***Assignment 4:***

***Application development using UDP Socket***

***PRESENTED BY 🡪 2021CSB071 PRAYAS MAZUMDER***

Objective :

This assignment aims to find the Round Trip Time (RTT) in a Client to Server communication using a datagram (UDP) socket. The Server and Client should situate on two different physical computers in Ethernet LAN. The aim is also to understand how the Round Trip Time changes in various loading conditions in the network.

Technique:

The process starts with the Client generating and sending a packet to the Server. The packet

(datagram) format to be used is as follows.

|----------|----------|----------|----------|

| Sequence | TTL | Payload | Payload |

| Number | | Length | bytes |

| (**4 bytes)| (1 byte) | (2 bytes)| (P bytes)**|

|----------|- --------|----------|----------|

 ***Sequence Number***:Uniquely identifies individual packet.

 ***Time-to-live field (TTL):*** A non-negative even integer with an initial value of T.

 ***Payload Length:*** The length of the payload bytes P.

 ***Payload Bytes***: Arbitrary payload bytes of size P bytes.

When the Server receives a datagram from the Client, it checks the Payload Length field and compares the number of received bytes to determine the sanity of the packet. The packet is dropped at the Server if it is found to be inconsistent and responds with an error message as “MALFORMED PACKET” in the payload (keeping SN and TTL unchanged). Otherwise it decrements the TTL value by one in the datagram and sends the same datagram (with the new TTL) back to the Client. Note that the Server should minimize the processing delay by not incorporating any printing or queuing of the received packet.

On reception of the packets at the Client back from the Server, calculate the Round Trip Time (RTT) delay for each packet and calculate the average RTT delay across the packets. In case the packet is reported as “MALFORMED PACKET”, it prints in the console.

**Development and Running Environment:**

Write two socket programs in ClientPacketGen.c/cpp and ServerPacketFwd.c/cpp,

which communicate using Datagram Sockets (UDP).

The server application should be executed using the following command:

**$>./serverPacketfwd <ServerPort>**

The client application should be executed using the following command:

**$>./clientPacketGen <ServerIP> <ServerPort> <P> <TTL> <NumPackets>**

Here NumPackets could be anything between 1 to 50, and for each packet, a different

sequence number should be generated starting from 0. The P value should be within 100 to

1000 bytes, and T between 2 and 20 (and must be even).

TCP VS UDP :

Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are both part of the internet protocol suite. They are foundational pillars of the internet that enable different types of data transmission.

**Here are some differences between TCP and UDP:**

* Connection: TCP is a connection-based protocol, while UDP is connectionless.
* Speed: UDP is faster than TCP because it doesn't require additional responses from the receiver.
* Data integrity: TCP only transmits complete sets of data packets, while UDP transmits whatever it can, even if some packets are lost along the way.
* Reliability: TCP is more reliable, while UDP prioritizes speed and efficiency.
* Retransmission: TCP is the only protocol that can retransmit lost data packets.
* Bandwidth and resources: TCP/IP consumes more bandwidth and resources than UDP.

With UDP, the programmer manages the transmission to network directly, and has to take care of lost and out-of order packets as well as flow control and fragmenting data to packets that can be transmitted over the network connection.

***UDP SERVER CODE:***

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h> // for perror() , exit() etc ..

#include <string.h> // FOR STRING RELATED FUNCTIONS ..

#include <unistd.h>

#include <arpa/inet.h> // for network related functions like inet\_pton etc …

#define MAX\_PAYLOAD\_SIZE 1000

// as told in the qustion as the max size of a payload

*// Structure to represent the packet format*

*struct* Packet

{

*uint32\_t* sequence\_number; *// Sequence number (4 bytes)*

*uint8\_t* TTL; *// Time-to-live (1 byte)*

*uint16\_t* payload\_length; *// Payload length (2 bytes)*

*char* payload[MAX\_PAYLOAD\_SIZE]; *// Payload data (variable length)*

};

// uint32\_t guruntees us 32 bits of length , same with uint8\_t , uint16\_t …

Our main function :

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

{

    if (argc != 2) *// command line inputs*

    {

        printf("Usage: %s <ServerPort>\n", argv[0]);

        exit(1);

    }

*// server port   numeber conversion*

*// atoi 🡪 ASCII to Integer*

*int* server\_port = atoi(argv[1]);

*// Create socket*

*/ // server\_socket 🡪 file descriptor*

*int* server\_socket = socket(AF\_INET, SOCK\_DGRAM, 0); *// DGRAM for udp ,AF\_INET 🡪 for Ipv4 address*

    if (server\_socket < 0)

    {

        perror("Socket creation failed");

        exit(1);

    }

*// error checking comments*

*// printf("ok till now 1 \n");*

*// Server address of type ‘sockaddr\_in’*

*struct* sockaddr\_in server\_address, client\_address;

// initialisinng server\_address as all 0’s so that it does not contain any garbage value if not specified value anywehere in the code

    memset(&server\_address, 0, sizeof(server\_address));

// server address is of IPv4 type

    server\_address.sin\_family = AF\_INET;

// htons 🡪 host byte order to network byte order(short,16bit)

// ensures that network byte is in proper order

// NETWORK BYTE IS generally in BIG-ENDIAN …

    server\_address.sin\_port = htons(server\_port);

// now we will set the IP address of the server\_address structure

// using INADDR\_ANY 🡪 server will listen to any IP available

// htonl 🡪 host to network byte long (32 bit)

    server\_address.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    printf("Server is listening on port %d\n", server\_port);

*// Bind socket to address*

*// int bind(int socket, const struct sockaddr \*address,  
       socklen\_t address\_len); 🡪 structure of bind function*

*// bind() function returns 0 upon succcesful completion , -1 on error*

*int* bind\_val = bind(server\_socket, (*struct* sockaddr \*)&server\_address, sizeof(server\_address));

*// if (bind(server\_socket, (struct sockaddr \*)&server\_address, sizeof(server\_address)) < 0)*

    if (bind\_val < 0)

    {   perror("Binding failed");

        exit(1);

    }

*// error checking comments*

*// printf("ok till now 2");*

*// Loop to receive and forward packets*

*struct* Packet packet;

*socklen\_t* client\_address\_len = sizeof(client\_address);

    while (1)

    {

*// Receive packet from client*

*// ssize\_t recvfrom(int sockfd, void \*buf, size\_t len, int flags,*

*struct sockaddr \*src\_addr, socklen\_t \*addrlen);*

*ssize\_t* bytes\_received = recvfrom(server\_socket, &packet, sizeof(packet), 0, (*struct* sockaddr \*)&client\_address, &client\_address\_len);

        if (bytes\_received < 0)

        {

            perror("Error receiving packet");

            continue; *// Continue to next iteration of the loop*

        }

*// Check payload length*

*uint16\_t* payload\_length = ntohs(packet.payload\_length);

// this converts recceived network byte order to host order for 16bits

        if (payload\_length > MAX\_PAYLOAD\_SIZE)

        {

            printf("MALFORMED PACKET: Payload length too large\n");

            continue; *// Continue to next iteration of the loop*

        }

*// Decrement TTL*

*/\** *Otherwise it decrements the TTL value by one in the datagram and sends the same datagram (with the new TTL) back to the Client.\*/*

        if (packet.TTL > 0)

        {

            packet.TTL--;

        }

*// Forward same packet back to client*

*// ssize\_t sendto(int sockfd, const void \*buf, size\_t len, int flags,const struct sockaddr \*dest\_addr, socklen\_t addrlen);*

        sendto(server\_socket, &packet, bytes\_received, 0, (*struct* sockaddr \*)&client\_address, sizeof(client\_address));

    }

    close(server\_socket);

    return 0;

}

***UDP CLIENT CODE :***

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/time.h>

#define MAX\_PAYLOAD\_SIZE 1000

*// Structure to represent the packet format*

*struct* Packet

{

*uint32\_t* sequence\_number; *// Sequence number (4 bytes)*

*uint8\_t* TTL; *// Time-to-live (1 byte)*

*uint16\_t* payload\_length; *// Payload length (2 bytes)*

*char* payload[MAX\_PAYLOAD\_SIZE]; *// Payload data (variable length)*

};

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

{

    if (argc != 6)

    {

        printf("Usage: <ServerIP> <ServerPort> <P> <TTL> <NumPackets>\n");

        exit(1);

    }

*// Parse command-line arguments*

*char* \*server\_ip\_str = argv[1];

*int* server\_port = atoi(argv[2]);

*int* payload\_size = atoi(argv[3]);

*int* ttl = atoi(argv[4]);

*int* num\_packets = atoi(argv[5]);

*// Validate inputs*

    if (payload\_size < 100 || payload\_size > 1000 || ttl < 2 || ttl > 20 || ttl % 2 != 0 || num\_packets < 1 || num\_packets > 50)

    {

        printf("Invalid input parameters.\n");

        exit(1);

    }

*// Create socket*

*int* client\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);

    if (client\_socket < 0)

    {

        perror("Socket creation failed");

        exit(1);

    }

*// Server address*

*struct* sockaddr\_in server\_address;

    memset(&server\_address, 0, sizeof(server\_address));

    server\_address.sin\_family = AF\_INET;

    server\_address.sin\_port = htons(server\_port);

*// Convert server IP address from presentation to network format*

*struct* in\_addr server\_addr;

    if(inet\_pton(AF\_INET, server\_ip\_str, &server\_addr)<= 0)

    {

        perror("Invalid server IP address");

        exit(1);

    }

    server\_address.sin\_addr = server\_addr;

*struct* timeval start\_time, end\_time;

*double* total\_rtt = 0.0; *// Total round trip time*

*// initialised to zero initially …*

    for (*int* i = 0; i < num\_packets; i++)

    {

*// Create packet*

*struct* Packet packet;

        packet.sequence\_number = i; *// Sequence number*

        packet.TTL = ttl; *// TTL*

        packet.payload\_length = htons(payload\_size); *// Payload length*

*// Fill payload with random data (for demonstration purposes)*

        for (*int* j = 0; j < payload\_size; j++)

        {

            packet.payload[j] = 'A' + (rand() % 26); *// Random uppercase letter*

        }

*// Print payload before sending*

        printf("Sending payload data which is : %s\n\n", packet.payload);

*// Record start time*

*clock\_t* start\_time = clock();

*// Send packet to server*

        sendto(client\_socket, &packet, sizeof(packet), 0, (*struct* sockaddr \*)&server\_address, sizeof(server\_address));

*// Receive packet back from server*

        recvfrom(client\_socket, &packet, sizeof(packet), 0, NULL, NULL);

*// Print payload before sending*

        printf("Received payload data which is : %s\n\n", packet.payload);

// note : sent and receivedd data will be same since we are forwarding the same datagram

*// Record end time*

*clock\_t* end\_time = clock();

*// Calculate RTT for this packet*

*double* rtt = ((*double*)(end\_time - start\_time) / CLOCKS\_PER\_SEC) \* 1000.0;

// for milliseconds we have multiplied by 1000

        total\_rtt += rtt;

        printf("RTT for packet %d: %.2f ms\n\n", i, rtt);

        sleep(1); *// Add delay of 1sec between packets*

*// just to realise if things are working properly*

    }

*// Calculate and print average RTT across all packets*

    printf("Average RTT: %.2f ms\n\n", total\_rtt / num\_packets);

    close(client\_socket);

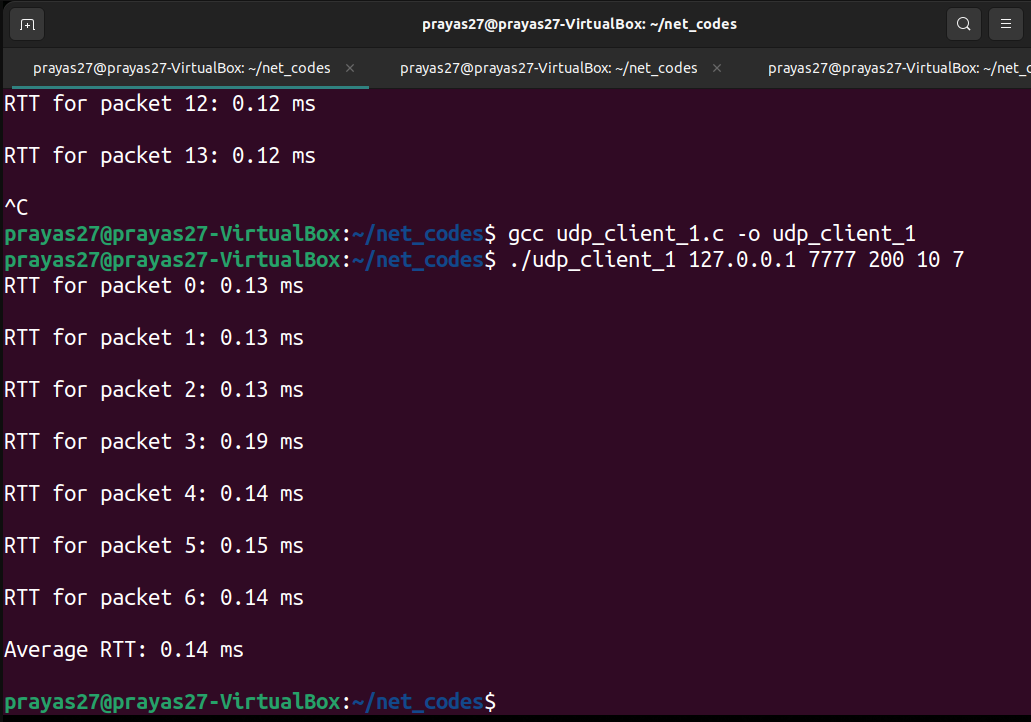
    return 0;

}

**Below is some demo SS from virtualbox UBUNTU terminal**

**The server and port both are running at port number 7777 …**





**Here IP 🡪 127.0.0.1 (local host)**

**7777 🡪 port number , 200 🡪 payload length , 10 🡪 TTL , 7 🡪 num\_packets**

**PART 2**

Note: Once the aboveproblem is completed, then attempt the following extension –

This assignment is an extension of above problem. Create a copy of the developed code and make the required changes in this extension. Only ClientPacketGen.c/cpp will need some modification.

Technique

On receiving a datagram from Server, the Client program decrements the TTL value and checks if this new value is zero. If the new TTL is not zero, the Client sends the datagram (with the decremented TTL value) back to Server. However, if TTL is zero, the Client calculates the difference between the current time and the time of datagram first sent. Let's call this as the “Cumulative RTT”. The Client saves the “Cumulative RTT” value to a file (in a new line).

Running Environment:

As stated earlier, run the Serve on your laptop/terminal computer, and the Client is at Hamsa (10.2.1.40) or Hanau (10.2.1.41).

Every time you run the Client, it generates new datagrams with TTL set to T and payload length P. The value of P and T and the name of the output file for storing the “*Cumulative RTT*” should be entered as command line arguments when executing the Client. The P should be within the range of 100 to 1000 bytes, and the T between 2 and 20 (and must be even). The Client program

initiates the packet transmission process for a total of 50 datagrams (using the sequence numbers 0, 1, 2, ..., 49) and the stops.

At first, run the Client for T = 2 and different values of P = 100, 200, 300, . . . , 1000. Plot a scatter-plot (using any suitable software, such as Matlab, Gnuplot, etc.) of “Cumulative RTT” for all 50 datagrams vs. P for the different values of P when T = 2.

What do you observe? What information does the slope of the graph contain?

Repeat the process when T = 8 and 16 and mention your understanding.

***Client code***

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <time.h>

#define MAX\_PACKET\_SIZE 1024

#define MAX\_FILENAME\_SIZE 256

#define NUM\_PACKETS 50

*// Packet structure*

typedef *struct* {

*int* sequence\_number;

*int* ttl;

*int* payload\_length;

*char* payload[MAX\_PACKET\_SIZE];

} Packet;

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

*// Check for correct number of arguments*

    if (argc != 6) {

        fprintf(stderr, "Usage: %s <ServerIP> <ServerPort> <P> <TTL> <output\_file>\n", argv[0]);

        exit(EXIT\_FAILURE);

    }

*// Parse command-line arguments*

*char* \*server\_ip = argv[1];

*int* server\_port = atoi(argv[2]);

*int* payload\_length = atoi(argv[3]);

*int* ttl = atoi(argv[4]);

*char* \*output\_file = argv[5];

*// Open file for saving Cumulative RTT values*

    FILE \*file = fopen(output\_file, "a");

    if (file == NULL) {

        perror("Error opening file");

        exit(1);

    }

*// Create socket*

*int* client\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);

    if (client\_socket < 0) {

        perror("Socket creation failed");

        exit(1);

    }

*// Server address*

*struct* sockaddr\_in server\_address;

    memset(&server\_address, 0, sizeof(server\_address));

    server\_address.sin\_family = AF\_INET;

    server\_address.sin\_port = htons(server\_port);

    server\_address.sin\_addr.s\_addr = inet\_addr(server\_ip);

*// Generate payload*

    srand(time(NULL));

    Packet packet;

    packet.payload\_length = payload\_length;

    for (*int* j = 0; j < packet.payload\_length; j++) {

        packet.payload[j] = 'A' + (rand() % 26); *// Random uppercase letter*

    }

*// Send packets to server*

*struct* timespec start\_time, end\_time;

*double* total\_rtt = 0.0; *// Total round trip time*

*int* num\_sent = 0; *// Number of packets sent*

    for (*int* i = 0; i < NUM\_PACKETS; i++)

    {

*// Record start time*

        clock\_gettime(CLOCK\_MONOTONIC, &start\_time);

*// Send packet to server*

        packet.sequence\_number = i;

        packet.ttl = ttl;

        sendto(client\_socket, &packet, sizeof(Packet), 0, (*struct* sockaddr \*)&server\_address, sizeof(server\_address));

*// Receive response from server*

*// printf("") ;*

*// Record end time*

*//clock\_gettime(CLOCK\_MONOTONIC, &end\_time);*

*// Calculate RTT for this packet*

*//double rtt = (end\_time.tv\_sec - start\_time.tv\_sec) \* 1000.0 + (end\_time.tv\_nsec - start\_time.tv\_nsec) / 1000000.0;*

*// total\_rtt += rtt;*

*// num\_sent++;*

*// If all packets sent for this TTL and payload length, calculate and save cumulative RTT*

        if (num\_sent == NUM\_PACKETS)

        {

            fprintf(file, "Cumulative RTT for TTL=%d, P=%d: %.2f\n", ttl, payload\_length, total\_rtt);

        }

*// Sleep for demonstration purposes*

*// usleep(10000); // Sleep for 10 milliseconds between each packet*

        sleep(1) ;

    }

    clock\_gettime(CLOCK\_MONOTONIC, &end\_time);

*double* rtt = (end\_time.tv\_sec - start\_time.tv\_sec) \* 1000.0 + (end\_time.tv\_nsec - start\_time.tv\_nsec) / 1000000.0;

*// total\_rtt += rtt;*

    fprintf(file, "Cumulative RTT for TTL=%d, P=%d: %.4f\n", ttl, payload\_length, rtt);

*// Close file*

    fclose(file);

*// Close socket*

    close(client\_socket);

    return 0;

}

Since server code is provided by sir in **10.2.1.44 , port 5555** , which is similar to our code in the prev assignment .

