

IMPROVED SELECTIVE REPEATE ARQ IMPLEMENTATION

BY - KARTIKAY NARULA (201427), SAMAGRA DVIVEDI (201422)





CONTENT



01 ABOUT THIS PROJECT

02 PROJECT TIMELINE

03 CODE SNIPPETS

04 RESULTS

05 CONCLUSION

ABOUT THE PROJECT

IMPLEMENTATION FOR IMPROVED SELECTIVE REPEAT ARQ

The project aims to implement Improved Selective Repeat ARQ by E.J.Weldon Jr.

The basic idea of the strategy is to repeat NACKed blocks multiple times, with the number of repeats increasing as the receive buffer approaches overflow.

- Level 0 If an ACK is received at the transmitter, send a new block.
- Level 1 If a NACK is received after transmitting a new block, the block is repeated n_1 times.
- Level 2 If all n_1 copies are received in error (corresponding to n_1 NACK's) the block is repeated n_2 times.
- Level q If all n_4 - copies are received in error, buffer overflow occurs. The block is repeated n_4 times.
- Level $q + 1$ If all n_g copies are received in error, buffer overflow occurs. The block is repeated n_4 times.



PROJECT TIMELINE

1. Gathering Information about Selective Repeat ARQ
2. What improvements can be made in Selective Repeat ARQ.
3. Building basic simulator for Selective Repeat ARQ in C++
4. Adding error checks
5. Adding further improvements to the code.

CODE SNIPPETS

```
// Packet class representing a packet
class Packet {
public:
    int seq_num;
    string data;
    Packet(int seq) : seq_num(seq), data("Packet " +
to_string(seq)) {}
};
```



```

class Sender {
private:
    int window_size;
    int max_seq_num;
    int send_base;
    int next_seq_num;
    vector<Packet> window;

    bool wait_for_ack() {
        double timeout = (double)rand() / RAND_MAX + 0.5;
        cout << "Waiting for acknowledgment with timeout " <<
        timeout << "s" << endl;
        this_thread::sleep_for(chrono::milliseconds((int)
        (timeout * 1000)));
        return (rand() % 2) == 1;
    }
    void slide_window() {
        cout << "Sliding window" << endl;
        window.erase(window.begin());
        send_base++;
    }

    void resend_packets() {
        cout << "Resending packets" << endl;
        start_timer();
    }
}

```

```

public:
    Sender(int window, int max_seq) :
    window_size(window), max_seq_num(max_seq),
    send_base(0), next_seq_num(0) {}

    void send_packets() {
        while (send_base < max_seq_num) {
            while (window.size() < window_size &&
            next_seq_num < max_seq_num) {
                Packet packet(next_seq_num++);
                window.push_back(packet);
                cout << "Sending packet with sequence number
                " << packet.seq_num << endl;
            }
            start_timer();
            bool ack_received = wait_for_ack();
            if (ack_received) {
                slide_window();
            } else {
                resend_packets();
            }
        }
    }
}

```



```

class Receiver {
private:
    int max_seq_num;
    int expected_seq_num;
    double error_probability;

    bool is_packet_corrupted() {
        return (double)rand() / RAND_MAX < error_probability;
    }

    void send_ack() {
        cout << "Sending acknowledgment for sequence
number " << expected_seq_num << endl;
    }

public:
    Receiver(int max_seq, double error_prob) :
max_seq_num(max_seq), expected_seq_num(0),
error_probability(error_prob) {}

```

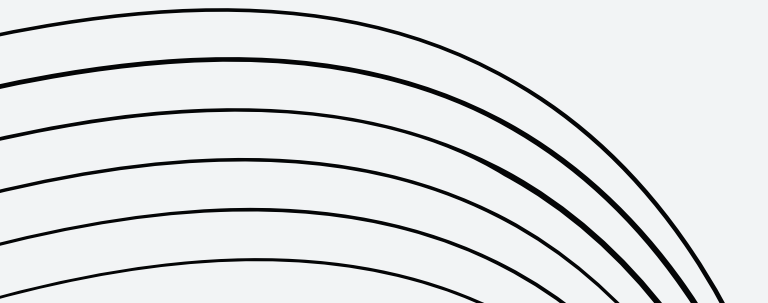
```

void receive_packets() {
    while (expected_seq_num < max_seq_num) {
        int seq_num = rand() % max_seq_num;
        Packet packet(seq_num);
        cout << "Received packet with sequence number "
<< packet.seq_num << endl;
        if (is_packet_corrupted()) {
            cout << "Packet with sequence number " <<
packet.seq_num << " is corrupted" << endl;
            continue;
        }
        if (packet.seq_num == expected_seq_num) {
            cout << "Sending acknowledgment for sequence
number " << expected_seq_num << endl;
            send_ack();
            expected_seq_num++;
        } else {
            cout << "Discarding packet with sequence number "
<< packet.seq_num << endl;
        }
    }
}

```



RESULT AND CONCLUSION

- Multiple repeats should be used if the number of in-flight errors (SP) exceeds three.
 - For channels with very high error probability, the number of repetitions which maximize throughput is significantly greater than one.
 - Regardless of the value of n_1 , throughput is significantly less than channel capacity for error probability > 0.1
 - Generally speaking, error burstiness increases channel capacity. However, throughput does not necessarily increase with capacity; rather, it reflects a system's ability to exploit capacity.
 - The new strategy compares favourably with older continuous - ARQ procedures even with the receive buffer size limited to q (number of levels) = 1
- 

THANK YOU