THEORY

TERMWORK 1: Implementing IPC using Pipes and message queues.

-- This experiment explores Inter Process Communication (IPC) through the implementation of pipes and message queues.
Pipes enable one-way communication within a process,
while message queues facilitate communication between different processes,
emphasizing the importance of coordinated message passing in concurrent computing.

TERMWORK 2: Implementing client server communication using socket programming that uses connection-oriented protocol at transport layer.

-- Socket programming is a way of connecting two nodes on a network to communicate with each other.
One socket (node) listens on a particular port at an IP, while the other socket reaches out to form a connection.
The server forms the listener socket, and the client reaches out to the server. When creating a connection between client and server using TCP, functionalities include high reliability, less critical transmission time, and TCP's handling of reliability, congestion control, error checking, and error recovery.

TERMWORK 3: Implement the distance vector routing algorithm

-- Routing Protocols: Distance vector routing is a class of routing protocols used in computer networks to determine the best path for data packets to travel from the source to the destination.
An internet needs dynamic routing table in order to incorporate changes into the table whenever there is a change in internet.
They need to be updated when the route is down, or when a better route is created.

Distance Vector routing:

Dynamic Routing: Routers quickly adapt routing decisions in milliseconds, using Vector tables for distance information, updated through neighbour exchanges.
Vector Table Entries: Entries for each destination router contain preferred routes and estimated hop distances, enabling rapid routing adjustments.

TERMWORK 1: Implementing IPC using Pipes and message queues.

WRITER.C

```c
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX 50

struct msg_buffer {
   long mesg_type;
   char mesg_text[100];

}message;

int main() {
   key_t key;
   int msgid;

   key = ftok("progfile", 65);
   msgid = msgget(key, 0666 | IPC_CREAT);
   message.mesg_type = 1;
   printf("Write data: \n");
   fgets(message.mesg_text, MAX, stdin);
   msgsnd(msgid, &message, sizeof(message), 0);
   printf("Data sent is : %s\n", message.mesg_text);
   return 0;
}
```

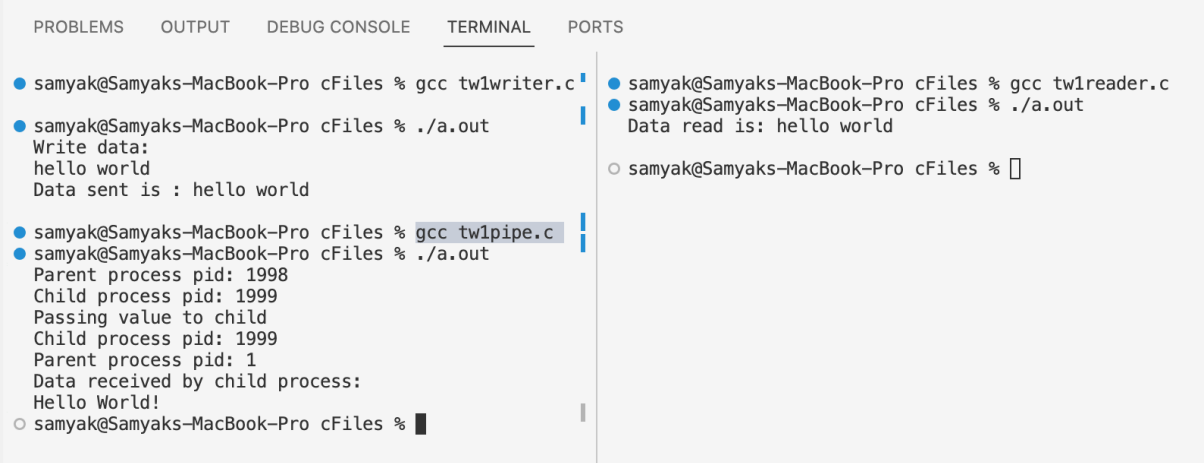
READER.C

```c
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX 50

struct msg_buffer {
   long mesg_type;
   char mesg_text[100];

}message;

int main() {
   key_t key;
   int msgid;
   key = ftok("progfile", 65);
   msgid = msgget(key, 0666 | IPC_CREAT);
   msgrcv(msgid, &message, sizeof(message), 1, 0);
   printf("Data read is: %s\n", message.mesg_text);
   msgctl(msgid, IPC_RMID, NULL);
   return 0;
}
```

PIPE.C

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>

int main() {
    int fd[2], n;
    char buffer[100];
    pid_t p;

    pipe(fd);
    p = fork();

    if (p > 0) {
        printf("Parent process pid: %d\n", getpid());
        printf("Child process pid: %d\n", p);
        printf("Passing value to child\n");
        write(fd[1], "Hello World!\n", 13);
    } else {
        printf("Child process pid: %d\n", getpid());
        printf("Parent process pid: %d\n", getppid());
        n = read(fd[0], buffer, 100);
        printf("Data received by child process: \n");
        write(1, buffer, n);
    }

    return 0;
}
```

TW1 OUTPUT:



TERMWORK 2: Implementing client server communication using socket programming that uses connection-oriented protocol at transport layer.

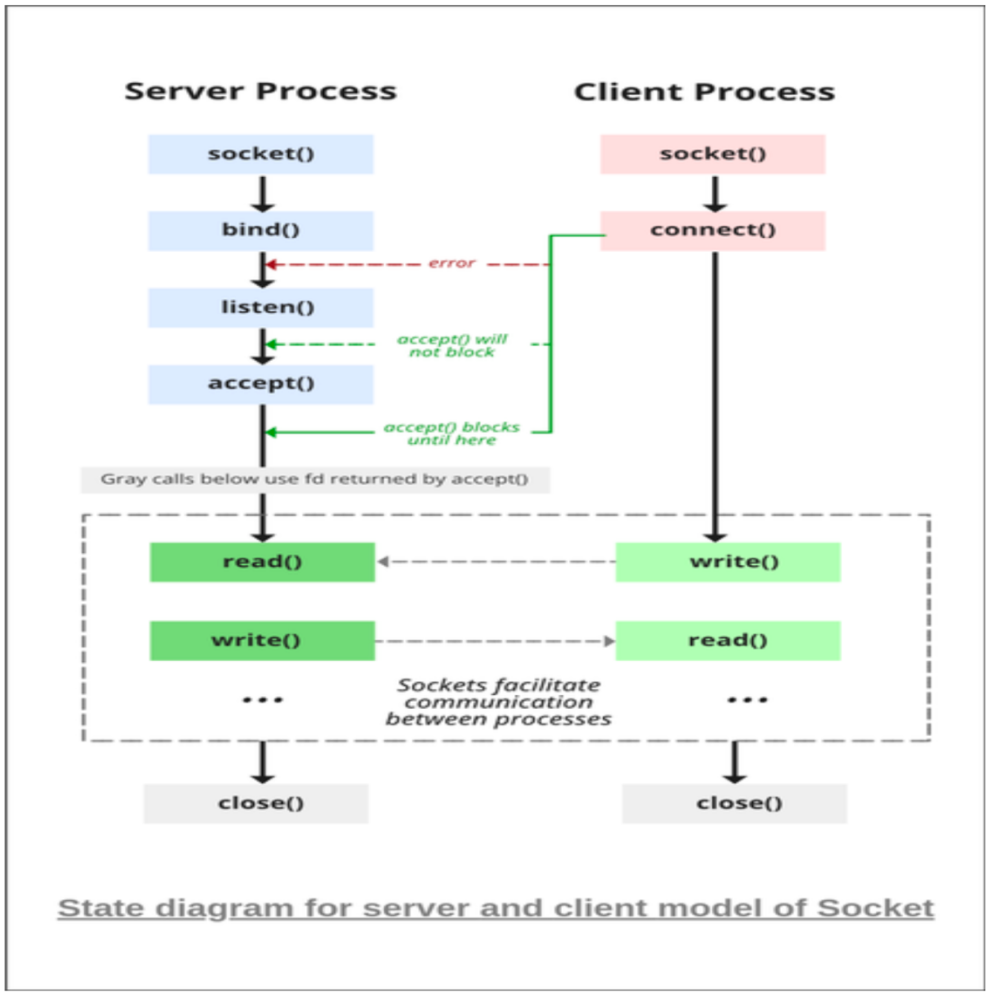
SERVER.C

```c
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <string.h>
#define PORT 4444

int main() {
   int listenfd, connfd;
   struct sockaddr_in servAddr, cliAddr;
   socklen_t clilen;
   char buffer[1024];

   listenfd = socket(AF_INET, SOCK_STREAM, 0);
   printf("[+] Server socket created successfully\n");

   bzero(&servAddr, sizeof(servAddr));
   servAddr.sin_family = AF_INET;
   servAddr.sin_port = htons(PORT);
   servAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

   bind(listenfd, (struct sockaddr *) &servAddr,
sizeof(servAddr));
   printf("[+] Bind to PORT %d successful\n", PORT);

   listen(listenfd, 5);
   printf("[+] Listening...\n");

   connfd = accept(listenfd, (struct sockaddr *) &cliAddr,
&clilen);

   strcpy(buffer, "Hello World!");
   send(connfd, buffer, strlen(buffer), 0);
   printf("[+] Data sent to client: %s\n", buffer);

   printf("[+] Closing the connection\n");
   return 0;
}
```

CLIENT.C

```c
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define PORT 4444

int main() {
   int sockfd;
   struct sockaddr_in servAddr;
   char buffer[1024];

   sockfd = socket(AF_INET, SOCK_STREAM, 0);
   printf("[+] Client socket created successfully\n");

   bzero(&servAddr, sizeof(servAddr));
   servAddr.sin_family = AF_INET;
   servAddr.sin_port = htons(PORT);
   servAddr.sin_addr.s_addr = inet_addr("127.0.0.1");

   connect(sockfd, (struct sockaddr *) &servAddr,
sizeof(servAddr));
   printf("[+] Connected to server\n");

   recv(sockfd, buffer, 1024, 0);
   printf("[+] Data received from server: %s\n", buffer);
   printf("[+] Closing the connection\n");
   return 0;
}
```

(optional)



State diagram for server and client model of Socket

TW2 OUTPUT:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
samyak@Samyaks-MacBook-Pro cFiles % gcc tw2server.c

samyak@Samyaks-MacBook-Pro cFiles % ./a.out
  [+] Server socket created successfully
  [+] Bind to PORT 4444 successful
  [+] Listening...
  [+] Data sent to client: Hello World!
  [+] Closing the connection
samyak@Samyaks-MacBook-Pro cFiles %
```

```
samyak@Samyaks-MacBook-Pro cFiles % gcc tw2client.c
samyak@Samyaks-MacBook-Pro cFiles % ./a.out
  [+] Client socket created successfully
  [+] Connected to server
  [+] Data received from server: Hello World!
  [+] Closing the connection
samyak@Samyaks-MacBook-Pro cFiles %
```

TERMWORK 3: Implement the distance vector routing algorithm

TW3.C

```c
#include <stdio.h>
#define NODES 10
#define NO_ROUTE 999
#define NO_HOP 1000

int no;

struct node {
   int a[NODES][4];
} router[NODES];

void init(int r) {
   int i;
   for (i = 1; i <= no; i++) {
      router[r].a[i][1] = i;
      router[r].a[i][2] = NO_ROUTE;
      router[r].a[i][3] = NO_HOP;
   }
   router[r].a[r][2] = 0;
   router[r].a[r][3] = r;
}

void inp(int r) {
   int i;
   printf("\nEnter distance from node %d to other nodes\n",
r);
   printf("Enter 999 if there is no direct route\n");
   for (i = 1; i <= no; i++) {
      if (i != r) {
         printf("Enter distance to node %d: ", i);
         scanf("%d", &router[r].a[i][2]);
         router[r].a[i][3] = i;
      }
   }
}

void display(int r) {
   int i;
   printf("\nThe routing table for node %d is as follows", r);
   for (i = 1; i <= no; i++) {
      if (router[r].a[i][2] == 999)
         printf("\n%d \t no link \t no hop",
router[r].a[i][1]);
      else
         printf("\n%d \t %d \t %d", router[r].a[i][1],
router[r].a[i][2], router[r].a[i][3]);
   }
}
```

```c
void dv_algo(int r) {
   int i, j, z;
   for (i = 1; i <= no; i++) {
      if (router[r].a[i][2] != 999 && router[r].a[i][2] != 0)
{
         for (j = 1; j <= no; j++) {
            z = router[r].a[i][2] + router[i].a[j][2];
            if (z < router[r].a[j][2]) {
               router[r].a[j][2] = z;
               router[r].a[j][3] = i;
            }
         }
      }
   }
}

int main() {
   int i, x, y;
   char choice = 'y';
   printf("Enter the number of nodes: ");
   scanf("%d", &no);
   for (i = 1; i <= no; i++) {
      init(i);
      inp(i);
   }
   printf("\nThe routing tables of nodes after initialization
are as follows");
   for (i = 1; i <= no; i++)
      display(i);
   printf("\n\nComputing shortest paths...\n");
   for (i = 1; i <= no; i++)
      dv_algo(i);
   printf("\nThe routing tables of nodes after computation of
shortest paths are as follows");
   for (i = 1; i <= no; i++)
      display(i);
   printf("\n");
   while (choice != 'n') {
      printf("\nEnter the nodes between which the shortest
distance is to be found: ");
      scanf("%d %d", &x, &y);
      getchar();
      printf("The length of the shortest path between nodes %d
and %d is %d\n", x, y, router[x].a[y][2]);
      printf("Continue? (y/n): ");
      scanf(" %c", &choice);
   }
   return 0;
}
```

TW3 OUTPUT:

```
samyak@Samyaks-MacBook-Pro cFiles %
gcc tw3.c
samyak@Samyaks-MacBook-Pro cFiles % ./a.out
Enter the number of nodes: 5

Enter distance from node 1 to other nodes
Enter 999 if there is no direct route
Enter distance to node 2: 1
Enter distance to node 3: 999
Enter distance to node 4: 999
Enter distance to node 5: 999

Enter distance from node 2 to other nodes
Enter 999 if there is no direct route
Enter distance to node 1: 1
Enter distance to node 3: 3
Enter distance to node 4: 4
Enter distance to node 5: 5

Enter distance from node 3 to other nodes
Enter 999 if there is no direct route
Enter distance to node 1: 999
Enter distance to node 2: 2
Enter distance to node 4: 3
Enter distance to node 5: 999

Enter distance from node 4 to other nodes
Enter 999 if there is no direct route
Enter distance to node 1: 999
Enter distance to node 2: 4
Enter distance to node 3: 3
Enter distance to node 5: 999

Enter distance from node 5 to other nodes
Enter 999 if there is no direct route
Enter distance to node 1: 999
Enter distance to node 2: 5
Enter distance to node 3: 999
Enter distance to node 4: 999

The routing tables of nodes after initialization
are as follows
The routing table for node 1 is as follows
1       0       1
2       1       2
3       no link         no hop
4       no link         no hop
5       no link         no hop
The routing table for node 2 is as follows
1       1       1
2       0       2
3       3       3
4       4       4
5       5       5
The routing table for node 3 is as follows
1       no link         no hop
2       2       2
3       0       3
4       3       4
5       no link         no hop
The routing table for node 4 is as follows
1       no link         no hop
2       4       2
3       3       3
4       0       4
5       no link         no hop
The routing table for node 5 is as follows
1       no link         no hop
2       5       2
3       no link         no hop
4       no link         no hop
5       0       5

Computing shortest paths...

The routing tables of nodes after computation of
shortest paths are as follows
The routing table for node 1 is as follows
1       0       1
2       1       2
3       4       2
4       5       2
5       6       2
The routing table for node 2 is as follows
1       1       1
2       0       2
3       3       3
4       4       4
5       5       5
The routing table for node 3 is as follows
1       3       2
2       2       2
3       0       3
4       3       4
5       7       2
The routing table for node 4 is as follows
1       5       2
2       4       2
3       3       3
4       0       4
5       9       2
The routing table for node 5 is as follows
1       6       2
2       5       2
3       8       2
4       9       2
5       0       5

Enter the nodes between which the shortest
distance is to be found: 1 5
The length of the shortest path between nodes
1 and 5 is 6
```

(optional)

Algorithm:

1. Send my routing table to all my neighbours whenever my link table changes.

2. When I get a routing table from a neighbour on port **P** with link metric **M:**

   - ❖ add **L** to each of the neighbour's metrics.
   - ❖ for each entry **(D, P', M')** in the updated neighbour's table

   i) if I do not have an entry for **D**, add **(D, P, M')** to my routing table

   ii) if I have an entry for **D** with metric **M"**, add **(D, P, M')** to my routing table if **M' < M"**

3. If my routing table has changed, send all the new entries to all my neighbours.

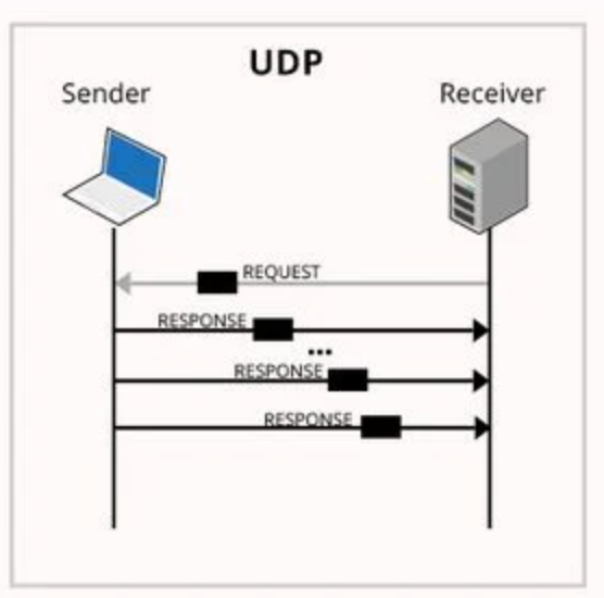
TERMWORK 4: Using WIRESHARK observe the data transferred in client server communication using UDP and identify the UDP datagram.

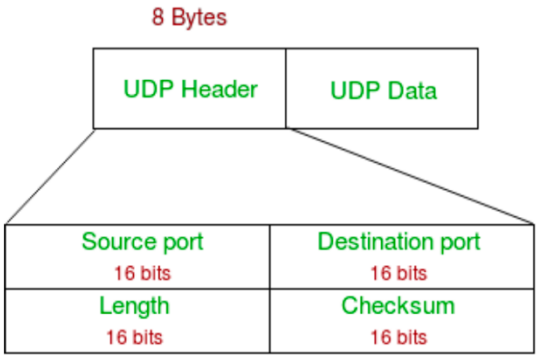
Program Execution Steps:

Capturing UDP packets with browser:

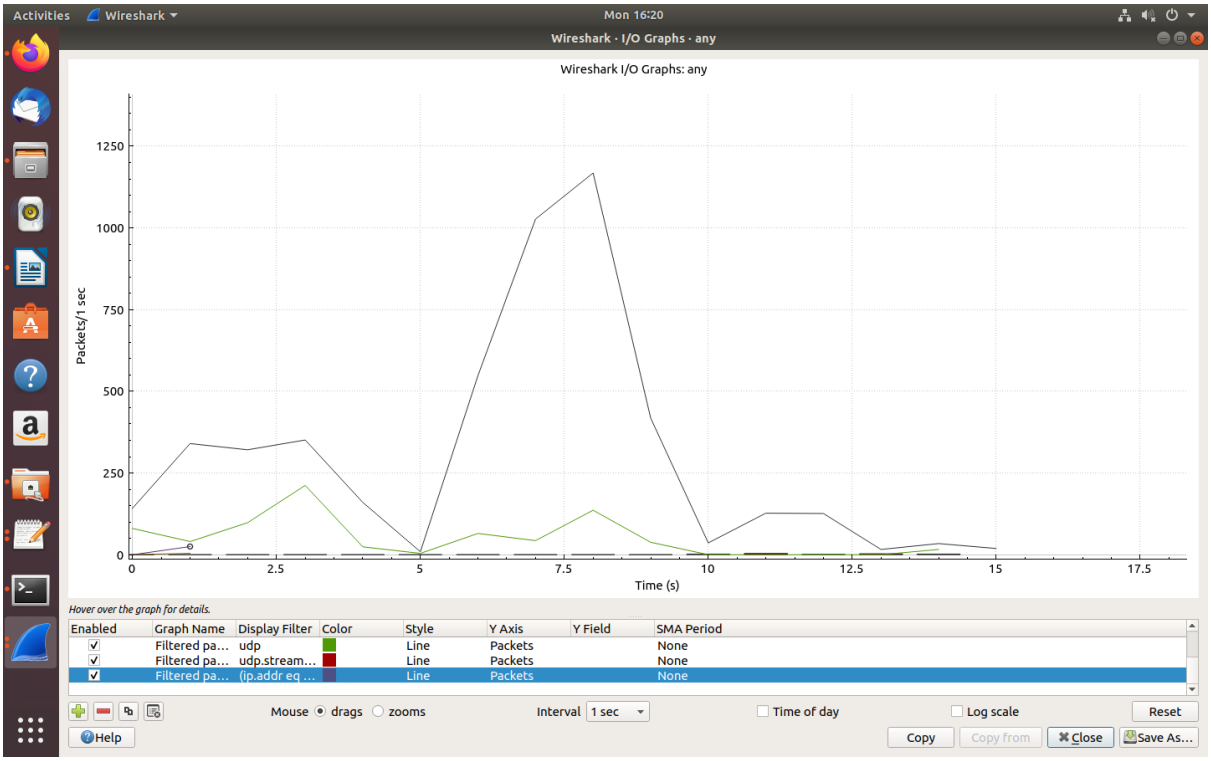
1. **Open Wireshark** and double-click on **any**-interface to start the packet capture process.
2. **Open** the **browser** and enter any website's fully qualified domain name in the browser address bar and hit enter.
3. After the site is fully loaded, **stop** the capturing process
4. in Wireshark go to edit in the menu bar and select find packet option or just press **CTRL+F.**
5. In Find Packet menu bar, select the String option in the display filter drop-down menu and enter the name of the website in the next box and click on find.
6. The **arrow** indicating **towards** the packet is the **request packet**, and the **arrow** coming **out** from the packet is the **response packet**.
7. **Click on any request or response** DNS packet and examine **UDP** packet.
8. Go to **statistics**: **Generate I/O Graph**, **Flow Graph** and study and analyze both the graphs

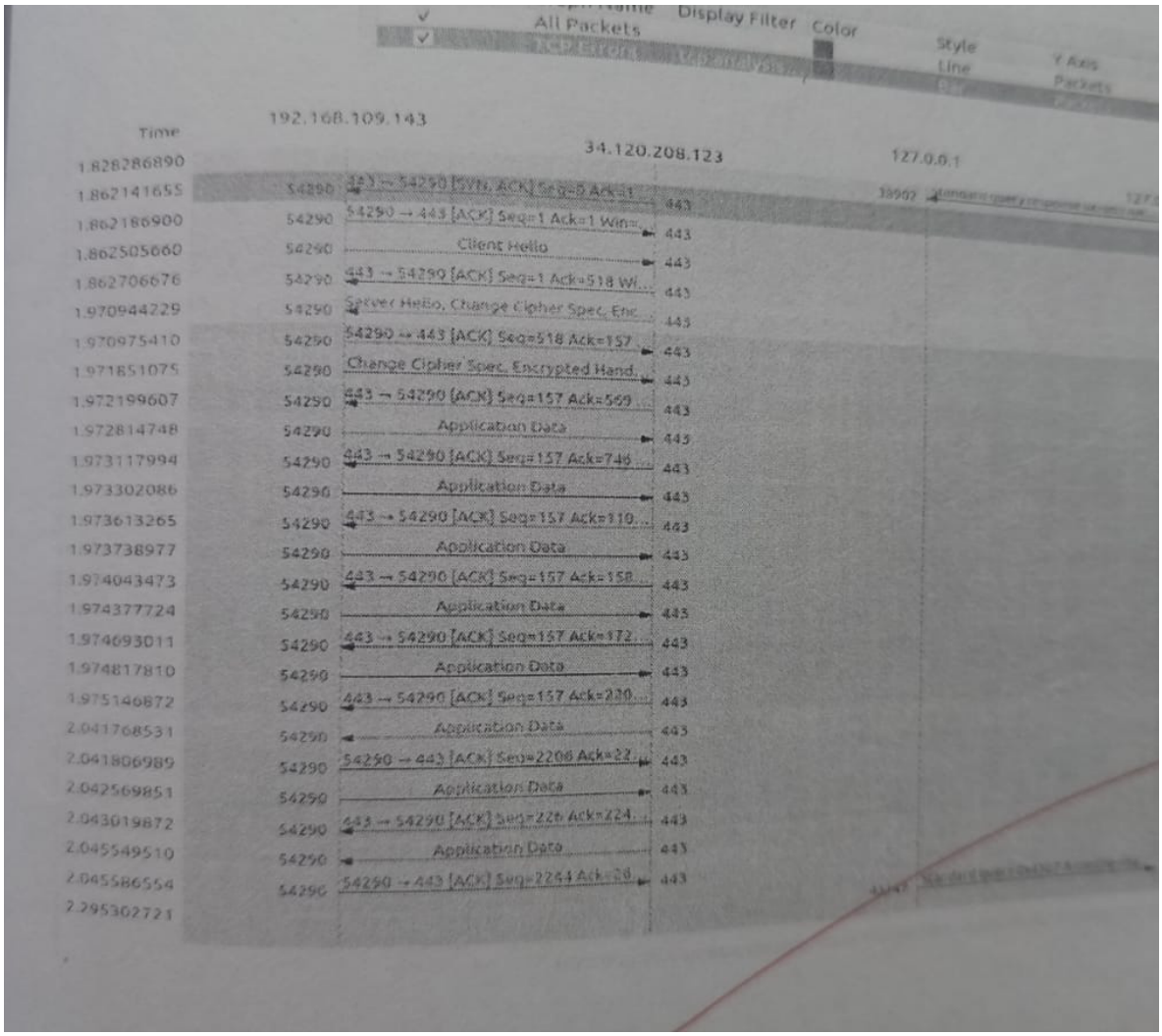
(optional)





TW4 OUTPUT:


Activities
Wireshark
Mon 16:20
Wireshark · I/O Graphs · any
Wireshark I/O Graphs: any
Packets/1 sec
Time (s)
Hover over the graph for details.
Enabled
Graph Name
Display Filter
Color
Style
Y Axis
Y Field
SMA Period
Filtered pa… udp Line Packets None
Filtered pa… udp.stream… Line Packets None
Filtered pa… (ip.addr eq … Line Packets None
Mouse drags zooms
Interval 1 sec
Time of day
Log scale
Reset
Help
Copy
Copy from
Close
Save As…


All Packets
Display Filter
Color
Style
Line
Y Axis
Packets
Time
192.168.109.143
34.120.208.123
127.0.0.1
1.828286890
1.862141655
1.862186900
1.862505060
1.862706676
1.970944229
1.970975410
1.971851075
1.972199607
1.972814748
1.973117994
1.973302086
1.973613265
1.973738977
1.974043473
1.974377724
1.974693011
1.974817810
1.975146872
2.041768531
2.041806989
2.042569851
2.043019872
2.045549510
2.045586554
2.295302721
54290 → 443 [SYN] Seq=0 Ack=1
54290 → 443 [ACK] Seq=1 Ack=1 Win=
Client Hello
443 → 54290 [ACK] Seq=1 Ack=518 Wi…
Server Hello, Change Cipher Spec, Enc…
54290 → 443 [ACK] Seq=518 Ack=157
Change Cipher Spec, Encrypted Hand…
443 → 54290 [ACK] Seq=157 Ack=569
Application Data
443 → 54290 [ACK] Seq=157 Ack=746
Application Data
443 → 54290 [ACK] Seq=157 Ack=110…
Application Data
443 → 54290 [ACK] Seq=157 Ack=158
Application Data
443 → 54290 [ACK] Seq=157 Ack=172…
Application Data
443 → 54290 [ACK] Seq=157 Ack=230…
Application Data
54290 → 443 [ACK] Seq=2206 Ack=22…
Application Data
443 → 54290 [ACK] Seq=228 Ack=224…
Application Data
54290 → 443 [ACK] Seq=2264 Ack=28…

TERMWORK 5: Using WIRESHARK analyse three-way handshaking connection establishment, data transfer and connection termination in client server communication using TCP.

Program Execution Steps:

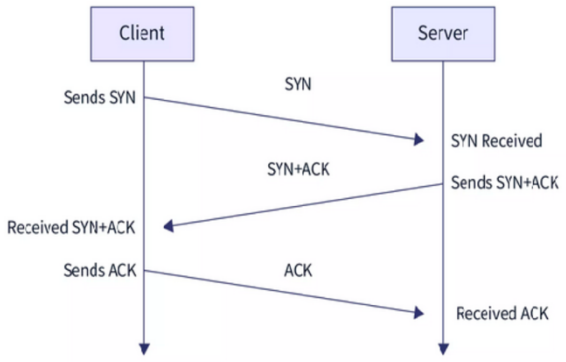
Capturing TCP packets with browser:

1. **Open Wireshark** and double-click on **any**-interface to start the packet capture process.
2. **Open** the **browser** and enter any website's fully qualified domain name in the browser address bar and hit enter.
3. After the site is fully loaded, **stop** the capturing process
4. Type the following in, apply a filter column and hit-enter:

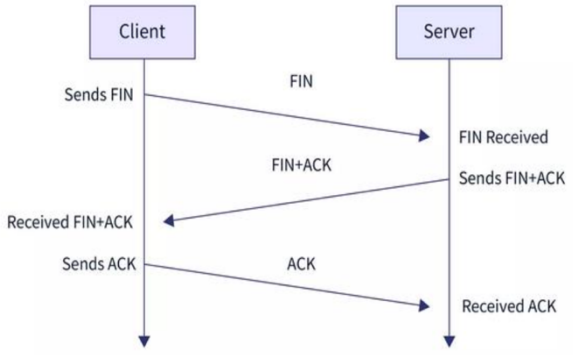
   tcp.flags.fin==1 and tcp.flags.ack==1

5. Select any one of these listed packets, right-click and hover on conversation filter and select **TCP**
6. Go to **statistics**: **Generate I/O Graph**, **Flow Graph** and study and analyze both the graphs

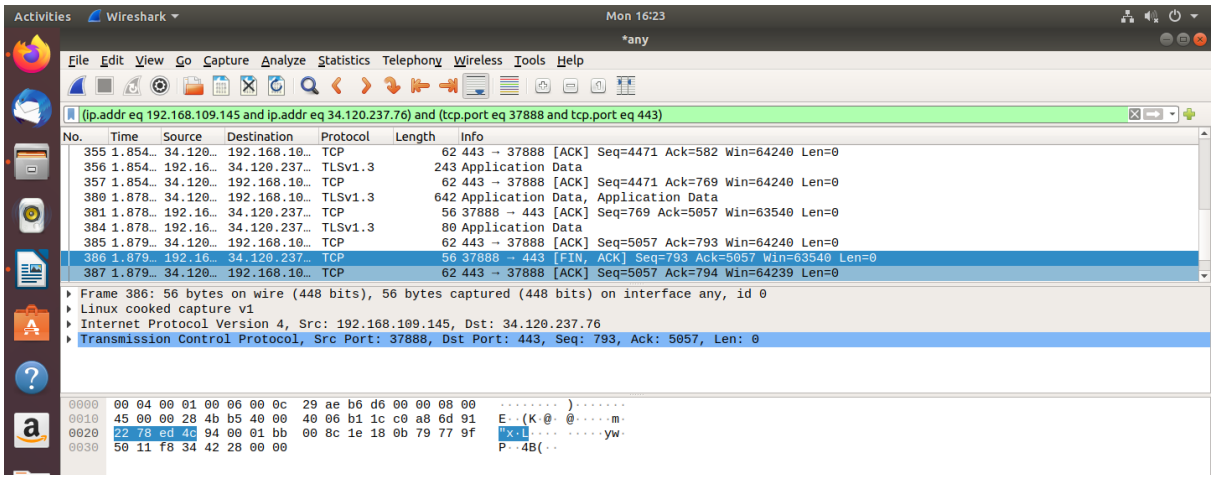
Diagram (on the blank page):



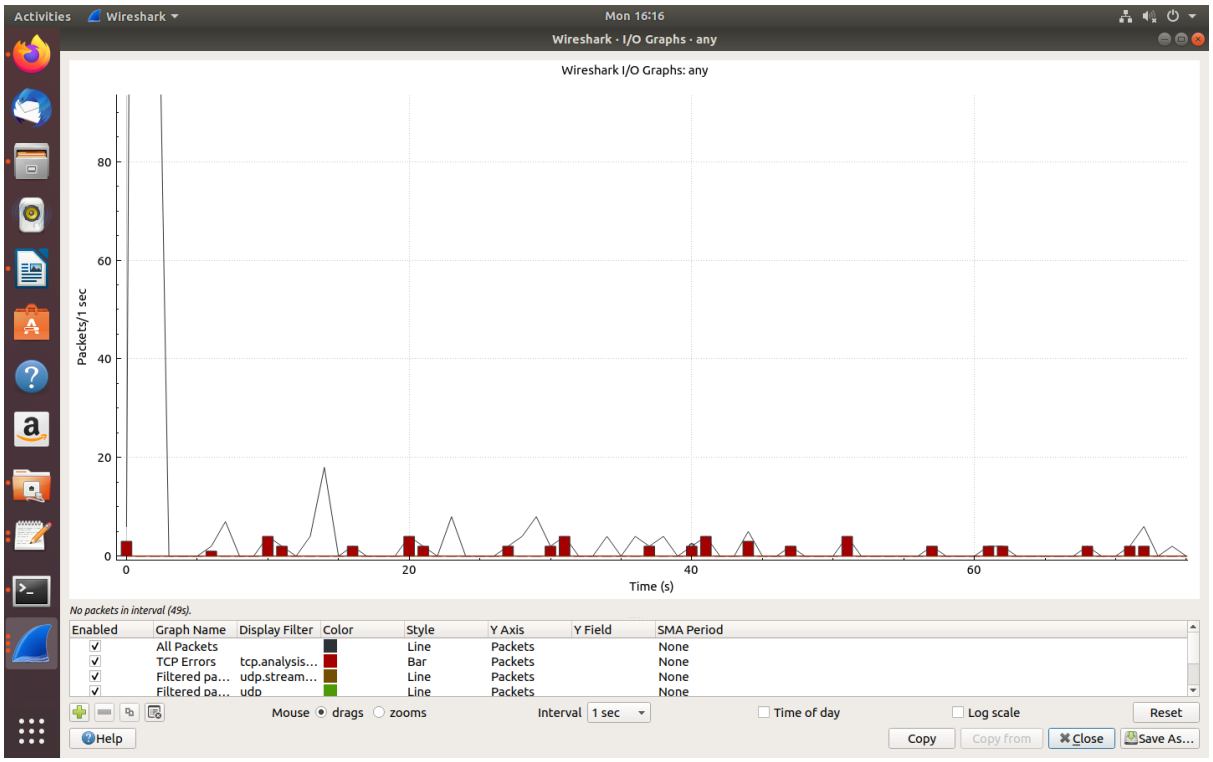
TCP 3-way handshake process

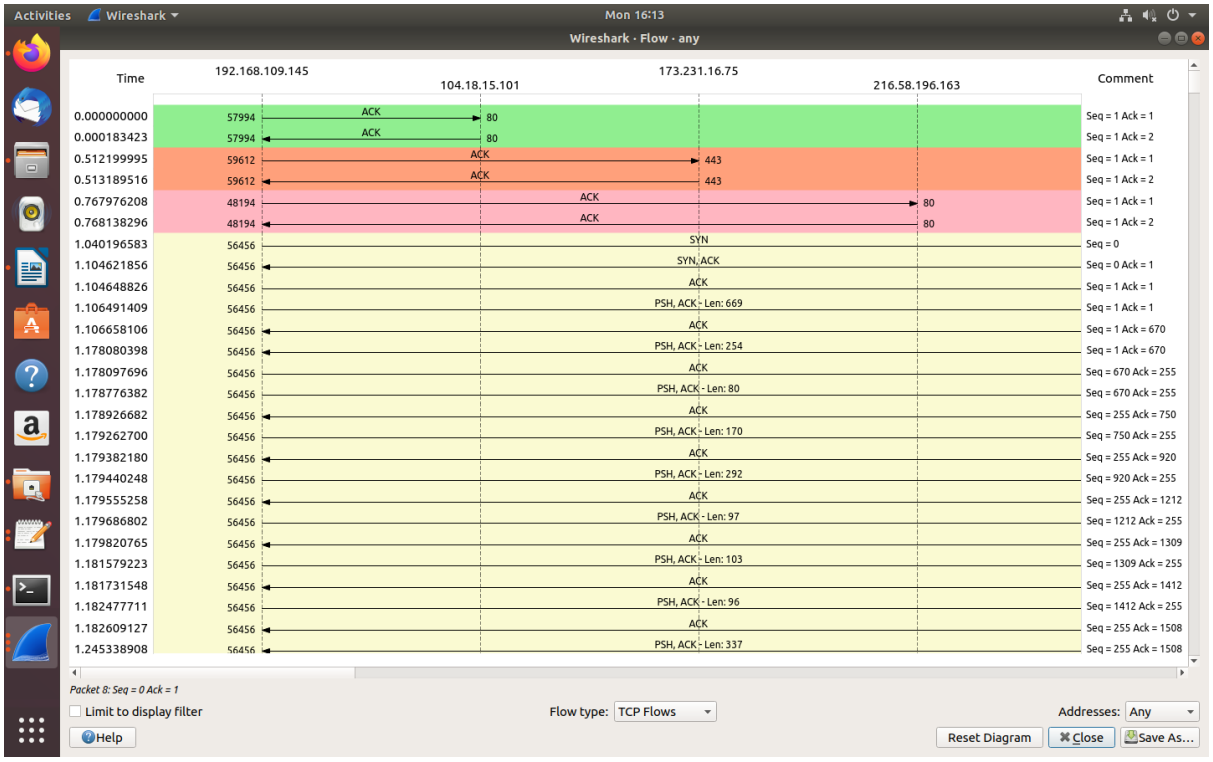


TCP 3-Way Handshake for Terminating the Connection



TW5 OUTPUT:


Activities
Wireshark
Mon 16:16
Wireshark · I/O Graphs · any
Wireshark I/O Graphs: any
Packets/1 sec
Time (s)
No packets in interval (49s).
Enabled
Graph Name
Display Filter
Color
Style
Y Axis
Y Field
SMA Period
All Packets
Line
Packets
None
TCP Errors
tcp.analysis...
Bar
Packets
None
Filtered pa...
udp.stream...
Line
Packets
None
Filtered pa...
udp
Line
Packets
None
Mouse drags zooms
Interval 1 sec
Time of day
Log scale
Reset
Help
Copy
Copy from
Close
Save As...


Activities
Wireshark
Mon 16:13
Wireshark · Flow · any
Time
192.168.109.145
104.18.15.101
173.231.16.75
216.58.196.163
Comment
0.000000000 57994 ACK 80 Seq = 1 Ack = 1
0.000183423 57994 ACK 80 Seq = 1 Ack = 2
0.512199995 59612 ACK 443 Seq = 1 Ack = 1
0.513189516 59612 ACK 443 Seq = 1 Ack = 2
0.767976208 48194 ACK 80 Seq = 1 Ack = 1
0.768138296 48194 ACK 80 Seq = 1 Ack = 2
1.040196583 56456 SYN Seq = 0
1.104621856 56456 SYN, ACK Seq = 0 Ack = 1
1.104648826 56456 ACK Seq = 1 Ack = 1
1.106491409 56456 PSH, ACK - Len: 669 Seq = 1 Ack = 1
1.106658106 56456 ACK Seq = 1 Ack = 670
1.178080398 56456 PSH, ACK - Len: 254 Seq = 1 Ack = 670
1.178097696 56456 ACK Seq = 670 Ack = 255
1.178776382 56456 PSH, ACK - Len: 80 Seq = 670 Ack = 255
1.178926682 56456 ACK Seq = 255 Ack = 750
1.179262700 56456 PSH, ACK - Len: 170 Seq = 750 Ack = 255
1.179382180 56456 ACK Seq = 255 Ack = 920
1.179440248 56456 PSH, ACK - Len: 292 Seq = 920 Ack = 255
1.179555258 56456 ACK Seq = 255 Ack = 1212
1.179686802 56456 PSH, ACK - Len: 97 Seq = 1212 Ack = 255
1.179820765 56456 ACK Seq = 255 Ack = 1309
1.181579223 56456 PSH, ACK - Len: 103 Seq = 1309 Ack = 255
1.181731548 56456 ACK Seq = 255 Ack = 1412
1.182477711 56456 PSH, ACK - Len: 96 Seq = 1412 Ack = 255
1.182609127 56456 ACK Seq = 255 Ack = 1508
1.245338908 56456 PSH, ACK - Len: 337 Seq = 255 Ack = 1508
Packet 8: Seq = 0 Ack = 1
Limit to display filter
Flow type: TCP Flows
Addresses: Any
Help
Reset Diagram
Close
Save As...

TERMWORK 6: Simulate a Full duplex connection in a wired network using NS3.

Step 1 : Open UBUNTU and locate and open **ns-allinone-3.28** folder on Desktop.

Step 2 : Go to **ns-3.28** folder and open **examples**->**tutorial**->**first.cc**

Step 3 : Copy first.cc and **paste** it in ns-3.28->**scratch** folder.

Remember that scratch folder should contain only one .cc example file and it must contain scratch executable file named scratchsimulator.cc and other files can be deleted.

Step 4 : In first.cc , include the following code. (Before – "Simulator::Run ();")

```
#include "ns3/netanim-module.h"
AnimationInterface anim("first,   xml");
AsciiTraceHelper ascii;
pointToPoint.EnableAsciiAll(ascii.CreateFileStream("first.tr"));
pointToPoint.EnablePcapAll("first");
```

Step 5 : Right click->**Open in terminal** and change working directory to Desktop by cd Desktop and type following commands to go to location where scratch executable file is located i.e. scratch folder.

Step 6 : cd ns-allinone-3.28 -> **cd ns-3.28**

Step 7 : ./waf build

Step 8 : Run the first.cc by entering following command.

./waf --run scratch/first

Step 9 : Once build is successful,

cd ../ (return to ns-allinone-3.28 folder)

ls

cd netanim-3.108 (enter into netanim-3.108)

Step 10 : Now to see the animation,

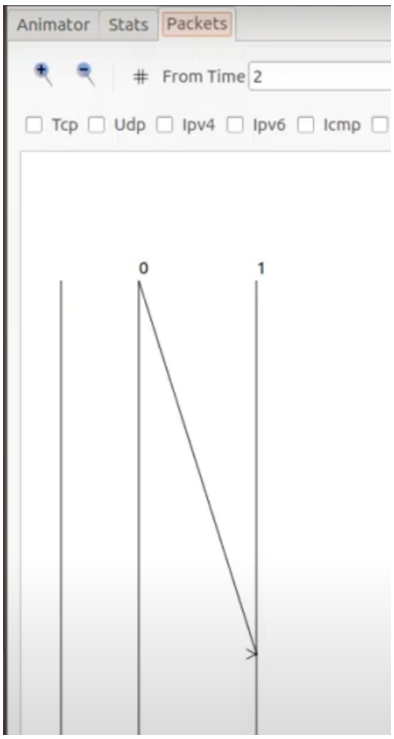
./NetAnim (we have to open NetAnim software)

Step 11 : In NetAnim, open **first , xml** by clicking on open XML trace file icon.

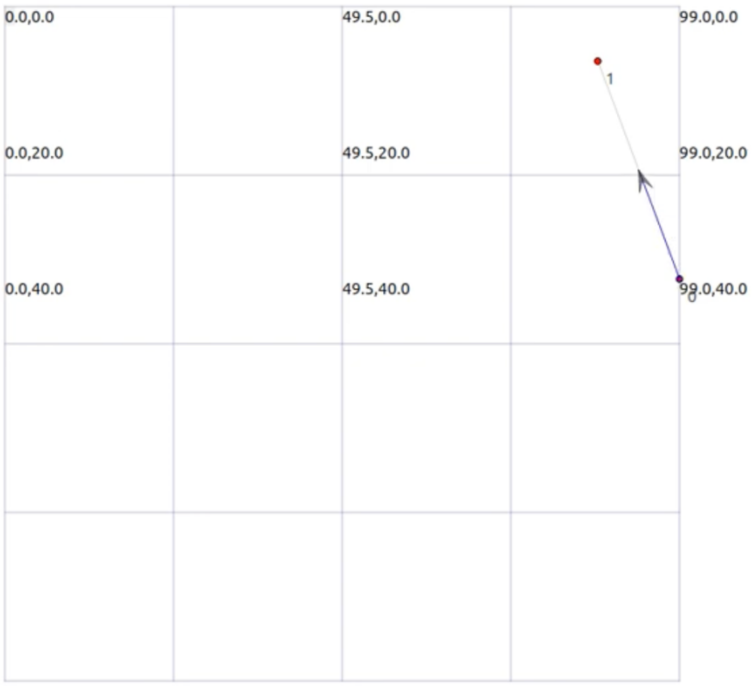
Step 12 : Click on **run** option/icon to see the animation. To see the packet transfer, open Packets Tab.

TW6 OUTPUT:

Packets



Animator



TERMWORK 7: Simulate a simple Wireless UDP application using NS3.

Step 1 : Open UBUNTU and locate and open **ns-allinone-3.28** folder on Desktop.

Step 2 : Go to **ns-3.28** folder and open **examples**->**tutorial**->**second.cc**

Step 3 : Copy second.cc and **paste** it in ns-3.28->**scratch** folder.

Remember that scratch folder should contain only one .cc example file and it must contain scratch executable file named scratchsimulator.cc and other files can be deleted.

Step 4 : In second.cc , include the following code. (Before – "Simulator::Run ();")

```
#include "ns3/netanim-module.h"
AnimationInterface anim("second, xml");
AsciiTraceHelper ascii;
pointToPoint.EnableAsciiAll(ascii.CreateFileStream("second.tr"));
pointToPoint.EnablePcapAll("second");
```

Step 5 : Right click->**Open in terminal** and change working directory to Desktop by cd Desktop and type following commands to go to location where scratch executable file is located i.e. scratch folder.

Step 6 : cd ns-allinone-3.28 -> **cd ns-3.28**

Step 7 : ./waf build

Step 8 : Run the second.cc by entering following command.

./waf --run scratch/second

Step 9 : Once build is successful,

cd ../ (return to ns-allinone-3.28 folder)

ls

cd netanim-3.108 (enter into netanim-3.108)

Step 10 : Now to see the animation,

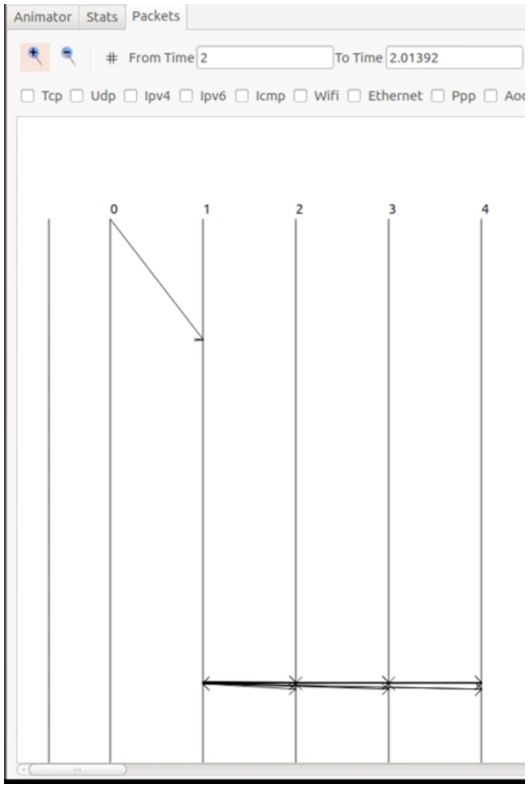
./NetAnim (we have to open NetAnim software)

Step 11 : In NetAnim, open **second , xml** by clicking on open XML trace file icon.

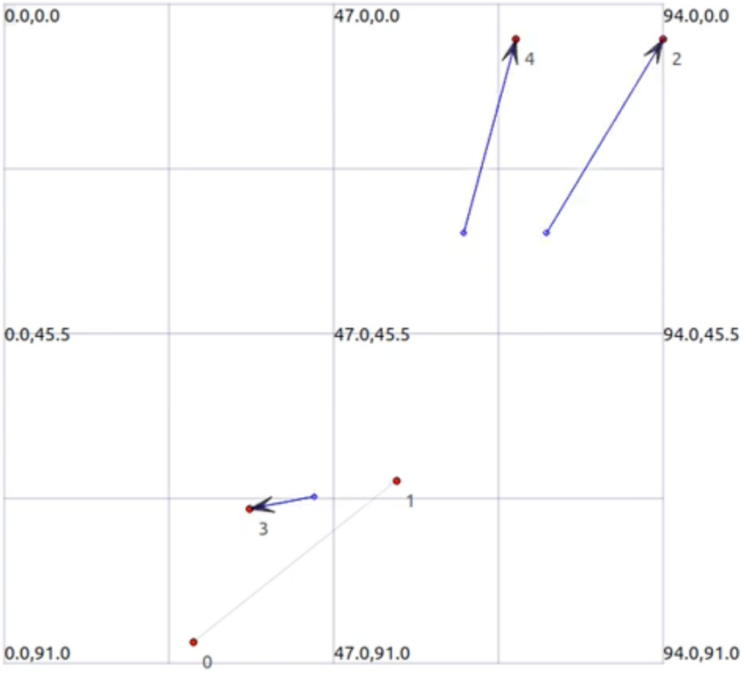
Step 12 : Click on **run** option/icon to see the animation. To see the packet transfer, open Packets Tab.

TW7 OUTPUT:

Packets



Animator



TERMWORK 8: Simulate a simple 5G Network application using NS3.

Step 1 : Open UBUNTU and locate and open **ns-allinone-3.28** folder on Desktop.

Step 2 : Go to **ns-3.28** folder and open **examples**->**tutorial**->**second.cc**

Step 3 : Copy second.cc and **paste** it in ns-3.28->**scratch** folder.

Remember that scratch folder should contain only one .cc example file and it must contain scratch executable file named scratchsimulator.cc and other files can be deleted.

Step 4 : In third.cc , include the following code. (Before – "Simulator::Run ();")

```
#include "ns3/netanim-module.h"
AnimationInterface anim("third, xml");
AsciiTraceHelper ascii;
pointToPoint.EnableAsciiAll(ascii.CreateFileStream("third.tr"));
pointToPoint.EnablePcapAll("third");
```

Step 5 : Right click->**Open in terminal** and change working directory to Desktop by cd Desktop and type following commands to go to location where scratch executable file is located i.e. scratch folder.

Step 6 : cd ns-allinone-3.28 -> **cd ns-3.28**

Step 7 : ./waf build

Step 8 : Run the third.cc by entering following command.

./waf --run scratch/third

Step 9 : Once build is successful,

cd ../ (return to ns-allinone-3.28 folder)

ls

cd netanim-3.108 (enter into netanim-3.108)

Step 10 : Now to see the animation,

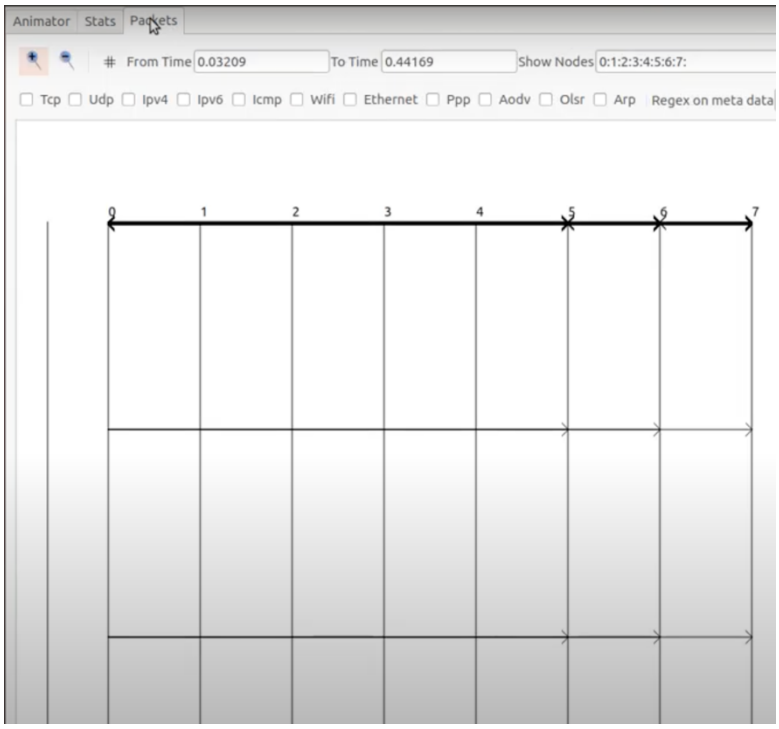
./NetAnim (we have to open NetAnim software)

Step 11 : In NetAnim, open **third , xml** by clicking on open XML trace file icon.

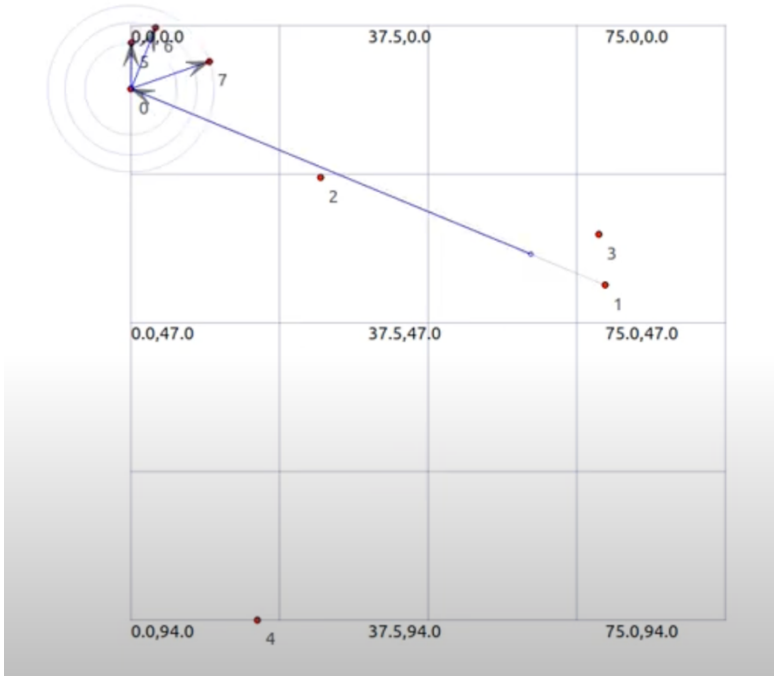
Step 12 : Click on **run** option/icon to see the animation. To see the packet transfer, open Packets Tab.

TW8 OUTPUT:

Packets



Animator



TERMWORK 9: Understanding the working of Ipv6 in Low power lossy network

Process Execution steps:

Go to the Location **contiki-ng/tools/cooja/** with commands

1. **Goto root** directory
2. **cd contiki-ng**
3. **cd tools**
4. **cd cooja**

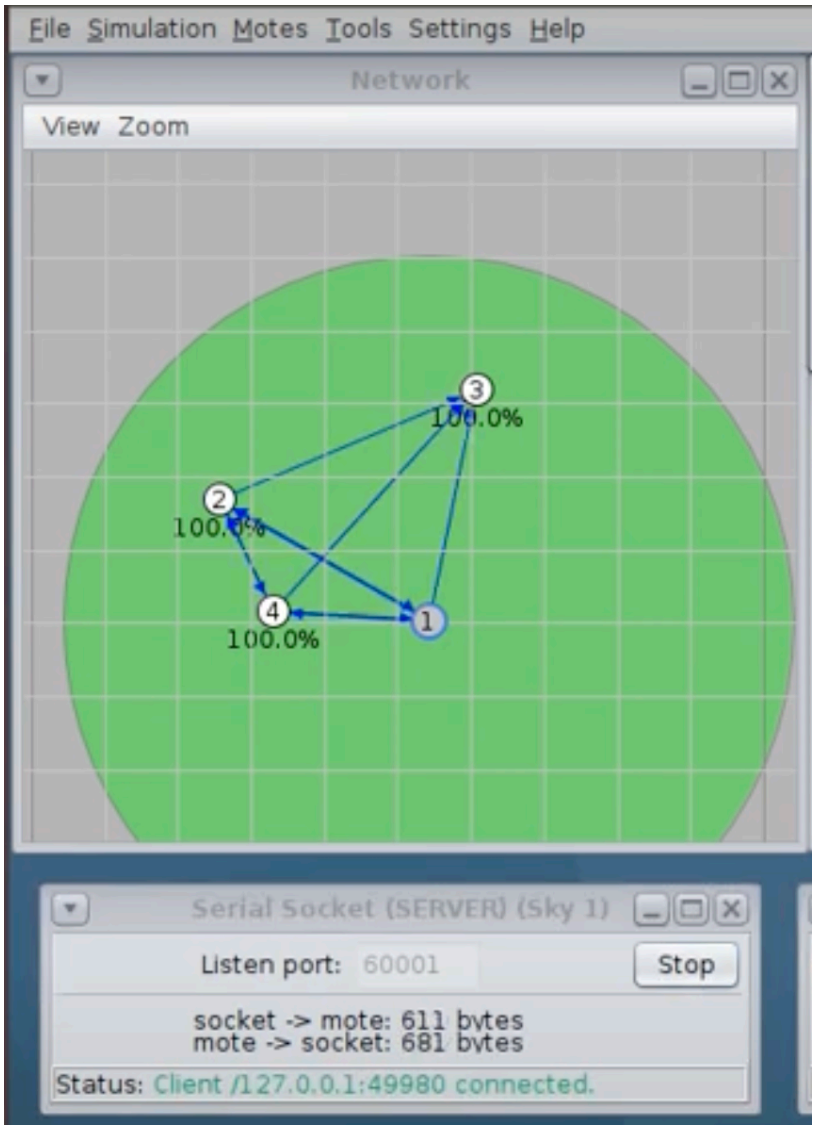
Run the cooja simulator with

5. **ant run**

Steps to create motes and configure them as server and client

1. Goto **File** -> **New Simulation**
2. Name the simulation and click on create
3. Click on **Motes** -> **Add motes** -> **Create a new mote type** -> **Sky mote**
4. Click on Browse and select **ipv6-hooks.c** (**/contiki-ng/examples/libs/ipv6-hooks**)
5. Click on **open** and then on **compile** and then on **create**
6. Enter the number of motes as **4** and click **on Add** motes
7. Place all motes close to each other such that the **coverage** is **100%** for each of them
8. Right click on **mote 1** and then click More tools for Sky 1 and then on Serial Socket **(SERVER)**. Mote 1 has been configured as Server.
9. Similarly, configure **motes 2, 3 and 4** as **clients**.
10. **Cop**y the **server's listening port** number and **paste** it as the port number for all **clients.**
11. **Start** the **server** and connect the client to the server.
12. Run the simulation by clicking on Simulation -> **Run Simulation**

TW9 OUTPUT:

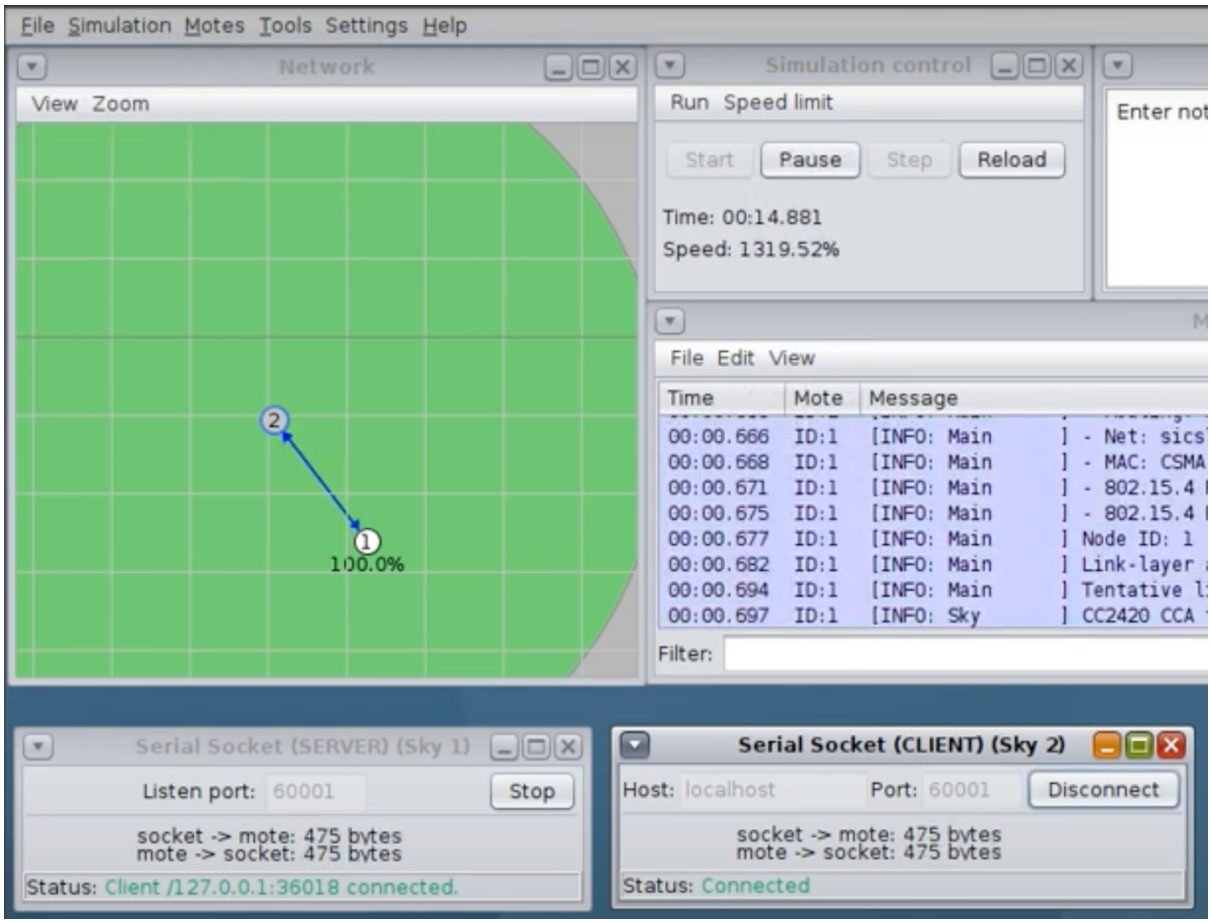

File Simulation Motes Tools Settings Help
Network
View Zoom
3
100.0%
2
100.0%
4
100.0%
1
Serial Socket (SERVER) (Sky 1)
Listen port: 60001
Stop
socket -> mote: 611 bytes
mote -> socket: 681 bytes
Status: Client /127.0.0.1:49980 connected.

TERMWORK 10: Understanding the working of IoT routing using RPL protocol

Process Execution steps:

1. Goto **File** -> **New Simulation**
2. Name the simulation and click on create
3. Click on **Motes** -> **Add motes** -> **Create a new mote type** -> **Sky mote**
4. Click on Browse and select **rpl-udp(/contiki-ng/examples/libs/rpl-udp**)
5. Create **udp-server.c** and add **1 mote** by clicking Motes -> Add new Mote -> Browse
6. Create **udp-client.c** and add **1 mote**
7. Place both the motes close to each other
8. Configure **1** as **server** and **2** as **client**
9. **Copy** the **server's listening port** number and **paste** it as the port number for all **clients.**
10. **Start** the **server** and connect the client to the server.
11. Run the simulation by clicking on Simulation -> **Run Simulation**

TW10 OUTPUT:



NP VIVA

1. What is socket?

It is endpoint of interface between transport layer process and application layer process.

2. Name the seven layers of the OSI model?

1- Application layer
2- Presentation layer
3- Session layer
4- Transport layer
5- Network layer
6- Data link layer
7- Physical layer

3. What is the difference between TCP and UDP?

Parameter-	TCP	UDP
Type of Service-	Connection oriented	Connectionless
Reliability-	More	less
Acknowledgement-	Yes	No
Delivery-	Inorder	not guaranteed
Retransmission-	yes for lost segments	no
Handshaking-	Yes	no
Stream type-	Bytes	message

4. What does socket consists of?

A socket has three parts: protocol, local-address, local-port.

5. What is firewall?

A Firewall is a network security device that monitors and filters incoming and outgoing network traffic.

6. How do I monitor the activity of sockets?

The socket is bind to specific port and IP address, helps for monitoring the activities over socket.

7. What is the role of TCP protocol and IP protocol?

Transmission Control Protocol/Internet Protocol is a suite of communication protocols used to interconnect network devices on the internet.

8. How should I choose a port number for my server?

Any port no can be chosen but the no must be above 1023 , as (0-1023) are system reserved port that can’t be used.

9. What is routing?

Routing is the process of selecting a path for traffic in a network or between or across multiple networks.

10. What is VPN?

A virtual private network, or VPN, is an encrypted connection over the Internet from a device to a network.

11. How do I open a socket?

The socket command may be used to open either the client or server side of a connection,

12. How do I create an input stream?

getInputStream() method is used to initialize the inputstream for the socket.

13. How do I close a socket?

Close() method is used to close the socket.

14. What is echo server?

An EchoServer is an application that allows a client and a server to connect so a client can send a message to the server and the server can receive the message and send, or echo, it back to the client.

15. What this function bind() does?

bind() assigns the address specified by addr to the socket referred to by the file descriptor sockfd. Addrlen.

16. What this function socket() does?

socket() creats the new socket and returns a socket descriptor for use in later system calls or -1 on error.

17. What is IP address?

A unique number assigned to machine or host for network connection.

18. What are network host names?

Hostname is an alias given to a computer on a TCP/IP network to identify it on the network. Hostnames are a friendlier way of identifying TCP/IP hosts than IP address.

19. How to find a machine address?

Open cmd and type command: ipconfig/all in windows, open terminal and type command: ifconfig in Linux.

20. What is MAC address?

It is a unique 48 bit address given the NIC vendor for creation of network.

21. What is multicasting?

One sender and few recievers- group communication.

22. What is DNS?

It makes translation of Domain name to equivalent IP address to lacate the host on the network.

23. How does TCP handshaking works?

It is three phase:
Phase1- connection request
Phase2- data transfer
Phase3- connection termination

24. What is Wireshark?

It is protocol and packet analyzer tool.

25. Which wireshark filter can be used to check all incoming requests to a HTTP web server?

TCP port 80

26. How to capture packets using Wireshark in a switched Ethernet network?

Choose the interface as etho or any relevant interface and click start it will capture the packets.

27. Explain the following features of NS3: i) Tracing ii) NetAnim

Tracing: ASCII and pcap tracing- network traffic analysis
NetAnim: animation interface for NS3

28. Explain about the following : a. Pcap file b. gnuplot

a. Pcap file: a file that contains captured packtes information from wireshark.
b. Gnuplot: tool used for ploting graph against network traffic captured.

29. How does IPv6 solve the problem of IPv4 exhaustion?

Larger address space over IPv4, no NAT, increased security, true sense of Type Of Service(TOS) etc features.

30. What is RPL protocol? Explain its significance in IoT.

The protocol suitable for routing of packets in Low Power lossy network applications like IoT. Since IoT needs a protocol with less utilization of battery power and more performance , then RPL becomes choice for usage in IoT deployment.