Network Lab: Sliding Window Protocol

V Venkataraman 106118106

server.c

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#define PORT 8080
typedef enum{ DATA,ACK }   MSGKIND;
struct timeval timeout;

struct MESSAGE
{
    MSGKIND type;
    int seq;
    unsigned int len;
    int msg;
};
```

```c
int main(int argc, char *argv[])
{
    timeout.tv_sec = 1;
    timeout.tv_usec = 0;
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
    {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    if (setsockopt(server_fd, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout, sizeof(timeout))
< 0)
    {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons( PORT );

    if (bind(server_fd, (struct sockaddr *)&address,
                                 sizeof(address))<0)
    {
```

```
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    if (listen(server_fd, 3) < 0)
    {
        perror("listen");
        exit(EXIT_FAILURE);
    }
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
                        (socklen_t*)&addrlen))<0)
    {
        perror("accept");
        exit(EXIT_FAILURE);
    }
    if (setsockopt(new_socket, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout, sizeof(timeout))
 < 0)
    {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }
```

```
        int serv_win = 3;
        int flag=1, s=0;
        int count=0;
        int i=0;
        int max=10;
        int num=0;
```

```
        if(argc>2)
        exit(0);
        printf(argv[1]);
        if(strcmp(argv[1],"1")==0)
        {
        //Go back N
        printf("Go back n\n");
        int count=0;
        while(count<max)
        {
```

```
            int right= s+serv_win;
            for(int count=s;count<right;count++)
            {
                //Send all messages in window
                num++;
                struct MESSAGE* Message = (struct MESSAGE*) malloc(sizeof(struct MESSAGE))
;
                struct MESSAGE* Acknowledge = (struct MESSAGE*) malloc(sizeof(struct MESSA
GE));
                Message->type = DATA;
                Message->len = 1;
                Message->msg=count;
                Message->seq = count;
                // if(num!=2)
```

```
                {
                    send(new_socket,(void*)Message, sizeof(struct MESSAGE), 0);
                    printf("MSG: %d\n",count);
                }
                if(recv(new_socket, Acknowledge, sizeof(struct MESSAGE), 0) > 0)
                {
                    printf("ACK: %d\n",Acknowledge->seq);
                    if(Acknowledge->type == ACK && Acknowledge->seq == s)
                    {
                        sleep(1);
                        //move winddow left limit
                        s++;
                        if(count>max)
                        break;
                    }
                }
```

```
            }
        }
        }
        else if(strcmp(argv[1],"2")==0)
        //Selective Repeat
        {
            printf("Selective Repeat\n");
            int buffer[serv_win];
            memset(buffer,0,sizeof(buffer));
            int num=0;
            int right= s+serv_win;
            while(count<max)
            {

            for(int count=s;count<right;count++)
            {
                num++;
                if(count>max)
                break;
```

```
                //Send message only if not acknowledged
                if( count>=s && buffer[count-s]==0)
                {
                struct MESSAGE* Message = (struct MESSAGE*) malloc(sizeof(struct MESSAGE))
;
                Message->type = DATA;
                Message->len = 1;
                Message->msg=count;
                Message->seq = count;
                    // if(num!=2)
                    {
                    send(new_socket,(void*)Message, sizeof(struct MESSAGE), 0);
                    printf("MSG: %d s=%d\n",count,s);
                    }
                }
```

```
                //Receive ACK
```

```
                struct MESSAGE* Acknowledge = (struct MESSAGE*) malloc(sizeof(struct MESSA
GE));
                if(recv(new_socket, Acknowledge, sizeof(struct MESSAGE), 0) > 0)
                {
                    printf("ACK: %d\n",Acknowledge->seq);
                    if(Acknowledge->type == ACK && Acknowledge->seq >=s && Acknowledge->se
q < s+serv_win && buffer[Acknowledge->seq-s]==0)
                    {
                        sleep(1);
                        //Window element acknowledged
                        buffer[Acknowledge->seq-s]=1;
                    }
                }
```

```
                int flag=1;
                for(int p=0;p<serv_win;p++)
                {
                    if(buffer[p]==0)
                    {
                        flag=-1;
                        break;
                    }
                }
                if(flag==1)
                {
                    s+=serv_win;
                    right+=serv_win;
                    memset(buffer,0,sizeof(buffer));
                }
            }
            }
        }

    close(new_socket);
    return 0;
}
```

client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#define PORT 8080
typedef enum{ DATA,ACK }   MSGKIND;


struct MESSAGE
{
    MSGKIND type;
```

```
    int seq;
    unsigned int len;
    int msg;
};
```

```
int main(int argc, char *argv[])
{
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char buffer[1024] = {0};
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0)
    {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        printf("\nConnection Failed \n");
        return -1;
    }

            int s=0;
            char null[1]={'N'};
            int i=0;
            int count=0;
            int max=10;
            int recv_win=3;
    if(argc>2)
        exit(0);
    printf(argv[1]);
    if(strcmp(argv[1],"1")==0)
    //Go Back n
    {
        printf("Goback n\n");
        int num=0;
        int temp;
            while(s<max)
            {
                num++;
                struct MESSAGE* Message = (struct MESSAGE*) malloc(sizeof(struct MESSAGE))
;
                if(recv(sock, Message, sizeof(struct MESSAGE), 0) > 0)
                    {
                        if(Message->type == DATA && Message->seq == s)
```

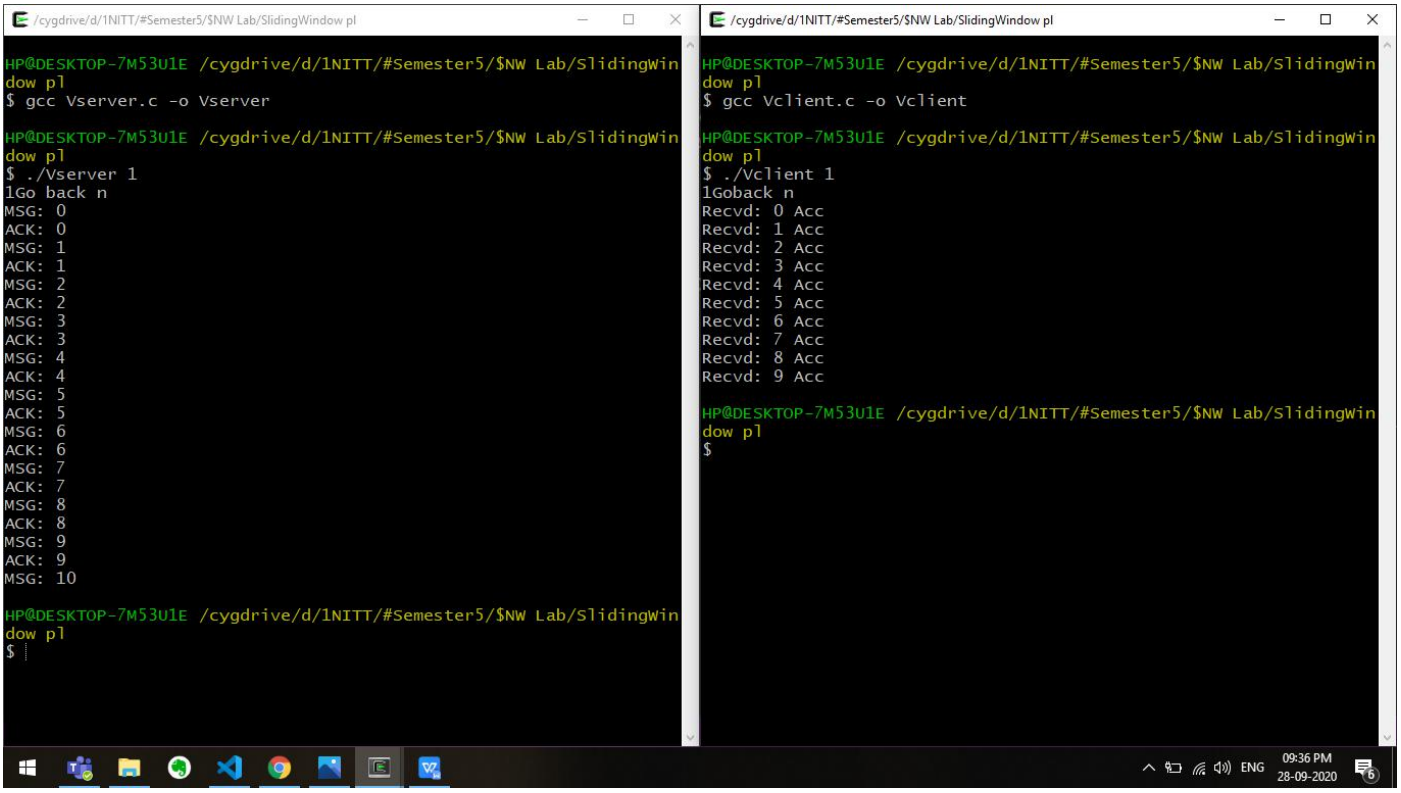
```
                        {
                            printf("Recvd: %d Acc\n",Message->seq);
                            s++;
                            temp=s;
                        }
                        else
                        {
                            printf("Recvd: %d Disc expecting %d\n",Message->seq,s);
                            if(Message->seq<s)
                            {
                                temp=s;
                                s=Message->seq;
                            }
                        }

                    }
                struct MESSAGE* Acknowledge = (struct MESSAGE*) malloc(sizeof(struct MESSA
GE));
                Acknowledge->type = ACK;
                Acknowledge->len = 0;
                Acknowledge->msg=-1;
                Acknowledge->seq=s-1;

                send(sock,(void*)Acknowledge, sizeof(struct MESSAGE), 0);
```

```
                if(temp!=s)
                s=temp;
            }
    }
    else if(strcmp(argv[1],"2")==0)
    {
    // Selective repeat
    printf("Selective Repeat\n");
        int ind=0;
        int temp;
        count=0;
        int buffer[recv_win];
        memset(buffer,0,sizeof(buffer));
        //for(int count=s;count<s+recv_win;s++)
        int left=0;
        int right=recv_win-1;
        int num=0;
        while(left<max)
        {
                num++;
                struct MESSAGE* Message = (struct MESSAGE*) malloc(sizeof(struct MESSAGE));

                if(recv(sock, Message, sizeof(struct MESSAGE), 0) > 0)
                    {
                        if(Message->type == DATA && Message->seq >= left && Message->seq <
= right && buffer[Message->seq-left]==0)
                        {
                            printf("Recvd: %d Acc L:%d R:%d\n",Message->seq,left,right);
                            buffer[Message->seq-left]=1;
```

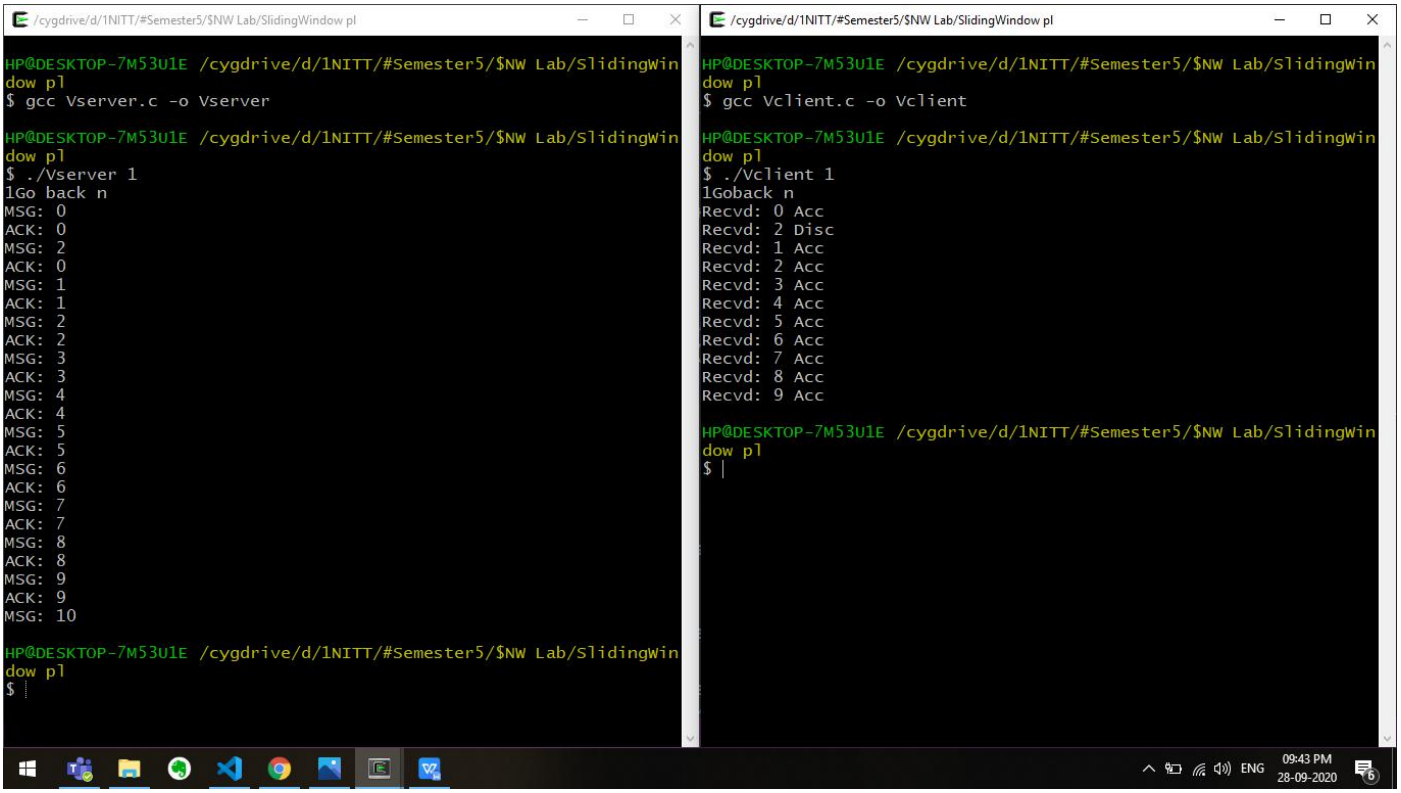
```
                        ind=Message->seq;
                        temp=ind;
                    }
                    else
                    {
                        printf("Recvd: %d Disc L:%d R:%d\n",Message->seq,left,right);
                        if(Message->seq<=left)
                        {
                            temp=ind;
                            ind=Message->seq;
                        }
                    }
                }
```

```
                struct MESSAGE* Acknowledge = (struct MESSAGE*) malloc(sizeof(struct M
ESSAGE));
                Acknowledge->type = ACK;
                Acknowledge->len = 0;
                Acknowledge->msg=-1;
                Acknowledge->seq=ind;
                    send(sock,(void*)Acknowledge, sizeof(struct MESSAGE), 0);

                if(ind!=temp)
                    ind=temp;

            int flag=1;
            for(int p=left;p<=right;p++)
            {
                if(buffer[p-left]==0)
                {
                    flag=-1;
                    //printf("%d",p);
                    break;
                }
            }
            if(flag==1)
            {
                left+=recv_win;
                right+=recv_win;
                memset(buffer,0,sizeof(buffer));
            }


        }
    }


    close(sock);
    return 0;
}
```

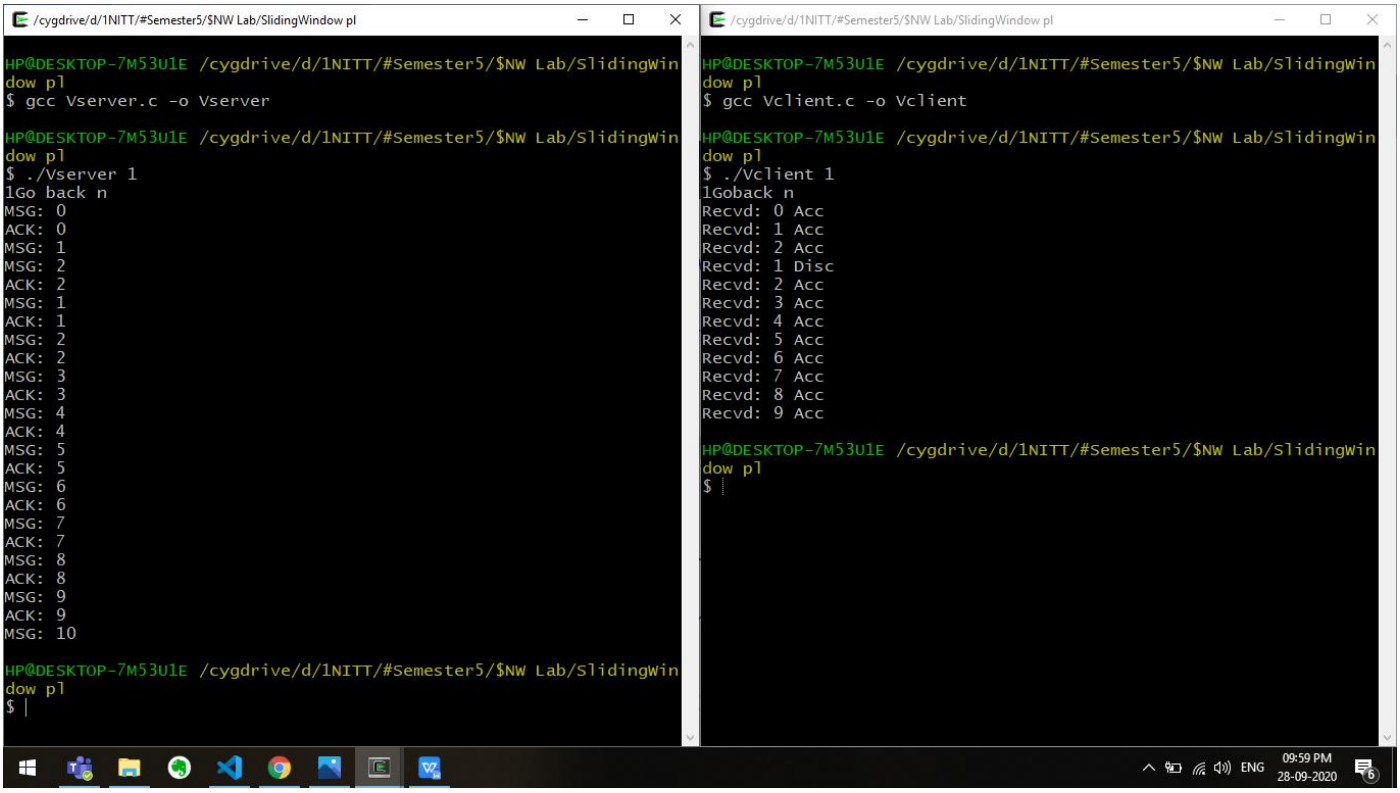
Go Back n Normal Transmission



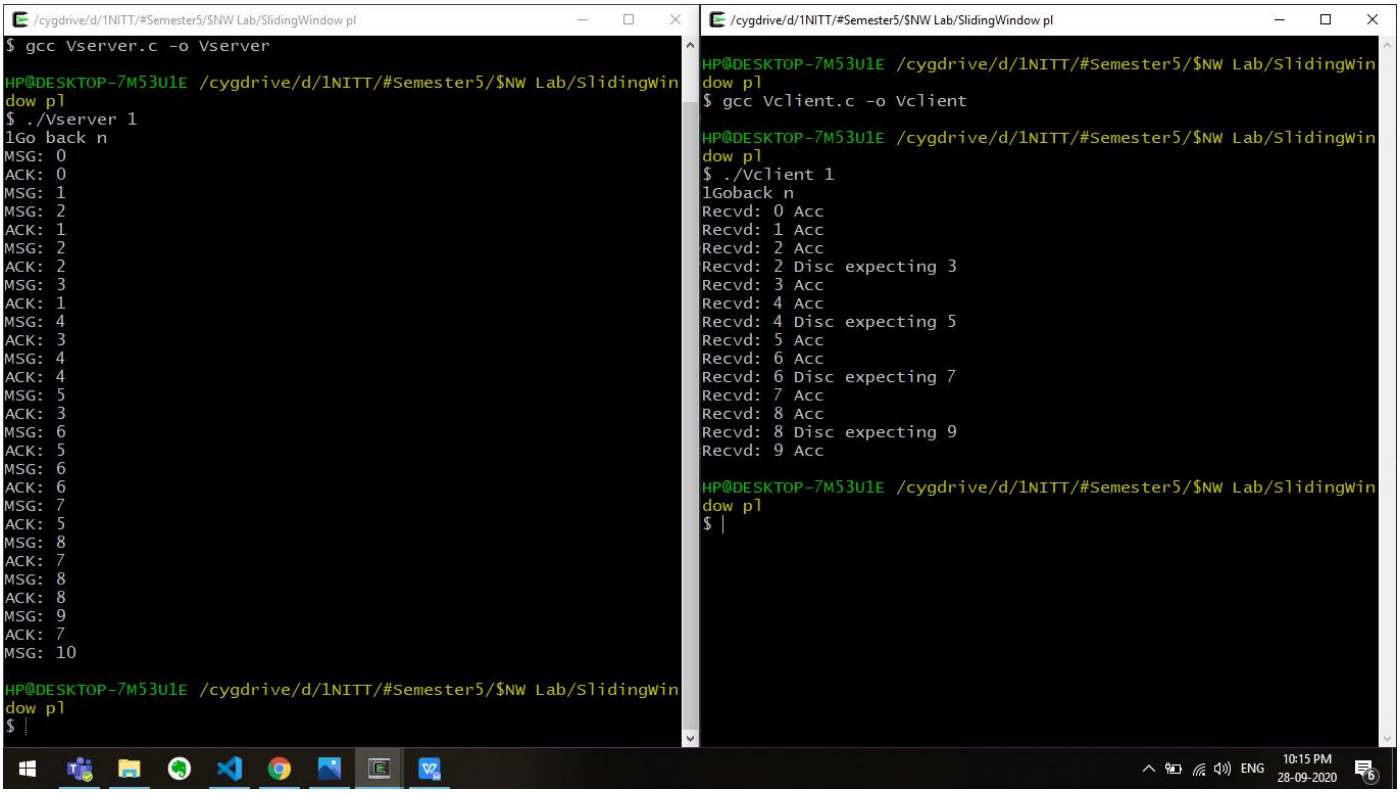
Go Back n Lost Packet(msg 1)



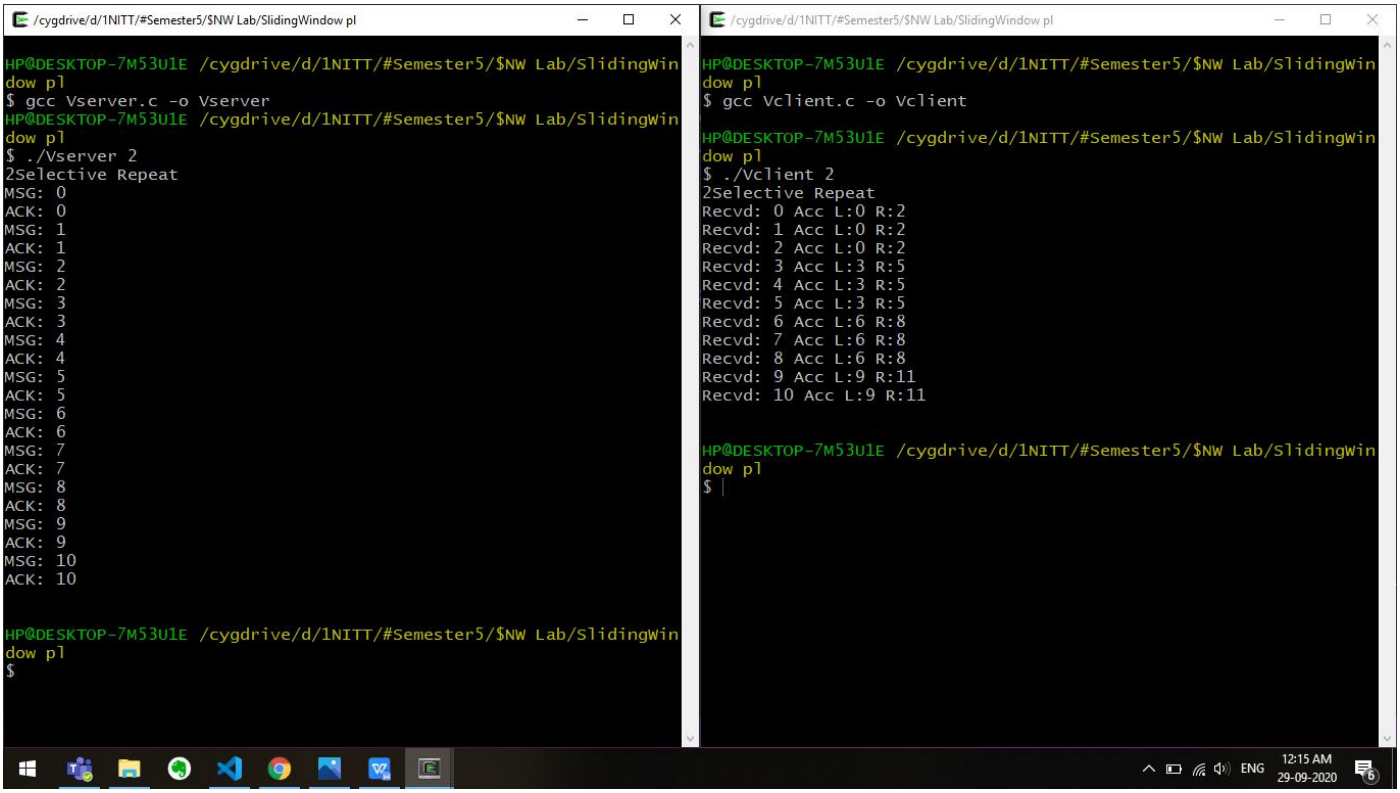
Go Back n Lost Acknowledgement(ack 1)



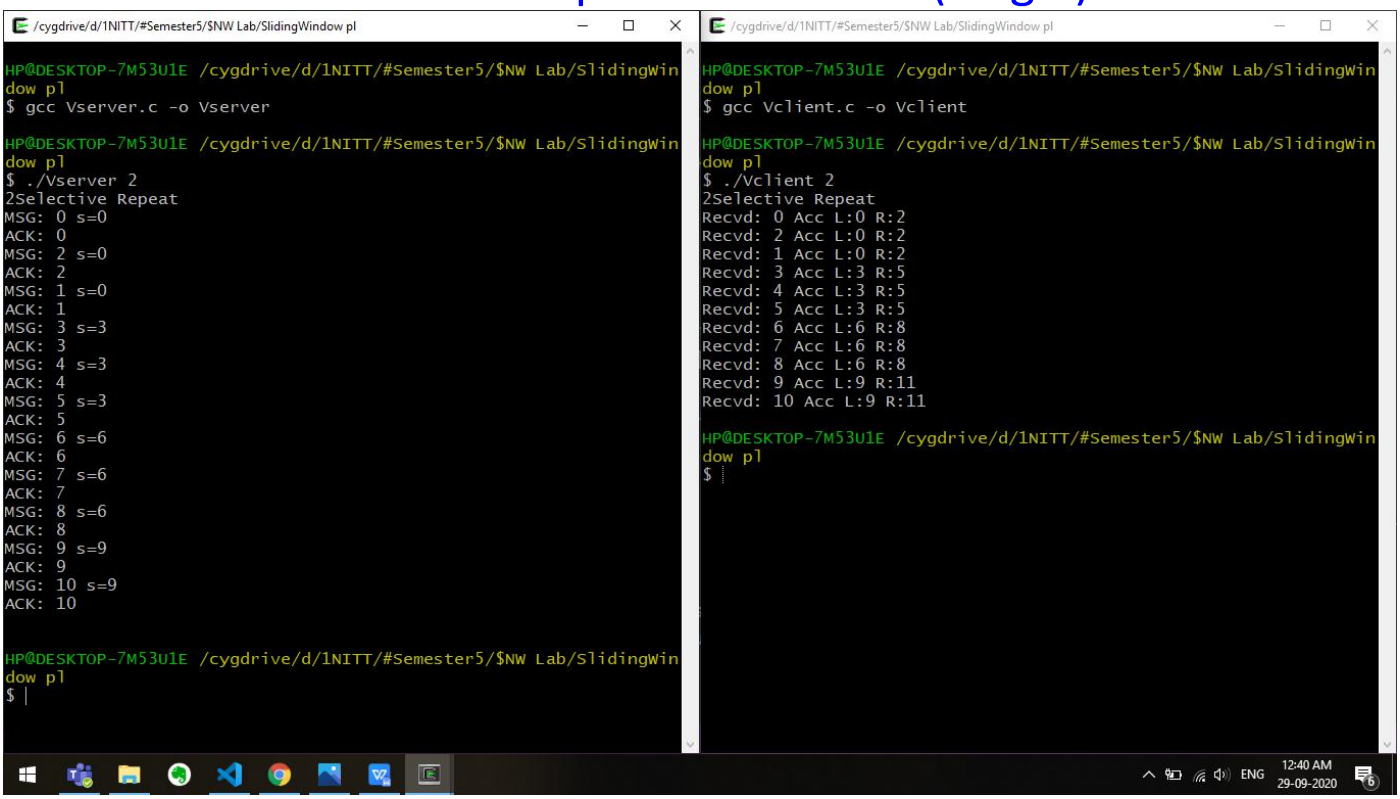
Go Back n Timeout (ack 1)



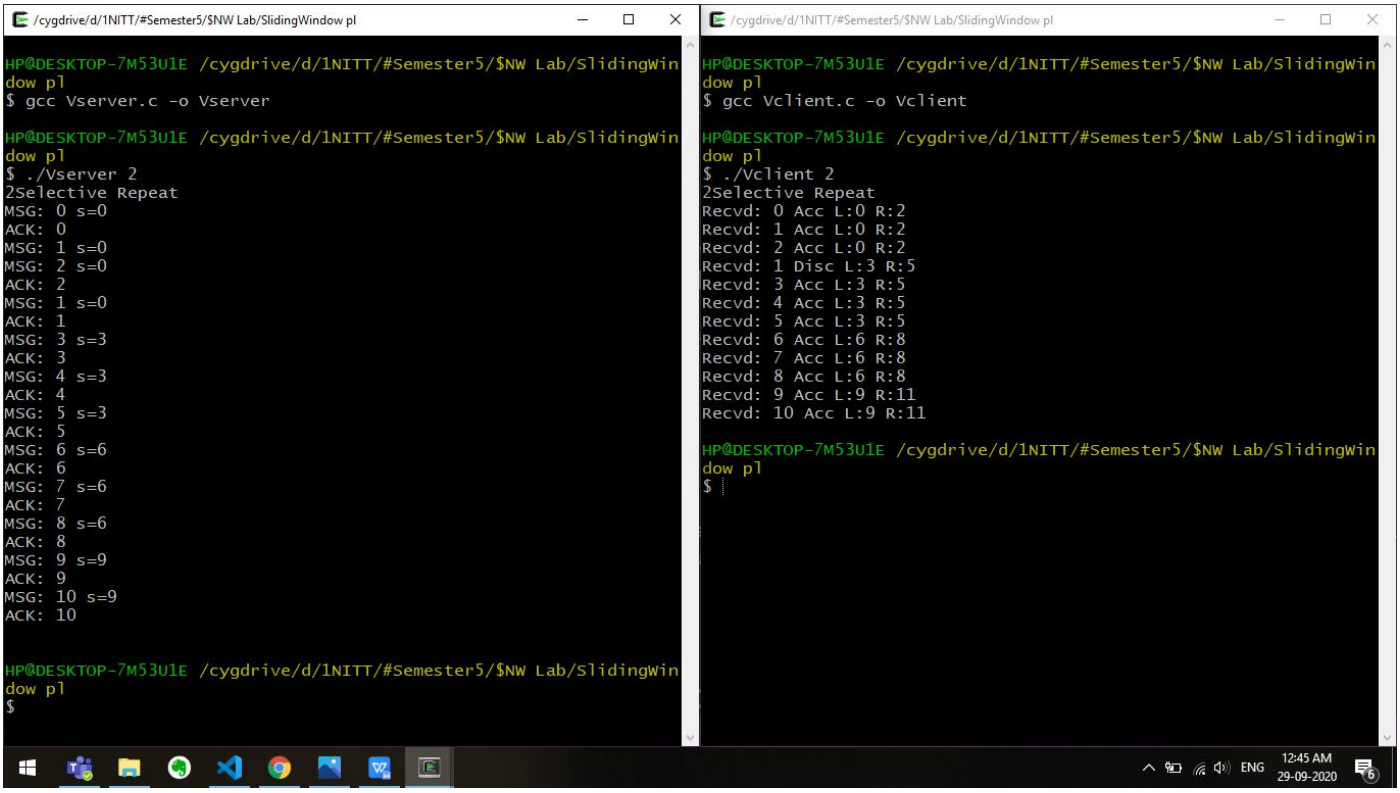
Selective Repeat Normal Transmission



Selective Repeat Lost Packet(msg 1)



Selective Repeat Lost Acknowledgement(ack 1)



Selective Repeat Timeout(ack 1)