

## TW-1 (Message Queue)

### Writer

```
#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

```
#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;
}
```

## Output

### Writer

```
lab2@lab2-virtual-machine:~/Aniket_NP-Lab/TW-1$ gcc TW-1_writer.c
lab2@lab2-virtual-machine:~/Aniket_NP-Lab/TW-1$ ./a.out
Write data: Hello World
Data sent is : Hello World
```

### Reader

```
lab2@lab2-virtual-machine:~/Aniket_NP-Lab/TW-1$ gcc TW-1_reader.c
lab2@lab2-virtual-machine:~/Aniket_NP-Lab/TW-1$ ./a.out
Data read is: Hello World
```

## TW-1 (Pipe)

```
#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", getppid());
        printf("Child process pid: %d\n", p);
        printf("Passing value 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;
}
```

## Output

```
lab2@lab2-virtual-machine:~/Aniket_NP-Lab/TW-1/pipe$ gcc TW-1.c
lab2@lab2-virtual-machine:~/Aniket_NP-Lab/TW-1/pipe$ ./a.out
Parent process pid: 7437
Child process pid: 7618
Passing value child
lab2@lab2-virtual-machine:~/Aniket_NP-Lab/TW-1/pipe$ Child process
pid: 7618
Parent process pid: 1511
Data received by child process:
Hello World!
```

## TW-2

### Server

```
#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

```
#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;
}
```

# Output

## Server

```
lab2@lab2-virtual-machine:~/Aniket_NP-Lab/TW-2$ cc server.c
lab2@lab2-virtual-machine:~/Aniket_NP-Lab/TW-2$ ./a.out
[+]Server socket created successfully.
[+]Bind to port number 4444.
[+]Listening...
[+]Data sent to client: Hello.
[+]Closing the connection
```

## Client

```
lab2@lab2-virtual-machine:~/Aniket_NP-Lab/TW-2$ cc client.c
lab2@lab2-virtual-machine:~/Aniket_NP-Lab/TW-2$ ./a.out
[+]Client socket created successfully.
[+]Connected to server.
[+]Data received: Hello
[+]Closing the connection.
```

## TW-3

```
#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]);
    }
}

void dv_algo(int r) {
    int i, j, z;
    for (i = 1; i <= no; i++) {
        // r → source router
        // i → step taken (via which router to reach the dest router)
        // j → destination router
        // cannot jump from the source router or to a router which is not
        // reachable or from the source router
        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, j, 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 is 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 is as follows");
    for (i = 1; i <= no; i++)
        display(i);
}

```



```

        printf("\n");
        while (choice != 'n'){
            printf("\nEnter the nodes between which 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;
    }
}

```

## Output

```

aniket@aniket-Lenovo-IdeaPad-S540-15IML-D:~/NP-Lab/TW-3$ gcc dvalgo.c
aniket@aniket-Lenovo-IdeaPad-S540-15IML-D:~/NP-Lab/TW-3$ ./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 is 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 is 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 shortest distance is to be found: 1 5

The length of the shortest path between nodes 1 and 5 is 6

Continue? (y/n): y

Enter the nodes between which shortest distance is to be found: 1 4

The length of the shortest path between nodes 1 and 4 is 5

Continue? (y/n): n

# TW-4

## UDP Program execution steps

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 in Wireshark, goto edit in the menu bar and select find packet option or just press Ctrl+F.
4. In the 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.
5. The arrow pointing towards the packet is the request packet, and the arrow coming out from the packet is the response packet.
6. Click on any request or response DNS packet and examine the UDP packet.

The image shows the Wireshark network protocol analyzer interface. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Wireless, Tools, and Help. Below the menu is a toolbar with various icons. The main window is divided into three panes. The top pane shows a list of captured packets with columns for No., Time, Source, Destination, Protocol, Length, and Info. The middle pane shows the packet details for the selected packet (No. 686), including Ethernet II, Internet Protocol Version 4, and User Datagram Protocol. The bottom pane shows the packet bytes in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
683	5.076574613	192.168.109.143	142.251.42.110	QUIC	80	Protected Payload (KP0), DCID=dfedee50f6931f21
684	5.076996681	142.251.42.110	192.168.109.143	QUIC	231	Protected Payload (KP0), DCID=52d3ec
685	5.076974417	192.168.109.143	142.251.42.110	QUIC	75	Protected Payload (KP0), DCID=dfedee50f6931f21
686	5.116935205	127.0.0.1	127.0.0.53	DNS	88	Standard query 0xf1ba A mail.google.com OPT
687	5.116188969	192.168.109.143	192.168.109.2	DNS	77	Standard query 0xf747 A mail.google.com
688	5.117042492	192.168.109.2	192.168.109.143	DNS	83	Standard query response 0xf747 A mail.google.com A 142.250.67.197
689	5.117143130	127.0.0.53	127.0.0.1	DNS	104	Standard query response 0xf1ba A mail.google.com A 142.250.67.197
690	5.117202704	127.0.0.1	127.0.0.53	DNS	88	Standard query 0xf5c4 AAAA mail.google.com OPT
691	5.117285877	192.168.109.143	192.168.109.2	DNS	77	Standard query 0xe754 AAAA mail.google.com
692	5.117886282	192.168.109.2	192.168.109.143	DNS	105	Standard query response 0xe754 AAAA mail.google.com AAAA 2404:6800:
693	5.117977402	127.0.0.53	127.0.0.1	DNS	116	Standard query response 0xf5c4 AAAA mail.google.com AAAA 2404:6800:
694	5.127552963	192.168.109.143	142.250.67.197	TCP	76	41246 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=
695	5.130251529	142.251.42.110	192.168.109.143	QUIC	72	Protected Payload (KP0), DCID=52d3ec
696	5.153877716	192.168.109.143	142.251.42.36	QUIC	1401	Protected Payload (KP0), DCID=c754bcc684e6f6bd
697	5.154067401	192.168.109.143	142.251.42.36	QUIC	206	Protected Payload (KP0), DCID=c754bcc684e6f6bd
698	5.155308699	142.250.67.197	192.168.109.143	TCP	62	443 → 41246 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
699	5.155328429	192.168.109.143	142.250.67.197	TCP	56	41246 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
700	5.157074210	192.168.109.143	142.250.67.197	TLSv1.3	573	Client Hello
701	5.157269454	142.250.67.197	192.168.109.143	TCP	62	443 → 41246 [ACK] Seq=1 Ack=518 Win=64240 Len=0
702	5.181685839	142.251.42.36	192.168.109.143	QUIC	75	Protected Payload (KP0), DCID=2b830f
703	5.202909274	192.168.109.143	142.251.42.36	QUIC	78	Protected Payload (KP0), DCID=c754bcc684e6f6bd
704	5.252829974	142.251.42.36	192.168.109.143	QUIC	83	Protected Payload (KP0), DCID=2b830f

Frame 686: 88 bytes on wire (704 bits), 88 bytes captured (704 bits) on interface any, id 0  
Interface id: 0 (any)  
Encapsulation type: Linux cooked-mode capture v1 (25)  
Arrival Time: Dec 19, 2022 09:23:14.833884997 IST  
[Time shift for this packet: 0.000000000 seconds]  
Epoch Time: 1679903594.833884997 seconds  
[Time delta from previous captured frame: 0.037060788 seconds]  
[Time delta from previous displayed frame: 0.037060788 seconds]  
[Time since reference or first frame: 5.116935205 seconds]  
Frame Number: 686  
Frame Length: 88 bytes (704 bits)  
Capture Length: 88 bytes (704 bits)  
[Frame is marked: False]  
[Frame is ignored: False]

Time	127.0.0.1	127.0.0.53	192.168.109.143	192.168.109.2	Comment
0.000000000					
0.000164837	39118	Standard query 0x3e2bA incoming.tel...	53		DNS: Standard query 0x3e2bA incoming.tel...
0.000300711	39118	Standard query 0x683a AAAA incommi...	53		DNS: Standard query 0x683a AAAA incoming tel...
0.000374082				51446	DNS: Standard query 0x1d8b AAAA incoming tel...
0.001030767	41686	Standard query 0xa06c A incoming.tel...	53		DNS: Standard query 0xa06c A incoming tel...
0.001279186				51446	DNS: Standard query response 0x1d8b AAAA incommi...
0.001468264				49266	DNS: Standard query 0xf158 AAAA prod.in...
0.002109315				49266	DNS: Standard query response 0xf158 AAAA prod.in...
0.002211465	39118	Standard query response 0x683a AA...	53		DNS: Standard query response 0x683a AAAA incommi...
0.027654601				40411	DNS: Standard query response 0xf884 A incoming.tel...
0.027906955	39118	Standard query response 0x3e2b A in...	53		DNS: Standard query response 0x3e2b A incoming.tel...
0.028008365	41686	Standard query response 0xa06c A in...	53		DNS: Standard query response 0xa06c A incoming.tel...
0.028523107				60558	TCP: 60558 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=
0.051705744	36631	Standard query 0x42fc A incoming.tel...	53		DNS: Standard query 0x42fc A incoming tel...
0.051905746	36631	Standard query response 0x42fc A in...	53		DNS: Standard query response 0x42fc A incoming.tel...
0.051925769	36631	Standard query 0xa806 AAAA incommi...	53		DNS: Standard query 0xa806 AAAA incoming tel...
0.052081193				56755	DNS: Standard query 0xfae0 AAAA prod.in...
0.052889504				56755	DNS: Standard query response 0xfae0 AAAA prod.in...
0.052970376	36631	Standard query response 0xa806 AA...	53		DNS: Standard query response 0xa806 AAAA incommi...
0.056745078				60558	TCP: 443 → 60558 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=
0.056769929				60558	TCP: 60558 → 443 [ACK] Seq=1 Ack=1 Win=64240 Len=0
0.057263531				60558	TLSv1.2: Client Hello
0.057439130				60558	TCP: 443 → 60558 [ACK] Seq=1 Ack=518 Win=64240 Len=0
0.194051945				60558	TLSv1.2: Server Hello, Change Cipher Spec, Encrypted Handshake Message
0.194083373				60558	TCP: 60558 → 443 [ACK] Seq=518 Ack=157 Win=64084 Len=0
0.194832376				60558	TLSv1.2: Change Cipher Spec, Encrypted Handshake Message

# TW-5

## TCP Program execution steps

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.flags.syn == 1 && tcp.flags.ack == 1

No.	Time	Source	Destination	Protocol	Length	Info
24	4.279670381	34.120.208.123	192.168.109.143	TCP	62	443 → 36828 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
59	4.499669718	34.120.115.102	192.168.109.143	TCP	62	443 → 32870 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
72	4.547974158	34.120.237.76	192.168.109.143	TCP	62	443 → 50380 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
356	13.248804942	104.16.92.83	192.168.109.143	TCP	62	80 → 43684 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
358	13.248864883	104.16.92.83	192.168.109.143	TCP	62	80 → 43690 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
375	13.646024395	104.16.93.83	192.168.109.143	TCP	62	443 → 56012 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
491	14.213531518	142.250.182.202	192.168.109.143	TCP	62	443 → 38698 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
495	14.231104966	104.16.92.83	192.168.109.143	TCP	62	443 → 40954 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
497	14.231227483	104.16.92.83	192.168.109.143	TCP	62	443 → 40938 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
499	14.231262608	104.16.92.83	192.168.109.143	TCP	62	443 → 40984 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
501	14.231281961	104.16.92.83	192.168.109.143	TCP	62	443 → 40968 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
503	14.231300631	104.16.92.83	192.168.109.143	TCP	62	443 → 40988 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
689	14.447544998	104.16.93.83	192.168.109.143	TCP	62	443 → 56000 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

Frame 24: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface any, id 0

Linux cooked capture v1

Internet Protocol Version 4, Src: 34.120.208.123, Dst: 192.168.109.143

Transmission Control Protocol, Src Port: 443, Dst Port: 36828, Seq: 0, Ack: 1, Len: 0

Source Port: 443  
Destination Port: 36828  
[Stream index: 1]  
[TCP Segment Len: 0]  
Sequence Number: 0 (relative sequence number)  
Sequence Number (raw): 2046311603  
[Next Sequence Number: 1 (relative sequence number)]  
Ack=1  
Ackedgment Number: 1 (relative ack number)  
Ackedgment number (raw): 148274904  
0110 .... = Header Length: 24 bytes (6)  
Flags: 0x012 (SYN, ACK)  
Window: 64240

0000 00 00 00 01 00 06 00 50 56 f9 6c 60 00 00 08 00 .....P V..  
0010 45 00 00 2c e3 27 00 00 00 06 30 79 22 78 d0 7b E... ..6y"x{  
0020 c0 a8 6d 8f 01 bb 8f dc 79 f8 3c b3 08 d6 7e d8 ..m.....y<.....  
0030 60 12 fa f0 ac 08 00 00 02 04 05 b4 00 00 00 .....

Time 34.120.208.123 192.168.109.143 34.120.115.102 34.120.237.76 Comment

4.279670381	443	443 → 36828 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	36828		TCP: 443 → 36828 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
4.499669718		443 → 32870 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	32870	443	TCP: 443 → 32870 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
4.547974158		443 → 50380 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	50380	443	TCP: 443 → 50380 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
13.248804942		80 → 43684 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	43684		TCP: 80 → 43684 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
13.248864883		80 → 43690 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	43690		TCP: 80 → 43690 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
13.646024395		443 → 56012 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	56012		TCP: 443 → 56012 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14.213531518		443 → 38698 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	38698		TCP: 443 → 38698 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14.231104966		443 → 40954 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	40954		TCP: 443 → 40954 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14.231227483		443 → 40938 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	40938		TCP: 443 → 40938 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14.231262608		443 → 40984 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	40984		TCP: 443 → 40984 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14.231281961		443 → 40968 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	40968		TCP: 443 → 40968 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14.231300631		443 → 40988 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	40988		TCP: 443 → 40988 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14.447544998		443 → 56000 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	56000		TCP: 443 → 56000 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14.470054032		443 → 38700 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	38700		TCP: 443 → 38700 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14.470860956		443 → 38704 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	38704		TCP: 443 → 38704 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14.474607081		443 → 38714 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	38714		TCP: 443 → 38714 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14.744700144		443 → 54166 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	54166		TCP: 443 → 54166 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14.747704968		443 → 54172 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	54172		TCP: 443 → 54172 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14.757058208		443 → 54182 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	54182		TCP: 443 → 54182 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14.763894826		443 → 54198 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	54198		TCP: 443 → 54198 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14.767094559		443 → 54202 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	54202		TCP: 443 → 54202 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14.771794421		443 → 54216 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	54216		TCP: 443 → 54216 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
15.522290903		443 → 34764 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460	34764		TCP: 443 → 34764 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

Packet 1547: TCP: 443 → 54172 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

Limit to display filter

Flow type: All Flows

Addresses: Any

Reset Diagram Close Save As...

## TW-6

Load for

First.cc → TW6

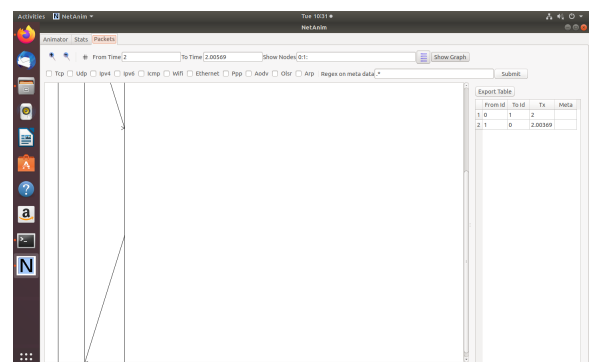
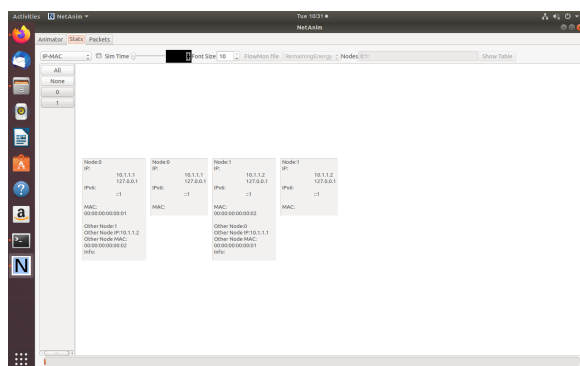
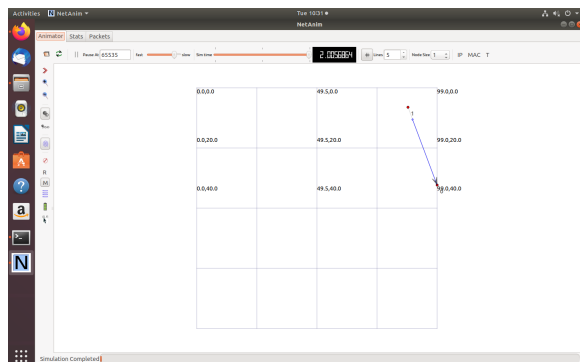
Second.cc → TW7

Third.cc → TW8

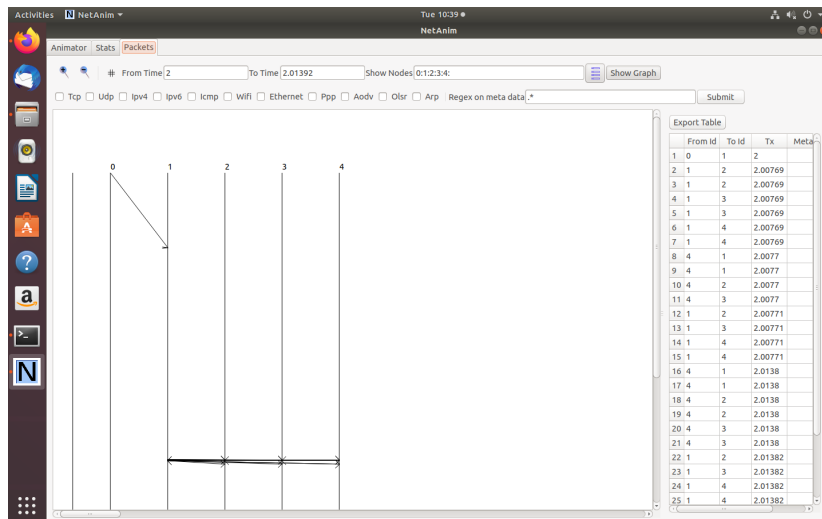
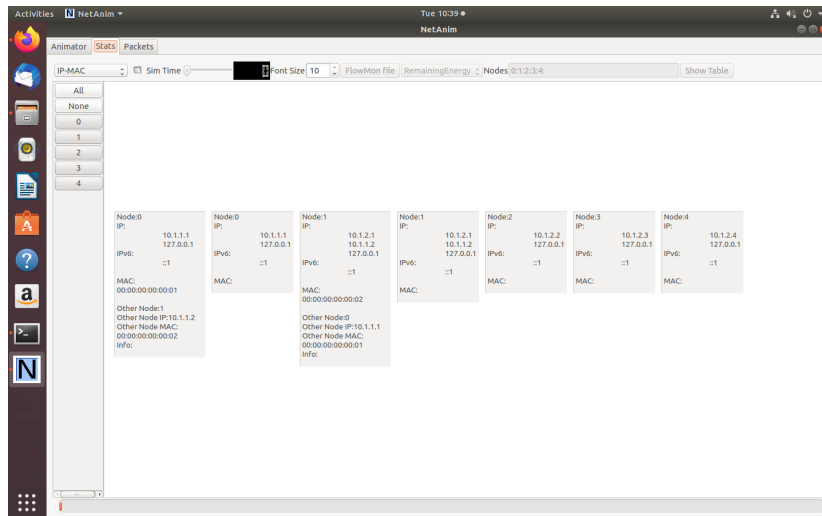
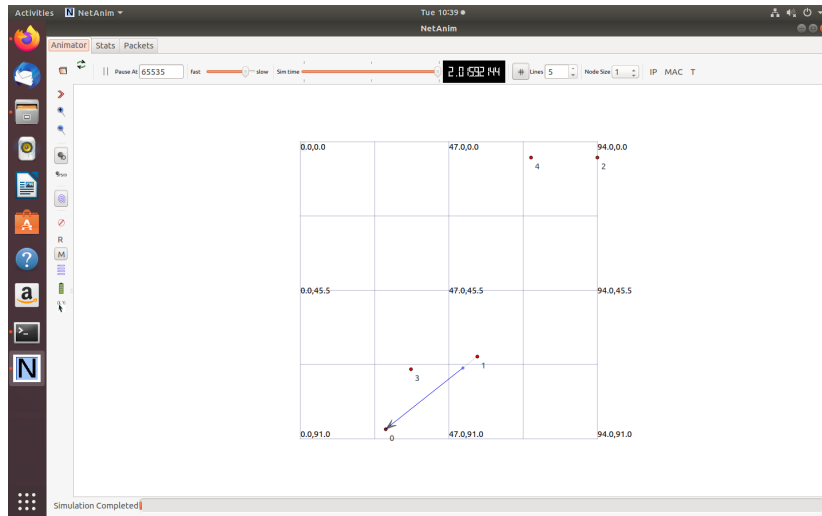
\*\*\*IMPORTANT - Load **NS3-WORKING** virtual machine\*\*\*

- 1) Copy the required TW file from examples/tutorial folder to scratch folder
- 2) Add the below four lines at the end of the program before "Simulator::Run()"  

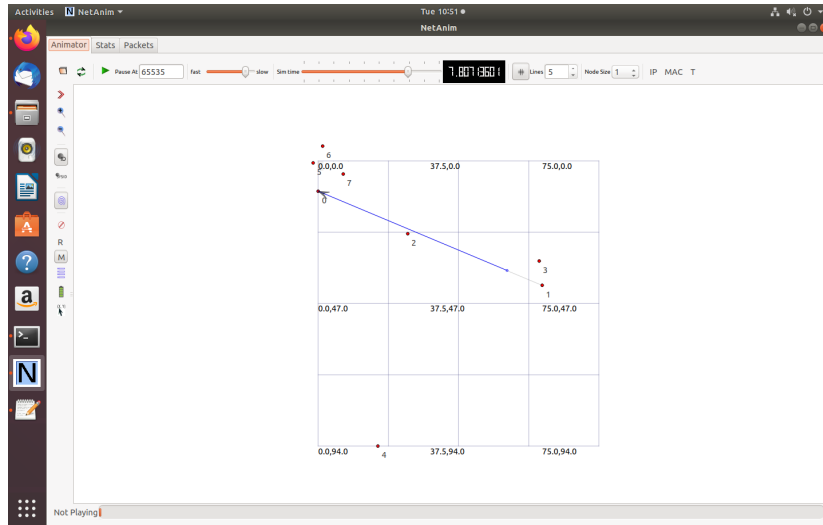
```
#include "ns3/netanim-module.h"
AnimationInterface anim("first, xml");
AsciiTraceHelper ascii;
pointToPoint.EnableAsciiAll(ascii.CreateFileStream("first.tr"))
;
pointToPoint.EnablePcapAll("first");
```
- 3) Open terminal in **ns-3.28** and run command **./waf build**
- 4) Run command **./waf --run scratch/[first/second/third]**
- 5) **cd .. (to move back a folder)**
- 6) Change directory to **netanim-3.1** and run the command **./NetAnim** to run the network animator
- 7) Load the XML file **[first.xml / second.xml / third.xml ]** and run it



# TW-7

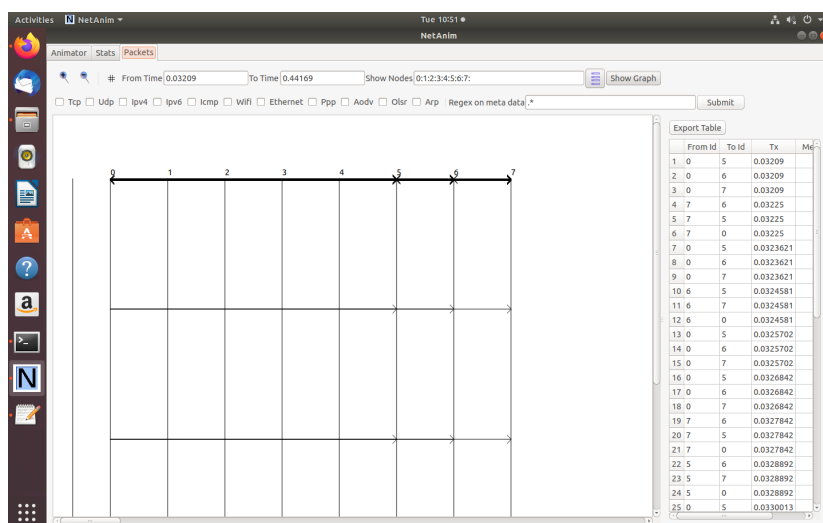


# TW-8



The NetAnim Stats window displays details for eight nodes. The 'IP-MAC' dropdown is set to 'IP-MAC'. The 'Font Size' is 10. The 'FlowMon file' is 'RemainingEnergy'. The 'Nodes' dropdown is set to '0:1:2:3:4:5:6:7'. The 'Show Table' button is visible.

Node	IP	IPv6	MAC	Other Node
Node0	10.1.1.1	10.1.3.4	127.0.0.1	Other Node1
Node1	10.1.1.1	10.1.3.4	127.0.0.1	Other Node0
Node2	10.1.2.1	10.1.2.2	127.0.0.1	Other Node0
Node3	10.1.2.3	10.1.2.4	127.0.0.1	Other Node0
Node4	10.1.2.4	10.1.2.5	127.0.0.1	Other Node0
Node5	10.1.3.1	10.1.3.2	127.0.0.1	Other Node0
Node6	10.1.3.2	10.1.3.3	127.0.0.1	Other Node0
Node7	10.1.3.3	10.1.3.4	127.0.0.1	Other Node0





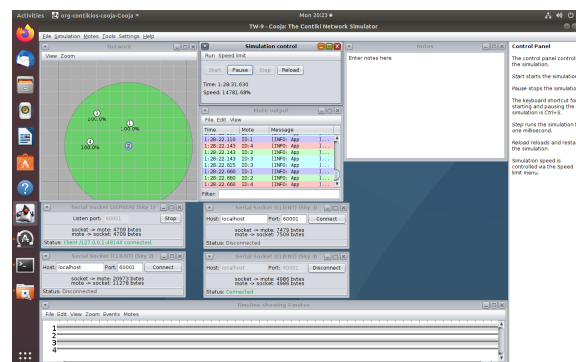
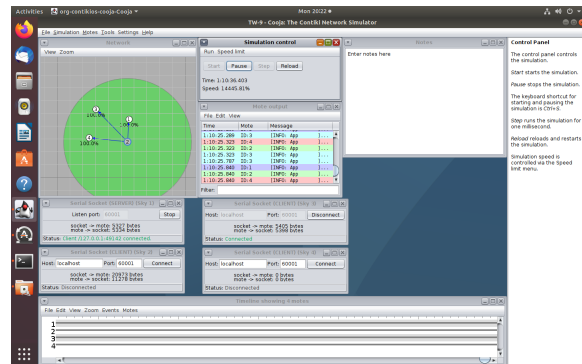
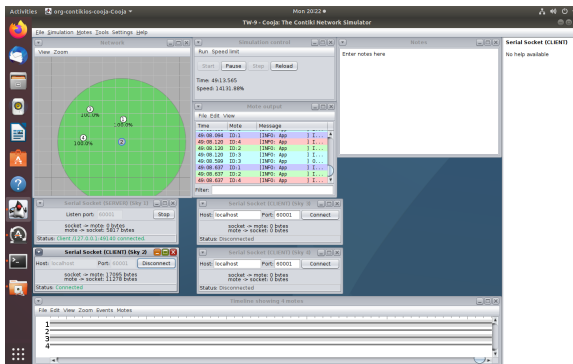
# TW-9

## Steps to open the cooja simulator

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



## TW-10

Follow the exact same steps to create 2 motes (client and server).

1. Click on Browse and select rpl-udp(/contiki-ng/examples/libs/rpl-udp)
2. Create udp-server.c and add 1 mote by clicking Motes -> Add new Mote -> Browse
3. Create udp-client.c and add 1 mote
4. Place both the motes close to each other
5. Configure 1 as server and 2 as client
6. Copy the server's port number to the client.
7. Start the server and connect the client.
8. Run the simulation.

