UNIVERSIDADE FEDERAL DO PARANÁ

EVANDRO LUIS COPERCINI
JOEL APARECIDO BARBOSA JUNIOR

DESENVOLVIMENTO DE NOVO ALGORITMO DE ESCALONAMENTO PARA MELHORAR A RELAÇÃO DE USUÁRIOS DE VÍDEO ATENDIDOS COM QUALIDADE EM REDES LTE

CURITIBA
2016

EVANDRO LUIS COPERCINI
JOEL APARECIDO BARBOSA JUNIOR

DESENVOLVIMENTO DE NOVO ALGORITMO DE ESCALONAMENTO PARA MELHORAR A RELAÇÃO DE USUÁRIOS DE VÍDEO ATENDIDOS COM QUALIDADE EM REDES LTE

Trabalho de conclusão de curso apresentado como requisito parcial para obtenção do grau de Engenheiro Eletricista no Curso de Graduação em Engenharia Elétrica, Setor de Tecnologia da Universidade Federal do Paraná

Orientador: Prof. Dr. Carlos Marcelo Pedroso

CURITIBA

2016

TERMO DE APROVAÇÃO

EVANDRO LUIS COPERCINI
JOEL APARECIDO BARBOSA JUNIOR

DESENVOLVIMENTO DE NOVO ALGORITMO DE ESCALONAMENTO PARA MELHORAR A RELAÇÃO DE USUÁRIOS DE VÍDEO ATENDIDOS COM QUALIDADE EM REDES LTE

Trabalho de conclusão de curso aprovado como requisito parcial para obtenção do grau de Engenheiro Eletricista no Curso de Graduação em Engenharia Elétrica, Setor de Tecnologia da Universidade Federal do Paraná, pela seguinte banca examinadora:

_____

Prof. Dr. Carlos Marcelo Pedroso

Orientador – Departamento de Engenharia Elétrica, UFPR

_____

Prof. Dr. Evelio Martín García Fernández

Departamento de Engenharia Elétrica, UFPR

_____

Prof. Ph.D. Horácio Tertuliano dos Santos Filho,

Departamento de Engenharia Elétrica, UFPR

Curitiba, 13 de dezembro de 2016

AGRADECIMENTOS

Esses anos foram muito importantes para nossa formação como profissionais de engenharia elétrica, muitas pessoas fizeram parte desse caminho, em especial nossos pais e familiares, que sempre nos apoiaram durante essa trajetória, a Deus por todas as graças que Ele nos concedeu, a todos os professores e técnicos que auxiliaram em nossa graduação de alguma forma, direta ou indiretamente, em especial ao nosso orientador Prof. Dr. Carlos Marcelo Pedroso por sua dedicação e apoio nessa jornada. Aos nossos amigos que estiveram ao nosso lado e ajudaram torna essa caminhada mais fácil.

RESUMO

A popularização dos *smartphones* levou a um aumento acentuado do tráfego de vídeo em redes móveis. É um desafio conectar esses usuários a internet da forma mais eficiente e atendê-los de forma satisfatória, principalmente em tráfegos *Real-time,* como vídeo *streaming* e voz sobre IP que demandam requisitos de qualidade, como atraso de pacotes e *jitter* menores e taxas de transmissão mais elevadas que a maioria dos fluxos comuns.

Um dos elementos fundamentais para garantir a qualidade de serviço em sistemas LTE é o método utilizado para distribuição dos recursos disponíveis na célula coberta por uma estação rádio base. No ambiente de vídeo é necessário um *throughput* mínimo, bem como manter o atraso e o *jitter* menores que um certo limite para obter uma qualidade de experiência satisfatória.

O presente trabalho apresenta um novo algoritmo de escalonamento de recursos LTE visando aumentar o número de usuários satisfeitos com aplicações de vídeo *streaming*. O novo algoritmo de escalonamento leva em consideração o indicador de qualidade do canal do LTE para alocação de recursos de forma a atender o maior número possível de usuários de vídeo *streaming* com qualidade.

Os resultados obtidos com o algoritmo desenvolvido foram comparados com os dos escalonadores mais utilizados atualmente através de simulações computacionais e demonstraram um aumento de usuários satisfeitos, para aplicação de vídeo, considerando requisitos mínimos de atraso, *jitter* e *throughput.*

Palavras chave: Escalonador, LTE, vídeo

ABSTRACT

The popularization of smartphones has seen a marked increase in video traffic on mobile networks. It is a challenge to be able to connect these users to the Internet in the most efficient and satisfactory manner, especially in real-time traffic, such as video streaming and voice over IP that require quality requirements such as lower delay and jitter, and higher transmission rates than a majority of common flows.

One of the key elements for ensuring service quality in LTE systems is the method used to distribute the resources available in the cell covered by a base station. In the video environment, a minimum throughput is required, as well as keeping the delay and jitter less than a certain threshold to obtain a satisfactory quality of experience.

This work presents a new algorithm for scheduling LTE resources aiming to increase the number of satisfied users with video streaming applications. The new scheduling algorithm takes the LTE channel quality indicator to resource allocation so the largest possible number of users of streaming video quality.

The results obtained with the developed algorithm were compared with the most used schedulers in computational simulations and demonstrated an increase of satisfied users for video application, considering some minimum requirements of delay, jitter and throughput.

Keywords: scheduler, LTE, video

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

3GPP	3rd *Generation Partnership Project*
AMC	*Adaptive Modulation and Coding*
TBFQ	*Adaptive Token Bucket*
ARQ	*Automatic Repeat Request*
BET	*Blind Equal Throughput*
CQA	*Channel and QoS-aware*
CQI	*Channel Quality Indicator*
CP	*Cyclic Prefix*
FD	Domínio da Frequência
TD	Domínio do Tempo
EPS	*Envolved Packet System*
ERB	*Estação Radio Base*
eNodeB	*Evolved NodeB*
EPC	*Evolved Packet Core*
E-UTRAN	*Evolved UMTS Terrestrial Radio Access Network*
GNU	*General Public License*
GSM	*Global System for Mobile Communications*
GBR	*Guarantee Bit Rate*
HOL	*Head Of Line*
HD	*High* D*efinition*
HARQ	*Hybrid Automatic Repeat Request*
LTE	*Long Term Evolution Standard*
MT	*Maximum Throughput*
MAC	*Medium Access Control*
MME	*Mobility Management Entity*
MCS	*Modulation and Coding Scheme*
MIMO	*Multiple-input Multiple-output*
MRC	Maximum-ratio combining
NS-3	*Network Simulator 3*
OFDMA	*Orthogonal Frequency Division Multiple Access*
OFDM	*Orthogonal Frequency Division Multiplexing*

PDN	*Packet Data Network*
PGW	*Packet Data Network Gateway*
PLMN	*Public Land Mobile Network*
PSNR	*Peak Signal-to-Noise Ratio*
PC	*Personal Computer*
PDCCH	*Physical Downlink Control Channel*
PHY	*Physical Layer*
PSS	*Priority set scheduling*
PF	*Proportional Fair*
QAM	*Quadrature Amplitude Modulation*
QPSK	*Quadrature Phase Shift Keying*
QoS	*Quality of Service*
RLC	*Radio Link Control*
RRC	*Radio Resource Control*
RRM	*Radio Resource Management*
RB	*Resource Block*
RBG	*Resource Block Group*
RR	*Round Robin*
SGW	*Serving Gateway*
SINR	*Signal to Noise Ratio*
SC-FDMA	*Single-Carrier Frequency Division Multiple Access*
SISO	*Single-input Single-output*
SSIM	*Structural Similarity*
TBR	*Target Bit Rate*
TTA	*Throughput to Average*
TCP	*Transmission Control Protocol*
TTI	*Transmission Time Interval*
TB	*Transport Block*
UHD	*Ultra-high* Definition
UMTS	*Universal Mobile Telecommunication System*
UDP	*User Datagram Protocol*
UE	*User Equipment*
VoD	*Video-On-Demand*
VOIP	*VOIP Voice over IP*

SUMÁRIO

1 INTRODUÇÃO

1.1 JUSTIFICATIVA

O estudo *Visual Networking Index* [1] estima que até 2020, de todo tráfego IP, 82% será de vídeos e se, por exemplo, uma pessoa resolvesse capturar um mês desse tráfego e assistisse aos vídeos transferidos, levaria 5 milhões de anos para assistir tudo. O mesmo estudo aponta que o tráfego IP utilizado por *smartphones* (8% em 2015) irá superar o utilizado por computadores pessoais (53% em 2015) até o ano de 2020, no qual prevê-se 30% para *smartphones* e 29% para computadores pessoais, segundo o mesmo estudo, a demanda de diversos serviços de *streaming* de vídeo como *Netflix, Skype, Youtube* tem crescido nos últimos anos. O uso de internet para TV apresenta crescimento acelerado, conforme ilustrado pela TABELA 1 que demonstra o consumo mensal esperado em *petabytes* para os anos de 2015 a 2020.

	2015	2016	2017	2018	2019	2020
Por tipo de rede (Petabytes por mês)						
Fixa	27,011	34,978	45,134	57,656	73,413	90,239
Móvel	1,756	3,138	5,378	8,607	13,295	19,668
Por categoria (Petabytes por mês)						
Video	22,344	29,046	38,297	50,596	67,423	86,704
Videos da internet para TV	6,424	9,070	12,215	15,667	19,284	23,203

TABELA 1: PREVISÃO DE CONSUMO GLOBAL DE VÍDEO
FONTE: CISCO (2016) [1]

A estimativa também prevê que até 2020, o tráfego de *video-on-demand* (VoD) será 20,7% em *ultra-high* definition (UHD) e 53% em *high definition* (HD).

Conforme dados da Agência Nacional de Telecomunicações [2] o Brasil fechou o mês de julho de 2016 com o crescimento anual de 193% em acessos utilizando LTE, partindo de 14,6 milhões em julho de 2015 para 42,9 milhões em julho de 2016. A cobertura do sinal está presente em 560 cidades, que concentram mais da metade (58%) da população nacional.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Desenvolver um escalonador de recursos que melhore a quantidade de usuários satisfeitos em transmissões de vídeo no *downlink* de sistemas LTE.

1.2.2 Objetivos específicos

a. revisão do estado da arte dos algoritmos de escalonamento existentes;
b. realizar uma análise comparativa entre os principais escalonadores do LTE existentes com o que será desenvolvido, na aplicação de vídeo *streaming*;
c. realização de simulações para comparar o desempenho da solução proposta utilizando o *Network Simulator* 3 (NS-3).

1.3 NOVO ESCALONADOR LTE

O papel do escalonador é determinar como será realizada a alocação de recursos disponíveis para um conjunto de UEs (*User Equipaments*).

O 3rd *Generation Partnership Project* (3GPP) não especifica a política de alocação de recursos ou o escalonador que deve ser utilizado, deixando a critério dos desenvolvedores de equipamentos e fabricantes a escolha do algoritmo a ser implementado [3].

Atualmente, existem diversos escalonadores com variados objetivos e estratégias para distribuição de recursos, dentre eles, se destacam: *Round Robin*, *Proportional Fair* e *Maximum Throughput*. O primeiro distribui recursos de forma rotativa, sem se preocupar com qualidade de canal, *Quality of Service* (QoS) ou priorizar algum usuário. O segundo busca uma distribuição de forma justa, alocando menos recursos para quem possuir um *Channel Quality Indicator* (CQI) melhor (e provavelmente está mais próximo a antena de transmissão) e mais recursos para quem está com um CQI pior (possivelmente mais longe), de forma que todos tenham o *throughput* mais igualitário possível. O *Maximum Throughput* busca obter a maior taxa de transmissão na célula, alocando todos os recursos para o usuário com melhor CQI.

Existem outros escalonadores que levam em conta a qualidade do canal, qualidade de serviço e outros critérios, além dos três já citados. A seção 2.3 deste trabalho aborda-os de forma mais detalhada.

A estratégia adotada pelo novo escalonador é distribuir recursos para os UEs por ordem de melhor CQI, até atingir uma taxa limite definida pelo operador do sistema, visando atender o maior número possível de usuários de vídeo *streaming*. O capitulo 3 detalha a proposta do novo escalonador. Esta estratégia é inédita pois não existem algoritmos de escalonamento projetados com este objetivo para o LTE.

A implementação no simulador seguiu as normas do *Small Cell Forum* [4], o que permite que o código desenvolvido no simulador possa ser utilizado em equipamentos físicos sem grandes modificações.

1.4 METODOLOGIA

Para validação do novo escalonador, foi utilizado o simulador de redes NS-3, expandido com o módulo do projeto LENA [5] para simulações com LTE e também com o módulo do EVALVID [6] para transmissões de vídeo. O escalonador foi desenvolvido na linguagem de programação C++ como um módulo do NS-3.

Foram simulados variados números de usuários, com variadas distâncias até a estação rádio base e com os recursos escalonados de diversas formas, a seção 4 apresenta um comparativo entre o escalonador desenvolvido e os mais populares no LTE.

2 REDES LTE

O LTE começou a ser desenvolvido em 2004 pelo 3GPP. As principais vantagens dessa tecnologia são as altas taxas de transmissão de dados, baixa latência e melhor eficiência espectral em relação aos padrões anteriores [7].

Para aumentar a eficiência espectral, o LTE utiliza para *downlink* o *Orthogonal Frequency Division Multiplexing* (OFDM), cuja ideia chave é dividir a largura de banda total em subportadoras ortogonais espaçadas em 15khz, o que permite fornecer recursos para múltiplos usuários em um mesmo *slot* de tempo, além de uma maior imunidade a interferência inter simbólica (IIS) e desvanecimento seletivo em frequência [8].

2.1 TOPOLOGIA DA REDE

Um *User Equipment* (UE) pode ser qualquer dispositivo que o usuário final utilize para se conectar à rede LTE através do eNodeB. *Smartphones* e computadores com modem LTE são alguns exemplos de UE. Esses equipamentos do usuário são classificados em categorias ou classes que definem suas especificações de desempenho e permitem que as EnodeBs sejam capazes de se comunicar efetivamente com eles, sabendo seus níveis de desempenho. A TABELA 2 lista algumas características das categorias mais utilizadas.

Categoria	Taxa máxima de *downlink* (Mbit/s)	MIMO máximo	Taxa máxima de *uplink* (Mbit/s)	3GPP *Release*
0	1.0	1	1.0	Rel 12
1	10.3	1	5.2	Rel 8
2	51.0	2	25.5	Rel 8
3	102.0	2	51.0	Rel 8
4	150.8	2	51.0	Rel 8
5	299.6	4	75.4	Rel 8
6	301.5	2 ou 4	51.0	Rel 10
7	301.5	2 ou 4	102.0	Rel 10
8	2998.6	8	1497.8	Rel 10
9	452.2	2 ou 4	51.0	Rel 11
10	452.2	2 ou 4	102.0	Rel 11

TABELA 2: CATEGORIAS DE UES.
FONTE: 3GPP (2016) [14]

As taxas máximas de *downlink* e *uplink* apresentadas na TABELA 2 são para 20 MHz de largura de banda de canal. As categorias 6 e acima podem combinar vários canais de 20 MHz. A taxa máxima será menor se for utilizada menos largura de banda. Na Categoria 8, a taxa de dados de ~3,0 Gbps de *downlink* e ~1,5 Gbps de *uplink* especificada é a taxa de pico para um setor de estação base. Uma taxa de dados máxima mais realista para um único usuário é 1.2 Gbps (*downlink*) e 600 Mbps (*uplink*) [9]. A NOKIA [10] demonstrou taxas de *downlink* de 1,4 Gbps utilizando 100 MHz de espectro agregado. A categoria 0 foi pensada para dispositivos com baixo poder de processamento, como a internet das coisas.

A E-UTRAN *Node* B, também conhecida como *Evolved Node* B ou eNodeB é semelhante às estações rádio base (ERBs) do sistema *Global System for Mobile Communications (*GSM), que no LTE executa diversas funções, incluindo gerenciamento de recursos de rádio, controle de admissão, **escalonamento de recursos**, *broadcast* de informações da célula, criptografia, compressão e descompressão de cabeçalhos de pacotes. Além de fazer a interface com os UEs, a EnodeB hospeda as camadas *PHYsical* (PHY), *Radio Link Control* (RLC), *Medium Access Control* (MAC), *Packet Data Control Protocol* (PDCP) e a funcionalidade de *Radio Resource Control* (RRC).

A *Mobility Management Entity* (MME) é responsável pelo procedimento de rastreamento, controle da rede de acesso e paginação, incluindo retransmissões dos UEs. Gera identidades temporárias e as aloca aos UEs. Autentica os UEs. Entre as suas funções está também a autorização UE para a Rede Pública Terrestre Móvel, *Public Land Mobile Network* (PLMN) e a imposição das restrições de *roaming* do UE se houver.

O *Serving Gateway* (SGW) é responsável pela transferência do UE para o eNodeB vizinho caso o usuário esteja em movimento (*handover*), junto com a transferência dos dados que estavam sendo transmitidos. As suas funções se estendem também para a transferência do usuário para outras redes como 2G e 3G. O SGW monitora e mantém informações de contexto relacionadas com o UE durante

o seu estado inativo. A SGW também é responsável pelo monitoramento do tráfego do usuário em caso de exigências judiciais.

O *Packet Data Network Gateway* (PGW), proporciona conectividade entre os UEs e redes de dados externas, como, por exemplo, a Internet. O PGW executa aplicação de políticas, filtragem de pacotes para cada usuário, interceptação legal e triagem de dados.

A FIGURA 1 fornece um exemplo de interação entre UEs, EnodeBs, MME, SGW, PGW e rede IP.

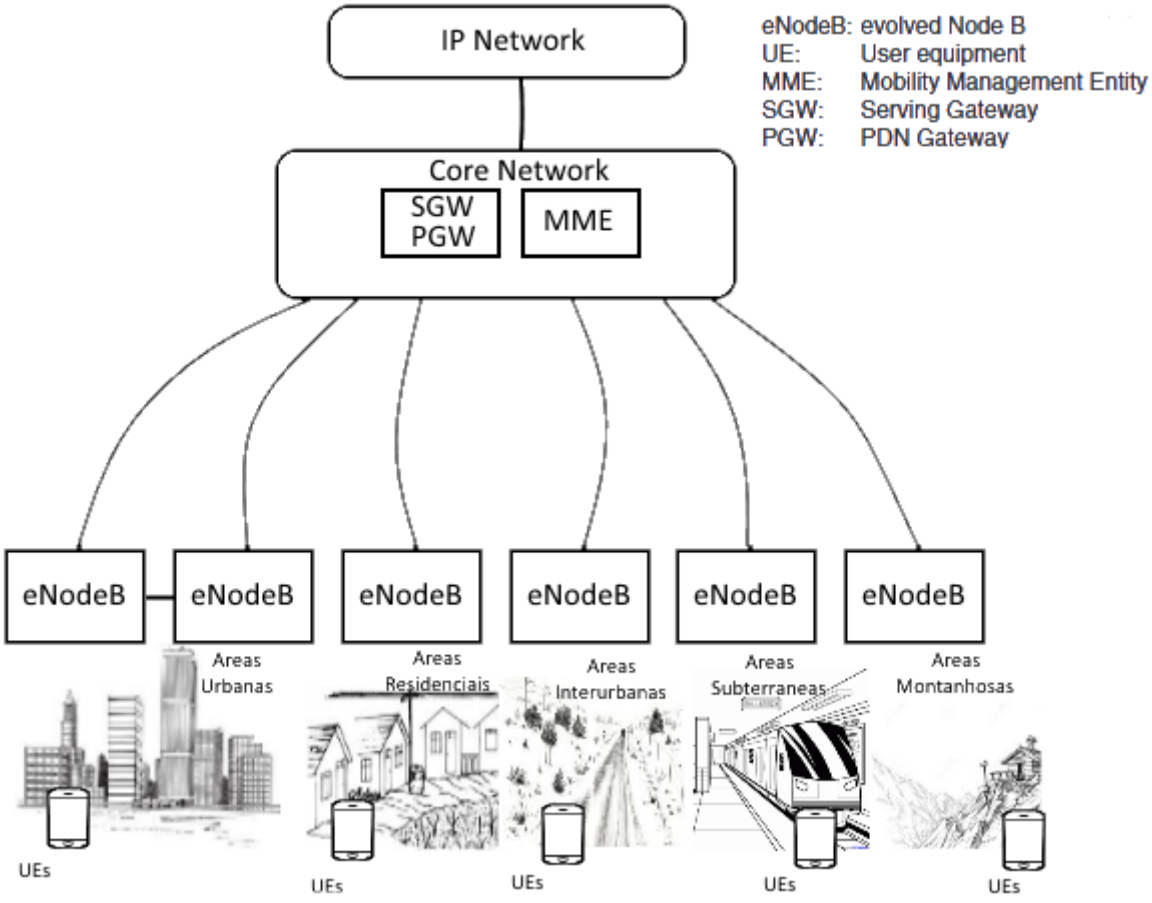

FIGURA 1: EXEMPLO DE INTERAÇÃO ENTRE UES, ENODEBS, MME, SGW, PGW E REDE IP
FONTE: WATANABE e MACHIDA (2012) [11]

O LTE pode opcionalmente utilizar múltiplos transmissores e múltiplos receptores (MIMO) tanto na eNodeB quanto no UE, de modo a aumentar a robustez da conexão e aumentar as taxas de dados. Em particular, a *Maximum-ratio combining* (MRC), combinação de proporção máxima, é utilizada para aumentar a confiabilidade da conexão em condições de propagação ruim, quando a intensidade

do sinal é baixa e as condições de multi-percurso são desafiadoras. Devido a limitações de espaço físico e energia, os UEs geralmente possuem apenas uma única antena LTE, restringindo essa técnica.

2.2 ALOCAÇÃO DE RECURSOS (RB, TTI, CQI)

Os *Resource Blocks* (RBs) são agrupamentos de recursos de 180khz (12 subportadoras OFDM) e 7 símbolos OFDM a cada 0,5ms. Um *Transmission Time Interval* (TTI) geralmente tem a duração de 1ms e engloba temporalmente 2 RBs. A TABELA 3 relaciona a largura de banda do canal com a quantidade de RBs e subportadoras OFDM.

Largura de banda do canal (MHz)	1,4	3	5	10	15	20
Número de *resourse blocks* (RBs)	6	15	25	50	75	100
Número de subportadoras ocupadas	72	180	300	600	900	1200

TABELA 3: RELAÇÃO ENTRE BANDA DO CANAL, NÚMERO DE RBS E DE SUBPORTADORAS OCUPADAS.
FONTE: 3GPP (2016) [12]

RBs adjacentes são agrupados em *Resource Block Groups* (RBGs), respeitando a TABELA 4, em um único intervalo de tempo. Em cada TTI, o escalonador na eNodeB é responsável por alocar os RBGs para um ou mais UEs, de acordo com seu algoritmo. A FIGURA 2 ilustra esse agrupamento.

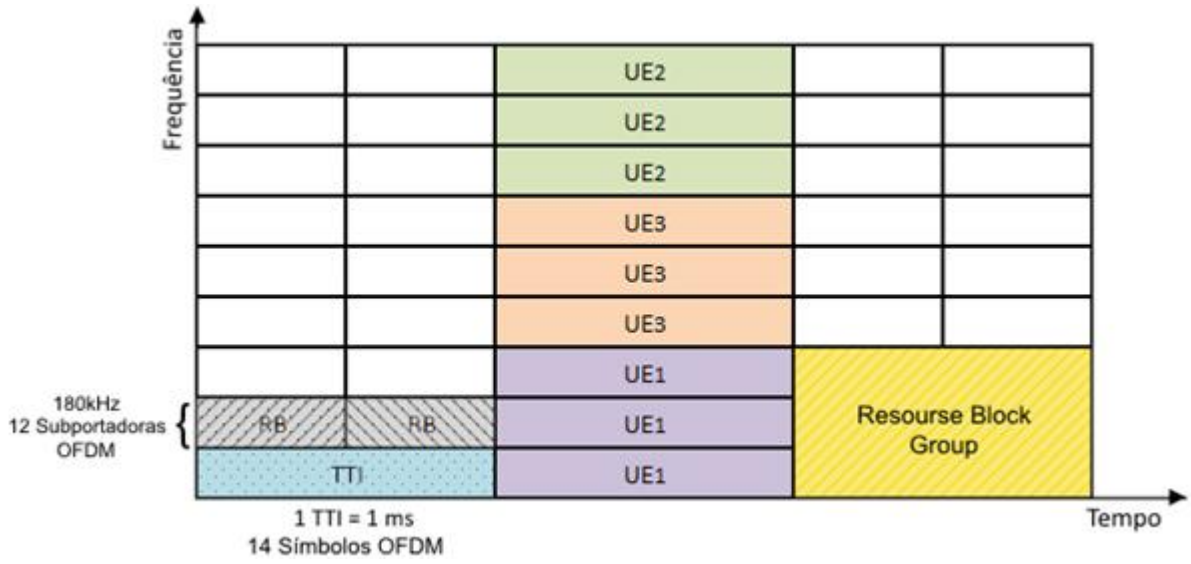

FIGURA 2: DIAGRAMA DE TEMPO VS FREQUÊNCIA, COM 3 RBS POR RBG (10MHZ)
FONTE: Grondalen et al. (2016) [13]

O *Resource Allocation Type* especifica a forma como serão agrupados vários RBs em RBGs de acordo com a largura de banda utilizada, o NS-3 segue as políticas da TABELA 4, por exemplo, com largura de banda disponível de 10Mhz, haverá 50 RBs disponíveis para alocação (TABELA 3), reunidos em grupos de 3 (TABELA 4), formando assim, 16 RBGs disponíveis para alocação pelo escalonador. Nota-se que por a divisão 50/3 não ser inteira, 2 RBs não são alocados [14].

Largura de banda do sistema (MHz)	Tamanho do RBG (em RBs)
1.4	1
3	2
5	2
10	3
15	4
20	4

TABELA 4: POLÍTICA DE AGRUPAMENTO DE RBS EM FUNÇÃO DA LARGURA DE BANDA. FONTE: 3GPP (2016) [12]

Alguns escalonadores podem utilizar a informação de CQI, estimada através do *Radio resource management* (RRM), para decidir como será feita a alocação de recursos, existem duas estimativas disponíveis: a *wideband* CQI que engloba uma avaliação da qualidade do canal para toda a banda utilizada, geralmente utilizada em escalonamentos no domínio do tempo e a *subband* CQI que é avaliada para cada RB, normalmente utilizada no domínio da frequência. No novo escalonador desenvolvido fez-se uso desta última alternativa.

A partir do CQI é possível, também, estimar a taxa máxima que cada UE pode atingir naquele TTI, pois é a partir do CQI que a eNodeB decide o *modulation and coding scheme* (MCS), ou seja, a codificação que será utilizada para transmissão de dados para determinado usuário, podendo ser *Quadrature Phase Shift Keying* (QPSK), 16 *Quadrature Amplitude Modulation* (16 QAM) ou 64 QAM em vários níveis, conforme a TABELA 5, então, a partir do MCS escolhido define-se o tamanho do *transport block* (TB), que armazena a carga útil de dados, e assim a quantidade de bits que será transportado no TTI para cada UE [8] [15].

Code rate é a razão entre o total de bit úteis pelo total de bits transmitidos (são acrescentados bits de correção de erro), no LTE é dado pela equação:

Code Rate = (*Transport Block Size* + CRC *Bits*)/(Número de Resourse Elements x *Bits* por símbolo)

CQI	Esquema de modulação	Coding rate	SINR(dB)	Eficiência espectral (bps/Hz)
0	-	-	-	-
1	QPSK	0.076	< -7.27	0.1523
2	QPSK	0.12	< -4.76	0.2344
3	QPSK	0.19	< -2.06	0.377
4	QPSK	0.3	< 0.61	0.6016
5	QPSK	0.44	< 2.81	0.877
6	QPSK	0.59	< 4.69	1.1758
7	16QAM	0.37	< 6.29	1.4766
8	16QAM	0.48	< 8.69	1.9141
9	16QAM	0.6	< 11.37	2.4063
10	64QAM	0.45	< 13.11	2.7305
11	64QAM	0.55	< 16.44	3.3223
12	64QAM	0.65	< 19.62	3.9023
13	64QAM	0.75	< 23.01	4.5234
14	64QAM	0.85	< 26.19	5.1152
15	64QAM	0.93	< 28.66	5.5547

TABELA 5: NÍVEIS DE CQI DEFINIDOS PELO 3GPP.
FONTE: BARAYAN e KOSTANIC (2013) [8]

Exemplo de cálculo de *throughput* máximo para largura de banda de 10MHz:

Para uma largura de banda de 10MHz e considerando *Cyclic Prefix* (CP) Normal, são utilizadas 600 subportadoras (TABELA 3). Nas melhores condições de canal, será utilizada a modulação 64QAM, permitindo que cada símbolo carregue 6 bits. O total de bits transportados por 600 subportadoras na duração de um símbolo LTE é $600 * 6 = 3600\ bits$, como a duração de um símbolo LTE é de 71,4 microssegundos, então a taxa de dados nessas condições é de $3600/71{,}4 = 50{,}4Mbps$.

A TABELA 6 sintetiza os resultados para diversas codificações e larguras de banda.

Largura de banda do canal	1.4 MHz	3 MHz	5 MHz	10 MHz	15 MHz	20 MHz
Resource Blocks no domínio da frequência	6	15	25	50	75	100
Símbolos OFDMA por milissegundo	14					
Modulation symbol rate (Msps)	1.0	2.5	4.2	8.4	12.6	16.8
Taxa QPSK (Mbps)	2.0	5.0	8.4	16.8	25.2	33.6
Taxa 16QAM (Mbps)	4.0	10.1	16.8	33.6	50.4	67.2
Taxa 64QAM (Mbps)	6.1	15.1	25.2	50.4	75.6	100.8

TABELA 6: CÁLCULO DA MÁXIMA TAXA ATINGÍVEL, CONSIDERANDO CYCLIC PREFIX (CP) NORMAL E SEM CONSIDERAR OVERHEAD.
FONTE: 3GPP (2016) [12]

Estima-se que aproximadamente 25% da taxa é usada pra *Physical Downlink Control Channel* (PDCCH), sinal de referência, sinais de sincronismo, *Physical Broadcast Channel* (PBCH) entre outras coisas, sendo então aproximadamente 75% da taxa da TABELA 6 estará disponível para escalonamento entre as UEs.

As taxas podem alcançar valores superiores se utilizado MIMO, que é uma técnica que utiliza várias antenas transmissoras e receptoras.

2.3 PRINCIPAIS ESCALONADORES

2.3.1 *Round Robin* (RR)

É um dos escalonadores mais simples, cujo algoritmo aloca os RGBs de maneira igual e rotativa entre todas as UEs independentemente da qualidade do canal ou requisitos de QoS. O objetivo é ser justo com todas as UEs, mas por não considerar o CQI, fornece um menor *throughput* geral [16].

2.3.2 Proportional Fair

O algoritmo de escalonamento *Proportional Fair* (PF) tenta encontrar um equilíbrio entre justiça e eficiência espectral, o RB atual é alocado para o UE que obtiver a maior razão entre a vazão atingível naquele instante e a última média da vazão atingida.

Esse equilíbrio é fornecido utilizando o *throughput* médio passado como um fator de ponderação da taxa de dados esperada, o que resulta em servir os usuários em condições de canal ruim dentro de um certo período de tempo [14].

2.3.3 Maximum Throughput

Maximum Throughput (MT): MT é um escalonador que utiliza CQI, mas não utiliza QoS para tomada de decisões. No MT, o eNodeB atribui o RBG à UE com a maior taxa de dados suportável instantânea calculada pelo *subband* CQI no domínio da frequência, visando maximizar o *throughput* do EnodeB.

Esta estratégia pode deixar alguns usuários com condições de canal ruim sem nenhum ou com poucos recursos, pois faz um compartilhamento de recursos “injusto” [17].

2.3.4 Blind Equal Throughput

Blind Equal Throughput (BET): BET é um escalonador que não utiliza CQI ou QoS para suas decisões. Ele designa a mesma taxa de transferência para todos os usuários, independentemente da qualidade do seu canal de rádio. A prioridade é o recíproco do *throughput* médio passado.

$$m_{i,k}^{BET} = 1/\overline{R^i}(t-1)$$

Sendo $\overline{R^i}(t) = \beta\overline{R^i}(t-1) + (1-\beta)r^i(t)$

i é i-ésimo User Equipment

k é k-ésimo Resourse Block

$\overline{R^i}(t)$ é taxa média passada;

$r^i(t)$ é a taxa arquivada no tempo t;

β é coeficiente de alisamento exponencial .

Conforme a equação, o BET vai alocando recursos para usuários com a menor taxa de transferência média. Portanto, para os usuários com uma condição de canal ruim são alocados mais recursos do que outros, o que, por sua vez, melhora a equidade, mas resulta em uma redução do rendimento total do sistema [17].

2.3.5 Throughput To Average

Throughput to Average (TTA): TTA é um escalonador que utiliza CQI mas não utiliza QoS para tomada de decisões.

A métrica de prioridade é igual a taxa esperada para o UE j no tempo t no k-ésimo RBG (estimativa utilizando *subband* CQI) dividido pela taxa esperada para o mesmo UE em toda a largura de banda, (*wideband* CQI). Por utilizar o cálculo de *subband* CQI, esse escalonador só pode ser utilizado domínio da frequência (FD) [17].

2.3.6 Adaptive Token Bucket

Adaptive Token Bucket (TBFQ): O TBFQ é um escalonador que utiliza tanto CQI como QoS para tomada de decisões. Esse escalonador deriva do mecanismo de “balde furado” o que garante justiça entre os UEs através da utilização de um banco de tokens compartilhados. Além disso, cada fluxo tem uma taxa de geração de tokens. A QoS é garantida definindo a taxa de geração de tokens em diferentes valores, como o *Guarantee Bit Rate* (GBR) no parâmetro QoS. No TBFQ, fluxos pertencentes a UEs que estão sofrendo interferência severa, e condições de sombreamento, terão maior prioridade [18] [19].

2.3.7 Priority Set Scheduling

Priority set scheduling (PSS): O PSS é um escalonador que combina operações de escalonamento no domínio do tempo e da frequência em uma só política e utiliza tanto CQI como QoS para tomada de decisões. Tem como objetivo fornecer uma taxa especifica para todos os UEs, o *Target Bit Rate* (TBR). O PSS funciona selecionando UEs cujo buffer RLC (*Radio Link Control*) não está vazio. Os UEs são então divididos em dois grupos de acordo com o seu TBR:

Grupo 1: usuários abaixo do TBR, são alocados recursos com um escalonador estilo BET;

Grupo 2: usuários acima do TBR, são alocados recursos com um escalonador estilo PF;

Os usuários pertencentes ao grupo 1 têm maior prioridade do que aqueles no grupo 2 [20] [21].

2.3.8 Channel And Qos-Aware

Channel and QoS-aware (CQA): O CQA é um escalonador que utiliza tanto CQI como QoS para tomada de decisões. Além disso leva em consideração o atraso *head of line* (HOL), ou seja, o atraso do primeiro pacote a ser transmitido pelo UE; De forma semelhante ao PSS, o CQA divide os UEs de acordo com a sua prioridade, considerando primeiro os fluxos com maior atraso HOL a qualidade do canal pode ser calculada usando dois métodos que são PF ou desvanecimento seletivo da frequência [20].

2.4 SIMULADOR NS-3

As simulações permitem o estudo e a avaliação de sistemas complexos. Os simuladores de rede, como o NS-3 desempenham um papel importante na tarefa de desenvolver, analisar e aperfeiçoar protocolos de comunicação. Segundo Guedes, Conceição, Carvalho e Rodrigues [22] destacam-se três principais vantagens do uso de simulação:

1) Permitem testar o comportamento dos protocolos utilizados em diversas redes e ambientes, cuja preparação num laboratório poderia ser impraticável, isso por questão de custos, tempo de instalação, ou até mesmo do ponto de vista administrativo;

2) Facilitam a execução de testes em um ambiente controlado, onde é mais fácil variar determinados parâmetros, mantendo os restantes constantes;

3) Facilitam a criação de múltiplos cenários de execução.

No presente trabalho foi utilizado o NS-3 [23], que é um simulador de redes baseado em eventos discretos, desenvolvido para pesquisa e uso educacional. É um software livre, licenciado sob a licença GNU GPLv2. Seu download, junto com mais informações e manual de uso estão disponíveis em https://www.nsnam.org/

2.4.1 FlowMonitor

O objetivo do módulo *flow monitor* do NS-3 é fornecer um sistema flexível para medir o desempenho dos fluxos da rede. Ele faz o monitoramento dos nós instalados na rede, para rastrear os pacotes trocados e medir valores médios de atraso, jitter, *throughput* e perda de pacotes. Os pacotes são divididos de acordo com o fluxo ao qual pertencem, no caso utilizado são classificados como IP:porta_de_origem e IP:porta_de_destino.

As estatísticas são coletadas para cada fluxo podem ser exportadas em formato XML ou CSV. No presente trabalho foram coletadas as estatísticas do tráfego de pacotes e gerados os gráficos da seção Resultados.

2.4.2 Recomendações para uma qualidade de experiência satisfatória

O relatório técnico TR-126 do DSL FORUM [24] faz recomendações de recursos mínimos para uma qualidade de experiência (QoE) de tráfego de vídeo satisfatória. A TABELA 7 apresenta esses valores.

O *delay* é o tempo necessário para que o vídeo percorrer a rede e ser reconstruído no receptor. É comumente chamado de latência. Quando o *delay* é muito prolongado, pode causar interrupções na exibição do vídeo.

A variação no atraso, causada por diferenças no tempo levado para pacotes para atravessar a rede, é chamado de *jitter*.

Perdas de pacotes podem ocorrer durante a transmissão ou descartados caso chegem no receptor com muito atraso. As informações ausentes degradam a qualidade do vídeo e podem causar uma qualidade de experiência (QoE) insatisfatória para o usuário.

Taxa em Mbps	Latência	Jitter	Duração máxima de um único erro	Perda correspondente em pacotes IP	Distância entre perdas	Taxa média de perda no fluxo de vídeo
1.75	<200 ms	<50 ms	<= 16 ms	4 pacotes IP	1 erro por hora	<= 6.68E-06
2.0	<200 ms	<50 ms	<= 16 ms	5 pacotes IP	1 erro por hora	<= 7.31E-06
2.5	<200 ms	<50 ms	<= 16 ms	5 pacotes IP	1 erro por hora	<= 5.85E-06
3.0	<200 ms	<50 ms	<= 16 ms	6 pacotes IP	1 erro por hora	<= 5.85E-06

TABELA 7: RECOMENDAÇÕES DE RECURSOS MÍNIMOS PARA UMA QUALIDADE DE EXPERIÊNCIA (QOE) SATISFATÓRIA PARA VÍDEOS CODIFICADOS COM CODEC H.264/AVC E FORMATO SD.
FONTE: DSL FORUM (2006) [24]

3 NOVO ESCALONADOR

3.1 CLASSIFICAÇÃO DO ESCALONADOR

O novo escalonador se enquadra na categoria *Channel-aware* por consultar o CQI para tomar as decisões na distribuição de recursos e *QoS-unaware* por não dar utilizar osistema de QoS do LTE para priorização de fluxos. Faz o escalonamento de recursos no domínio da frequência.

Para o desenvolvimento do escalonador foram seguidas as normas do Small Cell Forum [24], estabelecidas pelo conjunto das maiores operadoras de telecomunicações, fabricantes de equipamentos e fabricantes de circuitos eletrônicos do mundo, possibilitando futuramente implementações em equipamentos físicos com a mesma estrutura de código.

3.2 ESTRATÉGIA DE ESCALONAMENTO

Na estrutura do LTE, o escalonador é requisitado uma vez por TTI (por padrão 1 milissegundo) para definir como será o mapa de alocação de recursos.

Primeiramente são alocados recursos para *Hybrid automatic repeat request* (HARQ) - retransmissão de dados perdidos, então, no novo escalonador, os RBGs restantes são alocados conforme o fluxograma mostrado na FIGURA 3. Verifica-se os UEs que desejam transmitir, ordena-os por ordem de CQI e aloca os RBGs para o UE com maior CQI que tenha *throughput* médio abaixo de um certo limite. Caso o UE com maior CQI naquele TTI já atingiu a taxa limite, aloca-se então para o UE com segundo maior CQI, até que este UE atinja o limite e assim sucessivamente.

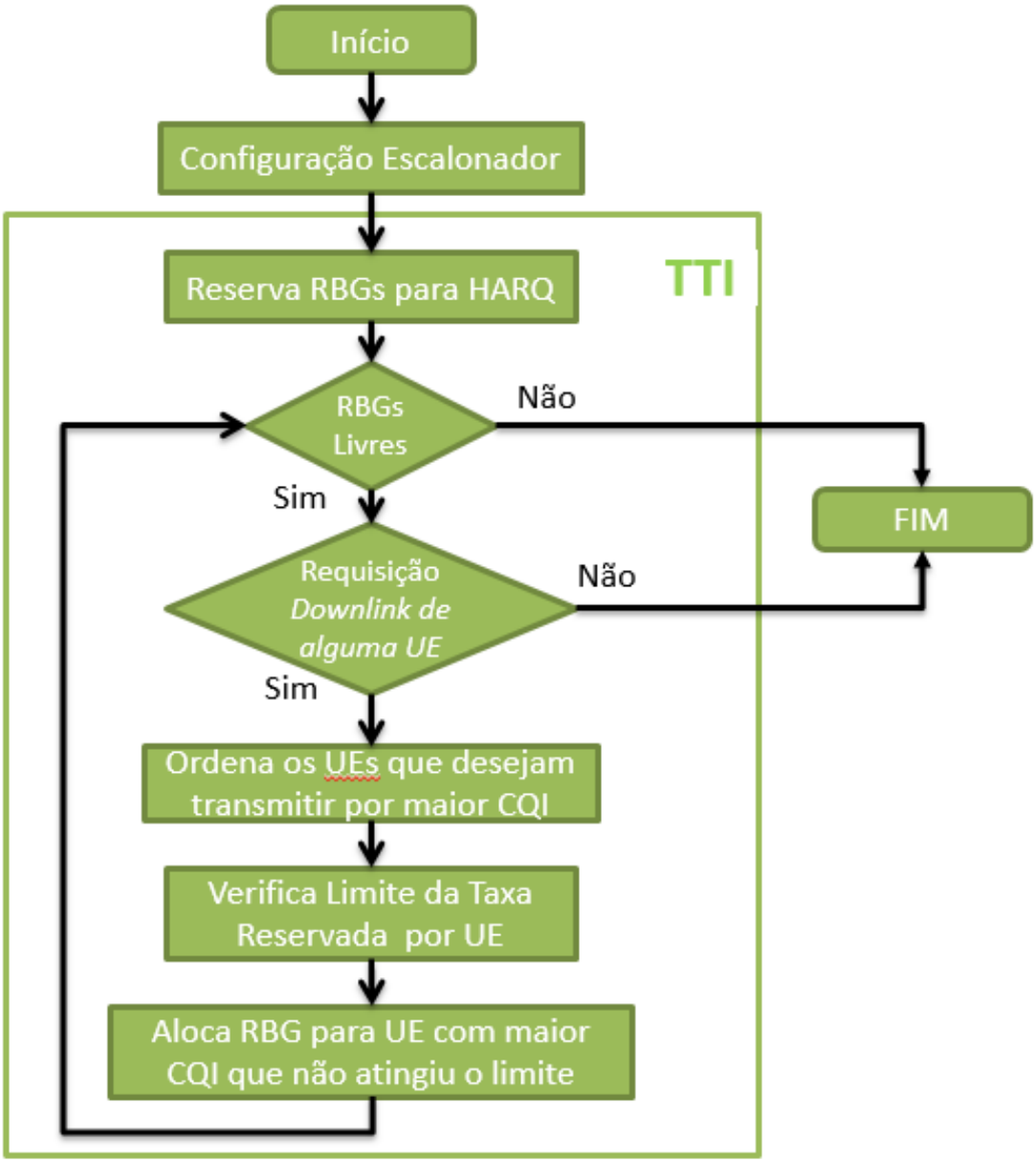

FIGURA 3: FLUXOGRAMA DO MÉTODO PROPOSTO, ACIONADO A CADA TTI.

O ordenamento por CQI tem o objetivo de dar prioridade aos usuários que utilizam uma codificação mais eficiente, melhorando assim o número de UEs atendidos satisfatoriamente, uma vez que um UE com CQI baixo necessita de uma maior quantidade de RBGs para ser atendido com o mesmo *throughput* de um UE com CQI alto.

O limite da taxa é definido previamente pelo operador do sistema e evita abusos de UEs, permitindo um maior número de usuários satisfeitos.

A FIGURA 4 compara a taxa (*bitrate*) necessário para atingir resoluções específicas nos principais serviços de video *streaming*.

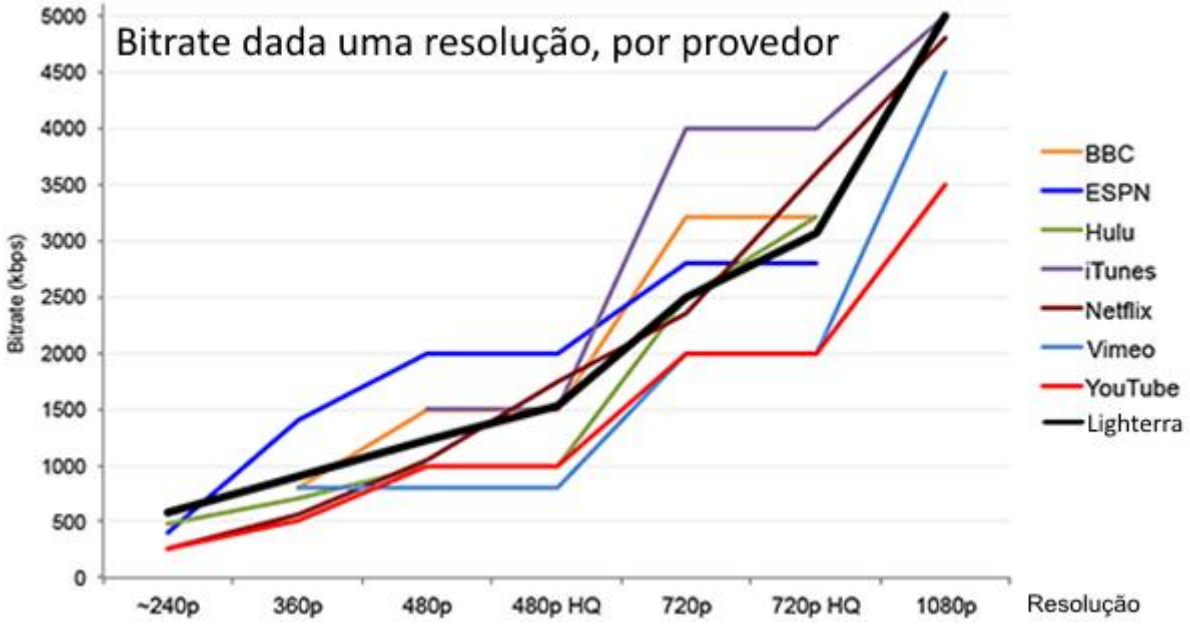

FIGURA 4: BITRATE EM RELAÇÃO A RESOLUÇÃO DO VÍDEO NOS PRINCIPAIS SERVIÇOS DE STREAMING.
FONTE: PATTERSON (2012) [25]

Para armazenar a taxa reservada para cada UE foi utilizada uma técnica de média móvel com alisamento exponencial simples, que possui uma complexidade computacional baixa pois é possível adicionar memória sem a necessidade de armazenar valores passados em vetores. Outra vantagem é permitir rajadas.

A cada TTI é calculada a taxa em uso para cada UE utilizando a média móvel com alisamento exponencial.

$$TR_S(TTI) = \alpha * TR(TTI) + (1 - \alpha) * TR_S(TTI - 1)$$

Onde: TR_S é a Taxa Reservada Suavizada;

TR é a Taxa Reservada;

α é o coeficiente de alisamento exponencial e deve estar entre 0 e 1

A partir de uma análise experimental, foi definido o valor de alfa em 0,2 ponderando assim 80% do valor passado e 20% do valor presente para a taxa reservada suavizada.

Quando a taxa reservada suavizada de um UE atinge o limite máximo definido ele deixa de receber recursos e a cada TTI sua taxa é reduzida, como o valor da taxa atual é zero, a média irá decair até estar abaixo do limite e o usuário se credenciar novamente para receber recursos.

Alguns algoritmos de escalonamento que permitem a reserva de recursos (QoS) procuram atender todos os elementos da célula que estão requisitando um fluxo priorizado, mesmo que eles possuam CQI baixo, o que diminui o *throughput* global da célula e faz com que o número de usuários satisfeitos seja menor caso aja um grande volume ou todo seja tráfego priorizado.

As principais alterações do novo escalonador em relação aos outros estão na função *DoSchedDlTriggerReq().* Essa função é chamada pelo método *SchedDlTriggerReq()* da classe *MemberSchedSapProvider* no seu atributo, que é uma instância do tipo de escalonador. O objetivo principal da função DoSchedDlTriggerReq() é fazer o escalonamento do *downlink*. Portanto, é chamada uma vez por TTI. O código da função *DoSchedDlTriggerReq()* do escalonador desenvolvido está no Anexo 2.

4 RESULTADOS

A FIGURA 5 ilustra a topologia adotada nas simulações realizadas. O servidor de vídeo tem a função de transmissão de um vídeo para os UEs, um servidor de tráfego de fundo realiza a transmissão de pacotes com intevalos exponencialmente distribuídos e uma taxa média de 5Mbps, o Roteador 1 encaminha pacotes de dados dos servidores para o PGW, que provê conectividade entre a rede de dados externa e os UEs, sendo o nó de saída e entrada do tráfego de dados dos usuários. O EnodeB está conectado à rede de telefonia móvel que se comunica diretamente com os equipamentos dos usuários que, para o experimento 1 são posicionados de forma fixa e para o experimento 2 são distribuídos espacialmente de forma uniforme, dentro de um raio pré-definido.

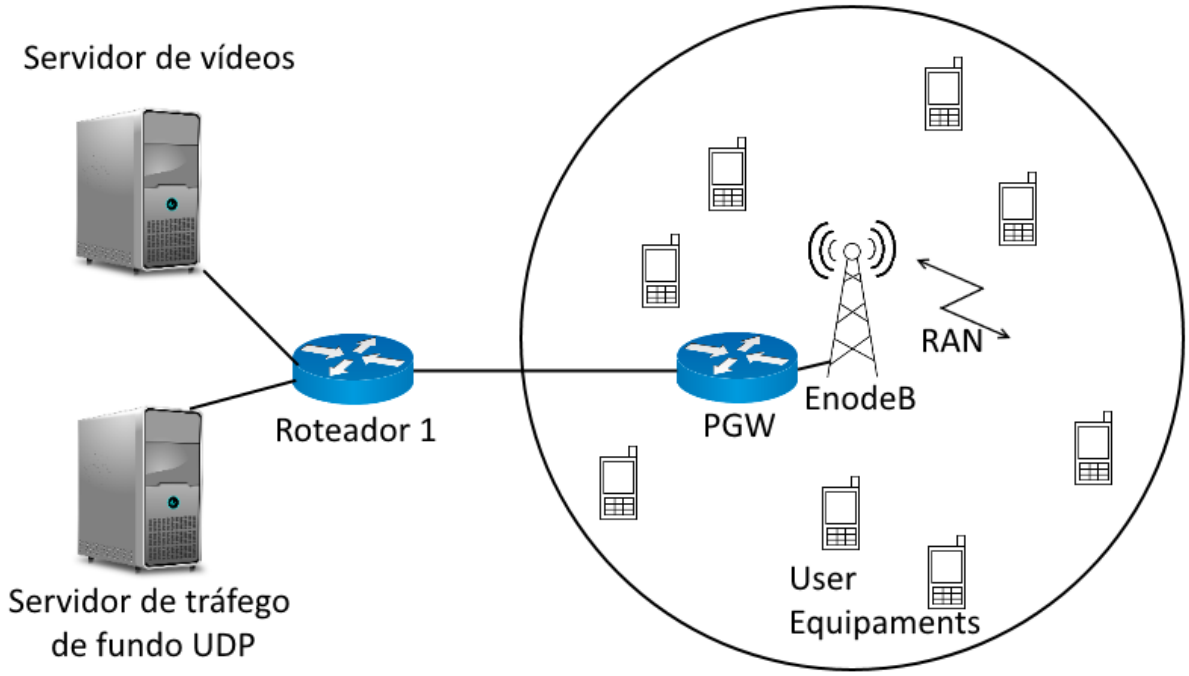

FIGURA 5: TOPOLOGIA UTILIZADA NAS SIMULAÇÕES DE VALIDAÇÃO.

Os parâmetros de simulação são sumarizados na TABELA 8. A largura de banda usada foi de 10 MHz. As potências da eNodeB e EU são as de uso típico na prática, o modelo de perda usado é o Friss, no modelo de perdas de Friis, a potência recebida decai proprorcionalmente ao quadrado da distância do transmissor, a uma taxa de 20dB por década em relação à distância, e considera que há visada direta entre o transmissor e receptor. Esse modelo não é o mais realista para ambientes urbanos, mas isso de certa forma não interfere nos resultados, pois o parâmetro para a tomada de decisões é o valor do CQI. Nesse modelo a distância está diretamente associada a variação de CQI, por esse motivo as simulações consideram variação entre distância entre UE e eNodeB, quantidade de usuários e o escalonador utilizado [26]. As distâncias entre eNodeB e EU são maiores que as observadas em ambientes urbanos.

Parâmetro	Valor
Largura de banda	10Mhz
Número de RBs	50
RBs por RBG	3
Downlink EARFCN	100
Frequência portadora de *downlink*	2120 Mhz
Uplink EARFCN	18100
Frequência portadora de *uplink*	1930 Mhz
Modelo AMC e cálculo de CQI	Piro (2011)
Modelo de perdas	Friis
Potência de transmissão do eNodeB	30 dBm
Potência de transmissão dos UEs	10 dBm
Configuração das antenas	SISO
Transmission Time Interval (TTI)	1ms

TABELA 8: PARÂMETROS DE SIMULAÇÃO

4.1 EXPERIMENTO 1: CENÁRIOS CRÍTICOS

Foi simulado um cenário crítico onde todas as UEs estavam à uma mesma distância da eNodeB, tendo assim a mesma qualidade no canal, gerando uma taxa de transmissão de 5 Mbps para cada UE. O limite foi configurado em 3 Mbps no escalonador desenvolvido. Foi considerado um usuário satisfeito aquele possuir os requisitos mínimos para receber um vídeo na qualidade 720p(HD), uma taxa mínima de 2,350 Mbps segundo a FIGURA 4, um atraso médio máximo de 200ms e *jitter* médio menor que 50 ms [24].

A FIGURA 6 representa o número de usuários satisfeitos com 4 UEs requisitando recursos primeiramente a 150m, depois a 3000m, 6000m e por último a 9000m de distância da eNodeB. As TABELAS 9 e 10 apresentam respectivamente as médias para o atraso e o *jitter*.

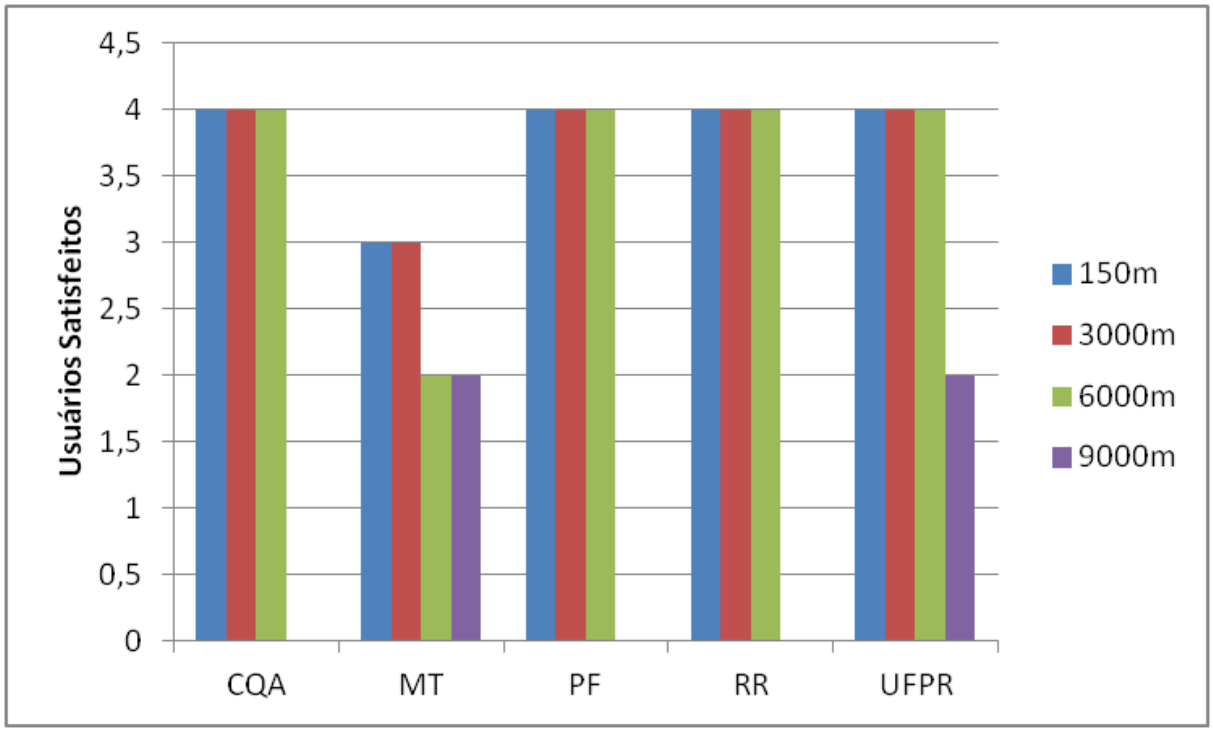

FIGURA 6: USUÁRIOS SATISFEITOS COM 4 UES

Distância	Cqa	FdMt	Pf	Rr	Ufpr
150m	14,26	38,08	10,52	9,79	25,70
3000m	18,40	98,71	12,24	15,10	26,30
6000m	30,34	123,30	23,03	26,53	28,85
9000m	41,37	496,37	34,85	38,65	38,22

TABELA 9: DELAY MÉDIO EM (MS) COM 4 UES

Distância	Cqa	FdMt	Pf	Rr	Ufpr
150m	2,22	7,87	1,27	0,74	2,57
3000m	2,99	20,02	1,58	1,76	2,72
6000m	4,83	23,04	2,72	2,61	2,90
9000m	5,16	49,55	2,93	3,47	4,00

TABELA 10: JITTER MÉDIO EM (MS) COM 4 UES

A FIGURA 7 representa o número de usuários satisfeitos com 6 UEs requisitando recursos primeiramente a 150m, depois a 3000m, 6000m e por último a 9000m de distância da eNodeB. As TABELAS 11 e 12 apresentam respectivamente as médias para o atraso e o *jitter*.

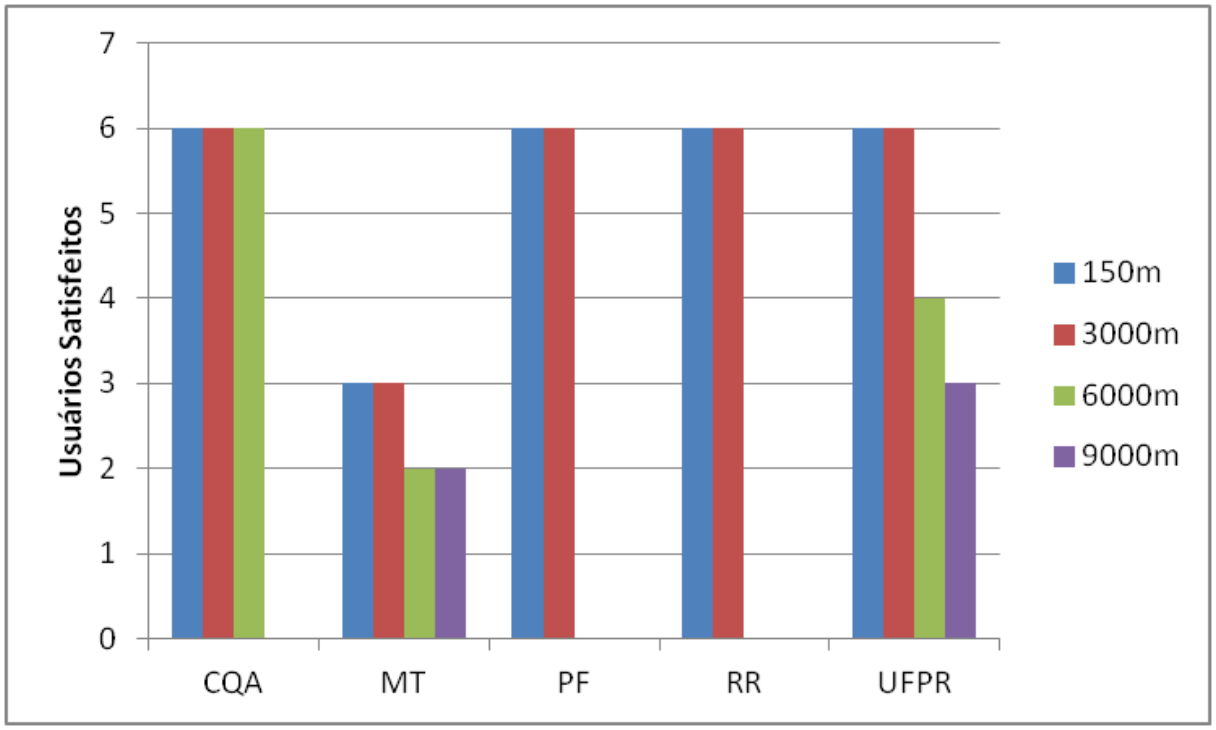

FIGURA 7: USUÁRIOS SATISFEITOS COM 6 UES

Distância	Cqa	FdMt	Pf	Rr	Ufpr
150m	19,14	1463,26	13,39	19,67	28,04
3000m	25,12	437,05	20,04	30,77	28,41
6000m	38,59	31,91	34,49	44,68	56,77
9000m	56,27	921,42	53,18	64,60	38,49

TABELA 11: DELAY MÉDIO EM (MS) COM 6 UES

Distância	Cqa	FdMt	Pf	Rr	Ufpr
150m	3,52	17,87	2,03	1,73	2,35
3000m	4,13	75,20	2,58	2,23	2,49
6000m	6,04	2,50	4,58	2,78	4,95
9000m	6,61	106,70	4,09	3,55	2,89

TABELA 12: JITTER MÉDIO EM (MS) COM 6 UES

A FIGURA 8 representa o número de usuários satisfeitos com 8 UEs requisitando recursos primeiramente a 150m, depois a 3000m, 6000m e por último a 9000m de distância da eNodeB. As TABELAS 13 e 14 apresentam respectivamente as médias para o atraso e o *jitter*.

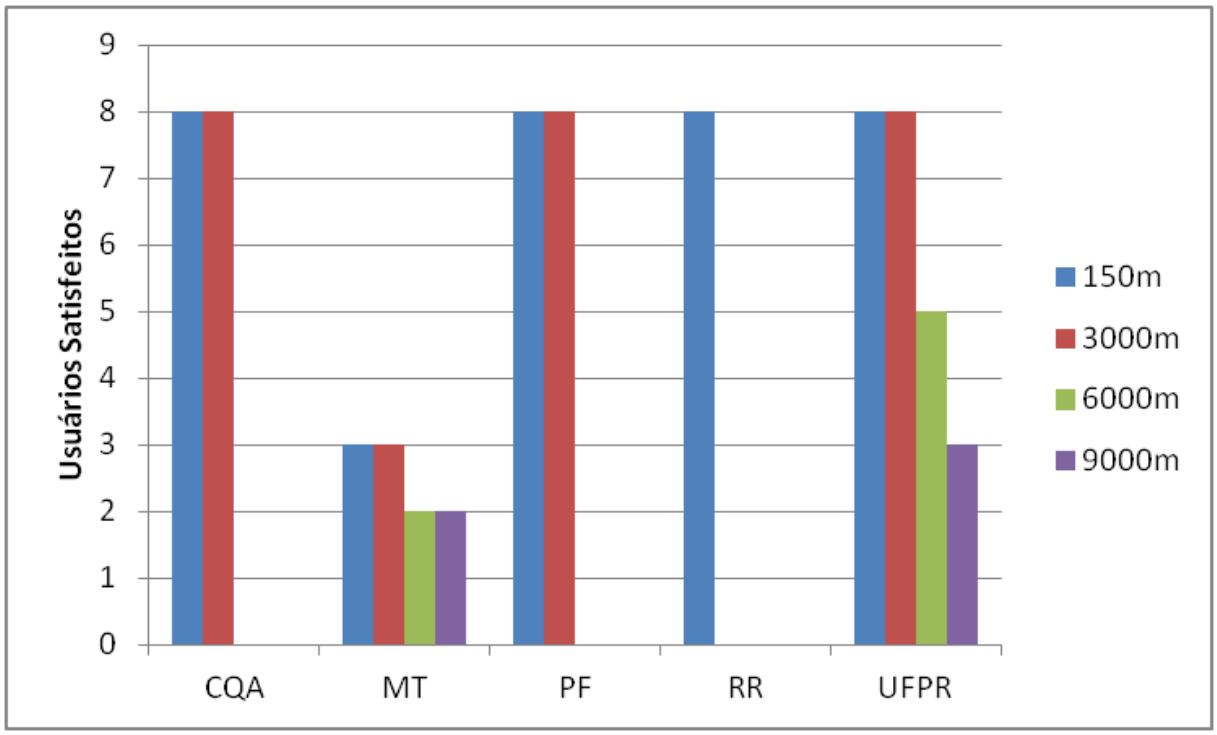

FIGURA 8: USUÁRIOS SATISFEITOS COM 8 UES

Distância	Cqa	FdMt	Pf	Rr	Ufpr
150m	23,40	1184,61	18,98	27,26	31,45
3000m	30,12	58,90	26,95	46,73	31,31
6000m	48,34	31,15	46,16	77,89	47,88
9000m	71,58	429,06	72,19	120,49	34,74

TABELA 13: DELAY MÉDIO EM (MS) COM 8 UES

Distância	Cqa	FdMt	Pf	Rr	Ufpr
150m	4,05	4,72	2,91	2,48	2,29
3000m	4,59	11,88	3,47	3,51	2,42
6000m	7,85	5,64	6,56	3,76	1,96
9000m	8,35	80,17	4,93	4,42	1,82

TABELA 14: JITTER MÉDIO EM (MS) COM 8 UES

A FIGURA 9 representa o número de usuários satisfeitos com 10 UEs requisitando recursos primeiramente a 150m, depois a 3000m, 6000m e por último a 9000m de distância da eNodeB. As TABELAS 15 e 16 apresentam respectivamente as médias para o atraso e o *jitter*.

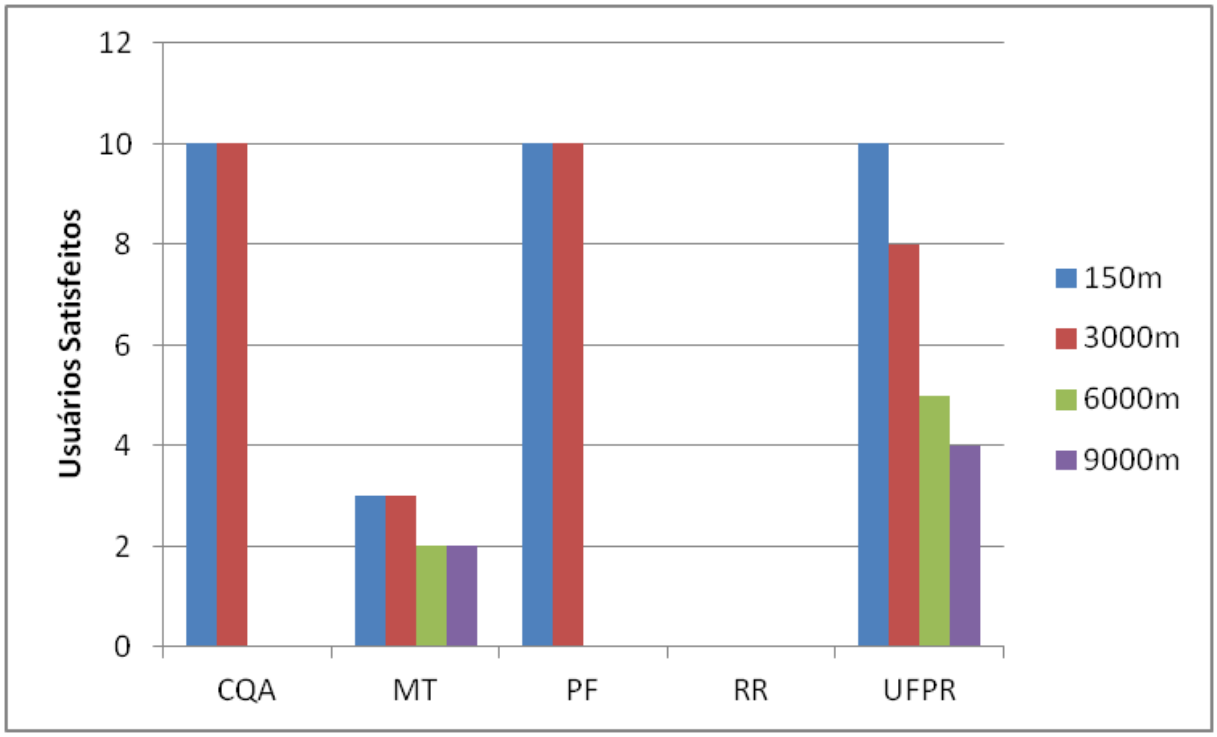

FIGURA 9: USUÁRIOS SATISFEITOS COM 10 UES

Distância	Cqa	FdMt	Pf	Rr	Ufpr
150m	27,33	27,80	24,24	38,99	30,91
3000m	35,26	64,09	33,57	59,31	66,52
6000m	58,56	1986,56	58,35	88,34	55,38
9000m	89,43	316,39	89,44	135,71	34,72

TABELA 15: DELAY MÉDIO EM (MS) COM 10 UES

Distancia	Cqa	FdMt	Pf	Rr	Ufpr
150m	4,57	5,38	3,86	2,26	2,16
3000m	5,32	13,30	4,48	2,64	7,12
6000m	9,67	5,38	8,55	3,35	4,01
9000m	9,35	68,19	5,42	3,97	1,81

TABELA 16: JITTER MÉDIO EM (MS) COM 10 UES

A FIGURA 10 representa o número de usuários satisfeitos com 12 UEs requisitando recursos primeiramente a 150m, depois a 3000m, 6000m e por último a 9000m de distância da eNodeB. As TABELAS 17 e 18 apresentam respectivamente as médias para o atraso e o *jitter*.

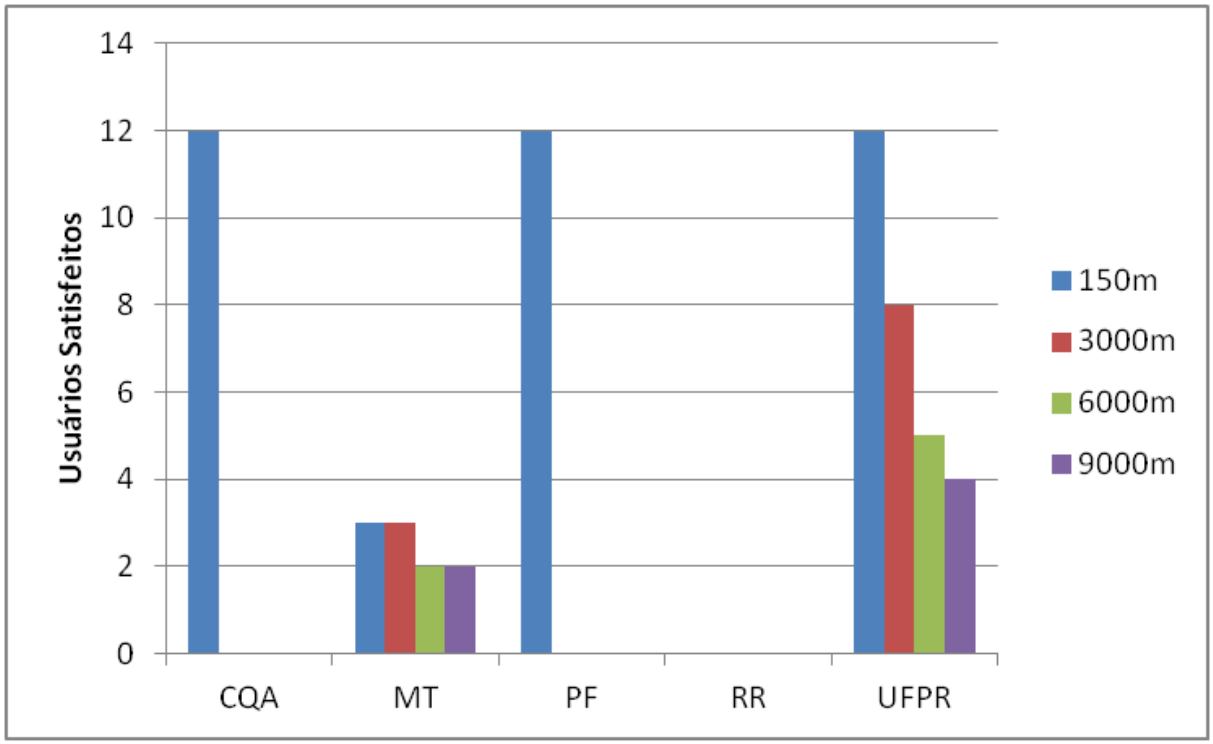

FIGURA 10: USUÁRIOS SATISFEITOS COM 12 UES

Distância	Cqa	FdMt	Pf	Rr	Ufpr
150m	31,32	1188,80	29,26	39,48	32,20
3000m	41,03	65,09	40,30	60,17	32,13
6000m	68,78	31,13	70,22	89,67	47,84
9000m	105,38	374,18	106,74	137,59	34,75

TABELA 17: DELAY MÉDIO EM (MS) COM 12 UES

Distância	Cqa	FdMt	Pf	Rr	Ufpr
150m	5,27	195,41	4,90	2,17	2,14
3000m	6,03	13,41	5,57	2,48	2,53
6000m	11,64	5,69	10,34	3,11	1,96
9000m	11,41	81,21	5,86	3,68	1,80

TABELA 18: JITTER MÉDIO EM (MS) COM 12 UES

Analisando os resultados obtidos no experimento 1 podemos observar que o escalonador proposto se destaca em relação aos outros em cenários com a rede congestionada atendendo um maior número de usuários de forma satisfatória, também apresenta um atraso e *jitter* médios baixos e não apresentam uma grande variação com a distância entre UE e eNodeB.

4.2 EXPERIMENTO 2: UES DISTRIBUÍDAS UNIFORMEMENTE

Foi simulado outro cenário onde as UEs foram distribuídas uniformemente em raio de 10 km da eNodeB com um número de UEs variando entre 4 e 20, foram realizadas 18 rodadas de simulações com sementes diferentes para aumentar a confiabilidade dos resultados.

É possivel verificar que o atraso médio observado com o uso do escalonador proposto possui um melhor desempenho em cenários com a rede congestionada se comparado aos demais métodos simulados, conforme demonstrado na TABELA 19 e na FIGURA 11, o congestionamento da rede está representada pela linha vertical localizada em 7 UEs.

	Cqa	FdMT	Pf	Rr	Ufpr
2 UEs	12,90±2,01	14,22±1,66	13,49±2,01	11,79±1,51	27,77±0,98
4 UEs	29,79±3,28	57,12±18,83	31,2±3,74	29,78±3,37	35,42±1,47
6 UEs	42,84±2,50	37,69±10,75	44,91±2,95	57,35±2,88	37,67±1,99
8 UEs	55,52±3,58	126,4±223,59	57,98±4,02	56,74±3,04	39,90±4,86
10 UEs	66,88±4,04	37,03±23,12	69,72±4,46	109,38±5,99	49,91±13,74
12 UEs	78,65±4,08	37,12±25,35	81,69±4,49	108,25±4,92	41,06±5,99
14 UEs	92,93±4,06	157,41±286,59	96,26±4,53	110,26±4,14	46,88±10,53
16 UEs	105,97±4,37	39,32±27,24	109,32±4,95	110,42±3,78	45,46±7,19
18 UEs	119,23±4,53	45,16±33,92	122,91±5,13	123,82±4,14	42,95±4,38
20 UEs	133,81±4,97	48,24±30,60	137,53±5,63	138,49±4,47	47,19±13,66

TABELA 19: ATRASO MÉDIO EM MILISSEGUNDOS COM DISTRIBUIÇÃO UNIFORME:

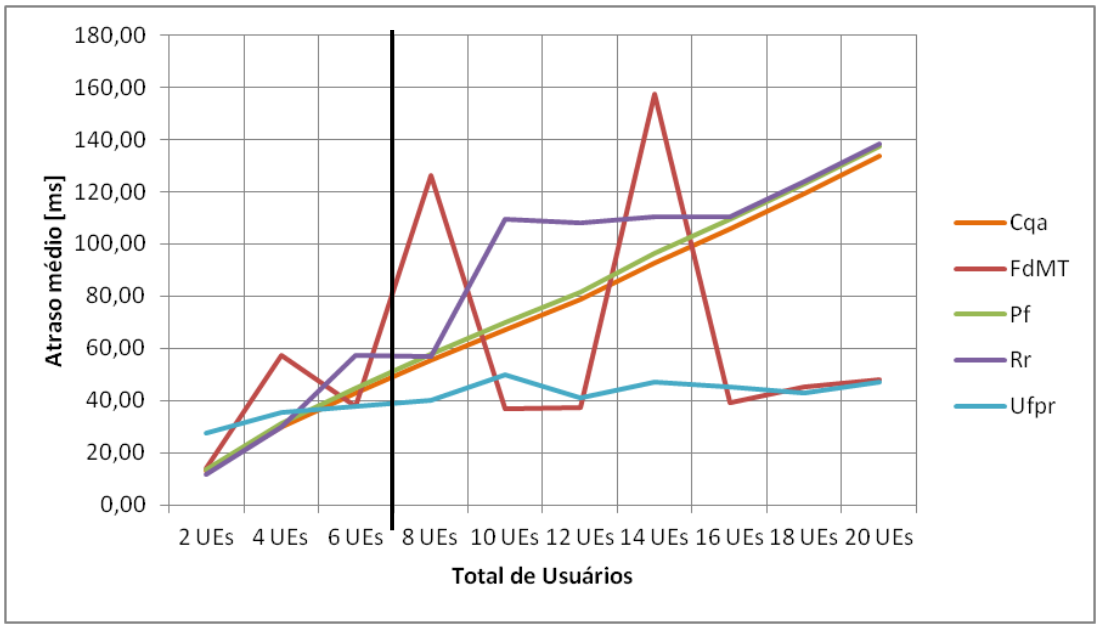

FIGURA 11: ATRASO MÉDIO EM MILISSEGUNDOS COM DISTRIBUIÇÃO UNIFORME

O escalonador proposto também apresentou um *jitter* baixo se aproximando dos resultados do escalonador RR com pouca variação em relação a ocupação total da rede, o *jitter* médio está demonstrado na TABELA 20 e na FIGURA 12.

	Cqa	FdMT	Pf	Rr	Ufpr
2 UEs	1,30±0,13	1,23±0,11	1,17±0,09	0,91±0,10	1,58±0,03
4 UEs	3,07±0,19	8,44±3,08	2,22±0,11	1,44±0,10	1,81±0,04
6 UEs	4,72±0,18	5,47±1,82	3,15±0,15	1,64±0,03	1,99±0,32
8 UEs	6,40±0,30	6,82±4,59	4,08±0,17	1,64±0,02	1,86±0,03
10 UEs	7,94±0,33	5,66±4,16	5,05±0,23	1,84±0,03	2,03±0,20
12 UEs	9,47±0,31	5,93±5,02	5,98±0,30	1,83±0,03	1,91±0,07
14 UEs	11,27±0,38	7,24±6,30	6,90±0,39	1,84±0,02	2,05±0,24
16 UEs	12,90±0,34	6,07±4,92	7,83±0,49	1,85±0,02	2,01±0,18
18 UEs	14,52±0,38	7,07±5,88	8,78±0,49	1,96±0,03	1,96±0,09
20 UEs	16,34±0,31	7,38±5,13	9,78±0,53	2,03±0,03	2,06±0,21

TABELA 20: JITTER MÉDIO EM MILISSEGUNDOS COM DISTRIBUIÇÃO UNIFORME

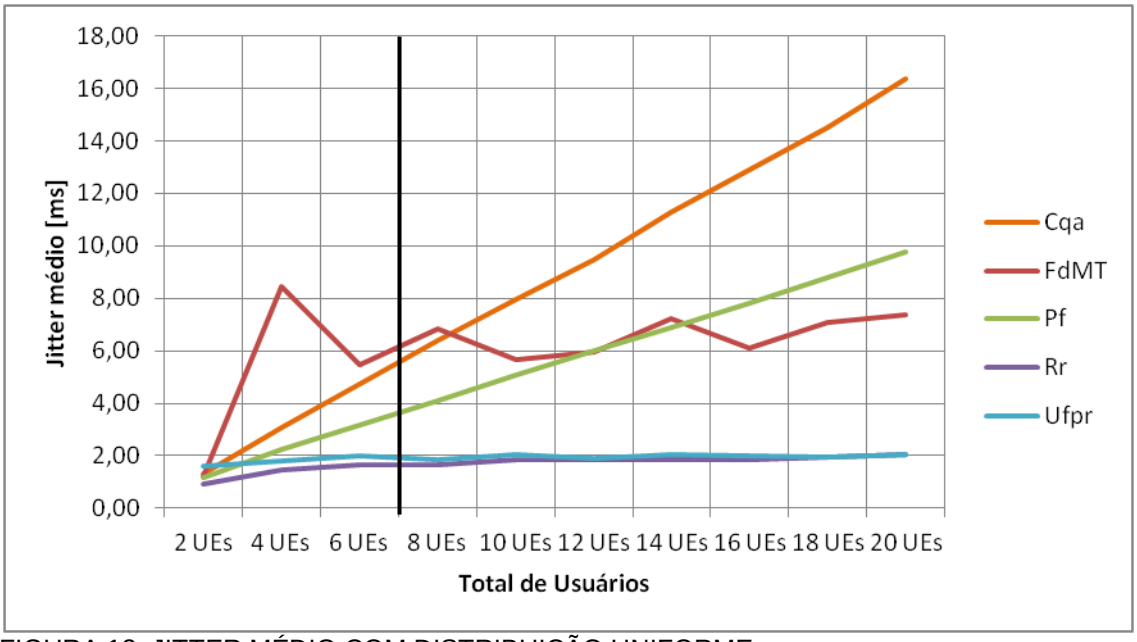

FIGURA 12: JITTER MÉDIO COM DISTRIBUIÇÃO UNIFORME

Também é possivel verificar que a taxa total na rede atinge um valor maior comparado com os outros escalonadores, taxa total na rede está demonstrada na TABELA 21 e na FIGURA 13.

	Cqa	FdMT	Pf	Rr	Ufpr
2 UEs	9,93±0,26	9,25±0,37	9,70±0,34	10,02±0,19	7,08±0,19
4 UEs	12,19±1,44	10,81±0,67	12,61±1,42	13,20±1,31	10,84±0,59
6 UEs	12,25±0,78	11,40±0,57	13,37±1,34	10,34±0,98	13,53±1,05
8 UEs	12,43±0,85	11,86±0,60	13,70±1,53	13,95±1,26	14,47±2,28
10 UEs	12,75±0,77	12,35±0,74	14,20±1,52	8,78±0,80	15,61±1,11
12 UEs	12,90±0,68	14,07±3,68	14,47±1,31	10,73±0,83	16,51±1,12
14 UEs	12,65±0,58	15,25±3,10	14,13±1,09	12,22±0,81	17,33±1,32
16 UEs	12,64±0,53	15,34±3,39	14,11±0,91	13,96±0,77	18,12±1,39
18 UEs	12,61±0,51	17,65±5,49	14,09±0,91	13,96±0,78	19,33±2,03
20 UEs	12,47±0,48	19,26±7,86	13,84±0,84	13,70±0,71	21,74±2,75

TABELA 21: TAXA TOTAL DA REDE (MBPS) COM DISTRIBUIÇÃO UNIFORME

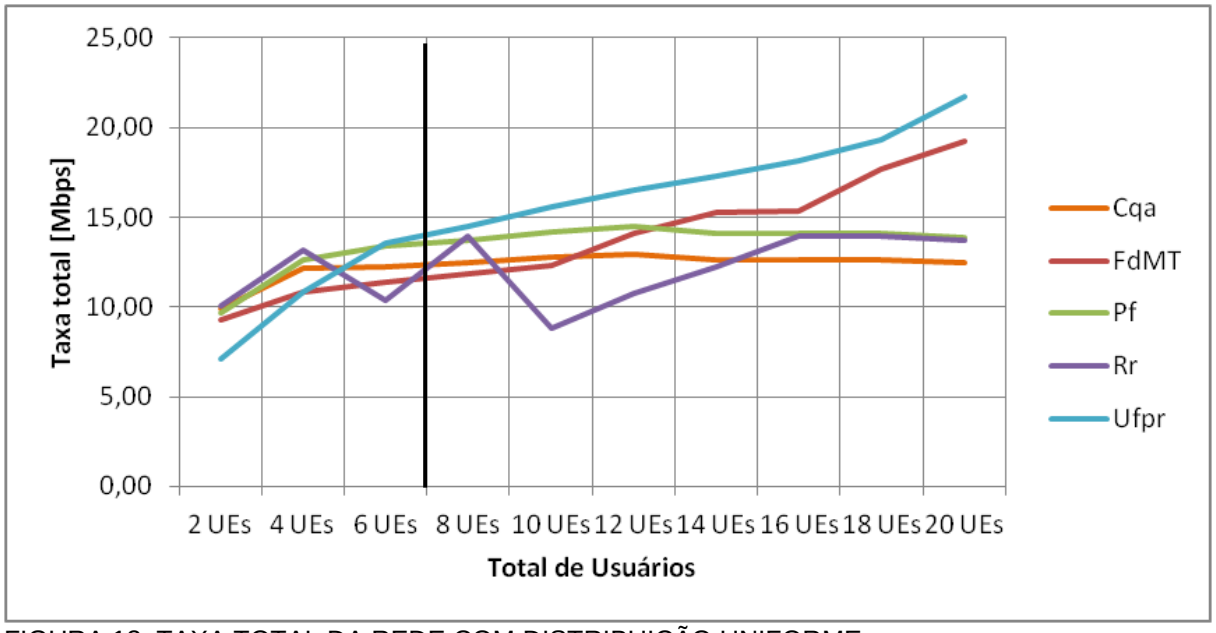

FIGURA 13: TAXA TOTAL DA REDE COM DISTRIBUIÇÃO UNIFORME

É possivel notar que em casos onde a ocupação da rede é baixa o algoritmo proposto apresenta uma taxa total inferior aos demais, isto por que, ele aloca recursos somente para atender os usuários até o limite estabelecido.

O número de usuários satisfeitos está demonstrado na TABELA 22 e na FIGURA 14

	Cqa	FdMt	Pf	Rr	Ufpr
2UEs	2±0	2±0	2±0	2±0	2±0
4 UEs	4±0	2,05±0,14	3,66±0,33	3,83±0,23	3,88±0,19
6 UEs	0,33±0,85	2,11±0,19	2,11±0,77	0,55±0,42	4,66±0,41
8 UEs	0±0	2,22±0,25	1,16±0,89	0,77±0,53	5,16±0,82
10 UEs	0±0	2,44±0,3	0,83±0,67	0±0	5,55±0,42
12 UEs	0±0	2,66±0,35	0,66±0,62	0±0	5,94±0,43
14 UEs	0±0	2,77±0,39	0,5±0,59	0±0	6±0,35
16 UEs	0±0	2,88±0,35	0±0	0±0	6,33±0,35
18 UEs	0±0	3,05±0,48	0±0	0±0	6,55±0,47
20 UEs	0±0	3,11±0,54	0±0	0±0	7±0,46

TABELA 22: USUÁRIOS SATISFEITOS COM DISTRIBUIÇÃO UNIFORME

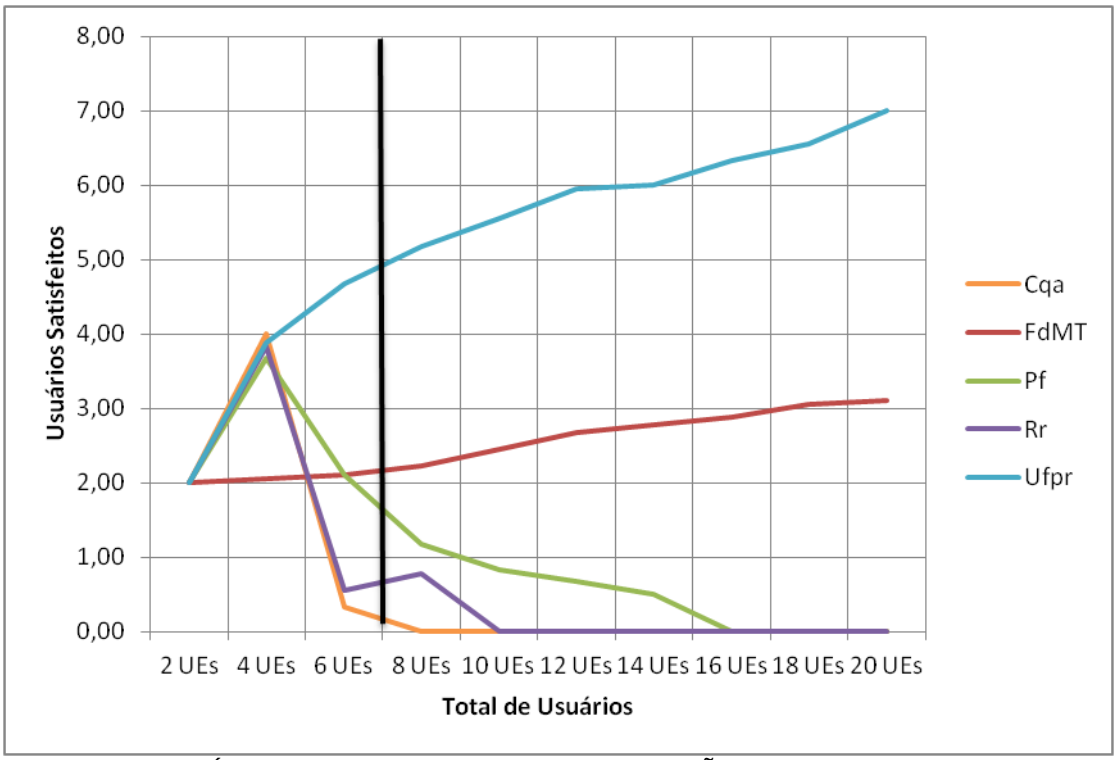

FIGURA 14: USUÁRIOS SATISFEITOS COM DISTRIBUIÇÃO UNIFORME

Analisando os resultados do experimento 2 verificou-se um aumento significativo de usuários satisfeitos em relação aos escalonadores existentes em cenários com sobrecarga na rede. Teve um delay e atraso médio aceitáveis para a aplicação de vídeo.

4.3 EXPERIMENTO 3: SIMULAÇÃO DO PSNR e SSIM

Neste experimento foi transmitido um vídeo com a duração de 20 segundos na resolução de 720p e calculado o PSNR e o SSIM fazendo uso do módulo do EVALVID, com as UEs que receberam o vídeo localizadas a 150m, 3000m, 6000m e 9000m da eNodeB. O PSNR e SSIM dos escalonadores avaliados estão demonstrados nas FIGURAS 15 a 22.

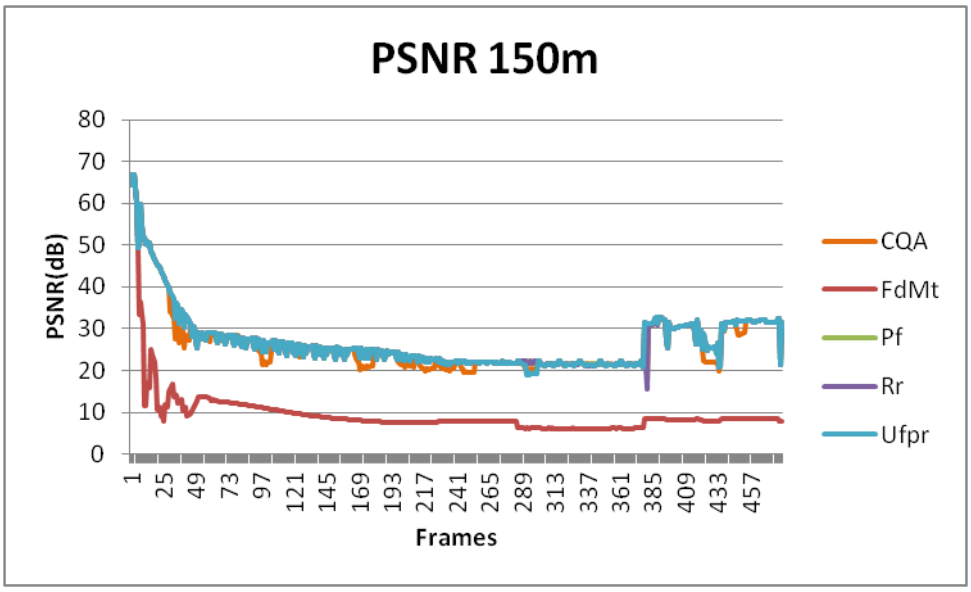

FIGURA 15: SIMULAÇÃO PSNR 150M

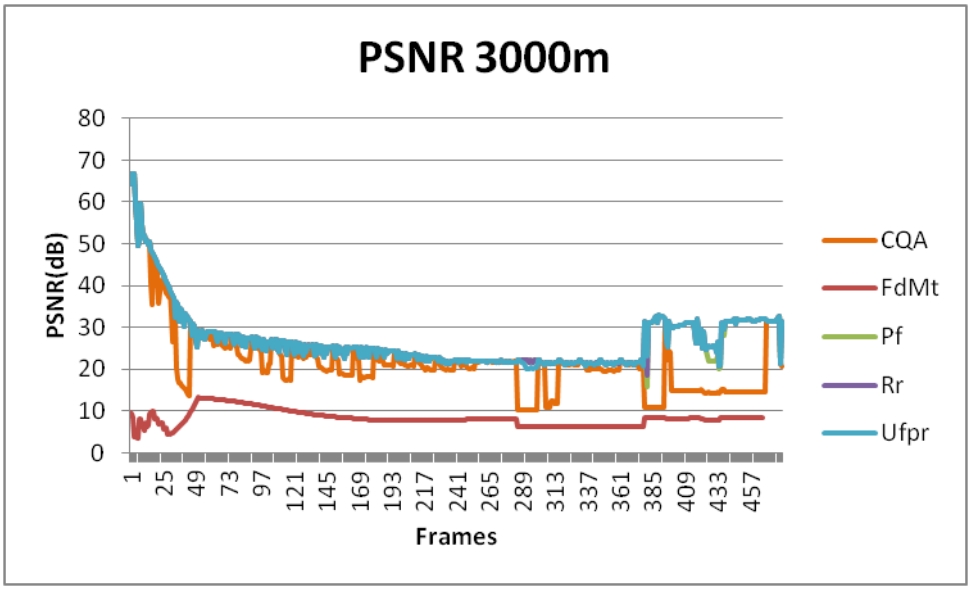

FIGURA 16: SIMULAÇÃO PSNR 3000M

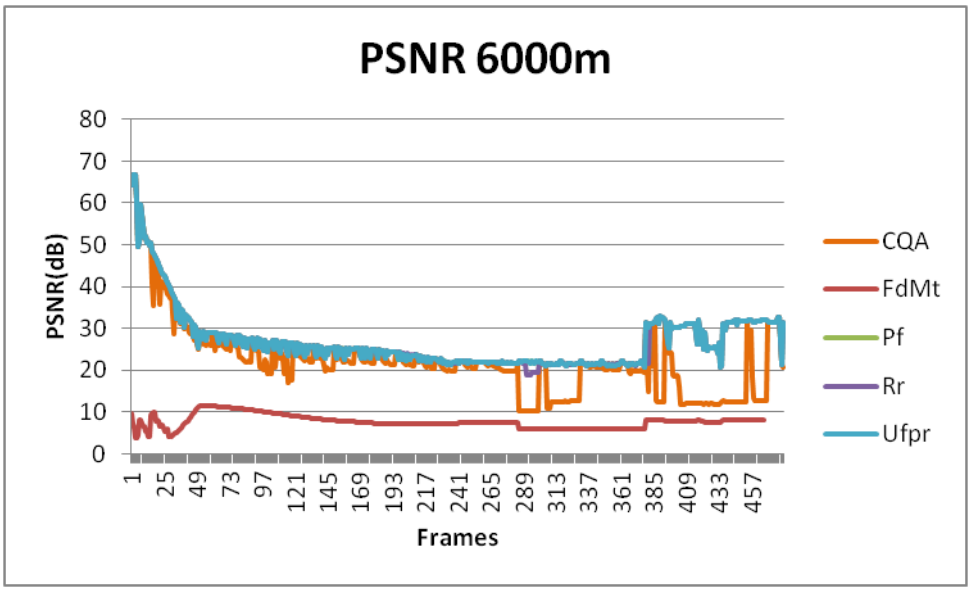

FIGURA 17: SIMULAÇÃO PSNR 6000M

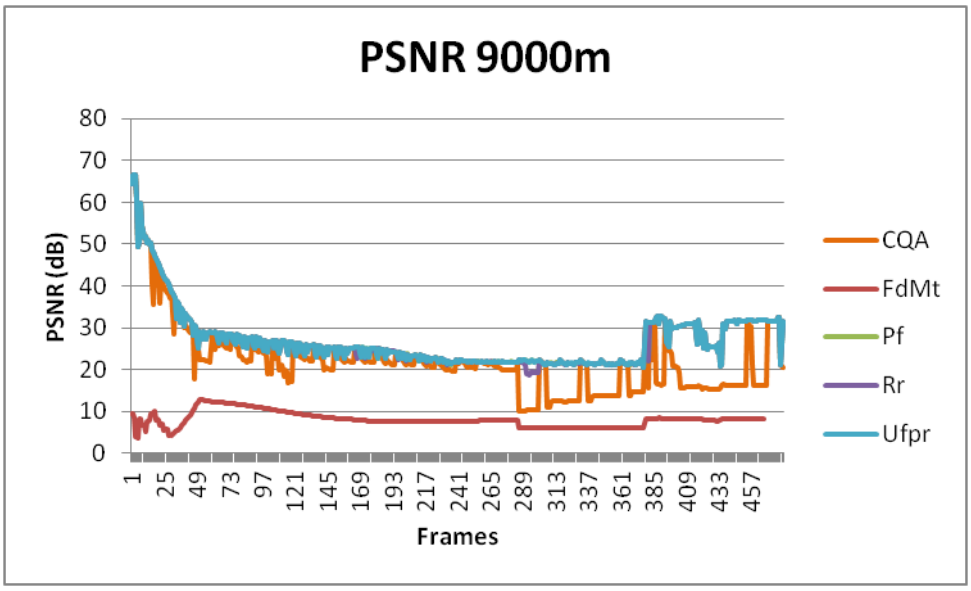

FIGURA 18: SIMULAÇÃO PSNR 6000M

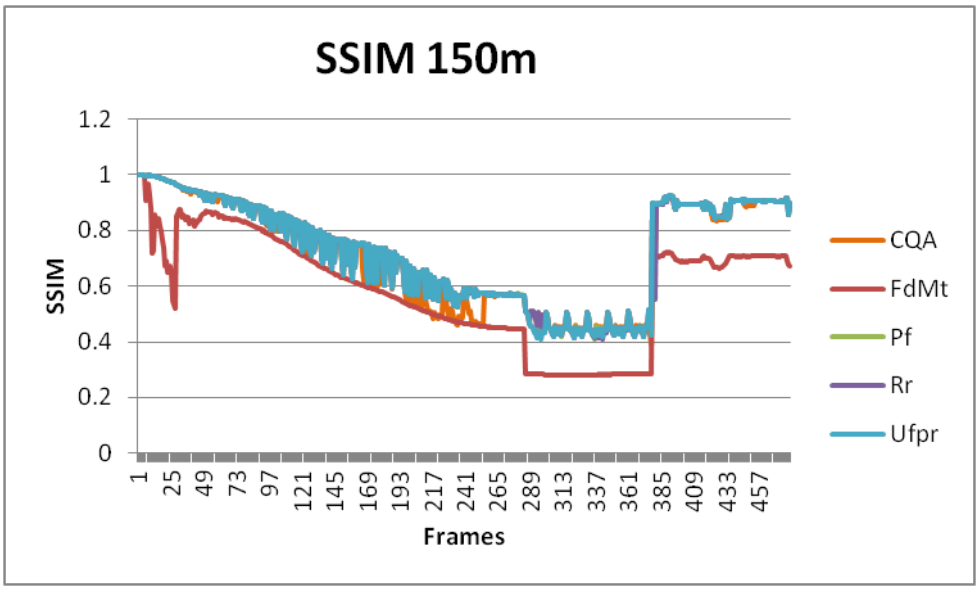

FIGURA 19: SIMULAÇÃO SSIM 150M

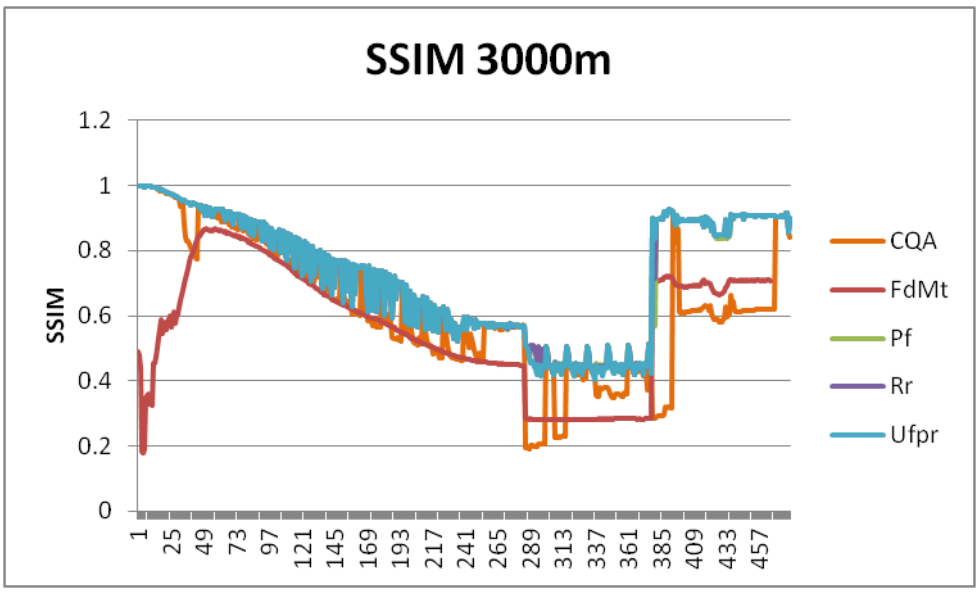

FIGURA 20: SIMULAÇÃO SSIM 3000M

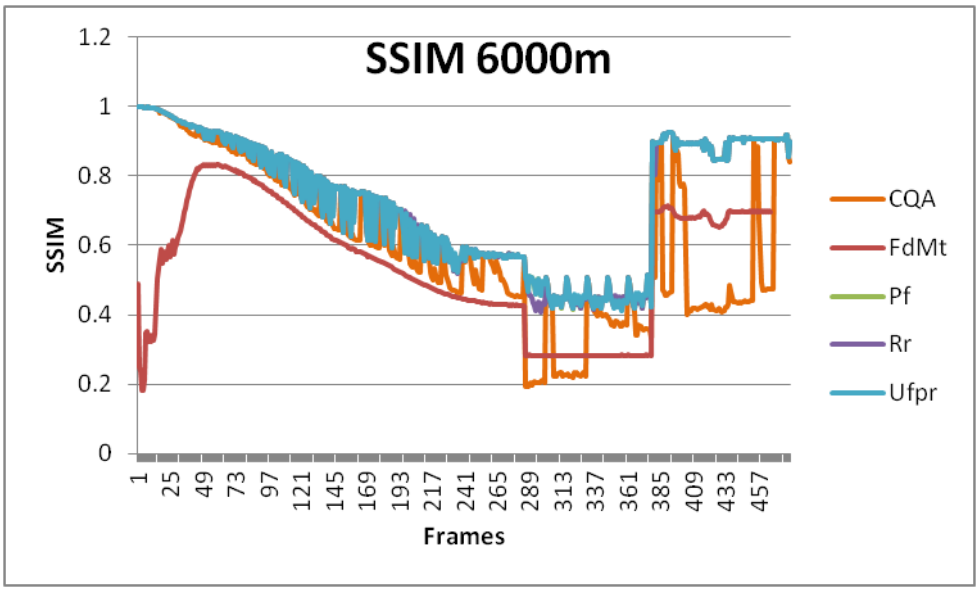

FIGURA 21: SIMULAÇÃO SSIM 6000M

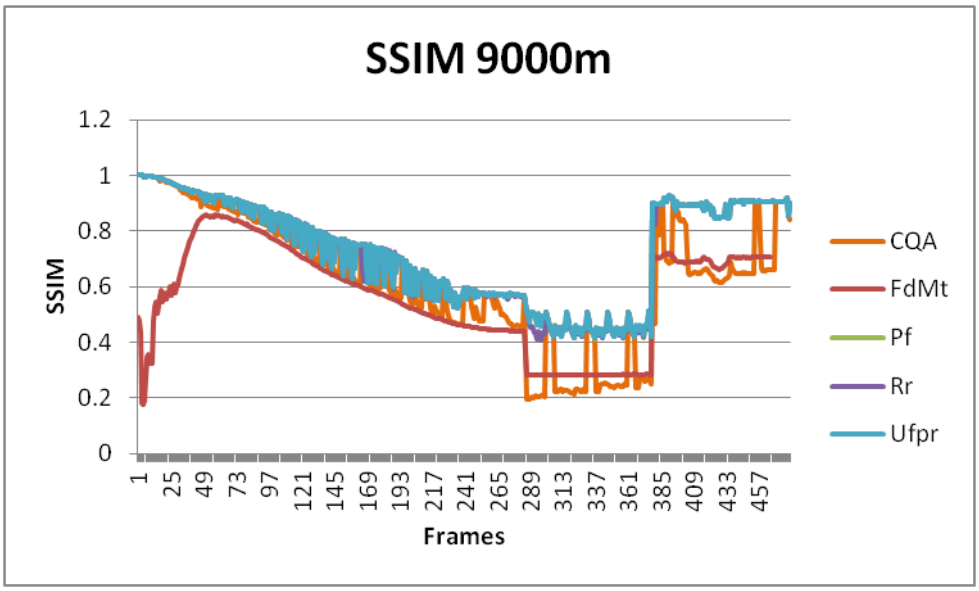

FIGURA 22: SIMULAÇÃO SSIM 9000M

5 CONCLUSÃO

Com o aumento do tráfego de vídeo nas redes LTE é necessário se atentar para a satisfação dos usuários com o serviço prestado, por este motivo foi proposto neste trabalho um novo algoritmo de escalonador de recursos que melhore a relação de usuários satisfeitos.

O escalonador foi desenvolvido em linguagem C++ como módulo do simulador NS-3 e testado através de simulações computacionais. Seu princípio de funcionamento é alocar recursos para o UE que está com melhor qualidade de sinal e não atingiu um certo limite de banda, que é atualizado e suavizado a cada TTI.

Os resultados obtidos indicaram que o número de usuário satisfeitos aumentou em relação aos escalonadores existentes, principalmente em cenários com a rede sobrecarregada.

O novo escalonador atende prioritariamente os usuários com melhor qualidade de sinal, já algoritmos que utilizam QoS atendem os usuários por ordem de chegada sendo assim um usuário com melhor qualidade de sinal que solicitar recursos pode não ser atendido, pois os outros com menor qualidade já estarão recebendo recursos,

Para trabalhos futuros, é possível, aumentar limite estabelecido caso houver sobra de RBGs no TTI, considerar *handoff, f*azer previsão da taxa de vídeo do usuário para ajustar a média móvel exponencial, , ampliar simulações considerando PSNR e SSIM,testar outros modelos de desvanecimento, principalmente para simular ambientes urbanos.

REFERÊNCIAS BIBIOGRÁFICAS

[1] CISCO SYSTEMS. **Cisco VNI Forecast and Methodology, 2015-2020.** 2016. Disponível em: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html>. Acesso em: 02 nov. 2016.

[2] ANATEL. **Dados de acessos de dispositivos móveis pessoais por tecnologia.** 2016. Dados considerados de julho de 2015 a julho de 2016. Disponível em: <http://ftp.anatel.gov.br/dados/Acessos/Movel_Pessoal/Por_Tecnologia/csv/>. Acesso em: 10 nov. 2016.

[3] ALI-YAHIYA, Tara. **Understanding LTE and its Performance.** S. L: Springer New York, 2011. 250 p.

[4] SMALL CELL FORUM. **LTE eNB L1 API Definition v1.1.** (Femto Forum Technical Paper), 2010. Disponível em <http://www.smallcellforum.org/resources-technical-papers>. Acesso em: out. 2016.

[5] CTTC, Centre Tecnològic de Telecomunicacions de Catalunya. **LENA - LTE-EPC Network SimAtor.** Disponível em: <http://networks.cttc.es/mobile-networks/software-tools/lena/>. Acesso em: 10 nov. 2016.

[6] KLAUE, Jirka; RATHKE, Berthold; WOLISZ, Adam. EvalVid – A Framework for Video Transmission and Quality Evaluation. In: MODELLING TECHNIQUES AND TOOLS, 13., 2003, Urbana. **Proceedings... .** Urbana: Springer Berlin Heidelberg, 2003.

[7] 3RD GENERATION PARTNERSHIP PROJECT. **TS 36.322 RELEASE 13**: Evolved Universal Terrestrial Radio Access (E-UTRA): Radio Link Control (RLC) protocol specification. [s. L.], 2016. Disponível em: <http://www.3gpp.org/dynareport/36322.htm>. Acesso em: 04 nov. 2016.

[8] BARAYAN, Yaser; KOSTANIC, Ivica. Performance Evaluation of Proportional Fairness Scheduling in LTE. In: PROCEEDINGS OF THE WORLD CONGRESS ON ENGINEERING AND COMPUTER SCIENCE, 2013, San Francisco. **Proceedings... .** San Francisco: Wcecs, 2013. v. 2, p. 23 - 25.

[9] ICT. **3GPP LTE / LTE-A Standardization: Status and Overview of Technologie.** [s. L.]: Ict, 2015. Color.

[10] NOKIA (Illinois). **4G speed record smashed with 1.4 Gigabits-per-second mobile call #MWC12.** 2012. Disponível em: <http://company.nokia.com/en/news/press-releases/2012/02/27/4g-speed-record-smashed-with-14-gigabits-per-second-mobile-call-mwc12>. Acesso em: 06 nov. 2016.

[11] WATANABE, Kimio; MACHIDA, Mamoru. 27 FUJITSU Sci. Tech. J., Vol. 48, No. 1, pp. 27–32 (January 2012) Outdoor LTE Infrastructure Equipment (eNodeB). **Fujitsu Scientific & Technical Journal.** [s. L.], p. 27-32. jan. 2012.

[12] 3RD GENERATION PARTNERSHIP PROJECT. **TS 36.331 RELEASE 14**: Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification. [s. L.], 2016. Disponível em: <http://www.3gpp.org/dynareport/36331.htm>. Acesso em: 04 nov. 2016.

[13] GRONDALEN, O. et al. Scheduling Policies in Time and Frequency Domains for LTE Downlink Channel: a Performance Comparison. **IEEE Transactions On Vehicular Technology.** [s. L.], p. 1-1. jul. 2016.

[14] SESIA, Stefania; TOUFIK, Issam; BAKER, Matthew. **LTE - The UMTS Long Term Evolution: From Theory to Practice.** 2. ed. [s. L.]: John Wiley & Sons, 2011.

[15] SALMAN, Mustafa Ismael et al. CQI-MCS Mapping for Green LTE Downlink Transmission. In: PROCEEDINGS OF THE APAN NETWORK RESEARCH WORKSHOP, 36., 2013, [s. L.]. **Proceedings.** [s. L.]: Sfu, 2013.

[16] SUBRAMANIAN, Ramprasad et al. Survey of LTE downlink schedulers algorithms in open access simulation tools NS-3 and LTE-sim. **International Journal Of Wireless & Mobile Networks (IJWMN).** Sydney. abr. 2015.

[17] F. CAPOZZI et al. Downlink Packet Scheduling in LTE Cellular Networks: Key Design Issues and a Survey. **IEEE Comm. Surveys And Tutorials.** [s. L.]. 06 jul. 2012.

[18] BOKHARI, F.a.; YANIKOMEROGLU, H.; WONG, W.k.. Cross-Layer Resource Scheduling for Video Traffic in the Downlink of OFDMA-Based Wireless 4G Networks. **Eurasip Journal On Wireless Communications And Networking.** [s. L.], p. 1-10. jan. 2009.

[19] WONG, W. K.; TANG, H. Y.; LEUNG, V. C. M.. Token bank fair queuing: a new scheduling algorithm for wireless multimedia services. **International Journal Of Communication Systems.** [s. L.], p. 591-614. ago. 2004.

[20] G.MONGHA et al. QoS Oriented Time and Frequency Domain Packet Schedulers for The UTRAN Long Term Evolution. In: PROCEEDINGS OF IEEE VEHICULAR TECHNOLOGY CONFERENCE (VTC), 68., 2008, Singapora. **Conference.** Singapora: Ieee, 2008. p. 2532 - 2536.

[21] NS-3 PROJECT. **Design Documentation.** Disponível em: <https://www.nsnam.org/docs/models/html/lte-design.html>. Acesso em: 02 nov. 2016.

[22] GUEDES, Susana; CONCEIÇÃO, Valter; CARVALHO, Nuno. Plataforma de Desenvolvimento e Simulação de Protocolos. In: OITAVA CONFERÊNCIA SOBRE REDES DE COMPUTADORES (CRC'05), 2005, Portalegre, Portugal. **Artigo.** Portalegre, Portugal: Fc-di-lasige, 2005.

[23] FOUNDATION, National Science. “**Network Simulator 3”.** Disponível em: <https://www.nsnam.org/>. Acesso em: 02 ago. 2016.

[24] DIGITAL SUBSCRIBER LINE FORUM. **TR-126**: Triple-play services quality of experience (QoE) requirements. [s. L.]: [s. E.], 2006.

[25] PATTERSON, Jason Robert Carey. **Resolutions & Bitrates.** 2012. Disponível em: <http://www.lighterra.com/papers/videoencodingh264/>. Acesso em: 06 nov. 2016.

[26] PHILLIPS, C.; SICKER, D.; GRUNWALD, D.. A Survey of Wireless Path Loss Prediction and Coverage Mapping Methods. **Ieee Communications Surveys & Tutorials.** [s. L.], p. 255-270. jan. 2013.

ANEXO 01: FLUXOGRAMA E CÓDIGO UTILIZADO NOS TESTES DE VALIDAÇÃO.

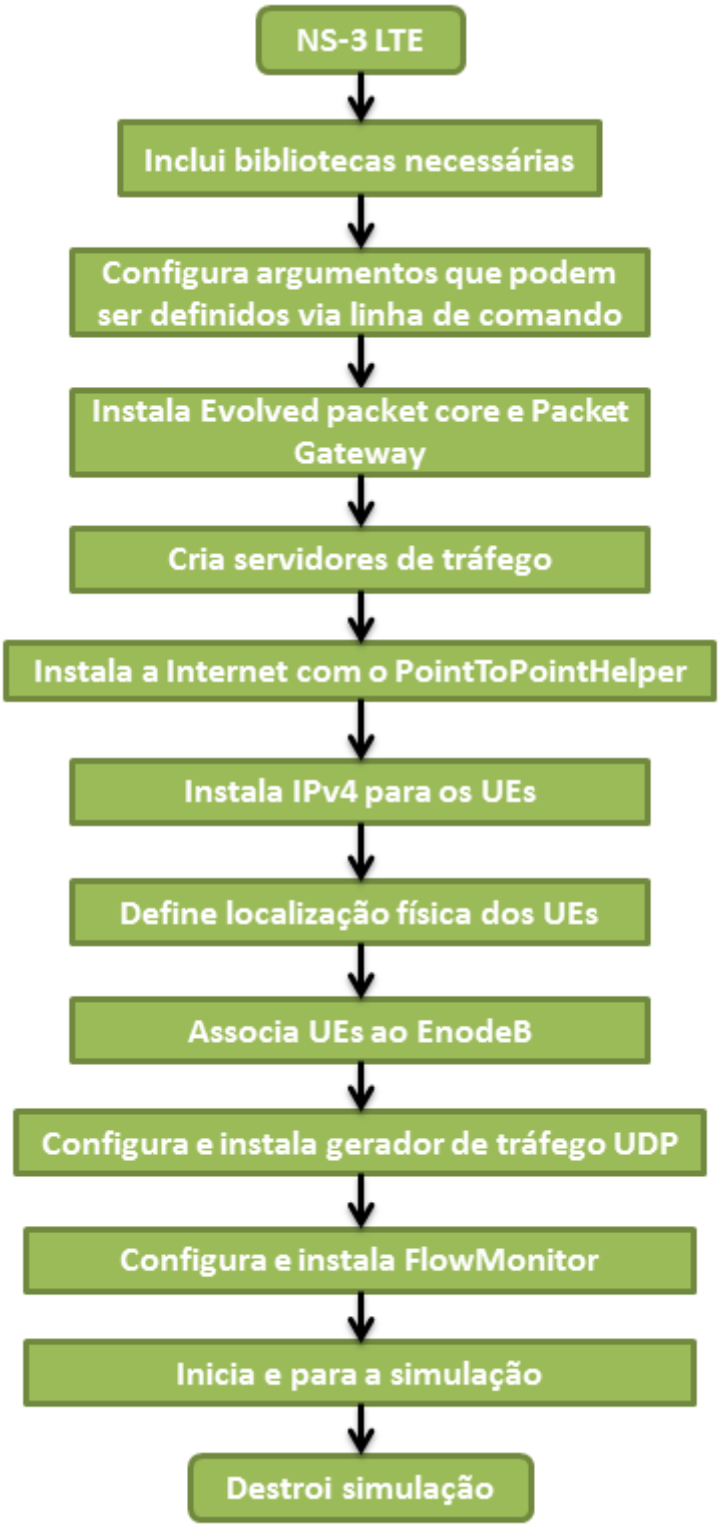

FIGURA 23: DIAGRAMA DE BLOCOS DO SCRIPT DE VALIDAÇÃO.

```
/*=============================================================================
 *  Título: SCRIPT DE VALIDAÇÃO DO NOVO ESCALONADOR LTE
 *
 *  Trabalho de Conclusão de Curso apresentado como exigência parcial
 *  aprovação na disciplina de Trabalho de Conclusão do Curso de Engenharia
 *  Elétrica, Setor de Tecnologia, Universidade Federal do Paraná
 *
 *    Alunos:          Evandro Luis Copercini
 *                     Joel Aparecido Barbosa Junior
 *
 *    Orientador: Prof. Dr. Carlos Marcelo Pedroso
 */

#include <fstream>
#include <string.h>
#include <iostream>

#include "ns3/csma-helper.h"
#include "ns3/evalvid-client-server-helper.h"

#include "ns3/lte-helper.h"
#include "ns3/epc-helper.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/ipv4-global-routing-helper.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-helper.h"
#include "ns3/config-store.h"
//#include "ns3/netanim-module.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/random-variable-stream.h"

#define ALFA 0.2

//#include "ns3/gtk-config-store.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("EvalvidLTE");


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

  uint16_t seed = 1;
  uint16_t numberOfNodes1 = 8;  //  Tráfego de fundo
  uint16_t numberOfNodes2 = 1;
  uint16_t distOfUEs1 = 0;
  uint16_t distOfUEs2 = 0;
  uint16_t numberOfUEs1 = 0;
  uint16_t numberOfUEs2 = 0;
  std::string schedType("ns3::FdMtFfMacScheduler");

  //LogComponentEnable ("EvalvidClient", LOG_LEVEL_INFO);
  //LogComponentEnable ("EvalvidServer", LOG_LEVEL_INFO);
  LogComponentEnable ("EvalvidLTE", LOG_LEVEL_INFO);
```

```
  CommandLine cmd;
  cmd.AddValue("seed", "Seed of simulation", seed);
  cmd.AddValue("numberOfNodes1", "Number of UE1 pairs", numberOfNodes1);
  cmd.AddValue("distOfUEs1", "Distance of UE1 pairs", distOfUEs1);
  cmd.AddValue("distOfUEs2", "Distance of UE2 pairs", distOfUEs2);
  cmd.AddValue("numberOfUEs1", "Number of UE1 pairs", numberOfUEs1);
  cmd.AddValue("numberOfUEs2", "Number of UE2 pairs", numberOfUEs2);
  cmd.AddValue("schedType", "Scheduler Type", schedType);
  cmd.Parse(argc, argv);

  ns3::SeedManager::SetSeed(seed);

   uint16_t port = 8000;
  //std::string jitterBufferLength("0.2s");
  std::string simulationDuration("10s");
  std::string transmitterStartTime("0s");
  std::string transmitterStopTime("10s");
  std::string receiverStartTime("0s");
  std::string receiverStopTime("10s");


  Ptr<LteHelper> LteObjeto = CreateObject<LteHelper> ();
  //Ptr<EpcHelper>  epcHelper = CreateObject<EpcHelper> ();
  Ptr<PointToPointEpcHelper>  EpcObjeto =
CreateObject<PointToPointEpcHelper> ();
  LteObjeto->SetEpcHelper (EpcObjeto);
  //LteObjeto->SetSchedulerType("ns3::RrFfMacScheduler");
  //LteObjeto->SetSchedulerType ("ns3::PfFfMacScheduler");
 // LteObjeto->SetSchedulerType ("ns3::ufprMacScheduler");
 //LteObjeto->SetSchedulerType ("ns3::FdMtFfMacScheduler");
  LteObjeto->SetSchedulerType (schedType);

  LteObjeto->SetEnbDeviceAttribute ("UlBandwidth", UintegerValue (50)); //
Alterando banda para 50 RBs = 10 MHz
  LteObjeto->SetEnbDeviceAttribute ("DlBandwidth", UintegerValue (50));
  LteObjeto->SetEnbAntennaModelType ("ns3::IsotropicAntennaModel");
  LteObjeto->SetEnbDeviceAttribute ("DlEarfcn", UintegerValue (100));
  LteObjeto->SetEnbDeviceAttribute ("UlEarfcn", UintegerValue (100 +
18000));



// Default config.
//default ns3::LteHelper::PathlossModel
"ns3::FriisSpectrumPropagationLossModel"
//default ns3::LteEnbNetDevice::UlBandwidth "25"
//default ns3::LteEnbNetDevice::DlBandwidth "25"
//default ns3::LteEnbNetDevice::DlEarfcn "100"
//default ns3::LteEnbNetDevice::UlEarfcn "18100"
//default ns3::LteUePhy::TxPower "10"
//default ns3::LteUePhy::NoiseFigure "9"
//default ns3::LteEnbPhy::TxPower "30"
//default ns3::LteEnbPhy::NoiseFigure "5"



  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> UDPServer = 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, UDPServer);


  // ************************************
  // *   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
(UDPServer->GetObject<Ipv4> ());
  RotaDefault2->SetDefaultRoute (interfacesIP[2].GetAddress (0), 1);



  // ************************************
  // *     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(0,9000), rand-
>GetInteger(0,9000), 0));
}

//x,y,z
/*
for(uint16_t i =0; i < numberOfUEs1;i++)
{
positionAlloc1-> Add(Vector(distOfUEs1, 0, 0)); //UEs dist1
}
for(uint16_t i =0; i < numberOfUEs2;i++)
{
positionAlloc1-> Add(Vector(distOfUEs2, 0, 0)); //UEs dist2
}
*/
Ptr<ListPositionAllocator> positionAlloc2 =
CreateObject<ListPositionAllocator> ();

 for (uint16_t i = 0; i < numberOfNodes2; i++)
    {
      positionAlloc2->Add (Vector(distOfUEs1,0, 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));

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

std::cout << "\nEscalonador: " << schedType << "\n";

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

for (uint32_t u = 0; u < ueNodesA.GetN (); u++)
    {
      Ptr<Node> ueNode_udp = ueNodesA.Get (u);
      Ptr<Ipv4StaticRouting> ueStaticRouting1 =
ipv4Roteamento.GetStaticRouting (ueNode_udp->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        *
  // ************************************

  //lteHelper->ActivateEpsBearer (ueLteDevs, EpsBearer
(EpsBearer::NGBR_VIDEO_TCP_DEFAULT), EpcTft::Default ());

  NS_LOG_INFO ("Create Applications.");

  EvalvidServerHelper server(port);
  server.SetAttribute ("SenderTraceFilename",
StringValue("blue_sky_1080p25.st"));
  server.SetAttribute ("SenderDumpFilename", StringValue("sd_a01_lte"));
  ApplicationContainer apps = server.Install(VideoServer);
  apps.Start (Time(receiverStartTime));
  apps.Stop (Time(receiverStopTime));

  EvalvidClientHelper client ( interfacesIP[1].GetAddress (1) ,port);
//Video Server IP and port
  client.SetAttribute ("ReceiverDumpFilename", StringValue("rd_a01_lte"));
  apps = client.Install (ueNodesB.Get(0));
  apps.Start (Time(transmitterStartTime));
  apps.Stop (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.GetAddr
ess(u), ulPort));
          ctTcpSender.SetAttribute("MaxBytes", UintegerValue(0));
```

```
          ctSenderContainer = ctTcpSender.Install(UDPServer);

          // 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));
     }
*/


  // ***********************************
  // *      Trafego de Fundo em UDP     *
  // ***********************************
    //Ptr<UniformRandomVariable> rg = CreateObject<UniformRandomVariable>
();
    //rg->SetAttribute ("Min", DoubleValue (0.0));
    //rg->SetAttribute ("Max", DoubleValue (1000.0));
    //char str[100];

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

        uint16_t uePort = 1000;
        ++uePort;
        ApplicationContainer ueClientApps;
        ApplicationContainer ueServerApps;
        Address sinkLocalAddress (InetSocketAddress (Ipv4Address::GetAny
(), uePort));
        PacketSinkHelper sinkHelper
("ns3::UdpSocketFactory",sinkLocalAddress);
        //ueClientApps = sinkHelper.Install(UDPServer); // modificado para
trocar o sentido do tráfego de fundo
        ueClientApps = sinkHelper.Install(ueNodesA.Get(u));
        UdpClientHelper ueClient (uE_Interface1.GetAddress(u) , uePort);
        ueClient.SetAttribute ("Interval", TimeValue (Seconds (0.0016)));
        ueClient.SetAttribute ("MaxPackets", UintegerValue(100000000));
        ueClient.SetAttribute ("PacketSize", UintegerValue (1000)); //1000
bytes
        Config::SetDefault ("ns3::UdpSocket::RcvBufSize", UintegerValue
(1000000000));
        //ueServerApps = ueClient.Install (ueNodesA.Get(u)); // modificado
para trocar o sentido do tráfego de fundo
        ueServerApps = ueClient.Install (UDPServer);
        ueClientApps.Start (Time(transmitterStartTime));
        ueClientApps.Stop (Time(transmitterStopTime));
        //sprintf(str,"%dms",(int)rg->GetValue());
        cout<<"\nIniciando transmissão da UE " << u <<" em
"<<transmitterStartTime;
        ueServerApps.Start (Time(transmitterStartTime));

        ueServerApps.Stop (Time(simulationDuration));
```

```
    }

  //Animation
  //  AnimationInterface anim ("evalvidLTE.xml");

  // Flow monitor setup

  FlowMonitorHelper flowmon_helper;
  Ptr<FlowMonitor> monitor = flowmon_helper.InstallAll();

  NS_LOG_INFO ("Run Simulation.");
  Simulator::Stop(Time(simulationDuration));
  Simulator::Run ();

  std::stringstream result;
  result << schedType << "-"<< numberOfNodes1 << "UEs"<< "-
DistribuicaoUniforme-5Mbps.xml";
  monitor->SerializeToXmlFile(result.str(),false,true);

    std::stringstream resultcsv;
  resultcsv << schedType << "-"<< numberOfNodes1 << "UEs"<< "-
DistribuicaoUniforme-5Mbps.csv";



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

  ofstream csvfile;
  csvfile.open (resultcsv.str());


  csvfile << "SchedulerType,"<<schedType<< ",numberOfUEs1,"<<numberOfUEs1<<
",numberOfUEs2,"<<numberOfUEs2<< ",distOfUEs1"<<distOfUEs1<<
",distOfUEs2"<<distOfUEs2<< "\n";

      csvfile << "flowId,Source-
>Destination,timeFirstTxPacket,timeFirstRxPacket,timeLastTxPacket,timeLastR
xPacket,delaySum,jitterSum,txBytes,rxBytes,txPackets,rxPackets,lostPackets,
timesForwarded\n";



  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;
```

```
      double TimeFirstTxPacket = (iterator-
>second.timeFirstTxPacket).GetNanoSeconds();
      double TimeFirstRxPacket = (iterator-
>second.timeFirstRxPacket).GetNanoSeconds();
      double TimeLastTxPacket = (iterator-
>second.timeLastTxPacket).GetNanoSeconds();
      double TimeLastRxPacket = (iterator-
>second.timeLastRxPacket).GetNanoSeconds();
      double DelaySum = (iterator->second.delaySum).GetNanoSeconds();
      double JitterSum = (iterator->second.jitterSum).GetNanoSeconds();
      double TxBytes = iterator->second.txBytes;
      double RxBytes = iterator->second.rxBytes;
      double TxPackets = iterator->second.txPackets;
      double RxPackets = iterator->second.rxPackets;
      double LostPackets = iterator->second.lostPackets;
      double TimesForwarded = iterator->second.timesForwarded;



      csvfile << iterator->first << ", ( " << sourceAddress << ":" <<
sourcePort <<
          " -> " << destinationAddress << ":" << destinationPort << " ),";
      csvfile << std::fixed
        << std::setw(8) << std::setprecision(0) << TimeFirstTxPacket << ","
        << std::setw(8) << std::setprecision(0) << TimeFirstRxPacket << ","
        << std::setw(8) << std::setprecision(0) << TimeLastTxPacket << ","
        << std::setw(8) << std::setprecision(0) << TimeLastRxPacket << ","
        << std::setw(10) << std::setprecision(0) << DelaySum << ","
        << std::setw(10) << std::setprecision(0) << JitterSum << ","
        << std::setw(10) << std::setprecision(0) << TxBytes << ","
        << std::setw(10) << std::setprecision(0) << RxBytes << ","
        << std::setw(8) << std::setprecision(0) << TxPackets << ","
        << std::setw(8) << std::setprecision(0) << RxPackets << ","
        << std::setw(8) << std::setprecision(0) << LostPackets << ","
        << std::setw(8) << std::setprecision(0) << TimesForwarded << "\n";

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



  Simulator::Destroy ();


   NS_LOG_INFO ("Done.");
  return 0;

}
```

ANEXO 02: TRECHO DO ESCALONADOR DESENVOLVIDO

```
/*============================================================================
 *  Título: FUNÇÃO DoSchedDlTriggerReq() DO ESCALONADOR DESENVOLVIDO
 *
 *  Trabalho de Conclusão de Curso apresentado como exigência parcial
 *  aprovação na disciplina de Trabalho de Conclusão do Curso de Engenharia
 *  Elétrica, Setor de Tecnologia, Universidade Federal do Paraná
 *
 *    Alunos:          Evandro Luis Copercini
 *                     Joel Aparecido Barbosa Junior
 *
 *    Orientador: Prof. Dr. Carlos Marcelo Pedroso
 */



Void FdMtFfMacScheduler::DoSchedDlTriggerReq (const struct
FfMacSchedSapProvider::SchedDlTriggerReqParameters& params)
{
  NS_LOG_FUNCTION (this << " Frame no. " << (params.m_sfnSf >> 4) << "
subframe no. " << (0xF & params.m_sfnSf));
  // API generated by RLC for triggering the scheduling of a DL subframe


  // evaluate the relative channel quality indicator for each UE per each
RBG
  // (since we are using allocation type 0 the small unit of allocation is
RBG)
  // Resource allocation type 0 (see sec 7.1.6.1 of 36.213)

  RefreshDlCqiMaps ();

  int rbgSize = GetRbgSize (m_cschedCellConfig.m_dlBandwidth);
  int rbgNum = m_cschedCellConfig.m_dlBandwidth / rbgSize;
  std::map <uint16_t, std::vector <uint16_t> > allocationMap; // RBs map
per RNTI
  std::vector <bool> rbgMap;  // global RBGs map
  uint16_t rbgAllocatedNum = 0;
  std::set <uint16_t> rntiAllocated;
  rbgMap.resize (m_cschedCellConfig.m_dlBandwidth / rbgSize, false);
  FfMacSchedSapUser::SchedDlConfigIndParameters ret;

  //   update UL HARQ proc id
  std::map <uint16_t, uint8_t>::iterator itProcId;
  for (itProcId = m_ulHarqCurrentProcessId.begin (); itProcId !=
m_ulHarqCurrentProcessId.end (); itProcId++)
    {
      (*itProcId).second = ((*itProcId).second + 1) % HARQ_PROC_NUM;
    }

  // RACH Allocation
  m_rachAllocationMap.resize (m_cschedCellConfig.m_ulBandwidth, 0);
  uint16_t rbStart = 0;
  std::vector <struct RachListElement_s>::iterator itRach;
  for (itRach = m_rachList.begin (); itRach != m_rachList.end (); itRach++)
    {
      NS_ASSERT_MSG (m_amc->GetTbSizeFromMcs (m_ulGrantMcs,
m_cschedCellConfig.m_ulBandwidth) > (*itRach).m_estimatedSize, " Default UL
Grant MCS does not allow to send RACH messages");
```

```
      BuildRarListElement_s newRar;
      newRar.m_rnti = (*itRach).m_rnti;
      // DL-RACH Allocation
      // Ideal: no needs of configuring m_dci
      // UL-RACH Allocation
      newRar.m_grant.m_rnti = newRar.m_rnti;
      newRar.m_grant.m_mcs = m_ulGrantMcs;
      uint16_t rbLen = 1;
      uint16_t tbSizeBits = 0;
      // find lowest TB size that fits UL grant estimated size
      while ((tbSizeBits < (*itRach).m_estimatedSize) && (rbStart + rbLen <
m_cschedCellConfig.m_ulBandwidth))
        {
          rbLen++;
          tbSizeBits = m_amc->GetTbSizeFromMcs (m_ulGrantMcs, rbLen);
        }
      if (tbSizeBits < (*itRach).m_estimatedSize)
        {
          // no more allocation space: finish allocation
          break;
        }
      newRar.m_grant.m_rbStart = rbStart;
      newRar.m_grant.m_rbLen = rbLen;
      newRar.m_grant.m_tbSize = tbSizeBits / 8;
      newRar.m_grant.m_hopping = false;
      newRar.m_grant.m_tpc = 0;
      newRar.m_grant.m_cqiRequest = false;
      newRar.m_grant.m_ulDelay = false;
      NS_LOG_INFO (this << " UL grant allocated to RNTI " <<
(*itRach).m_rnti << " rbStart " << rbStart << " rbLen " << rbLen << " MCS "
<< m_ulGrantMcs << " tbSize " << newRar.m_grant.m_tbSize);
      for (uint16_t i = rbStart; i < rbStart + rbLen; i++)
        {
          m_rachAllocationMap.at (i) = (*itRach).m_rnti;
        }

      if (m_harqOn == true)
        {
          // generate UL-DCI for HARQ retransmissions
          UlDciListElement_s uldci;
          uldci.m_rnti = newRar.m_rnti;
          uldci.m_rbLen = rbLen;
          uldci.m_rbStart = rbStart;
          uldci.m_mcs = m_ulGrantMcs;
          uldci.m_tbSize = tbSizeBits / 8;
          uldci.m_ndi = 1;
          uldci.m_cceIndex = 0;
          uldci.m_aggrLevel = 1;
          uldci.m_ueTxAntennaSelection = 3; // antenna selection OFF
          uldci.m_hopping = false;
          uldci.m_n2Dmrs = 0;
          uldci.m_tpc = 0; // no power control
          uldci.m_cqiRequest = false; // only period CQI at this stage
          uldci.m_ulIndex = 0; // TDD parameter
          uldci.m_dai = 1; // TDD parameter
          uldci.m_freqHopping = 0;
          uldci.m_pdcchPowerOffset = 0; // not used

          uint8_t harqId = 0;
          std::map <uint16_t, uint8_t>::iterator itProcId;
          itProcId = m_ulHarqCurrentProcessId.find (uldci.m_rnti);
```

```
          if (itProcId == m_ulHarqCurrentProcessId.end ())
            {
              NS_FATAL_ERROR ("No info find in HARQ buffer for UE " <<
uldci.m_rnti);
            }
          harqId = (*itProcId).second;
          std::map <uint16_t, UlHarqProcessesDciBuffer_t>::iterator itDci =
m_ulHarqProcessesDciBuffer.find (uldci.m_rnti);
          if (itDci == m_ulHarqProcessesDciBuffer.end ())
            {
              NS_FATAL_ERROR ("Unable to find RNTI entry in UL DCI HARQ
buffer for RNTI " << uldci.m_rnti);
            }
          (*itDci).second.at (harqId) = uldci;
        }

      rbStart = rbStart + rbLen;
      ret.m_buildRarList.push_back (newRar);
    }
  m_rachList.clear ();


  // Process DL HARQ feedback
  RefreshHarqProcesses ();
  // retrieve past HARQ retx buffered
  if (m_dlInfoListBuffered.size () > 0)
    {
      if (params.m_dlInfoList.size () > 0)
        {
          NS_LOG_INFO (this << " Received DL-HARQ feedback");
          m_dlInfoListBuffered.insert (m_dlInfoListBuffered.end (),
params.m_dlInfoList.begin (), params.m_dlInfoList.end ());
        }
    }
  else
    {
      if (params.m_dlInfoList.size () > 0)
        {
          m_dlInfoListBuffered = params.m_dlInfoList;
        }
    }
  if (m_harqOn == false)
    {
      // Ignore HARQ feedback
      m_dlInfoListBuffered.clear ();
    }
  std::vector <struct DlInfoListElement_s> dlInfoListUntxed;
  for (uint16_t i = 0; i < m_dlInfoListBuffered.size (); i++)
    {
      std::set <uint16_t>::iterator itRnti = rntiAllocated.find
(m_dlInfoListBuffered.at (i).m_rnti);
      if (itRnti != rntiAllocated.end ())
        {
          // RNTI already allocated for retx
          continue;
        }
      uint8_t nLayers = m_dlInfoListBuffered.at (i).m_harqStatus.size ();
      std::vector <bool> retx;
      NS_LOG_INFO (this << " Processing DLHARQ feedback");
      if (nLayers == 1)
        {
```

```
          retx.push_back (m_dlInfoListBuffered.at (i).m_harqStatus.at (0)
== DlInfoListElement_s::NACK);
          retx.push_back (false);
        }
      else
        {
          retx.push_back (m_dlInfoListBuffered.at (i).m_harqStatus.at (0)
== DlInfoListElement_s::NACK);
          retx.push_back (m_dlInfoListBuffered.at (i).m_harqStatus.at (1)
== DlInfoListElement_s::NACK);
        }
      if (retx.at (0) || retx.at (1))
        {
          // retrieve HARQ process information
          uint16_t rnti = m_dlInfoListBuffered.at (i).m_rnti;
          uint8_t harqId = m_dlInfoListBuffered.at (i).m_harqProcessId;
          NS_LOG_INFO (this << " HARQ retx RNTI " << rnti << " harqId " <<
(uint16_t)harqId);
          std::map <uint16_t, DlHarqProcessesDciBuffer_t>::iterator itHarq
= m_dlHarqProcessesDciBuffer.find (rnti);
          if (itHarq == m_dlHarqProcessesDciBuffer.end ())
            {
              NS_FATAL_ERROR ("No info find in HARQ buffer for UE " <<
rnti);
            }

          DlDciListElement_s dci = (*itHarq).second.at (harqId);
          int rv = 0;
          if (dci.m_rv.size () == 1)
            {
              rv = dci.m_rv.at (0);
            }
          else
            {
              rv = (dci.m_rv.at (0) > dci.m_rv.at (1) ? dci.m_rv.at (0) :
dci.m_rv.at (1));
            }

          if (rv == 3)
            {
              // maximum number of retx reached -> drop process
              NS_LOG_INFO ("Maximum number of retransmissions reached ->
drop process");
              std::map <uint16_t, DlHarqProcessesStatus_t>::iterator it =
m_dlHarqProcessesStatus.find (rnti);
              if (it == m_dlHarqProcessesStatus.end ())
                {
                  NS_LOG_ERROR ("No info find in HARQ buffer for UE (might
change eNB) " << m_dlInfoListBuffered.at (i).m_rnti);
                }
              (*it).second.at (harqId) = 0;
              std::map <uint16_t, DlHarqRlcPduListBuffer_t>::iterator
itRlcPdu =  m_dlHarqProcessesRlcPduListBuffer.find (rnti);
              if (itRlcPdu == m_dlHarqProcessesRlcPduListBuffer.end ())
                {
                  NS_FATAL_ERROR ("Unable to find RlcPdcList in HARQ buffer
for RNTI " << m_dlInfoListBuffered.at (i).m_rnti);
                }
              for (uint16_t k = 0; k < (*itRlcPdu).second.size (); k++)
                {
                  (*itRlcPdu).second.at (k).at (harqId).clear ();
```

```
                }
              continue;
            }
          // check the feasibility of retransmitting on the same RBGs
          // translate the DCI to Spectrum framework
          std::vector <int> dciRbg;
          uint32_t mask = 0x1;
          NS_LOG_INFO ("Original RBGs " << dci.m_rbBitmap << " rnti " <<
dci.m_rnti);
          for (int j = 0; j < 32; j++)
            {
              if (((dci.m_rbBitmap & mask) >> j) == 1)
                {
                  dciRbg.push_back (j);
                  NS_LOG_INFO ("\t" << j);
                }
              mask = (mask << 1);
            }
          bool free = true;
          for (uint8_t j = 0; j < dciRbg.size (); j++)
            {
              if (rbgMap.at (dciRbg.at (j)) == true)
                {
                  free = false;
                  break;
                }
            }
          if (free)
            {
              // use the same RBGs for the retx
              // reserve RBGs
              for (uint8_t j = 0; j < dciRbg.size (); j++)
                {
                  rbgMap.at (dciRbg.at (j)) = true;
                  NS_LOG_INFO ("RBG " << dciRbg.at (j) << " assigned");
                  rbgAllocatedNum++;
                }

              NS_LOG_INFO (this << " Send retx in the same RBGs");
            }
          else
            {
              // find RBGs for sending HARQ retx
              uint8_t j = 0;
              uint8_t rbgId = (dciRbg.at (dciRbg.size () - 1) + 1) %
rbgNum;
              uint8_t startRbg = dciRbg.at (dciRbg.size () - 1);
              std::vector <bool> rbgMapCopy = rbgMap;
              while ((j < dciRbg.size ())&&(startRbg != rbgId))
                {
                  if (rbgMapCopy.at (rbgId) == false)
                    {
                      rbgMapCopy.at (rbgId) = true;
                      dciRbg.at (j) = rbgId;
                      j++;
                    }
                  rbgId = (rbgId + 1) % rbgNum;
                }
              if (j == dciRbg.size ())
                {
                  // find new RBGs -> update DCI map
```

```
                  uint32_t rbgMask = 0;
                  for (uint16_t k = 0; k < dciRbg.size (); k++)
                    {
                      rbgMask = rbgMask + (0x1 << dciRbg.at (k));
                      rbgAllocatedNum++;
                    }
                  dci.m_rbBitmap = rbgMask;
                  rbgMap = rbgMapCopy;
                  NS_LOG_INFO (this << " Move retx in RBGs " << dciRbg.size
()); 
                }
              else
                {
                  // HARQ retx cannot be performed on this TTI -> store it
                  dlInfoListUntxed.push_back (m_dlInfoListBuffered.at (i));
                  NS_LOG_INFO (this << " No resource for this retx ->
buffer it");
                }
            }
          // retrieve RLC PDU list for retx TBsize and update DCI
          BuildDataListElement_s newEl;
          std::map <uint16_t, DlHarqRlcPduListBuffer_t>::iterator itRlcPdu
=  m_dlHarqProcessesRlcPduListBuffer.find (rnti);
          if (itRlcPdu == m_dlHarqProcessesRlcPduListBuffer.end ())
            {
              NS_FATAL_ERROR ("Unable to find RlcPdcList in HARQ buffer for
RNTI " << rnti);
            }
          for (uint8_t j = 0; j < nLayers; j++)
            {
              if (retx.at (j))
                {
                  if (j >= dci.m_ndi.size ())
                    {
                      // for avoiding errors in MIMO transient phases
                      dci.m_ndi.push_back (0);
                      dci.m_rv.push_back (0);
                      dci.m_mcs.push_back (0);
                      dci.m_tbsSize.push_back (0);
                      NS_LOG_INFO (this << " layer " << (uint16_t)j << " no
txed (MIMO transition)");
                    }
                  else
                    {
                      dci.m_ndi.at (j) = 0;
                      dci.m_rv.at (j)++;
                      (*itHarq).second.at (harqId).m_rv.at (j)++;
                      NS_LOG_INFO (this << " layer " << (uint16_t)j << " RV
" << (uint16_t)dci.m_rv.at (j));
                    }
                }
              else
                {
                  // empty TB of layer j
                  dci.m_ndi.at (j) = 0;
                  dci.m_rv.at (j) = 0;
                  dci.m_mcs.at (j) = 0;
                  dci.m_tbsSize.at (j) = 0;
                  NS_LOG_INFO (this << " layer " << (uint16_t)j << " no
retx");
                }
```

```
            }
          for (uint16_t k = 0; k < (*itRlcPdu).second.at (0).at
(dci.m_harqProcess).size (); k++)
            {
              std::vector <struct RlcPduListElement_s> rlcPduListPerLc;
              for (uint8_t j = 0; j < nLayers; j++)
                {
                  if (retx.at (j))
                    {
                      if (j < dci.m_ndi.size ())
                        {
                          rlcPduListPerLc.push_back ((*itRlcPdu).second.at
(j).at (dci.m_harqProcess).at (k));
                        }
                    }
                }

              if (rlcPduListPerLc.size () > 0)
                {
                  newEl.m_rlcPduList.push_back (rlcPduListPerLc);
                }
            }
          newEl.m_rnti = rnti;
          newEl.m_dci = dci;
          (*itHarq).second.at (harqId).m_rv = dci.m_rv;
          // refresh timer
          std::map <uint16_t, DlHarqProcessesTimer_t>::iterator itHarqTimer
= m_dlHarqProcessesTimer.find (rnti);
          if (itHarqTimer== m_dlHarqProcessesTimer.end ())
            {
              NS_FATAL_ERROR ("Unable to find HARQ timer for RNTI " <<
(uint16_t)rnti);
            }
          (*itHarqTimer).second.at (harqId) = 0;
          ret.m_buildDataList.push_back (newEl);
          rntiAllocated.insert (rnti);
        }
      else
        {
          // update HARQ process status
          NS_LOG_INFO (this << " HARQ received ACK for UE " <<
m_dlInfoListBuffered.at (i).m_rnti);
          std::map <uint16_t, DlHarqProcessesStatus_t>::iterator it =
m_dlHarqProcessesStatus.find (m_dlInfoListBuffered.at (i).m_rnti);
          if (it == m_dlHarqProcessesStatus.end ())
            {
              NS_FATAL_ERROR ("No info find in HARQ buffer for UE " <<
m_dlInfoListBuffered.at (i).m_rnti);
            }
          (*it).second.at (m_dlInfoListBuffered.at (i).m_harqProcessId) =
0;
          std::map <uint16_t, DlHarqRlcPduListBuffer_t>::iterator itRlcPdu
=  m_dlHarqProcessesRlcPduListBuffer.find (m_dlInfoListBuffered.at
(i).m_rnti);
          if (itRlcPdu == m_dlHarqProcessesRlcPduListBuffer.end ())
            {
              NS_FATAL_ERROR ("Unable to find RlcPdcList in HARQ buffer for
RNTI " << m_dlInfoListBuffered.at (i).m_rnti);
            }
          for (uint16_t k = 0; k < (*itRlcPdu).second.size (); k++)
            {
```

```
              (*itRlcPdu).second.at (k).at (m_dlInfoListBuffered.at
(i).m_harqProcessId).clear ();
            }
        }
    }
  m_dlInfoListBuffered.clear ();
  m_dlInfoListBuffered = dlInfoListUntxed;

  if (rbgAllocatedNum == rbgNum)
    {
      // all the RBGs are already allocated -> exit
      if ((ret.m_buildDataList.size () > 0) || (ret.m_buildRarList.size ()
> 0))
        {
          m_schedSapUser->SchedDlConfigInd (ret);
        }
      return;
    }

  // Finalizou a alocação de RBGs para HARQ
  // Vai iniciar a alocação dos RBGs que sobraram para demais UEs
  tticont++;
  std::cout << " \n ---------------------- TTI " << tticont << "-----------
------------";

  for (int x=0; x<1000;x++,ueTaxaReservada[x]=0); // taxa já reservada de
cada UE, entre todos os RGBs

  for (int i = 0; i < rbgNum; i++)
    {
      for (int x=0; x<1000;x++,ueTaxaMaxima[x]=0); // para controle da taxa
maxima de cada UE no RBG
      NS_LOG_INFO (this << " ALLOCATION for RBG " << i << " of " <<
rbgNum);
      if (rbgMap.at (i) == false) //RGB free!!!
        {
          std::set <uint16_t>::iterator it;
          std::set <uint16_t>::iterator itMax = m_flowStatsDl.end ();
          double rcqiMax = 0.0;
          for (it = m_flowStatsDl.begin (); it != m_flowStatsDl.end ();
it++) // procura por UEs que desejam transmitir - o objetivo deste for é
montar um vetor com as taxas atingíveis dos UEs que querem transmitir nesse
TTI
            {
              std::set <uint16_t>::iterator itRnti = rntiAllocated.find
((*it));
              if ((itRnti != rntiAllocated.end
())||(!HarqProcessAvailability ((*it))))
                {
                  // UE already allocated for HARQ or without HARQ process
available -> drop it
                  if (itRnti != rntiAllocated.end ())
                  {
                    NS_LOG_DEBUG (this << " RNTI discared for HARQ tx" <<
(uint16_t)(*it));
                  }
                  if (!HarqProcessAvailability ((*it)))
                  {
                    NS_LOG_DEBUG (this << " RNTI discared for HARQ id" <<
(uint16_t)(*it));
                  }
```

```
                  continue; //UE já vai fazer HARQ alocado para HARQ; faz
it++ e procura próxima UE que seja transmitir
                }
              // se chegou aqui a UE "it" quer transmitir e não tem RGBs
alocados
              std::map <uint16_t,SbMeasResult_s>::iterator itCqi;
              itCqi = m_a30CqiRxed.find ((*it)); // itCqi é o CQI desta it
(UE)
              std::map <uint16_t,uint8_t>::iterator itTxMode;
              itTxMode = m_uesTxMode.find ((*it)); // itTxMode desta it
(UE)
              if (itTxMode == m_uesTxMode.end ())
                {
                  NS_FATAL_ERROR ("No Transmission Mode info on user " <<
(*it));
                }
              int nLayer = TransmissionModesLayers::TxMode2LayerNum
((*itTxMode).second);
              std::vector <uint8_t> sbCqi;
              if (itCqi == m_a30CqiRxed.end ())
                {
                  for (uint8_t k = 0; k < nLayer; k++)
                    {
                      sbCqi.push_back (1);  // start with lowest value
                    }
                }
              else
                {
                  sbCqi = (*itCqi).second.m_higherLayerSelected.at
(i).m_sbCqi;
                }
              uint8_t cqi1 = sbCqi.at (0);
              uint8_t cqi2 = 1;
              if (sbCqi.size () > 1)
                {
                  cqi2 = sbCqi.at (1);
                }
              if ((cqi1 > 0)||(cqi2 > 0)) // CQI == 0 means "out of range"
(see table 7.2.3-1 of 36.213)
                {
                  if (LcActivePerFlow ((*it)) > 0)  // UE tem fluxos ativos
para transmitir
                    {
                      // this UE has data to transmit
                      double achievableRate = 0.0;
                      uint8_t mcs = 0;
                      for (uint8_t k = 0; k < nLayer; k++)  // neste for
está calculando a taxa máxima de transmissão para esta UE
                        {
                          if (sbCqi.size () > k)
                            {
                              mcs = m_amc->GetMcsFromCqi (sbCqi.at (k));
                            }
                          else
                            {
                              // no info on this subband -> worst MCS
                              mcs = 0;
                            }
                          achievableRate += ((m_amc->GetTbSizeFromMcs (mcs,
rbgSize) / 8) / 0.001);   // = TB size / TTI
                        }
```

```
                        double rcqi = achievableRate; //CALCULOU  a taxa
máxima para esta UE (it)
                        NS_LOG_INFO (this << " RNTI " << (*it) << " MCS " <<
(uint32_t)mcs << " achievableRate " << achievableRate << " RCQI " << rcqi);

                        // Coloca taxa máxima atingível de cada UE no vetor
ueTaxamaxima
                        ueTaxaMaxima[(*it)]=rcqi;
                       /* if (rcqi > rcqiMax) // algoritmo de procura pelo
maior
                          {
                            rcqiMax = rcqi; // rcqiMax é a maior taxa
encontrada até o momento
                            itMax = it;     // itMax é a UE com maior taxa
encontrada até o momento.
                          }
                        */
                      }
                  }   // end if cqi

             } // end for m_rlcBufferReq

            // Inclui aqui: procura no vetor ueTaxamaxima
            rcqiMax=0;
             for (it = m_flowStatsDl.begin (); it != m_flowStatsDl.end ();
it++) //Varre todas as UEs que tem tráfego no downlink
             {
               if(ueTaxaReservadaSuavizada[(*it)]>600000) // 600kbytes por
seg, Verifica no vetor ueTaxaReservadaSuavizada se a taxa da itMax é maior
que o limite. Esse limite pode ser alterado para a taxa desejada.
                   continue; //UE acima do limite, pula para a próxima

               else // Menor que o limite, procura a próxima UE no vetor com
taxa menor que o limite
               {
                 if (ueTaxaMaxima[(*it)]>rcqiMax)
                 {
                   // Faz itMax = nova ue
                   rcqiMax=ueTaxaMaxima[(*it)]; // rcqiMax é a maior taxa
encontrada até o momento, dentro do limite
                   itMax = it;                  // itMax é o UE com maior
taxa encontrada até o momento, dentro do limite
                 }
               }
             }  //cálculo usando média móvel exponencial com alpha=0.2

            ueTaxaReservada[(*itMax)]+=rcqiMax;  //atualiza o vetor
ueTaxaReservada

            //atualização da média móvel exponencial será realizado uma vez
só e não cumulativamente

//ueTaxaReservadaSuavizada[(*itMax)]=0.8*ueTaxaReservadaSuavizada[(*itMax)]
+0.2*ueTaxaReservada[(*itMax)];  // atualiza o vetor ueTaxaReservada


          // Finalizou o "for" e econtrou a UE que pode transmitir com
maior taxa no RBG em questão (representado por it)
          if (itMax == m_flowStatsDl.end ()) // não encontrou nenhuma ue
querendo transmitir (rcqiMax vai ser 0)
```

```
            {
              // no UE available for this RB
              //NS_LOG_INFO (this << " any  found");
              std::cout << "\n RGB: "<< i << " - not alocated, no UE
available for this RB"; // modificado
            }
          else
            {
              rbgMap.at (i) = true; // começou a alocar o RGB i para a UE
itMax
              std::map <uint16_t, std::vector <uint16_t> >::iterator itMap;
              itMap = allocationMap.find ((*itMax));
              if (itMap == allocationMap.end ()) // primeira alocação de
RBG para este TTI para a UE itMax
                {
                  // insert new element
                  std::vector <uint16_t> tempMap;
                  tempMap.push_back (i);
                  allocationMap.insert (std::pair <uint16_t, std::vector
<uint16_t> > ((*itMax), tempMap)); // alocou RGB i para UE itMax
                }
              else  // já havia uma alocação de RGB para itMax neste TTI
                {
                  (*itMap).second.push_back (i);
                }
              //NS_LOG_INFO (this << " UE assigned " << (*itMax));
              std::cout << "\n RGB: "<< i << " - assigned to UE " <<
(*itMax) << "  TaxaReservada " << ueTaxaReservada[(*itMax)] << "
TaxaSuavizada " << ueTaxaReservadaSuavizada[(*itMax)]; // modificado

            }
        } // end for RBG free
        else
           std::cout << "\n RGB: " << i << " - assigned to HARQ or
something else";
    } // end for RBGs

   for (int i = 0; i != 1000; i++) {//Varre todas as UEs
        //cálculo usando média móvel exponencial com alpha=0.2
        // if (ueTaxaReservada[i]==0)

ueTaxaReservadaSuavizada[i]=0.9*ueTaxaReservadaSuavizada[i]+0.1*ueTaxaReser
vada[i];  // atualiza o vetor ueTaxaReservada
       //(debug) std::cout << "\n UE "<< i << " Taxa Reservada: " <<
ueTaxaReservada[i] << "Taxa Suavizada: " << ueTaxaReservadaSuavizada[i];
   }


  // generate the transmission opportunities by grouping the RBGs of the
same RNTI and
  // creating the correspondent DCIs
  std::map <uint16_t, std::vector <uint16_t> >::iterator itMap =
allocationMap.begin ();
  while (itMap != allocationMap.end ())
    {
      // create new BuildDataListElement_s for this LC
      BuildDataListElement_s newEl;
      newEl.m_rnti = (*itMap).first; // o Numero da UE
      // create the DlDciListElement_s
      DlDciListElement_s newDci;
      newDci.m_rnti = (*itMap).first;
```

```
      newDci.m_harqProcess = UpdateHarqProcessId ((*itMap).first);

      uint16_t lcActives = LcActivePerFlow ((*itMap).first);
      NS_LOG_INFO (this << "Allocate user " << newEl.m_rnti << " rbg " <<
lcActives);
      // std::cout << "\n Allocate user " << newEl.m_rnti << " rbg " <<
lcActives;
      if (lcActives == 0)
        {
          // Set to max value, to avoid divide by 0 below
          lcActives = (uint16_t)65535; // UINT16_MAX;
        }
      uint16_t RgbPerRnti = (*itMap).second.size ();
      std::map <uint16_t,SbMeasResult_s>::iterator itCqi;
      itCqi = m_a30CqiRxed.find ((*itMap).first);
      std::map <uint16_t,uint8_t>::iterator itTxMode;
      itTxMode = m_uesTxMode.find ((*itMap).first);
      if (itTxMode == m_uesTxMode.end ())
        {
          NS_FATAL_ERROR ("No Transmission Mode info on user " <<
(*itMap).first);
        }
      int nLayer = TransmissionModesLayers::TxMode2LayerNum
((*itTxMode).second);
      std::vector <uint8_t> worstCqi (2, 15);
      if (itCqi != m_a30CqiRxed.end ())
        {
          for (uint16_t k = 0; k < (*itMap).second.size (); k++)
            {
              if ((*itCqi).second.m_higherLayerSelected.size () >
(*itMap).second.at (k))
                {
                  NS_LOG_INFO (this << " RBG " << (*itMap).second.at (k) <<
" CQI " << (uint16_t)((*itCqi).second.m_higherLayerSelected.at
((*itMap).second.at (k)).m_sbCqi.at (0)) );
                //  std::cout <<"\n" << this << " RBG " <<
(*itMap).second.at (k) << " CQI " <<
(uint16_t)((*itCqi).second.m_higherLayerSelected.at ((*itMap).second.at
(k)).m_sbCqi.at (0));
                  for (uint8_t j = 0; j < nLayer; j++)
                    {
                      if ((*itCqi).second.m_higherLayerSelected.at
((*itMap).second.at (k)).m_sbCqi.size () > j)
                        {
                          if (((*itCqi).second.m_higherLayerSelected.at
((*itMap).second.at (k)).m_sbCqi.at (j)) < worstCqi.at (j))
                            {
                              worstCqi.at (j) =
((*itCqi).second.m_higherLayerSelected.at ((*itMap).second.at
(k)).m_sbCqi.at (j));
                            }
                        }
                      else
                        {
                          // no CQI for this layer of this suband -> worst
one
                          worstCqi.at (j) = 1;
                        }
                    }
                }
              else
```

```
                {
                  for (uint8_t j = 0; j < nLayer; j++)
                    {
                      worstCqi.at (j) = 1; // try with lowest MCS in RBG
with no info on channel
                    }
                }
            }
        }
      else
        {
          for (uint8_t j = 0; j < nLayer; j++)
            {
              worstCqi.at (j) = 1; // try with lowest MCS in RBG with no
info on channel
            }
        }
      for (uint8_t j = 0; j < nLayer; j++)
        {
          NS_LOG_INFO (this << " Layer " << (uint16_t)j << " CQI selected "
<< (uint16_t)worstCqi.at (j));
         // std::cout << "\n" << this << " Layer " << (uint16_t)j << " CQI
selected " << (uint16_t)worstCqi.at (j);
        }
      uint32_t bytesTxed = 0;
      for (uint8_t j = 0; j < nLayer; j++)
        {
          newDci.m_mcs.push_back (m_amc->GetMcsFromCqi (worstCqi.at (j)));
          int tbSize = (m_amc->GetTbSizeFromMcs (newDci.m_mcs.at (j),
RgbPerRnti * rbgSize) / 8); // (size of TB in bytes according to table
7.1.7.2.1-1 of 36.213)
          newDci.m_tbsSize.push_back (tbSize);
          NS_LOG_INFO (this << " Layer " << (uint16_t)j << " MCS selected"
<< m_amc->GetMcsFromCqi (worstCqi.at (j)));
        //  std::cout << "\n" << this << " Layer " << (uint16_t)j << " MCS
selected" << m_amc->GetMcsFromCqi (worstCqi.at (j));
          bytesTxed += tbSize;
        }

      newDci.m_resAlloc = 0;  // only allocation type 0 at this stage
      newDci.m_rbBitmap = 0; // TBD (32 bit bitmap see 7.1.6 of 36.213)
      // VER ESSE RBG MASK **********************
      uint32_t rbgMask = 0;
      for (uint16_t k = 0; k < (*itMap).second.size (); k++)
        {
          rbgMask = rbgMask + (0x1 << (*itMap).second.at (k));
          NS_LOG_INFO (this << " Allocated RBG " << (*itMap).second.at
(k));
        //  std::cout << "\n" << this << " Allocated RBG " <<
(*itMap).second.at (k);
        }
      newDci.m_rbBitmap = rbgMask; // (32 bit bitmap see 7.1.6 of 36.213)

      // create the rlc PDUs -> equally divide resources among actives LCs
      std::map <LteFlowId_t,
FfMacSchedSapProvider::SchedDlRlcBufferReqParameters>::iterator itBufReq;
      for (itBufReq = m_rlcBufferReq.begin (); itBufReq !=
m_rlcBufferReq.end (); itBufReq++)
        {
          if (((*itBufReq).first.m_rnti == (*itMap).first)
              && (((*itBufReq).second.m_rlcTransmissionQueueSize > 0)
```

```
                    || ((*itBufReq).second.m_rlcRetransmissionQueueSize > 0)
                    || ((*itBufReq).second.m_rlcStatusPduSize > 0) ))
              {
                std::vector <struct RlcPduListElement_s> newRlcPduLe;
                for (uint8_t j = 0; j < nLayer; j++)
                  {
                    RlcPduListElement_s newRlcEl;
                    newRlcEl.m_logicalChannelIdentity =
(*itBufReq).first.m_lcId;
                    newRlcEl.m_size = newDci.m_tbsSize.at (j) / lcActives;
                    NS_LOG_INFO (this << " LCID " << (uint32_t)
newRlcEl.m_logicalChannelIdentity << " size " << newRlcEl.m_size << " layer
" << (uint16_t)j);
                    newRlcPduLe.push_back (newRlcEl);
                    UpdateDlRlcBufferInfo (newDci.m_rnti,
newRlcEl.m_logicalChannelIdentity, newRlcEl.m_size);
                    if (m_harqOn == true)
                      {
                        // store RLC PDU list for HARQ
                        std::map <uint16_t,
DlHarqRlcPduListBuffer_t>::iterator itRlcPdu =
m_dlHarqProcessesRlcPduListBuffer.find ((*itMap).first);
                        if (itRlcPdu == m_dlHarqProcessesRlcPduListBuffer.end
())
                          {
                            NS_FATAL_ERROR ("Unable to find RlcPdcList in
HARQ buffer for RNTI " << (*itMap).first);
                          }
                        (*itRlcPdu).second.at (j).at
(newDci.m_harqProcess).push_back (newRlcEl);
                      }
                  }
                newEl.m_rlcPduList.push_back (newRlcPduLe);
              }
            if ((*itBufReq).first.m_rnti > (*itMap).first)
              {
                break;
              }
          }
        for (uint8_t j = 0; j < nLayer; j++)
          {
            newDci.m_ndi.push_back (1);
            newDci.m_rv.push_back (0);
          }

        newDci.m_tpc = 1; //1 is mapped to 0 in Accumulated Mode and to -1 in
Absolute Mode

        newEl.m_dci = newDci;

        if (m_harqOn == true)
          {
            // store DCI for HARQ
            std::map <uint16_t, DlHarqProcessesDciBuffer_t>::iterator itDci =
m_dlHarqProcessesDciBuffer.find (newEl.m_rnti);
            if (itDci == m_dlHarqProcessesDciBuffer.end ())
              {
                NS_FATAL_ERROR ("Unable to find RNTI entry in DCI HARQ buffer
for RNTI " << newEl.m_rnti);
              }
            (*itDci).second.at (newDci.m_harqProcess) = newDci;
```

```
          // refresh timer
          std::map <uint16_t, DlHarqProcessesTimer_t>::iterator itHarqTimer
=  m_dlHarqProcessesTimer.find (newEl.m_rnti);
          if (itHarqTimer== m_dlHarqProcessesTimer.end ())
            {
              NS_FATAL_ERROR ("Unable to find HARQ timer for RNTI " <<
(uint16_t)newEl.m_rnti);
            }
          (*itHarqTimer).second.at (newDci.m_harqProcess) = 0;
        }


      ret.m_buildDataList.push_back (newEl);

      itMap++;
    } // end while allocation
  ret.m_nrOfPdcchOfdmSymbols = 1;

  m_schedSapUser->SchedDlConfigInd (ret);


  return;
}
```