BITS Pilani - Hyderabad Campus
CS F303 (Computer Networks)
Second Semester 2023-24, Lab Sheet 5
Stop and Wait Protocol

1. Overview

In this lab, we will implement Stop and Wait protocol. We will use the Socket programming to utilize sockets for the communication between application and transport layer. Additionally, we will also utilize the Mininet, to implement a virtualized environment for simulating hosts, switches, and routers. We will emulate network configurations such as delay and packet losses to study their impact on the Stop and Wait protocol.

2. Stop and Wait Protocol

A basic stop and wait protocol is given by finite state machines (FSMs) in Figure 1 (Sender) and Figure 2 (Receiver). This basic stop and wait works well only in error-free and loss-free channel.

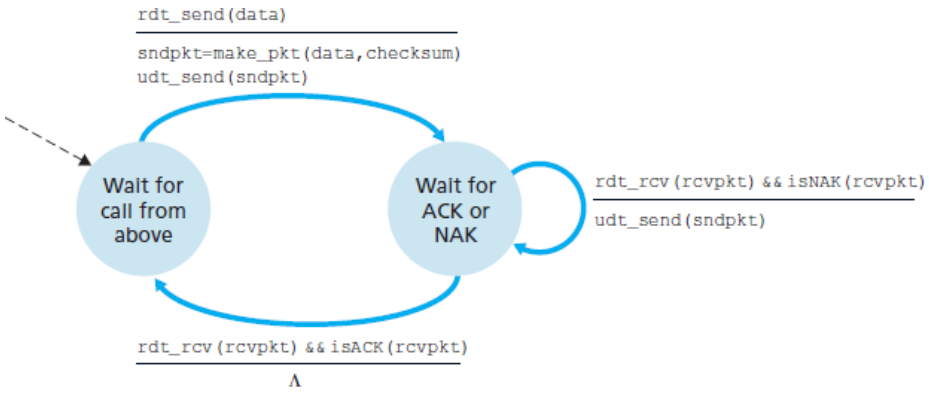

Figure 1 Stop and Wait (Sender Side)

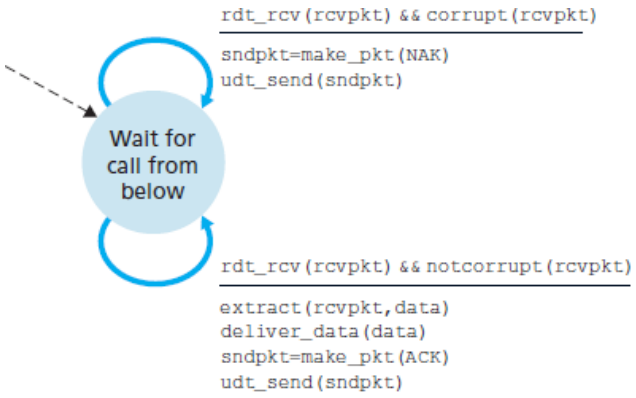

Figure 2 Stop and Wait (Receiver Side)

3. Deployment of Reliable Stop and Wait protocol over Mininet

In this lab, we will be using Mininet to develop a network topology (one sender and one receiver) and implement stop and wait protocol. Please follow these steps:

- To setup and use Mininet, follow the previous labsheets.
- Create a standard topology as shown in Figure 3.

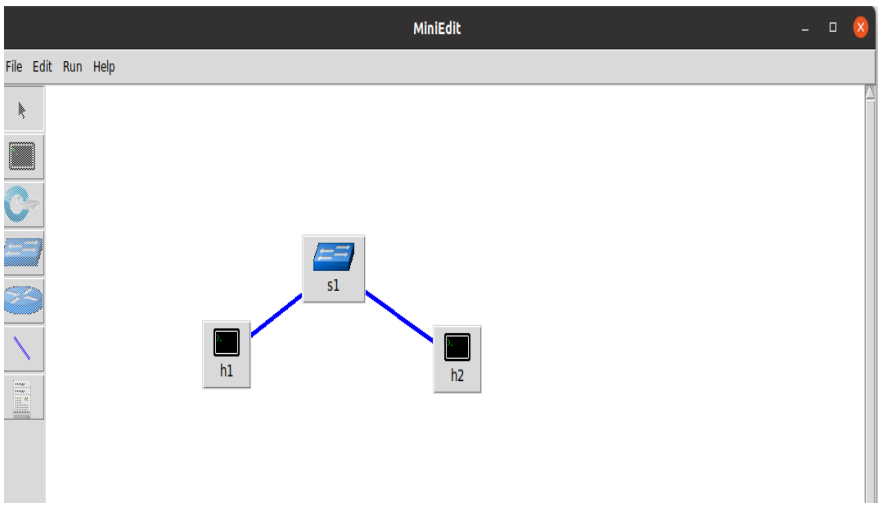

Figure 3 Topology (One sender and one receiver)

2.1)

- Configure IP addresses of h1 and h2 nodes, and start running mininet.
- Develop C programs for both the sender and receiver functionalities. Place the **sender.c file** on host 1 (h1) and the **receiver.c** file on host 2 (h2). These codes are given in the next section.
- Execute the program without adding delay by running receiver.c first and then sender.c.
- Add a delay on both the link between h1-s1 and s1-h2. Execute the programs and observe the impact of the delay introduced.

Note: Prior to compilation, ensure to input the IP address of the receiver into the sender program. Then, compile both the sender and receiver programs on mininet terminals of both h1 and h2.

4. C programs for implementation of basic stop and wait

Code (sender.c)

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#define PORT 8080
#define MAX_FRAME_SIZE 100
void error(const char *msg) {
   perror(msg);
   exit(1);
}
int main() {
   int sockfd;
   struct sockaddr_in serv_addr;
   char buffer[MAX_FRAME_SIZE];
   // Create socket
   sockfd = socket(AF_INET, SOCK_STREAM, 0);
   if (sockfd < 0)
      error("ERROR opening socket");
   // Server address configuration
   bzero((char *) &serv_addr, sizeof(serv_addr));
   serv_addr.sin_family = AF_INET;
   serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
   serv_addr.sin_port = htons(PORT);
   // Connect to the server
   if (connect(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
      error("ERROR connecting");
   // Number of frames to be sent
   int n;
   printf("Enter the number of frames to be sent: ");
   scanf("%d", &n);
   // Send frames
   for (int i = 0; i < n; i++) {
      printf("Sending frame %d\n", i);
      sprintf(buffer, "Frame %d", i);
      write(sockfd, buffer, strlen(buffer));
      // Wait for acknowledgment
      bzero(buffer, MAX_FRAME_SIZE);
      read(sockfd, buffer, MAX_FRAME_SIZE);
```

```
        printf("Received acknowledgment: %s\n", buffer);
    }
    // Send exit signal
    strcpy(buffer, "exit");
    write(sockfd, buffer, strlen(buffer));
    close(sockfd);
    return 0;
}
```

Code (receiver.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#define PORT 8080
#define MAX_FRAME_SIZE 100
void error(const char *msg) {
    perror(msg);
    exit(1);
}
int main() {
    int sockfd, newsockfd, clilen;
    int flag=1;
    struct sockaddr_in serv_addr, cli_addr;
    char buffer[MAX_FRAME_SIZE];
    // Create socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
    // Server address configuration
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(PORT);
    // Bind socket
    if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
        error("ERROR on binding");
```

```
    // Listen for connections
    listen(sockfd, 5);
    clilen = sizeof(cli_addr);
    // Accept incoming connection
    newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
    if (newsockfd < 0)
        error("ERROR on accept");
    // Receive frames
    while (1) {
        bzero(buffer, MAX_FRAME_SIZE);
        read(newsockfd, buffer, MAX_FRAME_SIZE);
        if (strcmp(buffer, "exit") == 0)
            break;

        printf("Received frame: %s\n", buffer);
        // Send acknowledgment
        strcpy(buffer, "ACK");
       // if(flag){
        write(newsockfd, buffer, strlen(buffer));
        //flag=0;}
    }
    close(newsockfd);
    close(sockfd);
    return 0;
}
```

5. Lab Exercises

Modify a program (at sender side) such that you virtually drop some frames. Observe the behavior and identify the possible issues. Now, modify the code such that the issues are resolved and your program also works well over the lossy channel.