Relatório de Avaliação de Rede Sem Fio IEEE 802.11 em modo Infraestruturado no simulador *Network Simulator 3*

Joahannes B. D. da Costa[1], **Wellington V. Lobato Junior**[1]

[1]Laboratório de Redes de Computadores (LRC) – Instituto de Computação (IC)
Universidade Estadual de Campinas (UNICAMP)
Campinas – São Paulo – Brasil

{joahannes,wellington}@lrc.ic.unicamp.br

Resumo. *Este documento apresenta uma avaliação da comunicação de rede sem fio 802.11 em modo infraestruturada e é parte dos critérios de avaliação da disciplina MO655 - Gerência de Redes de Computadores, do Instituto de Computação (IC), da UNICAMP, sob supervisão do Prof. Dr. Edmundo Madeira. Desta forma, os tráfegos de comunicação gerados foram o de Constant Bit Rate (CBR), utilizando o protocolo UDP, e tráfego em Rajadas, que por sua vez utiliza o protocolo TCP. O número de clientes conectados na rede varia de 1 a 6. Ainda, dois cenários de mobilidade foram utilizados, o primeiro sendo os nós parados em volta do AP e o segundo com os clientes se deslocando até a borda da rede. Os resultados mostraram que a presença de mobilidade nos nós influencia diretamente nas taxas de atraso, vazão e perda de pacotes.*

1. Introdução

O surgimento de várias tecnologias nos últimos anos é decorrência do avanço nas tecnologias de comunicação, que procuram sempre atendar as necessidades e demandas de seus usuários com a melhor qualidade possível. Especialmente a comunicação sem fio ganhou espaço considerável no universo de transmissão de dados, deixando de existir apenas nas comunicações de longa distância, como em satélites, para fazer parte de ambientes locais. Essa utilização mais comum foi fortalecida pelo investimento da academia e indústria no sentido de aplicar a comunicação sem fio em redes de computadores [Garcia 2012].

Os primeiros produtos para redes sem fio começaram a sere introduzidos no início da década de 90. Dentro deste contexto notou-se que faltava uma padronização de uso e compatibilidade, observando esta lacuna o IEEE (*Institute of Electrical and Eletronics Engineers*) desenvolveu e aprovou normas para o uso, ficando conhecidas como IEEE 802.11, tendo sua primeira versão publicada em 1997 [Batalha 2018]. O padrão 802.11 não parou na primeira versão Experimental [Tanenbaum et al. 2003], ao decorrer dos anos a tecnologia sofreu evoluções e com elas novas opções para o seu uso e manuseio.

O padrão IEEE 802.11 define basicamente uma arquitetura para as Redes Locais Sem Fio (*Wireless Local Area Network - WLAN*) que abrange os níveis físico e de enlace. Transmissões com frequência de rádio são consideradas no nível físico, embora outras formas de transmissão sem fio possam ser usadas, como microondas, infravermelho e laser. Para o nível de enlace, o IEEE definiu um protocolo de controle de acesso ao meio (protocolo MAC), bastante semelhante ao protocolo usado em redes locais Ethernet (CSMA/CD) [Garcia 2012]. O padrão IEEE 802.11 possibilita a

transmissão de dados numa velocidade que atingem até 1300 Mbit/s (IEEE 802.11ac) [CISCO 2012, Bejarano et al. 2013].

Levando isso em consideração, este documento traz algumas avaliações realizadas em uma Rede Sem Fio 802.11 em modo infraestruturado. São utilizados 3 tipos de tráfegos diferentes, sendo CBR, Rajada e CBR/Rajada. Ainda, os impactos da adição de mobilidade aos clientes conectados à rede também são analisados.

O restante deste relatório está organizado como se segue. A Seção 2 apresenta uma revisão da literatura de conceitos utilizados neste experimento. Na Seção 4 as características do exercício proposto, o software utilizado nas simulações e definições importantes para o entendimento do trabalho. A Seção 5 apresenta os resultados obtidos nos experimentos. Finalmente, a Seção 6 apresenta as considerações finais acerca dos resultados e do experimento como um todo. **Vale ressaltar que ao final deste documento encontram-se os principais códigos utilizados nas simulações**.

2. Fundamentação Teórica

Esta Seção descreve os conceitos de redes de computadores utilizados para realizar as simulações. Aqui são descritos os protocolos utilizados, bem como os conceitos relacionados as métricas coletadas.

2.1. Protocolo UDP

O protocolo UDP (*User Datagram Protocol*) é um protocolo não orientado à conexão da camada de transporte do modelo TCP/IP. Ele não fornece nenhum tipo de tratamento e controle de erros por isso é considerado como um protocolo simples. De forma resumida, o UDP recebe mensagens do processo da aplicação que deseja enviar dados, anexa os campos de número de porta de origem e destino para o serviço de multiplexação/demultiplexação e transmite o segmento resultante para a camada de rede. A camada de rede encapsula o segmento da camada de transporte em um datagrama[1] IP e, em seguida, faz uma tentativa de melhor esforço para entregar o segmento ao destinatário [Kurose and Ross 2013].

Se o segmento conseguir chegar ao destinatário, o UDP usará o número da porta de destino para entregar os dados do segmento ao processo da aplicação correto. Observe que, com o UDP, não há *handshaking* entre o envio e o recebimento de entidades da camada de transporte. Por essa razão, o UDP é dito como não orientado à conexão. O objetivo dele é acelerar o processo de envio de dados, visto que todas as etapas de comunicação necessárias para verificar a integridade de um pacote contribuem para deixá-lo mais lento.

Ou seja, quando o protocolo UDP é acionado, ele apenas manda informações a um destinatário, sem a preocupação se elas irão ser recebidas de forma correta. Caso ocorram erros, simplesmente envia o próximo pacote e os anteriores não podem ser recuperados. Esse método garante uma comunicação rápida entre dois computadores, mesmo potencializando a ocorrência de erros. Graças à essas características, o protocolo é bastante usado em situações nas quais a correção de erros não é exatamente desejada, como *streamings* de vídeo ou jogos *online*, por exemplo.

[1] O datagrama IP é a unidade básica de dados no nível IP

2.2. Protocolo TCP

O Protocolo de Controle de Transmissão, ou apenas TCP (*Transmission Control Protocol*) é um dos principais protocolos da camada de transporte do modelo TCP/IP e é conhecido por ser o protocolo de transporte da Internet, confiável e orientado à conexão [Kurose and Ross 2013]. Ele permite gerenciar os dados vindo da camada inferior do modelo (ou seja, o protocolo IP). Quando os dados são fornecidos ao protocolo IP, este encapsula-os em datagramas IP. O TCP é um protocolo orientado à conexão, ou seja, ele permite que duas máquinas se comuniquem entre elas durante o processo de envio, que é conhecido como *handshaking*, além de controlar o estado da transmissão.

O TCP não só envia os dados como também recebe informações para se assegurar que os pacotes foram recebidos sem erros. Ele também adota um mecanismo próprio para se assegurar que os pacotes enviados vão chegar ao destino na ordem correta. Caso o receptor não receba um pacote corretamente, a informação é enviada novamente até que chegue de forma correta ao destino. Também há uma checagem se as informações não foram corrompidas no trajeto entre emissor e receptor.

2.3. Definições

- **Intervalo de confiança**: Um intervalo de confiança (IC) é um intervalo estimado de um parâmetro de interesse de uma população. Em vez de estimar o parâmetro por um único valor, é dado um intervalo de estimativas prováveis. Intervalos de confiança são usados para indicar a confiabilidade de uma estimativa [Shimakura 2012].
- **Vazão**: A vazão é definida como o número de bits que podem ser transmitidos sobre a rede num dado tempo, sendo expressa em bits/segundo (b/s). Por exemplo, numa rede Ethernet podemos ter como vazão 10Mb/s. Muitas vezes usa-se o termo taxa de transmissão para se referir a vazão [Cantu 2015].
- **Atraso**: O atraso é o tempo que um pacote leva para atravessar uma rede desde a origem até o destino, passando pelos roteadores e enlaces intermediários. Ao percorrer uma rede, um pacote sofre uma série de atrasos em cada um dos nós do caminho. Este atraso em cada nó da rede tem quatro componentes principais: o atraso de processamento, o atraso de fila, o atraso de transmissão e o atraso de propagação [Cantu 2015].
- **Perda de pacotes**: A perda de pacotes ocorre quando a capacidade de armazenamento da fila de um roteador se esgota. Neste caso os novos pacotes que chegam são descartados (*dropped*) e são considerados perdidos. A fração dos pacotes perdidos aumenta a medida que a intensidade de tráfego aumenta. Pacotes corrompidos totalmente também podem ser considerados como perdidos [Cantu 2015].
- **Tráfego CBR**: Aplicado a conexões que necessitam de banda fixa (estática) devido aos requisitos de tempo bastante apertados entre a origem e o destino. Aplicações típicas deste serviço são: áudio interativo (telefonia), distribuição de áudio e vídeo (televisão, pay-per-view, etc), e áudio e vídeo *on demand* [em Telecomunicações 2012].
- **Tráfego em Rajada**: Aplicado a conexões que transportam tráfego em rajadas que necessitam de garantia de banda, embora a taxa de bits possa variar. Aplicações típicas deste serviço são as interligações entre redes e a emulação de LAN's [em Telecomunicações 2012].

3. Exercício proposto

O exercício proposto para as avaliação se refere à simular uma Rede sem Fio 802.11 no modo infraestruturado com um Ponto de Acesso (*Access Point - AP*). Mostrar Vazão, Atraso e Perda de Pacotes para um dispositivo móvel que se move do *AP* para a borda da rede. Em seguida, mostrar Vazão, Atraso e Perda para os casos de 2, 4 e 6 dispositivos móveis transmitindo simultaneamente, nos cenários com e sem mobilidade. Considerar a Rede sem Fio conectada à Internet por um enlace cabeado de 100 Mbps.

Usar tráfego em rajadas e fontes CBR (*Constant Bit Rate*). Tamanho de quadro: 512 (CBR) e 1500 bytes (Rajada). Mostrar os resultados para tráfego TCP, UDP e 50% de cada, chamada aqui de CBR/Rajadas, (usar intervalo de confiança de 95%).

4. Avaliação

Esta seção descreve a metodologia e métricas utilizadas para avaliar a eficiência das aplicações de CBR, Rajada e CBR/Rajada em termos de atraso médio, vazão e número pacotes perdidos. Foram realizadas simulações com diferentes números de clientes a fim de identificar o impacto disso para a rede.

4.1. Metodologia

As simulações foram realizadas no Network Simulator 3 (NS3)[2], o qual implementa a pilha de protocolo do padrão IEEE 802.11a para comunicação e atenuação de sinal. O IEEE 802.11a está definido como padrão no NS3 por isso optou-se por utilizá-lo neste trabalho. Para estabelecer um cenário de avaliação, utilizou-se uma área de 140m x 140m. Para os nós estáticos foi utilizado o modelo de mobilidade `ConstantPositionMobility` e para o cenário com mobilidade foi utilizado o modelo `ConstantVelocityMobility`, ambos fornecido pelo NS3.

A taxa de dados foi definida em 512 kbits e a potência de transmissão em 16 dbm, conforme padrão estabelecido no modelo `YansWifiPhy` do NS3[3]. Esses parâmetros, juntamente com o modelo de propagação `FriisPropagationLossModel`, fornecem um raio de comunicação de $\sim$ 70 metros. Cada simulação foi executada 33 vezes com diferentes sementes de aleatoriedade e os resultados apresentam os valores com um intervalo de confiança de 95%. O tempo de simulação foi de 60 segundos.

Para quantificar a evolução do tráfego neste cenário, variou-se o número de clientes conectados à rede em $1, 2, 4$ e 6 dispositivos. Ainda, dois cenários de mobilidade foram definidos. O primeiro com ausência de mobilidade, onde os nós são dispostos próximos ao AP e ficam estáticos até o final da simulação. No segundo cenário, os nós clientes possuem mobilidade que varia entre 1 e 2 m/s (igual a 3.6 e 7.2 km/h), o que se aproxima da mobilidade de pessoas caminhando. Considerou-se que os clientes se distanciam do AP até próximo à borda da rede.

A conexão estabelecida entre o *Router*, que representa a Internet, e o *AP* se dá através de um enlace Ethernet de 100Mbps com atraso de 2 milissegundos, conforme especificação padrão do próprio NS3 através da classe `ns3::CsmaHelper`. O algoritmo utilizado para controle de taxa foi o AARF que é disponibilizado através da

[2] `https://www.nsnam.org/`

[3] `https://www.nsnam.org/doxygen/classns3_1_1_yans_wifi_phy.html`

classe `ns3::AarfWifiManager`. Controle de taxa é a determinação da taxa de transmissão de dados ideal mais apropriada para as condições atuais do canal sem fio. Consiste em avaliar as condições do canal e, consequentemente, ajustar a taxa. A adaptação da taxa é bastante desafiadora devido às flutuações das condições do canal [Lacage et al. 2004, Wong et al. 2006].

A organização dos dispositivos no cenário se dá da seguinte forma: o $Router$ está posicionado nas coordenadas $x = 0, y = 0$, o *AP* está posicionado no centro do cenário ($x = 70, y = 70$), e os clientes são alocados em suas posições iniciando de $x = 70, y = 70$ com $\Delta X = 5$, $\Delta Y = 5$ e organização de layout $RowFirst = 3$. Tais configurações fazem que o primeiro cliente fique bem embaixo do *AP*, cada cliente fique à 5 metros de distância um do outro e organizados em grupos de 3 em 3 por linha, conforme ilustrado na Figura 1.

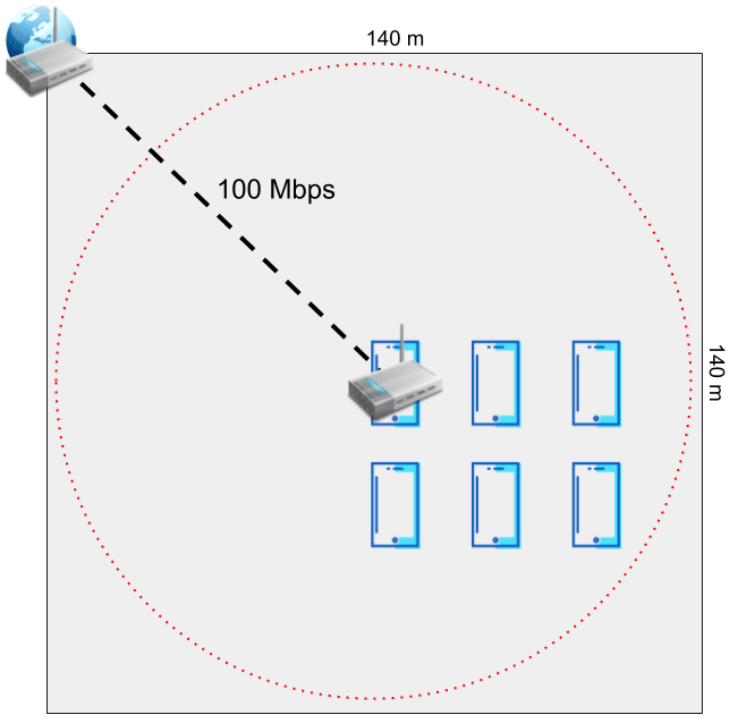


Figura 1. Cenário dos experimentos.

Para capturar os resultados gerados em cada simulação foi utilizada a classe de rastreamento de fluxos `ns3::FlowMonitor`. Ao final de cada simulação, as informações foram guardadas em arquivos de texto e depois tratadas com a linguagem Python para organização e plotagem dos gráficos. Um exemplo do arquivo gerado com 1 cliente e tráfego CBR é exibido abaixo.

```
Build commands will be stored in build/compile_commands.json
'build' finished successfully (3.827s)
FlowID[1]   Trafego   CBR
FlowID[1]   Cliente   1
FlowID[1]   Fluxo   10.1.2.2 -----> 192.168.0.2
FlowID[1]   Duracao   58.9903
FlowID[1]   Vazao(Mbps)   0.51693
```

```
FlowID[1]   Atraso   0.00215035
FlowID[1]   PacotesTransmitidos   7899
FlowID[1]   PacotesPerdidos   0
```

A atribuição de tráfego em cada nó cliente é estabelecida através de uma variável chamada $trafego$ que por padrão recebe valor 0. Assim, quando $trafego = 0$, o tráfego CBR é instalado em todos os clientes e quando $trafego = 1$, o tráfego Rajada é instalado em todos os clientes. Porém, quando o tráfego escolhido for $trafego = 2$, 50% dos clientes devem utilizar tráfego CBR e os outros 50% utilizar tráfego em Rajada. Para essa atribuição, é feita uma verificação através do id de cada cliente, conforme mostrado na Equação 1.

$$trafego(id) = \begin{cases} CBR, & \text{se } trafego = 0 \\ Rajada, & \text{se } trafego = 1 \\ CBR/Rajada, & \text{se } trafego = 2 \begin{cases} CBR, & \text{se } id \bmod 2 = 0 \\ Rajada, & \textit{Caso contrário} \end{cases} \end{cases} \quad (1)$$

Se um cenário de $trafego = 2$ possuir apenas 1 cliente, o mesmo receberá o tráfego CBR, já que o id do cliente na estrutura que comporta os nós no simulador (`NodeContainer`) é igual a 0 e $0 \bmod 2 = 0$. Porém, se um cenário de mesmo tráfego possuir 2 clientes, o cliente de $id = 0$ receberá o tráfego CBR e o de $id = 1$ receberá o tráfego Rajada, já que $1 \bmod 2 \neq 0$. Tornando assim a simulação com 50% de tráfego CBR e 50% de Rajada.

Por fim, a Tabela 1 resume os parâmetros utilizados nas simulações para ambos os cenários estabelecidos.

Tabela 1. Parâmetros de simulação para os cenários

Parâmetro	**Valor**
Tempo de simulação	60 s
Número de cliente	1, 2, 4 e 6
Tamanho do cenário	140m x 140m
Padrão de comunicação	IEEE 802.11a
Potência de transmissão	16 dbm
Velocidade dos clientes no cenário estático	0 km/h
Velocidade dos clientes no cenário móvel	3.6 a 7.2 km/h
Modelo de mobilidade para nós estáticos	`ConstantPositionMobility`
Modelo de mobilidade para nós móveis	`ConstantVelocityMobility`
Taxa de dados	512 kbps
Tamanho do quadro UDP	512 bytes
Tamanho do quadro TCP	1500 bytes
Número de antenas no AP	1
Aplicações	CBR, Rajada e CBR/Rajada

5. Resultados

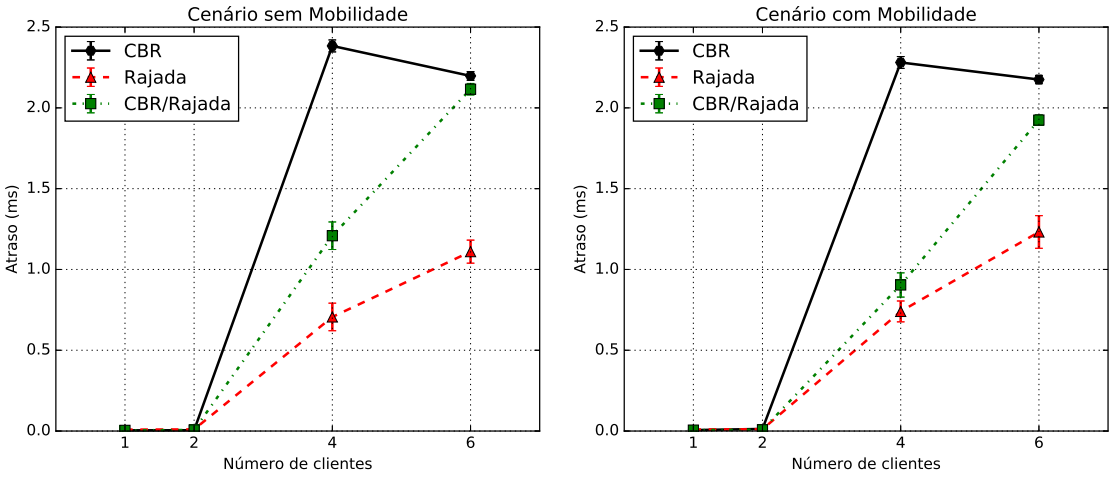
Os resultados obtidos nas simulações são apresentados nas subseções 5.1, 5.2 e 5.3 referentes ao Atraso, Vazão e Perda de Pacotes, respectivamente. Todos os resultados contam com os 3 tipos de tráfegos utilizados, sendo CBR, Rajada e CBR/Rajada.

5.1. Atraso

A Figura 2 exibe os resultados de atraso, em milissegundos, obtidos nos cenários sem e com mobilidade. Especificamente na Figura 2(a), pode-se notar que com 1 e 2 clientes conectados o atraso é bem baixo para ambos os tráfegos. No entanto, tem um crescimento com 4 clientes e decaimento com 6 clientes conectados. Especialmente os dispositivos com tráfego CBR tiveram maiores taxas de atraso, seguidos pelo CBR/Rajada e Rajadas. O aumento do atraso do CBR pode ser justificado pelo fato da fila de pacotes do UDP estar enchendo, o que ocasiona aumento no atraso de espera de cada pacote até sua transmissão.

Outro ponto importante é que o aumento do atraso de 2 para 4 clientes conectados ocasionado pelo enchimento da fila de transmissão tem impacto direto no número de pacotes perdidos, já que a fila sem espaço livre faz com que os pacotes que chegam para transmissão sejam perdidos. Esse comportamento será melhor visualizado na Figura 4(a) mais à frente.

O gráfico ilustrado na Figura 2(b) segue o padrão descrito anteriormente. Porém, podemos notar que a mobilidade presente nos nós faz com que o atraso com 4 e 6 clientes na rede seja maior que o cenário sem mobilidade. Ou seja, a mobilidade faz com que os algoritmos de controle levem em consideração essa movimentação para entrega dos pacotes.



(a) Sem mobilidade (b) Com mobilidade

Figura 2. Resultados de atraso nos cenários sem e com mobilidade.

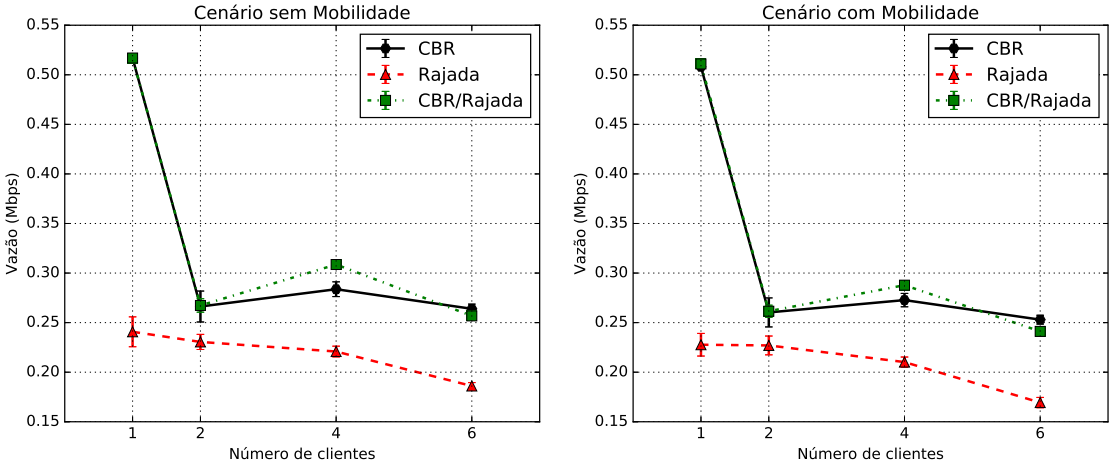
5.2. Vazão

A Figura 3 exibe o resultado de Vazão, em Megabit por segundo (Mbps), nos cenários estabelecidos. Podemos observar que o tráfego de Rajada possui menor vazão, em relação ao CBR e CBR/Rajada. Tanto CBR quanto CBR/Rajada utilizam toda a vazão disponível

no ambiente com 1 cliente conectado, porém essa vazão sofre queda a partir do momento que outro dispositivo passa a concorrer pelo canal de comunicação. Esse comportamento pode ser visualizado tanto na Figura 3(a) quanto na Figura 3(b), entretanto, no cenário com mobilidade temos taxas menores de vazão para todos os números de clientes conectados.

Conforme o número de clientes conectados à rede cresce, os clientes que realizam o tráfego em Rajada apresentaram queda na Vazão. Esse comportamento é justificado pela utilização do protocolo TCP, que para controlar as taxas de perdas de pacotes durante a transmissão aumenta sua janela de contenção. Em contrapartida, os clientes que utilizam tráfego CBR possuem maior Vazão pelo fato desse tipo de tráfego empregar taxa constante de transferência e não levar em consideração o número de colisões e pacotes que possam se perder devido ao crescimento do número de clientes conectados à rede.

Especialmente no tráfego CBR/Rajada no cenário em que apenas 1 cliente está conectado, a Vazão é a máxima do canal pelo fato da atribuição de tráfego realizada nos experimentos seguir a regra explicada anteriormente, na Subseção 4.1. Ou seja, apenas o tráfego CBR está transmitindo neste cenário específico.



(a) Sem mobilidade (b) Com mobilidade

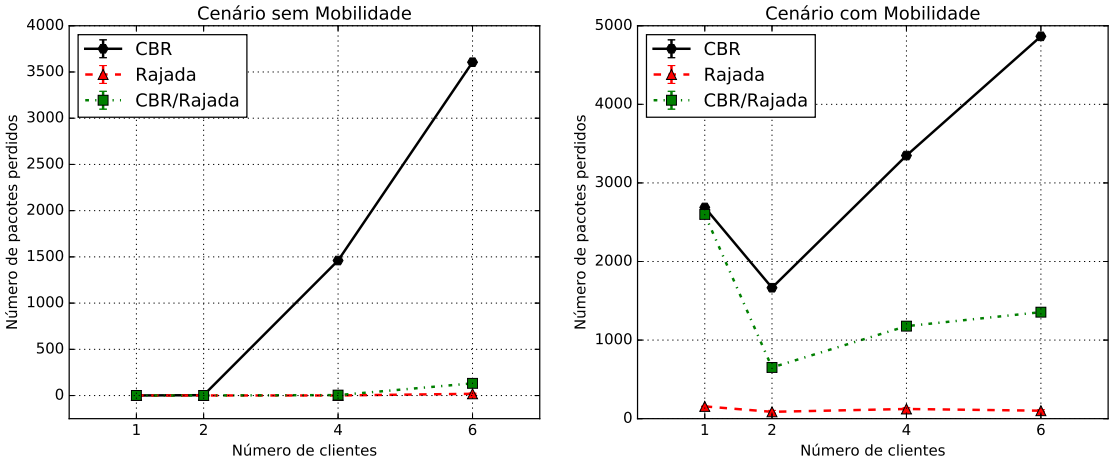
Figura 3. Resultados de vazão nos cenários sem e com mobilidade.

5.3. Perda de Pacotes

A Figura 4 exibe o resultado da Perda de Pacotes em valores absolutos. Na Figura 4(b), podemos visualizar que nos cenários contendo 1 e 2 clientes conectados o número de perda de pacotes é próximo à zero, tendo aumento à medida que o número de clientes conectados também aumenta. As redes que utilizaram tráfego em rajada não apresentaram um aumento significativo no número de pacotes perdidos durante a simulação, resultados justificados pelo uso do protocolo TCP nesse cenário. Ou seja, ao passo que o TCP detecta um aumento no número de colisões devido ao aumento no número de dispositivos atuantes na rede, o TCP aumenta sua janela de congestionamento, diminuindo a vazão do link, conforme visto anteriormente na Figura 3, e assim mantém a taxa de perda de pacotes estável.

No entanto, ao se analisar o gráfico mostrado na Figura 4(b) podemos notar que a mobilidade influencia significativamente na perda de pacotes mesmo em cenários com

poucos dispositivos atuantes na rede. Tanto o tráfego CBR quanto CBR/Rajada tiveram baixas taxas de perdas de pacotes no cenário sem mobilidade com 1 e 2 clientes, porém com a presença de mobilidade as taxas de perda aumentam drasticamente. Ainda, o CBR/-Rajada tem destaque pelo aumento significativo de perda em todos os cenários, diferente da avaliação sem mobilidade.



(a) Sem mobilidade (b) Com mobilidade

Figura 4. Resultados de perda de pacotes nos cenários sem e com mobilidade.

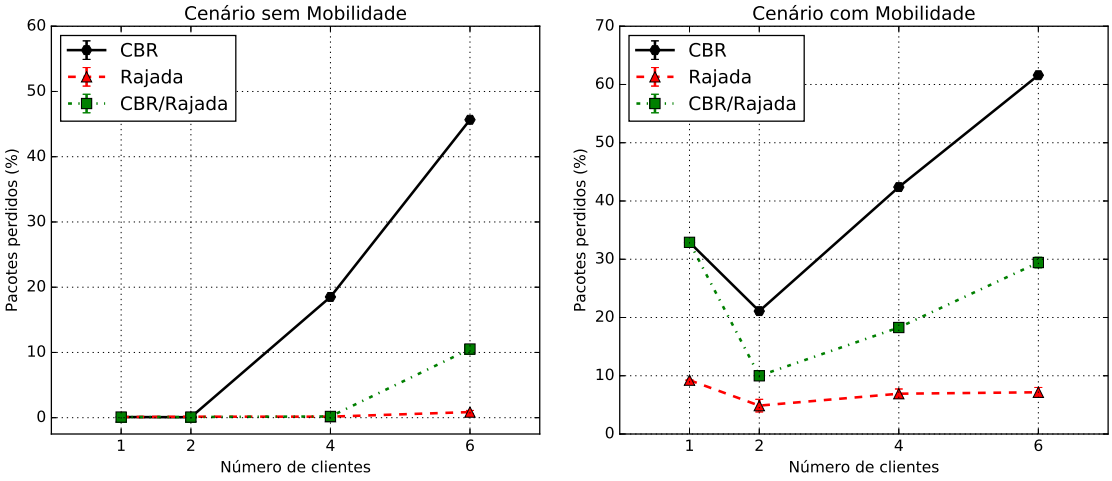
A Figura 5 também exibe o resultado da Perda de Pacotes, porém em porcentagem. Essa avaliação é interessante pois mostra a porcentagem de pacotes que foram perdidos em relação ao total de pacotes transmitidos. Assim, podemos ter uma ideia melhor do impacto que essas perdas podem ocasionar às aplicações que utilizam os tráfegos avaliados. Considerando o maior número de clientes conectados à rede no cenário sem mobilidade, as taxas de perda de pacotes são 45% para CBR, 10.1% para CBR/Rajada e 1% para Rajada. Já no cenário com mobilidade, e também com 6 clientes conectados, as taxas são 82% para CBR, 39% para CBR/Rajada e 7% para Rajada. Assim, evidencia-se ainda mais que a mobilidade impacta nos ambientes de comunicação em redes de computadores.

5.4. Avaliação extra

Para melhor visualização do impacto de utilização dos diferentes tipos de tráfegos com diferentes números de clientes, foi montado um conjunto de novas simulações com um número maior de clientes conectados ao AP. Assim, as configurações de comunicação são as mesmas utilizadas nas simulações anteriores, porém com o número de clientes variando em $6, 12, 24$ e 48. A Tabela 2 resume os parâmetros utilizados neste experimento extra.

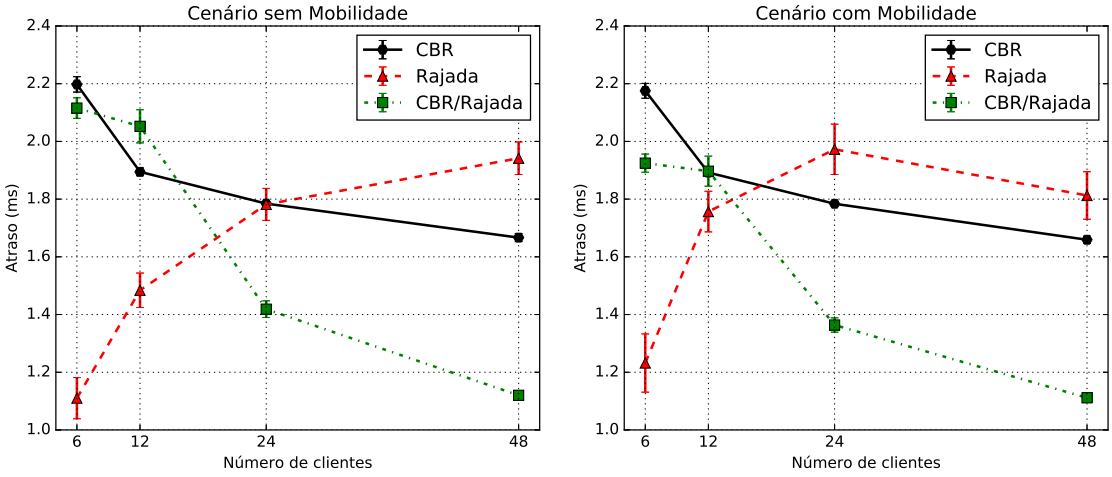
Desta forma, a Figura 6 exibe o Atraso dos dois cenários, com 6, 12, 24 e 48 clientes atuantes na rede. Podemos observar que o atraso do tráfego em rajada fica superior ao CBR e CBR/Rajada a partir de 24 clientes tanto no cenário sem mobilidade quanto no cenário com mobilidade. Isso se justifica pelo elevado número de clientes e maior controle de tráfego realizado pelo TCP que é utilizado no tráfego em Rajada.

A Figura 7 exibe a Vazão para este mesmo cenário. Pode-se observar que a vazão mantém o comportamento de decaimento à medida que o número de clientes cresce na rede. O tráfego em Rajada mantém seu comportamento de menor vazão que os tráfegos



(a) Porcentagem - Sem mobilidade (b) Porcentagem - Com mobilidade

Figura 5. Resultados de perda de pacotes em porcentagem nos cenários sem e com mobilidade.



(a) Sem mobilidade (b) Com mobilidade

Figura 6. Resultados de atraso nos cenários de 6, 12, 24 e 48 clientes sem e com mobilidade.

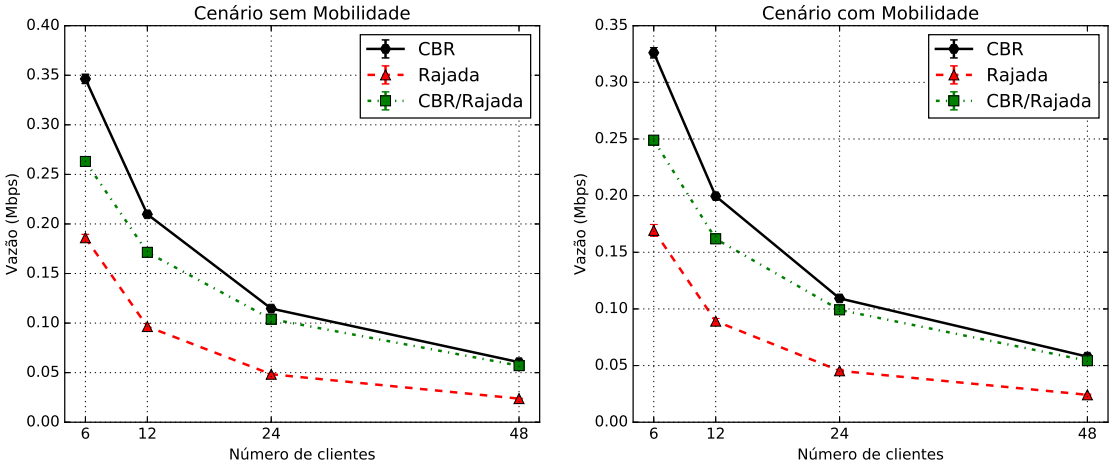
CBR e CBR/Rajada. Um ponto importante para observação é que o padrão de menor vazão no cenário com mobilidade também se mantém aqui, reforçando ainda mais o que foi discutido acerca da mobilidade influenciar diretamente o comportamento da rede como um todo.

A Figura 8 exibe os resultados de Perda de Pacotes em valores absolutos. Assim, podemos visualizar que o tráfego Rajada tem menores taxas de perda também nos cenários com maior número de clientes atuantes na rede. E ambos os tráfegos CBR e CBR/Rajada mantém seus comportamentos, onde o CBR/Rajada fica no meio termo entre CBR e Rajada por utilizar 50% de cada tráfego. Aqui também podemos visualizar o impacto que a mobilidade exerce na rede.

Por fim, a Figura 9 exibe as taxas de Perda de Pacotes em porcentagem para cada

Tabela 2. Parâmetros de simulação para o com maior número de clientes

Parâmetro	Valor
Tempo de simulação	60 s
Número de cliente	6, 12, 24 e 48
Tamanho do cenário	140m x 140m
Padrão de comunicação	IEEE 802.11a
Potência de transmissão	16 dbm
Velocidade dos clientes no cenário estático	0 km/h
Velocidade dos clientes no cenário móvel	3.6 a 7.2 km/h
Modelo de mobilidade para nós estáticos	`ConstantPositionMobility`
Modelo de mobilidade para nós móveis	`ConstantVelocityMobility`
Taxa de dados	512 kbps
Tamanho do quadro UDP	512 bytes
Tamanho do quadro TCP	1500 bytes
Número de antenas no AP	1
Aplicações	CBR, Rajada e CBR/Rajada



(a) Sem mobilidade (b) Com mobilidade

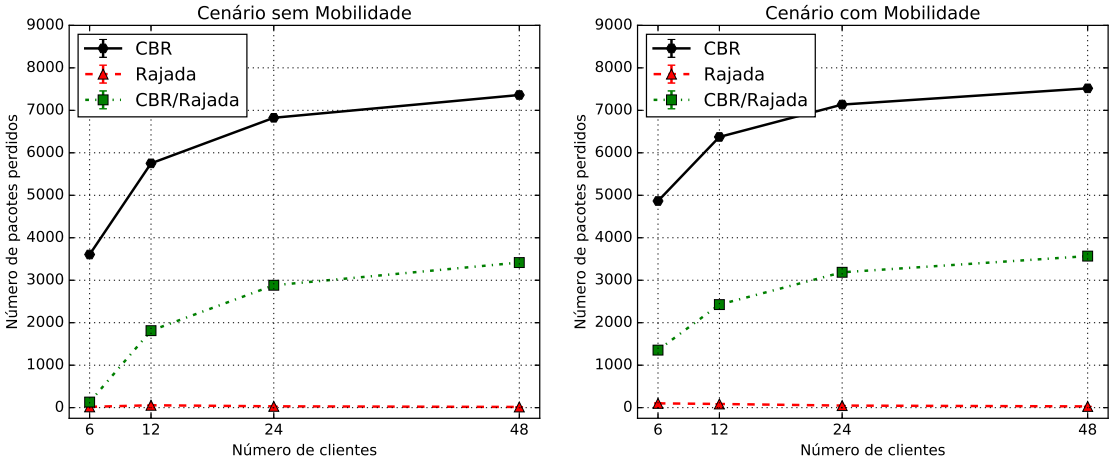
Figura 7. Resultados de vazão nos cenários de 6, 12, 24 e 48 clientes sem e com mobilidade.

tráfego utilizado. Pode-se observar aqui que a mobilidade exerce influência significativa até no tráfego em Rajada, que utiliza o TCP e possui maior controle nas transmissões, fazendo com que a diferença de da taxa de perda suba de 4% no cenário sem mobilidade para 20% no cenário com mobilidade. Os outros tráfegos, conforme esperado, também sofrem aumentos em suas taxas de perda de pacotes.

6. Considerações finais

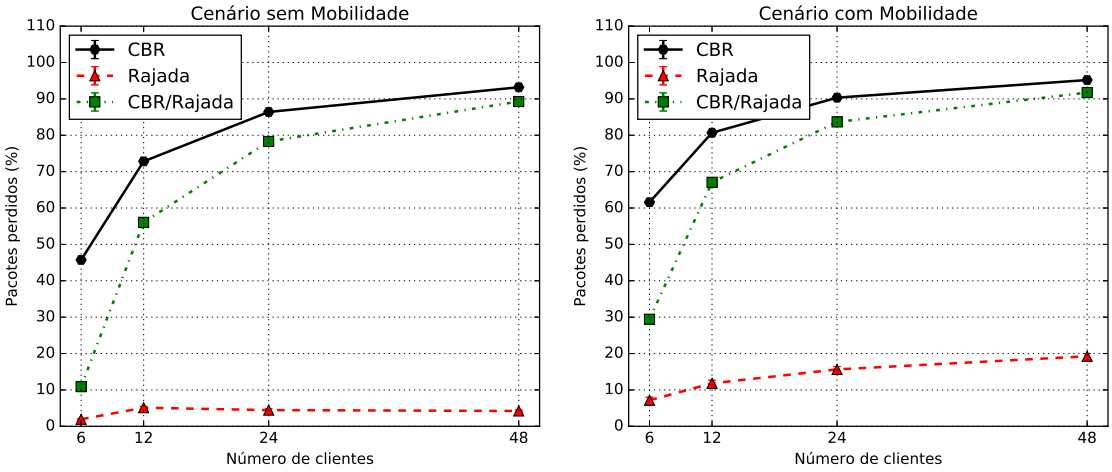
Com base nos resultados apresentados na Seção 5 pôde ser observado que os clientes que utilizam tráfego CBR para suas transmissões apresentaram maiores taxas de atraso comparados com os clientes que utilizam o tráfego em Rajada, tais taxas podem ser justificadas por conta do enchimento da fila de transmissão utilizada no protocolo UDP.

Em relação à Vazão, os clientes que utilizam CBR possuem maiores taxas de vazão



(a) Sem mobilidade (b) Com mobilidade

Figura 8. Resultados de perda de pacotes nos cenários de 6, 12, 24 e 48 clientes sem e com mobilidade.



(a) Porcentagem - Sem mobilidade (b) Porcentagem - Com mobilidade

Figura 9. Resultados de perda de pacotes nos cenários de 6, 12, 24 e 48 clientes sem e com mobilidade.

em função do protocolo UDP não realizar nenhum controle na rede e tomar proveito de todo recurso que estiver disponível no momento da transmissão. Diferente do tráfego em Rajada, que faz uso do protocolo TCP, e existe um controle da rede para evitar colisões e, consequentemente, perda de pacotes.

No que se refere à Perda de Pacotes, observou-se que tanto o tráfego CBR quanto CBR/Rajada tiveram maiores taxas de perda de pacotes em comparação ao tráfego em Rajada. Isso de deu pelo fato do tráfego em Rajada utilizar o protocolo TCP e seus procedimentos de controle, diferente do UDP que não utiliza nenhum procedimento que garanta a entrega dos pacotes.

Observou-se também que em ambas avaliações com a presença de mobilidade nos clientes atuantes na rede as taxas de Atraso, Vazão e Perda de Pacotes sofreram impacto direto. Por fim, os resultados se mostram coerentes em ambos os cenários quando compa-

rados Atraso, Vazão e Perda de Pacotes juntos. Onde, a medida que o número de clientes cresce na rede, a vazão diminui e, dependendo do tipo de tráfego, o atraso de entrega sofre aumenta ou diminui.

Referências

Batalha, I. (2018). Estudo da tecnologia ieee 802.11ac para o desenvolvimento de modelos empírico e cross-layer. Master's thesis, Universidade Federal do Pará.

Bejarano, O., Knightly, E. W., and Park, M. (2013). Ieee 802.11 ac: from channelization to multi-user mimo. *IEEE Communications Magazine*, 51(10):84–90.

Cantu, E. (2015). Vazão, atraso e perda de pacotes.

CISCO (2012). 802.11ac: The fifth generation of wi-fi technical white paper. Technical report, CISCO, https://www.cisco.com/c/dam/en/us/products/collateral/wireless/aironet-3600-series/white-paper-c11-713103.pdf.

em Telecomunicações, T. C. (2012). Comunicação de dados: Tecnologias.

Garcia, L. G. U. (2012). Redes 802.11 (camada de enlace).

Kurose, J. F. and Ross, K. W. (2013). *Computer networking: a top-down approach: international edition*. Pearson Higher Ed.

Lacage, M., Manshaei, M. H., and Turletti, T. (2004). Ieee 802.11 rate adaptation: a practical approach. In *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 126–134. ACM.

Shimakura, S. E. (2012). Intervalo de confiança.

Tanenbaum, A. S. et al. (2003). Computer networks, 4-th edition. *ed: Prentice Hall*.

Wong, S. H., Yang, H., Lu, S., and Bharghavan, V. (2006). Robust rate adaptation for 802.11 wireless networks. In *Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 146–157. ACM.

Códigos utilizados nas simulações

Código "gerencia2018.cc" do NS3

Foi utilizada a opção de passagem de parâmetros via terminal do simulador NS3, habilitada através da classe `ns3::CommandLine` (Linha 92). Assim, foram definidos os seguintes parâmetros para as simulações: *seed, nwifi, mobilidade* e *trafego*. Tais parâmetros são passados diretamente via terminal no *script* `run_gerencia2018.sh` que automatiza o processo de avaliação.

Para atribuir velocidade aos clientes no cenário com mobilidade foi utilizada a classe `ns3::ConstantVelocityMobility` e seu método para configuração de velocidade `SetVelocity`. Assim, a velocidade dos clientes foi atualizada a cada 0.5 segundos (Linha 46) no método `AtualizaVelocidade`. A velocidade configurada é definida a cada chamada de `AtualizaVelocidade` (Linha 200) obedecendo o intervalo definido entre 1 e 2 m/s, para que a velocidade de cada cliente seja ligeiramente diferente e deixe os experimentos mais próximo do que acontece no mundo real.

```
1  /* Projeto: Rede 802.11
2   * Disciplina: MO655 - Gerencia de Redes de Computadores - 2018
3   * Professor: Edmundo Madeira
4   * Autores: Joahannes B. D. da Costa e Wellington V. Lobato
       Junior.
5   * Email: joahannes, wellington{@lrc.ic.unicamp.br}
6   * Laboratorio de Redes de Computadores - LRC
7   */
8
9  #include "ns3/core-module.h"
10 #include "ns3/network-module.h"
11 #include "ns3/applications-module.h"
12 #include "ns3/wifi-module.h"
13 #include "ns3/mobility-module.h"
14 #include "ns3/internet-module.h"
15 #include "ns3/csma-module.h"
16 #include "ns3/point-to-point-module.h"
17 #include "ns3/netanim-module.h"
18 //Monitor de fluxo
19 #include "ns3/flow-monitor-module.h"
20 #include "ns3/flow-monitor.h"
21 #include "ns3/flow-monitor-helper.h"
22 #include "ns3/rng-seed-manager.h"
23 #include "ns3/random-variable-stream.h"
24
25 // Topologia da rede
26 //
27 //    10.1.1.0
28 // r0 *--------* AP
29 //                |        192.168.0.0
30 //                *    *    *    *    *    *
31 //                |    |    |    |    |    |
32 //                c0   c1   c2   c3   c4   c5
```

```
33
34 using namespace ns3;
35
36 NS_LOG_COMPONENT_DEFINE ("projetoGerencia2018");
37
38 //Mobilidade para os dispositivos moveis
39 void AtualizaVelocidade(Ptr<Node> node) {
40     Ptr<UniformRandomVariable> rvar = CreateObject<
       UniformRandomVariable>();
41     Ptr <ConstantVelocityMobilityModel> mobility = node ->
       GetObject<ConstantVelocityMobilityModel>();
42
43     double velocidade = rvar->GetValue(1, 2); // 3.6 a 7.2 km/h
44     mobility->SetVelocity (Vector (velocidade, 0.0, 0.0));
45
46     Simulator::Schedule (Seconds (0.5), AtualizaVelocidade, node
       );
47 }
48
49 void ImprimeMetricas (FlowMonitorHelper* fmhelper, Ptr<
   FlowMonitor> monitor, int trafego, int clientes){
50         double totalThroughput = 0.0;
51         monitor->CheckForLostPackets();
52         std::map<FlowId, FlowMonitor::FlowStats> flowStats =
           monitor->GetFlowStats();
53         Ptr<Ipv4FlowClassifier> classifier = DynamicCast<
           Ipv4FlowClassifier> (fmhelper->GetClassifier());
54
55         for (std::map<FlowId, FlowMonitor::FlowStats>::
           const_iterator stats = flowStats.begin (); stats !=
           flowStats.end (); ++stats)
56         {
57                 Ipv4FlowClassifier::FiveTuple fiveTuple =
                   classifier->FindFlow (stats->first);
58                 std::string flowidhost = "FlowID[" + std::
                   to_string(stats->first) + "]";
59
60                 if(trafego == 0)
61                         std::cout << flowidhost <<"   Trafego
                           UDP" << std::endl;
62                 else if(trafego == 1)
63                         std::cout << flowidhost <<"   Trafego
                           TCP" << std::endl;
64                 else if(trafego == 2)
65                         std::cout << flowidhost <<"   Trafego
                           UDP/TCP" << std::endl;
66
67                 std::cout << flowidhost <<"   Clientes   " <<
                   clientes << std::endl;
```

```
68                  std::cout << flowidhost <<"   Fluxo   "<<
                       fiveTuple.sourceAddress <<" -----> "<<
                       fiveTuple.destinationAddress<<std::endl;
69                  std::cout << flowidhost <<"   Duracao   " <<
                       stats->second.timeLastRxPacket.GetSeconds() -
                        stats->second.timeFirstTxPacket.GetSeconds()
                       <<std::endl;
70                  totalThroughput = (stats->second.rxBytes * 8.0 /
                        (stats->second.timeLastRxPacket.GetSeconds()
                        - stats->second.timeFirstTxPacket.GetSeconds
                       ())/1024/1024);
71                  std::cout << flowidhost <<"   Vazao(Mbps)   "<<
                       totalThroughput <<std::endl;
72                  std::cout << flowidhost <<"   Atraso   " <<
                       stats->second.delaySum.GetSeconds () / stats
                       ->second.rxPackets << std::endl;
73                  std::cout << flowidhost <<"
                       PacotesTransmitidos   " << stats->second.
                       txPackets << std::endl;
74                  std::cout << flowidhost <<"   PacotesPerdidos
                       " << stats->second.lostPackets << std::endl;
75          }
76          Simulator::Schedule(Seconds(1),&ImprimeMetricas,
               fmhelper, monitor, trafego, clientes);
77  }
78
79
80  int main (int argc, char *argv[])
81  {
82    uint16_t seed = 1; //Seeds
83    bool verbose = true; //Verbose
84    uint32_t nWifi = 1; //Quantidade de nos WiFi
85    uint32_t apWifi = 1; //Quantidade de APs
86    bool mobilidade = false; //Sem mobilidade por padrao
87    double distance = 5.0; //Distancia entre os nos
88    double simTime = 60.0; //Tempo de simulacao
89    int trafego = 0; //UDP por padrao
90
91  //Parametros por linha de comando
92    CommandLine cmd;
93    cmd.AddValue ("seed", "Semente de simulacao", seed);
94    cmd.AddValue ("nWifi", "Numero de clientes", nWifi);
95    cmd.AddValue ("trafego", "(0) UDP, (1) TCP ou (2) para ambos",
          trafego);
96    cmd.AddValue ("mobilidade", "(0) Sem mobilidade e (1) Com
         mobilidade", mobilidade);
97    cmd.Parse (argc,argv);
98
99  //Seed
```

```
100   RngSeedManager::SetSeed(12);
101   RngSeedManager::SetRun(seed);
102   Ptr<UniformRandomVariable> u = CreateObject<
          UniformRandomVariable> ();
103
104   if (nWifi > 250)
105   {
106     std::cout << "Muitos nos wifi criados. Nosso limite e' 250."
              << std::endl;
107     return 1;
108   }
109
110 //Cria apenas nos WiFi
111   NodeContainer wifiStaNodes;
112   wifiStaNodes.Create (nWifi);
113
114 //Cria no AP
115   NodeContainer wifiApNode;
116   wifiApNode.Create (apWifi);
117
118 //Cria Router
119   NodeContainer csmaNodes;
120   csmaNodes.Add(wifiApNode.Get(0));
121   csmaNodes.Create (1);
122
123 //Cria conexao entre Router e AP
124   CsmaHelper csma;
125   csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"))
          ;
126   csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)
          ));
127
128   NetDeviceContainer csmaDevices;
129   csmaDevices = csma.Install (csmaNodes);
130
131 //Cria os nos WiFi e os interliga por meio de um canal
132   YansWifiChannelHelper channel = YansWifiChannelHelper::Default
          ();
133   YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
134   phy.SetChannel (channel.Create ());
135
136 //Algoritmo de controle de de taxa, que neste caso e o AARF
137   WifiHelper wifi;
138   wifi.SetRemoteStationManager ("ns3::AarfWifiManager");
139
140   WifiMacHelper mac;
141
142 //SSID do AP
143   Ssid ssid = Ssid ("GerenciaRedes2018");
```

```
144     //Clientes
145     mac.SetType ("ns3::StaWifiMac", "Ssid", SsidValue (ssid), "
           ActiveProbing", BooleanValue (false));
146     NetDeviceContainer staDevices = wifi.Install (phy, mac,
           wifiStaNodes);
147     //AP
148     mac.SetType ("ns3::ApWifiMac", "Ssid", SsidValue (ssid));
149     NetDeviceContainer apDevices = wifi.Install (phy, mac,
           wifiApNode);
150
151   //Instancia a Mobilidade
152     //AP_node
153     Ptr<ListPositionAllocator> positionAlloc0 = CreateObject<
           ListPositionAllocator> ();
154     positionAlloc0->Add (Vector(70, 70, 0));
155
156     MobilityHelper mobilityAp;
157     mobilityAp.SetMobilityModel("ns3::
           ConstantPositionMobilityModel");
158     mobilityAp.SetPositionAllocator(positionAlloc0);
159     mobilityAp.Install(wifiApNode);
160
161     //Router
162     Ptr<ListPositionAllocator> positionAllocRouter = CreateObject<
           ListPositionAllocator> ();
163     positionAllocRouter->Add (Vector(0, 0, 0));
164
165     MobilityHelper router;
166     router.SetMobilityModel("ns3::ConstantPositionMobilityModel");
167     router.SetPositionAllocator(positionAllocRouter);
168     router.Install(csmaNodes.Get(1));
169
170     //WiFi_nodes
171     if (mobilidade == false)
172     {
173       MobilityHelper mobilityWifi;
174       mobilityWifi.SetPositionAllocator ("ns3::
             GridPositionAllocator",
175           "MinX", DoubleValue (70.0),
176           "MinY", DoubleValue (70.0),
177           "DeltaX", DoubleValue (distance),
178           "DeltaY", DoubleValue (distance),
179           "GridWidth", UintegerValue (3),
180           "LayoutType", StringValue ("RowFirst"));
181       mobilityWifi.SetMobilityModel("ns3::
             ConstantVelocityMobilityModel");
182       mobilityWifi.Install (wifiStaNodes);
183     }
184     else
```

```
185   {
186     MobilityHelper mobilityWifi;
187     mobilityWifi.SetPositionAllocator("ns3::
          GridPositionAllocator",
188         "MinX", DoubleValue (70.0),
189         "MinY", DoubleValue (70.0),
190         "DeltaX", DoubleValue (distance),
191         "DeltaY", DoubleValue (distance),
192         "GridWidth", UintegerValue (3),
193         "LayoutType", StringValue ("RowFirst"));
194     mobilityWifi.SetMobilityModel("ns3::
          ConstantVelocityMobilityModel");
195     mobilityWifi.Install (wifiStaNodes);
196
197     //Atualiza velocidade dos nos
198     for (uint16_t i = 0; i < nWifi; i++)
199     {
200       AtualizaVelocidade(wifiStaNodes.Get(i));
201     }
202   }
203
204 //Pilha de internet
205   InternetStackHelper stack;
206   stack.Install (csmaNodes);
207   stack.Install (wifiStaNodes);
208
209   Ipv4AddressHelper address;
210
211   //Cria endereco para CSMA
212   address.SetBase ("10.1.2.0", "255.255.255.0");
213   Ipv4InterfaceContainer csmaInterfaces = address.Assign (
        csmaDevices);
214
215   //Cria endereco para os Clientes
216   address.SetBase ("192.168.0.0", "255.255.255.0");
217   Ipv4InterfaceContainer interfacesAp = address.Assign (
        apDevices);
218   Ipv4InterfaceContainer interfacesWifi = address.Assign (
        staDevices);
219
220   if(trafego == 0)
221   {
222     //Aplicacao UDP
223     uint16_t port = 9;
224     for (uint32_t i = 0; i < wifiStaNodes.GetN(); i++)
225     {
226       uint16_t m_port = port + i;
227
228       OnOffHelper onoff ("ns3::UdpSocketFactory", Address(
```

```
            InetSocketAddress(csmaInterfaces.GetAddress(1), m_port)
            ));
229         onoff.SetAttribute ("Remote",  AddressValue(
            InetSocketAddress(interfacesWifi.GetAddress(i), m_port)
            ));
230         onoff.SetAttribute ("OnTime", StringValue ("ns3::
            ConstantRandomVariable[Constant=1]"));
231         onoff.SetAttribute ("OffTime", StringValue ("ns3::
            ConstantRandomVariable[Constant=0]"));
232         onoff.SetAttribute ("DataRate", DataRateValue ( DataRate (
            "512kbps")));
233         onoff.SetAttribute ("PacketSize", UintegerValue (478)); //
             +34 header
234
235         ApplicationContainer server = onoff.Install(csmaNodes.Get
            (1));
236         server.Start(Seconds (1.0));
237         server.Stop(Seconds (simTime));
238
239         PacketSinkHelper sink ("ns3::UdpSocketFactory",
            InetSocketAddress(interfacesWifi.GetAddress(i), m_port)
            );
240
241         ApplicationContainer client = sink.Install(wifiStaNodes.
            Get(i));
242         client.Start(Seconds (2.0));
243         client.Stop(Seconds (simTime));
244       }
245     }
246   else if(trafego == 1)
247   {
248     //Aplicacao TCP
249     uint16_t port = 8080;
250     for (uint32_t i = 0; i < wifiStaNodes.GetN(); i++)
251     {
252       uint16_t m_port = port + i;
253
254       OnOffHelper onoff ("ns3::TcpSocketFactory", Address(
            InetSocketAddress(csmaInterfaces.GetAddress(1), m_port)
            ));
255       onoff.SetAttribute ("Remote",  AddressValue(
            InetSocketAddress(interfacesWifi.GetAddress(i), m_port)
            ));
256       onoff.SetAttribute ("OnTime", StringValue("ns3::
            NormalRandomVariable[Mean=5.|Variance=1.|Bound=10.]"));
257       onoff.SetAttribute ("OffTime", StringValue("ns3::
            NormalRandomVariable[Mean=7.|Variance=1.|Bound=10.]"));
258       onoff.SetAttribute ("DataRate", DataRateValue ( DataRate (
            "512kbps")));
```

```
259         onoff.SetAttribute ("PacketSize", UintegerValue (1440));
               // +60 header
260
261         ApplicationContainer server = onoff.Install(csmaNodes.Get
               (1));
262         server.Start (Seconds (1.0));
263         server.Stop (Seconds (simTime));
264
265         PacketSinkHelper sink ("ns3::TcpSocketFactory",
               InetSocketAddress(interfacesWifi.GetAddress(i), m_port)
               );
266
267         ApplicationContainer client = sink.Install(wifiStaNodes.
               Get(i));
268         client.Start(Seconds (2.0));
269         client.Stop(Seconds (simTime));
270
271       }
272
273     }
274     else if(trafego == 2)
275     {
276           //Aplicacao UDP e TCP
277           for (uint32_t i = 0; i < wifiStaNodes.GetN(); i++)
278           {
279                   uint32_t udp_port = 9;
280                   uint32_t tcp_port = 8080;
281
282                   if(i \% 2 == 0)
283                   {
284                           //Aplicacao UDP
285                           OnOffHelper onoff ("ns3::
                              UdpSocketFactory", Address(
                              InetSocketAddress(csmaInterfaces.
                              GetAddress(1), udp_port+i)));
286                           onoff.SetAttribute ("Remote",
                              AddressValue(InetSocketAddress(
                              interfacesWifi.GetAddress(i),
                              udp_port+i)));
287                           onoff.SetAttribute ("OnTime",
                              StringValue ("ns3::
                              ConstantRandomVariable[Constant=1]"))
                              ;
288                           onoff.SetAttribute ("OffTime",
                              StringValue ("ns3::
                              ConstantRandomVariable[Constant=0]"))
                              ;
289                           onoff.SetAttribute ("DataRate",
                              DataRateValue ( DataRate ("512kbps"))
```

```
                    );
290                 onoff.SetAttribute ("PacketSize",
                    UintegerValue (478)); // +34 header
291
292                 ApplicationContainer server = onoff.
                    Install(csmaNodes.Get(1));
293                 server.Start(Seconds (1.0));
294                 server.Stop(Seconds (simTime));
295
296                 PacketSinkHelper sink ("ns3::
                    UdpSocketFactory", InetSocketAddress(
                    interfacesWifi.GetAddress(i),
                    udp_port+i));
297
298                 ApplicationContainer client = sink.
                    Install(wifiStaNodes.Get(i));
299                 client.Start(Seconds (2.0));
300                 client.Stop(Seconds (simTime));
301         }
302         else
303         {
304                 //Aplicacao TCP
305                 OnOffHelper onoff ("ns3::
                    TcpSocketFactory", Address(
                    InetSocketAddress(csmaInterfaces.
                    GetAddress(1), tcp_port+i)));
306                 onoff.SetAttribute ("Remote",
                    AddressValue(InetSocketAddress(
                    interfacesWifi.GetAddress(i),
                    tcp_port+i)));
307                 onoff.SetAttribute ("OnTime",
                    StringValue("ns3::
                    NormalRandomVariable[Mean=5.|Variance
                    =1.|Bound=10.]"));
308                 onoff.SetAttribute ("OffTime",
                    StringValue("ns3::
                    NormalRandomVariable[Mean=7.|Variance
                    =1.|Bound=10.]"));
309                 onoff.SetAttribute ("DataRate",
                    DataRateValue ( DataRate ("512kbps"))
                    );
310                 onoff.SetAttribute ("PacketSize",
                    UintegerValue (1440)); // +60 header
311
312                 ApplicationContainer server = onoff.
                    Install(csmaNodes.Get(1));
313                 server.Start (Seconds (1.0));
314                 server.Stop (Seconds (simTime));
315
```

```
316                             PacketSinkHelper sink ("ns3::
                                   TcpSocketFactory", InetSocketAddress(
                                   interfacesWifi.GetAddress(i),
                                   tcp_port+i));
317
318                             ApplicationContainer client = sink.
                                   Install(wifiStaNodes.Get(i));
319                             client.Start(Seconds (2.0));
320                             client.Stop(Seconds (simTime));
321                     }
322             }
323     }
324
325     Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
326
327 //Monitor de fluxo
328     Ptr<FlowMonitor> monitor;
329     FlowMonitorHelper fmhelper;
330     monitor = fmhelper.InstallAll();
331
332     Simulator::Stop (Seconds (simTime));
333     Simulator::Run ();
334
335 //Imprime metricas n
336     if (verbose)
337     {
338       ImprimeMetricas (&fmhelper, monitor, trafego, nWifi);
339     }
340
341     Simulator::Destroy ();
342     return 0;
343 }
```

Script "run_gerencia2018.sh" para executar as simulações de forma automatizada

Para execução dos diferentes tráfegos com as 33 *seeds*, basta rodar o *script* abaixo passando os devidos parâmetros, como por exemplo `./run_gerencia.sh 33 6 0 0` que resulta em 33 simulações ($seed = 33$) no cenário com 6 clientes ($nWifi = 6$) utilizando tráfego CBR ($trafego = 0$) e sem mobilidade ($mobilidade = 0$).

Os parâmetros passados no *script* são diretamente adicionados à chamada de simulação do NS3, da seguinte forma: `./waf --run "scratch/gerencia2018 --seed=$i --nWifi=$2 --trafego=$3 --mobilidade=$4"`. Onde `$1` é o primeiro parâmetro passado, `$2` o segundo parâmetro e assim sucessivamente.

```
1 #!/bin/bash
2 #Autores: Joahannes B. D. da Costa e Wellington V. Lobato Junior
3 #Data: 23.09.2018
4 #Nota: script para rodar simulacao de MO655 - Gerencia de Redes
     - 2018
```

```
5  #Rodar: ./run_gerencia.sh numero_seeds numero_clientes trafego
       mobilidade
6
7  seeds=$1
8  #$2 = clientes
9  #$3 = trafego
10
11 #Vai para diretorio ns-3.29
12 cd ..
13
14 echo
15 echo " * MO655 - Gerencia de Redes - 2018"
16 echo " * Numero de seeds: $1"
17 echo " * Numero de clientes: $2"
18 if [ $4 == 0 ]
19 then
20         echo " * Mobilidade: Ausente"
21 else
22         echo " * Mobilidade: Presente"
23 fi
24 if [ $3 == 0 ]
25 then
26         echo " * Trafego: UDP"
27 elif [ $3 == 1 ]
28 then
29         echo " * Trafego: TCP"
30 else
31         echo " * Trafego: 50% UDP e 50%TCP"
32 fi
33
34 for ((i=0; i<$seeds; i++))
35 do
36         if [ $4 == 0 ]
37         then
38                 #Controle para mensagens
39                 if [ $3 == 0 ]
40                 then
41                         echo "Executando seed [$i] com $2
                               clientes ESTATICOS e trafego UDP..."
42                         #Chamada da simulacao
43                         ./waf --run "scratch/gerencia2018 --seed
                               =$i --nWifi=$2 --trafego=$3 --
                               mobilidade=$4" > resultados/UDP/
                               results-$2-est-UDP-$i.txt 2>&1
44                 elif [ $3 == 1 ]
45                 then
46                         echo "Executando seed [$i] com $2
                               clientes ESTATICOS e trafego TCP..."
47                         #Chamada da simulacao
```

```
48                          ./waf --run "scratch/gerencia2018 --seed
                               =$i --nWifi=$2 --trafego=$3 --
                               mobilidade=$4" > resultados/TCP/
                               results-$2-est-TCP-$i.txt 2>&1
49                  else
50                          echo "Executando seed [$i] com $2
                               clientes ESTATICOS e trafegos UDP/TCP
                               ..."
51                          #Chamada da simulacao
52                          ./waf --run "scratch/gerencia2018 --seed
                               =$i --nWifi=$2 --trafego=$3 --
                               mobilidade=$4" > resultados/UDP-TCP/
                               results-$2-est-AMBOS-$i.txt 2>&1
53                  fi
54          else
55                  #Controle para mensagens
56                  if [ $3 == 0 ]
57                  then
58                          echo "Executando seed [$i] com $2
                               clientes MOVEIS e trafego UDP..."
59                          #Chamada da simulacao
60                          ./waf --run "scratch/gerencia2018 --seed
                               =$i --nWifi=$2 --trafego=$3 --
                               mobilidade=$4" > resultados/UDP/
                               results-$2-mov-UDP-$i.txt 2>&1
61                  elif [ $3 == 1 ]
62                  then
63                          echo "Executando seed [$i] com $2
                               clientes MOVEIS e trafego TCP..."
64                          #Chamada da simulacao
65                          ./waf --run "scratch/gerencia2018 --seed
                               =$i --nWifi=$2 --trafego=$3 --
                               mobilidade=$4" > resultados/TCP/
                               results-$2-mov-TCP-$i.txt 2>&1
66                  else
67                          echo "Executando seed [$i] com $2
                               clientes MOVEIS e trafegos UDP/TCP...
                               "
68                          #Chamada da simulacao
69                          ./waf --run "scratch/gerencia2018 --seed
                               =$i --nWifi=$2 --trafego=$3 --
                               mobilidade=$4" > resultados/UDP-TCP/
                               results-$2-mov-AMBOS-$i.txt 2>&1
70                  fi
71          fi
72  done
73
74  echo "------------------------------------------------------"
75
```

```
#EOF
```