UNIVERSIDADE FEDERAL DO PARANÁ

ALEXANDRE MATEUS ZAVADNIAK

AVALIAÇÃO DOS MÉTODOS DE ESCALONAMENTO DO LTE (LONG TERM EVOLUTION) NA QUALIDADE DE VÍDEO STREAMING

CURITIBA

2016

ALEXANDRE MATEUS ZAVADNIAK

AVALIAÇÃO DOS MÉTODOS DE ESCALONAMENTO DO LTE (LONG TERM EVOLUTION) NA QUALIDADE DE VÍDEO STREAMING

Monografia apresentada como trabalho de conclusão de curso apresentado como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica no curso de graduação de Engenharia Elétrica com Ênfase em Sistemas Eletrônicos Embarcados, Setor de Tecnologia da Universidade Federal do Paraná.

Orientador: Prof. Dr. Carlos Marcelo Pedroso

CURITIBA

2016

Aos meus pais e familiares, que foram grandes incentivadores e que sempre acreditaram nos meus sonhos. Por toda base e estrutura que me dedicaram para eu alcançar meus objetivos e trilhar meu futuro.

RESUMO

As tecnologias de transmissão de vídeos em *streaming* vêm aumentando em uma velocidade considerável nos últimos anos. Incentivadas no âmbito da telefonia móvel, com o surgimento da tecnologia LTE (*Long Term Evolution*), possibilitam taxas de transmissão de até 100 Mbps de *download* em aparelhos portáteis. Um novo mercado surgiu e uma nova oportunidade na área de pesquisa: a transmissão de vídeos de alta resolução em redes móveis. Grandes empresas de tecnologia como a Google e NetFlix vêm visando o mercado de telefonia móvel e, consequentemente, investindo nesta área. Para que exista a transmissão em um meio compartilhado, surge a necessidade de escalonadores para distribuir os recursos. Este estudo avalia o desempenho de dois dos escalonadores mais utilizados no LTE: *Proportional Fair* e *Round Robin*, em diferentes situações de congestionamento da rede e compactação temporal do vídeo.

Palavras-chave: **Vídeo *streaming*. *Long Term Evolution* . *Escaladores*. *Tamanho do GOP*.**

ABSTRACT

The video streaming technologies are increasing their capabilities, allowing transmission rate of over 100 Mbps in LTE (Long Term Evolution) networks. A new market has emerged so as a new opportunity in the high quality video streaming over mobile networks. Technology companies such as Google and Netflix are focusing in the mobile phone market. The resource allocation in LTE networks is performed in radio base stations. This study aims to analyze the performance of two schedulers: proportional fair and round robin, in in different situations of network congestion and video encoding characteristics, such as the size of GOP (Group of Pictures) .
Keywords: ***Video streaming, Long Term Evolution, Scheduler, GOP size***

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

3GPP	3rd Generation Partnership Project
4G	4th Generation
ATSV	Advanced Television System Committee
AVC	Advanced Video Coding
DL	Downlink
DVB-T	Digital Video Broadcasting - Terrestrial
ENB	Evolved NodeB
EPC	Evolved Packet Core
FTP	File Transfer Protocol
HD	High Definition
HSPA	High Speed Packet Access
IC	Intervalo de Confiança
IP	Internet Protocol
ISDB-T	Integrated Services Digital Broadcasting - Terrestrial
ITU	International Telecommunication Union
LTE	Long Term Evolution
MIMO	Multiple-Input Multiple-Output
MOS	Mean Opinion Score
MPEG	Moving Picture Experts Group
MSE	Mean Square Error

NIST	National Institute of Standards and Technology
NS3	Network Simulator 3
OFDM	Orthogonal frequency-division multiplexing
OFDMA	Orthogonal frequency-division multiplexing Access
PAPR	Peak to Average Power Ratio
PF	Proportional Fair
PGW	Packet Data Network Gateway
PSNR	Peak Signal Noise Ratio
GOP	Group of Picture
QAM	Quadrature Amplitude Modulation
QoE	Quality of Experience
QoS	Quality of Service
QPSK	Quadrature Phase Shift Keying
RAN	Radio Access Network
RB	Resource Block
SGW	Serving Gateway
SBTVD	Sistema Brasileiro de Televisão Digital
SC-FDMA	Single Carrier Frequency Division Multiple Access
SSIM	Structural Similarity
M-LWDF	Maximum Largest Weighted Delay First
TCP	Transmission Control Protocol
TTI	Transmission Time Interval

UDP	User Datagram Protocol
UE	User Equipament
UL	Uplink

SUMÁRIO

1 INTRODUÇÃO

Segundo estudo realizado pela [Cisco, 2015] , intitulado Visual Networking Index, o total de tráfego dedicado a vídeo está em franca expansão. Em 2019 a cada segundo serão transmitidos o equivalente a um milhão de minutos em vídeos. Segundo o mesmo estudo, o tráfego de dados móveis deve crescer a uma taxa anual de 57% entre os anos de 2014 e 2019, atingindo 24,3 exabytes por mês em 2019. Em 2014 as redes móveis de quarta geração (4G) geraram 10 vezes mais tráfego do que a média dos sistemas não-4G.

Conforme estudo "*4G Mobile Broadband Evolution*"[Americas, 2015], o tráfego de dados móveis foi quase 30 vezes o tamanho da Internet global em 2000 até a metade de 2015. Os *Smartphones* foram responsáveis por quase 88% do meio trilhão de conexões adicionadas ao final do ano de 2014.

1.1 PÚBLICO ALVO

O escalonamento e recursos na transmissão LTE define qual equipamento de usuário podendo acessar o meio de transmissão nos domínios do tempo e frequência. A tendência é que o aumento do tráfego de vídeos de *streaming* impulsione a área de pesquisa e o mercado para melhor atender os principais requisitos de qualidade de serviço para esta aplicação.

O público alvo são as empresas de transmissão de vídeo *streaming* como o Netflix, o Youtube, e empresas fabricantes de hardware para estações rádio base. O escalonamento é realizado pela estação rádio base de forma centralizada, sendo disponíveis diversos algoritmos de escalonamento. Existem algoritmos para realizar a divisão "justa"de recursos entre os equipamentos de usuário, para maximizar o *throughput*, para atender requisitos de atraso de propagação, entre outros objetos. A aplicação de vídeo *streaming* necessita de parâmetros específicos de qualidade de serviço.

1.2 DIFERENCIAL DO PROJETO EM RELAÇÃO A OUTROS JÁ EXECUTADOS

Esta proposta de conclusão de curso tem como principal diferencial em relação a outros trabalhos publicados a avaliação de desempenho de dois algoritmos de escalonamento: o *Proportional Fair* (PF) e o *Round Robin* (RR).

As características de compactação do vídeo também podem influenciar na qualidade do vídeo. Foi adotado o codec MPEG 4 part 10, ou H.264/AVC (*Advanced Video Coding*), analisando a influência do tamanho do GOP no desempenho do algoritmo de escalonamento em uso.

2 OBJETIVOS

2.1 OBJETIVO GERAL

Avaliar a influência dos algoritmos de escalonamento RR e PF do LTE na qualidade de vídeo *streaming* através de métricas objetivas de avaliação de qualidade.

2.2 OBJETIVOS ESPECÍFICOS

Dentre os objetivos específicos podemos citar como prioridade:

- Revisão da literatura.
  - Sistemas LTE;
  - Codificação de vídeo, padrão H.264/AVC;
  - Métodos para avaliação da qualidade de vídeo;
- Definição da topologia de simulação para vídeo *streaming* sobre LTE utilizando o simulador NS3. O software *network simulator* 3 é a ferramenta proposta para executar a simulação. O NS3 é um simulador baseado em eventos discretos que tem sido utilizado pela comunidade de pesquisa na área de redes e possui módulos para simulação de sistemas LTE e para avaliação de qualidade de vídeo.
- Avaliação da qualidade do vídeo usando a Relação Sinal Ruído de Pico (PSNR) e Índice de Similaridade Estrutural (SSIM).
- Avaliação dos intervalos de confiança e realização análise estatística dos resultados de simulação e comparação.

3 METODOLOGIA

Foi utilizado o programa de simulação *Network Simulator versão* 3, instalado no servidor disponível no Departamento de Engenharia Elétrica. O simulador é baseado em eventos discretos, sendo desenvolvido especialmente para pesquisa e uso educacional. O NS-3 é um *software* livre, sob a licença GNU GPLv2.

O acesso ao servidor é por uma conta de usuário e para acessá-lo pode se utilizar uma conexão ssh, sendo possível se conectar ao servidor de qualquer lugar com acesso à Internet. Para codificação de vídeo foi usado o software FFmpeg, que é um programa em linha de comando que grava, converte e cria *stream* de áudio e vídeo em diversos formatos, sendo software livre de código aberto e domínio público.

A simulação ocorre em um cenário com diversos dispositivos móveis lançando tráfego concorrente, com o intuito de criar uma perturbação no meio de transmissão. Inicialmente foi utilizado como tráfego de fundo transferências de arquivo utilizando o protocolo UDP. A Figura 3.1 mostra a representação gráfica da proposta de projeto, onde:

- Codificação do arquivo de vídeo original formato YUV para MPEG 4 part 10 ou H.264/AVC;
- Simulação da transmissão do vídeo MPEG4 utilizando a rede LTE com diversos cenário de simulação, alterando os algoritmos de escalonamento e a quantidade de UE's;
- Decodificação do vídeo MPEG4 recebido para YUV;
- Análise da qualidade comparando-se o arquivo YUV original com o arquivo YUV recebido. A avaliação de qualidade foi realizada através das métricas PSNR (*Peak Signal to Noise Ratio*) e SSIM (*Structural similarity*).

3.1 STREAMING DE VÍDEO

Streaming de vídeo ou qualquer outro tipo de multimídia, é caracterizado como distribuição de dados, geralmente de multimídia em uma rede através de pacotes, enviando de forma continua para reprodução ao usuário.

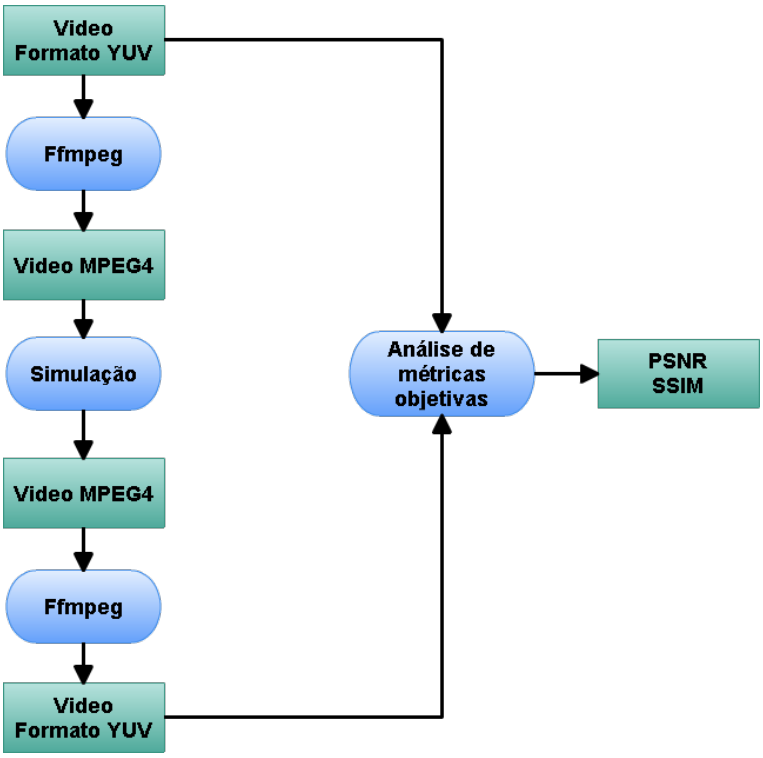

FIGURA 3.1 – REPRESENTAÇÃO GRÁFICA DA PROPOSTA DO PROJETO

Com o aumento da popularização de serviços de Internet de alta velocidade, o número de acessos para reprodução de mídia via *streaming* tem crescido consideravelmente. Dentro os principais fornecedores deste tipo de serviço temos o *Youtube* e o *Netflix*.

Durante o processo de transmissão podem ocorrer perdas de pacotes, devido a características da rede, dentre elas erros no meio físico e latência dos pacotes maiores que o tempo máximo do *buffer* de reprodução. Entre os principais defeitos que afetam o QoE tem-se demonstrado na Figura 3.2: (a) mosaico (b) *blocking*, (c) pixelização, fantasmas e congelamentos de quadros [Greengrass et al., 2009].

FIGURA 3.2 – PRINCIPAIS ERROS DE CODIFICAÇÃO E DECODIFICAÇÃO DE VIDEO

FONTE: [Greengrass et al., 2009]

3.2 AVALIAÇÃO DE QUALIDADE DE VÍDEO

Conforme [de M. Cardoso et al., 2012] a qualidade do vídeo é um fator subjetivo, logo depende não só das distorções presentes, mas também da definição de qualidade por parte de cada individuo. O modelo ideal para se verificar a qualidade visual de uma sequência de vídeo é a avaliação subjetiva. O principal fator de peso, é que a realização de testes subjetivos agregam, alto custo, tempo e uma complexa metodologia. Os modelos de avaliação objetiva são mais rápidos e possuem menor custo do que as soluções subjetivas, além disso, tais modelos indicam a existência de degradações imperceptíveis a visão humana.

As métricas subjetivas são baseadas na experiência do usuário como um todo, porém não estão relacionadas necessariamente ao vídeo, afetando principalmente a experiência do usuário sobre o serviço. Pode-se citar como um exemplo de influência nas métricas indiretas o tempo de resposta entre o usuário clicar no vídeo até sua reprodução.

As métricas objetivas possuem relações quantitativas entre o vídeo e o áudio transmitido, levando em consideração degradações que não podem ser percebidas por seres humanos.

3.2.1 Mean Opinion Score

Segundo [Martins and de Souza Pinheiro, 2014] o (*Mean Opinion Score*) é um método subjetivo para cálculo do QoE, que em obter um valor para a qualidade de vídeo através das opiniões dos espectadores. É necessário que todas as variáveis dos testes sejam controladas, incluindo desde o ambiente de exibição, a tela da projeção, distância e ângulo para a tela, vídeos a serem testados e inclusive os próprios espectadores. Os espectadores avaliam os vídeos atribuindo notas de 1 a 5, onde o maior valor representa uma melhor qualidade

3.2.2 PSNR

PSNR é a relação Sinal Ruído de Pico, definido como a relação entre a máxima energia de um sinal e seu ruído, para a análise da qualidade de vídeo é utilizando como referência vídeo original e vídeo degradado. Esta é uma técnica objetiva, com referência completa e direta. A unidade para sua representação é o dB (*decibel*). O valor de PSNR é baseado no erro

quadrático médio (*Mean Square Error*), MSE, definido por:

$$MSE = \frac{1}{l.c} \sum_{i=1}^{l} \sum_{j=1}^{c} [Y_o(i,j) - Y_r(i,j)]^2 \tag{3.1}$$

Onde l e c representam o número de linhas e colunas do quadro, $Y_o(i,j)$ representa o valor de luminância de um pixel especifico do vídeo original e $Y_r(i,j)$ a luminância do vídeo recebido. O mesmo procedimento pode ser realizado com os valores do croma Cb e do coma Cr.

O PSNR pode ser obtido através de

$$PSNR = 20.log_{10} \left(\frac{MAX}{\sqrt{MSE}} \right) \tag{3.2}$$

Onde MAX é o valor máximo do pixel, baseado no número de bits para a sua representação. Quanto maior o valor do PSNR menor a degradação do vídeo, este pode variar de zero a infinito. Porém para evitar a representação numérica do infinito, neste trabalho o limite do valor máximo do PSNR foi definido em 99 dB.

3.2.3 SSIM

SSIM, índice de similaridade estrutural, conforme afirmam [Wang et al., 2004] e [Cantarutti, 2013] é uma técnica métrica objetiva, com referência completa e direta que leva em consideração a medição da distorção estrutural da imagem, diferente do PSNR que leva em conta a medição do erro. O método leva em conta para a medição da degradação a correlação de perdas, distorção de luminância e distorção de contraste.

O índice de similaridade estrutural é calculado utilizando a seguinte relação:

$$SSIM = \frac{(2.\mu_x\mu_y + c_1)(2.\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{3.3}$$

Onde a média de x e y é representado por μ_x e μ_y, σ_x^2 e σ_y^2 representa a variância e σ_{xy} a covariância entre x e y. As variáveis c_1 e c_2 são utilizadas para estabilizar a divisão para valores baixos, são baseadas nos valores do número de bits para representar um pixel, conforme estabelece [Wang et al., 2004].

A concepção do SSIM é baseada na ideia que os pixels possuem interdependência, principalmente quando estão próximos. O valor do índice pode variar de 0 a 1, sendo 1 quando a imagem recebida é exatamente igual a imagem original. Na Tabela 1 está representada a correlação entre PSNR e SSIM facilitando assim a análise dos resultados em correlação aos diferentes tipos de métricas.

A Tabela 3.1 mostra a correlação entre o PSNR, SSIM e MOS.

TABELA 3.1 – Correlação PSNR, MOS E SSIM

Qualidade	**PSNR (dB)**	**MOS**	**SSIM**
Excelente	$>$ 37	5	$>$ 0,99
Bom	31 - 37	4	0,95 - 0,99
Razoável	25 - 31	3	0,88 - 0,95
Pobre	20 - 25	2	0,50 - 0,88
Péssimo	$<$ 20	1	$<$ 0,50

FONTE: [Zinner et al., 2010]

3.2.4 INTERVALO DE CONFIANÇA

Para garantir resultados mais confiáveis serão utilizadas ferramentas estatísticas baseada em simulações com sementes diferentes. Com base nesta gama de valores é possível chegar a um intervalo de confiança para o resultado.

A ideia é construir um intervalo de confiança para o parâmetro com uma probabilidade de $1-\alpha$ (nível de confiança) de que o intervalo contenha o verdadeiro parâmetro. Para cada conFiguração de simulação serão utilizadas as equações abaixo que respectivamente calculam o desvio padrão equação 3.4 e o IC na **??**

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2} \tag{3.4}$$

$$erro = \left[\overline{x} - a\frac{\sigma}{\sqrt{n}}; \overline{x} + a\frac{\sigma}{\sqrt{n}}\right] \tag{3.5}$$

O fator a é obtido através dos valores críticos da distribuição de t-Student com o nível de confiança e graus de liberdade necessários.

3.3 LTE - *LONG TERM EVOLUTION*

O *Long Term Evolution* (LTE) é atualmente a tecnologia utilizada para implementação da 4ª geração de comunicações móveis via rádio (4G), sendo introduzido na *Release* 8 e 9 do 3º *Generation Partnership Project* (3GPP).

As principais características da tecnologia LTE são:

- LTE suporta velocidades de até 100 Mbps no *downlink* e 50 Mbps no *uplink*, quando está usando uma largura de banda de 20 MHz, uma única antena de transmissão para o equipamento de utilizador (UE), e quatro antenas de recepção na estação de base.
- LTE gera três a quatro vezes a taxa de transferência no *downlink* e duas a três vezes a taxa de transferência no *uplink* em relação ao 3GPP no Release 6.
- LTE melhora a eficiência do espectro, se comparando ao Release 6 (HSDPA).
- LTE tem métodos duplex flexíveis. *Ambos Frequency Division Duplex* (FDD) e *Time Division Duplex* (TDD) são válidas alocações de espectro e permitir LTE acomodar várias larguras de banda de canais no espectro disponível.
- LTE suporta larguras de banda escalável de canais de RF. Os valores permitidos são 1,4, 3, 5, 10, 15 e 20 MHz.

3.4 RB - *RESOURCE BLOCK*

Segundo [Zhou et al., 2013] o acesso sem fio de LTE é baseado em um canal *Orthogonal frequency-division multiplexing Access* (OFDMA) de downlink de portadora única em divisão de acesso múltiplo (SC-FDMA). A unidade de mínima para alocação de recursos que o escalonador pode atribuir a um UE corresponde a dois RBs, como mostrado na Figura 3.3. A unidade mínima de alocação é chamada de PRB. Especificamente, um RB no domínio do tempo é um intervalo de tempo de transmissão (TTI), que tem a duração de tempo de 0,5m e no domínio da frequência o agrupamento de 12 *sub-carriers* de 180kHz.

O escalonador MAC está ativo para todos os TTI, alocando os RB's para as UEs de acordo com o algoritmo de escalonamento. O algoritmo de escalonamento não é definido pela norma e deve ser escolhido pelo fabricante da estação rádio base.

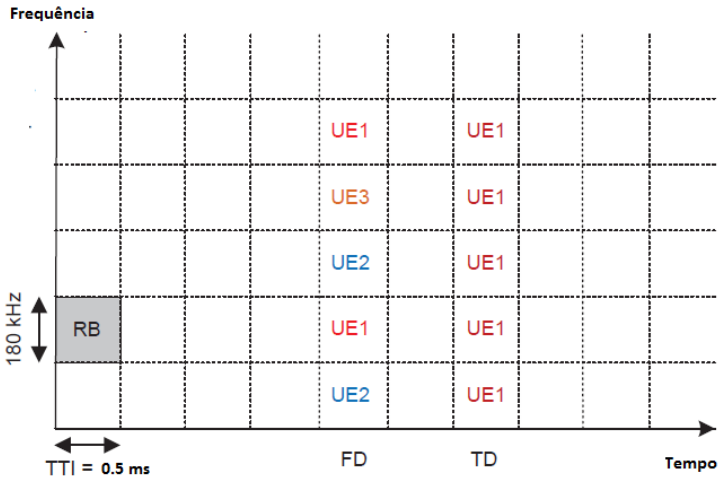

FIGURA 3.3 – ALOCAÇÃO DE RECURSOS NO DOMÍNIO DO TEMPO X DOMÍNIO DA FREQUÊNCIA

FONTE: [Zhou et al., 2013]

Cada RB pode ocupar um *slot* de 0,5 ms e o mesmo pode ocupar 7 símbolos em um portadora de 180 KHz divididos em 12 *sub-carriers* de 15 KHz criando uma relação de 84 símbolos. A taxa de transmissão pode variar dependendo do tipo de modulação em que ocorre a transmissão, sendo elas QPSK, 8-QAM, 16-QAM e 64-QAM. Cada modulação é decidida pela relação sinal ruído entre a UE e estação rádio base, caso o sinal entre os dois esteja fraco ocorre uma seleção diferente de modulação entre os aparelhos. Por exemplo, a modulação DPSK envia apenas dois símbolos por *time slot*, ideal para ambientes como muito ruído.

3.5 ESCALONADORES

O escalonador é responsável por alocar os PRBs. Nesta seção serão apresentados os escalonadores RR e PF, que são os escalonares mais usados atualmente.

3.5.0.1 ROUND ROBIN

Round-Robin é mais simples dos algoritmos de escalonamento. Em uma unidade de tempo TTI todos os PRB's representados por uma fila. Considerando-se o escalonamento de downlink, cria-se uma lista de UEs que devem receber dados no próximo ciclo de escalonamento. O primeiro PRB é alocado ao primeiro UE, o segundo PRB é alocado ao segundo UE e assim por diante. Quando atinge-se o último UE, alocação continua com o primeiro UE.

3.5.0.2 PROPORTIONAL FAIR

Segundo os autores [Dahlman et al., 2008] o escalonador de pacotes PF é uma das estratégias de escalonamento mais conhecidas e pesquisadas para a rede LTE.

O escalonador PF utiliza a informação de qualidade do canal para realizar uma alocação justa de recurso entre as diversas UEs. Por exemplo, em uma situação com duas UEs ativas, mas onde uma delas transmite com quatro vezes a velocidade da outra, o escalonador PF irá alocar quatro vezes mais PRBs para a UE com a mais baixa taxa, de forma que a taxa de transmissão seja aproximadamente equivalente.

Na prática o usuários que estão na borda do alcance da estação rádio base consomem mais recursos do que estão nas proximidades, pois o escalonador PF vai forçar uma alocação mais "justa". Por exemplo, caso um usuário 1 tenha um RB reservado pra garantir uma taxa de 1Mbit/s e outro usuário ira necessitar de 2 RB's pra garantir a mesma taxa de transmissão. Isso se deve pelo fato do tipo de modulação, variando entre BPSK, 4-QAM, 16-QAM e 64-QAM. Cada uma delas possui uma taxa de transmissão relacionada a quantidade de símbolos transmitido. A escolha da modulação e feita pela estação rádio base através da relação sinal-ruído.

3.6 CODIFICAÇÃO DE VÍDEO

Consiste em verificar as condições de entropia dos *frames*, e organizar o fluxo de bits de forma a remover as redundâncias e consequentemente comprimindo o vídeo. A compactação possui benefícios como menor espaço de armazenamento do vídeo e menor uso de banda para

transmissão.

Os métodos de compactação de vídeo podem ser classificados em métodos com e sem perdas. No primeiro caso o vídeo reconstituído após a codificação é idêntico ao vídeo original, entretanto com uma capacidade de compactação bastante limitada.

Segundo [Yamada et al., 2004] um sinal de vídeo possui uma taxa de bits que pode superar 1 Gbps, impossibilitando a transmissão em sistemas de TV digitais, como ATSV, DVB-T e ISDB-T, que transmitem uma taxa máxima de 20 Mbps, necessitam assim a compressão do vídeo para a transmissão.

Atualmente estes padrões de transmissão digital utilizam o MPEG2 para a compressão de vídeo. Em 1986 a ISO (*International Standard Organization*) organizou um grupo, denominado JPEG (*Joint Photograph Expert Group*), para desenvolver e padronizar métodos de compressão de sinal de imagens estáticas.

Em 1988 institutos desenvolveram e padronizaram sistemas de compactação de vídeo, baseados no JPEG. A evolução da codificação do video digital da elaboração destes padrões realizado pela ITU-T (*International Telecommunication Union - Telecommunication Standardization Sector*) e MPEG (Moving Picture Experts Group). O objeto de estudo deste trabalho está focado no padrão H264/MPEG-4 AVC.

Segundo [Manoel, 2007] a padronização de técnicas de codificação de vídeo digital é essencial para permitir que equipamentos de vídeo digital de diferentes fabricantes se comuniquem. Atualmente, os principais órgãos responsáveis por tal padronização são o ITU-T *Vídeo Coding Experts Group* (VCEG) e o ISO/IEC *Moving Picture Experts Group* (MPEG).

3.6.1 YUV

O padrão YUV é derivado do padrão RGB, sendo as três letras as iniciais das cores primárias, Red (Vermelho), Green (Verde) e Blue (Azul). Com as junções destas três cores com intensidades diferentes é possível resultar em diversas cores diferentes. Utilizando um byte para representar cada cor é possível obter mais de 16 milhões de combinações de tons, saturação e brilho.

O padrão YUV [Manoel, 2007], conhecido também como CCIR 601 ou YCbCr é um

padrão utilizado onde pode ser representada luminância e crominância. Onde Y representa a luminosidade (luminância), U ou Cb representa a informação da cor azul e V ou Cr representa a informação da cor vermelha.

Este padrão foi desenvolvido para transmissão de TV analógica possibilitando que o mesmo sinal seja utilizado tanto para televisores a cores e televisores preto e branco, pois a informação de luminância (Y) já é suficiente para imagens em tons de cinza Os valores da luminância Y e das componentes U e V podem ser correlacionados com os valores das componentes RGB, o padrão YUV tem uma nomenclatura para a representação ou não da subamostragem dos sinais de croma, Cb e Cr. No formato 4:4:4 para cada amostra de Y há uma amostra de Cb e Cr. No formato 4:2:2 a amostragem dos sinais de croma são reduzidos pela metade horizontalmente. No formato 4:2:0 a amostragem é reduzida pela metade horizontalmente e verticalmente, resultando em quatro amostragens de Y para cada amostra de Cb e Cr.

3.6.2 MPEG-4 (*Moving Pictures Experts Group*)

O MPEG-4 é destinado a codificação de dados multimídia em objetos numéricos, com o objetivo de aumentar a taxa de compreensão e garantir uma transmissão segura, são amplamente usados no mundo web e aparelhos portáteis. Segundo [Manoel, 2007] a melhoria de desempenho do MPEG-4 em relação ao MPEG-2 é considerável, tendo uma taxa de bits 50% menor, mantendo a qualidade semelhante.

A codificação MPEG-4 é uma das mais utilizadas atualmente, embora já existam outras soluções. Este trabalho será baseado na codificação MPEG-4 AVC (*Advanced Video Coding*) ou H.264. Existem dois tipos de redundância nos vídeos, as quais podem ser utilizadas para compactação. São elas: espacial e temporal. A espacial leva em consideração informações similares no mesmo quadro, como por exemplo um fundo preto. Já a redundância temporal é baseada em semelhanças numa sequência de quadros.

O MPEG explora a redundância temporal criando uma estrutura chamada GOP (Group of pictures), contendo os seguintes tipos de quadros:

- *Intra coded frames* (*frames* I): são codificadas separadamente, sem fazer referência

as imagens anteriores;

- *Predictive coded frames* (*frames* P): são codificadas utilizando informações do quadro I ou P anterior;
- *Bidirectionally predictive coded frames* (*frames* B): são codificadas utilizando informações do quadro I ou P anterior e posterior.

O GOP inicia sempre com um quadro I, seguido de quadros P E B. O tamanho do GOP compreende o número de quadros entre dois quadros I. Quanto maior o tamanho do GOP, maior a compactação e maior a perda de qualidade.

3.7 CENÁRIO DE SIMULAÇÃO

Foi proposto um cenário para cada método de avaliação e seus respectivos intervalos de confiança de 97,5%, sendo utilizado o PSNR e SSIM com relação a cada escalonador testado. Serão analisadosconfigurações de GOP alternando o número de UE's do tráfego de fundo em 10 , 20 , 30 , 40 e 50 concorrentes na RAN. Devido a uma limitação do módulo QoE monitor não serão utilizados quadros do tipo P.

Na codificação do vídeo do formato YUV para MPEG-4, será utilizada a seguinte linha de comando da ferramenta ffmpeg:

```
ffmpeg -s 4cif -r 30 -f rawvideo -vcodec rawvideo -pix_fmt yuv420p -i
video_entrada.yuv -vcodec libx264 -g 15 -bf 0 -s 4cif video_saida.mp4
```

Os vídeos para a simulação tem origem de um banco de vídeos de uso na comunidade científica, sendo de domínio público, disponíveis na biblioteca de vídeos para teste da Universidade de Hanôver Gottfried Wilhelm Leibniz, na Alemanha, acessado em ftp://ftp.tnt.uni-hannover.de/pub/svc/testsequences/.

3.7.1 Características dos vídeos do experimento

A Tabela 3.2 mostra as principais características de cada vídeo utilizado no estudo. Foi utilizada a codificação H.264 alterando o GOP.

Os vídeos escolhidos tem como características grandes mudanças de um *frame* para

TABELA 3.2 – Principais características dos vídeos de teste

Caracteristicas	**Decrição**	
Video	Soccer	Crew
Resolução	4CIF (704x576)	4CIF (704x576)
Frames	300	300
Cadência	30 fps	30 fps
Grau de movimento	Moderado	Grande movimento do fundo
GoP	06; 18; 30	06; 18; 30

outro, sendo bem movimentado e com diferenças significavas na compactação espacial do vídeo.

3.7.2 Simulador

O software para a simulação utilizado neste trabalho foi o *Network Simulator* versão 3, abreviado: NS-3, que é um simulador baseado em eventos discretos e foi desenvolvido especialmente para pesquisa e uso educacional. O NS-3 é um software livre, sob a licença GNU GPLv2 e está disponível ao público, elaborado em C++.

3.7.3 QoE monitor

O QoE Monitor é um módulo para o software de simulação de redes NS3 capaz de calcular a qualidade de experiência para *streaming* de vídeos através de técnicas objetivas, sendo elas PSNR e SSIM [Saladino et al., 2013].

A ferramenta suporta diversos formatos de vídeos, tal como, AVI, YUV, MP4, WMV e VOB. Independentemente do formato de entrada é inserido pelo usuário um arquivo YUV original e a ferramenta gera um arquivo YUV recebido, este último baseado no vídeo recebido no mesmo formato do enviado. Após o envio total é realizada o cálculo de PSNR e SSIM. Os resultados de PSNR e SSIM ao final da transmissão são representados em planilhas eletrônicas. São calculados valores de PSNR para as componentes Y, U e V. Todos os valores de SSIM e PSNR são calculados e apresentados quadro a quadro. Como resultado também são apresentados os valores de *jitter* e atrasos de cada pacote.

No artigo apresentado por [Saladino et al., 2013], é realizada uma comparação com

os resultados do QoE monitor com a ferramenta *Evalvid*, sendo esta amplamente utilizada pela comunidade para análise da qualidade de experiência. Tal comparação resultou em um erro máximo de 0,002 dB para PSNR e 0,02 para SSIM.

3.7.4 Tráfego de fundo

O TCP (*Transmission Control Protocol*) é um protocolo da camada de transporte que possui controle de fluxo, congestionamento e retransmissão de pacotes, sendo dimensionado para oferecer confiabilidade na transmissão em redes não-confiáveis e com parâmetros distintos.

O TCP detecta condições de congestionamento da rede e reduz a taxa de transmissão para adaptar-se à capacidade de transmissão disponível.

3.7.5 Topologia de rede

A Figura 3.4 apresenta a topologia da rede simulada. O PGW (*PDN Gateway*) é responsável pela conectividade entre os UE's e o roteador da rede cabeada. Ele é um elemento que permite implementação de políticas de controle, filtragem de pacotes por usuários, tarifação, espelhamento de tráfego e interceptação legal. A principal função é atuar como mediador de mobilidade entre as redes 2G/3G/4G.

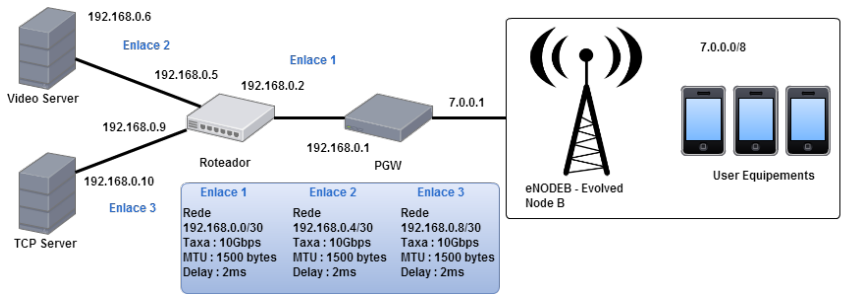

FIGURA 3.4 – CONFIGURAÇÃO DA REDE SIMULADA

4 RESULTADOS OBTIDOS

Em todos os testes foi calculado o erro para um nível de confiança de 97,5%. Foram executadas cinco simulações em cada cenário. A rede foi congestionada de forma a identificar principais influências entre o grau de utilização da rede, a variação do GOP, tipos de vídeos e escalonadores de recursos.

4.1 VÍDEO CREW

Nas tabelas 4.1, 4.3 e 4.2, 4.4 são apresentados os resultados de avaliação PSNR e SSIM respectivamente do vídeo crew. Com o aumento do número de usuários ativos simultaneamente a qualidade de vídeo se degrada de forma proporcional, conforme mostrado no gráfico da Figura 4.1.

A variação do GOP teve grande influência na qualidade. Observa-se que a degradação de qualidade é mais suave para as configurações de GOP maiores. Isso é causado pela maior exigência de banda dos vídeos codificados com GOP pequeno - neste caso, a perda de qualidade na codificação é menor, mas a compactação é também menor.

Não é possível observar diferença de desempenho na utilização do escalonador PF e RR.

TABELA 4.1 – Tabela de resultados vídeo Crew PF - PSNR

PF	CREW- PSNR (dB)					
N° de UE	GOP6	erro	GOP18	erro	GOP30	erro
10	37,05	0,62	39,75	0,00	39,66	0,00
20	29,65	0,79	32,22	1,72	33,18	1,38
30	29,26	0,33	31,89	0,14	32,20	0,33
40	28,16	0,27	30,32	0,52	30,50	0,67
50	28,36	0,66	29,66	0,79	30,94	0,81

TABELA 4.2 – Tabela de resultados vídeo Crew PF- SSIM

PF	CREW- SSIM					
N° de UE	GOP6	erro	GOP18	erro	GOP30	erro
10	0,89	0,016	0,97	0,000	0,97	0,000
20	0,71	0,021	0,76	0,048	0,79	0,033
30	0,70	0,006	0,75	0,003	0,77	0,014
40	0,67	0,014	0,70	0,011	0,73	0,012
50	0,67	0,018	0,69	0,012	0,71	0,019

TABELA 4.3 – Tabela de resultados vídeo Crew RR - PSNR

RR	CREW- PSNR(dB)					
N° de UE	GOP6	erro	GOP18	erro	GOP30	erro
10	34,35	0,07	39,74	0,00	39,66	0,00
20	29,37	1,59	32,98	0,59	33,44	0,03
30	28,96	0,44	30,94	0,25	32,04	0,54
40	28,28	0,41	30,40	0,49	30,69	1,63
50	28,17	0,67	30,39	0,34	30,46	1,27

TABELA 4.4 – Tabela de resultados vídeo Crew RR - SSIM

RR	CREW- SSIM					
N° de UE	GOP6	erro	GOP18	erro	GOP30	erro
10	0,82	0,001	0,97	0,000	0,97	0,000
20	0,70	0,038	0,77	0,043	0,80	0,057
30	0,70	0,009	0,72	0,014	0,76	0,025
40	0,68	0,021	0,71	0,011	0,73	0,026
50	0,67	0,013	0,71	0,008	0,73	0,030

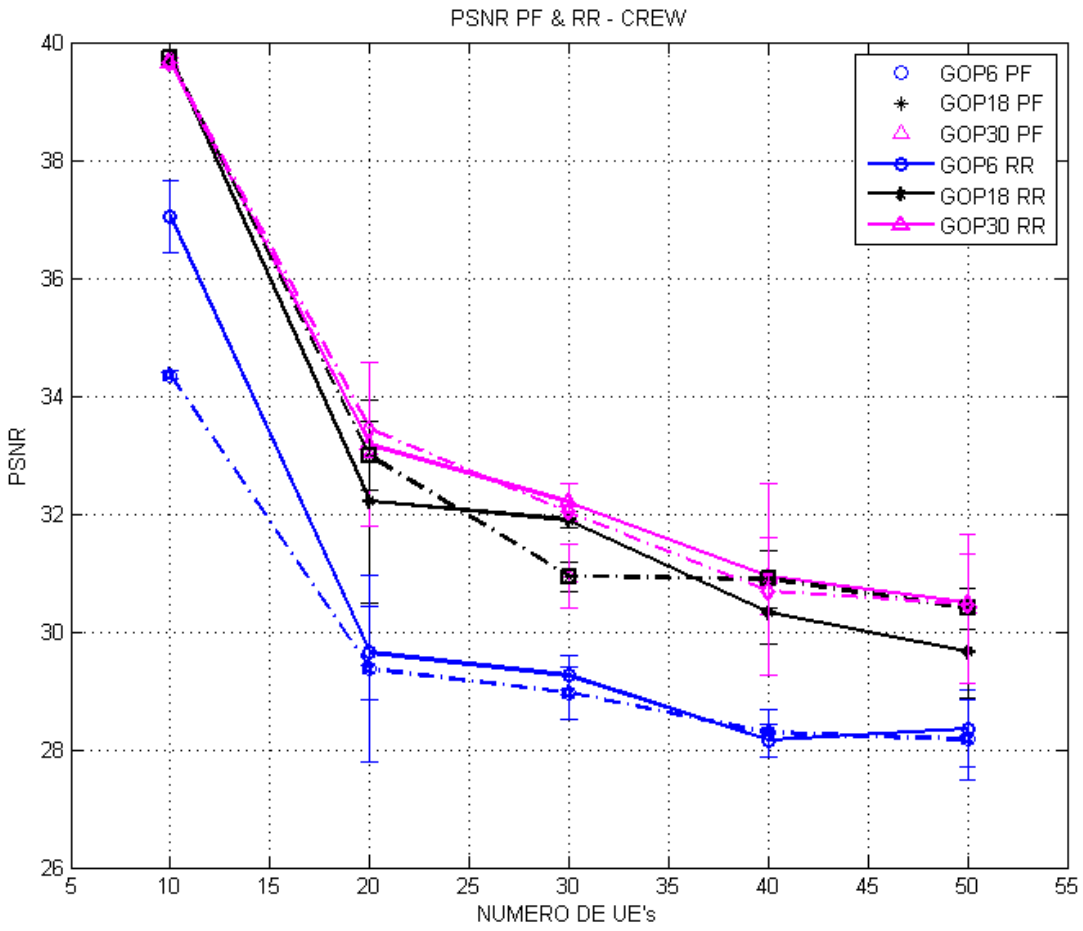

FIGURA 4.1 – RESULTADO GRÁFICO DO VÍDEO CREW GOP X UE'S- PSNR

4.2 SOCCER

Nas tabelas 4.5, 4.6 e 4.6, 4.8 são apresentados os valores do PSNR e SSIM respectivamente para simulação do vídeo Soccer. Com o aumento do usuários a qualidade percebida degrada de certa forma proporcional, sendo de fácil visualização no gráfico da Figura 4.2.

Da mesma forma que no vídeo anterior, pode ser observador que a degradação de qualidade é mais acentuada para as configuração de GOP menores.

Não é possível observar diferença de desempenho na utilização do escalonador PF e RR.

TABELA 4.5 – Tabela de resultados vídeo Soccer PF- PSNR

PF	SOCCER - PSNR (dB)					
N° de UE	GOP6	erro	GOP18	erro	GOP30	erro
10	37,24	0,35	39,75	0,00	40,05	0,00
20	29,73	0,55	32,22	1,72	36,08	1,17
30	28,49	0,25	31,89	0,14	35,12	0,56
40	28,81	0,21	30,32	0,52	33,80	1,12
50	28,12	1,10	29,66	0,79	32,75	0,36

TABELA 4.6 – Tabela de resultados vídeo Soccer PF- SSIM

PF	SOCCER - SSIM					
N° de UE	GOP6	erro	GOP18	erro	GOP30	erro
10	0,89	0,016	0,97	0,000	0,97	0,000
20	0,71	0,021	0,76	0,048	0,79	0,033
30	0,70	0,006	0,75	0,003	0,77	0,014
40	0,67	0,014	0,70	0,011	0,73	0,012
50	0,67	0,018	0,69	0,012	0,71	0,019

TABELA 4.7 – Tabela de resultados vídeo Soccer RR - PSNR

RR	SOCCER - PSNR (dB)					
N° de UE	GOP6	erro	GOP18	erro	GOP30	erro
10	36,07	0,01	40,24	0,00	40,05	0,00
20	29,57	1,64	35,30	1,41	35,88	1,21
30	29,13	0,89	33,29	0,72	35,77	0,18
40	28,67	0,39	31,70	0,79	33,61	0,65
50	28,76	0,44	30,93	1,02	32,33	0,99

TABELA 4.8 – Tabela de resultados vídeo Soccer RR - SSIM

RR	SOCCER - SSIM					
N° de UE	GOP6	erro	GOP18	erro	GOP30	erro
10	0,86	0,000	0,97	0,000	0,97	0,000
20	0,71	0,042	0,83	0,052	0,86	0,079
30	0,68	0,020	0,80	0,028	0,85	0,006
40	0,67	0,042	0,76	0,016	0,81	0,043
50	0,66	0,021	0,73	0,012	0,77	0,033

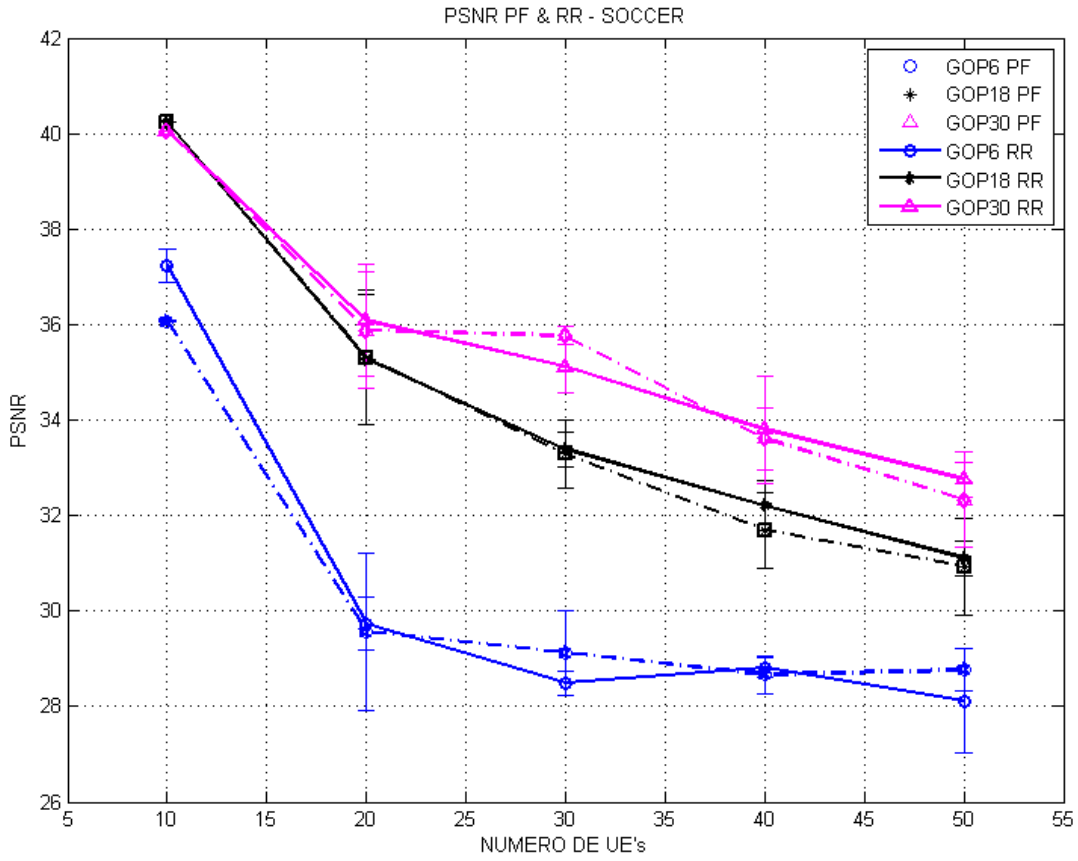

FIGURA 4.2 – RESULTADO GRÁFICO DO VÍDEO SOCCER GOP X UE'S- PSNR

4.3 DISCUSSÕES E ANÁLISE DE RESULTADOS

Analisando os resultados dos testes apresentados nos itens anteriores, é possível avaliar a qualidade de vídeo *streaming* de acordo com a variação do número de usuários de tráfego de fundo.

Não há variações significativas de qualidade em relação aos dois escalonadores testados. Isso se deve à distribuição espacial das UEs na célula. Foi adotado um posicionamento aleatório com distribuição uniforme entre 0 a 1000 metros da estação rádio base. Possivelmente a escolha da distância das UEs e o modelo de desvanecimento do canal levem a alteração de desempenho entre os algoritmos de escalonamento, mas não foi possível conduzir testes adicionais para confirmar esta hipótese.

Os gráficos mostram que o desempenho do RR é equivalente ao PF. No vídeo Crew é bem visível uma súbita melhora a partir de 30 UE's e ao stress de 50 UE's a qualidade

volta a piorar. A partir de 50 usuários existe uma forte influência de um fenômeno de gargalo localizado na eNODEb, o vídeo não é transmitido completamente, este fenômeno está ligado diretamente ao aumento de usuários de tráfego de fundo.

Nos testes é constatado que o GOP e o formato de compactação do vídeo influencia diretamente na qualidade de vídeo Uma das principais contribuições deste trabalho foi a observação de um fenômeno paradoxal: a vídeos codificados com GOP menor se degradam mais rapidamente conforme o congestionamento da rede aumenta. Isto pode ser explicado pela maior necessidade de banda de vídeos codificados com GOP menor.

5 CONCLUSÕES

Com o aumento dos usuários de equipamentos portáteis que utilizam a tecnologia de transmissão LTE surge o aumento de usuários de vídeo *streaming*. Impulsionando o sociedade cientifica a buscar novos caminhos e estudos do comportamento das redes LTE junto a transmissão de vídeo *streaming*.

Neste trabalho foram executados testes de transmissão de vídeo sobre redes LTE. A rede LTE foi estressada com a intenção de degradar a qualidade dos vídeos, alterando o número de usuários concorrentes ao meio de transmissão, de 10 UE's a 50 UE's. Foi estudada a influência do tamanho do GOP e do algoritmo de escalonamento de recursos do LTE. Foram alterados os escalonadores de canal de *downlink* utilizando os escalonadores RR e PF.

Foi observado que não houve diferença significativa no desempenho dos escalonadores PF e RR, sendo a variação de qualidade é imperceptível perante aos usuários. Foi observado que os vídeos com GOP maior degradam-se de forma mais suave na presença do congestionamento.

Como trabalho futuro, sugere-se o uso do protocolo UDP para congestionar a rede, além de alterar o modelo de posicionamento da UEs em torno da estação rádio base. Sugere-se também o uso de outros algoritmos de escalonamento.

BIBLIOGRAFIA

[Americas, 2015] Americas, G. (2015). 4G mobile broadband evolution: 5g release 12 & release 13 and beyond. *Report 4G Americas*.

[Cantarutti, 2013] Cantarutti, R. C. M. (2013). Modelo para previsão da qualidade de experiência na transmissão d vídeo sobre ip. Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Paraná.

[Cisco, 2015] Cisco (2015). Cisco visual networking index: Global mobile data traffic forecast update, 2014/2019. *Annual Update Cisco*.

[Dahlman et al., 2008] Dahlman, E., Parkvall, S., Skold, J., and Beming, P. (2008). *3G Evolution, Second Edition: HSPA and LTE for Mobile Broadband*. Academic Press, 2 edition.

[de M. Cardoso et al., 2012] de M. Cardoso, J. V., Mariano, A. C. S., Regis, C. D. M., and Alencar, M. S. (2012). Comparação das métricas objetivas de qualidade de vídeos baseadas na similaridade estrutural e na sensibilidade ao erro. 1.

[Greengrass et al., 2009] Greengrass, J., Evans, J., and Begen, A. (2009). Not all packets are equal, part i: Streaming video coding and sla requirements. *Internet Computing, IEEE*, 13(1):70–75.

[Manoel, 2007] Manoel, E. T. M. (2007). Codificação de vídeo h.264 - estudo de codificação mista de macroblocos.

[Martins and de Souza Pinheiro, 2014] Martins, D. F. and de Souza Pinheiro, F. (2014). AvaliaÇÃo da qualidade de streaming de vÍdeo em sistemas lte curitiba.

[Saladino et al., 2013] Saladino, D., Paganelli, A., and Casoni, M. (2013). A tool for multimedia quality assessment in ns3: Qoe monitor. *Simulation Modelling Practice and Theory*, 32:30–41.

[Wang et al., 2004] Wang, Z., Bovik, A., Sheikh, H., and Simoncelli, E. (2004). Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612.

[Yamada et al., 2004] Yamada, F., Sukys, F., Gunnar Bedicks Jr, C. A., Raunheitte, L. T. M., and E.Dantas, C. (2004). *Revista de Engenharia e Computação*. Instituto Presbiteriano Mackenzie, 5 edition.

[Zhou et al., 2013] Zhou, D., Song, W., Baldo, N., and Miozzo, M. (2013). Evaluation of tcp performance with lte downlink schedulers in a vehicular environment. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*, pages 1064–1069.

[Zinner et al., 2010] Zinner, T., Abboud, O., Hohlfeld, O., Hossfeld, T., and Tran-Gia, P. (2010). Towards qoe management for scalable video streaming. In *21th ITC Specialist Seminar on Multimedia Applications - Traffic, Performance and QoE*, Miyazaki, Jap.

A ANEXOS

A.1 CÓDIGO PF

```
/*  -*-UNIVERSIDADE FEDERAL DO PARANÁ -*-
*

* Título: DESEMPENHO DE STREAMING DE VIDEO EM TRANSMISSÃO LTE
*
*  Trabalho de Conclusão de Curso apresentado como exigência parcial
* para obtenção do grau de Bacharel em Engenharia Elétrica, Ênfase
* Sistemas Eletrônicos Embarcados, à Universidade Federal do Paraná.
*
* Aluno: Alexandre Mateus Zavadniak
*
*
*
* Orientador: Prof. Dr. Carlos Marcelo Pedroso
*/

#include "ns3/internet-stack-helper.h"
#include "ns3/ipv4-address-helper.h"
#include "ns3/ipv4-interface-container.h"
#include "ns3/ipv4-routing-table-entry.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/simulation-dataset.h"
#include "ns3/h264-packetizer.h"
#include "ns3/multimedia-application-sender.h"
#include "ns3/multimedia-application-receiver.h"
#include "ns3/multimedia-file-rebuilder.h"
#include "ns3/error-model.h"
#include "ns3/mpeg4-container.h"
#include "ns3/psnr-metric.h"
#include "ns3/ssim-metric.h"
#include "ns3/nstime.h"
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/udp-client-server-helper.h"
#include "ns3/packet-sink-helper.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/applications-module.h"
#include "ns3/lte-helper.h"
#include "ns3/epc-helper.h"
#include "ns3/mobility-module.h"
```

```
#include "ns3/point-to-point-helper.h"
#include "ns3/config-store.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/tcp-socket.h"
#include "ns3/olsr-module.h"
#include "ns3/dsdv-module.h"
#include <iostream>
#include <cstring>
#include <cassert>

#include "ns3/core-module.h"
#include "ns3/lte-module.h"
#include "ns3/config-store.h"

/*#include <ns3/tdmt-ff-mac-scheduler.h>
#include <ns3/ff-mac-scheduler.h>
#include <ns3/rr-ff-mac-scheduler.h>
#include <ns3/pf-ff-mac-scheduler.h>*/


using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("Simulacao - Escalonadores");

/* Simple FFMpeg wrapper for encoding/decoding purposes */
int Ffmpeg(std::string inputFileName, std::string outputFileName)
{
std::stringstream command;

command << "ffmpeg -i " << inputFileName <<" -vcodec rawvideo -pix_fmt yuv420p
 " << outputFileName;
std::cout << "ffmpeg -i " << inputFileName <<" -vcodec rawvideo -pix_fmt yuv420p
" << outputFileName;
/* I execute the command */
int returnCode = system(command.str().c_str());
if (returnCode != 0)
{
std::cout << "Error while calling FFMpeg, abort.\n";
}
return returnCode;
}

void ImprimirRotas (Ptr<Node>& n, std::string nome)
{
Ptr<Ipv4StaticRouting> routing = 0;
Ipv4StaticRoutingHelper routingHelper;
Ptr<Ipv4> ipv4 = n->GetObject<Ipv4> ();
uint32_t nbRoutes = 0;
```

```
Ipv4RoutingTableEntry route;

routing = routingHelper.GetStaticRouting (ipv4);

std::cout << "\nTabela de rotas do " << nome << std::endl;
std::cout << "Destination\t" << "Mask\t\t\t" << "Gateway\t\t" << "Interface\t"
<<  std::endl;

nbRoutes = routing->GetNRoutes ();
for (uint32_t i = 0; i < nbRoutes; i++)
{
route = routing->GetRoute (i);
std::cout << route.GetDest () << "\t"
<< route.GetDestNetworkMask() << "\t\t\t"
<< route.GetGateway   () << "\t\t"
<< route.GetInterface   () << "\t\t"
<< std::endl;
}
}
/**************************************
/*                                    *
/*         Rotina Principal           *
/*                                    *
/**************************************/

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


/*******************************
/* Inicialização das variáveis *
/*******************************/

uint16_t seed = 1;
uint16_t numberOfNodes1 = 15;
uint16_t numberOfNodes2 = 1;
std::string name ("TCC_versao_1.0");


CommandLine cmd;
cmd.AddValue("seed", "Seed of simulation", seed);
cmd.AddValue("numberOfNodes1", "Number of UE1 pairs", numberOfNodes1);
cmd.Parse(argc, argv);

ns3::SeedManager::SetSeed(seed);

std::string jitterBufferLength("200ms");
```

```
std::string simulationDuration("80s");
std::string transmitterStartTime("2s");
std::string transmitterStopTime("70s");
std::string receiverStartTime("1s");
std::string receiverStopTime("71s");
unsigned int mtu = 1400;

/************************************
/*    Extract the coded filename    *
/************************************/

std::string codedFilename(argv[1]);


/*************************
/* Opções para simulação *
/*************************/

bool enablePsnr = true;
bool enableSsim = true;

/************************************
/*       Filenames creation         *
/************************************/

size_t extPosition = codedFilename.rfind(".mp4");
std::string fileIdentifier = codedFilename.substr(0, extPosition);

std::string rawFilename = fileIdentifier + ".yuv";
std::string receivedFilename = fileIdentifier + ".received.mp4";
std::string receivedRawFilename = fileIdentifier + ".received.yuv";

std::string traceFileID = fileIdentifier + "-trace";
std::string metricFile = fileIdentifier + "-metric";

/************************************
/*         QoE monitor setup        *
/************************************/

SimulationDataset* dataset = new SimulationDataset();

dataset->SetOriginalRawFile(rawFilename);
dataset->SetOriginalCodedFile(codedFilename);
dataset->SetReceivedCodedFile(receivedFilename);
dataset->SetReceivedReconstructedFile(receivedRawFilename);
dataset->SetTraceFileId(traceFileID);

Mpeg4Container mpeg4ReadingContainer = Mpeg4Container(dataset->
```

```
GetOriginalCodedFile(),
Container::READ, AVMEDIA_TYPE_VIDEO);
mpeg4ReadingContainer.InitForRead();
dataset->SetSamplingInterval(mpeg4ReadingContainer.GetSamplingInterval());

H264Packetizer videoPacketizer(mtu, dataset);

/************************************
/* Cria elementos principais do LTE *
/************************************/

Ptr<LteHelper> LteObjeto = CreateObject<LteHelper> ();

LteObjeto->SetSchedulerType("ns3::PfFfMacScheduler");
// LteObjeto->SetSchedulerType ("ns3::FdMtFfMacScheduler"); // FD-MT scheduler
 - Maximum Throughput (MT) scheduler
// LteObjeto->SetSchedulerType ("ns3::TdMtFfMacScheduler"); // TD-MT scheduler
// LteObjeto->SetSchedulerType ("ns3::TtaFfMacScheduler"); // TTA scheduler -
Throughput to Average (TTA) scheduler
// LteObjeto->SetSchedulerType ("ns3::FdBetFfMacScheduler"); // FD-BET scheduler
 -  Blind Equal Throughput (BET) scheduler
// LteObjeto->SetSchedulerType ("ns3::TdBetFfMacScheduler"); // TD-BET scheduler


Ptr<EpcHelper>  EpcObjeto = CreateObject<EpcHelper> ();
LteObjeto -> SetEpcHelper (EpcObjeto);

/************************************
/*         Parâmetros do LTE        *
/************************************/

LteObjeto -> SetEnbAntennaModelType ("ns3::CosineAntennaModel");
LteObjeto -> SetEnbAntennaModelAttribute ("Orientation", DoubleValue (0));
LteObjeto -> SetEnbAntennaModelAttribute ("Beamwidth", DoubleValue (60));
LteObjeto -> SetEnbAntennaModelAttribute ("MaxGain", DoubleValue (0.0));


Ptr<Node> pgw = EpcObjeto->GetPgwNode ();

/************************************
/* Cria os elementos dos Servidores *
/************************************/

NodeContainer Servidores;
Servidores.Create (3);
Ptr<Node> RouterServer = Servidores.Get(0);
Ptr<Node> VideoServer = Servidores.Get(1);
Ptr<Node> TcpServer = Servidores.Get(2);
```

```
/***********************************
/* Cria conexão entre os elementos  *
/***********************************/

PointToPointHelper p2ph;
p2ph.SetDeviceAttribute ("DataRate", DataRateValue (DataRate ("10Gbps")));
// Enlace = 10Gbps
p2ph.SetDeviceAttribute ("Mtu", UintegerValue (1500));
// MTU = 1500 bits
p2ph.SetChannelAttribute ("Delay", StringValue("2ms"));
// Delay do canal = 2ms

NetDeviceContainer Enlace[3];

Enlace[0] = p2ph.Install (RouterServer, pgw);
Enlace[1] = p2ph.Install (RouterServer, VideoServer);
Enlace[2] = p2ph.Install (RouterServer, TcpServer);

/***********************************
/*   Cria rede entre os elementos   *
/***********************************/

InternetStackHelper internet;
internet.Install (Servidores);

Ipv4AddressHelper ipv4h;
ipv4h.SetBase ("192.168.0.0", "255.255.255.252");

Ipv4InterfaceContainer interfacesIP[3];

for(unsigned int i = 0; i < 3; i++)
{
interfacesIP[i] = ipv4h.Assign(Enlace[i]);
ipv4h.NewNetwork();
}

Ipv4StaticRoutingHelper ipv4Roteamento;
Ptr<Ipv4StaticRouting> RotaEstaticaEnlace1 = ipv4Roteamento.GetStaticRouting
 (RouterServer->GetObject<Ipv4> ());
RotaEstaticaEnlace1 -> AddNetworkRouteTo (Ipv4Address ("7.0.0.0"), Ipv4Mask
("255.0.0.0"), 1);

Ptr<Ipv4StaticRouting> RotaEstaticaEnlace2 = ipv4Roteamento.GetStaticRouting
 (pgw->GetObject<Ipv4> ());
//RotaEstaticaEnlace2-> AddNetworkRouteTo (Ipv4Address ("192.168.0.0"), Ipv4Mask
("255.255.0.0"), 2);
RotaEstaticaEnlace2-> SetDefaultRoute (interfacesIP[0].GetAddress (0), 2);
```

```
Ptr<Ipv4StaticRouting> RotaDefault1 = ipv4Roteamento.GetStaticRouting (VideoServer->
GetObject<Ipv4> ());
RotaDefault1->SetDefaultRoute (interfacesIP[1].GetAddress (0), 1);

Ptr<Ipv4StaticRouting> RotaDefault2 = ipv4Roteamento.GetStaticRouting (TcpServer->
GetObject<Ipv4> ());
RotaDefault2->SetDefaultRoute (interfacesIP[2].GetAddress (0), 1);

// Print IPs

std::cout << "IP Router/pgw: " << interfacesIP[0].GetAddress (0) << "\n";
std::cout << "IP Router/VideoServer: " << interfacesIP[1].GetAddress (0) << "\n";
std::cout << "IP Router/TcpServer: " << interfacesIP[2].GetAddress (0) << "\n";
std::cout << "IP Pgw: " << interfacesIP[0].GetAddress (1) << "\n";
std::cout << "IP VideoServer: " << interfacesIP[1].GetAddress (1) << "\n";
std::cout << "IP TcpServer: " << interfacesIP[2].GetAddress (1) << "\n";

// Print Routes Router

std::string routerString("Router");
std::string tcpString("TCPServer");
std::string videoString("VideoServer");
std::string pgwString("PGW");

ImprimirRotas(RouterServer,routerString);

ImprimirRotas(VideoServer,tcpString);

ImprimirRotas(TcpServer,videoString);

ImprimirRotas(pgw,pgwString);


/************************************
/*     Cria os elementos Móveis     *
/************************************/

NodeContainer ueNodesA;
NodeContainer ueNodesB;
NodeContainer enbNodes;
enbNodes.Create(1);
ueNodesA.Create(numberOfNodes1);
ueNodesB.Create(numberOfNodes2);

/************************************
/*  Instala o modelo de Mobilidade  *
/************************************/
```

```
Ptr<ListPositionAllocator> positionAlloc0 = CreateObject<ListPositionAllocator> ();
positionAlloc0->Add (Vector(0, 0, 0));

Ptr<UniformRandomVariable> rand;
rand = CreateObject<UniformRandomVariable>();
//rand -> SetAttribute("Min",IntegerValue (0));
//rand -> SetAttribute("Max",IntegerValue (500));


Ptr<ListPositionAllocator> positionAlloc1 = CreateObject<ListPositionAllocator> ();
for (uint16_t i = 0; i < numberOfNodes1; i++)
{
positionAlloc1->Add (Vector(rand->GetInteger(0,1000), rand->GetInteger(0,1000), 0));
}
Ptr<ListPositionAllocator> positionAlloc2 = CreateObject<ListPositionAllocator> ();

for (uint16_t i = 0; i < numberOfNodes2; i++)
{
positionAlloc2->Add (Vector(rand->GetInteger(0,500), rand->GetInteger(0,500), 0));
}

MobilityHelper mobility0;
mobility0.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility0.SetPositionAllocator(positionAlloc0);
mobility0.Install(enbNodes);

MobilityHelper mobility1;
mobility1.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility1.SetPositionAllocator(positionAlloc1);
mobility1.Install(ueNodesA);

MobilityHelper mobility2;
mobility2.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility2.SetPositionAllocator(positionAlloc2);
mobility2.Install(ueNodesB);

/************************************
/* Instala Dispositivos LTE nos nós *
/************************************/

NetDeviceContainer enbDevs;
NetDeviceContainer ueDevsA;
NetDeviceContainer ueDevsB;
enbDevs = LteObjeto->InstallEnbDevice (enbNodes);
ueDevsA = LteObjeto->InstallUeDevice (ueNodesA);
ueDevsB = LteObjeto->InstallUeDevice (ueNodesB);
```

```
/***********************************
/*      Instala IP nos UEs          *
/***********************************/

internet.Install (ueNodesA);
Ipv4InterfaceContainer uE_Interface1;
uE_Interface1 = EpcObjeto->AssignUeIpv4Address (NetDeviceContainer (ueDevsA));

internet.Install (ueNodesB);
Ipv4InterfaceContainer uE_Interface2;
uE_Interface2 = EpcObjeto->AssignUeIpv4Address (NetDeviceContainer (ueDevsB));

/***************************************
/* Defini EPC como gateway para os UEs *
/***************************************/

std::cout << "\nNumero de ueA: " << ueNodesA.GetN () << "\n";

for (uint32_t u = 0; u < ueNodesA.GetN (); u++)
{
Ptr<Node> ueNode_tcp = ueNodesA.Get (u);
Ptr<Ipv4StaticRouting> ueStaticRouting1 = ipv4Roteamento.GetStaticRouting
(ueNode_tcp->GetObject<Ipv4> ());
ueStaticRouting1->SetDefaultRoute (EpcObjeto->GetUeDefaultGatewayAddress (), 1);
std::cout << "IP do ueNodes1(" << u << "): " << uE_Interface1.GetAddress(u) << "\n";
}

std::cout << "\nNumero de ueB: " << ueNodesB.GetN () << "\n";

for (uint32_t u = 0; u < ueNodesB.GetN (); u++)
{
Ptr<Node> ueNode_video = ueNodesB.Get (u);
Ptr<Ipv4StaticRouting> ueStaticRouting2 = ipv4Roteamento.GetStaticRouting
 (ueNode_video->GetObject<Ipv4> ());
ueStaticRouting2->SetDefaultRoute (EpcObjeto->GetUeDefaultGatewayAddress (), 1);
std::cout << "IP do ueNodes2(" << u << "): " << uE_Interface2.GetAddress(u) << "\n";
}

/***********************************
/*      Anexa os UEs no eNB         *
/***********************************/

LteObjeto->Attach (ueDevsA, enbDevs.Get (0));
LteObjeto->Attach (ueDevsB, enbDevs.Get (0));


/***********************************
```

```
/*      Transmissão do video        *
/***********************************/

/* Receiver side application setup
* First I create an mpeg4 container for the received file (output) */
Mpeg4Container mpeg4Container = Mpeg4Container(dataset->GetReceivedCodedFile(),
Container::WRITE, AVMEDIA_TYPE_VIDEO);

/* Now I take the codec context from the reading context (input) and I copy
* its settings to the output codec context */
mpeg4Container.SetCodecContext(mpeg4ReadingContainer.GetCodecContext());

/* I copy the stream's settings from the reading context (input) to the
output context */
mpeg4Container.SetStream(mpeg4ReadingContainer.GetStream());

/* Now I can initialize the output container for writing */
mpeg4Container.InitForWrite();

/* Create the receiver's multimedia file rebuilder */
MultimediaFileRebuilder fileRebuilder(&mpeg4Container, dataset);

/* Create the receiver's side application */
Ptr<MultimediaApplicationReceiver> multimediaReceiver = CreateObject<
MultimediaApplicationReceiver> (ueNodesB.Get(0), dataset,
 Time(std::string(ererBufferLength)));

multimediaReceiver->SetupReceiverPort(400);
multimediaReceiver->SetupFileRebuilder(&fileRebuilder);

ueNodesB.Get(0)->AddApplication(multimediaReceiver);

multimediaReceiver->SetStartTime(Time(receiverStartTime));
multimediaReceiver->SetStopTime(Time(receiverStopTime));

/* Create the sender's side application */
Ptr<MultimediaApplicationSender> multimediaSender = CreateObject
<MultimediaApplicationSender>(VideoServer, &videoPacketizer, dataset);

multimediaSender->SetupDestinationAddress(uE_Interface2.GetAddress(0), 400);
VideoServer->AddApplication(multimediaSender);

multimediaSender->SetStartTime(Time(transmitterStartTime));
multimediaSender->SetStopTime(Time(transmitterStopTime));

/***********************************
/*      Trafego de Fundo em TCP     *
/***********************************/
```

```
uint16_t ulPort = 2000;
for (uint32_t u = 0; u < ueNodesA.GetN (); u++)
{

ApplicationContainer ctSenderContainer, ctReceiverContainer;

/* Sender side */
BulkSendHelper ctTcpSender("ns3::TcpSocketFactory",InetSocketAddress
(uE_Interface1.GetAddress(u), ulPort));
ctTcpSender.SetAttribute("MaxBytes", UintegerValue(0));
ctSenderContainer = ctTcpSender.Install(TcpServer);

/* Receiver side */

PacketSinkHelper ctTcpReceiver("ns3::TcpSocketFactory",InetSocketAddress
(Ipv4Address::GetAny(),ulPort));
ctReceiverContainer = ctTcpReceiver.Install(ueNodesA.Get(u));

ulPort++;

ctSenderContainer.Start(Seconds (1));
ctSenderContainer.Stop(Time(receiverStopTime));

/*ctReceiverContainer.Start(Seconds (1));
ctReceiverContainer.Stop(Time(receiverStopTime)); */
}


/* ========================================================*/

/* Flow monitor setup */
FlowMonitorHelper flowmon_helper;
Ptr<FlowMonitor> monitor = flowmon_helper.InstallAll();

monitor->SetAttribute("DelayBinWidth", ns3::DoubleValue(0.0005));
monitor->SetAttribute("JitterBinWidth", ns3::DoubleValue(0.0005));
monitor->SetAttribute("PacketSizeBinWidth", ns3::DoubleValue(20));


/* Actual simulation start */
std::cout << "Starting simulation\n";

Simulator::Stop (Time(simulationDuration));

Simulator::Run();

/* Print traces to file */
```

```
dataset->PrintTraces(true);

/* Decoding received file */
std::cout << "Received file decoding with FFMpeg...\n";
//assert(Ffmpeg(receivedFilename, receivedRawFilename) == 0);
Ffmpeg(receivedFilename, receivedRawFilename);

/* Decode the original h264 file to perform video comparison */
//std::cout << "Original YUV (raw) file decoding with FFMpeg...\n";
//assert(Ffmpeg(codedFilename, rawFilename) == 0);

if (enablePsnr)
{
/* Computing PSNR */
std::cout << "PSNR computing...";
std::cout.flush();

PsnrMetric psnr;
psnr.EvaluateQoe(rawFilename, receivedRawFilename);

/* Print the metric output without any header */
psnr.PrintResults(metricFile.c_str(), true);
std::cout << " done!\n";
}

if (enableSsim)
{
/* Computing SSIM */
std::cout << "SSIM computing...";
std::cout.flush();

SsimMetric ssim;
ssim.EvaluateQoe(rawFilename, receivedRawFilename);

/* Print the metric output without any header */
ssim.PrintResults(metricFile.c_str(), true);
std::cout << " done!\n";
}

/* Flow monitor post-processing */
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>
 (flowmon_helper.GetClassifier ());
std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();

for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator iterator =
 stats.begin ();
 iterator != stats.end (); iterator++)
{
```

```
/* Extract the classifier info. for the current flow */
Ipv4FlowClassifier::FiveTuple tuple = classifier->FindFlow (iterator->first);

uint16_t destinationPort = tuple.destinationPort;
uint16_t sourcePort = tuple.sourcePort;
Ipv4Address sourceAddress = tuple.sourceAddress;
Ipv4Address destinationAddress = tuple.destinationAddress;

/* I extract loss, throughput (in bps), delay, jitter and packet size */
double senderStreamDuration = (iterator->second.timeLastTxPacket - iterator->
second.timeFirstTxPacket).GetSeconds();
double receiverStreamDuration = (iterator->second.timeLastRxPacket - iterator->
second.timeFirstRxPacket).GetSeconds();
double sampleLoss = (iterator->second.txPackets - iterator->
second.rxPackets)/(double (iterator->second.txPackets));
double sampleThroughput = ((iterator->second.rxBytes * 8)/(double ((iterator->
second.timeLastRxPacket -
iterator->second.timeFirstRxPacket).GetSeconds())));
double sampleDelay = (iterator->second.delaySum).GetSeconds()/(double (iterator->
second.rxPackets));
double sampleJitter = (iterator->second.jitterSum).GetSeconds()/(double (iterator->
second.rxPackets - 1));
double samplePacketSize = (iterator->second.rxBytes)/(double (iterator->
second.rxPackets));
uint32_t sampledPacketSent = iterator->second.txPackets;
uint32_t samplePacketReceived = iterator->second.rxPackets;

std::cout << "Throughput, delay, jitter, loss, packet size, packet sent,
packet received\n";
std::cout << "Flow " << iterator->first << " ( " << sourceAddress << ":"
<< sourcePort <<
" -> " << destinationAddress << ":" << destinationPort << " )\n";
std::cout << std::fixed
<< std::setw(6) << std::setprecision(0) << sampleThroughput << ","
<< std::setw(8) << std::setprecision(6) << sampleDelay << ","
<< std::setw(8) << std::setprecision(6) << sampleJitter << ","
<< std::setw(8) << std::setprecision(6) << sampleLoss << ","
<< std::setw(5) << std::setprecision(0) << samplePacketSize << ","
<< std::setw(8) << std::setprecision(0) << sampledPacketSent << ","
<< std::setw(8) << std::setprecision(0) << samplePacketReceived << "\n";

std::cout.precision(6);
// printf '%s\n' sampleThroughput | paste -sd ' ' >> taxas.csv;
std::cout << "Sender stream duration: " << senderStreamDuration << "\n";
std::cout << "Receiver stream duration: " << receiverStreamDuration << "\n";
}

delete dataset;
```

```
Simulator::Destroy();
return 0;

}
```

A.2 CÓDIGO RR

```
/*  -*-UNIVERSIDADE FEDERAL DO PARANÁ -*-
*

* Título: DESEMPENHO DE STREAMING DE VIDEO EM TRANSMISSÃO LTE
*
*  Trabalho de Conclusão de Curso apresentado como exigência parcial
* para obtenção do grau de Bacharel em Engenharia Elétrica, Ênfase
* Sistemas Eletrônicos Embarcados, à Universidade Federal do Paraná.
*
* Aluno: Alexandre Mateus Zavadniak
*
*
*
* Orientador: Prof. Dr. Carlos Marcelo Pedroso
*/

#include "ns3/internet-stack-helper.h"
#include "ns3/ipv4-address-helper.h"
#include "ns3/ipv4-interface-container.h"
#include "ns3/ipv4-routing-table-entry.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/simulation-dataset.h"
#include "ns3/h264-packetizer.h"
#include "ns3/multimedia-application-sender.h"
#include "ns3/multimedia-application-receiver.h"
#include "ns3/multimedia-file-rebuilder.h"
#include "ns3/error-model.h"
#include "ns3/mpeg4-container.h"
#include "ns3/psnr-metric.h"
#include "ns3/ssim-metric.h"
#include "ns3/nstime.h"
#include "ns3/core-module.h"
#include "ns3/internet-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/udp-client-server-helper.h"
#include "ns3/packet-sink-helper.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/applications-module.h"
#include "ns3/lte-helper.h"
#include "ns3/epc-helper.h"
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-helper.h"
#include "ns3/config-store.h"
#include "ns3/flow-monitor-helper.h"
```

```
#include "ns3/tcp-socket.h"
#include "ns3/olsr-module.h"
#include "ns3/dsdv-module.h"
#include <iostream>
#include <cstring>
#include <cassert>

#include "ns3/core-module.h"
#include "ns3/lte-module.h"
#include "ns3/config-store.h"

/*#include <ns3/tdmt-ff-mac-scheduler.h>
#include <ns3/ff-mac-scheduler.h>
#include <ns3/rr-ff-mac-scheduler.h>
#include <ns3/pf-ff-mac-scheduler.h>*/


using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("Simulacao - Escalonadores");

/* Simple FFMpeg wrapper for encoding/decoding purposes */
int Ffmpeg(std::string inputFileName, std::string outputFileName)
{
std::stringstream command;

command << "ffmpeg -i " << inputFileName <<" -vcodec rawvideo
-pix_fmt yuv420p " << outputFileName;
std::cout << "ffmpeg -i " << inputFileName <<" -vcodec rawvideo
-pix_fmt yuv420p " << outputFileName;
/* I execute the command */
int returnCode = system(command.str().c_str());
if (returnCode != 0)
{
std::cout << "Error while calling FFMpeg, abort.\n";
}
return returnCode;
}

void ImprimirRotas (Ptr<Node>& n, std::string nome)
{
Ptr<Ipv4StaticRouting> routing = 0;
Ipv4StaticRoutingHelper routingHelper;
Ptr<Ipv4> ipv4 = n->GetObject<Ipv4> ();
uint32_t nbRoutes = 0;
Ipv4RoutingTableEntry route;

routing = routingHelper.GetStaticRouting (ipv4);
```

```
std::cout << "\nTabela de rotas do " << nome << std::endl;
std::cout << "Destination\t" << "Mask\t\t\t" << "Gateway\t\t"
<< "Interface\t" <<  std::endl;

nbRoutes = routing->GetNRoutes ();
for (uint32_t i = 0; i < nbRoutes; i++)
{
route = routing->GetRoute (i);
std::cout << route.GetDest () << "\t"
<< route.GetDestNetworkMask() << "\t\t\t"
<< route.GetGateway   () << "\t\t"
<< route.GetInterface   () << "\t\t"
<< std::endl;
}
}
/**************************************
/*                                    *
/*         Rotina Principal           *
/*                                    *
/**************************************/

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


/*******************************
/* Inicialização das variáveis *
/*******************************/

uint16_t seed = 1;
uint16_t numberOfNodes1 = 15;
uint16_t numberOfNodes2 = 1;
std::string name ("TCC_versao_1.0");


CommandLine cmd;
cmd.AddValue("seed", "Seed of simulation", seed);
cmd.AddValue("numberOfNodes1", "Number of UE1 pairs", numberOfNodes1);
cmd.Parse(argc, argv);

ns3::SeedManager::SetSeed(seed);

std::string jitterBufferLength("200ms");
std::string simulationDuration("80s");
std::string transmitterStartTime("2s");
std::string transmitterStopTime("70s");
```

```
std::string receiverStartTime("1s");
std::string receiverStopTime("71s");
unsigned int mtu = 1400;

/***********************************
/*    Extract the coded filename    *
/***********************************/

std::string codedFilename(argv[1]);


/************************
/* Opções para simulação *
/************************/

bool enablePsnr = true;
bool enableSsim = true;

/***********************************
/*       Filenames creation         *
/***********************************/

size_t extPosition = codedFilename.rfind(".mp4");
std::string fileIdentifier = codedFilename.substr(0, extPosition);

std::string rawFilename = fileIdentifier + ".yuv";
std::string receivedFilename = fileIdentifier + ".received.mp4";
std::string receivedRawFilename = fileIdentifier + ".received.yuv";

std::string traceFileID = fileIdentifier + "-trace";
std::string metricFile = fileIdentifier + "-metric";

/***********************************
/*         QoE monitor setup        *
/***********************************/

SimulationDataset* dataset = new SimulationDataset();

dataset->SetOriginalRawFile(rawFilename);
dataset->SetOriginalCodedFile(codedFilename);
dataset->SetReceivedCodedFile(receivedFilename);
dataset->SetReceivedReconstructedFile(receivedRawFilename);
dataset->SetTraceFileId(traceFileID);

Mpeg4Container mpeg4ReadingContainer = Mpeg4Container(dataset->
GetOriginalCodedFile(),
Container::READ, AVMEDIA_TYPE_VIDEO);
mpeg4ReadingContainer.InitForRead();
```

```
dataset->SetSamplingInterval(mpeg4ReadingContainer.GetSamplingInterval());

H264Packetizer videoPacketizer(mtu, dataset);

/************************************
/* Cria elementos principais do LTE *
/************************************/

Ptr<LteHelper> LteObjeto = CreateObject<LteHelper> ();

LteObjeto->SetSchedulerType("ns3::RrFfMacScheduler");
// LteObjeto->SetSchedulerType("ns3::PfFfMacScheduler");
// LteObjeto->SetSchedulerType ("ns3::FdMtFfMacScheduler");
// FD-MT scheduler - Maximum Throughput (MT) scheduler
// LteObjeto->SetSchedulerType ("ns3::TdMtFfMacScheduler");
// TD-MT scheduler
// LteObjeto->SetSchedulerType ("ns3::TtaFfMacScheduler");
// TTA scheduler - Throughput to Average (TTA) scheduler
// LteObjeto->SetSchedulerType ("ns3::FdBetFfMacScheduler");
// FD-BET scheduler -  Blind Equal Throughput (BET) scheduler
// LteObjeto->SetSchedulerType ("ns3::TdBetFfMacScheduler");
// TD-BET scheduler

Ptr<EpcHelper>  EpcObjeto = CreateObject<EpcHelper> ();
LteObjeto -> SetEpcHelper (EpcObjeto);

/************************************
/*         Parâmetros do LTE        *
/************************************/

LteObjeto -> SetEnbAntennaModelType ("ns3::CosineAntennaModel");
LteObjeto -> SetEnbAntennaModelAttribute ("Orientation", DoubleValue (0));
LteObjeto -> SetEnbAntennaModelAttribute ("Beamwidth", DoubleValue (60));
LteObjeto -> SetEnbAntennaModelAttribute ("MaxGain", DoubleValue (0.0));

/*ConfigStore inputConfig;
inputConfig.ConfigureDefaults();
inputConfig.SetFilename("input-default.txt");*/

Ptr<Node> pgw = EpcObjeto->GetPgwNode ();

/************************************
/* Cria os elementos dos Servidores *
/************************************/

NodeContainer Servidores;
Servidores.Create (3);
Ptr<Node> RouterServer = Servidores.Get(0);
```

```
Ptr<Node> VideoServer = Servidores.Get(1);
Ptr<Node> TcpServer = Servidores.Get(2);

/************************************
/* Cria conexão entre os elementos  *
/************************************/

PointToPointHelper p2ph;
p2ph.SetDeviceAttribute ("DataRate", DataRateValue (DataRate ("10Gbps")));
// Enlace = 10Gbps
p2ph.SetDeviceAttribute ("Mtu", UintegerValue (1500));
// MTU = 1500 bits
p2ph.SetChannelAttribute ("Delay", StringValue("2ms"));
// Delay do canal = 2ms

NetDeviceContainer Enlace[3];

Enlace[0] = p2ph.Install (RouterServer, pgw);
Enlace[1] = p2ph.Install (RouterServer, VideoServer);
Enlace[2] = p2ph.Install (RouterServer, TcpServer);

/************************************
/*   Cria rede entre os elementos   *
/************************************/

InternetStackHelper internet;
internet.Install (Servidores);

Ipv4AddressHelper ipv4h;
ipv4h.SetBase ("192.168.0.0", "255.255.255.252");

Ipv4InterfaceContainer interfacesIP[3];

for(unsigned int i = 0; i < 3; i++)
{
interfacesIP[i] = ipv4h.Assign(Enlace[i]);
ipv4h.NewNetwork();
}

Ipv4StaticRoutingHelper ipv4Roteamento;
Ptr<Ipv4StaticRouting> RotaEstaticaEnlace1 = ipv4Roteamento.GetStaticRouting
 (RouterServer->GetObject<Ipv4> ());
RotaEstaticaEnlace1 -> AddNetworkRouteTo (Ipv4Address ("7.0.0.0"), Ipv4Mask
("255.0.0.0"), 1);

Ptr<Ipv4StaticRouting> RotaEstaticaEnlace2 = ipv4Roteamento.GetStaticRouting
(pgw->GetObject<Ipv4> ());
//RotaEstaticaEnlace2-> AddNetworkRouteTo (Ipv4Address ("192.168.0.0"), Ipv4Mask
```

```
("255.255.0.0"), 2);
RotaEstaticaEnlace2-> SetDefaultRoute (interfacesIP[0].GetAddress (0), 2);

Ptr<Ipv4StaticRouting> RotaDefault1 = ipv4Roteamento.GetStaticRouting
(VideoServer->GetObject<Ipv4> ());
RotaDefault1->SetDefaultRoute (interfacesIP[1].GetAddress (0), 1);

Ptr<Ipv4StaticRouting> RotaDefault2 = ipv4Roteamento.GetStaticRouting
(TcpServer->GetObject<Ipv4> ());
RotaDefault2->SetDefaultRoute (interfacesIP[2].GetAddress (0), 1);

// Print IPs

std::cout << "IP Router/pgw: " << interfacesIP[0].GetAddress (0) << "\n";
std::cout << "IP Router/VideoServer: " << interfacesIP[1].GetAddress (0) << "\n";
std::cout << "IP Router/TcpServer: " << interfacesIP[2].GetAddress (0) << "\n";
std::cout << "IP Pgw: " << interfacesIP[0].GetAddress (1) << "\n";
std::cout << "IP VideoServer: " << interfacesIP[1].GetAddress (1) << "\n";
std::cout << "IP TcpServer: " << interfacesIP[2].GetAddress (1) << "\n";

// Print Routes Router

std::string routerString("Router");
std::string tcpString("TCPServer");
std::string videoString("VideoServer");
std::string pgwString("PGW");

ImprimirRotas(RouterServer,routerString);

ImprimirRotas(VideoServer,tcpString);

ImprimirRotas(TcpServer,videoString);

ImprimirRotas(pgw,pgwString);


/************************************
/*     Cria os elementos Móveis     *
/************************************/

NodeContainer ueNodesA;
NodeContainer ueNodesB;
NodeContainer enbNodes;
enbNodes.Create(1);
ueNodesA.Create(numberOfNodes1);
ueNodesB.Create(numberOfNodes2);

/************************************
```

```
/*  Instala o modelo de Mobilidade  *
/***********************************/

Ptr<ListPositionAllocator> positionAlloc0 = CreateObject<ListPositionAllocator> ();
positionAlloc0->Add (Vector(0, 0, 0));

Ptr<UniformRandomVariable> rand;
rand = CreateObject<UniformRandomVariable>();
//rand -> SetAttribute("Min",IntegerValue (0));
//rand -> SetAttribute("Max",IntegerValue (500));


Ptr<ListPositionAllocator> positionAlloc1 = CreateObject<ListPositionAllocator> ();
for (uint16_t i = 0; i < numberOfNodes1; i++)
{
positionAlloc1->Add (Vector(rand->GetInteger(0,1000), rand->GetInteger(0,1000), 0));
}
Ptr<ListPositionAllocator> positionAlloc2 = CreateObject<ListPositionAllocator> ();

for (uint16_t i = 0; i < numberOfNodes2; i++)
{
positionAlloc2->Add (Vector(rand->GetInteger(0,500), rand->GetInteger(0,500), 0));
}

MobilityHelper mobility0;
mobility0.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility0.SetPositionAllocator(positionAlloc0);
mobility0.Install(enbNodes);

MobilityHelper mobility1;
mobility1.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility1.SetPositionAllocator(positionAlloc1);
mobility1.Install(ueNodesA);

MobilityHelper mobility2;
mobility2.SetMobilityModel("ns3::ConstantPositionMobilityModel");
mobility2.SetPositionAllocator(positionAlloc2);
mobility2.Install(ueNodesB);

/***********************************
/* Instala Dispositivos LTE nos nós *
/***********************************/

NetDeviceContainer enbDevs;
NetDeviceContainer ueDevsA;
NetDeviceContainer ueDevsB;
enbDevs = LteObjeto->InstallEnbDevice (enbNodes);
ueDevsA = LteObjeto->InstallUeDevice (ueNodesA);
```

```
ueDevsB = LteObjeto->InstallUeDevice (ueNodesB);


/************************************
/*      Instala IP nos UEs          *
/************************************/

internet.Install (ueNodesA);
Ipv4InterfaceContainer uE_Interface1;
uE_Interface1 = EpcObjeto->AssignUeIpv4Address (NetDeviceContainer (ueDevsA));

internet.Install (ueNodesB);
Ipv4InterfaceContainer uE_Interface2;
uE_Interface2 = EpcObjeto->AssignUeIpv4Address (NetDeviceContainer (ueDevsB));

/************************************
/* Defini EPC como gateway para os UEs *
/************************************/

std::cout << "\nNumero de ueA: " << ueNodesA.GetN () << "\n";

for (uint32_t u = 0; u < ueNodesA.GetN (); u++)
{
Ptr<Node> ueNode_tcp = ueNodesA.Get (u);
Ptr<Ipv4StaticRouting> ueStaticRouting1 = ipv4Roteamento.GetStaticRouting
 (ueNode_tcp->GetObject<Ipv4> ());
ueStaticRouting1->SetDefaultRoute (EpcObjeto->GetUeDefaultGatewayAddress (), 1);
std::cout << "IP do ueNodes1(" << u << "): " << uE_Interface1.GetAddress(u) << "\n";
}

std::cout << "\nNumero de ueB: " << ueNodesB.GetN () << "\n";

for (uint32_t u = 0; u < ueNodesB.GetN (); u++)
{
Ptr<Node> ueNode_video = ueNodesB.Get (u);
Ptr<Ipv4StaticRouting> ueStaticRouting2 = ipv4Roteamento.GetStaticRouting
(ueNode_video->GetObject<Ipv4> ());
ueStaticRouting2->SetDefaultRoute (EpcObjeto->GetUeDefaultGatewayAddress (), 1);
std::cout << "IP do ueNodes2(" << u << "): " << uE_Interface2.GetAddress(u) << "\n";
}

/************************************
/*      Anexa os UEs no eNB         *
/************************************/

LteObjeto->Attach (ueDevsA, enbDevs.Get (0));
LteObjeto->Attach (ueDevsB, enbDevs.Get (0));
```

```
/***********************************
/*      Transmissão do video        *
/***********************************/

/* Receiver side application setup
* First I create an mpeg4 container for the received file (output) */
Mpeg4Container mpeg4Container = Mpeg4Container(dataset->GetReceivedCodedFile(),
Container::WRITE, AVMEDIA_TYPE_VIDEO);

/* Now I take the codec context from the reading context (input) and I copy
* its settings to the output codec context */
mpeg4Container.SetCodecContext(mpeg4ReadingContainer.GetCodecContext());

/* I copy the stream's settings from the reading context (input) to the output
 context */
mpeg4Container.SetStream(mpeg4ReadingContainer.GetStream());

/* Now I can initialize the output container for writing */
mpeg4Container.InitForWrite();

/* Create the receiver's multimedia file rebuilder */
MultimediaFileRebuilder fileRebuilder(&mpeg4Container, dataset);

/* Create the receiver's side application */
Ptr<MultimediaApplicationReceiver> multimediaReceiver = CreateObject<
MultimediaApplicationReceiver> (ueNodesB.Get(0), dataset, Time(std::string(jitterBuf

multimediaReceiver->SetupReceiverPort(400);
multimediaReceiver->SetupFileRebuilder(&fileRebuilder);

ueNodesB.Get(0)->AddApplication(multimediaReceiver);

multimediaReceiver->SetStartTime(Time(receiverStartTime));
multimediaReceiver->SetStopTime(Time(receiverStopTime));

/* Create the sender's side application */
Ptr<MultimediaApplicationSender> multimediaSender = CreateObject<MultimediaApplicatio
(VideoServer, &videoPacketizer, dataset);

multimediaSender->SetupDestinationAddress(uE_Interface2.GetAddress(0), 400);
VideoServer->AddApplication(multimediaSender);

multimediaSender->SetStartTime(Time(transmitterStartTime));
multimediaSender->SetStopTime(Time(transmitterStopTime));

/***********************************
/*      Trafego de Fundo em TCP     *
```

```
/***********************************/

uint16_t ulPort = 2000;
for (uint32_t u = 0; u < ueNodesA.GetN (); u++)
{

ApplicationContainer ctSenderContainer, ctReceiverContainer;

/* Sender side */
BulkSendHelper ctTcpSender("ns3::TcpSocketFactory",InetSocketAddres
s(uE_Interface1.GetAddress(u), ulPort));
ctTcpSender.SetAttribute("MaxBytes", UintegerValue(0));
ctSenderContainer = ctTcpSender.Install(TcpServer);

/* Receiver side */

PacketSinkHelper ctTcpReceiver("ns3::TcpSocketFactory",InetSocketAddress
(Ipv4Address::GetAny(), ulPort));
ctReceiverContainer = ctTcpReceiver.Install(ueNodesA.Get(u));

ulPort++;

ctSenderContainer.Start(Seconds (1));
ctSenderContainer.Stop(Time(receiverStopTime));

/*ctReceiverContainer.Start(Seconds (1));
ctReceiverContainer.Stop(Time(receiverStopTime)); */
}


/* ======================================================*/

/* Flow monitor setup */
FlowMonitorHelper flowmon_helper;
Ptr<FlowMonitor> monitor = flowmon_helper.InstallAll();

monitor->SetAttribute("DelayBinWidth", ns3::DoubleValue(0.0005));
monitor->SetAttribute("JitterBinWidth", ns3::DoubleValue(0.0005));
monitor->SetAttribute("PacketSizeBinWidth", ns3::DoubleValue(20));


/* Actual simulation start */
std::cout << "Starting simulation\n";

Simulator::Stop (Time(simulationDuration));

Simulator::Run();
```

```
/* Print traces to file */
dataset->PrintTraces(true);

/* Decoding received file */
std::cout << "Received file decoding with FFMpeg...\n";
//assert(Ffmpeg(receivedFilename, receivedRawFilename) == 0);
Ffmpeg(receivedFilename, receivedRawFilename);

/* Decode the original h264 file to perform video comparison */
//std::cout << "Original YUV (raw) file decoding with FFMpeg...\n";
//assert(Ffmpeg(codedFilename, rawFilename) == 0);

if (enablePsnr)
{
/* Computing PSNR */
std::cout << "PSNR computing...";
std::cout.flush();

PsnrMetric psnr;
psnr.EvaluateQoe(rawFilename, receivedRawFilename);

/* Print the metric output without any header */
psnr.PrintResults(metricFile.c_str(), true);
std::cout << " done!\n";
}

if (enableSsim)
{
/* Computing SSIM */
std::cout << "SSIM computing...";
std::cout.flush();

SsimMetric ssim;
ssim.EvaluateQoe(rawFilename, receivedRawFilename);

/* Print the metric output without any header */
ssim.PrintResults(metricFile.c_str(), true);
std::cout << " done!\n";
}

/* Flow monitor post-processing */
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>
(flowmon_helper.GetClassifier ());
std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats();

for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator iterator =
stats.begin (); iterator != stats.end (); iterator++)
{
```

```
/* Extract the classifier info. for the current flow */
Ipv4FlowClassifier::FiveTuple tuple = classifier->FindFlow (iterator->first);

uint16_t destinationPort = tuple.destinationPort;
uint16_t sourcePort = tuple.sourcePort;
Ipv4Address sourceAddress = tuple.sourceAddress;
Ipv4Address destinationAddress = tuple.destinationAddress;

/* I extract loss, throughput (in bps), delay, jitter and packet size */
double senderStreamDuration = (iterator->second.timeLastTxPacket - iterator->
second.timeFirstTxPacket).GetSeconds();
double receiverStreamDuration = (iterator->second.timeLastRxPacket - iterator->
second.timeFirstRxPacket).GetSeconds();
double sampleLoss = (iterator->second.txPackets - iterator->
second.rxPackets)/(double (iterator->second.txPackets));
double sampleThroughput = ((iterator->second.rxBytes * 8)/(double ((iterator->
second.timeLastRxPacket -
iterator->second.timeFirstRxPacket).GetSeconds())));
double sampleDelay = (iterator->second.delaySum).GetSeconds()/(double (iterator->
second.rxPackets));
double sampleJitter = (iterator->second.jitterSum).GetSeconds()/(double (iterator->
second.rxPackets - 1));
double samplePacketSize = (iterator->second.rxBytes)/(double (iterator->
second.rxPackets));
uint32_t sampledPacketSent = iterator->second.txPackets;
uint32_t samplePacketReceived = iterator->second.rxPackets;

std::cout << "Throughput, delay, jitter, loss, packet size, packet sent,
packet received\n";
std::cout << "Flow " << iterator->first << " ( " << sourceAddress << ":"
<< sourcePort <<
" -> " << destinationAddress << ":" << destinationPort << " )\n";
std::cout << std::fixed
<< std::setw(6) << std::setprecision(0) << sampleThroughput << ","
<< std::setw(8) << std::setprecision(6) << sampleDelay << ","
<< std::setw(8) << std::setprecision(6) << sampleJitter << ","
<< std::setw(8) << std::setprecision(6) << sampleLoss << ","
<< std::setw(5) << std::setprecision(0) << samplePacketSize << ","
<< std::setw(8) << std::setprecision(0) << sampledPacketSent << ","
<< std::setw(8) << std::setprecision(0) << samplePacketReceived << "\n";

std::cout.precision(6);
// printf '%s\n' sampleThroughput | paste -sd ' ' >> taxas.csv;
std::cout << "Sender stream duration: " << senderStreamDuration << "\n";
std::cout << "Receiver stream duration: " << receiverStreamDuration << "\n";
}

delete dataset;
```

```
Simulator::Destroy();
return 0;

}
```