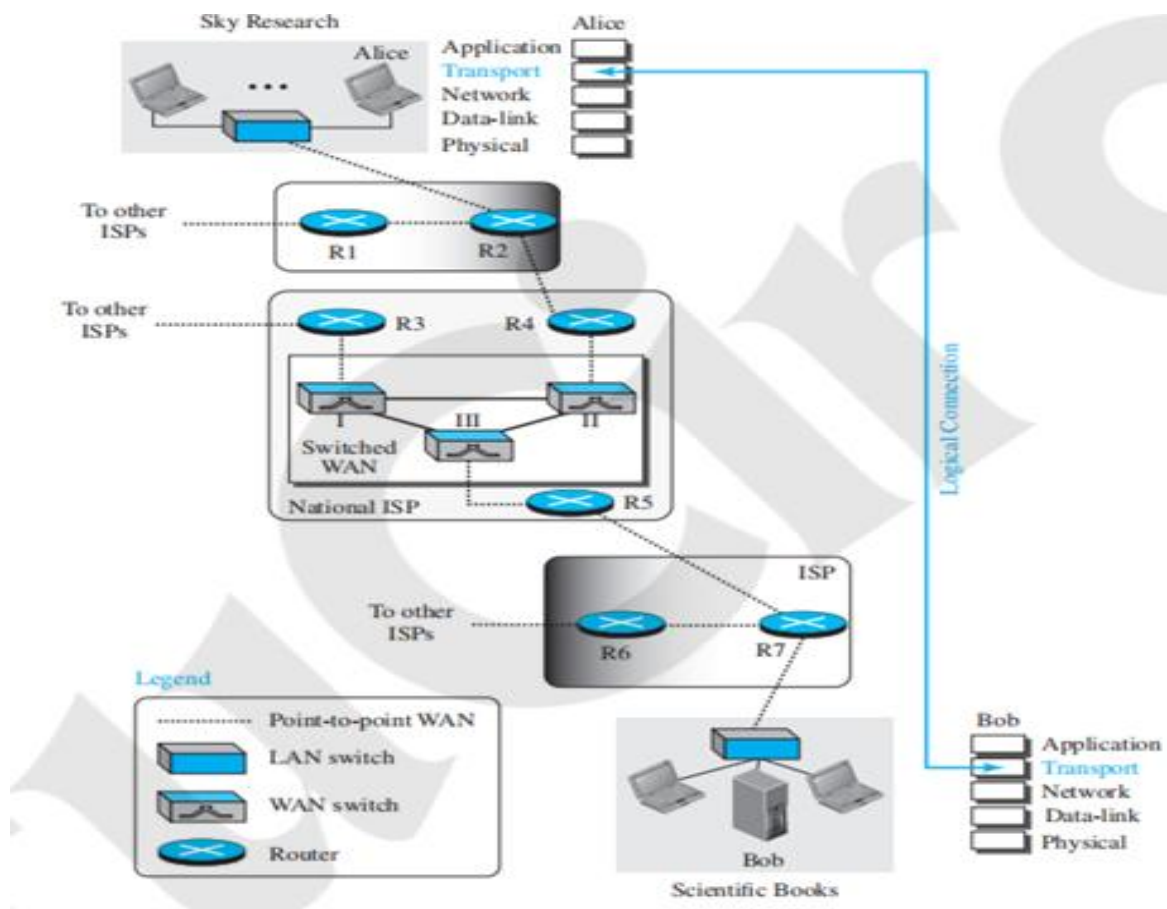


MODULE 4

TRANSPORT LAYER

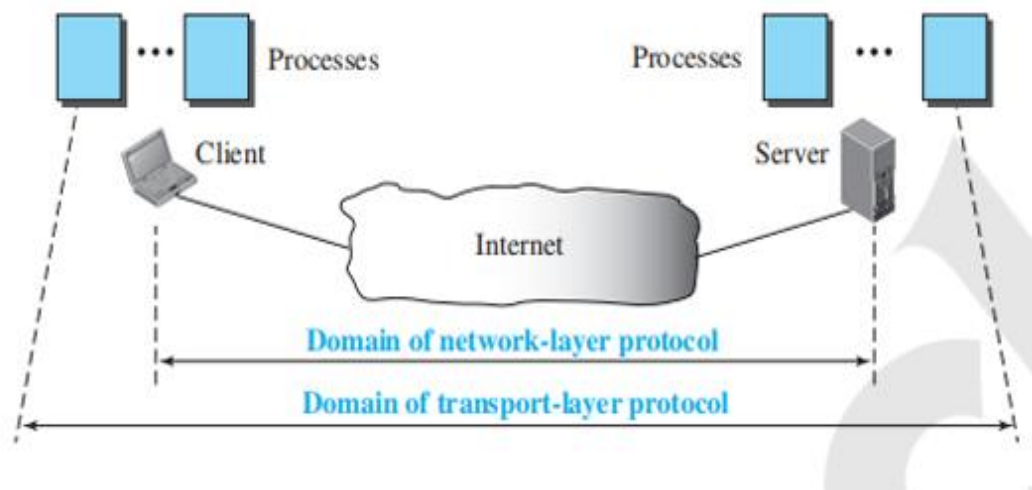
INTRODUCTION

- The transport layer is located between the application layer and the network layer.
- It provides a process-to-process communication between two application layers, one at the local host and the other at the remote host.
- Communication is provided using a logical connection, which means that the two application layers, which can be located in different parts of the globe, assume that there is an imaginary direct connection through which they can send and receive messages.



-Layer Services

- The transport layer provides services to the application layer.
- It receives necessary services from the network layer.
- Its main responsibility is to enable process-to-process communication.
- A process means a running program in the application layer.
- Transport layer helps one process on the source computer communicate with a process on the destination computer.
- It ensures end-to-end delivery between the correct processes.
- It makes sure data reaches the intended application (like browser, email, etc.).
- The network layer handles communication from one computer to another (host-to-host communication).
- It can deliver a message only up to the destination computer.
- This delivery is not complete because the message still needs to reach the correct process (program) inside that computer.
- The transport layer takes over after the network layer finishes its job.
- The transport layer is responsible for delivering the message to the appropriate process on the destination computer.



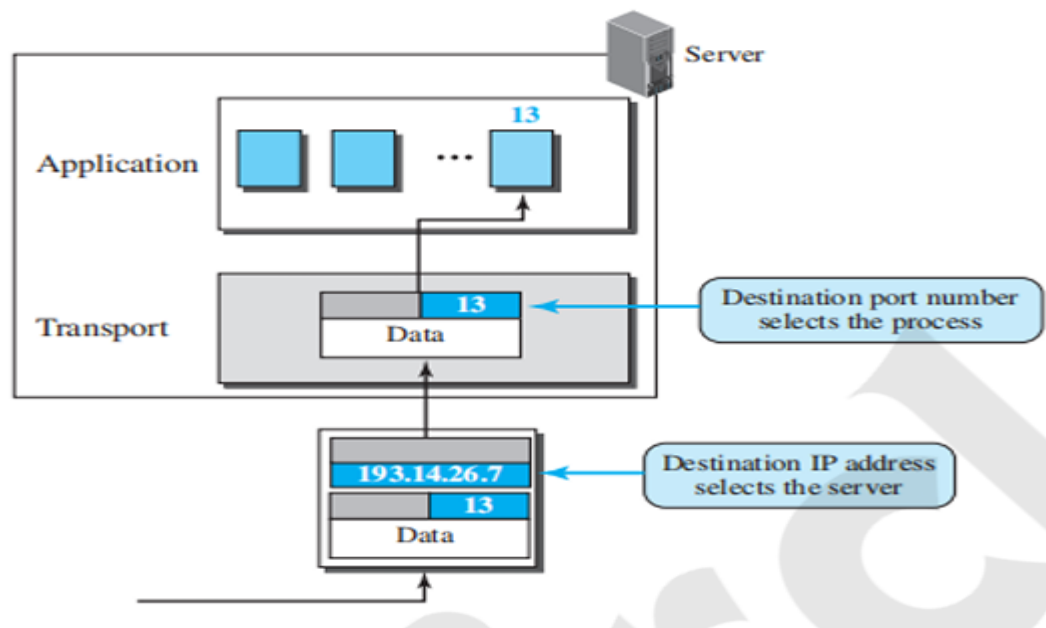
Addressing:

Port Numbers Although there are a few ways to achieve process-to-process communication, the most common is through the 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.

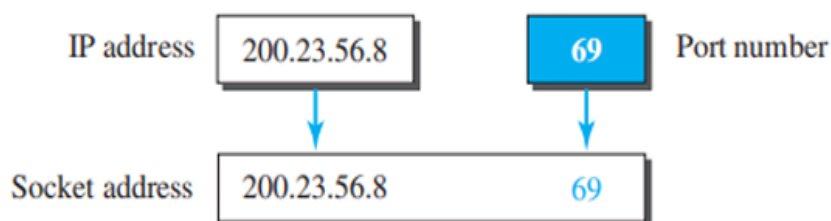
→ The operating systems today support both multiuser and multiprogramming environments. A remote computer can run multiple server programs at once, just like different local computers can each run one or more client programs at the same time.

→ For communication, we must define the local host, local process, remote host, and remote process. The local host and the remote host are defined using IP addresses. To define the processes, we need port numbers (0-65,535 (16 bits)).



Socket Addresses

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

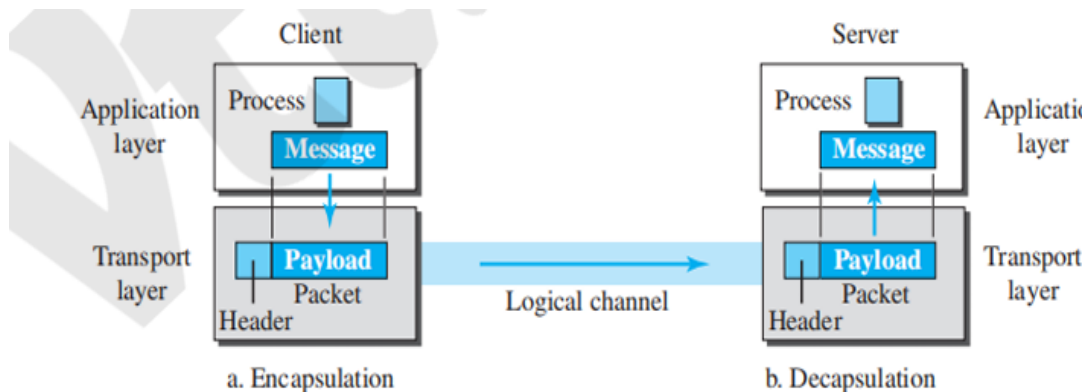


Encapsulation and Decapsulation

Encapsulation happens on the sender's side: When a program wants to send a message, it gives the message to the transport layer, along with the socket addresses and other necessary info. The transport layer then adds a header to the message, creating a packet.

Depending on the transport protocol, this packet might be called a user datagram, segment, or just a packet.

Decapsulation happens on the receiver's side: When the packet reaches its destination, the transport layer removes the header and delivers the message to the application program. The sender's socket address is also passed along so the program can respond if needed.

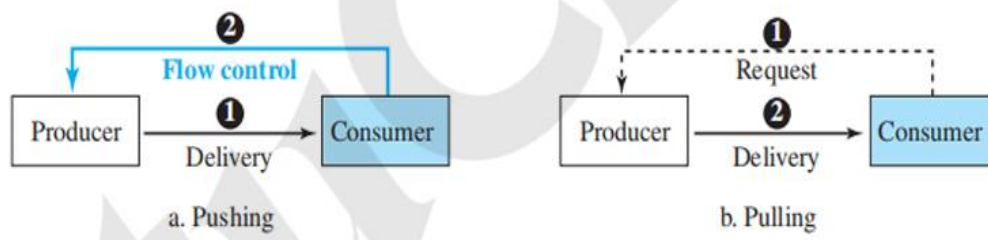


Multiplexing and Demultiplexing

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

Pushing or Pulling

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



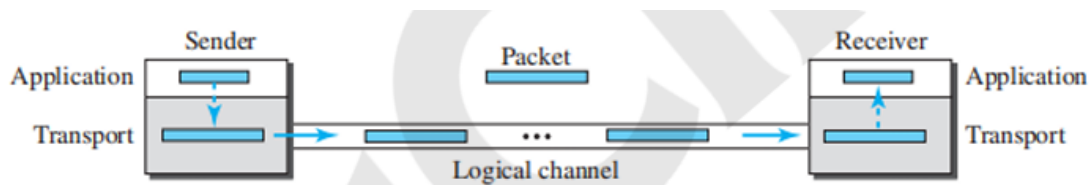
TRANSPORT-LAYER PROTOCOLS

There are 4 types of Transport layer Protocols they are:

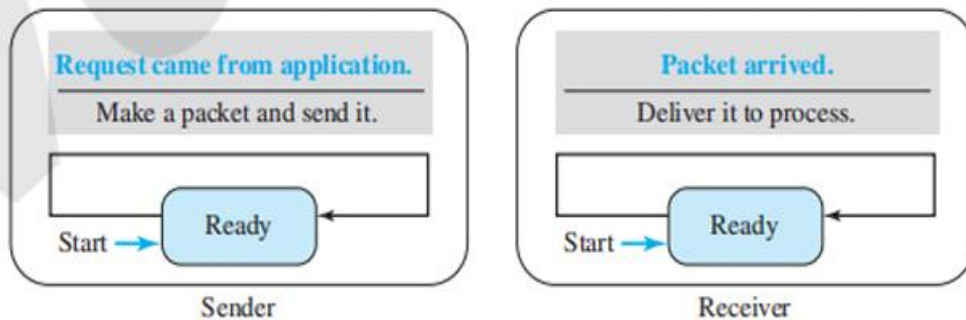
1. Simple Protocol
2. Stop and Wait Protocol
3. Go-Back-N Protocol (GBN)
4. Selective-Repeat Protocol

1. Simple Protocol

Our first protocol is a simple connectionless protocol with neither flow nor error control. We assume that the receiver can immediately handle any packet it receives.



- The transport layer at the sender receives a message from the application layer.
- It creates a packet from this message.
- It then sends the packet to the network layer.
- The transport layer at the receiver gets the packet from its network layer.
- It extracts the original message from the packet.
- Finally, it delivers the message to the application layer.

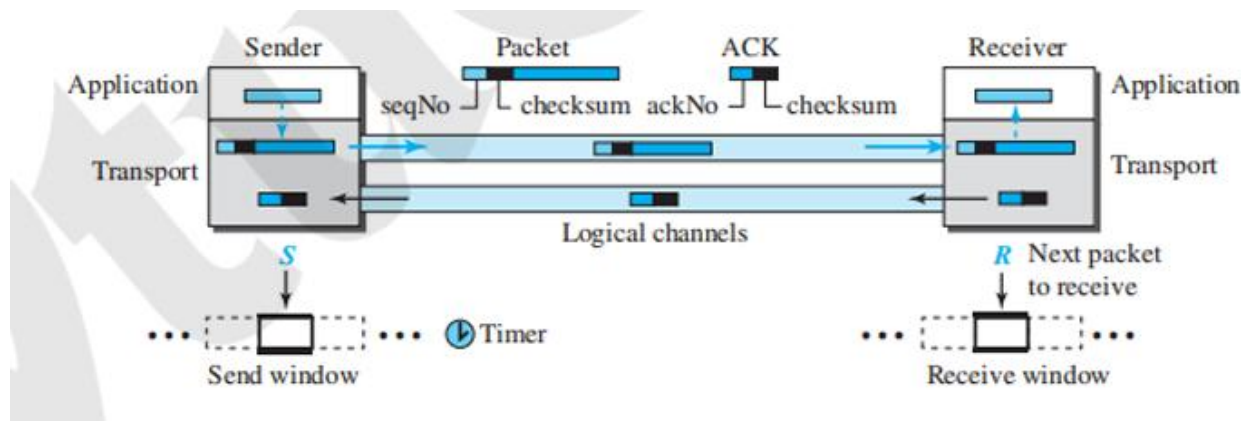
FSMs

- The sender sends a packet only when the application layer gives a message.
- The receiver delivers a message to its application layer only when a packet arrives.
- Two FSMs (Finite State Machines) are used to show this; each has only one state called **ready**.
- The sender stays in the ready state until it gets a message from the application layer; then it makes a packet and sends it.
- The receiver stays in the ready state until a packet arrives; then it decapsulates the message and delivers it to its application layer.

2. Stop-and-Wait Protocol

- Stop-and-Wait is a connection-oriented protocol that provides flow control and error control.
- Both sender and receiver use a sliding window of size 1.
- The sender sends one packet at a time and waits for an acknowledgment (ACK) before sending the next.
- Each packet has a checksum to detect corruption.
- If a packet arrives with a wrong checksum, the receiver discards it silently.
- Silence from the receiver indicates that the packet was lost or corrupted.
- After sending a packet, the sender starts a timer.
- If the ACK comes before the timer expires, the sender stops the timer and sends the next packet.
- If the timer expires, the sender retransmits the last packet, assuming it was lost or corrupted.

- The sender must keep a copy of the last packet until it receives its acknowledgment.



Sequence Number

- Acknowledgment numbers indicate the next packet the receiver is expecting.
- Because the protocol is connection-oriented, both sides must be in the established state before sending data.
- The sender's and receiver's states work inside this established state.

Sender

- Sender starts in ready state with $S = 0$.
- Ready state:
 - Waits for a message from the application layer.
 - When a message arrives, the sender makes a packet with sequence number S , saves a copy, sends it, starts the timer, and moves to blocking state.
- Blocking state: Three things can happen:
 - Correct ACK arrives with $\text{ackNo} = (S + 1) \bmod 2$
 - Stop the timer
 - Update $S = (S + 1) \bmod 2$
 - Move to ready state
 - Wrong or corrupted ACK arrives

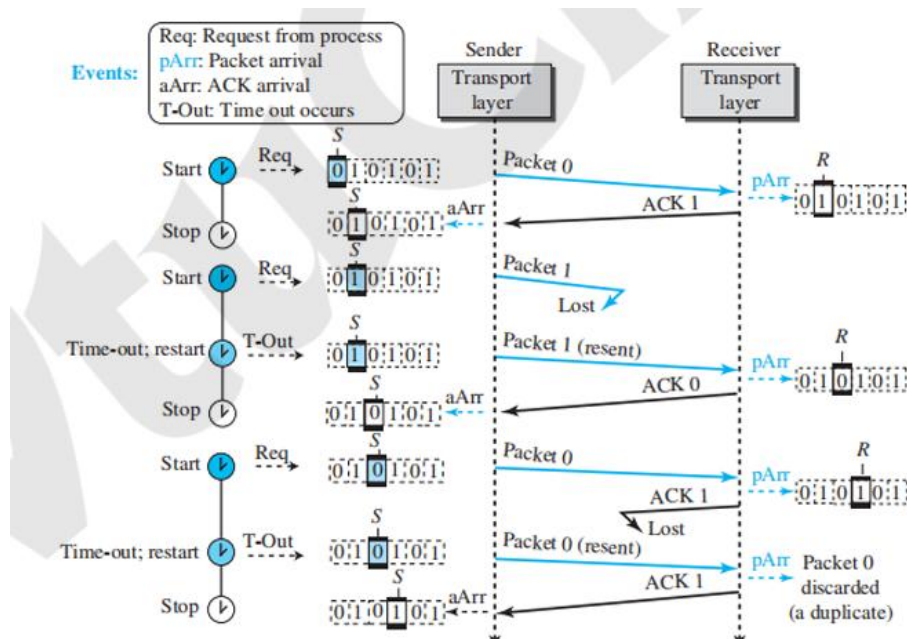
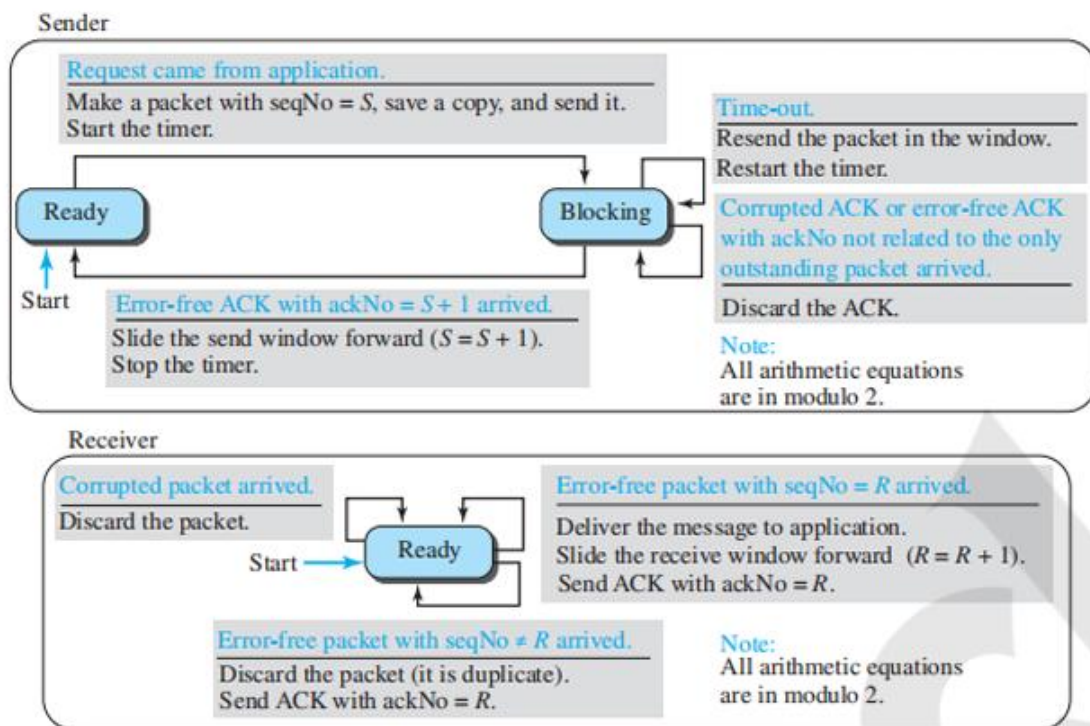
- Discard the ACK

3. Timeout occurs

- Resend the stored packet
- Restart the timer

Receiver

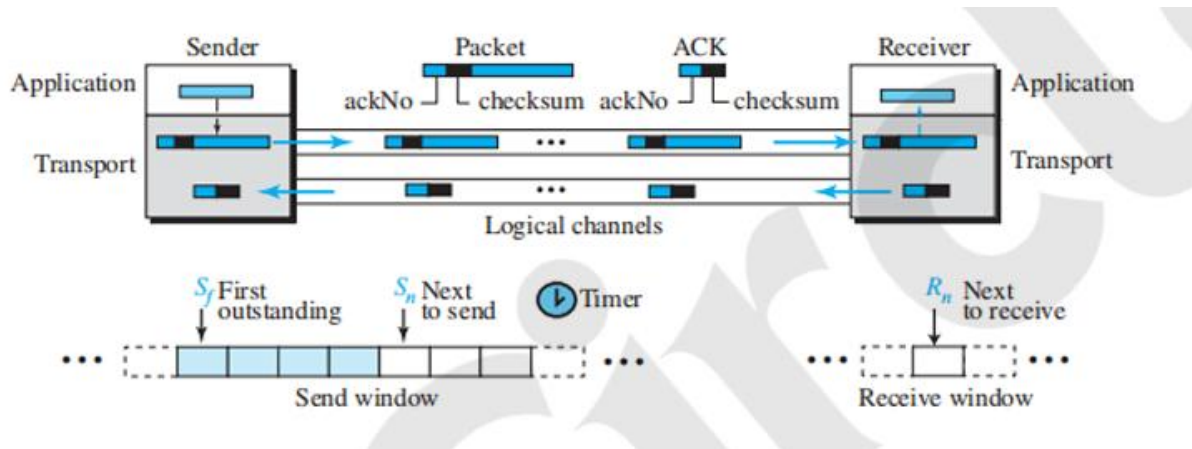
- Receiver always stays in ready state.
- Three possibilities:
 1. Correct packet arrives with $\text{seqNo} = R$
 - Deliver message to application layer
 - Update $R = (R + 1) \bmod 2$
 - Send ACK with $\text{ackNo} = R$
 2. Correct packet but wrong sequence number ($\text{seqNo} \neq R$)
 - Discard packet
 - Send ACK with $\text{ackNo} = R$ again
 3. Corrupted packet arrives
 - Discard packet (no delivery)



3.Go-Back-N Protocol (GBN)

The key to Go-back-N is that we can send several packets before receiving acknowledgments, but the receiver can only buffer one packet. We keep a copy of the sent packets until the

acknowledgments arrive. Figure shows the outline of the protocol. Note that several data packets and acknowledgments can be in the channel at the same time.



Sequence Numbers

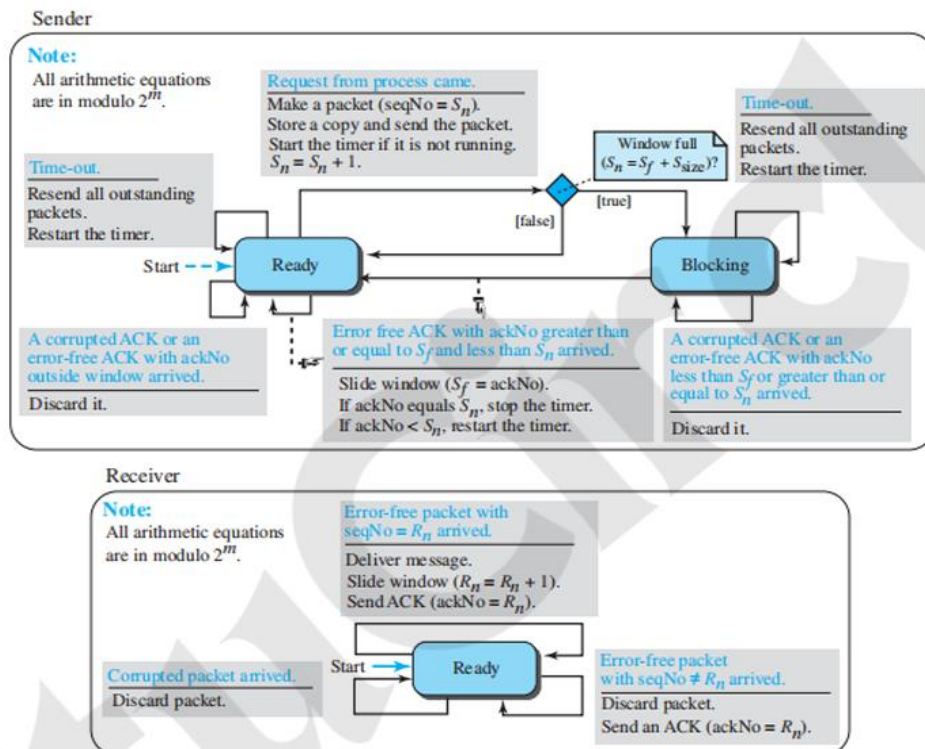
The sequence numbers are modulo $2m$, where m is the size of the sequence number field in bits.

Acknowledgment Numbers

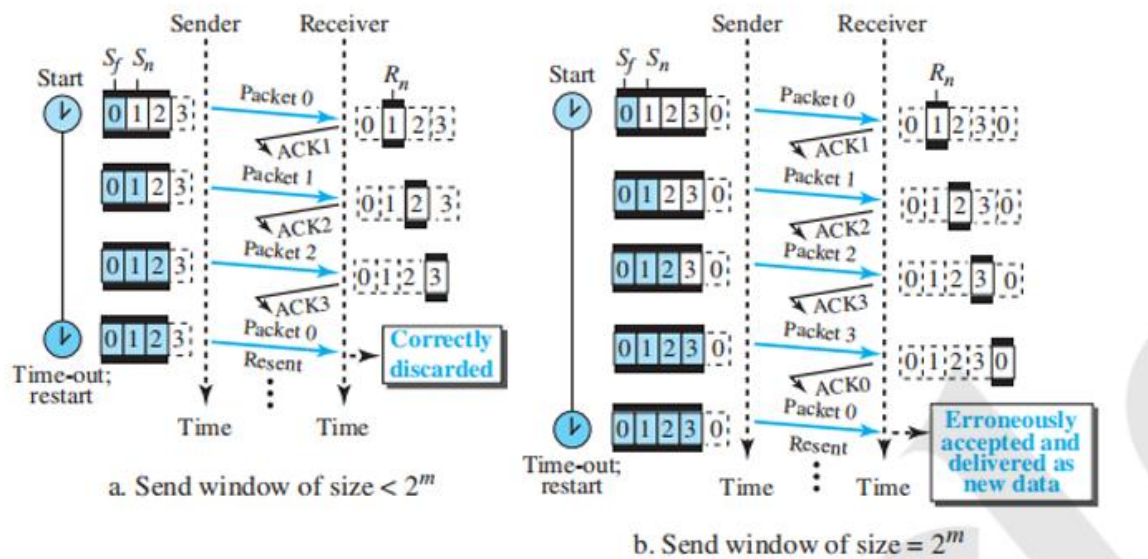
An acknowledgment number in this protocol is cumulative and defines the sequence number of the next packet expected. For example, if the acknowledgment number (ackNo) is 7, it means all packets with sequence number up to 6 have arrived, safe and sound, and the receiver is expecting the packet with sequence number 7.

Send Window

The send window is an imaginary box covering the sequence numbers of the data packets that can be in transit or can be sent. In each window position, some of these sequence numbers define the packets that have been sent; others define those that can be sent. The maximum size of the window is $2m - 1$.

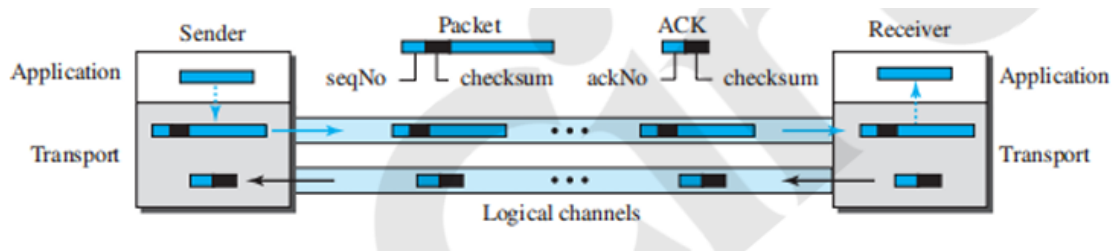
FSMs**Send Window Size**

We can now show why the size of the send window must be less than $2m$. As an example, we choose $m = 2$, which means the size of the window can be $2m - 1$, or 3.



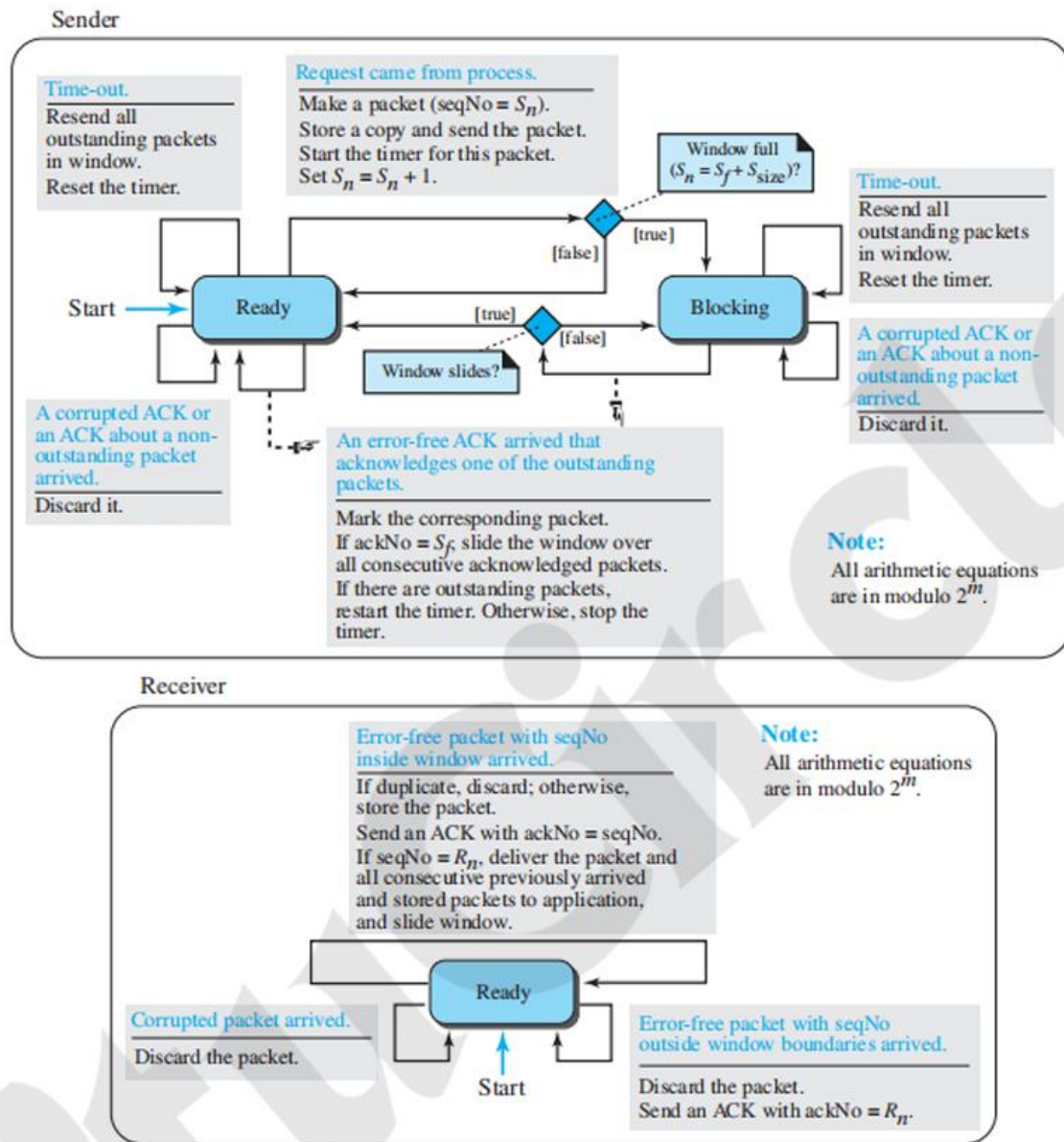
4. Selective-Repeat Protocol

Selective-Repeat (SR) protocol, as the name implies, resends only selective packets, those that are actually lost.



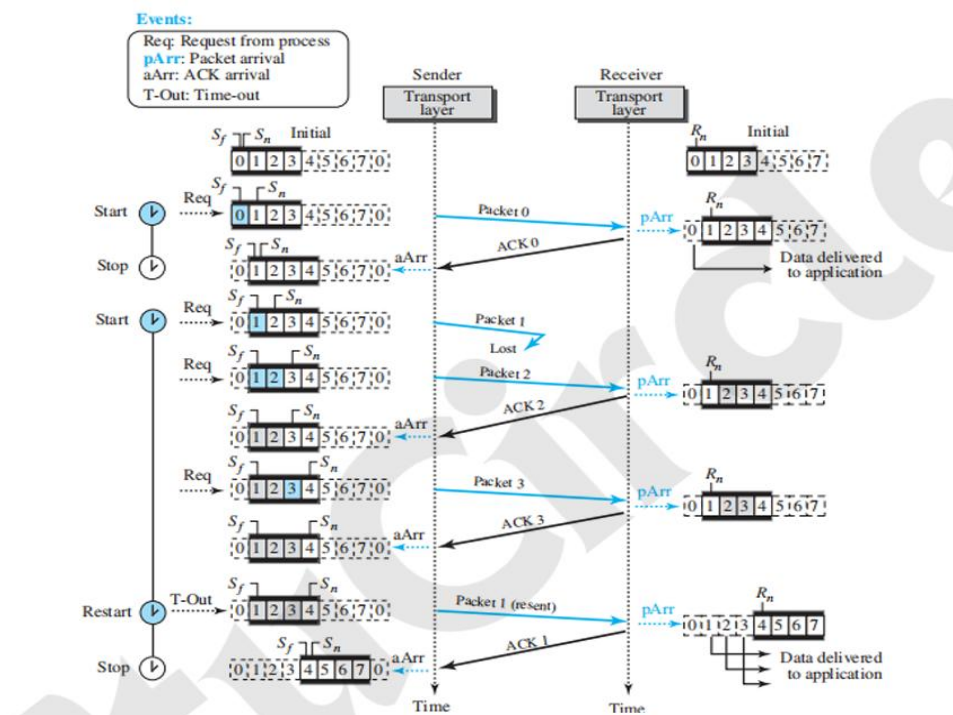
Acknowledgments

There is yet another difference between the two protocols. In GBN an ackNo is cumulative; it defines the sequence number of the next packet expected, confirming that all previous packets have been received safe and sound. The semantics of acknowledgment is different in SR. In SR, an ackNo defines the sequence number of a single packet that is received safe and sound; there is no feedback for any other.

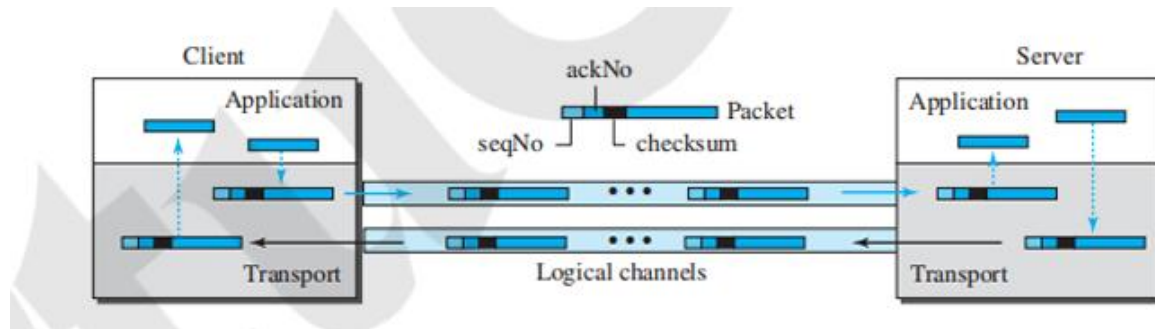


Example

In this example packet 1 is lost. We show how Selective-Repeat behaves in this case. Figure shows the situation.

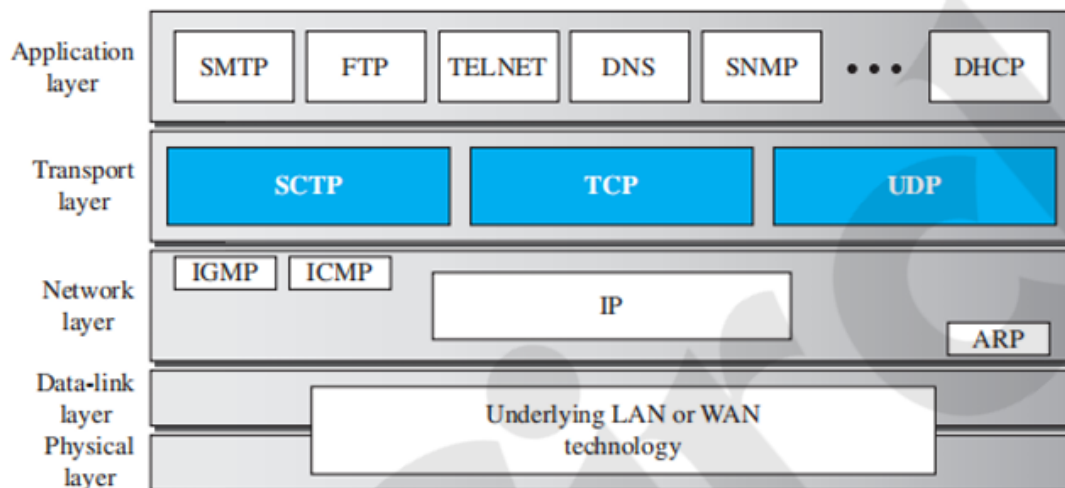
**Bidirectional Protocols:**

- The earlier protocols are unidirectional, meaning data flows only one way and acknowledgments flow the opposite way.
- In real communication, data usually flows both ways (client → server and server → client).
- Therefore, acknowledgments also need to flow in both directions.
- Piggybacking is used to make bidirectional communication more efficient.
- A data packet going from A to B can also carry the ACK for packets received from B.
- Similarly, a packet going from B to A can carry the ACK for packets received from A.
- In the bidirectional GBN protocol with piggybacking, both client and server use two windows:
 - A send window
 - A receive window



Transport-Layer Protocols Introduction (Message Format)

- In the TCP/IP model, the transport layer lies between the application layer and the network layer.
- It provides services to the application layer (programs like browser, email, etc.).
- It receives services from the network layer (which handles routing and delivery between hosts).
- The transport layer works as a bridge between a client process and a server process.
- It provides process-to-process communication.
- It is the core (heart) of the TCP/IP suite.
- It offers end-to-end logical communication for transferring data across the Internet.



Services

Each protocol provides a different type of service and should be used appropriately.

UDP

UDP is an unreliable connectionless transport-layer protocol used for its simplicity and efficiency in applications where error control can be provided by the application-layer process.

TCP

TCP is a reliable connection-oriented protocol that can be used in any application where reliability is important.

SCTP

SCTP is a new transport-layer protocol that combines the features of UDP and TCP.

Port Numbers

One of the responsibilities of transport layer is to create a process-to-process communication; these protocols use port numbers to accomplish this. Port numbers provide end-to-end addresses at the transport layer and allow multiplexing and demultiplexing at this layer, just as IP addresses do at the network layer. Table gives some common port numbers for all three protocols.

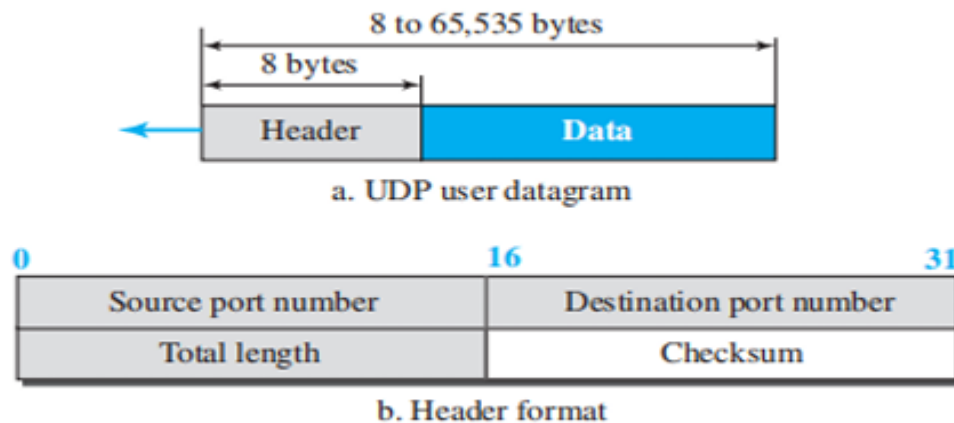
USER DATAGRAM PROTOCOL

- UDP (User Datagram Protocol) is a connectionless and unreliable transport protocol.
- It is very simple and uses minimum overhead.
- A process can use UDP when it wants to send small messages and does not require high reliability.
- Sending data through UDP needs less interaction between sender and receiver compared to TCP.

User Datagram (Packet) Format:

- UDP packets are called user datagrams.
- Each user datagram has a fixed 8-byte header.
- The header has four fields, each 2 bytes (16 bits).
- Source port number (2 bytes).

- Destination port number (2 bytes).
- Total length of the datagram (header + data), ranging from 0 to 65,535 bytes.
- Actual size is smaller because it must fit inside an IP datagram (max 65,535 bytes).
- The last field is an optional checksum for error detection.



UDP Services

Process-to-Process Communication

UDP provides process-to-process communication using socket addresses, a combination of IP addresses and port numbers.

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. There is no connection establishment and no connection termination. This means that each user datagram can travel on a different path.

Flow Control

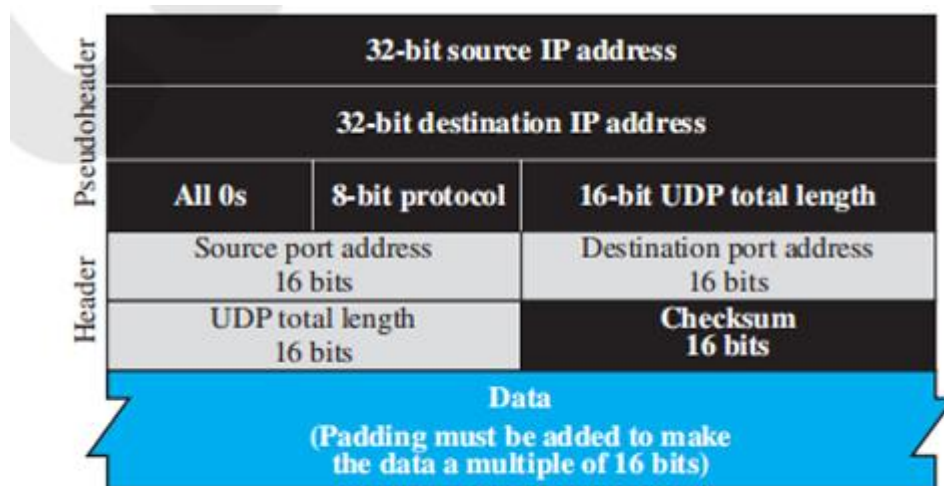
UDP is a very simple protocol. There is no flow control, and hence no window mechanism. The receiver may overflow with incoming messages. The lack of flow control means that the process using UDP should provide for this service, if needed.

Error Control

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.

Checksum

UDP checksum calculation includes three sections: a pseudo header, the UDP header, and the data coming from the application layer. The pseudo header is the part of the header of the IP packet in which the user datagram is to be encapsulated with some fields filled with 0s.



TRANSMISSION CONTROL PROTOCOL

Transmission Control Protocol (TCP) is a connection-oriented, reliable protocol. TCP explicitly defines connection establishment, data transfer, and connection tear down phases to provide a connection-oriented service. TCP uses a combination of GBN and SR 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.

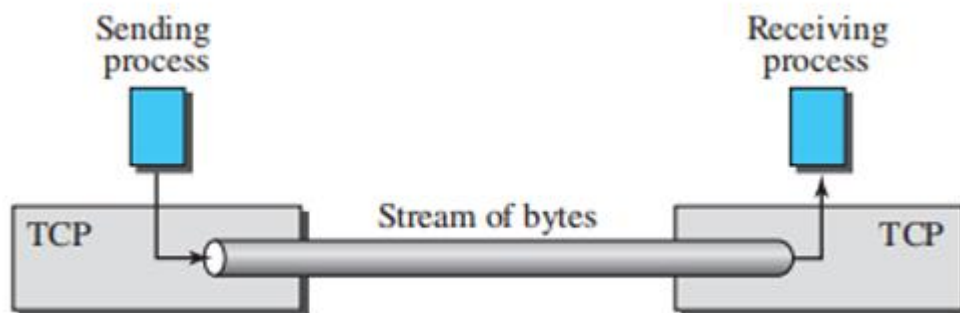
TCP SERVICES

1. Process-to-Process Communication

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

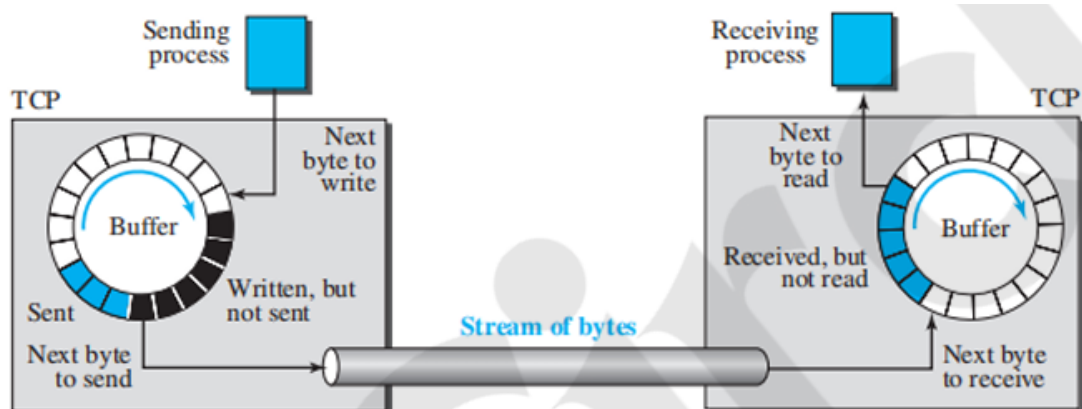
2. Stream Delivery Service

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 bytes across the Internet. This imaginary environment is depicted in Figure. The sending process produces (writes to) the stream and the receiving process consumes (reads from) it.



3. 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.



- TCP uses buffers for flow control and error control.
- A buffer is usually a circular array of many bytes (hundreds or thousands).
- The sender's buffer has three parts:

- Empty space (white) where new data from the application can be written.
- Sent but not yet acknowledged data (colored), kept until an ACK is received.
- Data ready to send (shaded), but TCP may send only a part depending on network or receiver speed.
- When data is acknowledged, its space in the buffer becomes free again—this is why a circular buffer is useful.
- The receiver's buffer has two parts:
 - Empty chambers (white) for incoming data.
 - Filled chambers (colored) with received bytes.
- Segment sizes vary in examples, but in real TCP, segments usually carry hundreds or thousands of bytes.

TCP FEATURES

Numbering System Although the TCP software keeps track of the segments being transmitted or received, there is no field for a segment number value in the segment header. Instead, there are two fields, called the sequence number and the acknowledgment number. These two fields refer to a byte number and not a segment number

1. **Byte Number**- TCP numbers all data bytes transmitted in a connection, with independent numbering in each direction. The numbering doesn't start from 0 but from an arbitrary number between 0 and $2^{32} - 1$. For example, if the starting number is 1057 and 6000 bytes are to be sent, the bytes will be numbered from 1057 to 7056. This numbering is used for flow and error control.

2. **Sequence Number** -After the bytes have been numbered, TCP assigns a sequence number to each segment that is being sent. The sequence number, in each direction, is defined as follows: 1. The sequence number of the first segment is the ISN (initial sequence number), which is a random number. 2. The sequence number of any other segment is the sequence number of the previous segment plus the number of bytes (real or imaginary) carried by the previous segment. Later, we show that some control segments are thought of as carrying one imaginary byte.

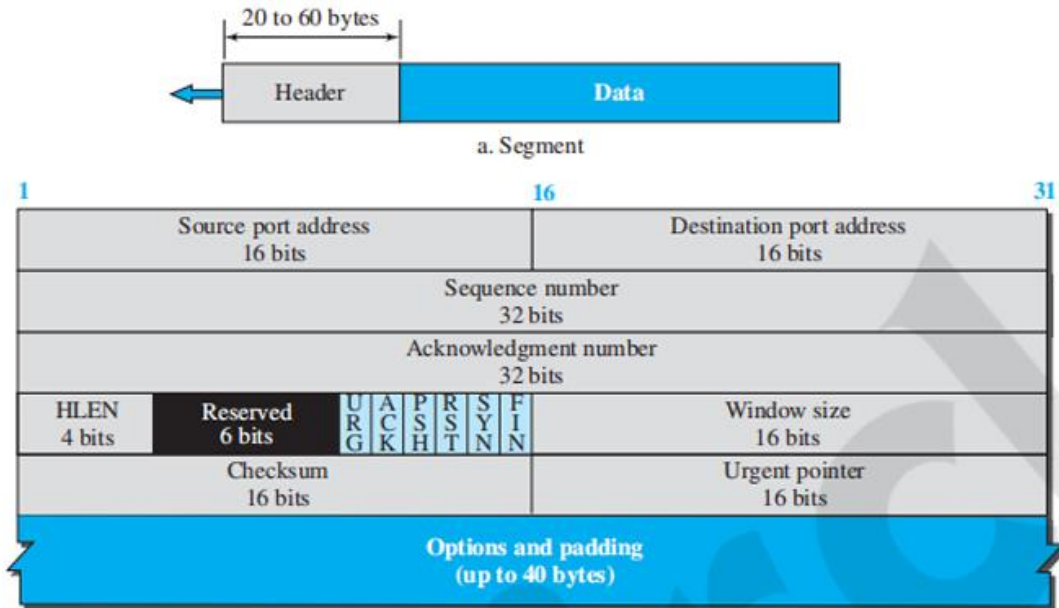
3. Acknowledgment Number - In TCP's full duplex communication, both parties can send and receive data simultaneously. Each party numbers the bytes, typically starting with different numbers. The sequence number in each direction indicates the first byte in the segment, while the acknowledgment number confirms receipt of bytes and specifies the next expected byte. The acknowledgment number is cumulative, meaning if it's 5643, all bytes up to 5642 have been received, though the first byte may not be 0.

SEGMENT

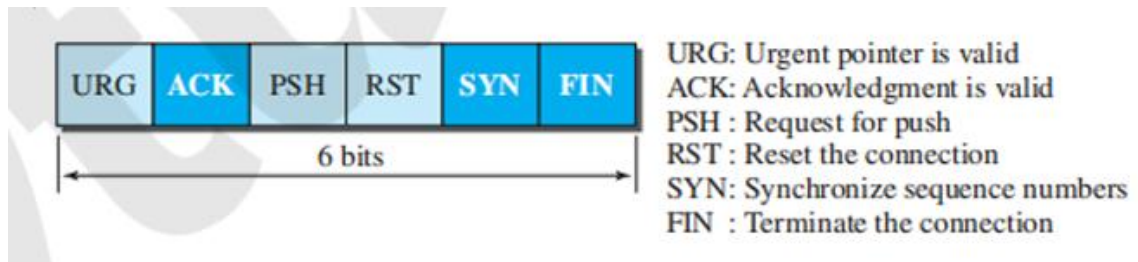
A packet in TCP is called a segment.

Format The format of a segment is shown in Figure . The segment consists of a header of 20 to 60 bytes, 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. TCP is a stream transport protocol. To ensure connectivity, each byte to be transmitted is numbered. The sequence number tells the destination which byte in this sequence is the first byte in the segment. During connection establishment each party uses a random number generator to create an initial sequence number (ISN), which is usually different in each direction.



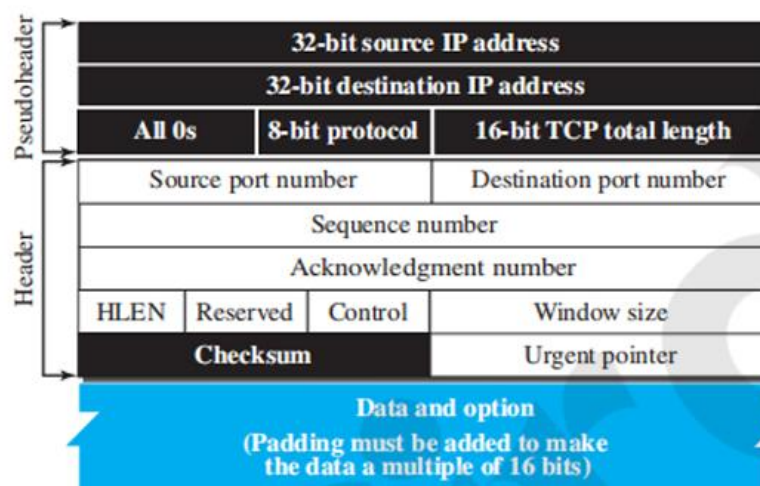
- Acknowledgment number (32 bits):
- Shows the next byte the receiver expects.
- If the receiver gets byte x , it sends back $x + 1$ as the acknowledgment number.
- ACK information can be piggybacked with data.
- Header length (4 bits):
- Indicates the length of the TCP header in 4-byte words.
- Header size ranges from 20 to 60 bytes.
- So this field's value is between 5 and 15.
- Control field:
- Contains 6 control bits (flags).
- One or more flags may be ON at the same time.
- These flags handle flow control, connection setup, connection termination, aborting connections, and data transfer modes in TCP.



Window size. This field defines the window size of the sending TCP in bytes. 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. 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. The same pseudo header, serving the same purpose, is added to the segment. For the TCP pseudo header, the value for the protocol field.



TCP CONNECTION

- TCP is a connection-oriented protocol.

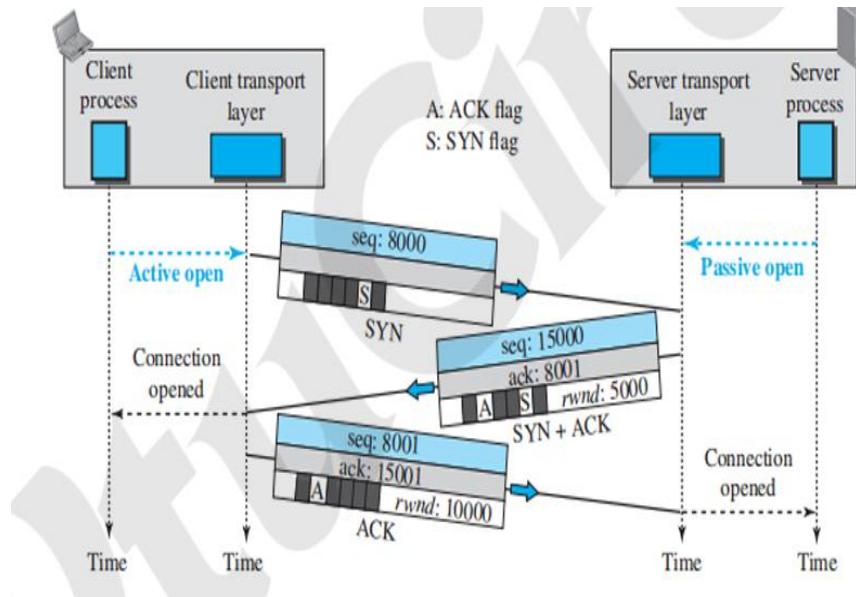
- It creates a logical connection between sender and receiver before data transfer begins.
- All segments of a message are sent through this same logical path.
- Using one logical path makes acknowledgment and retransmission easier.
- The connection is logical, not physical (no actual wires or dedicated path).
- TCP works above IP; IP is connectionless and only delivers individual segments.
- If a segment is lost or damaged, TCP retransmits it (IP does not).
- If segments arrive out of order, TCP stores them and waits for the missing ones.
- IP does not handle ordering or retransmission; TCP manages these tasks.
- 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.

Three-Way Handshaking

- TCP uses three-way handshaking to establish a connection.
- A client wants to connect to a server using TCP.
- The server first performs a passive open, telling TCP it is ready to accept connections.
- The server cannot start a connection on its own; it only waits.
- The client performs an active open, telling its TCP to connect to the server.
- After this, TCP begins the three-step handshake between client and server to establish the connection.



- **Step 1 – Client Sends SYN**

- The client sends a SYN segment to the server.
- Only the SYN flag is set.
- This segment is used to synchronize sequence numbers.
- The client includes its Initial Sequence Number (ISN).
- No acknowledgment number is included.
- SYN is a control segment and carries no data.

- **Step 2 – Server Sends SYN + ACK**

- The server replies with a SYN + ACK segment.
- Both SYN and ACK flags are set.
- It serves two purposes:
 - Sends the server's own SYN to establish communication in the reverse direction.
 - Sends an ACK to confirm the client's SYN.

- The server also includes its own ISN for synchronization.
- The server includes its own **sequence number** to number the bytes it will send to the client.
- It also **acknowledges** the client's SYN by setting the **ACK flag**.
- The acknowledgment number shows the **next byte** the server expects from the client.
- Since this segment contains an acknowledgment, it must include the **acknowledgment number field**.

- **Step 3 – Client Sends ACK**

- The client sends a final ACK segment.
- Only the ACK flag is set.
- It acknowledges the server's SYN.
- After this, the connection is established, and data transfer can begin.
- This segment acknowledges the server's SYN.
- Only the **ACK flag** is set.
- The acknowledgment number confirms that the second segment from the server was received.

Flow Control

- Flow control keeps the **sender's data rate** and the **receiver's processing rate** balanced.
- In TCP, **flow control** is handled separately from **error control**.

Opening and Closing Windows

- TCP uses **sliding windows** for flow control.
- Although buffer size is fixed during connection setup, the **window size changes**.
- The **receive window closes** (left edge moves right) when new bytes arrive.

- The **receive window opens** (right edge moves right) when the application consumes data.

Shrinking of Windows

- **Receive window cannot shrink.**
- **Send window can shrink** if the receiver advertises a smaller rwnd.

Window Shutdown

- Shrinking the send window is normally avoided.
- But the receiver can **temporarily shut down** the window by advertising rwnd = 0.
- This tells the sender: **“Stop sending for now.”**

Silly Window Syndrome

- Occurs when:
 - The sender produces data **very slowly**, or
 - The receiver consumes data **very slowly**.
- This leads to sending **very tiny segments** (example: only 1 byte of data).
- Causes **extreme inefficiency** (41 bytes of header for 1 byte of data).
- Makes network capacity **wasteful and ineffective**.
- This problem is known as **silly window syndrome**.

ERROR CONTROL

1. Checksum

- Every TCP segment contains a **16-bit checksum**.
- Purpose: Detect errors (corruption) in the segment during transmission.
- If checksum does not match → Segment is **discarded** and treated as **lost**.
- The sender will later retransmit it.

2. Acknowledgment (ACK)

TCP uses ACKs to confirm that data has been received.

- If a segment carries data, it **MUST** be acknowledged.
- Even control segments (like SYN, FIN) consume sequence numbers and are acknowledged.
- ACK segments themselves are **not** acknowledged.

Types of Acknowledgments

A. Cumulative Acknowledgment (Traditional TCP ACK)

- TCP tells the **next byte number** it expects.
- If bytes 1–100 are received → ACK = 101
- Out-of-order segments are ignored temporarily

B. Selective Acknowledgment (SACK)

- Used in modern TCP implementations.
- Reports received **blocks of data** even if some parts are missing.
- Helps sender know exactly **which segments** were lost.
- Implemented as a **TCP option**, not part of main header.

Example:

Received blocks = 100–200, 300–400

Missing = 200–300

SACK can report the two valid blocks.

Rules for Generating ACKs

TCP follows some rules when to send ACKs:

Rule 1: Piggyback ACKs

If receiver also has data to send, it adds ACK inside the segment.

Rule 2: Delayed ACK

If an in-order segment arrives, receiver waits **up to 500 ms** before sending ACK.

This reduces unnecessary ACK traffic.

Rule 3: Immediate ACK for new in-order segment

If previous segment was not acknowledged, ACK is sent immediately.

Rule 4: Out-of-order segment arrives → Immediate ACK

Receiver **repeats the ACK** for the next expected byte.

This helps sender detect missing segments quickly.

Rule 5: Missing segment arrives later

Receiver sends ACK for next expected byte, confirming the missing part is now okay.

Rule 6: Duplicate segment arrives

Receiver discards it, but sends an ACK again.

Helps if earlier ACK was lost.

4. Retransmission

Retransmission is the **core of TCP error control**.

A sent segment is kept in a **retransmission queue** until it is acknowledged.

Retransmission occurs in two ways:

A. Retransmission after RTO (Retransmission Time-Out)

- TCP calculates an **RTO timer** for each segment.
- If no ACK is received before timeout → resend the segment.
- RTO is dynamic, based on Round-Trip Time (RTT).

Example:

If RTT increases due to congestion, RTO will also increase.

B. Fast Retransmission (after 3 duplicate ACKs)

- If sender receives **3 duplicate ACKs**, it assumes a segment is lost.
- Immediately retransmits missing segment **without waiting for RTO**.

✓ Faster

✓ Reduces delay

✓ Improves performance

Example:

Receiver keeps saying ACK = 101, ACK = 101, ACK = 101 → means segment starting from byte 101 is missing.

Sender instantly retransmits segment 101.

CONGESTION DETECTION

TCP understands congestion using **feedback from ACKs**.

There are **two major signals** of congestion:

A. Time-Out → Severe Congestion

- If the sender does NOT receive an ACK before the timer expires...
- It assumes the segment is **lost due to heavy congestion**.
- This is considered **strong congestion**.

B. Three Duplicate ACKs → Mild Congestion

- Receiver sends several identical ACKs (ACK, ACK, ACK, ACK).
- This means **one packet is missing** but others arrived.
- The link is congested, but not too bad.

This is called **Fast Retransmit** condition

Indicates **mild congestion**, not severe

How Different TCP Versions Handle These Signals

Earlier version: **TCP Tahoe**

- Treated both events (timeout + duplicate ACKs) the same
- Always assumed severe congestion.

Later version: **TCP Reno**

- More intelligent:
 - **Timeout = strong congestion**
 - **3 duplicate ACKs = mild congestion**

This allows better network utilization and faster recovery.

Important Idea: TCP Uses ACKs Only

TCP does *not* receive congestion information from routers.

Instead, it guesses congestion by checking:

- Missing ACKs
- Delayed ACKs
- Multiple duplicate ACKs

If ACKs slow down → network is congested.

If ACKs arrive fast → network is fine.

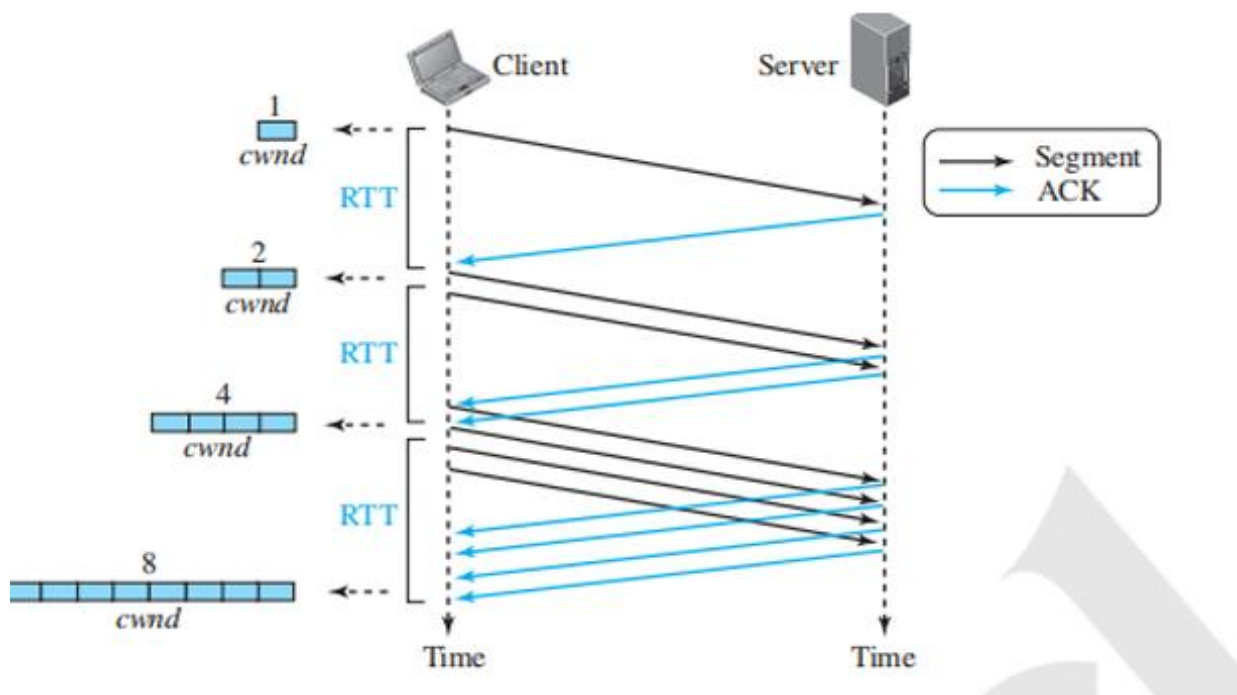
TCP CONGESTION POLICIES – SLOW START

TCP uses **three main algorithms** to control and avoid congestion:

1. **Slow Start**
2. **Congestion Avoidance**
3. **Fast Recovery**

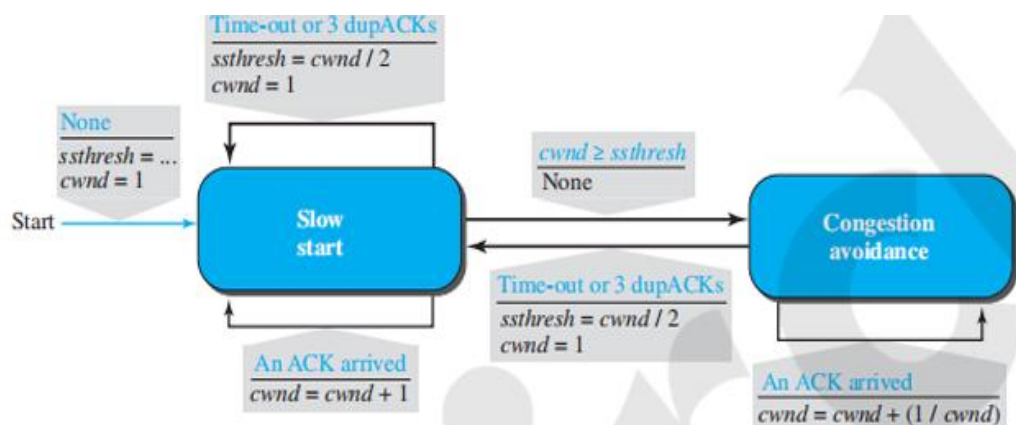
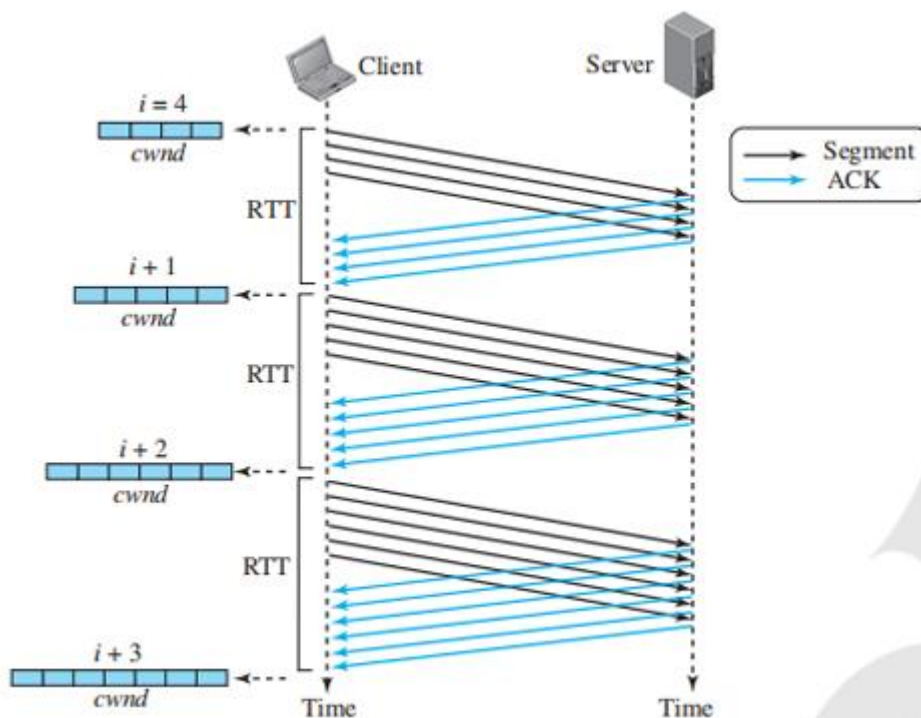
Slow Start: Exponential Increase

- The slow-start algorithm begins with the congestion window (cwnd) set to one maximum segment size (MSS) and increases by one MSS for each received acknowledgment.
- The MSS is negotiated during connection establishment. Despite the name, the algorithm grows exponentially.
- Initially, the sender transmits one segment, and upon receiving the ACK, the cwnd increases by 1, allowing the sender to transmit two segments.
- Each acknowledgment further increases cwnd, doubling the number of segments the sender can transmit, resulting in rapid growth as long as no congestion is detected.
- The size of the congestion window in this algorithm is a function of the number of ACKs arrived and can be determined as follows. If an **ACK arrives**, $\text{cwnd} = \text{cwnd} + 1$.

**Congestion Avoidance:**

Additive Increase The slow-start algorithm in TCP increases the size of the congestion window (cwnd) exponentially, which can lead to congestion. To mitigate this, TCP employs

the congestion avoidance algorithm, which increases $cwnd$ additively rather than exponentially. When $cwnd$ reaches the slow-start threshold (i), the slow-start phase ends, and the additive phase begins. In this algorithm, for each set of acknowledged segments (the entire “window”), $cwnd$ is incremented by one. A window represents the number of segments sent during a round-trip time (RTT).



- Tahoe uses slow start, congestion avoidance, and restarts slow start after congestion.

- At connection start:
 $\text{cwnd} = 1 \text{ MSS}$, $\text{ssthresh} = \text{predefined value}$.
- Congestion detected when:
 - timeout occurs
 - OR three duplicate ACKs arrive
(Tahoe treats both the same)
- On congestion:
 - $\text{ssthresh} = \text{cwnd} / 2$
 - $\text{cwnd} = 1 \text{ MSS}$
 - restart slow start
- During slow start, cwnd increases exponentially ($\text{cwnd} = \text{cwnd} + 1$ per ACK).
- When cwnd reaches ssthresh , TCP switches to congestion avoidance.
- In congestion avoidance, cwnd increases linearly (1 MSS per RTT).