Department of Computer Science and Engineering

S.G.Shivanirudh , 185001146, Semester V

16 September 2020

UCS1511 - Networks Laboratory

Exercise 5: Address Resolution Protcol

Objective:

Simulate ARP using socket programming.

Code:

Packet Structure:

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
```

```c
#include<unistd.h>
#include<sys/time.h>

struct Packet{
    char *sip;            //Source IP address
    char *smac;           //Source MAC address
    char *dip;            //Destination IP address
    char *dmac;           //Destination MAC address
    char *arp_packet;     //ARP packet
    char *data;           //Data
};

typedef struct Packet ARP;

void init(ARP* packet){
    packet->sip = (char*)calloc(100,sizeof(char));
    packet->smac = (char*)calloc(100, sizeof(char));
    packet->dip = (char*)calloc(100,sizeof(char));
    packet->dmac = (char*)calloc(100, sizeof(char));
    packet->arp_packet = (char*)calloc(100, sizeof(char));
    packet->data = (char*)calloc(100, sizeof(char));
}

void acceptPacket(ARP *packet){
    printf("\nEnter the details of packet received. \n");
    printf("\nSource IP address: ");scanf(" %s", packet->sip)
   ;
    printf("\nSource MAC address: ");scanf(" %s", packet->
   smac);
    printf("\nDestination IP address: ");scanf(" %s", packet
   ->dip);
    printf("\n16 Bit data: ");scanf(" %s", packet->data);
}

void developPacket(ARP *packet){
    printf("\nDeveloping ARP packet details.\n");
    strcpy(packet->arp_packet, packet->sip);strcat(packet->
   arp_packet, "|");
    strcat(packet->arp_packet, packet->smac);strcat(packet->
   arp_packet, "|");
    strcat(packet->arp_packet, packet->dip);
}

void get_destmac(ARP *packet, char* buffer){
    int count = 0, k = 0;
    for(int i =0; buffer[i];i++){
        if(count == 3)
            packet->dmac[k++] = buffer[i];
        if(buffer[i] == '|')
            count++;
    }
    packet->dmac[k] = '\0';
}

void develop_msg(ARP *packet, char* buffer){
    strcpy(buffer, packet->arp_packet);strcat(buffer, "|");
    strcat(buffer, packet->dmac); strcat(buffer, "|");
    strcat(buffer, packet->data);
}
```

Server:

```c
#include "Packet.h"

int main(int argc,char **argv){
    //Server and Client addresses
    struct sockaddr_in server_address, client_address;
    //Buffer to handle messages
    char buffer[1024];
    //Storing sockets for client
    int client_sockets[30];
    //Set of file descriptors
    fd_set clientfds;
    //Socket file descriptor for accepting connections
    int newfd;

    if(argc > 1){
        perror("Error: No arguments needed to run server. \n"
    );
    }

    //ARP Packet structure
    ARP packet;

    //Initialising ARP packet
    init(&packet);

    //Accepting packet details
    acceptPacket(&packet);

    //Developing ARP request packet
    developPacket(&packet);
    printf("%s\n", packet.arp_packet);

    for(int i = 0; i < 30; i++)
        client_sockets[i] = 0;

    int sockfd = socket(AF_INET, SOCK_STREAM, 0);//domain =
   IPv4, type = TCP, protocol = IP
    if(sockfd < 0)
        perror("Error: Unable to create socket");

    //Filling server_address with null bytes
    bzero(&server_address, sizeof(server_address));

    server_address.sin_family = AF_INET;// Uses the Internet
   address family
    server_address.sin_addr.s_addr = INADDR_ANY;// Use any of
    the available addresses
    server_address.sin_port = htons(4500);// Connect to
   specified port 4500

    //Bind socket to the specified port
    if(bind(sockfd, (struct sockaddr*)&server_address, sizeof
   (server_address))<0)
        perror("Bind error");

    //Look for clients to serve, with a maximum limit of 5.
    listen(sockfd, 5);

    //New socket file descriptor to handle connections.
    int len = sizeof(client_address);
    while(1){
        //Clears socket set
        FD_ZERO(&clientfds);

        //Add main socket to the set
        FD_SET(sockfd, &clientfds);
        int max_sd = sockfd;

        //Adding valid secondary sockets to the set
        for(int i = 0;i < 30;i++){
            int sd = client_sockets[i];
            //Checking validity
            if(sd > 0)
                FD_SET(sd, &clientfds);

            //Store highest valued file descriptor
            if(sd > max_sd)
                max_sd = sd;
        }

        //Wait indefinitely for action on one of the sockets
        int action = select(max_sd + 1, &clientfds, NULL,
    NULL, NULL);
        if(action<0){
            perror("\nSelect error!\n");
        }

        //A change in main socket descriptor value implies
    that it is an incoming connection request
        if(FD_ISSET(sockfd, &clientfds)){
            newfd = accept(sockfd, (struct sockaddr*)&
    client_address, &len);
            if(newfd < 0)
                perror("\nUnable to accept new connection.\n"
    );

            //Send ARP Request
            strcpy(buffer, packet.arp_packet);
            write(newfd, buffer, sizeof(buffer));
            //Add new client socket to list of sockets
            for(int i =0;i<30;i++){
                if(client_sockets[i] == 0){
                    client_sockets[i] = newfd;
                    break;
                }
            }
        }
        //Broadcasting on an established connection
        for(int i = 0;i<30;i++){
            int sd = client_sockets[i];
            bzero(buffer, sizeof(buffer));
            //Check for change in descriptors
            if(FD_ISSET(sd, &clientfds)){
                read(sd, buffer, sizeof(buffer));

```

```c
                //Check ARP response
                if(buffer[0]){
                    printf("\nARP Reply received: %s\n", buffer);
                    get_destmac(&packet, buffer);

                    printf("\nSending packet to %s\n", packet.dmac);
                    bzero(buffer, sizeof(buffer));

                    //Create data message
                    develop_msg(&packet, buffer);

                    //Write message in buffer
                    write(newfd, buffer, sizeof(buffer));
                    printf("\nPacket Sent: %s\n", buffer);
                }

            }
        }

    }

    return 0;
}
```

Client:

```c
#include "Packet.h"

int main(int argc,char** argv){
    //Server and client addresses
    struct sockaddr_in server_address, client_address;
    //Buffer to handle messages
    char buffer[1024];

    //Check arguments
    if(argc<2){
        perror("\nError: IP address to be passed in command line.\n");
    }

    //ARP Packet structure
    ARP packet;

    //Initialising ARP packet
    init(&packet);

    //Accepting addresses
    printf("\nEnter IP address: ");scanf(" %s", packet.sip);
    printf("\nEnter MAC address: ");scanf(" %s", packet.smac);

    //Server socket file descriptor
    int sockfd = socket(AF_INET, SOCK_STREAM, 0);//(domain = Ipv4, type = TCP, protocol = 0
    if(sockfd < 0)
        perror("Error: Unable to create socket");
    //Filling server address with null bytes
    bzero(&server_address, sizeof(server_address));

    server_address.sin_family = AF_INET;//Use the Internet address family
    server_address.sin_addr.s_addr = inet_addr(argv[1]);//Use ip address passed as command line argument
    server_address.sin_port = htons(4500);//Connect socket to port 4500

    //Attempt to connect client to socket on specified port
    connect(sockfd, (struct sockaddr*)&server_address, sizeof(server_address));

    int len = sizeof(client_address);

    bzero(buffer, sizeof(buffer));
    read(sockfd, buffer, sizeof(buffer));
    printf("\nARP Request received: %s\n", buffer);

    int count = 0, k = 0;
    for(int i =0; buffer[i];i++){
        if(count == 2)
            packet.dip[k++] = buffer[i];
        if(buffer[i] == '|')
            count++;
    }
    packet.dip[k] = '\0';

    //Check ARP request packet
    if(strcmp(packet.dip, packet.sip) == 0){
        printf("\nIP address matches.\n");
        //Write message in buffer
        strcat(buffer, "|");strcat(buffer, packet.smac);
        write(sockfd, buffer, sizeof(buffer));
        printf("\nARP reply sent: %s\n", buffer);

        bzero(buffer, sizeof(buffer));
        read(sockfd, buffer, sizeof(buffer));
        printf("\nPacket received: %s\n", buffer);
    }
    else{
        printf("\nIP address does not match.\n");
    }

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

Output:

Server:

```
Enter the details of packet received.

Source IP address: 123.128.34.56

Source MAC address: AF-45-E5-00-97-12

Destination IP address: 155.157.65.128

16 Bit data: 1011110000101010

Developing ARP packet details.
123.128.34.56|AF-45-E5-00-97-12|155.157.65.128

ARP Reply received: 123.128.34.56|AF-45-E5
    -00-97-12|155.157.65.128|45-D4-62-21-1A-B2

Sending packet to 45-D4-62-21-1A-B2
```

```

Packet Sent: 123.128.34.56|AF-45-E5
    -00-97-12|155.157.65.128|45-D4-62-21-1A-B2
    |1011110000101010
```

Client 1:

```
Enter IP address: 165.43.158.158

Enter MAC address: 09-DF-90-26-6C-09

ARP Request received: 123.128.34.56|AF-45-E5
    -00-97-12|155.157.65.128

IP address does not match.
```

Client 2:

```
Enter IP address: 155.157.65.128

Enter MAC address: 45-D4-62-21-1A-B2

ARP Request received: 123.128.34.56|AF-45-E5
    -00-97-12|155.157.65.128

IP address matches.

ARP reply sent: 123.128.34.56|AF-45-E5
    -00-97-12|155.157.65.128|45-D4-62-21-1A-B2

Packet received: 123.128.34.56|AF-45-E5
    -00-97-12|155.157.65.128|45-D4-62-21-1A-B2
    |1011110000101010
```

Client 3:

```
Enter IP address: 15.143.158.18

Enter MAC address: 19-0F-01-63-C7-D4

ARP Request received: 123.128.34.56|AF-45-E5
    -00-97-12|155.157.65.128

IP address does not match.
```