8 Program to simulate UDP Client Server.

What is UDP ?

UDP stands for User Datagram Protocol. It's one of the core protocols of the Internet Protocol (IP) suite, often used for sending short messages called datagrams between devices on a network.

Key Features of UDP:

- Connectionless: Unlike TCP, UDP doesn't establish a connection before sending data. It just sends it.
- Fast but unreliable: There's no guarantee of delivery, order, or error checking. If data gets lost or arrives out of order, UDP doesn't correct it.
- Low overhead: Because it skips checks and handshakes, it's faster and more lightweight than TCP.

Common Uses of UDP:

- Streaming (video/audio): Like Zoom, Skype, or YouTube live, where speed is more important than perfect delivery.
- Online gaming: Fast response is critical, and occasional data loss is acceptable.
- DNS (Domain Name System): When you type a website address, your computer sends a quick UDP query to get the IP.

```
#include <fstream>

#include "ns3/core-module.h"

#include "ns3/csma-module.h"

#include "ns3/applications-

module.h" #include "ns3/internet-

module.h"


using namespace ns3;


NS_LOG_COMPONENT_DEFINE ("UdpClientServerExample");


int

main (int argc, char *argv[])

{
```

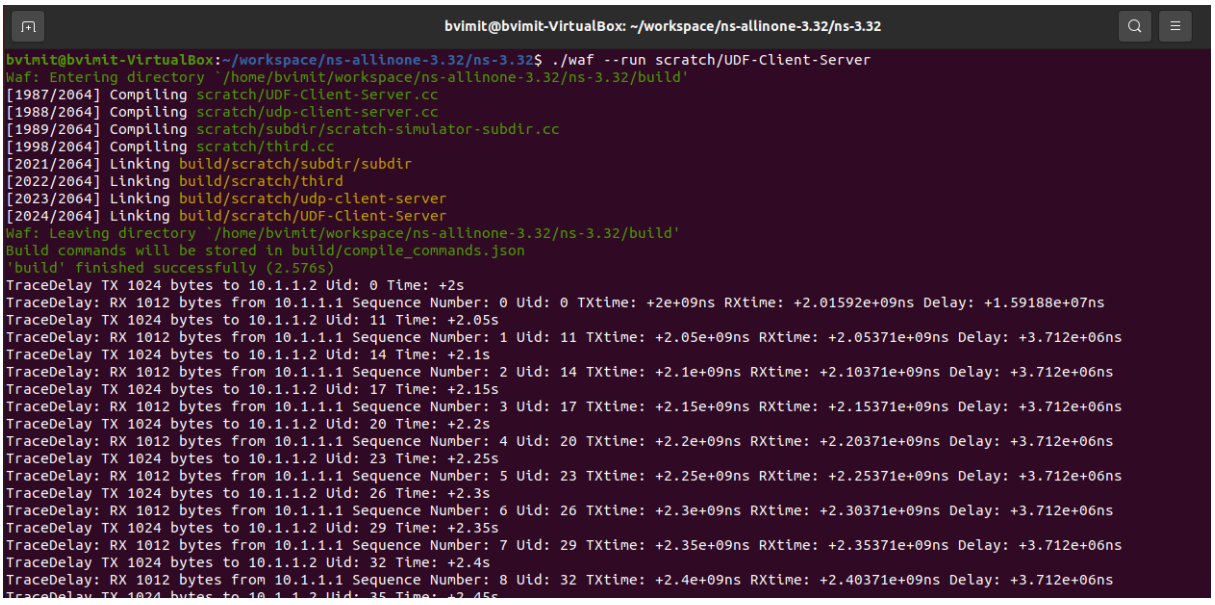
```
LogComponentEnable ("UdpClient",
LOG_LEVEL_INFO); LogComponentEnable
("UdpServer", LOG_LEVEL_INFO);

bool useV6 = false;
bool tracing = true;
Address
serverAddress;

CommandLine cmd (__FILE__);
cmd.AddValue ("useIpv6", "Use Ipv6", useV6);
cmd.AddValue("tracing","Enable pcap tracing",
tracing);

cmd.Parse (argc, argv);


NS_LOG_INFO ("Create
nodes."); NodeContainer n;
n.Create (2);

InternetStackHelper internet;
internet.Install (n);

NS_LOG_INFO ("Create channels.");

CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", DataRateValue (DataRate
(5000000))); csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds
(2))); csma.SetDeviceAttribute ("Mtu", UintegerValue (1400));
NetDeviceContainer d = csma.Install (n);
```

```
NS_LOG_INFO ("Assign IP
Addresses."); if (useV6 == false)
 {
   Ipv4AddressHelper ipv4;
   ipv4.SetBase ("10.1.1.0", "255.255.255.0");
   Ipv4InterfaceContainer i = ipv4.Assign (d);
   serverAddress = Address (i.GetAddress (1));
 }
else
 {
   Ipv6AddressHelper ipv6;
   ipv6.SetBase ("2001:0000:f00d:cafe::", Ipv6Prefix
   (64)); Ipv6InterfaceContainer i6 = ipv6.Assign (d);
   serverAddress = Address(i6.GetAddress (1,1));
 }

NS_LOG_INFO ("Create Applications.");

uint16_t port = 4000;
UdpServerHelper server (port);
ApplicationContainer apps = server.Install (n.Get
(1)); apps.Start (Seconds (1.0));
apps.Stop (Seconds (10.0));



uint32_t MaxPacketSize = 1024;
Time interPacketInterval = Seconds (0.05);
uint32_t maxPacketCount = 320;
UdpClientHelper client (serverAddress, port);
client.SetAttribute ("MaxPackets", UintegerValue (maxPacketCount));
client.SetAttribute ("Interval", TimeValue (interPacketInterval));
```

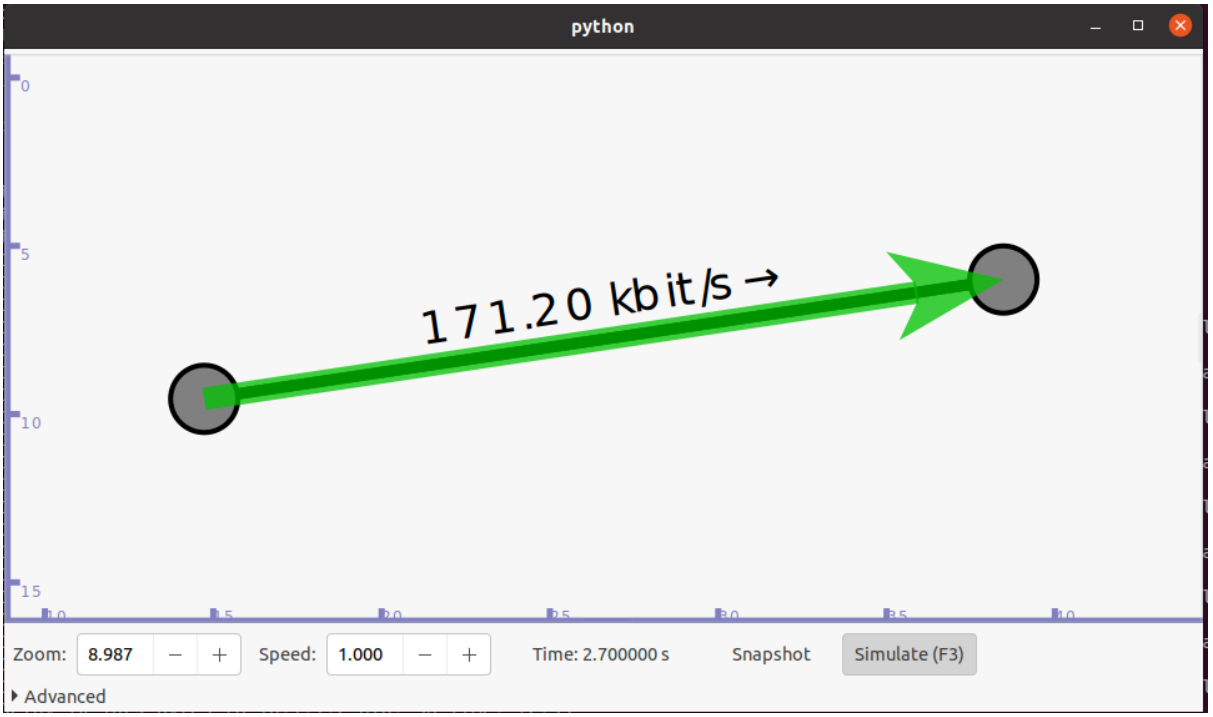
```
client.SetAttribute ("PacketSize", UintegerValue
(MaxPacketSize)); apps = client.Install (n.Get (0));
apps.Start (Seconds (2.0));
apps.Stop (Seconds (10.0));


NS_LOG_INFO ("Run
Simulation."); Simulator::Run
(); Simulator::Destroy ();
NS_LOG_INFO ("Done.");
}
```

Output:-

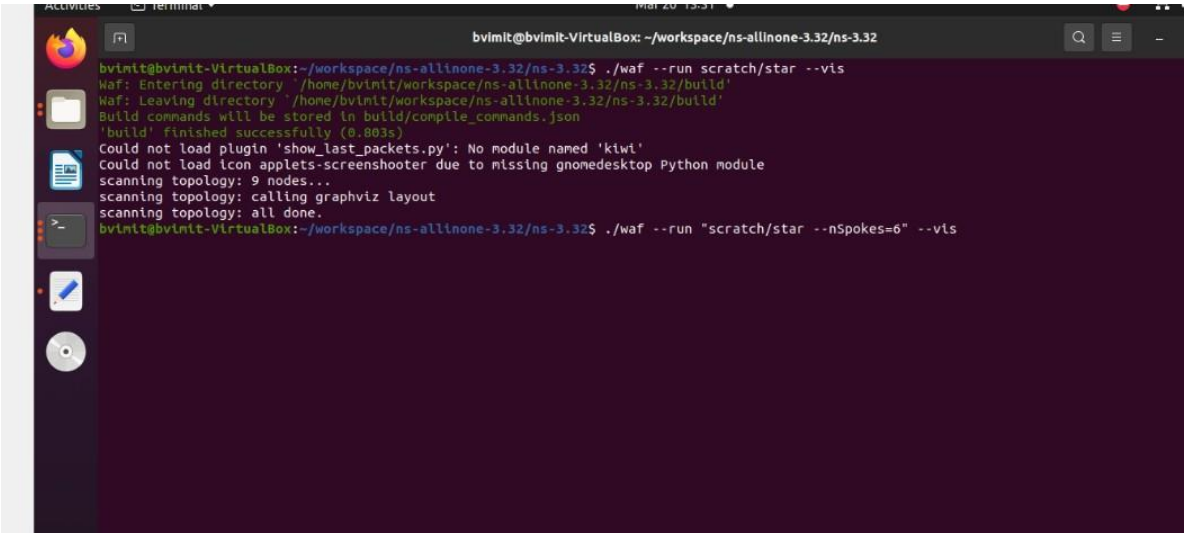
./waf --run scratch/UDF-Client-Server



./waf --run scratch/UDF-Client-Server --vis



Command line input



9 Program to simulate DHCP server and Clients.

What is DHCP?

DHCP stands for **Dynamic Host Configuration Protocol**. It's a network management protocol used to **automatically assign IP addresses** and other network configuration details to devices (clients) on a network.

DHCP Server

The **DHCP Server** is the system or device (like a router or dedicated server) that:

- Manages a pool of IP addresses.
- Assigns an IP address to each device that connects.
- Provides additional info like:
  - Subnet mask
  - Default gateway
  - DNS server

Think of it like a hotel front desk giving out room numbers and Wi-Fi details to guests when they check in.

? DHCP Clients

The **DHCP Clients** are devices like:

- Laptops
- Phones
- Printers
- Smart TVs
- Any network-enabled device

```
/*
 * Network layout:
 *
 * R0 is a DHCP server. The DHCP server announced R1 as the default router.
 * Nodes N1 will send UDP Echo packets to node A.
 *
 *
 *          ┌-----------------------------------------------------┐
 *          | DHCP Clients                        |
 *          |                    172.30.0.14   |
 *          |                    DHCP static   |
 *          | ┌──────────┐   ┌──────────┐   ┌──────────┐   |
 *          | | N0 |   | N1 |   | N2 |   |       ┌──────────┐
 *          | └──────────┘   └──────────┘   └──────────┘   |   ┌──────| A |
 *          |   |          |          |     |   └──────────┘
 *          └ -------|--------------|---------------|----------┘   | 172.30.1.2
 *  DHCP Server       |          |          |           |
 *     ┌──────────┐   |          |          |   ┌──────────┐   |
 *     | R0 |──────────┘ ──────────────────┘ ──────────────────────┘ ──────| R1
|──────┘
 *     └──────────┘                          └──────────┘ 172.30.1.1
 *       172.30.0.12                 172.30.0.17
 *
 * Things to notice:
 * 1) The routes in A are manually set to have R1 as the default router,
 *    just because using a dynamic outing in this example is an overkill.
```

```
 * 2) R1's address is set statically though the DHCP server helper interface.
 *    This is useful to prevent address conflicts with the dynamic pool.
 *    Not necessary if the DHCP pool is not conflicting with static addresses.
 * 3) N2 has a dynamically-assigned, static address (i.e., a fixed address assigned via DHCP).
 *
 */

#include "ns3/core-module.h"
#include "ns3/internet-apps-
module.h" #include "ns3/csma-
module.h"  #include "ns3/internet-
module.h" #include "ns3/point-to-
point-module.h" #include
"ns3/applications-module.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("DhcpExample");

int
main (int argc, char *argv[])
{
  CommandLine cmd (__FILE__);

  bool verbose = true;
  bool tracing = true;
```

```
cmd.AddValue ("verbose", "turn on the logs", verbose);

cmd.AddValue ("tracing", "turn on the tracing", tracing);


cmd.Parse (argc, argv);


// GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));


if (verbose)
 {
   LogComponentEnable ("DhcpServer", LOG_LEVEL_ALL);
   LogComponentEnable ("DhcpClient", LOG_LEVEL_ALL);
   LogComponentEnable ("UdpEchoServerApplication",
   LOG_LEVEL_INFO); LogComponentEnable
   ("UdpEchoClientApplication", LOG_LEVEL_INFO);
 }


Time stopTime = Seconds (20);


NS_LOG_INFO ("Create
nodes."); NodeContainer
nodes; NodeContainer router;
nodes.Create (3);
router.Create (2);


NodeContainer net (nodes, router);
```

```
NS_LOG_INFO ("Create
channels."); CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue
("5Mbps")); csma.SetChannelAttribute ("Delay", StringValue
("2ms")); csma.SetDeviceAttribute ("Mtu", UintegerValue
(1500)); NetDeviceContainer devNet = csma.Install (net);

NodeContainer p2pNodes;
p2pNodes.Add (net.Get (4));  //R1 router. Adding R1 to point to point network
p2pNodes.Create (1);  // Creating node A

PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

//netdevice for Point to point
NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);

InternetStackHelper tcpip;
tcpip.Install (nodes);
tcpip.Install (router);
tcpip.Install (p2pNodes.Get (1));  //A

Ipv4AddressHelper address;
```

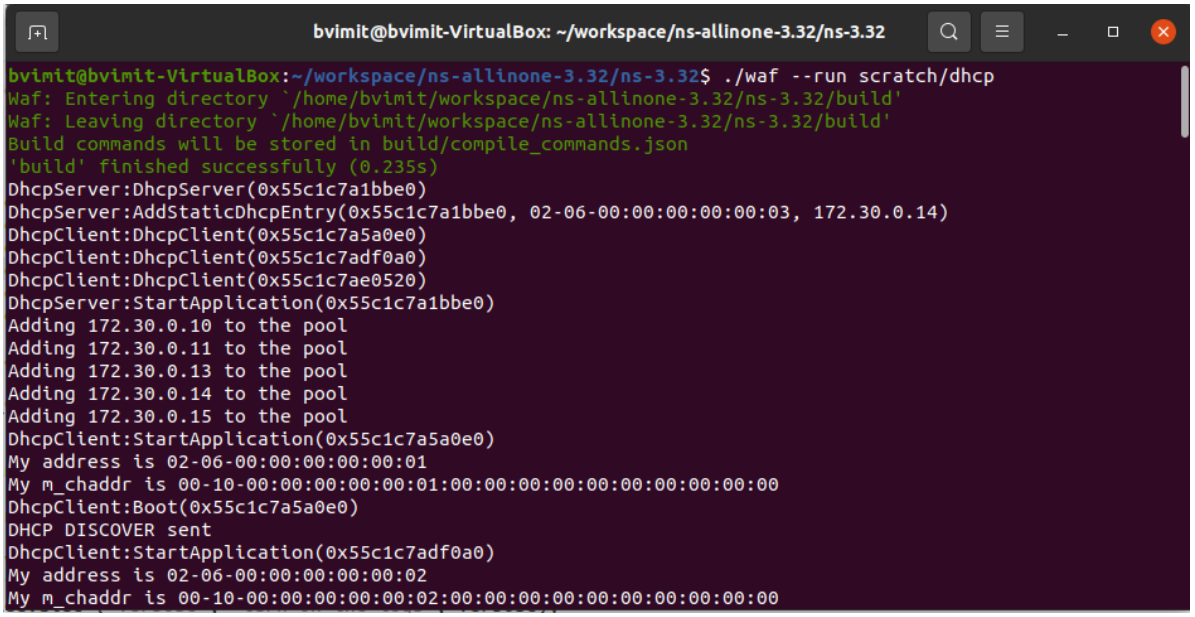
```
address.SetBase ("172.30.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign
(p2pDevices);

// manually add a routing entry because we don't want to add a dynamic routing
Ipv4StaticRoutingHelper ipv4RoutingHelper;
Ptr<Ipv4> ipv4Ptr = p2pNodes.Get (1)->GetObject<Ipv4> ();  //A
Ptr<Ipv4StaticRouting> staticRoutingA = ipv4RoutingHelper.GetStaticRouting
(ipv4Ptr); staticRoutingA->AddNetworkRouteTo (Ipv4Address ("172.30.0.0"),
Ipv4Mask ("/24"),
                    Ipv4Address ("172.30.1.1"), 1);

NS_LOG_INFO ("Setup the IP addresses and create DHCP
applications."); DhcpHelper dhcpHelper;
// The router must have a fixed IP.
/*InstallFixedAddress   (          Ptr< NetDevice >
                    netDevice, Ipv4Address          addr,
          Ipv4Mask        mask
 )        */
//Assign IP address of R1
Ipv4InterfaceContainer fixedNodes = dhcpHelper.InstallFixedAddress (devNet.Get (4), Ipv4Address
("172.30.0.17"), Ipv4Mask ("/24"));
// Not really necessary, IP forwarding is enabled by default in IPv4.
fixedNodes.Get (0).first->SetAttribute ("IpForward", BooleanValue
(true));
```

```
  // Configuring DHCP server R0
  /*DhcpHelper::InstallDhcpServer       (          Ptr< NetDevice >
                          netDevice, Ipv4Address
                          serverAddr, //IP address of R0
Ipv4Address
                          poolAddr
, Ipv4Mask
                          poolMas
k, Ipv4Address            minAddr,
Ipv4Address               maxAddr,
            Ipv4Address   gateway = Ipv4Address ()// R1
   )        */
  ApplicationContainer dhcpServerApp = dhcpHelper.InstallDhcpServer (devNet.Get (3), Ipv4Address
("172.30.0.12"),
                                     Ipv4Address ("172.30.0.0"), Ipv4Mask ("/24"),
                                     Ipv4Address ("172.30.0.10"), Ipv4Address
                                     ("172.30.0.15"),
                                     Ipv4Address ("172.30.0.17"));
  // This is just to show how it can be done.
  //DHCP static IP for N2
  DynamicCast<DhcpServer> (dhcpServerApp.Get (0))->AddStaticDhcpEntry (devNet.Get (2)-
>GetAddress (), Ipv4Address ("172.30.0.14"));
  dhcpServerApp.Start (Seconds (0.0));
  dhcpServerApp.Stop (stopTime);
 // Configuring DHCP clients
    // Netdevice for DHCP clients
  NetDeviceContainer dhcpClientNetDevs;
```

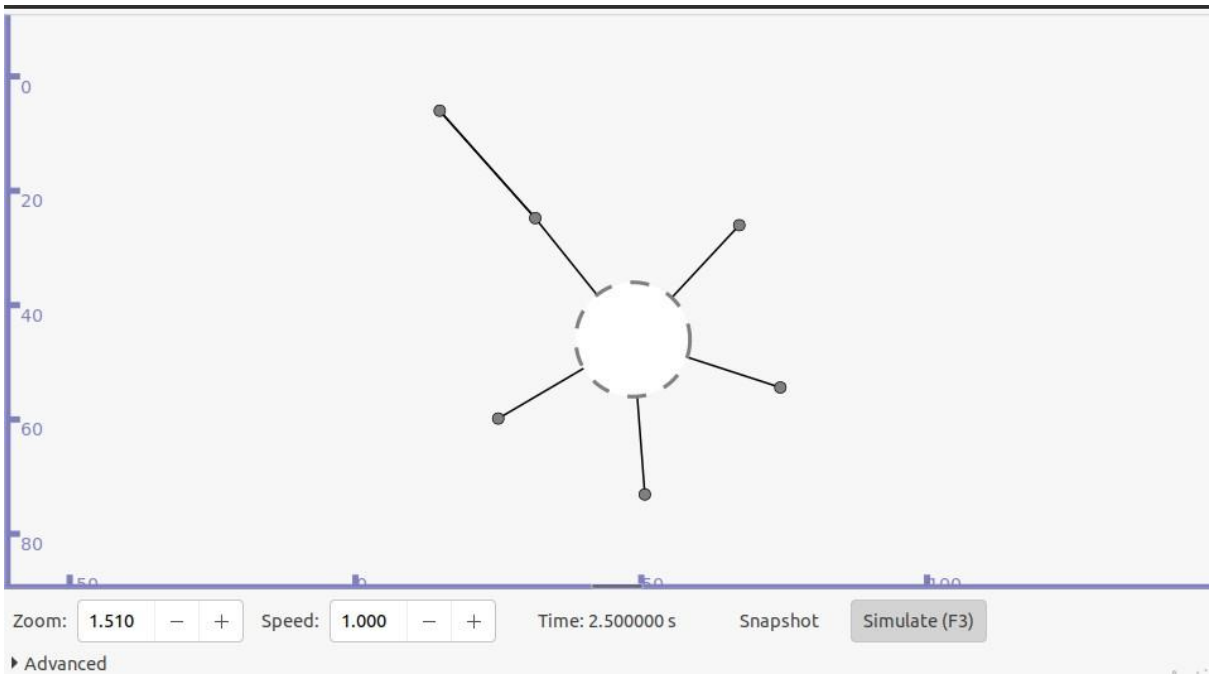
```
dhcpClientNetDevs.Add (devNet.Get (0));  //
N0    dhcpClientNetDevs.Add    (devNet.Get
(1));      //      N2      dhcpClientNetDevs.Add
(devNet.Get (2));   //N3


// installing DHCP client Application on N0, N1 and N2
ApplicationContainer dhcpClients = dhcpHelper.InstallDhcpClient (dhcpClientNetDevs);


dhcpClients.Start (Seconds
(1.0)); dhcpClients.Stop
(stopTime);


UdpEchoServerHelper echoServer (9);


ApplicationContainer serverApps = echoServer.Install
(p2pNodes.Get (1)); serverApps.Start (Seconds (0.0));
serverApps.Stop (stopTime);

UdpEchoClientHelper echoClient (p2pInterfaces.GetAddress
(1), 9); echoClient.SetAttribute ("MaxPackets", UintegerValue
(100)); echoClient.SetAttribute ("Interval", TimeValue
(Seconds (1.0))); echoClient.SetAttribute ("PacketSize",
UintegerValue (1024));
ApplicationContainer clientApps = echoClient.Install (nodes.Get (1));
clientApps.Start (Seconds (10.0));
clientApps.Stop (stopTime);
```

```
 Simulator::Stop (stopTime + Seconds (10.0));


 if (tracing)
  {
    csma.EnablePcapAll ("dhcp-csma");
    pointToPoint.EnablePcapAll ("dhcp-
    p2p");
  }


 NS_LOG_INFO ("Run
 Simulation."); Simulator::Run
 (); Simulator::Destroy ();
 NS_LOG_INFO ("Done.");
}
```

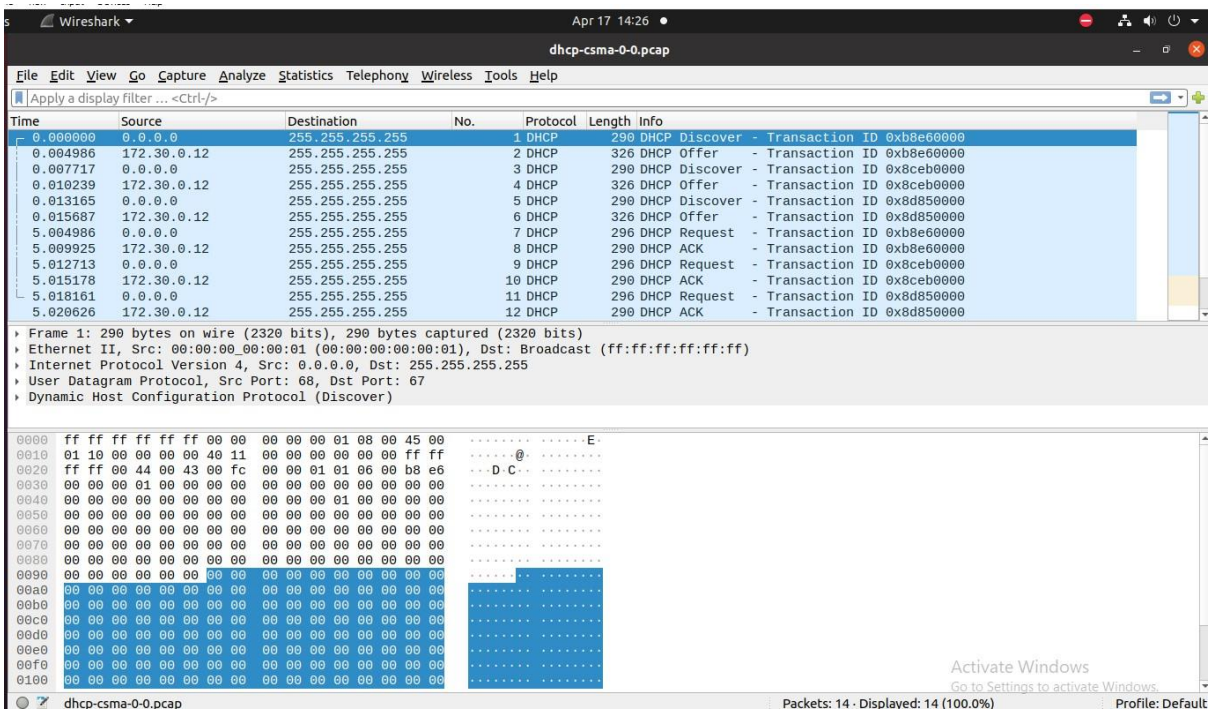
./waf --run sratch/dhcp.cc



./waf --run sratch/dhcp.cc --vis



Wireshark



10 Program to simulate FTP using TCP.

What is FTP ?

FTP stands for **File Transfer Protocol**. It is a standard network protocol used to transfer files between a client and a server over a computer network, such as the internet or an intranet.

FTP works on a client-server model, where:

- **FTP Client** is a program or application used to connect to an FTP server.
- **FTP Server** is a machine that stores files and allows clients to upload, download, or manage those files.

Key features of FTP:

1. **File Transfers**: FTP allows users to upload and download files between their local machine and a remote server.
2. **File Management**: It provides capabilities for renaming, deleting, and managing files and directories on the server.
3. **Authentication**: FTP usually requires a username and password for access (though there are anonymous FTP servers that allow access without authentication).
4. **Modes**: FTP operates in two modes:
    - **Active Mode**: The client opens a port for data transfer, and the server connects to the client.
    - **Passive Mode**: The server opens a port for data transfer, and the client connects to it.

Common FTP Commands:

- `get`: Download a file from the server.
- `put`: Upload a file to the server.
- `ls`: List files in the current directory.
- `cd`: Change the directory on the server.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-
module.h" #include
"ns3/applications-module.h"
#include "ns3/csma-module.h"
#include  "ns3/ipv4-global-routing-helper.h"

using namespace ns3;


NS_LOG_COMPONENT_DEFINE ("FTP_Simulation");

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

        Time::SetResolution (Time::NS);
        LogComponentEnable ("OnOffApplication", LOG_LEVEL_INFO);
        LogComponentEnable ("OnOffApplication", LOG_LEVEL_INFO);


        NodeContainer csmaNodes;

        csmaNodes.Create(2);

// using csma helper to configure attributes
// and creating connectivity between two nodes
```

```
        CsmaHelper csma;

// set channel attributes for device
        csma.SetChannelAttribute ("DataRate",StringValue("1Mbps"));
        csma.SetChannelAttribute ("Delay",StringValue("20ms"));



// installing nodes to create, configure, install devices of required type
        NetDeviceContainer csmaDevices;

        csmaDevices = csma.Install(csmaNodes);

// installing protocol stack in nodes; once executed will install TCP,UDP, IP etc. on each node of the nodeContainer

        InternetStackHelper stack;
        stack.Install(csmaNodes);

// assigning IP address to the devices, setting base address here
        Ipv4AddressHelper address;
        address.SetBase ("10.10.10.0","255.255.255.0");


        Ipv4InterfaceContainer csmaInterfaces =address.Assign(csmaDevices);
// assigning ip address to the devices, since devices contain nodes, while ip stack is created within a node

//      unint32_t port_num=21; // port number for Rx
```

Address RxAddress(InetSocketAddress (csmaInterfaces.GetAddress(1), 21)); //Rx obtaining a IPv4 address, and the port number to listen

PacketSinkHelper sinkHelper ("ns3::TcpSocketFactory", RxAddress); // PacketSinkHelper is a class reference for creating socket references and passing address at which Rx is instantiated

ApplicationContainer sinkApp = sinkHelper.Install (csmaNodes.Get(1)); // ApplicationContainer, installs application on the node, in our case it is packet sink helper ; node here is the server node

sinkApp.Start (Seconds (1.0));

sinkApp.Stop (Seconds (10.0));

//creating a TCP transmitter

// set up address and port number to send TCP traffic to the server

Address TxAddress(InetSocketAddress(csmaInterfaces.GetAddress(1),21));

OnOffHelper clientHelper ("ns3::TcpSocketFactory", TxAddress);

// setting up attributes for onoff application helper

clientHelper.SetAttribute("DataRate",StringValue("1Mbps"));

clientHelper.SetAttribute("PacketSize",UintegerValue(1280));

ApplicationContainer Tx = clientHelper.Install (csmaNodes.Get

(0));

Tx.Start (Seconds (1.0));

Tx.Stop (Seconds (10.0));

```
csma.EnablePcap("ftpeg",csmaDevices.Get(1),true);


AsciiTraceHelper ascii;
csma.EnableAsciiAll(ascii.CreateFileStream
("ftp_trace.tr"));


                        NS_LOG_INFO ("Run
                        Simulation."); Simulator::Run
                        ();
                        NS_LOG_INFO ("Done");
        return(0);
}
```

```
bvimit@bvimit-VirtualBox:~/workspace/ns-allinone-3.32/ns-3.32$ ./waf --run scratch/ftp
Waf: Entering directory `/home/bvimit/workspace/ns-allinone-3.32/ns-3.32/build'
Waf: Leaving directory `/home/bvimit/workspace/ns-allinone-3.32/ns-3.32/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (0.230s)
At time +2.01024s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 1280 bytes
At time +2.02048s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 2560 bytes
At time +2.03072s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 3840 bytes
At time +2.04096s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 5120 bytes
At time +2.0512s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 6400 bytes
At time +2.06144s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 7680 bytes
At time +2.07168s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 8960 bytes
At time +2.08192s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 10240 bytes
At time +2.09216s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 11520 bytes
At time +2.1024s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 12800 bytes
At time +2.11264s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 14080 bytes
At time +2.12288s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 15360 bytes
At time +2.13312s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 16640 bytes
At time +2.14336s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 17920 bytes
At time +2.1536s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 19200 bytes
At time +2.16384s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 20480 bytes
At time +2.17408s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 21760 bytes
At time +2.18432s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 23040 bytes
At time +2.19456s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 24320 bytes
At time +2.2048s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 25600 bytes
At time +2.21504s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 26880 bytes
At time +2.22528s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 28160 bytes
At time +2.23552s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 29440 bytes
At time +2.24576s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 30720 bytes
At time +2.256s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 32000 bytes
At time +2.26624s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 33280 bytes
At time +2.27648s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 34560 bytes
At time +2.28672s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 35840 bytes
At time +2.29696s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 37120 bytes
At time +2.3072s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 38400 bytes
At time +2.31744s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 39680 bytes
At time +2.32768s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 40960 bytes
At time +2.33792s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 42240 bytes
At time +2.34816s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 43520 bytes
At time +2.3584s on-off application sent 1280 bytes to 10.10.10.2 port 21 total Tx 44800 bytes
```

Wireshark · Apr 12 16:35 ·

ftpeg-1-0.pcap

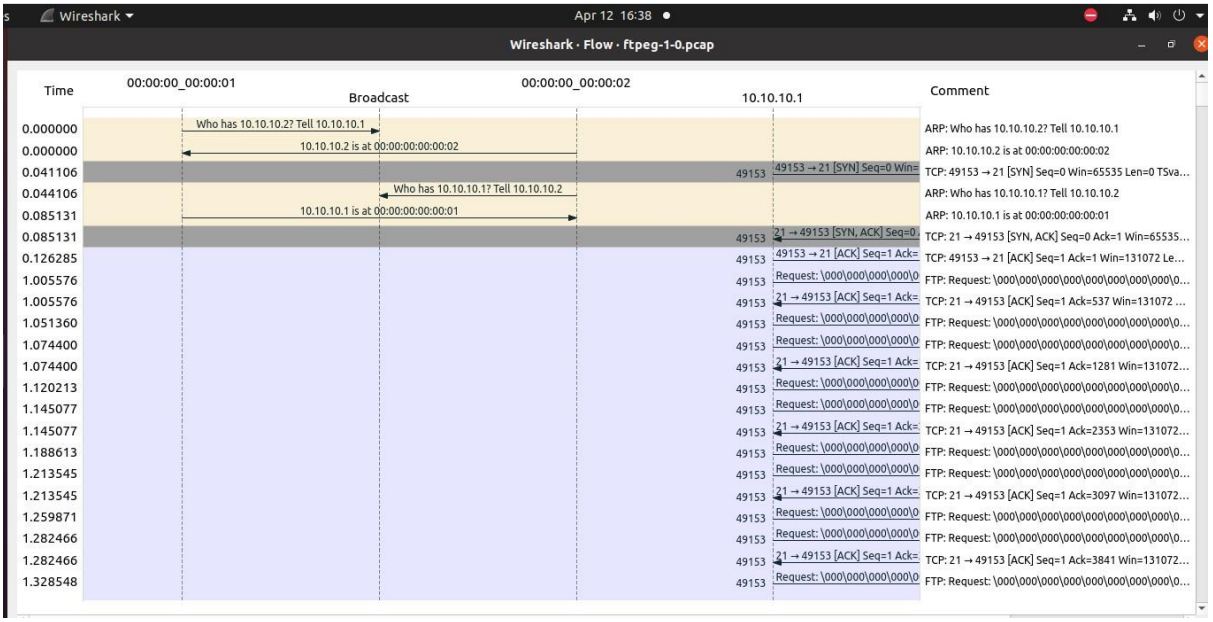
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

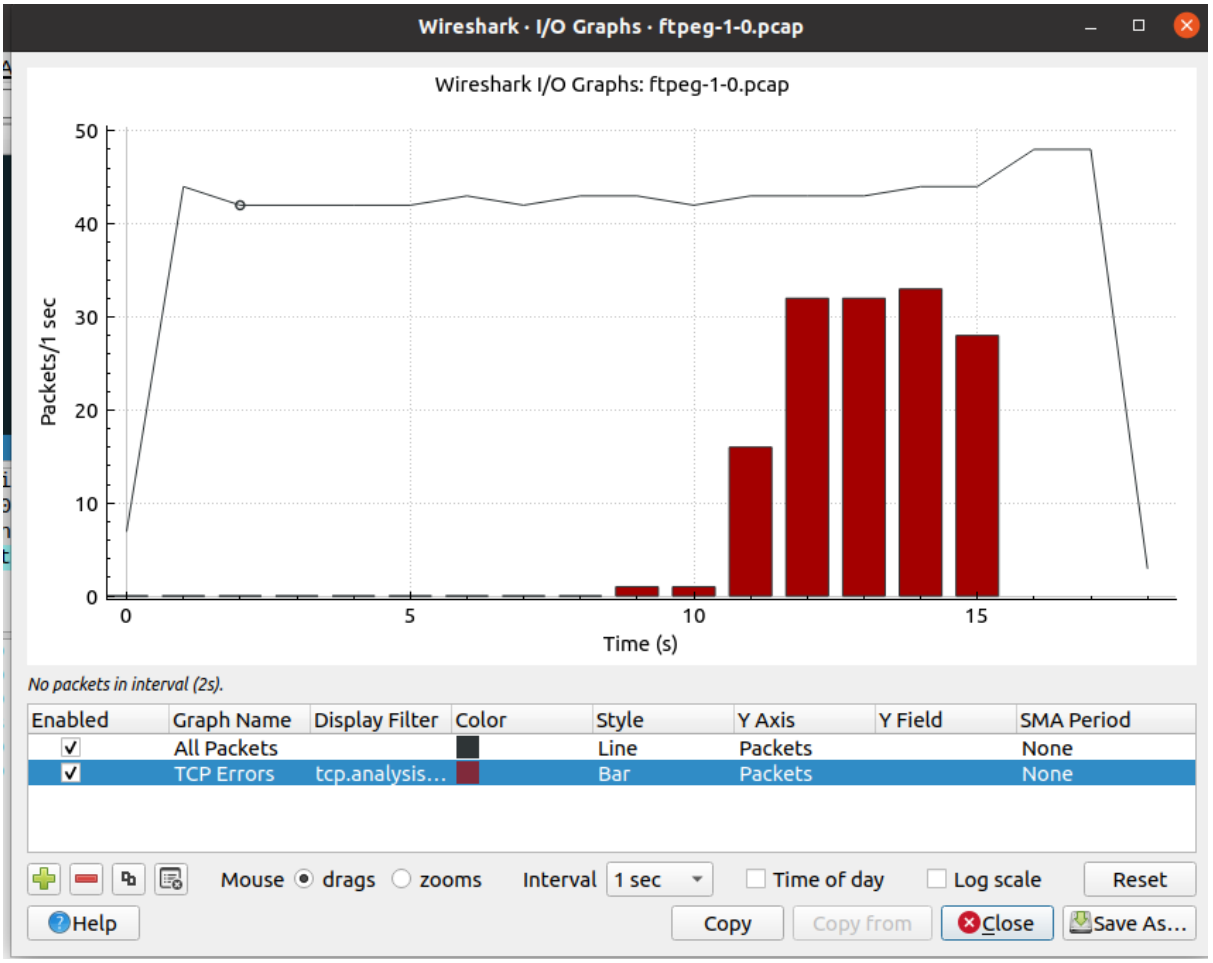
Apply a display filter … <Ctrl-/>

Time	Source	Destination	No.	Protocol	Length	Info
0.000000	00:00:00_00:00:01	Broadcast	1	ARP	64	Who has 10.10.10.2? Tell 10.10.10.1
0.000000	00:00:00_00:00:02	00:00:00_00:00:01	2	ARP	64	10.10.10.2 is at 00:00:00:00:00:02
0.041106	10.10.10.1	10.10.10.2	3	TCP	74	49153 → 21 [SYN] Seq=0 Win=65535 Len=0 TSval=1000 TSecr=0 WS=…
0.044106	00:00:00_00:00:02	Broadcast	4	ARP	64	Who has 10.10.10.1? Tell 10.10.10.2
0.085131	00:00:00_00:00:01	00:00:00_00:00:02	5	ARP	64	10.10.10.1 is at 00:00:00:00:00:01
0.085131	10.10.10.2	10.10.10.1	6	TCP	74	21 → 49153 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 TSval=1070 …
0.126285	10.10.10.1	10.10.10.2	7	TCP	70	49153 → 21 [ACK] Seq=1 Ack=1 Win=131072 Len=0 TSval=1135 TSec…
1.005576	10.10.10.1	10.10.10.2	8	FTP	606	Request: \000\000\000\000\000\000\000\000\000\000\000\000\000…
1.005576	10.10.10.2	10.10.10.1	9	TCP	70	21 → 49153 [ACK] Seq=1 Ack=537 Win=131072 Len=0 TSval=2035 TS…
1.051360	10.10.10.1	10.10.10.2	10	FTP	606	Request: \000\000\000\000\000\000\000\000\000\000\000\000\000…
1.074400	10.10.10.1	10.10.10.2	11	FTP	278	Request: \000\000\000\000\000\000\000\000\000\000\000\000\000…
1.074400	10.10.10.2	10.10.10.1	12	TCP	70	21 → 49153 [ACK] Seq=1 Ack=1281 Win=131072 Len=0 TSval=2103 T…

```
▸ Frame 1: 64 bytes on wire (512 bits), 64 bytes captured (512 bits)
▸ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Address Resolution Protocol (request)
```

```
0000  ff ff ff ff ff ff 00 00  00 00 00 01 08 06 00 01   ........ ........
0010  08 00 06 04 00 01 00 00  00 00 00 01 0a 0a 0a 01   ........ ........
0020  ff ff ff ff ff ff 0a 0a  0a 02 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
```

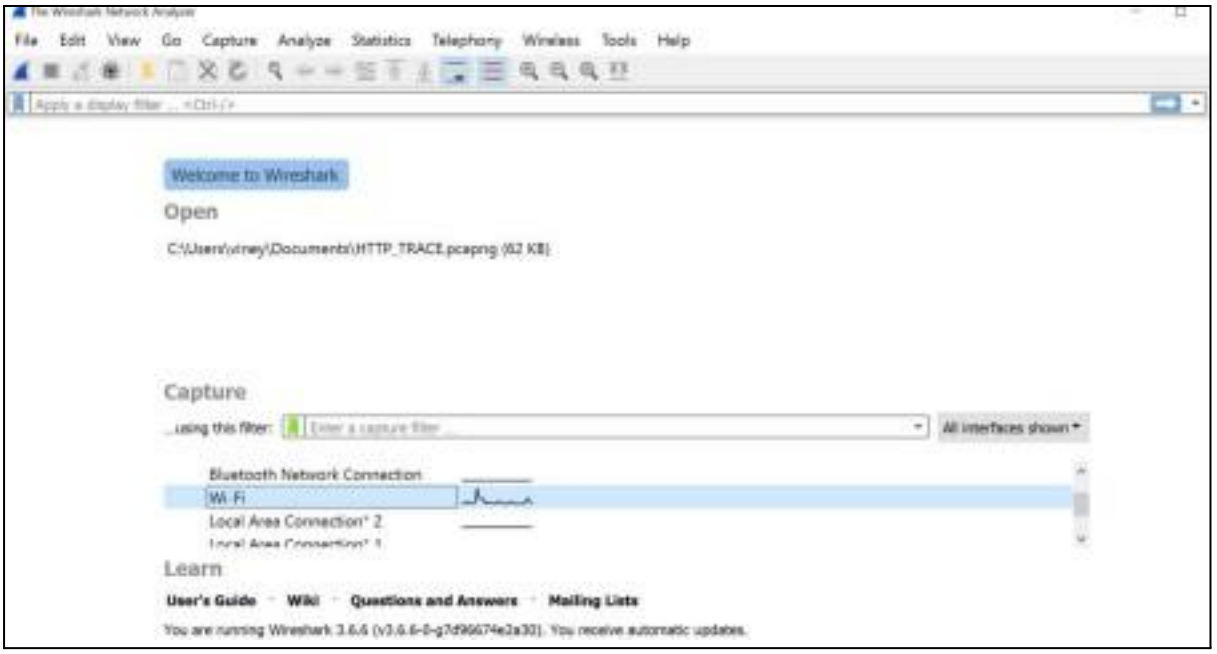

Wireshark
Apr 12 16:38
Wireshark · Flow · ftpeg-1-0.pcap
Time
00:00:00_00:00:01
Broadcast
00:00:00_00:00:02
10.10.10.1
Comment
0.000000 Who has 10.10.10.2? Tell 10.10.10.1
ARP: Who has 10.10.10.2? Tell 10.10.10.1
0.000000 10.10.10.2 is at 00:00:00:00:00:02
ARP: 10.10.10.2 is at 00:00:00:00:00:02
0.041106 49153 → 21 [SYN] Seq=0 Win=
TCP: 49153 → 21 [SYN] Seq=0 Win=65535 Len=0 TSva...
0.044106 Who has 10.10.10.1? Tell 10.10.10.2
ARP: Who has 10.10.10.1? Tell 10.10.10.2
0.085131 10.10.10.1 is at 00:00:00:00:00:01
ARP: 10.10.10.1 is at 00:00:00:00:00:01
0.085131 21 → 49153 [SYN, ACK] Seq=0
TCP: 21 → 49153 [SYN, ACK] Seq=0 Ack=1 Win=65535...
0.126285 49153 → 21 [ACK] Seq=1 Ack=
TCP: 49153 → 21 [ACK] Seq=1 Ack=1 Win=131072 Le...
1.005576 Request: \000\000\000\000\0
FTP: Request: \000\000\000\000\000\000\000\000\0...
1.005576 21 → 49153 [ACK] Seq=1 Ack=
TCP: 21 → 49153 [ACK] Seq=1 Ack=537 Win=131072 ...
1.051360 Request: \000\000\000\000\0
FTP: Request: \000\000\000\000\000\000\000\000\0...
1.074400 Request: \000\000\000\000\0
FTP: Request: \000\000\000\000\000\000\000\000\0...
1.074400 21 → 49153 [ACK] Seq=1 Ack=
TCP: 21 → 49153 [ACK] Seq=1 Ack=1281 Win=131072...
1.120213 Request: \000\000\000\000\0
FTP: Request: \000\000\000\000\000\000\000\000\0...
1.145077 Request: \000\000\000\000\0
FTP: Request: \000\000\000\000\000\000\000\000\0...
1.145077 21 → 49153 [ACK] Seq=1 Ack=
TCP: 21 → 49153 [ACK] Seq=1 Ack=2353 Win=131072...
1.188613 Request: \000\000\000\000\0
FTP: Request: \000\000\000\000\000\000\000\000\0...
1.213545 Request: \000\000\000\000\0
FTP: Request: \000\000\000\000\000\000\000\000\0...
1.213545 21 → 49153 [ACK] Seq=1 Ack=
TCP: 21 → 49153 [ACK] Seq=1 Ack=3097 Win=131072...
1.259871 Request: \000\000\000\000\0
FTP: Request: \000\000\000\000\000\000\000\000\0...
1.282466 Request: \000\000\000\000\0
FTP: Request: \000\000\000\000\000\000\000\000\0...
1.282466 21 → 49153 [ACK] Seq=1 Ack=
TCP: 21 → 49153 [ACK] Seq=1 Ack=3841 Win=131072...
1.328548 Request: \000\000\000\000\0
FTP: Request: \000\000\000\000\000\000\000\000\0...


Wireshark · I/O Graphs · ftpeg-1-0.pcap
Wireshark I/O Graphs: ftpeg-1-0.pcap
Packets/1 sec
Time (s)
0
10
20
30
40
50
0
5
10
15
No packets in interval (2s).
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
Mouse
drags
zooms
Interval
1 sec
Time of day
Log scale
Reset
Help
Copy
Copy from
Close
Save As...

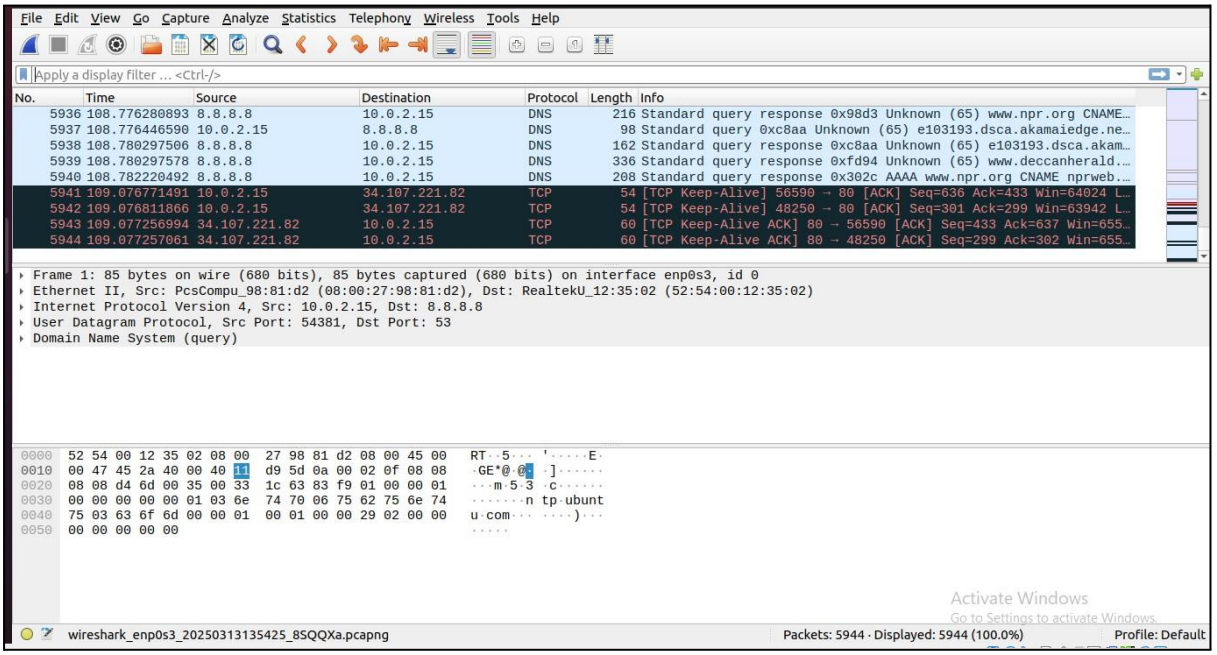
11. Exercises for analyzing the network protocols using Wireshark · Capture the packets while browsing the any web site · Analyze the header fields of various protocols.

Analyse the network traffic using Wireshark

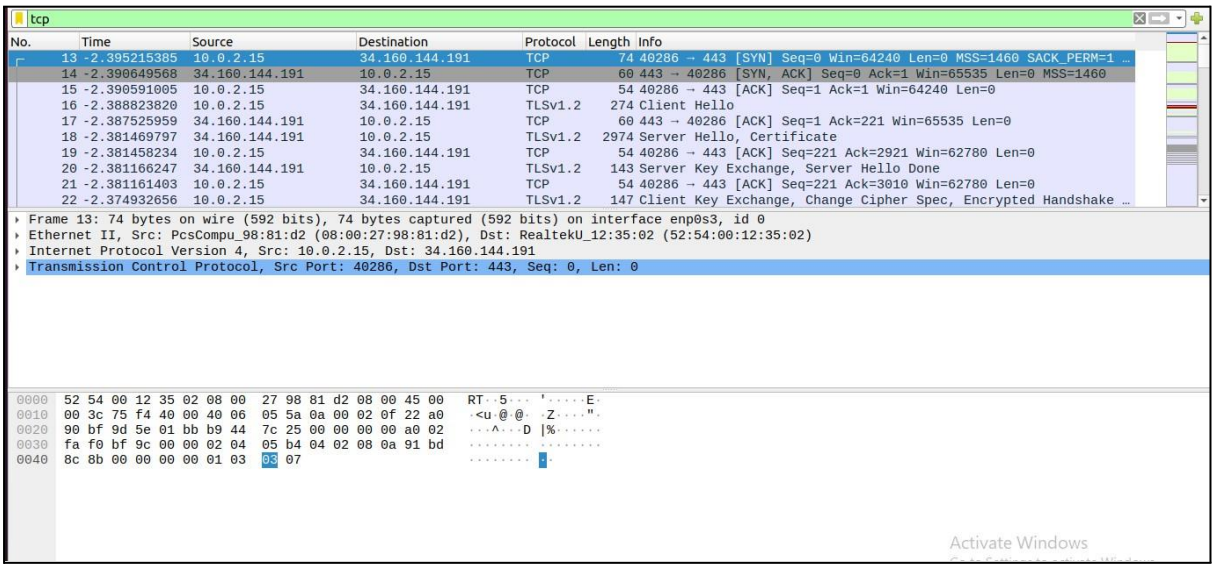
Step 1: Open Wireshark and select the network connection, which is Wifi.



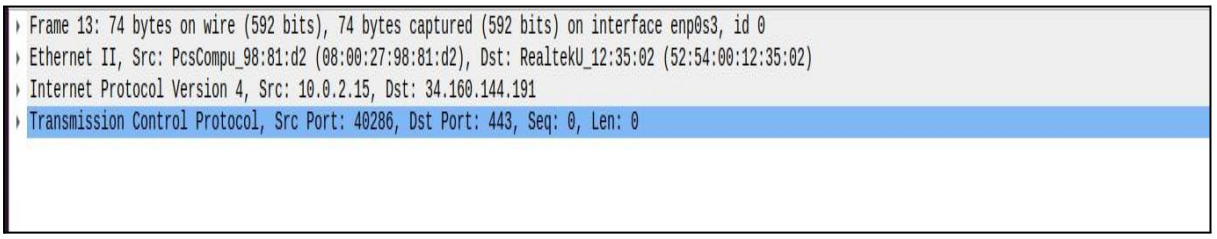
Step 2: Start capturing packets, which will record the browsing history. And then stop capturing.



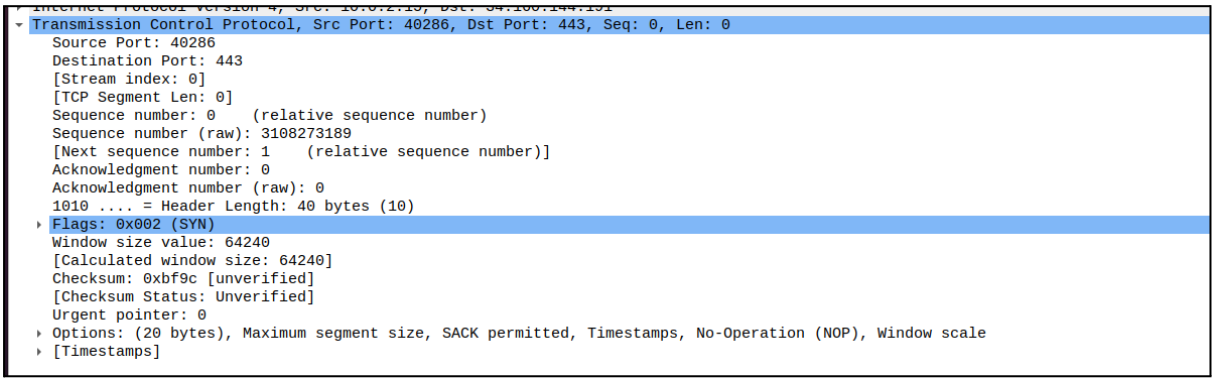
Step 3: Search for all TCP protocol packets.



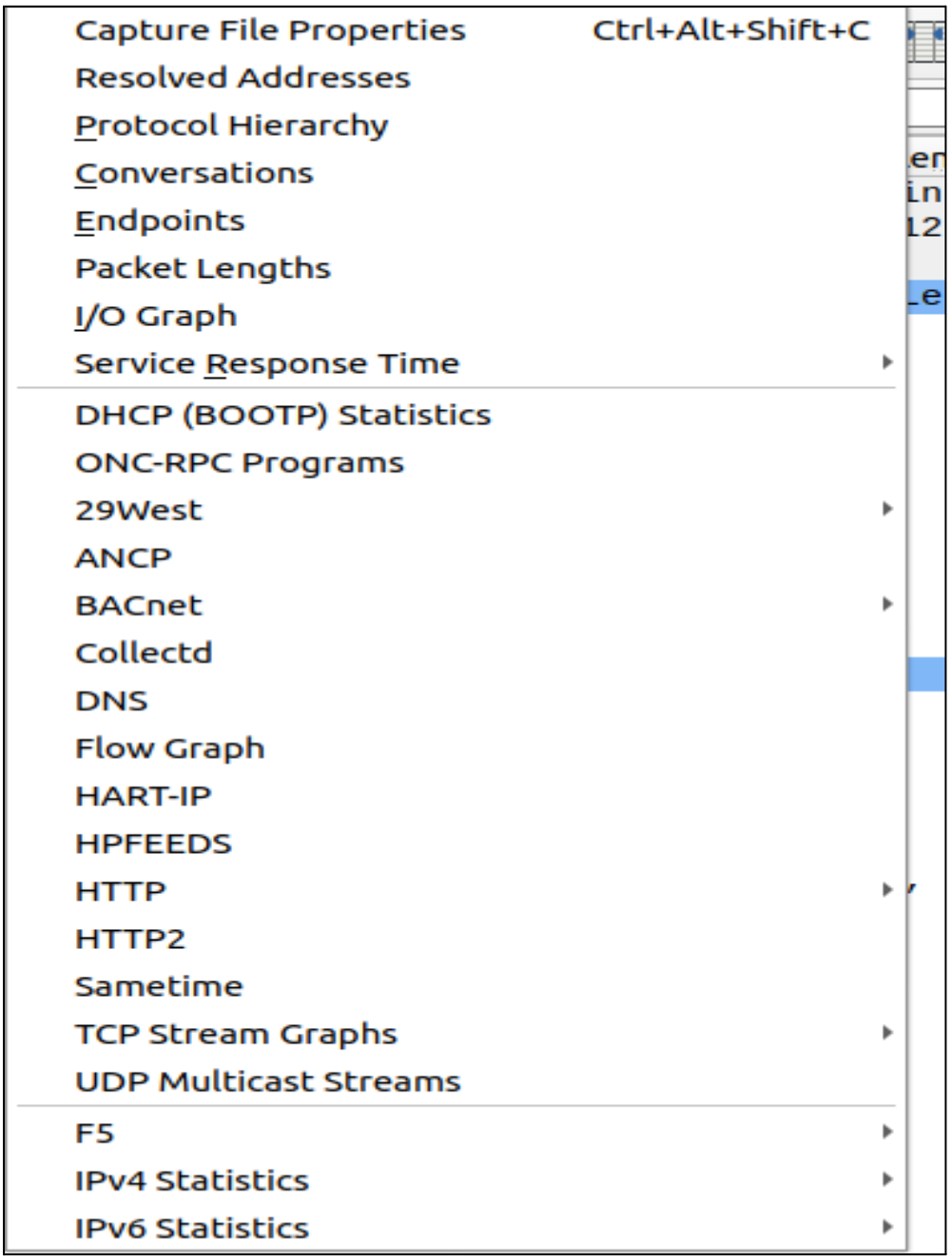
You can select any packets to get the information about that.

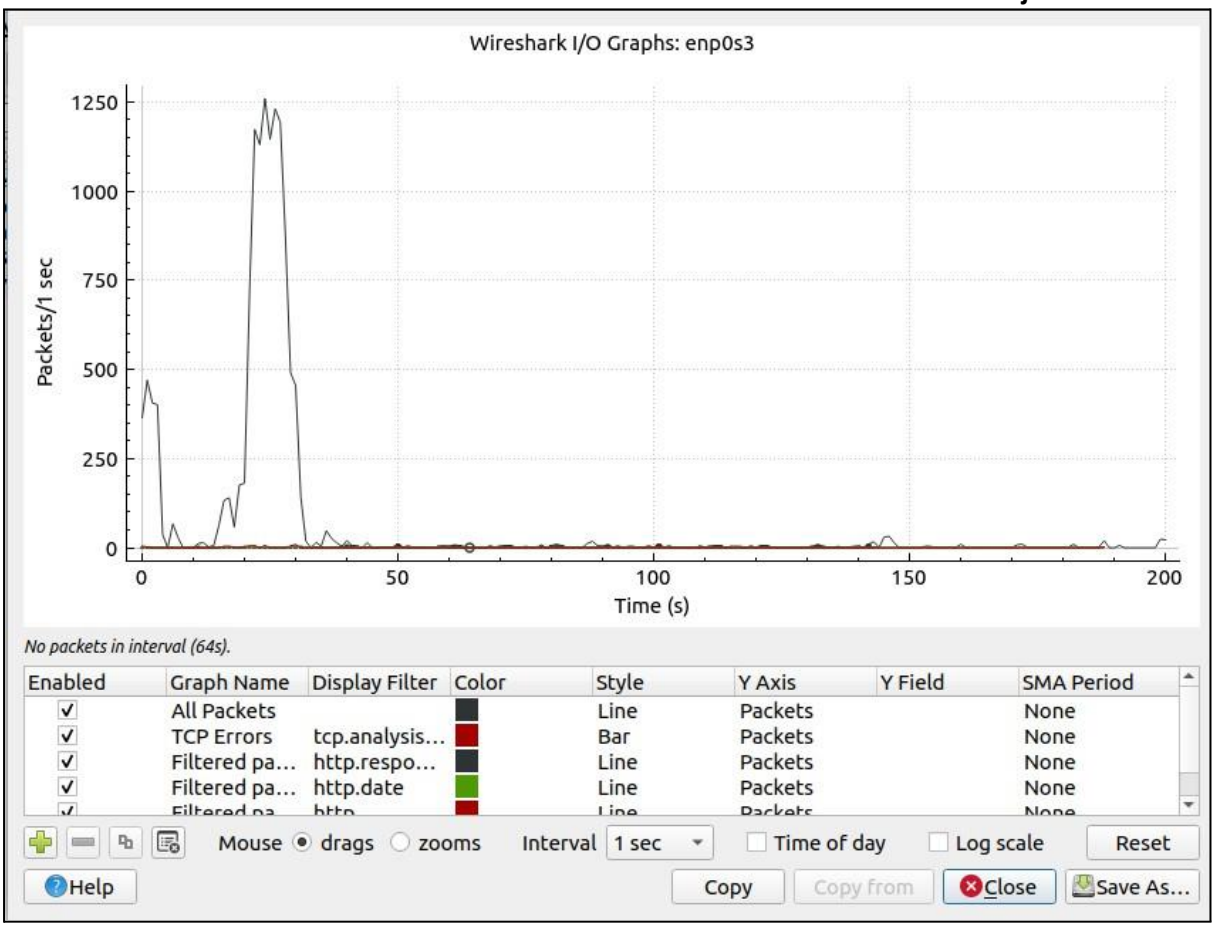


Then expand the Transmission Control Protocol column to get additional details.

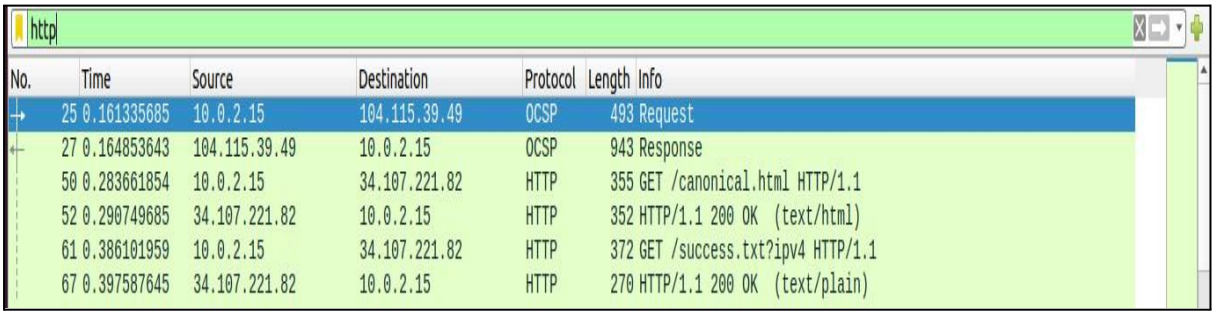


You can analyze the statistics graphically. Click on the statistics option.

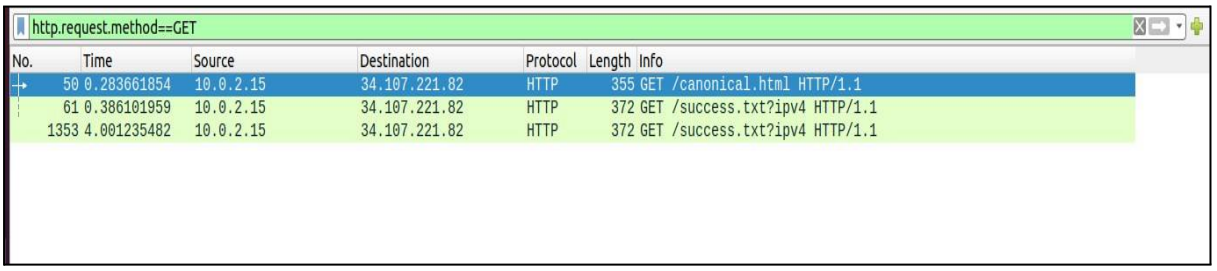




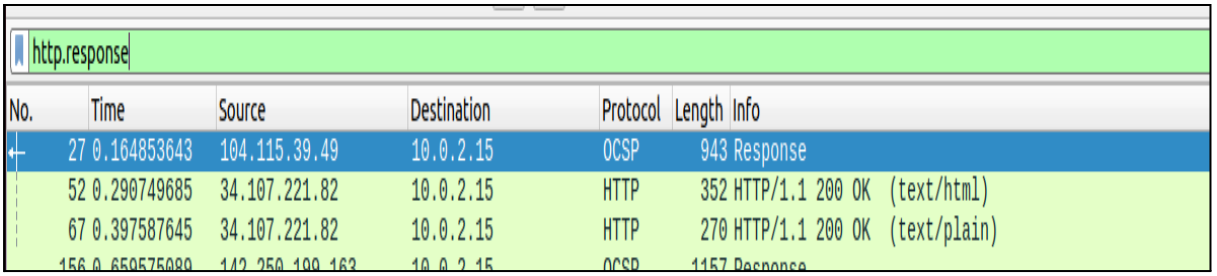
Step 4: Search for all HTTP protocol packets.



Step 5: Search for all HTTP request protocol packets with the GET method.



Step 6: Search for all HTTP response protocol packets.



12. Evaluate the network performance using metrics: throughput, delay, response time, packet loss, dropped packets etc. (All Topology).

Evaluating **network performance** involves analyzing several **key metrics** across all types of **topologies** (like star, bus, ring, mesh, etc.).

1. Throughput

Definition:
The amount of **data successfully transmitted** from source to destination in a given time, usually in **bps (bits per second)**.

High throughput = Good network performance.

How to measure:

- Use tools like iPerf, Wireshark, or real-time monitoring systems.
- Formula:
  Throughput = (Total data received / Total time)

Factors affecting it:

- Bandwidth
- Network congestion
- Protocol overhead
- Packet loss

2. Delay (Latency)

Definition:
The time it takes for a packet to travel from source to destination.

Measured in: milliseconds (ms)
Types of delay:

- **Propagation delay:** Time to travel across the medium
- **Transmission delay:** Time to push bits onto the link
- **Processing delay:** Time to process the packet
- **Queuing delay:** Time waiting in queues

Lower delay = Better performance.

3. Response Time

Definition:
Time taken from the **user's request to the receipt of a response.**
Includes:

- Transmission time
- Server processing time
- Return time

Used for: Web services, applications, client-server models
Ideal: <100ms for web apps, <1s for interactive systems

4. Packet Loss

Definition:
The **percentage of packets that never reach** their destination.

Packet loss (%) = (Lost packets / Sent packets) × 100

Causes:

- Network congestion
- Faulty hardware
- Weak Wi-Fi signals

Low packet loss (ideally 0%) is desired.

5. Dropped Packets

Definition:
Packets that are intentionally **discarded** due to full buffers, congestion, or routing errors.

Difference from packet loss:
All dropped packets are lost, but not all lost packets are dropped intentionally.

Measured via:

- Router/switch logs
- Monitoring tools (e.g., SNMP, Wireshark)

Example Evaluation Table (All Topologies)

Metric	Star Topology	Mesh Topology	Ring Topology	Bus Topology
Throughput	High (centralized)	Very High (redundant links)	Moderate	Moderate (shared medium)
Delay	Low	Low (short paths)	Medium (one-way path)	High under load
Response Time	Fast	Fast	Slower with distance	Slower with collisions
Packet Loss	Low	Very Low (backup links)	Medium	High during collisions
Dropped Packets	Low	Very Low	Moderate	High under load

Tools You Can Use:

- **iPerf / iPerf3** (Throughput)
- **Ping** (Delay, packet loss)
- **Traceroute** (Delay across hops)
- **Wireshark** (Detailed analysis)
- **NetFlow / SNMP** (Enterprise monitoring)
- **Network Simulator (NS2/NS3, Cisco Packet Tracer)** for topology-based simulations