**Implementation of The Onion Routing (TOR) Network Using NS3 Network Simulator**

Project guide – Prof. Jyoti Ramteke

Group Members:

Pratik Panjwani – 48

Rahul Rai – 52

Soumil Rao – 53

**Problem Definition/Statement:**

Implementation of Onion Routing Network using NS3 for User Anonymity.

**Onion Routing:**

Onion routing is a technique for [anonymous](http://en.wikipedia.org/wiki/Anonymity) communication over a [computer network](http://en.wikipedia.org/wiki/Computer_network). In an onion network, messages are encapsulated in layers of [encryption](http://en.wikipedia.org/wiki/Encryption), analogous to layers of the vegetable [onion](http://en.wikipedia.org/wiki/Onion). The encrypted data is transmitted through a series of network nodes called onion routers, each of which "peels" away a single layer, uncovering the data's next destination. When the final layer is decrypted, the message arrives at its destination. The sender remains anonymous because each intermediary knows only the location of the immediately preceding and following nodes.

**The Onion Data Structure:**

An onion is the data structure formed by "wrapping" a message with successive layers of encryption to be decrypted ("peeled" or "unwrapped") by as many intermediary computers as there are layers before arriving at its destination. The original message remains hidden as it is transferred from one node to the next, and no intermediary knows both the origin and final destination of the data, allowing the sender to remain anonymous.

**How Onion Routing Works:**

To create and transmit an onion, the originator selects a set of nodes from a list provided by a "directory node". The chosen nodes are arranged into a path, called a "chain" or "circuit", through which the message will be transmitted. Using [asymmetric key cryptography](http://en.wikipedia.org/wiki/Asymmetric_key_cryptography), the originator obtains a [public key](http://en.wikipedia.org/wiki/Public_key) from the directory node to send an encrypted message to the first ("entry") node, establishing a connection and a [shared secret](http://en.wikipedia.org/wiki/Shared_secret) ("session key"). Using the established encrypted link to the entry node, the originator can then relay a message through the first node to a second node in the chain. When the second node receives the message, it establishes a connection with the first node. This extends the “chain” to the second node. Similarly, the “chain” is extended till the exit node, with a unique, secret “session key” shared between each Tor node and the originator. Since the session key between each node and the originator is unique, no Tor node can decrypt a layer of encryption apart from the layer it is meant to decrypt.

The number of hops in the chain is limited to preserve performance. If the chain is too long, the response time becomes substantially high.

**Software Requirements:**

1. GNU/Linux Operating System
2. GCC compiler
3. Python interpreter
4. Network Simulator 3 installed (Optional – Network Animator)
5. Wireshark to read packet capture traces

**Hardware Requirements:**

Machine with Linux OS installed and at least 1.5 GB of free memory for installation of NS3

**Objective:**

To implement and simulate working of a Tor Network in order to establish user anonymity. If packets during transmission are intercepted by malicious users, the original source and destination of the packet as well as the packet data should not be revealed.

**Literature Work/Study:**

Several reliable sources on the internet were used and a few publications were referred, including:

1. A Formal Treatment of Onion Routing

<http://cs.brown.edu/~anna/papers/cl05.pdf>

1. Onion Routing for Anonymous Communications

<http://ntrg.cs.tcd.ie/undergrad/4ba2.05/group10/>

1. Onion Sleeve Routing Algorithm

<http://ijates.com/images/short_pdf/1401971204_P26-36.pdf>

1. Onion Routing for Anonymous and Private Internet Connections

<http://www.onion-router.net/Publications/CACM-1999.pdf>

**Implementation:**

Network Simulator 3 is a discrete event network simulator with scripting capability in C++ and Python. It is a powerful simulation tool with provisions to study and simulate widely used protocols, design new protocols, study wireless networks, topologies and analyse performance. It also has a Network Animator tool for graphical visualization of networks.

For the sake of simplicity, we assume that the secure key exchange between the originator and each node in the Tor network is complete. Each node in the Tor network shares a unique secret key with the originator. We also assume that the number of hops in the chain is 3.

i.e. Originator -> Entry Node -> Intermediate Node -> Exit Node -> Destination

We implement the Tor network protocol by designing a new Header for the packets called the “Tor Header”. The structure of the Tor Header is as follows:

|  |  |
| --- | --- |
| Hop Count | |
| FP Flag 0 | Forward Address 0 |
| FP Flag 1 | Forward Address 1 |
| FP Flag 2 | Forward Address 2 |

Hop Count is a 1 byte integer.

Each Forward Packet (FP) Flag is 1 byte long.

Each Forward Address is 4 bytes long.

Therefore, the total length of the Tor Header is 19 bytes.

Each Forward Address is encrypted. ith Forward Address is encrypted with the session key of the ith node in the chain. Thus, no node can view the forward address meant for subsequent nodes. This ensures that each node is only aware of the immediate next hop address and not the final destination. Upon receiving a packet, the node views the contents of Tor Header. The 0th FP Flag determines whether to forward the packet or not.

FP Flag[0] = 0 indicates that the destination is reached. FP Flag[0] = 1 indicates that the destination is not yet reached and the remaining packet contents are to be forwarded *as is* to the address indicated by Forward Address[0].

At each hop, the 0th FP Flag and Forward Address are decrypted, utilized and then removed from the Tor Header. Thus, “peeling off” one layer of encryption of the “onion”. The remaining FP Flags and Forward Addresses are shifted one place above in the Tor Header. At the destination node, all FP Flags and Forward Addresses are found to be 0.

In the implementation, the topology is first defined. An adjacency matrix is randomly populated to indicate network traffic. The path with the minimum value of traffic is selected. The packet data is taken as input. The encrypted forward address, FP Flags and packet data are added to the packet. This indicates a ready “onion”. This packet is then sent to the entry node.

**C++ file**

**socket.cc**

#include <iostream>

#include <fstream>

#include <string>

#include <cassert>

#include <time.h>

#include "ns3/packet.h"

#include "ns3/netanim-module.h"

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

#include "ns3/point-to-point-module.h"

#include "ns3/ipv4-static-routing-helper.h"

#include "ns3/ipv4-list-routing-helper.h"

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("SocketBoundRoutingExample");

void SendStuff (Ptr<Socket> sock, Ipv4Address dstaddr, uint16\_t port, struct parameters packetParameters); //flag: 1 - Forward, 0 - Dummy

void BindSock (Ptr<Socket> sock, Ptr<NetDevice> netdev);

void socketRecv (Ptr<Socket> socket);

struct parameters{

uint8\_t flags[3];

uint8\_t hopCount;

uint32\_t addresses[3];

uint8\_t packetData[5];

};

uint16\_t port = 12345;

uint32\_t keys[3];

uint32\_t adjacencyMatrix[9][9] = {{0,1,1,1,0,0,0,0,0},

{0,0,0,0,1,1,1,0,0},

{0,0,0,0,1,1,1,0,0},

{0,0,0,0,1,1,1,0,0},

{0,0,0,0,0,0,0,1,1},

{0,0,0,0,0,0,0,1,1},

{0,0,0,0,0,0,0,1,1},

{0,0,0,0,0,0,0,0,0}

};

uint32\_t entryNode,intermediateNode,exitNode;

class TorHeader : public Header

{

public:

TypeId GetInstanceTypeId (void) const

{

static TypeId tid = TypeId ("TorHeader")

.SetParent<Header> ()

.AddConstructor<TorHeader> ()

;

return tid;

}

uint32\_t GetSerializedSize (void) const

{

return 16;

}

void Serialize (Buffer::Iterator start) const

{

start.WriteU8(hopCount);

uint32\_t i = 0;

for(i = 0; i < 3; ++i)

{

start.WriteU8(FPFlag[i]); //FP FLAG

start.WriteU32(forwardAddress[i]); //Forwarding Address

}

}

uint32\_t Deserialize (Buffer::Iterator start)

{

hopCount = start.ReadU8();

for(uint32\_t i = 0; i < 3; ++i)

{

FPFlag[i] = start.ReadU8();

forwardAddress[i] = start.ReadU32 ();

}

return 16; // the number of bytes consumed.

}

void SetFlag(uint8\_t flag, uint32\_t index)

{

FPFlag[index] = flag;

}

void SetForwardAddress(uint32\_t address, uint32\_t index)

{

forwardAddress[index] = address;

}

void SetHopCount(uint32\_t hop)

{

hopCount = hop;

}

void Print (std::ostream &os) const

{

os <<"Tor Header Contents:\n---------------------------------------------\nHop Count: " << (uint32\_t)hopCount << "\n";

for(uint32\_t i = 0; i < 3; ++i)

{

os << "\nFP Flag "<< i <<": "<< (uint32\_t)FPFlag[i]

<< "\tForward Address "<< i <<": " << (forwardAddress[i]) <<"\n";

}

os << "\n---------------------------------------------\n";

};

uint8\_t \* GetFlags()

{

return FPFlag;

}

uint32\_t \* GetForwardAddresses()

{

return forwardAddress;

}

uint8\_t GetHopCount()

{

return hopCount;

}

private:

uint8\_t FPFlag[3]; //1 - Forward

uint32\_t forwardAddress[3];

uint8\_t hopCount;

};

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

{

CommandLine cmd;

cmd.Parse (argc, argv);

Ptr<Node> nN0 = CreateObject<Node> ();

Ptr<Node> nN1 = CreateObject<Node> ();

Ptr<Node> nN2 = CreateObject<Node> ();

Ptr<Node> nN3 = CreateObject<Node> ();

Ptr<Node> nN4 = CreateObject<Node> ();

Ptr<Node> nN5 = CreateObject<Node> ();

Ptr<Node> nN6 = CreateObject<Node> ();

Ptr<Node> nN7 = CreateObject<Node> ();

Ptr<Node> nN8 = CreateObject<Node> ();

Ptr<Node> nN9 = CreateObject<Node> ();

NodeContainer c = NodeContainer (nN0, nN1, nN2, nN3, nN4);

c.Add(nN5);

c.Add(nN6);

c.Add(nN7);

c.Add(nN8);

c.Add(nN9);

InternetStackHelper internet;

internet.Install (c);

NodeContainer nN0N1 = NodeContainer (nN0, nN1);

NodeContainer nN0N2 = NodeContainer (nN0, nN2);

NodeContainer nN0N3 = NodeContainer (nN0, nN3);

NodeContainer nN1N4 = NodeContainer (nN1, nN4);

NodeContainer nN1N5 = NodeContainer (nN1, nN5);

NodeContainer nN1N6 = NodeContainer (nN1, nN6);

NodeContainer nN2N4 = NodeContainer (nN2, nN4);

NodeContainer nN2N5 = NodeContainer (nN2, nN5);

NodeContainer nN2N6 = NodeContainer (nN2, nN6);

NodeContainer nN3N4 = NodeContainer (nN3, nN4);

NodeContainer nN3N5 = NodeContainer (nN3, nN5);

NodeContainer nN3N6 = NodeContainer (nN3, nN6);

NodeContainer nN4N7 = NodeContainer (nN4, nN7);

NodeContainer nN4N8 = NodeContainer (nN4, nN8);

NodeContainer nN5N7 = NodeContainer (nN5, nN7);

NodeContainer nN5N8 = NodeContainer (nN5, nN8);

NodeContainer nN6N7 = NodeContainer (nN6, nN7);

NodeContainer nN6N8 = NodeContainer (nN6, nN8);

NodeContainer nN8N9 = NodeContainer (nN8, nN9);

NodeContainer nN7N9 = NodeContainer (nN7, nN9);

PointToPointHelper p2p;

p2p.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));

p2p.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer dN0dN1= p2p.Install(nN0N1);

NetDeviceContainer dN0dN2= p2p.Install(nN0N2);

NetDeviceContainer dN0dN3= p2p.Install(nN0N3);

NetDeviceContainer dN1dN4= p2p.Install(nN1N4);

NetDeviceContainer dN1dN5= p2p.Install(nN1N5);

NetDeviceContainer dN1dN6= p2p.Install(nN1N6);

NetDeviceContainer dN2dN4= p2p.Install(nN2N4);

NetDeviceContainer dN2dN5= p2p.Install(nN2N5);

NetDeviceContainer dN2dN6= p2p.Install(nN2N6);

NetDeviceContainer dN3dN4= p2p.Install(nN3N4);

NetDeviceContainer dN3dN5= p2p.Install(nN3N5);

NetDeviceContainer dN3dN6= p2p.Install(nN3N6);

NetDeviceContainer dN4dN7= p2p.Install(nN4N7);

NetDeviceContainer dN4dN8= p2p.Install(nN4N8);

NetDeviceContainer dN5dN7= p2p.Install(nN5N7);

NetDeviceContainer dN5dN8= p2p.Install(nN5N8);

NetDeviceContainer dN6dN7= p2p.Install(nN6N7);

NetDeviceContainer dN6dN8= p2p.Install(nN6N8);

NetDeviceContainer dN7dN9= p2p.Install(nN7N9);

NetDeviceContainer dN8dN9= p2p.Install(nN8N9);

Ipv4AddressHelper ipv4;

ipv4.SetBase ("10.1.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN0iN1 = ipv4.Assign (dN0dN1);

ipv4.SetBase ("20.2.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN0iN2 = ipv4.Assign (dN0dN2);

ipv4.SetBase ("30.3.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN0iN3 = ipv4.Assign (dN0dN3);

ipv4.SetBase ("78.6.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN1iN4 = ipv4.Assign (dN1dN4);

ipv4.SetBase ("67.3.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN1iN5 = ipv4.Assign (dN1dN5);

ipv4.SetBase ("42.1.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN1iN6 = ipv4.Assign (dN1dN6);

ipv4.SetBase ("52.10.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN2iN4 = ipv4.Assign (dN2dN4);

ipv4.SetBase ("13.1.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN2iN5 = ipv4.Assign (dN2dN5);

ipv4.SetBase ("18.6.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN2iN6 = ipv4.Assign (dN2dN6);

ipv4.SetBase ("8.9.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN3iN4 = ipv4.Assign (dN3dN4);

ipv4.SetBase ("19.4.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN3iN5 = ipv4.Assign (dN3dN5);

ipv4.SetBase ("17.8.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN3iN6 = ipv4.Assign (dN3dN6);

ipv4.SetBase ("11.4.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN4iN7 = ipv4.Assign (dN4dN7);

ipv4.SetBase ("25.3.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN4iN8 = ipv4.Assign (dN4dN8);

ipv4.SetBase ("53.4.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN5iN7 = ipv4.Assign (dN5dN7);

ipv4.SetBase ("23.9.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN5iN8 = ipv4.Assign (dN5dN8);

ipv4.SetBase ("52.4.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN6iN7 = ipv4.Assign (dN6dN7);

ipv4.SetBase ("67.9.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN6iN8 = ipv4.Assign (dN6dN8);

ipv4.SetBase ("27.4.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN7iN9 = ipv4.Assign (dN7dN9);

ipv4.SetBase ("29.1.2.0", "255.255.255.0");

Ipv4InterfaceContainer iN8iN9 = ipv4.Assign (dN8dN9);

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

Ipv4Address addresses[10];

addresses[0] = nN0->GetObject<Ipv4>()->GetAddress(1, 0).GetLocal();

addresses[1] = nN1->GetObject<Ipv4>()->GetAddress(1, 0).GetLocal();

addresses[2] = nN2->GetObject<Ipv4>()->GetAddress(1, 0).GetLocal();

addresses[3] = nN3->GetObject<Ipv4>()->GetAddress(1, 0).GetLocal();

addresses[4] = nN4->GetObject<Ipv4>()->GetAddress(1, 0).GetLocal();

addresses[5] = nN5->GetObject<Ipv4>()->GetAddress(1, 0).GetLocal();

addresses[6] = nN6->GetObject<Ipv4>()->GetAddress(1, 0).GetLocal();

addresses[7] = nN7->GetObject<Ipv4>()->GetAddress(1, 0).GetLocal();

addresses[8] = nN8->GetObject<Ipv4>()->GetAddress(1, 0).GetLocal();

addresses[9] = nN9->GetObject<Ipv4>()->GetAddress(1, 0).GetLocal();

Ptr<Socket> nN0Socket = Socket::CreateSocket (nN0, TypeId::LookupByName ("ns3::UdpSocketFactory"));

InetSocketAddress nN0SocketAddr = InetSocketAddress (addresses[0], port);

nN0Socket->Bind (nN0SocketAddr);

nN0Socket->SetRecvCallback (MakeCallback (&socketRecv));

Ptr<Socket> nN1Socket = Socket::CreateSocket (nN1, TypeId::LookupByName ("ns3::UdpSocketFactory"));

InetSocketAddress nN1SocketAddr = InetSocketAddress (addresses[1], port);

nN1Socket->Bind (nN1SocketAddr);

nN1Socket->SetRecvCallback (MakeCallback (&socketRecv));

Ptr<Socket> nN2Socket = Socket::CreateSocket (nN2, TypeId::LookupByName ("ns3::UdpSocketFactory"));

InetSocketAddress nN2SocketAddr = InetSocketAddress (addresses[2], port);

nN2Socket->Bind (nN2SocketAddr);

nN2Socket->SetRecvCallback (MakeCallback (&socketRecv));

Ptr<Socket> nN3Socket = Socket::CreateSocket (nN3, TypeId::LookupByName ("ns3::UdpSocketFactory"));

InetSocketAddress nN3SocketAddr = InetSocketAddress (addresses[3], port);

nN3Socket->Bind (nN3SocketAddr);

nN3Socket->SetRecvCallback (MakeCallback (&socketRecv));

Ptr<Socket> nN4Socket = Socket::CreateSocket (nN4, TypeId::LookupByName ("ns3::UdpSocketFactory"));

InetSocketAddress nN4SocketAddr = InetSocketAddress (addresses[4], port);

nN4Socket->Bind (nN4SocketAddr);

nN4Socket->SetRecvCallback (MakeCallback (&socketRecv));

Ptr<Socket> nN5Socket = Socket::CreateSocket (nN5, TypeId::LookupByName ("ns3::UdpSocketFactory"));

InetSocketAddress nN5SocketAddr = InetSocketAddress (addresses[5], port);

nN5Socket->Bind (nN5SocketAddr);

nN5Socket->SetRecvCallback (MakeCallback (&socketRecv));

Ptr<Socket> nN6Socket = Socket::CreateSocket (nN6, TypeId::LookupByName ("ns3::UdpSocketFactory"));

InetSocketAddress nN6SocketAddr = InetSocketAddress (addresses[6], port);

nN6Socket->Bind (nN6SocketAddr);

nN6Socket->SetRecvCallback (MakeCallback (&socketRecv));

Ptr<Socket> nN7Socket = Socket::CreateSocket (nN7, TypeId::LookupByName ("ns3::UdpSocketFactory"));

InetSocketAddress nN7SocketAddr = InetSocketAddress (addresses[7], port);

nN7Socket->Bind (nN7SocketAddr);

nN7Socket->SetRecvCallback (MakeCallback (&socketRecv));

Ptr<Socket> nN8Socket = Socket::CreateSocket (nN8, TypeId::LookupByName ("ns3::UdpSocketFactory"));

InetSocketAddress nN8SocketAddr = InetSocketAddress (addresses[8], port);

nN8Socket->Bind (nN8SocketAddr);

nN8Socket->SetRecvCallback (MakeCallback (&socketRecv));

Ptr<Socket> nN9Socket = Socket::CreateSocket (nN9, TypeId::LookupByName ("ns3::UdpSocketFactory"));

InetSocketAddress nN9SocketAddr = InetSocketAddress (addresses[9], port);

nN9Socket->Bind (nN9SocketAddr);

nN9Socket->SetRecvCallback (MakeCallback (&socketRecv));

AsciiTraceHelper ascii;

p2p.EnableAsciiAll (ascii.CreateFileStream ("socket-bound-static-routing.tr"));

p2p.EnablePcapAll ("socket-bound-static-routing");

//LogComponentEnableAll (LOG\_PREFIX\_TIME);

LogComponentEnable ("SocketBoundRoutingExample", LOG\_LEVEL\_INFO);

srand(time(NULL));

struct parameters packetParameters;

uint32\_t i = 0,minimum=99;

int32\_t j;

uint32\_t pathLength = 3;

uint32\_t path[4];

path[0]=0;

for(i=0;i<8;i++){

for(j=0;j<9;j++){

if(adjacencyMatrix[i][j]==0){

adjacencyMatrix[i][j]=99;

}

else if(adjacencyMatrix[i][j]==1){

adjacencyMatrix[i][j]=rand()%10;

}

}

}

printf("Adjacency Matrix:\n");

printf("\n\t");

for(i = 0; i < 9; ++i){

printf("N%d\t", i);

}

printf("\n");

for(i=0;i<8;i++){

for(j=-1;j<9;j++){

if(j != -1){

if(adjacencyMatrix[i][j] == 99)

printf("-\t");

else

printf("%d\t",adjacencyMatrix[i][j]);

}

else{

printf("N%d \t", i);

}

}

printf("\n");

}

for(i=0;i<pathLength;i++){

minimum=99;

for(j=0;j<9;j++){

if(adjacencyMatrix[path[i]][j]< minimum){

minimum=adjacencyMatrix[path[i]][j];

path[i+1]=j;

}

}

}

printf("\nIP Address:\n");

for(uint32\_t k = 0; k < 10; k++){

NS\_LOG\_INFO("N" << k << "\t" << addresses[k]);

}

printf("\nPath:\n");

printf("N%d -> N%d -> N%d\n", path[1],path[2],path[3]);

for(i = 0; i < pathLength; ++i)

{

keys[i] = (rand() % 100000000000) + 1;

packetParameters.flags[i] = 1;

}

packetParameters.addresses[0] = addresses[path[2]].Get() - keys[0];

packetParameters.addresses[1] = addresses[path[3]].Get() - keys[1];

packetParameters.addresses[2] = addresses[9].Get() - keys[2];

printf("\nEnter Packet Data to be sent (Upto 5 characters):\n");

std::cin >> packetParameters.packetData;

for(i = 0; i < 5; ++i){

packetParameters.packetData[i] -= (keys[0] + keys[1] + keys[2]);

}

packetParameters.packetData[5] = '\0';

packetParameters.hopCount = 0;

Simulator::Schedule (Seconds (1), &SendStuff, nN0Socket, addresses[path[1]], port, packetParameters);

AnimationInterface anim("socket.xml");

anim.SetConstantPosition(nN0, 3, 7);

anim.SetConstantPosition(nN1, 2, 5);

anim.SetConstantPosition(nN2, 3, 5);

anim.SetConstantPosition(nN3, 4, 5);

anim.SetConstantPosition(nN4, 2, 3);

anim.SetConstantPosition(nN5, 3, 3);

anim.SetConstantPosition(nN6, 4, 3);

anim.SetConstantPosition(nN7, 2, 1);

anim.SetConstantPosition(nN8, 4, 1);

anim.SetConstantPosition(nN9, 3, 0);

p2p.EnablePcapAll ("bhaiyya");

Simulator::Run ();

Simulator::Destroy ();

return 0;

}

void SendStuff (Ptr<Socket> sock, Ipv4Address dstaddr, uint16\_t port, struct parameters packetParameters)

{

uint32\_t size = 5;

uint32\_t i = 0;

Address addr;

sock->GetSockName (addr);

InetSocketAddress iaddr = InetSocketAddress::ConvertFrom (addr);

Packet p1(packetParameters.packetData, size);

Ptr<Packet> p = p1.Copy();

TorHeader torHeader;

for(i = 0; i < 3; ++i)

{

torHeader.SetFlag(packetParameters.flags[i], i);

if(packetParameters.flags[i] == 1)

torHeader.SetForwardAddress(packetParameters.addresses[i], i);

else

torHeader.SetForwardAddress(0, i);

}

torHeader.SetHopCount(packetParameters.hopCount);

p->AddHeader(torHeader);

NS\_LOG\_INFO("\n" << iaddr.GetIpv4() << " Sending packet to: " << dstaddr);

sock->SendTo (p, 0, InetSocketAddress (dstaddr,port));

return;

}

void socketRecv (Ptr<Socket> socket)

{

Address from;

uint32\_t size;

uint8\_t \* tempFlags;

uint32\_t \* tempAddr;

uint32\_t i = 0;

struct parameters packetParameters;

Ptr<Packet> packet = socket->RecvFrom (from);

packet->RemoveAllPacketTags ();

packet->RemoveAllByteTags ();

NS\_LOG\_INFO ("" << "Received " << packet->GetSize () << " bytes from " << InetSocketAddress::ConvertFrom (from).GetIpv4 ());

TorHeader receivedTorHeader;

packet->RemoveHeader(receivedTorHeader);

NS\_LOG\_INFO("" << receivedTorHeader);

size = packet->GetSize();

packet->CopyData(packetParameters.packetData, size);

NS\_LOG\_INFO ("" << "Packet data: " << packetParameters.packetData);

for(i = 0; i < 5; ++i){

packetParameters.packetData[i] += keys[receivedTorHeader.GetHopCount()];

}

tempFlags = receivedTorHeader.GetFlags();

tempAddr = receivedTorHeader.GetForwardAddresses();

NS\_LOG\_INFO("" << "Decrypted Forward Address: " << Ipv4Address(tempAddr[0] + keys[receivedTorHeader.GetHopCount()]));

if(tempFlags[0] == 1)

{

for(i = 0; i < 2; ++i)

{

packetParameters.flags[i] = tempFlags[i + 1];

packetParameters.addresses[i] = tempAddr[i + 1];

}

packetParameters.flags[2] = 0;

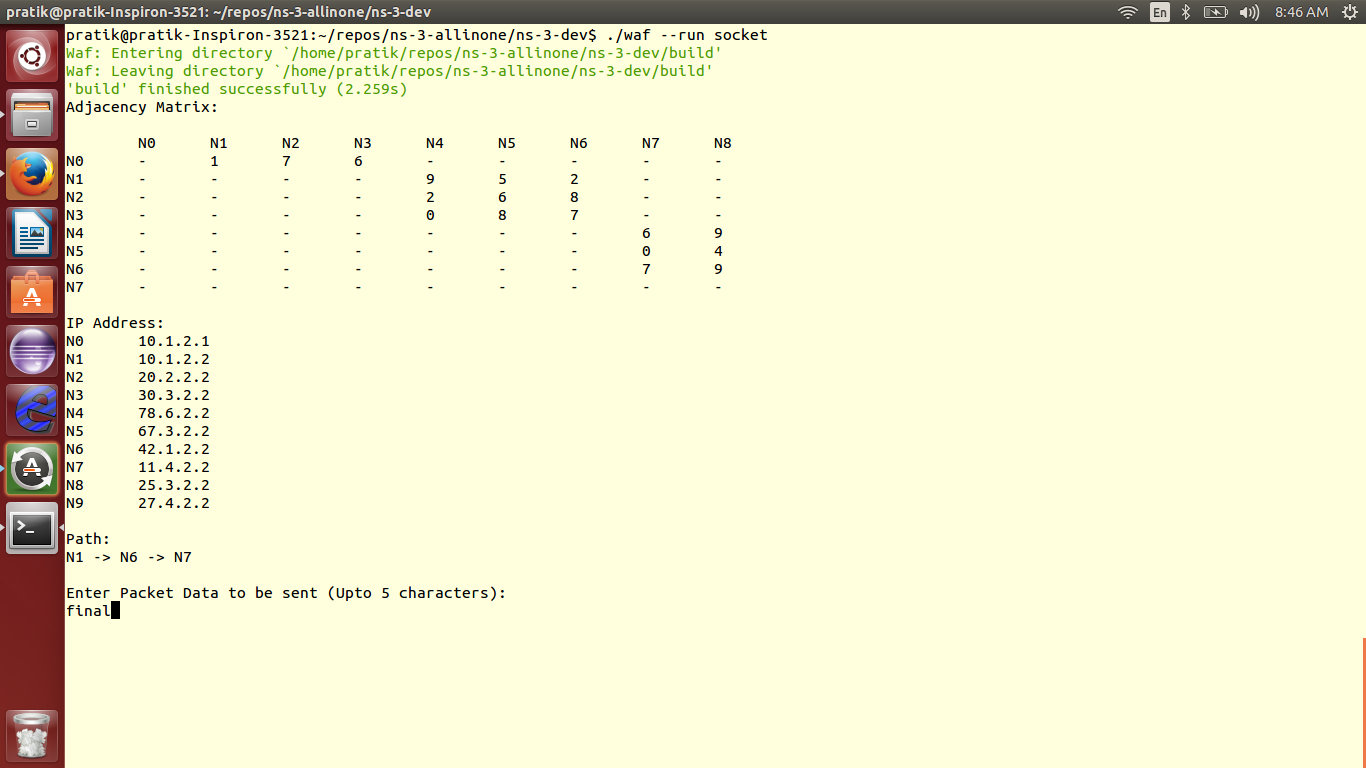
packetParameters.addresses[2] = 0;

packetParameters.hopCount = receivedTorHeader.GetHopCount() + 1;

SendStuff (socket, Ipv4Address (tempAddr[0] + keys[receivedTorHeader.GetHopCount()]), port, packetParameters);

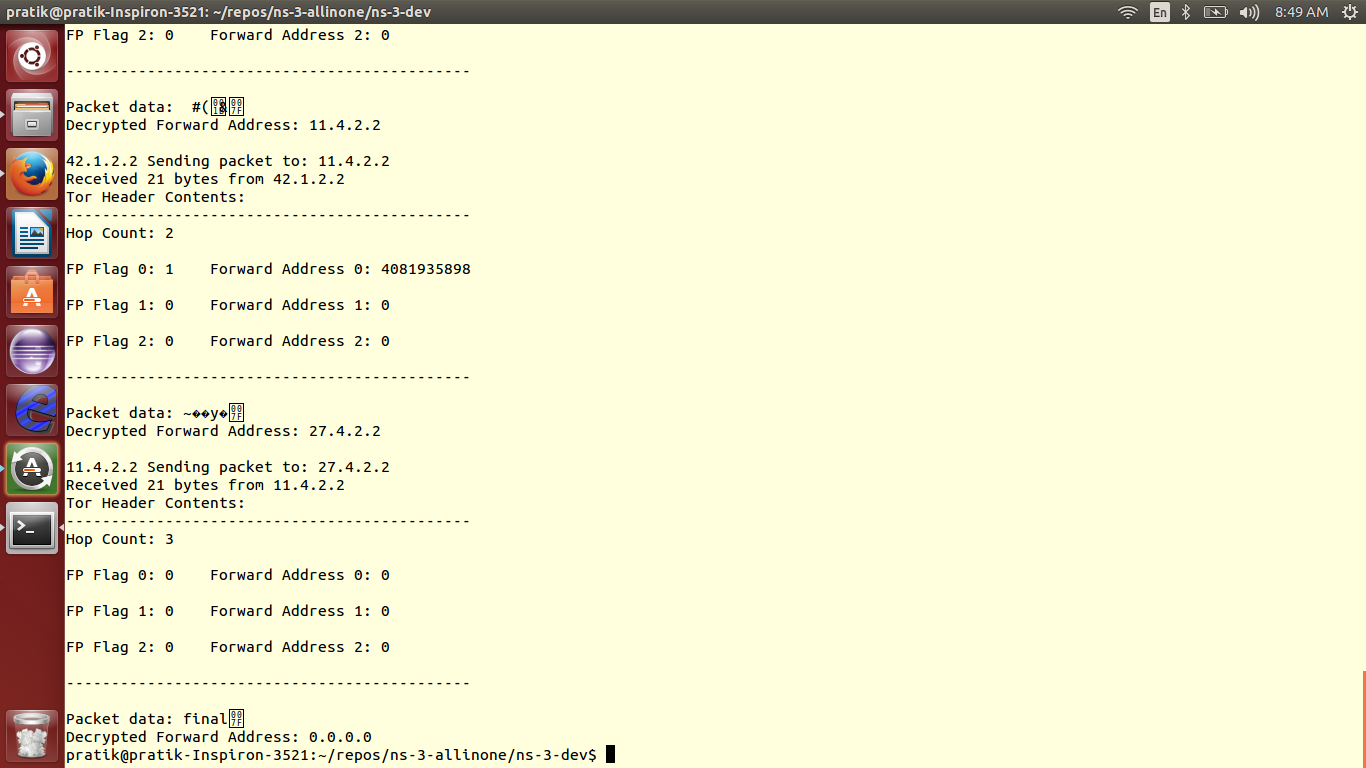
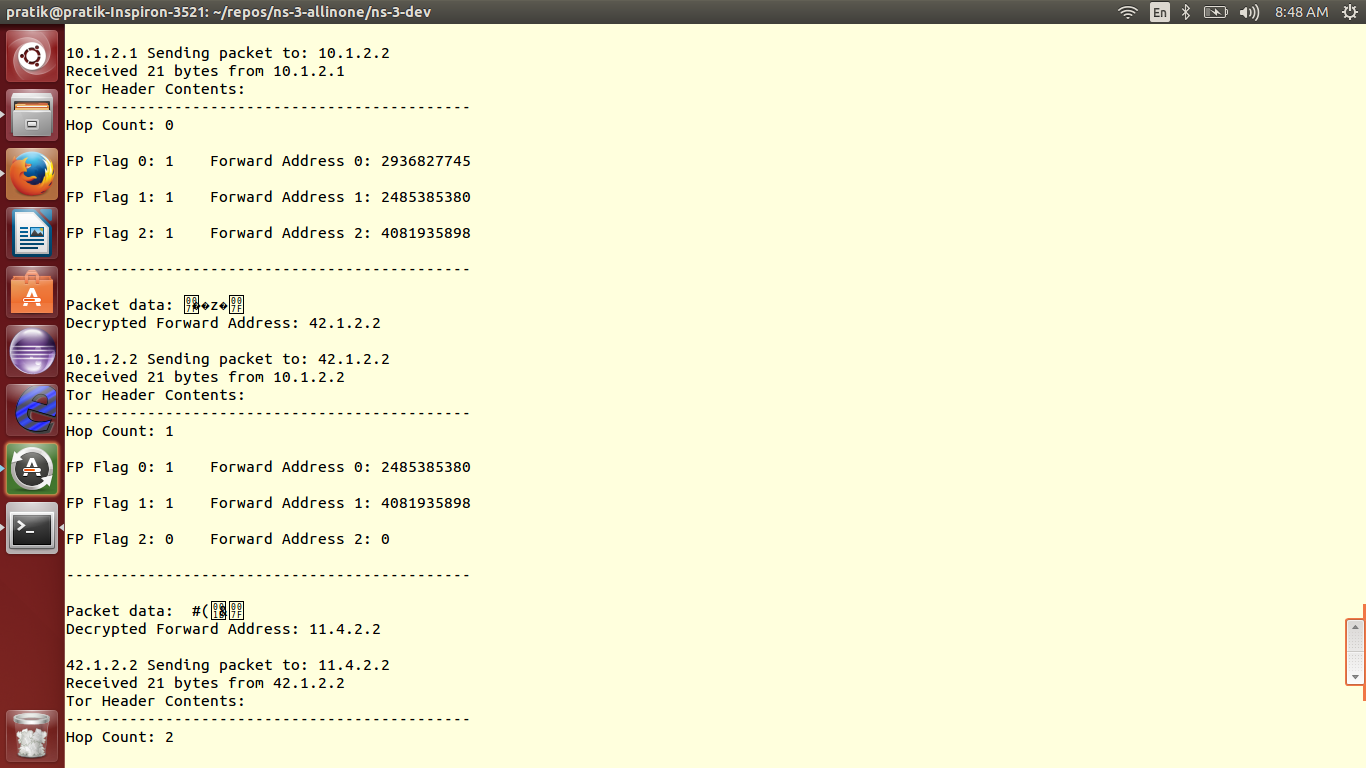
}

}

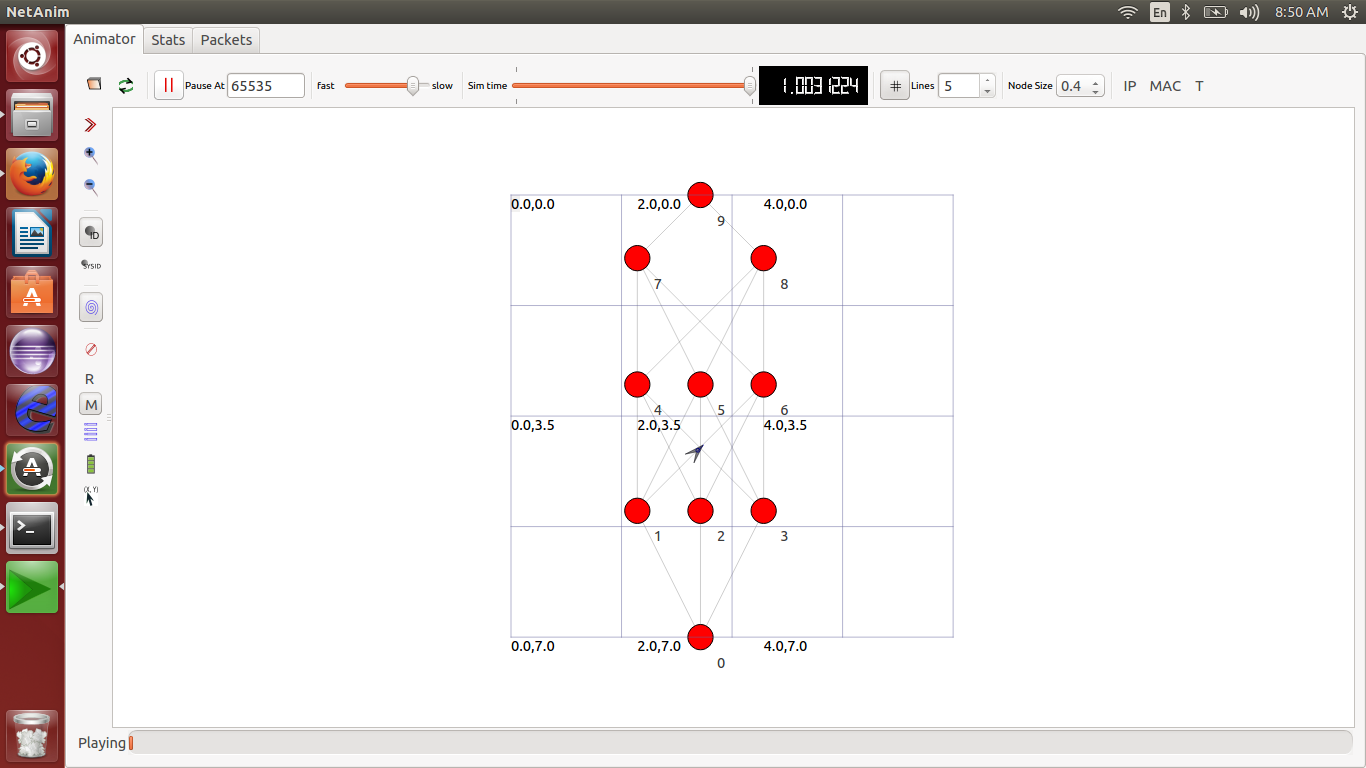


Here the adjacency matrix for the connected nodes is shown. The cost values are determined using random function and the path is decided based on on which path costs the lowest. Here, the path choosen is N1 -> N6-> N7

The message to be passed is also taken as input from the user



Here how the message is passed across the TOR nodes is shown, the encrypted message is decrypted at each stage and the encrypted is shown as well

Here 

The topology for the TOR network is shown in the above figure

**References:**

1. <http://www.onion-router.net/>
2. <http://en.wikipedia.org/wiki/Onion_routing>
3. <http://en.wikipedia.org/wiki/File:Onion_diagram.svg>
4. <http://en.wikipedia.org/wiki/Ns_(simulator)>
5. <https://www.nsnam.org/wiki/>
6. <https://www.nsnam.org/doxygen/index.html>
7. <https://www.torproject.org/about/overview>
8. <https://www.torproject.org/docs/documentation.html.en>
9. <http://cs.brown.edu/~anna/papers/cl05.pdf>
10. <http://ntrg.cs.tcd.ie/undergrad/4ba2.05/group10/>
11. <http://ijates.com/images/short_pdf/1401971204_P26-36.pdf>
12. <http://www.onion-router.net/Publications/CACM-1999.pdf>