Computer Network

Lecture taken by
Dharmendra Kumar
(Associate Professor)
United College of Engineering and Research, Prayagraj

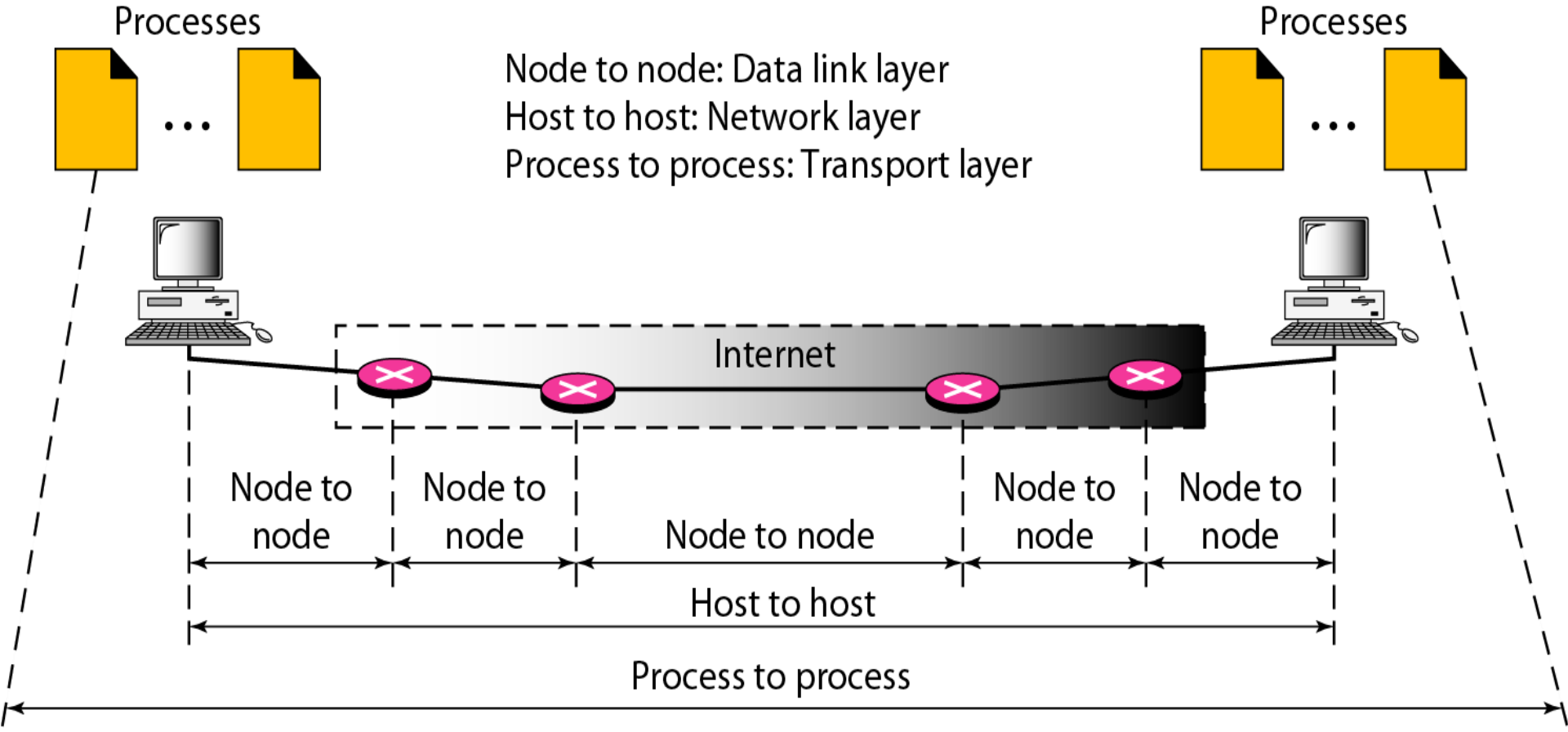
Unit-4

Transport Layer

Process-to-Process Delivery

Process-to-Process Delivery

The transport layer is responsible for process-to process delivery-the delivery of a packet, part of a message, from one process to another.



Process-to-Process Delivery

Client/Server Paradigm

- A process on the local host, called a client, needs services from a process usually on the remote host, called a server.
- Both processes (client and server) have the same name. For example, to get the day and time from a remote machine, we need a Daytime client process running on the local host and a Daytime server process running on a remote machine.

Process-to-Process Delivery

Addressing

- At the transport layer, we need a transport layer address, called a **port number**, to choose among multiple processes running on the destination host. The destination port number is needed for delivery; the source port number is needed for the reply.
- In the Internet model, the port numbers are 16-bit integers between 0 and 65,535.

Process-to-Process Delivery

IANA Ranges

The IANA (Internet Assigned Number Authority) has divided the port numbers into three ranges: well known, registered, and dynamic (or private).

Well-known ports: The ports ranging from 0 to 1023 are assigned and controlled by IANA. These are the well-known ports.

Registered ports: The ports ranging from 1024 to 49,151 are not assigned or controlled by IANA. They can only be registered with IANA to prevent duplication.

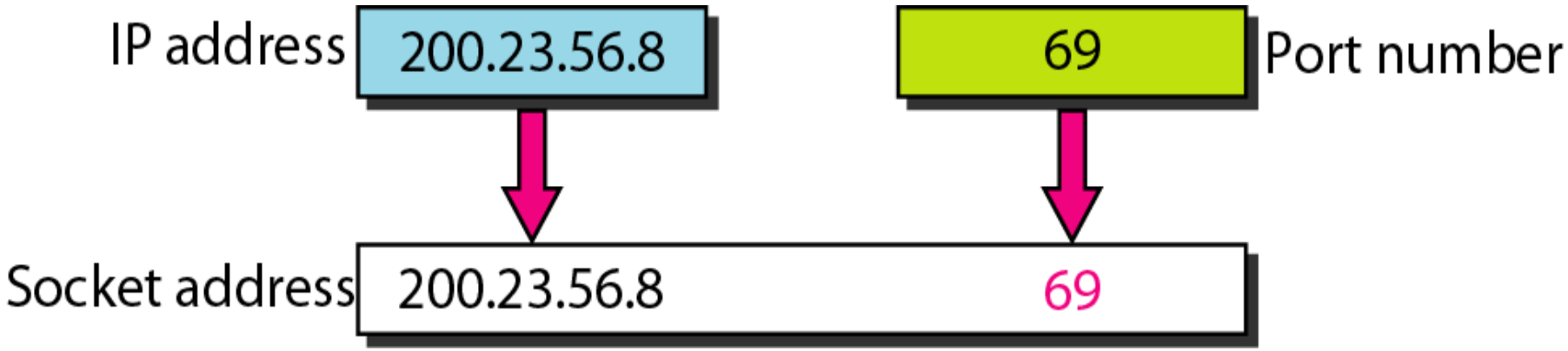
Dynamic ports: The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used by any process. These are the ephemeral ports.

Process-to-Process Delivery

Socket Addresses

Process-to-process delivery needs two identifiers, 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.



Process-to-Process Delivery

Connectionless Versus Connection-Oriented Service

A transport layer protocol can either be connectionless or connection-oriented.

Connectionless Service

In a connectionless service, the packets are sent from one party to another with no need for connection establishment or connection release. The packets are not numbered; they may be delayed or lost or may arrive out of sequence. There is no acknowledgment.

Connection-Oriented Service

In a connection-oriented service, a connection is first established between the sender and the receiver. Data are transferred. At the end, the connection is released.

Process-to-Process Delivery

Reliable Versus Unreliable

- The transport layer service can be reliable or unreliable. If the application layer program needs reliability, we use a reliable transport layer protocol by implementing flow and error control at the transport layer.
- On the other hand, if the application program does not need reliability because it uses its own flow and error control mechanism or it needs fast service or the nature of the service does not demand flow and error control (real-time applications), then an unreliable protocol can be used.

Process-to-Process Delivery

- In the Internet, there are three different transport layer protocols, UDP, TCP and SCTP.
- UDP is connectionless and unreliable;
- TCP and SCTP are connection-oriented and reliable.

Process-to-Process Delivery

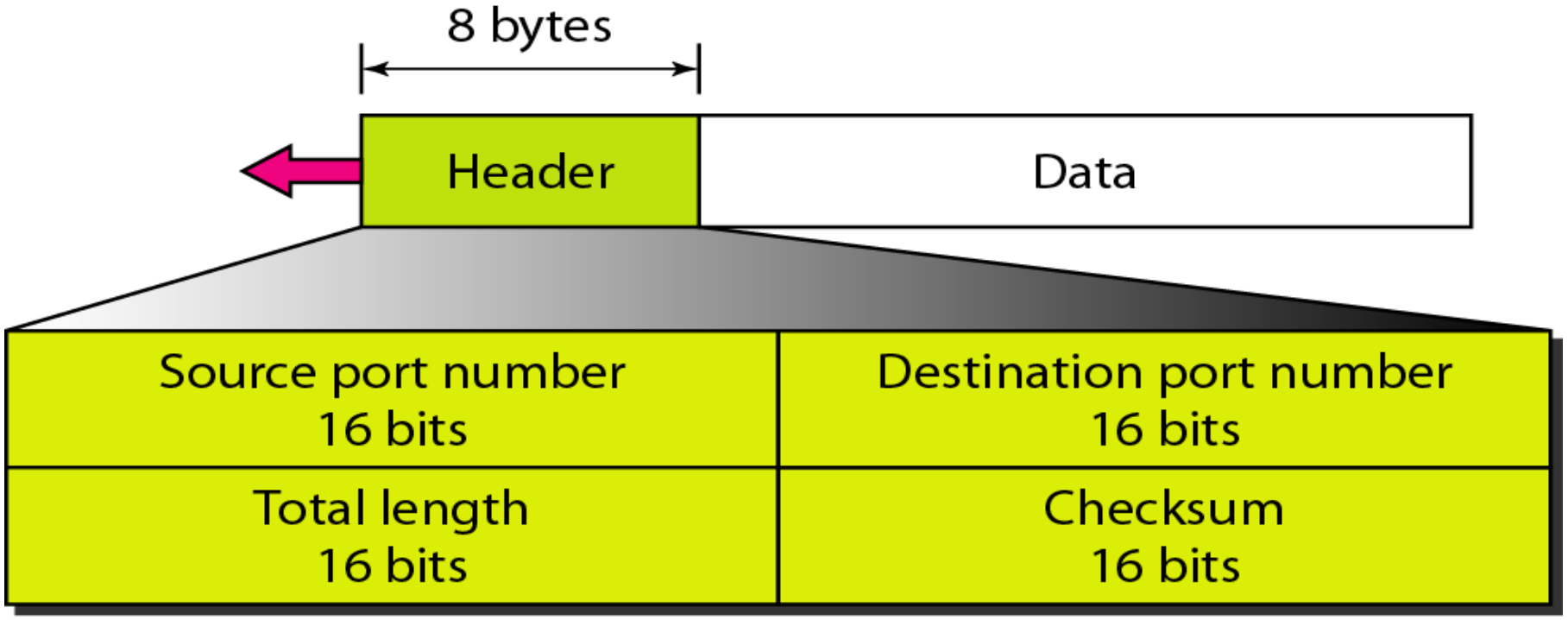
USER DATAGRAM PROTOCOL (UDP)

- The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol. It does not add anything to the services of IP except to provide process-to-process communication instead of host-to-host communication. Also, it performs very limited error checking.
- 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 by using UDP takes much less interaction between the sender and receiver than using TCP or SCTP.

Process-to-Process Delivery

User Datagram

UDP packets, called user datagrams, have a fixed-size header of 8 bytes. Following figure shows the format of a user datagram.



Process-to-Process Delivery

All the fields are explained as following:-

Source port number:

This is the port number used by the process running on the source host. It is 16 bits long, which means that the port number can range from 0 to 65,535. If the source host is the client (a client sending a request), the port number, in most cases, is an **ephemeral port number** requested by the process and chosen by the UDP software running on the source host. If the source host is the server (a server sending a response), the port number, in most cases, is a **well-known port number**.

Process-to-Process Delivery

Destination port number: This is the port number used by the process running on the destination host. It is also 16 bits long. If the destination host is the server (a client sending a request), the port number, in most cases, is a well-known port number. If the destination host is the client (a server sending a response), the port number, in most cases, is an ephemeral port number. In this case, the server copies the ephemeral port number it has received in the request packet.

Total length: This is a 16-bit field that defines the total length of the user datagram, header plus data. The 16 bits can define a total length of 0 to 65,535 bytes.

UDP datagram length = IP datagram length – IP header length

Checksum: This field is used to detect errors over the entire user datagram (header plus data).

Process-to-Process Delivery

UDP Operation

Connectionless Services

- UDP provides a connectionless service. This means that each user datagram sent by UDP is an independent datagram.
- There is no relationship between the different user datagrams even if they are coming from the same source process and going to the same destination program.
- The user datagrams are not numbered.
- Also, there is no connection establishment and no connection termination, as is the case for TCP. This means that each user datagram can travel on a different path.
- Only those processes sending short messages should use UDP.

Process-to-Process Delivery

Flow and Error Control

- UDP is a very simple, unreliable transport protocol. There is no flow control and hence no window mechanism. The receiver may overflow with incoming messages.
- There is no error control mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently discarded.

Process-to-Process Delivery

Use of UDP

The following are some uses of the UDP protocol:

- UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control.
- 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).

Process-to-Process Delivery

Transmission Control Protocol(TCP)

- Unlike UDP, TCP is a connection oriented protocol; it creates a virtual connection between two TCPs to send data. In addition, TCP uses flow and error control mechanisms at the transport level.

- TCP is called a connection-oriented, reliable transport protocol. It adds connection-oriented and reliability features to the services of IP.

Process-to-Process Delivery

TCP Services

Following are the services offered by TCP to the processes at the application layer.

Process-to-Process Communication

Like UDP, TCP provides process-to-process communication using port numbers.

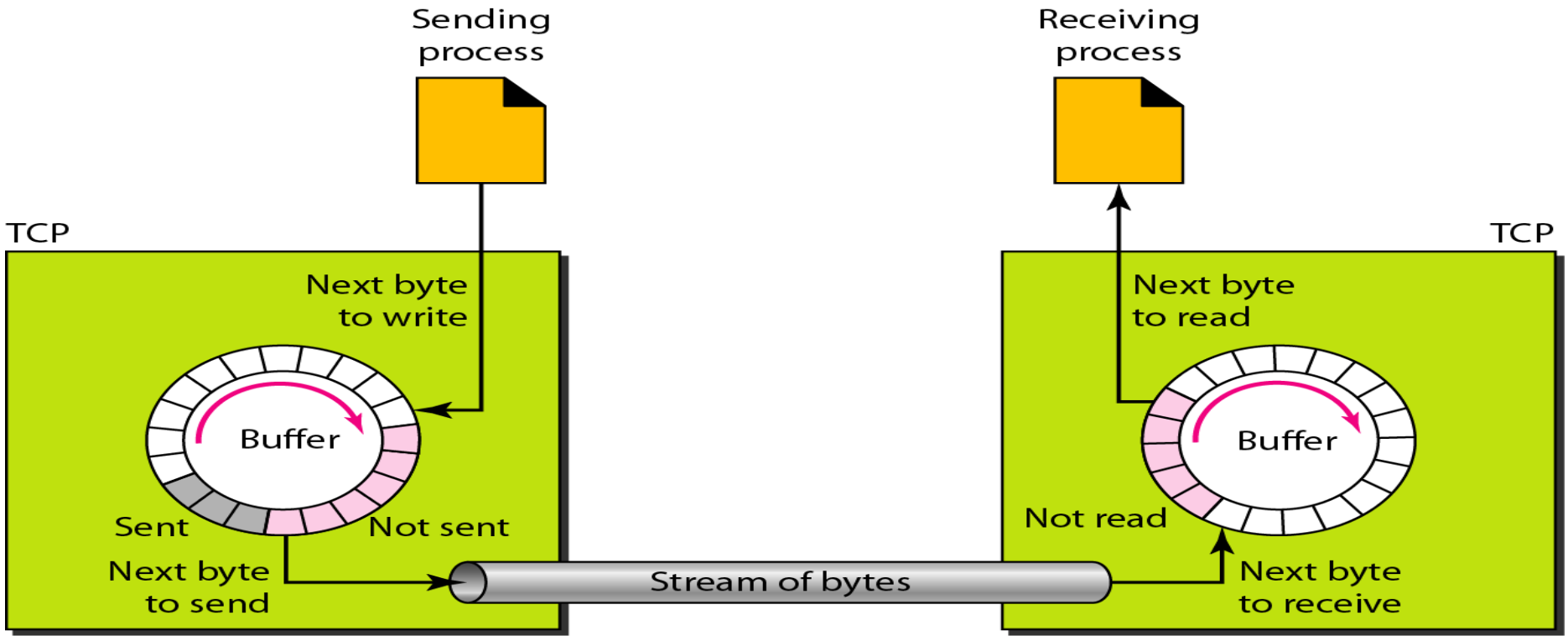
Stream Delivery Service

- TCP, unlike UDP, is a stream-oriented protocol.
- TCP 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 data across the Internet.
- The sending process produces (writes to) the stream of bytes, and the receiving process consumes (reads from) them.

Process-to-Process Delivery

Sending and Receiving Buffers

Because the sending and the receiving processes may not write or read data at the same speed, TCP needs buffers for storage. There are two buffers, the sending buffer and the receiving buffer, one for each direction.



Process-to-Process Delivery

- At the sending site, the buffer has three types of chambers. The white section contains empty chambers that can be filled by the sending process (producer). The gray area holds bytes that have been sent but not yet acknowledged. TCP keeps these bytes in the buffer until it receives an acknowledgment. The colored area contains bytes to be sent by the sending TCP. After the bytes in the gray chambers are acknowledged, the chambers are recycled and available for use by the sending process.
- The operation of the buffer at the receiver site is simpler. The circular buffer is divided into two areas (shown as white and colored). The white area contains empty chambers to be filled by bytes received from the network. The colored sections contain received bytes that can be read by the receiving process. When a byte is read by the receiving process, the chamber is recycled and added to the pool of empty chambers.

Process-to-Process Delivery

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 IP layer for transmission. The segments are encapsulated in IP datagrams and transmitted.

Full-Duplex Communication

TCP offers full-duplex service, in which data can flow in both directions at the same time. Each TCP then has a sending and receiving buffer, and segments move in both directions.

Process-to-Process Delivery

Connection-Oriented Service

TCP, unlike UDP, is a connection-oriented protocol.

Reliable Service

TCP is a reliable transport protocol. It uses an acknowledgment mechanism to check the safe and sound arrival of data.

Process-to-Process Delivery

TCP Features

To provide the services mentioned in the previous section, TCP has several features.

Numbering System

To keeps track of the segments being transmitted or received, TCP uses two fields called the sequence number and the acknowledgment number. These two fields refer to the byte number and not the segment number.

Byte Number

TCP numbers all data bytes that are transmitted in a connection. Numbering is independent in each direction. When TCP receives bytes of data from a process, it stores them in the sending buffer and numbers them. The numbering does not necessarily start from 0. Instead, TCP generates a random number between 0 and 232 - 1 for the number of the first byte. For example, if the random number happens to be 1057 and the total data to be sent are 6000 bytes, the bytes are numbered from 1057 to 7056.

Process-to-Process Delivery

Sequence Number

After the bytes have been numbered, TCP assigns a sequence number to each segment that is being sent. The sequence number for each segment is the number of the first byte carried in that segment.

Acknowledgment Number

The acknowledgment number defines the number of the next byte that the party expects to receive. The acknowledgment number is cumulative.

Process-to-Process Delivery

Flow Control

TCP, unlike UDP, provides flow control. The receiver of the data controls the amount of data that are to be sent by the sender. This is done to prevent the receiver from being overwhelmed with data. The numbering system allows TCP to use a byte-oriented flow control.

Error Control

To provide reliable service, TCP implements an error control mechanism.

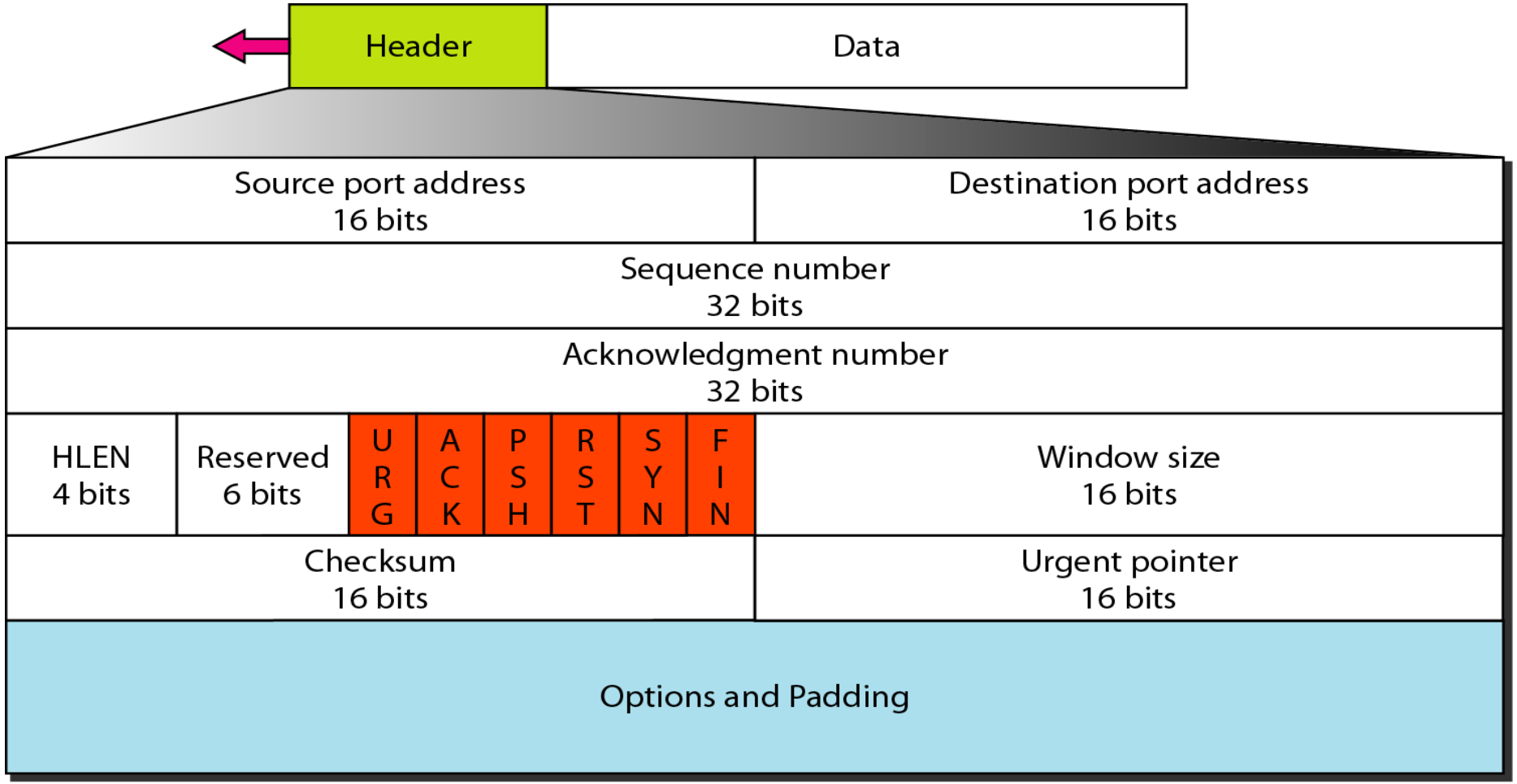
Congestion Control

TCP, unlike UDP, takes into account congestion in the network. The amount of data sent by a sender is not only controlled by the receiver (flow control), but is also determined by the level of congestion in the network.

Process-to-Process Delivery

TCP segment format

The format of a segment is shown in the following figure:-



Process-to-Process Delivery

The segment consists of a 20- to 60-byte header, followed by data from the application program. The header is 20 bytes if there are no options and up to 60 bytes if it contains options.

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. If the receiver of the segment has successfully received byte number x from the other party, it defines x + 1 as the acknowledgment number. Acknowledgment and data can be piggybacked together.

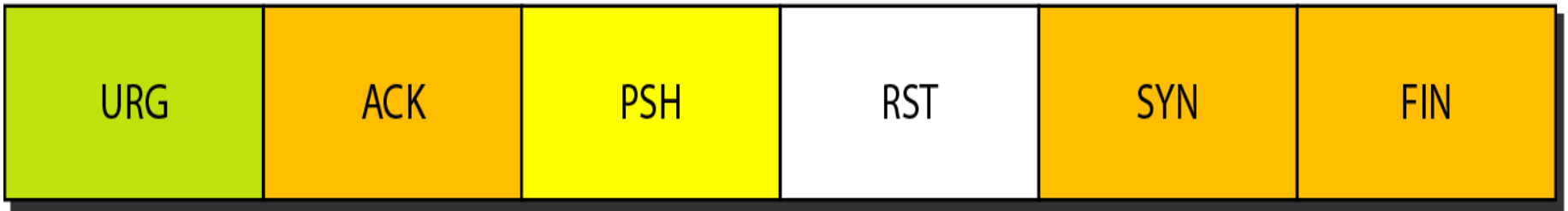
Process-to-Process Delivery

Header length: This 4-bit field indicates the number of 4-byte words in the TCP header. The length of the header can be between 20 and 60 bytes.

Reserved: This is a 6-bit field reserved for future use.

Control: This field defines 6 different control bits or flags as shown in figure. One or more of these bits can be set at a time.

URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection



Process-to-Process Delivery

These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP.

Flag	*Description*
URG	The value of the urgent pointer field is valid.
ACK	The value of the acknowledgment field is valid.
PSH	Push the data.
RST	Reset the connection.
SYN	Synchronize sequence numbers during connection.
FIN	Terminate the connection.

Process-to-Process Delivery

Window size: This field defines the size of the window, in bytes, that the other party must maintain. Note that the length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes. This value is normally referred to as the receiving window (rwnd) and is determined by the receiver. The sender must obey the dictation of the receiver in this case.

Checksum: This 16-bit field contains the checksum.

Urgent pointer: This 16-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data. It defines the number that must be added to the sequence number to obtain the number of the last urgent byte in the data section of the segment.

Options: There can be up to 40 bytes of optional information in the TCP header.

Process-to-Process Delivery

TCP Connection

In TCP, connection-oriented transmission requires three phases: connection establishment, data transfer, and connection termination.

Connection Establishment

TCP transmits data in full-duplex mode. When two TCPs in two machines are connected, they are able to send segments to each other simultaneously. This implies that each party must initialize communication and get approval from the other party before any data are transferred.

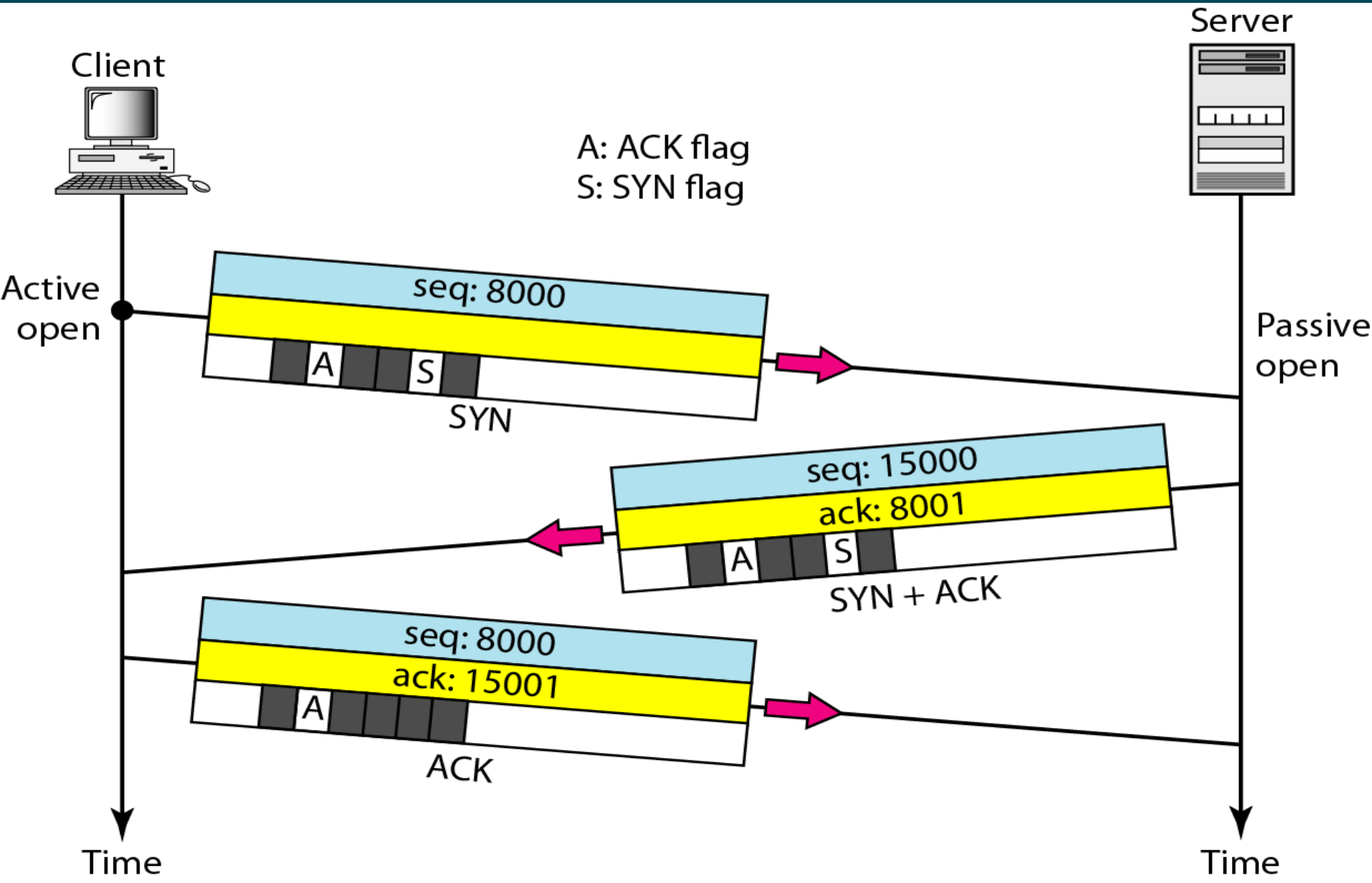
Process-to-Process Delivery

Three-Way Handshaking

The connection establishment in TCP is called three way handshaking. In our example, an application program, called the client, wants to make a connection with another application program, called the server, using TCP as the transport layer protocol.

The process starts with the server. The server program tells its TCP that it is ready to accept a connection. This is called a request for a passive open. Although the server TCP is ready to accept any connection from any machine in the world, it cannot make the connection itself.

Process-to-Process Delivery



Process-to-Process Delivery

The three steps in this phase are as follows:-

(1) The client sends the first segment, a SYN segment, in which only the SYN flag is set. This segment is for synchronization of sequence numbers. It consumes one sequence number. When the data transfer starts, the sequence number is incremented by 1. We can say that the SYN segment carries no real data, but we can think of it as containing 1 imaginary byte.

(2) The server sends the second segment, a SYN +ACK segment, with 2 flag bits set: SYN and ACK. This segment has a dual purpose. It is a SYN segment for communication in the other direction and serves as the acknowledgment for the SYN segment. It consumes one sequence number.

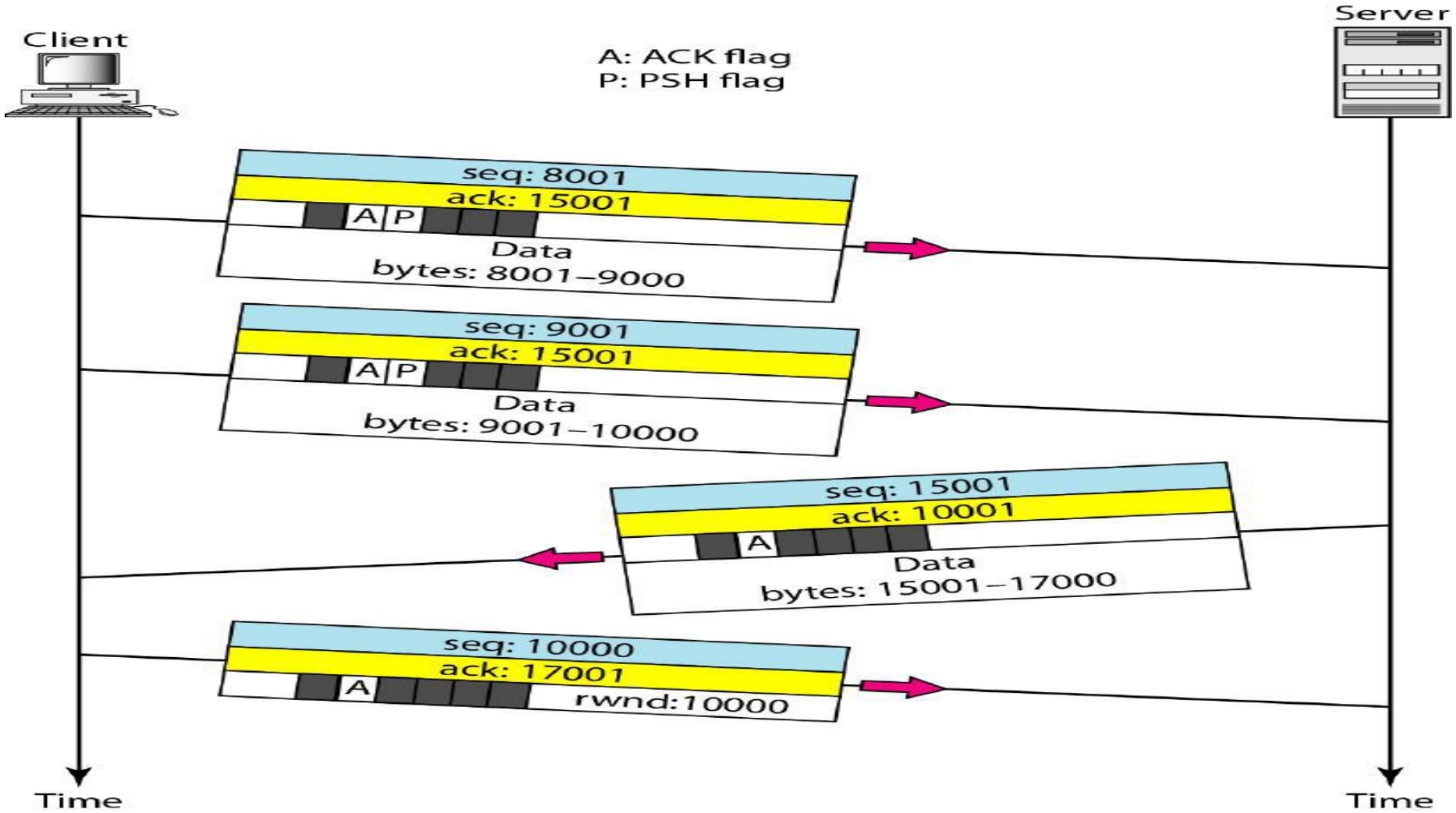
Process-to-Process Delivery

(3) The client sends the third segment. This is just an ACK segment. It acknowledges the receipt of the second segment with the ACK flag and acknowledgment number field. Note that the sequence number in this segment is the same as the one in the SYN segment; the ACK segment does not consume any sequence numbers.

Process-to-Process Delivery

Data Transfer

This process is shown in the following figure:-



Process-to-Process Delivery

In this figure, after connection is established , the client sends 2000 bytes of data in two segments. The server then sends 2000 bytes in one segment. The client sends one more segment. The first three segments carry both data and acknowledgment, but the last segment carries only an acknowledgment because there are no more data to be sent. The data segments sent by the client have the PSH (push) flag set so that the server TCP knows to deliver data to the server process as soon as they are received. The segment from the server, on the other hand, does not set the push flag.

Pushing Data

The application program at the sending site can request a push operation. This means that the sending TCP must not wait for the window to be filled. It must create a segment and send it immediately. The sending TCP must also set the push bit (PSH) to let the receiving TCP know that the segment includes data that must be delivered to the receiving application program as soon as possible and not to wait for more data to come.

Process-to-Process Delivery

Urgent Data

- URG bit is set to send urgent data.
- The sending application program tells the sending TCP that the piece of data is urgent. The sending TCP creates a segment and inserts the urgent data at the beginning of the segment. The rest of the segment can contain normal data from the buffer. The urgent pointer field in the header defines the end of the urgent data and the start of normal data.
- When the receiving TCP receives a segment with the URG bit set, it extracts the urgent data from the segment, using the value of the urgent pointer, and delivers them, out of order, to the receiving application program.

Process-to-Process Delivery

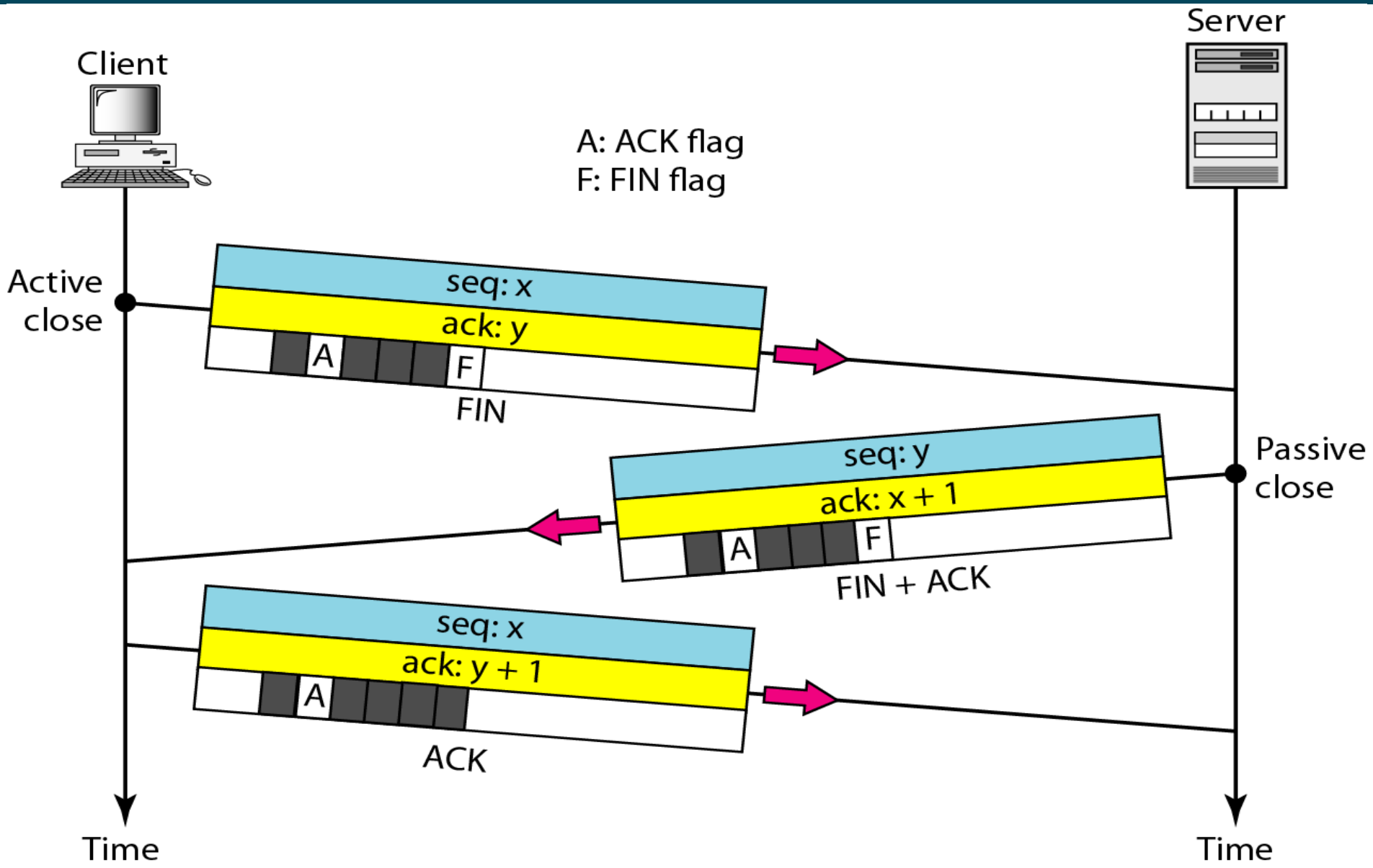
Connection Termination

Any of the two parties involved in exchanging data (client or server) can close the connection, although it is usually initiated by the client. There are two methods to close the connection as three-way handshaking and four-way handshaking with a half-close option.

Three-Way Handshaking

Most implementations today allow three-way handshaking for connection termination. It is shown in the following figure:-

Process-to-Process Delivery



Process-to-Process Delivery

1. In a normal situation, the client TCP, after receiving a close command from the client process, sends the first segment, a FIN segment in which the FIN flag is set. Note that a FIN segment can include the last chunk of data sent by the client, or it can be just a control segment as shown in Figure 23.20. If it is only a control segment, it consumes only one sequence number.
2. The server TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment, a FIN +ACK segment, to confirm the receipt of the FIN segment from the client and at the same time to announce the closing of the connection in the other direction. This segment can also contain the last chunk of data from the server. If it does not carry data, it consumes only one sequence number.

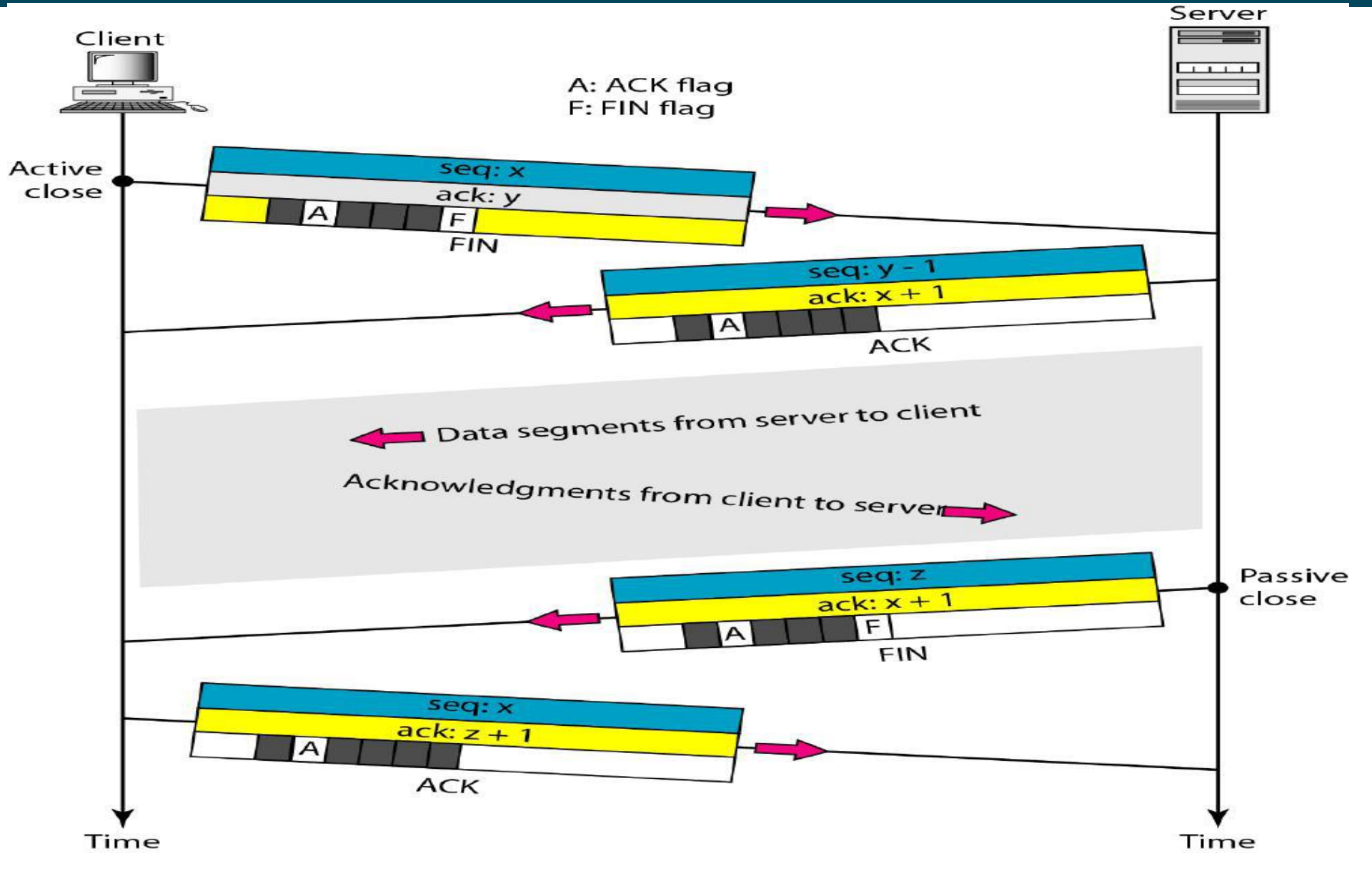
Process-to-Process Delivery

3. The client TCP sends the last segment, an ACK segment, to confirm the receipt of the FIN segment from the TCP server. This segment contains the acknowledgment number, which is 1 plus the sequence number received in the FIN segment from the server. This segment cannot carry data and consumes no sequence numbers.

Half-Close

In TCP, one end can stop sending data while still receiving data. This is called a half-close. Although either end can issue a half-close, it is normally initiated by the client.

Process-to-Process Delivery



Process-to-Process Delivery

This figure shows an example of a half-close. The client half-closes the connection by sending a FIN segment. The server accepts the half-close by sending the ACK segment. The data transfer from the client to the server stops. The server, however, can still send data. When the server has sent all the processed data, it sends a FIN segment, which is acknowledged by an ACK from the client.

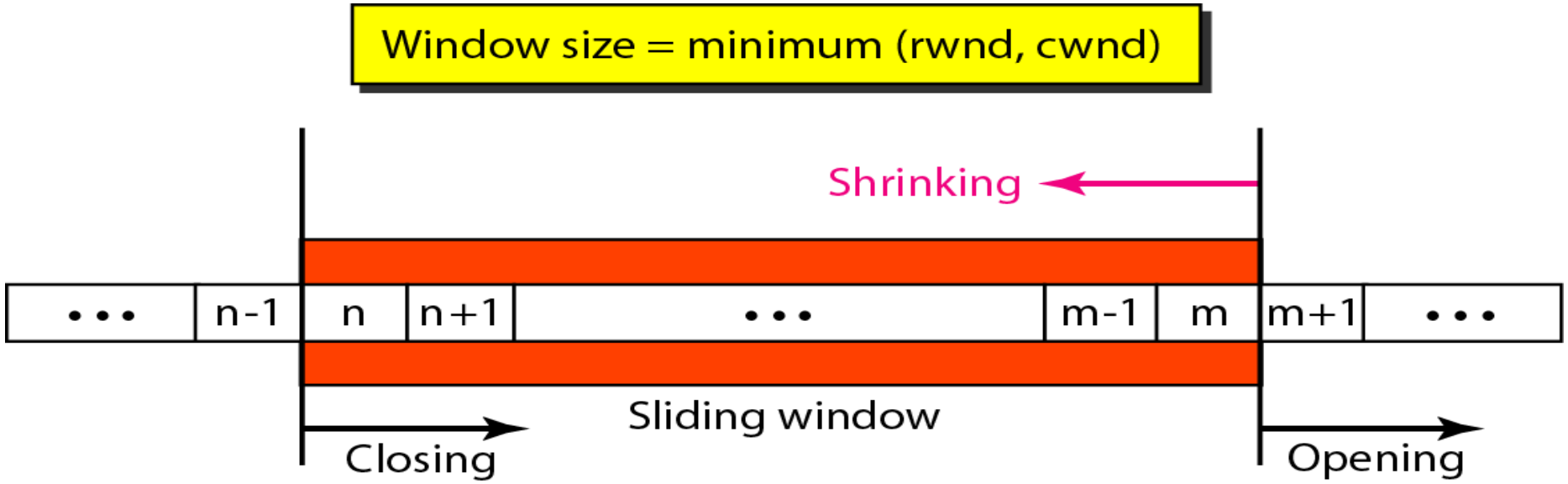
After half-closing of the connection, data can travel from the server to the client and acknowledgments can travel from the client to the server. The client cannot send any more data to the server. Note the sequence numbers we have used. The second segment (ACK) consumes no sequence number. Although the client has received sequence number y - 1 and is expecting y, the server sequence number is still y - 1. When the connection finally closes, the sequence number of the last ACK segment is still x, because no sequence numbers are consumed during data transfer in that direction.

Process-to-Process Delivery

Flow Control or TCP Sliding Window

- TCP uses a sliding window, to handle flow control. The sliding window protocol used by TCP, however, is something between the Go-Back-N and Selective Repeat sliding window.
- The sliding window protocol in TCP looks like the Go-Back-N protocol because it does not use NAKs; it looks like Selective Repeat because the receiver holds the out-of-order segments until the missing ones arrive.
- There are two big differences between this sliding window and the one we used at the data link layer.
  - The sliding window of TCP is byte-oriented; the one we discussed in the data link layer is frame-oriented.
  - The TCP's sliding window is of variable size; the one we discussed in the data link layer was of fixed size.

Process-to-Process Delivery



The window is opened, closed, or shrunk. These three activities, are in the control of the receiver (and depend on congestion in the network), not the sender.

Process-to-Process Delivery

The sender must obey the commands of the receiver in this matter.

Opening a window means moving the right wall to the right. This allows more new bytes in the buffer that are eligible for sending.

Closing the window means moving the left wall to the right. This means that some bytes have been acknowledged and the sender need not worry about them anymore.

Shrinking the window means moving the right wall to the left.

Process-to-Process Delivery

The size of the window at one end is determined by the lesser of two values: receiver window (rwnd) or congestion window (cwnd).

The **receiver window** is the value advertised by the opposite end in a segment containing acknowledgment. It is the number of bytes the other end can accept before its buffer overflows and data are discarded.

The **congestion window** is a value determined by the network to avoid congestion

Process-to-Process Delivery

Silly Window Syndrome

Silly Window Syndrome is a problem that arises due to the poor implementation of TCP flow control.

It degrades the TCP performance and makes the data transmission extremely inefficient.

It causes the sender window size to shrink to a silly value.

The window size shrinks to such an extent where the data being transmitted is smaller than TCP Header.

Process-to-Process Delivery

Causes-

The problem arises due to following causes-

1. Sender transmitting data in small segments repeatedly
2. Receiver accepting only few bytes at a time repeatedly

Cause-01: Sender Transmitting Data In Small Segments Repeatedly-

- Consider application generates one byte of data to send at a time.
- The poor implementation of TCP causes the sender to send each byte of data in an individual TCP segment.

This problem is solved using Nagle's Algorithm.

Nagle's Algorithm-

Process-to-Process Delivery

Nagle's Algorithm-

Nagle's algorithm suggests-

- Sender should send only the first byte on receiving one byte data from the application.
- Sender should buffer all the rest bytes until the outstanding byte gets acknowledged.
- In other words, sender should wait for 1 RTT.
- After receiving the acknowledgement, sender should send the buffered data in one TCP segment.
- Then, sender should buffer the data again until the previously sent data gets acknowledged.

Process-to-Process Delivery

Cause-02: Receiver Accepting Only Few Bytes Repeatedly-

- Consider the receiver continues to be unable to process all the incoming data.
- In such a case, its window size becomes smaller and smaller.
- A stage arrives when it repeatedly sends the window size of 1 byte to the sender.

This problem is solved using Clark's Solution.

Clark's Solution-

Clark's solution suggests-

- Receiver should not send a window update for 1 byte.
- Receiver should wait until it has a decent amount of space available.
- Receiver should then advertise that window size to the sender.

Process-to-Process Delivery

Error Control

TCP is a reliable transport layer protocol. This means that an application program that delivers a stream of data to TCP relies on TCP to deliver the entire stream to the application program on the other end in order, without error, and without any part lost or duplicated.

TCP provides reliability using error control. Error control includes mechanisms for detecting corrupted segments, lost segments, out-of-order segments, and duplicated segments. Error control also includes a mechanism for correcting errors after they are detected. Error detection and correction in TCP is achieved through the use of three simple tools: **checksum, acknowledgment, and time-out.**

Process-to-Process Delivery

Checksum

Each segment includes a checksum field which is used to check for a corrupted segment. If the segment is corrupted, it is discarded by the destination TCP and is considered as lost. TCP uses a 16-bit checksum that is mandatory in every segment.

Acknowledgment

TCP uses acknowledgments to confirm the receipt of data segments. Control segments that carry no data but consume a sequence number are also acknowledged. ACK segments are never acknowledged.

ACK segments do not consume sequence numbers and are not acknowledged.

Process-to-Process Delivery

Retransmission

The heart of the error control mechanism is the retransmission of segments. When a segment is corrupted, lost, or delayed, it is retransmitted.

In modern implementations, a segment is retransmitted on two occasions: when a retransmission timer expires or when the sender receives three duplicate ACKs.

Out-of-Order Segments

- When a segment is delayed, lost, or discarded, the segments following that segment arrive out of order.
- Originally, TCP was designed to discard all out-of-order segments, resulting in the retransmission of the missing segment and the following segments.
- Most implementations today do not discard the out-of-order segments. They store them temporarily and flag them as out-of-order segments until the missing segment arrives. TCP guarantees that data are delivered to the process in order.

Process-to-Process Delivery

Exercise

1. The following is a dump of a UDP header in hexadecimal format.

0632000D00 1CE217

a. What is the source port number?

b. What is the destination port number?

c. What is the total length of the user datagram?

d. What is the length of the data?

e. Is the packet directed from a client to a server or vice versa?

f. What is the client process?

Process-to-Process Delivery

2. The following is a dump of a TCP header in hexadecimal format.

05320017 00000001 00000000 500207FF 00000000

a. What is the source port number?

b. What is the destination port number?

c. What the sequence number?

d. What is the acknowledgment number?

e. What is the length of the header?

f. What is the type of the segment?

g. What is the window size?

Process-to-Process Delivery

3. If WAN link is 2 Mbps and RTT between source and destination is 300 msec, what would be the optimal TCP window size needed to fully utilize the line?
   a) 60,000 bits
   b) 75,000 bytes
   c) 75,000 bits
   d) 60,000 bytes
4. Suppose host A is sending a large file to host B over a TCP connection. The two end hosts are 10 msec apart (20 msec RTT) connected by a 1 Gbps link. Assume that they are using a packet size of 1000 bytes to transmit the file. For simplicity, ignore ack packets. At least how big would the window size (in packets) have to be for the channel utilization to be greater than 80%?
   a) 1000
   b) 1500
   c) 2000
   d) 2500

Process-to-Process Delivery

5. A TCP machine is sending windows of 65535 B over a 1 Gbps channel that has a 10 msec one way delay.
   a) What is the maximum throughput achievable?
   b) What is the line efficiency?

6. Consider the three-way handshake mechanism followed during TCP connection establishment hosts P and Q. Let X and Y be two random 32-bit starting sequence numbers chosen by P and Q respectively. Suppose P sends a TCP connection request message to Q with a TCP segment having SYN bit = 1, SEQ number = X, and ACK bit = 0. Suppose Q accepts the connection request. Which one of the following choices represents the information present in the TCP segment header that is sent by Q to P?

(A) SYN bit = 0, SEQ number = X + 1, ACK bit = 0, ACK number = Y, FIN bit = 1
(B) SYN bit = 1, SEQ number = Y, ACK bit = 1, ACK number = X + 1, FIN bit = 0
(C) SYN bit = 1, SEQ number = Y, ACK bit = 1, ACK number = X, FIN bit = 0
(D) SYN bit = 1, SEQ number = X + 1, ACK bit = 0, ACK number = Y, FIN bit = 0

Process-to-Process Delivery

5. Consider a TCP connection in a state where there are no outstanding ACKs. The sender sends two segments back to back. The sequence numbers of the first and second segments are 230 and 290 respectively. The first segment was lost but the second segment was received correctly by the receiver. Let X be the amount of data carried in the first segment (in bytes) and Y be the ACK number sent by the receiver. The values of X and Y are-

a) 60 and 290
b) 230 and 291
c) 60 and 231
d) 60 and 230

Congestion Control and Quality of Service

Congestion Control and Quality of Service

- Congestion control and quality of service are two issues so closely bound together that improving one means improving the other and ignoring one usually means ignoring the other.
- Most techniques to prevent or eliminate congestion also improve the quality of service in a network.

Congestion Control and Quality of Service

CONGESTION

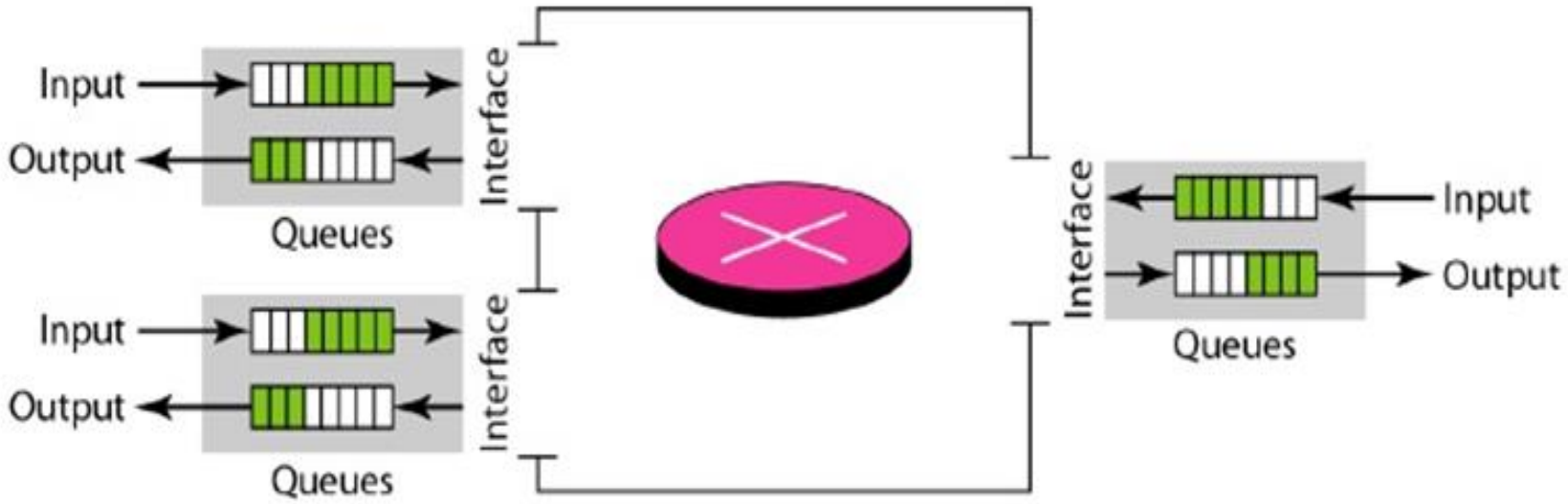
Congestion in a network may occur if the load on the network (the number of packets sent to the network) is greater than the capacity of the network (the number of packets a network can handle).

Congestion control refers to the mechanisms and techniques to control the congestion and keep the load below the capacity.

Congestion happens in any system that involves waiting. For example, congestion happens on a freeway because any abnormality in the flow, such as an accident during rush hour, creates blockage.

Congestion Control and Quality of Service

Congestion in a network or internetwork occurs because routers and switches have queues-buffers that hold the packets before and after processing. A router, for example, has an input queue and an output queue for each interface. When a packet arrives at the incoming interface, it undergoes three steps before departing, as shown in following figure .



Congestion Control and Quality of Service

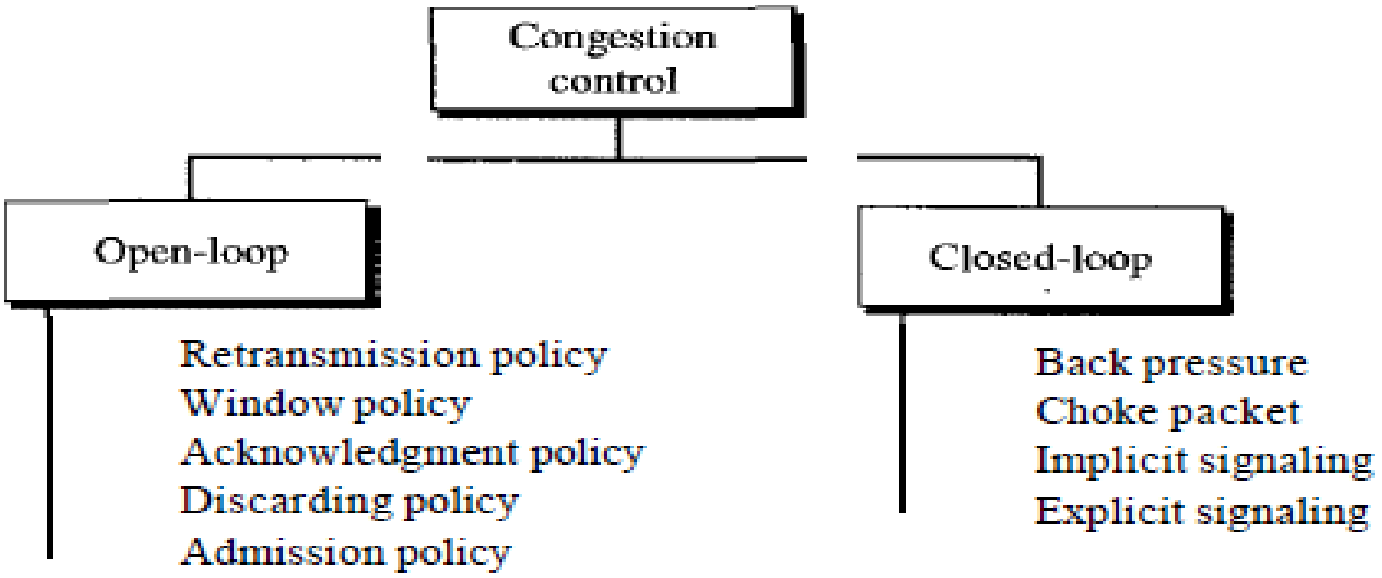
1. The packet is put at the end of the input queue while waiting to be checked.
2. The processing module of the router removes the packet from the input queue once it reaches the front of the queue and uses its routing table and the destination address to find the route.
3. The packet is put in the appropriate output queue and waits its tum to be sent.

We need to be aware of two issues. First, if the rate of packet arrival is higher than the packet processing rate, the input queues become longer and longer. Second, if the packet departure rate is less than the packet processing rate, the output queues become longer and longer.

Congestion Control and Quality of Service

CONGESTION CONTROL

- Congestion control refers to techniques and mechanisms that can either prevent congestion, before it happens, or remove congestion, after it has happened.
- In general, we can divide congestion control mechanisms into two broad categories: open-loop congestion control (prevention) and closed-loop congestion control (removal) as shown in following figure:



Congestion Control and Quality of Service

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.

Retransmission Policy

- Retransmission is sometimes unavoidable.
- If the sender feels that a sent packet is lost or corrupted, the packet needs to be retransmitted.
- Retransmission in general may increase congestion in the network. However, a good retransmission policy can prevent congestion. The retransmission policy and the retransmission timers must be designed to optimize efficiency and at the same time prevent congestion.
- For example, the retransmission policy used by TCP is designed to prevent or alleviate congestion.

Congestion Control and Quality of Service

Window Policy

The type of window at the sender may also affect congestion. The Selective Repeat window is better than the Go-Back-N window for congestion control. In the Go-Back-N window, when the timer for a packet times out, several packets may be resent, although some may have arrived safe and sound at the receiver. This duplication may make the congestion worse. The Selective Repeat window, on the other hand, tries to send the specific packets that have been lost or corrupted.

Acknowledgment Policy

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

Several approaches are used in this case. A receiver may send an acknowledgment only if it has a packet to be sent or a special timer expires. A receiver may decide to acknowledge only N packets at a time. We need to know that the acknowledgments are also part of the load in a network. Sending fewer acknowledgments means imposing less load on the network.

Congestion Control and Quality of Service

Discarding Policy

A good discarding policy by the routers may prevent congestion and at the same time may not harm the integrity of the transmission. For example, in audio transmission, if the policy is to discard less sensitive packets when congestion is likely to happen, the quality of sound is still preserved and congestion is prevented or alleviated.

Admission Policy

An admission policy, which is a quality-of-service mechanism, can also prevent congestion in virtual-circuit networks. Switches in a flow first check the resource requirement of a flow before admitting it to the network. A router can deny establishing a virtual circuit connection if there is congestion in the network or if there is a possibility of future congestion.

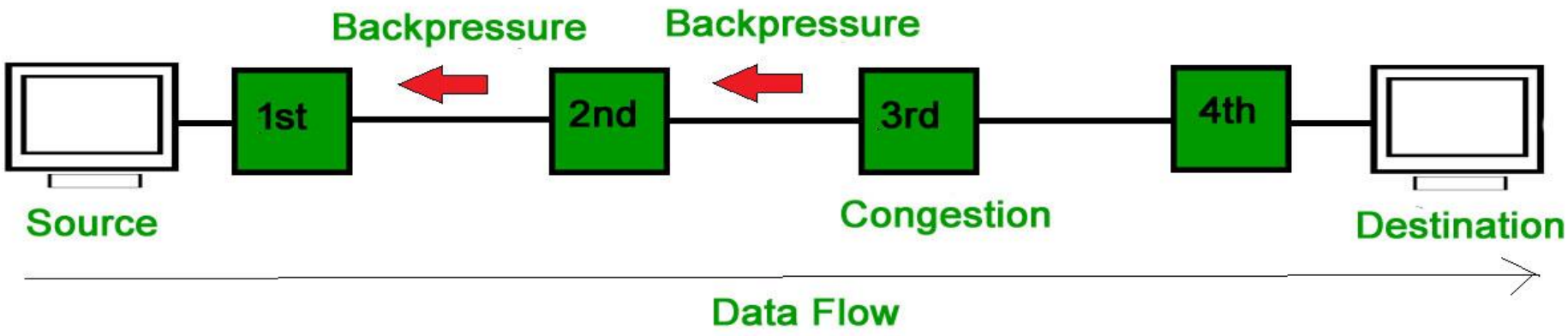
Congestion Control and Quality of Service

Closed-Loop Congestion Control

Closed-loop congestion control mechanisms try to alleviate congestion after it happens. Several mechanisms have been used by different protocols.

Backpressure

Backpressure is a node-to-node congestion control that starts with a node and propagates, in the opposite direction of data flow, to the source. The backpressure technique can be applied only to virtual circuit networks, in which each node knows the upstream node from which a flow of data is coming. Following figure shows the idea of backpressure.



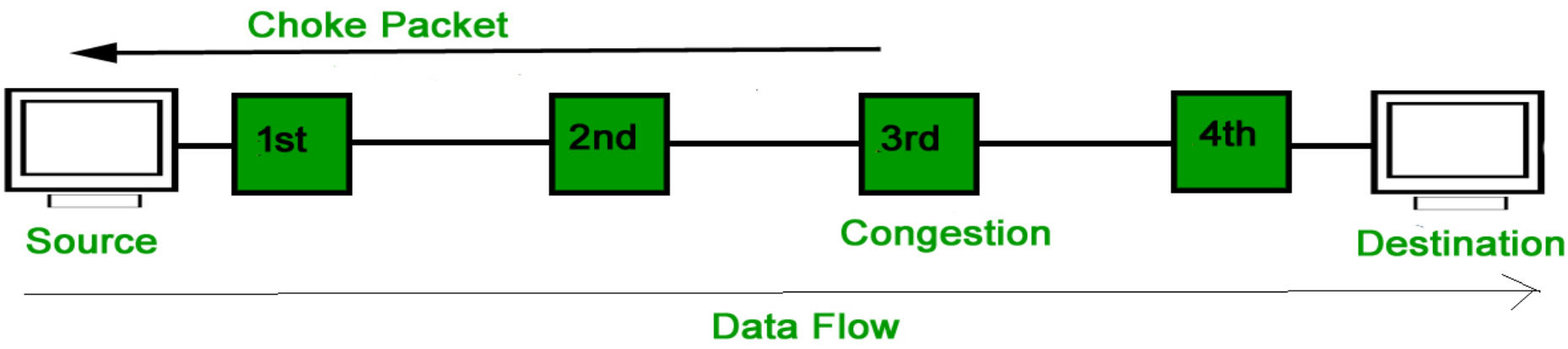
Congestion Control and Quality of Service

Choke Packet

A choke packet is a packet sent by a node to the source to inform it of congestion.

In backpressure, the warning is from one node to its upstream node, although the warning may eventually reach the source station.

In the choke packet method, the warning is from the router, which has encountered congestion, to the source station directly. The intermediate nodes through which the packet has traveled are not warned. An example of this type of control in ICMP.



Congestion Control and Quality of Service

Implicit Signaling

In implicit signaling, there is no communication between the congested node or nodes and the source. The source guesses that there is a congestion somewhere in the network from other symptoms. For example, when a source sends several packets and there is no acknowledgment for a while, one assumption is that the network is congested. The delay in receiving an acknowledgment is interpreted as congestion in the network; the source should slow down. It is used in the TCP congestion control technique.

Explicit Signaling

The node that experiences congestion can explicitly send a signal to the source or destination. The explicit signaling method, however, is different from the choke packet method. In the choke packet method, a separate packet is used for this purpose; in the explicit signaling method, the signal is included in the packets that carry data. Explicit signaling, used in Frame Relay congestion control, can occur in either the forward or the backward direction.

Congestion Control and Quality of Service

Backward Signaling

A bit can be set in a packet moving in the direction opposite to the congestion. This bit can warn the source that there is congestion and that it needs to slow down to avoid the discarding of packets.

Forward Signaling

A bit can be set in a packet moving in the direction of the congestion. This bit can warn the destination that there is congestion. The receiver in this case can use policies, such as slowing down the acknowledgments, to alleviate the congestion.

Congestion Control and Quality of Service

Congestion Control in TCP

Congestion Window

The sender's window size is determined not only by the receiver but also by congestion in the network.

The sender has two pieces of information: the receiver advertised window size(rwnd) and the congestion window size(cwnd). The actual size of the window is the minimum of these two i.e.

Actual window size = minimum (rwnd, cwnd)

Congestion Policy

TCP's general policy for handling congestion is based on three phases: slow start, congestion avoidance, and congestion detection.

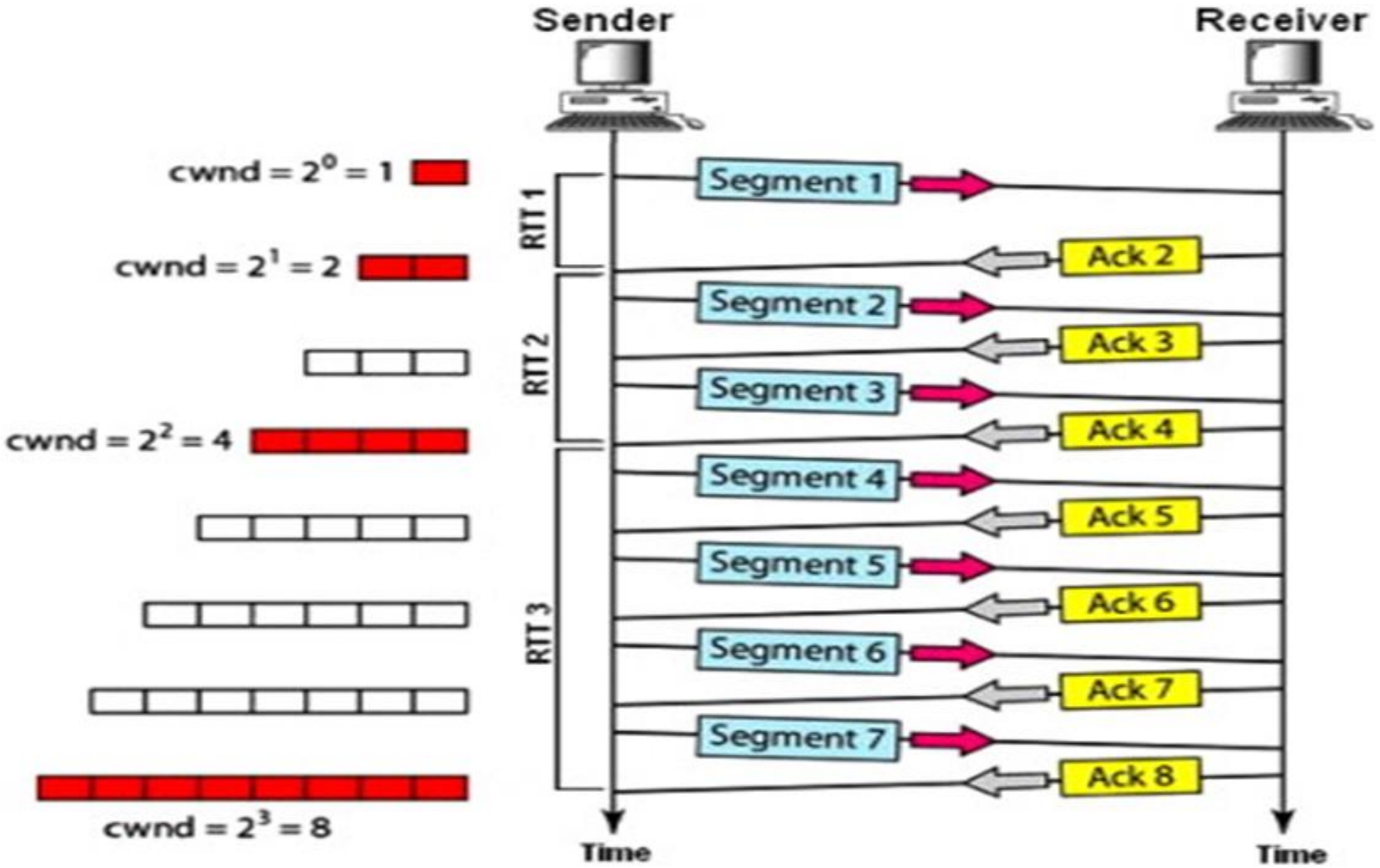
Congestion Control and Quality of Service

Slow Start: Exponential Increase:

This algorithm is based on the idea that the size of the congestion window (cwnd) starts with one maximum segment size (MSS). The size of the window increases one MSS each time an acknowledgment is received. As the name implies, the window starts slowly, but grows exponentially. It is shown in the following figure:-

In this figure, we have assumed that rwnd is much higher than cwnd, so that the sender window size always equals cwnd. We have assumed that each segment is acknowledged individually.

Congestion Control and Quality of Service



Congestion Control and Quality of Service

The sender starts with cwnd =1 MSS. This means that the sender can send only one segment. After receipt of the acknowledgment for segment 1, the size of the congestion window is increased by 1, which means that cwnd is now 2. Now two more segments can be sent. When each acknowledgment is received, the size of the window is increased by 1 MSS. When all seven segments are acknowledged, cwnd = 8.

Congestion Control and Quality of Service

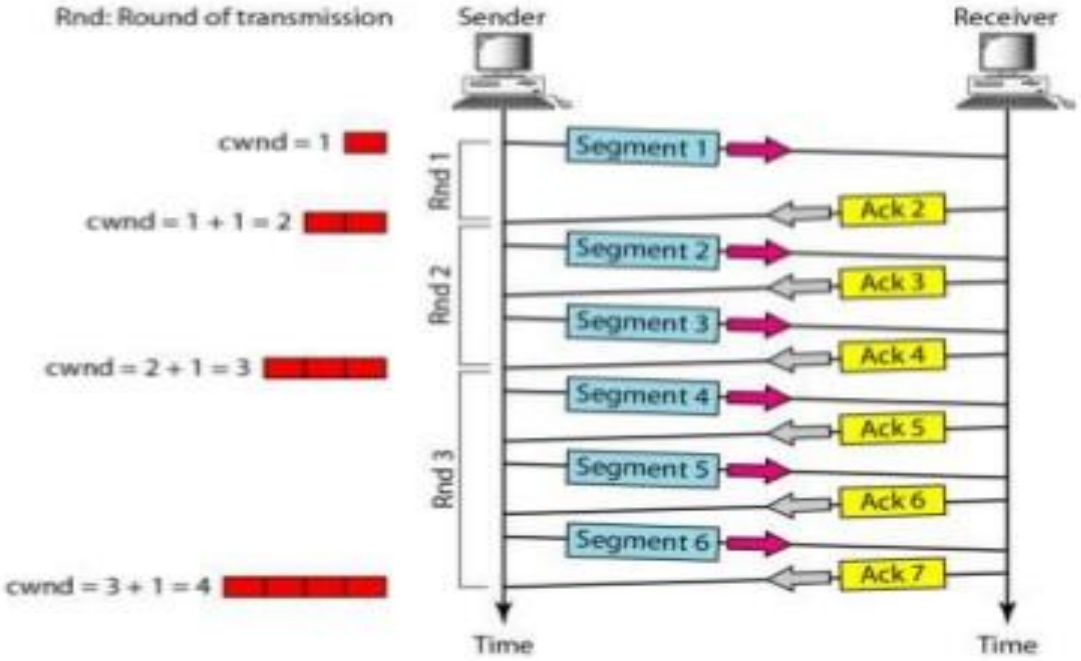
Congestion Avoidance: Additive Increase

TCP defines another algorithm called congestion avoidance, which undergoes an additive increase instead of an exponential one. When the size of the congestion window reaches the slow-start threshold, the slow-start phase stops and the additive phase begins. In this algorithm, each time the whole window of segments is acknowledged (one round), the size of the congestion window is increased by 1. It is shown in the following figure:-

Congestion Control and Quality of Service

CONGESTION CONTROL IN TCP

Congestion avoidance, additive increase:



Congestion Control and Quality of Service

Congestion Detection: Multiplicative Decrease

If congestion occurs, the congestion window size must be decreased. The only way the sender can guess that congestion has occurred is by the need to retransmit a segment. However, retransmission can occur in one of two cases: when a timer times out or when three ACKs are received. In both cases, the size of the threshold is dropped to one-half, a multiplicative decrease. Most TCP implementations have two reactions:

1. If a time-out occurs, there is a stronger possibility of congestion; a segment has probably been dropped in the network, and there is no news about the sent segments.

In this case TCP reacts strongly:

a. It sets the value of the threshold to one-half of the current window size.

b. It sets cwnd to the size of one segment.

c. It starts the slow-start phase again.

Congestion Control and Quality of Service

2. If three ACKs are received, there is a weaker possibility of congestion; a segment may have been dropped, but some segments after that may have arrived safely since three ACKs are received. This is called fast transmission and fast recovery. In this case, TCP has a weaker reaction:

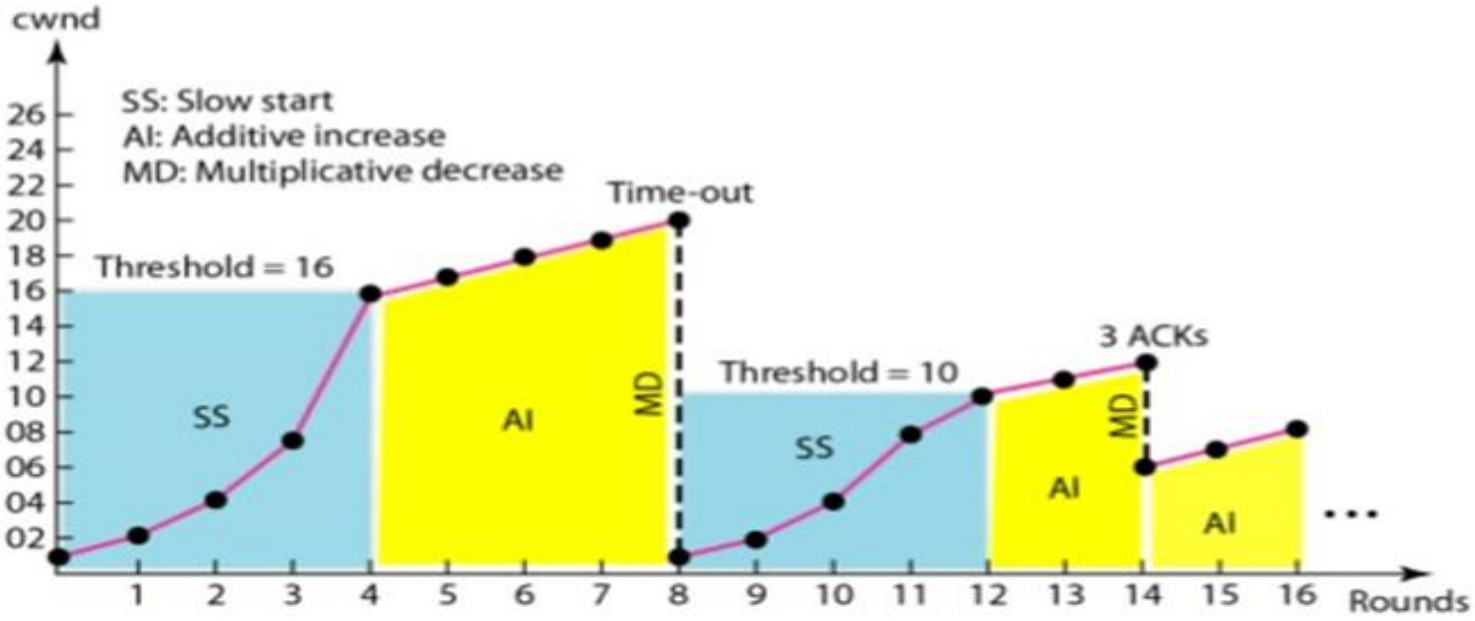
   a. It sets the value of the threshold to one-half of the current window size.

   b. It sets cwnd to the value of the threshold (some implementations add three segment sizes to the threshold).

   c. It starts the congestion avoidance phase.

Congestion Control and Quality of Service

Example:



In this figure, the time-out occurs when the window size is 20. At this moment, the multiplicative decrease procedure takes over and reduces the threshold to one-half of the previous window size. The previous window size was 20 when the time-out happened so the new threshold is now 10.

Congestion Control and Quality of Service

TCP moves to slow start again and starts with a window size of 1, and TCP moves to additive increase when the new threshold is reached. When the window size is 12, a three-ACKs event happens. The multiplicative decrease procedure takes over again. The threshold is set to 6 and TCP goes to the additive increase phase this time. It remains in this phase until another time-out or another three ACKs happen.

Congestion Control and Quality of Service

QUALITY OF SERVICE

We can informally define quality of service as something a flow seeks to attain.

Flow Characteristics

There are four types of characteristics attributed to a flow: reliability, delay, jitter, and bandwidth.

Reliability

Reliability is a characteristic that a flow needs. Lack of reliability means losing a packet or acknowledgment, which entails retransmission. However, the sensitivity of application programs to reliability is not the same. For example, it is more important that electronic mail, file transfer, and Internet access have reliable transmissions than telephony or audio conferencing.

Congestion Control and Quality of Service

Delay

Source-to-destination delay is another flow characteristic. Again applications can tolerate delay in different degrees. In this case, telephony, audio conferencing, video conferencing, and remote log-in need minimum delay, while delay in file transfer or e-mail is less important.

Jitter

Jitter is the variation in delay for packets belonging to the same flow. For example, if four packets depart at times 0, 1, 2, 3 and arrive at 20, 21, 22, 23, all have the same delay, 20 units of time. On the other hand, if the above four packets arrive at 21, 23, 21, and 28, they will have different delays: 21,22, 19, and 24.

For applications such as audio and video, the first case is completely acceptable; the second case is not.

Bandwidth

Different applications need different bandwidths. In video conferencing we need to send millions of bits per second to refresh a color screen while the total number of bits in an e-mail may not reach even a million.

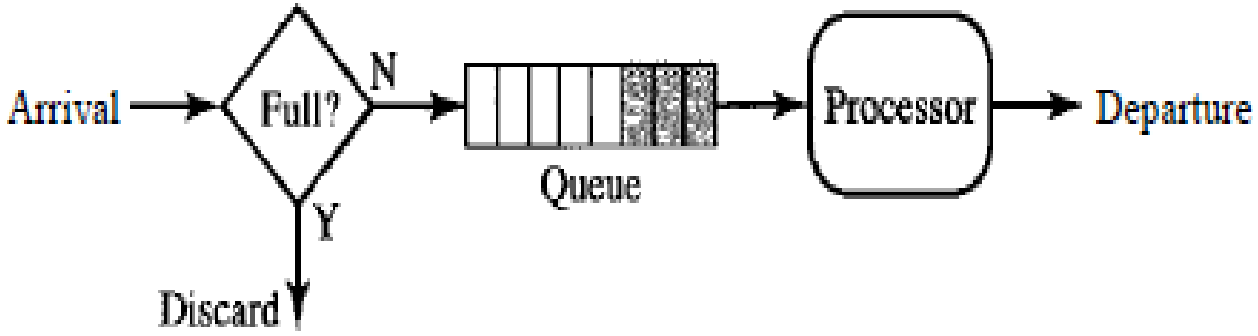
Congestion Control and Quality of Service

TECHNIQUES TO IMPROVE QoS

There are four common methods to improve the quality of service: scheduling, traffic shaping, admission control, and resource reservation.

FIFO Queuing

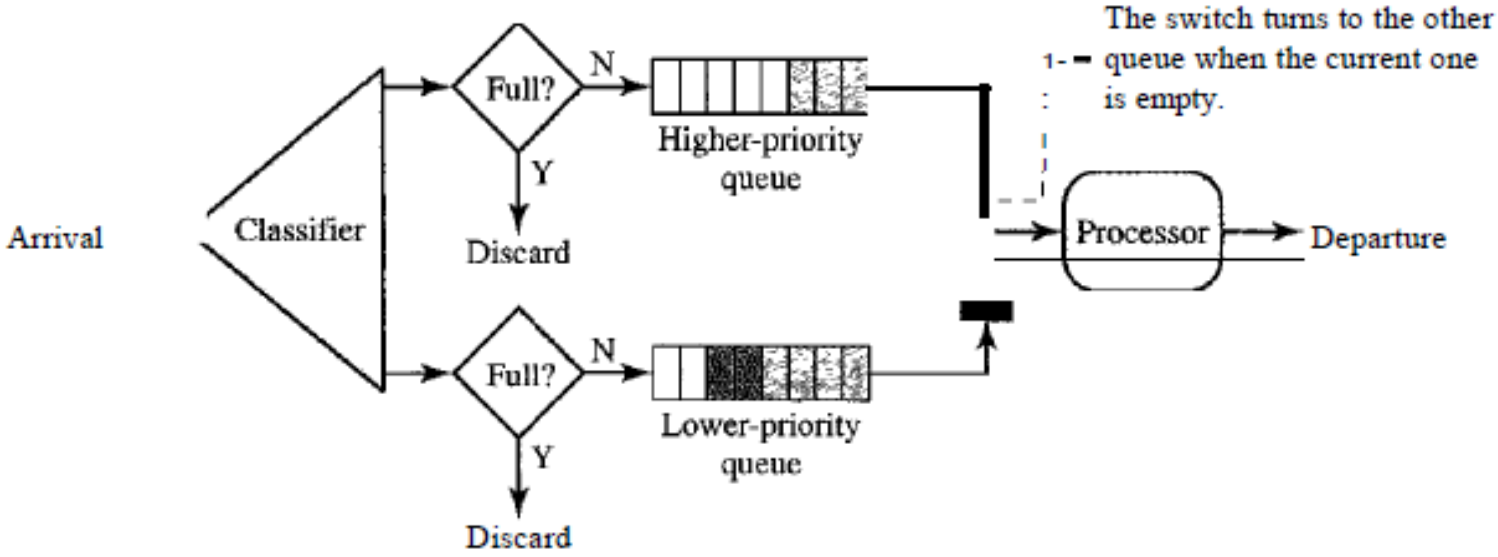
In first-in, first-out (FIFO) queuing, packets wait in a buffer (queue) until the node(router or switch) is ready to process them. If the average arrival rate is higher than the average processing rate, the queue will fill up and new packets will be discarded.



Congestion Control and Quality of Service

Priority Queuing

In priority queuing, packets are first assigned to a priority class. Each priority class has its own queue. The packets in the highest-priority queue are processed first. Packets in the lowest-priority queue are processed last. Note that the system does not stop serving a queue until it is empty. Following figure shows priority queuing with two priority levels.



A priority queue can provide better QoS than the FIFO queue because higher priority traffic, such as multimedia, can reach the destination with less delay.

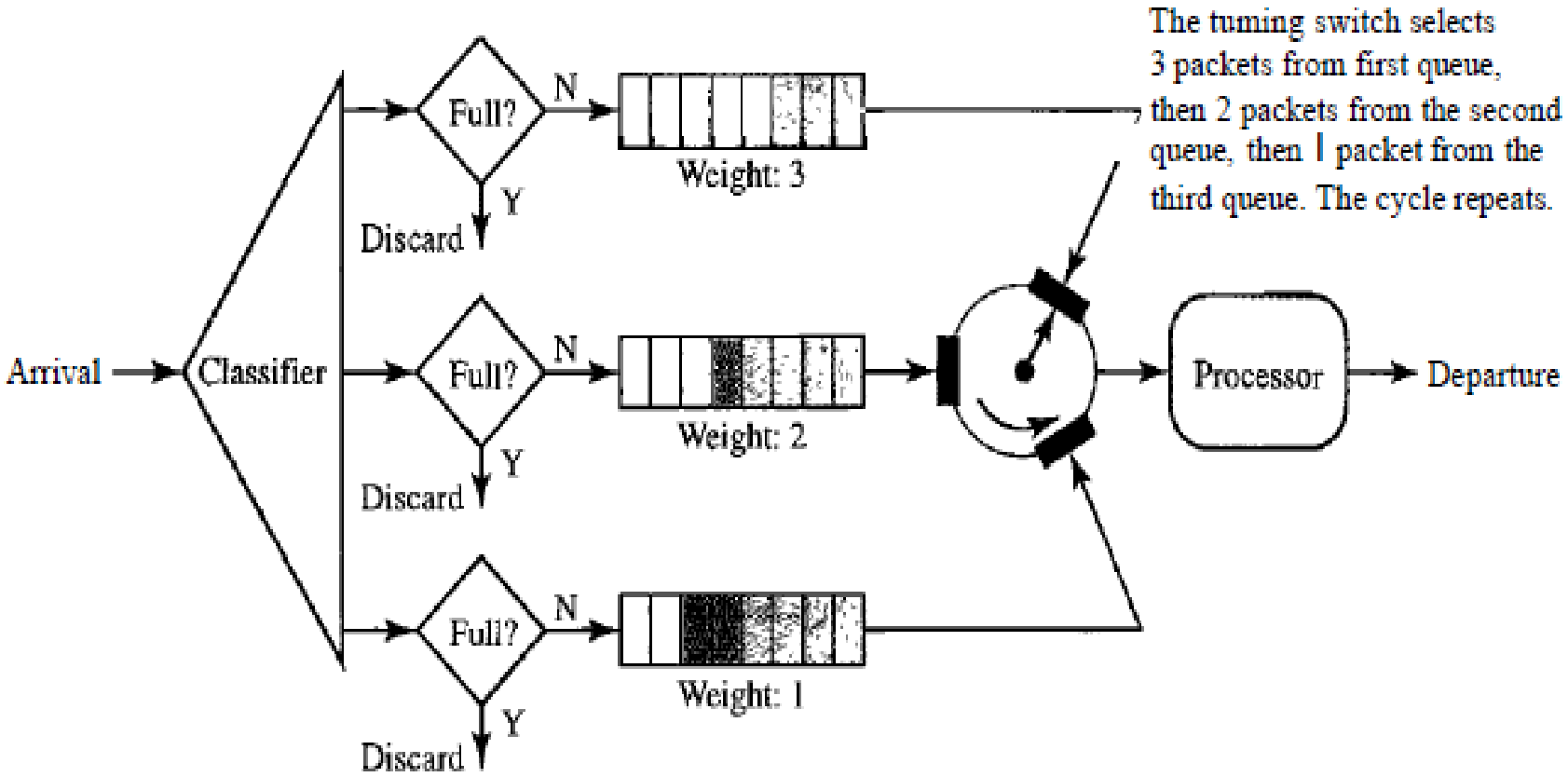
Congestion Control and Quality of Service

Weighted Fair Queuing

A better scheduling method is weighted fair queuing. In this technique, the packets are still assigned to different classes and admitted to different queues. The queues, however, are weighted based on the priority of the queues; higher priority means a higher weight.

The system processes packets in each queue in a round-robin fashion with the number of packets selected from each queue based on the corresponding weight. For example, if the weights are 3, 2, and 1, three packets are processed from the first queue, two from the second queue, and one from the third queue.

Congestion Control and Quality of Service



Congestion Control and Quality of Service

Traffic Shaping

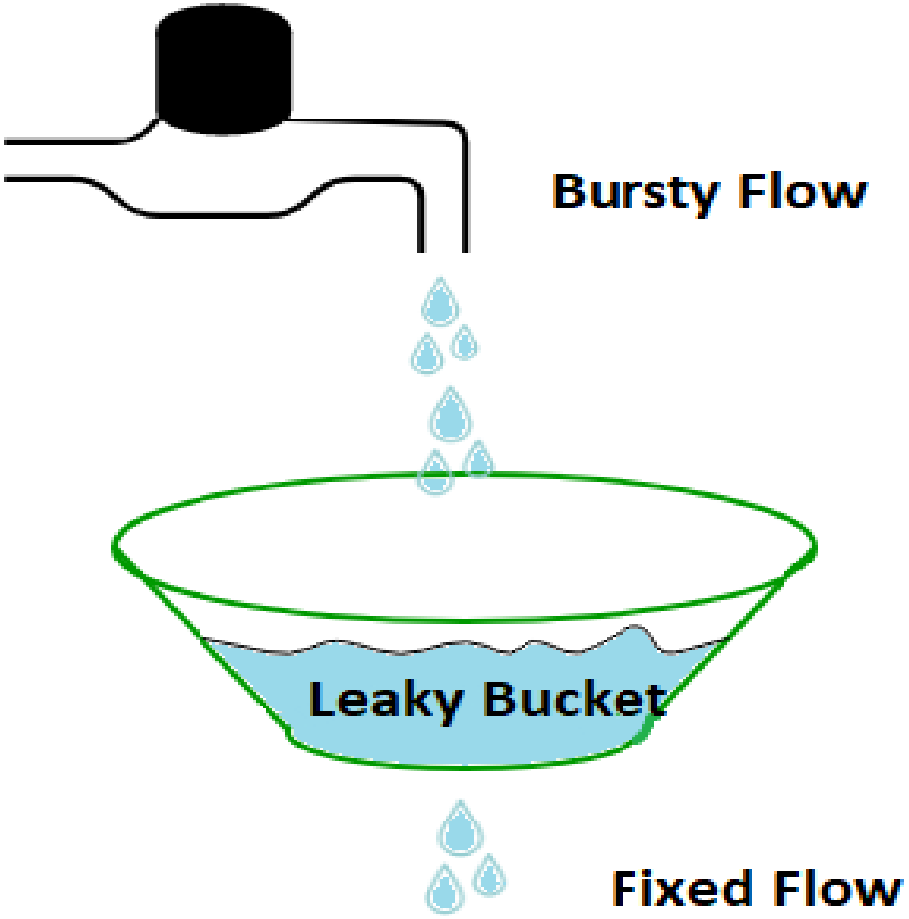
Traffic shaping is a mechanism to control the amount and the rate of the traffic sent to the network. Two techniques can shape traffic: leaky bucket and token bucket.

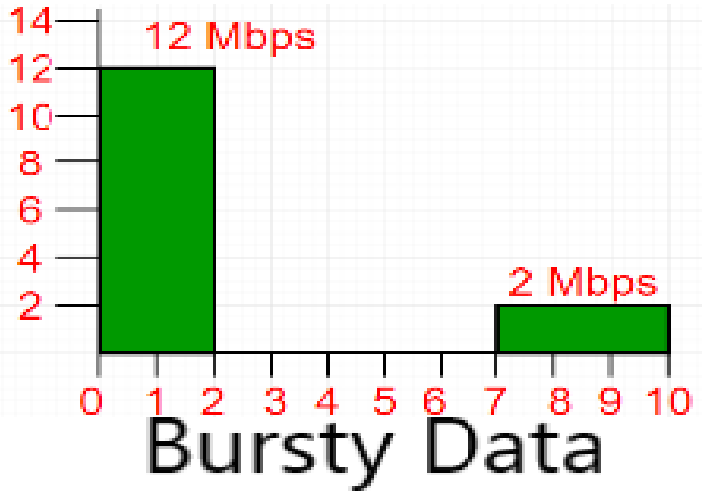
Leaky Bucket

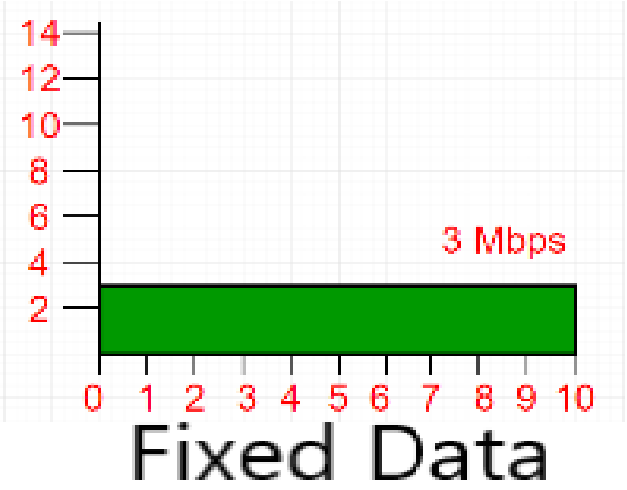
If a bucket has a small hole at the bottom, the water leaks from the bucket at a constant rate as long as there is water in the bucket. The rate at which the water leaks does not depend on the rate at which the water is input to the bucket unless the bucket is empty. The input rate can vary, but the output rate remains constant.

Similarly, in networking, leaky bucket technique can smooth out bursty traffic. Bursty chunks are stored in the bucket and sent out at an average rate. Following figure shows a leaky bucket and its effect.

Congestion Control and Quality of Service





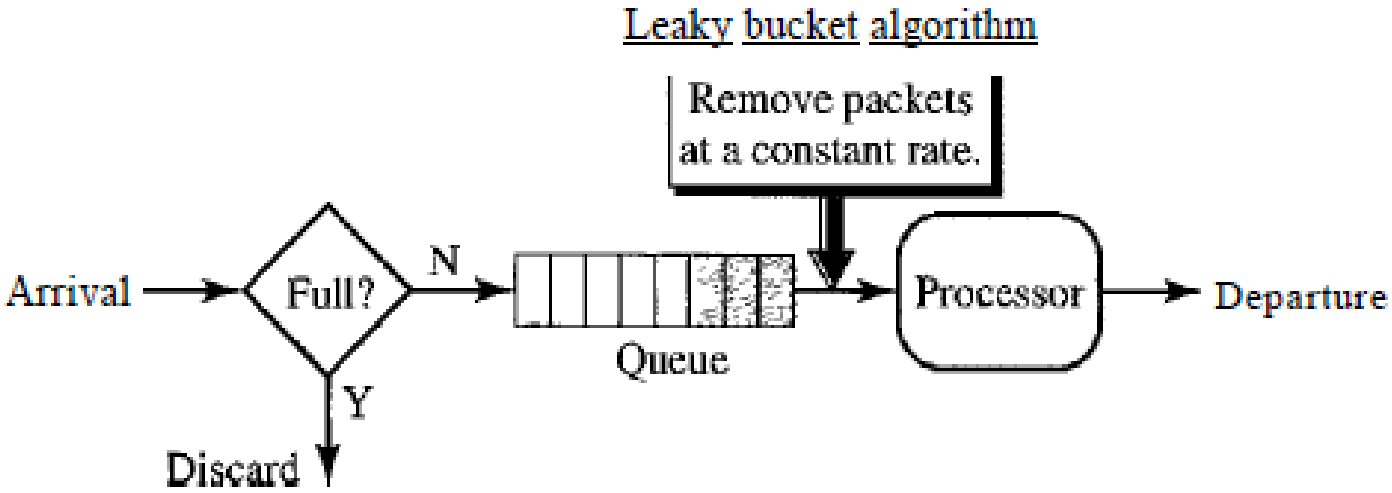


Congestion Control and Quality of Service

In this figure, the host sends a burst of data at a rate of 12 Mbps for 2s, for a total of 24 Mbits of data. The host is silent for 5s and then sends data at a rate of 2 Mbps for 3s, for a total of 6 Mbits of data. In all, the host has sent 30 Mbits of data in 10s. The leaky bucket smooths the traffic by sending out data at a rate of 3 Mbps during the same 10s.

Note: Leaky bucket may prevent congestion.

Leaky bucket implementation



Congestion Control and Quality of Service

In above figure, FIFO queue holds the packets. If the traffic consists of fixed-size packets, then the process removes a fixed number of packets from the queue at each tick of the clock. If the traffic consists of variable-length packets, the fixed output rate must be based on the number of bytes or bits.

The following is an algorithm for variable-length packets:

1. Initialize a counter to n at the tick of the clock.
2. If n is greater than the size of the packet, send the packet and decrement the counter by the packet size. Repeat this step until n is smaller than the packet size.
3. Reset the counter and go to step 1.

Congestion Control and Quality of Service

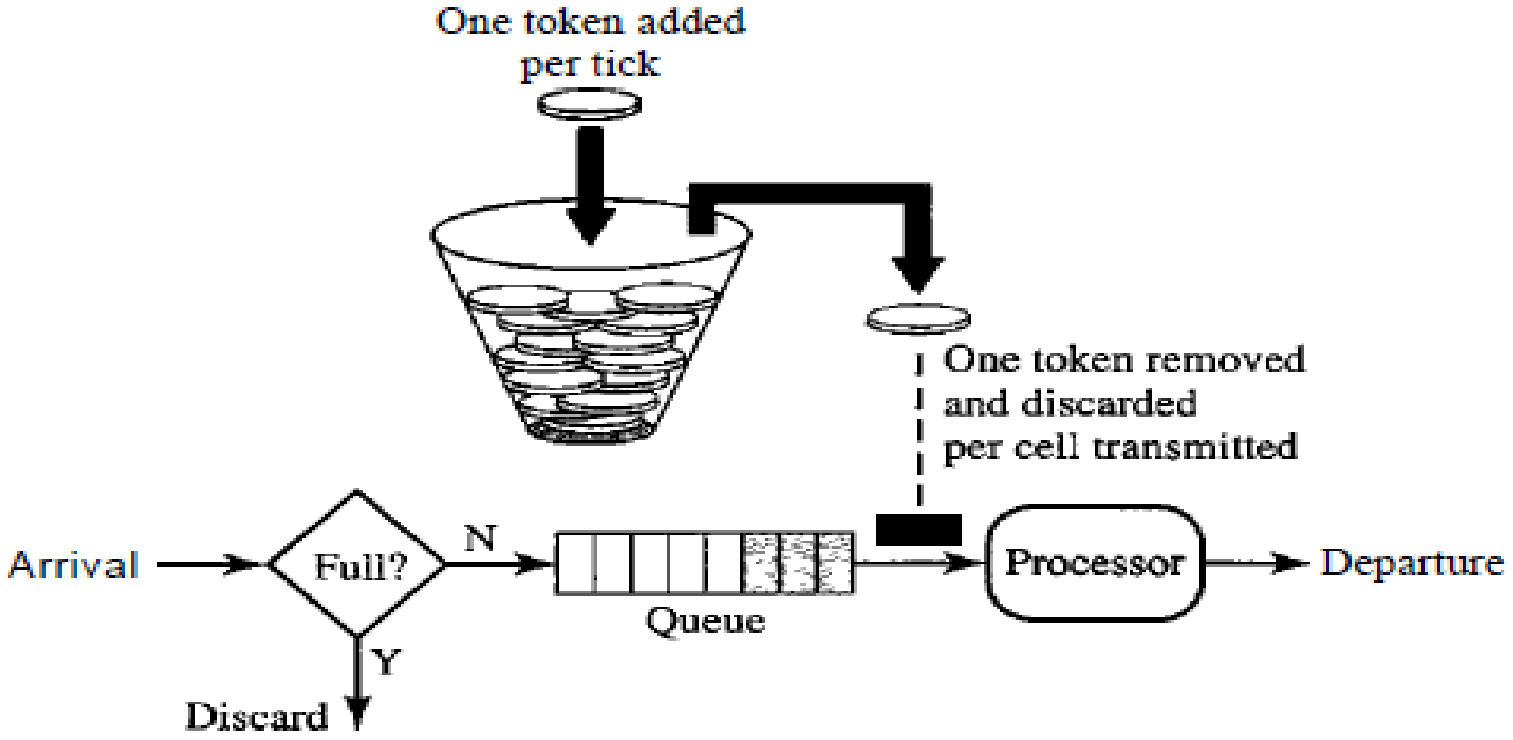
Token Bucket

The leaky bucket is very restrictive. It does not credit an idle host. For example, if a host is not sending for a while, its bucket becomes empty. Now if the host has bursty data, the leaky bucket allows only an average rate. The time when the host was idle is not taken into account.

On the other hand, the token bucket algorithm allows idle hosts to accumulate credit for the future in the form of tokens. For each tick of the clock, the system sends n tokens to the bucket. The system removes one token for every cell (or byte) of data sent. For example, if n is 100 and the host is idle for 100 ticks, the bucket collects 10,000 tokens. Now the host can consume all these tokens in one tick with 10,000 cells, or the host takes 1000 ticks with 10 cells per tick. In other words, the host can send bursty data as long as the bucket is not empty.

Congestion Control and Quality of Service

It is shown in the following figure:-



The token bucket can easily be implemented with a counter. The token is initialized to zero. Each time a token is added, the counter is incremented by 1. Each time a unit of data is sent, the counter is decremented by 1. When the counter is zero, the host cannot send data.

Congestion Control and Quality of Service

AKTU Examination Questions

1. What are the services of Transport Layer?
2. Discuss TCP window management in detail. Also explain silly window syndrome and their solution.
3. What is Congestion? Differentiate between congestion control and flow control with example. Also discuss congestion prevention policies.
4. Provide few reasons for congestion in a network.
5. How does transport layer perform duplication control?
6. What is congestion? Briefly describe the techniques that prevent congestion.
7. Enumerate on TCP header and working of TCP and differentiate TCP and UDP with frame format.

Process-to-Process Delivery

8. Enumerate how the transport layer unsure that the complete message arrives at the destination and in the proper order.
9. Explain the three way handshaking protocol to establish the transport level connection.
10. Explain about the TCP header and working of TCP protocol and differentiate between TCP and UDP with frame format.
11. The following is the dump of a TCP header in hexa decimal format:

    05320017 00000001 00000000 500207FF 00000000

    (i) What is the sequence number?

    (ii) What is the destination port number?

    (iii) What is the acknowledgment number?

    (iv) What is the window size?
12. What do you understand by Quality of service, parameters? List various Quality of service parameters.

Process-to-Process Delivery

13. How does transport layer perform duplication control?
14. Enumerate on TCP header and working of TCP and differentiate TCP and UDP with frame format.
15. Explain the three way handshaking protocol to establish the transport level connection.
16. Enumerate how the transport layer unsure that the complete message arrives at the destination and in the proper order.