FACULDADE DE COMPUTAÇÃO

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

MATHEUS PASSOS CASTRO

ANÁLISE DE COMPORTAMENTO DE REDES 4G/LTE SOB O IMPACTO DE HANDOVER

CASTANHAL

2017

MATHEUS PASSOS CASTRO

ANÁLISE DE COMPORTAMENTO DE REDES 4G/LTE SOB O IMPACTO DE HANDOVER

Monografia apresentada a Faculdade de Computação da Universidade Federal do Pará como exigência parcial para obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. José Jailton Henrique Ferreira Júnior.

CASTANHAL

2017

MATHEUS PASSOS CASTRO

ANÁLISE DE COMPORTAMENTO DE REDES 4G/LTE SOB O IMPACTO DE HANDOVER.

Monografia apresentada a Faculdade de Computação da Universidade Federal do Pará como exigência parcial para obtenção do título de Bacharel em Engenharia de Computação.

Data: 17 de Dezembro de 2023

ABSTRACT

Since its conception, 4G technology has been the subject of research in the scientific community and has focused efforts of researchers aiming to improve services provided to users. The featured work aims to conduct a study on the performance of 4G networks, focusing on Long Term Evolution (LTE) technology, in order to evaluate the service performance in different scenarios regarding the number of users and their speed during occurrences of handover in a network. The presented research was developed with the assistance of the Network Simulator, a specific tool for simulating different LTE scenarios and thereby facilitating the study of the technology. The collected results will be presented graphically to facilitate the understanding and performance evaluation of the technology.

RESUMO

Desde a sua concepção, a tecnologia 4G tem sido objeto de pesquisas na comunidade científica e tem concentrado esforços de pesquisadores com o intuito de melhorar os serviços prestados aos usuários. O trabalho em destaque tem por objetivo realizar um estudo sobre o desempenho de redes 4G com foco na tecnologia *Long Term Evolution* (LTE), de forma a avaliar o desempenho do serviço em diferentes cenários em relação a quantidade de usuários e a velocidade dos mesmos nas ocorrências de *handover* em uma rede. A pesquisa apresentada, foi desenvolvida com o auxílio do simulador *Network Simulator*, ferramenta específica para fazer simulações de cenários LTEs distintos e, dessa forma, facilitar o estudo sobre a tecnologia. Os resultados coletados serão apresentados graficamente com o objetivo de facilitar o entendimento e a avaliação de desempenho da tecnologia.

Palavras-Chave: Avaliação de Desempenho; Redes 4G; Tecnologia LTE; *Handover*; *Network Simulator.*

1 - INTRODUÇÃO

A evolução contínua das redes de comunicação móvel tem proporcionado avanços significativos na conectividade e na prestação de serviços aos usuários. Dentro desse contexto, o handover - a transição contínua de um dispositivo entre diferentes células ou pontos de acesso em uma rede sem fio - desempenha um papel fundamental na manutenção da conectividade e na garantia de uma experiência de usuário ininterrupta. Nas redes 4G, especialmente na tecnologia Long Term Evolution (LTE), o handover representa um elemento crítico para a eficiência do sistema, impactando diretamente a qualidade dos serviços oferecidos aos usuários finais.

Esta transição entre células ou pontos de acesso é essencial para evitar interrupções nas chamadas, transferências de dados ou streaming de mídia durante a movimentação do usuário. No entanto, a eficácia e a fluidez desse processo são influenciadas por uma série de variáveis, incluindo a quantidade de usuários na rede, a velocidade do movimento do usuário e a carga de tráfego em determinadas áreas geográficas.

Neste contexto, este estudo busca explorar e analisar os diferentes aspectos do handover, com foco na tecnologia LTE das redes 4G, com o objetivo de compreender os desafios, tendências e estratégias de otimização para aprimorar a eficiência desse processo. Ao examinar a dinâmica do handover em ambientes com variações de usuários e velocidades, pretende-se identificar padrões e possíveis melhorias que possam contribuir para a melhoria contínua da conectividade e qualidade dos serviços oferecidos aos usuários finais.

O objetivo deste trabalho é realizar uma análise do desempenho de uma rede LTE quando exposta à diversos cenários relacionados a quantidade de usuários conectados e a velocidade do deslocamento de tais usuários. Outros objetivos específicos também serão apresentados, tais como:

- Realizar um estudo sobre o comportamento da rede em relação as ocorrências de *handovers.*
- Apontar graficamente os resultados obtidos nas simulações.

2 - O que é *Handover*

O fenômeno de *handover*(transferência) é definido, segundo o artigo "*Handover Management in GSM cellular system*", como procedimento de transferência de uma chamada contínua de uma célula pra outra devido o distanciamento do usuário da cobertura, logo, numa rede LTE o *handover* ocorre quando há o distanciamento do dispositivo móvel do usuário da torre anteriormente conectada e uma aproximação de outra torre. Como o afastamento da torre a qual o usuário estava conectado gera uma deficiência na conexão, aumentando a quantidade de pacotes perdidos, então a troca de torre conectada automática é uma possível solução.

Para que tal fenômeno ocorra, é necessário que haja pelo menos duas torres e um dispositivo conectado a uma delas, no qual o dispositivo precisa se movimentar saindo da área de cobertura da torre inicialmente conectada e se aproximando da torre que irá iniciar uma nova conexão. A imagem a seguir demonstra o que foi descrito.

Baseando – se na locomoção do usuário, podemos obter dois tipos de *handover*, o vertical e o horizontal. O handover vertical ocorre quando um dispositivo muda de uma estação base para outra dentro da mesma rede, mas em diferentes frequências ou tecnologias. Por exemplo, em uma rede móvel, quando um dispositivo se move de uma célula para outra dentro da mesma operadora, mas utilizando frequências diferentes (por exemplo, de uma banda de frequência mais alta para uma mais baixa ou vice-versa), isso é um exemplo de handover vertical. Esse tipo de handover é fundamental para garantir uma transição suave e eficiente, mantendo a continuidade dos serviços sem interrupções perceptíveis para o usuário. Por outro lado o handover horizontal ocorre quando um dispositivo se move de uma estação base para outra, mas dentro da mesma frequência ou tecnologia de rede. Por exemplo, quando um dispositivo móvel se move de uma célula para outra, ambas operadas pela mesma frequência e tecnologia, como LTE, sem a mudança entre frequências, isso é considerado um handover horizontal. Esse tipo de handover é crucial para manter a conectividade durante o deslocamento do usuário e garantir uma transição contínua entre diferentes células ou pontos de acesso, mantendo a qualidade do serviço

oferecido aos usuários finais. Ambos os tipos de handover estão representados na figura 3.

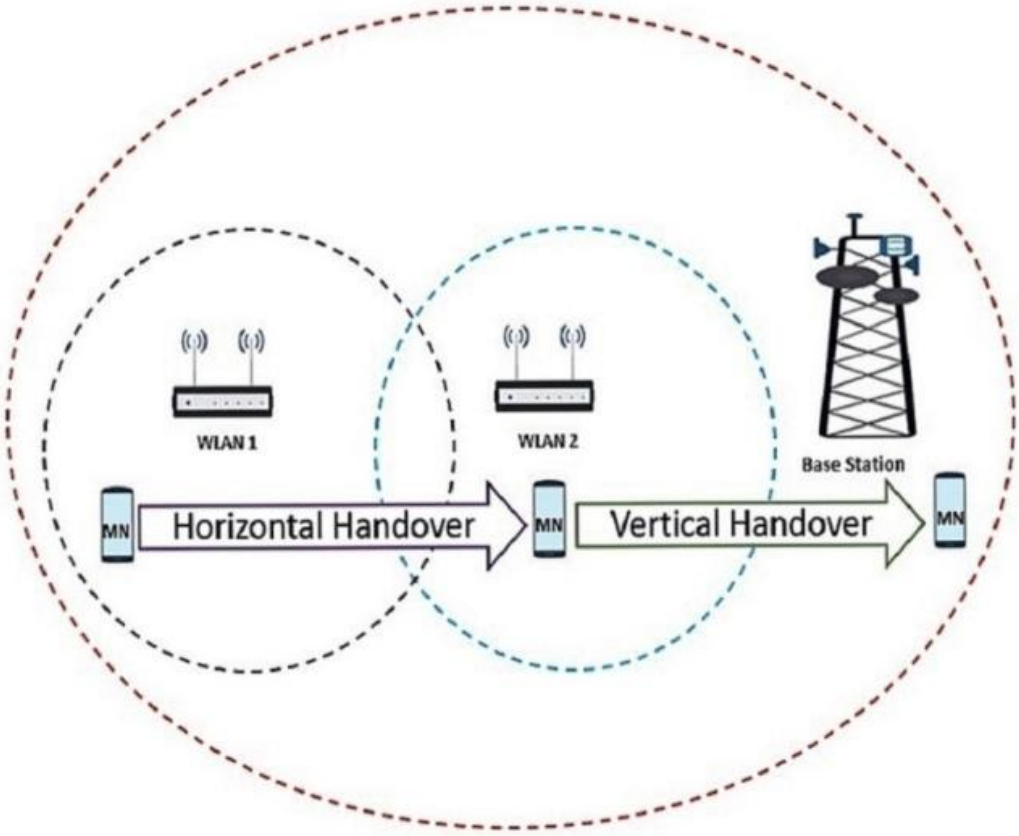


Figura 1 – Cenário Típico para o Handover Horizontal e Vertical

Fonte: Ayoub Bahnasse, https://www.researchgate.net/figure/Horizontal-and-Vertical-Handover-From-Study-and-Evaluation-of-Vertical-and-Horizontal_fig2_320197239

2.1 - Impactos Causados Pelo *Handover*

Dá-se o *handouver* como um “procedimento custoso porque envolve diversas tarefas que podem causar interrupções no fornecimento de serviços e degradação no desempenho das aplicações” (AREZIO, 2009, p.46). Tal situação se agrava a partir do contexto onde haja um aumento no número de transições. O maior obstáculo a ser ultrapassado é fazer com que tais transições sejam imperceptíveis para o usuário, para os protocolos das camadas superiores e

A ocorrência de handouver se dá a partir do momento em que o usuário está em movimento e transita saindo da área de alcance de uma torre de conexão pra outra. Aqui temos os principais erros causados pelo handouver numa rede LTE a qual é onde são maiores os danos:

- **Perda de pacotes**: Apesar de ser algo comum, no momento do *hanouver* na rede LTE é acentuada a perda de pacotes.
- **Vazão**: A ocorrência do processo de transação de células, a taxa de transmissão diminui drasticamente.
- **Quebra na conexão**: Dentre os impactos do *handouver*, o mais preocupante e perceptível, pois o usuário comum pode detectá-lo facilmente.

No entanto, é bom frisar que o handover não é a problemática, mas sim uma solução pra um problema, o qual consiste na manutenção no fluxo do serviço utilizado pelo usuário, na qual o mesmo enfrente dificuldades de conexão por conta da distância da célula conectada. Trabalha-se hoje para a amenização do impacto gerado por esta alternativa chamada handover.

3 - RESULTADOS

Nesta parte do trabalho serão apresentados os resultados das simulações no software NS-3(Network Simulator - 3) sobre o handover e seus impactos com as variações de quantidade de usuários e suas respectivas velocidades. Tal análise é feita por pontos(Flowid) na simulação que indicam os resultados em um período de tempo de conexão.

3.1 - O Simulador NS-3

Para atingir os objetivos da pesquisa, foi feito uso de simulações de eventos discretos. A simulação é uma técnica de avaliação e desempenho de um sistema baseada no desenvolvimento de modelos, ou seja, abstrações da realidade, que capturam aspectos essenciais do objeto em estudo.

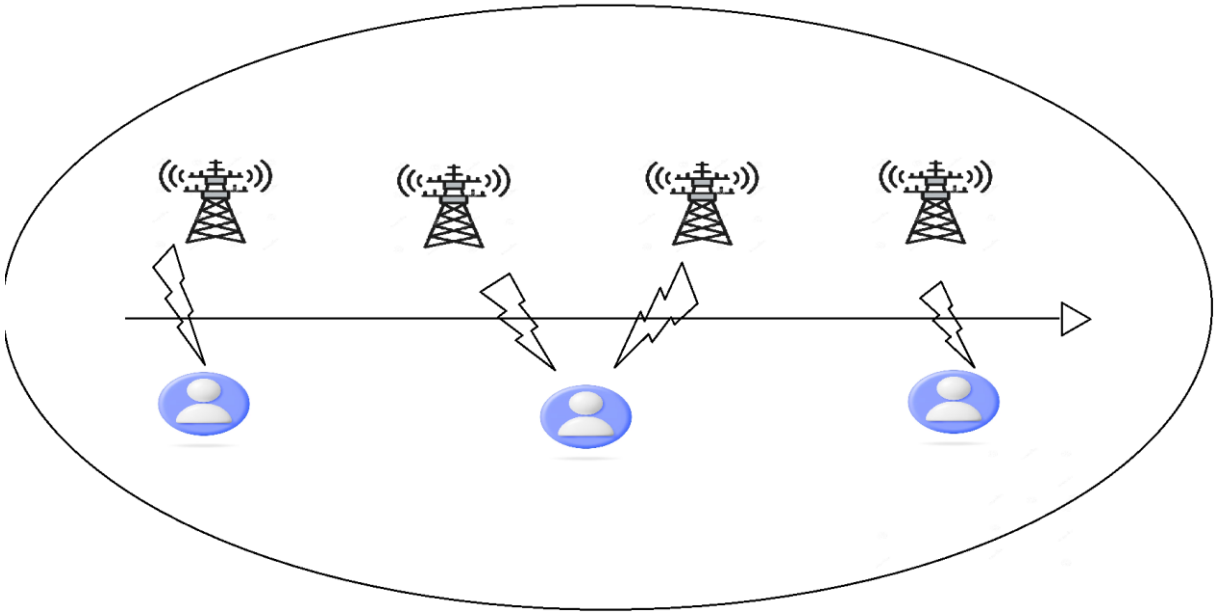
Como já mencionado, o simulador utilizado na pesquisa para realizar as simulações foi o *Network Simulator 3* (NS-3) na versão 3.30, que se trata de uma ferramenta de simulações de eventos discretos para sistemas de Internet (NS-3 TUTORIAL, 2012) e é amplamente usufruída pela comunidade científica internacional de pesquisa em redes de computadores. A principal diferença e vantagem do simulador é a oferta de uma massa de modelos de tecnologias diferentes disponíveis para simulações, em especial os modelos de sistemas *Wireless* e uma variedade de protocolos.

Os cenários esquematizados no simulador foram concebidos utilizando a biblioteca LTE da ferramenta, que dentre várias funções, disponibiliza a ferramenta *LteHelper* que permite estruturar todos os nós do cenário (*UEs* e *eNBs*) com as tecnologias particulares de cada um, pertencentes a rede LTE. Vale ressaltar também o uso do módulo *MobilityModel* para conceder e gerenciar a mobilidade dos nós. As métricas utilizadas para fazer as análises dos resultados das simulações foram coletadas com classe *FlowMonitor* pertencente ao próprio simulador.

Como o simulador é um software livre sob a licença *GNU GPLv2*, uma grande massa de desenvolvedores e pesquisadores realizam contribuições com o software, permitindo a criação de mais módulos, suprindo algumas limitações e melhorando a ferramenta.

A Figura 2 exibe o cenário esquematizado no simulador para a efetuação das simulações e suas respectivas análises.

Figura 2 – Cenário da Primeira Simulação



Fonte: O Autor

S

3.2 - Primeiro Cenário

Na primeira simulação foi descrito o cenário com quatro *eNBs*, utilizado o algoritmo *A2A4RsrqHandoverAlgorithm* somente variando a velocidade e o número de usuários para realizar o *handover*. Tal simulação relata sobre os impactos do handover em um rede LTE, principalmente a sua vazão, que seria a sua taxa de dados, o atraso(delay) e o jitter. O Quadro 1 informa os parâmetros usados para a primeira simulação

Quadro 1 – Parâmetros para a Primeira Simulação

Quantidade de Usuários	Dois nós conectados
Tempo de Simulação	250 segundos
Tecnologia	Rede LTE
Tráfego Internet	100 Gigabit por segundo
Tráfego da Aplicação	1 Gigabit por segundo
Tamanho do pacote da aplicação	100000
Velocidade do nó	10 metros por segundo

Os resultados do primeiro cenário são retirados em pontos de análise do fluxo com a somatória do atraso, do jitter e de vazão, como mostrado nos gráficos um,dois e três

Gráfico 1, Somatória de vazão por ponto de fluxo analisado

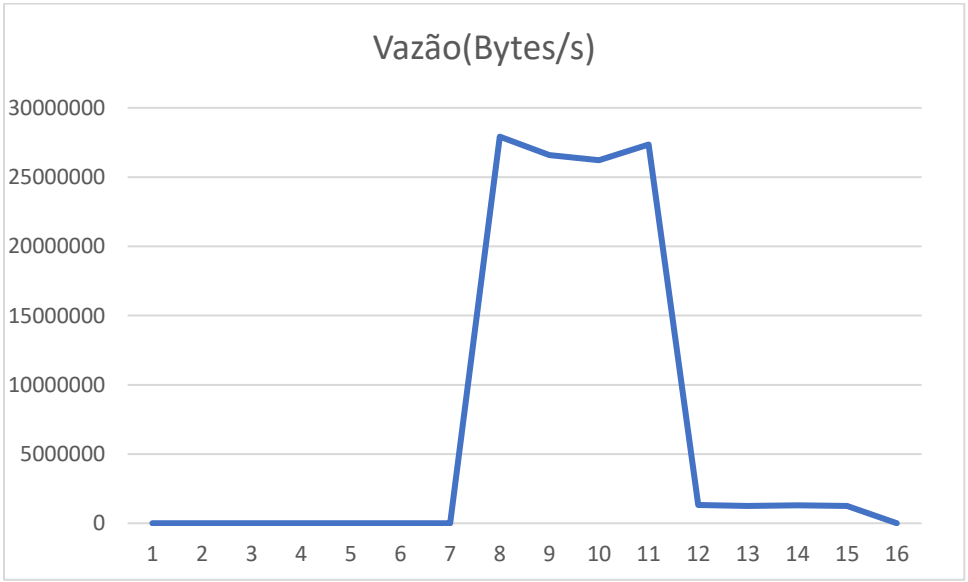


Gráfico 2, Somatória de atraso por ponto de fluxo analisado

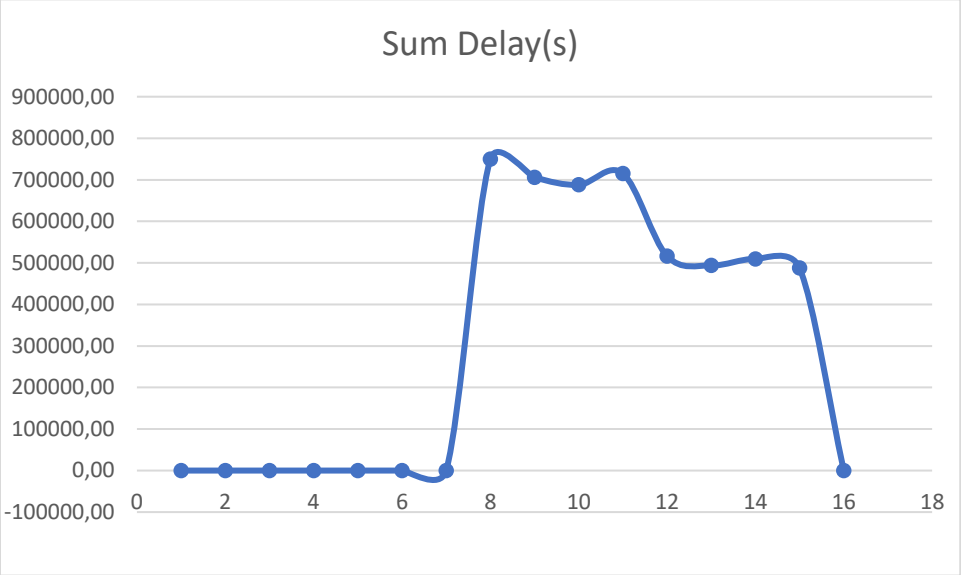
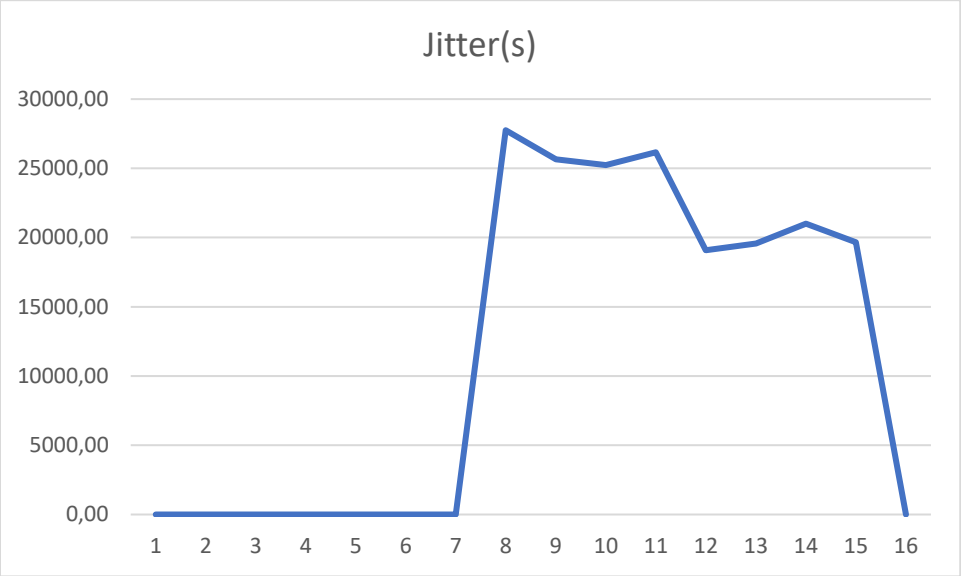


Gráfico 3, Somatória de Jitter por ponto de fluxo analisado



Nesse primeiro cenário verificamos que a rede se mantém estável até o período que o handover ocorre, fazendo com que haja um pico no jitter, atraso e vazão, porém tal pico se mantém por um tempo maior no jitter e no ataso, enquanto na vazão, logo normaliza.

3.3 - Segundo cenário

Quadro 2 – Parâmetros para a Segunda Simulação

Quantidade de Usuários	Dois nós conectados
Tempo de Simulação	125 segundos
Tecnologia	Rede LTE
Tráfego Internet	100 Gigabit por segundo
Tráfego da Aplicação	1 Gigabit por segundo
Tamanho do pacote da aplicação	100000
Velocidade do nó	20 metros por segundo

No segundo cenário iremos verificar o impacto do handover aumentando a velocidade em relação a primeira análise.

Gráfico 4 – Somatório de vazão por ponto de fluxo

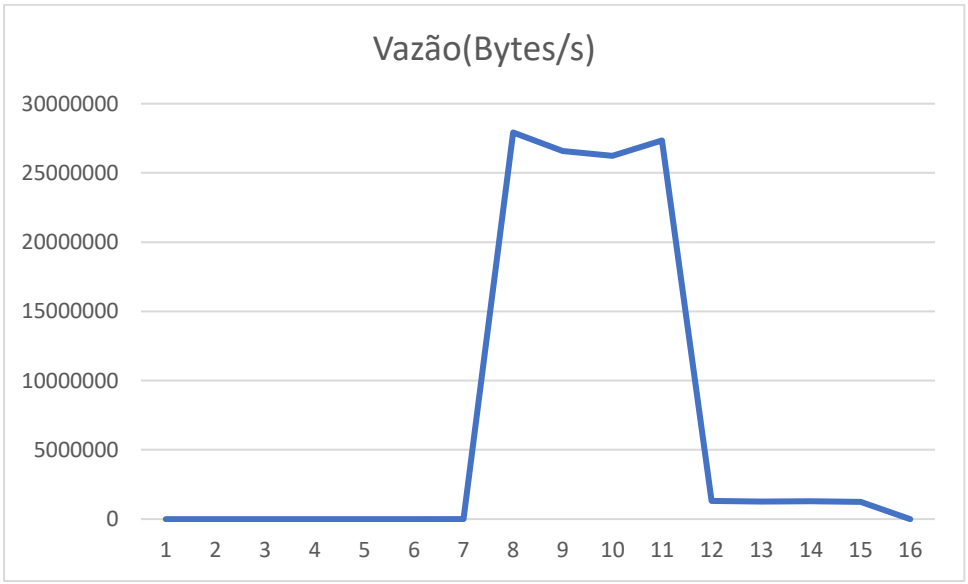


Gráfico 5 – Somatório de atraso por ponto de fluxo

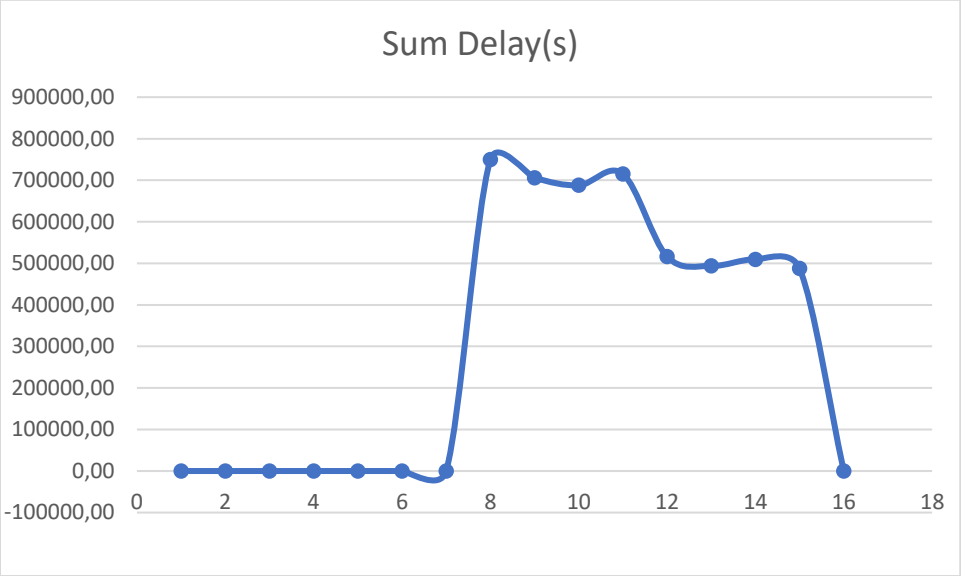
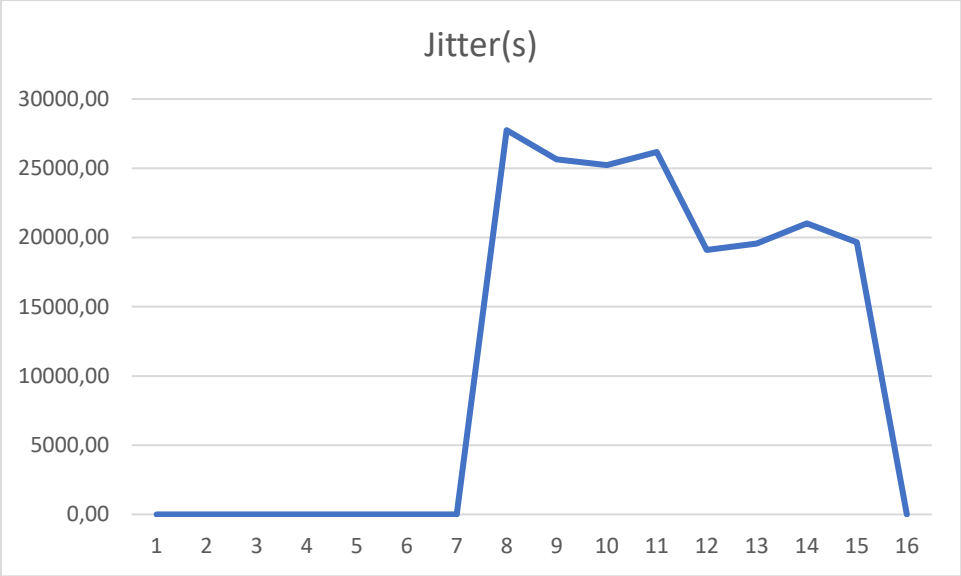


Gráfico 6 – Somatório de Jitter por ponto de fluxo



Nesta simulação tivemos um resultado bem parecido com o anterior, porém em um espaço de tempo menor, ou seja, os pontos de análise de fluxo são menores e o handover não demonstra um acréscimo em seus impactos em relação a um acréscimo de 10m/s de velocidade.

3.4 - Terceiro Cenário

Quadro 3 – Parâmetros para a terceiro Simulação

Quantidade de Usuários	Quatro nós conectados
Tempo de Simulação	250 segundos
Tecnologia	Rede LTE
Tráfego Internet	100 Gigabit por segundo
Tráfego da Aplicação	1 Gigabit por segundo
Tamanho do pacote da aplicação	100000
Velocidade do nó	10 metros por segundo

No cenário atual será acrescentado mais dois usuários totalizando quatro pra análise no ambiente montado na simulação pra vermos o impacto gerado.

Gráfico 7 – Somatório de vazão por ponto de fluxo

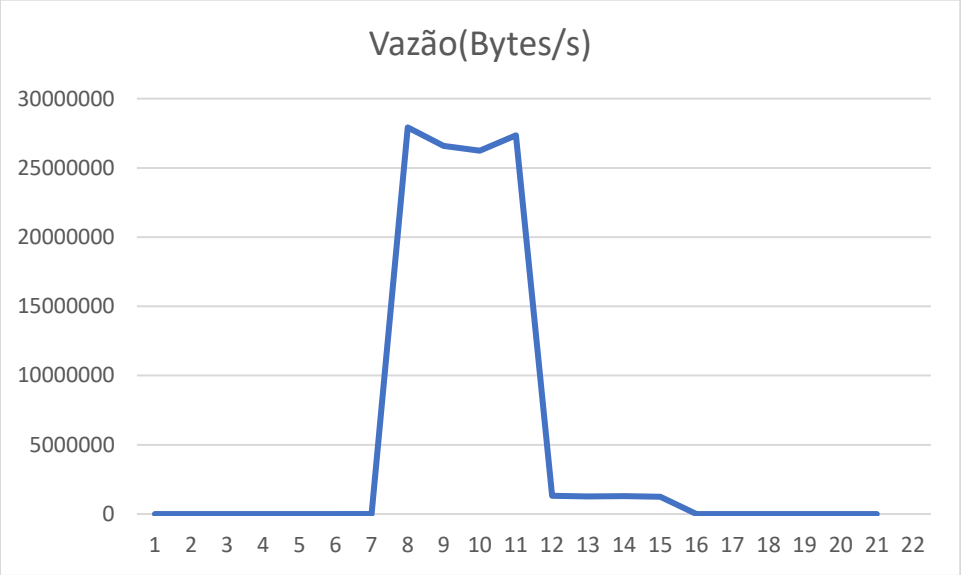


Gráfico 8 – Somatório de atraso por ponto de fluxo

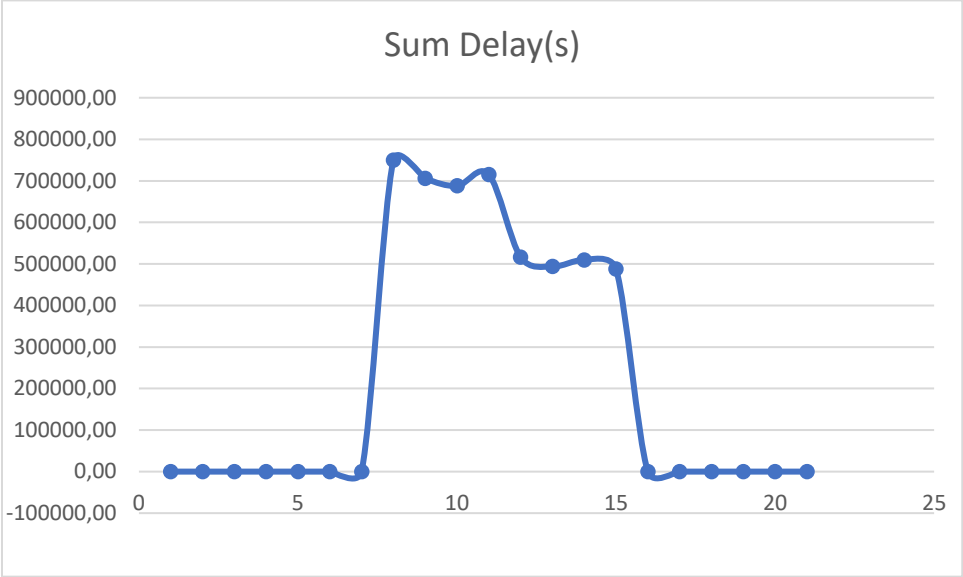
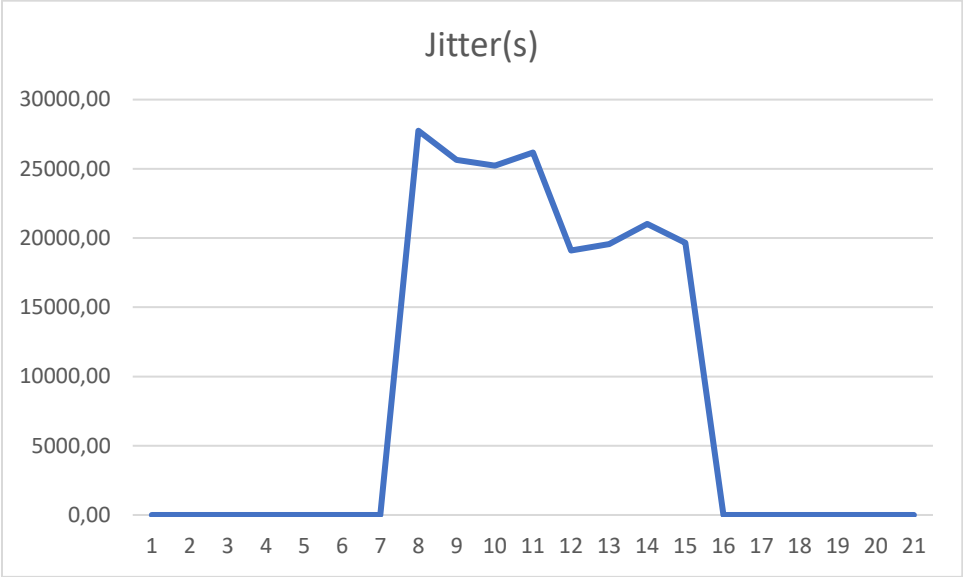


Gráfico 9 – Somatório de jitter por ponto de fluxo



Nesse caso, vemos que houve um aumento nos momentos de pico onde o atraso e o jitter tiveram um pico, pois permaneceram mais tempo em alta, no entanto a vazão teve um pico maior, porém, o período no pico não foi maior do que as anteriores.

3.5 - Quarto cenário

Quadro 4 – Parâmetros para a Quarta Simulação

Quantidade de Usuários	Quatro nós conectados
Tempo de Simulação	125 segundos
Tecnologia	Rede LTE
Tráfego Internet	100 Gigabit por segundo
Tráfego da Aplicação	1 Gigabit por segundo
Tamanho do pacote da aplicação	100000
Velocidade do nó	20 metros por segundo

Agora analisaremos aumentando a velocidade nos mesmos quatro usuários pra que identifiquemos os impactos do handover na rede.

Gráfico 10 – Somatório de vazão por ponto de fluxo

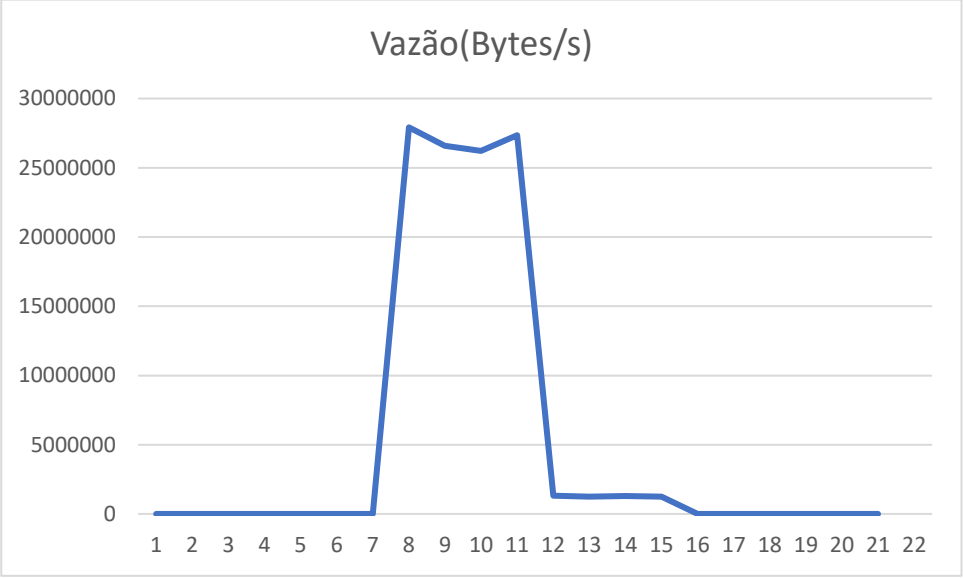


Gráfico 11 – Somatório de atraso por ponto de fluxo

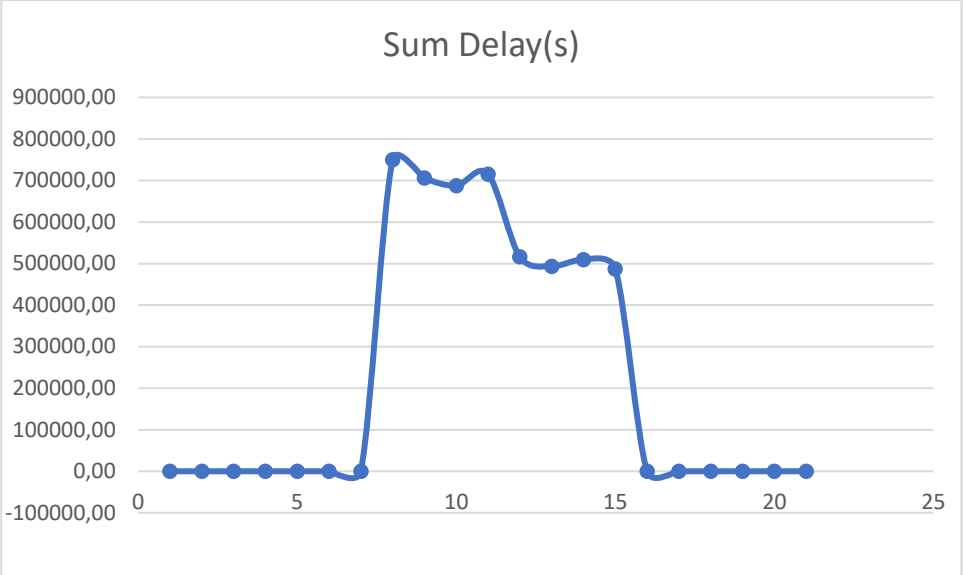
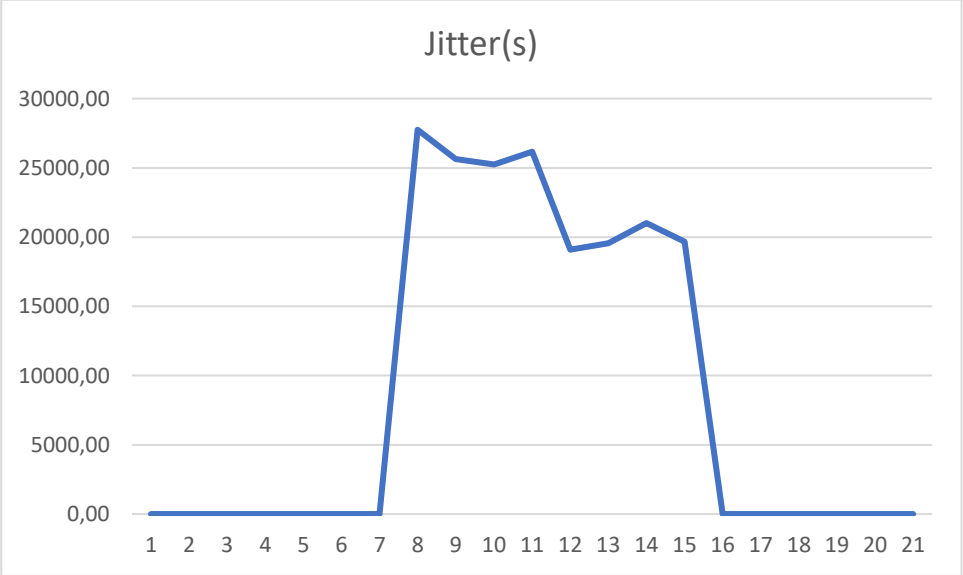


Gráfico 12 – Somatório de jitter por ponto de fluxo



Novamente o impacto gerado pelo aumento de velocidade no handover na rede é somente no tempo a qual é executado, no entanto, isso indica que o impacto maior é o aumento de usuários.

4 - CONCLUSÃO

Em suma, a análise detalhada do handover dentro das redes 4G, focando na tecnologia LTE, revelou insights significativos sobre a dinâmica e desempenho desses sistemas. Observamos que a quantidade de usuários e suas velocidades têm um impacto direto na eficiência do handover, influenciando a continuidade dos serviços oferecidos. Os resultados destacaram a importância de estratégias de otimização para gerenciar o handover de maneira mais eficaz, especialmente em ambientes com alto tráfego e demanda variável. Além disso, este estudo ressalta a necessidade contínua de aprimoramentos técnicos e algoritmos mais sofisticados para lidar com os desafios do handover, visando aprimorar a experiência do usuário e a estabilidade das redes. Assim, o handover permanece um aspecto crucial nas comunicações móveis modernas, e o contínuo aprofundamento deste campo é fundamental para o desenvolvimento e evolução das futuras gerações de redes de comunicação.

5 - REFERÊNCIA

AGRAWAL, Sonal D. Mobility Management Vertical and Horizontal Handover Decisions in Heterogeneous Wireless Networks Using Ominet. 2016. Disponível em: <https://www.ijser.org/researchpaper> Acesso em 12 de março de 2017.

ALMEIDA, C.; MELO, C.; SANTOS, C.; COSTA, G.; RODRIGUES, P. Redes sem fios: Gerações de Telemóveis. 2013.

BARUMERLI, Roberto. Studio e Simulazione delle Procedure di Handover nello Standard per Reti Cellulari LTE. 2013**.** Disponível em: <https://tesi.cab.unipd.it/43854/> Acesso em 09 de fevereiro de 2017.

D'ÁVILA, César Kyn. **LTE**: Long Term Evolution – Arquitetura Básica e Acesso Múltiplo. 2009. Disponível em

<http://www.cedet.com.br/index.php?/Tutoriais/Telecom/lte-long-term-evolutionarquitetura-basica-e-acesso-multiplo.html> Acesso em agosto de 2017.

DHIMAN, Abhishek. Vertical And Horizontal Handover In Heterogeneous Wireless Networks. 2012. Dissertação (Mestrado em Engenharia Eletrônica e Comunicação), Departamento de Eletrônica e Engenharia de Comunicação, Thapar University, Patiala, 2012.

ERMOLAYEV, Victor. Handover Parameter Optimisation in LTE. 2016. Disponível em: <https://www.mera.com/media/attachments/2016/05/29/handover-parameteroptimisation-in-lte.pdf> Acesso em 15 de junho de 2017.

FOLHA DE SÃO PAULO. Número de smartphones em uso no Brasil chega a 168 milhões, diz estudo. 2016. Disponível em <http://www1.folha.uol.com.br/mercado/2016/04/1761310-numero-de-smartphonesem-uso-no-brasil-chega-a-168-milhoes-diz-estudo.shtml> Acesso em 27 de setembro de 2017.

HEO, H.; LEE, W.; KIM, H.; LEE, B.; JEONG, B.; KIM, N. Analysis of 3GPP LTE Handover using NS-3 SIMULATOR. 2016.

HERMAN, B.; PETROV, D.; PUTTONEN, J.; KURJENNIEMI, J. A3-Based Measurements and Handover Model for NS-3 LTE. 2013.

KATTI, N.; SHIVAPUR, S.; VIJAYALAKSHMI, M. Optimization of QoS in 4G Networks Using Handover Management. 2015. Disponível em: <http://www.ijetcse.com> Acesso em fevereiro de 2017.

KHAN, Jahangir. Handover Management in GSM Cellular System. 2010. Disponível em: <https://www.ijcaonline.org/archives/volume8/number12/1257-1763> Acesso em 23 de novembro de 2016.

KUROSE, J.; ROSS, K. Redes de Computadores e a Internet. 3. ed. São Paulo: Pearson, 2006.

MÄKELÄ, Juha-Pekka. Effects of Handoff Algorithms on the Performance of Multmedia Wireless Networks. 2008. Dissertação (Mestrado em Engenharia Elétrica e Informação), Faculdade de Tecnologia, Universidade de Oulu, Oulu, 2008.

MAKINO, Iudy. Desempenho de Sistemas Smart Grid Sob Plataforma de Transmissão lte. 2013. Disponível em: <https://www.eletrica.ufpr.br/pedroso/Monografias/TCC-IudyMakino-2013.pdf> Acesso em 05 de outubro de 2016.

MAZZONI, Victor de Souza. Análise Histórica e Funcional das Redes 4G LTE. 2014. Disponível em: <http://monografias.nrc.ice.ufjf.br/tcc-web/downloadPdf?id=175> Acesso em janeiro de 2017.

MENDES, Cristian Tiago Erazo. Avaliação de Desempenho de Algoritmos de Escalonamento de Dados em Redes LTE. 2014. Disponível em: <https://repositorio.ufla.br/bitstream/1/10713/1 > Acesso em 30 de outubro de 2016.

NAKANO, Camila L.; VIANA, Thayane R. Desempenho de Redes *AD HOC* para Escalonamento de Tráfego de Redes Celulares. 2014. Disponível em <http://www.ene.unb.br/mmcarvalho/pubs/tcc/final.camila.thayane.pdf> Acesso em 03 de dezembro de 2016.

NAGAMUTA, Vera. Seamless Handover em Redes Móveis sem Fio. 2003. Disponível em <https://www.ime.usp.br/~cpgmac/Disciplinas_Passadas/2003i/mac5701/Relatorios/Vera_rel.OLD.pdf> Acesso em junho de março de 2017.

ns-3 Model Library: Release ns-3.14. 2012. Disponível em: <https://www.nsnam.org/docs/models/html/lte.html> Acesso em 10 de agosto de 2016.

ns-3 Tutorial: Release ns-3.14. 2012. Disponível em: <https://www.nsnam.org/docs/tutorial-pt-br/html/index.html> Acesso em 05 julho de 2016.

RICARDO, Claudia Arezio. Otimização do Processo de Decisão no Handover Vertical em Redes Baseadas no Subsistema Multimídia IP (IMS). 2009. Dissertação (Mestrado em Informática Aplicada), Pontifícia Universidade Católica, Curitiba, 2009.

SALES, Diego Fernandes. Análise de Handover inter e intra – célula em um Sistema de Telefonia Celular através do Método de Medição Simplificado. 2009. Dissertação (Mestrado em Engenharia Elétrica e Computação), Centro de Tecnologia, Universidade Federal do Rio Grande do Norte, Natal, 2009.

SILVA, Ketyllen da Costa. Análise de Handover a Partir do uso de Femtocells em Redes LTE: Abordagem baseada em Simulação Discreta. 2014. Dissertação (Mestrado em Engenharia Elétrica), Instituto de Tecnologia, Universidade Federal do Pará, Belém, 2014.

TANENBAUM, A. Redes de Computadores. 5. ed. São Paulo: Pearson, 2011.

3GPP, Overview 3GPP Release 8. Disponível em <http://www.3gpp.org/specifications/releases/72-release-8> Acesso em 20 de dezembro de 2016.

6 - APÊNDÍCE A – SCRIPT BASE DE SIMULAÇÃO

```
/* -*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*- */
/*
 * Copyright (c) 2013 Centre Tecnologic de Telecomunicacions de Catalunya (CTTC)
 *
 * 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
 *
 * Author: Manuel Requena <manuel.requena@cttc.es>
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/lte-module.h"
#include "ns3/applications-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/config-store-module.h"
#include "ns3/netanim-module.h"

//added lib
#include "ns3/gnuplot.h"
#include "ns3/flow-monitor-module.h"
#include <ns3/flow-monitor-helper.h>


using namespace ns3;

void ThroughputMonitor (FlowMonitorHelper *fmhelper, Ptr<FlowMonitor> flowMon,Gnuplot2dDataset DataSet);
void JitterMonitor(FlowMonitorHelper *fmHelper, Ptr<FlowMonitor> flowMon, Gnuplot2dDataset Dataset2);
void DelayMonitor(FlowMonitorHelper *fmHelper, Ptr<FlowMonitor> flowMon, Gnuplot2dDataset Dataset3);

NS_LOG_COMPONENT_DEFINE ("LenaX2HandoverMeasures");

void
NotifyConnectionEstablishedUe (std::string context,
                               uint64_t imsi,
                               uint16_t cellid,
                               uint16_t rnti)
{
  std::cout << context
            << " UE IMSI " << imsi
            << ": connected to CellId " << cellid
            << " with RNTI " << rnti
            << std::endl;
}

void
NotifyHandoverStartUe (std::string context,
                       uint64_t imsi,
                       uint16_t cellid,
                       uint16_t rnti,
                       uint16_t targetCellId)
```

```
{
  std::cout << context
            << " UE IMSI " << imsi
            << ": previously connected to CellId " << cellid
            << " with RNTI " << rnti
            << ", doing handover to CellId " << targetCellId
            << std::endl;
}

void
NotifyHandoverEndOkUe (std::string context,
                       uint64_t imsi,
                       uint16_t cellid,
                       uint16_t rnti)
{
  std::cout << context
            << " UE IMSI " << imsi
            << ": successful handover to CellId " << cellid
            << " with RNTI " << rnti
            << std::endl;
}

void
NotifyConnectionEstablishedEnb (std::string context,
                                uint64_t imsi,
                                uint16_t cellid,
                                uint16_t rnti)
{
  std::cout << context
            << " eNB CellId " << cellid
            << ": successful connection of UE with IMSI " << imsi
            << " RNTI " << rnti
            << std::endl;
}

void
NotifyHandoverStartEnb (std::string context,
                        uint64_t imsi,
                        uint16_t cellid,
                        uint16_t rnti,
                        uint16_t targetCellId)
{
  std::cout << context
            << " eNB CellId " << cellid
            << ": start handover of UE with IMSI " << imsi
            << " RNTI " << rnti
            << " to CellId " << targetCellId
            << std::endl;
}

void
NotifyHandoverEndOkEnb (std::string context,
                        uint64_t imsi,
                        uint16_t cellid,
                        uint16_t rnti)
{
  std::cout << context
            << " eNB CellId " << cellid
            << ": completed handover of UE with IMSI " << imsi
            << " RNTI " << rnti
            << std::endl;
}
void
CourseChange(std::string foo, Ptr<const MobilityModel> mobility)
{
```

```
Vector pos = mobility->GetPosition();
Vector vel = mobility->GetVelocity();
std::cout << "Tempo =" << Simulator::Now().GetSeconds() << ",Model =" << mobility << ", POS X =" << pos.x << " ,POS Y = "
<< pos.y << " ,VEL = "<< vel.x << std::endl;
}


/**
 * Sample simulation script for an automatic X2-based handover based on the RSRQ measures.
 * It instantiates two eNodeB, attaches one UE to the 'source' eNB.
 * The UE moves between both eNBs, it reports measures to the serving eNB and
 * the 'source' (serving) eNB triggers the handover of the UE towards
 * the 'target' eNB when it considers it is a better eNB.
 */
int
main (int argc, char *argv[])
{
  // LogLevel logLevel = (LogLevel)(LOG_PREFIX_ALL | LOG_LEVEL_ALL);

  // LogComponentEnable ("LteHelper", logLevel);
  // LogComponentEnable ("EpcHelper", logLevel);
  // LogComponentEnable ("EpcEnbApplication", logLevel);
  // LogComponentEnable ("EpcX2", logLevel);
  // LogComponentEnable ("EpcSgwPgwApplication", logLevel);

  // LogComponentEnable ("LteEnbRrc", logLevel);
  // LogComponentEnable ("LteEnbNetDevice", logLevel);
  // LogComponentEnable ("LteUeRrc", logLevel);
  // LogComponentEnable ("LteUeNetDevice", logLevel);
  // LogComponentEnable ("A2A4RsrqHandoverAlgorithm", logLevel);
  // LogComponentEnable ("A3RsrpHandoverAlgorithm", logLevel);

  uint16_t numberOfUes = 2;
  uint16_t numberOfEnbs = 4;
  //uint16_t numBearersPerUe = 1;
  double distance = 500.0; // m
  //double yForUe = 100.0;   // m
  double speed = 10;       // m/s
  double simTime = (double)(numberOfEnbs + 1) * distance / speed; // 1500 m / 20 m/s = 75 secs
  // double simTime = 200;
  double enbTxPowerDbm = 46.0;
  uint16_t seed = 1; // SEMENTE DE SIMULAÇÃO

  // change some default attributes so that they are reasonable for
  // this scenario, but do this before processing command line
  // arguments, so that the user is allowed to override these settings
  Config::SetDefault ("ns3::UdpClient::Interval", TimeValue (MilliSeconds (10)));
  Config::SetDefault ("ns3::UdpClient::MaxPackets", UintegerValue (1000000));
  Config::SetDefault ("ns3::LteHelper::UseIdealRrc", BooleanValue (true));

  // Command line arguments
  CommandLine cmd;
  cmd.AddValue("seed", "Seed of simulation", seed);
  cmd.AddValue ("simTime", "Total duration of the simulation (in seconds)", simTime);
  cmd.AddValue ("speed", "Speed of the UE (default = 20 m/s)", speed);
  cmd.AddValue ("enbTxPowerDbm", "TX power [dBm] used by HeNBs (defalut = 46.0)", enbTxPowerDbm);

  cmd.Parse (argc, argv);
  ns3::SeedManager::SetSeed(seed);


  Ptr<LteHelper> lteHelper = CreateObject<LteHelper> ();//cria um objeto do tipo Lte
  Ptr<PointToPointEpcHelper> epcHelper = CreateObject<PointToPointEpcHelper> ();//cria um objeto p simular um Ponto-a-
Ponto -> Topologia
  lteHelper->SetEpcHelper (epcHelper);//Informa ao lte que sera usado o EPC
```

```
  lteHelper->SetSchedulerType ("ns3::RrFfMacScheduler");//Informa ao lte o modelo de programador a ser usado.

  lteHelper->SetHandoverAlgorithmType ("ns3::A2A4RsrqHandoverAlgorithm");//informando que sera utilizado o handover para
tratar a transição de Ues entre Enbs
  lteHelper->SetHandoverAlgorithmAttribute ("ServingCellThreshold", UintegerValue (30));//Configuração do algoritmo de
entrega - PADRAO
  lteHelper->SetHandoverAlgorithmAttribute ("NeighbourCellOffset",//Configuração do algoritmo de entrega - PADRAO
                                    UintegerValue (1));

  //  lteHelper->SetHandoverAlgorithmType ("ns3::A3RsrpHandoverAlgorithm");
  //  lteHelper->SetHandoverAlgorithmAttribute ("Hysteresis",
  //                                    DoubleValue (3.0));
  //  lteHelper->SetHandoverAlgorithmAttribute ("TimeToTrigger",
  //                                    TimeValue (MilliSeconds (256)));

  Ptr<Node> pgw = epcHelper->GetPgwNode ();//criando um objeto PGW para tratar a entrada e saida de dados do usuario.

  // Create a single RemoteHost
  NodeContainer remoteHostContainer;//instanciando um objeto do tipo NodeContainer
  remoteHostContainer.Create (1);//cria-se somente 1 host
  Ptr<Node> remoteHost = remoteHostContainer.Get (0);//informa que o nosso host (posicao 0) vai ser um host remoto
  InternetStackHelper internet;//criamos um objeto que representa os protocolos da internet ip tcp udp
  internet.Install (remoteHostContainer);//instalamos a pilha de protocolos no host remoto

  // Create and configuring the Internet
  PointToPointHelper p2ph;
  p2ph.SetDeviceAttribute ("DataRate", DataRateValue (DataRate ("100Gb/s")));
  p2ph.SetDeviceAttribute ("Mtu", UintegerValue (1500));
  p2ph.SetChannelAttribute ("Delay", TimeValue (Seconds (0.010)));
  NetDeviceContainer internetDevices = p2ph.Install (pgw, remoteHost);
  Ipv4AddressHelper ipv4h;
  ipv4h.SetBase ("1.0.0.0", "255.0.0.0");
  Ipv4InterfaceContainer internetIpIfaces = ipv4h.Assign (internetDevices);
  //Ipv4Address remoteHostAddr = internetIpIfaces.GetAddress (1);


  // Routing of the Internet Host (towards the LTE network) - Criando rotas para o Host ate a Net
  Ipv4StaticRoutingHelper ipv4RoutingHelper;
  Ptr<Ipv4StaticRouting> remoteHostStaticRouting = ipv4RoutingHelper.GetStaticRouting (remoteHost->GetObject<Ipv4> ());
  // interface 0 is localhost, 1 is the p2p device
  remoteHostStaticRouting->AddNetworkRouteTo (Ipv4Address ("7.0.0.0"), Ipv4Mask ("255.0.0.0"), 1);//Nesse caso Ip e Mask
sao padrao

  /*
   * Network topology:
   *
   *      |     + -------------------------------------------------------->
   *      |     UE
   *      |
   *      |             d                 d                 d
   *    y |     |-------------------x-------------------x--------------------
   *      |     |               eNodeB              eNodeB
   *      |   d |
   *      |     |
   *      |     |                                        d = distance
   *            o (0, 0, 0)                                y = yForUe
   */

  NodeContainer ueNodes;//instanciando um objeto que represente os Ues
  NodeContainer enbNodes;//instanciando um objeto que represente as Enbs
  enbNodes.Create (numberOfEnbs);//criando verdadeiramente  as Enbs - 2
  ueNodes.Create (numberOfUes);//criando verdadeiramente o Ue - 1

  // Install Mobility Model in eNB
  Ptr<ListPositionAllocator> enbPositionAlloc = CreateObject<ListPositionAllocator> ();
```

```
 enbPositionAlloc->Add (Vector(100.0, 100.0, 0.0));
 enbPositionAlloc->Add (Vector(100.0, 500.0, 0.0));
 enbPositionAlloc->Add (Vector(500.0, 500.0, 0.0));
 enbPositionAlloc->Add (Vector(500.0, 100.0, 0.0));

 //for (uint16_t i = 0; i < numberOfEnbs; i++)
  //{
    //Vector enbPosition (distance * (i + 1), distance, 0);
    //enbPositionAlloc->Add (enbPosition);
  //}

 Ptr<ListPositionAllocator> uePositionAlloc = CreateObject<ListPositionAllocator> ();
 uint32_t var = 0;
 for(var =0; var < numberOfUes; var++){
 uePositionAlloc-> Add (Vector(0.0, (var+100.0), 0.0));
 if(var==1){
 uePositionAlloc-> Add (Vector(0.0, 500.0, 0.0));

 }
 }


 MobilityHelper enbMobility;//criando uma variavel que represente a mobilidade
 enbMobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");//configurando a mobilidade como posicao constante
 enbMobility.SetPositionAllocator (enbPositionAlloc);//definindo a mobilidade nas posicoes das enbs
 enbMobility.Install (enbNodes);//instalando a mobilidade nas enbs

 // Install Mobility Model in UE
 MobilityHelper ueMobility;//criando uma variavel que represente a mobilidade
 ueMobility.SetPositionAllocator(uePositionAlloc);
 ueMobility.SetMobilityModel ("ns3::RandomDirection2dMobilityModel", "Bounds", RectangleValue (Rectangle (0, 500, 0, 500)),
                              "Speed", StringValue ("ns3::ConstantRandomVariable[Constant=20.0]"),
                              "Pause", StringValue ("ns3::ExponentialRandomVariable[Mean=0.5]"));//configurando a mobilidade
 ueMobility.Install (ueNodes);// instalando a mobilidade nas Ues
 //ueNodes.Get (0)->GetObject<MobilityModel> ()->SetPosition (Vector (0, yForUe, 0));//definindo a posicao(X,Y) e o
deslocamento das Ues
 //ueNodes.Get (0)->GetObject<ConstantVelocityMobilityModel> ()->SetVelocity (Vector (speed, 0, 0));//definindo a velocidade
nesse esquema de codigo a Ue ira se deslocar paralelo ao eixo X

//teste
Config::Connect("/NodeList/*/$ns3::MobilityModel/CourseChange",
            MakeCallback(&CourseChange));


 // Install LTE Devices in eNB and UEs
 Config::SetDefault ("ns3::LteEnbPhy::TxPower", DoubleValue (enbTxPowerDbm));
 NetDeviceContainer enbLteDevs = lteHelper->InstallEnbDevice (enbNodes);//criando um objeto que representa o dispositivo
de rede e adicionamos a ele o dispositivo LTE para as Enbs
 NetDeviceContainer ueLteDevs = lteHelper->InstallUeDevice (ueNodes);//criando um objeto que representa o dispositivo de
rede e adicionamos a ele o dispositivo LTE para os Ues

 // Install the IP stack on the UEs
 internet.Install (ueNodes);//instalamos a pilha de protocolos da net nos Ues
 Ipv4InterfaceContainer ueIpIfaces;//criando uma objeto p representar a associacao ip-no
 ueIpIfaces = epcHelper->AssignUeIpv4Address (NetDeviceContainer (ueLteDevs));//acontece a alocacao de ips a cada
dispositivo de rede dos Ues a partir de uma faixa

 // Attach all UEs to the first eNodeB
 for (uint16_t i = 0; i < numberOfUes; i++)//for para contar a quantidades de Ues
  {
    lteHelper->Attach (ueLteDevs.Get (i), enbLteDevs.Get (0));//anexamos a Ue da posicao i ao primeiro enb (0)
    if(i==1){
            lteHelper->Attach (ueLteDevs.Get (i), enbLteDevs.Get (1));
    }
```

```
  }

            //----------RODANDO APLICAÇÕES FTP/TCP----------

 NS_LOG_LOGIC ("setting up applications");

 for(uint32_t u= 0; u<ueNodes.GetN(); ++u){//Obs
 Ptr<Node> ue = ueNodes.Get (u);//definindo um objeto que representara cada UE a cada laco
      // Set the default gateway for the UE definindo o gateway padrao p UE
      Ptr<Ipv4StaticRouting> ueStaticRouting = ipv4RoutingHelper.GetStaticRouting (ue->GetObject<Ipv4> ());//definindo o
objeto para fazer o roteamento estatico ipv4 dos UEs
      ueStaticRouting->SetDefaultRoute (epcHelper->GetUeDefaultGatewayAddress (), 1);//O gateway padrao definindo a rota
dos UEs

            uint16_t dlPort = 10000;
            ++dlPort;

            ApplicationContainer clientApps2; // CRIAR INTERFACE CLIENTE
            ApplicationContainer serverApps2; // CRIAR INTERFACE SERVIDOR
            Address sinkLocalAddress (InetSocketAddress (Ipv4Address::GetAny (), dlPort));
            PacketSinkHelper sinkHelper ("ns3::TcpSocketFactory",sinkLocalAddress);
            clientApps2 = sinkHelper.Install(ueNodes.Get(u));   // INSTALA INTERFACE SERVIDOR NO NO BACKGROUND
            OnOffHelper sourceHelper ("ns3::TcpSocketFactory", Address ());
            sourceHelper.SetAttribute ("OnTime", StringValue ("ns3::ConstantRandomVariable[Constant=1.0]"));
            sourceHelper.SetAttribute ("OffTime", StringValue ("ns3::ConstantRandomVariable[Constant=0.0]"));
            AddressValue remoteAddress (InetSocketAddress (ueIpIfaces.GetAddress(u), dlPort));
                  sourceHelper.SetAttribute ("Remote", remoteAddress);
                  sourceHelper.SetAttribute ("DataRate", DataRateValue(DataRate("1Gbps"))); // TAXA DE TRANSMISSAO
                  sourceHelper.SetAttribute ("PacketSize", UintegerValue (100000)); // DEFINE TAMANHO DO PACOTE
                  Config::SetDefault ("ns3::TcpSocket::RcvBufSize", UintegerValue (1000000000)); // DEFINE TAMANHO
BUFFER
                  serverApps2 = sourceHelper.Install (remoteHost);
                  clientApps2.Start (Seconds (1.0)); // INICIA APLICAÇÃO CLIENT
                  clientApps2.Stop (Seconds (simTime)); // FINALIZA APLICAÇÃO CLIENT
                  serverApps2.Start (Seconds (1.0)); // INICIA APLICAÇÃO SERVER
                  serverApps2.Stop (Seconds (simTime)); // FINALIZA APLICAÇÃO SERVER

                 //Ativando a portadora de radio
                        Ptr<EpcTft> tft = Create<EpcTft> ();
                        EpcTft::PacketFilter dlpf;
                        dlpf.localPortStart = dlPort;
                        dlpf.localPortEnd = dlPort;
                        tft->Add (dlpf);
                        EpcTft::PacketFilter ulpf;
                        ulpf.remotePortStart = dlPort;
                        ulpf.remotePortEnd = dlPort;
                        tft->Add (ulpf);
                        EpsBearer bearer (EpsBearer::NGBR_VIDEO_TCP_DEFAULT);
                        lteHelper->ActivateDedicatedEpsBearer (ueLteDevs.Get (u), bearer, tft);

 }
// lteHelper->EnableTraces ();

       /*Time startTime = Seconds (startTimeSeconds->GetValue ());
       serverApps.Start (startTime);
       clientApps.Start (startTime);*/

      // end for b



 // Add X2 inteface
 lteHelper->AddX2Interface (enbNodes);//configurando uma interface X2 entre os enbs

 // X2-based Handover
```

```
  //lteHelper->HandoverRequest (Seconds (0.100), ueLteDevs.Get (0), enbLteDevs.Get (0), enbLteDevs.Get (1));

  // Uncomment to enable PCAP tracing
  // p2ph.EnablePcapAll("lena-x2-handover-measures");

  lteHelper->EnablePhyTraces ();
  lteHelper->EnableMacTraces ();
  lteHelper->EnableRlcTraces ();
  lteHelper->EnablePdcpTraces ();
  Ptr<RadioBearerStatsCalculator> rlcStats = lteHelper->GetRlcStats ();
  rlcStats->SetAttribute ("EpochDuration", TimeValue (Seconds (1.0)));
  Ptr<RadioBearerStatsCalculator> pdcpStats = lteHelper->GetPdcpStats ();
  pdcpStats->SetAttribute ("EpochDuration", TimeValue (Seconds (1.0)));

  // connect custom trace sinks for RRC connection establishment and handover notification
  Config::Connect ("/NodeList/*/DeviceList/*/LteEnbRrc/ConnectionEstablished",
                   MakeCallback (&NotifyConnectionEstablishedEnb));
  Config::Connect ("/NodeList/*/DeviceList/*/LteUeRrc/ConnectionEstablished",
                   MakeCallback (&NotifyConnectionEstablishedUe));
  Config::Connect ("/NodeList/*/DeviceList/*/LteEnbRrc/HandoverStart",
                   MakeCallback (&NotifyHandoverStartEnb));
  Config::Connect ("/NodeList/*/DeviceList/*/LteUeRrc/HandoverStart",
                   MakeCallback (&NotifyHandoverStartUe));
  Config::Connect ("/NodeList/*/DeviceList/*/LteEnbRrc/HandoverEndOk",
                   MakeCallback (&NotifyHandoverEndOkEnb));
  Config::Connect ("/NodeList/*/DeviceList/*/LteUeRrc/HandoverEndOk",
                   MakeCallback (&NotifyHandoverEndOkUe));

  AnimationInterface anim ("simulation3.xml");
  anim.SetConstantPosition (enbNodes.Get(0), 100, 100, 0);
  anim.SetConstantPosition (enbNodes.Get(1), 100, 500, 0);
  anim.SetConstantPosition (enbNodes.Get(2), 500, 500, 0);
  anim.SetConstantPosition (enbNodes.Get(3), 500, 100, 0);

  //anim.SetConstantPosition (ueNodes.Get(0), 0, 100.0, 0);

 //Gnuplot parameters

    std::string fileNameWithNoExtension = "FlowVSThroughput_";
    std::string graphicsFileName        = fileNameWithNoExtension + ".png";
    std::string plotFileName            = fileNameWithNoExtension + ".plt";
    std::string plotTitle               = "Flow vs Throughput";
    std::string dataTitle               = "Throughput";

    // Instantiate the plot and set its title.
    Gnuplot gnuplot (graphicsFileName);
    gnuplot.SetTitle (plotTitle);

    // Make the graphics file, which the plot file will be when it
    // is used with Gnuplot, be a PNG file.
    gnuplot.SetTerminal ("png");

    // Set the labels for each axis.
    gnuplot.SetLegend ("Flow", "Throughput");


   Gnuplot2dDataset dataset;
   dataset.SetTitle (dataTitle);
   dataset.SetStyle (Gnuplot2dDataset::LINES_POINTS);

  //flowMonitor declaration
  FlowMonitorHelper fmHelper;
  Ptr<FlowMonitor> allMon = fmHelper.InstallAll();
  // call the flow monitor function
  ThroughputMonitor(&fmHelper, allMon, dataset);
```

```
  //-----------------FlowMonitor-JITTER----------------

    std::string fileNameWithNoExtension2 = "FlowVSJitter_";
    std::string graphicsFileName2                = fileNameWithNoExtension2 + ".png";
    std::string plotFileName2                    = fileNameWithNoExtension2 + ".plt";
    std::string plotTitle2                       = "Flow vs Jitter";
    std::string dataTitle2                       = "Jitter";

    Gnuplot gnuplot2 (graphicsFileName2);
    gnuplot2.SetTitle(plotTitle2);

  gnuplot2.SetTerminal("png");

  gnuplot2.SetLegend("Flow", "Jitter");

  Gnuplot2dDataset dataset2;
  dataset2.SetTitle(dataTitle2);
  dataset2.SetStyle(Gnuplot2dDataset::LINES_POINTS);

  //FlowMonitorHelper fmHelper;
  //Ptr<FlowMonitor> allMon = fmHelper.InstallAll();

  JitterMonitor(&fmHelper, allMon, dataset2);

//--------------FlowMonitor-DELAY-----------------

  std::string fileNameWithNoExtension3 = "FlowVSDelay_";
  std::string graphicsFileName3                = fileNameWithNoExtension3 + ".png";
  std::string plotFileName3                    = fileNameWithNoExtension3 + ".plt";
  std::string plotTitle3                       = "Flow vs Delay";
  std::string dataTitle3                       = "Delay";

      Gnuplot gnuplot3 (graphicsFileName3);
      gnuplot3.SetTitle(plotTitle3);

      gnuplot3.SetTerminal("png");

      gnuplot3.SetLegend("Flow", "Delay");

      Gnuplot2dDataset dataset3;
      dataset3.SetTitle(dataTitle3);
      dataset3.SetStyle(Gnuplot2dDataset::LINES_POINTS);

      DelayMonitor(&fmHelper, allMon, dataset3);


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

//Gnuplot ...continued
  gnuplot.AddDataset (dataset);
  // Open the plot file.
  std::ofstream plotFile (plotFileName.c_str());
  // Write the plot file.
  gnuplot.GenerateOutput (plotFile);
  // Close the plot file.
  plotFile.close ();

  //---ContJITTER---

  gnuplot2.AddDataset(dataset2);;
  std::ofstream plotFile2 (plotFileName2.c_str());
  gnuplot2.GenerateOutput(plotFile2);
  plotFile2.close();
```

```
  //---ContDelay---

  gnuplot3.AddDataset(dataset3);;
  std::ofstream plotFile3 (plotFileName3.c_str());
  gnuplot3.GenerateOutput(plotFile3);
  plotFile3.close();


  // GtkConfigStore config;
  // config.ConfigureAttributes ();

  Simulator::Destroy ();
  return 0;


}

void ThroughputMonitor (FlowMonitorHelper *fmhelper, Ptr<FlowMonitor> flowMon,Gnuplot2dDataset DataSet)
{
       double localThrou=0;
std::map<FlowId, FlowMonitor::FlowStats> flowStats = flowMon->GetFlowStats();
Ptr<Ipv4FlowClassifier> classing = DynamicCast<Ipv4FlowClassifier> (fmhelper->GetClassifier());
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator stats = flowStats.begin (); stats != flowStats.end (); ++stats)
{
                //     if(stats->first == 1){//IFFFFFFFFFFFFFFFFFFFFFFF
               Ipv4FlowClassifier::FiveTuple fiveTuple = classing->FindFlow (stats->first);
               std::cout<<"Flow ID                          : " << stats->first <<" ; "<< fiveTuple.sourceAddress <<" ----->
"<<fiveTuple.destinationAddress<<std::endl;
               std::cout<<"Tx Packets = " << stats->second.txPackets<<std::endl;
               std::cout<<"Rx Packets = " << stats->second.rxPackets<<std::endl;
          std::cout<<"Duration                 : "<<(stats->second.timeLastRxPacket.GetSeconds()-stats-
>second.timeFirstTxPacket.GetSeconds())<<std::endl;
               std::cout<<"Last Received Packet          : "<< stats->second.timeLastRxPacket.GetSeconds()<<"
Seconds"<<std::endl;
               std::cout<<"Throughput: " << stats->second.rxBytes * 8.0 / (stats->second.timeLastRxPacket.GetSeconds()-stats-
>second.timeFirstTxPacket.GetSeconds())/1024/1024  << " Mbps"<<std::endl;
          localThrou=(stats->second.rxBytes * 8.0 / (stats->second.timeLastRxPacket.GetSeconds()-stats-
>second.timeFirstTxPacket.GetSeconds())/1024/1024);
          DataSet.Add((double)Simulator::Now().GetSeconds(),(double) localThrou);
               std::cout<<"---------------------------------------------------------------------------"<<std::endl;
//             }//IFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
}
               Simulator::Schedule(Seconds(1),&ThroughputMonitor, fmhelper, flowMon,DataSet);
  //if(flowToXml)
     {
flowMon->SerializeToXmlFile ("ThroughputMonitor.xml", true, true);
     }

}

//------------------------Metodo-JITTER----------------------

double atraso1=0;
void JitterMonitor(FlowMonitorHelper *fmHelper, Ptr<FlowMonitor> flowMon, Gnuplot2dDataset Dataset2)
{
      double localJitter=0;
      double atraso2 =0;

      std::map<FlowId, FlowMonitor::FlowStats> flowStats2 = flowMon->GetFlowStats();
      Ptr<Ipv4FlowClassifier> classing2 = DynamicCast<Ipv4FlowClassifier> (fmHelper->GetClassifier());
      for(std::map<FlowId, FlowMonitor::FlowStats>::const_iterator stats2 = flowStats2.begin(); stats2 != flowStats2.end();
++stats2)
      {
                                    if(stats2->first == 1){//IFFFFFFFFFFF
```

```
        Ipv4FlowClassifier::FiveTuple fiveTuple2 = classing2->FindFlow (stats2->first);
std::cout<<"Flow ID : "<< stats2->first <<";"<< fiveTuple2.sourceAddress <<"------>"
<<fiveTuple2.destinationAddress<<std::endl;
std::cout<<"Tx Packets = " << stats2->second.txPackets<<std::endl;
std::cout<<"Rx Packets = " << stats2->second.rxPackets<<std::endl;
std::cout<<"Duration  : "<<(stats2->second.timeLastRxPacket.GetSeconds()-stats2-
>second.timeFirstTxPacket.GetSeconds())<<std::endl;
std::cout<<"Atraso: "<<stats2->second.timeLastRxPacket.GetSeconds()-stats2->second.timeLastTxPacket.GetSeconds()
<<"s"<<std::endl;
atraso2 = stats2->second.timeLastRxPacket.GetSeconds()-stats2->second.timeLastTxPacket.GetSeconds();
//atraso1 = stats2->second.timeFirstRxPacket.GetSeconds()-stats2->second.timeFirstTxPacket.GetSeconds();
std::cout<<"Jitter: "<< atraso2-atraso1 <<std::endl;
localJitter= atraso2-atraso1;//Jitter
Dataset2.Add((double)Simulator::Now().GetSeconds(), (double) localJitter);
std::cout<<"-----------------------------------------------------------------------------"<<std::endl;
                                    }//IFFFFFFFFFF

    atraso1 = atraso2;
    }

    Simulator::Schedule(Seconds(1),&JitterMonitor, fmHelper, flowMon, Dataset2);
    {
            flowMon->SerializeToXmlFile("JitterMonitor.xml", true, true);
    }
}

//-------------------Metodo-DELAY-----------------------------------

void  DelayMonitor(FlowMonitorHelper *fmHelper, Ptr<FlowMonitor> flowMon, Gnuplot2dDataset Dataset3)
{
double localDelay=0;

std::map<FlowId, FlowMonitor::FlowStats> flowStats3 = flowMon->GetFlowStats();
Ptr<Ipv4FlowClassifier> classing3 = DynamicCast<Ipv4FlowClassifier> (fmHelper->GetClassifier());
for(std::map<FlowId, FlowMonitor::FlowStats>::const_iterator stats3 = flowStats3.begin(); stats3 != flowStats3.end(); ++stats3)
{
        if(stats3->first == 1){//IFFFFFFFFFFF
      Ipv4FlowClassifier::FiveTuple fiveTuple3 = classing3->FindFlow (stats3->first);
      std::cout<<"Flow ID : "<< stats3->first <<";"<< fiveTuple3.sourceAddress <<"------>"
<<fiveTuple3.destinationAddress<<std::endl;
      localDelay = stats3->second.timeLastRxPacket.GetSeconds()-stats3->second.timeLastTxPacket.GetSeconds();
   Dataset3.Add((double)Simulator::Now().GetSeconds(), (double) localDelay);
   std::cout<<"-----------------------------------------------------------------------------"<<std::endl;
}//IFFFFFFFFF
}
Simulator::Schedule(Seconds(1),&DelayMonitor, fmHelper, flowMon, Dataset3);
{
 flowMon->SerializeToXmlFile("DelayMonitor.xml", true, true);
}
}
```