**AIM:**

Program to implement **Sliding Window Protocol using Selective Repeat**.

**DESCRIPTION:**

1. **Selective Repeat (SR)** is a reliable data transfer protocol that allows the sender to transmit multiple frames before waiting for acknowledgments (ACKs), similar to the Go-Back-N protocol.
2. The **receiver in SR accepts out-of-order frames** and temporarily buffers them until any missing frames are received.
3. Only **lost or erroneous frames** are retransmitted, rather than the entire window, improving efficiency and reducing network congestion.
4. Both the **sender and receiver maintain window sizes**, which define how many frames can be sent or received before acknowledgment.
5. Each frame is assigned a **sequence number** to help the receiver identify the correct order and detect missing frames.
6. The sender keeps track of **sent but unacknowledged frames**, and retransmits only those that are not acknowledged within a timeout period.
7. The receiver sends **individual acknowledgments (ACKs)** for each correctly received frame, which allows the sender to slide the window forward selectively.
8. SR improves **throughput** because the sender does not have to retransmit successfully received frames, unlike Go-Back-N.
9. SR is particularly useful in **networks with high latency or unreliable channels**, where retransmitting the entire window would be inefficient.
10. The protocol ensures **reliable and ordered delivery of frames**, making it suitable for applications like file transfer, streaming, and other data communication systems.

**1. What is the Sliding Window Protocol?**

**Answer:**  
The Sliding Window Protocol is a method used in data communication to **reliably send multiple frames over a network** without waiting for each acknowledgment individually. It manages **flow control and ensures ordered delivery** of frames.

**2. What is Selective Repeat (SR) in the Sliding Window Protocol?**

**Answer:**  
Selective Repeat is a variant of the Sliding Window Protocol where the **receiver can accept out-of-order frames** and buffer them. Only **lost or erroneous frames are retransmitted**, making it more efficient than Go-Back-N.

**3. How does Selective Repeat differ from Go-Back-N?**

**Answer:**

* **Go-Back-N:** retransmits all frames after a lost or erroneous frame.
* **Selective Repeat:** retransmits **only the specific lost or erroneous frames**, reducing unnecessary retransmissions.

**4. What are sequence numbers used for in SR?**

**Answer:**  
Sequence numbers **identify each frame uniquely**, help the receiver **reorder frames**, and detect **missing frames** for selective retransmission.

**5. How does the sender manage frames in SR?**

**Answer:**  
The sender maintains a **window of sent but unacknowledged frames** and retransmits only frames for which **ACKs are not received within a timeout**.

**6. How does the receiver handle out-of-order frames?**

**Answer:**  
The receiver **buffers out-of-order frames** until the missing frames arrive, ensuring **correct order** before delivering them to the application layer.

**7. What is the advantage of Selective Repeat?**

**Answer:**

* Efficient use of bandwidth by **avoiding unnecessary retransmissions**
* Improves **throughput**, especially in networks with **high latency or error rates**

**8. What happens when a frame is lost in SR?**

**Answer:**  
Only the **lost frame is retransmitted**, while all correctly received frames remain in the buffer.

**9. What are the key components of SR?**

**Answer:**

* **Sender window**: tracks sent but unacknowledged frames
* **Receiver window**: tracks received frames and buffers out-of-order frames
* **Sequence numbers**: identify frames uniquely
* **Acknowledgments (ACKs)**: confirm successful reception

**10. Where is Selective Repeat used in real life?**

**Answer:**  
Used in **file transfer protocols, streaming applications, TCP communication**, and other **network systems requiring reliable and ordered delivery** of data over unreliable channels.

**AIM:**

**To write a program to implement congestion control using the Leaky Bucket algorithm.**

**DESCRIPTION:**

1. **The Leaky Bucket algorithm is a traffic shaping mechanism used in computer networks to control data transmission rates and avoid congestion.**
2. **It uses a bucket (buffer) of fixed capacity that holds incoming packets temporarily before they are transmitted to the network.**
3. **Packets arrive at variable rates, simulating bursty traffic conditions where data can arrive unpredictably and in uneven quantities.**
4. **The packets leave the bucket at a constant output rate, ensuring that data is sent out smoothly over time, avoiding sudden bursts in transmission.**
5. **If the bucket overflows (i.e., the size of the incoming packet is greater than the remaining bucket capacity), the extra packets are discarded, representing packet loss.**
6. **This mechanism effectively regulates the flow of packets into the network, maintaining stability and preventing congestion at routers and switches.**
7. **It acts as a rate limiter, ensuring that the data transmission rate does not exceed the predefined network capacity.**
8. **By converting irregular or bursty traffic into a steady and predictable output stream, the algorithm helps improve Quality of Service (QoS).**
9. **The Leaky Bucket algorithm is simple to implement and is widely used in network devices, routers, and operating systems for bandwidth management.**
10. **Overall, it plays a crucial role in congestion control, traffic shaping, and fair bandwidth allocation, preventing packet loss and ensuring efficient network performance.**

**🎯 Viva Questions and Answers**

**1. What is the Leaky Bucket algorithm?**

**Answer:**  
The Leaky Bucket algorithm is a traffic shaping and congestion control mechanism used in computer networks. It regulates the data transmission rate by allowing packets to enter a fixed-size bucket and leave it at a constant rate, preventing network congestion.

**2. What is the main purpose of the Leaky Bucket algorithm?**

**Answer:**  
The main purpose is to **control network congestion** and **smooth out bursty traffic** by sending data at a uniform rate, ensuring stability and avoiding packet loss.

**3. How does the Leaky Bucket algorithm work?**

**Answer:**

* Incoming packets are stored in a bucket (buffer) with limited capacity.
* The bucket leaks (transmits) data at a fixed constant rate.
* If packets arrive faster than they can be sent and the bucket is full, the extra packets are discarded.

**4. What happens when the bucket overflows?**

**Answer:**  
When the bucket overflows, the incoming packets are **discarded or dropped**, representing **packet loss** due to congestion or excessive traffic.

**5. What are the main parameters used in the Leaky Bucket algorithm?**

**Answer:**

1. **Bucket capacity (C):** The maximum amount of data the bucket can hold.
2. **Output rate (r):** The constant rate at which data is transmitted.
3. **Incoming packet size:** The size of packets entering the bucket.

**6. How does the Leaky Bucket help in congestion control?**

**Answer:**  
It prevents network congestion by **limiting the rate of outgoing traffic**. Even if packets arrive at a high rate, they are sent out at a controlled constant speed, avoiding sudden data bursts in the network.

**7. What is the difference between the Leaky Bucket and Token Bucket algorithms?**

**Answer:**

* **Leaky Bucket:** Sends packets at a fixed rate; bursty traffic is smoothed out.
* **Token Bucket:** Allows bursts of traffic as long as tokens are available, offering more flexibility in rate control.

**8. Where is the Leaky Bucket algorithm used in real life?**

**Answer:**  
It is used in **network routers, traffic shapers, and operating systems** for **rate limiting**, **bandwidth management**, and **Quality of Service (QoS)** control.

**9. What are the advantages of the Leaky Bucket algorithm?**

**Answer:**

* Simple and easy to implement.
* Prevents network congestion.
* Provides smooth and predictable data flow.
* Ensures fair bandwidth usage.

**10. What are the limitations of the Leaky Bucket algorithm?**

**Answer:**

* It cannot handle bursty traffic efficiently (packets may be dropped).
* The constant output rate may underutilize available bandwidth.
* Not suitable for applications requiring variable data rates.

**Bit Stuffing — extended description (10 points)**

1. **Purpose & context:** Bit stuffing is a data-link layer technique used to preserve frame boundary markers (flags) and provide *data transparency* — i.e., allow arbitrary bit patterns in payload without confusing them with control flags.
2. **Flag pattern:** A fixed 8-bit flag like 01111110 (HDLC-style) is placed at frame start and end. The receiver recognizes these exact patterns as frame delimiters.
3. **Stuffing rule:** During transmission, whenever the sender’s bit-stream contains **five consecutive 1 bits** in the payload, the transmitter automatically inserts a single 0 bit immediately after those five 1s. This prevents the occurrence of the 01111110 flag inside the payload.
4. **Destuffing at receiver:** The receiver scans incoming bits; when it sees five consecutive 1s followed by a 0, it removes that 0 (destuffs) and continues. If it sees five 1s followed by another 1, that indicates the flag boundary (or an error).
5. **Transparency & correctness:** Because stuffing only happens after five 1s, the exact flag pattern (01111110) cannot appear in the stuffed payload — the inserted 0 breaks it — so flags reliably mark frame boundaries.
6. **Bit-level vs byte-level:** Bit stuffing works at the bit-stream level (not on bytes). This makes it independent of any byte/character encoding and robust for mixed / binary data.
7. **Error detection interplay:** Bit stuffing only handles delimiting; it is normally used together with error detection (e.g., CRC) so the receiver can detect when bit errors make stuffing/flags ambiguous.
8. **Overhead & efficiency:** Stuffed bits add overhead: in the worst case (payload full of 1s), every 5 data 1s causes an extra 0. On average overhead is small but depends on data content. There is no fixed alignment cost like byte stuffing may incur.
9. **Implementation details:** Implementations typically maintain a counter of consecutive 1s while scanning/transmitting. On transmit: when counter reaches 5, transmit 0 and reset counter. On receive: when counter reaches 5, check next bit — if it’s 0, drop it and reset; if it’s 1, treat it as flag (or error if context unexpected).
10. **Alternatives & extensions:** Alternatives include byte stuffing (escape bytes), using length fields instead of flags, or using higher-level protocols that avoid in-band delimiters. Bit stuffing remains popular in simple, continuous bit-stream protocols (e.g., HDLC, PPP’s bit-oriented modes).

**Viva / oral-exam questions (with concise answers)**

1. **Q:** What is bit stuffing and why is it used?  
   **A:** Inserting a 0 after five consecutive 1s in the payload to prevent the frame flag (01111110) from appearing inside data; it preserves frame boundaries and data transparency.
2. **Q:** What is the flag sequence commonly used with bit stuffing?  
   **A:** 01111110 (the HDLC flag).
3. **Q:** Explain the transmitter algorithm for bit stuffing.  
   **A:** Maintain a count of consecutive 1s. For each outgoing bit: if it's 1, increment count; if count becomes 5, transmit the 1 then insert a 0 and reset count. If bit is 0, transmit and reset count.
4. **Q:** How does the receiver detect and remove stuffed bits?  
   **A:** Maintain the same consecutive-1s counter. After seeing five 1s, examine next bit: if 0 — drop it (destuff) and reset; if 1 — that's part of the flag (01111110) or an error.
5. **Q:** What happens if a bit error flips a stuffed 0 to 1?  
   **A:** The receiver may misinterpret the pattern as a flag boundary or an invalid sequence, causing framing errors. That’s why CRC/error detection is needed.
6. **Q:** Compare bit stuffing with byte stuffing. Give one advantage and one disadvantage.  
   **A:** Advantage: bit stuffing works at bit level so it's encoding-agnostic and more space-efficient for some data. Disadvantage: implementation is slightly more complex (bit-level operations) and may be slower in software.
7. **Q:** Does bit stuffing change the length of the transmitted frame?  
   **A:** Yes — it increases frame length by the number of stuffed 0s inserted; the receiver removes them to recover original length.
8. **Q:** Why is bit stuffing independent of character encoding?  
   **A:** Because it operates on raw bits, not on byte or character boundaries, so any binary payload is handled uniformly.
9. **Q:** In an extreme worst-case payload (all 1s), how often are 0s inserted?  
   **A:** After every five 1s a 0 is inserted — so for every 5 data 1s, one extra 0 (20% overhead for long runs of 1s).
10. **Q:** How does the receiver differentiate between a stuffed 0 and a legitimate 0 in the data?  
    **A:** It uses the context: only the 0 that immediately follows five consecutive 1s is considered a stuffed bit and is removed; other 0s are treated as normal data.
11. **Q:** Can bit stuffing be used with other flag patterns?  
    **A:** Yes — the rule adapts to prevent the flag pattern from appearing; HDLC uses five 1s rule because its flag has six consecutive 1s surrounded by 0s.
12. **Q:** Why is error detection (like CRC) necessary when using bit stuffing?  
    **A:** Because bit errors can corrupt flags or stuffed bits and cause mis-framing; CRC helps detect corrupted frames so they can be discarded or retransmitted.
13. **Q:** How would you implement bit stuffing in code (briefly)?  
    **A:** Iterate over bits; keep ones\_count. Append current bit to output; if bit is 1, increment ones\_count and if ones\_count==5 append a 0 and reset ones\_count. If bit is 0 reset ones\_count.
14. **Q:** What effect does bit-stuffing have on latency and throughput?  
    **A:** Slightly reduces throughput due to extra bits; processing overhead may add minimal latency at transmitter/receiver, but impact is generally small.

**AIM:**

Program to implement on a data set of characters the three CRC polynomials – **CRC-12**, **CRC-16**, and **CRC-CCITT**.

**🔹 Extended Description (10 Points)**

1. **Definition:**  
   CRC (Cyclic Redundancy Check) is an **error detection technique** used in the **data link layer** to ensure data integrity during transmission.
2. **Purpose:**  
   The main goal of CRC is to detect **accidental changes**, such as noise or bit flips, in transmitted data across communication channels.
3. **Basic Principle:**  
   It treats the binary data as a **polynomial** and divides it by a **generator polynomial** (representing CRC-12, CRC-16, or CRC-CCITT) using **modulo-2 arithmetic**.
4. **Generator Polynomials:**
   * **CRC-12:** x^12 + x^11 + x^3 + x^2 + x + 1 → binary: 1100000001111
   * **CRC-16:** x^16 + x^15 + x^2 + 1 → binary: 11000000000000101
   * **CRC-CCITT:** x^16 + x^12 + x^5 + 1 → binary: 10001000000100001
5. **Data Preparation:**  
   Before division, zeros equal to the **degree of the polynomial** are appended to the input data.  
   Example: For CRC-16, append **16 zeros** to the data bits.
6. **Modulo-2 Division (XOR Operation):**  
   CRC calculation uses **bitwise XOR** for division since addition and subtraction are equivalent under modulo-2 arithmetic.
7. **Remainder (CRC Code):**  
   The remainder obtained after the division is the **CRC value**. This is appended to the data to form the **transmitted frame**.
8. **Error Detection at Receiver:**  
   At the receiver end, the same division is performed.
   * If the remainder is **0**, no error occurred.
   * If the remainder is **non-zero**, an **error** is detected.
9. **Advantages of CRC:**
   * Detects **single-bit**, **double-bit**, and **burst errors** efficiently.
   * Simple to implement using hardware shift registers or software XOR logic.
10. **Practical Applications:**  
    CRC is widely used in **Ethernet frames**, **HDLC**, **PPP**, **USB**, **Bluetooth**, and **storage systems** to ensure data reliability.

**🔹 Common Viva Questions (with Answers)**

1. **Q:** What is CRC and why is it used?  
   **A:** CRC (Cyclic Redundancy Check) is an error detection method that identifies accidental changes in transmitted data using polynomial division.
2. **Q:** What is the difference between CRC-12, CRC-16, and CRC-CCITT?  
   **A:** They differ in their **generator polynomials** and **polynomial lengths**, which determine how strong their error detection capability is.
3. **Q:** Why are zeros appended to the data before CRC calculation?  
   **A:** To make space for the CRC bits that will be generated during division. The number of zeros equals the degree of the polynomial.
4. **Q:** What mathematical operation is used in CRC computation?  
   **A:** **Modulo-2 division**, which uses **XOR** instead of subtraction.
5. **Q:** What happens if the remainder after checking is non-zero?  
   **A:** It indicates that an error occurred during transmission.
6. **Q:** Can CRC correct errors?  
   **A:** No, CRC only **detects errors**, not corrects them.
7. **Q:** Why is modulo-2 arithmetic used?  
   **A:** Because CRC works on binary data where each bit can be represented by 0 or 1, making XOR operations sufficient for division.
8. **Q:** What are the typical lengths of CRC codes?  
   **A:** CRC-12 → 12 bits, CRC-16 → 16 bits, CRC-CCITT → 16 bits.
9. **Q:** What types of errors can CRC detect?  
   **A:** It can detect **single-bit**, **double-bit**, **odd number of bit errors**, and **burst errors** (up to polynomial length).
10. **Q:** Where is CRC commonly used in networking?  
    **A:** In **Ethernet frames**, **HDLC**, **PPP**, and **wireless protocols** for ensuring frame-level data integrity.

**AIM:**

Program to implement Sliding Window Protocol for **Go-Back-N (GBN)**.

**🔹 Extended Description (10 Points)**

1. **Definition:**  
   Go-Back-N (GBN) is a **sliding window protocol** used for **reliable data transfer** in computer networks at the **Data Link Layer** or **Transport Layer**.
2. **Purpose:**  
   The protocol ensures **error-free and ordered delivery** of frames between sender and receiver, even in the presence of errors such as frame loss or corruption.
3. **Sender Window Concept:**  
   The sender maintains a **window of size N**, meaning it can transmit **N unacknowledged frames** before waiting for an acknowledgment (ACK).
4. **Frame Transmission:**  
   The sender continuously sends frames within the window limit without waiting for each individual ACK, thus improving **channel utilization** compared to Stop-and-Wait.
5. **Acknowledgment Mechanism:**  
   The receiver sends a **cumulative ACK**, acknowledging all frames up to the last correctly received in-order frame.
6. **Error Handling (Go-Back behavior):**  
   If a frame is **lost or corrupted**, the receiver **discards all subsequent frames**, and the sender **retransmits all frames** starting from the erroneous one.
7. **Timer Management:**  
   The sender starts a **timer** for the first unacknowledged frame. If the timer expires before receiving an ACK, it **goes back** and retransmits all pending frames.
8. **Receiver Behavior:**  
   The receiver does not buffer out-of-order frames; it only accepts frames in sequential order to maintain reliable delivery.
9. **Throughput & Efficiency:**  
   GBN achieves higher throughput than Stop-and-Wait because it allows multiple frames to be in transit simultaneously. However, retransmitting multiple frames after an error can reduce efficiency.
10. **Applications:**  
    Go-Back-N is used in various reliable communication protocols such as **HDLC (High-Level Data Link Control)** and **TCP (in simplified form)** for efficient data transfer.

**🔹 Viva Questions (with Answers)**

1. **Q:** What is the main purpose of the Go-Back-N protocol?  
   **A:** To enable reliable and efficient data transmission by allowing multiple frames to be sent before waiting for acknowledgments.
2. **Q:** How does Go-Back-N differ from Stop-and-Wait?  
   **A:** In Stop-and-Wait, the sender transmits one frame at a time, while in GBN, the sender can send up to **N** frames before waiting for an acknowledgment.
3. **Q:** What does the “N” represent in Go-Back-N?  
   **A:** It represents the **window size**, or the maximum number of unacknowledged frames the sender can send.
4. **Q:** What happens when a frame is lost or corrupted in Go-Back-N?  
   **A:** The sender retransmits **that frame and all subsequent frames** in the current window.
5. **Q:** Why is it called “Go-Back-N”?  
   **A:** Because upon detecting a missing or erroneous frame, the sender **goes back** and retransmits **N frames** starting from the erroneous one.
6. **Q:** What type of acknowledgment is used in Go-Back-N?  
   **A:** **Cumulative acknowledgment** – acknowledges all frames up to the last correctly received frame.
7. **Q:** What is the role of the timer in Go-Back-N?  
   **A:** The timer is started for the first unacknowledged frame. If it expires, all unacknowledged frames are retransmitted.
8. **Q:** What happens to out-of-order frames in Go-Back-N?  
   **A:** They are **discarded** by the receiver since the protocol only accepts in-order frames.
9. **Q:** What is the major disadvantage of Go-Back-N?  
   **A:** When a single frame is lost, **all following frames must be retransmitted**, leading to **wasted bandwidth**.
10. **Q:** How does Go-Back-N improve throughput?  
    **A:** It allows the sender to send **multiple frames continuously** before receiving ACKs, keeping the channel busy and increasing efficiency.
11. **Q:** Which protocol uses Go-Back-N in practice?  
    **A:** **HDLC** uses Go-Back-N ARQ mechanism for error control.
12. **Q:** What is the difference between Go-Back-N and Selective Repeat?  
    **A:** In GBN, all frames after a lost one are retransmitted; in Selective Repeat, only the **specific lost frame** is retransmitted.