CSE351s

Computer Networks

GO-BACK-N PROTOCOL

Faculty Of Engineering
Ain-Shams University

GO-BACK-N Protocol

- A PYTHON IMPLEMENTATION -

TEAM MEMBERS

سكشن *Section*	الكود ID	الاسم Name
2	2100767	عمر مامون حامد
2	2100792	حبيبه السيد موافي
2	2100482	الاء عاطف محمد مصطفى
2	2101016	اسراء عاطف محمد مصطفى

PROJECT WAS DELIVERED AS PART OF
CSE351s | COMPUTER NETWORKS COURSE
SENIOR 1 CSE – FACULTY OF ENGINEERING, AIN SHAMS UNIVERSITY
UNDER SUPERVISION OF **DR. HOSSAM FAHMY** AND **ENG. NOHA WAHDAN**

Python Code

VISIT THE Repository

```python
import random
#test branch
def protocol5(window_size, number_of_frames, message):
    message_after_received = ""
    failed_frames = 0  # Count the no of frames that fail to send
    total_sent_frames = 0  # Total frames sent, including retransmissions
    last_successful_frame = 0  # Tracks the last successfully sent frame

    # Send the first set of frames in the window
    for j in range(1, min(window_size + 1, number_of_frames + 1)):
        frame_data = message[(j - 1) * 2:j * 2]  # Get 2 characters for the frame
        print(f"Sending Frame {j}: Data: {frame_data}, Binary Representation: {convert_to_8bit_binary(frame_data)}")
        total_sent_frames += 1

    i = 1
    while i <= number_of_frames:
        for j in range(i, min(i + window_size, number_of_frames + 1)):
            random_result = random.randint(0, 1)  # Simulate frame sending success or failure

            if random_result:
                # Frame sent successfully
                print(f"Frame {j} sent successfully")
                frame_data = message[(j - 1) * 2:j * 2]
                message_after_received += convert_to_8bit_binary(frame_data)
                last_successful_frame = j

                if j + window_size <= number_of_frames:
                    next_frame_data = message[(j + window_size - 1) * 2: (j + window_size) * 2]
                    print(
                        f"Sending Frame {j + window_size}: Data: {next_frame_data}, Binary Representation: {convert_to_8bit_binary(next_frame_data)}")
                    total_sent_frames += 1

            else:
                # Frame failed, all frames in the current window are resent
                failed_frames += 1
                print(f"Frame {j} execution timeout")
                for k in range(j, min(j + window_size, number_of_frames + 1)):

                    frame_data = message[(k - 1) * 2:k * 2]
                    print(
                        f"Resending Frame {k}: Data: {frame_data}, Binary Representation: {convert_to_8bit_binary(frame_data)}")
                    total_sent_frames += 1
                break  # Exit to resend the current window
        i = last_successful_frame + 1  # Update i to the next frame to send

    # Calculate wire efficiency
```

```
    wire_efficiency = ((total_sent_frames - failed_frames) /
total_sent_frames) * 100

    # Print the statistics after completion
    print("nTransmission complete!")
    print(f"Received Message (binary): {message_after_received}")
    print(f"Decoded Message:
{binary_to_text_8bit(message_after_received)}")
    print(f"Total frames sent (including retransmissions):
{total_sent_frames}")
    print(f"Number of failed frames: {failed_frames}")
    print(f"Wire efficiency: {wire_efficiency:.2f}%")

def convert_to_8bit_binary(string):
    if len(string) == 1:
        string += ''  # Zero-extend if only one character remains
    return ''.join(format(ord(char), '08b') for char in string)

def binary_to_text_8bit(binary_string):
    chars = [binary_string[i:i + 8] for i in range(0, len(binary_string),
8)]
    text = ''.join(chr(int(char, 2)) for char in chars)
    return text

def main():
    message = input("Enter message: ")

    if len(message) % 2 != 0:
        message += ''  # Zero-extend if the length is odd

    number_of_frames = len(message) // 2  # frame consists of 2 characters

    try:
        window_size = int(input("Enter window size: "))
        if window_size <= 0:
            print("Invalid window size")
            return

        if number_of_frames <= 0:
            print("Invalid message length")
            return

        protocol5(window_size, number_of_frames, message)

    except ValueError:  #exception of window_size
        print("Invalid input. Please enter integer values.")  #message to
enter another number
if __name__ == "__main__":
    main()
```

Note:

Copying and running code directly from a PDF may cause **indentation issues**.
We recommend **visiting our repository by clicking the GitHub icon above the code** to download the app files or view the code directly.

Result Visualizations

```
Enter message: Hello CSE351s
Enter window size: 4
Sending Frame 1: Data: He, Binary Representation: 0100100001100101
Sending Frame 2: Data: ll, Binary Representation: 0110110001101100
Sending Frame 3: Data: o , Binary Representation: 0110111100100000
Sending Frame 4: Data: CS, Binary Representation: 0100001101010011
Frame 1 sent successfully
Sending Frame 5: Data: E3, Binary Representation: 0100010100110011
Frame 2 execution timeout
Resending Frame 2: Data: ll, Binary Representation: 0110110001101100
Resending Frame 3: Data: o , Binary Representation: 0110111100100000
Resending Frame 4: Data: CS, Binary Representation: 0100001101010011
Resending Frame 5: Data: E3, Binary Representation: 0100010100110011
Frame 2 sent successfully
Sending Frame 6: Data: 51, Binary Representation: 0011010100110001
Frame 3 sent successfully
Sending Frame 7: Data: s, Binary Representation: 0111001100000000
Frame 4 sent successfully
Frame 5 execution timeout
Resending Frame 5: Data: E3, Binary Representation: 0100010100110011
Resending Frame 6: Data: 51, Binary Representation: 0011010100110001
Resending Frame 7: Data: s, Binary Representation: 0111001100000000
Frame 5 sent successfully
Frame 6 sent successfully
Frame 7 sent successfully

Transmission complete!
Received Message (binary): 0100100001100101011011000110110001101111001000000100001101010011010001010011001100110101001100010111001100000000
Decoded Message: Hello CSE351s
Total frames sent (including retransmissions): 14
Number of failed frames: 2
Wire efficiency: 85.71%
```

Handled Cases:

1. General Input Validation

- **The user provides invalid input for the window size.**

  If invalid, the program prints an error message ("Invalid window size") and exits

- **The input message length is invalid or empty.**

  prints "Invalid message length" and exits

2. Message Encoding

- **The input message has an odd number of characters.**

  Appends a null character ('\0') to the message for proper 2-character framing

3. Frame Transmission

- **On failure.**

  Logs failure ("Frame execution timeout") and resends all frames in the current window.

Conceptual Overview of the Code

1. Sliding Window Protocol:

The sliding window protocol manages the transmission of data frames between a sender and a receiver. Key features include:

- **Window Size:** Controls the number of frames that can be sent before waiting for acknowledgments.
- **Acknowledgment Mechanism:** Ensures successful delivery of frames by retransmitting lost or corrupted frames.

2. Key Functionalities:

- Frame Sending and Retransmission:
  - Frames are sent sequentially, with failures simulated randomly. When a frame fails, all frames within the current window are retransmitted.
- Binary Conversion of Message:
  - Each character in the message is converted to its binary representation for transmission, simulating real-world data encoding.
- Wire Efficiency Calculation:
  - After transmission, the program calculates the efficiency of the process based on the ratio of successful transmissions to total transmissions.

3. Code Components:

- protocol5 Function:
  - Sends frames within the window size.
  - Detects and handles transmission failures.
  - Maintains counters for total frames sent, failed frames, and successfully transmitted frames.
  - Constructs the received message by appending binary data of successfully transmitted frames.

- convert_to_binary Function:

- ✅ Converts characters of the message to their 8-bit binary equivalents for simulation purposes.

- main Function:

- ✅ Takes user input for the message and window size.
- ✅ Validates inputs and initializes the protocol simulation.

4. Simulated Real-World Behavior:

- Random Frame Success/Failure:

The use of random.randint(0, 1) simulates the uncertainty of real-world network transmission.

- Timeout and Retransmission:

If a frame times out (fails), all subsequent frames in the window are retransmitted, mimicking network retransmission protocols.

5. Output and Statistics:

- Received Message:

The program reconstructs the transmitted message from binary data and converts it back to text.

- Efficiency Metrics:

Provides insights into transmission performance, including wire efficiency, total sent frames, and failed frames.

Go Back N Protocol

1. Overview of the System

The Go-Back-N protocol is a reliable data transmission protocol used in network communication. It is a type of Automatic Repeat Request (ARQ) protocol designed to ensure data is delivered accurately and in order, even over unreliable networks. It achieves this by combining sliding window techniques with error detection and retransmission strategies.

2. Key Concepts of Go-Back-N Protocol

- **Sliding Window:**

The sender maintains a window of "N" frames that can be sent without waiting for acknowledgment. This enhances efficiency by allowing continuous transmission until the window limit is reached.

- **Acknowledgments (ACKs):**

The receiver sends an acknowledgment for the last correctly received and sequential frame. If any frame within the window is lost or corrupted, subsequent frames are ignored until the lost frame is retransmitted.

- **Retransmission:**

In case of a failure (e.g., a lost or corrupted frame), the sender "goes back" and retransmits the affected frame and all subsequent frames in the window. This ensures the receiver's buffer remains synchronized with the sender's data stream.

3. How Go-Back-N Works:

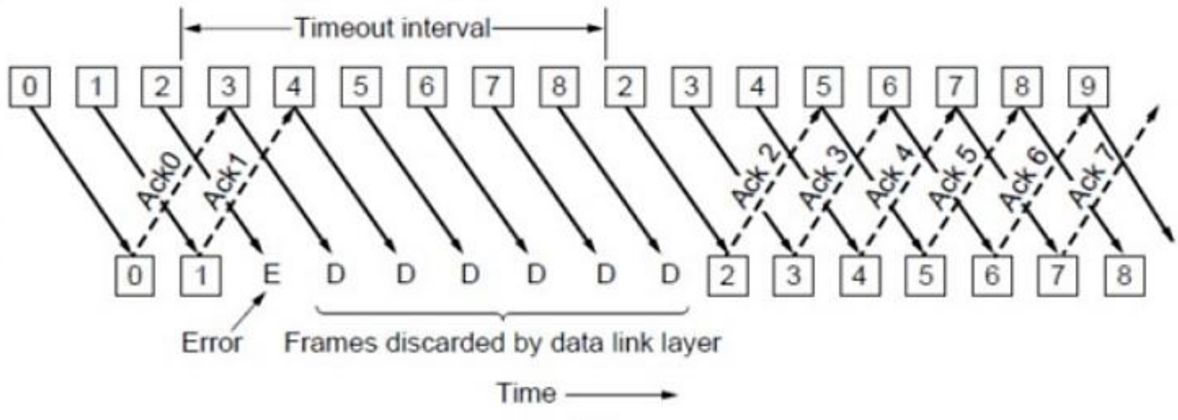
- Sender's Role:

- The **sender** transmits frames sequentially, maintaining a window of N frames.
- It keeps a timer for the oldest unacknowledged frame. If the timer expires (indicating a potential loss), all frames starting from that unacknowledged frame are retransmitted.

- Receiver's Role:

✅ The **receiver** processes frames in order. If a frame is out of sequence, it discards it and does not send an acknowledgment for it.

✅ Once the expected frame is received, it sends an acknowledgment for that frame, signaling the sender to move the window forward.



(a)

4. Advantages:

- **Efficiency:**

(Go Back N) allows multiple frames to be transmitted at once, increasing throughput on high-latency networks.

- **Simplicity:**

The protocol is easy to implement compared to more complex ARQ protocols like Selective Repeat.

5. Limitations:

- **Redundant Transmissions:**

If a single frame is lost, all subsequent frames in the window are retransmitted, even if they were received correctly.

- **Receiver Idle Time:**

The receiver must wait for the retransmitted frames to arrive before proceeding, leading to potential delays.