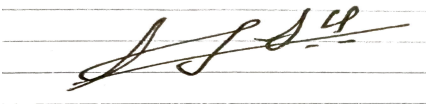


Name of Student : Sunny Satish Halkatti		
Roll Number : 17		LAB Assignment Number: 9
Title of LAB Assignment: Program to simulate FTP using TCP protocol.		
DOP : 11-05-2023		DOS : 25-05-2023
CO Mapped : CO2	PO Mapped: PO1, PO2, PO3, PO5, PO7, PSO1	Signature : 

NWL - Practical - 9

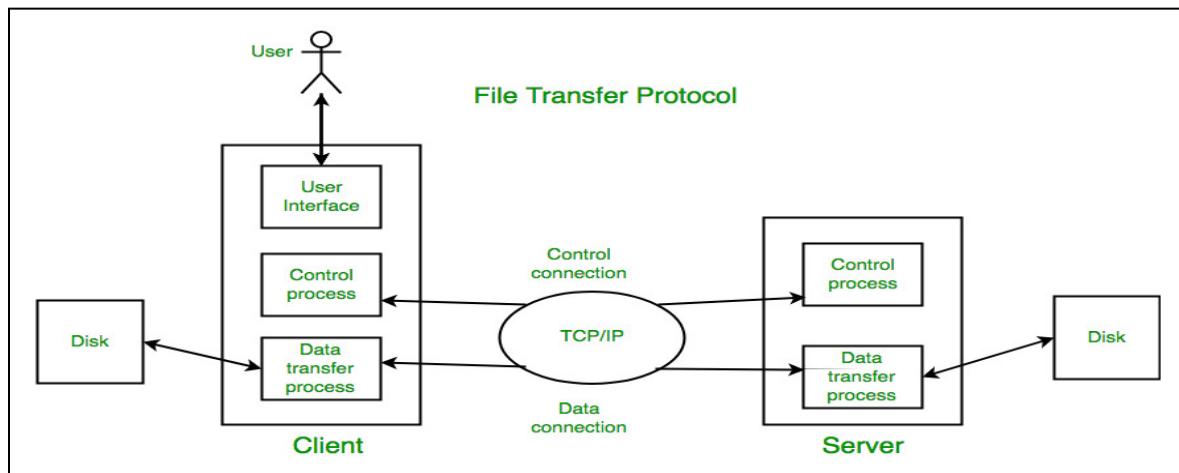
Aim: Program to simulate FTP using TCP protocol.

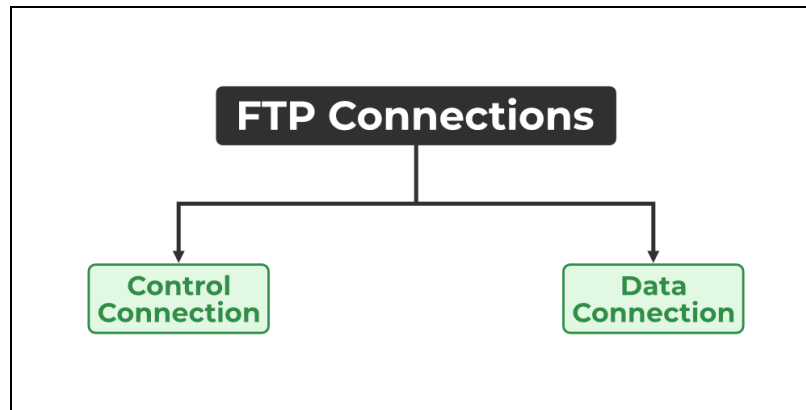
Theory:

File Transfer Protocol(FTP) is an application layer protocol that moves files between local and remote file systems. It runs on top of TCP, like HTTP. To transfer a file, 2 TCP connections are used by FTP in parallel: control connection and data connection.

Why FTP?

FTP is a standard communication protocol. There are various other protocols like HTTP which are used to transfer files between computers, but they lack clarity and focus as compared to FTP. Moreover, the systems involved in connection are heterogeneous systems, i.e. they differ in operating systems, directories, structures, character sets, etc the FTP shields the user from these differences and transfers data efficiently and reliably. FTP can transfer ASCII, EBCDIC, or image files. The ASCII is the default file share format, in this, each character is encoded by NVT ASCII. In ASCII or EBCDIC the destination must be ready to accept files in this mode. The image file format is the default format for transforming binary files.





1. **Control Connection:** For sending control information like user identification, password, commands to change the remote directory, commands to retrieve and store files, etc., FTP makes use of a control connection. The control connection is initiated on port number 21.

2. **Data connection:** For sending the actual file, FTP makes use of a data connection. A data connection is initiated on port number 20. FTP sends the control information out-of-band as it uses a separate control connection. Some protocols send their request and response header lines and the data in the same TCP connection. For this reason, they are said to send their control information in-band. HTTP and SMTP are such examples.

Characteristics of FTP

FTP uses TCP as a transport layer protocol.

It is good for simple file transfers, such as during boot time.

Errors in the transmission (lost packets, checksum errors) must be handled by the TFTP server.

It uses only one connection through well-known port 69.

TFTP uses a simple lock-step protocol (each data packet needs to be acknowledged). Thus the throughput is limited.

Code:

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */
//
// Network topology
//
// 10Mb/s, 10ms    10Mb/s, 10ms
// n0-----n1-----n2
//
//
// - Tracing of queues and packet receptions to file
// "tcp-large-transfer.tr"
// - pcap traces also generated in the following files
// "tcp-large-transfer-$n-$i.pcap" where n and i represent node and interface
// numbers respectively
// Usage (e.g.): ./waf --run tcp-large-transfer
#include <iostream>
#include <fstream>
#include <string>
#include "ns3/core-module.h"
#include "ns3/applications-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/netanim-module.h"
```

```
using namespace ns3;
NS_LOG_COMPONENT_DEFINE("TcpLargeTransfer");
// The number of bytes to send in this simulation.
static const uint32_t totalTxBytes = 2000000;
static uint32_t currentTxBytes = 0;
// Perform series of 1040 byte writes (this is a multiple of 26 since
// we want to detect data splicing in the output stream)
static const uint32_t writeSize = 1040;
uint8_t data[writeSize];
// These are for starting the writing process, and handling the sending
// socket's notification upcalls (events). These two together more or less
// implement a sending "Application", although not a proper ns3::Application
// subclass.
void StartFlow(Ptr<Socket>, Ipv4Address, uint16_t);
void WriteUntilBufferFull(Ptr<Socket>, uint32_t);
static void
CwndTracer(uint32_t oldval, uint32_t newval)
{
    NS_LOG_INFO("Moving cwnd from " << oldval << " to " << newval);
}
int main(int argc, char *argv[])
{
    // Users may find it convenient to turn on explicit debugging
    // for selected modules; the below lines suggest how to do this
    // LogComponentEnable("TcpL4Protocol", LOG_LEVEL_ALL);
    // LogComponentEnable("TcpSocketImpl", LOG_LEVEL_ALL);
    // LogComponentEnable("PacketSink", LOG_LEVEL_ALL);
    // LogComponentEnable("TcpLargeTransfer", LOG_LEVEL_ALL);
    CommandLine cmd(__FILE__);
    cmd.Parse(argc, argv);
    // initialize the tx buffer.
    for (uint32_t i = 0; i < writeSize; ++i)
    {
        char m = toascii(97 + i % 26);
        data[i] = m;
    }
    // Here, we will explicitly create three nodes. The first container contains
    // nodes 0 and 1 from the diagram above, and the second one contains nodes
    // 1 and 2. This reflects the channel connectivity, and will be used to
    // install the network interfaces and connect them with a channel.
```

```
NodeContainer n0n1;
n0n1.Create(2);
NodeContainer n1n2;
n1n2.Add(n0n1.Get(1));
n1n2.Create(1);
// We create the channels first without any IP addressing information
// First make and configure the helper, so that it will put the appropriate
// attributes on the network interfaces and channels we are about to install.
PointToPointHelper p2p;
p2p.SetDeviceAttribute("DataRate", DataRateValue(DataRate(10000000)));
p2p.SetChannelAttribute("Delay", TimeValue(MilliSeconds(10)));
// And then install devices and channels connecting our topology.
NetDeviceContainer dev0 = p2p.Install(n0n1);
NetDeviceContainer dev1 = p2p.Install(n1n2);
// Now add ip/tcp stack to all nodes.
InternetStackHelper internet;
internet.InstallAll();
// Later, we add IP addresses.
Ipv4AddressHelper ipv4;
ipv4.SetBase("10.1.3.0", "255.255.255.0");
ipv4.Assign(dev0);
ipv4.SetBase("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer ipInterfs = ipv4.Assign(dev1);
// and setup ip routing tables to get total ip-level connectivity.
Ipv4GlobalRoutingHelper::PopulateRoutingTables();
////////////////////////////////////
// Simulation 1
//
// Send 2000000 bytes over a connection to server port 50000 at time 0
// Should observe SYN exchange, a lot of data segments and ACKS, and FIN
// exchange. FIN exchange isn't quite compliant with TCP spec (see release
// notes for more info)
//
////////////////////////////////////
uint16_t servPort = 50000;
// Create a packet sink to receive these packets on n2...
PacketSinkHelper sink("ns3::TcpSocketFactory",
                      InetSocketAddress(Ipv4Address::GetAny(), servPort));
ApplicationContainer apps = sink.Install(n1n2.Get(1));
apps.Start(Seconds(0.0));
```

```
apps.Stop(Seconds(3.0));
// Create a source to send packets from n0. Instead of a full Application
// and the helper APIs you might see in other example files, this example
// will use sockets directly and register some socket callbacks as a sending
// "Application".
// Create and bind the socket...
Ptr<Socket> localSocket =
    Socket::CreateSocket(n0n1.Get(0), TcpSocketFactory::GetTypeId());
localSocket->Bind();
// Trace changes to the congestion window

Config::ConnectWithoutContext("/NodeList/0/$ns3::TcpL4Protocol/SocketList/0/CongestionWi
ndow", MakeCallback(&CwndTracer));
// ...and schedule the sending "Application"; This is similar to what an
// ns3::Application subclass would do internally.
Simulator::ScheduleNow(&StartFlow, localSocket,
    ipInterfs.GetAddress(1), servPort);
// One can toggle the comment for the following line on or off to see the
// effects of finite send buffer modelling. One can also change the size of
// said buffer.

// localSocket->SetAttribute("SndBufSize", UintegerValue(4096));
// Ask for ASCII and pcap traces of network traffic
AsciiTraceHelper ascii;
p2p.EnableAsciiAll(ascii.CreateFileStream("tcp-large-transfer.tr"));
p2p.EnablePcapAll("tcp-large-transfer");

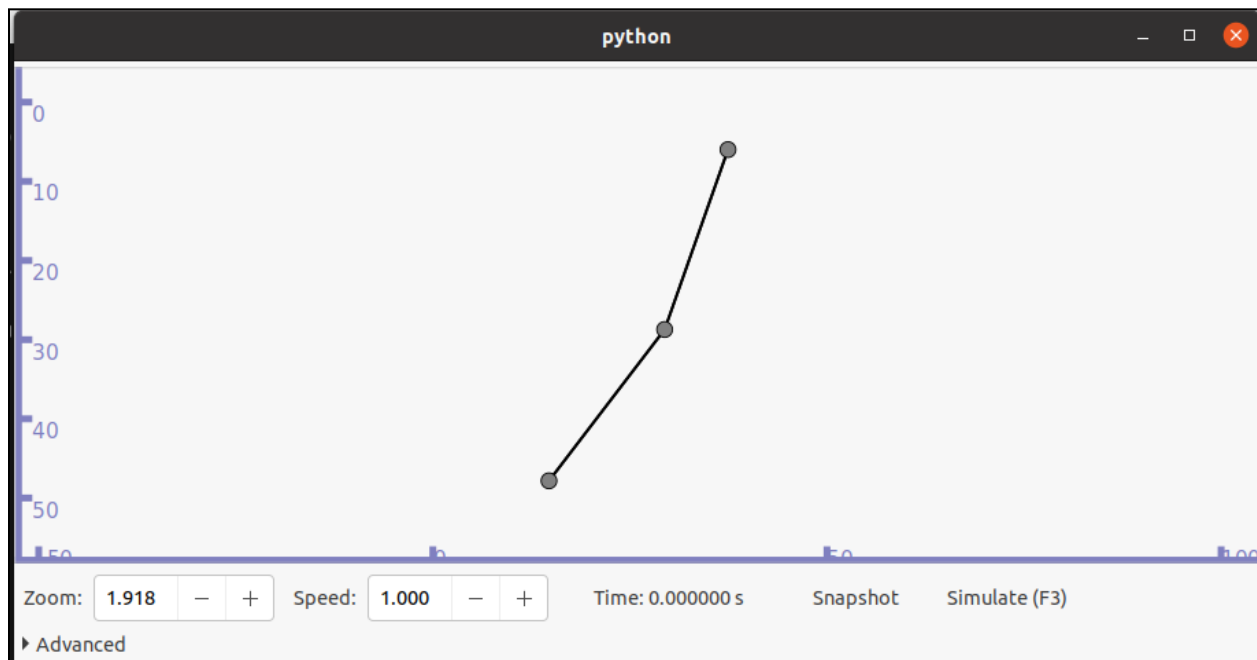
AnimationInterface anim("ftp-tcp.xml");
// Finally, set up the simulator to run. The 1000 second hard limit is a
// failsafe in case some change above causes the simulation to never end
Simulator::Stop(Seconds(1000));
Simulator::Run();
Simulator::Destroy();
}
//-----
//-----
//-----
// begin implementation of sending "Application"
void StartFlow(Ptr<Socket> localSocket,
    Ipv4Address servAddress,
```

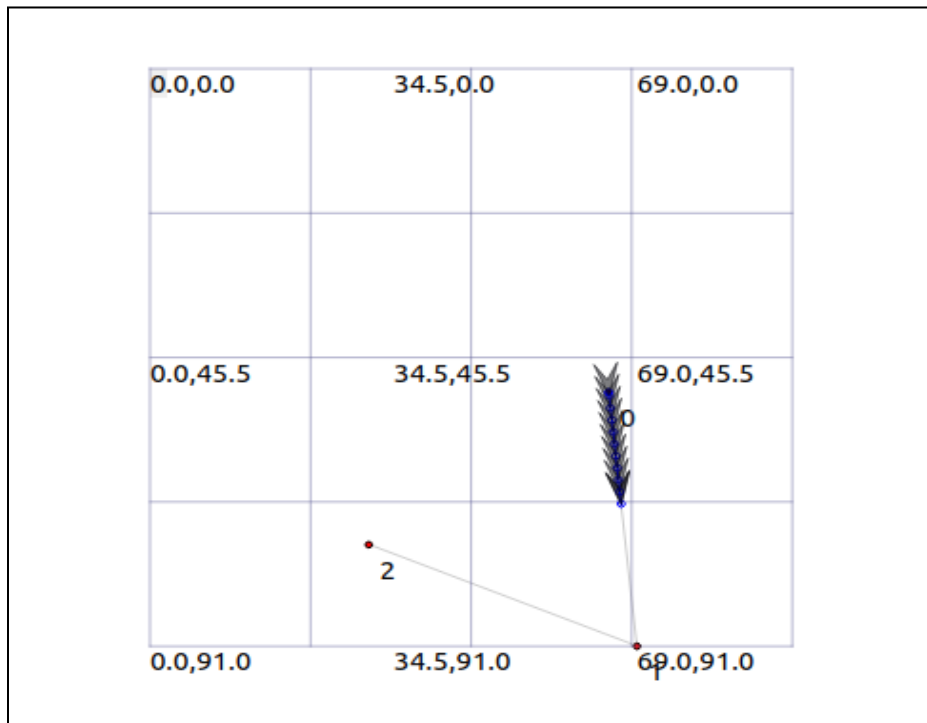
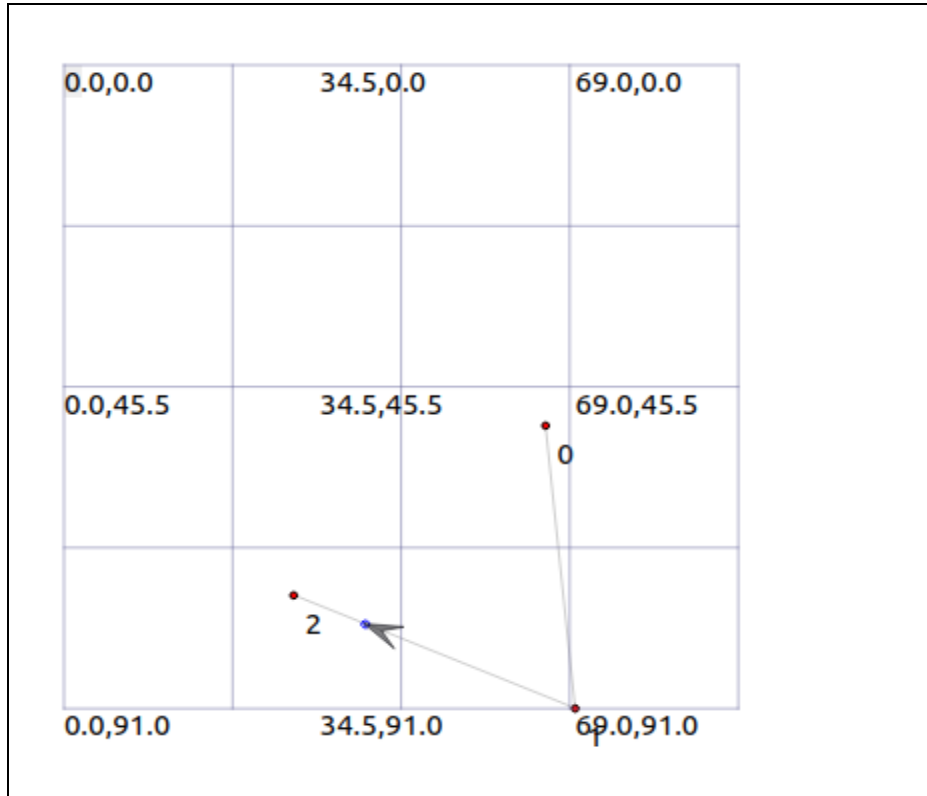
```
        uint16_t servPort)
    {
        NS_LOG_LOGIC("Starting flow at time " << Simulator::Now().GetSeconds());
        localSocket->Connect(InetSocketAddress(servAddress, servPort)); // connect
        // tell the tcp implementation to call WriteUntilBufferFull again
        // if we blocked and new tx buffer space becomes available
        localSocket->SetSendCallback(MakeCallback(&WriteUntilBufferFull));
        WriteUntilBufferFull(localSocket, localSocket->GetTxAvailable());
    }
void WriteUntilBufferFull(Ptr<Socket> localSocket, uint32_t txSpace)
{
    while (currentTxBytes < totalTxBytes && localSocket->GetTxAvailable() > 0)
    {
        uint32_t left = totalTxBytes - currentTxBytes;
        uint32_t dataOffset = currentTxBytes % writeSize;
        uint32_t toWrite = writeSize - dataOffset;
        toWrite = std::min(toWrite, left);

        toWrite = std::min(toWrite, localSocket->GetTxAvailable());
        int amountSent = localSocket->Send(&data[dataOffset], toWrite, 0);
        if (amountSent < 0)
        {
            // we will be called again when new tx space becomes available.
            return;
        }
        currentTxBytes += amountSent;
    }
    if (currentTxBytes >= totalTxBytes)
    {
        localSocket->Close();
    }
}
```


Output:

```
vaish@vaish-VirtualBox:~/Workspace/ns-allinone-3.32/ns-3.32$ ./waf --run prac9Retry --vis
Waf: Entering directory `/home/vaish/Workspace/ns-allinone-3.32/ns-3.32/build'
Waf: Leaving directory `/home/vaish/Workspace/ns-allinone-3.32/ns-3.32/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (0.458s)
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:2 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:2 Does not have a mobility model. Use SetConstantPosition if it is stationary
Could not load plugin 'show_last_packets.py': No module named 'kiwi'
Could not load icon applets-screenshooter due to missing gnomedesktop Python module
scanning topology: 3 nodes...
scanning topology: calling graphviz layout
scanning topology: all done.
vaish@vaish-VirtualBox:~/Workspace/ns-allinone-3.32/ns-3.32$
```



**Conclusions:**

We have successfully implemented a program to simulate FTP using TCP protocol in this practical.