Pratham U-210905407

Computer Networks Lab 3: Wireshark & GNS3

1) CODE

1. Write a UDP client-server program where t he client sends rows of a matrix, and the servercombines them together as a matrix.

2. Write a client program to send a manually crafted HTTP request packet to a Web Server and display all fields received in HTTP Response at client Side. (for this question same code is tested over different computers)

Solution:

server.c

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

#define PORT 5001
#define MAX_ROWS 100

void process_client(int client_socket, int cols, struct sockaddr_in client_address) {
   int matrix[MAX_ROWS][cols];
   int rows = 0;

   while (1) {
      int row_data[cols];
      ssize_t n = recv(client_socket, row_data, cols * sizeof(int), 0);
      if (n <= 0) {
         perror("Error receiving data");
         break;
      }
      if (row_data[0] == -1) {
         break;
      }
      for (int j = 0; j < cols; j++) {
         matrix[rows][j] = row_data[j];
      }
      rows++;

      int choice = 0;
      printf("Do you want to add another row (1:YES 0:NO): ");
      scanf("%d", &choice);

      n = sendto(client_socket, &choice, sizeof(int), 0, (struct sockaddr *)&client_address,
sizeof(client_address));
      if (n < 0) {
```

```
            perror("Error sending data");
            break;
        }

        if (choice == 0) {
            break;
        }

        if (rows >= MAX_ROWS) {
            printf("Matrix has reached its max size\n");
            break;
        }
    }
    printf("Received matrix from client.\n");
    printf("Matrix:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int server_socket, client_socket;
    struct sockaddr_in server_address, client_address;
    socklen_t len;
    int cols;

    server_socket = socket(AF_INET, SOCK_DGRAM, 0);
    if (server_socket == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }
    bzero(&server_address, sizeof(server_address));
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(PORT);
    server_address.sin_addr.s_addr = inet_addr("172.16.48.71");
    bind(server_socket, (struct sockaddr *)&server_address, sizeof(server_address));

    len = sizeof(client_address);
    recvfrom(server_socket, &cols, sizeof(int), 0, (struct sockaddr *)&client_address, &len);
    printf("Connected to client.\n");

    // Pass the client socket and client address to the processing function
    process_client(server_socket, cols, client_address);

    close(server_socket);
    printf("Server closed.\n");
    return 0;
}
```

client.c

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <stdio.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdlib.h>

#define PORT 5001

void clifunc(int sockfd, int arr_size, struct sockaddr_in *server_address) {
    int n;
    int arr[arr_size];
    while (1) {
        for (int i = 0; i < arr_size; i++) {
            printf("Enter element %d: ", i);
            scanf("%d", &arr[i]);
        }
        n = sendto(sockfd, arr, arr_size * sizeof(int), 0, (struct sockaddr *)server_address,
sizeof(*server_address));
        if (n < 0) {
            perror("Error sending data");
            break;
        }

        int choice = 0;
        n = recvfrom(sockfd, &choice, sizeof(int), 0, NULL, NULL);
        if (n < 0) {
            perror("Error receiving data");
            break;
        }

        if (choice == 0) {
            break;
        }
    }
}

int main() {
    int sockfd;
    int len;
    struct sockaddr_in server_address;
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }
    bzero(&server_address, sizeof(server_address));
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(PORT);
    server_address.sin_addr.s_addr = inet_addr("172.16.48.84");

    int col;
    printf("Enter the number of columns: ");
    scanf("%d", &col);
    sendto(sockfd, &col, sizeof(int), 0, (struct sockaddr *)&server_address, sizeof(server_address));
    clifunc(sockfd, col, &server_address);
```

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

```
student@selab-25:~/210905407/week3$ ./client
Enter the number of columns: 3
Enter element 0: 1
Enter element 1: 2
Enter element 2: 3
Enter element 0: 1
Enter element 1: 2
Enter element 2: 3
```

```
student@selab-25:~/210905407/week3$ ./server
Connected to client.
Do you want to add another row (1:YES 0:NO): 1
Do you want to add another row (1:YES 0:NO): 0
Received matrix from client.
Matrix:
1 2 3
1 2 3
Server closed.
```

```
student@selab-25:~/210905407/week3$ ./client
Enter the number of columns: 3
Enter element 0: 1
Enter element 1: 2
Enter element 2: 3
Enter element 0: 4
Enter element 1: 5
Enter element 2: 6
Enter element 0: 7
Enter element 1: 8
Enter element 2: 9
student@selab-25:~/210905407/week3$
```

2. Analyzing UDP datagrams using Wireshark:

• Start your web browser and clear the browser's cache memory, but do not access any website yet.

• Open Wireshark and start capturing.

• Go back to your web browser and retrieve any file from a website. Wireshark starts capturing packets.

• After enough packets have been captured, stop Wireshark, and save the captured file.

• Using the captured file, analyze TCP & UDP packets captured .Note: DNS uses UDP for name resolution & HTTP uses TCP

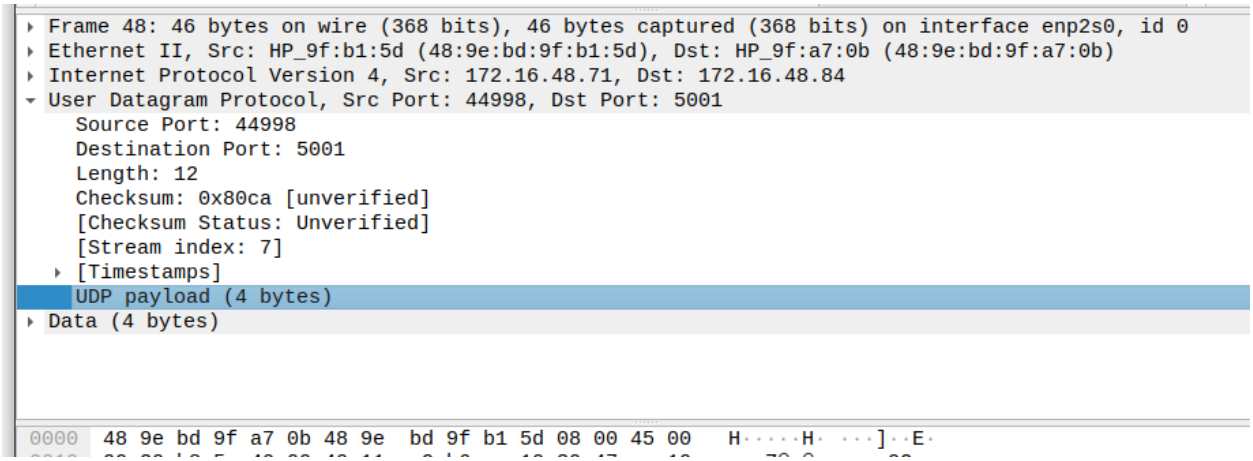
Using the captured information, answer the following questions in your lab report.

A.

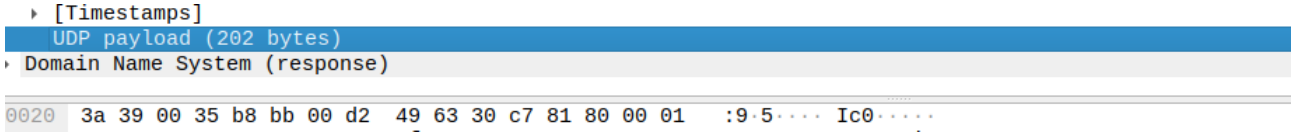
In the packet list pane, select the first DNS packet. In the packet detail pane, select the

User Datagram Protocol. The UDP hexdump will be highlighted in the packet byte lane.
Using the hexdump, Answer the following:
a. the source port number.
b. the destination port number.
c. the total length of the user datagram.
d. the length of the data.
whether the packet is directed from a client to a server or vice versa.
f. the application-layer protocol.
g. whether a checksum is calculated for this packet or not.
B. What are the source and destination IP addresses in the DNS query message? What are those
addresses in the response message? What is the relationship between the two?
C. What are the source and destination port numbers in the query message? What are those
addresses in the response message? What is the relationship between the two? Which port
number is a well-known port number?
D. What is the length of the first packet? How many bytes of payload are carried by the first
packet?

Solution:
A)f) UDP
B) The relationship between the two: sources = client , responses = server
A) B) C) D)



Part I: Connection-Establishment Phase
Identify the TCP packets used for connection establishment. Note that the last packet used for
connection establish may have the application-layer as the source protocol.



Questions

Using the captured information, answer the following question in your lab report about packets
used for connection establishment.
1. What are the socket addresses for each packet?

```
    [Stream index: 7]
  ▸ [Timestamps]
    UDP payload (4 bytes)
▸ Data (4 bytes)
```

2. What flags are set in each packet?

```
Flags: 0x40, Don't fragment
```

3. What are the sequence number and acknowledgment number of each packet?

```
[TCP Segment Len: 0]
Sequence Number: 0    (relative sequence number)
Sequence Number (raw): 3255551709
[Next Sequence Number: 1    (relative sequence number)]
Acknowledgment Number: 0
Acknowledgment number (raw): 0
1000 .... = Header Length: 32 bytes (8)
```

4. What are the window size of each packet?
Ans :64240

Part II: Data-Transfer Phase
The data-transfer phase starts with an HTTP GET request message and ends with an HTTP OK
message.
Questions
Using the captured information, answer the following question in your lab report about packets
used for data transfer.

```
 608 13.26802479  185.27.134.126   172.16.48.71     HTTP   558 HTTP/1.1 302 Found  (text/html)
1173 13.661149818 172.16.48.71     35.232.111.17    HTTP   153 GET / HTTP/1.1
```

1. What TCP flags are set in the first data-transfer packet (HTTP GET message)?

```
▸ Flags: 0x40, Don't fragment
  ...0 0000 0000 0000 = Fragment Offset: 0
```

2. How many bytes are transmitted in this packet?
87

3. How often does the receiver generate an acknowledgment? To which acknowledgment rule
(defined in Page 200 in the textbook) does your answer correspond to?
In many standard implementations of communication protocols like TCP 1 acknowledgment per received segment.

4. How many bytes are transmitted in each packet? How are the sequence and acknowledgment
numbers related to number of bytes transmitted?

```
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 139
```

139-20=119

5. What are the original window sizes that are set by the client and the server? Are these numbers
expected? How do they change as more segments are received by the client?

```
Window: 502
[Calculated window size: 64256]
[Window size scaling factor: 128]
```

The initial window sizes can vary based on the specific implementation, operating system, and network configuration. They are chosen to provide a balance between efficient data transfer and avoiding overwhelming the receiver. Generally, larger initial window sizes can lead to faster data transfer but could potentially cause congestion if the network path has limited capacity.

6. Explain how the window size is used in flow control:

In flow control, the window size refers to the number of bytes or packets that a sender can transmit to a receiver before expecting an acknowledgment. It regulates the rate at which data is sent, preventing the sender from overwhelming the receiver. As the receiver successfully receives data, it sends acknowledgments, allowing the sender to adjust the window size and continue sending more data. The window size effectively balances the sender's transmission speed with the receiver's processing capabilities.

7. What is the purpose of the HTTP OK message in the data transfer phase: The HTTP OK message, also known as the "200 OK" status code, is an HTTP response status code indicating that the request has succeeded. In the context of the data transfer phase, this response is crucial for indicating that the client's request for a particular resource or action was successful. It serves as a confirmation that the server has processed the client's request and is ready to provide the requested data or continue with the requested action. This status code is an important part of the client-server communication protocol, allowing effective data transfer and interaction between web clients and servers.

Part III: Connection Termination Phase
The data-transfer phase is followed by the connection termination phase. Note that some packets
used in the connection-termination phase may have the source or sink protocol at the application

layer. Find the packets used for connection termination.

Questions

Using the captured information, answer the following question in your lab report about packets
used for connection termination.

How many TCP segments are exchanged for this phase?

- In this captured connection termination phase, a total of 4 TCP segments were exchanged.

2. **Which end point started the connection termination phase?**

- The client end initiated the connection termination phase.

3. **What flags are set in each of the segments used for connection termination?**

- Segment 1 (Client to Server):
  - Flags: FIN (Finish)
  - Purpose: The client initiates termination by sending a FIN flag to the server, indicating it has finished sending data.
- Segment 2 (Server to Client):
  - Flags: ACK (Acknowledgment)
  - Purpose: The server acknowledges the client's FIN flag, confirming the receipt of all client data.
- Segment 3 (Server to Client):
  - Flags: FIN (Finish)
  - Purpose: The server initiates its termination process by sending a FIN flag to the client, indicating it has finished sending data.
- Segment 4 (Client to Server):
  - Flags: ACK (Acknowledgment)
  - Purpose: The client acknowledges the server's FIN flag, confirming the receipt of all server data and completing the termination process.