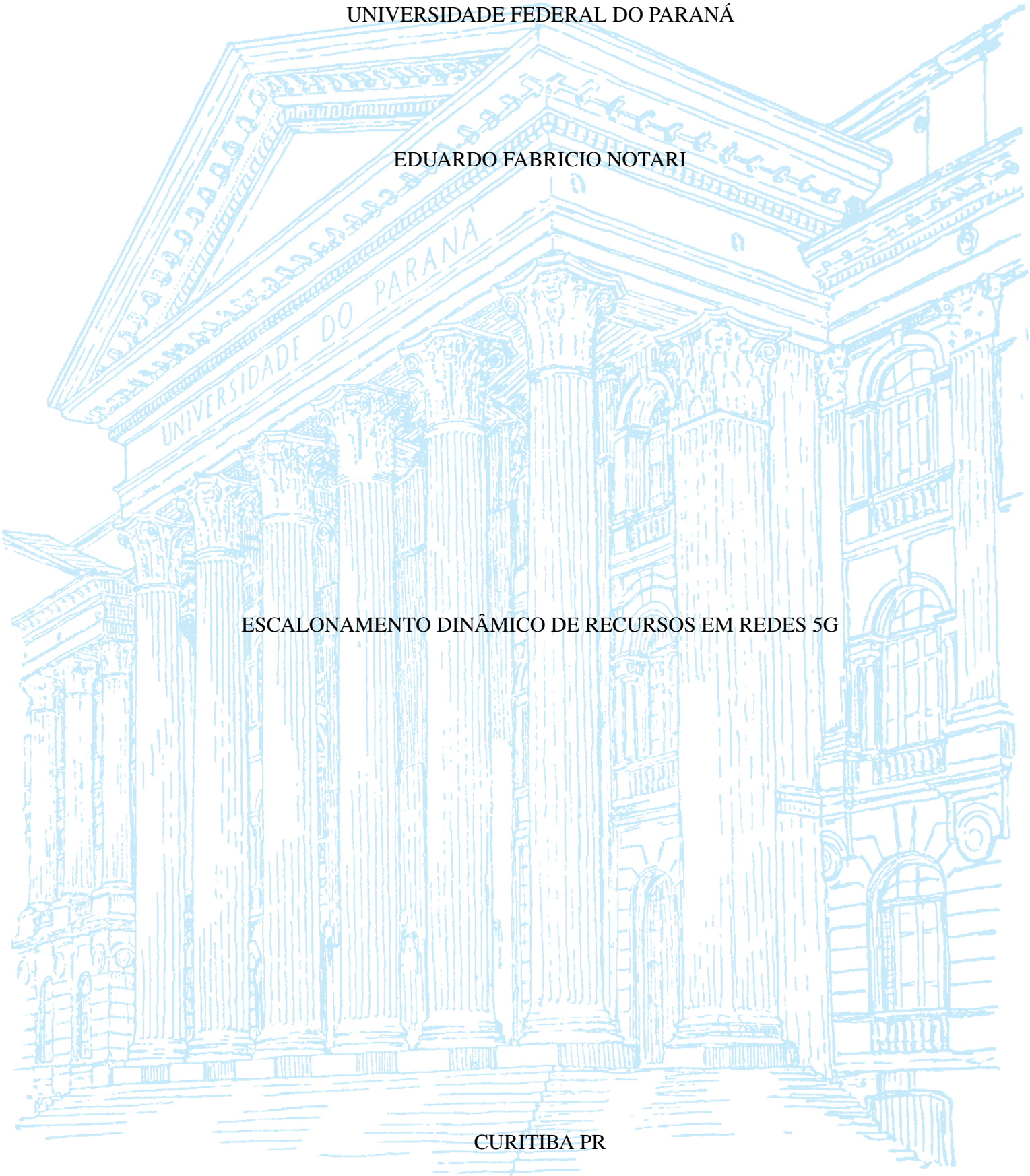
UNIVERSIDADE FEDERAL DO PARANÁ

EDUARDO FABRICIO NOTARI



ESCALONAMENTO DINÂMICO DE RECURSOS EM REDES 5G

CURITIBA PR

2024

EDUARDO FABRICIO NOTARI

ESCALONAMENTO DINÂMICO DE RECURSOS EM REDES 5G

Tese apresentada como requisito parcial à obtenção do grau de Mestre em Engenharia Elétrica no Programa de Pós-Graduação em Engenharia Elétrica, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Engenharia Elétrica, Telecomunicações, Sistemas de Comunicação*.

Orientador: Dr. Carlos Marcelo Pedroso.

CURITIBA PR

2024

Ficha catalográfica

Substituir o arquivo `0-iniciais/catalografica.pdf` pela ficha catalográfica fornecida pela Biblioteca da UFPR (PDF em formato A4).

Instruções para obter a ficha catalográfica e fazer o depósito legal da tese/dissertação (contribuição de André Hochuli, abril 2019. Links atualizados Wellton Costa, Nov 2022):

1. Estas instruções se aplicam a dissertações de mestrado e teses de doutorado. Trabalhos de conclusão de curso de graduação e textos de qualificação não precisam segui-las.
2. Verificar se está usando a versão mais recente do modelo do PPGInf e atualizar, se for necessário (`https://gitlab.c3sl.ufpr.br/maziero/tese`).
3. conferir o *checklist* de formato do Sistema de Bibliotecas da UFPR, em `https://bibliotecas.ufpr.br/servicos/normalizacao/`
4. Enviar e-mail para "`referencia.bct@ufpr.br`" com o arquivo PDF da dissertação/tese, solicitando a respectiva ficha catalográfica.
5. Ao receber a ficha, inseri-la em seu documento (substituir o arquivo `0-iniciais/catalografica.pdf` do diretório do modelo).
6. Emitir a Certidão Negativa (CND) de débito junto a biblioteca, em `https://bibliotecas.ufpr.br/servicos/certidao-negativa/`
7. Avisar a secretaria do PPGInf que você está pronto para o depósito. Eles irão mudar sua titulação no SIGA, o que irá liberar uma opção no SIGA pra você fazer o depósito legal.
8. Acesse o SIGA (`http://www.prppg.ufpr.br/siga`) e preencha com cuidado os dados solicitados para o depósito da tese.
9. Aguarde a confirmação da Biblioteca.
10. Após a aprovação do pedido, informe a secretaria do PPGInf que a dissertação/tese foi depositada pela biblioteca. Será então liberado no SIGA um link para a confirmação dos dados para a emissão do diploma.

Ficha de aprovação

Substituir o arquivo 0-iniciais/aprovacao.pdf pela ficha de aprovação fornecida pela secretaria do programa, em formato PDF A4.

It has to start somewhere, it has to start sometime, what better place than here, what better time than now!
RATM

AGRADECIMENTOS

Ao Professor Doutor Carlos Marcelo Pedroso, meu orientador, pelos conhecimentos adquiridos e oportunidade de conhecer e desenvolver o meio científico, participando de forma ativa na contriubuição e progresso da ciência.

RESUMO

A rede 5G *New Radio* foi desenvolvida para suportar aplicações que não eram atendidas pela geração anterior da tecnologia de comunicações móveis. Particularmente, a rede 4G LTE (*Long Term Evolution*) não suportava aplicações da Internet das Coisas (*Internet of Things*, IoT) por causa do limite no número de UEs (*User Equipments*), da limitação de banda e da quantidade mínima de recursos alocados para uma UE na área da cobertura da estação rádio-base. O *massive Machine Type Communication* (mMTC), tem se destacado habilitando IoT na indústria atual. Esta tecnologia utiliza da automação e inteligência para coletar dados, e a partir dessas informações, tomar decisões que tragam benefícios aos usuários finais. Aplicações mMTC normalmente exigem baixo atraso de propagação, baixa probabilidade de perda de pacotes e uma alta densidade de dispositivos. No entanto, a taxa de transmissão exigida por um dispositivo mMTC é muito baixa. Para atender estes requerimentos, é necessário distribuir recursos escassos do espectro, distribuídos pela estação rádio-base, e priorizar o mMTC sobre as demais aplicações existentes na rede. Ao mesmo tempo, a rede tem que atender aplicações que exigem a transmissão de grandes volumes de informações, principalmente com transmissão de vídeo. A convivência destes dois tipos e aplicações é muito difícil, uma vez que os requisitos envolvidos são opostos entre eles. Neste estudo foi proposto o Algoritmo de Escalonamento com Prioridade Dinâmica *"Dynamic Priority Scheduler"*, capaz de identificar e priorizar a transmissão de usuários mMTC, trazendo uma melhoria na utilização desse serviço quando concorrendo com o tráfego de aplicações do tipo *enhanced Mobile BroadBand* (eMBB). Os resultados obtidos mostram que o método proposto identifica aplicações mMTC e efetivamente tem sucesso em priorizar o tráfego mMTC sobre o eMBB, permitindo que um maior número de aplicações mMTC possam existir na rede mesmo quando há concorrência com aplicações eMBB.

Palavras-chave: Redes Móveis 1. 5G NR 2. Escalonamento 3. Internet das Coisas 4.

ABSTRACT

The 5G New Radio network was developed to support applications not supported by the previous generation of mobile communications technology. Particularly, the 4G LTE (Long Term Evolution) network did not support Internet of Things (IoT) applications because of the limit on the number of UEs (User Equipments), bandwidth limitation. Supporting the massive Machine Type Communication (mMTC) is necessary for enabling IoT in current industry. This technology uses automation and intelligence to collect data, and based on this information, make decisions that bring benefits to end users. mMTC applications typically require low propagation delay, low packet loss probability, and a high device density. However, the transmission rate required by a mMTC device is very low. To meet these requirements, it is necessary to distribute scarce spectrum resources, distributed by the base station and prioritize the choice of mMTC over other applications on the network. At same time, the network has to serve applications that require transmission of large volumes of information, especially video streaming. The coexistence of these two types and applications is very difficult, since the requirements involved are opposites to each other. In this dissertation, a new resource scheduler algorithm called Dynamic Priority Scheduler (DPS) is proposed. The DPS is capable of identifying and prioritizing the transmission of mMTC users, bringing an improvement in the use of this service when competing with the traffic of enhanced Mobile BroadBand (eMBB) applications. The simulated results show that the proposed method identifies mMTC applications and effectively succeeds in prioritizing mMTC traffic over eMBB, allowing a greater number of mMTC applications to exist in the network when there is competition with eMBB applications.

Keywords: Mobile Networks 1. 5G 2. Scheduler 3. Internet of Things 4.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

3GPP	*Third Generation Partnership Project*
5G NR	*5th Generation New Radio*
BPSK	*Binary Phase Shift Keying*
BWP	*Bandwidth*
CQI	*Channel Quality Indicator*
D2D	*Device-to-Device Communication*
DF	Domínio da Frequência
DPS	*Dynamic Priority Scheduler*
DQN	*Deep Q Network*
DT	Domínio do Tempo
DUD	Distribuição Uniforme Dinâmica
EE	*Energy Efficieny*
eMBB	*enhanced Mobile Broadband*
FR	*Frequency Range*
gNodeB	Estação Rádio-base
ICR	*Interference Contribution Ratio*
IMT	*International Mobile Telecommunications*
IoT	*Internet of Things*
ITU	*International Telecommunication Union*
LTE	*Long Term Evolution*
M2M	*Machine-to-Machine Communication*
MCS	*Modulation and Code Scheme*
MDP	*Markov Decision Process*
mMTC	*massive Machine Type Communication*
MT	*Maximum Throuthput*
N-PF	*Neighbors-Aware Proportional Fair*
NS3	*Network Simulator*
OFDMA	*Orthogonal Frequency-Division Multiple Access*
PF	*Proportional Fair*
PRACH	*Physical Random Access Channel*
QAM	*Quadrature Amplitude Modulation*
QoE	*Quality of Experience*
QoS	*Quality of Service*
QPSK	*Quadrature Phase Shift Keying*
RA	*Random Access*

RAN	*Radio Access Network*
RB	*Resource Block*
RE	*Resource Element*
RMa_LOS	*Rural Macrocell Path Loss Model*
RR	*Round Robin*
RRM	*Radio Resource Management*
SE	*Spectral Efficiency*
SINR	*Signal-Noise plus Interference Rate*
TTI	*Time Transmission Interval*
UE	*User Equipment*
UMTS	*Universal Mobile Telecommunication System*
uRLLC	*ultra-Realiable Low-Latency Communication*

LISTA DE SÍMBOLOS

γ	Parâmetro da média móvel exponencial da taxa de transmissão de uma UE.
H	Parâmetro de Hurst.
D	Variável de decremento da prioridade.
I	Variável de incremento da prioridade.
R_i	Média móvel exponencial ponderada.
r_i	Taxa de transmissão instantânea.
$\text{SNIR}_k^i(t)$	Razão sinal ruído-mais-interferência da UE_i no RB_k no tempo t.
t	Variável temporal.
T_i	Tempo do último instante que a UE_i recebeu alocação.
P_i	Prioridade dinâmica da UE_i.
α	Parâmetro de forma da distribuição de Pareto
β	Parâmetro de escala da distribuição de Pareto
μ	Taxa média da distribuição Exponencial
γ	Taxa de transmissão em quadros por segundo.
J	Índice de justiça de Jain
ι	Variável Jitter

SUMÁRIO

1 INTRODUÇÃO

Com a evolução da indústria e a necessidade de atender a demanda das novas aplicações, desenvolveu-se o *5G New Radio* (5G NR), projetado para atender um número massivo de dispositivos, suportar as tecnologias anteriores e ter um melhor aproveitamento dos recursos disponibilizados pela estação rádio-base. Suporta aplicações como *enhanced Mobile Broadband* (eMBB) proveniente do *Long Term Evolution* (LTE) e habilita novas aplicações como o *massive Machine Type Communication* (mMTC) e aplicações de missão-crítica com requisitos exigentes chamado *ultra-Realiable Low-Latency Communication* (uRLLC) [Sauter, 2021].

Uma das motivações para o desenvolvimento do 5G NR foi a necessidade da troca de informação entre máquinas usando comunicações *Machine-to-Machine* (M2M) ou *Device-to-Device* (D2D) para suportar aplicações da Internet das Coisas (IoT), e também serviços críticos como telemedicina, internet táctil, carros autônomos, realidade virtual e realidade aumentada, dentre outras [ROHDESCHWARZ, 2023]. A transmissão deve ser confiável a ponto de executar tarefas remotamente, coletar informações e possivelmente utilizar da Inteligência Artificial para tomada de decisões, oferecendo melhores serviços.

A característica típica da comunicação M2M envolve a transmissão de pequenos pacotes de dados, de forma periódica ou assíncrona. Porém, dependem do alcance proporcionado pela rede móvel e tempo de resposta suficiente para atender os requisitos básicos para a operação da solução. Esses dados competem com o tráfego existente na rede, exigindo uma configuração de rede e métodos que organizem a transmissão de maneira a obter o melhor aproveitamento dos recursos existentes.

O sistema 5G NR utiliza *frames* de transmissão de período 10 ms subdivididos em *subframes* de 1 ms que, como no sistema LTE, é o menor intervalo de transmissão e é chamado de *TTI (Time Transmission Interval)*. Introduzido na Release-15 do *Third Generation Partnership Project* (3GPP), o conceito de numerologia possibilita uma janela de transmissão menor se comparado com o LTE. Desta maneira, o 5G permite atrasos menores e habilita o suporte à applicações críticas.

As informações são transmitidas para o equipamento do usuário (*User Equipment, UE*), através de esquemas de modulação definidos pelo 3GPP [3GPP, 2019] de acordo com a estimativa da relação sinal-ruído do canal. A modulação trabalha com símbolos que representam de 1 a 8 bits, que são transportados no *Resource Block* (RB), que é o menor elemento de alocação de recursos na rede de rádio. Em cada TTI, dois RBs carregam 14 símbolos em 12 subportadoras de largura de banda variável definida pela numerologia escolhida. O 5G utiliza o *Orthogonal Frequency-Division Multiple Access* (OFDMA) que organiza as subportadoras de cada banda de maneira ortogonal, evitando a interferência intercanal. O número de RBs transmitidos em um TTI varia com a largura de banda, a numerologia e a utilização de banda de guarda.

De forma a transmitir os dados dos usuários, as UEs competem pelos RBs disponíveis, que são alocados pela estação rádio-base (chamada de gNodeB). Assim são implementados algoritmos escalonadores que determinam como os RBs serão distribuídos para as UEs. Cada tipo de aplicação possui uma característica específica, assim sendo, cada tipo de aplicação tem um algoritmo que melhor se adapta àquele tipo de transmissão, trazendo uma melhor experiência ao usuário final. Os algoritmos escalonadores usam de técnicas diferenciadas, e até mesmo inteligência artificial para seu propósito. Pode-se computar informações do sistema, como por exemplo, o indicador da qualidade do canal ou a necessidade da qualidade de serviço para

alocação dos RBs aos UEs para obter o resultado desejado e até mesmo aumentar a capacidade de usuários no sistema.

De acordo com Cappozi et al. [Capozzi et al., 2013], os escalonadores podem ser classificados de acordo com a seguinte estratégia de abordagem:

- Insensíveis ao canal (*channel-unaware*);
- Sensíveis ao canal e insensíveis à qualidade de serviço (*channel-aware/QoS-unaware*), considerando os índices de qualidade do canal, desprezando a qualidade de serviço;
- Sensíveis ao canal e sensíveis à qualidade de serviço (*channel-aware/QoS-aware*), considerando à qualidade do canal e de serviço;
- Semi-persistente (*semi-persistent*), preservando a taxa alocada pelo tempo previsto de duração da sessão, usado na transmissão de voz;
- Sensíveis a energia (*energy-aware*), considerando os dados da potência utilizada com a finalidade de economizar a bateria do dispositivo.

Os algoritmos de escalonamento mais usados são o *Round Robin* (RR), o *Proportional Fair* (PF) e o *Maximum Throughput* (MT) [Maryam Imran Sheik Mamode, 2020]. O RR distribui os RBs disponíveis de forma sequencial sem considerar a qualidade de canal de cada UE. O PF considera a qualidade de canal dos UEs e procura distribuir os RBs de forma que todas as UEs tenham *throughput* semelhante, auxiliando UEs que possuem baixa qualidade de sinal da estação rádio-base e distribuindo os recursos de uma maneira mais "justa". O MT provê maior *throughput* médio para a célula rádio, priorizando elementos com melhor qualidade de sinal da rede rádio. Este algoritmo pode não dar oportunidade aos elementos com baixa qualidade de canal a transmitirem os seus dados, pois os elementos com alta qualidade de canal podem utilizar todos os recursos existentes.

O 5G NR é capaz de separar a largura de banda disponível em diversas partições (*Bandwidth Partitions*, BWP) de forma a aplicar configuração e métodos para suportar diferentes aplicações em cada BWP. No entanto, a situação de uso mais comum é a configuração de um único BWP, uma vez que a separação da largura de banda disponível implica também em redução de capacidade do BWP dividido. Para que o uso de mais de um BWP seja economicamente viável, devem existir clientes arcando com os custos da alocação da banda para aplicações específicas.

As principais aplicações suportadas no 5G NR são o eMBB, que envolve aplicações que necessitam de grandes taxas de transmissão (como vídeo *streaming*), o mMTC, que representa aplicações IoT e D2D, e aplicações uRLLC [Fehrenbach et al., 2018], que é a categoria de aplicações com exigências estritas de atraso, *jitter*, perda e taxa de transmissão. As aplicações uRLLC necessitam de escalonadores com suporte à qualidade de serviço e possivelmente um BWP com configuração específica para permitir seu correto funcionamento.

Neste trabalho será abordado um cenário onde em um BWP existem aplicações eMBB e mMTC competindo por recursos. Esta é uma situação típica de convivência de aplicações, em um cenário de uso que pode ser necessário o uso do 5G NR sem a necessidade da aplicação de mecanismos de qualidade de serviço, ou classificação de tráfego, o que exigiria a identificação prévia dos UEs gerando cada tipo de aplicação. A proposta deste trabalho é desenvolver um escalonador de recursos capaz de identificar e priorizar a alocação de recursos para aplicações mMTC em detrimento das aplicações eMBB, de forma automática.

Além desta seção introdutória, esta dissertação está dividida em seis capítulos. No Capítulo 2, são apresentados os objetivos gerais e específicos da pesquisa e a metodologia

utilizada. Em seguida, o Capítulo 3 apresenta os tópicos relevantes sobre a tecnologia de rádio móvel 5G NR, normalização, características, tipos de dispositivos e transmissões utilizadas, assim como as restrições de operação de cada tipo, escalonadores concorrentes e informações sobre o simulador utilizado. No Capítulo 4, é apresentado o método proposto e sua implementação, com a avaliação de desempenho realizada através de simulações computacionais sendo apresentadas no Capítulo 5. As conclusões são apresentadas no Capítulo 6.

2 OBJETIVOS E METODOLOGIA

2.1 OBJETIVOS

O objetivo geral é desenvolver um algoritmo escalonador sensível ao canal e insensível a qualidade de serviço capaz de detectar automaticamente e priorizar a transmissão de aplicações mMTC em redes 5G em situações onde há concorrência entre aplicações eMBB e mMTC na mesma célula de rádio.

São objetivos específicos deste projeto:

- Levantar os métodos disponíveis de escalonamento em redes 5G NR para aplicações eMBB e mMTC.
- Projetar o algoritmo escalonador.
- Implementar o escalonador desenvolvido no simulador NS-3.
- Validar a implementação do método no simulador NS-3.
- Realizar a análise do desempenho do método proposto através de simulações computacionais.
- Comparar o desempenho do método proposto com demais métodos disponíveis.

2.2 METODOLOGIA

O desenvolvimento do novo método de escalonamento foi realizado através de pesquisa sobre os métodos de escalonamento dinâmico de recursos na área de Sistemas Operacionais e a sua possibilidade de adaptação para o uso no cenário de redes 5G com aplicações tipo mMTC e eMBB concorrentes. Foram pesquisados algoritmos escalonadores que têm como objetivo a convivência entre as aplicações eMBB e mMTC e suas formas de implementação e avaliação dos resultados. O projeto e a especificação do modelo proposto foi realizado através de equações algébricas e algoritmo.

A avaliação de desempenho foi realizada através de simulações computacionais utilizando o simulador NS-3 [Patriciello et al., 2019]. O NS-3 é um simulador baseado em eventos discretos, com código fonte aberto e largamente utilizado para desenvolvimento de pesquisa na área de redes e sistemas de comunicação [Riley e Henderson, 2010]. O módulo 5G-LENA desenvolvido pelo *"Centre Tecnològic de Telecomunicacions de Catalunya"* possibilita a simulação de redes 5G no NS-3 [5G LENA, 2023]. Na simulação foi utilizado um servidor AMD 5950x de 16 processadores, 64 GB de memória RAM trabalhando em *multi-threading*, com sistema operacional linux Fedora.

O método proposto foi implementado em linguagem de programação C++ e incorporado ao 5G-LENA.

Para validar o método, foram realizados testes de execução, utilização do simulador para a verificação do código e também a verificação se os resultados condiziam com o esperado pelo programa.

Foi definida uma topologia de simulação que demonstre o funcionamento de uma rede móvel 5G com suas características próximas da real, para que traga resultados relevantes e

generalizáveis. Nas simulações foi utilizado um cenário com uma célula e características de ambiente rural. Na rede, há um servidor de aplicação que transmite dados para as UEs conectados à rede móvel. A simulação utiliza o *downlink* para avaliação do funcionamento do algoritmo escalonador.

Nas simulações, devem haver UEs gerando tráfego eMBB e mMTC. A aplicação eMBB escolhida imita uma transmissão de vídeo *streaming* que gera tráfego com características auto-similares através do modelo ON-OFF com o estado ON modelado por uma Distribuição de Pareto e estado OFF com uma Distribuição Exponencial. O número de fontes de tráfego aumentou até que a rede ficou congestionada e, assim, observou-se o desempenho do método proposto que foi comparado com os métodos de referência. As métricas de desempenho avaliadas foram o atraso fim-a-fim, o *jitter*, o *throughput* e a perda de pacotes.

O *jitter* é a variação no atraso de pacotes que são transmitidos por uma rede de dados. Os fatores que podem influenciar são o congestionamento da rede, a diferença nas rotas dos pacotes ou as variações na carga de trabalho dos elementos da rede. A RFC 8238 define o *jitter* como

$$\iota(i) = \iota(i-1) + (|D(i-1,i)| - \iota(i-1))/16 \tag{2.1}$$

onde ι é o *jitter* é D é o atraso no instante i [Avramov, 2017].

A perda de pacotes ocorre quando um ou mais pacotes de dados não chegam ao seu destino. A perda de pacotes é ocasionada por erros na transmissão ou congestionamento na rede. É calculada através da razão entre pacotes perdidos e pacotes enviados. Em aplicações de tempo-real como vídeo *streaming*, a perda de pacotes afeta a qualidade da experiência (QoE) do usuário [Molisch, 2012].

Além disso, foi avaliada a justiça na distribuição de recursos e determinou-se uma diferença significativa de desempenho entre serviços dependendo do algoritmo de escalonamento usado. Foi utilizado o índice de justiça de Jain [Jain, 1991], definido por

$$J = \frac{(\sum_{j=1}^{n} x_j)^2}{n \sum_{j=1}^{n} x_j^2} \tag{2.2}$$

onde J é o índice de justiça, n é o número de medidas e x_j representa uma medida. O valor de J deve estar entre 0 e 1, sendo 1 representando a maior justiça possível e 0 a menor.

O atraso fim-a-fim, é o intervalo de tempo entre a transmissão e a respectiva recepção de um pacote. O atraso fim-a-fim é importante para que se meça a qualidade de serviço oferecido. Ele é influenciado por fatores como tempo de processamento, tamanho de filas nos *buffers* dos equipamentos de rede e largura de banda nos meios de transmissão. Quanto menor o atraso melhor a qualidade da rede. Certas aplicações dependem desta performance para que seja viável o seu funcionamento.

O *throughput* é uma medida da quantidade de dados que podem ser transmitidos e recebidos através de uma rede em um determinado período de tempo, expressa em bits por segundo (bps). O *throughput* é um indicador importante do desempenho de um aplicativo de rede, pois reflete a eficiência com que os dados são transferidos entre diferentes dispositivos ou pontos na rede. Ela pode ser afetada por diversos fatores, incluindo a largura de banda da rede, o atraso fim-a-fim, a qualidade da conexão, a capacidade do hardware e a sobrecarga do protocolo de comunicação.

3 5G NEW RADIO

O 4G LTE, quarta geração de telefonia móvel sem fio, foi desenvolvido inicialmente para prover acesso internet de alta velocidade aos seus usuários finais, para além das capacidades do 3G *Universal Mobile Telecommunication System* (UMTS), terceira geração de telefonia móvel sem fio [Sauter, 2021]. Seguindo a evolução, na quinta geração de redes de telefonia móvel sem fio 5G NR, implementou-se a flexibilização dos padrões fixos existentes no 4G LTE, possibilitando maiores taxas de transmissão e suporte a aplicação com requisitos de baixo atraso, além de suportar maior densidade de dispositivos na rede rádio [Dahlman et al., 2018]. Também se introduziu o conceito de numerologia, onde cada frequência da largura de banda da subportadora dentro de um RB pode ser multiplicada pela numerologia, dividindo em partes menores o intervalo de transmissão para este elemento. Deste modo, adequou-se os sistemas existentes para as novas aplicações como IoT, mMTC ou uRLLC.

Essas aplicações que utilizam a comunicação M2M como *wearables*, sensores industriais e aplicações residenciais, possuem características diferentes que a utilizada na navegação da internet ou vídeo *streaming*. Além de ter um número massivo de usuários conectados, as aplicações M2M tem como característica transmitir pacotes de dados com pequeno tamanho em um espaço curto de tempo. No LTE, este tipo de transmissção tem dois problemas. O primeiro é a concorrência pelo canal de acesso aleatório (PRACH, *Physical Random Access Channel*), que é usado pelas UEs para solicitar a alocação de recursos pela eNodeB. O controle de acesso a este canal segue o método *Sloted Aloha*, o que aumenta muito a probabilidade da colisão conforme o aumento no número de UEs na rede. O segundo é a granularidade da alocação, que é de no mínimo de 1 RB (12 portadoras com 14 símbolos em 1 ms). Um dispositivo IoT pode não utilizar a capacidade toda do RB.

Verificados os problemas encontrados, o 3GPP desenvolveu uma evolução dos conceitos existentes que pudesse atender os requisitos das novas soluções e possibilite integrar futuras outras tecnologias a medida que elas surjam. O 5G NR possui taxa de *download* que pode chegar a 10 Gbps, utilizando larguras de banda maiores e em frequências mais altas, se comparado ao LTE [Zaidi et al., 2018]. Porém, a utilização de altas frequências tornam o sistema mais suscetível a interferências e diminui o alcance devido à propagação das ondas eletromagnéticas. Portanto as células de cada estação rádio-base são menores, apesar de suportar maior número de usuários. Utilizando as chamadas **ondas milimétricas** (mmWave) terão alcance de centenas de metros, permitindo maiores velocidades de transmissão num ambiente de uma fábrica, por exemplo.

3.1 NORMALIZAÇÃO

O 3GPP é composto por um número de organizações padronizadoras que desenvolvem protocolos para telecomunicações móveis sem fio [3GPP, 2020]. Depois do 4G LTE implementado, o 3GPP iniciou uma série de discussões sobre a geração seguinte. Levaram cerca de dois anos e muitos relatórios técnicos com diversas análises das propostas para o 5G NR, até que em 2008, os conjuntos de relatórios se tornaram a especificação técnica *3GPP Release 15*. O principal documento é o 3GPP TS38.401 [3GPP, 2019], que descreve de uma maneira geral o 5G RAN (*Radio Access Network*). Muitos outros documentos compõem as séries 38 que detalham como o 5G NR deve ser implementado.

Estes padrões devem ser seguidos para que diferentes fabricantes e novas tecnologias possam ser incluídas nos sistemas existentes. O 3GPP define que uma nova geração de telefonia móvel deve suportar as tecnologias anteriores e ser flexível a ponto de que suporte novas funcionalidades e soluções para futuros problemas que possam ser introduzidas com facilidade, sem a necessidade de grandes alterações de hardware e/ou configurações da rede já em operação.

3.2 FREQUÊNCIAS UTILIZADAS

O 3GPP definiu faixas três faixas de frequência para o NR. Para frequências de 410 MHz à 7,125 MHz a *Frequency Range 1* (FR1); de 24,250 MHz à 52,600 MHz, a *Frequency Range 2-1*; e de 52,600 MHz à 71,000 MHz, a *Frequency Range 2-2* [3GPP, 2018b]. Na faixa FR-2 estão as chamadas ondas milimétricas (mmWave), de acordo com a Tabela 3.1. A vantagem do mmWave é a maior largura de banda máxima a qual é de 400 MHz, enquanto na FR1 o máximo é de 100 MHz. A desvantagem do FR2 é o alcance, que é afetado por obstáculos.

Nome	Frequência
FR1	410 MHz - 7,125 MHz
FR2-1	24,250 MHz - 52,600 MHz
FR2-2	52,600 MHz 71,000 MHz

Tabela 3.1: Ranges de frequências definidas pelo 3GPP.

3.3 RADIO ACCESS NETWORK

A interface rádio RAN é especificada pelo padrão TS 38.300 [3GPP, 2018a], e se baseia no uso do OFDMA para o *Downlink* e *Uplink*. O controle de acesso é realizado de forma centralizada pela gNodeB.

3.4 CAMADA FÍSICA

Na interface de rádio do 5G NR, um *slot* consiste em um RB com 14 símbolos OFDM, com a palavra código variando a quantidade de bits transmitida, de acordo com a modulação configurada pela análise da resposta da relação **sinal-ruído mais interferência** (SINR), reportada pelo *Channel Quality Indicator* (CQI). Cada símbolo, segundo a modulação empregada, transporta de um a oito bits por símbolo [Abu-Rgheff, 2019], conforme informado na Tabela 3.2.

Tipo de Modulação	Bits por Símbolo
BPSK (Binary Phase Shift Keying)	1
QPSK (Quadrature Phase Shift Keying)	2
16QAM (Quadrature Amplitude Modulation)	4
64QAM	6
256QAM	8

Tabela 3.2: Esquemas de modulação para o 5G NR [Abu-Rgheff, 2019].

De acordo com Dahlman etal. [Dahlman et al., 2018], o tempo de transmissão do *slot* depende da numerologia empregada. Ela também permite a utilização de aplicações de baixa

latência, dividindo os *slots* em *minislots*. O tempo de *subframe* e *frame* no 5G NR é fixo em 1 ms e 10 ms, respectivamente. Cada divisão vertical ilustrada na Figura 3.1 indica um símbolo OFDM. É possível visualizar que na numerologia 0, 14 símbolos ocupam uma largura de banda de 15 kHz em 1 milisegundo (destaque em cinza escuro na figura). Já na numerologia 1, 14 símbolos ocupam 30 kHz em 0,5 milisegundos, e assim por diante. O número de *slots* dentro de um *subframe* é variável e depende da largura de banda da portadora definida através da numerologia utilizada. A Tabela 3.3 mostra as diferentes numerologias possíveis no NR.

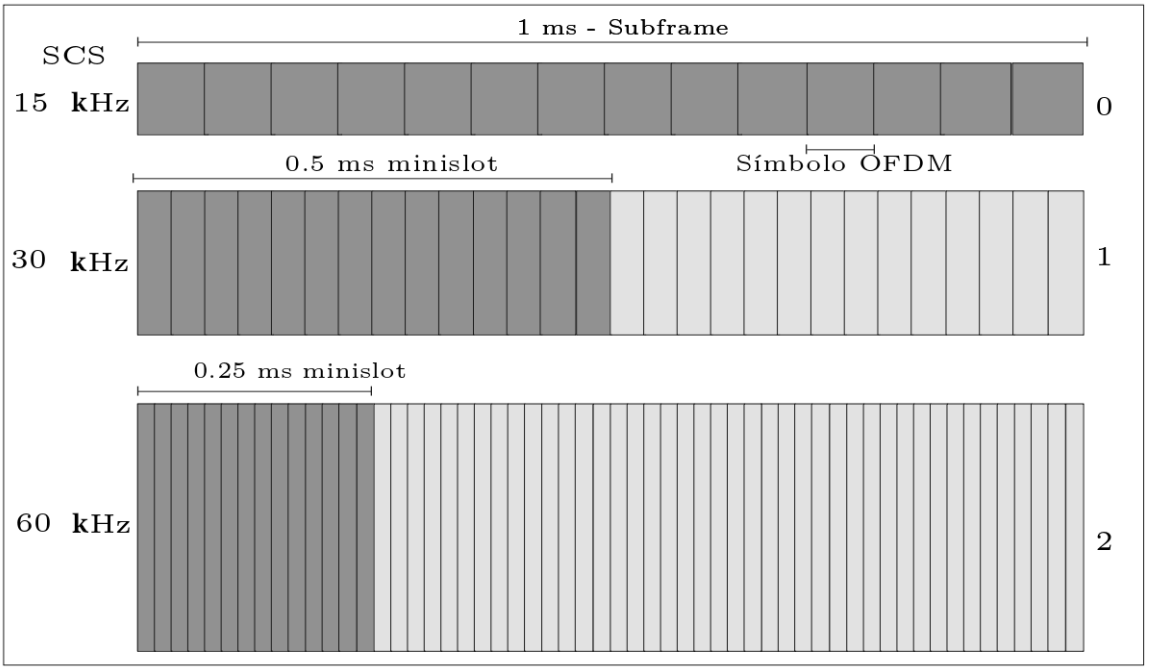


Figura 3.1: Esquemas de modulação para o 5G NR.

	Subportadora kHz	BW do RB kHz	Tempo do *Slot* ms	*Slots* por *Subframe*	Símbolos por *Slots*	*Slots* por *Frame*
0	15	180	1	1	14	10
1	30	360	0.5	2	14	20
2	60	720	0.25	4	14	40
3	120	1440	0.125	8	14	80
4	240	2880	0.0625	16	14	160

Tabela 3.3: Características da numerologia no NR [Dahlman et al., 2018].

Dahlman etal. [Dahlman et al., 2018] explica que cada RBs possui 12 subportadoras e se constitui na menor unidade de alocação de recursos de rede. O algoritmo de escalonamento de recursos terá efeito importante na qualidade percebida pelas aplicações, independente da numerologia em uso.

3.5 APLICAÇÕES

3.5.1 Aplicações eMBB

A principal característica desta categoria de aplicações do 5G NR é a necessidade de altas taxas de transmissão em áreas de cobertura abrangente conectando muitos usuários dentro de uma célula. O eMBB é considerado uma extensão dos serviços 4G LTE, porém utilizando recursos do 5G NR, com maiores taxas de transmissão e menor latência [Ahmadi, 2019]. As aplicações eMBB incluem Internet de alta capacidade, *streaming* de vídeo, jogos, reuniões virtuais com vídeos em 360^o com traduções simultâneas em tempo-real para diferentes línguas, e aplicações imersivas como a realidade virtual e realidade aumentada.

3.5.2 Aplicações mMTC

Este tipo é utilizado por comunicações entre máquinas e sua característica é uma quantidade pequena de dados transmitidos. Podem ser periódicos como leituras em sensores eletrônicos, ou assíncronos como envio de informações a partir de um gatilho como o disparo de um sensor de alarme. Hoje é muito utilizado para tecnologias IoT que precisam trocar informações com servidores para que sejam analisados e possam transformar os dados em informações para usuários, como a geração de gráficos informativos. Um exemplo de aplicação é o uso de atuadores em dispositivos eletrônicos, como a abertura de um portão. Outra grande expectativa é a utilização da Inteligência Artificial para gerar respostas automáticas ao identificar determinadas situações. As aplicações mMTC normalmente exigem baixa latência e baixa probabilidade de perda de pacotes, além de normalmente possuir uma grande densidade de dispositivos.

3.5.3 Aplicações uRLLC

O uRLLC tem como requisitos principais a baixa latência e a alta confiabilidade para suportar soluções como cirurgias remotas, veículos autônomos ou Internet táctil. Essas aplicações suscetíveis a latência devem transmitir os dados através dos elementos da rede com uma resposta típica máxima de 1 ms [3GPP, 2020], conforme estipulado pelo 3GPP. Para tanto, é preciso uma transmissão confiável que evite perdas de pacotes e a necessidade de retransmissões, que dependendo do número de pacotes para repor os dados perdidos, aumentam o atraso do sistema inviabilizando a solução. A confiabilidade definida pelo 3GPP para o uRLLC tem uma razão de perda de pacotes de 10^{-5}.

O presente trabalho não endereça aplicações uRLLC, apenas considera a competição entre aplicações eMBB e mMTC. Isso se justifica porque tecnologias mMTC estão mais presentes no dia a dia dos usuários e sistemas estão sendo introduzidos nas soluções comerciais existentes, passando a concorrer com aplicações eMBB tradicionais. As aplicações uRLLC exigem uso de escalonadores específicos para garantir a qualidade de serviço necessária.

3.6 ESCALONAMENTO DE RECURSOS

O algoritmo de escalonamento é responsável por determinar qual UE pode utilizar cada RB disponível. O mapa de utilização pode ser atualizado a cada milisegundo e enviado para as UEs utilizando um canal de sinalização.

A decisão do escalonador é realizada com informações coletadas pelo sistema, como o estado da fila de transmissão, a qualidade do canal reportada pelo CQI, o histórico de alocação,

o estado do *buffer*, entre outros. Também é necessário atentar para limitações do sistema para que o algoritmo seja eficiente, como a verificação do canal de *uplink*, o controle de *overhead*, multi-diversidade dos usuários e o consumo de energia [Maryam Imran Sheik Mamode, 2020].

Existem vários tipos de escalonadores provenientes do 4G LTE utilizados no 5G NR. Temos como exemplos algoritmos como o *Roudin Robin* (RR), *Maximum Throuput* (MT) e *Proportional Fair* (PF). Estes algoritmos servem de base para a confecção de outros, quando o desenvolvedor implementa melhorias para solucionar a utilização dos RBs.

3.7 PRINCIPAIS ALGORITMOS DE ESCALONAMENTO

A estratégia de escalonamento pode ser definida formalmente através da métrica [Mamane et al., 2022]:

$$m_{j,k} = \max_{i}\{m_{i,k}\} \tag{3.1}$$

onde j é o índice da UE que receberá a alocação do RB de índice k. O índice i varia entre todas as UEs que desejam receber recursos. Desta forma, conforme especificado em 3.1, o RB_k será alocado para UE_j que possuir o maior valor para a métrica $m_{i,k}$.

3.7.1 Round Robin (RR)

Este método atribui os RBs em esquema de fila circular, de acordo com a métrica a seguir:

$$m_{i,k}^{\mathrm{RR}} = t - T_i \tag{3.2}$$

onde t representa o tempo t atual, T_i representa o último instante de tempo que a UE_i recebeu a alocação para o k-ésimo RB. De acordo com 3.2, a alocação não depende da qualidade de canal e pode gerar situações onde *throughput* dos fluxos fique muito diferente entre as UEs, uma vez que a taxa de transmissão possível com um RB varia de acordo com a estimação da qualidade de canal para UE.

3.7.2 Maximum Throughput (MT)

O objetivo do *Maximum Throughput* (MT) é obter o maior *throughput* médio na célula rádio, mesmo que isto sacrifique a justiça na distribuição de recursos. A distribuição de recursos é realizada de acordo com a métrica a seguir:

$$m_{i,k}^{\mathrm{MT}} = \log[1 + \mathrm{SNIR}_k^i(t)] \tag{3.3}$$

onde SNIR_k^i é a relação **sinal-ruído mais interferência** estimada para a UE_i em relação ao RB_k no tempo t atual.

Neste caso, a UE que possuir melhor qualidade de canal irá receber a alocação do RB disponível. Como a UE que possui melhor qualidade de canal estará usando a técnica de modulação com maior número de bits por símbolo, o *throughput* médio será o maior possível.

No entando grandes injustiças no pior caso, é possível que determinadas UEs nunca consigam alocar recursos.

3.7.3 Proportional Fair (PF)

PF é um algoritmo que considera a justiça entre usuários e eficiência espectral. O objetivo do PF é obter justiça na divisão do *throughput* entre as UEs, utilizando a seguinte métrica:

$$\begin{aligned} m_{i,k}^{PF} &= \log[1 + \text{SINR}_k^i(t)]/\overline{R^i}(t-1), \\ \overline{R^i}(t) &= \gamma * \overline{R_i}(t-1) + (1-\gamma) r^i(t) \end{aligned} \tag{3.4}$$

onde $r^i(t)$ é a taxa de transmissão instantânea da UE_i, $\overline{R^i}(t)$ representa uma média móvel exponencial ponderada de $r^i(t)$ com parâmetro γ.

Com o uso do PF as UEs que possuem qualidade de canal mais baixa tendem a receber a alocação de um número maior de RBs para obter um *throughput* médio (considerando a média móvel exponencial ponderada) equivalente às UEs que possuem uma melhor qualidade de canal.

3.8 ESCALONADORES PARA CONVIVÊNCIA EMBB, MMTC E URLLC

Além dos algoritmos básicos principais, existem outras abordagens alternativas que buscam garantir uma distribuição justa entre os usuários, podendo organizar a grade de recursos de maneira mais eficiente, de acordo com o objetivo específico de cada aplicação. No campo científico, há uma diversidade de estudos, cada um com um foco e uma abordagem distintos, dependendo do tipo de aplicação e dos objetivos a serem alcançados. A seguir, são descritos algoritmos relacionados ao estudo de serviços mMTC e eMBB que competem por recursos.

Em um *survey* com dezenas de propostas no escalonamento de recursos no problema da coexistência entre aplicações eMBB, mMTC e uRLLC nas redes móveis 5G, Kumar et al. [Kumar et al., 2024] descreve os algoritmos escalonadores e formula questões sobre o assunto, desafios e direções futuras.

Katila et al. [Katila et al., 2017] propõem um algoritmo *channel-aware/QoS-unaware*, capaz de escalonar dispositivos denominados escalonáveis e não-coordenados em uma mesma célula. O algoritmo proposto chamado Neighbors-Aware Proportional Fair (N-PF) considera as condições do estado do canal do *uplink* e o número de nós não-coordenados vizinhos de cada nó escalonado agregado ao cálculo da métrica do PF, assim maximizando a probabilidade de sucesso nas transmissões dos pacotes de dados. Também é utilizado um esquema de distribuição uniforme dinâmica (DUD) para transmitir pacotes de diversos tamanhos. Os resultados obtidos foram comparados ao PF, mostrando superioridade de 35% no ganho de *goodput* sem comprometer a justiça entre os usuários.

Liu et al. [Liu et al., 2019] sugere como objetivo do seu modelo escalonador a melhoria da eficiência espectral (SE) e eficiência energética (EE) em infraestrutura ultra densa. Neste tipo de topologia diferentes tipos de dispositivos inteligentes e número massivo de aplicações IoT, operam em uma rede de raio reduzido resultando no aumento da interferência entre as células. A proposta para a solução do problema é utilizar o *deep reinforcement learning* para encontrar a solução que considera as informações do SE e EE obtidas pelo CQI, para otimizar a distribuição dos recursos da rede utilizando Processos de Decisão de Markov (MDP) [FAGUNDES, 2011].

Um algoritmo *energy-aware*, chamado *Dueling Deep Q Network Strategy*, foi criado para escalonar os recursos em uma rede ultra-densa e comparado com a alocação dos RBs utilizando os tradicionais *Q-learning* e *Deep Q Network* DQN. De acordo com os testes realizados o Dueling DQN demonstrou superioridade aos algoritmos envolvidos no estudo quando comparada sua eficiência energética espectral.

Em outro estudo Kooshki et al. [Kooshki et al., 2023], cria uma estratégia *channel-aware/QoS-unaware/preemption-aware* para escalonar usuários uRRLC conjuntamente com eMBB. A proposta desenvolve um novo esquema para o *radio resource management* (RRM) que considera a combinação de escalonadores do domínio do tempo (DT) e domínio da frequência (DF) num algoritmo escalonador único. O algoritmo DT usa uma métrica normalizada para priorizar as aplicações uRLLC e maximizar o *throughput* das aplicações eMBB. O algoritmo DF utiliza uma métrica baseada no *throughput* para aplicações uRLLC e para as aplicações eMBB pode ser usada a métrica calculada pela *interference contribution ratio* (ICR) e também baseada pelo *throughput*. O algoritmo proposto teve uma melhoria de 29% na latência para usuários uRLLC e melhoria de 90% na SINR das aplicações eMBB, comparadas ao algoritmo de referência sem prioridade para a aplicação uRLLC no escalonador DT e utilizando o MT no escalonador DF [Karimi et al., 2020].

Chukwu et. al. [Chukwu et al., 2020] desenvolve um novo algoritmo escalonador *channel-aware/QoS-unaware*, e compara sua utilização com aplicações conjuntas eMBB-uRLLC e eMBB-mMTC. Ele utiliza para a confecção da métrica um novo esquema de codificação, *network-slicing* e numerologia. Os resultados foram comparados, e observado a capacidade de canal, a taxa de erro, o *throughput* e a eficiência espectral. Com os resultados encontrados foi possível observar as vantagens e desvantagens da utilização de aplicações diferentes na mesma estação rádio-base, quando utilizada bandas de frequência de larguras diferentes e distâncias entre as UEs e a gNodeB. Quando as UEs eMBB-uRLLC e eMBB-mMTC estão à 50 m da gNodeB a capacidade média está em 1700 Mbps e 1950 Mbps respectivamente. Acima desta distância, em 300m, a capadidade média cai para as aplicações eMBB-uRLLC para 850 Mbps e para as aplicações eMBB-mMTC 200 Mbps.

A Tabela 3.4 mostra a relação dos algoritmos estudados e suas principais características.

Algoritmo	Característica
RR	Distribuiçao jsuta dos recursos em frequência entre usuários.
MT	Atinge o máximo *throughput* dentro de uma célula.
PF	Justo em relação ao *throughput* entre os usuários no sistema.
N-PF	Aloca elementos com maior interfência nos melhores canais diminuíndo os erros.
Dueling DQN	Considera eficiência espectral e energética na utilização dos recursos em Processo de Decisão de Markov.
RRM	No domínio do tempo prioriza o uRLLC e maximiza o eMBB, e no domínio da frequência métrica calculada pela *interference contribution ratio* (ICR) e também pelo *throughput*.
Multi 5G-NR Apps	Métrica com um novo esquema de codificação, *network-slicing* e numerologia.

Tabela 3.4: Comparação entre os Algoritmos estudados.

Nenhuma das estratégias descritas é capaz de priorizar o tráfego mMTC em relação ao eMBB. Para que isto ocorra, deveria ser usado um escalonador que prevê QoS. No entanto, nestes casos, a operadora do sistema deve identificar explicitamente quais UEs geram cada tipo de tráfego, implicando em uma relação comercial específica, com a cobrança de valores e reserva

de recursos na rede. Neste trabalho o objetivo é produzir um algoritmo capaz de identificar e priorizar o tráfego mMTC sobre o eMBB de forma automática, sem exigir a implementação de um escalonador projetado para QoS. Não é do conhecimento do autor a existência de algoritmos de escalonamento projetados com este propósito na literatura. Os resultados obtidos mostram seu funcionamento em relação à justiça e sua superioridade entre os algoritmos mais conhecidos como RR, MT e PF. O método de referência utilizado pela maioria dos trabalhos relacionados é o PF, o que fundamenta a escolha para a comparação dos resultados neste trabalho. O PF é muito utilizado na indústria pelos fabricantes de equipamentos [AlQahtani e Alhassany, 2013]. O RR e o MT foram preservados para determinar seus desempenhos e compará-los com o desempenho do algoritmo proposto.

4 ESCALONAMENTO COM PRIORIDADE DINÂMICA

O algoritmo proposto é o Escalonamento com Prioridade Dinâmica (*Dynamic Priority Scheduler*, DPS) inspirado nos algoritmos de escalonamento utilizados em sistemas operacionais de tempo compartilhado. Nestes sistemas o uso do DPS identifica automaticamente processos consumidores de CPU e processos interativos. O objetivo é garantir que processos interativos sejam escalonados o mais brevemente possível para não impactar na qualidade percebida da aplicação. São exemplos de processos interativos as interfaces com usuários, servidores de rede e servidores de banco de dados. A ideia central é que o benefício obtido ao priorizar a aplicação interativa é muito superior ao custo, que é o aumentar o tempo de execução de uma tarefa consumidora de CPU. Neste contexto, as aplicações eMBB seriam os processos consumidores de CPU e as aplicações mMTC os processos interativos.

O DPS utiliza duas prioridades: a prioridade base e a prioridade dinâmica. A prioridade base representa a maior prioridade que pode ser atribuída a um UE. Inicialmente, a prioridade dinâmica é igual a prioridade base. Um RB será reservado para a UE que possuir a maior prioridade dinâmica. Quando uma UE receber um RB, sua prioridade dinâmica será decrementada em um valor fixo D até o mínimo de 0. Caso uma UE não receba recursos em um dado TTI, sua prioridade dinâmica será incrementada em um valor fixo I, respeitando o limite máximo da prioridade base.

Com o uso do DPS, espera-se obter os seguintes efeitos:

1. Aplicações eMBB demandam grande quantidade de recursos e terão sua prioridade dinâmica quase sempre baixa, só recebendo recursos caso não hajam aplicações mMTC solicitando recursos.

2. Aplicações mMTC raramente solicitam recursos, de forma que a tendência é que sua prioridade dinâmica seja quase sempre alta. Desta forma, quando uma aplicação mMTC solicitar recursos ela terá prioridade alta se comparada à aplicações eMBB.

Pode haver redução nos recursos alocados às aplicações eMBB para acomodar este tipo de priorização. No entanto, isto é aceitável e a maioria das aplicações eMBB é capaz de adaptar-se às condições de congestinamento como, por exemplo, o vídeo *streaming* adaptativo amplamente utilizado atualmente. Desta forma, a métrica para reserva de recursos é definida como:

$$m_{i,k}^{\mathrm{DPS}} = P_i^d + \frac{m_{i,k}^{\mathrm{RR}}}{\max\limits_i \{m_{i,k}^{\mathrm{RR}}\}} \tag{4.1}$$

onde P_i^d é a prioridade dinâmica da UE_i, e $m_{i,k}^{\mathrm{RR}}$ é a métrica para o *Round Robin* definida em (3.2).

A prioridade dinâmica inicial de uma UE será igual à prioridade base. O Fluxograma 4.1 mostra os detalhes do incremento e decremento da prioridade dinâmica que deve ser executado a cada TTI pela gNodeB. Quando a UE recebe a alocação de um RB sua prioridade dinâmica será decrementada em D, respeitando-se o limite mínimo de 0. Caso a UE_i não receba a alocação de um RB no TTI, o contador C_i será incrementado em 1. Quando o contador C_i atingir o valor I, a prioridade dinâmica será incrementada em 1 unidade, respeitando-se o limite da prioridade base.

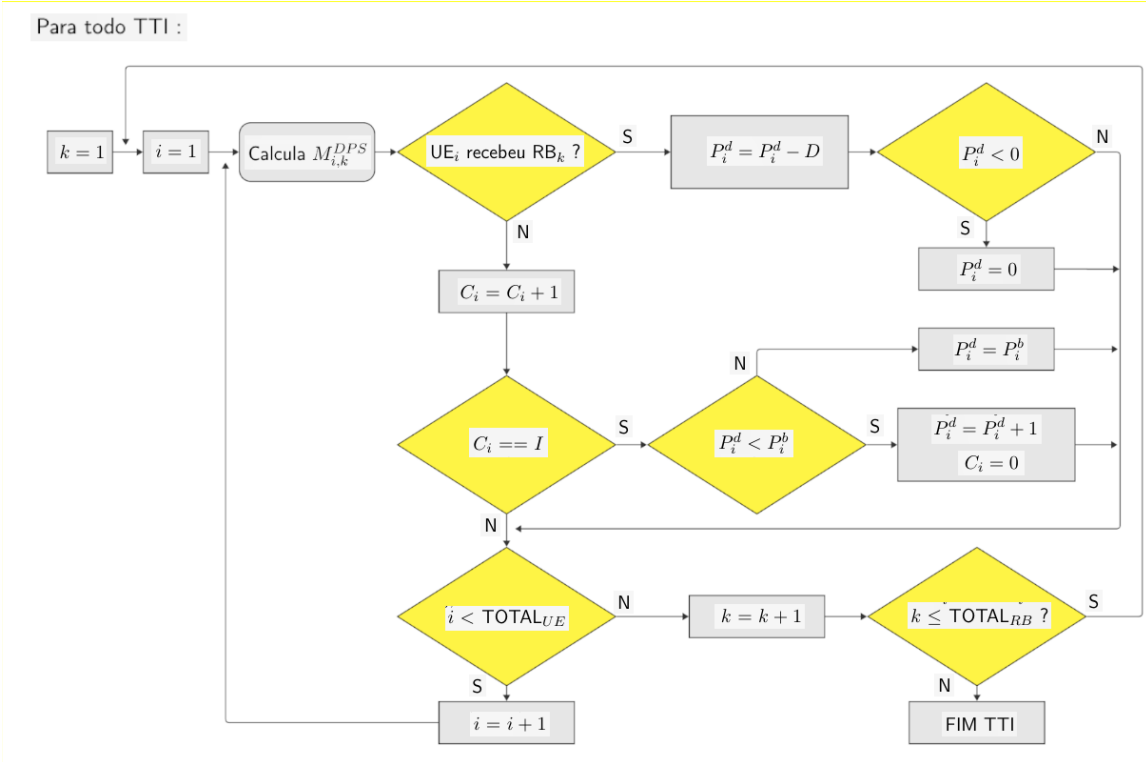


Figura 4.1: Fluxograma do Algoritmo DPS.

Caso haja um empate na prioridade dinâmica é utilizado a métrica para o algoritmo Round Robin. No caso de utilizar o PF como critério de desempate, a métrica pode ser escrita como:

$$m_{i,k}^{\text{DPS}} = P_i^d + \frac{m_{i,k}^{\text{PF}}}{\underset{i}{max}\{m_{i,k}^{\text{PF}}\}} \tag{4.2}$$

onde P_i^d é a prioridade dinâmica e $m_{i,k}^{PF}$ a métrica para o *Proportional Fair*.

Também é importante notar que, caso não hajam dispositivos mMTC na rede rádio e apenas dispositivos eMBB ativos, o comportamento no limite converge para o método usado como critério de desempate, ou seja, RR no caso de (4.1) e PF em (4.2). Na implementação realizada optou-se por usar (4.2). Em ambos os casos deve ser utilizado o Algoritmo 4.1 para incremento/decremento da prioridade dinâmica.

Quando ocorre o empate no cálculo da métrica utilizando as prioridades das UEs no algoritmo DPS e é utilizado o critério de desempate, $m_{i,k}^{\text{PF}}/\underset{i}{max}\{m_{i,k}^{\text{PF}}\}$ tem o seu valor entre 0 e 1.

A notação *big O* expressa a complexidade computacional em termos do tamanho da entrada n. Ela descreve um limite superior assintótico para o tempo de execução ou a utilização da memória, ignorando as outras variáveis de sistema de ordem inferior. Verificando o Algoritmo 4.1, a complexidade computacional é $O(n)$, neste caso é a complexidade mínima do OFDMA. O método proposto também depende do critério de desempate, que no caso foi o PF, que também possui complexidade no pior caso de $O(n)$.

5 AVALIAÇÃO DE DESEMPENHO

5.1 CENÁRIO

5.1.1 Topologia

Foi configurado no NS-3 na parte da RAN um sistema contendo uma gNodeB centralizada na célula em um ambiente rural com visada direta, no ns-3 designado RMa_LOS. Este modelo permite que trabalhe variando as distâncias e assim gerando uma diferença no tipo de codificação da modulação (*Modulation and Code Scheme*, MCS) utilizada na transmissão, e assim variando a quantidade de RBs necessárias para cada fluxo das UEs presentes na simulação. A gNodeB foi posicionada no centro da célula. Para as aplicações eMBB, foram realizados testes com 3 e 5 usuários, os mesmos posicionados entre 0 e 40,000 m da gNodeB. Os elementos utilizam um espaçamento que foi calculado a partir da distância máxima dividida pelo número de aplicações eMBB, desta maneira estavam posicionados nas distâncias de 13,333 m, 26,666 m e 39,999 m no **teste 1** com 3 aplicações eMBB, e 8,000 m, 16,000 m, 24,000 m, 32,000 m e 40,000 m para o **teste 2** com 5 usuários eMBB. Para os usuários mMTC, que estão em número maior, foram posicionados de maneira que o espaçamento entre os elementos seja a distância máxima de 40,000 m dividida pelo número de aplicações mMTC, de 0 à 40, na medida que foram introduzidas nas simulações. O core da rede 5G NR, foi conectado à servidores externo de aplicação de video *streaming* e aplicação IoT, através de um link conectado ao *User Plane Function* (UPF) e este conectado à gNodeB. Os servidores utilizam UDP, que faz a transmissão de dados no sentido do *downlink*, ou seja, do servidor para as UEs, conforme ilustrado pela Figura 5.1. Desta forma, o escalonador projetado será avaliado considerando o tráfego concorrente no *downlink* da rede rádio.

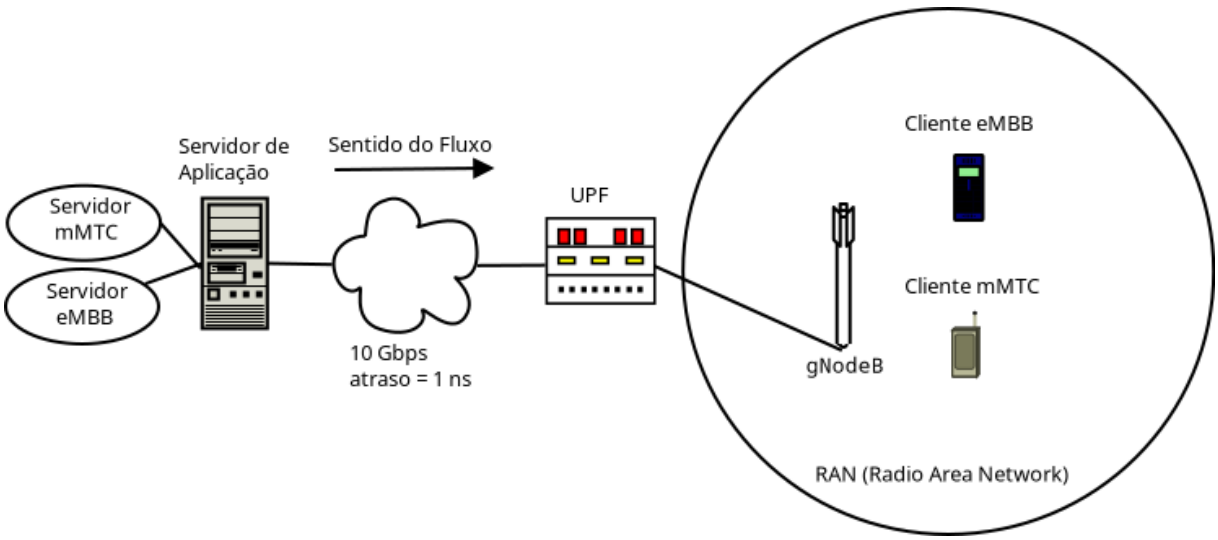


Figura 5.1: Cenário criado no NS-3 para os testes.

A conexão entre servidor de aplicação e a UPF foi realizada com um enlace de taxa muito alta (10 Gbps) e atraso desprezível (1 ns) para não influenciar nas medições de desempenho, uma vez que deseja-se avaliar o desempenho da RAN.

O modelo de propagação utilizado nos testes foi o de espaço livre. Este é um modelo simples, que supõem que os elementos se comunicando estejam em visada direta, sem obstáculos.

O uso deste modelo não implica em perda de generalidade na avaliação do método proposto, uma vez que para a avaliação de desempenho do método proposto é necessário que haja variação da qualidade de canal, mas a forma como isto ocorre não impede ou invalida os resultados. No entanto, com o uso do modelo de propagação de espaço livre, o raio da célula rádio foi mantido em 40,000 metros para garantir que haja variação de qualidade de canal entre as UEs.

5.1.2 Parâmentros de Simulação

Para esta simulação foram definidos valores para a frequência central, largura de banda, número de canal por banda, cenário, potência de transmissão e recepção, tempo de simulação, numerologia, altura da antena e altura do UE. Os parâmetros de simulação são apresentados na Tabela 5.1.

Parâmetro	Valor
Frequência Central	3,510 GHz
Largura de banda	20 MHz
Num. de CC por banda	1
Cenário	RMa_LOS
Potência de transmissão da gNodeB	30 dBm
Potência de transmissão da UE	28 dBm
Tempo de Simulação	600s
Numerologia	0
Altura gNodeB	50 m
Altura UEs	1,5 m
Escalonador	RR,PF,MT,DPS

Tabela 5.1: Parâmetros utilizados.

5.1.3 Geradores de Tráfego

Para cada tipo de aplicação foi utilizada uma forma de gerar o tráfego esperado. No caso do mMTC foi realizada uma taxa constante de transmissão de 360 kbps para simular uma aplicação mMTC [Mehmeti e La Porta, 2022]. A Tabela 5.2 informa os parâmetros utilizados no gerador de tráfego mMTC.

Parâmetro	Valor
Intervalo de Transmissão	0.005 s
Tamanho de Pacotes	228 Bytes

Tabela 5.2: Características de transmissão usuários mMTC.

Para o eMBB, foi utilizado o modelo ON/OFF, com o estado ON sendo modelado por uma distribuição de Pareto e o estado OFF por uma distribuição exponencial. Este é um dos modos mais populares para geração de tráfego de vídeo com características auto-similares. Neste caso, o parâmetro de *Hurst* é dado por $H = \frac{(3-\alpha)}{2}$ [Rusev e Skorikov, 2023] , onde α é o

parâmetro de forma da distribuição de Pareto. Deseja-se obter H = 0,85, que é um valor típico para transmissão de vídeo, logo α pode ser calculado como 1,3.

Considere que o tempo de permanência nos estados ON e OFF são representados respectivamente pelas variáveis aleatórias X e Y. A esperança para a distribuição de Pareto é dada por $E[X] = \alpha\beta/(\alpha - 1)$. Deseja-se imitar o tráfego de vídeo [Rao et al., 2011] a 30 fps, sendo definido E[X]= 1/60 (tempo ON + tempo OFF = 1/30). Com isso foi obtido o parâmetro β, com valor de 0,003846154. Para o estado ON, foi definido E[Y] = 1/60. Como a esperança da distribuição exponencial é dada por $E[Y] = \mu$, então $\mu = 1/60$.

A taxa média resultante pode ser obtida através da taxa de transmissão dividida por 2, pois a esperança do tempo ON foi definida igual à esperança do tempo OFF. A taxa de geração do fluxo eMBB foi configurada em 5,8 Mbps, o que leva uma taxa média de transmissão de 2,9 Mbps por fluxo.

Parâmetro	Valor
Data Rate	5.8 Mbps
Tamanho de Pacotes	1024 Bytes
ON TIME	VariávelRandômicadePareto(α=1.3| β=57480.9)
OFF TIME	VariávelRandômicaExponencial(μ=1/60)

Tabela 5.3: Características de transmissão usuários eMBB.

Foi usada a largura de banda de 20 MHz, que é menor do que o uso típico de sistemas 5G. Isso se justifica porque o uso de uma largura de banda maior exigiria uma quantidade maior de geradores de tráfego, o que aumentaria muito o tempo de simulações considerando a grande quantidade de testes necessários, além de usar um número bem maior de UEs na geração de tráfego.

Para o parâmetro D, foi atribuído o valor 1 para o decremento no cálculo da prioridade dinâmica, desta maneira diminuíndo o valor da prioridade do elemento a cada iteração até o valor mínimo de 0. Para calibrar o algoritmo, foi utilizada a variável I para uma unidade da prioridade dinâmica. Foram realizados diversos testes com os valores de I com 10, 12, 20, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 48, 50, 51, 52, 53, 54, 60 e 72. Após verificar os resultados obtidos, o valor 53 foi escolhido por apresentar melhor desempenho. Os resultados obtidos foram incluídos no Apêndice A.6.

5.1.4 Resultados

5.1.4.1 Teste 1

A Figura 5.2 apresenta as respostas das simulações computacionais para os dispositivos mMTC. Nesta simulação, 3 dispositivos eMBB estão ativos simultaneamente, transmitindo a taxas de 2,8 Mbps. Com a introdução de 0 a 40 dispositivos mMTC, pode-se notar que o atraso médio do algoritmo proposto é menor que o encontrado nos algoritmos PF e RR. O atraso médio do MT foi baixo, mas isso ocorre porque diversos fluxos não realizaram transmissão alguma, o que melhora o atraso médio artificialmente. Vizualizando o *throughput*, o DPS tem um decaimento menor que os algoritmos concorrentes, mantendo a taxa de transmissão constante. O *jitter* apresenta o mesmo comportamento que os outros algoritmos. No gráfico da perda de pacote, nota-se que o DPS tem a menor perda em relação aos concorrentes.

No mesmo teste, foi apresentado na Figura 5.3 a resposta média para os 3 dispositivos eMBB participantes das simulações. O atraso médio das aplicações eMBB são maiores para os

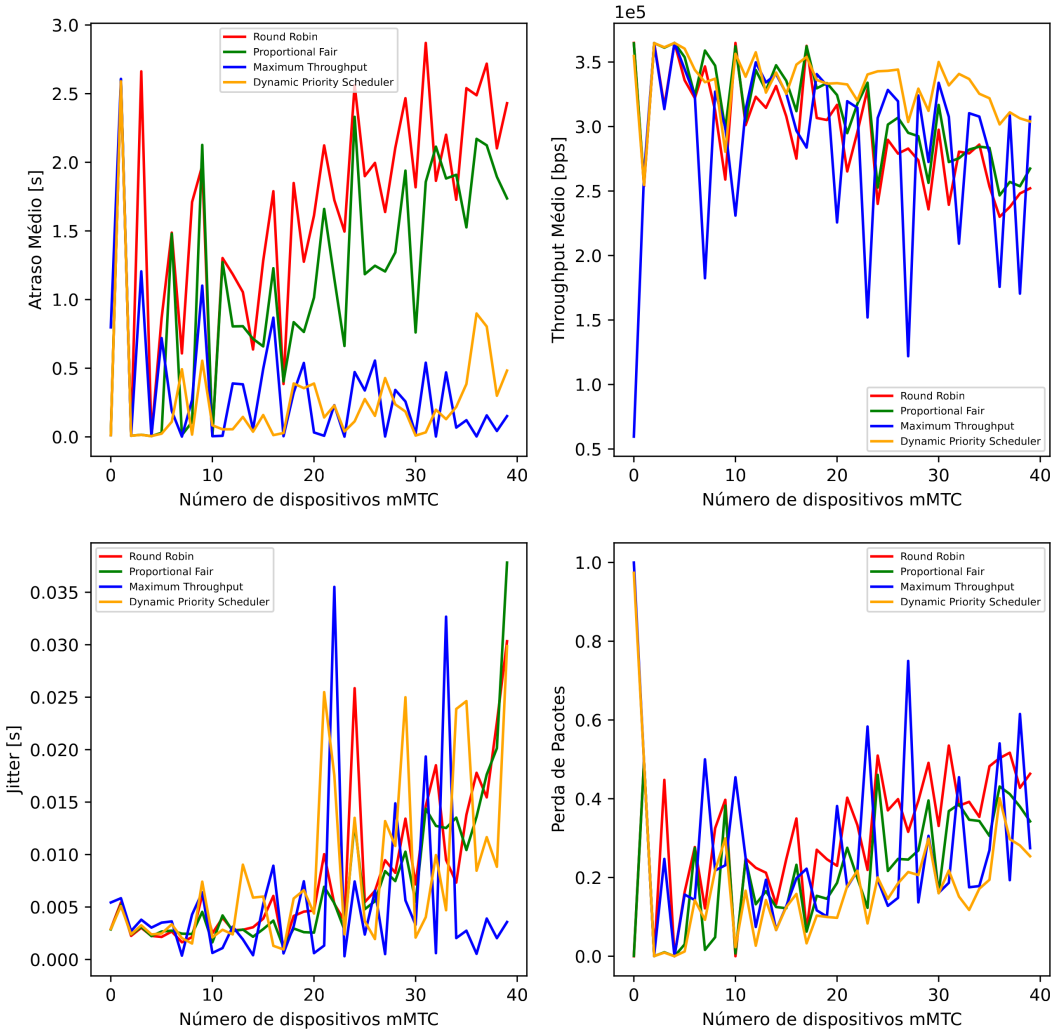


Figura 5.2: Resultados do teste 1 para usuários mMTC.

algoritmos RR e PF. No caso do *throughput* o DPS apresenta taxas de transmissão média um pouco superior que o PF e RR, mostrando que mesmo nas aplicações eMBB é possível notar uma melhoria. O *jitter* apresenta similiaridade em relação aos outros algoritmos e a perda de pacotes é mais baixa que os concorrentes.

5.1.4.2 Teste 2

Na Figura 5.4 apresenta os resultados simulados utilizando 5 usuários eMBB, cada um transmitindo 2,8 Mbps, totalizando 14 Mbps. Da mesma maneira que o teste anterior, aplicações mMTC foram inseridas de 0 a 40 usuários. Nos gráficos, observa-se o atraso médio dos algoritmos PF e RR superiores ao DPS. Para o MT ocorre o caso da melhoria artificial do atraso sendo que diversos fluxos não transmitem, não contabilizando esses fluxos. Nota-se a diminuição do *throughput* nos algoritmos concorrentes enquanto o DPS continua constante, suportando uma quantidade maior de aplicações com as taxas de transmissão esperadas. Nota-se que ao atingir 40 usuários mMTC o *throughput* médio está 300 Kbps para o DPS, 250 Kbps para o PF e 200 Kbps para o MT e RR. O *jitter* apresenta similiaridade entre todos os algoritmos. Isto acontece por sofrerem interferência parecida durante o tempo da simulação. No caso da perda de pacotes, pode-se notar que a perda do algoritmos proposto é menor que os outros algoritmos envolvidos no teste.

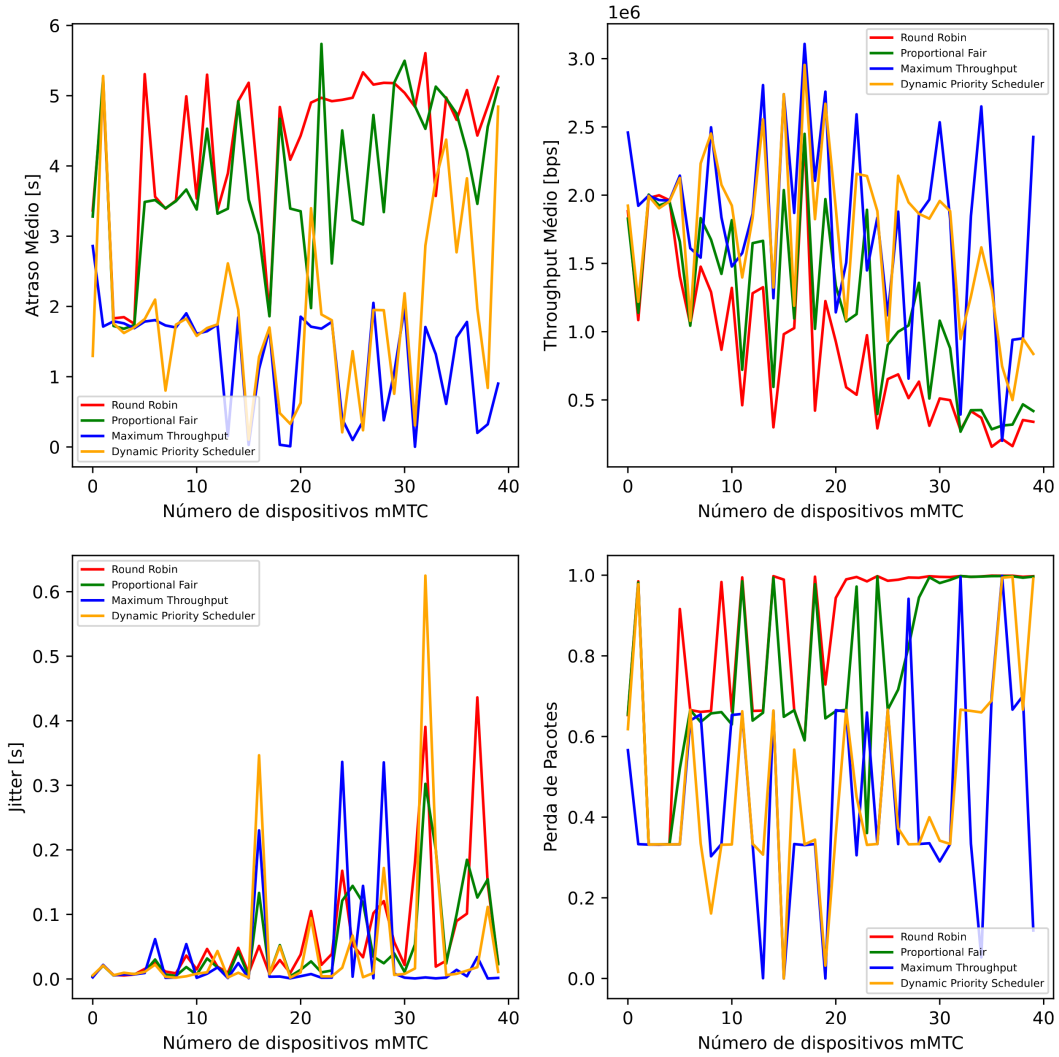


Figura 5.3: Resultados do teste 1 para usuários eMBB.

Para as aplicações eMBB, mostrados na Figura 5.5, o atraso médio notado do DPS é menor que o PF e RR. Vizualizando o *throughput*, observa-se que as aplicações eMBB sofrem grande degradação a medida que as aplicações mMTC são introduzidas. Ainda assim, nota-se que o DPS tem um melhor resultado que os demais. Comparando os resultados do *jitter*, vemos que o valor encontrado para o DPS é relativamente menor que os demais, desconsiderando o atraso médio artificial do MT. A perda de pacotes apresentada no gráfico é menor para o algoritmo proposto, desconsiderando o MT que não contabilizou os pacotes perdidos das aplicações que não transmitiram devido o funcionamento lógico do algoritmo MT.

Os resultados também mostram que o *throughtput* das aplicações eMBB não foi reduzido severamente em função da inserção de dispositivos mMTC.

Além disso, foi avaliada a justiça na distribuição de recursos e determinou-se uma diferença significativa de desempenho entre os diversos fluxos dependendo do algoritmo de escalonamento usado. Foi utilizado o índice de Jain [Jain, 1991], definido por

$$J = \frac{(\sum_{j=1}^{n} x_j)^2}{n \sum_{j=1}^{n} x_j^2}, \tag{5.1}$$

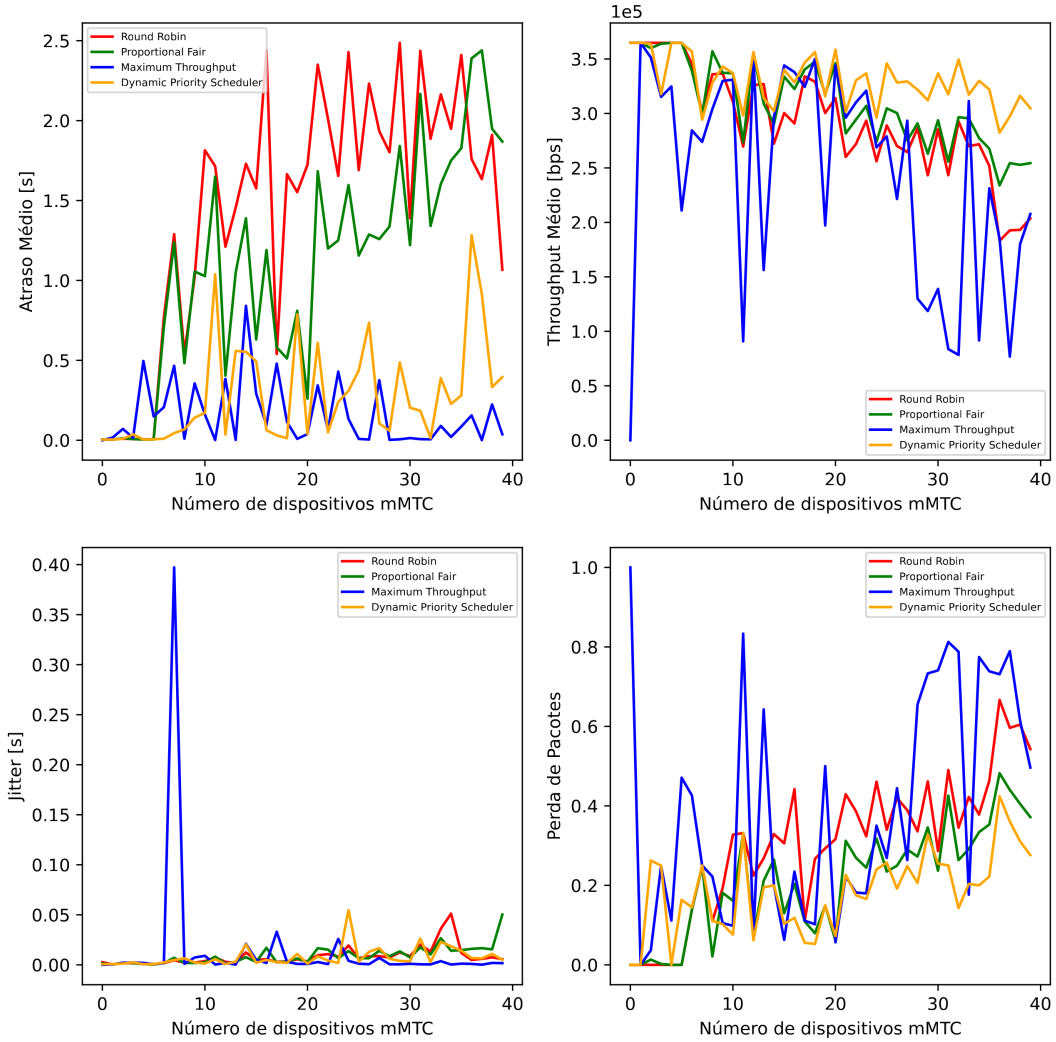


Figura 5.4: Resultados do teste 2 para usuários mMTC.

onde J é o índice de justiça de Jain, n é o número de medidas e x_j representa uma medida. O valor de J deve estar entre 0 e 1, sendo 1 representando a maior justiça possível e 0 a menor.

O índice de justiça de Jain do *throughput* das aplicações mMTC foi calculado para os testes 1 e 2, e apresentados na Figura 5.6. Para o caso do teste 2, com maior congestionamento, é possível observar que o algoritmo DPS mantém a justiça à medida que o congestionamento aumenta, apresentando quando inseridos 40 usuários mMTC na simulação o valor de 0,9. No entanto, o algoritmo PF apresenta valor de 0,73, e o MT e RR de 0,6 aproximadamente, quando existem 40 usuários mMTC no sistema, resultando em um ganho de 23% e 50% respectivamente. Comprova-se a superioridade do DPS em relação aos algoritmos concorrentes.

Assim como os algoritmos apresentados no item 3.8, houve uma melhoria na justiça entre os usuários para a distribuição dos recursos da gNodeB. Existem inúmeras abordagens para priorizar e escolher os dados a serem escalonados na grade de recursos. O melhor algoritmo à ser utilizado depende do tipo de aplicação escolhida, dos objetivos e das variáveis do ambiente onde a estação rádio-base está implementada.

Comparando aos resultados obtidos com os outros autores, no caso do N-PF, os nós sofrem grande interferência pela densificação das multi-células envolvidas no teste, o índice de Jain apresentou melhoria de 35% em relação ao PF e o DPS com menor interferência 23%. A complexidade computacional do DPS é muito inferior $O(n)$, equivalente ao PF, destancando

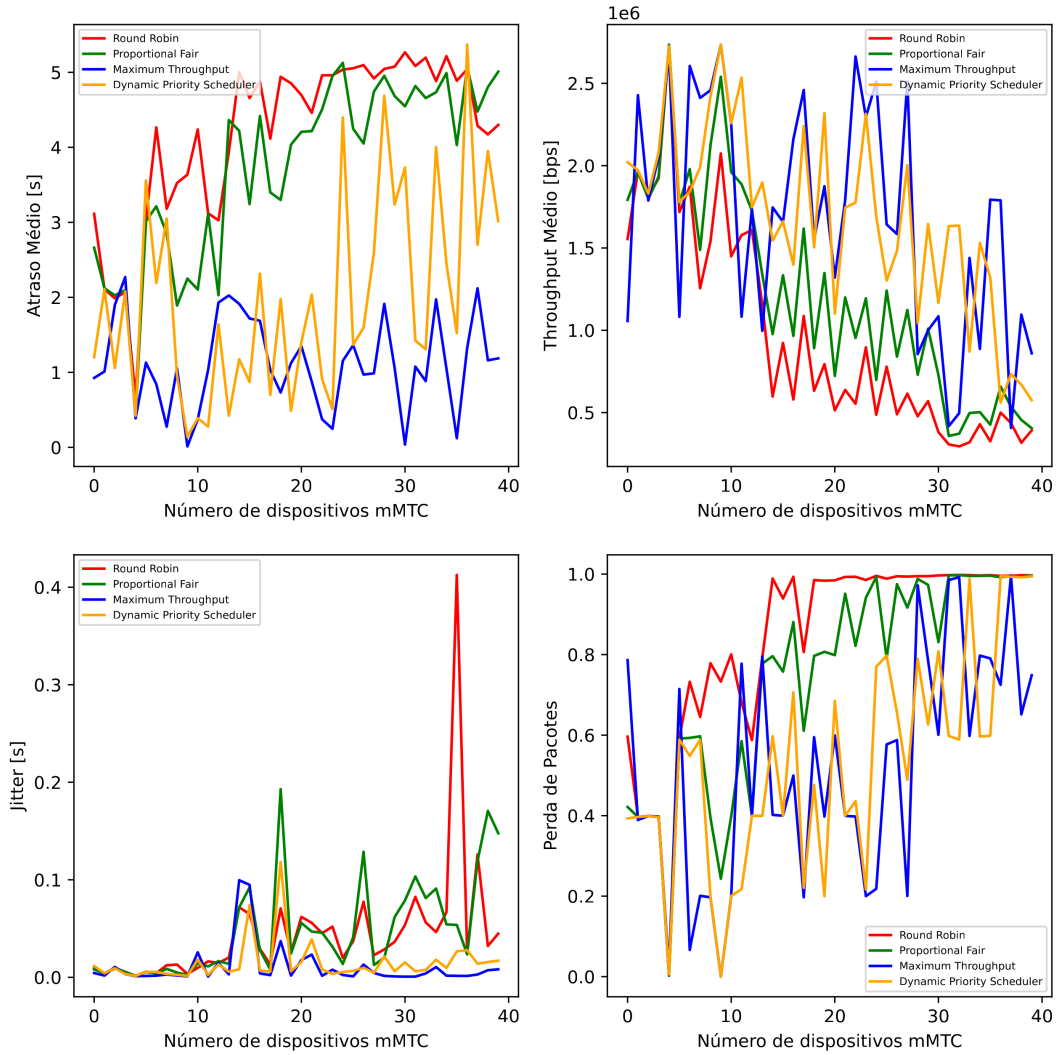


Figura 5.5: Resultados do teste 2 para usuários eMBB.

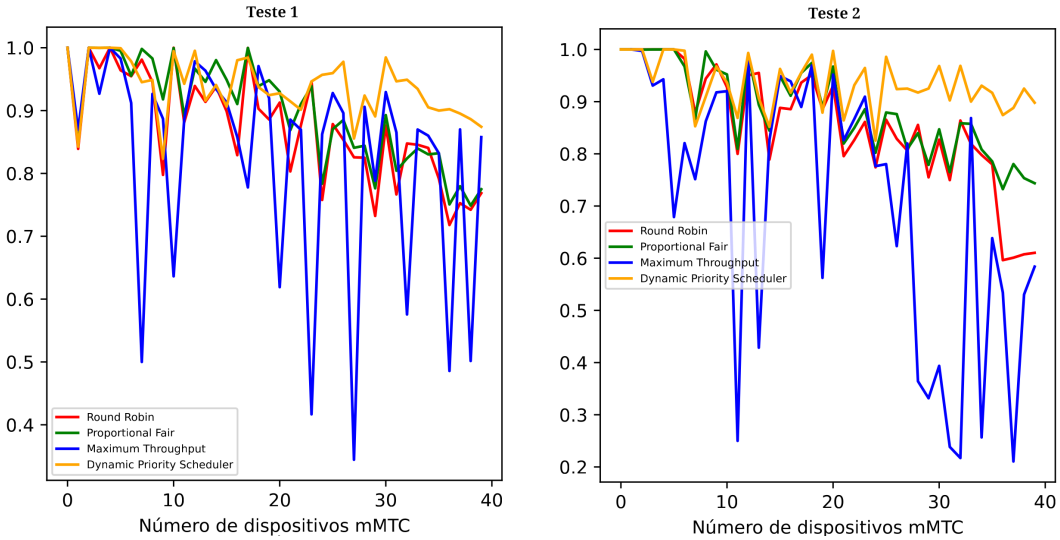


Figura 5.6: Índice de Jain calculado para os testes 1 e 2 respectivamente.

facilidade na implementação devido a urgência da resposta em tempo quase real (NearRT). No caso do estudo apresentado por Kooshki, a complexidade computacional é $O((S+1)^K)$. Observa-

se que o DPS suporta um número significavelmente maior de UEs com a rede congestionada com o serviço mMTC.

6 CONCLUSÕES E TRABALHOS FUTUROS

Durante as simulações, foi verificado que mesmo quando o sistema está congestionado, utilizando o algoritmo proposto, o sistema suportou um número maior de aplicações mMTC em comparação com os algoritmos concorrentes. Os resultados indicam que o menor atraso observado nas aplicações mMTC foi obtido a um custo de aumento no atraso das aplicações eMBB. No entanto, a penalização imposta às aplicações eMBB não foi grande e pode ser absorvida pelas aplicações, que são tipicamente de vídeo *streaming*. O impacto ao usuário seria a redução da qualidade de vídeo, o que se configura numa boa troca considerando o ganho com o suporte à um número maior de aplicações mMTC.

O *throughput* médio dos dados das aplicações eMBB diminui a medida que aplicações mMTC são inseridas no sistema, por causa do congestionamento dos dados na fila de entrada até que atingem o ponto de saturação e neste ponto há uma queda abrupta do *throughput* médio recebido. No caso dos usuários mMTC, o *throughput* médio das aplicações permanece constante até o ponto de saturação. Perecebe-se que quando o DPS é utilizado para o escalonamento, o sistema suporta uma quantidade maior de aplicações e o mantém operacional em maiores níveis de congestionamento. O método se torna imbatível nos casos de elementos que iniciam a transmissão por métodos de disparo, onde sua prioridade estaria muito mais alta em relação à prioridade do tráfego existente.

O algoritmo de escalonamento desenvolvido, *channel-aware/QoS-unaware*, permite a convivência de aplicações eMBB e mMTC sem a necessidade de implementar mecanismos de QoS em redes 5G New Radio. O algoritmo proposto *Dynamic Priority Scheduler* tem capacidade de identificar o tráfego de tipo máquina e priorizá-lo na utilização dos recursos disponíveis na interface de rádio do sistema. Os testes realizados no simulador NS-3 demonstram que a abordagem proposta apresenta uma superioridade de até 23% na justiça em relação ao *throughput*, priorizando aplicações mMTC, e é 50% mais eficaz na justiça para a alocação de recursos comparada aos algoritmos RR e MT.

6.1 SUGESTÕES DE TRABALHOS FUTUROS:

1. Pode ser realizada uma análise mais extensa a respeito do número de níveis de prioridades das base usadas. Neste trabalho foram usados 20 níveis, com inspiração no Sistema Operacional Linux. No entando um número maior ou menor pode se mostrar mais apropriado.

2. Pode ser realizado um estudo sobre o impacto do decremento da prioridade no desempenho do sistema. Neste trabalho, o decremento foi mantido constante, sendo variado apenas o valor do incremento.

3. Uso de uma estrutura de árvore tipo *red-black* para classificar as UEs de forma ordenada utilizando o critério temporal de uso de recursos. Esta técnica é utilizada no Sistema Operacional Linux e reduz a complexidade do incremento/decremento dinâmico de prioridade dinâmica.

Referências Bibliográficas

3GPP (2018a). 5G NR. Technical Specification Group Radio Access Network, NR, and NG-RAN Overall Description (Release 15). Technical Specification (TS) 38.300, 3rd Generation Partnership Project (3GPP).

3GPP (2018b). 5G NR. User Equipment (UE) radio transmission and reception (Release 15). Technical Specification (TS) 38.101-1, 3rd Generation Partnership Project (3GPP).

3GPP (2019). 5G NR; Base Station (BS) conformance testing Part 1: Conducted conformance testing Release 15. Technical Specification (TS) 38.141-1, 3rd Generation Partnership Project (3GPP).

3GPP (2019). 5G NR. Physical channels and modulation (Release 15). Technical Specification (TS) 38.211, 3rd Generation Partnership Project (3GPP).

3GPP (2020). NG-RAN; Architecture description Release 15. Technical Specification (TS) 38.401, 3rd Generation Partnership Project 3GPP.

5G LENA (2023). 5G-LENA Simulator. `https://https://5g-lena.cttc.es/`. Acessado em 03/12/2023.

Abu-Rgheff, M. (2019). *5G Physical Layer Technologies*. IEEE Press. Wiley.

Ahmadi, S. (2019). *5G NR: Architecture, Technology, Implementation, and Operation of 3GPP New Radio Standards*. Elsevier Science.

AlQahtani, S. A. e Alhassany, M. (2013). Comparing different LTE scheduling schemes. Em *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, páginas 264–269.

Avramov, L. (2017). Data Center Benchmarking Terminology. RFC 8238.

Capozzi, F., Piro, G., Grieco, L. A., Boggia, G. e Camarda, P. (2013). Downlink Packet Scheduling in LTE Cellular Networks: Key Design Issues and a Survey . *IEEE Communications Surveys and Tutorials* , 15(2):678–700.

Chukwu, E. C., Abdullahi, U. S., Koyunlu, G., Sanusi, J., Sani, G. e Gangfada, I. A. (2020). Performance Evaluation of Multiplexed 5G-New Radio Network Services of Different Usage Scenarios. *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, páginas 335–342.

Dahlman, E., Parkvall, S. e Skold, J. (2018). *5G NR: The Next Generation Wireless Access Technology*. Academic Press, Inc., USA, 1st edition.

FAGUNDES, P. C. (2011). Processo de Decisão de Markov. Dissertação de Mestrado, Universidade Federal de Juiz de Fora.

Fehrenbach, T., Datta, R., Göktepe, B., Wirth, T. e Hellge, C. (2018). URLLC Services in 5G - Low Latency Enhancements for LTE.

Jain, R. (1991). *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling*. Wiley professional computing. Wiley.

Karimi, A., Pedersen, K. I. e Mogensen, P. (2020). Low-Complexity Centralized Multi-Cell Radio Resource Allocation for 5G URLLC. Em *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, páginas 1–6.

Katila, C. J., Buratti, C., Abrignani, M. D. e Verdone, R. (2017). Neighbors-Aware Proportional Fair scheduling for future wireless networks with mixed MAC protocols. *EURASIP Journal on Wireless Communications and Networking*, (93).

Kooshki, F., Armada, A. G., Mowla, M. M. e Flizikowski, A. (2023). Radio Resource Management Scheme for URLLC and eMBB Coexistence in a Cell-Less Radio Access Network. *IEEE Access*, 11:25090–25101.

Kumar, R., Sinwar, D. e Singh, V. (2024). QoS aware resource allocation for coexistence mechanisms between eMBB and URLLC: Issues, challenges, and future directions in 5G. *Computer Communications*, 213:208–235.

Liu, Z., Chen, X., Chen, Y. e Li, Z. (2019). Deep Reinforcement Learning Based Dynamic Resource Allocation in 5G Ultra-Dense Networks. Em *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, páginas 168–174.

Mamane, A., Fattah, M., El Ghazi, M., Bekkali, M., Younes, B. e Mazer, S. (2022). Scheduling Algorithms for 5G Networks and Beyond: Classification and Survey. *IEEE Access*, 10:1–1.

Maryam Imran Sheik Mamode, T. P. F. (2020). Survey of Scheduling Schemes in 5G Mobile Communication Systems. *Journal of Electrical Engineering, Electronics, Control and Computer Science – JEEECCS*, 6(20):21–30.

Mehmeti, F. e La Porta, T. F. (2022). Modeling and Analysis of mMTC Traffic in 5G Base Stations. Em *2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC)*, páginas 652–660.

Molisch, A. (2012). Wireless Communications. IEEE Press, página 888. Wiley.

Patriciello, N., Lagen, S., Bojovic, B. e Giupponi, L. (2019). An E2E Simulator for 5G NR Networks.

Rao, A., Legout, A., Lim, Y.-s., Towsley, D., Barakat, C. e Dabbous, W. (2011). Network characteristics of video streaming traffic. Em *Proceedings of the Seventh Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, New York, NY, USA. Association for Computing Machinery.

Riley, G. F. e Henderson, T. R. (2010). The ns-3 Network Simulator. Em Wehrle, K., Günes, M. e Gross, J., editores, *Modeling and Tools for Network Simulation*, páginas 15–34. Springer.

ROHDESCHWARZ (2023). 5G New Radio. ROHDESCHWARZ. Acessado em 30/05/2024.

Rusev, V. e Skorikov, A. (2023). The Asymptotics of Moments for the Remaining Time of Heavy-Tail Distributions. *Computer Sciences & Mathematics Forum*, 7(1).

Sauter, M. (2021). *From GSM to LTE-Advanced Pro and 5G: An Introduction to Mobile Networks and Mobile Broadband*. Wiley.

Zaidi, A., Athley, F., Medbo, J., Gustavsson, U., Durisi, G. e Chen, X. (2018). *5G Physical Layer: Principles, Models and Technology Components*. Academic Press, Inc., USA, 1st edition.

APÊNDICE A – ANEXOS

A.1 ED_19_CTTC-3GPP-CHANNEL-NUMS.CC

```
#include <cstdio>
#include <string.h>
#include "ns3/antenna-module.h"
#include "ns3/applications-module.h"
#include "ns3/config-store-module.h"
#include "ns3/config-store.h"
#include "ns3/core-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/internet-apps-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/network-module.h"
#include "ns3/nr-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/nr-mac-scheduler-ofdma-rr-weight.h"
#include <ns3/lte-module.h>
#include <ns3/position-allocator.h>
#include <ns3/lte-enb-rrc.h>
#include <ctime>
#include <cstdlib>
#include <iostream>
#include <fstream>
#include "ns3/rng-seed-manager.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("3gppChannelNumerologiesExample");

// Prints actual position of nodes
static void
nodePos(NodeContainer nodeType,const char* nodeName, std::ofstream& MyFile)
{
for (NodeContainer::Iterator j = nodeType.Begin(); j != nodeType.End(); ++j)
{
Ptr<Node> object = *j;
Ptr<MobilityModel> position = object->GetObject<MobilityModel>();
NS_ASSERT(position);
Vector pos = position->GetPosition();
MyFile << "" << pos.x << "," << pos.y << "," << pos.z << ","<< nodeName << "," << j-nodeType.Begin() << std::endl;
}
}

// iterate our nodes and print their position.


int
main(int argc, char* argv[])
{
// enable logging or not
bool logging = false;
if (logging)
{
//LogComponentEnable("UdpClient", LOG_LEVEL_INFO);
//LogComponentEnable("UdpServer", LOG_LEVEL_INFO);
//LogComponentEnable("LtePdcp", LOG_LEVEL_INFO);
//LogComponentEnable ("EvalvidClient", LOG_LEVEL_INFO);
//LogComponentEnable ("EvalvidServer", LOG_LEVEL_INFO);
LogComponentEnable ("NrMacSchedulerUeInfoRRW", LOG_LEVEL_DEBUG);
}

// set simulation time and mobility
double simTime = 1;           // seconds

// other simulation parameters default values
uint16_t numerology = 0;
uint16_t gNbNum = 1;
uint16_t eMBB = 1;
uint16_t mTC = 1;

double centralFrequency = 7e9;
double bandwidth = 100e6;
double txPower = 14;
double lambda = 1000;

//uint8_t fixedMcs = 28;
//bool useFixedMcs = true;
uint8_t scheduler = 1;
uint8_t seed= 1;

// Random Variable
auto xRand = CreateObject<UniformRandomVariable>();
xRand->SetAttribute("Min", DoubleValue(0));
xRand->SetAttribute("Max", DoubleValue(999));
auto yRand = CreateObject<UniformRandomVariable>();
yRand->SetAttribute("Min", DoubleValue(0));
yRand->SetAttribute("Max", DoubleValue(999));
```

```
// Where we will store the output files.
std::string simTag = "results.xml";
std::string outputDir = "./";

// Position log file
std::string posFileOut;
std::string posFileIn;

CommandLine cmd(__FILE__);
cmd.AddValue("scheduler", "Mac Scheduler 1=RR, 2=PF, 3=MR, 4=RRW", scheduler);
cmd.AddValue("simTime", "The simulation time to run.", simTime);
cmd.AddValue("posFileOut", "Position log file.", posFileOut);
cmd.AddValue("posFileIn", "Position log file.", posFileIn);
cmd.AddValue("seed", "Seed of random variable.", seed);
cmd.AddValue("eMBB", "eMBB users in attached.", eMBB);
cmd.AddValue("mTC", "mTC users in attached.", mTC);
cmd.AddValue("gNbNum", "The number of gNbs in multiple-ue topology", gNbNum);
cmd.AddValue("numerology", "The numerology to be used.", numerology);
cmd.AddValue("txPower", "Tx power to be configured to gNB", txPower);
cmd.AddValue("simTag",
"tag to be appended to output filenames to distinguish simulation campaigns",
simTag);
cmd.AddValue("outputDir", "directory where to store simulation results", outputDir);
cmd.AddValue("frequency", "The system frequency", centralFrequency);
cmd.AddValue("bandwidth", "The system bandwidth", bandwidth);
cmd.AddValue("lambda", "Number of UDP packets per second", lambda);
//cmd.AddValue(
//    "fixedMcs",
//    "The fixed MCS that will be used in this example if useFixedMcs is configured to true (1).",
//    fixedMcs);
//cmd.AddValue("useFixedMcs",
//            "Whether to use fixed mcs, normally used for testing purposes",
//            useFixedMcs);

cmd.Parse(argc, argv);

printf("\n\n###################################\n####### Starting Simulation #######\n###################################\n");
printf("\n Central Frequency: %.02f Hz",centralFrequency);
printf("\n Bandwidth:          %.02f Hz",bandwidth);
printf("\n Numerology: %d",numerology);
printf("\n eMBB users:  %02d",eMBB);
printf("\n mTC users: %02d",mTC);
printf("\n Simulation Time:   %.02f s",simTime);

// Use current time as seed for random generator
ns3::RngSeedManager::SetSeed(seed);
ns3::RngSeedManager::SetRun(4);


// Setup NR Simulation
Ptr<NrHelper> nrHelper = CreateObject<NrHelper>();

// Configure Bandwidth
CcBwpCreator ccBwpCreator;
//const uint8_t numCcPerBand = 1; // in this example, both bands have a single CC
//BandwidthPartInfo::Scenario scenario = BandwidthPartInfo::UMa_nLoS;
//BandwidthPartInfo::Scenario scenario = BandwidthPartInfo::RMa_LoS;

// Create the configuration for the CcBwpHelper. SimpleOperationBandConf creates
// a single BWP per CC
CcBwpCreator::SimpleOperationBandConf bandConf(3.51e9,//Frequência Central
20e6,//Largura de Banda
1,//Component Carrier = 1
BandwidthPartInfo::RMa_LoS);
//CcBwpCreator::SimpleOperationBandConf bandConf(centralFrequency,
//                                               bandwidth,
//                                               numCcPerBand,
//                                               scenario);

// By using the configuration created, it is time to make the operation bands
OperationBandInfo band = ccBwpCreator.CreateOperationBandContiguousCc(bandConf);

// Force Dynamic fixedMCS
nrHelper->SetSchedulerAttribute("FixedMcsDl", BooleanValue(false));
nrHelper->SetSchedulerAttribute("FixedMcsUl", BooleanValue(false));

/*
* Initialize channel and pathloss, plus other things inside band1. If needed,
* the band configuration can be done manually, but we leave it for more
* sophisticated examples. For the moment, this method will take care
* of all the spectrum initialization needs.
*/
nrHelper->InitializeOperationBand(&band);

BandwidthPartInfoPtrVector allBwps = CcBwpCreator::GetAllBwps({band});

nrHelper->SetGnbPhyAttribute("TxPower", DoubleValue(30));
nrHelper->SetGnbPhyAttribute("Numerology", UintegerValue(0));

// UE TxPower
nrHelper->SetUePhyAttribute("TxPower", DoubleValue(28));

// Scheduler
switch(scheduler)
{
case 1:
nrHelper->SetSchedulerTypeId(TypeId::LookupByName("ns3::NrMacSchedulerOfdmaRR"));
printf("\n\n####### Scheduler RR #######\n");
break;
```

```
case 2:
nrHelper->SetSchedulerTypeId(TypeId::LookupByName("ns3::NrMacSchedulerOfdmaPF"));
printf("\n\n####### Scheduler PF #######\n");
break;
case 3:
nrHelper->SetSchedulerTypeId(TypeId::LookupByName("ns3::NrMacSchedulerOfdmaMR"));
printf("\n\n####### Scheduler MR #######\n");
break;
case 4:
nrHelper->SetSchedulerTypeId(TypeId::LookupByName("ns3::NrMacSchedulerOfdmaRRW"));
printf("\n\n####### Scheduler RRW #######\n");
break;
}

Config::SetDefault("ns3::LteRlcUm::MaxTxBufferSize", UintegerValue(4294967295));

// Antennas for all the UEs
nrHelper->SetUeAntennaAttribute("NumRows", UintegerValue(2));
nrHelper->SetUeAntennaAttribute("NumColumns", UintegerValue(4));
nrHelper->SetUeAntennaAttribute("AntennaElement",
PointerValue(CreateObject<IsotropicAntennaModel>()));

// Antennas for all the gNbs
nrHelper->SetGnbAntennaAttribute("NumRows", UintegerValue(4));
nrHelper->SetGnbAntennaAttribute("NumColumns", UintegerValue(8));
nrHelper->SetGnbAntennaAttribute("AntennaElement",
PointerValue(CreateObject<ThreeGppAntennaModel>()));

// Beamforming method
Ptr<IdealBeamformingHelper> idealBeamformingHelper = CreateObject<IdealBeamformingHelper>();
idealBeamformingHelper->SetAttribute("BeamformingMethod",
TypeIdValue(DirectPathBeamforming::GetTypeId()));
nrHelper->SetBeamformingHelper(idealBeamformingHelper);

Config::SetDefault("ns3::ThreeGppChannelModel::UpdatePeriod", TimeValue(MilliSeconds(0)));
//nrHelper->SetChannelConditionModelAttribute ("UpdatePeriod", TimeValue(MilliSeconds(0)));
nrHelper->SetPathlossAttribute("ShadowingEnabled", BooleanValue(false));



// Create EPC helper
Ptr<NrPointToPointEpcHelper> epcHelper = CreateObject<NrPointToPointEpcHelper>();
nrHelper->SetEpcHelper(epcHelper);

// MME > gNB - S1uLinkDataRate 10Gb/s
epcHelper->SetAttribute("S1uLinkDelay", TimeValue(MilliSeconds(0)));
epcHelper->SetAttribute("S1uLinkMtu", UintegerValue(9000));
epcHelper->SetAttribute("S1uLinkDataRate", DataRateValue(DataRate("100Gb/s")));

// gNb routing between Bearer and bandwidh part (https://cttc-lena.gitlab.io/nr/html/classns3_1_1_bwp_manager_algorithm_static.html)
//uint32_t bwpIdForBearer = 0;

// Increase periodicity (more users)
Config::SetDefault("ns3::LteEnbRrc::SrsPeriodicity", UintegerValue(320));

// Initialize nrHelper
nrHelper->Initialize();

//Create the gNB and UE nodes according to the network topology
NodeContainer gNbNodes;
NodeContainer eMBBue;
NodeContainer mTCue;
MobilityHelper mobility;
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
Ptr<ListPositionAllocator> bsPositionAlloc = CreateObject<ListPositionAllocator>();
Ptr<ListPositionAllocator> utPositionAlloc = CreateObject<ListPositionAllocator>();

// Creating Topology
//const double gNbHeight = 15;
const double ueHeight = 1.5;

gNbNodes.Create(1);
eMBBue.Create(eMBB);
mTCue.Create(mTC);

int32_t xValue = 0.0;
int32_t yValue = 0.0;

bsPositionAlloc->Add(Vector(0.0, 0.0, 50));
//utPositionAlloc->Add(posFileIn, ueHeight);

mobility.SetPositionAllocator(bsPositionAlloc);
mobility.Install(gNbNodes);

// Configuração da Posição das UEs


for (uint32_t i = 1; i <= eMBB ; i++)
{

xValue = static_cast<int> (i) * (40000/eMBB);
yValue = 0;

utPositionAlloc->Add (Vector (xValue, yValue, ueHeight));
}

for (uint32_t i = 1; i <= mTC ; i++)
{
```

```
xValue = static_cast<int> (i) * (40000/mTC + 20);
yValue = 0;

utPositionAlloc->Add (Vector (xValue, yValue, ueHeight));
}

mobility.SetPositionAllocator(utPositionAlloc);
mobility.Install(eMBBue);
mobility.Install(mTCue);

// Create posFile
std::ofstream MyFile(posFileOut);
MyFile << "x,y,z,type,tag" << std::endl;
nodePos(gNbNodes,"gNodeB",MyFile);
nodePos(eMBBue,"eMBB",MyFile);
nodePos(mTCue,"mTC",MyFile);
MyFile.close();

// Install NR net devices
NetDeviceContainer gNbNetDev = nrHelper->InstallGnbDevice(gNbNodes, allBwps);
NetDeviceContainer eMBBueNetDev = nrHelper->InstallUeDevice(eMBBue, allBwps);
NetDeviceContainer mTCueNetDev = nrHelper->InstallUeDevice(mTCue, allBwps);

// When all the configuration is done, explicitly call UpdateConfig ()
for (auto it = gNbNetDev.Begin(); it != gNbNetDev.End(); ++it)
{
DynamicCast<NrGnbNetDevice>(*it)->UpdateConfig();
}
for (auto it = eMBBueNetDev.Begin(); it != eMBBueNetDev.End(); ++it)
{
DynamicCast<NrUeNetDevice>(*it)->UpdateConfig();
}
for (auto it = mTCueNetDev.Begin(); it != mTCueNetDev.End(); ++it)
{
DynamicCast<NrUeNetDevice>(*it)->UpdateConfig();
}

// create the internet and install the IP stack on the UEs
// get SGW/PGW and create a single RemoteHost
Ptr<Node> pgw = epcHelper->GetPgwNode();
NodeContainer remoteHostContainer;
remoteHostContainer.Create(1);
Ptr<Node> remoteHost = remoteHostContainer.Get(0);
InternetStackHelper internet;
internet.Install(remoteHostContainer);
internet.Install(eMBBue);
internet.Install(mTCue);

// connect a remoteHost to pgw. Setup routing too
PointToPointHelper p2ph;
p2ph.SetDeviceAttribute("DataRate", DataRateValue(DataRate("100Gb/s")));
p2ph.SetDeviceAttribute("Mtu", UintegerValue(9000));
p2ph.SetChannelAttribute("Delay", TimeValue(Seconds(0.000)));
NetDeviceContainer internetDevices = p2ph.Install(pgw, remoteHost);
Ipv4AddressHelper ipv4h;
ipv4h.SetBase("1.0.0.0", "255.0.0.0");
Ipv4InterfaceContainer internetIpIfaces = ipv4h.Assign(internetDevices);
Ipv4StaticRoutingHelper ipv4RoutingHelper;
Ptr<Ipv4StaticRouting> remoteHostStaticRouting =
ipv4RoutingHelper.GetStaticRouting(remoteHost->GetObject<Ipv4>());
remoteHostStaticRouting->AddNetworkRouteTo(Ipv4Address("7.0.0.0"), Ipv4Mask("255.0.0.0"), 1);

Ipv4InterfaceContainer eMBBueIpIface = epcHelper->AssignUeIpv4Address(NetDeviceContainer(eMBBueNetDev));
Ipv4InterfaceContainer mTCueIpIface = epcHelper->AssignUeIpv4Address(NetDeviceContainer(mTCueNetDev));

// Set the default gateway for the UEs eMMB
for (uint32_t j = 0; j < eMBBue.GetN(); ++j)
{
Ptr<Ipv4StaticRouting> ueStaticRouting =
ipv4RoutingHelper.GetStaticRouting(eMBBue.Get(j)->GetObject<Ipv4>());
ueStaticRouting->SetDefaultRoute(epcHelper->GetUeDefaultGatewayAddress(), 1);
}
// Set the default gateway for the UEs mTC
for (uint32_t j = 0; j < mTCue.GetN(); ++j)
{
Ptr<Ipv4StaticRouting> ueStaticRouting =
ipv4RoutingHelper.GetStaticRouting(mTCue.Get(j)->GetObject<Ipv4>());
ueStaticRouting->SetDefaultRoute(epcHelper->GetUeDefaultGatewayAddress(), 1);
}

// attach UEs to the closest eNB
nrHelper->AttachToClosestEnb(eMBBueNetDev, gNbNetDev);
nrHelper->AttachToClosestEnb(mTCueNetDev, gNbNetDev);

NS_LOG_INFO ("Create Applications.");

// Install Application Video Server
uint16_t eMBBport = 8000;
ApplicationContainer clientOnOff;
ApplicationContainer serverOnOff;
for (uint32_t i = 0; i < eMBBue.GetN(); ++i)
{
//Ptr<UniformRandomVariable> xRand = CreateObject<UniformRandomVariable>();
uint32_t myRandomNoX = xRand->GetValue();
printf("\n This is the RANDOM Number: %d",myRandomNoX);
printf("\n## eMBB UE: %02d port: %d",i,eMBBport+i);
PacketSinkHelper sinkOnOff ("ns3::UdpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), eMBBport+i));
clientOnOff.Add (sinkOnOff.Install (eMBBue.Get (i)));
```

```
OnOffHelper onoff ("ns3::UdpSocketFactory", InetSocketAddress (eMBBueIpIface.GetAddress (i), eMBBport+i));
//onoff.SetAttribute("OnTime", StringValue("ns3::WeibullRandomVariable[Shape=10.2063|Scale=57480.9]"));
onoff.SetAttribute("OnTime", StringValue("ns3::ParetoRandomVariable[Shape=1.3|Scale=0.003846154]"));
onoff.SetAttribute("OffTime", StringValue("ns3::ExponentialRandomVariable[Mean=0.0166666667|Bound=0.0]"));
onoff.SetAttribute("PacketSize", UintegerValue (1024));
onoff.SetAttribute("DataRate", DataRateValue(DataRate("5.6Mb/s")));
//onoff.SetAttribute("MaxBytes",UintegerValue(0));
onoff.SetAttribute ("StartTime", TimeValue (MilliSeconds(myRandomNoX)));
serverOnOff.Add (onoff.Install (remoteHost));
}
serverOnOff.Stop (Seconds (simTime));
clientOnOff.Stop (Seconds (simTime));

// Install Application mTC
uint16_t mTCport = 9000;
ApplicationContainer clientApp;
ApplicationContainer serverApp;
for (uint32_t i = 0; i < mTCue.GetN(); ++i)
{
//Ptr<UniformRandomVariable> yRand = CreateObject<UniformRandomVariable>();
uint32_t myRandomNoY = yRand->GetValue();
printf("\n This is the RANDOM Number: %d",myRandomNoY);
printf("\n##  mTC UE: %02d port: %d", i, mTCport+i);
PacketSinkHelper dlPacketSinkHelper ("ns3::UdpSocketFactory", InetSocketAddress (Ipv4Address::GetAny (), mTCport+i ));
clientApp.Add (dlPacketSinkHelper.Install (mTCue.Get(i)));
UdpClientHelper dlClient (mTCueIpIface.GetAddress (i), mTCport+i );
dlClient.SetAttribute ("Interval", TimeValue (MilliSeconds(5)));
dlClient.SetAttribute ("PacketSize", UintegerValue (200));
dlClient.SetAttribute ("StartTime", TimeValue (MilliSeconds(myRandomNoY)));
dlClient.SetAttribute ("MaxPackets", UintegerValue (4294967295));
serverApp.Add (dlClient.Install (remoteHost));
}
printf("\n");
clientApp.Stop (Seconds (simTime));
serverApp.Stop (Seconds (simTime));

// Flow monitor
Ptr<FlowMonitor> flowMonitor;
FlowMonitorHelper flowHelper;
flowMonitor = flowHelper.Install(eMBBue);
flowMonitor = flowHelper.Install(mTCue);
flowMonitor = flowHelper.Install(remoteHostContainer);

NS_LOG_INFO("Run Simulation.");
Simulator::Stop(Seconds(simTime));
Simulator::Run();

flowMonitor->SerializeToXmlFile(simTag, false, false);

Simulator::Destroy();


printf("\n######## End of Simulation ########\n");

NS_LOG_INFO("Done.");
return 0;
}

/*
# Added 10s application finishes before simulation.
# Flow Monitor only in nodes.

*/

*/
```

A.2 NR-MAC-SCHEDULER-DPS.CC

```
/* -*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*- */

// Copyright (c) 2019 Centre Tecnologic de Telecomunicacions de Catalunya (CTTC)
//
// SPDX-License-Identifier: GPL-2.0-only

#include "nr-mac-scheduler-ofdma-rr-weight.h"

#include "nr-mac-scheduler-ue-info-rr-weight.h"

#include <ns3/double.h>
#include <ns3/log.h>

#include <algorithm>

namespace ns3
{

NS_LOG_COMPONENT_DEFINE("NrMacSchedulerOfdmaRRW");
NS_OBJECT_ENSURE_REGISTERED(NrMacSchedulerOfdmaRRW);



TypeId
NrMacSchedulerOfdmaRRW::GetTypeId()
{
static TypeId tid =
TypeId("ns3::NrMacSchedulerOfdmaRRW")
.SetParent<NrMacSchedulerOfdmaRR>()
.AddConstructor<NrMacSchedulerOfdmaRRW>()
.AddAttribute("FairnessIndex",
"Value (between 0 and 1) that defines the RRW metric (1 is the "
"traditional 3GPP RRW, 0 is RR in throughput",
DoubleValue(1),
MakeDoubleAccessor(&NrMacSchedulerOfdmaRRW::SetFairnessIndex,
&NrMacSchedulerOfdmaRRW::GetFairnessIndex),
MakeDoubleChecker<float>(0, 1))
.AddAttribute(
"LastAvgTPutWeight",
"Weight of the last average throughput in the average throughput calculation",
DoubleValue(99),
MakeDoubleAccessor(&NrMacSchedulerOfdmaRRW::SetTimeWindow,
&NrMacSchedulerOfdmaRRW::GetTimeWindow),
MakeDoubleChecker<float>(0));
return tid;
}

NrMacSchedulerOfdmaRRW::NrMacSchedulerOfdmaRRW()
: NrMacSchedulerOfdmaRR()
{
}

void
NrMacSchedulerOfdmaRRW::SetFairnessIndex(double v)
{
NS_LOG_FUNCTION(this);
m_alpha = v;
}

double
NrMacSchedulerOfdmaRRW::GetFairnessIndex() const
{
NS_LOG_FUNCTION(this);
return m_alpha;
}

void
NrMacSchedulerOfdmaRRW::SetTimeWindow(double v)
{
NS_LOG_FUNCTION(this);
m_timeWindow = v;
}

double
NrMacSchedulerOfdmaRRW::GetTimeWindow() const
{
NS_LOG_FUNCTION(this);
return m_timeWindow;
}

std::shared_ptr<NrMacSchedulerUeInfo>
NrMacSchedulerOfdmaRRW::CreateUeRepresentation(
const NrMacCschedSapProvider::CschedUeConfigReqParameters& params) const
{
NS_LOG_FUNCTION(this);
return std::make_shared<NrMacSchedulerUeInfoRRW>(
m_alpha,
params.m_rnti,
params.m_beamConfId,
std::bind(&NrMacSchedulerOfdmaRRW::GetNumRbPerRbg, this));
}

std::function<bool(const NrMacSchedulerNs3::UePtrAndBufferReq& lhs,
const NrMacSchedulerNs3::UePtrAndBufferReq& rhs)>
NrMacSchedulerOfdmaRRW::GetUeCompareDlFn() const
{
```

```
return NrMacSchedulerUeInfoRRW::CompareUeWeightsDl;
}

std::function<bool(const NrMacSchedulerNs3::UePtrAndBufferReq& lhs,
const NrMacSchedulerNs3::UePtrAndBufferReq& rhs)>
NrMacSchedulerOfdmaRRW::GetUeCompareUlFn() const
{
return NrMacSchedulerUeInfoRRW::CompareUeWeightsUl;
}

void
NrMacSchedulerOfdmaRRW::AssignedDlResources(const UePtrAndBufferReq& ue,
[[maybe_unused]] const FTResources& assigned,
const FTResources& totAssigned) const
{
NS_LOG_FUNCTION(this);
auto uePtr = std::dynamic_pointer_cast<NrMacSchedulerUeInfoRRW>(ue.first);
uePtr->UpdateDlRRWMetric(totAssigned, m_timeWindow, m_dlAmc);

}

void
NrMacSchedulerOfdmaRRW::NotAssignedDlResources(
const NrMacSchedulerNs3::UePtrAndBufferReq& ue,
[[maybe_unused]] const NrMacSchedulerNs3::FTResources& assigned,
const NrMacSchedulerNs3::FTResources& totAssigned) const
{
NS_LOG_FUNCTION(this);
auto uePtr = std::dynamic_pointer_cast<NrMacSchedulerUeInfoRRW>(ue.first);
uePtr->UpdateDlRRWMetric(totAssigned, m_timeWindow, m_dlAmc);
}

void
NrMacSchedulerOfdmaRRW::AssignedUlResources(const UePtrAndBufferReq& ue,
[[maybe_unused]] const FTResources& assigned,
const FTResources& totAssigned) const
{
NS_LOG_FUNCTION(this);
auto uePtr = std::dynamic_pointer_cast<NrMacSchedulerUeInfoRRW>(ue.first);
uePtr->UpdateUlRRWMetric(totAssigned, m_timeWindow, m_ulAmc);
}

void
NrMacSchedulerOfdmaRRW::NotAssignedUlResources(
const NrMacSchedulerNs3::UePtrAndBufferReq& ue,
[[maybe_unused]] const NrMacSchedulerNs3::FTResources& assigned,
const NrMacSchedulerNs3::FTResources& totAssigned) const
{
NS_LOG_FUNCTION(this);
auto uePtr = std::dynamic_pointer_cast<NrMacSchedulerUeInfoRRW>(ue.first);
uePtr->UpdateUlRRWMetric(totAssigned, m_timeWindow, m_ulAmc);
}

void
NrMacSchedulerOfdmaRRW::BeforeDlSched(const UePtrAndBufferReq& ue,
const FTResources& assignableInIteration) const
{
NS_LOG_FUNCTION(this);
auto uePtr = std::dynamic_pointer_cast<NrMacSchedulerUeInfoRRW>(ue.first);
uePtr->CalculatePotentialTPutDl(assignableInIteration, m_dlAmc);
}

void
NrMacSchedulerOfdmaRRW::BeforeUlSched(const UePtrAndBufferReq& ue,
const FTResources& assignableInIteration) const
{
NS_LOG_FUNCTION(this);
auto uePtr = std::dynamic_pointer_cast<NrMacSchedulerUeInfoRRW>(ue.first);
uePtr->CalculatePotentialTPutUl(assignableInIteration, m_ulAmc);
}

} // namespace ns3
```

A.3 NR-MAC-SCHEDULER-DPS.H

```
/* -*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*- */

// Copyright (c) 2019 Centre Tecnologic de Telecomunicacions de Catalunya (CTTC)
//
// SPDX-License-Identifier: GPL-2.0-only

#pragma once
#include "nr-mac-scheduler-ofdma-rr.h"

namespace ns3
{

/**
* \ingroup scheduler
* \brief Assign frequencies in a proportional fair fashion
*
* Sort the UE by their current throughput. Number of symbols is fixed depending
* on the beam requirements.
*
* Details of the sorting function in the class NrMacSchedulerUeInfoRRW.
*/
class NrMacSchedulerOfdmaRRW : public NrMacSchedulerOfdmaRR
{
public:
/**
* \brief GetTypeId
* \return The TypeId of the class
*/
static TypeId GetTypeId();
/**
* \brief NrMacSchedulerOfdmaRRW constructor
*/
NrMacSchedulerOfdmaRRW();

/**
* \brief ~NrMacSchedulerOfdmaRRW deconstructor
*/
~NrMacSchedulerOfdmaRRW() override
{
}

/**
* \brief Set the value of attribute "FairnessIndex"
* \param v
*/
void SetFairnessIndex(double v);

/**
* \brief Get the value of attribute "FairnessIndex"
* @return
*/
double GetFairnessIndex() const;

/**
* \brief Set the attribute "LastAvgTPutWeight"
* \param v the value to save
*/
void SetTimeWindow(double v);
/**
* \brief Get the attribute "LastAvgTPutWeight"
* \return the value of the attribute
*/
double GetTimeWindow() const;

protected:
// inherit
/**
* \brief Create an UE representation of the type NrMacSchedulerUeInfoRRW
* \param params parameters
* \return NrMacSchedulerUeInfoRR instance
*/
std::shared_ptr<NrMacSchedulerUeInfo> CreateUeRepresentation(
const NrMacCschedSapProvider::CschedUeConfigReqParameters& params) const override;

/**
* \brief Return the comparison function to sort DL UE according to the scheduler policy
* \return a pointer to NrMacSchedulerUeInfoRRW::CompareUeWeightsDl
*/
std::function<bool(const NrMacSchedulerNs3::UePtrAndBufferReq& lhs,
const NrMacSchedulerNs3::UePtrAndBufferReq& rhs)>
GetUeCompareDlFn() const override;

/**
* \brief Return the comparison function to sort UL UE according to the scheduler policy
* \return a pointer to NrMacSchedulerUeInfoRRW::CompareUeWeightsUl
*/
std::function<bool(const NrMacSchedulerNs3::UePtrAndBufferReq& lhs,
const NrMacSchedulerNs3::UePtrAndBufferReq& rhs)>
GetUeCompareUlFn() const override;

/**
* \brief Update the UE representation after a symbol (DL) has been assigned to it
* \param ue UE to which a symbol has been assigned
* \param assigned the amount of resources assigned
* \param totAssigned the total amount of resources assigned in the slot
*
```

```
* The DL metrics (current Throughput and average Throughput) will be updated
* by calling the NrMacSchedulerUeInfoRRW::UpdateDlRRWMetric, which in turn will
* call NrMacSchedulerUeInfo::UpdateDlMetric in order to get the tbSize
* based on the resources assigned to the user. This will help the sorting
* function to sort the UEs for resource allocation.
*/
void AssignedDlResources(const UePtrAndBufferReq& ue,
const FTResources& assigned,
const FTResources& totAssigned) const override;

/**
* \brief Update DL metrics by calling NrMacSchedulerUeInfoRRW::UpdateRRWDlMetric
* \param ue UE to update (ue that didn't get any resources)
* \param notAssigned the amount of resources not assigned
* \param totalAssigned the total amount of resources assigned in the slot
*
* Even if the UE did not get any resource assigned, change its current throughput
* over the total number of symbols assigned.
*
* The DL metrics (current Throughput and average Throughput) will be updated
* by calling the NrMacSchedulerUeInfoRRW::UpdateDlRRWMetric, which in turn will
* call NrMacSchedulerUeInfo::UpdateDlMetric in order to get the tbSize
* based on the resources assigned to the user. Since no resources have been
* assigned, the tbSize will be zero. This will help the sorting function to
* sort the UEs for resource allocation.
*/
void NotAssignedDlResources(const UePtrAndBufferReq& ue,
const FTResources& notAssigned,
const FTResources& totalAssigned) const override;

/**
* \brief Update the UE representation after a symbol (UL) has been assigned to it
* \param ue UE to which a symbol has been assigned
* \param assigned the amount of resources assigned
* \param totAssigned the total amount of resources assigned in the slot
*
* The UL metrics (current Throughput and average Throughput) will be updated
* by calling the NrMacSchedulerUeInfoRRW::UpdateUlRRWMetric, which in turn will
* call NrMacSchedulerUeInfo::UpdateUlMetric in order to get the tbSize
* based on the resources assigned to the user. This will help the sorting
* function to sort the UEs for resource allocation.
*/
void AssignedUlResources(const UePtrAndBufferReq& ue,
const FTResources& assigned,
const FTResources& totAssigned) const override;

/**
* \brief Update UL metrics by calling NrMacSchedulerUeInfoRRW::UpdateRRWUlMetric
* \param ue UE to update (ue that didn't get any resources)
* \param notAssigned the amount of resources not assigned
* \param totalAssigned the total amount of resources assigned in the slot
*
* Even if the UE did not get any resource assigned, change its current throughput
* over the total number of symbols assigned.
*
* The UL metrics (current Throughput and average Throughput) will be updated
* by calling the NrMacSchedulerUeInfoRRW::UpdateUlRRWMetric, which in turn will
* call NrMacSchedulerUeInfo::UpdateUlMetric in order to get the tbSize
* based on the resources assigned to the user. Since no resources have been
* assigned, the tbSize will be zero. This will help the sorting function to
* sort the UEs for resource allocation.
*/
void NotAssignedUlResources(const UePtrAndBufferReq& ue,
const FTResources& notAssigned,
const FTResources& totalAssigned) const override;

/**
* \brief Calculate the potential throughput for the DL based on the available resources
* \param ue UE to which a rgb has been assigned
* \param assignableInIteration the minimum amount of resources to be assigned
*
* Calculates the the potential throughput by calling
* NrMacSchedulerUeInfoRRW::CalculatePotentialTPutDl.
*/
void BeforeDlSched(const UePtrAndBufferReq& ue,
const FTResources& assignableInIteration) const override;

/**
* \brief Calculate the potential throughput for the UL based on the available resources
* \param ue UE to which a rbg has been assigned
* \param assignableInIteration the minimum amount of resources to be assigned
*
* Calculates the the potential throughput by calling
* NrMacSchedulerUeInfoRRW::CalculatePotentialTPutUl.
*/
void BeforeUlSched(const UePtrAndBufferReq& ue,
const FTResources& assignableInIteration) const override;

private:
double m_timeWindow{
99.0}; //!< Time window to calculate the throughput. Better to make it an attribute.
double m_alpha{0.0}; //!< RRW Fairness index
};

} // namespace ns3
```

A.4 NR-MAC-SCHEDULER-UE-INFO-DPS.CC

```
/* -*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*- */
/*
*   Copyright (c) 2019 Centre Tecnologic de Telecomunicacions de Catalunya (CTTC)
*
*   This program is free software; you can redistribute it and/or modify
*   it under the terms of the GNU General Public License version 2 as
*   published by the Free Software Foundation;
*
*   This program is distributed in the hope that it will be useful,
*   but WITHOUT ANY WARRANTY; without even the implied warranty of
*   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*   GNU General Public License for more details.
*
*   You should have received a copy of the GNU General Public License
*   along with this program; if not, write to the Free Software
*   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*
*/

#include "nr-mac-scheduler-ue-info-rr-weight.h"
#include <ns3/log.h>
#include <string>
#include <iostream>


namespace ns3 {

NS_LOG_COMPONENT_DEFINE ("NrMacSchedulerUeInfoRRW");

void
NrMacSchedulerUeInfoRRW::UpdateDlRRWMetric (const NrMacSchedulerNs3::FTResources &totAssigned,
double timeWindow,
const Ptr<const NrAmc> &amc)
{
int dec =0;
//NS_LOG_FUNCTION (this);

NrMacSchedulerUeInfo::UpdateDlMetric (amc);



uint32_t tbSize = 0;

// OK: this is for RRW. I dont need that, but its here.
// ------------------------------------
for (const auto &it:m_dlTbSize)
{
tbSize += it;
}
m_currTputDl = static_cast<double> (tbSize) / (totAssigned.m_sym);
m_avgTputDl = ((1.0 - (1.0 / static_cast<double> (timeWindow))) * m_lastAvgTputDl) +
((1.0 / timeWindow) * m_currTputDl);

NS_LOG_DEBUG ("Update DL RRW Metric for UE " << m_rnti << " DL TBS: " << tbSize <<
" Updated currTputDl " << m_currTputDl << " avgTputDl " << m_avgTputDl <<
" over n. of syms: " << +totAssigned.m_sym <<
", last Avg TH Dl " << m_lastAvgTputDl <<
" total sym assigned " << static_cast<uint32_t> (totAssigned.m_sym) <<
" updated DL metric: " << m_potentialTputDl / std::max (1E-9, m_avgTputDl));
// ------------------------------------

// Now: what I have to do is recalculate the dynamic priority for this UE


//dec = (m_dlRBG - m_lastdlRBG)/static_cast<uint32_t> (totAssigned.m_sym);

dec = m_dlRBG == static_cast<uint32_t> (m_lastdlRBG) ? 0 : 1;

m_dynamicPriority -= dec;

m_inc = dec == 0 ? m_inc+1 : m_inc;



// if this UE does not receive an RGB in 12, dynamic priorty is incremented!
// why 12 ? good question ... could be different? yes, sure. 12 is good? I dont know ...
if (m_inc == 53) {
m_dynamicPriority++;

NS_LOG_DEBUG ("    >>>>> "<< Now().GetSeconds() << "Update DL RRW Metric for UE " << m_rnti << " DL TBS: " << tbSize <<
" over n. of syms: " << +totAssigned.m_sym <<
" m_lastdlRBG: " << m_lastdlRBG <<
" m_dlRBG: " << m_dlRBG <<
" totAssigned.m_sym: " << static_cast<uint32_t> (totAssigned.m_sym) <<
" dynamic prio: " << m_dynamicPriority <<
" incremento: " << m_inc <<
" decremento: " << dec);


m_inc=0;
}

// =   m_dlRBG/+totAssigned.m_sym == 0 ? (m_dynamicPriority+1) : m_dynamicPriority;
m_dynamicPriority = m_dynamicPriority < 0 ? 0 : m_dynamicPriority;
m_dynamicPriority = m_dynamicPriority > 20 ? 20 : m_dynamicPriority;
```

```
if (m_dlRBG != static_cast<uint32_t> (m_lastdlRBG)) {
std::string part = "SIM" ;
}

NS_LOG_DEBUG ("    >>>>> Update DL RRW Metric for UE " << m_rnti << " DL TBS: " << tbSize <<
" over n. of syms: " << +totAssigned.m_sym <<
" m_lastdlRBG: " << m_lastdlRBG <<
" m_dlRBG: " << m_dlRBG <<
" totAssigned.m_sym: " << static_cast<uint32_t> (totAssigned.m_sym) <<
" dynamic prio: " << m_dynamicPriority <<
" incremento: " << m_inc <<
" decremento: " << dec <<
"      " << part.c_str());


NS_LOG_DEBUG("    >>>>> UE " << m_rnti << " DL TBS: " << tbSize <<
" over n. of syms: " << +totAssigned.m_sym <<
" m_lastdlRBG: " << m_lastdlRBG <<
" m_dlRBG: " << m_dlRBG <<
" totAssigned.m_sym: " << static_cast<uint32_t> (totAssigned.m_sym) <<
" dynamic prio: " << m_dynamicPriority <<
" incremento: " << m_inc <<
" decremento: " << dec) ;

m_lastdlRBG = m_dlRBG;
}

void
NrMacSchedulerUeInfoRRW::UpdateUlRRWMetric (const NrMacSchedulerNs3::FTResources &totAssigned,
double timeWindow,
const Ptr<const NrAmc> &amc)
{
NS_LOG_FUNCTION (this);

NrMacSchedulerUeInfo::UpdateUlMetric (amc);

m_currTputUl = static_cast<double> (m_ulTbSize) / (totAssigned.m_sym);
m_avgTputUl = ((1.0 - (1.0 / static_cast<double> (timeWindow))) * m_lastAvgTputUl) +
((1.0 / timeWindow) * m_currTputUl);

NS_LOG_DEBUG ("Update UL RRW Metric for UE " << m_rnti << " UL TBS: " << m_ulTbSize <<
" Updated currTputUl " << m_currTputUl << " avgTputUl " << m_avgTputUl <<
" over n. of syms: " << +totAssigned.m_sym <<
", last Avg TH Ul " << m_lastAvgTputUl <<
" total sym assigned " << static_cast<uint32_t> (totAssigned.m_sym) <<
" updated UL metric: " << m_potentialTputUl / std::max (1E-9, m_avgTputUl));
}

void
NrMacSchedulerUeInfoRRW::CalculatePotentialTPutDl (const NrMacSchedulerNs3::FTResources &assignableInIteration,
const Ptr<const NrAmc> &amc)
{
NS_LOG_FUNCTION (this);

uint32_t rbsAssignable = assignableInIteration.m_rbg * GetNumRbPerRbg ();
// Since we compute a new potential throughput every time, there is no harm
// in initializing it to zero here.
m_potentialTputDl = 0.0;

if (this->m_dlCqi.m_ri == 1)
{
std::vector<uint8_t>::const_iterator mcsIt;
mcsIt = std::max_element (m_dlMcs.begin(), m_dlMcs.end());
m_potentialTputDl =  amc->CalculateTbSize (*mcsIt, rbsAssignable);
}

if (this->m_dlCqi.m_ri == 2)
{
//if the UE supports two streams potential throughput is the sum of
//both the TBs.
for (const auto &it:m_dlMcs)
{
m_potentialTputDl +=  amc->CalculateTbSize (it, rbsAssignable);
}
}

m_potentialTputDl /= assignableInIteration.m_sym;

NS_LOG_INFO ("UE " << m_rnti << " potentialTputDl " << m_potentialTputDl <<
" lastAvgThDl " << m_lastAvgTputDl <<
" DL metric: " << m_potentialTputDl / std::max (1E-9, m_avgTputDl));
}

void
NrMacSchedulerUeInfoRRW::CalculatePotentialTPutUl (const NrMacSchedulerNs3::FTResources &assignableInIteration,
const Ptr<const NrAmc> &amc)
{
NS_LOG_FUNCTION (this);

uint32_t rbsAssignable = assignableInIteration.m_rbg * GetNumRbPerRbg ();
m_potentialTputUl =  amc->CalculateTbSize (m_ulMcs, rbsAssignable);
m_potentialTputUl /= assignableInIteration.m_sym;

NS_LOG_INFO ("UE " << m_rnti << " potentialTputUl " << m_potentialTputUl <<
" lastAvgThUl " << m_lastAvgTputUl <<
" UL metric: " << m_potentialTputUl / std::max (1E-9, m_avgTputUl));
}

} // namespace ns3
```

A.5 NR-MAC-SCHEDULER-UE-INFO-DPS.H

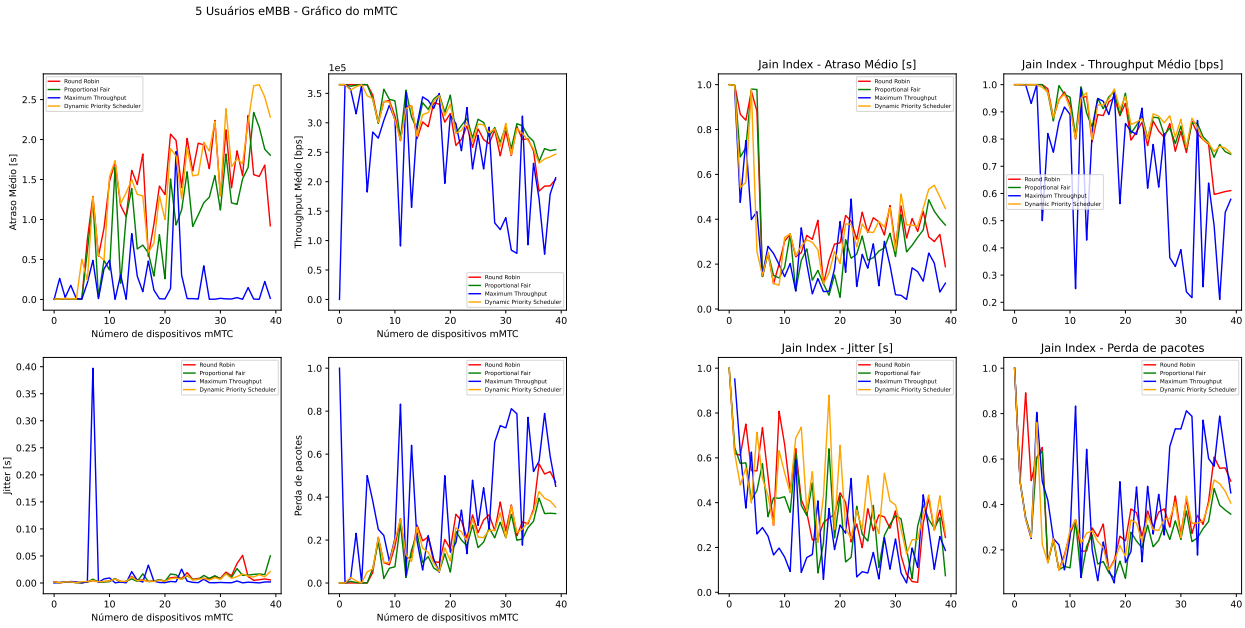
```
/* -*-  Mode: C++; c-file-style: "gnu"; indent-tabs-mode:nil; -*- */
/*
*   Copyright (c) 2019 Centre Tecnologic de Telecomunicacions de Catalunya (CTTC)
*
*   This program is free software; you can redistribute it and/or modify
*   it under the terms of the GNU General Public License version 2 as
*   published by the Free Software Foundation;
*
*   This program is distributed in the hope that it will be useful,
*   but WITHOUT ANY WARRANTY; without even the implied warranty of
*   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*   GNU General Public License for more details.
*
*   You should have received a copy of the GNU General Public License
*   along with this program; if not, write to the Free Software
*   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*
*/
#pragma once

#include "nr-mac-scheduler-ue-info-rr.h"

namespace ns3 {

/**
* \ingroup scheduler
* \brief UE representation for a proportional fair scheduler
*
* The representation stores the current throughput, the average throughput,
* and the last average throughput, as well as providing comparison functions
* to sort the UEs in case of a RRW scheduler.
*
* \see CompareUeWeightsDl
* \see CompareUeWeightsUl
*/
class NrMacSchedulerUeInfoRRW : public NrMacSchedulerUeInfo
{
public:
/**
* \brief NrMacSchedulerUeInfoRRW constructor
* \param rnti RNTI of the UE
* \param beamConfId BeamConfId of the UE
* \param fn A function that tells how many RB per RBG
*/
NrMacSchedulerUeInfoRRW (float alpha, uint16_t rnti, BeamConfId beamConfId, const GetRbPerRbgFn &fn)
: NrMacSchedulerUeInfo (rnti, beamConfId, fn),
m_alpha (alpha)
{
}

/**
* \brief Reset DL RRW scheduler info
*
* Set the last average throughput to the current average throughput,
* and zeroes the average throughput as well as the current throughput.
*
* It calls also NrMacSchedulerUeInfo::ResetDlSchedInfo.
*/
virtual void ResetDlSchedInfo () override
{
m_lastAvgTputDl = m_avgTputDl;
m_avgTputDl = 0.0;
m_currTputDl = 0.0;
m_potentialTputDl = 0.0;
NrMacSchedulerUeInfo::ResetDlSchedInfo ();
}

/**
* \brief Reset UL RRW scheduler info
*
* Set the last average throughput to the current average throughput,
* and zeroes the average throughput as well as the current throughput.
*
* It also calls NrMacSchedulerUeInfo::ResetUlSchedInfo.
*/
virtual void ResetUlSchedInfo () override
{
m_lastAvgTputUl = m_avgTputUl;
m_avgTputUl = 0.0;
m_currTputUl = 0.0;
m_potentialTputUl = 0.0;
NrMacSchedulerUeInfo::ResetUlSchedInfo ();
}

/**
* \brief Reset the DL avg Th to the last value
*/
virtual void ResetDlMetric () override
{
NrMacSchedulerUeInfo::ResetDlMetric ();
m_avgTputDl = m_lastAvgTputDl;
}

/**
* \brief Reset the UL avg Th to the last value
*/
```

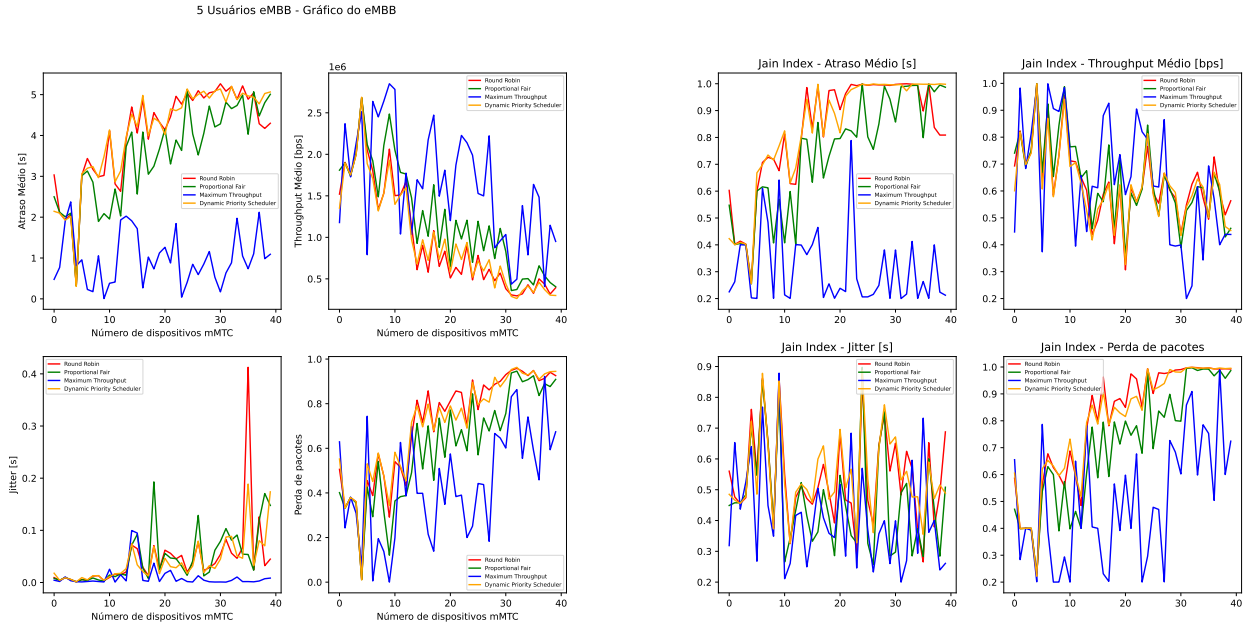
```
virtual void ResetUlMetric () override
{
NrMacSchedulerUeInfo::ResetUlMetric ();
m_avgTputUl = m_lastAvgTputUl;
}

/**
* \brief Update the RRW metric for downlink
* \param totAssigned the resources assigned
* \param timeWindow the time window
* \param amc a pointer to the AMC
*
* Updates m_currTputDl and m_avgTputDl by keeping in consideration
* the assigned resources (in form of TBS) and the time window.
* It gets the tbSise by calling NrMacSchedulerUeInfo::UpdateDlMetric.
*/
void UpdateDlRRWMetric (const NrMacSchedulerNs3::FTResources &totAssigned,
double timeWindow,
const Ptr<const NrAmc> &amc);

/**
* \brief Update the RRW metric for uplink
* \param totAssigned the resources assigned
* \param timeWindow the time window
* \param amc a pointer to the AMC
*
* Updates m_currTputUl and m_avgTputUl by keeping in consideration
* the assigned resources (in form of TBS) and the time window.
* It gets the tbSise by calling NrMacSchedulerUeInfo::UpdateUlMetric.
*/
void UpdateUlRRWMetric (const NrMacSchedulerNs3::FTResources &totAssigned,
double timeWindow,
const Ptr<const NrAmc> &amc);

/**
* \brief Calculate the Potential throughput for downlink
* \param assignableInIteration resources assignable
* \param amc a pointer to the AMC
*/
void CalculatePotentialTPutDl (const NrMacSchedulerNs3::FTResources &assignableInIteration,
const Ptr<const NrAmc> &amc);

/**
* \brief Calculate the Potential throughput for uplink
* \param assignableInIteration resources assignable
* \param amc a pointer to the AMC
*/
void CalculatePotentialTPutUl (const NrMacSchedulerNs3::FTResources &assignableInIteration,
const Ptr<const NrAmc> &amc);


/**
* \brief comparison function object (i.e. an object that satisfies the
* requirements of Compare) which returns true if the first argument is less
* than (i.e. is ordered before) the second.
* \param lue Left UE
* \param rue Right UE
* \return true if the RRW metric of the left UE is higher than the right UE
*
* The RRW metric is calculated as following:
*
* \f$ pfMetric_{i} = std::pow(potentialTPut_{i}, alpha) / std::max (1E-9, m_avgTput_{i}) \f$
*
* Alpha is a fairness metric. Please note that the throughput is calculated
* in bit/symbol.
*/
static bool CompareUeWeightsDl (const NrMacSchedulerNs3::UePtrAndBufferReq &lue,
const NrMacSchedulerNs3::UePtrAndBufferReq &rue)
{
auto luePtr = dynamic_cast<NrMacSchedulerUeInfoRRW*> (lue.first.get ());
auto ruePtr = dynamic_cast<NrMacSchedulerUeInfoRRW*> (rue.first.get ());

double lPfMetric = std::pow (luePtr->m_potentialTputDl, luePtr->m_alpha) / std::max (1E-9, luePtr->m_avgTputDl);
double rPfMetric = std::pow (ruePtr->m_potentialTputDl, ruePtr->m_alpha) / std::max (1E-9, ruePtr->m_avgTputDl);



if (luePtr->m_dynamicPriority > ruePtr->m_dynamicPriority)
return (true);

if (luePtr->m_dynamicPriority == ruePtr->m_dynamicPriority)
return (lPfMetric > rPfMetric);

return (false);
}

/**
* \brief comparison function object (i.e. an object that satisfies the
* requirements of Compare) which returns true if the first argument is less
* than (i.e. is ordered before) the second.
* \param lue Left UE
* \param rue Right UE
* \return true if the RRW metric of the left UE is higher than the right UE
*
* The RRW metric is calculated as following:
*
* \f$ pfMetric_{i} = std::pow(potentialTPut_{i}, alpha) / std::max (1E-9, m_avgTput_{i}) \f$
*
* Alpha is a fairness metric. Please note that the throughput is calculated
```

```
* in bit/symbol.
*/
static bool CompareUeWeightsUl (const NrMacSchedulerNs3::UePtrAndBufferReq &lue,
const NrMacSchedulerNs3::UePtrAndBufferReq &rue)
{
auto luePtr = dynamic_cast<NrMacSchedulerUeInfoRRW*> (lue.first.get ());
auto ruePtr = dynamic_cast<NrMacSchedulerUeInfoRRW*> (rue.first.get ());

double lPfMetric = std::pow (luePtr->m_potentialTputUl, luePtr->m_alpha) / std::max (1E-9, luePtr->m_avgTputUl);
double rPfMetric = std::pow (ruePtr->m_potentialTputUl, ruePtr->m_alpha) / std::max (1E-9, ruePtr->m_avgTputUl);

return (lPfMetric > rPfMetric);



}
int m_dynamicPriority {20};     // Dynamic priority is the current priority
int m_basePriority {20};       // Base priority is the maximum priority for an UE
int m_lastdynamicPriority {20};     // last Dynamic priority is the current priority
int m_lastbasePriority {20};       // last Base priority is the maximum priority for an UE
int m_lastdlRBG{0};                // I use this for discover how many RGBs where allocated for UE ... should be a better way, sorry.
int m_inc{0};                      // counter used to increment dynamic priority
std::string part = "";

// These are for RRW: UEs of same priority uses RRW... What? I ask myself the same ... !!!!

double m_currTputDl {0.0};    //!< Current slot throughput in downlink
double m_avgTputDl  {0.0};    //!< Average throughput in downlink during all the slots
double m_lastAvgTputDl {0.0}; //!< Last average throughput in downlink
double m_potentialTputDl {0.0}; //!< Potential throughput in downlink in one assignable resource (can be a symbol or a RBG)
float  m_alpha {0.0};         //!< RRW fairness metric

double m_currTputUl {0.0};    //!< Current slot throughput in uplink
double m_avgTputUl  {0.0};    //!< Average throughput in uplink during all the slots
double m_lastAvgTputUl {0.0}; //!< Last average throughput in uplink
double m_potentialTputUl {0.0}; //!< Potential throughput in uplink in one assignable resource (can be a symbol or a RBG)
};


} // namespace ns3
```

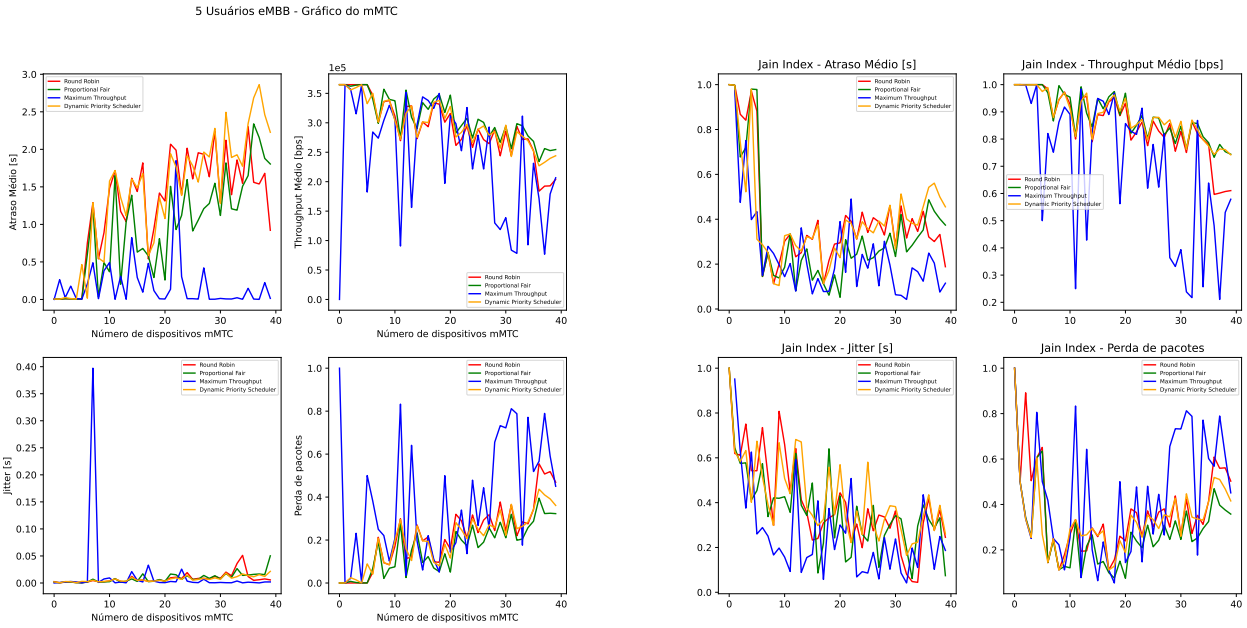
A.6 SIMULAÇÕES

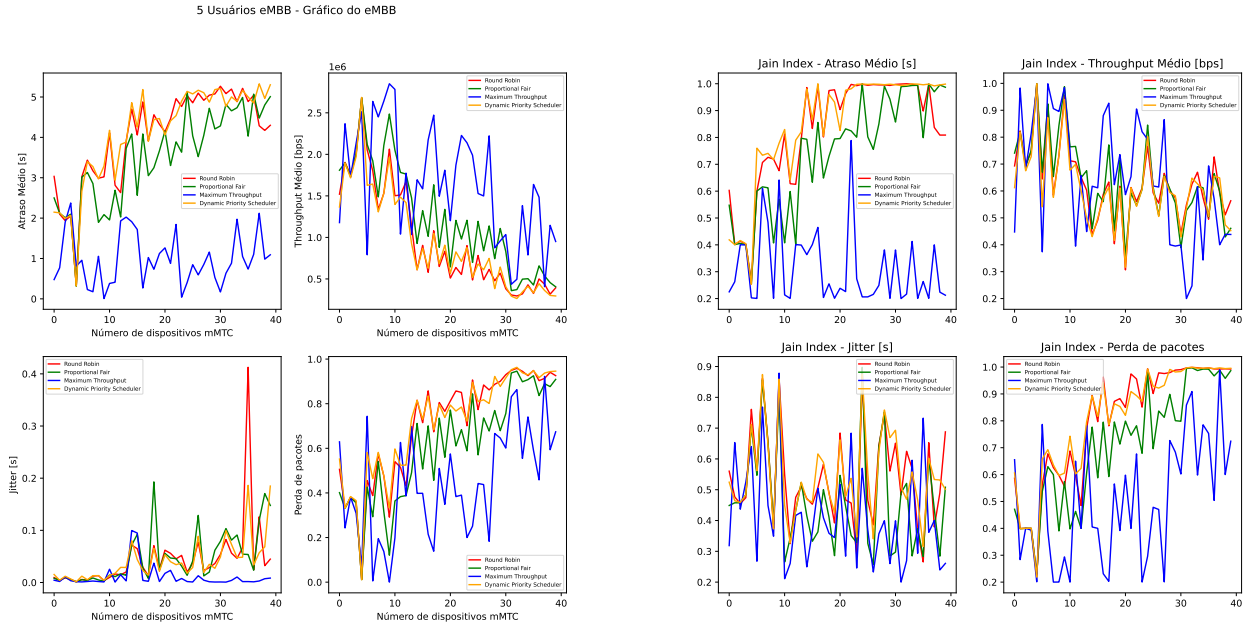
A.6.1 Weight 10



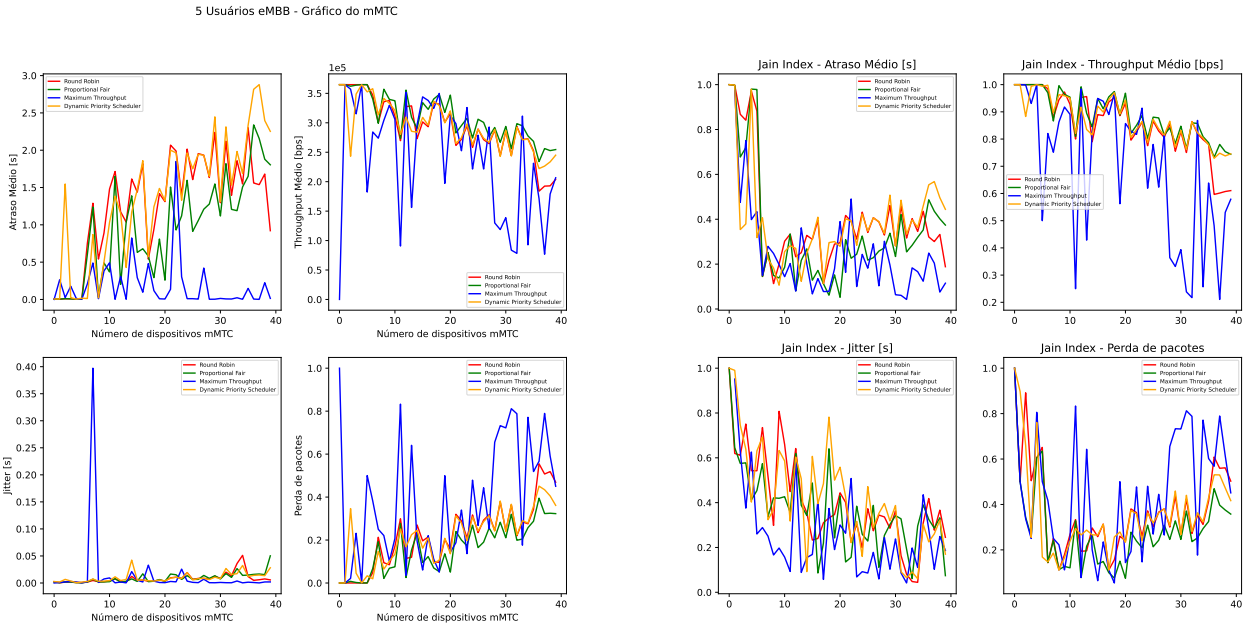


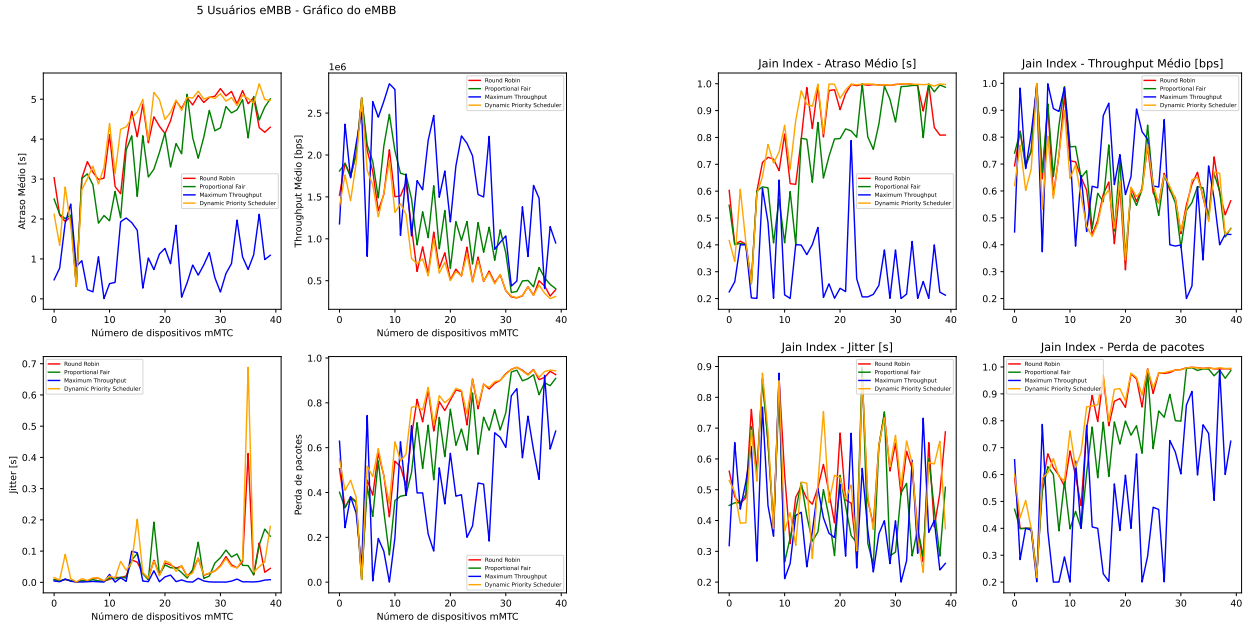
A.6.2 Weight 12



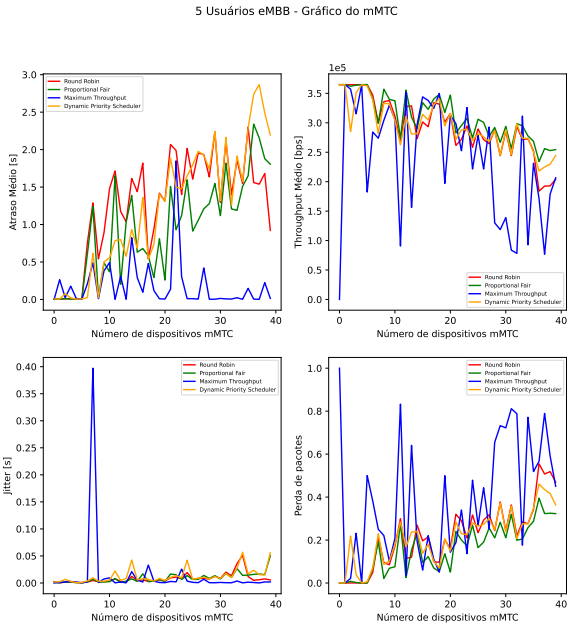


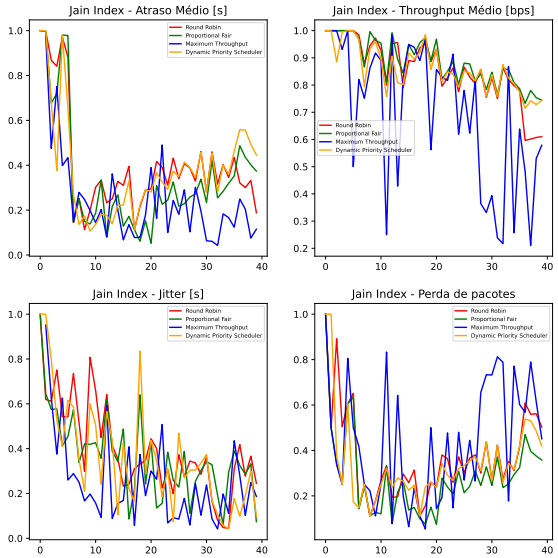
A.6.3 Weight 20

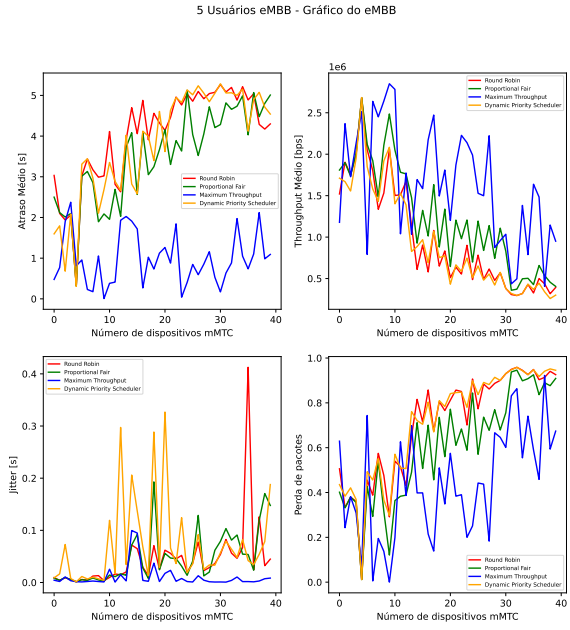


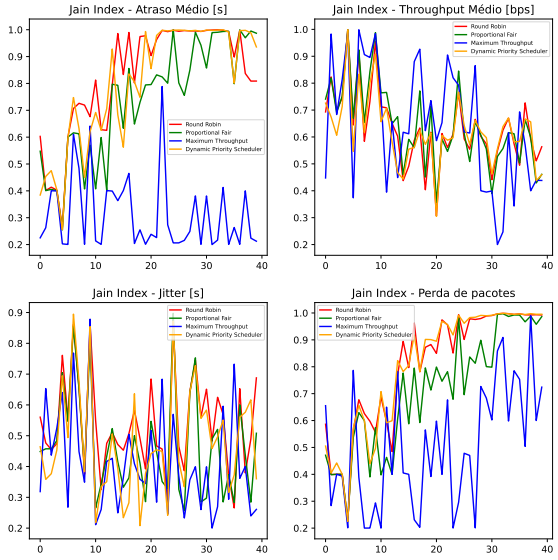


A.6.4 Weight 24

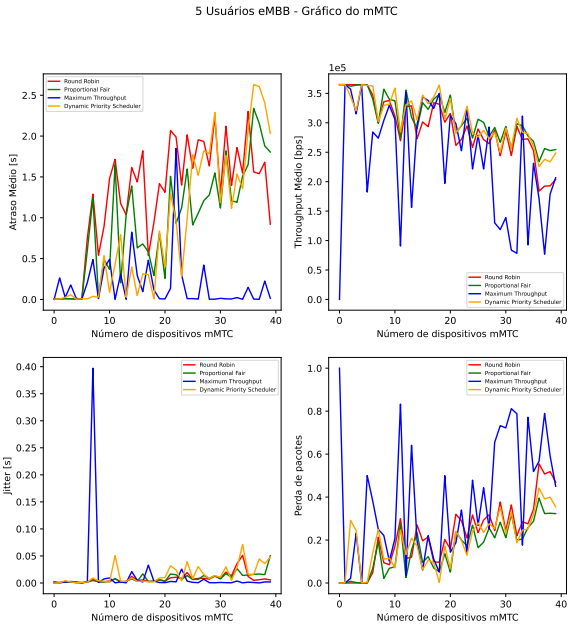


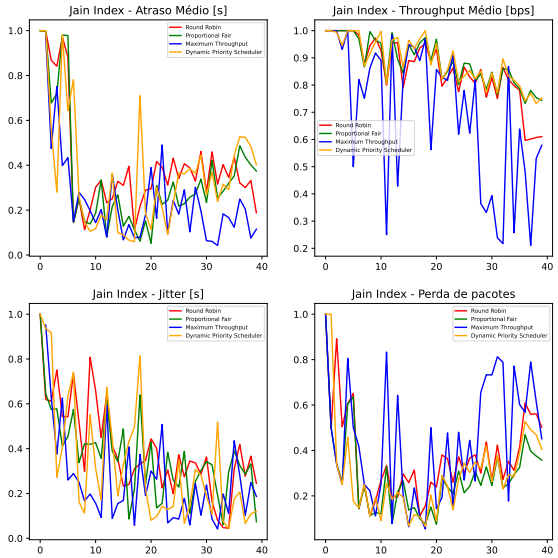


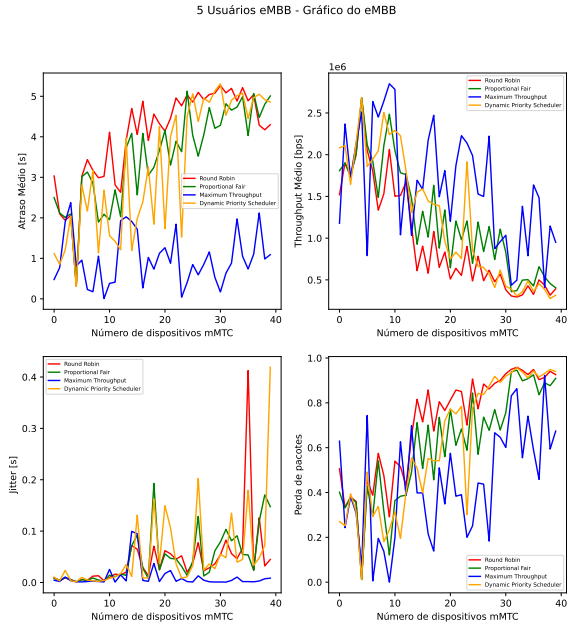


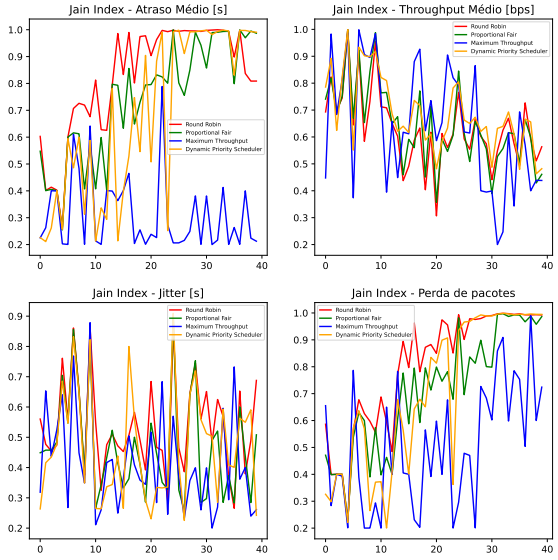


A.6.5 Weight 26

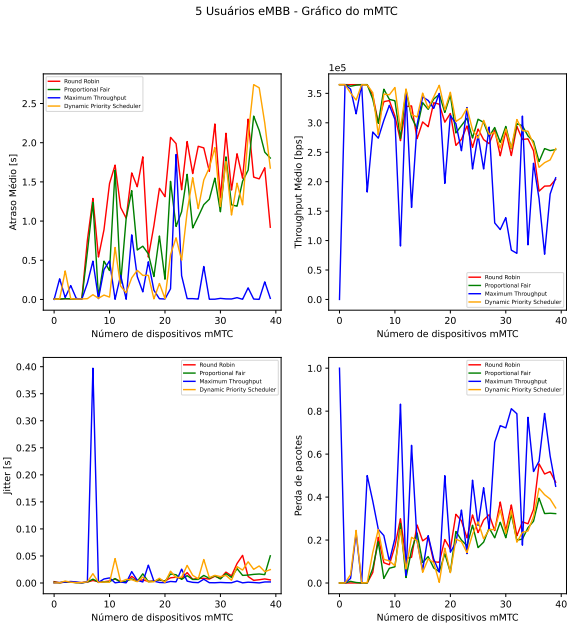


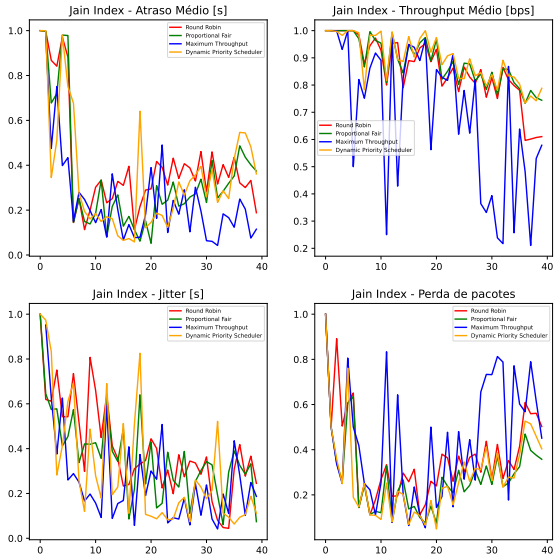


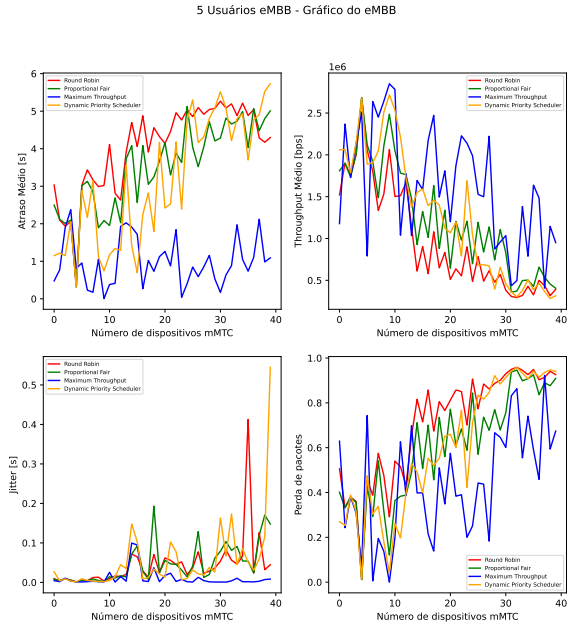


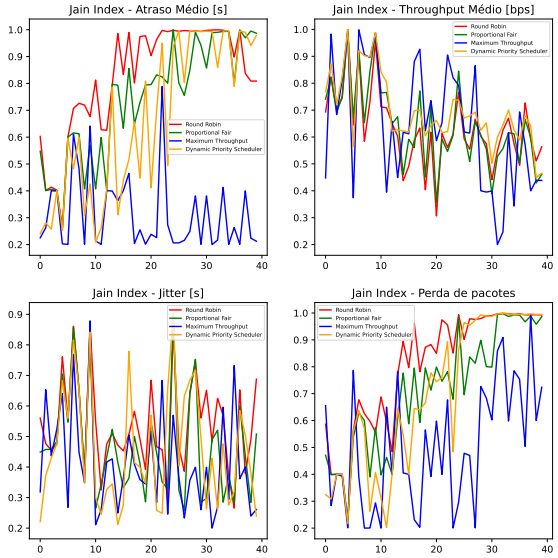


A.6.6 Weight 28

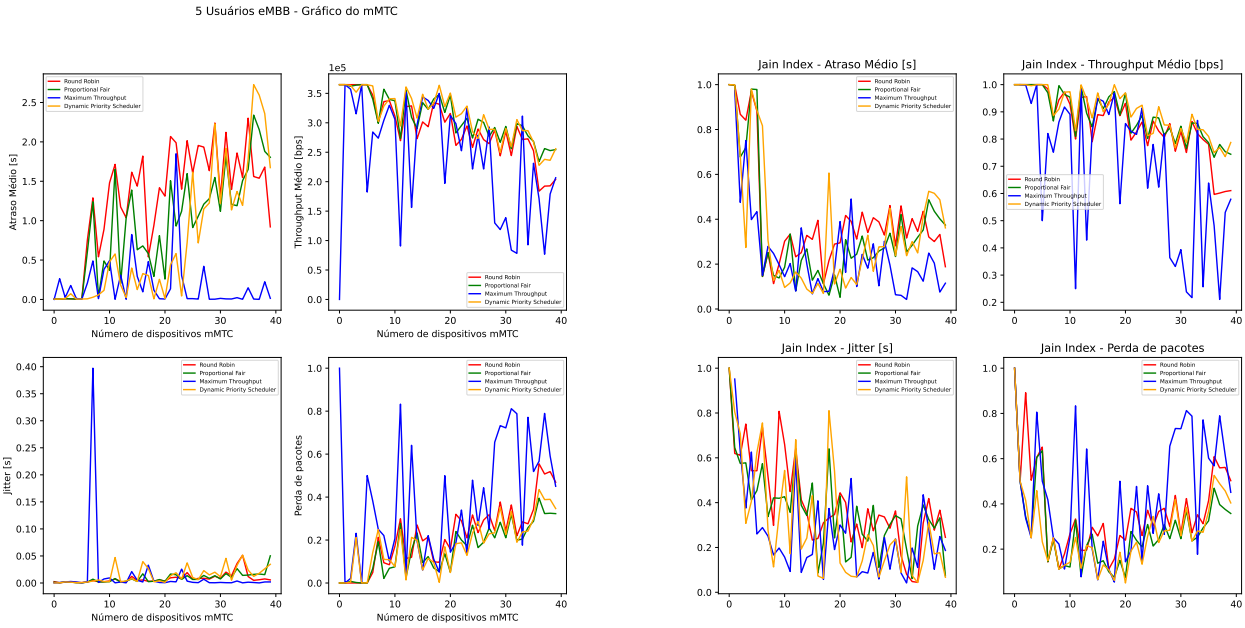


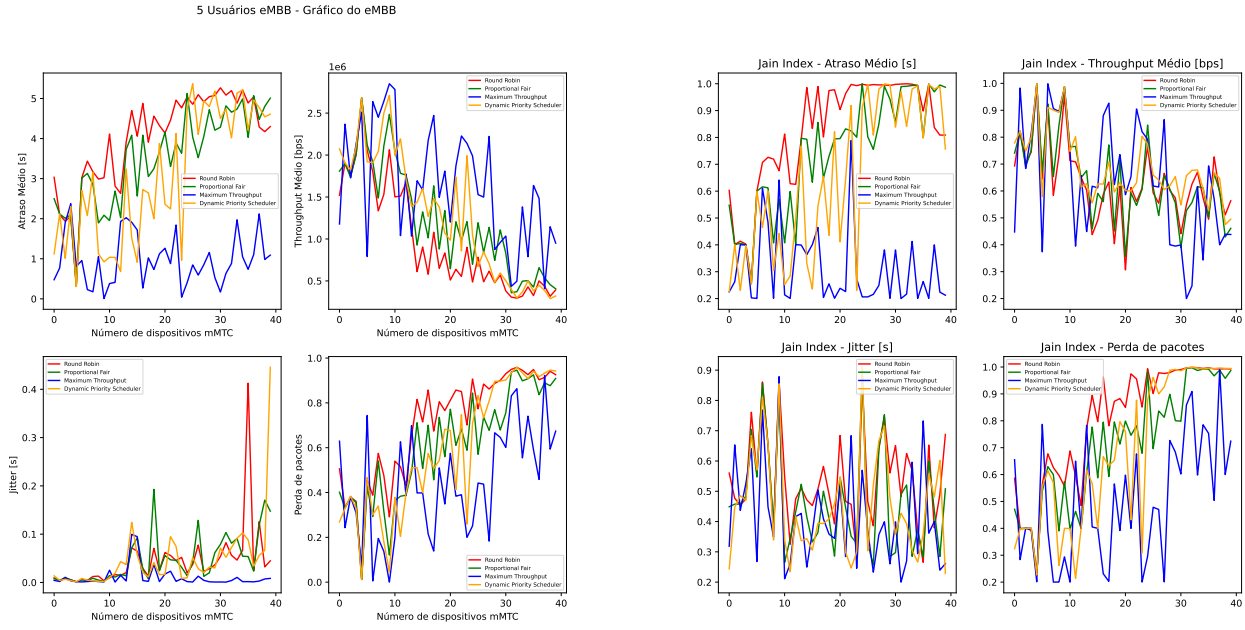




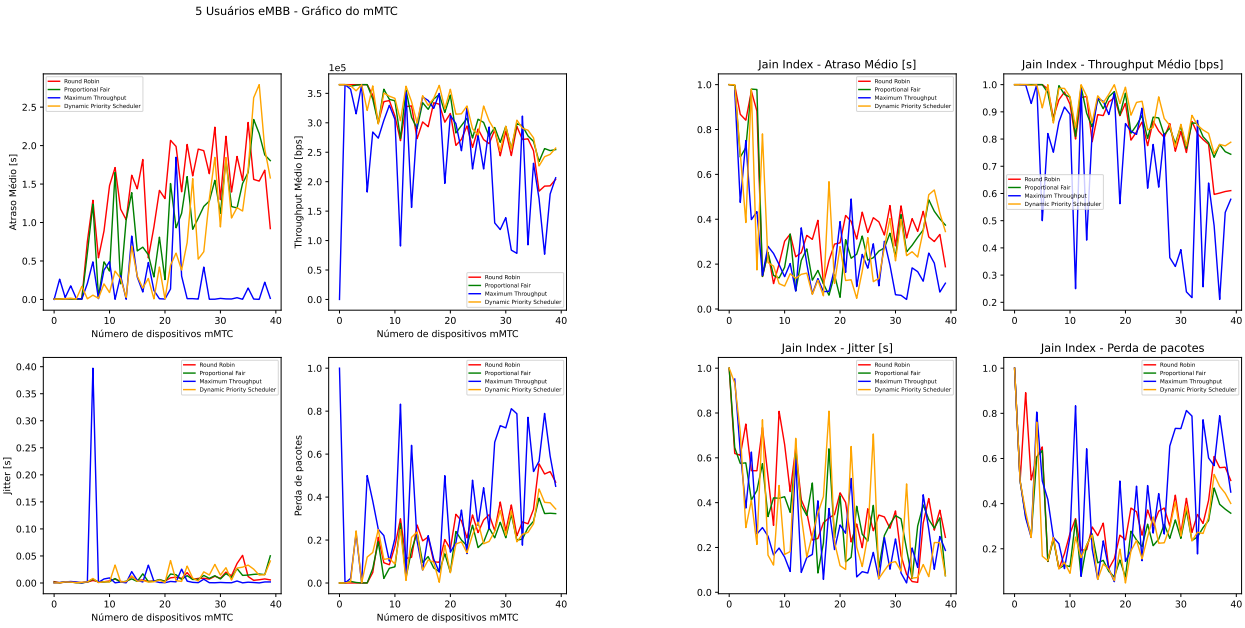


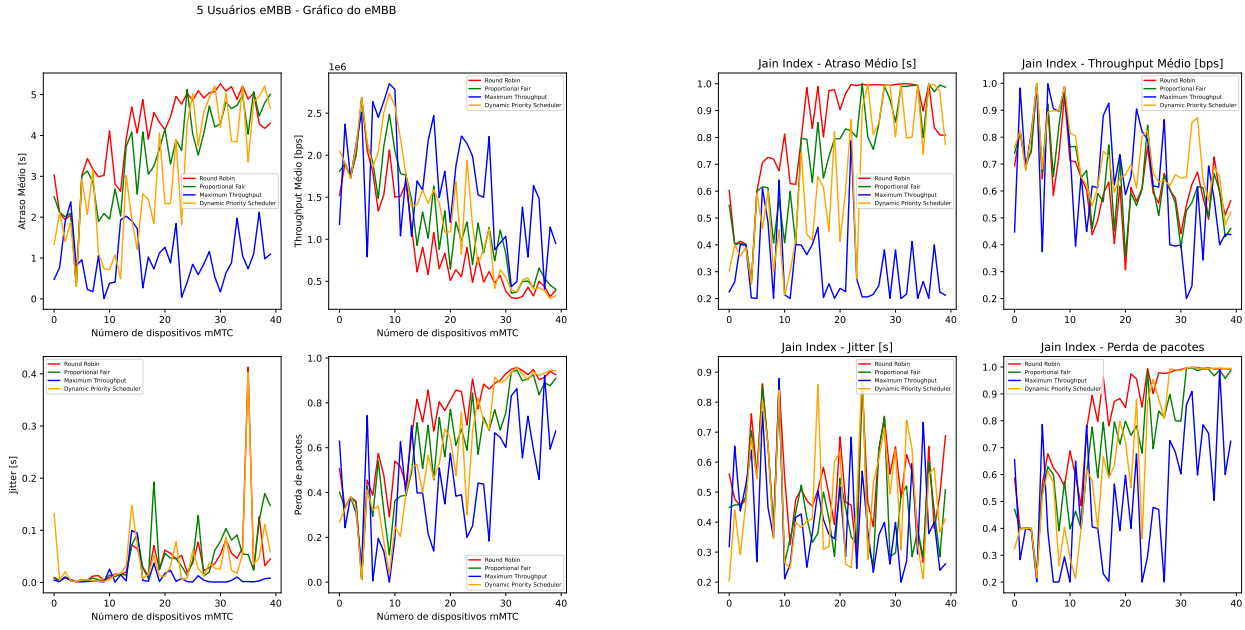
A.6.7 Weight 30



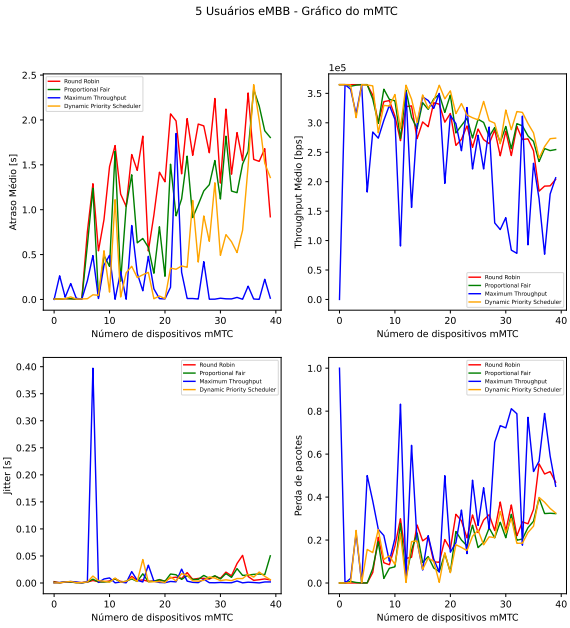


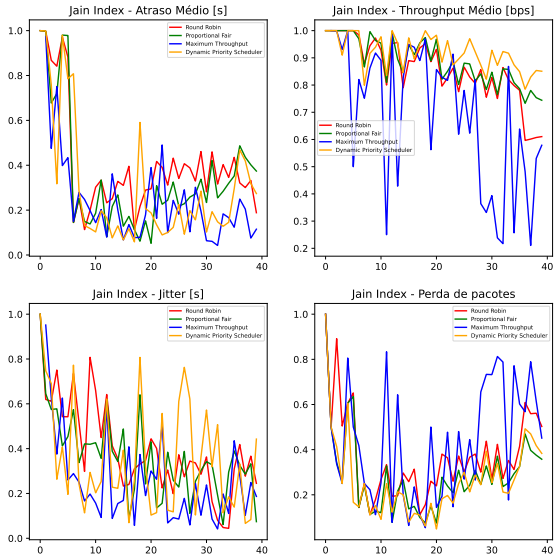
A.6.8 Weight 32

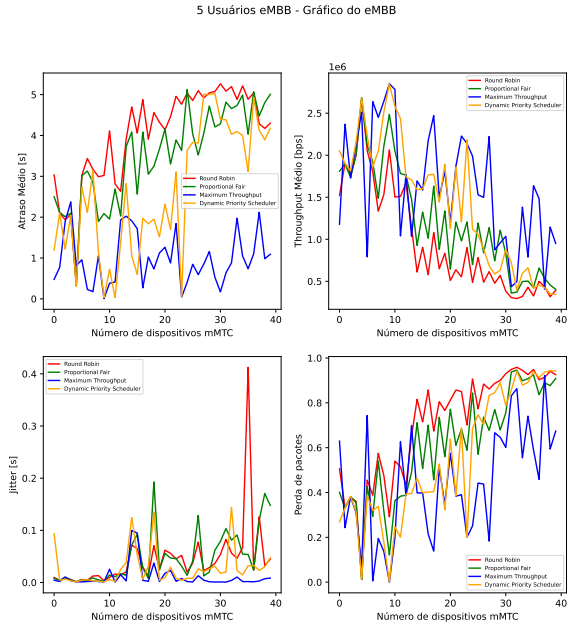


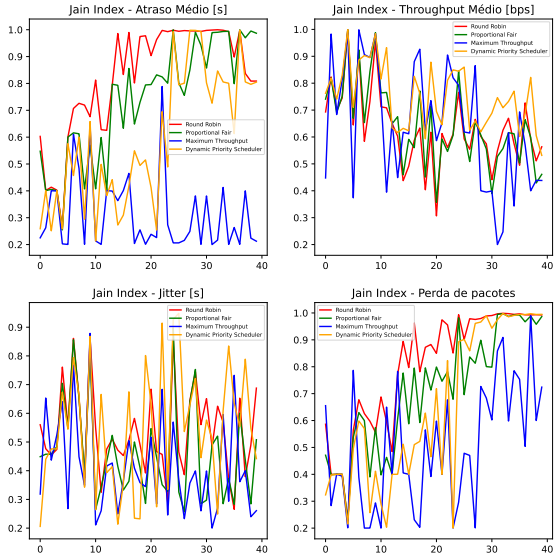


A.6.9 Weight 34

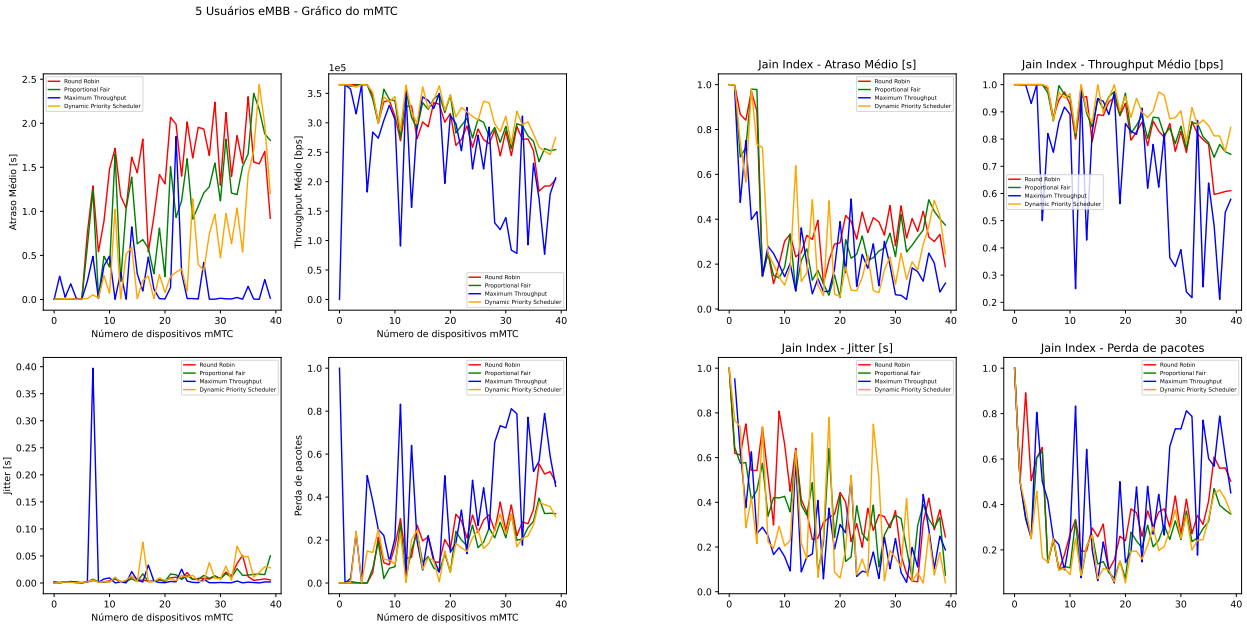


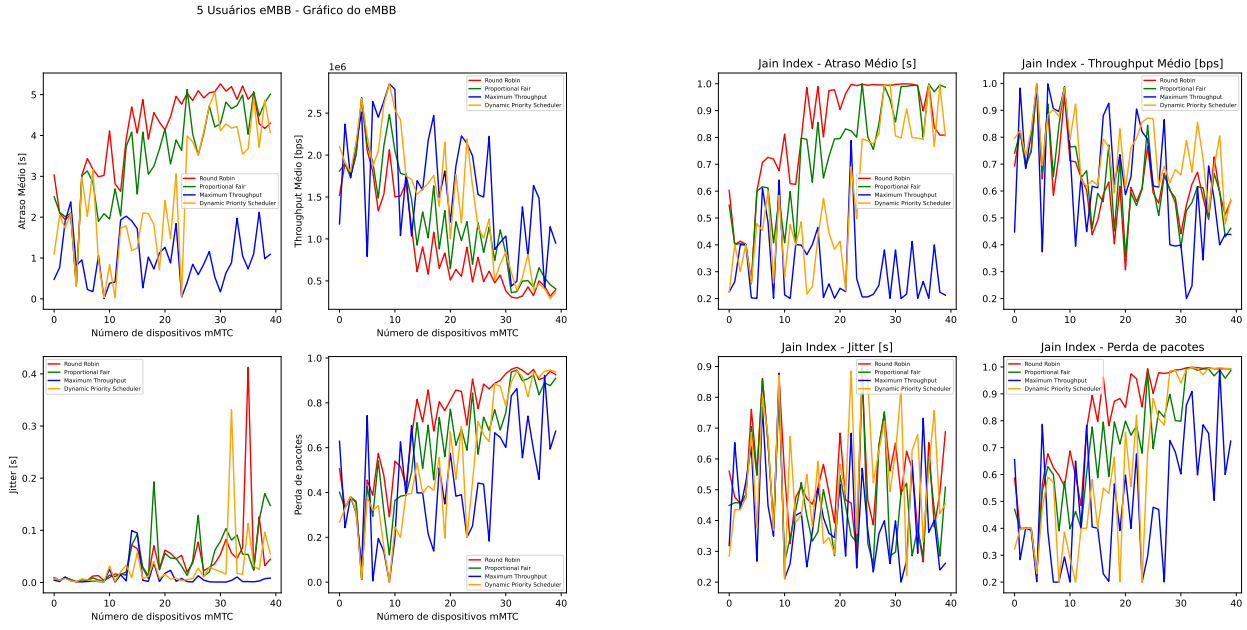




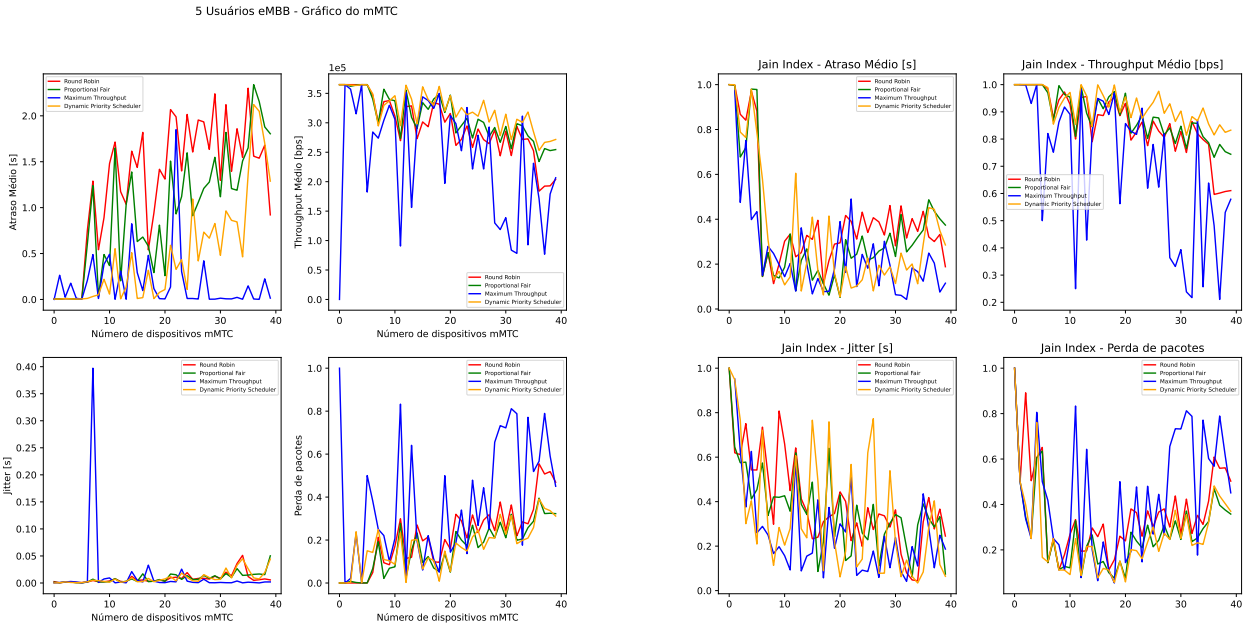


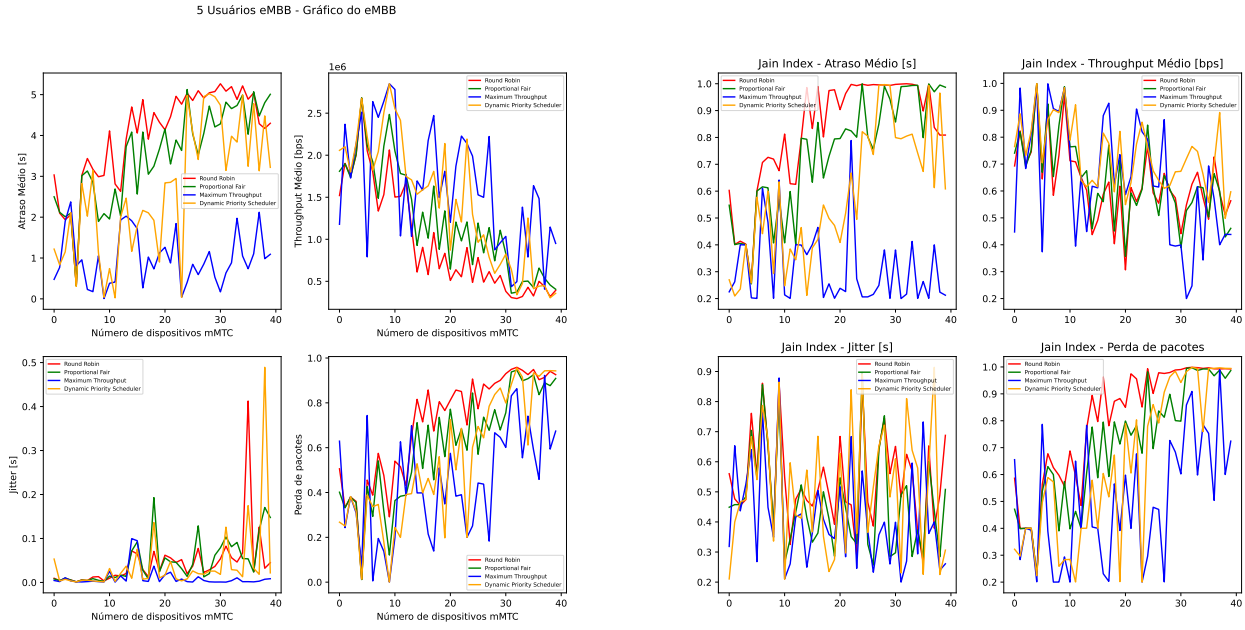
A.6.10 Weight 36



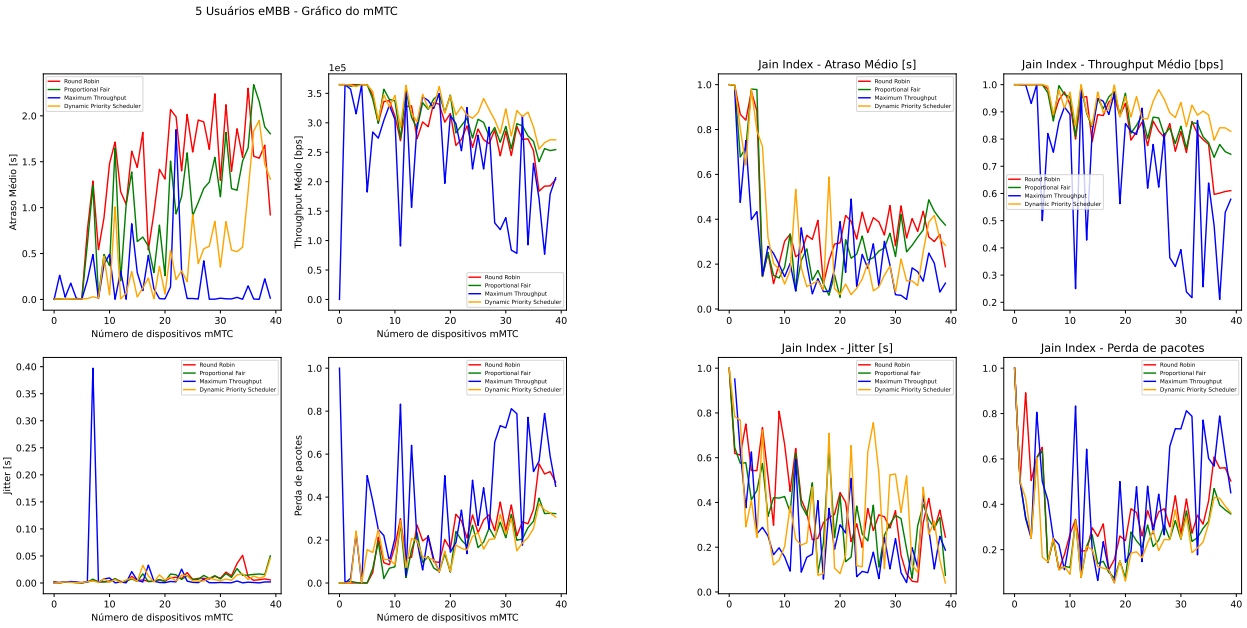


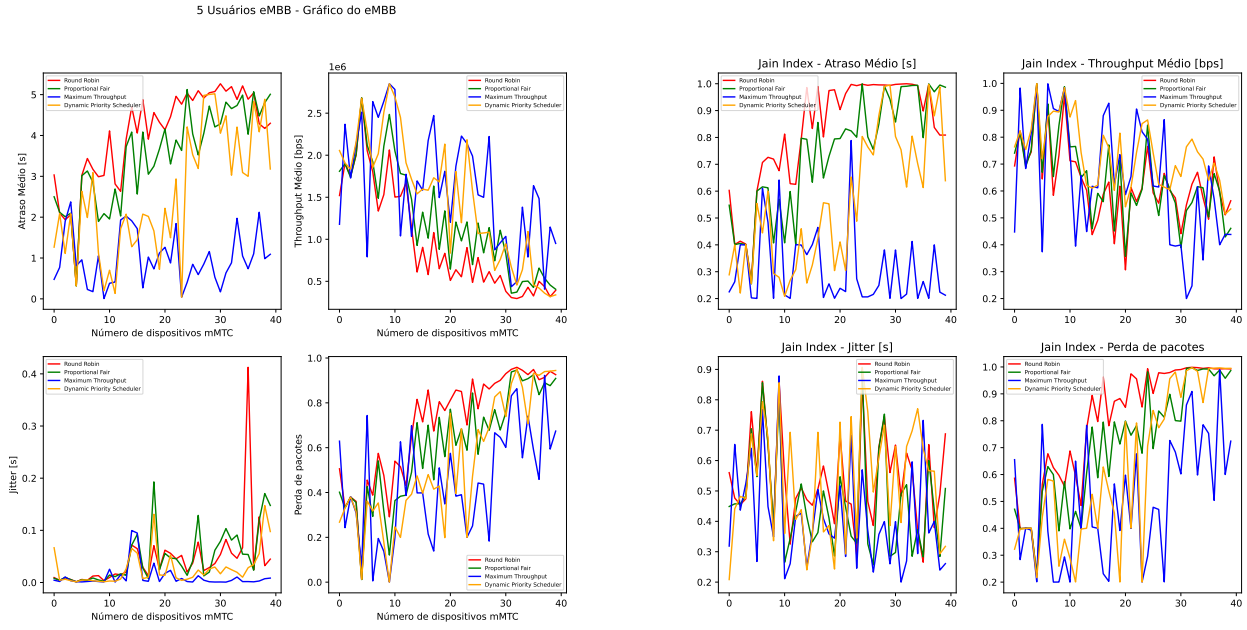
A.6.11 Weight 38



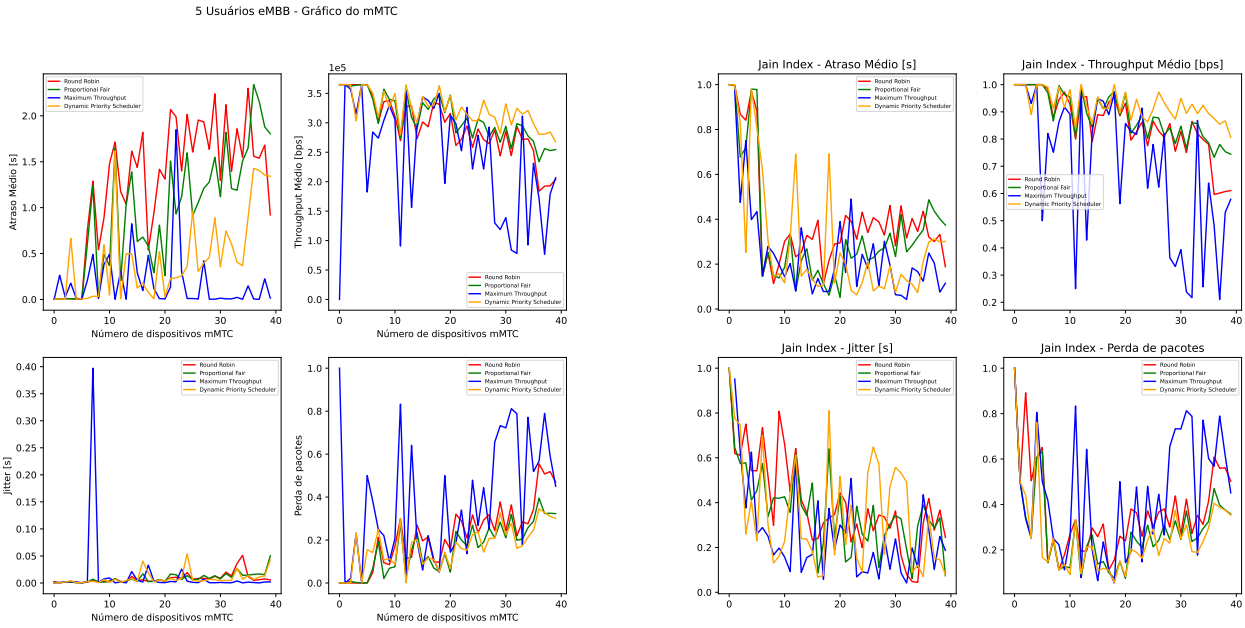


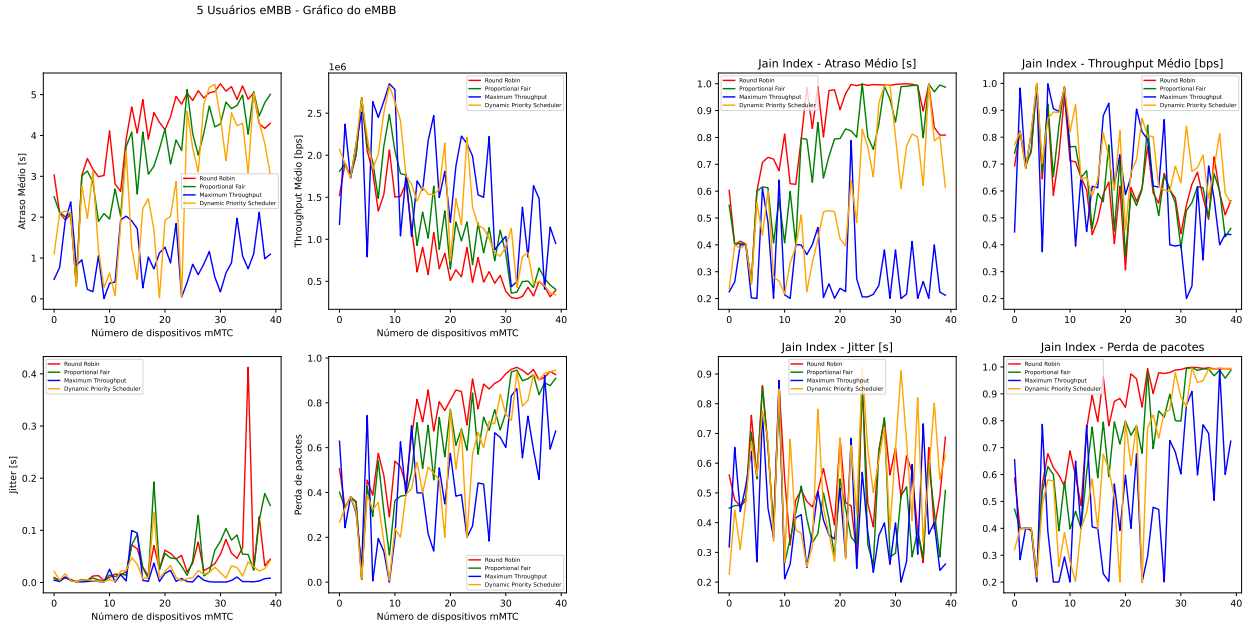
A.6.12 Weight 40



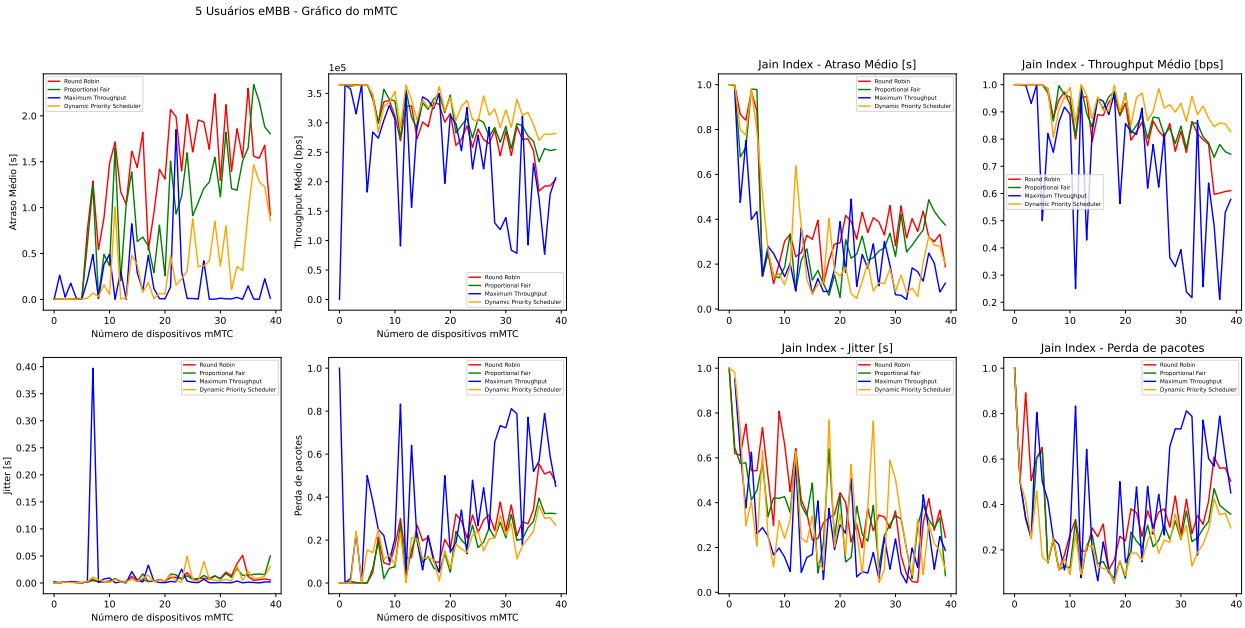


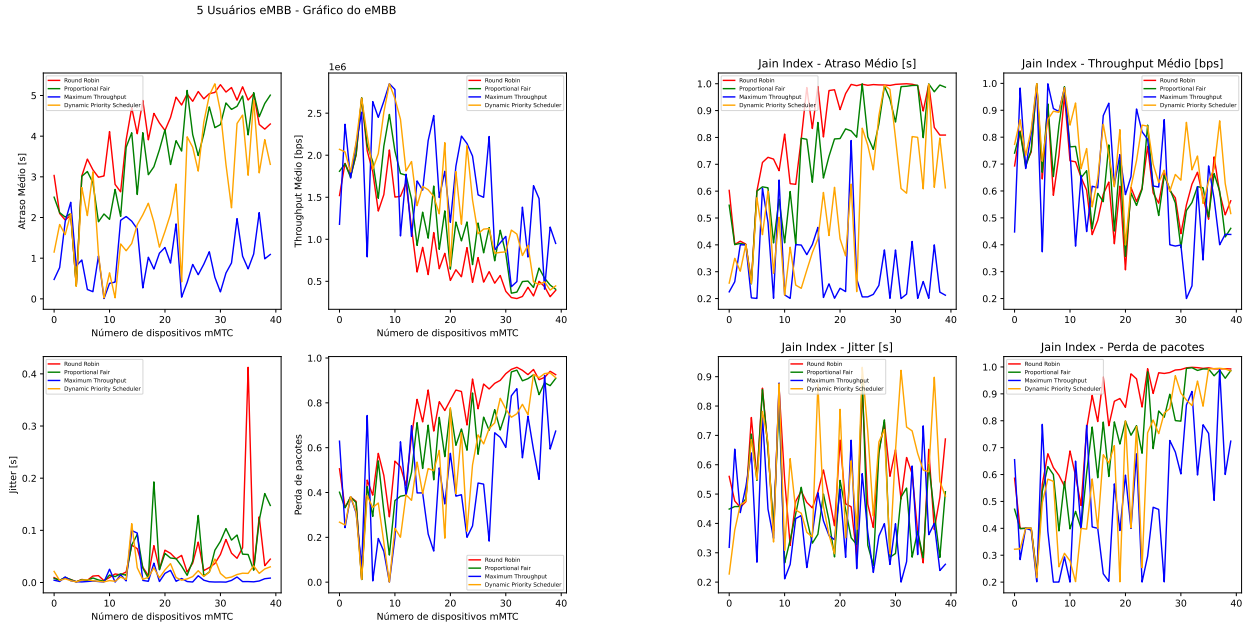
A.6.13 Weight 42



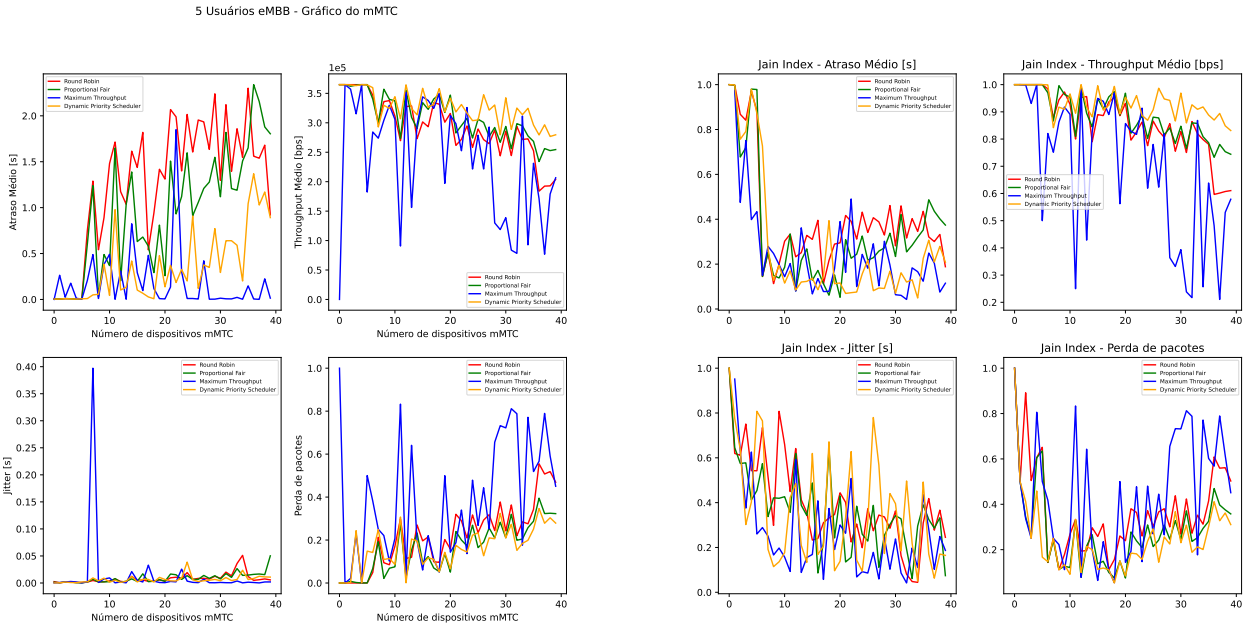


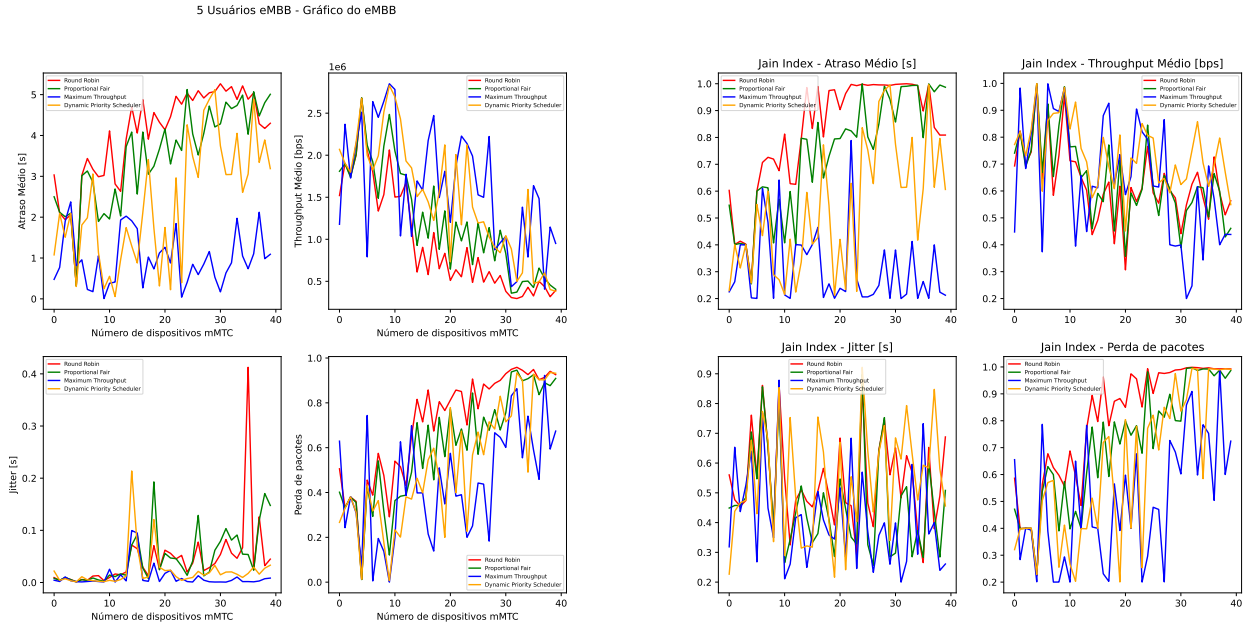
A.6.14 Weight 44



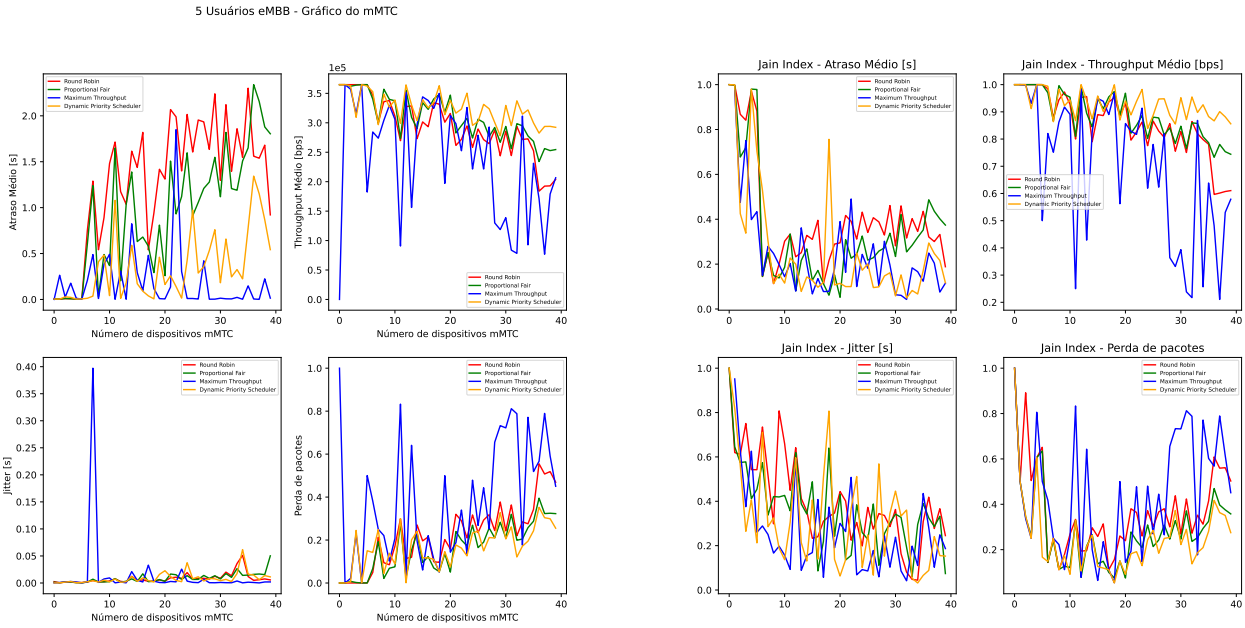


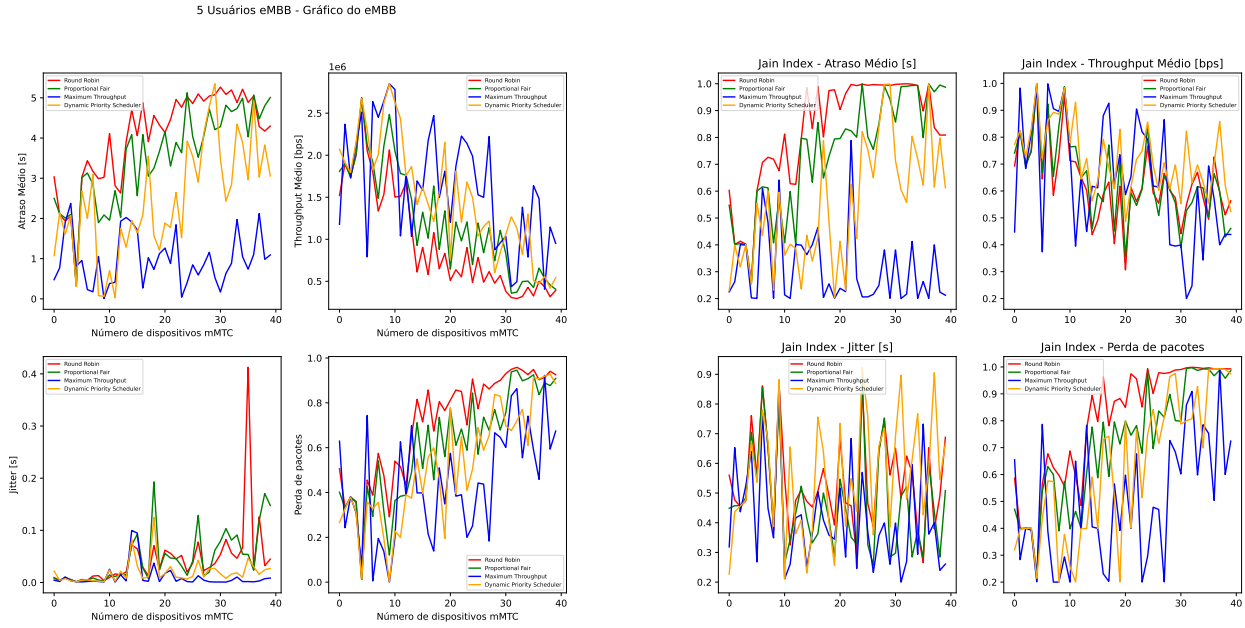
A.6.15 Weight 48



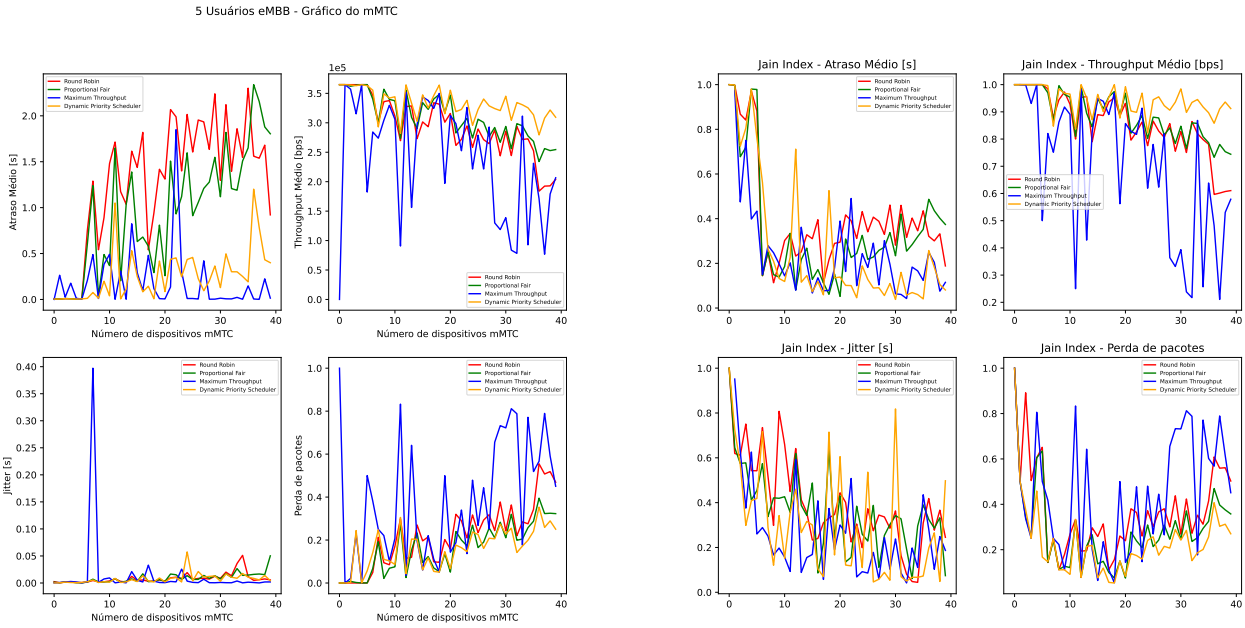


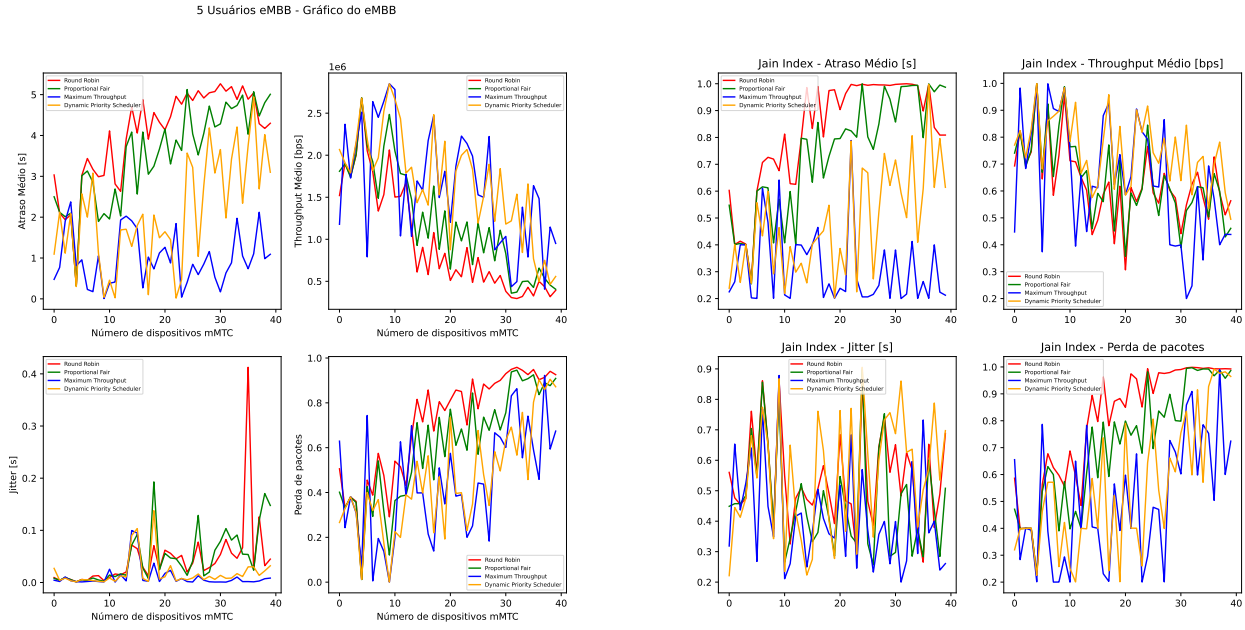
A.6.16 Weight 50



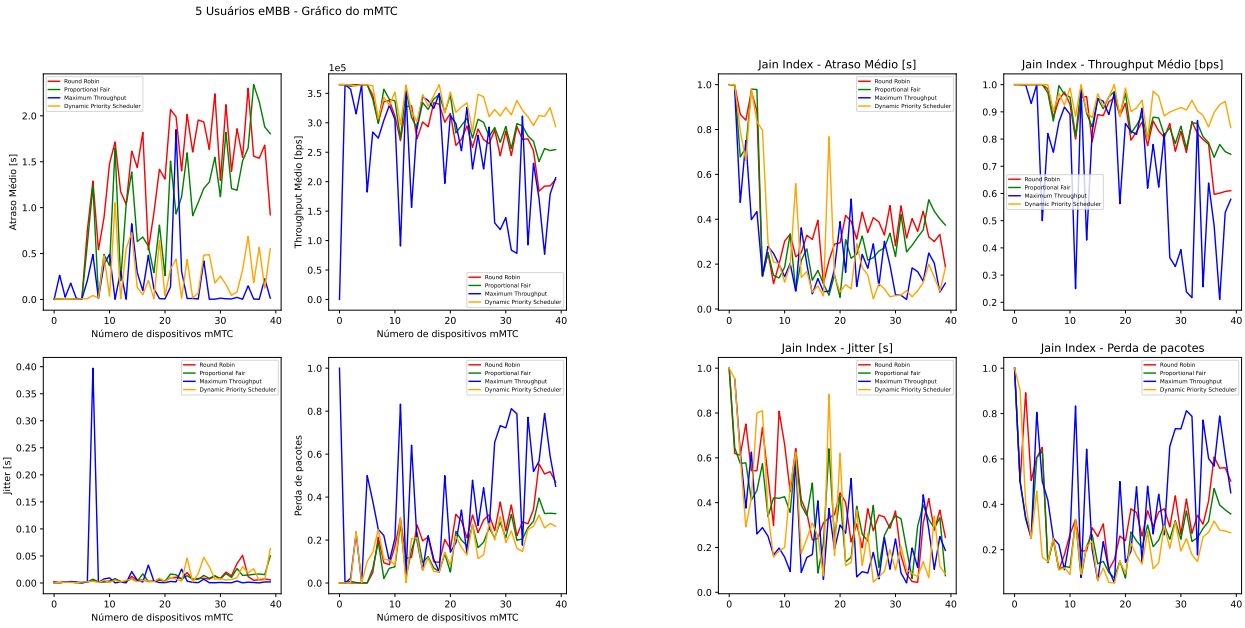


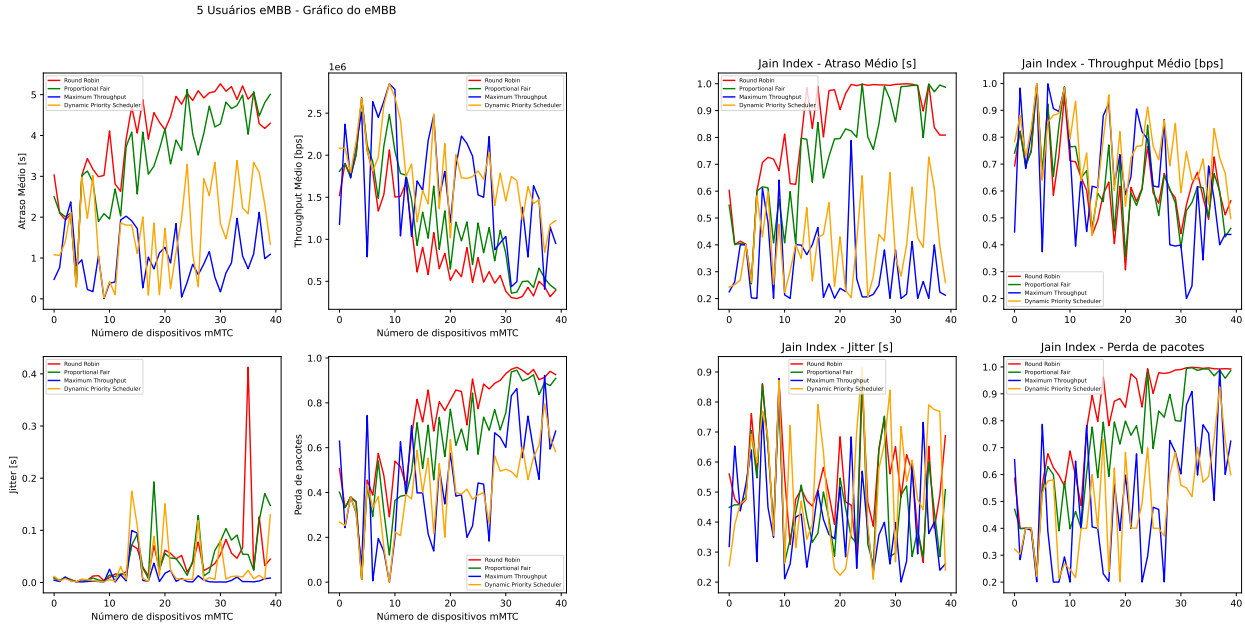
A.6.17 Weight 51



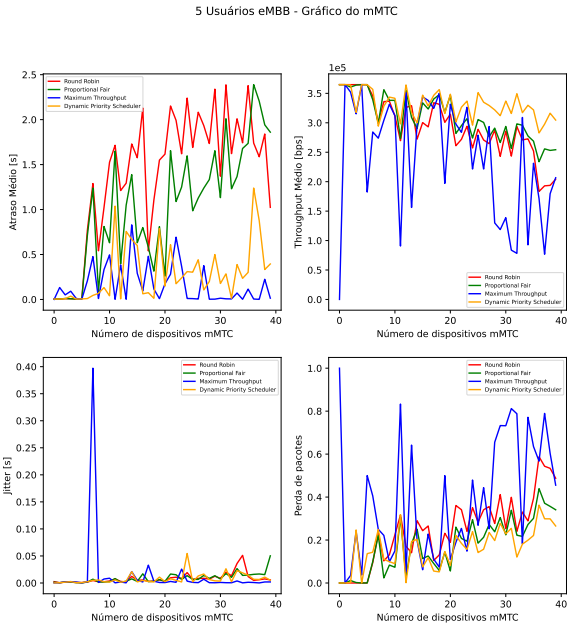


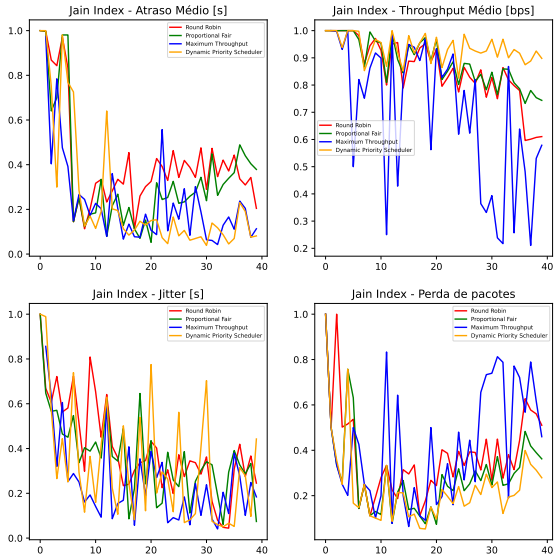
A.6.18 Weight 52

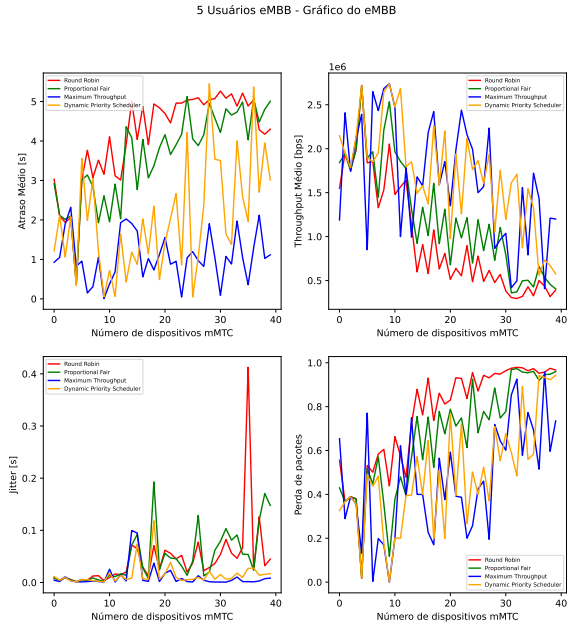


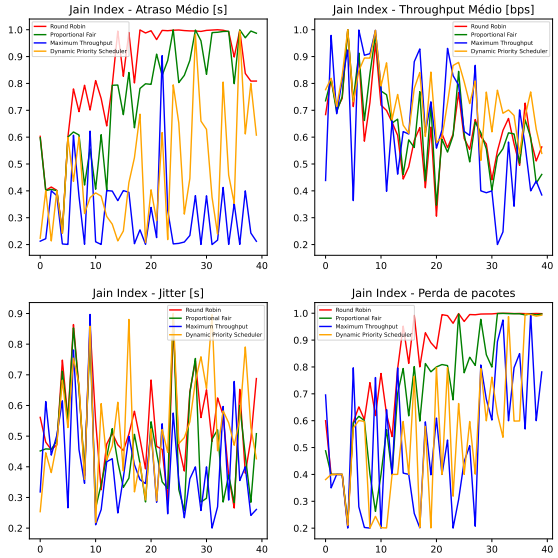


A.6.19 Weight 53

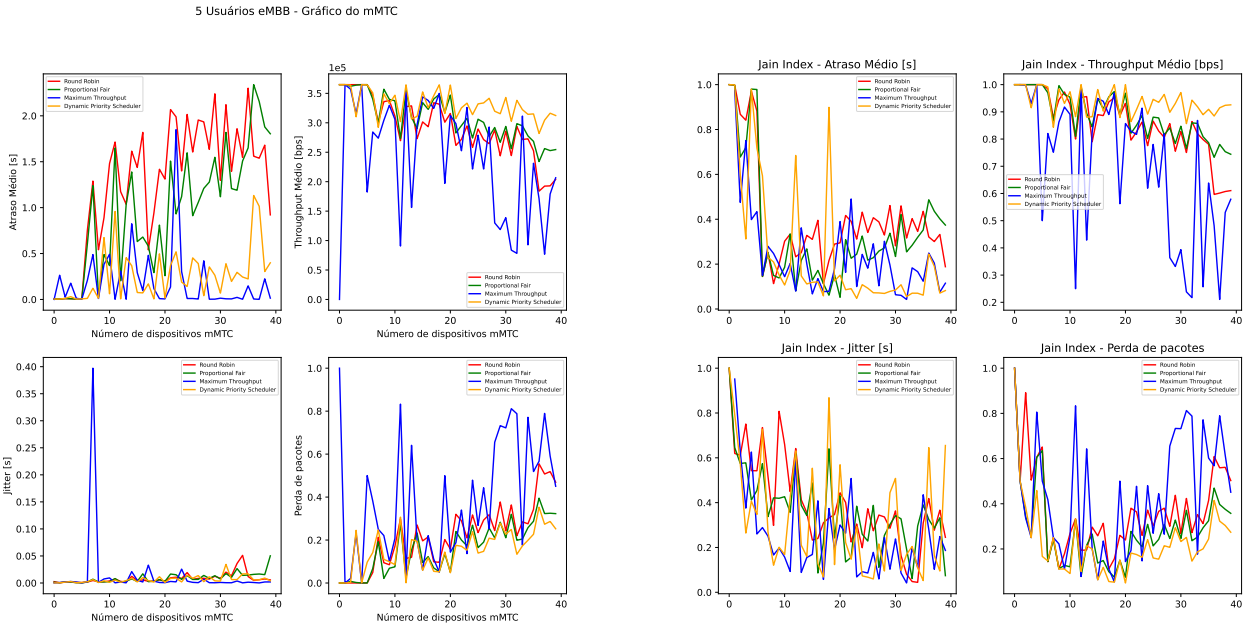


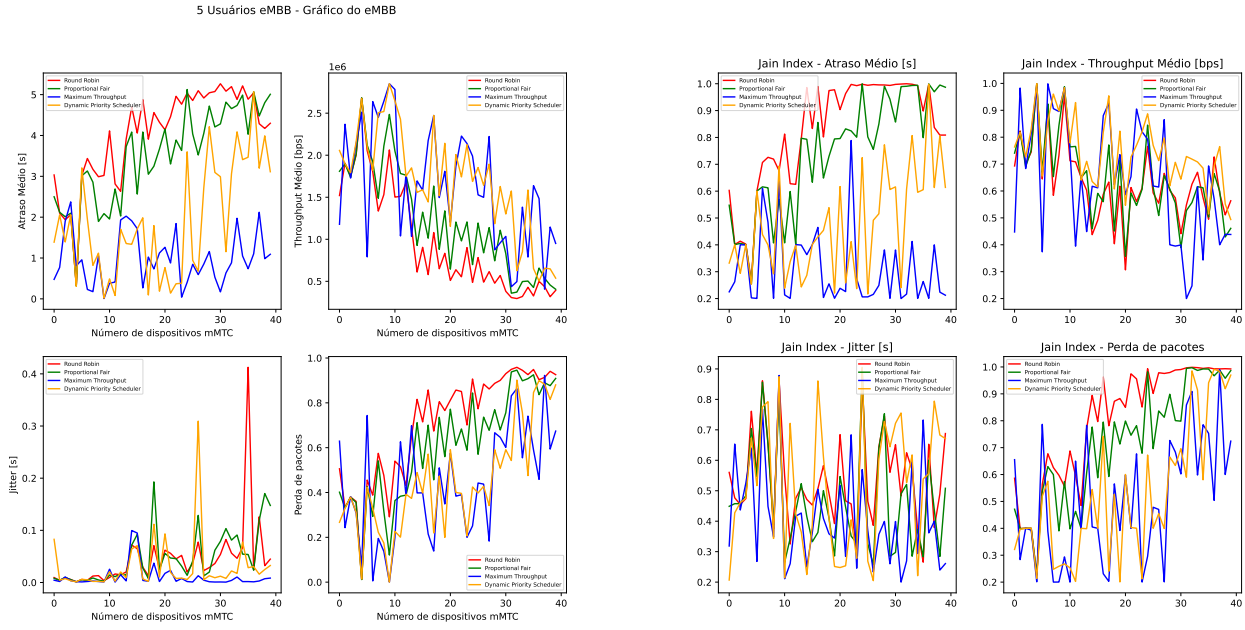




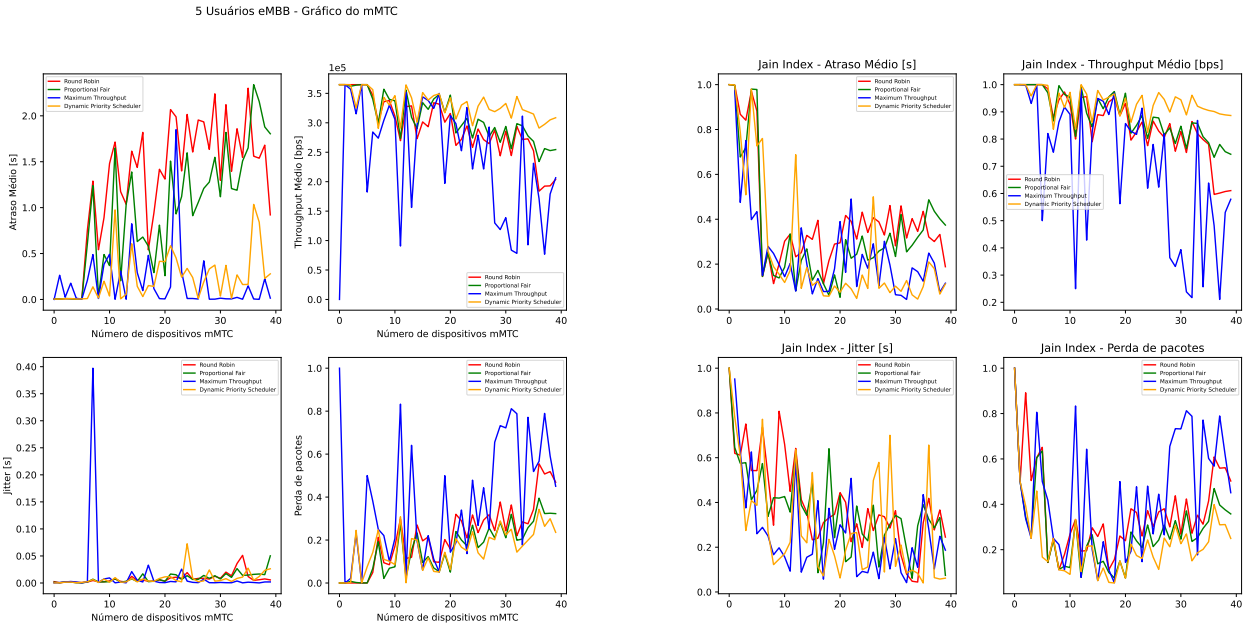


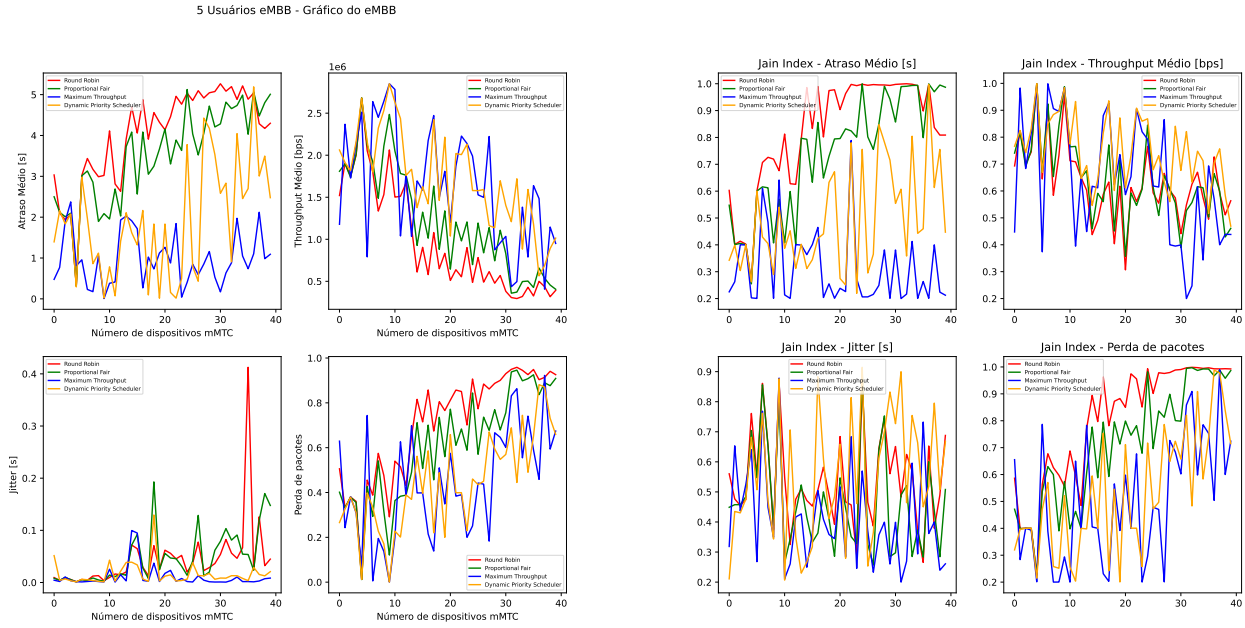
A.6.20 Weight 54





A.6.21 Weight 60





A.6.22 Weight 72