

SPTM - Dizajn i implementacija TOR modela u NS-3 simulatoru

Generated by Doxygen 1.9.8

1 Namespace Index	3
1.1 Namespace List	3
2 Class Index	5
2.1 Class List	5
3 File Index	7
3.1 File List	7
4 Namespace Documentation	9
4.1 ns3 Namespace Reference	9
4.1.1 Function Documentation	9
4.1.1.1 NS_LOG_COMPONENT_DEFINE() [1/2]	9
4.1.1.2 NS_LOG_COMPONENT_DEFINE() [2/2]	9
4.1.1.3 NS_OBJECT_ENSURE_REGISTERED() [1/2]	9
4.1.1.4 NS_OBJECT_ENSURE_REGISTERED() [2/2]	9
5 Class Documentation	11
5.1 PacketTrace Struct Reference	11
5.1.1 Detailed Description	11
5.1.2 Member Data Documentation	11
5.1.2.1 path	11
5.1.2.2 sendTime	11
6 File Documentation	13
6.1 TOR.cc File Reference	13
6.1.1 Detailed Description	14
6.1.2 Function Documentation	14
6.1.2.1 main()	14
6.1.2.2 NS_LOG_COMPONENT_DEFINE()	17
6.1.2.3 Ratio()	17
6.1.2.4 ReceivedPacket()	17
6.1.2.5 SentPacket()	18
6.1.3 Variable Documentation	19
6.1.3.1 g_firstPacket	19
6.1.3.2 g_firstPacketTime	19
6.1.3.3 g_lastPacketTime	19
6.1.3.4 m_bytes_received	20
6.1.3.5 m_bytes_sent	20
6.1.3.6 m_delayTable	20
6.1.3.7 m_packets_received	20
6.1.3.8 m_packets_sent	20
6.1.3.9 m_time	20
6.1.3.10 nodeNames	21

6.1.3.11 packetCount	21
6.1.3.12 PacketStartTimes	21
6.1.3.13 packetTracker	21
6.1.3.14 totalDelay	21
6.2 TOR.cc	22
6.3 udp-echo-client.cc File Reference	25
6.3.1 Detailed Description	25
6.4 udp-echo-client.cc	25
6.5 udp-echo-server.cc File Reference	31
6.5.1 Detailed Description	31
6.6 udp-echo-server.cc	31
Index	35

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

ns3	9
---------------------	-------	-------------------

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

PacketTrace	11
---------------------------------------	----

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

TOR.cc	This code will allow transfer of packets, udp-echo-client.cc located in src/applications/model allows sending of packets with string passed as data, while udp-echo-server.cc in the same directory allows the packets to be received	13
udp-echo-client.cc	This file allows the sending of packets in TOR.cc which is located in the scratch folder	25
udp-echo-server.cc	This file enables the process of receiving packets in TOR.cc file located in the ../../scratch folder	31

Chapter 4

Namespace Documentation

4.1 ns3 Namespace Reference

Functions

- [NS_LOG_COMPONENT_DEFINE](#) ("UdpEchoClientApplication")
- [NS_OBJECT_ENSURE_REGISTERED](#) (UdpEchoClient)
- [NS_LOG_COMPONENT_DEFINE](#) ("UdpEchoServerApplication")
- [NS_OBJECT_ENSURE_REGISTERED](#) (UdpEchoServer)

4.1.1 Function Documentation

4.1.1.1 NS_LOG_COMPONENT_DEFINE() [1/2]

```
ns3::NS_LOG_COMPONENT_DEFINE (
    "UdpEchoClientApplication" )
```

4.1.1.2 NS_LOG_COMPONENT_DEFINE() [2/2]

```
ns3::NS_LOG_COMPONENT_DEFINE (
    "UdpEchoServerApplication" )
```

4.1.1.3 NS_OBJECT_ENSURE_REGISTERED() [1/2]

```
ns3::NS_OBJECT_ENSURE_REGISTERED (
    UdpEchoClient )
```

4.1.1.4 NS_OBJECT_ENSURE_REGISTERED() [2/2]

```
ns3::NS_OBJECT_ENSURE_REGISTERED (
    UdpEchoServer )
```


Chapter 5

Class Documentation

5.1 PacketTrace Struct Reference

Public Attributes

- double [sendTime](#)
- std::string [path](#)

5.1.1 Detailed Description

Definition at line [24](#) of file [TOR.cc](#).

5.1.2 Member Data Documentation

5.1.2.1 path

```
std::string PacketTrace::path
```

Definition at line [26](#) of file [TOR.cc](#).

5.1.2.2 sendTime

```
double PacketTrace::sendTime
```

Definition at line [25](#) of file [TOR.cc](#).

The documentation for this struct was generated from the following file:

- [TOR.cc](#)

Chapter 6

File Documentation

6.1 TOR.cc File Reference

This code will allow transfer of packets, [udp-echo-client.cc](#) located in src/applications/model allows sending of packets with string passed as data, while [udp-echo-server.cc](#) in the same directory allows the packets to be received.

```
#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/ipv4-global-routing-helper.h"
#include "ns3/netanim-module.h"
#include "ns3/mobility-module.h"
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
```

Classes

- struct [PacketTrace](#)

Functions

- [NS_LOG_COMPONENT_DEFINE](#) ("SimpleTOR")
- static void [SentPacket](#) (Ptr< const Packet > p)
This is a function which handles the sending of packets.
- static void [ReceivedPacket](#) (Ptr< const Packet > p)
This is a function which handles the process of receiving packets.
- void [Ratio](#) ()
This function allows the tor network statistic to be printed out, which prints all relevant data about the network simulation.
- int [main](#) (int argc, char *argv[])
This is the main function which gets called once the simulation starts.

Variables

- `std::map< uint32_t, PacketTrace > packetTracker`
- `std::vector< std::string > nodeNames = {"Client", "Entry", "Relay1", "Relay2", "Relay3", "Exit", "Destination"}`
Here, we are declaring node names for all nodes in the network topology.
- `static Time g_firstPacketTime = Seconds(0.0)`
This is the variable which gets assigned a value of the time when the first packet was transmitted.
- `static Time g_lastPacketTime = Seconds(0.0)`
This is the variable which gets assigned a value of the time when the last packet was transmitted.
- `static bool g_firstPacket = true`
This is a boolean value for the first packet which can be true or false.
- `static std::map< uint32_t, double > PacketStartTimes`
- `static double totalDelay = 0.0`
This is a variable, type double, which means it stores decimal values, and it stores the value of the total delay of the network.
- `static int packetCount = 0`
- `uint32_t m_bytes_sent = 0`
This is an unsigned variable type, which means that it cannot have negative values, and the range for the numbers it can have is from 0 to 2^{32} . This variable stores the amount of sent bytes.
- `uint32_t m_bytes_received = 0`
This variable stores the amount of received bytes.
- `uint32_t m_packets_sent = 0`
This variable stores the amount of sent packets.
- `uint32_t m_packets_received = 0`
This variable stores the amount of received packets.
- `double m_time = 0`
- `std::map< uint32_t, double > m_delayTable`

6.1.1 Detailed Description

This code will allow transfer of packets, [udp-echo-client.cc](#) located in `src/applications/model` allows sending of packets with string passed as data, while [udp-echo-server.cc](#) in the same directory allows the packets to be received.

Definition in file [TOR.cc](#).

6.1.2 Function Documentation

6.1.2.1 main()

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

This is the main function which gets called once the simulation starts.

If `output.txt` exists in the directory where the simulation is started, it will be replaced with `output.txt` with no content. `std::ofstream output_file("output.txt"); // Replace output_txt if it was created before.`

Simulation time is assigned to this variable.

This variable stores the maximum amount of packets.

This allows the user to specify parameters which could be changed while running the simulation, such as: `./ns3 run scratch/TOR.cc --maxPackets=5 -simulationTime=30`

Node container is specified.

7 nodes are being created.

Data rate parameter is being assigned to the point-to-point link.

Delay parameter is being assigned to the point-to-point link.

This holds a collection of network devices, such as wifi or ethernet devices.

This holds IPv4 addresses of network devices.

This installs a network stack on nodes.

This assigns IPv4 addresses to network devices.

This creates a P2P link between nodes 0 and 1.

This specifies the initialisation time of the server.

This specifies the time when the server stops responding.

This creates and allocates a mobility model and installs it for the nodes.

This will create an xml file which can later be viewed in NetAnim which provides network visualisation.

This sets the maximum amounts of packets per trace file.

This specifies node descriptions.

Apart from this, we choose node colors.

Network tracing is enabled here.

This allows the simulation to start.

This frees up resources which were allocated to the simulation while it was running.

If 0 is returned, that means that code execution was successful.

Definition at line 220 of file TOR.cc.

```
00220                                     {
00225         // LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
00226
00227 Config::SetDefault ("ns3::Ipv4GlobalRouting::RespondToInterfaceEvents", BooleanValue(true));
00231
00232         double simulationTime = 20; // 20 seconds.
00236         double maxPackets = 10; // 10 packets.
00237
00238         Packet::EnablePrinting();
00239         PacketMetadata::Enable();
00240
00244         CommandLine cmd;
00245         cmd.AddValue ("simulationTime", "simulationTime", simulationTime);
00246         cmd.AddValue ("maxPackets", "maxPackets", maxPackets);
00247         cmd.Parse (argc, argv);
00248
00249         Time::SetResolution (Time::NS);
00250         //LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_ALL);
00251         //LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_ALL);
00252         LogComponentEnable ("SimpleTOR", LOG_LEVEL_ALL);
00253
00257         NodeContainer nodes;
00261         nodes.Create(7);
```

```

00262
00263 //Point to Point links
00264 PointToPointHelper pointToPoint;
00268 pointToPoint.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
00272 pointToPoint.SetChannelAttribute("Delay", StringValue("25ms"));
00273
00277 NetDeviceContainer devices[6];
00281 Ipv4InterfaceContainer interfaces[6];
00282
00286 InternetStackHelper stack;
00287 stack.Install(nodes);
00288
00292 Ipv4AddressHelper address;
00293
00297 address.SetBase("10.1.1.0", "255.255.255.0");
00298 interfaces[0] = address.Assign(pointToPoint.Install(nodes.Get(0), nodes.Get(1)));
00299
00300 address.SetBase("10.1.2.0", "255.255.255.0");
00301 interfaces[1] = address.Assign(pointToPoint.Install(nodes.Get(1), nodes.Get(2)));
00302
00303 address.SetBase("10.1.3.0", "255.255.255.0");
00304 interfaces[2] = address.Assign(pointToPoint.Install(nodes.Get(2), nodes.Get(3)));
00305
00306 address.SetBase("10.1.4.0", "255.255.255.0");
00307 interfaces[3] = address.Assign(pointToPoint.Install(nodes.Get(3), nodes.Get(4)));
00308
00309 address.SetBase("10.1.5.0", "255.255.255.0");
00310 interfaces[4] = address.Assign(pointToPoint.Install(nodes.Get(4), nodes.Get(5)));
00311
00312 address.SetBase("10.1.6.0", "255.255.255.0");
00313 interfaces[5] = address.Assign(pointToPoint.Install(nodes.Get(5), nodes.Get(6)));
00314
00315 Ipv4GlobalRoutingHelper::PopulateRoutingTables();
00316
00317 UdpEchoServerHelper echoServer(9);
00318 ApplicationContainer serverApp = echoServer.Install(nodes.Get(6));
00319
00323 serverApp.Start(Seconds(1.0));
00327 serverApp.Stop(Seconds(simulationTime));
00328
00329 UdpEchoClientHelper echoClient(interfaces[5].GetAddress(1), 9);
00330 echoClient.SetAttribute("MaxPackets", UintegerValue(maxPackets));
00331 echoClient.SetAttribute("Interval", TimeValue(Seconds(0.1)));
00332
00333 ApplicationContainer clientApp = echoClient.Install(nodes.Get(0));
00334 clientApp.Start(Seconds(1.0));
00335 clientApp.Stop(Seconds(simulationTime));
00336
00337 // Connect trace sources for packet tracking
00338 Config::ConnectWithoutContext ("/NodeList/*/ApplicationList/*/ns3::UdpEchoClient/Tx",
MakeCallback(&SentPacket));
00339 Config::ConnectWithoutContext ("/NodeList/*/ApplicationList/*/ns3::UdpEchoServer/Rx",
MakeCallback(&ReceivedPacket));
00340
00341 // Mobility Setup
00342 MobilityHelper mobility;
00346 mobility.SetPositionAllocator("ns3::GridPositionAllocator",
00347                               "MinX", DoubleValue(50.0),
00348                               "MinY", DoubleValue(80.0),
00349                               "DeltaX", DoubleValue(60.0),
00350                               "DeltaY", DoubleValue(70.0),
00351                               "GridWidth", UintegerValue(4),
00352                               "LayoutType", StringValue("RowFirst"));
00353
00354 mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
00355 mobility.Install(nodes);
00356
00357 // NetAnim
00361 AnimationInterface anim("TOR.xml");
00365 anim.SetMaxPktsPerTraceFile(5000);
00372 anim.UpdateNodeDescription(0, "Client");
00373 anim.UpdateNodeDescription(1, "Entry Guard");
00374 anim.UpdateNodeDescription(2, "Relay 1");
00375 anim.UpdateNodeDescription(3, "Relay 2");
00376 anim.UpdateNodeDescription(4, "Relay 3");
00377 anim.UpdateNodeDescription(5, "Exit");
00378 anim.UpdateNodeDescription(6, "Destination");
00379
00380 anim.UpdateNodeColor(0, 255, 0, 0); // Red for Client
00381 anim.UpdateNodeColor(1, 0, 255, 0); // Green for Entry Guard
00382 anim.UpdateNodeColor(2, 0, 0, 255); // Blue for Relay 1
00383 anim.UpdateNodeColor(3, 255, 255, 0); // Yellow for Relay 2
00384 anim.UpdateNodeColor(4, 255, 0, 255); // Purple for Relay 3
00385 anim.UpdateNodeColor(5, 0, 255, 255); // Cyan for Exit
00386 anim.UpdateNodeColor(6, 128, 128, 128); // Gray for Destination
00387 ;
00388

```

```

00392     pointToPoint.EnablePcapAll("tor_packet_trace");
00393
00394     Simulator::Schedule(Seconds(simulationTime), &Ratio);
00395
00399     Simulator::Run();
00403     Simulator::Destroy();
00404
00408     return 0;
00409 }

```

6.1.2.2 NS_LOG_COMPONENT_DEFINE()

```

NS_LOG_COMPONENT_DEFINE (
    "SimpleTOR" )

```

6.1.2.3 Ratio()

```

void Ratio ( )

```

This function allows the tor network statistic to be printed out, which prints all relevant data about the network simulation.

Definition at line 184 of file TOR.cc.

```

00184     {
00185
00186         std::cout << "\n=== TOR network statistics ===\n" << std::endl;
00187         std::cout << "Transmission summary:" << std::endl;
00188         std::cout << "-----" << std::endl;
00189         std::cout << "Total bytes sent:\t " << m_bytes_sent << std::endl;
00190         std::cout << "Total bytes received:\t " << m_bytes_received << std::endl;
00191         std::cout << "Total packets sent:\t " << m_packets_sent << std::endl;
00192         std::cout << "Total packets received:\t " << m_packets_received << std::endl;
00193         std::cout << "Delivery ratio (bytes):\t " << (float)m_bytes_received/(float)m_bytes_sent * 100 <<
00194         "%\n" << std::endl;
00195         std::cout << "Delivery ratio (packets): " << (float)m_packets_received/(float)m_packets_sent * 100 <<
00196         "%\n" << std::endl;
00197
00198         double duration = Simulator::Now().GetSeconds();
00199         double throughputBps = (m_bytes_received * 8.0) / duration;
00200         if (duration > 0){
00201             std::cout << "Troughput (bps):\t " << throughputBps << " bps " << std::endl;
00202             std::cout << "Troughput (kbps):\t " << throughputBps/1000.0 << " kbps " << std::endl;
00203         }
00204         if (packetCount > 0) {
00205             std::cout << "Average end-to-end delay: " << totalDelay/packetCount << "s" << std::endl;
00206         }
00207         std::cout << "-----" << std::endl;
00208
00209         std::cout << "Created output file: output.txt" << std::endl;
00210         std::cout << "-----" << std::endl;
00211
00212     }
00213 }
00214
00215 }

```

6.1.2.4 ReceivedPacket()

```

static void ReceivedPacket (
    Ptr< const Packet > p ) [static]

```

This is a function which handles the process of receiving packets.

This will allow an output file to be created, also, data can be appended to this file because of `std::ios::app` which is included. If we omit that, we would keep creating the file without appending data to it, which wouldn't be a solution

because we need to track the data for every packet so that we can eventually use gnuplot to plot the captured data, not just the final data for the final packet which was sent.

This variable stores the time when the packet was received.

This variable stores the time when the packet was sent.

This variable stores the packet delay which is calculated by subtracting start time from the end time.

This represents the time when the packet was received.

This variable stores the throughput value, which is calculated by multiplying received bytes by 8 and dividing that by the duration. The value for the throughput is displayed in bits per second.

Output file gets created with contents of:

- Duration;
- Sent packets;
- Received packets;
- Throughput in bits per second;
- Packet delay.

This allows the output file to be closed after writing.

This message prints the time when the packet was received.

Definition at line 116 of file [TOR.cc](#).

```
00116 {
00120     std::ofstream output_file("output.txt", std::ios::app); // This will create and open the file and
        append data to it.
00121
00122     m_bytes_received += p->GetSize();
00123     m_packets_received++;
00124
00125     /*
00126     //HELP LINES USED FOR TESTING
00127     std::cout << "\n .....ReceivedPacket....." << p->GetUid() << "..." << p->GetSize() <<
        "..... \n";
00128     p->Print(std::cout);
00129     std::cout << "\n ..... \n";
00130     */
00131
00135     double endTime = Simulator::Now().GetSeconds();
00139     double startTime = PacketStartTimes[p->GetUid()];
00143     double packetDelay = endTime - startTime;
00144
00145     //Ptr<Packet> packetCopy = p->Copy();
00146     //DecryptPacket (packetCopy);
00147
00148     totalDelay += packetDelay;
00149     packetCount++;
00153     double duration = Simulator::Now().GetSeconds();
00157     double throughputBps = (m_bytes_received * 8.0) / duration;
00158     //double averageDelay = totalDelay/packetCount;
00168     output_file << duration << " " << m_packets_sent << " " << m_packets_received << " " << throughputBps
        << " " << packetDelay << std::endl; // This will create an output file with: duration, sent packets,
        received packets, throughput and the packet delay with spaces between them.
00172     output_file.close(); // This closes the output file after writing.
00173
00177     std::cout << "\nPacket " << p->GetUid()+1 << " received at time " << endTime << "s with delay of:
        " << packetDelay << " s " << std::endl;
00178
00179 }
```

6.1.2.5 SentPacket()

```
static void SentPacket (
    Ptr< const Packet > p ) [static]
```

This is a function which handles the sending of packets.

Parameters

p	This is the packet that is being sent.
-----	--

The number of sent bytes gets increased, it adds the size of the packet that is being sent to the current amount of sent bytes.

The variable which tracks the number of sent packets increases.

This if statement checks if the first packet is being sent, and if it is, the variable which is declared for storing the time of the first sent packets actually gets that value assigned to itself.

The start time of each sent packet gets extracted. This applies to every packet and the Uid of that packet gets extracted so that the message which prints at what time the specific packet got sent.

This message prints when a certain packet got sent.

Definition at line 85 of file [TOR.cc](#).

```

00085                                     {
00086     m_bytes_sent += p->GetSize();
00087     m_packets_sent++;
00088     if (g_firstPacket) {
00089         g_firstPacketTime = Simulator::Now();
00090         g_firstPacket = false;
00091     }
00092     g_lastPacketTime = Simulator::Now();
00093     PacketStartTimes[p->GetUid()] = Simulator::Now().GetSeconds();
00094     std::cout << "\nPacket " << p->GetUid()+1 << " sent at time " << Simulator::Now().GetSeconds() <<
00095     "s" << std::endl;
00096 }
00097 }
```

6.1.3 Variable Documentation

6.1.3.1 g_firstPacket

```
bool g_firstPacket = true [static]
```

This is a boolean value for the first packet which can be true or false.

Definition at line 46 of file [TOR.cc](#).

6.1.3.2 g_firstPacketTime

```
Time g_firstPacketTime = Seconds(0.0) [static]
```

This is the variable which gets assigned a value of the time when the first packet was transmitted.

Definition at line 38 of file [TOR.cc](#).

6.1.3.3 g_lastPacketTime

```
Time g_lastPacketTime = Seconds(0.0) [static]
```

This is the variable which gets assigned a value of the time when the last packet was transmitted.

Definition at line 42 of file [TOR.cc](#).

6.1.3.4 m_bytes_received

```
uint32_t m_bytes_received = 0
```

This variable stores the amount of received bytes.

Definition at line 64 of file [TOR.cc](#).

6.1.3.5 m_bytes_sent

```
uint32_t m_bytes_sent = 0
```

This is an unsigned variable type, which means that it cannot have negative values, and the range for the numbers it can have is from 0 to 2^{32} . This variable stores the amount of sent bytes.

Definition at line 60 of file [TOR.cc](#).

6.1.3.6 m_delayTable

```
std::map<uint32_t, double> m_delayTable
```

Definition at line 78 of file [TOR.cc](#).

6.1.3.7 m_packets_received

```
uint32_t m_packets_received = 0
```

This variable stores the amount of received packets.

Definition at line 72 of file [TOR.cc](#).

6.1.3.8 m_packets_sent

```
uint32_t m_packets_sent = 0
```

This variable stores the amount of sent packets.

Definition at line 68 of file [TOR.cc](#).

6.1.3.9 m_time

```
double m_time = 0
```

Definition at line 75 of file [TOR.cc](#).

6.1.3.10 nodeNames

```
std::vector<std::string> nodeNames = {"Client", "Entry", "Relay1", "Relay2", "Relay3", "Exit",  
"Destination"}
```

Here, we are declaring node names for all nodes in the network topology.

Definition at line 33 of file [TOR.cc](#).

```
00033 {"Client", "Entry", "Relay1", "Relay2", "Relay3", "Exit", "Destination"};
```

6.1.3.11 packetCount

```
int packetCount = 0 [static]
```

@btief This is a variable, type int, and it stores packet count.

Definition at line 56 of file [TOR.cc](#).

6.1.3.12 PacketStartTimes

```
std::map<uint32_t, double> PacketStartTimes [static]
```

Definition at line 48 of file [TOR.cc](#).

6.1.3.13 packetTracker

```
std::map<uint32_t, PacketTrace> packetTracker
```

Definition at line 29 of file [TOR.cc](#).

6.1.3.14 totalDelay

```
double totalDelay = 0.0 [static]
```

This is a variable, type double, which means it stores decimal values, and it stores the value of the total delay of the network.

Definition at line 52 of file [TOR.cc](#).

6.2 TOR.cc

[Go to the documentation of this file.](#)

```

00001 // Including necessary libraries.
00007 #include "ns3/core-module.h"
00008 #include "ns3/network-module.h"
00009 #include "ns3/internet-module.h"
00010 #include "ns3/point-to-point-module.h"
00011 #include "ns3/applications-module.h"
00012 #include "ns3/ipv4-global-routing-helper.h"
00013 #include "ns3/netanim-module.h"
00014 #include "ns3/mobility-module.h"
00015 #include <iostream>
00016 #include <string>
00017 #include <vector>
00018 #include <fstream> // This is needed for creating the output file.
00019
00020 using namespace ns3;
00021
00022 NS_LOG_COMPONENT_DEFINE("SimpleTOR");
00023
00024 struct PacketTrace {
00025     double sendTime;
00026     std::string path;
00027 };
00028
00029 std::map<uint32_t, PacketTrace> packetTracker;
00033 std::vector<std::string> nodeNameNames = {"Client", "Entry", "Relay1", "Relay2", "Relay3", "Exit",
"Destination"};
00034
00038 static Time g_firstPacketTime = Seconds(0.0);
00042 static Time g_lastPacketTime = Seconds(0.0);
00046 static bool g_firstPacket = true;
00047
00048 static std::map<uint32_t, double> PacketStartTimes;
00052 static double totalDelay = 0.0;
00056 static int packetCount = 0;
00060 uint32_t m_bytes_sent = 0;
00064 uint32_t m_bytes_received = 0;
00068 uint32_t m_packets_sent = 0;
00072 uint32_t m_packets_received = 0;
00073
00074 //Create help variable m_time
00075 double m_time = 0;
00076
00077 //Create c++ map for measuring delay time
00078 std::map<uint32_t, double> m_delayTable;
00079
00085 static void SentPacket(Ptr<const Packet> p) {
00089     m_bytes_sent += p->GetSize();
00093     m_packets_sent++;
00095     if (g_firstPacket) {
00096         g_firstPacketTime = Simulator::Now();
00097         g_firstPacket = false;
00098     }
00099
00100     g_lastPacketTime = Simulator::Now();
00101
00105     PacketStartTimes[p->GetUid()] = Simulator::Now().GetSeconds();
00109     std::cout << "\nPacket " << p->GetUid()+1 << " sent at time " << Simulator::Now().GetSeconds() <<
"s" << std::endl;
00110
00111 }
00112
00116 static void ReceivedPacket(Ptr<const Packet> p) {
00120     std::ofstream output_file("output.txt", std::ios::app); // This will create and open the file and
append data to it.
00121
00122     m_bytes_received += p->GetSize();
00123     m_packets_received++;
00124
00125     /*
00126     //HELP LINES USED FOR TESTING
00127     std::cout << "\n .....ReceivedPacket....." << p->GetUid() << "..." << p->GetSize() <<
"..... \n";
00128     p->Print(std::cout);
00129     std::cout << "\n ..... \n";
00130     */
00131
00135     double endTime = Simulator::Now().GetSeconds();
00139     double startTime = PacketStartTimes[p->GetUid()];
00143     double packetDelay = endTime - startTime;
00144
00145     //Ptr<Packet> packetCopy = p->Copy();
00146     //DecryptPacket (packetCopy);

```



```

00147
00148     totalDelay += packetDelay;
00149     packetCount++;
00153     double duration = Simulator::Now().GetSeconds();
00157     double throughputBps = (m_bytes_received * 8.0) / duration;
00158     //double averageDelay = totalDelay/packetCount;
00168     output_file << duration << " " << m_packets_sent << " " << m_packets_received << " " << throughputBps
<< " " << packetDelay << std::endl; // This will create an output file with: duration, sent packets,
received packets, throughput and the packet delay with spaces between them.
00172     output_file.close(); // This closes the output file after writing.
00173
00177     std::cout << "\nPacket " << p->GetUid()+1 << " received at time " << endTime << "s with delay of:
" << packetDelay << " s " << std::endl;
00178
00179 }
00180
00184 void Ratio(){
00185
00186     std::cout << "\n== TOR network statistics ==\n" << std::endl;
00187     std::cout << "Transmission summary:" << std::endl;
00188     std::cout << "-----" << std::endl;
00189     std::cout << "Total bytes sent:\t " << m_bytes_sent << std::endl;
00190     std::cout << "Total bytes received:\t " << m_bytes_received << std::endl;
00191     std::cout << "Total packets sent:\t " << m_packets_sent << std::endl;
00192     std::cout << "Total packets received:\t " << m_packets_received << std::endl;
00193     std::cout << "Delivery ratio (bytes):\t " << (float)m_bytes_received/(float)m_bytes_sent * 100 <<
"% " << std::endl;
00194     std::cout << "Delivery ratio (packets): " << (float)m_packets_received/(float)m_packets_sent * 100 <<
"% " << std::endl;
00195
00196     double duration = Simulator::Now().GetSeconds();
00197     double throughputBps = (m_bytes_received * 8.0) / duration;
00198     if (duration > 0){
00199
00200         std::cout << "Troughput (bps):\t " << throughputBps << " bps " << std::endl;
00201         std::cout << "Troughput (kbps):\t " << throughputBps/1000.0 << " kbps " << std::endl;
00202     }
00203
00204     if (packetCount > 0) {
00205
00206         std::cout << "Average end-to-end delay: " << totalDelay/packetCount << "s" << std::endl;
00207
00208     }
00209     std::cout << "-----" << std::endl;
00210
00211
00212     std::cout << "Created output file: output.txt" << std::endl;
00213     std::cout << "-----" << std::endl;
00214
00215 }
00216
00220 int main(int argc, char *argv[]){
00225     // LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
00226
00227     Config::SetDefault ("ns3::Ipv4GlobalRouting::RespondToInterfaceEvents", BooleanValue(true));
00232     double simulationTime = 20; // 20 seconds.
00236     double maxPackets = 10; // 10 packets.
00237
00238     Packet::EnablePrinting();
00239     PacketMetadata::Enable();
00240
00244     CommandLine cmd;
00245     cmd.AddValue ("simulationTime", "simulationTime", simulationTime);
00246     cmd.AddValue ("maxPackets", "maxPackets", maxPackets);
00247     cmd.Parse (argc, argv);
00248
00249     Time::SetResolution (Time::NS);
00250     //LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_ALL);
00251     //LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_ALL);
00252     LogComponentEnable ("SimpleTOR", LOG_LEVEL_ALL);
00253
00257     NodeContainer nodes;
00261     nodes.Create(7);
00262
00263     //Point to Point links
00264     PointToPointHelper pointToPoint;
00268     pointToPoint.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
00272     pointToPoint.SetChannelAttribute("Delay", StringValue("25ms"));
00273
00277     NetDeviceContainer devices[6];
00281     Ipv4InterfaceContainer interfaces[6];
00282
00286     InternetStackHelper stack;
00287     stack.Install(nodes);
00288
00292     Ipv4AddressHelper address;
00293

```

```

00297     address.SetBase("10.1.1.0", "255.255.255.0");
00298     interfaces[0] = address.Assign(pointToPoint.Install(nodes.Get(0), nodes.Get(1)));
00299
00300     address.SetBase("10.1.2.0", "255.255.255.0");
00301     interfaces[1] = address.Assign(pointToPoint.Install(nodes.Get(1), nodes.Get(2)));
00302
00303     address.SetBase("10.1.3.0", "255.255.255.0");
00304     interfaces[2] = address.Assign(pointToPoint.Install(nodes.Get(2), nodes.Get(3)));
00305
00306     address.SetBase("10.1.4.0", "255.255.255.0");
00307     interfaces[3] = address.Assign(pointToPoint.Install(nodes.Get(3), nodes.Get(4)));
00308
00309     address.SetBase("10.1.5.0", "255.255.255.0");
00310     interfaces[4] = address.Assign(pointToPoint.Install(nodes.Get(4), nodes.Get(5)));
00311
00312     address.SetBase("10.1.6.0", "255.255.255.0");
00313     interfaces[5] = address.Assign(pointToPoint.Install(nodes.Get(5), nodes.Get(6)));
00314
00315     Ipv4GlobalRoutingHelper::PopulateRoutingTables();
00316
00317     UdpEchoServerHelper echoServer(9);
00318     ApplicationContainer serverApp = echoServer.Install(nodes.Get(6));
00319
00323     serverApp.Start(Seconds(1.0));
00327     serverApp.Stop(Seconds(simulationTime));
00328
00329     UdpEchoClientHelper echoClient(interfaces[5].GetAddress(1), 9);
00330     echoClient.SetAttribute("MaxPackets", UintegerValue(maxPackets));
00331     echoClient.SetAttribute("Interval", TimeValue(Seconds(0.1)));
00332
00333     ApplicationContainer clientApp = echoClient.Install(nodes.Get(0));
00334     clientApp.Start(Seconds(1.0));
00335     clientApp.Stop(Seconds(simulationTime));
00336
00337     // Connect trace sources for packet tracking
00338     Config::ConnectWithoutContext ("/NodeList/*/ApplicationList*/$ns3::UdpEchoClient/Tx",
MakeCallback(&SentPacket));
00339     Config::ConnectWithoutContext ("/NodeList/*/ApplicationList*/$ns3::UdpEchoServer/Rx",
MakeCallback(&ReceivedPacket));
00340
00341     // Mobility Setup
00342     MobilityHelper mobility;
00346     mobility.SetPositionAllocator("ns3::GridPositionAllocator",
00347                                   "MinX", DoubleValue(50.0),
00348                                   "MinY", DoubleValue(80.0),
00349                                   "DeltaX", DoubleValue(60.0),
00350                                   "DeltaY", DoubleValue(70.0),
00351                                   "GridWidth", UintegerValue(4),
00352                                   "LayoutType", StringValue("RowFirst"));
00353
00354     mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
00355     mobility.Install(nodes);
00356
00357     // NetAnim
00361     AnimationInterface anim("TOR.xml");
00365     anim.SetMaxPktsPerTraceFile(5000);
00372     anim.UpdateNodeDescription(0, "Client");
00373     anim.UpdateNodeDescription(1, "Entry Guard");
00374     anim.UpdateNodeDescription(2, "Relay 1");
00375     anim.UpdateNodeDescription(3, "Relay 2");
00376     anim.UpdateNodeDescription(4, "Relay 3");
00377     anim.UpdateNodeDescription(5, "Exit");
00378     anim.UpdateNodeDescription(6, "Destination");
00379
00380     anim.UpdateNodeColor(0, 255, 0, 0); // Red for Client
00381     anim.UpdateNodeColor(1, 0, 255, 0); // Green for Entry Guard
00382     anim.UpdateNodeColor(2, 0, 0, 255); // Blue for Relay 1
00383     anim.UpdateNodeColor(3, 255, 255, 0); // Yellow for Relay 2
00384     anim.UpdateNodeColor(4, 255, 0, 255); // Purple for Relay 3
00385     anim.UpdateNodeColor(5, 0, 255, 255); // Cyan for Exit
00386     anim.UpdateNodeColor(6, 128, 128, 128); // Gray for Destination
00387
00388
00392     pointToPoint.EnablePcapAll("tor_packet_trace");
00393
00394     Simulator::Schedule(Seconds(simulationTime), &Ratio);
00395
00399     Simulator::Run();
00403     Simulator::Destroy();
00404
00408     return 0;
00409 }
00410

```

6.3 udp-echo-client.cc File Reference

This file allows the sending of packets in [TOR.cc](#) which is located in the scratch folder.

```
#include "udp-echo-client.h"
#include "ns3/inet-socket-address.h"
#include "ns3/inet6-socket-address.h"
#include "ns3/ipv4-address.h"
#include "ns3/ipv6-address.h"
#include "ns3/log.h"
#include "ns3/nstime.h"
#include "ns3/packet.h"
#include "ns3/simulator.h"
#include "ns3/socket-factory.h"
#include "ns3/socket.h"
#include "ns3/trace-source-accessor.h"
#include "ns3/uinteger.h"
```

Namespaces

- namespace [ns3](#)

Functions

- [ns3::NS_LOG_COMPONENT_DEFINE](#) ("UdpEchoClientApplication")
- [ns3::NS_OBJECT_ENSURE_REGISTERED](#) (UdpEchoClient)

6.3.1 Detailed Description

This file allows the sending of packets in [TOR.cc](#) which is located in the scratch folder.

Definition in file [udp-echo-client.cc](#).

6.4 udp-echo-client.cc

[Go to the documentation of this file.](#)

```
00001 /*
00002  * Copyright 2007 University of Washington
00003  *
00004  * SPDX-License-Identifier: GPL-2.0-only
00005  */
00011 #include "udp-echo-client.h"
00012 #include "ns3/inet-socket-address.h"
00013 #include "ns3/inet6-socket-address.h"
00014 #include "ns3/ipv4-address.h"
00015 #include "ns3/ipv6-address.h"
00016 #include "ns3/log.h"
00017 #include "ns3/nstime.h"
00018 #include "ns3/packet.h"
00019 #include "ns3/simulator.h"
00020 #include "ns3/socket-factory.h"
00021 #include "ns3/socket.h"
00022 #include "ns3/trace-source-accessor.h"
00023 #include "ns3/uinteger.h"
00024
00025 namespace ns3
```

```

00026 {
00027
00028 NS_LOG_COMPONENT_DEFINE("UdpEchoClientApplication");
00029
00030 NS_OBJECT_ENSURE_REGISTERED(UdpEchoClient);
00031
00032 TypeId
00033 UdpEchoClient::GetTypeId()
00034 {
00035     static TypeId tid =
00036         TypeId("ns3::UdpEchoClient")
00037             .SetParent<Application>()
00038             .SetGroupName("Applications")
00039             .AddConstructor<UdpEchoClient>()
00040             .AddAttribute(
00041                 "MaxPackets",
00042                 "The maximum number of packets the application will send (zero means infinite)",
00043                 UIntegerValue(100),
00044                 MakeUIntegerAccessor(&UdpEchoClient::m_count),
00045                 MakeUIntegerChecker<uint32_t>())
00046             .AddAttribute("Interval",
00047                 "The time to wait between packets",
00048                 TimeValue(Seconds(1.0)),
00049                 MakeTimeAccessor(&UdpEchoClient::m_interval),
00050                 MakeTimeChecker())
00051             .AddAttribute("RemoteAddress",
00052                 "The destination Address of the outbound packets",
00053                 AddressValue(),
00054                 MakeAddressAccessor(&UdpEchoClient::m_peerAddress),
00055                 MakeAddressChecker())
00056             .AddAttribute("RemotePort",
00057                 "The destination port of the outbound packets",
00058                 UIntegerValue(0),
00059                 MakeUIntegerAccessor(&UdpEchoClient::m_peerPort),
00060                 MakeUIntegerChecker<uint16_t>())
00061             .AddAttribute("Tos",
00062                 "The Type of Service used to send IPv4 packets. "
00063                 "All 8 bits of the TOS byte are set (including ECN bits).",
00064                 UIntegerValue(0),
00065                 MakeUIntegerAccessor(&UdpEchoClient::m_tos),
00066                 MakeUIntegerChecker<uint8_t>())
00067             .AddAttribute(
00068                 "PacketSize",
00069                 "Size of echo data in outbound packets",
00070                 UIntegerValue(100),
00071                 MakeUIntegerAccessor(&UdpEchoClient::SetDataSize, &UdpEchoClient::GetDataSize),
00072                 MakeUIntegerChecker<uint32_t>())
00073             .AddTraceSource("Tx",
00074                 "A new packet is created and is sent",
00075                 MakeTraceSourceAccessor(&UdpEchoClient::m_txTrace),
00076                 "ns3::Packet::TracedCallback")
00077             .AddTraceSource("Rx",
00078                 "A packet has been received",
00079                 MakeTraceSourceAccessor(&UdpEchoClient::m_rxTrace),
00080                 "ns3::Packet::TracedCallback")
00081             .AddTraceSource("TxWithAddresses",
00082                 "A new packet is created and is sent",
00083                 MakeTraceSourceAccessor(&UdpEchoClient::m_txTraceWithAddresses),
00084                 "ns3::Packet::TwoAddressTracedCallback")
00085             .AddTraceSource("RxWithAddresses",
00086                 "A packet has been received",
00087                 MakeTraceSourceAccessor(&UdpEchoClient::m_rxTraceWithAddresses),
00088                 "ns3::Packet::TwoAddressTracedCallback");
00089     return tid;
00090 }
00091
00092 UdpEchoClient::UdpEchoClient()
00093 {
00094     NS_LOG_FUNCTION(this);
00095     m_sent = 0;
00096     m_socket = nullptr;
00097     m_sendEvent = EventId();
00098     m_data = nullptr;
00099     m_dataSize = 0;
00100 }
00101
00102 UdpEchoClient::~UdpEchoClient()
00103 {
00104     NS_LOG_FUNCTION(this);
00105     m_socket = nullptr;
00106
00107     delete[] m_data;
00108     m_data = nullptr;
00109     m_dataSize = 0;
00110 }
00111
00112 void

```

```

00113 UdpEchoClient::SetRemote(Address ip, uint16_t port)
00114 {
00115     NS_LOG_FUNCTION(this « ip « port);
00116     m_peerAddress = ip;
00117     m_peerPort = port;
00118 }
00119
00120 void
00121 UdpEchoClient::SetRemote(Address addr)
00122 {
00123     NS_LOG_FUNCTION(this « addr);
00124     m_peerAddress = addr;
00125 }
00126
00127 void
00128 UdpEchoClient::StartApplication()
00129 {
00130     NS_LOG_FUNCTION(this);
00131
00132     if (!m_socket)
00133     {
00134         TypeId tid = TypeId::LookupByName("ns3::UdpSocketFactory");
00135         m_socket = Socket::CreateSocket(GetNode(), tid);
00136         NS_ABORT_MSG_IF(m_peerAddress.IsValid(), "'RemoteAddress' attribute not properly set");
00137         if (Ipv4Address::IsMatchingType(m_peerAddress))
00138         {
00139             if (m_socket->Bind() == -1)
00140             {
00141                 NS_FATAL_ERROR("Failed to bind socket");
00142             }
00143             m_socket->SetIpTos(m_tos); // Affects only IPv4 sockets.
00144             m_socket->Connect(
00145                 InetSocketAddress(Ipv4Address::ConvertFrom(m_peerAddress), m_peerPort));
00146         }
00147         else if (Ipv6Address::IsMatchingType(m_peerAddress))
00148         {
00149             if (m_socket->Bind6() == -1)
00150             {
00151                 NS_FATAL_ERROR("Failed to bind socket");
00152             }
00153             m_socket->Connect(
00154                 Inet6SocketAddress(Ipv6Address::ConvertFrom(m_peerAddress), m_peerPort));
00155         }
00156         else if (InetSocketAddress::IsMatchingType(m_peerAddress))
00157         {
00158             if (m_socket->Bind() == -1)
00159             {
00160                 NS_FATAL_ERROR("Failed to bind socket");
00161             }
00162             m_socket->SetIpTos(m_tos); // Affects only IPv4 sockets.
00163             m_socket->Connect(m_peerAddress);
00164         }
00165         else if (Inet6SocketAddress::IsMatchingType(m_peerAddress))
00166         {
00167             if (m_socket->Bind6() == -1)
00168             {
00169                 NS_FATAL_ERROR("Failed to bind socket");
00170             }
00171             m_socket->Connect(m_peerAddress);
00172         }
00173         else
00174         {
00175             NS_ASSERT_MSG(false, "Incompatible address type: " « m_peerAddress);
00176         }
00177     }
00178
00179     m_socket->SetRecvCallback(MakeCallback(&UdpEchoClient::HandleRead, this));
00180     m_socket->SetAllowBroadcast(true);
00181     ScheduleTransmit(Seconds(0.));
00182 }
00183
00184 void
00185 UdpEchoClient::StopApplication()
00186 {
00187     NS_LOG_FUNCTION(this);
00188
00189     if (m_socket)
00190     {
00191         m_socket->Close();
00192         m_socket->SetRecvCallback(MakeNullCallback<void, Ptr<Socket>>());
00193         m_socket = nullptr;
00194     }
00195
00196     Simulator::Cancel(m_sendEvent);
00197 }
00198
00199 void

```

```

00200 UdpEchoClient::SetDataSize(uint32_t dataSize)
00201 {
00202     NS_LOG_FUNCTION(this « dataSize);
00203
00204     //
00205     // If the client is setting the echo packet data size this way, we infer
00206     // that she doesn't care about the contents of the packet at all, so
00207     // neither will we.
00208     //
00209     delete[] m_data;
00210     m_data = nullptr;
00211     m_dataSize = 0;
00212     m_size = dataSize;
00213 }
00214
00215 uint32_t
00216 UdpEchoClient::GetDataSize() const
00217 {
00218     NS_LOG_FUNCTION(this);
00219     return m_size;
00220 }
00221
00222 void
00223 UdpEchoClient::SetFill(std::string fill)
00224 {
00225     NS_LOG_FUNCTION(this « fill);
00226
00227     uint32_t dataSize = fill.size() + 1;
00228
00229     if (dataSize != m_dataSize)
00230     {
00231         delete[] m_data;
00232         m_data = new uint8_t[dataSize];
00233         m_dataSize = dataSize;
00234     }
00235
00236     memcpy(m_data, fill.c_str(), dataSize);
00237
00238     //
00239     // Overwrite packet size attribute.
00240     //
00241     m_size = dataSize;
00242 }
00243
00244 void
00245 UdpEchoClient::SetFill(uint8_t fill, uint32_t dataSize)
00246 {
00247     NS_LOG_FUNCTION(this « fill « dataSize);
00248     if (dataSize != m_dataSize)
00249     {
00250         delete[] m_data;
00251         m_data = new uint8_t[dataSize];
00252         m_dataSize = dataSize;
00253     }
00254
00255     memset(m_data, fill, dataSize);
00256
00257     //
00258     // Overwrite packet size attribute.
00259     //
00260     m_size = dataSize;
00261 }
00262
00263 void
00264 UdpEchoClient::SetFill(uint8_t* fill, uint32_t fillSize, uint32_t dataSize)
00265 {
00266     NS_LOG_FUNCTION(this « fill « fillSize « dataSize);
00267     if (dataSize != m_dataSize)
00268     {
00269         delete[] m_data;
00270         m_data = new uint8_t[dataSize];
00271         m_dataSize = dataSize;
00272     }
00273
00274     if (fillSize >= dataSize)
00275     {
00276         memcpy(m_data, fill, dataSize);
00277         m_size = dataSize;
00278         return;
00279     }
00280
00281     //
00282     // Do all but the final fill.
00283     //
00284     uint32_t filled = 0;
00285     while (filled + fillSize < dataSize)
00286     {

```

```

00287         memcpy(&m_data[filled], fill, fillSize);
00288         filled += fillSize;
00289     }
00290
00291     //
00292     // Last fill may be partial
00293     //
00294     memcpy(&m_data[filled], fill, dataSize - filled);
00295
00296     //
00297     // Overwrite packet size attribute.
00298     //
00299     m_size = dataSize;
00300 }
00301
00302 void
00303 UdpEchoClient::ScheduleTransmit(Time dt)
00304 {
00305     NS_LOG_FUNCTION(this << dt);
00306     m_sendEvent = Simulator::Schedule(dt, &UdpEchoClient::Send, this);
00307 }
00308
00312 void
00313 UdpEchoClient::Send()
00314 {
00315     NS_LOG_FUNCTION(this);
00316
00317     NS_ASSERT(m_sendEvent.IsExpired());
00318
00319     Ptr<Packet> p;
00320     if (m_dataSize)
00321     {
00322         //
00323         // If m_dataSize is non-zero, we have a data buffer of the same size that we
00324         // are expected to copy and send. This state of affairs is created if one of
00325         // the Fill functions is called. In this case, m_size must have been set
00326         // to agree with m_dataSize
00327         //
00328         NS_ASSERT_MSG(m_dataSize == m_size,
00329             "UdpEchoClient::Send(): m_size and m_dataSize inconsistent");
00330         NS_ASSERT_MSG(m_data, "UdpEchoClient::Send(): m_dataSize but no m_data");
00331         NS_LOG_INFO("HI");
00332         p = Create<Packet>(m_data, m_dataSize);
00333     }
00334     else
00335     {
00336         //
00337         // If m_dataSize is zero, the client has indicated that it doesn't care
00338         // about the data itself either by specifying the data size by setting
00339         // the corresponding attribute or by not calling a SetFill function. In
00340         // this case, we don't worry about it either. But we do allow m_size
00341         // to have a value different from the (zero) m_dataSize.
00342
00343         std::string message = "Hello World!";
00344         uint32_t dataSize = message.size();
00345         uint8_t* dataBuffer = new uint8_t[dataSize];
00346         memcpy(dataBuffer, message.c_str(), dataSize);
00347         uint8_t keys[] = {'A', 'B', 'C', 'F', 'E', 'D'};
00348         std::cout << "\n";
00349         std::cout << "-----" << std::endl;
00350         std::cout << "Packet data before encryption: ";
00351         for (uint32_t i = 0; i < dataSize; i++){
00352             std::cout << dataBuffer[i];
00353         }
00354
00355         std::cout << "\n" << std::endl;
00356         std::cout << "-----" << std::endl;
00357         std::cout << "\n" << std::endl;
00358         std::cout << "-----" << std::endl;
00359
00360         // XOR encryption.
00361         int xor_counter = 0;
00362         while (true){
00363             std::cout << "Packet data after encryption with layer " << xor_counter+1 << ": ";
00364             for (uint32_t i = 0; i < dataSize; i++) {
00365                 dataBuffer[i] ^= keys[xor_counter];
00366                 std::cout << dataBuffer[i];
00367             }
00368             std::cout << "\n" << std::endl;
00369             xor_counter++;
00370             if (xor_counter == 6){
00371                 break;
00372             }
00373         }
00374
00375         std::cout << "-----" << std::endl;
00376     }

```

```

00396         p = Create<Packet>(dataBuffer, dataSize);
00401         delete[] dataBuffer;
00402     }
00403     Address localAddress;
00404     m_socket->GetSockName(localAddress);
00405     // call to the trace sinks before the packet is actually sent,
00406     // so that tags added to the packet can be sent as well
00407     m_txTrace(p);
00408     if (Ipv4Address::IsMatchingType(m_peerAddress))
00409     {
00410         m_txTraceWithAddresses(
00411             p,
00412             localAddress,
00413             InetSocketAddress(Ipv4Address::ConvertFrom(m_peerAddress), m_peerPort));
00414     }
00415     else if (Ipv6Address::IsMatchingType(m_peerAddress))
00416     {
00417         m_txTraceWithAddresses(
00418             p,
00419             localAddress,
00420             Inet6SocketAddress(Ipv6Address::ConvertFrom(m_peerAddress), m_peerPort));
00421     }
00422     m_socket->Send(p);
00423     ++m_sent;
00424
00425     if (Ipv4Address::IsMatchingType(m_peerAddress))
00426     {
00427         NS_LOG_INFO("At time " « Simulator::Now().As(Time::S) « " client sent " « m_size
00428                     « " bytes to " « Ipv4Address::ConvertFrom(m_peerAddress)
00429                     « " port " « m_peerPort);
00430     }
00431     else if (Ipv6Address::IsMatchingType(m_peerAddress))
00432     {
00433         NS_LOG_INFO("At time " « Simulator::Now().As(Time::S) « " client sent " « m_size
00434                     « " bytes to " « Ipv6Address::ConvertFrom(m_peerAddress)
00435                     « " port " « m_peerPort);
00436     }
00437     else if (InetSocketAddress::IsMatchingType(m_peerAddress))
00438     {
00439         NS_LOG_INFO(
00440             "At time " « Simulator::Now().As(Time::S) « " client sent " « m_size « " bytes to "
00441             « InetSocketAddress::ConvertFrom(m_peerAddress).GetIpv4() « " port "
00442             « InetSocketAddress::ConvertFrom(m_peerAddress).GetPort());
00443     }
00444     else if (Inet6SocketAddress::IsMatchingType(m_peerAddress))
00445     {
00446         NS_LOG_INFO(
00447             "At time " « Simulator::Now().As(Time::S) « " client sent " « m_size « " bytes to "
00448             « Inet6SocketAddress::ConvertFrom(m_peerAddress).GetIpv6() « " port "
00449             « Inet6SocketAddress::ConvertFrom(m_peerAddress).GetPort());
00450     }
00451
00452     if (m_sent < m_count || m_count == 0)
00453     {
00454         ScheduleTransmit(m_interval);
00455     }
00456 }
00460 void
00461 UdpEchoClient::HandleRead(Ptr<Socket> socket)
00462 {
00463     NS_LOG_FUNCTION(this « socket);
00464     Ptr<Packet> packet;
00465     Address from;
00466     Address localAddress;
00467     while ((packet = socket->RecvFrom(from)))
00468     {
00469         if (InetSocketAddress::IsMatchingType(from))
00470         {
00471             NS_LOG_INFO("At time " « Simulator::Now().As(Time::S) « " client received "
00472                         « packet->GetSize() « " bytes from "
00473                         « InetSocketAddress::ConvertFrom(from).GetIpv4() « " port "
00474                         « InetSocketAddress::ConvertFrom(from).GetPort());
00475         }
00476         else if (Inet6SocketAddress::IsMatchingType(from))
00477         {
00478             NS_LOG_INFO("At time " « Simulator::Now().As(Time::S) « " client received "
00479                         « packet->GetSize() « " bytes from "
00480                         « Inet6SocketAddress::ConvertFrom(from).GetIpv6() « " port "
00481                         « Inet6SocketAddress::ConvertFrom(from).GetPort());
00482         }
00483         socket->GetSockName(localAddress);
00484         m_rxTrace(packet);
00485         m_rxTraceWithAddresses(packet, from, localAddress);
00486     }
00487 }
00488
00489 } // Namespace ns3

```


6.5 udp-echo-server.cc File Reference

This file enables the process of receiving packets in [TOR.cc](#) file located in the ../../scratch folder.

```
#include "udp-echo-server.h"
#include "ns3/address-utils.h"
#include "ns3/inet-socket-address.h"
#include "ns3/inet6-socket-address.h"
#include "ns3/ipv4-address.h"
#include "ns3/ipv6-address.h"
#include "ns3/log.h"
#include "ns3/nstime.h"
#include "ns3/packet.h"
#include "ns3/simulator.h"
#include "ns3/socket-factory.h"
#include "ns3/socket.h"
#include "ns3/udp-socket.h"
#include "ns3/uinteger.h"
```

Namespaces

- namespace [ns3](#)

Functions

- [ns3::NS_LOG_COMPONENT_DEFINE](#) ("UdpEchoServerApplication")
- [ns3::NS_OBJECT_ENSURE_REGISTERED](#) (UdpEchoServer)

6.5.1 Detailed Description

This file enables the process of receiving packets in [TOR.cc](#) file located in the ../../scratch folder.

Definition in file [udp-echo-server.cc](#).

6.6 udp-echo-server.cc

[Go to the documentation of this file.](#)

```
00001 /*
00002  * Copyright 2007 University of Washington
00003  *
00004  * SPDX-License-Identifier: GPL-2.0-only
00005  */
00010 #include "udp-echo-server.h"
00011
00012 #include "ns3/address-utils.h"
00013 #include "ns3/inet-socket-address.h"
00014 #include "ns3/inet6-socket-address.h"
00015 #include "ns3/ipv4-address.h"
00016 #include "ns3/ipv6-address.h"
00017 #include "ns3/log.h"
00018 #include "ns3/nstime.h"
00019 #include "ns3/packet.h"
00020 #include "ns3/simulator.h"
00021 #include "ns3/socket-factory.h"
00022 #include "ns3/socket.h"
00023 #include "ns3/udp-socket.h"
```

```

00024 #include "ns3/uinteger.h"
00025
00026 namespace ns3
00027 {
00028
00029 NS_LOG_COMPONENT_DEFINE("UdpEchoServerApplication");
00030
00031 NS_OBJECT_ENSURE_REGISTERED(UdpEchoServer);
00032
00033 TypeId
00034 UdpEchoServer::GetTypeId()
00035 {
00036     static TypeId tid =
00037         TypeId("ns3::UdpEchoServer")
00038             .SetParent<Application>()
00039             .SetGroupName("Applications")
00040             .AddConstructor<UdpEchoServer>()
00041             .AddAttribute("Port",
00042                 "Port on which we listen for incoming packets.",
00043                 UIntegerValue(9),
00044                 MakeUIntegerAccessor(&UdpEchoServer::m_port),
00045                 MakeUIntegerChecker<uint16_t>())
00046             .AddAttribute("Tos",
00047                 "The Type of Service used to send IPv4 packets. "
00048                 "All 8 bits of the TOS byte are set (including ECN bits).",
00049                 UIntegerValue(0),
00050                 MakeUIntegerAccessor(&UdpEchoServer::m_tos),
00051                 MakeUIntegerChecker<uint8_t>())
00052             .AddTraceSource("Rx",
00053                 "A packet has been received",
00054                 MakeTraceSourceAccessor(&UdpEchoServer::m_rxTrace),
00055                 "ns3::Packet::TracedCallback")
00056             .AddTraceSource("RxWithAddresses",
00057                 "A packet has been received",
00058                 MakeTraceSourceAccessor(&UdpEchoServer::m_rxTraceWithAddresses),
00059                 "ns3::Packet::TwoAddressTracedCallback");
00060     return tid;
00061 }
00062
00063 UdpEchoServer::UdpEchoServer()
00064 {
00065     NS_LOG_FUNCTION(this);
00066 }
00067
00068 UdpEchoServer::~UdpEchoServer()
00069 {
00070     NS_LOG_FUNCTION(this);
00071     m_socket = nullptr;
00072     m_socket6 = nullptr;
00073 }
00074
00075 void
00076 UdpEchoServer::StartApplication()
00077 {
00078     NS_LOG_FUNCTION(this);
00079
00080     if (!m_socket)
00081     {
00082         TypeId tid = TypeId::LookupByName("ns3::UdpSocketFactory");
00083         m_socket = Socket::CreateSocket(GetNode(), tid);
00084         InetSocketAddress local = InetSocketAddress(Ipv4Address::GetAny(), m_port);
00085         if (m_socket->Bind(local) == -1)
00086         {
00087             NS_FATAL_ERROR("Failed to bind socket");
00088         }
00089         if (addressUtils::IsMulticast(m_local))
00090         {
00091             Ptr<UdpSocket> udpSocket = DynamicCast<UdpSocket>(m_socket);
00092             if (udpSocket)
00093             {
00094                 // equivalent to setsockopt (MCAST_JOIN_GROUP)
00095                 udpSocket->MulticastJoinGroup(0, m_local);
00096             }
00097             else
00098             {
00099                 NS_FATAL_ERROR("Error: Failed to join multicast group");
00100             }
00101         }
00102     }
00103
00104     if (!m_socket6)
00105     {
00106         TypeId tid = TypeId::LookupByName("ns3::UdpSocketFactory");
00107         m_socket6 = Socket::CreateSocket(GetNode(), tid);
00108         Inet6SocketAddress local6 = Inet6SocketAddress(Ipv6Address::GetAny(), m_port);
00109         if (m_socket6->Bind(local6) == -1)
00110         {

```

```

00111         NS_FATAL_ERROR("Failed to bind socket");
00112     }
00113     if (addressUtils::IsMulticast(local6))
00114     {
00115         Ptr<UdpSocket> udpSocket = DynamicCast<UdpSocket>(m_socket6);
00116         if (udpSocket)
00117         {
00118             // equivalent to setsockopt (MCAST_JOIN_GROUP)
00119             udpSocket->MulticastJoinGroup(0, local6);
00120         }
00121         else
00122         {
00123             NS_FATAL_ERROR("Error: Failed to join multicast group");
00124         }
00125     }
00126 }
00127
00128 m_socket->SetIpTos(m_tos); // Affects only IPv4 sockets.
00129 m_socket->SetRecvCallback(MakeCallback(&UdpEchoServer::HandleRead, this));
00130 m_socket6->SetRecvCallback(MakeCallback(&UdpEchoServer::HandleRead, this));
00131 }
00132
00133 void
00134 UdpEchoServer::StopApplication()
00135 {
00136     NS_LOG_FUNCTION(this);
00137
00138     if (m_socket)
00139     {
00140         m_socket->Close();
00141         m_socket->SetRecvCallback(MakeNullCallback<void, Ptr<Socket>>());
00142     }
00143     if (m_socket6)
00144     {
00145         m_socket6->Close();
00146         m_socket6->SetRecvCallback(MakeNullCallback<void, Ptr<Socket>>());
00147     }
00148 }
00149
00150 void
00151 UdpEchoServer::HandleRead(Ptr<Socket> socket)
00152 {
00153     NS_LOG_FUNCTION(this « socket);
00154
00155     Ptr<Packet> packet;
00156     Address from;
00157     Address localAddress;
00158
00159     while ((packet = socket->RecvFrom(from)))
00160     {
00161         socket->GetSockName(localAddress);
00162         m_rxTrace(packet);
00163         m_rxTraceWithAddresses(packet, from, localAddress);
00164
00165         uint32_t packet_size = packet->GetSize();
00166         uint8_t* buffer = new uint8_t[packet_size];
00167         packet->CopyData(buffer, packet_size);
00168         uint8_t keys[] = {'A', 'B', 'C', 'F', 'E', 'D'};
00169         std::cout « " " « std::endl;
00170         std::cout « "-----" « std::endl;
00171         std::cout « "Packet data before decryption: ";
00172         for (uint32_t i = 0; i < packet_size; i++){
00173             std::cout « buffer[i];
00174         }
00175
00176         std::cout « " " « std::endl;
00177         std::cout « "-----" « std::endl;
00178         std::cout « " " « std::endl;
00179         std::cout « "-----" « std::endl;
00180
00181         // XOR decryption.
00182         int xor_counter = 0; // This counter is needed to not go over the limit of 6 encryption
00183         layers.
00184         int counter_for_keys = (sizeof(keys) / sizeof(keys[0]) - 1);
00185         while (true){
00186             std::cout « "Packet data after decryption on layer " « xor_counter+1 « ": ";
00187             for (uint32_t i = 0; i < packet_size; i++){
00188                 buffer[i] = buffer[i] ^ keys[counter_for_keys];
00189                 std::cout « buffer[i];
00190             }
00191             std::cout « " " « std::endl;
00192             counter_for_keys--;
00193             xor_counter++;
00194             if (xor_counter == 6){
00195                 break;
00196             }
00197         }
00198     }

```

```

00209     }
00210     std::cout << "-----" << std::endl;
00211
00212     if (InetSocketAddress::IsMatchingType(from))
00213     {
00214         NS_LOG_INFO("At time " << Simulator::Now().As(Time::S) << " server received "
00215                     << packet->GetSize() << " bytes from "
00216                     << InetSocketAddress::ConvertFrom(from).GetIpv4() << " port "
00217                     << InetSocketAddress::ConvertFrom(from).GetPort());
00218     }
00219     else if (Inet6SocketAddress::IsMatchingType(from))
00220     {
00221         NS_LOG_INFO("At time " << Simulator::Now().As(Time::S) << " server received "
00222                     << packet->GetSize() << " bytes from "
00223                     << Inet6SocketAddress::ConvertFrom(from).GetIpv6() << " port "
00224                     << Inet6SocketAddress::ConvertFrom(from).GetPort());
00225     }
00226
00227     packet->RemoveAllPacketTags();
00228     packet->RemoveAllByteTags();
00229
00230     NS_LOG_LOGIC("Echoing packet");
00231     socket->SendTo(packet, 0, from);
00232
00233     if (InetSocketAddress::IsMatchingType(from))
00234     {
00235         NS_LOG_INFO("At time " << Simulator::Now().As(Time::S) << " server sent "
00236                     << packet->GetSize() << " bytes to "
00237                     << InetSocketAddress::ConvertFrom(from).GetIpv4() << " port "
00238                     << InetSocketAddress::ConvertFrom(from).GetPort());
00239     }
00240     else if (Inet6SocketAddress::IsMatchingType(from))
00241     {
00242         NS_LOG_INFO("At time " << Simulator::Now().As(Time::S) << " server sent "
00243                     << packet->GetSize() << " bytes to "
00244                     << Inet6SocketAddress::ConvertFrom(from).GetIpv6() << " port "
00245                     << Inet6SocketAddress::ConvertFrom(from).GetPort());
00246     }
00247 }
00248 }
00249
00250 } // Namespace ns3

```

Index

- g_firstPacket
 - TOR.cc, [19](#)
- g_firstPacketTime
 - TOR.cc, [19](#)
- g_lastPacketTime
 - TOR.cc, [19](#)
- m_bytes_received
 - TOR.cc, [19](#)
- m_bytes_sent
 - TOR.cc, [20](#)
- m_delayTable
 - TOR.cc, [20](#)
- m_packets_received
 - TOR.cc, [20](#)
- m_packets_sent
 - TOR.cc, [20](#)
- m_time
 - TOR.cc, [20](#)
- main
 - TOR.cc, [14](#)
- nodeNames
 - TOR.cc, [20](#)
- ns3, [9](#)
 - NS_LOG_COMPONENT_DEFINE, [9](#)
 - NS_OBJECT_ENSURE_REGISTERED, [9](#)
- NS_LOG_COMPONENT_DEFINE
 - ns3, [9](#)
 - TOR.cc, [17](#)
- NS_OBJECT_ENSURE_REGISTERED
 - ns3, [9](#)
- packetCount
 - TOR.cc, [21](#)
- PacketStartTimes
 - TOR.cc, [21](#)
- PacketTrace, [11](#)
 - path, [11](#)
 - sendTime, [11](#)
- packetTracker
 - TOR.cc, [21](#)
- path
 - PacketTrace, [11](#)
- Ratio
 - TOR.cc, [17](#)
- ReceivedPacket
 - TOR.cc, [17](#)
- sendTime
- PacketTrace, [11](#)
- SentPacket
 - TOR.cc, [18](#)
- TOR.cc, [13](#)
 - g_firstPacket, [19](#)
 - g_firstPacketTime, [19](#)
 - g_lastPacketTime, [19](#)
 - m_bytes_received, [19](#)
 - m_bytes_sent, [20](#)
 - m_delayTable, [20](#)
 - m_packets_received, [20](#)
 - m_packets_sent, [20](#)
 - m_time, [20](#)
 - main, [14](#)
 - nodeNames, [20](#)
 - NS_LOG_COMPONENT_DEFINE, [17](#)
 - packetCount, [21](#)
 - PacketStartTimes, [21](#)
 - packetTracker, [21](#)
 - Ratio, [17](#)
 - ReceivedPacket, [17](#)
 - SentPacket, [18](#)
 - totalDelay, [21](#)
- totalDelay
 - TOR.cc, [21](#)
- udp-echo-client.cc, [25](#)
- udp-echo-server.cc, [31](#)