# Assignment-9

**Name: D Abhiraj**
**RollNo : 23CSB0A32**

Sender.cpp

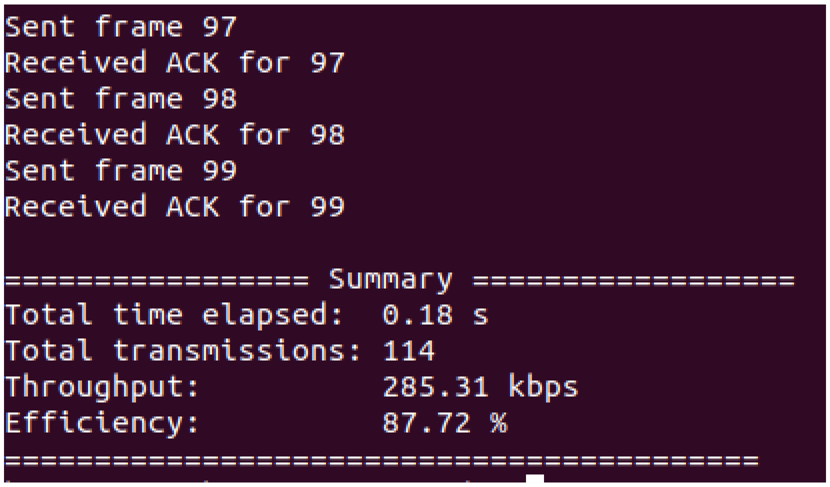
```
#include "protocol_common.h"

using namespace std;


class Receiver : public Socket {
private:
        struct sockaddr_in client_address;

public:
        Receiver();
        void runStopAndWait();
        void runSelectiveRepeat(int window_size);
};

Receiver::Receiver() {
        memset(&server_address, 0, sizeof(server_address));
        server_address.sin_family = AF_INET;
        server_address.sin_port = htons(constants::SERVER_PORT);
        server_address.sin_addr.s_addr = htonl(INADDR_ANY);

        if (bind(socket_fd, (struct sockaddr*)&server_address, sizeof(server_address)) == -1) {
        throw runtime_error("Bind failed");
        }
        cout << "Receiver listening on port " << constants::SERVER_PORT << "..." << endl;
}


void Receiver::runStopAndWait() {
        socklen_t addr_len = sizeof(client_address);
        int expected_sequence = 0;
        int frames_received_count = 0;

        while (frames_received_count < constants::TOTAL_FRAMES) {
        NetworkFrame frame;
        if (recvfrom(socket_fd, &frame, sizeof(frame), 0, (struct sockaddr*)&client_address,
&addr_len) == -1) {
        cerr << "recvfrom() error" << endl;
        continue;
        }
        cout << "Received DATA Frame with sequence " << frame.sequence_number << endl;

        NetworkFrame ack;
        ack.type = FrameType::AckFrame;

        if (frame.sequence_number == expected_sequence) {
        expected_sequence++;
        frames_received_count++;
        }

        ack.sequence_number = expected_sequence - 1;

        if (sendto(socket_fd, &ack, sizeof(ack), 0, (struct sockaddr*)&client_address, addr_len)
== -1) {
        cerr << "sendto() error for ACK" << endl;
        }
        cout << "Sent ACK for sequence " << ack.sequence_number << endl;
        }
}


void Receiver::runSelectiveRepeat(int window_size) {
        socklen_t addr_len = sizeof(client_address);
        int receive_base = 0;
        vector<bool> received_mask(constants::TOTAL_FRAMES, false);

        while (receive_base < constants::TOTAL_FRAMES) {
        NetworkFrame frame;
        if (recvfrom(socket_fd, &frame, sizeof(frame), 0, (struct sockaddr*)&client_address,
&addr_len) == -1) {
        cerr << "recvfrom() error" << endl;
        continue;
        }

        cout << "Received DATA Frame with sequence " << frame.sequence_number << endl;

        NetworkFrame ack;
        ack.type = FrameType::AckFrame;
        ack.sequence_number = frame.sequence_number;

        if (sendto(socket_fd, &ack, sizeof(ack), 0, (struct sockaddr*)&client_address, addr_len)
== -1) {
        cerr << "sendto() error for ACK" << endl;
        }
        cout << "Sent ACK for sequence " << ack.sequence_number << endl;

        if (frame.sequence_number >= receive_base && frame.sequence_number <
receive_base + window_size) {
        received_mask[frame.sequence_number] = true;
        while (receive_base < constants::TOTAL_FRAMES && received_mask[receive_base]) {
                receive_base++;
        }
        }
        }
}

int main(int argc, char* argv[]) {
        if (argc != 3) {
        cerr << "Usage: " << argv[0] << " <protocol: sw|sr> <window_size>" << endl;
        return 1;
        }
        string protocol = argv[1];
        int window_size = stoi(argv[2]);

        try {
        Receiver receiver;
        if (protocol == "sw") {
        receiver.runStopAndWait();
        } else if (protocol == "sr") {
        receiver.runSelectiveRepeat(window_size);
        } else {
        cerr << "Invalid protocol specified." << endl;
        return 1;
        }
        } catch (const runtime_error& e) {
        cerr << "Error: " << e.what() << endl;
        return 1;
        }

        return 0;
}
```

Receiver.cpp

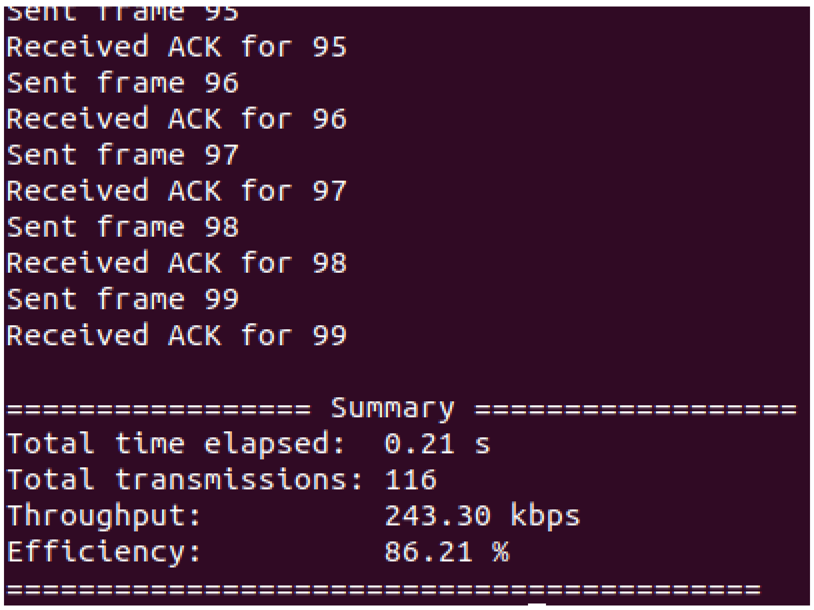
```
#include "protocol_common.h"
using namespace std;
using namespace chrono;
class Sender : public Socket {
```

```
private:
	int loss_probability;
	int total_transmissions = 0;

	void printSummary(const duration<double>& elapsed_time);
	bool shouldDropPacket();

public:
	Sender(int timeout_ms, int loss_prob);
	void executeStopAndWait();
	void executeSelectiveRepeat(int window_size);
};

Sender::Sender(int timeout_ms, int loss_prob) : loss_probability(loss_prob) {
	memset(&server_address, 0, sizeof(server_address));
	server_address.sin_family = AF_INET;
	server_address.sin_port = htons(constants::SERVER_PORT);

	if (inet_aton(constants::SERVER_ADDRESS, &server_address.sin_addr) == 0) {
	throw runtime_error("Invalid server address");
	}

	struct timeval timeout;
	timeout.tv_sec = timeout_ms / 1000;
	timeout.tv_usec = (timeout_ms % 1000) * 1000;
	if (setsockopt(socket_fd, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof(timeout)) <
0) {
	throw runtime_error("Failed to set socket timeout");
	}
	srand(time(nullptr));
}

bool Sender::shouldDropPacket() {
	return (rand() % 100) < loss_probability;
}


void Sender::printSummary(const duration<double>& elapsed_time) {
	double time_seconds = elapsed_time.count();
	double total_bits = constants::TOTAL_FRAMES * constants::PAYLOAD_SIZE * 8;
	double throughput_kbps = (time_seconds > 0) ? (total_bits / time_seconds) / 1000.0 : 0;
	double efficiency = (double)constants::TOTAL_FRAMES / total_transmissions * 100.0;

	cout << "\n================= Summary =================" << endl;
	cout << "Total time elapsed:  " << fixed << setprecision(2) << time_seconds << " s" <<
endl;
	cout << "Total transmissions: " << total_transmissions << endl;
	cout << "Throughput:          " << throughput_kbps << " kbps" << endl;
	cout << "Efficiency:          " << efficiency << " %" << endl;
	cout << "=========================================" << endl;
}


void Sender::executeStopAndWait() {
	int acked_frame_count = 0;
	auto start_time = steady_clock::now();

	while (acked_frame_count < constants::TOTAL_FRAMES) {
	NetworkFrame frame;
	frame.type = FrameType::DataFrame;
	frame.sequence_number = acked_frame_count;

	if (!shouldDropPacket()) {
	sendto(socket_fd, &frame, sizeof(frame), 0, (struct sockaddr*)&server_address,
sizeof(server_address));
	}
	total_transmissions++;
	cout << "Sent frame " << frame.sequence_number << endl;

	NetworkFrame ack;
	if (recvfrom(socket_fd, &ack, sizeof(ack), 0, NULL, NULL) != -1) {
	if (ack.type == FrameType::AckFrame && ack.sequence_number ==
acked_frame_count) {
		cout << "Received ACK for " << ack.sequence_number << endl;
		acked_frame_count++;
	}
	} else {
	cout << "Timeout occurred for frame " << frame.sequence_number << ", retransmitting."
<< endl;
	}
	}
	printSummary(steady_clock::now() - start_time);
}


void Sender::executeSelectiveRepeat(int window_size) {
	int base = 0;
	int next_sequence_num = 0;
```

```
}

        vector<bool> ack_status(constants::TOTAL_FRAMES + window_size, false);
        vector<steady_clock::time_point> sent_times(constants::TOTAL_FRAMES +
window_size);
        auto start_time = steady_clock::now();

        while (base < constants::TOTAL_FRAMES) {
        while (next_sequence_num < base + window_size && next_sequence_num <
constants::TOTAL_FRAMES) {
        NetworkFrame frame;
        frame.type = FrameType::DataFrame;
        frame.sequence_number = next_sequence_num;
        if (!shouldDropPacket()) {
                sendto(socket_fd, &frame, sizeof(frame), 0, (struct sockaddr*)&server_address,
sizeof(server_address));
        }
        cout << "Sent frame " << frame.sequence_number << endl;
        sent_times[next_sequence_num] = steady_clock::now();
        total_transmissions++;
        next_sequence_num++;
        }

        NetworkFrame ack;
        while (recvfrom(socket_fd, &ack, sizeof(ack), 0, NULL, NULL) != -1) {
        if (ack.type == FrameType::AckFrame && ack.sequence_number >= base) {
                cout << "Received ACK for " << ack.sequence_number << endl;
                ack_status[ack.sequence_number] = true;
        }
        }

        for (int i = base; i < next_sequence_num; ++i) {
        if (!ack_status[i] && duration_cast<milliseconds>(steady_clock::now() -
sent_times[i]).count() > 250) {
                NetworkFrame frame;
                frame.type = FrameType::DataFrame;
                frame.sequence_number = i;
                if (!shouldDropPacket()) {
                sendto(socket_fd, &frame, sizeof(frame), 0, (struct sockaddr*)&server_address,
sizeof(server_address));
                }
                cout << "Timeout! Resent frame " << frame.sequence_number << endl;
                sent_times[i] = steady_clock::now();
                total_transmissions++;
        }
        }
        while (base < constants::TOTAL_FRAMES && ack_status[base]) {
        base++;
        }
        }
        printSummary(steady_clock::now() - start_time);
}

int main(int argc, char* argv[]) {
        if (argc != 5) {
        cerr << "Usage: " << argv[0] << " <protocol: sw|sr> <window_size> <timeout_ms>
<loss_percent>" << endl;
        return 1;
        }
        string protocol = argv[1];
        int window_size = stoi(argv[2]);
        int timeout_ms = stoi(argv[3]);
        int loss_percent = stoi(argv[4]);

        try {
        Sender sender(timeout_ms, loss_percent);
        if (protocol == "sw") {
        sender.executeStopAndWait();
        } else if (protocol == "sr") {
        sender.executeSelectiveRepeat(window_size);
        } else {
        cerr << "Invalid protocol specified." << endl;
        return 1;
        }
        } catch (const runtime_error& e) {
        cerr << "Error: " << e.what() << endl;
        return 1;
        }

        return 0;
}
```



```
Sent frame 97
Received ACK for 97
Sent frame 98
Received ACK for 98
Sent frame 99
Received ACK for 99

================= Summary ==================
Total time elapsed:  0.18 s
Total transmissions: 114
Throughput:          285.31 kbps
Efficiency:          87.72 %
==========================================
```



```
Sent frame 95
Received ACK for 95
Sent frame 96
Received ACK for 96
Sent frame 97
Received ACK for 97
Sent frame 98
Received ACK for 98
Sent frame 99
Received ACK for 99

================= Summary ==================
Total time elapsed:  0.21 s
Total transmissions: 116
Throughput:          243.30 kbps
Efficiency:          86.21 %
==========================================
```