# Experiment 7: DLL ARQ Mechanisms using TCP Sockets

**Aim:** To use TCP Sockets to implement the ARQ mechanisms at the Data Link Layer

**Objective:** After carrying out this experiment, students will be able to:

* Apply TCP Socket programming technique to implement the ARQ mechanisms at the Data Link Layer

**Problem statement:** You are required to write programs to implement a TCP based server that receives frames sent to it by a client. The functionality of this server is that it should echo any data it receives from a client back to it. You are required to implement stop and wait, go back N and selective repeat ARQ mechanisms. Consider that you have to transmit and receive a total of 20 frames using WT=WR=1, WT=5 and WR=1 and WT=WR=5 for stop and wait, go back N and selective repeat respectively.

**Analysis:** While analyzing your program, you are required to address the following points:

* How does the functionality of the program differ when you have the accept() function call at the server within an infinite loop as opposed to having it outside?

**MARKS DISTRIBUTION**

|  |  |  |
| --- | --- | --- |
| **Component** | **Maximum Marks** | **Marks Obtained** |
| Preparation of Document | 7 |  |
| Results | 7 |  |
| Viva | 6 |  |
| **Total** | **20** |  |

Submitted by:

Register No:

1. Algorithm/Flowchart

Go-Back-N:

|  |
| --- |
| N = window size |
|  | Sn = sequence number |
|  | Sb = sequence base |
|  | Sm = sequence max |
|  | ack = ack number |
|  | nack = first non acknowledged |
|  | Receiver: |
|  | Do the following forever: |
|  | Randomly accept or reject packet |
|  |  |
|  | If the packet received and the packet is error free |
|  | Accept packet |
|  | Send a positive ack for packet |
|  | Else |
|  | Refuse packet |
|  | Send a negative ack for packet |
|  |  |
|  | Sender: |
|  | Sb = 0 |
|  | Sm = N − 1 |
|  | ack = 0 |
|  | Repeat the following steps forever: |
|  | Send packet with ack |
|  | If positively ack is recieved: |
|  | ack++ |
|  | Transmit a packet where Sb <= ack <= Sm. |
|  | packets are transmitted in order |
|  | Else |
|  | Enqueue the nack into the queue |
|  | //check if last packet in the window is sent |
|  | if(ack==Sm) |
|  | if(queue is not empty) |
|  | // start from the first nack packet |
|  | nack = queue.front(); |
|  | empty the queue |
|  | ack = nack |
|  |  |
|  | Sm = Sm + (ack − Sb) |
|  | Sb = ack |

Selective Repeat:

|  |
| --- |
| N = window size |
|  | Sn = sequence number |
|  | Sb = sequence base |
|  | Sm = sequence max |
|  | ack = ack number |
|  | nack = first non acknowledged |
|  | Receiver: |
|  | Do the following forever: |
|  | Randomly accept or reject packet |
|  |  |
|  | If the packet received and the packet is error free |
|  | Accept packet |
|  | Send a positive ack for packet |
|  | Else |
|  | Refuse packet |
|  | Send a negative ack for packet |
|  |  |
|  | Sender: |
|  | Sb = 0 |
|  | Sm = N − 1 |
|  | ack = 0 |
|  | Repeat the following steps forever: |
|  | If the packet was not already positively acknowledged by receiver |
|  | Send packet with ack |
|  | If positively ack is recieved: |
|  | Transmit a packet where Sb <= ack <= Sm. |
|  | packets are transmitted in order |
|  | Else |
|  | Enqueue the nack into the queue |
|  | ack++ |
|  | //check if last packet in the window is sent |
|  | if(ack==Sm) |
|  | if(queue is not empty) |
|  | // start from the first nack packet |
|  | nack = queue.front(); |
|  | empty the queue |
|  | ack = nack |
|  |  |
|  | Sm = Sm + (ack − Sb) |
|  | Sb = ack |

1. Program

**Client:**

// Client side C/C++ program to demonstrate Socket programming

#include <arpa/inet.h>

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#define PORT 8080

#include <iostream>

int main(int argc, char const \*argv[]) {

    std::cout << "Enter the ARQ Mechanism to use " << std::endl;

    std::cout

        << "1.Stop And Wait\t2.Go Back N\3.Selective Repeat\nYour Choice : "

        << std::endl;

    int choice;

    std::cin >> choice;

    std::cin.ignore();

    int sockfd = 0, valread;

    struct sockaddr\_in serv\_addr;

    if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

        printf("\n Socket creation error \n");

        exit(EXIT\_FAILURE);

    } else {

        std::cout << argv[0] << " : SOCKET CREATED " << std::endl;

    }

    serv\_addr.sin\_family = AF\_INET;

    serv\_addr.sin\_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary form

    if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) {

        printf("\nInvalid address/ Address not supported \n");

        return -1;

    } else {

        std::cout << argv[0] << " : ADDRESS CONVERTED" << std::endl;

    }

    if (connect(sockfd, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

        printf("\nConnection Failed \n");

        return -1;

    } else {

        std::cout << argv[0] << " : CONNECTION ESTABLISHED" << std::endl;

    }

    char buffer[1024] = {0};

    // create the data (20frames)

    std::string frames("");

    for (int i = 0; i < 20; i++) frames.push\_back(std::rand() % 10 + '0');

    std::cout << argv[0] << " : FRAMES TO TRANSMIT\n" << frames << std::endl;

    switch (choice) {

        case 1: {

        } break;

        case 2: {

            int windowSize;

            std::cout << "Enter the window size : ";

            std::cin >> windowSize;

            std::cin.ignore();

            int N = windowSize;

            std::string::iterator start = frames.begin();

            std::string::iterator end = frames.begin() + N >= frames.end()

                                            ? frames.end()

                                            : frames.begin() + N;

            while (true) {

                std::string to\_transmit(start, end);

                std::cout << "TRANSMITTING : " << to\_transmit << std::endl;

                if (to\_transmit.empty()) {

                    break;

                }

                write(sockfd, to\_transmit.c\_str(), to\_transmit.size());

                bzero(buffer, sizeof(buffer));

                read(sockfd, buffer, sizeof(buffer));

                std::string recieved(buffer);

                std::cout << "SERVER RECIEVED : " << recieved << std::endl;

                // error has occured

                int idx = recieved.size();

                start += idx;

                end = start + N >= frames.end() ? frames.end() : start + N;

            }

        } break;

        case 3: {

        } break;

        default: {

            send(sockfd, "EXIT", sizeof("EXIT"), 0);

            close(sockfd);

            exit(EXIT\_SUCCESS);

        }

    }

    close(sockfd);

    exit(EXIT\_SUCCESS);

    return 0;

}

**Server:**

#include <netinet/in.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#define PORT 8080

#include <algorithm>

#include <fstream>

#include <iostream>

#include <vector>

int main(int argc, char const \*argv[]) {

    int server\_fd, new\_socket, valread;

    struct sockaddr\_in address;

    int opt = 1;

    int addrlen = sizeof(address);

    // Creating socket file descriptor

    if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

        perror("socket failed");

        exit(EXIT\_FAILURE);

    } else {

        std::cout << argv[0] << " : "

                  << "SOCKET CREATED SUCCESSFULLY" << std::endl;

    }

    // Forcefully attaching socket to the port 8080

    if (setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR | SO\_REUSEPORT, &opt,

                   sizeof(opt))) {

        perror("setsockopt");

        exit(EXIT\_FAILURE);

    }

    address.sin\_family = AF\_INET;

    address.sin\_addr.s\_addr = INADDR\_ANY;

    address.sin\_port = htons(PORT);

    // Forcefully attaching socket to the port 8080

    if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

        perror("bind failed");

        exit(EXIT\_FAILURE);

    } else {

        std::cout << argv[0] << " : "

                  << "BIND SUCCESSFUL TO PORT " << PORT << std::endl;

    }

    if (listen(server\_fd, 3) < 0) {

        perror("listen");

        exit(EXIT\_FAILURE);

    } else {

        std::cout << argv[0] << " : "

                  << "NOW LISTENING" << std::endl;

    }

    if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address,

                             (socklen\_t \*)&addrlen)) < 0) {

        perror("accept");

        exit(EXIT\_FAILURE);

    } else {

        std::cout << argv[0] << " : CLIENT ACCEPTED" << std::endl;

    }

    char buffer[1024] = {0};

    // fetch the operation from the client

    while (true) {

        bzero(buffer, 1024);

        valread = read(new\_socket, buffer, sizeof(buffer));

        std::string operation(buffer);

        if (operation.empty()) {

            break;

        }

        std::cout << argv[0] << " : RECIEVED : " << buffer << std::endl;

        std::cout << "Enter the index of the byte from above frames which had "

                     "error\nenter -1 if there was no error :  ";

        int idx;

        std::cin >> idx;

        std::cin.ignore();

        std::string recieved(buffer);

        if (idx != -1) {

            recieved = recieved.substr(0, idx);

        }

        write(new\_socket, recieved.c\_str(), recieved.size());

    }

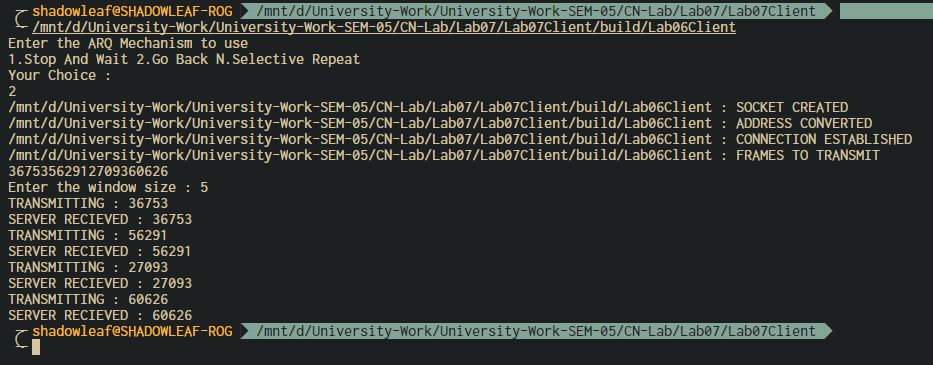
    // Close the Server File Descriptor

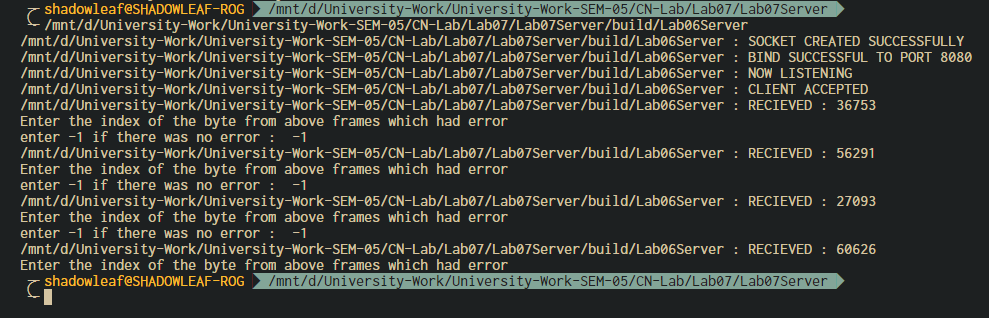
    close(server\_fd);

    return 0;

}

1. Results





1. Analysis and Discussions

accept - accept a new connection on a socket

#include <[sys/socket.h](https://pubs.opengroup.org/onlinepubs/009695399/basedefs/sys/socket.h.html)>  
  
int accept(int *socket*, struct sockaddr \*restrict *address*,  
       socklen\_t \*restrict *address\_len*);

The *accept*() function shall extract the first connection on the queue of pending connections, create a new socket with the same socket type protocol and address family as the specified socket, and allocate a new file descriptor for that socket.

Calling the accept function in an infinite loop will keep creating sockets for every new connection if possible, otherwise it will fail.

1. Conclusions

Stop and Wait –

The sender sends the packet and waits for the ACK (acknowledgement) of the packet. Once the ACK reaches the sender, it transmits the next packet in row. If the ACK is not received, it re-transmits the previous packet again.

Go Back N –

The sender sends N packets which is equal to the window size. Once the entire window is sent, the sender then waits for a cumulative ACK to send more packets. On the receiver end, it receives only in-order packets and discards out-of-order packets. As in case of packet loss, the entire window would be re-transmitted.

Selective Repeat –

The sender sends packet of window size N and the receiver acknowledges all packet whether they were received in order or not. In this case, the receiver maintains a buffer to contain out-of-order packets and sorts them. The sender selectively re-transmits the lost packet and moves the window forward.

| **PROPERTIES** | **STOP AND WAIT** | **GO BACK N** | **SELECTIVE REPEAT** |
| --- | --- | --- | --- |
| Sender window size | 1 | N | N |
| Receiver Window size | 1 | 1 | N |
| Minimum Sequence number | 2 | N+1 | 2N |
| Efficiency | 1/(1+2\*a) | N/(1+2\*a) | N/(1+2\*a) |
| Type of Acknowledgement | Individual | Cumulative | Individual |
| Supported order at Receiving end | – | In-order delivery only | Out-of-order delivery as well |
| Number of retransmissions in case of packet drop | 1 | N | 1 |

1. Comments
   1. Limitations of the experiment

The experiment is limited to 20 packets, which is very learn to simulate an ARQ mechanism for comparison.

* 1. Limitations of the results obtained

The results are obtained for very low number of packets, hence the packet dropping cannot be simulated well enough to learn the advantages and disadvantages of the individual algorithms.

* 1. Learning

We learnt how to implement sliding window ARQ mechanism using TCP Socket Programming.

* 1. Recommendations

Instead of reading the input from STDIN, introduce random packet drops from a poisson distribution to simulate a proper network, which will show the proper advantages of the different algorithms under different conditions.