Projeto de Simulação no NS-3 (Rede sem Fio)

Maria Inês da Silva, Marcelo Pinheiro Machado
Universidade de Campinas – Instituto de Computação, Campinas, São Paulo, Brasil
Contact: minessilva91@gmail.com RA190927, marcelopmachado@gmail.com, RA 210402

Resumo —
Este relatório apresenta os passos executados na construção de uma simulação de rede sem fio utilizando o simulador NS-3. A simulação consiste na avaliação de dispositivos que se comunicam através de uma rede sem fio, enviando pacotes para um servidor ligado o ponto de acesso (AP) sem fio, utilizando modelos de tráfego em rajada e CBR.
Os cenários avaliados levaram em consideração nós estáticos com 20% de efetividade em relação ao número total de pontos (10,20,30 e 40 pontos) em rajadas e CBR para UDP, TCP e 50% UDP e 50% TCP, medindo vazão, perda e atraso, para os pontos mais próximo e mais distante do AP.

I. Introdução

Este relatório técnico apresenta as configurações utilizadas para a implementação de três situações simuladas, sendo a medição de vazão, perda e atraso para os pontos mais perto e mais distante, em CBR (Constant Bit Rate) e rajada, para TCP, UDP e 50%-50% (UDP e TCP).

Foi utilizado o simulador NS-3 na versão 3.26. O código que realiza as simulações foi desenvolvido em C++ e as informações coletadas, foram armazenadas em arquivos (csv - Comma-separated values). Foi ainda utilizado um script shell para possibilitar a repetição da execução (30 repetições) de cada simulação para que se pudesse alcançar o intervalo de confiança de 95%.

Este relatório descreve as alterações que uma rede sem fio pode sofrer a medida em que se aumenta a quantidade de pontos efetivos levando em consideração a distância do AP, proporcionando uma análise comparativa entre os tipos de tráfegos, juntamente com a variação do tipo de protocolo utilizado (UDP ou TCP).

Na Seção II são apresentados os requisitos do projeto e critérios da simulação e na Seção III as premissas para desenvolvimento do código e como foram gerados os resultados.

Na Seção IV apresentamos as análises realizadas e conclusões

II. Projeto e Critérios de Implementação

O projeto a ser executado consiste na simulação de uma Rede sem Fio 802.11 no modo infraestruturado com um Ponto de Acesso realizando a interconexão entre os pontos sem-fio fixos a um servidor conectado a um ponto de acesso através uma conexão ponto a ponto.

A execução do projeto divide-se na simulação de três situações ou cenários envolvendo a rede sem fio descrita. As três situações dizem respeito ao protocolo utilizado, TCP, UDP e mix 50%-50% TCP e UDP.

Estarão sendo medidos, vazão de dados enviados, atraso no recebimento de pacotes e taxa de pacotes perdidos na transmissão

Os dispositivos estão distribuídos numa área de 100 m2 de forma aleatória e fixos, isto é, sem mobilidade.

O ponto de acesso, em todas as simulações, é definido no centro desse espaço (posição x=50 e y=50).

A comunicação entre os dispositivos e o ponto de acesso e realizada utilizando dois modelos de tráfego, um utilizando uma taxa constante de transmissão CBR e outro realizando tráfego em rajada. Foi utilizado o tamanho de quadro de 512(CBR) e 1500 bytes (rajada), com intervalo de confiança de 95% para as simulações.

III. Desenvolvimento

O código utilizado teve como base o exemplo 3 do próprio Simulador NS3 como diversas alterações para que pudesse levar em consideração a utilização dos protocolos TCP, UDP e mix de 50%-50%.

Além disso foram incluídas funcionalidades para que se pudesse alterar o número de pontos sem-fio fixos e uma maneira de se atribuir os protocolos para cada ponto.

Casa simulação gerou resultados em formato CSV para os pontos mais perto, mais longo e também para a média das interações.

Para que se pudesse atingir o intervalo que confiança de 95%, foi necessário realizar "n" simulações de cada caso, isto é, alterando-se o protocolo (UDP, TCP e mix) e alterando-se o número de pontos da rede sem fio. Um script shell foi desenvolvido para que se pudesse rodar as "n" simulações de cada cenário necessário. Foi utilizado o valor de "n" = 30.

IV. Resultados

ATRASO: Uma das métricas de avaliação dos cenários foi por meio da medida do tempo médio de atraso na entrega dos pacotes enviados pelos dispositivos para o servidor por meio de um ponto de acesso sem fio, para o ponto mais próximo e para o ponto mais longe.

Na Figura 1 e 2 são apresentados os resultados dos cenários por protocolo para os pontos mais perto e mais longe do AP. É apresentado os dados referente ao tempo de atraso para o modelo de tráfego em rajada e tráfego constante de dados (CBR), para os protocolos TCP, UDP e TCP/UDP (50%/50%).

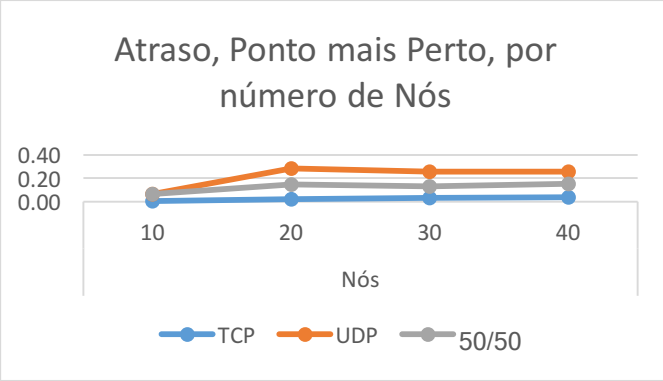


Fig.1

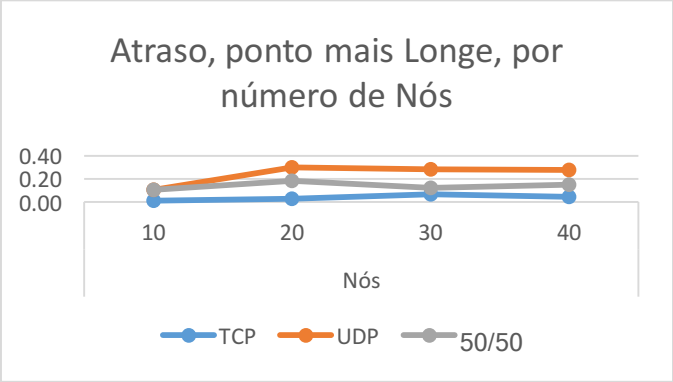


Fig. 2

Pode-se observar que houve um atraso maior para o cenário que utiliza o UDP (CBR) em comparação com o TCP que utiliza o modelo de rajada. Nota-se também que houve uma grande variação inicial no atraso em relação a variação do número de dispositivos (número de dispositivos de 10 para 20) para o UDP, e em seguida a variação do número de atraso permaneceu pequena tanto para o UDP, TCP quanto para o mix 50%-50%.

O comportamento foi semelhante tanto para o ponto mais perto quanto para o mais longe.

PERDA: A perda de pacotes apresentadas durante todo o tempo de execução da simulação, 60 segundos, são exibidas para cada um dos três cenários abordados, variando o modelo de trafego (CBR e rajada) e mix de 50%-50%.

As Figuras 3 e 4 apresentam os resultados para os cenários de ponto mais perto e mais longe do AP, respectivamente.

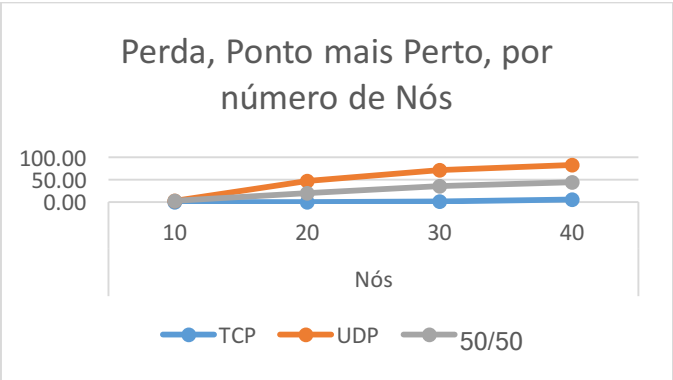


Fig. 3

No cenário de Ponto mais perto por número de nós (Figura 3) pode-se notar que a perda para o protocolo UDP aumenta consideravelmente em relação ao aumento de número de nós. Para o TCP esse número permaneceu zerado ou muito próximo disso.

No cenário de Ponto mais longe por número de nós (Figura 4), foi observado também que houve uma grande perda para o protocolo UDP, entretanto a surpresa foi em relação ao

protocolo TCP que deveria ter ficado em zero ou próximo disso. A explicação pode estar relacionada a deterioração do sinal causada pelo aumento da distância do AP.

A utilização do mix de 50%-50% de UDP e TCP causou um comportamento coerente entre a utilização dos dois modelos puros.

VAZÃO: Para as simulações foi determinado que a taxa de dados limite da rede sem fio fosse 512 Kbps e o link do AP até o servidor fosse de 5 Mbps.

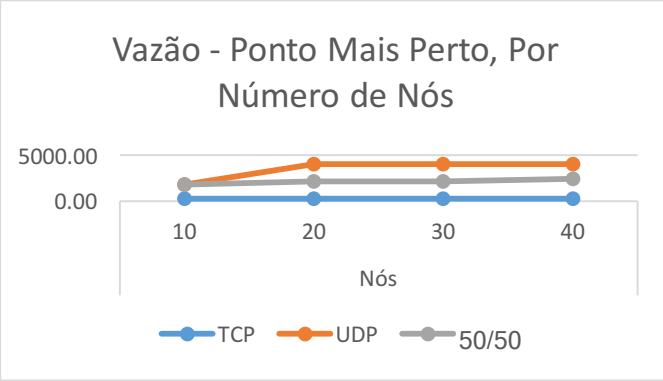


Fig. 5

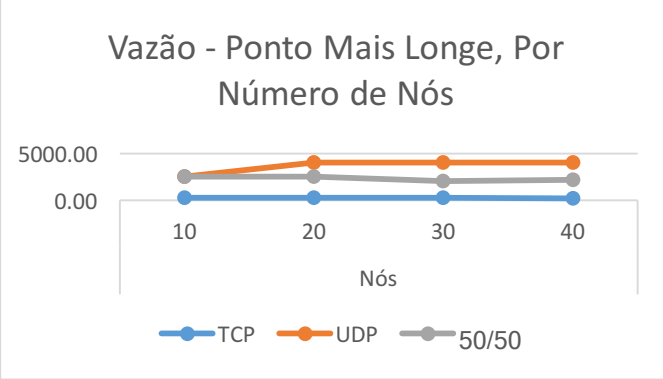


Fig. 6

Na Figura 5 e Figura 6 são apresentadas a Vazão para o Ponto mais perto, e Ponto mais longe, respectivamente, utilizando-se TCP, UDP e o mix 50%-50%. Nota-se que a vazão para o UDP (CBR) após um aumento inicial em relação ao número de dispositivos, confirma a taxa constante de transmissão.

Em relação ao TCP, observa-se que a Vazão é pequena em relação ao UDP, entretanto apresenta uma diminuição e variação pequena em relação ao número de nós.

A Vazão do UDP (CBR) é bem maior quando comparada ao TCP (rajada), tanto para os cenários do ponto mais perto, quanto no cenário do ponto mais longe.

A utilização do mix 50%-50% causou um comportamento do valor da vazão entre os dois protocolos.

V. Conclusões

O atraso na entrega de pacotes, os dispositivos que utilizaram CBR como modelo de transmissão de dados apresentaram um maior tempo de atraso comparados com os dispositivos que realizaram tráfego em rajada.

Comparando-se o cenário do ponto mais perto com o mais longe, houve um comportamento semelhante, isto é, a distância não afetou o atraso.

Em relação a perda de pacotes, o modelo que utiliza UDP apresentou valores bem maiores em relação ao modelo que utiliza o TCP (rajada). A distância entre o AP causou uma perda maior ao TCP quando se aumenta o número de pontos, entretanto essa taxa (perda) ainda é consideravelmente menor que a do UDP.

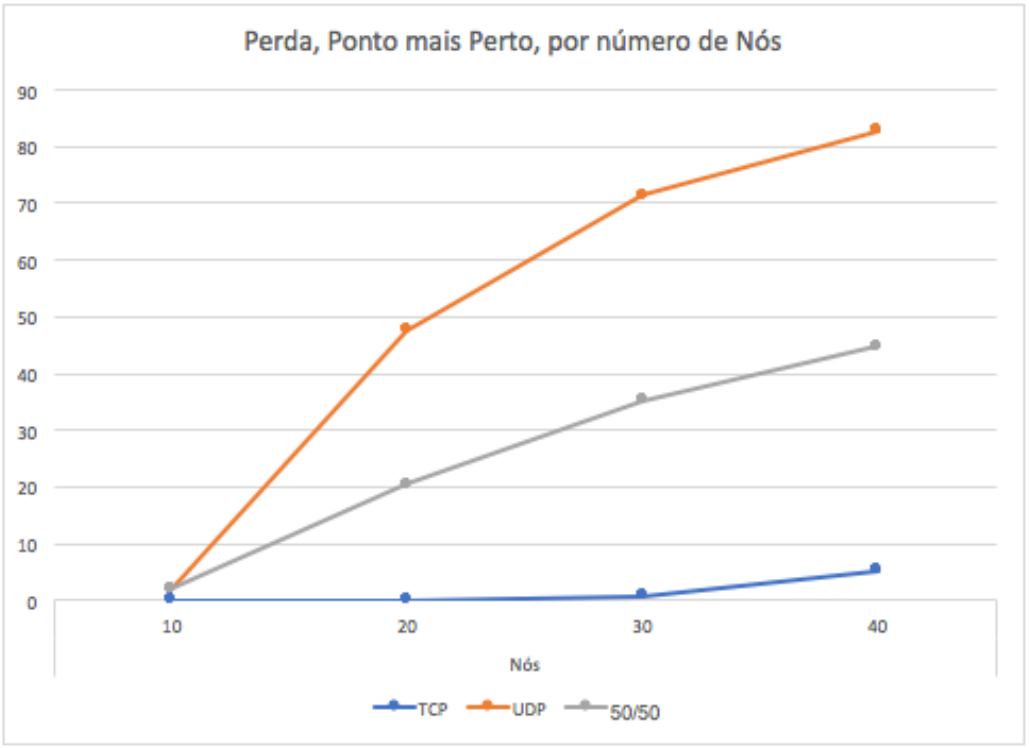
Analisando a vazão, quando se utiliza o TCP, essa apresenta uma ligeira queda quando se aumenta o número de nós, entretanto quando se verifica os valores de vazão do UDP, esses são significamente maiores quando comparados aqueles valores TCP. Esse comportamento pouco se altera em relação à distância do AP, isto é, ponto mais perto e ponto mais longe

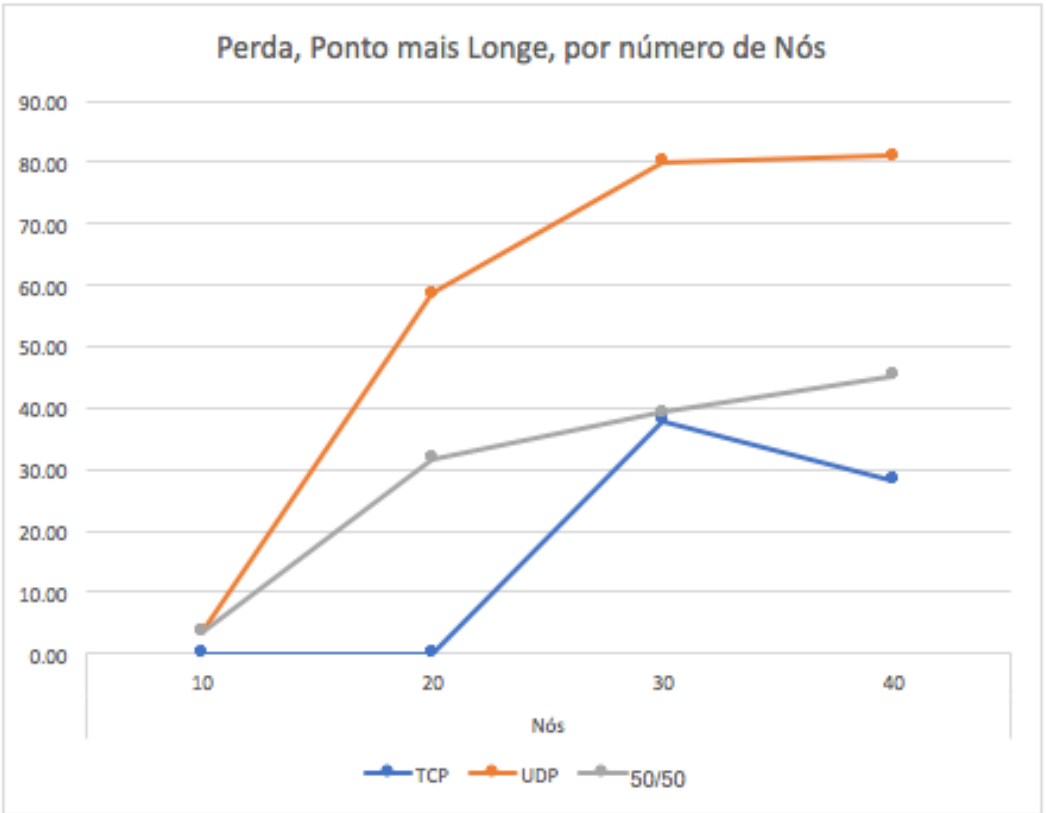
Referência

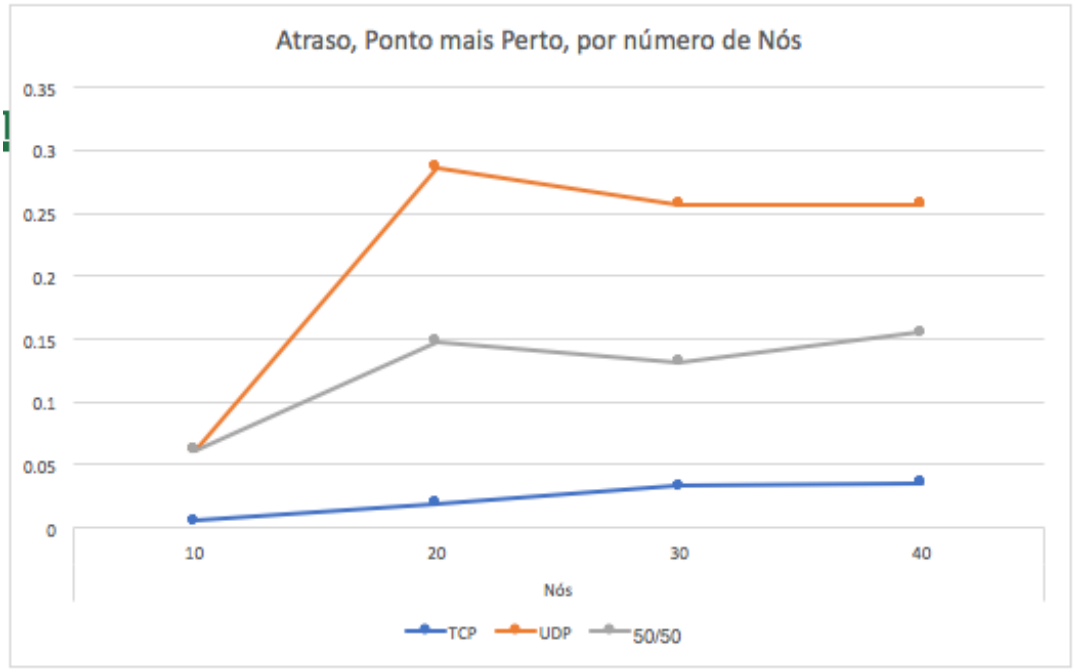
[1] Henderson, T. R., Lacage, M., Riley, G. F., Dowell, C., and Kopena, J. (2008). Network simulations with the ns-3 simulator. SIGCOMM demonstration, 15:17.

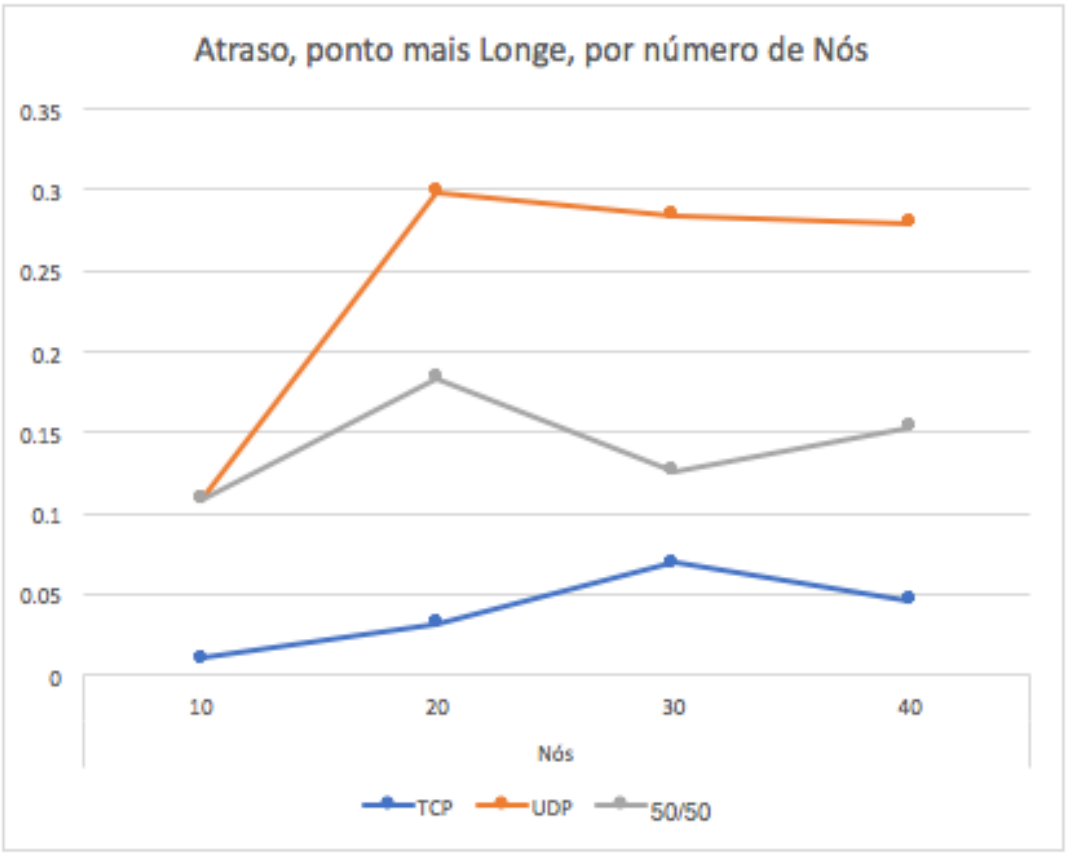
[2] Riley, G. F. and Henderson, T. R. (2010). The ns-3 network simulator. In Modeling and Tools for Network Simulation, pages 15–34. Springer.

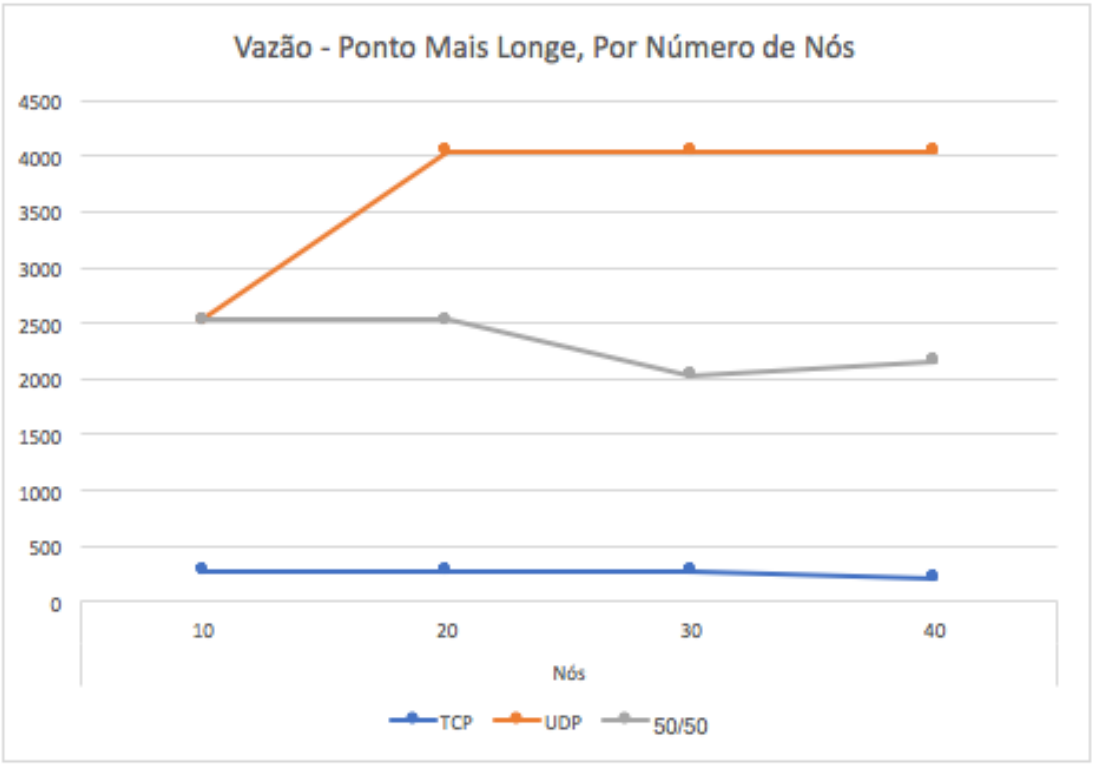
ANEXOS

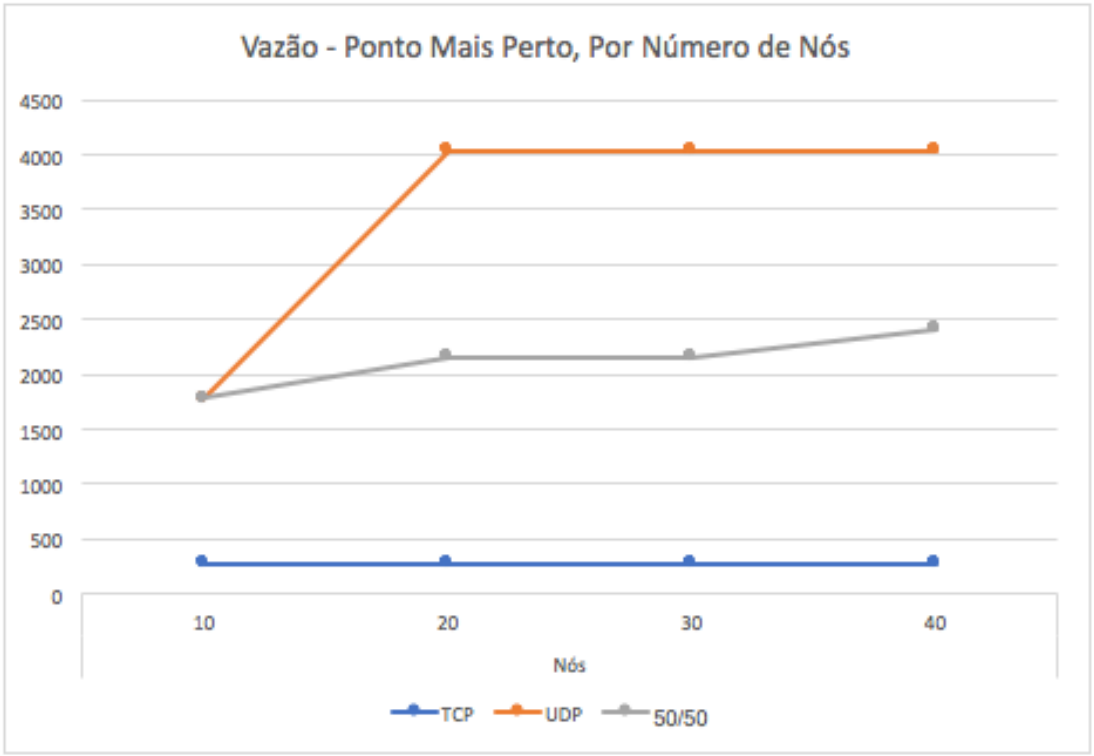

Perda, Ponto mais Perto, por número de Nós
90
80
70
60
50
40
30
20
10
0
10
20
30
40
Nós
TCP
UDP
50/50


Perda, Ponto mais Longe, por número de Nós
90.00
80.00
70.00
60.00
50.00
40.00
30.00
20.00
10.00
0.00
10
20
30
40
Nós
TCP
UDP
50/50


Atraso, Ponto mais Perto, por número de Nós
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
10
20
30
40
Nós
TCP
UDP
50/50


Atraso, ponto mais Longe, por número de Nós
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
10
20
30
40
Nós
TCP
UDP
50/50


Vazão - Ponto Mais Longe, Por Número de Nós
4500
4000
3500
3000
2500
2000
1500
1000
500
0
10
20
30
40
Nós
TCP
UDP
50/50


Vazão - Ponto Mais Perto, Por Número de Nós
4500
4000
3500
3000
2500
2000
1500
1000
500
0
10
20
30
40
Nós
TCP
UDP
50/50

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/propagation-module.h"
#include "ns3/flow-monitor-module.h"
#include <iostream>
#include <sstream>
#include <math.h>

// Default Network Topology
//
// Number of wifi or csma nodes can be increased up to 250
//                          |
//               Rank 0   |   Rank 1
// -------------------------|----------------------------
//   Wifi 10.1.3.0
//                AP
//  *    *    *    *
// |    |    |    |    10.1.1.0
// n5   n6   n7   n0 -------------- n1
//                   point-to-point  |
//                                   =
//                                   LAN 10.1.2.0

using namespace ns3;
using namespace std;


int myRand(int min, int max){

  Ptr<UniformRandomVariable> x = CreateObject<UniformRandomVariable> ();
```

```
    x->SetAttribute ("Min", DoubleValue (min));
    x->SetAttribute ("Max", DoubleValue (max));

    int value = x->GetValue ();

    return value;
   }

   //################ funcao_flow #################
   void funcao_flow(FlowMonitorHelper &flowmon, Ptr<FlowMonitor> &monitor, Ipv4InterfaceContainer &devicesIP,
Ipv4InterfaceContainer &p2pdeviceIP,  uint32_t nWifi, uint32_t porcentagem, int tipo_trafego, size_t nodenear, size_t nodefar){

   //variaveis iniciando com zero
    double taxaTransferencia = 0;
    double delay = 0;
    double somaTransferencia = 0.0;
    double mediaVazao = 0.0;
    double mediadelay = 0.0;
    double mediaPerdaPacote = 0.0;
    int count = 0;
    FILE *f;

    monitor->CheckForLostPackets();
    FlowMonitor::FlowStatsContainer stats = monitor->GetFlowStats ();

    Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());

    for (map<FlowId, FlowMonitor::FlowStats>::const_iterator i=stats.begin (); i != stats.end (); ++i, count++){
     Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);

     if (t.destinationAddress == p2pdeviceIP.GetAddress(0)){
      taxaTransferencia = (i->second.txBytes * 8) / ((i->second.timeLastTxPacket - i-
>second.timeFirstTxPacket).GetSeconds());


        cout << "Flowid                    = " << i->first << endl;
        cout << "Endereço de partida         = " << t.sourceAddress << endl;
        cout << "Endereço de destino         = " << t.destinationAddress << endl;
        cout << "Primeiro pacote Tx          = " << i->second.timeFirstTxPacket.GetSeconds() << endl;
        cout << "Primeiro pacote Rx          = " << i->second.timeFirstRxPacket.GetSeconds() << endl;
        cout << "Ultimo pacote Tx            = " << i->second.timeLastTxPacket.GetSeconds() << endl;
        cout << "Ultimo pacote Rx            = " << i->second.timeLastRxPacket.GetSeconds() << endl;
        cout << "Pacotes  Tx               = " << i->second.txPackets << endl;
        cout << "Pacotes  RX               = " << i->second.rxPackets << endl;
        cout << "Pacotes perdidos            = " << i->second.lostPackets << endl;
        cout << "Tx Bytes                  = " << i->second.txBytes << endl;
        cout << "RX bytes                  = " << i->second.rxBytes << endl;
        cout << "Atraso da soma              = " << i->second.delaySum.GetSeconds() << endl;
        cout << "Média de atraso do pacote     = " << i->second.delaySum.GetSeconds()/i->second.rxPackets << endl;
        cout << "Taxa de transferencia recebida = " << taxaTransferencia << " bps" << " " << taxaTransferencia/1024 << " kbps"
<< endl << endl;

       taxaTransferencia = ((taxaTransferencia > 0) ? taxaTransferencia : 0);
       somaTransferencia += taxaTransferencia;
       delay = i->second.delaySum.GetSeconds()/i->second.rxPackets;
```

```
    delay = ((delay > 0) ? delay : 0);
    mediadelay += delay;
    mediaPerdaPacote += i->second.lostPackets/((i->second.timeLastTxPacket - i->second.timeFirstTxPacket).GetSeconds());

      if(devicesIP.GetAddress(nodenear) == t.sourceAddress){
         std::stringstream ss;
         ss <<nWifi<<"_"<<"mais_proximo.csv";
         f = fopen(ss.str().c_str(), "a");
         fprintf(f, "%d;%.2f;%.2f;%d\n", nWifi, taxaTransferencia/1024, delay, i->second.lostPackets);
         fclose(f);
      }
       else if (devicesIP.GetAddress(nodefar) == t.sourceAddress){
         std::stringstream ss;
         ss <<nWifi<<"_"<<"mais_long.csv";
         f = fopen(ss.str().c_str(), "a");
         fprintf(f, "%d;%.2f;%.2f;%d\n", nWifi, taxaTransferencia/1024, delay, i->second.lostPackets);
         fclose(f);
      }
     }
   }



  mediaVazao  = somaTransferencia / porcentagem;
  mediadelay /= porcentagem;
  mediaPerdaPacote /= porcentagem;

  cout << "Valor de Vazão    : " << mediaVazao/1024 << " kbps"<< endl;
  cout << "Valor de Atraso   : " << mediadelay << endl;
  cout << "Valor de Perda    : " << mediaPerdaPacote << endl;



  // Caso seja escolhido as opções 0 será tcp, opção 1 será 50/50 e opção 2 será udp

      std::stringstream ss;
      switch(tipo_trafego){
            case 0:
                 ss <<nWifi<<"_"<<"tcp.csv";
            break;
            case 1:
                ss <<nWifi<<"_"<<"5050.csv";
            break;
            case 2:
                ss <<nWifi<<"_"<<"udp.csv";
            break;
            default:
            return;
      }
      f = fopen(ss.str().c_str(), "a");
      fprintf(f, "%d;%.2f;%.10f;%.2f;%.2f\n", nWifi, mediaVazao/1024, mediadelay, mediaPerdaPacote, somaTransferencia);


  fclose(f);
}
```

```
//#################### TCP ###################
void tcp (uint32_t porcentagem, Ipv4InterfaceContainer &csmaInterfaces, NodeContainer &wifiStaNodes, NodeContainer &wifiApNode)
{
 ApplicationContainer serverApp;
 ApplicationContainer sinkApp;

 std::ostringstream ossOnTime;
 ossOnTime << "ns3::ConstantRandomVariable[Constant=" << 0.001 << "]";
 std::ostringstream ossOffTime;
 ossOffTime << "ns3::ConstantRandomVariable[Constant=" << 0.001 << "]";

 /* Install TCP/UDP Transmitter on the station */
 for (uint32_t i = 0; i < porcentagem; i++){
 /* Install TCP Receiver on the access point */
  PacketSinkHelper sinkHelper ("ns3::TcpSocketFactory", InetSocketAddress (csmaInterfaces.GetAddress(0), i+10000));
  sinkApp = sinkHelper.Install (wifiStaNodes.Get(i));
  sinkApp.Add (sinkHelper.Install (wifiApNode.Get(0)));
  sinkApp.Start (Seconds (0.0));
  OnOffHelper server ("ns3::TcpSocketFactory", (InetSocketAddress (csmaInterfaces.GetAddress(0), i+10000)));
  server.SetAttribute ("PacketSize", UintegerValue (1484));
  server.SetAttribute ("OnTime", StringValue (ossOnTime.str()));
  server.SetAttribute ("OffTime", StringValue (ossOffTime.str()));
  server.SetAttribute ("DataRate", StringValue ("512kbps"));
  serverApp = server.Install (wifiStaNodes.Get(i));
  serverApp.Start (Seconds (1));
 }
  serverApp.Stop(Seconds(60+1));
}

//#################### UDP ###################
void udp(NodeContainer &wifiApNode, float time, Ipv4InterfaceContainer &csmaInterfaces, uint32_t porcentagem, uint32_t nWifi, NodeContainer &wifiStaNodes){

 UdpEchoServerHelper echoServer (9);

 ApplicationContainer serverApps = echoServer.Install (wifiApNode.Get (1));
 serverApps.Start (Seconds (1.0));
 serverApps.Stop (Seconds (time)); //pega o tempo da variavel TIME

UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (0), 9);
echoClient.SetAttribute ("Interval", TimeValue (Seconds (0.001)));
echoClient.SetAttribute ("DataRate", StringValue ("512kbps"));
echoClient.SetAttribute ("PacketSize", UintegerValue (484));

 uint32_t nnode;

 for ( uint32_t x=1 ; x <= porcentagem ; x++) {
        nnode = (nWifi - x);
        printf("%d\n", nnode);
        ApplicationContainer clientApps = echoClient.Install (wifiStaNodes.Get (nnode));
        clientApps.Start (Seconds (2.0));
        clientApps.Stop (Seconds (time));
 }
```

```
}

void tcp_udp (NodeContainer &wifiApNode, float time, Ipv4InterfaceContainer &csmaInterfaces, uint32_t porcentagem, uint32_t nWifi, NodeContainer &wifiStaNodes)
{

 ApplicationContainer serverApp;
 ApplicationContainer sinkApp;

 std::ostringstream ossOnTime;
 ossOnTime << "ns3::ConstantRandomVariable[Constant=" << 0.001 << "]";
 std::ostringstream ossOffTime;
 ossOffTime << "ns3::ConstantRandomVariable[Constant=" << 0.001 << "]";

 /* TCP/UDP */
 for (uint32_t i = 0; i < porcentagem/2; i++){
 /* Instala TCP no access point */
  PacketSinkHelper sinkHelper ("ns3::TcpSocketFactory", InetSocketAddress (csmaInterfaces.GetAddress(0), i+10000));
  sinkApp = sinkHelper.Install (wifiStaNodes.Get(i));
  sinkApp.Add (sinkHelper.Install (wifiApNode.Get(0)));
  sinkApp.Start (Seconds (0.0));
  OnOffHelper server ("ns3::TcpSocketFactory", (InetSocketAddress (csmaInterfaces.GetAddress(0), i+10000)));
  server.SetAttribute ("PacketSize", UintegerValue (1484));
  server.SetAttribute ("OnTime", StringValue (ossOnTime.str()));
  server.SetAttribute ("OffTime", StringValue (ossOffTime.str()));
  server.SetAttribute ("DataRate", StringValue ("512kbps"));
  serverApp = server.Install (wifiStaNodes.Get(i));
  serverApp.Start (Seconds (1));
 }
  serverApp.Stop(Seconds(60+1));

 UdpEchoServerHelper echoServer (9);

 ApplicationContainer serverApps = echoServer.Install (wifiApNode.Get (0));
 serverApps.Start (Seconds (1.0));
 serverApps.Stop (Seconds (time)); //pega o tempo da variavel TIME

UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (0), 9);
echoClient.SetAttribute ("Interval", TimeValue (Seconds (0.001)));
echoClient.SetAttribute ("DataRate", StringValue ("512kbps"));
echoClient.SetAttribute ("PacketSize", UintegerValue (484));

 uint32_t nnode;

 for ( uint32_t x=1 ; x <= porcentagem/2 ; x++) {
        nnode = (nWifi - x);
        printf("%d\n", nnode);
        ApplicationContainer clientApps = echoClient.Install (wifiStaNodes.Get (nnode));
        clientApps.Start (Seconds (2.0));
        clientApps.Stop (Seconds (time));
 }
}

double calcDistance (uint32_t x1, uint32_t y1, uint32_t x2, uint32_t y2){
```

```
  return sqrt( (x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
}



//#################### MAIN ###################
int main (int argc, char *argv[]){
  bool verbose = false;
  uint32_t nCsma = 1;
  float time = 60.0;
  uint32_t nWifi = 10;//###### ALTERAR AQUI  ###########
  bool tracing = false;
  int tipo_trafego=0;//0 tcp; 1 50/50; 2 udp ###### ALTERAR AQUI  ###########

size_t nodenear = -1;
size_t nodefar = -1;
double dnodenear;
double dnodefar;

  CommandLine cmd;
  cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
  cmd.AddValue ("time", "Number of \"extra\" CSMA nodes/devices", time);
  cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);
  cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);
  cmd.AddValue ("tracing", "Enable pcap tracing", tracing);
  cmd.Parse (argc,argv);

  uint32_t porcentagem = (nWifi * 0.2); // CALCULO DA PORCENTAGEM - NESSE CASO ESTOU UTILIZANDO 20% -
###### ALTERAR AQUI para 10 20 30 e 40 ###########
  printf("Minha porcentagem: %d\n", porcentagem);//Printf apenas para ver se está com parametro correto na porcentagem

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

  NodeContainer p2pNodes;
  p2pNodes.Create (2);

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

  NetDeviceContainer p2pDevices;
  p2pDevices = pointToPoint.Install (p2pNodes);

  NodeContainer wifiApNode;
  wifiApNode = p2pNodes.Get (1);

 MobilityHelper mobilityh;
  //Lista a posição
  Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
  //Nó ap estático
  uint32_t ApX = 50;
```

```
  uint32_t ApY = 50;
  positionAlloc->Add (Vector (ApX, ApY, 0.0));
  mobilityh.SetPositionAllocator (positionAlloc);
  mobilityh.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobilityh.Install (wifiApNode);
//  csmaNodes.Create (nCsma);

  CsmaHelper csma;
//  csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
//  csma.SetChannelAttribute ("delay", TimeValue (NanoSeconds (6560)));

  NodeContainer wifiStaNodes;
  wifiStaNodes.Create (nWifi);

  YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
  YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
  phy.SetChannel (channel.Create ());

  WifiHelper wifi;
  wifi.SetStandard (WIFI_PHY_STANDARD_80211b);

  wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

  WifiMacHelper mac;
  Ssid ssid = Ssid ("ns-3-ssid");
  mac.SetType ("ns3::StaWifiMac", "Ssid", SsidValue (ssid), "ActiveProbing", BooleanValue (false));

  NetDeviceContainer staDevices;
  staDevices = wifi.Install (phy, mac, wifiStaNodes);

  mac.SetType ("ns3::ApWifiMac", "Ssid", SsidValue (ssid));

  NetDeviceContainer apDevices;
  apDevices = wifi.Install (phy, mac, wifiApNode);


  InternetStackHelper stack;
  stack.Install (p2pNodes.Get(0));
  stack.Install (wifiApNode);
  stack.Install (wifiStaNodes);

  Ipv4AddressHelper address;

  address.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer p2pInterfaces;
  p2pInterfaces = address.Assign (p2pDevices);

  address.SetBase ("10.1.2.0", "255.255.255.0");
  Ipv4InterfaceContainer csmaInterfaces;
  csmaInterfaces = address.Assign (apDevices);
  Ipv4InterfaceContainer devicesInterfaces;
  devicesInterfaces = address.Assign (staDevices);

  switch(tipo_trafego)
  {
```

```
    case 0:
        tcp (porcentagem, csmaInterfaces, wifiStaNodes, wifiApNode);// Chama a função TCP
    break;
    case 1:
        tcp_udp(wifiApNode, time, csmaInterfaces, porcentagem, nWifi, wifiStaNodes);//Chama a função TCP e UDP -50%
    break;
    case 2:
        udp(wifiApNode, time, csmaInterfaces, porcentagem, nWifi, wifiStaNodes);//Chama a função UDP
    break;
    default:
    return 0;
}

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();//protocolo IP

MobilityHelper mobility;

for (size_t i = 0; i < nWifi; i++)
{
  uint32_t NodeX = myRand(0, 100); //O 100 é um tamanho bom!
  uint32_t NodeY = myRand(0, 100);
  double result = calcDistance(ApX, ApY, NodeX, NodeY);


  if(i == 0)
   {
   nodenear = i;
   nodefar = i;
   dnodenear = result;
   dnodefar = result;
   }
  else
  {
  if(result > dnodefar)
   {
   nodefar = i;
   dnodefar = result;
   }
  if(result < dnodenear)
   {
   nodenear = i;
   dnodenear = result;
   }
  }
  Ptr<ListPositionAllocator> positionAlloc = CreateObject<ListPositionAllocator> ();
  //positionAlloc = CreateObject<ListPositionAllocator> ();
  positionAlloc->Add (Vector (NodeX, NodeY, 0.0));
  mobility.SetPositionAllocator (positionAlloc);
  mobility.Install (wifiStaNodes.Get(i));

}

 Simulator::Stop (Seconds (time));

  if (tracing == true)
```

```
      {
       pointToPoint.EnablePcapAll ("third");
       phy.EnablePcap ("third", apDevices.Get (0));
       csma.EnablePcap ("third", apDevices.Get (0), true);
      }

   //################# FLOW MONITOR #################
      Ptr<FlowMonitor> flowMonitor;
      FlowMonitorHelper flowHelper;
      flowMonitor = flowHelper.InstallAll();

      Simulator::Stop (Seconds(time));
      Simulator::Run ();


      funcao_flow(flowHelper, flowMonitor, devicesInterfaces, csmaInterfaces, nWifi, porcentagem, tipo_trafego, nodenear, nodefar);//Chama a função funcao_flow
      Simulator::Destroy ();
      return 0;
   }
```

SCRIPT PARA RODAR AS SIMULAÇÕES - SCRIPT.SH

```
#!/bin/bash

protocolo="TCP" #ALTERAR QUANDO FOR EXERCUTAR tipo_trafego
n_execucoes=30

mkdir $protocolo
for i in {1..$n_execucoes};
	do
		NS_GLOBAL_VALUE="RngRun=$i" ./waf --run scratch/mt
	done
mv *.csv $protocolo
```