**For FIFO queing:**

Enable

Config

Interface GigabitEthernet0/0

No fair-queue

Exit exit

ping 192.168.1.10-n 100

**For priority queuing:**

Priority-list 1 protocol ip high tcp 80

Interface GigabitEthernet0/0

Priority-group 1

Exit exit

Write memory

Show interface GigabitEthernet0/0

**For Weighted fair queing:**

Fair-queue

**For ftp:**

ftp ping

put jay.txt

get jay.txt

**For checksum Calculation:**

#include <stdio.h>

// Function to calculate checksum

unsigned int calculateChecksum(int values[], int size) {

unsigned int checksum = 0;

for (int i = 0; i < size; i++) {

int value = values[i];

// Convert negative values to two’s complement in a 4-bit

range

if (value < 0) {

value = (16 + value) % 16;

}

// Ensure positive values are within the 4-bit range

value = value % 16;

// Add value to checksum and apply modulo operation to

ensure it stays within 4-bits

checksum = (checksum + value) % 16;

}

return checksum;

}

// Function to verify checksum at receiver side

4 | Page

int verifyChecksum(int received\_values[], int size, unsigned int

received\_checksum) {

unsigned int calculated\_checksum =

calculateChecksum(received\_values, size);

// Add received checksum to calculated checksum

unsigned int total = (calculated\_checksum + received\_checksum) %

16;

// Check if the result is 0xF (no error detected)

if (total == 0xF) {

return 0; // No error detected

} else {

return 1; // Error detected

}

}

int main() {

// Test data: 2, 3, 18, -6, 9

int values[] = {2, 3, 18, -6, 9};

int size = sizeof(values) / sizeof(values[0]);

// Calculate checksum for the sender side

unsigned int sender\_checksum = calculateChecksum(values, size);

printf("Sender's checksum: %X\n", sender\_checksum);

// Simulate receiver side checksum calculation (with no error)

int received\_values[] = {2, 3, 18, -6, 9};

unsigned int received\_checksum = 7; // Simulated checksum from

the sender

int error = verifyChecksum(received\_values, size,

received\_checksum);

if (error == 0) {

5 | Page

printf("No error detected on receiver side.\n");

} else {

printf("Error detected on receiver side.\n");

}

// Simulate an error (first value changed to 1)

int erroneous\_values[] = {1, 3, 18, -6, 9};

error = verifyChecksum(erroneous\_values, size,

received\_checksum);

if (error == 0) {

printf("No error detected on receiver side.\n");

} else {

printf("Error detected on receiver side.\n");

}

return 0;

}

**Simulation of Checksum Calculations for Text Inputs (C)**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

// Function to compute checksum (16-bit carry-around addition)

unsigned short calculate\_checksum(const char \*data, int length) {

3 | Page

unsigned int sum = 0;

// Process 16-bit blocks

for (int i = 0; i < length; i += 2) {

unsigned short word = (data[i] << 8) & 0xFF00; // High byte

if (i + 1 < length) {

word |= (data[i + 1] & 0x00FF); // Low byte

}

sum += word;

// Carry-around addition (if overflow occurs)

if (sum > 0xFFFF) {

sum = (sum & 0xFFFF) + 1;

}

}

// One's complement of sum

return (unsigned short)(~sum);

}

// Function to verify checksum at the receiver side

int verify\_checksum(const char \*data, int length, unsigned short received\_checksum) {

unsigned short computed\_checksum = calculate\_checksum(data, length);

return (computed\_checksum == received\_checksum);

}

// Function to handle text input

void process\_text\_input() {

char message[100];

char ErrorMessage[100];

// Taking text input

printf("Enter message: ");

fgets(message, sizeof(message), stdin);

// Removing newline character if present

message[strcspn(message, "\n")] = 0;

4 | Page

// Compute checksum for the original message

unsigned short original\_checksum = calculate\_checksum(message, strlen(message));

printf("Computed Checksum for Original Message: 0x%04X\n", original\_checksum);

// Taking error message input

printf("Enter the error-modified message: ");

fgets(ErrorMessage, sizeof(ErrorMessage), stdin);

// Removing newline character if present

ErrorMessage[strcspn(ErrorMessage, "\n")] = 0;

// Compute checksum for the error-modified message

unsigned short modified\_checksum = calculate\_checksum(ErrorMessage, strlen(ErrorMessage));

printf("Computed Checksum for Modified Message: 0x%04X\n", modified\_checksum);

// Compare the checksums to detect error

if (verify\_checksum(message, strlen(message), modified\_checksum)) {

printf("Checksum Verified: No Error Detected\n");

} else {

printf("Checksum Verification Failed: Error Detected!\n");

}

}

// Function to handle decimal input

void process\_decimal\_input() {

int num\_blocks, block\_size;

printf("Enter the block size (in bytes): ");

scanf("%d", &block\_size);

printf("Enter the number of blocks: ");

scanf("%d", &num\_blocks);

char data[100];

printf("Enter the decimal values (e.g., 0x1234 0x5678 ...): ");

for (int i = 0; i < num\_blocks \* block\_size; ++i) {

unsigned short word;

scanf("%hx", &word); // Read as hexadecimal input

data[i] = (char)(word & 0xFF); // Store the low byte

}

5 | Page

// Compute checksum

unsigned short checksum = calculate\_checksum(data, num\_blocks \* block\_size);

printf("Computed Checksum: 0x%04X\n", checksum);

// Simulate an error in the data (changing a byte)

data[3] = 'X'; // Introduce an error

printf("Modified Data (in decimal): ");

for (int i = 0; i < num\_blocks \* block\_size; ++i) {

printf("0x%02X ", data[i] & 0xFF);

}

printf("\n");

// Verify checksum with error

if (verify\_checksum(data, num\_blocks \* block\_size, checksum)) {

printf("Checksum Verified: No Error Detected\n");

} else {

printf("Checksum Verification Failed: Error Detected!\n");

}

}

// Function to handle binary input

void process\_binary\_input() {

int num\_blocks, block\_size;

printf("Enter the block size (in bytes): ");

scanf("%d", &block\_size);

printf("Enter the number of blocks: ");

scanf("%d", &num\_blocks);

char data[100];

printf("Enter the binary values (e.g., 10101010 11001100 ...): ");

for (int i = 0; i < num\_blocks \* block\_size; ++i) {

unsigned short word;

scanf("%hx", &word); // Read as hexadecimal input (for simplicity, using hex representation)

data[i] = (char)(word & 0xFF); // Store the low byte

}

6 | Page

// Compute checksum

unsigned short checksum = calculate\_checksum(data, num\_blocks \* block\_size);

printf("Computed Checksum: 0x%04X\n", checksum);

// Simulate an error in the data (changing a byte)

data[3] = 'X'; // Introduce an error

printf("Modified Data (in binary): ");

for (int i = 0; i < num\_blocks \* block\_size; ++i) {

printf("%08X ", data[i] & 0xFF);

}

printf("\n");

// Verify checksum with error

if (verify\_checksum(data, num\_blocks \* block\_size, checksum)) {

printf("Checksum Verified: No Error Detected\n");

} else {

printf("Checksum Verification Failed: Error Detected!\n");

}

}

// Function to handle hexadecimal input

void process\_hexadecimal\_input() {

int num\_blocks, block\_size;

printf("Enter the block size (in bytes): ");

scanf("%d", &block\_size);

printf("Enter the number of blocks: ");

scanf("%d", &num\_blocks);

char data[100];

printf("Enter the hexadecimal values (e.g., 0x1234 0x5678 ...): ");

for (int i = 0; i < num\_blocks \* block\_size; ++i) {

unsigned short word;

scanf("%hx", &word); // Read as hexadecimal input

data[i] = (char)(word & 0xFF); // Store the low byte

}

7 | Page

// Compute checksum

unsigned short checksum = calculate\_checksum(data, num\_blocks \* block\_size);

printf("Computed Checksum: 0x%04X\n", checksum);

// Simulate an error in the data (changing a byte)

data[3] = 'X'; // Introduce an error

printf("Modified Data (in hexadecimal): ");

for (int i = 0; i < num\_blocks \* block\_size; ++i) {

printf("0x%02X ", data[i] & 0xFF);

}

printf("\n");

// Verify checksum with error

if (verify\_checksum(data, num\_blocks \* block\_size, checksum)) {

printf("Checksum Verified: No Error Detected\n");

} else {

printf("Checksum Verification Failed: Error Detected!\n");

}

}

int main() {

int choice;

printf("Select input type:\n");

printf("1. Text\n");

printf("2. Decimal\n");

printf("3. Binary\n");

printf("4. Hexadecimal\n");

printf("Enter your choice (1/2/3/4): ");

scanf("%d", &choice);

getchar(); // Clear the newline character left in the buffer by scanf

switch (choice) {

case 1:

process\_text\_input();

break;

case 2:

process\_decimal\_input();

break;

case 3:

process\_binary\_input();

break;

8 | Page

case 4:

process\_hexadecimal\_input();

break;

default:

printf("Invalid choice!\n");

}

return 0;

}

**- Simulation of Stop-and-Wait ARQ Protocol**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#define MAX\_RETRANSMISSIONS 5

#define LOSS\_PROBABILITY 0.2 // 20% probability of data frame loss

#define ACK\_LOSS\_PROBABILITY 0.2 // 20% probability of ACK loss

#define ROUND\_TRIP\_TIME 1 // Round-trip time in seconds

#define TIMEOUT 2 // Timeout period for ACK in seconds

#define PACKET\_SIZE 1024 // Size of each packet in bytes (for throughput calculation)

// Function to simulate packet loss

int is\_packet\_lost() {

return rand() % 100 < (LOSS\_PROBABILITY \* 100);

}

// Function to simulate ACK loss

int is\_ack\_lost() {

return rand() % 100 < (ACK\_LOSS\_PROBABILITY \* 100);

}

// Function to simulate sending a packet and waiting for ACK

int send\_packet(int packet\_num) {

int retransmissions = 0;

time\_t start\_time;

int ack\_received = 0;

while (retransmissions < MAX\_RETRANSMISSIONS) {

// Simulate packet sending

printf("Sending packet %d...\n", packet\_num);

3 | Page

// Check if the packet is lost

if (is\_packet\_lost()) {

printf("Packet %d is lost.\n", packet\_num);

retransmissions++;

sleep(ROUND\_TRIP\_TIME); // Simulate time delay for retransmission

continue; // Skip sending ACK and move to retransmission

}

// Simulate waiting for ACK

start\_time = time(NULL);

while (difftime(time(NULL), start\_time) < TIMEOUT) {

if (is\_ack\_lost()) {

printf("ACK for packet %d is lost.\n", packet\_num);

break;

} else {

ack\_received = 1;

break;

}

}

if (ack\_received) {

printf("ACK for packet %d received.\n", packet\_num);

return retransmissions;

} else {

printf("ACK for packet %d not received, resending...\n", packet\_num);

retransmissions++;

sleep(ROUND\_TRIP\_TIME); // Simulate time delay for retransmission

}

}

// If MAX\_RETRANSMISSIONS is reached

printf("Packet %d failed to send after %d attempts.\n", packet\_num, MAX\_RETRANSMISSIONS);

return retransmissions;

}

4 | Page

int main() {

srand(time(NULL)); // Seed for random number generation

// Simulating file input as text data packets

char \*file[] = {"Packet 1", "Packet 2", "Packet 3", "Packet 4", "Packet 5"};

int total\_packets = 5;

int total\_retransmissions = 0;

int total\_sent\_packets = 0;

time\_t start\_time = time(NULL); // Start time for throughput calculation

for (int i = 0; i < total\_packets; i++) {

printf("\n----- Sending %s -----\n", file[i]);

int retransmissions = send\_packet(i + 1);

total\_retransmissions += retransmissions;

total\_sent\_packets++;

}

// Calculate total time taken to send all packets (including retransmissions)

time\_t end\_time = time(NULL);

double total\_time = difftime(end\_time, start\_time);

// Calculate total data sent (size of each packet \* number of packets sent)

double total\_data\_sent = total\_sent\_packets \* PACKET\_SIZE; // in bytes

double total\_time\_in\_seconds = total\_time; // in seconds

// Calculate throughput (in bytes per second)

double throughput = total\_data\_sent / total\_time\_in\_seconds;

// Output results

printf("\nTotal packets sent: %d\n", total\_sent\_packets);

printf("Total retransmissions: %d\n", total\_retransmissions);

printf("Protocol efficiency: %.2f%%\n", ((float)(total\_sent\_packets - total\_retransmissions) /

total\_sent\_packets) \* 100);

printf("Total time taken: %.2f seconds\n", total\_time);

printf("Throughput: %.2f bytes per second\n", throughput);

return 0;

}

**Simulation of Go-Back-N ARQ Protocol**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#define WINDOW\_SIZE 4 // Size of the sliding window

#define LOSS\_PROBABILITY 0.2 // 20% probability of data frame loss

#define ACK\_LOSS\_PROBABILITY 0.2 // 20% probability of ACK loss

#define TIMEOUT 7 // Timeout period in seconds

#define MAX\_RETRANSMISSIONS 5 // Max number of retransmissions allowed

// Hardcoded packet data (simulating the contents of an input file)

char \*packets[] = {

"Packet 1: Hello",

"Packet 2: World",

"Packet 3: This",

"Packet 4: is",

"Packet 5: a",

"Packet 6: Go-Back-N",

"Packet 7: ARQ",

"Packet 8: Protocol"

};

#define NUM\_PACKETS (sizeof(packets) / sizeof(packets[0])) // Number of packets

// Function to simulate packet loss

int is\_packet\_lost() {

return rand() % 100 < (LOSS\_PROBABILITY \* 100);

3 | Page

}

// Function to simulate ACK loss

int is\_ack\_lost() {

return rand() % 100 < (ACK\_LOSS\_PROBABILITY \* 100);

}

// Function to send a packet and handle ACK reception

void send\_packet(int packet\_num, int \*ack\_received\_for\_packet, int \*total\_retransmissions) {

time\_t start\_time;

int ack\_received = 0;

printf("Sending packet %d: %s\n", packet\_num + 1, packets[packet\_num]);

if (is\_packet\_lost()) {

printf("Packet %d is lost.\n", packet\_num + 1);

return;

}

// Simulate waiting for ACK

start\_time = time(NULL);

while (difftime(time(NULL), start\_time) < TIMEOUT) {

if (is\_ack\_lost()) {

printf("ACK for packet %d is lost.\n", packet\_num + 1);

break;

} else {

ack\_received = 1;

printf("ACK received for packet %d.\n", packet\_num + 1);

break;

}

}

if (ack\_received) {

ack\_received\_for\_packet[packet\_num % WINDOW\_SIZE] = 1;

} else {

printf("Timeout for packet %d, resending the window.\n", packet\_num + 1);

(\*total\_retransmissions)++;

4 | Page

}

}

// Function to simulate the Go-Back-N ARQ protocol with sliding window

void go\_back\_n\_arq\_simulation() {

int packet\_num = 0;

int total\_packets\_sent = 0;

int ack\_received\_for\_packet[WINDOW\_SIZE] = {0};

int total\_retransmissions = 0; // Total number of retransmissions

int total\_successful\_packets = 0; // Total number of successfully transmitted packets

time\_t start\_time, end\_time; // For calculating total time taken

// Start the time measurement for total time taken

start\_time = time(NULL);

while (packet\_num < NUM\_PACKETS) {

printf("\n----- Sending Window: Packets %d to %d -----\n", packet\_num + 1, packet\_num +

WINDOW\_SIZE);

// Send packets in the window

for (int i = packet\_num; i < packet\_num + WINDOW\_SIZE && i < NUM\_PACKETS; i++) {

send\_packet(i, ack\_received\_for\_packet, &total\_retransmissions);

total\_packets\_sent++;

}

// Check if all ACKs are received for the current window

int all\_acks\_received = 1;

for (int i = 0; i < WINDOW\_SIZE; i++) {

if (ack\_received\_for\_packet[i] == 0) {

all\_acks\_received = 0;

break;

}

}

if (all\_acks\_received) {

printf("All ACKs received for current window. Proceeding to next window.\n");

total\_successful\_packets += WINDOW\_SIZE;

5 | Page

} else {

printf("Some packets in the window were not acknowledged, retrying...\n");

}

// Reset the ACK reception for the window

for (int i = 0; i < WINDOW\_SIZE; i++) {

ack\_received\_for\_packet[i] = 0;

}

packet\_num += WINDOW\_SIZE; // Move the window forward

sleep(1); // Simulate delay before next window

}

// End the time measurement for total time taken

end\_time = time(NULL);

// Calculate protocol efficiency, total time taken, and throughput

double total\_time\_taken = difftime(end\_time, start\_time);

double protocol\_efficiency = ((double)total\_successful\_packets / (double)total\_packets\_sent) \*

100;

double throughput = (double)total\_successful\_packets / total\_time\_taken; // Packets per second

// Print metrics

printf("\nTotal packets sent: %d\n", total\_packets\_sent);

printf("Total retransmissions: %d\n", total\_retransmissions);

printf("Protocol Efficiency: %.2f%%\n", protocol\_efficiency);

printf("Total time taken: %.2f seconds\n", total\_time\_taken);

printf("Throughput: %.2f packets per second\n", throughput);

}

int main() {

srand(time(NULL)); // Seed the random number generator

go\_back\_n\_arq\_simulation();

return 0;

}

**Simulation of Selective Repeat ARQ Protocol**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <unistd.h>

#define WINDOW\_SIZE 4 // Size of the sliding window

#define LOSS\_PROBABILITY 0.2 // Probability of data packet loss

#define ACK\_LOSS\_PROBABILITY 0.2 // Probability of ACK loss

#define TIMEOUT 2 // Timeout period in seconds for retransmission

#define MAX\_RETRANSMISSIONS 5 // Max retransmissions allowed for lost packets

// Hardcoded data packets (simulated)

char \*packets[] = {

"Packet 1: Hello",

"Packet 2: World",

"Packet 3: This",

"Packet 4: is",

"Packet 5: a",

"Packet 6: Selective",

"Packet 7: Repeat",

"Packet 8: ARQ"

};

#define NUM\_PACKETS (sizeof(packets) / sizeof(packets[0])) // Number of packets

// Function to simulate packet loss

3 | Page

int is\_packet\_lost() {

return rand() % 100 < (LOSS\_PROBABILITY \* 100);

}

// Function to simulate ACK loss

int is\_ack\_lost() {

return rand() % 100 < (ACK\_LOSS\_PROBABILITY \* 100);

}

// Function to simulate sending a packet and handling ACK reception

void send\_packet(int packet\_num, int \*ack\_received\_for\_packet, int \*total\_retransmissions) {

time\_t start\_time;

int ack\_received = 0;

printf("Sending packet %d: %s\n", packet\_num + 1, packets[packet\_num]);

if (is\_packet\_lost()) {

printf("Packet %d is lost.\n", packet\_num + 1);

return;

}

// Simulate waiting for ACK

start\_time = time(NULL);

while (difftime(time(NULL), start\_time) < TIMEOUT) {

if (is\_ack\_lost()) {

printf("ACK for packet %d is lost.\n", packet\_num + 1);

break;

} else {

ack\_received = 1;

printf("ACK received for packet %d.\n", packet\_num + 1);

break;

}

}

if (ack\_received) {

ack\_received\_for\_packet[packet\_num] = 1;

} else {

4 | Page

printf("Timeout for packet %d, resending the packet.\n", packet\_num + 1);

(\*total\_retransmissions)++;

}

}

// Function to simulate the Selective Repeat ARQ protocol with sliding window and retransmission

void selective\_repeat\_arq\_simulation() {

int packet\_num = 0;

int total\_packets\_sent = 0;

int ack\_received\_for\_packet[NUM\_PACKETS] = {0}; // Track ACKs for each packet

int retransmissions[NUM\_PACKETS] = {0}; // Track retransmissions

int total\_retransmissions = 0; // Total retransmissions

int total\_successful\_packets = 0; // Total successful packets

time\_t start\_time, end\_time;

// Start the time measurement for total time taken

start\_time = time(NULL);

while (packet\_num < NUM\_PACKETS) {

printf("\n----- Sending Window: Packets %d to %d -----\n", packet\_num + 1, packet\_num +

WINDOW\_SIZE);

// Send packets in the current window

for (int i = packet\_num; i < packet\_num + WINDOW\_SIZE && i < NUM\_PACKETS; i++) {

send\_packet(i, ack\_received\_for\_packet, &total\_retransmissions);

total\_packets\_sent++;

}

// Handle retransmissions for lost packets

for (int i = packet\_num; i < packet\_num + WINDOW\_SIZE && i < NUM\_PACKETS; i++) {

if (ack\_received\_for\_packet[i] == 0 && retransmissions[i] < MAX\_RETRANSMISSIONS) {

printf("Retransmitting packet %d: %s\n", i + 1, packets[i]);

retransmissions[i]++;

total\_retransmissions++;

send\_packet(i, ack\_received\_for\_packet, &total\_retransmissions);

}

5 | Page

}

// Check if all ACKs are received for the current window

int all\_acks\_received = 1;

for (int i = packet\_num; i < packet\_num + WINDOW\_SIZE && i < NUM\_PACKETS; i++) {

if (ack\_received\_for\_packet[i] == 0) {

all\_acks\_received = 0;

break;

}

}

if (all\_acks\_received) {

printf("All ACKs received for the current window. Proceeding to next window.\n");

total\_successful\_packets += WINDOW\_SIZE;

} else {

printf("Some packets in the window were not acknowledged, retrying...\n");

}

// Move the window forward by the number of packets in the window

packet\_num += WINDOW\_SIZE;

sleep(1); // Simulate delay before the next window

}

// End the time measurement for total time taken

end\_time = time(NULL);

// Calculate protocol efficiency, total time taken, and throughput

double total\_time\_taken = difftime(end\_time, start\_time);

double protocol\_efficiency = ((double)total\_successful\_packets / (double)total\_packets\_sent) \*

100;

double throughput = (double)total\_successful\_packets / total\_time\_taken; // Packets per second

printf("\nTotal packets sent: %d\n", total\_packets\_sent);

printf("Total retransmissions: %d\n", total\_retransmissions);

printf("Protocol Efficiency: %.2f%%\n", protocol\_efficiency);

printf("Total time taken: %.2f seconds\n", total\_time\_taken);

printf("Throughput: %.2f packets per second\n", throughput);

6 | Page

}

int main() {

srand(time(NULL)); // Seed the random number generator

selective\_repeat\_arq\_simulation();

return 0;

}