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

Generated by Doxygen 1.9.8

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

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 successfull.

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);
```

```

    //Point to Point links
    PointToPointHelper pointToPoint;
    pointToPoint.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
    pointToPoint.SetChannelAttribute("Delay", StringValue("25ms"));

    NetDeviceContainer devices[6];
    Ipv4InterfaceContainer interfaces[6];

    InternetStackHelper stack;
    stack.Install(nodes);

    Ipv4AddressHelper address;

    address.SetBase("10.1.1.0", "255.255.255.0");
    interfaces[0] = address.Assign(pointToPoint.Install(nodes.Get(0), nodes.Get(1)));

    address.SetBase("10.1.2.0", "255.255.255.0");
    interfaces[1] = address.Assign(pointToPoint.Install(nodes.Get(1), nodes.Get(2)));

    address.SetBase("10.1.3.0", "255.255.255.0");
    interfaces[2] = address.Assign(pointToPoint.Install(nodes.Get(2), nodes.Get(3)));

    address.SetBase("10.1.4.0", "255.255.255.0");
    interfaces[3] = address.Assign(pointToPoint.Install(nodes.Get(3), nodes.Get(4)));

    address.SetBase("10.1.5.0", "255.255.255.0");
    interfaces[4] = address.Assign(pointToPoint.Install(nodes.Get(4), nodes.Get(5)));

    address.SetBase("10.1.6.0", "255.255.255.0");
    interfaces[5] = address.Assign(pointToPoint.Install(nodes.Get(5), nodes.Get(6)));

    Ipv4GlobalRoutingHelper::PopulateRoutingTables();

    UdpEchoServerHelper echoServer(9);
    ApplicationContainer serverApp = echoServer.Install(nodes.Get(6));

    serverApp.Start(Seconds(1.0));
    serverApp.Stop(Seconds(simulationTime));

    UdpEchoClientHelper echoClient(interfaces[5].GetAddress(1), 9);
    echoClient.SetAttribute("MaxPackets", UintegerValue(maxPackets));
    echoClient.SetAttribute("Interval", TimeValue(Seconds(0.1)));

    ApplicationContainer clientApp = echoClient.Install(nodes.Get(0));
    clientApp.Start(Seconds(1.0));
    clientApp.Stop(Seconds(simulationTime));

    // Connect trace sources for packet tracking
    Config::ConnectWithoutContext("/NodeList/*/ApplicationList/*/$ns3::UdpEchoClient/Tx",
MakeCallback(&SentPacket));
    Config::ConnectWithoutContext("/NodeList/*/ApplicationList/*/$ns3::UdpEchoServer/Rx",
MakeCallback(&ReceivedPacket));

    // Mobility Setup
    MobilityHelper mobility;
    mobility.SetPositionAllocator("ns3::GridPositionAllocator",
                                  "MinX", DoubleValue(50.0),
                                  "MinY", DoubleValue(80.0),
                                  "DeltaX", DoubleValue(60.0),
                                  "DeltaY", DoubleValue(70.0),
                                  "GridWidth", UintegerValue(4),
                                  "LayoutType", StringValue("RowFirst"));

    mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
    mobility.Install(nodes);

    // NetAnim
    AnimationInterface anim("TOR.xml");
    anim.SetMaxPktsPerTraceFile(5000);
    anim.UpdateNodeDescription(0, "Client");
    anim.UpdateNodeDescription(1, "Entry Guard");
    anim.UpdateNodeDescription(2, "Relay 1");
    anim.UpdateNodeDescription(3, "Relay 2");
    anim.UpdateNodeDescription(4, "Relay 3");
    anim.UpdateNodeDescription(5, "Exit");
    anim.UpdateNodeDescription(6, "Destination");

    anim.UpdateNodeColor(0, 255, 0, 0); // Red for Client
    anim.UpdateNodeColor(1, 0, 255, 0); // Green for Entry Guard
    anim.UpdateNodeColor(2, 0, 0, 255); // Blue for Relay 1
    anim.UpdateNodeColor(3, 255, 255, 0); // Yellow for Relay 2
    anim.UpdateNodeColor(4, 255, 0, 255); // Purple for Relay 3
    anim.UpdateNodeColor(5, 0, 255, 255); // Cyan for Exit
    anim.UpdateNodeColor(6, 128, 128, 128); // Gray for Destination
    ;

```

```
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 «
      "%" « 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 }
```

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.

```
                                          {
    m_bytes_sent += p->GetSize();
    m_packets_sent++;
    if (g_firstPacket) {
        g_firstPacketTime = Simulator::Now();
        g_firstPacket = false;
    }

    g_lastPacketTime = Simulator::Now();

    PacketStartTimes[p->GetUid()] = Simulator::Now().GetSeconds();
    std::cout « "\nPacket " « p->GetUid()+1 « " sent at time " «    Simulator::Now().GetSeconds() «
"s" « std::endl;

}
```

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.

```
// Including necessary libraries.
#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> // This is needed for creating the output file.

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("SimpleTOR");

struct PacketTrace {
    double sendTime;
    std::string path;
};

std::map<uint32_t, PacketTrace> packetTracker;
std::vector<std::string> nodeNames = {"Client", "Entry", "Relay1", "Relay2", "Relay3", "Exit",
"Destination"};

static Time g_firstPacketTime = Seconds(0.0);
static Time g_lastPacketTime = Seconds(0.0);
static bool g_firstPacket = true;

static std::map<uint32_t, double> PacketStartTimes;
static double totalDelay = 0.0;
static int packetCount = 0;
uint32_t m_bytes_sent = 0;
uint32_t m_bytes_received = 0;
uint32_t m_packets_sent = 0;
uint32_t m_packets_received = 0;

//Create help variable m_time
double m_time = 0;

//Create c++ map for measuring delay time
std::map<uint32_t, double> m_delayTable;

static void SentPacket(Ptr<const Packet> p) {
    m_bytes_sent += p->GetSize();
    m_packets_sent++;
    if (g_firstPacket) {
        g_firstPacketTime = Simulator::Now();
        g_firstPacket = false;
    }

    g_lastPacketTime = Simulator::Now();

    PacketStartTimes[p->GetUid()] = Simulator::Now().GetSeconds();
    std::cout « "\nPacket " « p->GetUid()+1 « " sent at time " «    Simulator::Now().GetSeconds() «
"s" « std::endl;

}

static void ReceivedPacket(Ptr<const Packet> p) {
    std::ofstream output_file("output.txt", std::ios::app); // This will create and open the file and
append data to it.

    m_bytes_received += p->GetSize();
    m_packets_received++;

    /*
    //HELP LINES USED FOR TESTING
    std::cout « "\n .................ReceivedPacket....." « p->GetUid() « "..." «  p->GetSize() «
".......  \n";
    p->Print(std::cout);
    std::cout « "\n ..........................................  \n";
    */

        double endTime = Simulator::Now().GetSeconds();
        double startTime = PacketStartTimes[p->GetUid()];
        double packetDelay = endTime - startTime;

        //Ptr<Packet> packetCopy = p->Copy();
          //DecryptPacket (packetCopy);

        totalDelay += packetDelay;
        packetCount++;
        double duration = Simulator::Now().GetSeconds();
        double throughputBps = (m_bytes_received * 8.0) / duration;
        //double averageDelay = totalDelay/packetCount;
        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.
        output_file.close(); // This closes the output file after writing.

        std::cout « "\nPacket " « p->GetUid()+1 « " received at time " « endTime « "s with delay of:
      "« packetDelay « " s " « std::endl;

}

void Ratio(){

    std::cout « "\n=== TOR network statistics ===\n" « std::endl;
    std::cout « "Transmission summary:" « std::endl;
    std::cout « "-------------------------------------" « std::endl;
    std::cout « "Total bytes sent:\t  " « m_bytes_sent « std::endl;
    std::cout « "Total bytes received:\t  " « m_bytes_received « std::endl;
    std::cout « "Total packets sent:\t  " « m_packets_sent « std::endl;
    std::cout « "Total packets received:\t  " « m_packets_received « std::endl;
    std::cout « "Delivery ratio (bytes):\t  " « (float)m_bytes_received/(float)m_bytes_sent * 100 «
      "%" « std::endl;
    std::cout « "Delivery ratio (packets): " « (float)m_packets_received/(float)m_packets_sent * 100 «
      "%" « std::endl;

    double duration = Simulator::Now().GetSeconds();
    double throughputBps = (m_bytes_received * 8.0) / duration;
    if (duration > 0){

        std::cout « "Troughput (bps):\t  " « throughputBps « " bps " « std::endl;
        std::cout « "Troughput (kbps):\t  " « throughputBps/1000.0 « " kbps " « std::endl;
    }

    if (packetCount > 0) {

       std::cout « "Average end-to-end delay: " « totalDelay/packetCount « "s" « std::endl;

    }
    std::cout « "-------------------------------------" « std::endl;


  std::cout « "Created output file: output.txt" « std::endl;
  std::cout « "-------------------------------------" « std::endl;

}

int main(int argc, char *argv[]){
    // LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);

Config::SetDefault ("ns3::Ipv4GlobalRouting::RespondToInterfaceEvents",BooleanValue(true));
    double simulationTime = 20; // 20 seconds.
    double maxPackets = 10; // 10 packets.

    Packet::EnablePrinting();
    PacketMetadata::Enable();

    CommandLine cmd;
    cmd.AddValue ("simulationTime", "simulationTime", simulationTime);
    cmd.AddValue ("maxPackets", "maxPackets", maxPackets);
    cmd.Parse (argc, argv);

    Time::SetResolution (Time::NS);
    //LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_ALL);
    //LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_ALL);
    LogComponentEnable ("SimpleTOR", LOG_LEVEL_ALL);

    NodeContainer nodes;
    nodes.Create(7);

    //Point to Point links
    PointToPointHelper pointToPoint;
    pointToPoint.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
    pointToPoint.SetChannelAttribute("Delay", StringValue("25ms"));

    NetDeviceContainer devices[6];
    Ipv4InterfaceContainer interfaces[6];

    InternetStackHelper stack;
    stack.Install(nodes);

    Ipv4AddressHelper address;

    address.SetBase("10.1.1.0", "255.255.255.0");
    interfaces[0] = address.Assign(pointToPoint.Install(nodes.Get(0), nodes.Get(1)));

    address.SetBase("10.1.2.0", "255.255.255.0");
    interfaces[1] = address.Assign(pointToPoint.Install(nodes.Get(1), nodes.Get(2)));

    address.SetBase("10.1.3.0", "255.255.255.0");
    interfaces[2] = address.Assign(pointToPoint.Install(nodes.Get(2), nodes.Get(3)));

    address.SetBase("10.1.4.0", "255.255.255.0");
    interfaces[3] = address.Assign(pointToPoint.Install(nodes.Get(3), nodes.Get(4)));

    address.SetBase("10.1.5.0", "255.255.255.0");
    interfaces[4] = address.Assign(pointToPoint.Install(nodes.Get(4), nodes.Get(5)));

    address.SetBase("10.1.6.0", "255.255.255.0");
    interfaces[5] = address.Assign(pointToPoint.Install(nodes.Get(5), nodes.Get(6)));

    Ipv4GlobalRoutingHelper::PopulateRoutingTables();

    UdpEchoServerHelper echoServer(9);
    ApplicationContainer serverApp = echoServer.Install(nodes.Get(6));

    serverApp.Start(Seconds(1.0));
    serverApp.Stop(Seconds(simulationTime));

    UdpEchoClientHelper echoClient(interfaces[5].GetAddress(1), 9);
    echoClient.SetAttribute("MaxPackets", UintegerValue(maxPackets));
    echoClient.SetAttribute("Interval", TimeValue(Seconds(0.1)));

    ApplicationContainer clientApp = echoClient.Install(nodes.Get(0));
    clientApp.Start(Seconds(1.0));
    clientApp.Stop(Seconds(simulationTime));

    // Connect trace sources for packet tracking
    Config::ConnectWithoutContext("/NodeList/*/ApplicationList/*/$ns3::UdpEchoClient/Tx", MakeCallback(&SentPacket));
    Config::ConnectWithoutContext("/NodeList/*/ApplicationList/*/$ns3::UdpEchoServer/Rx", MakeCallback(&ReceivedPacket));

    // Mobility Setup
    MobilityHelper mobility;
    mobility.SetPositionAllocator("ns3::GridPositionAllocator",
                                  "MinX", DoubleValue(50.0),
                                  "MinY", DoubleValue(80.0),
                                  "DeltaX", DoubleValue(60.0),
                                  "DeltaY", DoubleValue(70.0),
                                  "GridWidth", UintegerValue(4),
                                  "LayoutType", StringValue("RowFirst"));

    mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
    mobility.Install(nodes);

    // NetAnim
    AnimationInterface anim("TOR.xml");
    anim.SetMaxPktsPerTraceFile(5000);
    anim.UpdateNodeDescription(0, "Client");
    anim.UpdateNodeDescription(1, "Entry Guard");
    anim.UpdateNodeDescription(2, "Relay 1");
    anim.UpdateNodeDescription(3, "Relay 2");
    anim.UpdateNodeDescription(4, "Relay 3");
    anim.UpdateNodeDescription(5, "Exit");
    anim.UpdateNodeDescription(6, "Destination");

    anim.UpdateNodeColor(0, 255, 0, 0); // Red for Client
    anim.UpdateNodeColor(1, 0, 255, 0); // Green for Entry Guard
    anim.UpdateNodeColor(2, 0, 0, 255); // Blue for Relay 1
    anim.UpdateNodeColor(3, 255, 255, 0); // Yellow for Relay 2
    anim.UpdateNodeColor(4, 255, 0, 255); // Purple for Relay 3
    anim.UpdateNodeColor(5, 0, 255, 255); // Cyan for Exit
    anim.UpdateNodeColor(6, 128, 128, 128); // Gray for Destination
    ;

    pointToPoint.EnablePcapAll("tor_packet_trace");

    Simulator::Schedule(Seconds(simulationTime), &Ratio);

    Simulator::Run();
    Simulator::Destroy();

    return 0;
}

```

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.

```
/*
 * Copyright 2007 University of Washington
 *
 * SPDX-License-Identifier: GPL-2.0-only
 */
#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"

namespace ns3
```

```
{

NS_LOG_COMPONENT_DEFINE("UdpEchoClientApplication");

NS_OBJECT_ENSURE_REGISTERED(UdpEchoClient);

TypeId
UdpEchoClient::GetTypeId()
{
    static TypeId tid =
        TypeId("ns3::UdpEchoClient")
            .SetParent<Application>()
            .SetGroupName("Applications")
            .AddConstructor<UdpEchoClient>()
            .AddAttribute(
                "MaxPackets",
                "The maximum number of packets the application will send (zero means infinite)",
                UintegerValue(100),
                MakeUintegerAccessor(&UdpEchoClient::m_count),
                MakeUintegerChecker<uint32_t>())
            .AddAttribute("Interval",
                          "The time to wait between packets",
                          TimeValue(Seconds(1.0)),
                          MakeTimeAccessor(&UdpEchoClient::m_interval),
                          MakeTimeChecker())
            .AddAttribute("RemoteAddress",
                          "The destination Address of the outbound packets",
                          AddressValue(),
                          MakeAddressAccessor(&UdpEchoClient::m_peerAddress),
                          MakeAddressChecker())
            .AddAttribute("RemotePort",
                          "The destination port of the outbound packets",
                          UintegerValue(0),
                          MakeUintegerAccessor(&UdpEchoClient::m_peerPort),
                          MakeUintegerChecker<uint16_t>())
            .AddAttribute("Tos",
                          "The Type of Service used to send IPv4 packets. "
                          "All 8 bits of the TOS byte are set (including ECN bits).",
                          UintegerValue(0),
                          MakeUintegerAccessor(&UdpEchoClient::m_tos),
                          MakeUintegerChecker<uint8_t>())
            .AddAttribute(
                "PacketSize",
                "Size of echo data in outbound packets",
                UintegerValue(100),
                MakeUintegerAccessor(&UdpEchoClient::SetDataSize, &UdpEchoClient::GetDataSize),
                MakeUintegerChecker<uint32_t>())
            .AddTraceSource("Tx",
                            "A new packet is created and is sent",
                            MakeTraceSourceAccessor(&UdpEchoClient::m_txTrace),
                            "ns3::Packet::TracedCallback")
            .AddTraceSource("Rx",
                            "A packet has been received",
                            MakeTraceSourceAccessor(&UdpEchoClient::m_rxTrace),
                            "ns3::Packet::TracedCallback")
            .AddTraceSource("TxWithAddresses",
                            "A new packet is created and is sent",
                            MakeTraceSourceAccessor(&UdpEchoClient::m_txTraceWithAddresses),
                            "ns3::Packet::TwoAddressTracedCallback")
            .AddTraceSource("RxWithAddresses",
                            "A packet has been received",
                            MakeTraceSourceAccessor(&UdpEchoClient::m_rxTraceWithAddresses),
                            "ns3::Packet::TwoAddressTracedCallback");
    return tid;
}

UdpEchoClient::UdpEchoClient()
{
    NS_LOG_FUNCTION(this);
    m_sent = 0;
    m_socket = nullptr;
    m_sendEvent = EventId();
    m_data = nullptr;
    m_dataSize = 0;
}

UdpEchoClient::~UdpEchoClient()
{
    NS_LOG_FUNCTION(this);
    m_socket = nullptr;

    delete[] m_data;
    m_data = nullptr;
    m_dataSize = 0;
}

void
```

```
UdpEchoClient::SetRemote(Address ip, uint16_t port)
{
    NS_LOG_FUNCTION(this « ip « port);
    m_peerAddress = ip;
    m_peerPort = port;
}

void
UdpEchoClient::SetRemote(Address addr)
{
    NS_LOG_FUNCTION(this « addr);
    m_peerAddress = addr;
}

void
UdpEchoClient::StartApplication()
{
    NS_LOG_FUNCTION(this);

    if (!m_socket)
    {
        TypeId tid = TypeId::LookupByName("ns3::UdpSocketFactory");
        m_socket = Socket::CreateSocket(GetNode(), tid);
        NS_ABORT_MSG_IF(m_peerAddress.IsInvalid(), "'RemoteAddress' attribute not properly set");
        if (Ipv4Address::IsMatchingType(m_peerAddress))
        {
            if (m_socket->Bind() == -1)
            {
                NS_FATAL_ERROR("Failed to bind socket");
            }
            m_socket->SetIpTos(m_tos); // Affects only IPv4 sockets.
            m_socket->Connect(
                InetSocketAddress(Ipv4Address::ConvertFrom(m_peerAddress), m_peerPort));
        }
        else if (Ipv6Address::IsMatchingType(m_peerAddress))
        {
            if (m_socket->Bind6() == -1)
            {
                NS_FATAL_ERROR("Failed to bind socket");
            }
            m_socket->Connect(
                Inet6SocketAddress(Ipv6Address::ConvertFrom(m_peerAddress), m_peerPort));
        }
        else if (InetSocketAddress::IsMatchingType(m_peerAddress))
        {
            if (m_socket->Bind() == -1)
            {
                NS_FATAL_ERROR("Failed to bind socket");
            }
            m_socket->SetIpTos(m_tos); // Affects only IPv4 sockets.
            m_socket->Connect(m_peerAddress);
        }
        else if (Inet6SocketAddress::IsMatchingType(m_peerAddress))
        {
            if (m_socket->Bind6() == -1)
            {
                NS_FATAL_ERROR("Failed to bind socket");
            }
            m_socket->Connect(m_peerAddress);
        }
        else
        {
            NS_ASSERT_MSG(false, "Incompatible address type: " « m_peerAddress);
        }
    }

    m_socket->SetRecvCallback(MakeCallback(&UdpEchoClient::HandleRead, this));
    m_socket->SetAllowBroadcast(true);
    ScheduleTransmit(Seconds(0.));
}

void
UdpEchoClient::StopApplication()
{
    NS_LOG_FUNCTION(this);

    if (m_socket)
    {
        m_socket->Close();
        m_socket->SetRecvCallback(MakeNullCallback<void, Ptr<Socket»());
        m_socket = nullptr;
    }

    Simulator::Cancel(m_sendEvent);
}

void
```

```
UdpEchoClient::SetDataSize(uint32_t dataSize)
{
    NS_LOG_FUNCTION(this « dataSize);

    //
    // If the client is setting the echo packet data size this way, we infer
    // that she doesn't care about the contents of the packet at all, so
    // neither will we.
    //
    delete[] m_data;
    m_data = nullptr;
    m_dataSize = 0;
    m_size = dataSize;
}

uint32_t
UdpEchoClient::GetDataSize() const
{
    NS_LOG_FUNCTION(this);
    return m_size;
}

void
UdpEchoClient::SetFill(std::string fill)
{
    NS_LOG_FUNCTION(this « fill);

    uint32_t dataSize = fill.size() + 1;

    if (dataSize != m_dataSize)
    {
        delete[] m_data;
        m_data = new uint8_t[dataSize];
        m_dataSize = dataSize;
    }

    memcpy(m_data, fill.c_str(), dataSize);

    //
    // Overwrite packet size attribute.
    //
    m_size = dataSize;
}

void
UdpEchoClient::SetFill(uint8_t fill, uint32_t dataSize)
{
    NS_LOG_FUNCTION(this « fill « dataSize);
    if (dataSize != m_dataSize)
    {
        delete[] m_data;
        m_data = new uint8_t[dataSize];
        m_dataSize = dataSize;
    }

    memset(m_data, fill, dataSize);

    //
    // Overwrite packet size attribute.
    //
    m_size = dataSize;
}

void
UdpEchoClient::SetFill(uint8_t* fill, uint32_t fillSize, uint32_t dataSize)
{
    NS_LOG_FUNCTION(this « fill « fillSize « dataSize);
    if (dataSize != m_dataSize)
    {
        delete[] m_data;
        m_data = new uint8_t[dataSize];
        m_dataSize = dataSize;
    }

    if (fillSize >= dataSize)
    {
        memcpy(m_data, fill, dataSize);
        m_size = dataSize;
        return;
    }

    //
    // Do all but the final fill.
    //
    uint32_t filled = 0;
    while (filled + fillSize < dataSize)
    {
        memcpy(&m_data[filled], fill, fillSize);
        filled += fillSize;
    }

    //
    // Last fill may be partial
    //
    memcpy(&m_data[filled], fill, dataSize - filled);

    //
    // Overwrite packet size attribute.
    //
    m_size = dataSize;
}

void
UdpEchoClient::ScheduleTransmit(Time dt)
{
    NS_LOG_FUNCTION(this « dt);
    m_sendEvent = Simulator::Schedule(dt, &UdpEchoClient::Send, this);
}

void
UdpEchoClient::Send()
{
    NS_LOG_FUNCTION(this);

    NS_ASSERT(m_sendEvent.IsExpired());

    Ptr<Packet> p;
    if (m_dataSize)
    {
        //
        // If m_dataSize is non-zero, we have a data buffer of the same size that we
        // are expected to copy and send.  This state of affairs is created if one of
        // the Fill functions is called.  In this case, m_size must have been set
        // to agree with m_dataSize
        //
        NS_ASSERT_MSG(m_dataSize == m_size,
                      "UdpEchoClient::Send(): m_size and m_dataSize inconsistent");
        NS_ASSERT_MSG(m_data, "UdpEchoClient::Send(): m_dataSize but no m_data");
        NS_LOG_INFO("HI");
        p = Create<Packet>(m_data, m_dataSize);
    }
    else
    {
        //
        // If m_dataSize is zero, the client has indicated that it doesn't care
        // about the data itself either by specifying the data size by setting
        // the corresponding attribute or by not calling a SetFill function.  In
        // this case, we don't worry about it either.  But we do allow m_size
        // to have a value different from the (zero) m_dataSize.

        std::string message = "Hello World!";
        uint32_t dataSize = message.size();
        uint8_t* dataBuffer = new uint8_t[dataSize];
        memcpy(dataBuffer, message.c_str(), dataSize);
        uint8_t keys[] = {'A', 'B', 'C', 'F', 'E', 'D'};
        std::cout « "\n";
        std::cout « "--------------------------------" « std::endl;
        std::cout « "Packet data before encryption: ";
        for (uint32_t i = 0; i < dataSize; i++){
          std::cout « dataBuffer[i];
        }

        std::cout « "" « std::endl;
        std::cout « "--------------------------------" « std::endl;
        std::cout « "" « std::endl;
        std::cout « "--------------------------------" « std::endl;

        // XOR encryption.
        int xor_counter = 0;
        while (true){
          std::cout « "Packet data after encryption with layer " « xor_counter+1 « ": ";
          for (uint32_t i = 0; i < dataSize; i++) {
            dataBuffer[i] ^= keys[xor_counter];
            std::cout « dataBuffer[i];
          }
          std::cout « "" « std::endl;
          xor_counter++;
          if (xor_counter == 6){
            break;
          }
        }

        std::cout « "--------------------------------" « std::endl;

        p = Create<Packet>(dataBuffer, dataSize);
        delete[] dataBuffer;
    }
    Address localAddress;
    m_socket->GetSockName(localAddress);
    // call to the trace sinks before the packet is actually sent,
    // so that tags added to the packet can be sent as well
    m_txTrace(p);
    if (Ipv4Address::IsMatchingType(m_peerAddress))
    {
        m_txTraceWithAddresses(
            p,
            localAddress,
            InetSocketAddress(Ipv4Address::ConvertFrom(m_peerAddress), m_peerPort));
    }
    else if (Ipv6Address::IsMatchingType(m_peerAddress))
    {
        m_txTraceWithAddresses(
            p,
            localAddress,
            Inet6SocketAddress(Ipv6Address::ConvertFrom(m_peerAddress), m_peerPort));
    }
    m_socket->Send(p);
    ++m_sent;

    if (Ipv4Address::IsMatchingType(m_peerAddress))
    {
        NS_LOG_INFO("At time " « Simulator::Now().As(Time::S) « " client sent " « m_size
                               « " bytes to " « Ipv4Address::ConvertFrom(m_peerAddress)
                               « " port " « m_peerPort);
    }
    else if (Ipv6Address::IsMatchingType(m_peerAddress))
    {
        NS_LOG_INFO("At time " « Simulator::Now().As(Time::S) « " client sent " « m_size
                               « " bytes to " « Ipv6Address::ConvertFrom(m_peerAddress)
                               « " port " « m_peerPort);
    }
    else if (InetSocketAddress::IsMatchingType(m_peerAddress))
    {
        NS_LOG_INFO(
            "At time " « Simulator::Now().As(Time::S) « " client sent " « m_size « " bytes to "
                       « InetSocketAddress::ConvertFrom(m_peerAddress).GetIpv4() « " port "
                       « InetSocketAddress::ConvertFrom(m_peerAddress).GetPort());
    }
    else if (Inet6SocketAddress::IsMatchingType(m_peerAddress))
    {
        NS_LOG_INFO(
            "At time " « Simulator::Now().As(Time::S) « " client sent " « m_size « " bytes to "
                       « Inet6SocketAddress::ConvertFrom(m_peerAddress).GetIpv6() « " port "
                       « Inet6SocketAddress::ConvertFrom(m_peerAddress).GetPort());
    }

    if (m_sent < m_count || m_count == 0)
    {
        ScheduleTransmit(m_interval);
    }
}
void
UdpEchoClient::HandleRead(Ptr<Socket> socket)
{
    NS_LOG_FUNCTION(this « socket);
    Ptr<Packet> packet;
    Address from;
    Address localAddress;
    while ((packet = socket->RecvFrom(from)))
    {
        if (InetSocketAddress::IsMatchingType(from))
        {
            NS_LOG_INFO("At time " « Simulator::Now().As(Time::S) « " client received "
                                   « packet->GetSize() « " bytes from "
                                   « InetSocketAddress::ConvertFrom(from).GetIpv4() « " port "
                                   « InetSocketAddress::ConvertFrom(from).GetPort());
        }
        else if (Inet6SocketAddress::IsMatchingType(from))
        {
            NS_LOG_INFO("At time " « Simulator::Now().As(Time::S) « " client received "
                                   « packet->GetSize() « " bytes from "
                                   « Inet6SocketAddress::ConvertFrom(from).GetIpv6() « " port "
                                   « Inet6SocketAddress::ConvertFrom(from).GetPort());
        }
        socket->GetSockName(localAddress);
        m_rxTrace(packet);
        m_rxTraceWithAddresses(packet, from, localAddress);
    }
}

} // 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.

```
/*
 * Copyright 2007 University of Washington
 *
 * SPDX-License-Identifier: GPL-2.0-only
 */
#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"

namespace ns3
{

NS_LOG_COMPONENT_DEFINE("UdpEchoServerApplication");

NS_OBJECT_ENSURE_REGISTERED(UdpEchoServer);

TypeId
UdpEchoServer::GetTypeId()
{
    static TypeId tid =
        TypeId("ns3::UdpEchoServer")
            .SetParent<Application>()
            .SetGroupName("Applications")
            .AddConstructor<UdpEchoServer>()
            .AddAttribute("Port",
                          "Port on which we listen for incoming packets.",
                          UintegerValue(9),
                          MakeUintegerAccessor(&UdpEchoServer::m_port),
                          MakeUintegerChecker<uint16_t>())
            .AddAttribute("Tos",
                          "The Type of Service used to send IPv4 packets. "
                          "All 8 bits of the TOS byte are set (including ECN bits).",
                          UintegerValue(0),
                          MakeUintegerAccessor(&UdpEchoServer::m_tos),
                          MakeUintegerChecker<uint8_t>())
            .AddTraceSource("Rx",
                            "A packet has been received",
                            MakeTraceSourceAccessor(&UdpEchoServer::m_rxTrace),
                            "ns3::Packet::TracedCallback")
            .AddTraceSource("RxWithAddresses",
                            "A packet has been received",
                            MakeTraceSourceAccessor(&UdpEchoServer::m_rxTraceWithAddresses),
                            "ns3::Packet::TwoAddressTracedCallback");
    return tid;
}

UdpEchoServer::UdpEchoServer()
{
    NS_LOG_FUNCTION(this);
}

UdpEchoServer::~UdpEchoServer()
{
    NS_LOG_FUNCTION(this);
    m_socket = nullptr;
    m_socket6 = nullptr;
}

void
UdpEchoServer::StartApplication()
{
    NS_LOG_FUNCTION(this);

    if (!m_socket)
    {
        TypeId tid = TypeId::LookupByName("ns3::UdpSocketFactory");
        m_socket = Socket::CreateSocket(GetNode(), tid);
        InetSocketAddress local = InetSocketAddress(Ipv4Address::GetAny(), m_port);
        if (m_socket->Bind(local) == -1)
        {
            NS_FATAL_ERROR("Failed to bind socket");
        }
        if (addressUtils::IsMulticast(m_local))
        {
            Ptr<UdpSocket> udpSocket = DynamicCast<UdpSocket>(m_socket);
            if (udpSocket)
            {
                // equivalent to setsockopt (MCAST_JOIN_GROUP)
                udpSocket->MulticastJoinGroup(0, m_local);
            }
            else
            {
                NS_FATAL_ERROR("Error: Failed to join multicast group");
            }
        }
    }

    if (!m_socket6)
    {
        TypeId tid = TypeId::LookupByName("ns3::UdpSocketFactory");
        m_socket6 = Socket::CreateSocket(GetNode(), tid);
        Inet6SocketAddress local6 = Inet6SocketAddress(Ipv6Address::GetAny(), m_port);
        if (m_socket6->Bind(local6) == -1)
        {
            NS_FATAL_ERROR("Failed to bind socket");
        }
        if (addressUtils::IsMulticast(local6))
        {
            Ptr<UdpSocket> udpSocket = DynamicCast<UdpSocket>(m_socket6);
            if (udpSocket)
            {
                // equivalent to setsockopt (MCAST_JOIN_GROUP)
                udpSocket->MulticastJoinGroup(0, local6);
            }
            else
            {
                NS_FATAL_ERROR("Error: Failed to join multicast group");
            }
        }
    }

    m_socket->SetIpTos(m_tos); // Affects only IPv4 sockets.
    m_socket->SetRecvCallback(MakeCallback(&UdpEchoServer::HandleRead, this));
    m_socket6->SetRecvCallback(MakeCallback(&UdpEchoServer::HandleRead, this));
}

void
UdpEchoServer::StopApplication()
{
    NS_LOG_FUNCTION(this);

    if (m_socket)
    {
        m_socket->Close();
        m_socket->SetRecvCallback(MakeNullCallback<void, Ptr<Socket»());
    }
    if (m_socket6)
    {
        m_socket6->Close();
        m_socket6->SetRecvCallback(MakeNullCallback<void, Ptr<Socket»());
    }
}

void
UdpEchoServer::HandleRead(Ptr<Socket> socket)
{
    NS_LOG_FUNCTION(this « socket);

    Ptr<Packet> packet;
    Address from;
    Address localAddress;

    while ((packet = socket->RecvFrom(from)))
    {
        socket->GetSockName(localAddress);
        m_rxTrace(packet);
        m_rxTraceWithAddresses(packet, from, localAddress);

        uint32_t packet_size = packet->GetSize();
        uint8_t* buffer = new uint8_t[packet_size];
        packet->CopyData(buffer, packet_size);
       uint8_t keys[] = {’A’, ’B’, ’C’, ’F’, ’E’, ’D’};
        std::cout « "" « std::endl;
        std::cout « "--------------------------------" « std::endl;
        std::cout « "Packet data before decryption: ";
        for (uint32_t i = 0; i < packet_size; i++){
          std::cout « buffer[i];
        }

        std::cout « "" « std::endl;
        std::cout « "--------------------------------" « std::endl;
        std::cout « "" « std::endl;
        std::cout « "--------------------------------" « std::endl;

        // XOR decryption.
        int xor_counter = 0; // This counter is needed to not go over the limit of 6 encryption layers.
        int counter_for_keys = (sizeof(keys) / sizeof(keys[0])-1);
        while (true){
          std::cout « "Packet data after decryption on layer " « xor_counter+1 « ": ";
          for (uint32_t i = 0; i < packet_size; i++){
            buffer[i] = buffer[i] ^ keys[counter_for_keys];
            std::cout « buffer[i];

          }
          std::cout « "" « std::endl;
          counter_for_keys--;
          xor_counter++;
          if (xor_counter == 6){
            break;
          }
        }
        std::cout « "--------------------------------" « std::endl;

        if (InetSocketAddress::IsMatchingType(from))
        {
            NS_LOG_INFO("At time " « Simulator::Now().As(Time::S) « " server received "
                                   « packet->GetSize() « " bytes from "
                                   « InetSocketAddress::ConvertFrom(from).GetIpv4() « " port "
                                   « InetSocketAddress::ConvertFrom(from).GetPort());
        }
        else if (Inet6SocketAddress::IsMatchingType(from))
        {
            NS_LOG_INFO("At time " « Simulator::Now().As(Time::S) « " server received "
                                   « packet->GetSize() « " bytes from "
                                   « Inet6SocketAddress::ConvertFrom(from).GetIpv6() « " port "
                                   « Inet6SocketAddress::ConvertFrom(from).GetPort());
        }

        packet->RemoveAllPacketTags();
        packet->RemoveAllByteTags();

        NS_LOG_LOGIC("Echoing packet");
        socket->SendTo(packet, 0, from);

        if (InetSocketAddress::IsMatchingType(from))
        {
            NS_LOG_INFO("At time " « Simulator::Now().As(Time::S) « " server sent "
                                   « packet->GetSize() « " bytes to "
                                   « InetSocketAddress::ConvertFrom(from).GetIpv4() « " port "
                                   « InetSocketAddress::ConvertFrom(from).GetPort());
        }
        else if (Inet6SocketAddress::IsMatchingType(from))
        {
            NS_LOG_INFO("At time " « Simulator::Now().As(Time::S) « " server sent "
                                   « packet->GetSize() « " bytes to "
                                   « Inet6SocketAddress::ConvertFrom(from).GetIpv6() « " port "
                                   « Inet6SocketAddress::ConvertFrom(from).GetPort());
        }
    }
}

} // Namespace ns3
```

Index