COMPUTER NETWORKS (CS F303)

SECOND SEMESTER 2018-19

LAB-SHEET – 5

TOPIC: Socket Programming III

Learning Objectives:

a) To learn the concept of reliable protocol design and implementation

b) To learn Stop and Wait reliable protocol design and implementation using UDP sockets

---

Stop and Wait Protocol (rdt 2.5)

In Stop and Wait protocol sender sends a data packet and it waits till an acknowledgement (ACK) packet received from the receiver before it sends the next data packet. If ACK is not arrived at sender in a predefined time (Timeout) then sender retransmits that packet and starts timer again. Let us assume that a packet (Data packet and ACK packet) can be lost but it can't be corrupted for our current discussion. We call this modified version of rdt 3.0 protocol as rdt 2.5.

The behavior of rdt 2.5 sender and receiver is captured in the FSM shown in Fig. 1 (a) and Fig. 1 (b) respectively.

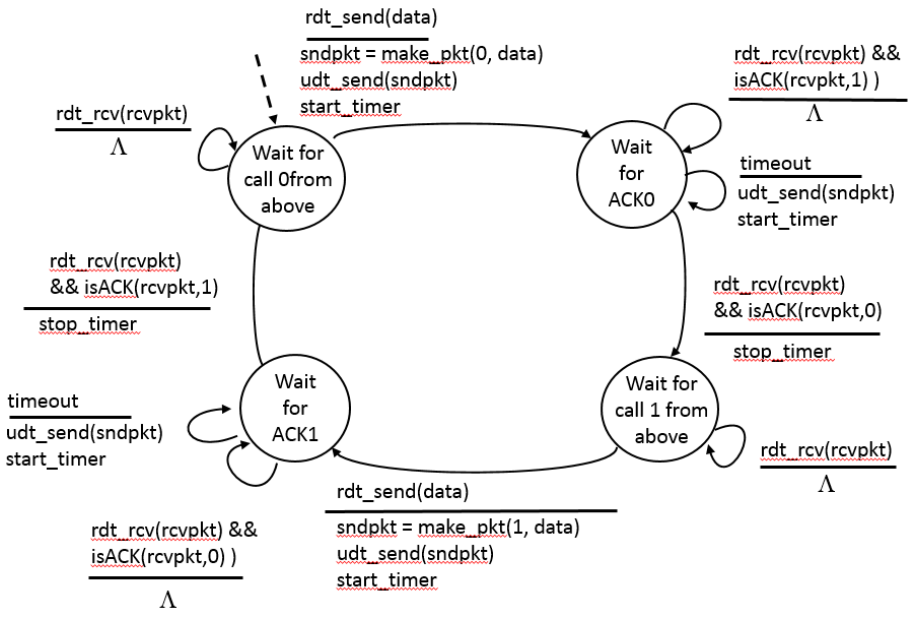

Fig. 1 (a) rdt 2.5 Sender FSM

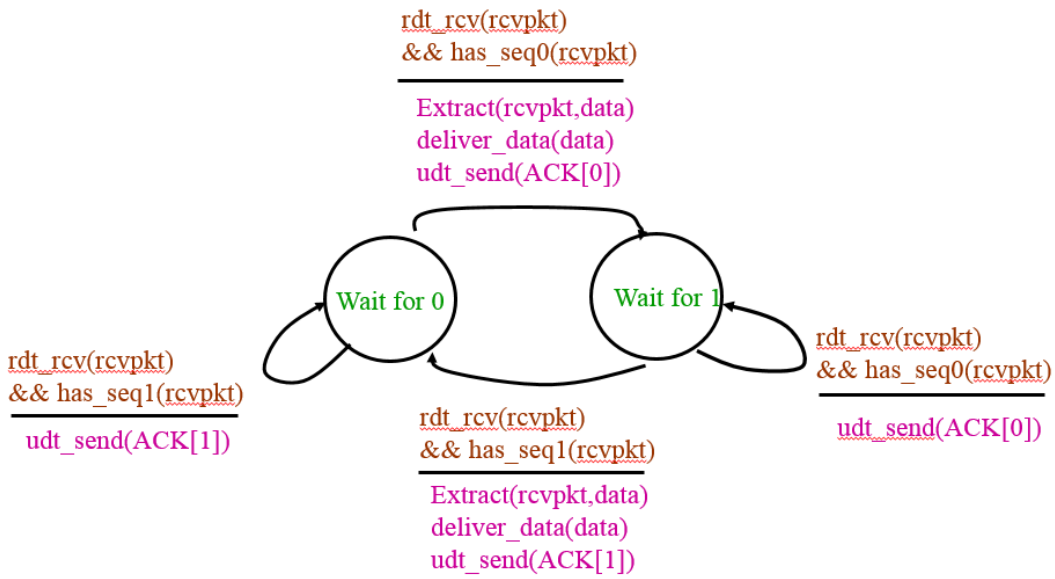

Fig. 1 (b) rdt 2.5 receiver FSM

rdt 2.5 Operation with no packet loss

We will implement rdt 2.5 using unreliable sockets (i.e., UDP). To make it simple, we will implement it in two steps. First we will implement it under no packet loss situation that means neither data packet nor ACK packet lost. The rdt 2.5 operation under no loss situation is demonstrated in the Fig. 2.

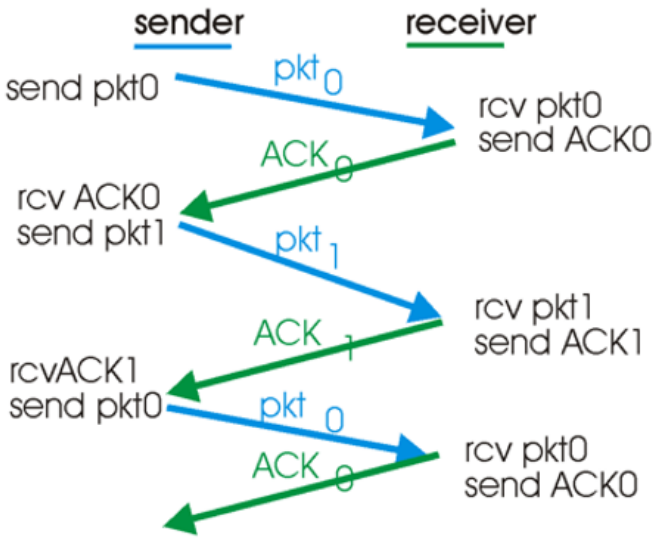

Fig. 2 rdt 2.5 without packet loss scenario

rdt 2.5 client (sender) implementation (udp_client.c)

Note: This is partial implementation of rdt 2.5 sender with no packet loss assumption. This code file is provided separately named as **udp_client.c**

```
/*
    Simple udp client with stop and wait functionality
*/
#include<stdio.h> //printf
#include<string.h> //memset
#include<stdlib.h> //exit(0);
#include<arpa/inet.h>
#include<sys/socket.h>

#define BUFLEN 512  //Max length of buffer
#define PORT 8882   //The port on which to send data

typedef struct packet1{
    int sq_no;
}ACK_PKT;

typedef struct packet2{
    int sq_no;
    char data[BUFLEN];
}DATA_PKT;

void die(char *s)
{
    perror(s);
    exit(1);
}

int main(void)
{
    struct sockaddr_in si_other;
    int s, i, slen=sizeof(si_other);
    char buf[BUFLEN];
    char message[BUFLEN];
    DATA_PKT send_pkt,rcv_ack;
    if ( (s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
    {
        die("socket");
    }

    memset((char *) &si_other, 0, sizeof(si_other));
    si_other.sin_family = AF_INET;
    si_other.sin_port = htons(PORT);
      si_other.sin_addr.s_addr = inet_addr("127.0.0.1");

 int state = 0;
    while(1)
    {

      switch(state)
```

```
      {   case 0: printf("Enter message 0: ");//wait for sending packet with
seq. no. 0
                  fgets(send_pkt.data,sizeof(send_pkt),stdin);
                  send_pkt.sq_no = 0;
                  if  (sendto(s,  &send_pkt,  sizeof(send_pkt),  0  ,  (struct
sockaddr *) &si_other, slen)==-1)
                  {
                    die("sendto()");
                  }
                  state = 1;
                  break;



         case 1:   //waiting for ACK 0
                  if  (recvfrom(s,  &rcv_ack,  sizeof(rcv_ack),  0,  (struct
sockaddr *) &si_other, &slen) == -1)
                  {
                          die("recvfrom()");
                  }
              if (rcv_ack.sq_no==0)
                     {  printf("Received ack seq. no. %d\n",rcv_ack.sq_no);
                             state = 2;
                       break;
                     }

          case 2:
                    printf("Enter message 1: ");
                   //wait for sending packet with seq. no. 1
                    fgets(send_pkt.data,sizeof(send_pkt),stdin);
                    send_pkt.sq_no = 1;
                   if  (sendto(s,  &send_pkt,  sizeof(send_pkt)  ,  0  ,  (struct
sockaddr *) &si_other, slen)==-1)
                   {
                     die("sendto()");
                   }
                   state = 3;
                   break;


         case 3:  //waiting for ACK 1
                   if(recvfrom(s, &rcv_ack, sizeof(rcv_ack), 0, (struct sockaddr
*) &si_other, &slen) == -1)
                    {
                      die("recvfrom()");
                    }
                   if (rcv_ack.sq_no==1)
                      { printf("Received ack seq. no. %d\n",rcv_ack.sq_no);
                        state = 0;
                        break;
                      }
      }

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

rdt 2.5 Receiver (Server) Implementation (udp_server.c)

Note: This is partial implementation of rdt 2.5 server with no packet loss assumption. This code file is provided separately named as **udp_server.c**

```
/* Simple udp server with stop and wait functionality */
#include<stdio.h> //printf
#include<string.h> //memset
#include<stdlib.h> //exit(0);
#include<arpa/inet.h>
#include<sys/socket.h>

#define BUFLEN 512  //Max length of buffer
#define PORT 8882   //The port on which to listen for incoming data

void die(char *s)
{
    perror(s);
    exit(1);
}

typedef struct packet1{
    int sq_no;
}ACK_PKT;

typedef struct packet2{
    int sq_no;
    char data[BUFLEN];
}DATA_PKT;

int main(void)
{
    struct sockaddr_in si_me, si_other;
    int s, i, slen = sizeof(si_other) , recv_len;
    //char buf[BUFLEN];
    DATA_PKT rcv_pkt;
    ACK_PKT  ack_pkt;
    //create a UDP socket
    if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
    {
        die("socket");
    }
    // zero out the structure
    memset((char *) &si_me, 0, sizeof(si_me));

    si_me.sin_family = AF_INET;
    si_me.sin_port = htons(PORT);
    si_me.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
    //bind socket to port
    if( bind(s , (struct sockaddr*)&si_me, sizeof(si_me) ) == -1)
    {
        die("bind");
    }
    int state =0;
    while(1)
    {
      switch(state)
        {  case 0:
            {   printf("Waiting for packet 0 from sender...\n");
                    fflush(stdout);

         //try to receive some data, this is a blocking call
         if ((recv_len = recvfrom(s, &rcv_pkt, BUFLEN, 0, (struct sockaddr *)
&si_other, &slen)) == -1)
                    {
                       die("recvfrom()");
                    }
                   if (rcv_pkt.sq_no==0)
                    {  printf("Packet received with seq. no. %d and Packet
                       content is = %s\n",rcv_pkt.sq_no, rcv_pkt.data);
                       ack_pkt.sq_no = 0;
        if (sendto(s, &ack_pkt, recv_len, 0, (struct sockaddr*) &si_other,
slen) == -1)
                {
                    die("sendto()");
                }
                      state = 1;
                      break;
                }
            }
         case 1:
                    {   printf("Waiting for packet 1 from sender...\n");
                        fflush(stdout);

                     //try to receive some data, this is a blocking call
         if ((recv_len = recvfrom(s, &rcv_pkt, BUFLEN, 0, (struct sockaddr *)
&si_other, &slen)) == -1)
                    {
                       die("recvfrom()");
                    }
              if (rcv_pkt.sq_no==1)
                 { printf("Packet received with seq. no.=1 %d and Packet content
                   is= %s\n",rcv_pkt.sq_no, rcv_pkt.data);
                   ack_pkt.sq_no = 1;
         if (sendto(s, &ack_pkt, recv_len, 0, (struct sockaddr*) &si_other,
slen) == -1)
                {    die("sendto()"); }
                     state = 0;
                     break;
```

```
            }

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

Exercise-1:

a) Compile and execute udp_client.c and udp_server.c programs separately and verify/understand the protocol behavior in no loss condition.

rdt 2.5 Operation with packet loss

Now we will add packet loss handling functionality in our phase-1 code to complete rdt 2.5 protocol implementation. The data packet loss operation and ACK packet loss operation is demonstrated in the Fig. 3(a) and 3(b) respectively.

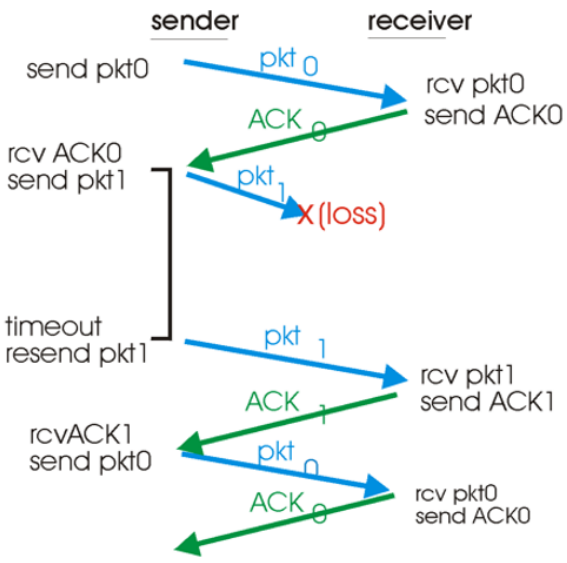

Fig. 3 (a) rdt 2.5 data packet loss scenario

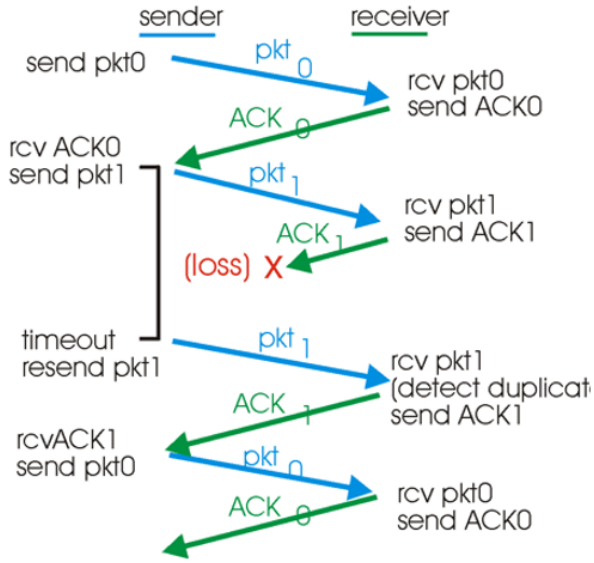

Fig. 3 (b) rdt 2.5 ACK packet loss scenario

Exercise-2:

a) Extend the given client (udp_client.c) and server (udp_server.c) programs to handle data packet and ACK packet loss.

Following description will be helpful to implement the desired functionality.

Packet loss Emulation:

Did you observe any packet loss when you run the given client server program? I hope your answer is NO. It is obvious, as your client and server both are running on same machine hence no packet loss happened. Even though, if you run this program on two different machines then also chances of packet loss is very less. But in real situation when client and server are distant apart then the packet loss probability increases.

Therefore to test the working of packet loss functionality for rdt 2.5 we need to introduce fake packet loss in our program. Let's think, how to do it….?????

Hint: Modify your receiver program such that when it receives a data packet through a recvfrom() system call it discards packet randomly (even it is with correct expected sequence number). In this manner some of the data packets are discarded. So for such packets receiver will not create and send ACK packet as per protocol semantics. This leads to timeout at sender.

Similarly, ACK packet loss can be fabricated at sender side.

You can use rand() function to randomly set a flag and based on that discard or receive a packet.

Timer Implementation:

Sender initiate a timer after sending a packet. If corresponding ACK packet is not received at sender (either data packet or ACK lost) before timer expires then it retransmits that packet and start times. Otherwise, it stops the timer and sends the packet with next sequence number.

The recvfrom() system call is a blocking call. So we have to make it non-blocking call. If recvfrom() call does not receive any data within a specified time (timeout value) then it should unblock.

Read select() system call to unblock recvfrom() call.

Alternative Approach

```
void handle_alarm( int sig ) {

{

        // specify action to be taken if timer expires

}
```

Timer setting using function **alarm(unsigned int seconds)** just after the sendto() system call is required.

****