Chapter 23

course	Computer Networks
Zohaib	none
Status	Not started
Resource	☐

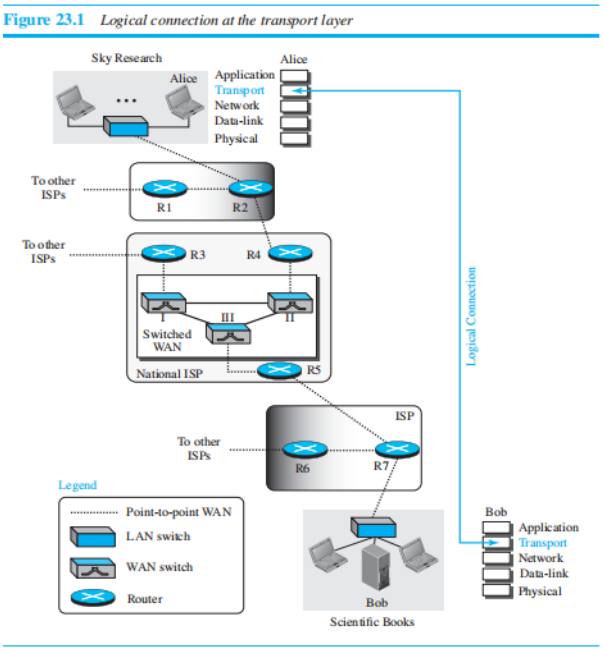
Notes

1. Introduction to the Transport Layer

The transport layer is situated between the application layer and the network layer. It facilitates process-to-process communication between application layers on the local and remote hosts. This communication is established using a logical connection, creating an illusion of a direct connection for the application layers.

Key Concepts:

- **Logical Connection:** Imaginary direct connection for sending and receiving messages between application layers on different hosts.
- **Transport Layer Interaction:** Only the end systems (e.g., Alice's and Bob's computers) use the transport layer services; intermediate routers utilize only the lower three layers (Physical, Data-Link, and Network).



2. Transport-Layer Services

The transport layer provides services to the application layer and receives services from the network layer.

Process-to-Process Communication:

- **Process:** An application-layer entity (running program) that uses transport layer services.
- **Host-to-Host vs. Process-to-Process Communication:**
  - **Host-to-Host (Network Layer):** Delivers messages to the destination computer.
  - **Process-to-Process (Transport Layer):** Delivers messages to the appropriate process within the destination computer.

Addressing: Port Numbers

- **Client-Server Paradigm:** Communication typically follows a client-server model where the client requests services from the server.
- **Port Numbers:** Identifiers for processes; ranges from 0 to 65,535.
  - **Ephemeral Port Numbers:** Used by clients, usually >1023.
  - **Well-Known Port Numbers:** Used by servers, predefined to facilitate easy access.

Port Number Classification by ICANN:

- **Well-Known Ports (0-1023):** Assigned and controlled by ICANN.
- **Registered Ports (1024-49,151):** Registered to prevent duplication.
- **Dynamic/Private Ports (49,152-65,535):** Temporary or private use.

Figure 23.3 *Port numbers*

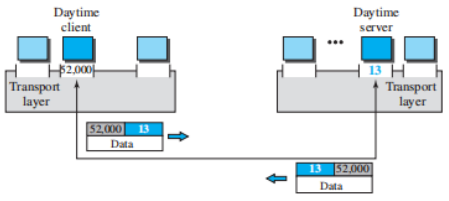
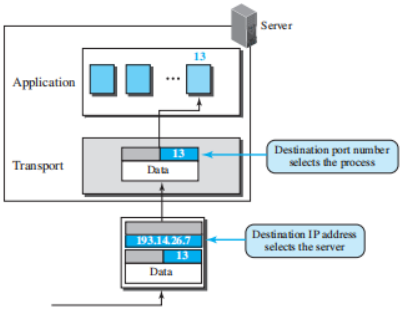


Figure 23.4 *IP addresses versus port numbers*



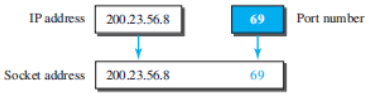
3. Socket Addresses

A transport-layer protocol requires both the IP address and the port number to make a connection, forming a **socket address**.

Example:

- **Client Socket Address:** Uniquely defines the client process.
- **Server Socket Address:** Uniquely defines the server process.

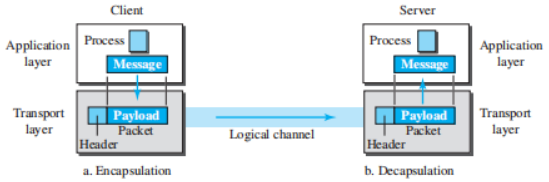
Figure 23.6 *Socket address*



4. Encapsulation and Decapsulation

- **Encapsulation:** At the sender's side, the transport layer adds a header to the message, forming a packet.
- **Decapsulation:** At the receiver's side, the transport layer removes the header and delivers the message to the appropriate process.

Figure 23.7 *Encapsulation and decapsulation*



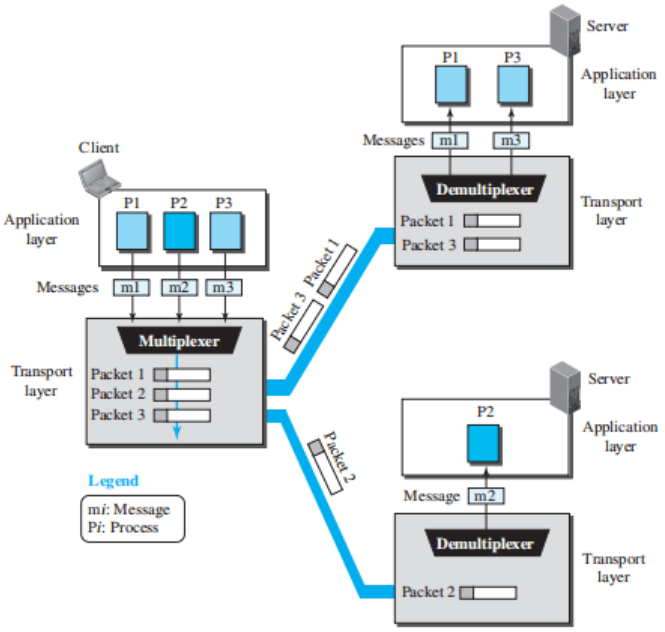
5. Multiplexing and Demultiplexing

- **Multiplexing:** Combining data from multiple processes for transmission.
- **Demultiplexing:** Separating received data to deliver it to the appropriate process.

Example:

- A client with three processes (P1, P2, P3) sends data to different servers. The transport layer at the client site multiplexes the data into packets, which are then demultiplexed at the server side to deliver to the correct process.

Figure 23.8 *Multiplexing and demultiplexing*



Summary Table

Concept	Description
Logical Connection	Imaginary direct connection for application layer communication
Process-to-Process Communication	Delivery of messages to specific processes within a host
Port Numbers	Identifiers for processes (0-65,535)
Socket Address	Combination of IP address and port number
Encapsulation	Adding transport layer header to messages
Decapsulation	Removing transport layer header at the receiver side
Multiplexing	Combining data from multiple sources for transmission
Demultiplexing	Distributing received data to the correct destination process

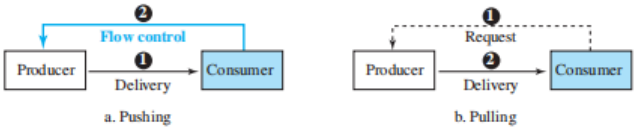
Flow Control

Flow control ensures a balance between production and consumption rates of data to prevent loss or inefficiency.

Methods of Delivery

- **Pushing**: Data is delivered without prior request from the consumer. Flow control is essential to prevent consumer overload.
- **Pulling**: Data is delivered upon consumer request, eliminating the need for flow control.

Figure 23.9 *Pushing or pulling*



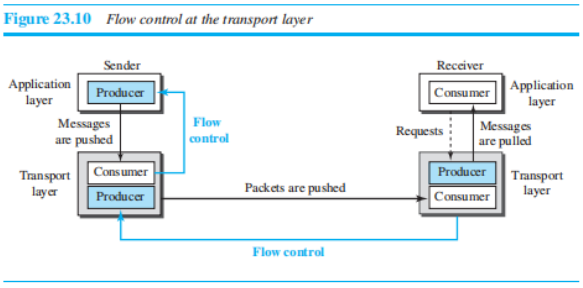
Flow Control at Transport Layer

Four entities are involved:

1. **Sender Process** (producer)
2. **Sender Transport Layer** (consumer and producer)
3. **Receiver Transport Layer** (consumer and producer)
4. **Receiver Process** (consumer)

Flow Control Mechanisms

- Buffers at both sender and receiver transport layers store packets temporarily.
- Communication between producers and consumers occurs when buffers are full or have vacancies.



Buffer Example

Using a single-slot buffer:

- Sender transport layer notifies the application layer when its slot is empty.
- Receiver transport layer acknowledges the sender when its slot is empty, signaling the sender to send the next packet.

Error Control

Ensures reliable communication by handling corrupted, lost, duplicate, or out-of-order packets.

Functions of Error Control

1. Detect and discard corrupted packets.
2. Track and resend lost packets.
3. Recognize and discard duplicate packets.
4. Buffer out-of-order packets until missing ones arrive.

Sequence Numbers

Packets are numbered sequentially within a defined range (e.g., 0 to 15 for a 4-bit sequence number field).

Acknowledgments

- Positive signals (ACKs) indicate successful packet receipt.
- Timers detect lost packets; if an ACK is not received, the sender resends the packet.

Combined Flow and Error Control

Combines the use of buffers with sequence and acknowledgment numbers to manage data transmission effectively.

Sliding Window Protocol

- Represents sequence numbers in a circular or linear format.
- Sliding window mechanism allows for efficient buffer management and error control.

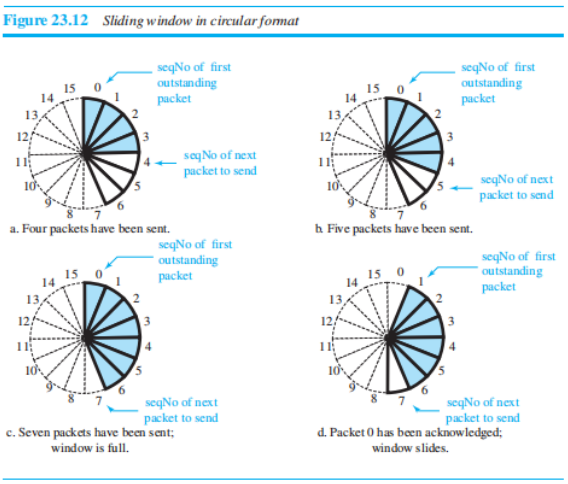


Figure 23.12 *Sliding window in circular format*

Congestion Control

Prevents network overload by controlling the number of packets sent.

Causes of Congestion

- Queues in routers and switches become overloaded when packet arrival rate exceeds processing rate.

Connectionless and Connection-Oriented Protocols

Connectionless Service

- Independent packet handling.
- No coordination between transport layers; packets may arrive out of order or get lost without detection.

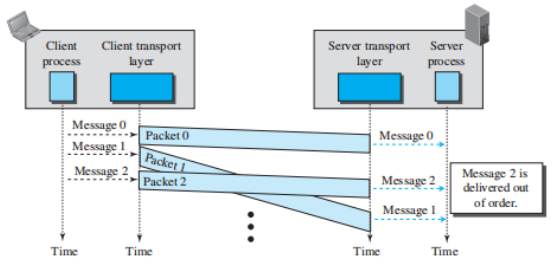


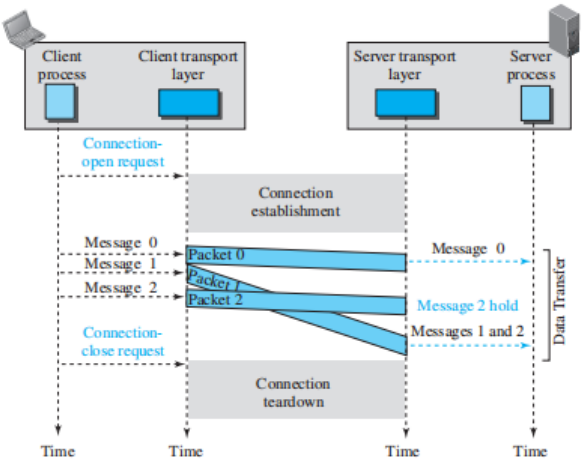
Figure 23.14 *Connectionless service*

Connection-Oriented Service

- Establishes a logical connection before data exchange.

- Allows for effective implementation of flow control, error control, and congestion control.

Figure 23.15 *Connection-oriented service*



Finite State Machine (FSM)

Used to represent the behavior of transport-layer protocols.

Connectionless FSM

- Single state: Established state, ready to send and receive packets.

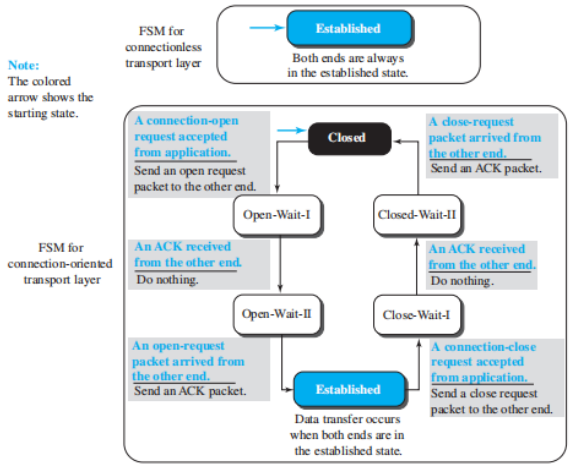
Connection-Oriented FSM

- Includes states for connection establishment, data transfer, and connection teardown.

FSM States for Connection-Oriented Service

1. **Closed**: No connection.
2. **Open-Wait-I**: Waiting for acknowledgment of connection request.
3. **Open-Wait-II**: Waiting for the other end to request a connection.
4. **Established**: Ready for data transfer.
5. **Close-Wait-I**: Waiting for acknowledgment of close request.
6. **Close-Wait-II**: Waiting for close request from the other end.

Figure 23.16 *Connectionless and connection-oriented service represented as FSMs*



Transport-Layer Protocols

1. Introduction

Transport-layer protocols combine various services to manage data transmission between systems. This section discusses basic to complex protocols, starting with unidirectional protocols and eventually covering bidirectional (full-duplex) protocols.

2. Simple Protocol

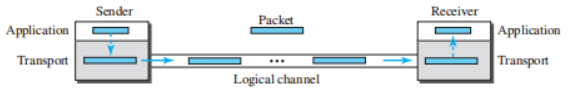
Overview

- **Connectionless Protocol:** No flow or error control.
- **Assumption:** Receiver can handle any packet immediately.

Process

1. **Sender:** Receives a message from the application layer, creates a packet, and sends it.
2. **Receiver:** Receives the packet, extracts the message, and delivers it to the application layer.

Figure 23.17 *Simple protocol*



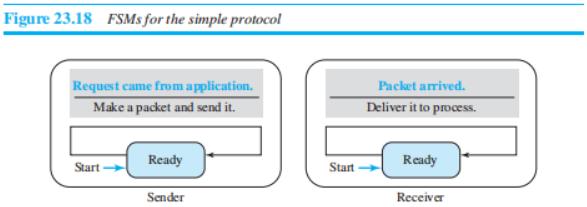
Finite State Machines (FSMs)

- **Sender FSM:**
  - **State:** Ready
  - **Event:** Application request
  - **Action:** Create and send packet
- **Receiver FSM:**
  - **State:** Ready

- **Event:** Packet arrival
- **Action:** Extract and deliver message

Example

- **Communication:** The sender sends packets without considering the receiver's state.



3. Stop-and-Wait Protocol

Overview

- **Connection-Oriented Protocol:** Includes flow and error control.
- **Sliding Window:** Size 1, meaning one packet sent at a time.

Process

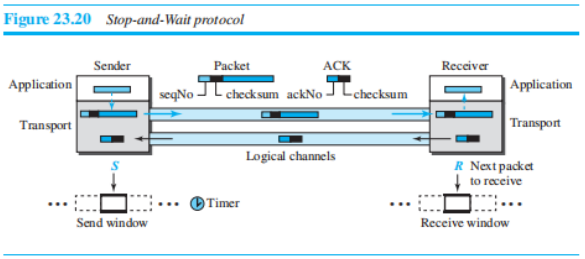
1. **Sender:**
   - Sends a packet and starts a timer.
   - Waits for acknowledgment (ACK) before sending the next packet.
2. **Receiver:**
   - Checks packet checksum.
   - Sends ACK if packet is correct; otherwise, silently discards the packet.

Sequence Numbers

- **Purpose:** Prevent duplicate packets.
- **Range:** Minimum range to avoid ambiguity (modulo 2 arithmetic).

Acknowledgment Numbers

- **Purpose:** Indicate the next expected packet.
- **Example:** If packet 0 is received, ACK with number 1 is sent, indicating the expectation of packet 1 next.



FSMs

- **Sender FSM:**
  - **States:** Ready and Blocking
  - **Transitions:**
    - From Ready: On application request, create and send packet, start timer.
    - From Blocking: On correct ACK, stop timer, slide window, move to Ready.
- **Receiver FSM:**
  - **State:** Ready
  - **Transitions:**
    - On correct packet: Deliver message, slide window, send ACK.
    - On incorrect or corrupted packet: Discard packet, send ACK.

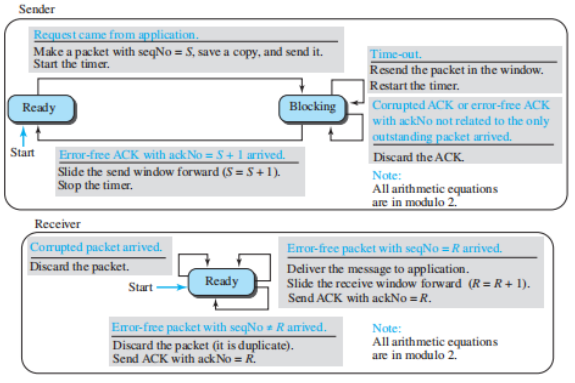
Example

- **Scenario:**
  - Packet 0 is sent and acknowledged.
  - Packet 1 is lost and resent after timeout.
  - Resent packet 1 is acknowledged.
  - ACK for packet 0 is lost, leading to packet resend and acknowledgment.

Efficiency

- **Issue:** Inefficiency in high bandwidth or long delay channels (bandwidth-delay product).
- **Example:** 1 Mbps bandwidth and 20ms round-trip delay leads to 5% utilization in Stop-and-Wait protocol.

Figure 23.21 *FSMs for the Stop-and-Wait protocol*



4. Pipelining

Concept

- **Definition:** Starting a task before the previous one completes.
- **Application:** Allows multiple packets to be sent before receiving feedback.
- **Benefit:** Increases efficiency for high bandwidth-delay products.

Comparison of Protocols

Protocol	Type	Control Mechanisms	Efficiency
Simple Protocol	Connectionless	None	Low (basic usage)
Stop-and-Wait	Connection-Oriented	Flow, Error, Sequence numbers	Moderate
Pipelined Protocols	Connection-Oriented	Flow, Error, Sequence numbers, Pipelining	High

Go-Back-N Protocol (GBN) - Notes

Introduction

The Go-Back-N (GBN) protocol is designed to improve the efficiency of data transmission by allowing multiple packets to be in transit simultaneously, rather than waiting for an acknowledgment (ACK) after each packet. This helps in maintaining a busy channel, ensuring efficient use of bandwidth.

Key Concepts

Sequence Numbers

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

  $2m$

  m

Acknowledgment Numbers

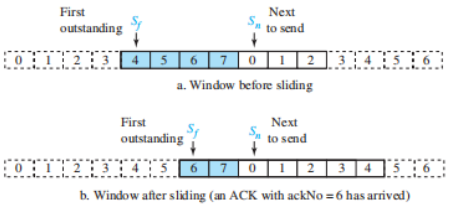
- ACK numbers are cumulative, indicating the sequence number of the next expected packet.
  - Example: If `ackNo = 7`, packets up to sequence number 6 have been received successfully.

Send Window

- The send window is an imaginary construct that defines the range of sequence numbers for packets that can be sent or are in transit.
- The maximum size of the send window is $2m-1$.

  $2m-1$
- The window is divided into four regions:
  1. **Acknowledged packets**: No longer need attention.
  2. **Outstanding packets**: Sent but not yet acknowledged.
  3. **Packets ready to be sent**: Data received from the application layer but not yet sent.
  4. **Future packets**: Cannot be sent until the window slides.
- Variables:
  - `Sf`: Sequence number of the first outstanding packet.
  - `Sn`: Sequence number of the next packet to be sent.
  - `Ssize`: Size of the send window.

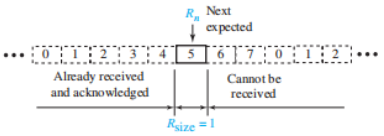
Figure 23.25 *Sliding the send window*



Receive Window

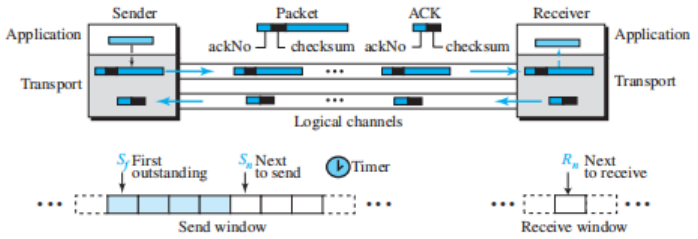
- The receive window ensures correct packet reception and acknowledgment.
- Size of the receive window is always 1.
- The receiver looks for a specific packet; out-of-order packets are discarded.

Figure 23.26 *Receive window for Go-Back-N*



- Variable:
  - `Rn`: Sequence number of the next expected packet.

Figure 23.23 *Go-Back-N protocol*



Protocol Mechanics

Timers

- A single timer is used for the first outstanding packet.
- If the timer expires, all outstanding packets are resent.

Resending Packets

- On timeout, the sender resends all outstanding packets from `Sf` to `Sn-1`.

Finite State Machines (FSM)

Sender FSM

- **Ready State**:

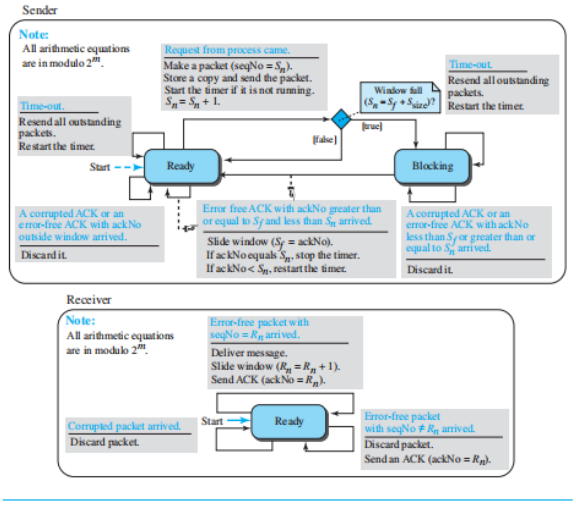
1. **Request from application**: Packet created and sent; `Sn` incremented.
2. **Error-free ACK**: Slide window; stop or restart timer as needed.
3. **Corrupted or unrelated ACK**: Discarded.
4. **Timeout**: Resend all outstanding packets.

- **Blocking State**:
  1. **Error-free ACK**: Slide window; stop or restart timer; move to ready state.
  2. **Corrupted or unrelated ACK**: Discarded.
  3. **Timeout**: Resend all outstanding packets.

Receiver FSM

- Always in ready state.
  1. **Error-free packet (seqNo = Rn)**: Deliver message; slide window; send ACK.
  2. **Error-free packet (seqNo ≠ Rn)**: Discard; send ACK.
  3. **Corrupted packet**: Discard.

Figure 23.27 *FSMs for the Go-Back-N protocol*



Send Window Size

- The send window size must be less than 2*m* to avoid errors.

  $2m$

  - Example: With *m*=2, the maximum window size is 3. If all ACKs are lost, retransmissions will be managed correctly. However, if the window size is 4, packet 0 could be mistakenly accepted as a new packet due to sequence number wraparound.

    m=2

Examples

Example 23.7 - Delayed ACKs

- Scenario where forward channel is reliable but reverse channel has delays/losses.

- Demonstrates cumulative ACK handling.

Example 23.8 - Lost Packet

- Shows how GBN handles lost packets by discarding out-of-order packets and using cumulative ACKs.

Comparison: Go-Back-N vs. Stop-and-Wait

- **Stop-and-Wait**: A special case of Go-Back-N with m=1, allowing only one packet in transit at a time (send window size of 1).

  m=1

Diagrams and Figures

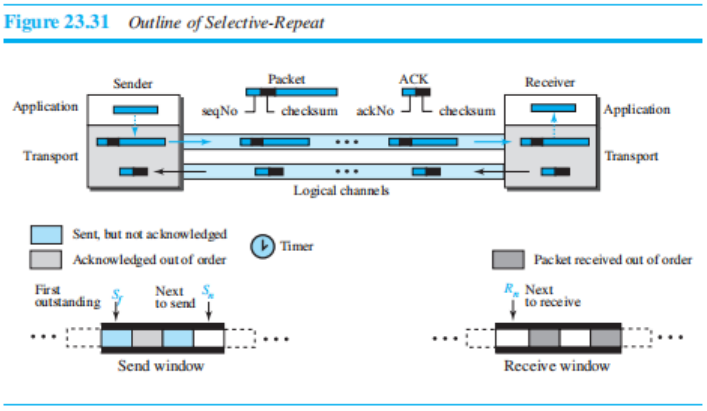
- **Send Window Sliding**: Illustrates how the send window slides upon receiving an ACK.
- **Receive Window**: Shows the single-slot receive window and its sliding mechanism.
- **FSMs**: Visual representation of sender and receiver states and transitions.

Conclusion

The Go-Back-N protocol enhances transmission efficiency by maintaining multiple packets in transit, managing retransmissions effectively with a cumulative acknowledgment strategy, and ensuring orderly delivery using the send and receive windows. Proper window management and handling of timeouts and ACKs are critical to its operation.

Selective-Repeat Protocol

The Selective-Repeat (SR) protocol is designed to improve efficiency over the Go-Back-N (GBN) protocol by selectively resending only those packets that are actually lost or corrupted. Here's an overview of the key concepts, functionalities, and comparisons between these protocols.



Overview

- **Go-Back-N (GBN)**: Simplifies receiver's process, discards out-of-order packets, inefficient in high-loss networks.
- **Selective-Repeat (SR)**: Resends only lost packets, maintains order at the receiver with two windows (send and receive).

Windows in Selective-Repeat

- **Send Window**: Maximum size is $2m-1$.

$2^m - 1$

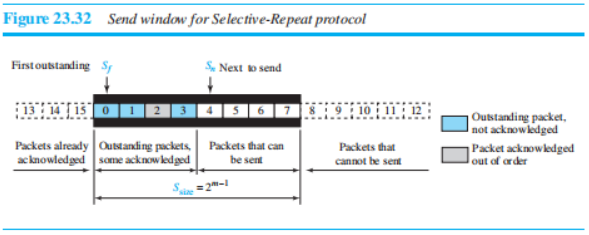
- **Receive Window**: Same size as send window, can buffer out-of-order packets.

Comparison of Windows

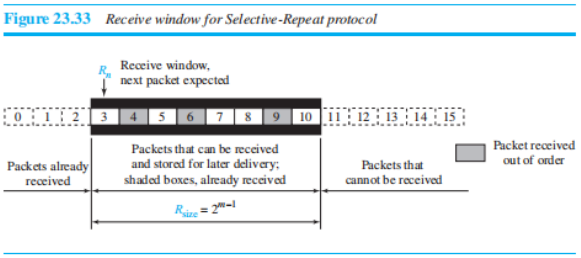
Feature	Go-Back-N	Selective-Repeat
Send Window Size	$2^m - 12^m - 1$	$2^m - 12^m - 1$
Receive Window Size	1	$2^m - 12^m - 1$
Handling Packets	Discards out-of-order packets	Buffers out-of-order packets

Working Mechanism

- **Sender**:
  - Maintains a window of sent but unacknowledged packets.
  - Uses one timer per packet (theoretically), but often implemented with a single timer.
  - Resends only the timed-out packet.



- **Receiver**:
  - Buffers out-of-order packets.
  - Sends an acknowledgment (ACK) for each received packet.
  - Delivers packets to the application layer in order when possible.



Protocol Functionality

- **Sending Packets**:
  - Packet creation and sending occur upon request from the application layer.
  - Sequence number (Sn) is incremented modulo 2*m*.

    2^m
  - If window is full, sender goes to blocking state.
- **Receiving ACKs**:
  - Marks packet as acknowledged.

    - Slides window if ACK matches the first outstanding packet.
    - Restarts timer if there are outstanding packets.
- **Timeout Handling**:
    - Resends all unacknowledged packets within the window.
    - Restarts timer.
- **Receiving Packets**:
    - Stores and ACKs packets within the window.
    - Discards and ACKs packets outside the window.
    - Delivers packets to the application layer in order.

Example Scenarios

Example 1: ACK Interpretation

Assume a sender sends packets 0, 1, 2, 3, 4, and 5. An ACK with ackNo = 3 is received.

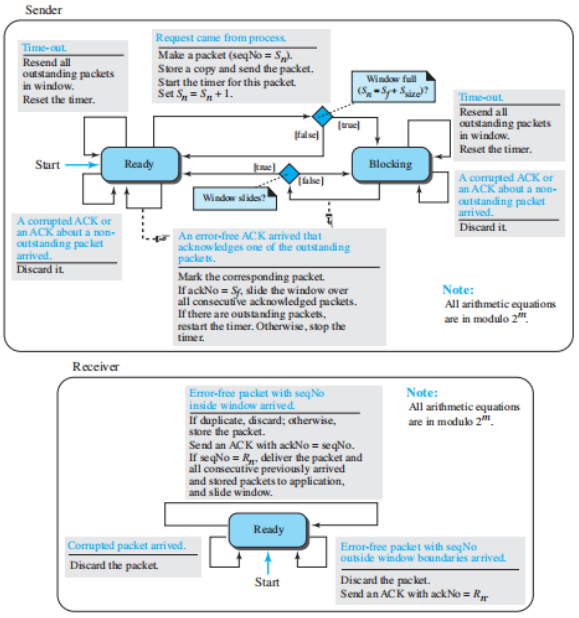
- **GBN**: Packets 0, 1, 2 received; expecting packet 3.
- **SR**: Packet 3 received; no information about other packets.

Example 2: Packet Loss

- **Scenario**: Packet 1 is lost; packets 2 and 3 are received out-of-order.
- **Sender**:
    - Resends only packet 1 upon timeout.
    - Slides window after receiving ACK for packet 1.
- **Receiver**:
    - Buffers packets 2 and 3.
    - Delivers packets to the application layer once packet 1 is received.

Protocol Finite State Machines (FSMs)

Figure 23.34 *FSMs for SR protocol*



Sender FSM

- **Ready State**:
  - Handles application requests, ACKs, and timeouts.
  - Moves to blocking state if the window is full.
- **Blocking State**:
  - Handles ACKs and timeouts.
  - Returns to ready state when the window slides.

Receiver FSM

- Always in the ready state.
- Handles packet arrivals (in window or out of window) and sends ACKs accordingly.

Practical Considerations

- **Window Size**: Must be $2m-1$ to prevent errors due to sequence number wraparound.

  $2m-1$
- **Efficiency**: SR is more efficient than GBN in networks with higher packet loss rates.

Bidirectional Protocols: Piggybacking

- **Concept**: ACKs are included in data packets to improve efficiency.
- **Implementation**: Each side maintains independent send and receive windows.

Example Layout: Bidirectional GBN with Piggybacking

- Client and server both use send and receive windows for data and ACKs.

By structuring these notes with headings, subheadings, and tables, the key points of the Selective-Repeat protocol and its comparison with Go-Back-N are captured clearly and concisely, making it easier to

understand and study.