Computer Networks [CSE351s]

Name	ID	SEC
Mosa Abdulaziz Morgan	2200257	4
Youssef Yacoub Radi Farid	2200649	4
Salma Ramadan Mohamed	2200381	2
Aya Mohamed Mohamed Kamel	2200716	3
Aaesha Mahmod Bedair	2200510	1

Language code : C++

GitHub_Link

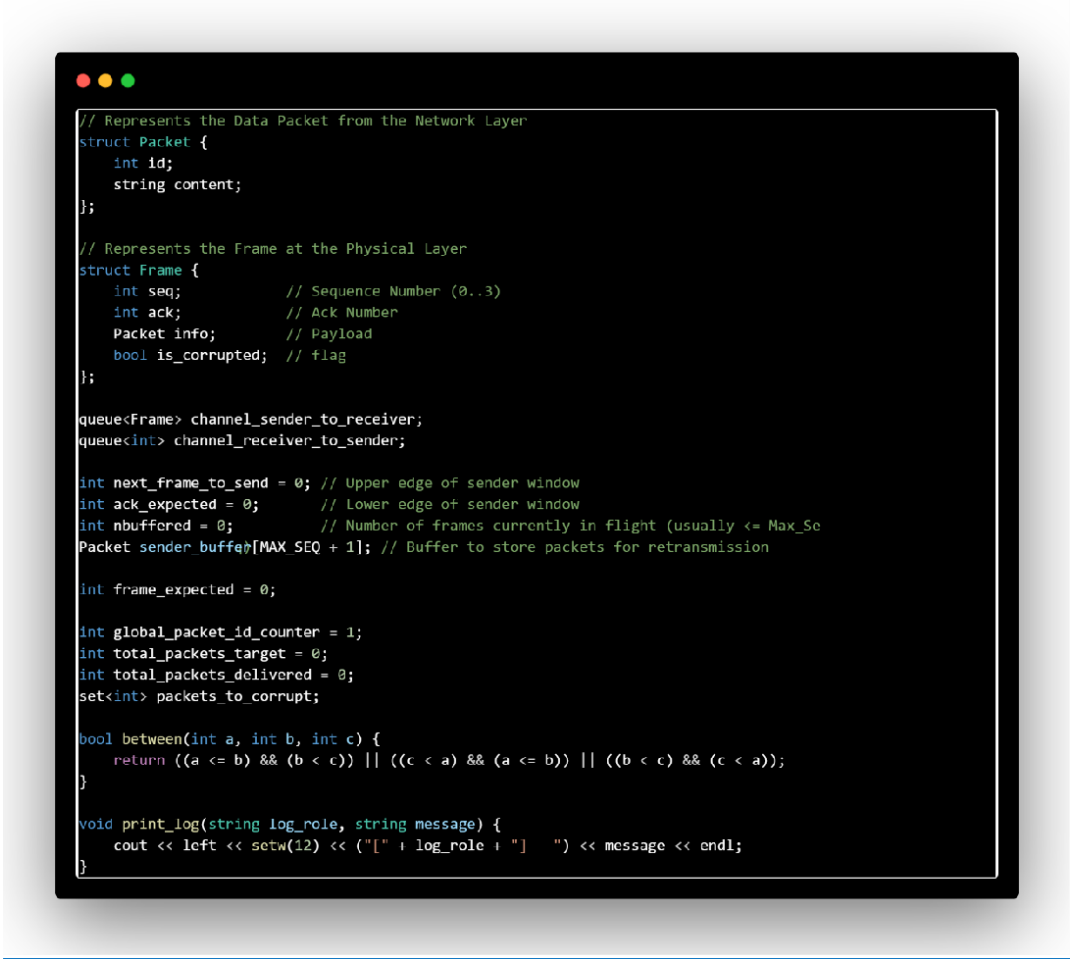
First Assumption

GO_Back_N Protocol

Introduction

In this project, we implemented the first assumption of the Go-Back-N (GBN) protocol using C++. Go-Back-N is a sliding-window protocol used to achieve reliable data transfer. In this assumption, the receiver does not buffer out-of-order frames, so if one frame is lost or has an error, the sender must resend that frame and all the frames after it. Our C++ code simulates how the sender and receiver communicate, handle acknowledgments, and manage errors to ensure the data is delivered correctly and in order.

Code:



```
// Represents the Data Packet from the Network Layer
struct Packet {
    int id;
    string content;
};

// Represents the Frame at the Physical Layer
struct Frame {
    int seq;            // Sequence Number (0..3)
    int ack;            // Ack Number
    Packet info;        // Payload
    bool is_corrupted;  // flag
};

queue<Frame> channel_sender_to_receiver;
queue<int> channel_receiver_to_sender;

int next_frame_to_send = 0; // Upper edge of sender window
int ack_expected = 0;       // Lower edge of sender window
int nbuffered = 0;          // Number of frames currently in flight (usually <= Max_Se
Packet sender_buffer[MAX_SEQ + 1]; // Buffer to store packets for retransmission

int frame_expected = 0;

int global_packet_id_counter = 1;
int total_packets_target = 0;
int total_packets_delivered = 0;
set<int> packets_to_corrupt;

bool between(int a, int b, int c) {
    return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}

void print_log(string log_role, string message) {
    cout << left << setw(12) << ("[" + log_role + "]   ") << message << endl;
}
```

```
void send_data(int frame_nr, int frame_exp, Packet p, bool is_retransmission) {
    Frame s;
    s.seq = frame_nr;
    s.ack = (frame_exp + MAX_SEQ) % (MAX_SEQ + 1);
    s.info = p;
    s.is_corrupted = false;
    if (packets_to_corrupt.count(p.id)) {
        s.is_corrupted = true;
        packets_to_corrupt.erase(p.id);
        string type = is_retransmission ? "Resending" : "Sending";
        print_log("WIRE", ">>> CORRUPTING " + type + " Packet #" + to_string(p.id) + " (Seq " + to_string
(frame_nr) + ") <<<");
    } else {
        string type = is_retransmission ? "Resending" : "Sending";
        print_log("SENDER", type + " Packet #" + to_string(p.id) + " (Seq " + to_string(frame_nr) + ")");
    }

    channel_sender_to_receiver.push(s);
}

// handles the time out
void event_timeout() {
    print_log("TIMER", "!!! TIMEOUT DETECTED for Seq " + to_string(ack_expected) + " !!!");
    print_log("SENDER", "Go-Back-N Triggered: Retransmitting window...");
    int next = ack_expected;
    for (int i = 1; i <= nbuffered; i++) {
        Packet p = sender_buffer[next];
        send_data(next, frame_expected, p, true); // true = is_retransmission
        next = (next + 1) % (MAX_SEQ + 1);
    }
}
// handles when an ACK arrives from the receiver.
void event_ack_arrival(int ack_val) {
    while (between(ack_expected, ack_val, next_frame_to_send)) {
        print_log("SENDER", "Received valid ACK for Seq " + to_string(ack_expected) + ". Window slides.");
        nbuffered--;
        ack_expected = (ack_expected + 1) % (MAX_SEQ + 1);
    }
}

// handles when a frame arrives from the wire.
void event_frame_arrival() {
    if (channel_sender_to_receiver.empty()) return;
    Frame r = channel_sender_to_receiver.front();
    channel_sender_to_receiver.pop();
    if (r.is_corrupted) {
        print_log("RECEIVER", "Frame " + to_string(r.seq) + " damaged. DISCARDING (Silence).");
        return;
    }
    if (r.seq == frame_expected) {
        print_log("RECEIVER", "Frame " + to_string(r.seq) + " Accepted. (Packet #" + to_string(r.info.id)
+ ")");
        total_packets_delivered++;
        frame_expected = (frame_expected + 1) % (MAX_SEQ + 1);
        print_log("RECEIVER", "Sending ACK " + to_string(r.seq));
        channel_receiver_to_sender.push(r.seq);

    } else {
        print_log("RECEIVER", "Frame " + to_string(r.seq) + " Unexpected (Expected " + to_string
(frame_expected) + "). DISCARDING.");
    }
}
```

```cpp
int main() {
    cout << "================================================\n";
    cout << "   GO-BACK-N PROTOCOL SIMULATION (Protocol 5)    \n";
    cout << "================================================\n";
    cout << "Enter total number of frames to send: ";
    cin >> total_packets_target;

    int num_bad;
    cout << "Enter number of frames to corrupt: ";
    cin >> num_bad;

    if (num_bad > 0) {
        for(int i=0; i<num_bad; i++) {
            cout << "Enter a Packet ID to corrupt between 1 and " << total_packets_target << " :";
            int id;
            cin >> id;
            packets_to_corrupt.insert(id);
        }
    }

    cout << "\n[SIM] Simulation Started. Window Size: " << MAX_SEQ << "\n\n";
    int idle_ticks = 0;
    const int TIMEOUT_THRESHOLD = 4;

    while (total_packets_delivered < total_packets_target) {
        bool activity = false;
        if (nbuffered < MAX_SEQ && global_packet_id_counter <= total_packets_target) {
            Packet p;
            p.id = global_packet_id_counter;
            p.content = "Data";
            sender_buffer[next_frame_to_send] = p;
            nbuffered++;
            send_data(next_frame_to_send, frame_expected, p, false); // false = new frame
            next_frame_to_send = (next_frame_to_send + 1) % (MAX_SEQ + 1);
            global_packet_id_counter++;
            activity = true;
        }
        if (!channel_sender_to_receiver.empty()) {
            event_frame_arrival();
            activity = true;
        }
        while (!channel_receiver_to_sender.empty()) {
            int ack = channel_receiver_to_sender.front();
            channel_receiver_to_sender.pop();
            event_ack_arrival(ack);
            activity = true;
        }
        if (activity) {
            idle_ticks = 0; // Reset timer if data moved
        } else {
            if (nbuffered > 0) {
                idle_ticks++;
                if (idle_ticks >= TIMEOUT_THRESHOLD) {
                    event_timeout();
                    idle_ticks = 0; // Reset after triggering
                }
            }
        }
        //simulating the time it takes for frames to travel through the physical layer
        this_thread::sleep_for(chrono::milliseconds(200));
    }
    cout << "\n================================================\n";
    cout << "   TRANSMISSION COMPLETE. ALL PACKETS DELIVERED.  \n";
    cout << "================================================\n";
    return 0;
}
```

Output:

```
================================================
   GO-BACK-N PROTOCOL SIMULATION (Protocol 5)
================================================
Enter total number of frames to send: 10
Enter number of frames to corrupt: 2
Enter a Packet ID to corrupt between 1 and 10 :4
Enter a Packet ID to corrupt between 1 and 10 :6

[SIM] Simulation Started. Window Size: 3

[SENDER]    Sending Packet #1 (Seq 0)
[RECEIVER]   Frame 0 Accepted. (Packet #1)
[RECEIVER]   Sending ACK 0
[SENDER]    Received valid ACK for Seq 0. Window slides.
[SENDER]    Sending Packet #2 (Seq 1)
[RECEIVER]   Frame 1 Accepted. (Packet #2)
[RECEIVER]   Sending ACK 1
[SENDER]    Received valid ACK for Seq 1. Window slides.
[SENDER]    Sending Packet #3 (Seq 2)
[RECEIVER]   Frame 2 Accepted. (Packet #3)
[RECEIVER]   Sending ACK 2
[SENDER]    Received valid ACK for Seq 2. Window slides.
[WIRE]      >>> CORRUPTING Sending Packet #4 (Seq 3) <<<
[RECEIVER]   Frame 3 damaged. DISCARDING (Silence).
[SENDER]    Sending Packet #5 (Seq 0)
[RECEIVER]   Frame 0 Unexpected (Expected 3). DISCARDING.
[WIRE]      >>> CORRUPTING Sending Packet #6 (Seq 1) <<<
[RECEIVER]   Frame 1 damaged. DISCARDING (Silence).
[TIMER]     !!! TIMEOUT DETECTED for Seq 3 !!!
[SENDER]    Go-Back-N Triggered: Retransmitting window...
[SENDER]    Resending Packet #4 (Seq 3)
[SENDER]    Resending Packet #5 (Seq 0)
[SENDER]    Resending Packet #6 (Seq 1)
[RECEIVER]   Frame 3 Accepted. (Packet #4)
[RECEIVER]   Sending ACK 3
[SENDER]    Received valid ACK for Seq 3. Window slides.
[SENDER]    Sending Packet #7 (Seq 2)
[RECEIVER]   Frame 0 Accepted. (Packet #5)
[RECEIVER]   Sending ACK 0
[SENDER]    Received valid ACK for Seq 0. Window slides.
[SENDER]    Sending Packet #8 (Seq 3)
[RECEIVER]   Frame 1 Accepted. (Packet #6)
[RECEIVER]   Sending ACK 1
[SENDER]    Received valid ACK for Seq 1. Window slides.
[SENDER]    Sending Packet #9 (Seq 0)
[RECEIVER]   Frame 2 Accepted. (Packet #7)
[RECEIVER]   Sending ACK 2
[SENDER]    Received valid ACK for Seq 2. Window slides.
[SENDER]    Sending Packet #10 (Seq 1)
[RECEIVER]   Frame 3 Accepted. (Packet #8)
[RECEIVER]   Sending ACK 3
[SENDER]    Received valid ACK for Seq 3. Window slides.
[RECEIVER]   Frame 0 Accepted. (Packet #9)
[RECEIVER]   Sending ACK 0
[SENDER]    Received valid ACK for Seq 0. Window slides.
[RECEIVER]   Frame 1 Accepted. (Packet #10)
[RECEIVER]   Sending ACK 1
[SENDER]    Received valid ACK for Seq 1. Window slides.

================================================
   TRANSMISSION COMPLETE. ALL PACKETS DELIVERED.
================================================
```

Second Assumption

Selective Repeat Protocol

Introduction

In the second assumption of the Go-Back-N protocol, we implemented the version where the receiver has a buffer. In this case, only the lost or corrupted frames need to be resent instead of the entire group. The receiver can store out-of-order frames and wait for the missing ones, which makes the process more efficient. Our C++ code shows how both sides manage their windows, buffering, and acknowledgments to ensure correct data delivery.

Code:

```cpp
const int WINDOW_SIZE = 3;         // The Sender and Receiver Window Size (N)
const int MAX_SEQ = 2 * WINDOW_SIZE - 1; // Sequence Space is 0..5 (2N - 1)

// Represents the Data Packet from the Network Layer
struct Packet {
    int id;
    string content;
};

// Represents the Frame at the Physical Layer
struct Frame {
    int seq;            // Protocol Sequence Number (0..MAX SEQ)
    int ack;            // Acknowledgement Number (Not strictly used in this SR sim, but included)
    Packet info;        // Payload
    bool is_corrupted;  // Simulation flag
};

queue<Frame> channel_sender_to_receiver;
queue<int> channel_receiver_to_sender;

int next_frame_to_send = 0; // Upper edge of sender window + 1
int ack_expected = 0;       // Lower edge of sender window
int nbuffered = 0;          // Number of frames currently in flight (<= WINDOW_SIZE)
Packet sender_buffer[MAX_SEQ + 1];
```

```
bool timer_running[MAX_SEQ + 1] = { false };
int timer_value[MAX_SEQ + 1] = { 0 }; // Timer counter for each frame

int frame_expected = 0;     // Lower edge of receiver window
int next_expected_ack = 0;  // Upper edge of receiver window + 1

Frame receiver_buffer[MAX_SEQ + 1]; // Buffer to store out-of-order frames
bool buffer_filled[MAX_SEQ + 1] = { false }; // Tracks which buffer slots are filled

int global_packet_id_counter = 1;
int total_packets_target = 0;
int total_packets_delivered = 0;
set<int> packets_to_corrupt; // User defined IDs to fail
const int TIMEOUT_THRESHOLD = 4; // Cycles of inactivity before timeout

bool between(int a, int b, int c) {
    return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}

void print_log(string log_role, string message) { //log printing formatter
    cout << left << setw(12) << ("[" + log_role + "]   ") << message << endl;
}

// handles Transmition Function
void send_data(int frame_nr, int frame_exp, Packet p, bool is_retransmission) {
    Frame s;
    s.seq = frame_nr;
    s.ack = (frame_exp + MAX_SEQ) % (MAX_SEQ + 1);
    s.info = p;
    s.is_corrupted = false;
    if (packets_to_corrupt.count(p.id)) {
        s.is_corrupted = true;
        packets_to_corrupt.erase(p.id);
        string type = is_retransmission ? "Resending" : "Sending";
        print_log("WIRE", ">>> CORRUPTING " + type + " Packet #" + to_string(p.id) + " (Seq " + to_string
(frame_nr) + ") <<<");
    }
    else {
        string type = is_retransmission ? "Resending" : "Sending";
        print_log("SENDER", type + " Packet #" + to_string(p.id) + " (Seq " + to_string(frame_nr) + ")");
    }
    timer_running[frame_nr] = true;
    timer_value[frame_nr] = 0;

    channel_sender_to_receiver.push(s);
}
```

```
void check_timers() {
    for (int i = 0; i <= MAX_SEQ; i++) {
        if (timer_running[i]) {
            timer_value[i]++;
            if (timer_value[i] >= TIMEOUT_THRESHOLD) {
                print_log("TIMER", "!!! TIMEOUT DETECTED for Seq " + to_string(i) + " !!!");
                print_log("SENDER", "Selective Retransmission: Retransmitting frame " + to_string(i) +
"...");
                send_data(i, frame_expected, sender_buffer[i], true);
            }
        }
    }
}

// handles Ack Arrival Event
void event_ack_arrival(int ack_val) {
    int upper_edge = (next_frame_to_send + MAX_SEQ + 1) % (MAX_SEQ + 1);
    if (between(ack_expected, ack_val, upper_edge)) {
        print_log("SENDER", "Received valid SACK for Seq " + to_string(ack_val) + ".");
        if (timer_running[ack_val]) {
            timer_running[ack_val] = false;
        }
        if (ack_val == ack_expected) {
            while (nbuffered > 0 && !timer_running[ack_expected]) {
                print_log("SENDER", "Window slides past Seq " + to_string(ack_expected) + ".");
                nbuffered--;
                ack_expected = (ack_expected + 1) % (MAX_SEQ + 1);
            }
        }
    }
}

// handles Frame Arrival Event
void event_frame_arrival() {
    if (channel_sender_to_receiver.empty()) return;
    Frame r = channel_sender_to_receiver.front();
    channel_sender_to_receiver.pop();
    if (r.is_corrupted) {
        print_log("RECEIVER", "Frame " + to_string(r.seq) + " damaged. DISCARDING (Silence).");
        return;
    }
    next_expected_ack = (frame_expected + WINDOW_SIZE) % (MAX_SEQ + 1);
    if (between(frame_expected, r.seq, next_expected_ack)) {
        print_log("RECEIVER", "Sending SACK " + to_string(r.seq));
        channel_receiver_to_sender.push(r.seq);
        if (!buffer_filled[r.seq]) {
            receiver_buffer[r.seq] = r;
            buffer_filled[r.seq] = true;
        }
        while (buffer_filled[frame_expected]) {
            Frame delivered_frame = receiver_buffer[frame_expected];
            print_log("RECEIVER", "Frame " + to_string(frame_expected) + " Delivered. (Packet #" +
to_string(delivered_frame.info.id) + ")");
            total_packets_delivered++;
            buffer_filled[frame_expected] = false;
            frame_expected = (frame_expected + 1) % (MAX_SEQ + 1);
        }

    }
    else {
        print_log("RECEIVER", "Frame " + to_string(r.seq) + " Outside Window (Exp " + to_string
(frame_expected) + "). DISCARDING.");
    }
}
```

```cpp
int main() {
    cout << "================================================\n";
    cout << "  SELECTIVE REPEAT PROTOCOL SIMULATION (SR)      \n";
    cout << "================================================\n";
    cout << "Enter total number of frames to send: ";
    cin >> total_packets_target;

    int num_bad;
    cout << "Enter number of frames to corrupt: ";
    cin >> num_bad;

    if (num_bad > 0) {
        for (int i = 0; i < num_bad; i++) {
            cout << "Enter a Packet ID to corrupt between 1 and " << total_packets_target << " :";
            int id;
            cin >> id;
            packets_to_corrupt.insert(id);
        }
    }

    cout << "\n[SIM] Simulation Started. Window Size: " << WINDOW_SIZE << ", Seq Space: 0-" << MAX_SEQ <<
"\n\n";

    while (total_packets_delivered < total_packets_target) {
        bool activity = false;
        if (nbuffered < WINDOW_SIZE && global_packet_id_counter <= total_packets_target) {
            Packet p;
            p.id = global_packet_id_counter;
            p.content = "Data";
            sender_buffer[next_frame_to_send] = p;
            nbuffered++;
            send_data(next_frame_to_send, frame_expected, p, false);
            next_frame_to_send = (next_frame_to_send + 1) % (MAX_SEQ + 1);
            global_packet_id_counter++;
            activity = true;
        }
        if (!channel_sender_to_receiver.empty()) {
            event_frame_arrival();
            activity = true;
        }
        while (!channel_receiver_to_sender.empty()) {
            int ack = channel_receiver_to_sender.front();
            channel_receiver_to_sender.pop();
            event_ack_arrival(ack);
            activity = true;
        }
        check_timers();
        this_thread::sleep_for(chrono::milliseconds(200));
    }

    cout << "\n================================================\n";
    cout << "   TRANSMISSION COMPLETE. ALL PACKETS DELIVERED.  \n";
    cout << "================================================\n";
    return 0;
}
```

Output:

```
============================================
  SELECTIVE REPEAT PROTOCOL SIMULATION (SR)
============================================
Enter total number of frames to send: 6
Enter number of frames to corrupt: 1
Enter a Packet ID to corrupt between 1 and 6 :3

[SIM] Simulation Started. Window Size: 3, Seq Space: 0-5

[SENDER]    Sending Packet #1 (Seq 0)
[RECEIVER]   Sending SACK 0
[RECEIVER]   Frame 0 Delivered. (Packet #1)
[SENDER]    Received valid SACK for Seq 0.
[SENDER]    Window slides past Seq 0.
[SENDER]    Sending Packet #2 (Seq 1)
[RECEIVER]   Sending SACK 1
[RECEIVER]   Frame 1 Delivered. (Packet #2)
[SENDER]    Received valid SACK for Seq 1.
[SENDER]    Window slides past Seq 1.
[WIRE]      >>> CORRUPTING Sending Packet #3 (Seq 2) <<<
[RECEIVER]   Frame 2 damaged.
[SENDER]    Sending Packet #4 (Seq 3)
[RECEIVER]   Sending SACK 3
[SENDER]    Received valid SACK for Seq 3.
[SENDER]    Sending Packet #5 (Seq 4)
[RECEIVER]   Sending SACK 4
[SENDER]    Received valid SACK for Seq 4.
[TIMER]     !!! TIMEOUT DETECTED for Seq 2 !!!
[SENDER]    Selective Retransmission: Retransmitting frame 2...
[SENDER]    Resending Packet #3 (Seq 2)
[RECEIVER]   Sending SACK 2
[RECEIVER]   Frame 2 Delivered. (Packet #3)
[RECEIVER]   Frame 3 Delivered. (Packet #4)
[RECEIVER]   Frame 4 Delivered. (Packet #5)
[SENDER]    Received valid SACK for Seq 2.
[SENDER]    Window slides past Seq 2.
[SENDER]    Window slides past Seq 3.
[SENDER]    Window slides past Seq 4.
[SENDER]    Sending Packet #6 (Seq 5)
[RECEIVER]   Sending SACK 5
[RECEIVER]   Frame 5 Delivered. (Packet #6)
[SENDER]    Received valid SACK for Seq 5.
[SENDER]    Window slides past Seq 5.

============================================
   TRANSMISSION COMPLETE. ALL PACKETS DELIVERED.
============================================
```