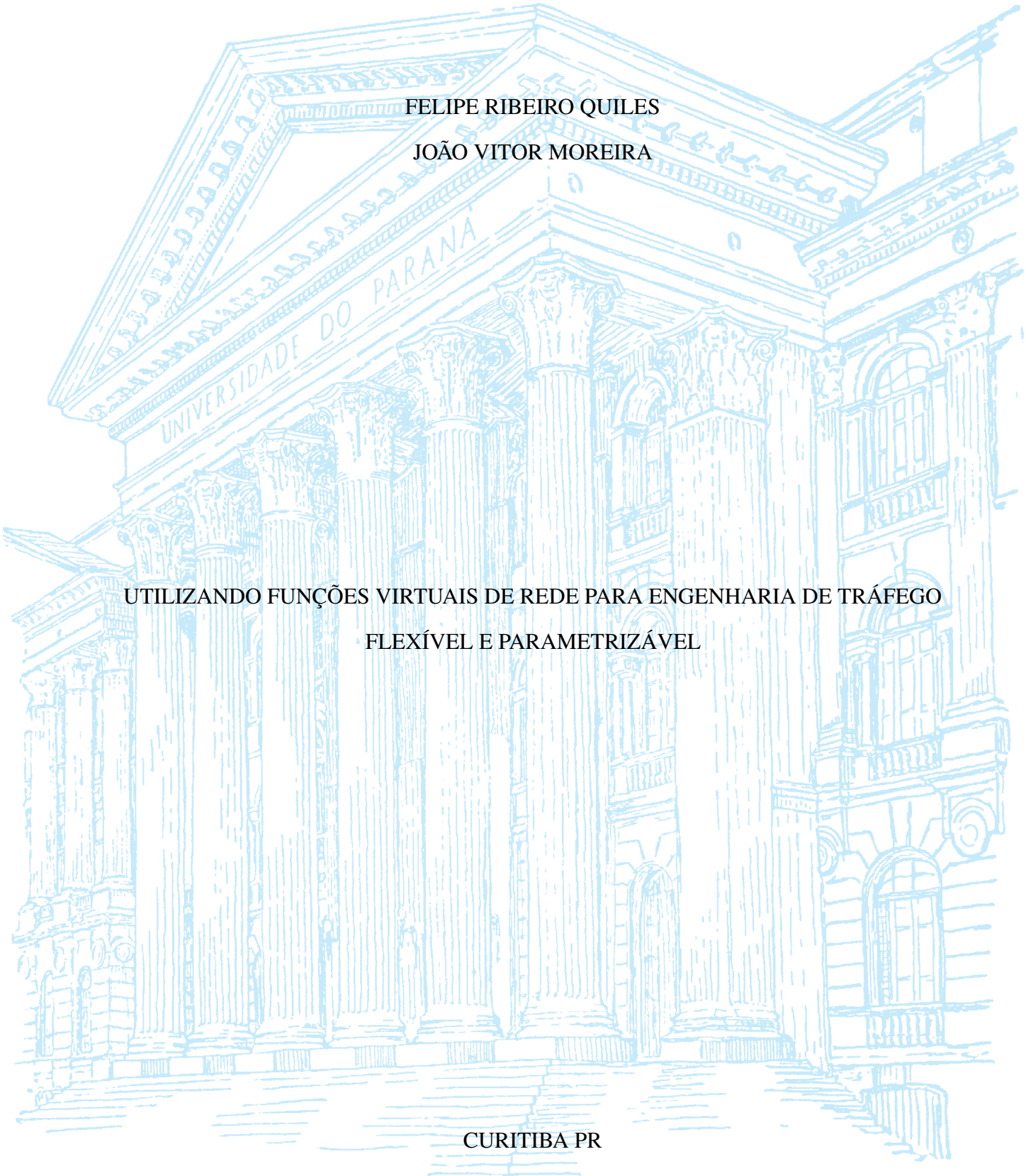
UNIVERSIDADE FEDERAL DO PARANÁ

FELIPE RIBEIRO QUILES

JOÃO VITOR MOREIRA



UTILIZANDO FUNÇÕES VIRTUAIS DE REDE PARA ENGENHARIA DE TRÁFEGO FLEXÍVEL E PARAMETRIZÁVEL

CURITIBA PR

2021

FELIPE RIBEIRO QUILES
JOÃO VITOR MOREIRA

UTILIZANDO FUNÇÕES VIRTUAIS DE REDE PARA ENGENHARIA DE TRÁFEGO FLEXÍVEL E PARAMETRIZÁVEL

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Elias Procópio Duarte Junior.

Coorientador: Vinícius Fülber Garcia.

CURITIBA PR

2021

RESUMO

O extraordinário avanço da Internet levou a um grande aumento no tráfego de dados pela rede. A engenharia de tráfego consiste em um conjunto de procedimentos relacionados ao monitoramento, caracterização, modelagem e controle do tráfego na rede. O objetivo da engenharia de tráfego é aprimorar o desempenho da rede, inclusive permitindo que sejam garantidos diferentes níveis de Qualidade de Serviço (*Quality of Service* - QoS). Os mecanismos de engenharia de tráfego podem atuar, por exemplo, classificando pacotes em diferentes categorias e/ou definindo taxas mínimas e máximas (*minimum and maximum rate enforcement*), evitando a sobrecarga da rede. Dois destes mecanismos são os *policers* e *shapers*, os quais atuam na suavização do tráfego da rede atrasando e descartando pacotes quando necessário. Deste modo, o seu uso contribui como uma forma de mitigar a dificuldade de se manter o desempenho das redes em níveis desejados devido à sua elevada e contínua demanda. Este trabalho propõe utilizar a tecnologia de Virtualização de Funções de Rede (*Network Function Virtualization* - NFV) para implementar um *framework* de engenharia de tráfego, o NFV-TE (NFV-*Traffic Engineering*). O paradigma NFV visa desacoplar o *software* do *hardware* no núcleo das redes. Para isso, Funções de Rede (*Network Functions* - NF), implementadas tradicionalmente em *hardware* dedicado, passam a ser disponibilizadas em um plano de *software* com o auxílio de tecnologias de virtualização. Assim, a implantação das NFs ocorre em dispositivos virtuais e passam a ser denominadas Funções Virtualizadas de Rede (*Virtualized Network Function* - VNF). Desta forma, as NFs podem ser executadas a partir de servidores de prateleira, não sendo necessária a aquisição de *hardware* especializado, flexibilizando a implementação, desenvolvimento e manutenção das redes de computadores. Já o NFV-TE implementa mecanismos de *policing* e *shaping* em um sistema parametrizável através de um arquivo JSON (*JavaScript Object Notation*) contendo a configuração desejada para o mecanismo escolhido. Além disso, as funções de rede geradas pelo NFV-TE dispõem da flexibilidade do paradigma NFV ao possibilitar que mecanismos de engenharia de tráfego sejam executados como VNFs. O NFV-TE foi testado gerando funções de engenharia de tráfego com diferentes parametrizações para cada mecanismo implementado pelo *framework*. Os mecanismos de *policing* implementados são: o *Single Rate Token Bucket Policer* (srTBP), o *Single Rate Three Color Marker Policer* (srTCM-*Policer*), o *Two Rate Three Color Marker Policer* (trTCM-*Policer*) e os *Color-Aware Policers*. Já os mecanismos de *shaping* implementados são: o *Single Rate Token Bucket Shaper* e o *Leaky Bucket*. Essas funções foram testadas em cenários com diferentes perfis de tráfego de rede. Assim, foi possível verificar, analisar e discutir o comportamento dos mecanismos. Os resultados obtidos demonstraram que todos os mecanismos realizam a limitação do tráfego de rede conforme os valores definidos através de seus parâmetros.

Palavras-chave: Rede. Engenharia de Tráfego. Modelagem. Controle. Virtualização. Funções Virtualizadas de Rede. Gerenciamento de Rede Baseado em Políticas.

ABSTRACT

The exceptional growth of Internet traffic is a formidable challenge for network management. Traffic engineering consists of a set of procedures related to the monitoring, characterization, modeling and control of network traffic. The goal of traffic engineering is to improve network performance, including providing different levels of Quality of Service (QoS). Traffic engineering mechanisms usually classify packets into different categories and/or enforce minimum and maximum rates, avoiding overloading the network. Two of those mechanisms are policers and shapers, which have the purpose to smooth network traffic by delaying and dropping packets when necessary. In that way, the mechanisms contribute to mitigate the difficulty of maintaining network performance at desired levels even under high and continuous demand. This work proposes the use of the Network Function Virtualization (NFV) to implement a traffic engineering framework, called NVF-Traffic Engineering (NFV-TE). The NFV paradigm aims to decouple software from hardware in the network core. Network Functions (NF), which are traditionally implemented as dedicated hardware, are made available in as Virtualized Network Functions (VNF) which are built and executed with virtualization technologies on off-the-shelf servers. Dropping the need for specialized hardware improves the flexibility to set up and operate the network. The NFT-TE framework implements policers and shapers that are configured using a JSON (JavaScript Object Notation) file which allow the specification of the desired settings for each traffic engineering mechanism. Furthermore, the network functions generated by the NFV-TE take advantage of the flexibility provided by the NFV paradigm by enabling traffic engineering mechanisms to be executed as VNFs. The NFV-TE framework was evaluated by generating traffic engineering functions with different parameters for each mechanism implemented. The policing mechanisms currently implemented by the framework are: the Single Rate Token Bucket Policer (srTBP), the Single Rate Three Color Marker Policer (srTCM-Policer), the Two Rate Three Color Marker Policer (trTCM-Policer) and the Color-Aware Policers. The shaping mechanisms implemented are: the Single Rate Token Bucket Shaper and the Leaky Bucket. All NFV-TE network functions were evaluated and discussed under different network traffic scenarios. Results obtained confirm that the mechanisms act on the network traffic according to the configuration parameters.

Keywords: Network. Traffic Engineering. Shaping. Policing. Virtualization. Network Function Virtualization. Policy-Based Network Management.

LISTA DE FIGURAS

2.1 Fluxograma *Single Rate Token Bucket Policer*. 12
2.2 Fluxograma srTCM-*Policer*. 13
2.3 Fluxograma trTCM-*Policer*. 14
2.4 Fluxograma srTCM-*Policer Color-Aware*. 15
2.5 Fluxograma trTCM-*Policer Color-Aware*. 16
2.6 Fluxograma *Single Rate Token Bucket Shaper*. 17
2.7 Fluxograma *Leaky Bucket*. 18

3.1 Comparação da estrutura entre as redes tradicionais e redes baseadas no paradigma NFV.. 20
3.2 Arquitetura de referência NFV proposto pela ETSI. 21
3.3 Módulos e conexões do bloco NFV-MANO. 22

5.1 Representação do Ambiente de Experimentos 27
5.2 Informações sobre os tipos de tráfego. 28
5.3 Transmissões realizadas pelo srTBP alternando a rajada e *rate* igual a 1500 para os diferentes tipos de tráfego. 29
5.4 Transmissões realizadas pelo srTBP alternando a rajada e *rate* igual a 2700 para os diferentes tipos de tráfego. 30
5.5 Transmissões realizadas pelo srTBP alternando a rajada e *rate* igual a 3900 para os diferentes tipos de tráfego. 31
5.6 Transmissões realizadas pelo srTBP para o tráfego de 10Mbps. 32
5.7 Transmissões realizadas pelo srTCM-*Policer* alterando o EBS e com CIR e CBS igual a 1500 e 4000, respectivamente, para os diferentes tipos de tráfego.. 33
5.8 Gráficos indicando a quantidade de ações verdes, amarelas e vermelhas tomadas pelo srTCM-*Policer* para os diferentes tráfegos. 34
5.9 Transmissões realizadas pelo srTCM-*Policer* alterando o EBS e com CIR e CBS igual a 2700 e 2900, respectivamente, para os diferentes tipos de tráfego.. 34
5.10 Gráficos indicando a quantidade de ações verdes, amarelas e vermelhas tomadas pelo srTCM-*Policer* para os diferentes tráfegos. 35
5.11 Transmissões realizadas pelo srTCM-*Policer* alterando o EBS e com CIR e CBS igual a 3900 e 7000, respectivamente, para os diferentes tipos de tráfego.. 35
5.12 Gráficos indicando a quantidade de ações verdes, amarelas e vermelhas tomadas pelo srTCM-*Policer* para os diferentes tráfegos. 36
5.13 Transmissões realizadas pelo trTCM-*Policer* alterando o PBS e CIR, PIR e CBS igual a 1500, 3400 e 2900, respectivamente, para os diferentes tipos de tráfego. . 37

5.14 Gráficos indicando a quantidade de ações verdes, amarelas e vermelhas tomadas pelo trTCM-*Policer* para os diferentes tráfegos. 37
5.15 Transmissões realizadas pelo trTCM-*Policer* alterando o PBS e CIR, PIR e CBS igual a 3900, 1200 e 7000, respectivamente, para os diferentes tipos de tráfego. . 38
5.16 Gráficos indicando a quantidade de ações verdes, amarelas e vermelhas tomadas pelo trTCM-*Policer* para os diferentes tráfegos. 39
5.17 Transmissões realizadas pelo trTCM-*Policer* alterando o PBS e CIR, PIR e CBS igual a 2700, 2300 e 4000, respectivamente, para os diferentes tipos de tráfego. . 39
5.18 Gráficos indicando a quantidade de ações verdes, amarelas e vermelhas tomadas pelo trTCM-*Policer* para os diferentes tráfegos. 40
5.19 Transmissões realizadas pelo srTCM-PCA alterando o EBS e com CIR e CBS iguais a 2700 e 2900, respectivamente, para os diferentes tipos de tráfego. 41
5.20 Comparação entre as cores recebidas e as cores das ações realizadas para o srTCM-PCA, tendo o valor de EBS alterado, e CIR e CBS iguais a 2700 e 2900, respectivamente. 42
5.21 Transmissões realizadas pelo srTCM-PCA alterando o EBS e com CIR e CBS iguais a 1500 e 4000, respectivamente, para os diferentes tipos de tráfego. 42
5.22 Quantidade de mudanças realizadas pelo srTCM-PCA entre a cor pré-definida e a ação realizada.. 43
5.23 Comparação entre as cores recebidas e as cores das ações realizadas para o srTCM-PCA, tendo o valor de EBS alterado, e CIR e CBS iguais a 1500 e 4000, respectivamente. 44
5.24 Transmissões realizadas pelo trTCM-PCA alterando o PBS e com CIR, PIR e CBS iguais a 1500, 3400 e 7000, respectivamente, para os diferentes tipos de tráfego. 44
5.25 Comparação entre as cores recebidas e as cores das ações realizadas para o trTCM-PCA, tendo o valor de PBS alterado, e CIR, PIR e CBS iguais a 1500, 3400 e 7000, respectivamente. 45
5.26 Comparação entre as cores recebidas e as cores das ações realizadas para o trTCM-PCA, tendo o valor de PBS alterado, e CIR, PIR e CBS iguais a 2700, 2300 e 7000, respectivamente. 46
5.27 Transmissões realizadas pelo srTBS alternando a rajada e *rate* igual a 1500 para os diferentes tipos de tráfego. 48
5.28 Tempo de espera médio dos pacotes da fila na execução do srTBS, alterando a rajada e taxa de reposição de *tokens* ao *bucket* igual a 1500. 49
5.29 Transmissões realizadas pelo srTBS alternando a rajada e *rate* igual a 2700 para os diferentes tipos de tráfego. 49
5.30 Tempo de espera médio dos pacotes da fila na execução do srTBS, alterando a rajada e taxa de reposição de *tokens* ao *bucket* igual a 2700. 50
5.31 Transmissões realizadas pelo LB alternando a taxa constante para os diferentes tipos de tráfego. 51
5.32 Transmissões realizadas pelo LB para o tráfego de 10Mbps.. 52

LISTA DE ACRÔNIMOS

QoS	*Quality of Service*
srTBP	*Single Rate Token Bucket Policer*
srTCM	*Single Rate Three Color Marker*
CIR	*Committed Information Rate*
CBS	*Committed Burst Size*
EBS	*Excess Burst Size*
trTCM	*Two Rate Three Color Marker*
PIR	*Peak Information Rate*
PBS	*Peak Burst Size*
srTBS	*Single Rate Token Bucket Shaper*
LB	*Leaky Bucket*
JSON	*JavaScript Object Notation*
NF	*Network Function*
NFV	*Network Function Virtualization*
VNF	*Virtualized Network Function*
ETSI	*European Telecommunications Standards Institute*
NAT	*Network Address Translation*
OPEX	*OPerational EXpenditures*
CAPEX	*CAPital EXpenditures*
DPI	*Deep Packet Inpection*
VPN	*Virtual Private Network*
NFVI	*NFV Infrastructure*
NFV-MANO	*NFV Management and Orchestration*
NFVO	NFV *Orchestrator*
VNFM	VNF *Manager*
VIM	*Virtualized Infrastructure Manager*
OSS/BSS	*Operations Support System/Business Support System*
VM	*Virtual Machines*
VNFD	VNF *Descriptor*
EM	*Element Manager*
FCAPS	*Fault, Configuration, Accounting, Performance and Security Management*
NFV-TE	*NFV-Traffic Engineering*
SDN	*Software-Defined Networking*
DDoS	*Distributed Denial of Service*

SUMÁRIO

1 INTRODUÇÃO 9
2 ENGENHARIA DE TRÁFEGO 11
2.1 *POLICERS* 11
2.1.1 *Single Rate Token Bucket Policer* 11
2.1.2 *Single Rate Three Color Marker Policer* 12
2.1.3 *Two Rate Three Color Marker Policer* 13
2.1.4 *Color-Aware Policers* 15
2.2 *SHAPERS* 16
2.2.1 *Single Rate Token Bucket Shaper* 16
2.2.2 *Leaky Bucket* 17
3 VIRTUALIZAÇÃO DE FUNÇÕES DE REDE 19
3.1 INTRODUÇÃO À VIRTUALIZAÇÃO DE FUNÇÕES DE REDE 19
3.2 ARQUITETURA DE REFERÊNCIA PARA NFV. 20
4 FUNÇÕES VIRTUAIS DE REDE PARA ENGENHARIA DE TRÁFEGO. . 23
4.1 TRABALHOS RELACIONADOS 23
4.2 *FRAMEWORK* NFV-TE 24
4.3 *POLICERS* 24
4.3.1 *Single Rate Token Bucket Policer* 24
4.3.2 *Single Rate Three Color Marker Policer* 25
4.3.3 *Two Rate Three Color Marker Policer* 25
4.3.4 *Color-Aware Policers* 25
4.4 *SHAPERS* 26
4.4.1 *Single Rate Token Bucket Shaper* 26
4.4.2 *Leaky Bucket* 26
5 RESULTADOS EXPERIMENTAIS 27
5.1 AMBIENTE DOS EXPERIMENTOS 27
5.2 *POLICERS* 28
5.2.1 *Single Rate Token Bucket Policer* 29
5.2.2 *Single Rate Three Color Marker Policer* 32
5.2.3 *Two Rate Three Color Marker Policer* 35
5.2.4 *Color-Aware Policers* 40
5.3 *SHAPERS* 47
5.3.1 *Single Rate Token Bucket Shaper* 47
5.3.2 *Leaky Bucket* 50

6 CONCLUSÃO 53
REFERÊNCIAS 54

1 INTRODUÇÃO

A engenharia de tráfego representa uma estratégia concreta para o provisionamento de Qualidade de Serviço (QoS - *Quality of Service*) em redes de computadores, principalmente em tempos de alta demanda e abundância de dados circulando (Nucci e Papagiannaki, 2009). O papel principal da engenharia de tráfego é aperfeiçoar o tráfego de dados na rede e manter a utilização de recursos computacionais em níveis adequados. Com a aplicação de mecanismos de engenharia de tráfego, é possível projetar redes mais robustas e resilientes, estas capazes de se adaptar e suportar situações críticas que, quando não gerenciadas apropriadamente, podem causar instabilidades e interrupções nos serviços prestados.

Dentre os principais mecanismos existentes de engenharia de tráfego estão os *policers* e os *shapers* (Evans e Filsfils, 2010). *Policers* têm o propósito de eliminar os picos no fluxo de dados, limitando-os a uma taxa máxima definida. Por outro lado, os *shapers* suavizam o perfil do tráfego de rede, atrasando os pacotes ao utilizar filas consumidas de acordo com taxas estabelecidas. Dessa forma, objetiva-se que o fluxo de dados não ultrapasse um limite definido.

Funções de Rede (*Network Functions* - NF) são comumente utilizadas para implementar mecanismos de engenharia de tráfego. Em particular, funções de rede são elementos que atuam no núcleo da rede realizando tarefas específicas, sendo historicamente implementadas em dispositivos físicos dedicados à execução das mesmas. Devido a isso, há um alto nível de acoplamento entre *hardware* e *software* (Sekar et al., 2012), fato este que normalmente implica em dificuldades de manutenção e escalabilidade das NFs. Desse modo, é justificada a análise de outras formas de se implementar NFs em uma rede.

A utilização de tecnologias de virtualização torna possível instanciar múltiplas máquinas virtuais (*software*) em um único *hardware* de propósito geral. Aplicando este conceito às Funções de Rede, é possível flexibilizar o desenvolvimento e gerenciamento de NFs (Cui et al., 2012). Assim surge o paradigma de Virtualização de Funções de Rede (*Network Function Virtualization* - NFV), em que as funções de rede são implementadas em um plano de *software* desacoplado do *hardware* físico que as executará. Com isso, considera-se que dispositivos virtuais serão empregados para executar tais funções, utilizando para isso as tecnologias de virtualização existentes. Destaca-se que instâncias de execução de NFs, no paradigma NFV, passam a ser denominadas Funções Virtualizadas de Rede (*Virtualized Network Functions* - VNF).

Diante do cenário apresentado, este trabalho apresenta o NFV-TE (NFV-*Traffic Engineering*), um *framework* parametrizável para geração de funções de rede de engenharia de tráfego. Destaca-se que as funções de rede provenientes do NFV-TE são desenvolvidas para o paradigma NFV, isto é, para serem executadas por tecnologias de virtualização disponibilizadas em *hardware* de propósito geral. A configuração do *framework* se dá pela especificação de um arquivo JSON (*JavaScript Object Notation*) contendo os parâmetros a serem utilizados na geração de uma função de rede, conforme os mecanismos disponíveis de engenharia de tráfego: *policers* e *shapers*.

O *framework* foi construído visando demonstrar o impacto da engenharia de tráfego em uma rede de computadores, bem como a viabilidade de sua implementação através de VNFs. Para isso foram realizados experimentos com as VNFs geradas pelo NFV-TE utilizando diferentes parâmetros e múltiplos perfis de tráfego para análise e avaliação dos resultados obtidos na rede. Nesse contexto, os mecanismos de *policing* considerados são: *Single Rate Token Bucket Policer* (srTBP), *Single Rate Three Color Marker Policer* (srTCM-*Policer*), *Two Rate Three*

Color Marker Policer (trTCM-*Policer*) e *Color-Aware Policers*. Já os mecanismos de *shaping* considerados são: *Single Rate Token Bucket Shaper* e *Leaky Bucket*.

Os experimentos foram executados em um ambiente controlado, com a utilização da tecnologia de virtualização de máquinas virtuais (*Virtual Machines* - VM). Três VMs foram instanciadas, todas executando o sistema Ubuntu 18.4. A primeira VM representa o cliente, a segunda representa um servidor e a terceira assume o papel de uma VNF de engenharia de tráfego gerada pelo NFV-TE. Assim, as interfaces de rede foram conectadas entre cliente e VNF e entre VNF e servidor de forma que, para se comunicar com o servidor, o cliente deve transmitir sua mensagem para a VNF, que, por sua vez, irá processar o pacote tomando a ação de encaminhá-lo ao servidor ou então descartá-lo, a depender da situação do tráfego de dados na rede em um dado momento. As VNFs foram testadas utilizando quatro perfis diferentes de tráfego: Alfa (Sarvotham et al., 2001), Beta (Sarvotham et al., 2001), Elefante (Hamdan et al., 2020) e Guepardo (Maji et al., 2017). Os resultados obtidos atestam o funcionamento correto das funções de rede geradas pelo NFV-TE, sendo apresentados e discutidos no decorrer deste trabalho.

O restante deste trabalho está organizado da seguinte forma. O Capítulo 2 apresenta uma visão geral sobre engenharia de tráfego e descreve os *shapers* e *policers* considerados neste trabalho. O Capítulo 3 introduz o paradigma NFV, apresentando também a arquitetura de referência proposta pela *European Telecommunications Standards Institute* (ETSI). O Capítulo 4 discorre sobre como o paradigma NFV pode ser utilizado para a implementação de engenharia de tráfego, junto aos trabalhos relacionados a este tópico; então descreve-se a implementação e funcionamento do NFV-TE. O Capítulo 5 detalha os experimentos realizados com as NFs geradas com o NFV-TE, apresentando a análise de cada resultado obtido. Por fim, a conclusão é apresentada no Capítulo 6, junto às propostas de trabalhos futuros.

2 ENGENHARIA DE TRÁFEGO

O aumento na oferta de serviços de dados e a crescente popularização da Internet desencadearam diversos desafios no núcleo das redes. Com o aumento extraordinário da demanda, inevitavelmente, manter o desempenho da rede sempre em níveis aceitáveis não é uma tarefa trivial (Nucci e Papagiannaki, 2009).

No cenário descrito, a engenharia de tráfego se torna essencial para a manutenção e monitoramento das redes de computadores, buscando assegurar boas marcas de Qualidade de Serviço (*Quality of Service* – QoS). Particularmente, a engenharia de tráfego pode ser compreendida como a área responsável por manipular e aperfeiçoar os níveis de utilização das redes. Nesse sentido, a definição adequada de estratégias de tratamento de tráfego é fundamental para alcançar um desempenho satisfatório e uma boa destinação de recursos computacionais nas redes de computadores.

Existem diversos mecanismos e maneiras de se empregar a engenharia de tráfego em uma rede de computadores. Entretanto, o presente trabalho foca em dois mecanismos específicos: *policers* e *shapers*. Em geral, *policers* atuam para eliminar picos de tráfego. Já os *shapers* buscam suavizar o perfil de tráfego, descartando ou atrasando pacotes (Evans e Filsfils, 2010). Mecanismos de *policing* relevantes são abordados na Seção 2.1, já alguns dos mecanismos de *shaping* essenciais são discutidos na Seção 2.2.

2.1 *POLICERS*

O objetivo geral de um *policer* é assegurar que o tráfego passando pela rede não exceda uma taxa máxima definida (Evans e Filsfils, 2010). A seguir, quatro importantes *policers* são descritos: *Single Rate Token Bucket Policer*, *Single Rate Three Color Marker Policer*, *Two Rate Three Color Marker Policer* e *Color-Aware Policers*.

2.1.1 *Single Rate Token Bucket Policer*

A implementação mais simples de um *Single Rate Token Bucket Policer* (srTBP) utiliza *tokens* para processar um fluxo de pacotes. Especificamente, os *tokens* são uma representação simplificada dos *bytes* dos pacotes, sendo um *token* correspondente a um *byte*. Tais *tokens* são consumidos durante o processamento dos pacotes pelo mecanismo de *policing*. No srTBP, também é necessária a existência de um *bucket*. Figurativamente, um *bucket* é um "balde"onde são armazenados os *tokens* disponíveis. É importante ressaltar que todo o *bucket* apresenta uma capacidade máxima de armazenamento de *tokens*. Essa capacidade é chamada rajada (*burst*), parâmetro que determina a profundidade do *bucket*. Além disso, há também a definição da taxa de reposição de *tokens* no *bucket* (*rate*), que indica quantas unidades de *tokens* são adicionados ao *bucket* em um dado momento. A adição de *tokens* no *bucket* acontece no disparo de um gatilho, como pelo processamento de um pacote pelo mecanismo de *policing* ou pela passagem de intervalos regulares de tempo (Evans e Filsfils, 2010).

Ao se aplicar o srTBP em um fluxo de tráfego, assim que um pacote chega ao *policer*, o tamanho do mesmo (normalmente definido pela sua quantidade de *bytes*) é comparado à quantidade de *tokens* presentes no *bucket*. No caso da quantidade de *tokens* disponíveis ser maior ou igual ao tamanho do pacote, este último é aceito pelo srTBP sendo transmitido. Assim, os *tokens* presentes no *bucket* são decrementados na quantidade equivalente ao tamanho do

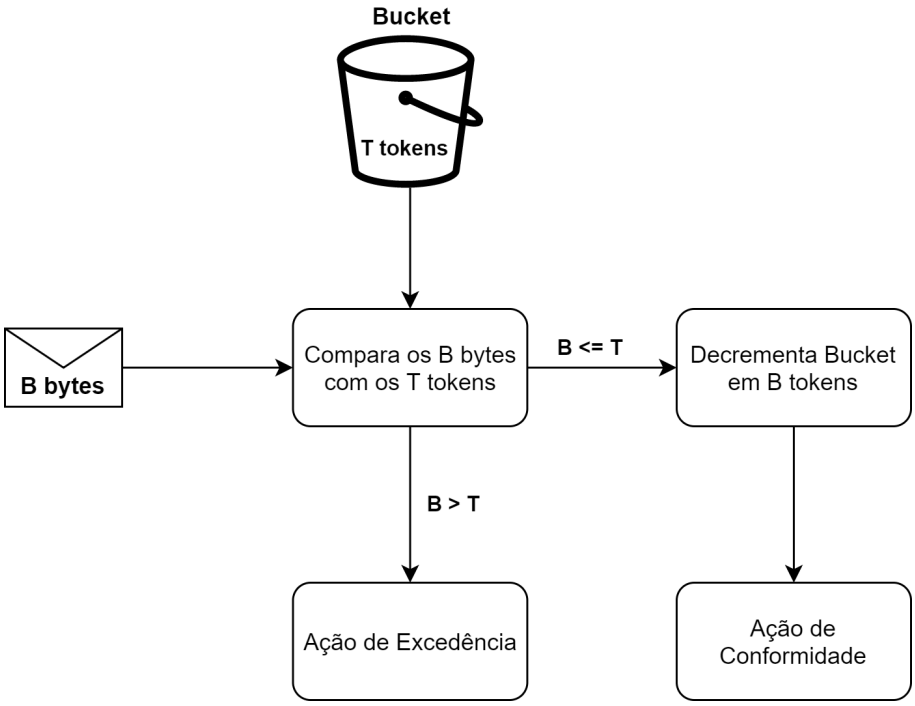


Figura 2.1: Fluxograma *Single Rate Token Bucket Policer*.

pacote. Caso contrário, o pacote é rejeitado, não sendo transmitido pelo srTBP. Além disso, nesse último caso, os *tokens* do *bucket* não são decrementados. Ressalta-se que diferentes ações podem ser efetuadas após a análise do pacote pelo srTBP. Porém, as ações tradicionais consistem na transmissão dos pacotes aceitos pelo srTBP e o descarte dos pacotes rejeitados. A Figura 2.1 exibe o fluxograma do comportamento do srTBP no processamento de um pacote de tamanho B, com o *bucket* tendo T *tokens* disponíveis no momento da chegada do pacote.

Finalmente, é importante destacar que o srTBP não atrasa o tráfego de rede, além de não reorganizar, nem priorizar determinados pacotes em detrimento de outros. Tais conclusões decorrem do fato deste mecanismo não armazenar pacotes. Sendo assim, à medida que os pacotes são recebidos, o srTBP processa os mesmos e executa imediatamente as ações atribuídas ao pacote.

2.1.2 *Single Rate Three Color Marker Policer*

O *Single Rate Three Color Marker* (srTCM) mede um fluxo de tráfego, realizando marcação dos pacotes nas cores verde, amarelo e vermelho. Os *markers* são mecanismos de engenharia de tráfego utilizados para classificação dos pacotes, realizando, normalmente, marcações de coloração verde, amarela e vermelha. Essas cores indicam se os pacotes estão ou não de acordo com definições estabelecidas na rede. O srTCM, apesar de ser particularmente definido por Heinanen e Guerin na RFC 2697 como um *marker*, é usado também em um mecanismo de *policer*, este chamado srTCM-*Policer*. Ao ser utilizado como *policer*, as três cores de marcação possíveis indicam ações específicas a serem realizadas em relação aos pacotes marcados (Evans e Filsfils, 2010).

A implementação do srTCM-*Policer* necessita de dois *token buckets*, denominados como Conforme (C) e Exceção (E), além dos próprios *tokens*. A cor de um pacote será estabelecida por meio da análise de seu tamanho em *bytes*, utilizando três parâmetros de tráfego, o *Committed Information Rate* (CIR), o *Committed Burst Size* (CBS) e o *Excess Burst Size* (EBS). Efetivamente,

o CBS e EBS, representam as quantidades máximas de *bytes* que podem ser recebidos para que os pacotes estejam em conformidade ou em exceção às definições dos *buckets* C e E, respectivamente. Por outro lado, o CIR é o parâmetro que representa a taxa de pacotes adequadas às definições do srTCM-*Policer*. Assim, o CIR define a quantidade de *tokens* que deve ser adicionada aos *buckets* a cada intervalo de tempo estipulado. Utilizando essas definições, tem-se a seguinte coloração: o pacote será marcado como verde, se não exceder o CBS, como amarelo, se excede o CBS, mas não o EBS, e como vermelho, se exceder ambos, CBS e EBS (Heinanen e Guerin, 1999a).

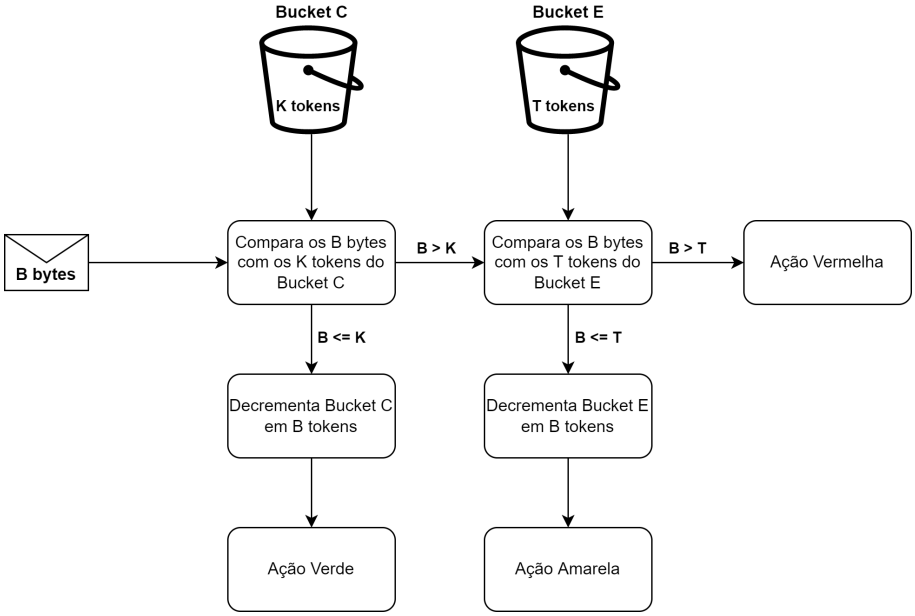


Figura 2.2: Fluxograma srTCM-*Policer*.

O comportamento do srTCM-*Policer* ao receber um pacote consiste em comparar o tamanho em *bytes* deste com a quantidade de *tokens* disponíveis, primeiramente, no *bucket* C. Se a quantidade de *tokens* é maior que o número de *bytes* do pacote, então o pacote está em conformidade com as definições do *bucket* C. Dessa forma, será tomada a ação verde e o *bucket* C decrementado pelo tamanho do pacote. Senão, o tamanho do pacote será comparado, também, aos *tokens* presentes no *bucket* E. Por conseguinte, sendo os *tokens* disponíveis suficientes, a ação amarela é tomada e o *bucket* E é decrementado considerando o tamanho do pacote, que está em conformidade com o mesmo. Se nenhum dos *buckets* sustenta uma quantidade de *tokens* maior ou igual aos *bytes* do pacote, este viola ambos os *buckets* e a ação vermelha será tomada. A Figura 2.2 exibe um fluxograma com o comportamento do srTCM-*Policer* ao realizar o processamento de um pacote de tamanho B, tendo na chegada do pacote, K *tokens* e T *tokens* disponíveis, respectivamente, nos *buckets* C e E.

Por fim, destaca-se que, caso o EBS seja igual a zero, existem efetivamente apenas duas ações a serem tomadas, a verde e a vermelha. Desta maneira, o comportamento do srTCM-*Policer* é equivalente ao do *srTBP*.

2.1.3 *Two Rate Three Color Marker Policer*

O *Two Rate Three Color Marker* (trTCM) é um mecanismo que quantifica o tráfego e realiza a coloração dos pacotes nas cores verde, amarela e vermelha. A implementação do trTCM necessita de dois *token buckets*, o *bucket* de Pico (P) e outro de Conformidade (C), além dos próprios *tokens*. A coloração dos pacotes é realizada segundo a definição de outros quatro parâmetros de tráfego: o *Peak Information Rate* (PIR) e sua associada, o *Peak Burst Size* (PBS),

além do *Committed Information Rate* (CIR) e sua associada, o *Committed Burst Size* (CBS). Concretamente, o PIR e o CIR são taxas que incrementam a quantidade de *tokens* dos *buckets* P e C, respectivamente, a cada intervalo de tempo. O PIR está relacionado à quantidade de *bytes* que se deseja suportar nos picos e o CIR relacionado aos *bytes* em conformidade com as definições de ambos os *buckets*. Já o PBS e CBS são a profundidade dos *buckets* P e C, respectivamente. Sendo assim, indicam a capacidade máxima de armazenamento de *tokens* nos *buckets*, também se relacionando com a quantidade de *bytes* que se quer suportar tanto no pico, como na taxa contratada (*commited*) de tráfego (Heinanen e Guerin, 1999b).

Assim como no caso do srTCM, o trTCM foi definido na RFC 2698 por Heinanen e Guerin como um mecanismo de *marking*. Porém, o mesmo também pode ser utilizado como mecanismo de *policing*, chamado trTCM-*Policer*. Ao ser utilizado como *policer*, ao invés de haver a coloração do pacote nas cores verde, amarelo e vermelho, são realizadas, diretamente, as ações correspondentes às cores (Evans e Filsfils, 2010).

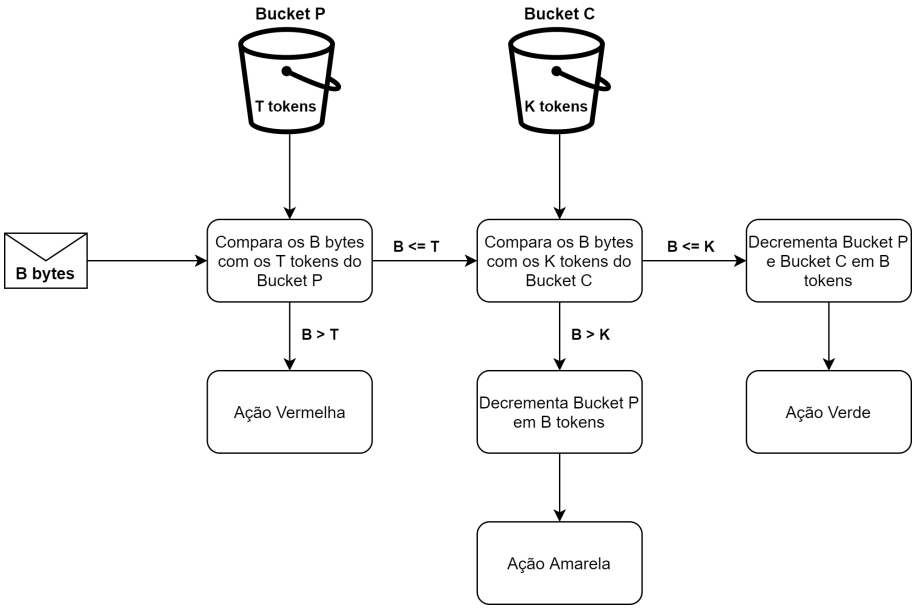


Figura 2.3: Fluxograma trTCM-*Policer*.

O comportamento do trTCM-*Policer* no recebimento de um pacote é da seguinte forma: o tamanho do pacote é comparado à quantidade de *tokens* disponíveis no *bucket* P. Se a quantidade de *tokens* é insuficiente, ou seja, menor se comparada ao número de *bytes* do pacote, a ação vermelha é tomada e nenhum dos *buckets* é decrementado. Se os *tokens* são suficientes, o tamanho do pacote é comparado à quantidade de *tokens* disponibilizados pelo *bucket* C. Se os *tokens* forem insuficientes, a ação amarela é realizada e o *bucket* P decrementado pelo tamanho do pacote. Entretanto, se também existirem *tokens* suficientes no *bucket* C, ambos os *buckets*, P e C, são decrementados pelo tamanho do pacote e a ação verde será realizada. A Figura 2.3, demonstra, por um fluxograma, o comportamento descrito sendo aplicado na chegada de um pacote de tamanho B, sendo K e T, a quantidade de *tokens* disponíveis nos *buckets* C e P, respectivamente.

Por fim, destaca-se que, se o PIR for igual ao CIR e o PBS igual ao CBS, apenas duas ações serão tomadas, a ação verde e a ação vermelha. Desta maneira, o comportamento do trTCM-*Policer* é equivalente ao comportamento do srTBP.

2.1.4 *Color-Aware Policers*

Os *Color-Aware Policers* (CAP) são tipicamente utilizados após um *marker*. Assim, o mecanismo de *policing* consiste em dois nodos. O primeiro nodo processa o fluxo de rede, marcando os pacotes e enviando os mesmos para o nodo seguinte. O segundo nodo é um *Color-Aware Policer* que verifica se o tráfego foi apropriadamente processado e marcado pelo nodo anterior (Evans e Filsfils, 2010).

Tanto o srTCM-*Policer*, quanto o trTCM-*Policer*, utilizando as mesmas definições, podem ser usados no contexto de *color-aware*, reconhecendo marcações de cores pré-definidas nos pacotes. Ressalta-se ainda que, na definição clássica, um pacote pode ser pré-colorido com uma de três cores: verde, amarela e vermelha.

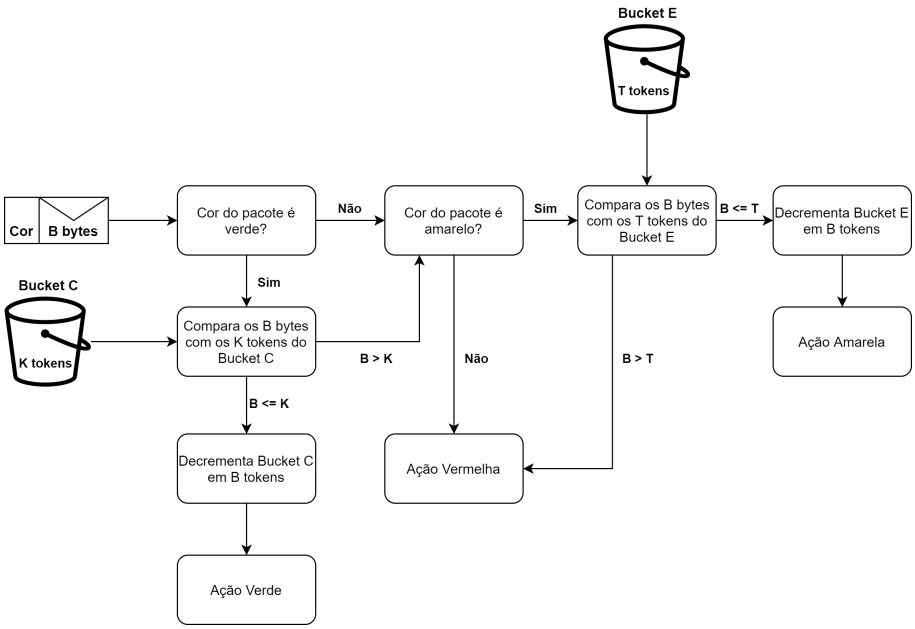


Figura 2.4: Fluxograma srTCM-*Policer Color-Aware*.

Ao se empregar o srTCM-PCA (*Policer Color-Aware* - PCA), considerando o tamanho em *bytes* do pacote e a sua cor pré-definida, o seguinte comportamento é esperado. Se o pacote é pré-colorido verde, a quantidade de *tokens* no *bucket* C é comparada com o tamanho do pacote. Se a quantidade de *tokens* disponíveis é maior, a ação verde será executada e o *bucket* C decrementado pelo tamanho do pacote. Porém, se pacote é verde e excede às definições do *bucket* C, ou é pré-colorido de amarelo, compara-se o tamanho do pacote com o número de *tokens* disponíveis no *bucket* E. Se o tamanho do pacote é menor que número de *tokens* no *bucket* E, a ação amarela é tomada e o respectivo *bucket* é decrementado na quantidade de *tokens* que representam o número de *bytes* do pacote. Se o pacote também exceder o *bucket* E ou está pré-colorido de vermelho, a ação vermelha é tomada e não ocorre decremento de *tokens* nos *buckets*. O processo descrito está representado por um fluxograma na Figura 2.4, considerando a chegada de um pacote pré-colorido e de tamanho B, com K e T *tokens* disponíveis nos buckets C e E, respectivamente, no momento da chegada do pacote.

A aplicação do trTCM-PCA (*Policer Color-Aware* - PCA) também considera o tamanho em *bytes* do pacote e a sua cor pré-definida. Assim, o comportamento esperado ao chegar um pacote é o seguinte. Se o pacote é pré-colorido vermelho, a ação vermelha será realizada. Senão, compara-se a quantidade de *tokens* do *bucket* P com o tamanho do pacote. Se não há *tokens* suficientes, o pacote excede o *bucket* P e a ação vermelha é tomada. Se existem *tokens* suficientes

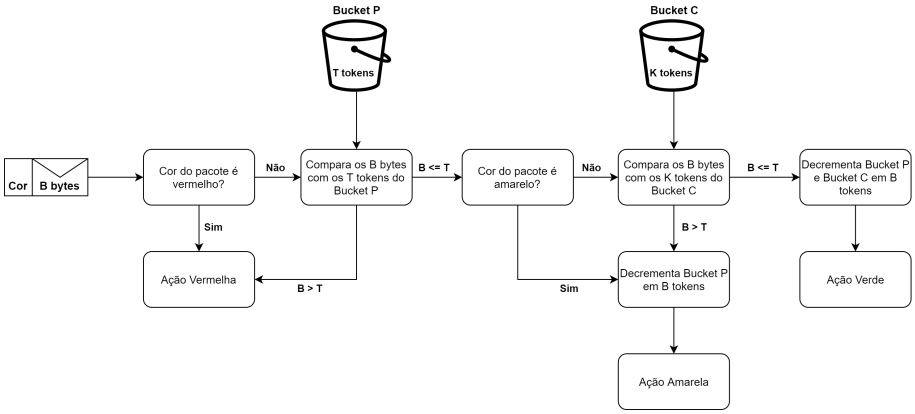


Figura 2.5: Fluxograma trTCM-*Policer Color-Aware*.

e o pacote está pré-colorido de amarelo, toma-se a ação amarela e decrementa-se o respectivo *bucket* pelo número de *tokens* que representa o tamanho do pacote. Entretanto, se o pacote está pré-colorido de verde, compara-se o número de *tokens* presentes no *bucket* C com o tamanho do pacote. Se o pacote não excede o *bucket* C, ou seja, há *tokens* suficientes, a ação verde é aplicada e são decrementados ambos os *buckets*, P e C, pelo tamanho do pacote. Se o pacote exceder o *bucket* C, decrementa-se o *bucket* P e a ação amarela é tomada. Na Figura 2.5 é utilizado um fluxograma para demonstrar os procedimentos descritos no recebimento de um pacote pré-colorido e de tamanho B, sendo que na chegada do mesmo, tem-se T e K *tokens* nos *buckets* P e C, respectivamente.

2.2 *SHAPERS*

Os *shapers*, assim como os *policers*, são mecanismos que se utilizam de uma taxa máxima definida para garantir que o fluxo de tráfego não ultrapasse um limite estabelecido. Porém, ao contrário do que ocorre nos *policers*, em que os pacotes são diretamente descartados, os *shapers* utilizam filas nas quais os pacotes são adicionados para sejam transmitidos posteriormente, quando as condições apropriadas forem atingidas.

A seguir são descritos dois dos principais *shapers*: o *Single Rate Token Bucket Shaper* e o *Leaky Bucket*.

2.2.1 *Single Rate Token Bucket Shaper*

O *Single Rate Token Bucket Shaper* (srTBS) é definido de forma análoga ao *policer* srTBP. Assim, é feita uma comparação do tamanho do pacote com a quantidade de *tokens* do *bucket*, promovendo a transmissão dos pacotes que estejam em conformidade.

Por outro lado, quando um pacote excede o *bucket*, ao invés de ser descartado, o mesmo é adicionado a uma fila, a qual é consumida, respeitando a ordem de chegada, à medida que *tokens* suficientes são adicionados ao *bucket*. No caso da fila estar cheia, então o pacote é diretamente descartado (Evans e Filsfils, 2010).

Dessa forma, ao aplicar um srTBS, o tamanho em *bytes* dos pacotes recebidos pelo mecanismo de *shaping* é comparado com a quantidade de *tokens* disponíveis no *bucket*. Com isso, se o tamanho do pacote é menor que o número de *tokens*, é atestada a sua conformidade e este é transmitido. Entretanto, se a quantidade de *tokens* no *bucket* é menor que o tamanho do pacote, o mesmo excede o limite máximo permitido. Assim, o pacote é adicionado, se possível, na fila de espera, sendo processado apenas quando estiver em conformidade com as definições do srTBS.

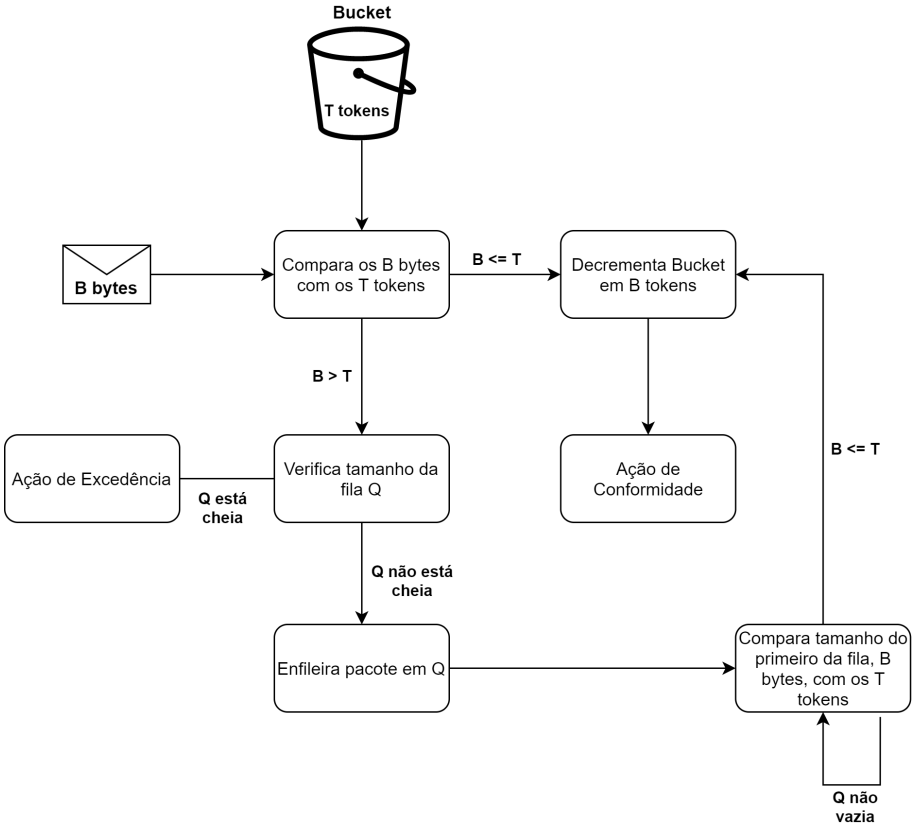


Figura 2.6: Fluxograma *Single Rate Token Bucket Shaper*.

Ou seja, o tamanho do pacote em *bytes* deve ser menor ou igual que a quantidade de *tokens* presentes no *bucket*. Desta maneira, o mecanismo pode gerar um atraso na rede, já que os pacotes podem não ser processados na ordem de chegada visto que pacotes menores que o primeiro da fila de espera podem chegar e estar em conformidade com as definições do srTBS mais rapidamente. Portanto, estes pacotes serão transmitidos antes dos da fila, quebrando a sequência de chegada e transmissão. No fluxograma da Figura 2.6 é demonstrado o comportamento descrito do srTBS ao se receber um pacote de tamanho B e tendo o *bucket* com T *tokens* disponíveis.

Apesar de não ser um objetivo primário dos mecanismos de *shaping*, o descarte de pacotes pode ocorrer. Porém, destaca-se que tais ações de descarte acontecem apenas se a fila de espera estiver totalmente preenchida, impedindo o mecanismo de armazenar novos pacotes.

2.2.2 *Leaky Bucket*

O *Leaky Bucket* (LB) é outro mecanismo de *shaping*. Entretanto, diferentemente do srTBS, onde, por definição, o *bucket* armazena *tokens*, no algoritmo do LB, o *bucket* é preenchido com os pacotes recebidos, armazenados em fila. Assim, a profundidade do *bucket* define a quantidade máxima de pacotes que podem ser enfileirados nele. Deste modo, ao se atingir a capacidade máxima, os pacotes que chegam não poderão mais ser armazenados e, portanto, serão descartados (Evans e Filsfils, 2010).

Ao se aplicar o LB em um fluxo de tráfego é necessário que a fila de pacotes do *bucket* seja consumida. Dessa forma, o consumo ocorre considerando uma taxa constante (*rate*), determinando a quantidade de pacotes que será reduzida do *bucket*, como se o mesmo estivesse com um furo na sua base. A taxa é aplicada utilizando um intervalo fixo de tempo. Assim, a cada intervalo, uma quantidade de pacotes equivalente ao valor da taxa será drenada do *bucket*.

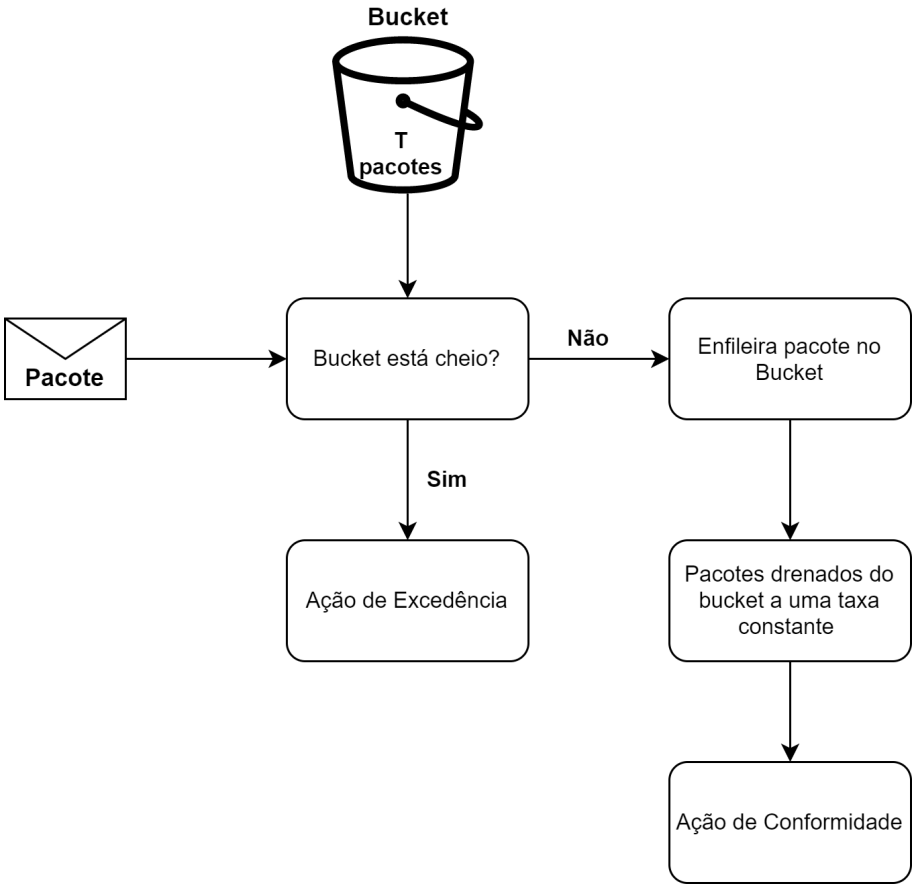


Figura 2.7: Fluxograma *Leaky Bucket*.

A Figura 2.7 mostra o processo, já descrito, realizado pelo algoritmo do LB após o recebimento de um pacote.

Por fim, destaca-se que o uso do LB tem como principal objetivo suavizar picos de tráfego na rede. Assim, é possível adequar a vazão da rede a uma taxa constante definida no *shaper*.

3 VIRTUALIZAÇÃO DE FUNÇÕES DE REDE

Este capítulo apresenta os principais conceitos de Virtualização de Funções de Rede (*Network Function Virtualization* - NFV). A Seção 3.1 introduz e discorre sobre as motivações e vantagens da utilização do paradigma NFV. Já a Seção 3.2 descreve a arquitetura de referência NFV proposta pela *European Telecommunications Standards Institute* (ETSI).

3.1 INTRODUÇÃO À VIRTUALIZAÇÃO DE FUNÇÕES DE REDE

As redes de computadores modernas dependem de diversas Funções de Rede (*Network Function* - NF) para operar. Funções de rede realizam tarefas bem definidas no núcleo da rede, tendo *interfaces* externas e comportamentos operacionais bem definidos (Rosa et al., 2014). *Firewalls*, dispositivos NAT (*Network Address Translation*) e sistemas de detecção de intrusão são exemplos de funções de rede. Tradicionalmente, as NFs são implementadas dispositivos de *hardware* dedicados à execução dessas funções, havendo o acoplamento entre o *software* e o *hardware* (Sekar et al., 2012). Assim, para usufruir de NFs é necessário adquirir e manter tais dispositivos, fato que impacta negativamente nos custos de uma rede, já que eles representam uma relevante parcela das despesas operacionais (OPEX - *OPerational EXpenditures*) e de capital (CAPEX - *CAPital EXpenditures*) (Cotroneo et al., 2014). Além disso, esses dispositivos apresentam alta complexidade ao se considerar os processos de implantação, gerência e solução de problemas. Esse fato ocorre devido ao recorrente uso de *hardware* e *software* proprietário e da necessidade de configurações realizadas de forma específica (Sherry et al., 2012). Por último, o uso de *hardware* dedicado muitas vezes não possibilita o redimensionamento dinâmico de recursos de rede conforme a demanda. Dessa forma, podem ocorrer desperdícios de recursos computacionais que se encontram disponíveis, mas permanecem ociosos (de Souza e Duarte Jr, 2020).

Nesse contexto surge o paradigma NFV, o qual objetiva implementar NFs em *software* utilizando tecnologias de virtualização. No paradigma NFV, as funções de rede são desenvolvidas em um plano de *software*, sendo desacopladas de *hardwares* físicos. Sendo assim, NFs são implementadas e operacionalizadas considerando a utilização de dispositivos virtuais. Esses dispositivos virtuais, quando executam uma função de rede, são chamados Funções Virtualizadas de Rede (*Virtualized Network Function* - VNF). Dessa forma, funções de rede podem ser disponibilizadas em servidores comerciais de prateleira. Através dessas e outras características, o paradigma NFV se torna uma alternativa ao uso de *hardware* dedicado, flexibilizando e simplificando o projeto, desenvolvimento e manutenção de NFs (Cui et al., 2012).

As redes tradicionais são compostas por diversos *hardwares* dedicados, responsáveis, por exemplo, pela execução de *Firewalls*, Inspeção Profunda de Pacotes (*Deep Packet Inpection* - DPI), Redes Virtuais Privadas (*Virtual Private Network* - VPN) e NAT. Já as redes NFV são suportadas por *hardwares* de prateleira com uma camada de virtualização. Com isso, múltiplas VNFs podem coexistir no mesmo equipamento através da instanciação e gerência de várias instâncias virtuais, estas criadas a partir de, por exemplo, virtualização completa, paravirtualização e conteinerização. Sendo assim, funções de rede no paradigma NFV tipicamente apresentam o prefixo "v", originado de virtual, como *vFirewalls*, vDPI, vVPN e vNAT. A Figura 3.1 apresenta uma comparação entre a estrutura das redes tradicionais, que utilizam *hardwares* dedicados, e as redes baseadas no paradigma NFV, que utilizam VNFs.

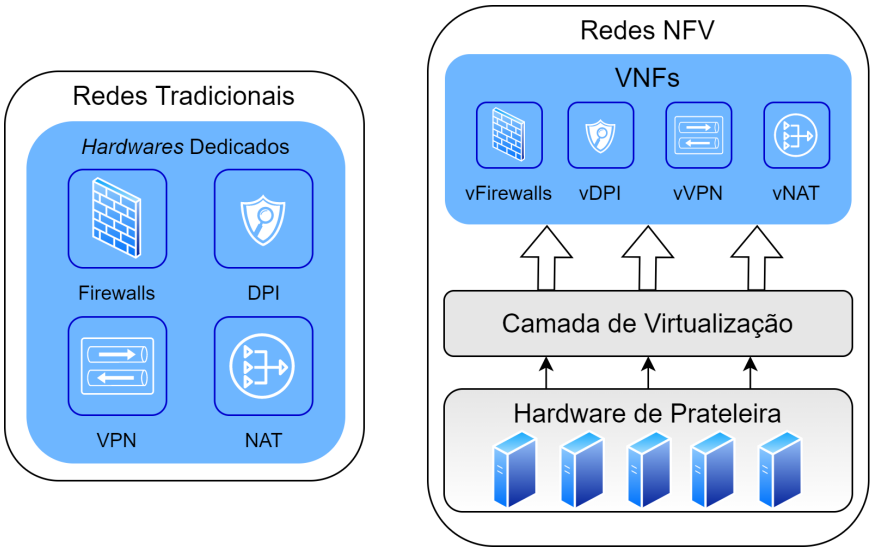


Figura 3.1: Comparação da estrutura entre as redes tradicionais e redes baseadas no paradigma NFV.

Ao se comparar as redes tradicionais às redes baseadas no paradigma NFV, verificam-se algumas vantagens das últimas. Em NFV, a execução de VNFs ocorre em ambientes virtualizados, ou seja, não necessitam de *hardwares* especializados. Desta maneira, a inclusão de novas VNFs na rede, bem como alterações e atualizações das já existentes, representam menor impacto nos custos financeiros e operacionais (OPEX e CAPEX), visto que não há necessidade de adquirir e instalar fisicamente novos equipamentos na rede. Ainda, destaca-se a flexibilização e agilidade no processo de implantação das VNFs, as quais também apresentam operação e gerenciamento facilitados em relação aos equipamentos físicos (Mijumbi et al., 2015).

Além disso, as VNFs tipicamente demandam menos energia e espaço físico já que estão desacopladas do *hardware* (Han et al., 2015). Por fim, o paradigma NFV reduz os problemas de ociosidade de recursos de rede, já que estes são redimensionados dinamicamente conforme a demanda. Desta forma, o emprego do paradigma NFV pode promover o uso eficiente e consciente dos recursos computacionais disponíveis na rede.

Como apresentado, as redes NFV contam com diversas vantagens em relação às redes tradicionais. Entretanto, também existem desvantagens e desafios que devem ser explorados. Uma desvantagem encontrada é a redução do desempenho ao realizar processamento de pacotes da rede, gerada devido à introdução da camada de virtualização no processo. Destaca-se que, tanto a academia quanto a indústria estão buscando tornar o paradigma NFV uma alternativa concreta para a utilização em redes de produção.

3.2 ARQUITETURA DE REFERÊNCIA PARA NFV

A ETSI, com o intuito de padronizar o uso da tecnologia de NFV, propôs uma arquitetura de referência para o paradigma, a qual é capaz de suportar a execução e integração de VNFs. A Figura 3.2 ilustra de maneira simplificada a arquitetura de referência NFV, contendo três principais blocos: VNFs, Infraestrutura NFV (*NFV Infrastructure* - NFVI) e Gerência e Orquestração NFV (*NFV Management and Orchestration* - NFV-MANO). Esses blocos são descritos a seguir.

O NFV-MANO é o bloco responsável por orquestrar e gerenciar o funcionamento das VNFs. Nele, se encontram os módulos de controle e orquestração, estes encarregados pelo ciclo de vida das VNFs, além de gerirem os recursos computacionais utilizados para executar as mesmas, ajustando, por exemplo, a quantidade de memória alocada em momentos de maior

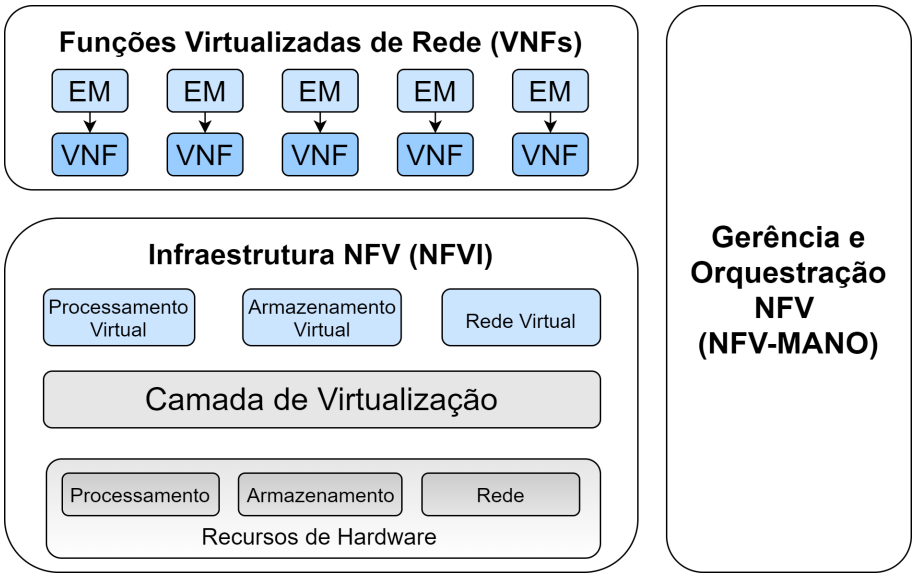


Figura 3.2: Arquitetura de referência NFV proposto pela ETSI.

demanda. Dentre as responsabilidades do NFV-MANO estão a criação das VNFs e até mesmo a composição de serviços entre elas. O NFV-MANO é dividido em três módulos: Orquestrador NFV (NFV *Orchestrator* - NFVO), Gerenciador de VNF (VNF *Manager* - VNFM) e Gerenciador da Infraestrutura Virtualizada (*Virtualized Infrastructure Manager* - VIM). Além desses, também existe o módulo OSS/BSS (*Operations Support System/Business Support System*), que não faz parte, estritamente, da arquitetura, mas representa os sistemas externos que interagem com a arquitetura (de Souza e Duarte Jr, 2020).

O bloco NFVI é a representação da infraestrutura virtualizada em que as VNFs são executadas. Uma parte do NFVI é relacionada aos recursos computacionais físicos, ou seja, recursos de *hardware* de propósito geral, sendo esses responsáveis pelo processamento, armazenamento e rede utilizados pelas VNFs. Entretanto, para que as VNFs possam usufruir desses recursos físicos há a introdução de uma Camada de Virtualização, responsável por abstrair os recursos físicos em recursos virtuais, desacoplando, assim, o *software* do *hardware*. Para auxiliar a Camada de Virtualização, há o *hypervisor*, que tem como responsabilidade a criação e gerência dos dispositivos virtuais, a exemplos de Máquinas Virtuais (*Virtual Machines* - VM) e *containers*. Dessa forma, a estratégia adotada pela ETSI para a arquitetura NFV permite a execução independente e isolada de cada VNF.

O terceiro bloco presente na arquitetura NFV é o das VNFs. Ele corresponde à representação das instâncias dos dispositivos virtuais, os quais são responsáveis pela execução, sobre o NFVI, das NFs. No bloco, cada VNF é associada a um Descritor VNF (VNFD - VNF *Descriptor*), no qual as VNFs são especificadas em termos dos seus requisitos operacionais e de implantação. Assim, os VNFDs possuem informações que podem ser utilizadas pelo bloco NFV-MANO, como, por exemplo, a existência de políticas de realocação de recursos computacionais conforme a demanda. Além disso, as VNFs vinculam-se a um Gerenciador de Elemento (*Element Manager* - EM), que será responsável pelas funcionalidades de gerência FCAPS (*Fault, Configuration, Accounting, Performance and Security Management*) da instância virtual.

A Figura 3.3 mostra as conexões e divisões na arquitetura do NFV-MANO. Como já descrito, o NFV-MANO é subdividido em três módulos: NFVO, VNFM e VIM, além do módulo OSS/BSS. O NFVO é o responsável pela gerência e orquestração dos recursos alocados nos dispositivos virtuais. Além disso, o módulo também é responsável pelo redimensionamento

dinâmico dos recursos da VNF de acordo com sua demanda, pelo gerenciamento das políticas e pela autorização e validação das requisições realizadas pelo NFVI.

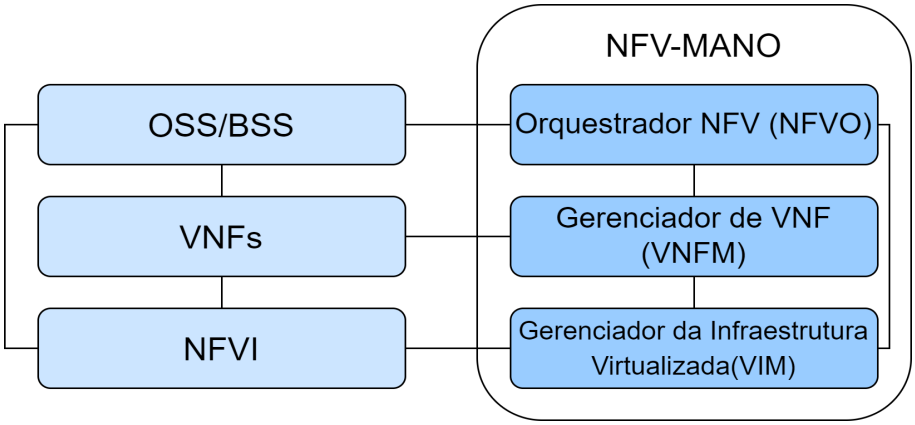


Figura 3.3: Módulos e conexões do bloco NFV-MANO.

Já o módulo VNFM é responsável, principalmente, pelo gerenciamento do ciclo de vida das VNFs. Ao VNFM são atribuídas as funções de instanciação, remoção, atualização, configuração, atualização e escalonamento de VNFs, além do monitoramento da utilização de recursos e notificações de falhas. Desta forma, o VNFM deve ter total acesso aos VNFDs para poder executar funcionalidades especificadas no mesmo, além de ter acesso aos repositórios e as diferentes versões das VNFs, para que todas as operações sejam suportadas. A arquitetura proposta pelo ETSI permite a utilização de um ou mais VNFMs, desde que as operações realizadas por eles sejam suportadas por qualquer VNF, ou seja, independentemente da sua implementação (ETSI, 2014).

O módulo VIM, por sua vez, é responsável pelo controle e gerência de recursos computacionais da infraestrutura virtualizada, ou seja, do processamento, armazenamento e rede. Assim, o VIM cria os dispositivos virtuais, modifica a alocação de recursos computacionais e monitora a infraestrutura. A arquitetura do NFV-MANO proposta pela ETSI também permite mais de uma instância de VIM, assim como no caso do VNFM, sendo que cada VIM é responsável por gerenciar uma infraestrutura virtualizada diferente.

Por fim, tem-se o módulo OSS/BSS, que não faz parte oficialmente da arquitetura proposta pela ETSI, mas representa os sistemas e aplicações externas que se comunicam com a rede NFV. O uso desse módulo se deve ao fato de que o paradigma NFV não permite que sistemas externos gerenciem as VNFs diretamente. Desta forma, o OSS/BSS é responsável por realizar a comunicação, através de *interfaces* padronizadas, do sistema externo com o bloco NFV-MANO para serem executadas requisições de gerência, como instanciação, atualização e remoção de VNFs.

Atualmente, diversas plataformas que implementam a arquitetura de referência NFV-MANO estão disponíveis. Entre elas encontram-se o Tacker (OPENSTACK, 2016), OSM (ETSI, 2020), OpenBaton (FRAUNHOFER-FOKUS e TU-BERLIN, 2017) e Vines (Flauzino et al., 2021). Existem também as chamadas plataformas de execução de VNFs, que proveem recursos que permitem a execução de NFs em instâncias virtuais. Exemplos de plataformas de execução de VNF incluem o ClickOS (Martins et al., 2014), Click-On-OSv (da Cruz Marcuzzo et al., 2017) e COVEN (Garcia et al., 2019). Associados a estas plataformas existem também sistemas que implementam estratégias eficientes para a instalação de funções e serviços de rede no substrato físico disponível, exemplos incluem o CUSCO (Fulber-Garcia et al., 2020) e Piecing NFV (Luizelli et al., 2015).

4 FUNÇÕES VIRTUAIS DE REDE PARA ENGENHARIA DE TRÁFEGO

A engenharia de tráfego tem papel fundamental para garantir o bom desempenho de uma rede. Em particular, é importante haver formas acessíveis e práticas de implementar, implantar e gerenciar as diversas técnicas de engenharia de tráfego na rede. A tecnologia NFV surge como uma dessas alternativas, em que os mecanismos de engenharia de tráfego são implementados como funções de rede, que podem ser implantadas em ambientes virtualizados de forma fácil e a um custo (OPEX e CAPEX) reduzido em relação a alternativas.

Essa abordagem, além de herdar todos os benefícios da tecnologia NFV, possibilita maior flexibilidade na utilização da engenharia de tráfego, facilitando sua configuração e permitindo a adoção de novas estratégias à medida que se tornem necessárias. Além disso, para executar os mecanismos são utilizados apenas os recursos computacionais necessários, tendo fácil escalabilidade conforme a demanda.

Assim, neste capítulo é proposto o NFV-TE (*NFV-Traffic Engineering*): um *framework* parametrizável para geração de funções de rede para engenharia de tráfego, usufruindo da tecnologia NFV. Na Seção 4.1 são apresentados trabalhos relacionados ao uso de NFV para engenharia de tráfego. Na Seção 4.2 é descrita a implementação do *framework* NFV-TE. As Seções 4.3 e 4.4 apresentam, respectivamente, os mecanismos de *policing* e *shaping* implementados no *framework*, bem como seus detalhes de configuração e funcionamento.

4.1 TRABALHOS RELACIONADOS

Esta seção apresenta alguns dos trabalhos relacionados ao uso de NFV para engenharia de tráfego. Dong e Clemm (2021) propõem a estratégia de Lavagem de Pacotes (*Packet Wash*) que visa garantir latência ponto-a-ponto com alta precisão, reduzindo o descarte de pacotes quando a rede se encontra congestionada. A Lavagem de Pacotes provê a funcionalidade de reduzir o tamanho do pacote, descartando seletivamente pedaços da carga útil do mesmo, sem haver interrupção da entrega. Os pacotes carregam informações sobre o processo de lavagem, para que não sejam interpretados como corrompidos ou comprometidos. O algoritmo sugerido pelos autores visa reduzir o tempo de permanência (*dwell time*) do pacote nos roteadores ao ter o risco do mesmo sofrer atraso. Assim, a proposta dos autores é negociar a transmissão de blocos de baixa prioridade para se atingir uma latência aprimorada, aumentando a possibilidade de o tráfego satisfazer o rigoroso prazo de latência de ponto-a-ponto em diferentes cenários.

Kuribayashi (2021) propõe um método de *shaping* dinâmico utilizando os paradigmas *Software-Defined Networking* (SDN) e NFV, o qual seleciona dinamicamente o fluxo de comunicação e também o ponto da rede ótimo para a realização do processo de *shaping*, quando necessário. O método implementado auxilia no uso da largura de banda de rede e na capacidade de processamento de pacotes retransmitidos, tornando-os mais flexíveis e eficientes. O autor disponibiliza e apresenta as funções utilizadas no método de *shaping* dinâmico proposto, sendo a viabilidade do método confirmada por um sistema de avaliação.

Garcia et al. (2018) propõem um sistema de mitigação de ataques *Distributed Denial of Service* (DDoS), chamado DeMONS, utilizando os conceitos de NFV em conjunto com alocação dinâmica e um mecanismo de reputação. O DeMONS é uma solução híbrida composta por cinco módulos principais, os quais são VNFs, sendo eles: o classificador de prioridade, o *firewall*, o alocador, o *policer* de tráfego e o gerenciador. O classificador de prioridade analisa os fluxos de rede e marca-os com uma reputação entre 0 e 1. Os fluxos marcados com reputação 0 são

bloqueados no *firewall*. Já fluxos marcados com reputação maior que 0 são alocados em diferentes túneis, sendo eles de baixa e alta prioridade. O módulo de *policer* de tráfego é executado se o túnel de baixa prioridade ficar sobrecarregado. Assim, o *policer* limita o tráfego de cada fluxo de rede baseando-se em suas prioridades. O módulo gerenciador é responsável por provisionar e controlar os ciclos de vida de todos os módulos. Os resultados obtidos no trabalho demonstram a viabilidade do DeMONS em mitigar eficientemente ataques DDoS.

4.2 *FRAMEWORK* NFV-TE

O Capítulo 2 apresentou o conceito de engenharia de tráfego, com o qual é possível empregar mecanismos de gerenciamento de pacotes em uma rede. Já no Capítulo 3 discorreu-se sobre a tecnologia de virtualização de funções de rede, a qual possibilita a implantação de funções de rede em *hardware* de propósito geral. Com base nesses conceitos, este trabalho propõe o *framework* NFV-TE para definição e geração de mecanismos de engenharia de tráfego no formato de funções de rede capazes de serem executados de forma virtualizada com o uso da tecnologia NFV.

A construção do *framework* foi feita utilizando-se a linguagem *Python 3*, conhecida por sua flexibilidade e amplo suporte em diversas plataformas, em particular com o uso da biblioteca "*socket*", nativa da linguagem. O funcionamento do NFV-TE é baseado em um arquivo de configurações no formato *JSON*, que deve incluir a categoria (*policing* ou *shaping*) e nome da função de rede desejada, bem como um conjunto de parâmetros a serem utilizados pela NF. Neste arquivo, devem ser definidos também os nomes das *interfaces* de rede do cliente e do servidor dos quais os pacotes serão enviados e recebidos. As funções de rede disponíveis e seus parâmetros são descritos nas Seções 4.3 e 4.4.

Após a leitura e subsequente validação dos parâmetros, é feita a geração de código-fonte, também em *Python 3*, para a função de rede designada utilizando variáveis com os valores correspondentes aos parâmetros definidos e gerando como resultado um arquivo com a extensão ".py"que pode ser executado em qualquer dispositivo com suporte à linguagem.

4.3 *POLICERS*

Para geração de NFs de *Policers* é preciso definir no arquivo de configurações o campo de categoria como "*policing*". Nesta categoria podem ser geradas funções de rede para três *policers*: *Single Rate Token Bucket Policer*, *Single Rate Three Color Marker Policer* e *Two Rate Three Color Marker Policer*, com a variação ou não de *Color-Aware* para os dois últimos.

4.3.1 *Single Rate Token Bucket Policer*

Conforme apresentado, o srTBP atua verificando o tamanho de cada pacote, transmitindo-os apenas caso haja *tokens* suficientes no *bucket*. Sua implementação como NF utiliza duas *threads*, uma responsável pelo processamento de novos pacotes que chegam na rede e outra para a reposição dos *tokens* a uma taxa constante.

Os parâmetros de configuração necessários para o srTBP são: "*bucket_size*"que corresponde ao tamanho inicial do *bucket*; "*bucket_max_size*"que corresponde ao tamanho máximo do *bucket*; "*interval*"que corresponde ao valor em segundos do intervalo entre cada reposição de *tokens*; e "*rate*"que corresponde à quantidade de *tokens* que são adicionados a cada intervalo.

4.3.2 *Single Rate Three Color Marker Policer*

O funcionamento do srTCM-*Policer*, conforme descrito anteriormente, se dá pela execução das ações referentes à marcação dos pacotes nas cores verde, amarela e vermelha, de acordo a quantidade de *tokens* presentes em dois *buckets*: o Conforme (C) e o de Exceção (E), que têm seus *tokens* incrementados a uma taxa constante e única para os dois. Caso o pacote recebido tenha tamanho B menor ou igual ao número de *tokens* no *bucket* C, a ação verde é executada e decrementa-se B *tokens* do *bucket* C. Caso contrário, se B for menor ou igual ao do número de *tokens* no *bucket* E, então a ação amarela é executada e decrementa-se B *tokens* do *bucket* E. Caso nenhuma das condições seja verdadeira, a ação vermelha é então executada.

A implementação do srTCM-*Policer* como NF foi feita utilizando-se duas *threads*, uma responsável por incrementar o número de *tokens* nos *buckets*; e outra responsável por avaliar os pacotes recebidos com os *buckets* C e E, executando a ação apropriada conforme a cor que o pacote é marcado.

Os parâmetros de configuração necessários para o *Single Rate Three Color Marker Policer* são: "*interval*"que corresponde ao valor em segundos do intervalo entre cada reposição de *tokens*; "*rate*"que corresponde à quantidade de *tokens* que são adicionados aos dois *buckets* a cada intervalo; "*bucketF_size*"e "*bucketS_size*"que correspondem às quantidades iniciais de *tokens* nos *buckets* C e E, respectivamente; "*bucketF_max_size*"e "*bucketS_max_size*"que correspondem ao número máximo de *tokens* para os *buckets* C e E, respectivamente.

4.3.3 *Two Rate Three Color Marker Policer*

O trTCM-*Policer* atua de forma similar ao srTCM-*Policer*, também executando ações correspondentes à marcação dos pacotes nas cores de verde, amarela e vermelha, conforme a quantidade de *tokens* em dois *buckets*: o de Pico (P) e o de Conformidade (C), que têm seus *tokens* incrementados a uma taxa constante, específica para cada *bucket*. Caso o pacote recebido tenha tamanho B maior do que o número de *tokens* disponíveis no *bucket* P, a ação vermelha é executada, caso contrário decrementa-se B *tokens* do *bucket* P sendo feita então a comparação com o tamanho do *bucket* C. Caso B seja maior que o número de *tokens* no *bucket* C, a ação amarela é executada, caso contrário decrementa-se B *tokens* do *bucket* C e a ação verde é então executada.

A implementação do trTCM-*Policer* como NF foi feita utilizando-se duas *threads*, uma responsável por incrementar o número de *tokens* nos *buckets* de acordo com as taxas específicas de cada *bucket*; e outra responsável por avaliar os pacotes recebidos com os *buckets* P e C, executando a ação apropriada segundo a cor com que o pacote é marcado.

Os parâmetros de configuração necessários para o *Two Rate Three Color Marker Policer* são: "*interval*"que corresponde ao valor em segundos do intervalo entre cada reposição; "*rateS*"e "rateF"que correspondem à quantidade de *tokens* que são adicionados aos *buckets* P e C, respectivamente, a cada intervalo; "*bucketS_size*"e "*bucketF_size*"que correspondem às quantidades iniciais de *tokens* nos *buckets* P e C, respectivamente; "*bucketS_max_size*"e "*bucketF_max_size*"que correspondem ao número máximo de *tokens* para os *buckets* P e C, respectivamente.

4.3.4 *Color-Aware Policers*

Um *Color-Aware Policer*, em geral, é utilizado após um *marker*, verificando a coloração atribuída a cada um dos pacotes que chegam até ele e decidindo por uma ação apropriada conforme a sua configuração.

Foram implementadas as versões *Color-Aware* do srTCM e do trTCM e ambas foram construídas de forma similar. Para a utilização da versão *Color-Aware*, o parâmetro de configuração *"color_aware"* deve ser marcado como verdadeiro. Com isso, os seguintes parâmetros também devem ser definidos: *"ca_bucketF_size"* e *"ca_bucketS_size"* que correspondem às quantidades iniciais de *tokens* nos *buckets* C e E no caso do srTCM-PCA e *buckets* C e P no caso do trTCM-PCA; *"ca_bucketF_max_size"* e *"ca_bucketS_max_size"* que correspondem ao número máximo de *tokens* para os *buckets* C e E no caso do srTCM-PCA e *buckets* C e P no caso do trTCM-PCA; o parâmetro *"ca_rate"* deve ser usado no srTCM-PCA correspondendo à quantidade de *tokens* que são adicionados aos *buckets* C e E a cada intervalo; já no trTCM-PCA são necessários os parâmetros *"ca_rateF"* e *"ca_rateS"* que correspondem à quantidade de *tokens* que são adicionados aos *buckets* C e P, respectivamente, a cada intervalo.

4.4 *SHAPERS*

Para a geração de *Shapers*, o campo de categoria deve ser definido como "*shaping*". Nesta categoria podem ser geradas funções para dois tipos de *shapers*: *Single Rate Token Bucket Shaper* e *Leaky Bucket*.

4.4.1 *Single Rate Token Bucket Shaper*

O srTBS atua de forma similar ao srTBP, porém quando um pacote excede a quantidade de *tokens* do *bucket*, ao invés de ser descartado ele é adicionado a uma fila, consumida à medida que novos *tokens* são adicionados ao *bucket*. Sua implementação como NF foi feita com o uso de duas *threads*, uma responsável pela reposição dos *tokens* no *bucket* e outra responsável pelo processamento dos pacotes trafegando pela rede.

Os parâmetros de configuração necessários para o *Single Rate Token Bucket Shaper* são: "*bucket_size*"que corresponde ao número inicial de *tokens* no *bucket*; "*bucket_max_size*"que corresponde ao número máximo de *tokens* no *bucket*; "*interval*"que corresponde ao valor em segundos do intervalo entre cada reposição; "*rate*"que corresponde à quantidade de *tokens* que são adicionados a cada intervalo; e "*queue_max_size*"que corresponde ao tamanho máximo da fila de pacotes.

4.4.2 *Leaky Bucket*

Como descrito anteriormente, o *Leaky Bucket* é um mecanismo de *shaping* que armazena os pacotes recebidos em um *bucket*, transmitindo-os a uma taxa constante em ordem de chegada. Se o *bucket* atingir sua capacidade máxima definida, os novos pacotes recebidos serão descartados.

Para geração do *Leaky Bucket* como NF, o *framework* NFV-TE implementa o *bucket* como uma fila, a qual é gerenciada por duas *threads*. Uma *thread* é responsável por adicionar os pacotes à fila, ou descartá-los caso a fila esteja cheia. A outra *thread* executa a transmissão dos pacotes da fila de forma periódica e constante.

Os parâmetros de configuração necessários para o *Leaky Bucket* são: "*bucket_max_size*"que corresponde ao tamanho máximo da fila (*bucket*); "*interval*"que corresponde ao valor em segundos do intervalo entre cada transmissão; e "*packets_to_release*"que corresponde à quantidade de pacotes transmitidos a cada intervalo.

5 RESULTADOS EXPERIMENTAIS

Neste capítulo são apresentados os resultados experimentais obtidos na execução dos mecanismos de engenharia de tráfego gerados pelo *framework* NFV-TE. A Seção 5.1 descreve como foram feitos os experimentos, incluindo ambiente de execução e perfis de tráfego aos quais os mecanismos foram submetidos. Na Seção 5.2 são apresentados os resultados obtidos para os mecanismos de *policing* implementados. Já na Seção 5.3 são apresentados os resultados obtidos para os mecanismos de *shaping* implementados.

5.1 AMBIENTE DOS EXPERIMENTOS

O ambiente para realização dos experimentos é composto por três VMs, todas executando o sistema Linux Ubuntu 18.4. A primeira tem a função de cliente, sendo assim, envia pacotes para um servidor. A segunda, tem a função de servidor, que recebe pacotes enviados por um cliente. Por último, situada entre o cliente e servidor, uma VM executa o *framework* NFV-TE, o qual pode colocar em funcionamento um dos mecanismos implementados de engenharia de tráfego e devidamente parametrizados, realizando o *policing* ou *shaping* do tráfego de pacotes entre o cliente e o servidor. Sendo assim, o funcionamento do ambiente se dá da seguinte forma: o cliente envia pacotes destinados ao servidor, mas antes de chegar ao servidor os pacotes são processados por um dos mecanismos de engenharia de tráfego do NVF-TE, o qual verifica a conformidade dos pacotes ao mesmo. Se conformes, os pacotes são transmitidos ao servidor, senão são descartados. A Figura 5.1 mostra a representação do ambiente de experimentos com as três VMs em execução.

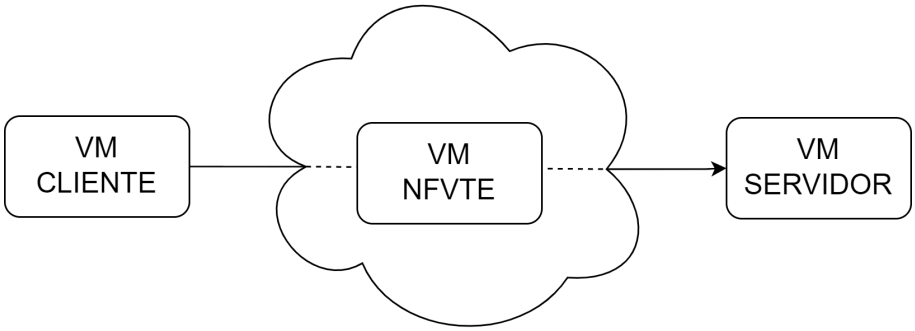


Figura 5.1: Representação do Ambiente de Experimentos

Para verificar o funcionamento dos mecanismos de *policing* e *shaping*, o cliente pode enviar a cada execução um de quatro perfis de tráfegos, sendo eles: Alfa (Sarvotham et al., 2001), Beta (Sarvotham et al., 2001), Elefante (Hamdan et al., 2020) e Guepardo (Maji et al., 2017). Tais perfis de tráfego são descritos a seguir.

O tráfego Alfa realiza transmissões de 10 pacotes em um segundo, assim o intervalo entre o envio de cada pacote é de 0,1 segundo. Entretanto, ao realizar o envio de 10 pacotes, o tráfego é paralisado por 2 segundos, ou seja, não há envio de pacotes para o servidor dentro desse tempo. No término dos 2 segundos de intervalo, o cliente volta a enviar os pacotes ao servidor, repetindo o procedimento descrito. No tráfego do Alfa são enviados pacotes com tamanho de

548 *bytes*, sendo assim, fora do período de paralisação são enviados para o servidor 5480 *bytes* por segundo.

No tráfego Beta há a transmissão de dois tipos de pacotes com diferentes tamanhos: 148 *bytes* e 348 *bytes*. O envio desses pacotes acontece de maneira alternada. Com isso, a cada 10 pacotes enviados troca-se o tipo do pacote. Além disso, são enviados 10 pacotes por segundo, sendo assim um pacote enviado a cada 0,1 segundos. Dessa forma, a quantidade de *bytes* transmitidos por segundo também é alternada, podendo ser de 1480 *bytes* ou 3480 *bytes*.

O tráfego Elefante realiza transmissões de pacotes maiores. O pacote transmitido, nesse caso, tem tamanho de 1048 *bytes*. Além disso, seu fluxo de envio é mais lento se comparado aos demais perfis de tráfego, realizando o envio de apenas 5 pacotes em 1 segundo, ou seja, tem um intervalo de 0,2 segundos a cada envio.

Já o Guepardo tem o fluxo mais rápido dentre os perfis de tráfego, realizado o envio de 20 pacotes em 1 segundo. Assim, tem-se o intervalo de 0,05 segundos a cada envio de pacote para o servidor. Entretanto, o tamanho dos pacotes enviados é pequeno se comparado aos demais, tendo apenas 68 *bytes*. Na Figura 5.2 são mostradas as informações de cada perfil de tráfego, incluindo a quantidade de pacotes por segundo, o tamanho de cada pacote, o número de *bytes* transmitidos por segundo e o intervalo entre os envios.

	Tamanho Pacote (Bytes)	Pacotes Transmitidos (1s)	Intervalo entre envios	Bytes Transmitidos (1s)
ALFA	348	10 ou 0*	0.1s a cada pacote + 2s após 10 pacotes	3480 ou 0*
BETA	148 ou 348 (2 fluxos)	10	0.1s a cada pacote	1480 ou 3480 (Alternado a cada 1s)
ELEFANTE	1048	5	0.2 a cada pacote	5240
GUEPARDO	68	20	0.05s a cada pacote	1360
*Será transmitido 0 pacotes em 1s e portanto 0 bytes em 1s se o tráfego estiver no intervalo de 2s após 10 pacotes.				

Figura 5.2: Informações sobre os tipos de tráfego.

Além desses tráfegos, foram realizadas medições ativas do *throughput* máximo da rede para avaliar o funcionamento dos mecanismos implementados no NFV-TE. Tais medições foram feitas especialmente para os mecanismos *Single Rate Token Bucket Policer* e o *Leaky Bucket*. Assim, é gerado um tráfego de rede de 10Mbps, sendo transmitidos 1000 pacotes de 1250 *bytes* por segundo, no período de 10 segundos, totalizando 10 mil pacotes. Dessa forma, é medido o *throughput* máximo da rede, sendo possível verificar a quantidade de pacotes descartados e a quantidade de *bytes* enviados pelo cliente e recebidos pelo servidor.

Vale ressaltar que nos experimentos o intervalo de tempo, seja para a adição de *tokens* aos *buckets* ou para o consumo do *bucket*, é definido em 1 segundo, padronizando os experimentos e tornando os resultados comparáveis. Os *buckets* são inicializados com sua capacidade máxima de *tokens* disponibilizados. Além disso, o número de pacotes transmitidos pelos tráfegos Alfa, Beta, Elefante e Guepardo será sempre de 500 pacotes, também para padronizar e facilitar a comparação entre os experimentos realizados nos diferentes mecanismos. As análises dos resultados experimentais obtidos se baseiam em como os diferentes valores para os parâmetros selecionados em cada mecanismo de *policing* e *shaping* impactam no número de transmissões e descartes realizados pelo mesmo.

5.2 *POLICERS*

Mecanismos de *policing* asseguram que o fluxo de tráfego que passa pela rede respeite uma taxa máxima definida, realizando a transmissão ou descarte de pacotes quando necessário. A seguir são mostrados os resultados experimentais obtidos na execução de quatro *policers*: o *Single Rate Token Bucket Policer* (srTBP), o *Single Rate Three Color Marker Policer* (srTCM-*Policer*), o *Two Rate Three Color Marker Policer* (trTCM-*Policer*) e os *Color-Aware Policers*.

5.2.1 *Single Rate Token Bucket Policer*

Os experimentos com o srTBP foram realizados alterando dois parâmetros: a rajada (*burst*) e a taxa de reposição de *tokens* ao *bucket* (*rate*). O primeiro indica a capacidade máxima de *tokens* do *bucket*. O segundo indica a quantidade de *tokens* que são adicionados ao *bucket* a cada intervalo. Para ambos parâmetros, os valores foram escolhidos baseados na quantidade de *bytes* transmitidos por segundo em cada perfil de tráfego: Alfa, Beta, Elefante e Guepardo. Após uma avaliação preliminar foram selecionados os seguintes valores para a taxa de reposição de *tokens* ao *bucket*: 1500, 2700 e 3900 *tokens*, e os seguintes valores para a rajada: 2700, 4000 e 7000 *tokens*.

Os resultados experimentais obtidos na execução do mecanismo srTBP na rede, para os diferentes valores da rajada e com a taxa de reposição de *tokens* ao *bucket* igual a 1500, podem ser visualizados na Figura 5.3. Para o tráfego Guepardo, o mecanismo de srTBP não executou descartes de pacotes, encaminhando a totalidade dos mesmos do cliente ao servidor. Por outro lado, o srTBP executou transmissões de, em média, 75,2%, 71% e 29,8%, da totalidade dos pacotes, para os tráfegos, respectivamente, Alfa, Beta e Elefante.

Ainda, é possível perceber que existem variações ínfimas na quantidade de pacotes transmitidos para os tráfegos Beta e Elefante conforme a variação da rajada. Por outro lado, nesse mesmo contexto, existe uma grande e uma ínfima mudança no número de transmissões realizadas para o tráfego Alfa, dependendo da variação da rajada. Assim, para o tráfego Alfa o aumento da rajada de 2900 para 4000 *tokens*, aumentou as transmissões em 45,6%, enquanto o aumento de 4000 para 7000 *tokens* causou um acréscimo ínfimo no número de transmissões, sendo de apenas 1,2%. Já o tráfego Guepardo, mantém-se em 100% de transmissões dos pacotes, independentemente do valor da rajada. A baixa variação no número de transmissões de pacotes, ou ausência dela, dada a variação da rajada, se deve ao fato da taxa de reposição de *tokens* ao *bucket* ser menor que a quantidade de *bytes* por segundo transmitidos pelos fluxos. Isso consome os *tokens* do *bucket* rapidamente, zerando os mesmos e, consequentemente, fazendo com que o comportamento do srTBP se mantenha praticamente constante, independentemente dos valores de rajada.

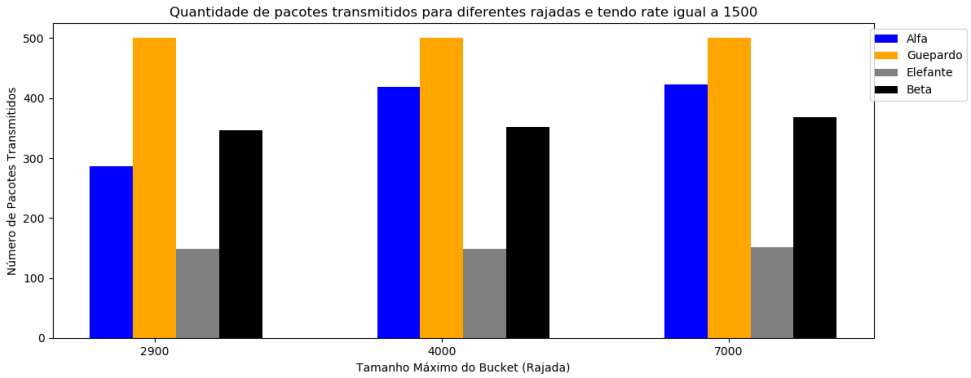


Figura 5.3: Transmissões realizadas pelo srTBP alternando a rajada e *rate* igual a 1500 para os diferentes tipos de tráfego.

Na Figura 5.4 são apresentados os resultados obtidos na execução do mecanismo srTBP na rede para diferentes valores de rajada e com a taxa de reposição de *tokens* ao *bucket* igual a 2700. Para o tráfego Beta, houve mudanças no comportamento do srTBP se comparado ao comportamento verificado na Figura 5.3, quando a taxa de reposição de *tokens* ao *bucket* era de 1500. É possível verificar um aumento no número de transmissões realizadas, atingindo 100%

de transmissões quando a rajada é igual a 4000 ou 7000 *tokens* e tendo um aumento de 69,2% para 93% de pacotes transmitidos dentre os 500 recebidos, quando a rajada é igual a 2900. Isso se deve ao fato de que no tráfego Beta são transmitidos 1480 *bytes* ou 3480 *bytes* alternados por segundo, valores menores e maiores, respectivamente, que a taxa de reposição de *tokens* ao *bucket*. Desta forma, quando a rajada é maior que 3480 *bytes*, sempre há disponibilidade de *tokens* no *bucket*, visto que quando recebidos 1480 *bytes* sobram 1220 *tokens*, que somados aos 2700 *tokens* adicionados são suficientes para os 3480 *bytes* seguintes. Entretanto, se o valor da rajada for menor que 3480 *bytes*, como no caso de 2900, atinge-se cerca de 93% de transmissões, pois o *bucket* não terá *tokens* suficientes para a transmissão dos 3480 *bytes*, aumentando o descarte à medida que a rajada diminui.

Para o tráfego Elefante, é possível verificar um aumento no número de transmissões ao se comparar com os resultados obtidos na Figura 5.3 para todos os diferentes valores de rajada. Assim, enquanto com a taxa de reposição de *tokens* ao *bucket* igual a 1500, foram obtidas, em média, 29,8% de transmissões. Já com a rajada igual a 2700 *tokens*, foram obtidos 40,4%, 52,4% e 53% dos pacotes transmitidos para as rajadas de 2900, 4000 e 7000 *tokens*, respectivamente, tendo, em média, 48,6% da totalidade dos pacotes transmitidos. Desta forma, é possível verificar que quando se aumenta a rajada inicialmente de 2900 para 4000 *tokens*, há um aumento significativo de 29,7% no número de transmissões realizadas pelo srTBP. Entretanto, ao aumentar a rajada de 4000 para 7000 *tokens*, o aumento é de 1,1%, indicando que acrescer a rajada acima de 4000 altera o resultado minimamente. Para o Guepardo, mantêm-se os 100% de pacotes transmitidos, assim como na Figura 5.3, quando a taxa de reposição de *tokens* ao *bucket* é igual a 1500.

Já para o Alfa, ao se comparar com os resultados apresentados na Figura 5.3, quando a taxa de reposição de *tokens* ao *bucket* é igual a 1500, é possível verificar o aumento no número de pacotes transmitidos pelo srTBP, acrescendo de 57,4%, 83,6% e 84,6% para 64,8%, 100% e 100% de pacotes transmitidos para as rajadas de 2900, 4000 e 7000 *tokens*, respectivamente. Isso se deve ao fato de o *bucket* ser preenchido com os *tokens* mesmo no período de não recebimento de pacotes, fazendo com que haja maior disponibilidade dos mesmos quando os pacotes forem recebidos.

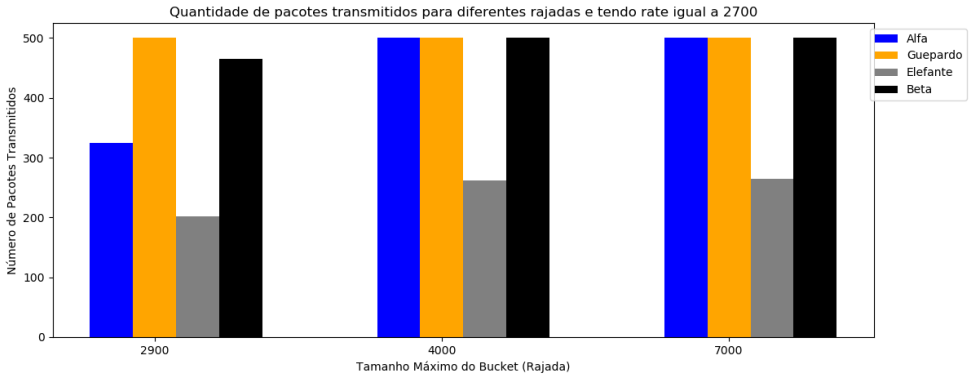


Figura 5.4: Transmissões realizadas pelo srTBP alternando a rajada e *rate* igual a 2700 para os diferentes tipos de tráfego.

A seguir, a Figura 5.5 apresenta os resultados obtidos na execução do mecanismo srTBP na rede, para diferentes valores de rajada e com a taxa de reposição de *tokens* ao *bucket* igual a 3900. Ao analisar os resultados obtidos para o tráfego Elefante, é possível verificar o aumento no número de transmissões realizadas pelo srTBP à medida que o valor da rajada aumenta, sendo obtidas taxas de 40,6%, 60,6% e 75,8% de pacotes transmitidos pelo mecanismo para os valores

de rajada 2900, 4000 e 7000 *tokens*, respectivamente. Isso ocorre já que a taxa de reposição de *tokens* ao *bucket* está próxima, mas ainda é inferior à quantidade de *bytes* transmitidos por segundo pelo tráfego Elefante. Assim, à medida que se aumenta o valor da rajada, demora mais tempo para que o *bucket* esvazie. Entretanto, o *bucket* ainda se esvaziará, visto que, por segundo, são consumidos mais *tokens* do *bucket* do que adicionados. Desta forma, o descarte de pacotes é apenas retardado.

Para o tráfego Beta, são obtidas taxas de 93,2%, 100% e 100% de pacotes transmitidos para as rajadas 2900, 4000 e 7000 *tokens*, respectivamente. Atingir 100% dos pacotes transmitidos se deve ao fato de tanto a rajada, quanto a taxa de reposição de *tokens* ao *bucket* serem maiores que a quantidade de *bytes* transmitidos por segundo pelo fluxo. Desta maneira, são consumidos menos *tokens* do que adicionados, fazendo com que haja sempre a disponibilidade dos mesmos ao se receber um pacote. Para o Guepardo, mantêm-se os 100% de transmissões, assim como nos resultados das Figuras 5.3 e 5.4, quando a taxa de reposição de *tokens* ao *bucket* é igual a 1500 e 2700, respectivamente.

Já para o tráfego Alfa, as taxas de transmissão obtidas foram de 75,4%, 100% e 100% para as rajadas de 2900, 4000 e 7000 *tokens*, respectivamente. Por último, é importante verificar que os resultados obtidos para as diferentes taxas de reposição de *tokens* ao *bucket*, 2700 e 3900, e com a rajada igual a 2900 têm poucas variações, havendo um aumento no número de pacotes transmitidos de 16,3%, 0,2%, 0,5% e 0%, respectivamente, para os tráfegos Alfa, Beta, Elefante e Guepardo. Isso se deve ao fato de que apesar de a taxa de reposição de *tokens* ao *bucket* ser de 3900, o *bucket* tem capacidade máxima para 2900 *tokens*, ou seja, a rede terá como limite máximo a rajada de 2900. Assim, esse limite de 2900 é próximo ao limite máximo estabelecido pela taxa de reposição de *tokens* ao *bucket* de 2700 da Figura 5.4. Portanto, os resultados obtidos para os diferentes tráfegos não sofrem grandes mudanças quando a taxa de reposição de *tokens* ao *bucket* é igual a 2700 ou 3900, e a rajada é igual a 2900. Vale ressaltar que a variação no caso do tráfego Guepardo é 0% já que em todos os testes este atingiu 100% de transmissões dos pacotes recebidos.

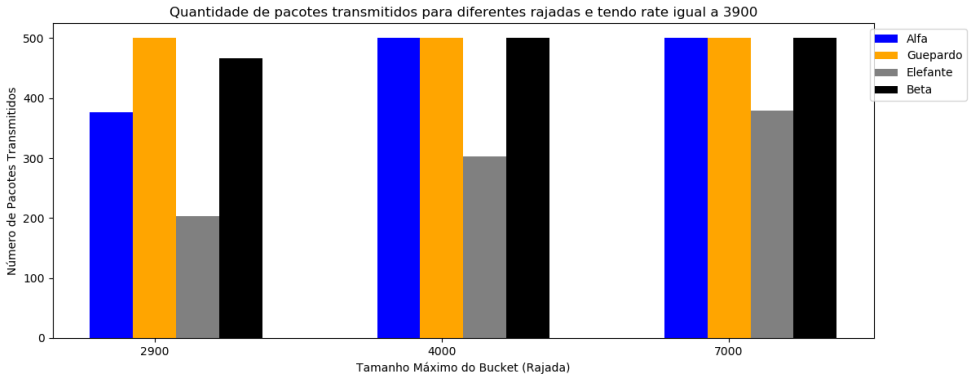


Figura 5.5: Transmissões realizadas pelo srTBP alternando a rajada e *rate* igual a 3900 para os diferentes tipos de tráfego.

Para os experimentos de *throughput* máximo, foram escolhidos para a rajada os valores de 700 mil, 900 mil e 1,1 milhões de *tokens*, e com taxa de reposição de *tokens* ao *bucket* igual a 1,1 milhões. Os resultados obtidos são apresentados na Figura 5.6. Após análise, é possível verificar que para a rajada igual a 700 mil *tokens*, o fluxo é limitado, visto que o valor da rajada é menor que a taxa de reposição de *tokens* ao *bucket*. Assim, atinge-se, em média, a transmissão de 6,7 Mbps. Para a rajada igual a 900 mil, é obtida, em média, a taxa de transmissão de 8,6 Mbps. Já para a rajada igual a 1,1 milhão, mesmo valor que a reposição de *tokens* ao *bucket*, é

obtido: 10 Mbps, sendo transmitidos todos os pacotes do tráfego. Os resultados obtidos para o tráfego de 10 Mbps mostram que o comportamento do srTBP para esse fluxo é semelhante ao encontrado nos experimentos realizados para os tráfegos Alfa, Beta, Elefante e Guepardo. Fato que mostra uma uniformidade no comportamento do mecanismo.

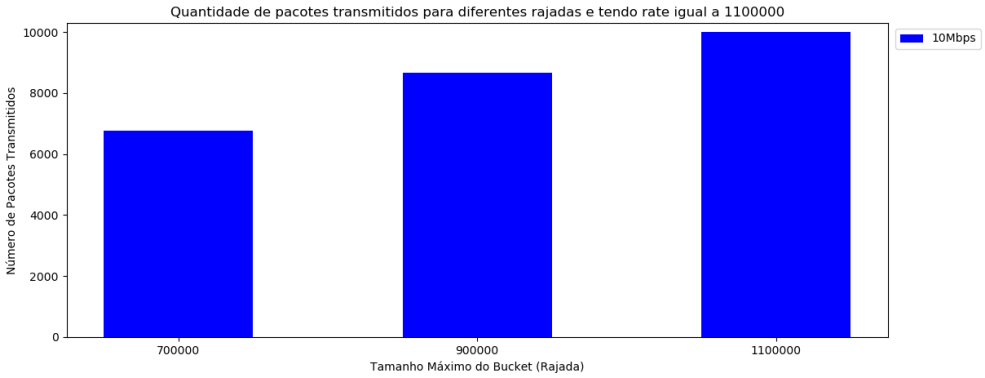


Figura 5.6: Transmissões realizadas pelo srTBP para o tráfego de 10Mbps.

Destarte, ao analisar todos os experimentos em conjunto é possível verificar os impactos gerados pelos dois parâmetros, a rajada e a taxa de reposição de *tokens* ao *bucket*, no comportamento do srTBP. Quanto aos valores das rajadas, percebe-se que aumentar a rajada pode gerar o aumento no número de transmissões realizadas pelo srTBP, mas não necessariamente evita que o mecanismo descarte pacotes dependendo do tempo de execução e número de pacotes recebidos. A exemplo, se a rajada for menor que a taxa de reposição de *tokens* ao *bucket* e a quantidade de *bytes* transmitidos por segundo pelo tráfego, é assegurado que a rede não ultrapassa o limite definido pela rajada, visto que a mesma indica a quantidade máxima de *tokens* disponíveis por segundo no *bucket*, realizando o descarte de pacotes que o excedam. O mesmo comportamento ocorre quando a taxa de reposição de *tokens* ao *bucket* tem valor inferior ao da rajada e da quantidade de *bytes* transmitidos por segundo pelo tráfego. Sendo assim, o tráfego não ultrapassa essa taxa máxima definida após um período, visto que o valor estabelecido para a rajada pode retardar o esvaziamento do *bucket*, fazendo com que seja necessário o recebimento de mais *bytes* para esvaziá-lo.

5.2.2 *Single Rate Three Color Marker Policer*

O srTCM-*Policer* utiliza dois *token buckets*, denominados de C e E, para verificar a conformidade dos pacotes. Desta forma, os experimentos no srTCM-*Policer* foram realizados alterando os valores de três parâmetros, sendo: o CIR, o CBS e o EBS. O primeiro define a quantidade de *tokens* que deve ser adicionada a ambos os *buckets* a cada intervalo de tempo. Enquanto os outros, CBS e EBS, representam as quantidades máximas de *tokens* suportados pelos *buckets* C e E, respectivamente. Os diferentes valores para os parâmetros foram escolhidos baseando-se na quantidade de *bytes* enviados por cada perfil de tráfego no intervalo de 1 segundo. Desta forma, foram selecionados os seguintes valores para o CIR: 1500, 2700 e 3900 *tokens*. Enquanto para o CBS foram selecionados os seguintes valores: 2900, 4000 e 7000 *tokens*. Já para o EBS foram selecionados: 2000, 3500 e 5500 *tokens*.

Na Figura 5.7 são apresentados os resultados obtidos executando o srTCM-*Policer*, para diferentes valores de EBS e com o CIR e CBS iguais a 1500 e 4000, respectivamente. É possível verificar que para os tráfegos Alfa, Beta e Guepardo, há a transmissão de 100% dos pacotes recebidos pelo mecanismo, independentemente dos valores de EBS adotados. Por outro lado,

para o tráfego Elefante são obtidos 49,8%, 58,8% e 59% de transmissão dos pacotes recebidos, quando EBS é igual a 2000, 3500 e 5500, respectivamente. Isso ocorre, pois, apesar de existirem dois *token buckets*, os valores de EBS e CBS, em conjunto com o CIR, ainda são menores que a quantidade de *bytes* enviados por segundo pelo tráfego Elefante.

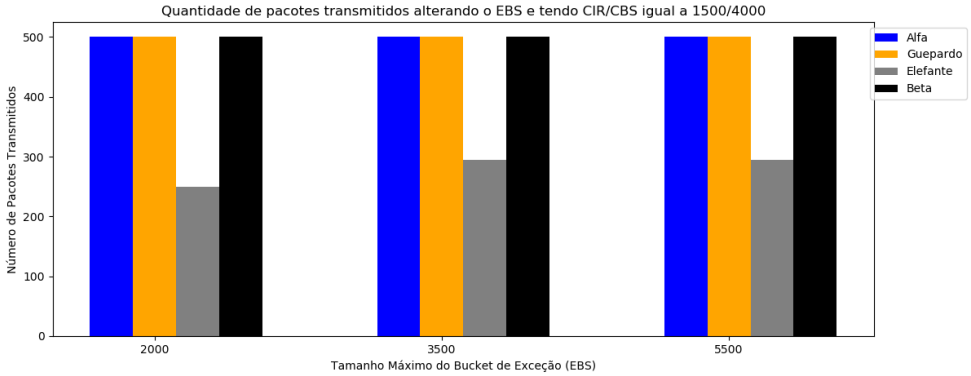


Figura 5.7: Transmissões realizadas pelo srTCM-*Policer* alterando o EBS e com CIR e CBS igual a 1500 e 4000, respectivamente, para os diferentes tipos de tráfego.

Entretanto, apesar dos tráfegos Alfa e Beta atingirem 100% de transmissões, a ação tomada nem sempre é a ação verde. Fato que acontece, visto que o CBS, com quantidade máxima de *tokens* do *bucket* C, em conjunto com a CIR, serem menores que a demanda de *bytes* do tráfego, fazendo com que o tamanho pacote seja comparado à quantidade de *tokens* do *bucket* E, que para o tráfego Alfa e Beta são suficientes para a demanda da rede. Desta forma, além de ações verdes, são tomadas também as ações amarelas, que indicam a conformidade do pacote com o *bucket* E. Para o tráfego Alfa, em média, 17,1% das ações realizadas foram de cor amarela, e 82,9% foram ações verdes. Para o tráfego Beta, em média, 28,5% e 71,5% das ações realizadas tem cor amarela e verde, respectivamente. Para o tráfego Guepardo 100% das transmissões foram ações verdes. Para o Elefante, em média, 44,1%, 26,2% e 29,7% das ações realizadas pelo mecanismo tem cor vermelha, amarela e verde, respectivamente. Na Figura 5.8 são mostradas a quantidade de ações verdes, amarelas e vermelhas tomadas pelo srTCM para os tráfegos Alfa, Beta e Elefante, utilizando CIR e CBS iguais a 1500 e 4000, respectivamente, além de alterar os valores do EBS em 2000, 3500 e 5500. O srTCM-*Policer* transmitiu todos os pacotes do tráfego Guepardo realizando ações verdes.

Os resultados experimentais obtidos na execução do mecanismo srTCM-*Policer* na rede, alterado o EBS e com o CIR e CBS iguais a 2700 e 2900, respectivamente, podem ser visualizados na Figura 5.9. Nesse experimento, os resultados obtidos para o tráfego Elefante mostram um aumento, em média, de 25,7% no número de transmissões de pacotes à medida que o valor de EBS aumenta. Este resultado ocorre porque quanto maior o EBS, mais ações amarelas são tomadas. Ainda assim, pacotes são descartados, indicando que o CIR é menor que a demanda de *bytes* do tráfego. Já para os tráfegos Guepardo e Beta são atingidos 100% de transmissão, independente dos valores dos parâmetros. Quanto ao tráfego Alfa, há um pequeno descarte de pacotes quando o EBS é igual a 2000 *tokens*, atingindo 97,4% de pacotes transmitidos, enquanto para o EBS igual a 3500 e 5500 *tokens*, obtêm-se 100% de pacotes transmitidos. A quantidade de ações verdes, amarelas e vermelhas realizadas para cada perfil de tráfego, alterado o EBS e com o CIR e CBS iguais a 2700 e 2900 *tokens*, respectivamente, são mostrados na Figura 5.10.

A seguir, a Figura 5.11 apresenta os resultados experimentais obtidos na execução do mecanismo srTCM-*Policer* na rede, alterado o EBS e com o CIR e CBS iguais a 3900 e 7000 *tokens*, respectivamente. Neste caso, apenas o tráfego Elefante, quando o EBS é igual a 2000,

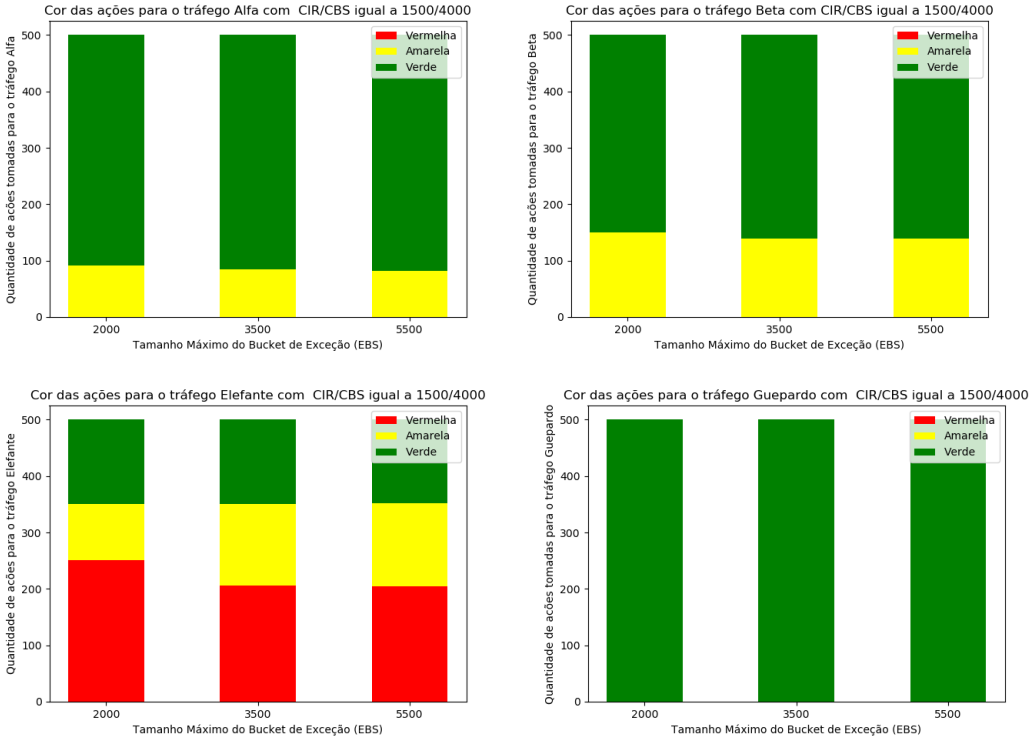


Figura 5.8: Gráficos indicando a quantidade de ações verdes, amarelas e vermelhas tomadas pelo srTCM-*Policer* para os diferentes tráfegos.

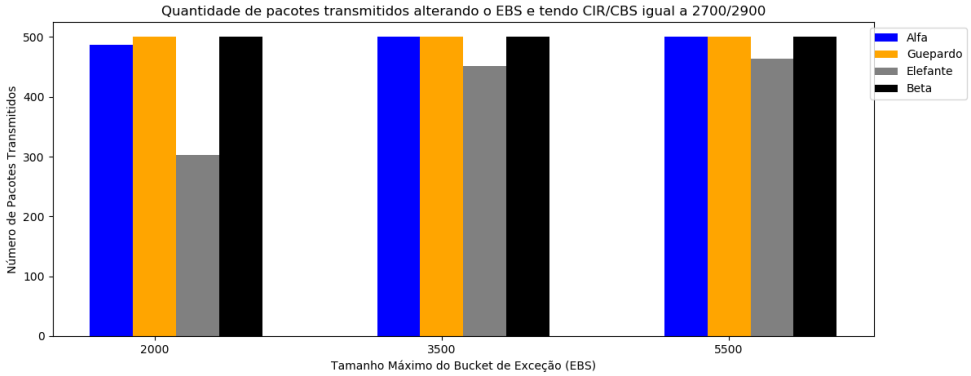


Figura 5.9: Transmissões realizadas pelo srTCM-*Policer* alterando o EBS e com CIR e CBS igual a 2700 e 2900, respectivamente, para os diferentes tipos de tráfego.

não atingiu o 100% de transmissão, tendo 95% de pacotes transmitidos. Entretanto, assim como ocorre no srTBP o descarte de pacotes pode ser apenas retardado. Para os tráfegos Alfa, Beta e Guepardo, 100% dos pacotes tiveram a ação verde realizada. Para o tráfego Elefante, quando o EBS é igual a 2000 *tokens*, obtêm-se 5%, 19,4% e 75,6% de ações vermelhas, amarelas e verdes, respectivamente. Já quando o EBS é igual a 4000 ou 7000, tem-se 24,2% de ações amarelas realizadas e 75,8% de ações verdes realizadas pelo mecanismo. Na Figura 5.12 é mostrada a quantidade de ações verdes, amarelas e vermelhas realizadas para cada perfil de tráfego, alterado o EBS e com o CIR e CBS iguais a 3900 e 7000, respectivamente.

Após análises dos experimentos, é possível visualizar os impactos gerados pelo aumento ou diminuição de cada um dos parâmetros no comportamento do srTCM-*Policer*. Também é possível verificar as correlações entre os parâmetros. Por exemplo, se o valor estipulado para

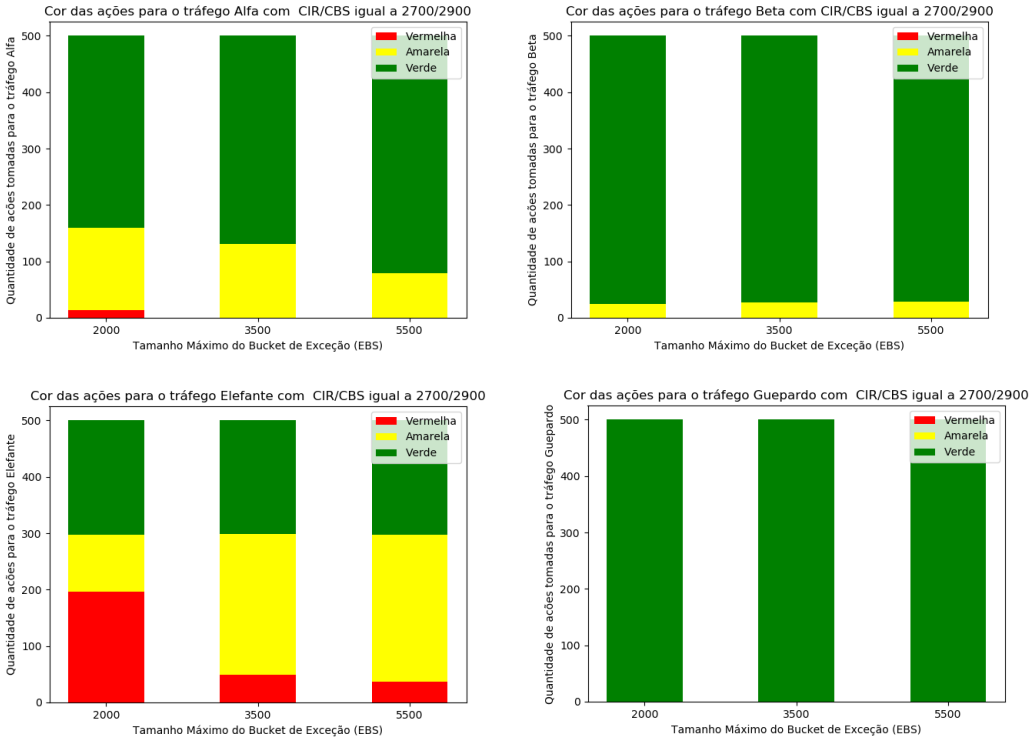


Figura 5.10: Gráficos indicando a quantidade de ações verdes, amarelas e vermelhas tomadas pelo srTCM-*Policer* para os diferentes tráfegos.

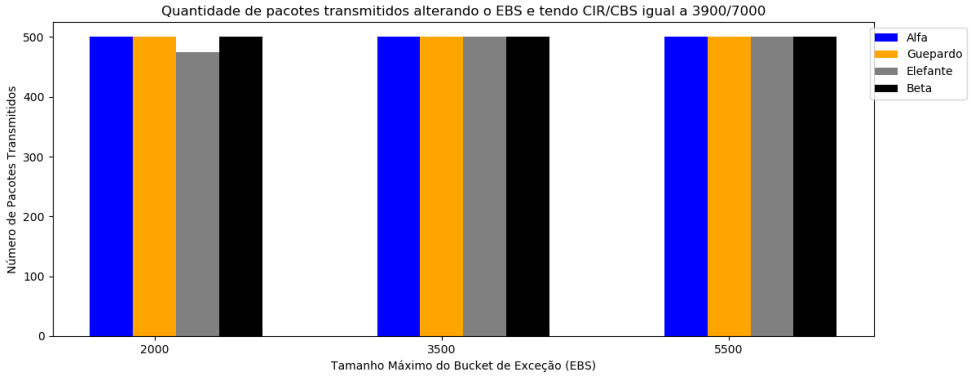


Figura 5.11: Transmissões realizadas pelo srTCM-*Policer* alterando o EBS e com CIR e CBS igual a 3900 e 7000, respectivamente, para os diferentes tipos de tráfego.

o CBS for menor que a quantidade de *bytes* transmitidos no fluxo por segundo, mas o EBS e CIR forem maiores, a quantidade de ações amarelas aumenta com relação ao número de ações verdes realizadas. Já se o CBS e CIR forem maiores, ocorre, em geral, a realização de ações verdes. Portanto, no srTCM-*Policer*, só há o descarte de pacotes quando a quantidade de *bytes* transmitidos no tráfego por segundo exceder os *buckets* C e E, limitando a rede aos valores dos parâmetros CBS, EBS e CIR.

5.2.3 *Two Rate Three Color Marker Policer*

O trTCM-*Policer*, assim como o srTCM-*Policer*, utiliza *token buckets* para verificar a conformidade dos pacotes, denominados, C e P. Entretanto, no trTCM-*Policer* cada *bucket* tem

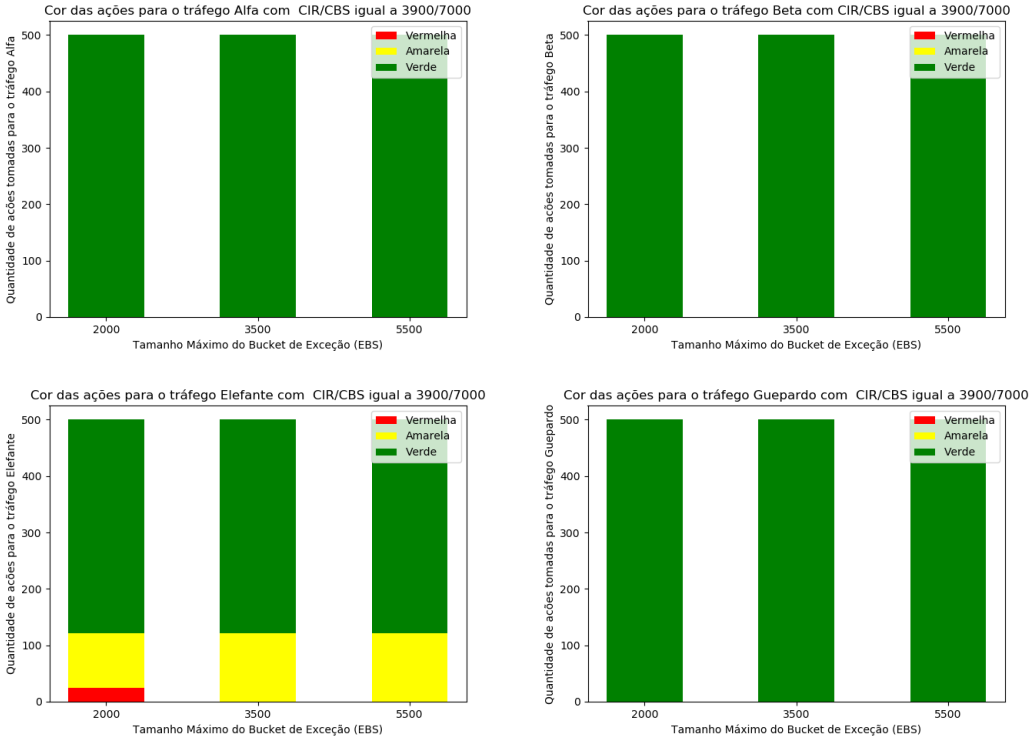


Figura 5.12: Gráficos indicando a quantidade de ações verdes, amarelas e vermelhas tomadas pelo srTCM-*Policer* para os diferentes tráfegos.

sua própria taxa de reposição de *tokens* ao *bucket*, a CIR e PIR, respectivamente. A quantidade máxima de *tokens* suportados por cada *bucket* também é definida, denominadas CBS e PBS, respectivamente. Os diferentes valores para os parâmetros foram escolhidos baseando-se na quantidade de *bytes* enviadas por cada perfil de tráfego no intervalo de 1 segundo. Desta forma, foram selecionados para o CIR os valores 1500, 2700 e 3900 *tokens*, e para o PIR os valores 1200, 2300 e 3400 *tokens*. Enquanto para o CBS foram selecionados os seguintes valores: 2900, 4000 e 7000 *tokens*. Já para o PBS foram selecionados: 2000, 3500 e 5500 *tokens*.

Um fato importante a se destacar é que no funcionamento do trTCM-*Policer* o pacote, ao ser recebido, tem seu tamanho em *bytes* comparado à quantidade de *tokens* presentes no *bucket* P, onde se não houver *tokens* suficientes, a ação vermelha é tomada. Se houverem *tokens* suficientes, compara-se então o tamanho do pacote à quantidade de *tokens* do *bucket* C, se não forem suficientes, decrementa-se a quantidade de *tokens* de P referentes ao tamanho do pacote e a ação amarela é tomada. Entretanto, se forem suficientes, a ação verde é realizada e ambos os *buckets*, C e P, são decrementados pelo tamanho do pacote.

Na Figura 5.13 são apresentados os resultados obtidos executando o trTCM-*Policer*, para os diferentes valores de PBS, e com CIR, PIR e CBS iguais a 1500, 3400 e 2900 *tokens*, respectivamente. Ao analisar os resultados, é possível verificar que o valor escolhido de 2000 *tokens* para o PBS é determinante para o menor número de transmissões realizadas pelo trTCM-*Policer* se comparados ao PBS igual 3500 *tokens*, tendo um aumento de 63,4%, 30,5% e 196% no número de transmissões realizadas para os tráfegos Alfa, Beta e Elefante, respectivamente. No trTCM-*Policer*, se o pacote não estiver em conformidade com o *bucket* P, a ação de descarte é realizada. Assim, ao aumentar o valor do PBS de 2000 para 3500 *tokens*, verifica-se esse aumento na quantidade de transmissões realizadas pelo trTCM. Para o tráfego Guepardo, atinge-se 100% de transmissões tanto para EBS igual a 2000 *tokens*, quanto para EBS igual a 3500.

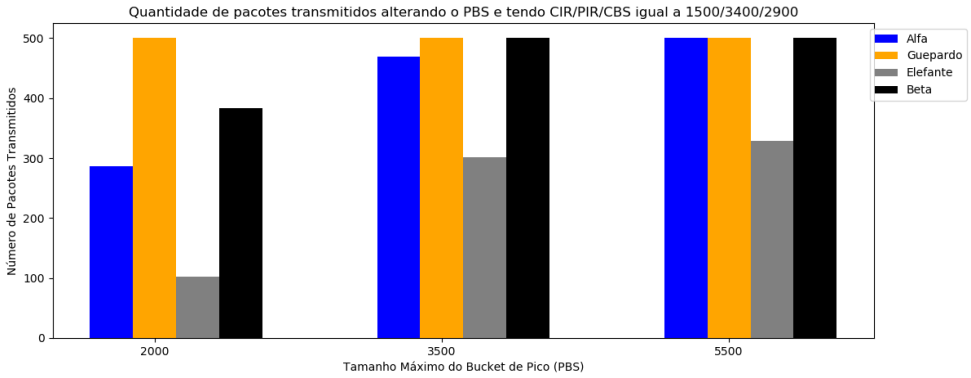


Figura 5.13: Transmissões realizadas pelo trTCM-*Policer* alterando o PBS e CIR, PIR e CBS igual a 1500, 3400 e 2900, respectivamente, para os diferentes tipos de tráfego.

Na Figura 5.14 é mostrada a quantidade de ações verdes, amarelas e vermelhas realizadas na execução do trTCM-*Policer* para cada perfil de tráfego, para os diferentes valores de PBS, e com CIR, PIR e CBS iguais a 1500, 3400 e 2900 *tokens*, respectivamente. Analisando os resultados obtidos é constatado que quando o PBS é igual a 2000 *tokens*, para os tráfegos Alfa e Elefante não são realizadas ações amarelas, tendo para o Alfa, 42,6% de ações vermelhas e 57,4% de ações verdes. Já para o tráfego Elefante existem 79,6% de ações vermelhas e 20,4% de ações verdes. Isto se deve ao fato de que o valor de CBS é superior ao de PBS, sendo assim, quando o pacote está em conformidade com P, também está com C. Entretanto, isto não necessariamente ocorre sempre, depende também do tamanho do pacote e das taxas PIR e CIR, como pode ser verificado nos resultados obtidos para o tráfego Beta, caso em que foi obtido 23,4% de ações vermelhas, 13,8% de ações amarelas e 62,8% de ações verdes realizadas. Para o tráfego Guepardo obteve-se 100% de ações verdes realizadas.

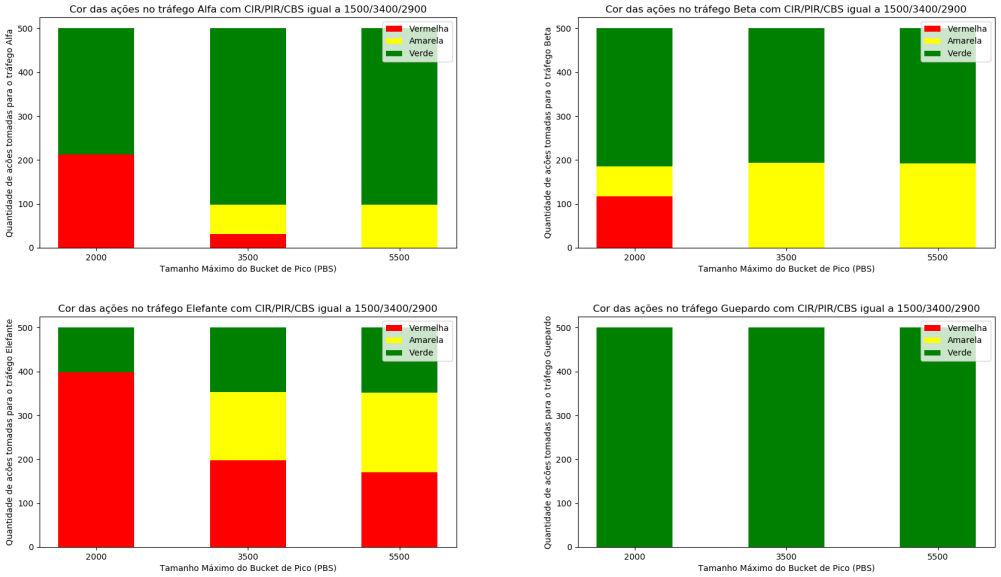


Figura 5.14: Gráficos indicando a quantidade de ações verdes, amarelas e vermelhas tomadas pelo trTCM-*Policer* para os diferentes tráfegos.

Na Figura 5.15 são apresentados os resultados obtidos para o trTCM-*Policer*, utilizando os diferentes perfis de tráfego, para os diferentes valores de PBS, e com CIR, PIR e CBS iguais a 3900, 1200 e 7000 *tokens*, respectivamente. Analisando as informações apresentadas, é possível

verificar que apesar de os valores de CIR e CBS serem, teoricamente, maiores que a demanda de *bytes* por segundo dos tráfegos, ocorre um valor baixo no número de transmissões realizadas, tendo, em média, 56,1%, 57,4% e 22,8% dos pacotes transmitidos para os tráfegos Alfa, Beta e Elefante, respectivamente. Isso ocorre visto que um PIR igual a 1200 *tokens* é menor que a quantidade de *bytes* enviados por segundo por esses tráfegos, fazendo com que pacotes excedam o *bucket* P, sendo realizada a ação vermelha. Para o tráfego Guepardo 100% dos pacotes são transmitidos, independente do valor atribuído para PBS.

Já na Figura 5.16 é apresentada a quantidade de ações verdes, amarelas e vermelhas realizadas na execução do trTCM-*Policer* para cada perfil de tráfego, para os diferentes valores de PBS, e com CIR, PIR e CBS iguais a 3900, 1200 e 7000 *tokens*, respectivamente. É possível verificar que não foram realizadas ações amarelas nenhuma vez para os tráfegos Alfa, Beta e Elefante, independente do valor de PBS, visto que o valor estipulado para o PIR é menor que CIR, CBS e a quantidade de *bytes* transmitidos por segundo nesses tráfegos. Desta forma, quando o pacote está conforme a P, o mesmo também está conforme a C, pois C recupera a quantidade de *tokens* mais rápido que P, visto que CIR é maior que PIR. Assim, são realizadas somente ações verdes ou vermelhas pelo trTCM-*Policer*.

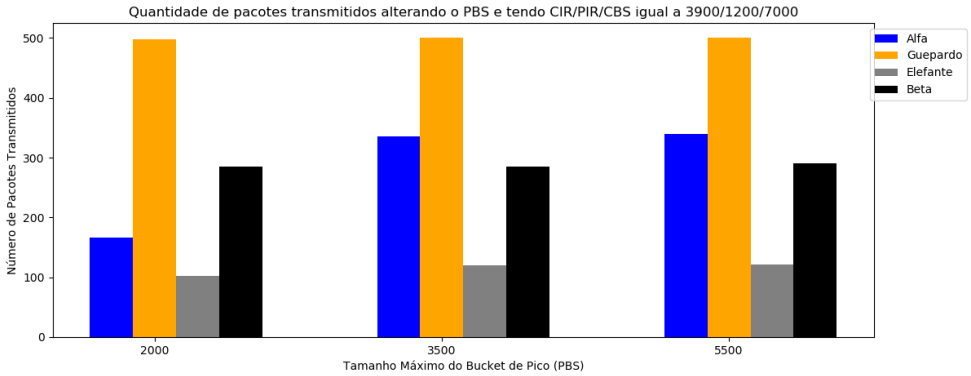


Figura 5.15: Transmissões realizadas pelo trTCM-*Policer* alterando o PBS e CIR, PIR e CBS igual a 3900, 1200 e 7000, respectivamente, para os diferentes tipos de tráfego.

Os resultados obtidos nos experimentos realizados no trTCM-*Policer* utilizando diferentes valores para o PBS e sendo CIR, PIR e CBS iguais a 2700, 2300 e 4000 *tokens*, respectivamente, podem ser visualizados na Figura 5.17. Após analisar os resultados é possível verificar que o aumento no valor do PBS impacta diretamente no número de transmissões realizadas pelo trTCM-*Policer* para o tráfego Alfa, tendo, em média, um aumento de 34,7%. Já para os tráfegos Elefante e Beta, há um aumento de 23,1% e 128,4%, respectivamente, no número de transmissões ao se comparar os resultados quando o PBS é igual a 2000 e 3500 *tokens*. Entretanto, esse aumento é quase imperceptível ao se comparar os resultados quando o PBS é igual a 3500 e 5500 *tokens*, sendo de 1,2% e 0,8%, respectivamente. Isso indica que aumentar o valor de PBS para mais de 3500 tem um menor impacto no número de transmissões realizadas. Já para o tráfego Guepardo, obteve-se 100% dos pacotes transmitidos, independente do valor de PBS.

Na Figura 5.18 é apresentada a quantidade de ações verdes, amarelas e vermelhas realizadas na execução do trTCM, alterando o valor de PBS, e tendo CIR, PIR e CBS iguais a 2700, 2300 e 4000 *tokens*, respectivamente. Analisando os resultados, nota-se que para o tráfego Alfa, quando PBS é igual a 5500, são realizadas apenas ações verdes e amarelas, diferentemente de quando PBS tem outros valores, para os quais são realizadas apenas ações verdes e vermelhas. Isso ocorre para o tráfego Alfa já que este tem 2 segundos de não transmissão, desta forma,

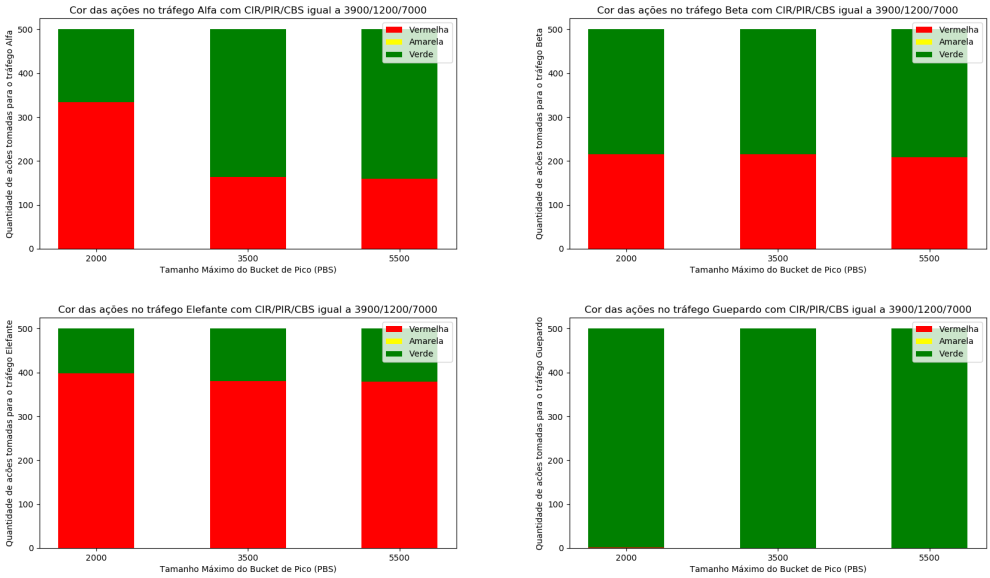


Figura 5.16: Gráficos indicando a quantidade de ações verdes, amarelas e vermelhas tomadas pelo trTCM-*Policer* para os diferentes tráfegos.

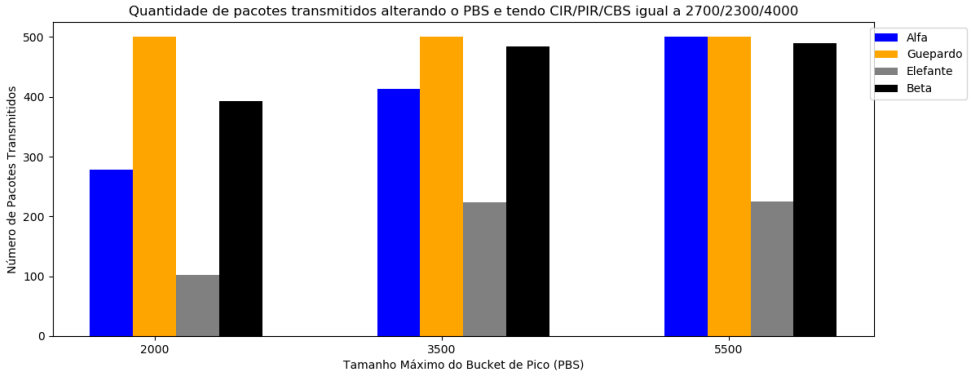


Figura 5.17: Transmissões realizadas pelo trTCM-*Policer* alterando o PBS e CIR, PIR e CBS igual a 2700, 2300 e 4000, respectivamente, para os diferentes tipos de tráfego.

tanto P quanto C recebem *tokens* de reposição, sem haver o consumo dos mesmos nesse período. Portanto, no caso de PBS ser 5500 *tokens*, apesar de CIR ser maior que PIR, o CBS é menor que o PBS, fazendo com que ao final dos 2 segundos de não transmissão o PBS tenha maior disponibilidade de *tokens*, realizando assim ações amarelas, pois C esvazia antes que P. Porém, nos outros casos, o PBS é menor que CBS, além de ter valor menor que a quantidade de *bytes* transmitidos em 1 segundo pelo tráfego Alfa. Desta maneira, P se esvazia antes de C, realizando as ações vermelhas. Quanto aos tráfegos Beta e Elefante, para todos os valores de PBS, o PIR ainda é menor que CIR e a quantidade de *bytes* transmitidos por segundo nesses fluxos, fazendo com que apenas ações verdes e vermelhas sejam realizadas pelo trTCM-*Policer*. Já para o Guepardo, é obtido 100% de ações verdes realizadas, independente do valor de PBS.

Após análise de todos os experimentos realizados para o trTCM-*Policer* é possível verificar a importância dos valores de PIR e PBS para o número de transmissões realizadas pelo mecanismo. O PIR e PBS tem impacto na quantidade de pacotes transmitidos, pois se o pacote recebido não estiver em conformidade com P, a ação vermelha é realizada, ou seja, o pacote é descartado. Já os valores de CIR e CBS, em conjunto com o PIR e PBS, impactam

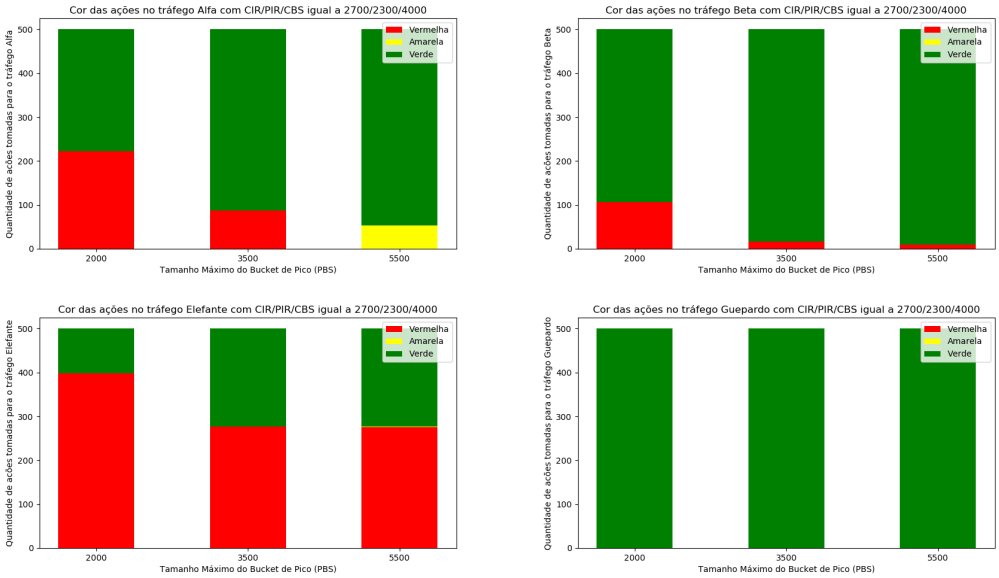


Figura 5.18: Gráficos indicando a quantidade de ações verdes, amarelas e vermelhas tomadas pelo trTCM-*Policer* para os diferentes tráfegos.

na quantidade de ações verdes ou amarelas realizadas pelo trTCM-*Policer*. Sendo assim, pelos experimentos é possível notar que as ações amarelas são realizadas apenas quando CIR e CBS são menores que os valores de PIR e PBS, de forma que no recebimento de um pacote, P tem *tokens* suficientes no *bucket* e C tem *tokens* insuficientes no *bucket*. Por fim, ao executar os testes para o trTCM-*Policer*, encontrar o equilíbrio entre os parâmetros se mostra complexo em relação aos outros mecanismos, fato que se deve à maior quantidade de parâmetros e de combinações de valores para os mesmos.

5.2.4 *Color-Aware Policers*

Os CAP recebem um tráfego de pacotes pré-coloridos, e analisam os mesmos baseando-se em suas cores e sua conformidade com o mecanismo. Desta forma, os CAP funcionam como um mecanismo de confiança, verificando se a coloração dos pacotes, realizada por um mecanismo de *marking*, está nos conformes da rede. Tanto o srTCM-*Policer*, quanto o trTCM-*Policer*, utilizando as mesmas definições, podem ser usados como CAPs, sendo denominados de srTCM-PCA e trTCM-PCA, os quais reconhecem marcações de cores pré-definidas nos pacotes. Ressalta-se ainda que, na definição clássica, um pacote pode ser pré-colorido com uma de três cores: verde, amarela e vermelha. É importante observar que o srTCM-CA e o trTCM-CA podem trocar a cor da ação dos pacotes, ou seja, mudar a cor dos mesmos, apenas se for para abaixar as suas prioridades. Por exemplo, pacotes amarelos não podem se tornar verdes, mesmo que fosse possível pela quantidade de *tokens* nos *buckets*, mas pacotes verdes podem se tornar amarelos havendo indisponibilidade *tokens*.

Os experimentos realizados no srTCM-PCA, foram realizados da seguinte forma: o mecanismo recebe um dos tráfegos (Alfa, Beta, Elefante ou Guepardo), sendo que os pacotes foram previamente coloridos nas cores verde, amarela ou vermelha por um *marker*, que nesse caso é o srTCM. Desta forma, se o srTCM e o srTCM-PCA tiverem os mesmos valores para seus parâmetros, o comportamento de ambos ao analisar o tráfego será igual, ou seja, o srTCM-PCA realizará as ações das respectivas cores do pacote, não havendo mudanças.

Os resultados obtidos na execução do srTCM-PCA tendo os mesmos parâmetros do srTCM, ou seja, alterando o EBS e com os valores de CIR e CBS iguais a 2700 e 2900 *tokens*,

respectivamente, podem ser visualizados na Figura 5.19. Após analisar os resultados é possível verificar que estes são os mesmos obtidos para o srTCM-*Policer* com mesmos parâmetros (Figura 5.9). Essa igualdade ocorre visto que a diferença entre o srTCM e srTCM-*Policer* é definida no contexto em que o primeiro ao invés de realizar a ação correspondente, realiza a marcação do pacote, e como o srTCM-PCA é um mecanismo de confiança e tem o mesmo comportamento, os resultados entre todos esses mecanismos são iguais. Na Figura 5.20 é possível visualizar os gráficos remetentes à quantidade de pacotes recebidos de cada coloração e à quantidade de ações de cada cor realizadas pelo srTCM-PCA para os tráfegos Alfa, Beta, Elefante e Guepardo. A quantidade de pacotes de cada cor corresponde ao número de ações executadas para cada cor, visto que o srTCM-PCA tem comportamento idêntico ao *marker* que realizou a marcação dos pacotes.

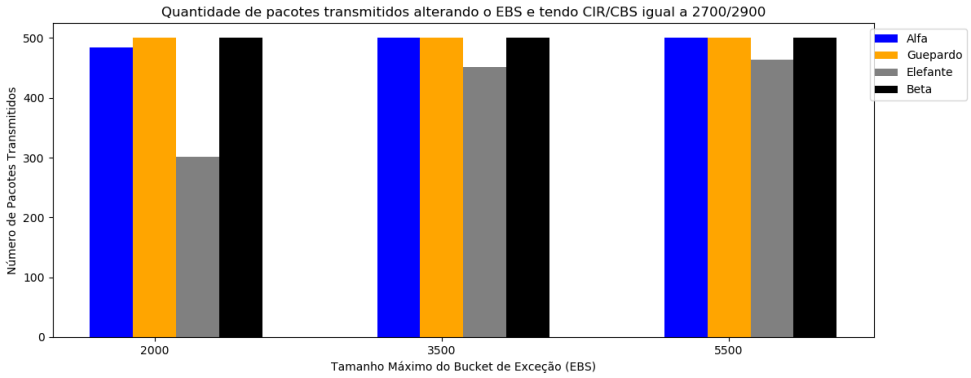


Figura 5.19: Transmissões realizadas pelo srTCM-PCA alterando o EBS e com CIR e CBS iguais a 2700 e 2900, respectivamente, para os diferentes tipos de tráfego.

Outro experimento foi realizado assumindo que o srTCM estava com problemas e colorindo erroneamente os pacotes, elevando a quantidade de pacotes verdes e amarelos na rede, fato que, nesse experimento, não corresponderia com os conformes da rede. Para o srTCM, assumem-se os valores de CIR e CBS iguais a 3900 e 7000 *tokens*, respectivamente, enquanto para o srTCM-PCA os valores são de 1500 e 4000 *tokens*, respectivamente. Na Figura 5.21 é apresentado o número de transmissões realizadas pelo srTCM-PCA para os diferentes tráfegos, sendo o parâmetro EBS alterado entre 2000, 3500 e 5500 *tokens*, enquanto o CIR e CBS são iguais a 1500 e 4000 *tokens*, respectivamente.

É interessante notar que os resultados obtidos neste experimento são iguais aos da Figura 5.7, que apresenta os resultados para o srTCM-*Policer*, com os mesmos valores para os parâmetros CIR e PIR. Isso indica que o srTCM-PCA não realizou as ações pela cor do pacote, pois o tráfego estava marcado com cores que extrapolavam os conformes da rede, sendo assim, realizaram-se mudanças nas cores das ações. Desta forma, ele readéqua o tráfego para os limites estipulados pelo srTCM-PCA. A quantidade de mudanças entre a cor pré-definida e a cor da ação realizada para os tráfegos Alfa, Beta e Elefante é apresentada na Figura 5.22. Já a quantidade de pacotes de cada cor recebidos e das ações realizadas pelo srTCM-CA, sendo alterado o EBS e com CIR e CBS iguais a 1500 e 4000, respectivamente, podem ser visualizados na Figura 5.23.

Para o tráfego Beta, quando EBS é igual a 2000, 3500 e 5500 *tokens*, 100% dos pacotes recebidos tinham cor verde, mas 34,8%, 36,8% e 36,8%, respectivamente, dos pacotes que chegaram ao mecanismo com a coloração verde, tiveram a ação amarela realizada. Para o tráfego Alfa, para todos os valores de EBS, chegaram ao mecanismo todos os pacotes pré-determinados com a cor verde, mas, quando EBS é igual a 2000, 3500 e 5500 de 1500 e 4000 *tokens*, respectivamente, 16,4%, 16,4% e 16,8% dos pacotes que chegaram ao mecanismo com a

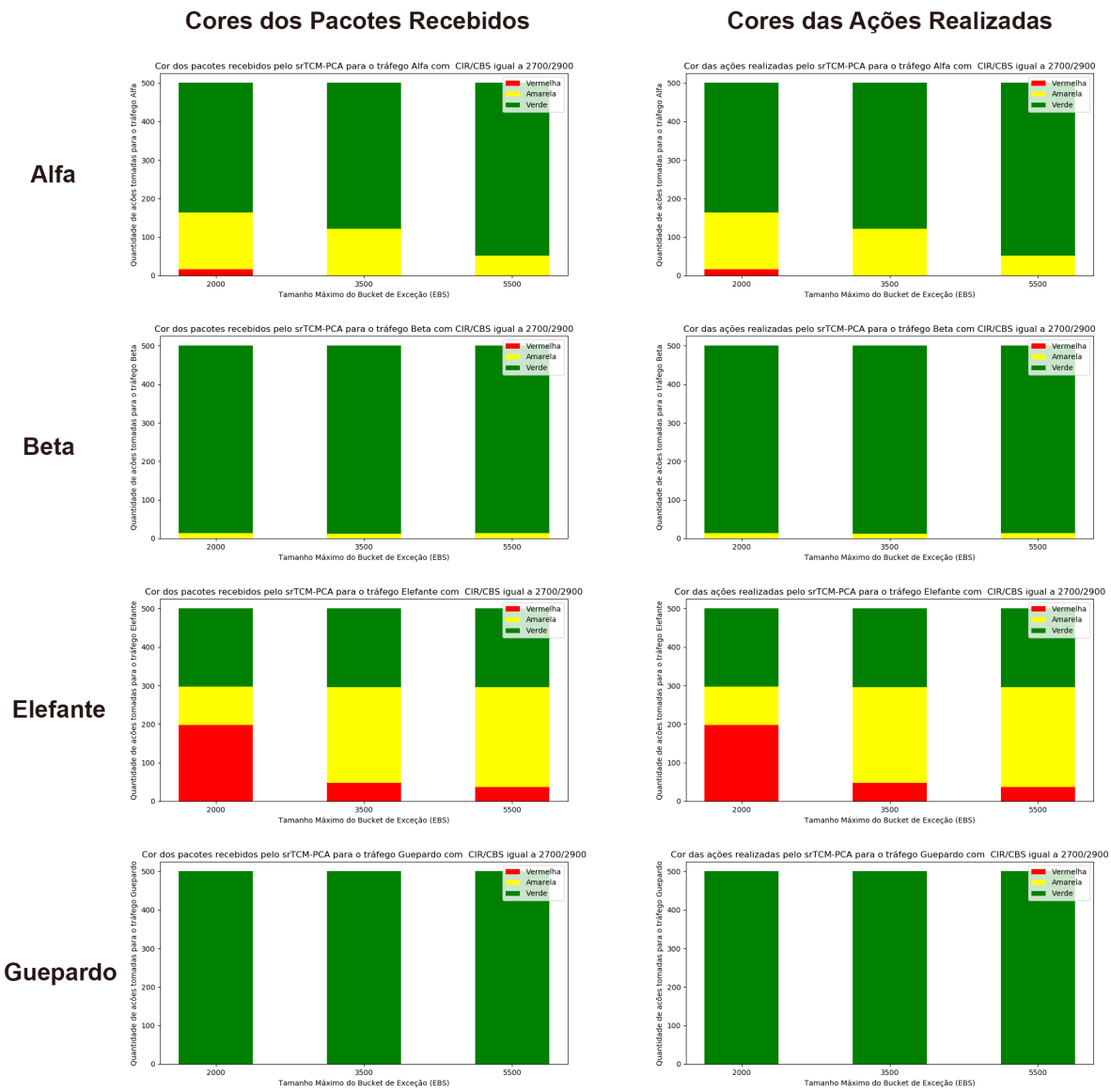


Figura 5.20: Comparação entre as cores recebidas e as cores das ações realizadas para o srTCM-PCA, tendo o valor de EBS alterado, e CIR e CBS iguais a 2700 e 2900, respectivamente.

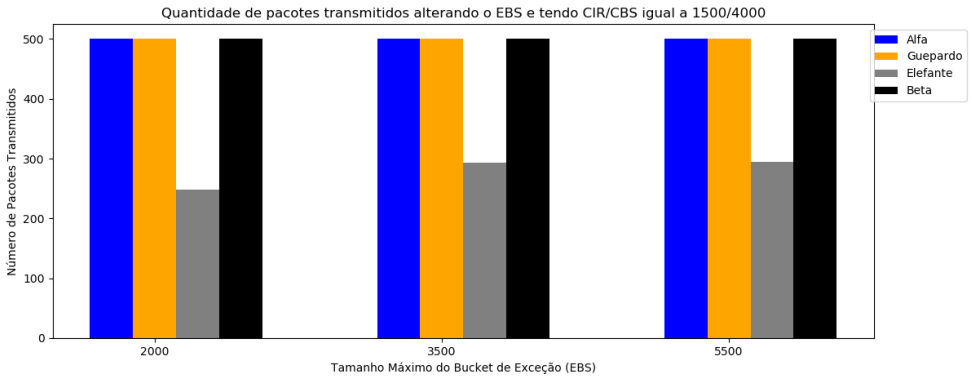


Figura 5.21: Transmissões realizadas pelo srTCM-PCA alterando o EBS e com CIR e CBS iguais a 1500 e 4000, respectivamente, para os diferentes tipos de tráfego.

coloração verde, tiveram a ação amarela realizada. Para o tráfego Guepardo, não houve diferenças entre a cor pré-determinada e a cor da ação realizada, mesmo para os diferentes valores de EBS.

Já para o tráfego Elefante, com o EBS igual a 2000 *tokens*, 24,4% dos pacotes recebidos tinham coloração amarela e 75,6% tinham a coloração verde, sendo que para 100% dos pacotes que chegaram ao mecanismo marcados de amarelo, a ação realizada para os mesmos foi a vermelha. Para os pacotes que chegaram marcados de verde, 26,2% tiveram a ação amarela realizada, 34,4% tiveram a ação vermelha realizada, e 39,4% mantiveram a ação verde. Para o EBS igual a 3500 *tokens*, 24,2% dos pacotes recebidos tinham coloração amarela e 75,8% tinham a coloração verde, sendo que para 93,4% dos pacotes que chegaram ao mecanismo marcados de amarelo, a ação realizada para os mesmos foi a vermelha, e 6,6% mantiveram a cor e realizaram a ação amarela. Já os pacotes que chegaram marcados de verde, 35,8% tiveram a ação amarela realizada, 24,8% tiveram a ação vermelha realizada, e 39,4% mantiveram a ação verde. Para o EBS igual a 5500 *tokens*, 24,2% dos pacotes recebidos tinham coloração amarela e 75,8% tinham a coloração verde, sendo que 93,4% dos pacotes que chegaram ao mecanismo marcados de amarelo, a ação realizada para os mesmos foi a vermelha, e 6,6% mantiveram a cor e realizaram a ação amarela. Já os pacotes que chegaram marcados de verde, 36,1% tiveram a ação amarela realizada, 24,3% tiveram a ação vermelha realizada, e 39,6% mantiveram a ação verde.

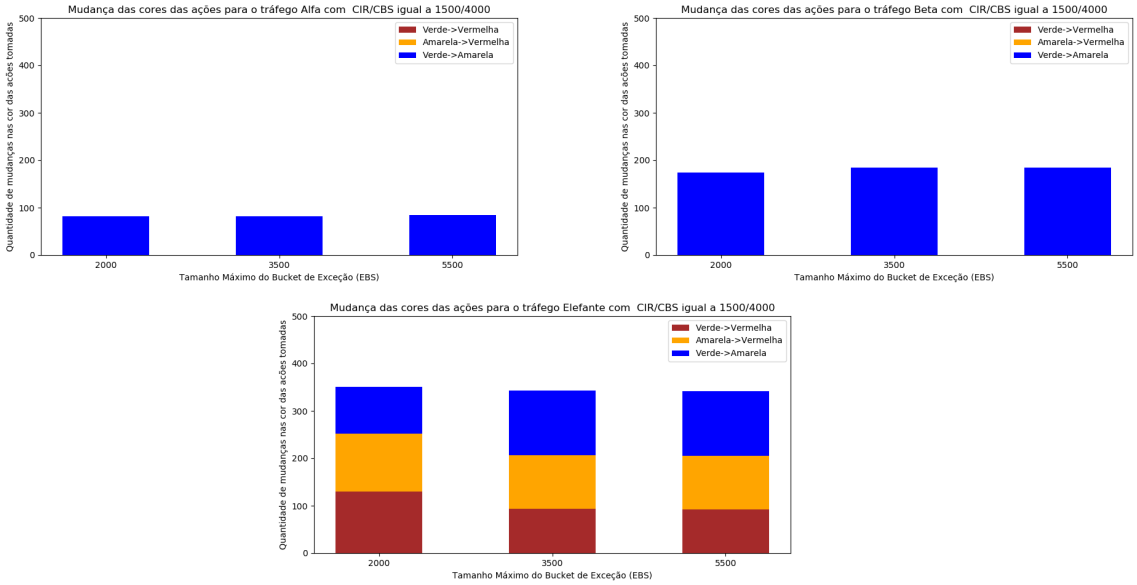


Figura 5.22: Quantidade de mudanças realizadas pelo srTCM-PCA entre a cor pré-definida e a ação realizada.

Já para os experimentos do trTCM-PCA, o *marker* utilizado foi o trTCM. Desta forma, assim como ocorre com o srTCM-PCA, se os parâmetros forem iguais, não há mudança nas cores das ações. Na Figura 5.24 é apresentada a quantidade de transmissões realizadas pelo trTCM-PCA para cada tráfego, sendo que o valor de PBS é alterado entre 2000, 3500 e 5500 *tokens*, enquanto CIR, PIR e CBS tem valores iguais a 1500, 3400 e 7000 *tokens*, respectivamente. Já na Figura 5.25 são mostradas as quantidades de pacotes de cada cor recebidos e também a quantidade de ações de cada cor realizadas pelo trTCM-PCA, com os mesmos parâmetros para os tráfegos Alfa, Beta, Elefante e Guepardo. Ao analisar os resultados é possível verificar que o trTCM-PCA não realiza mudanças de coloração no momento de realizar as ações para cada pacote, visto que trTCM e trTCM-PCA tem o mesmos valores para os parâmetros e o mesmo comportamento.

Outro experimento foi realizado com o trTCM-PCA, mas assumindo que o *marker* não está colorindo os pacotes corretamente, elevando o número de amarelos na rede, fato que

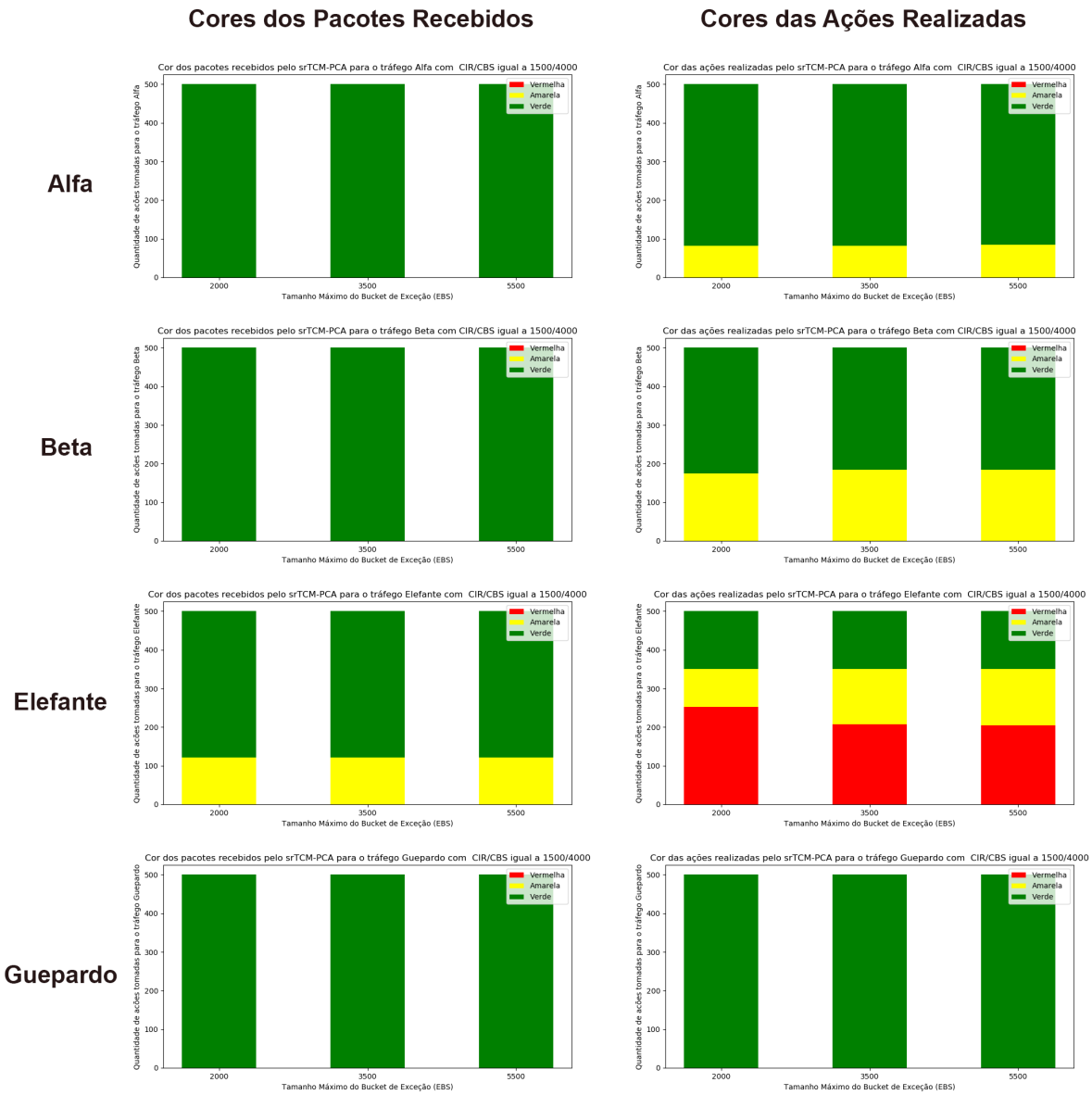


Figura 5.23: Comparação entre as cores recebidas e as cores das ações realizadas para o srTCM-PCA, tendo o valor de EBS alterado, e CIR e CBS iguais a 1500 e 4000, respectivamente.

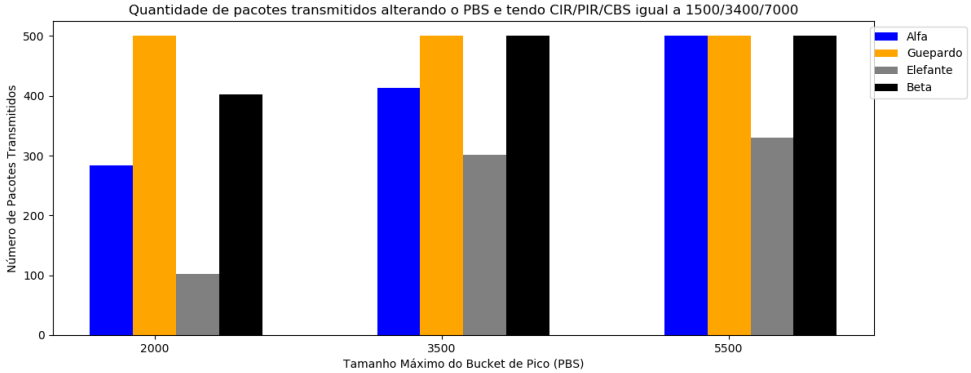


Figura 5.24: Transmissões realizadas pelo trTCM-PCA alterando o PBS e com CIR, PIR e CBS iguais a 1500, 3400 e 7000, respectivamente, para os diferentes tipos de tráfego.

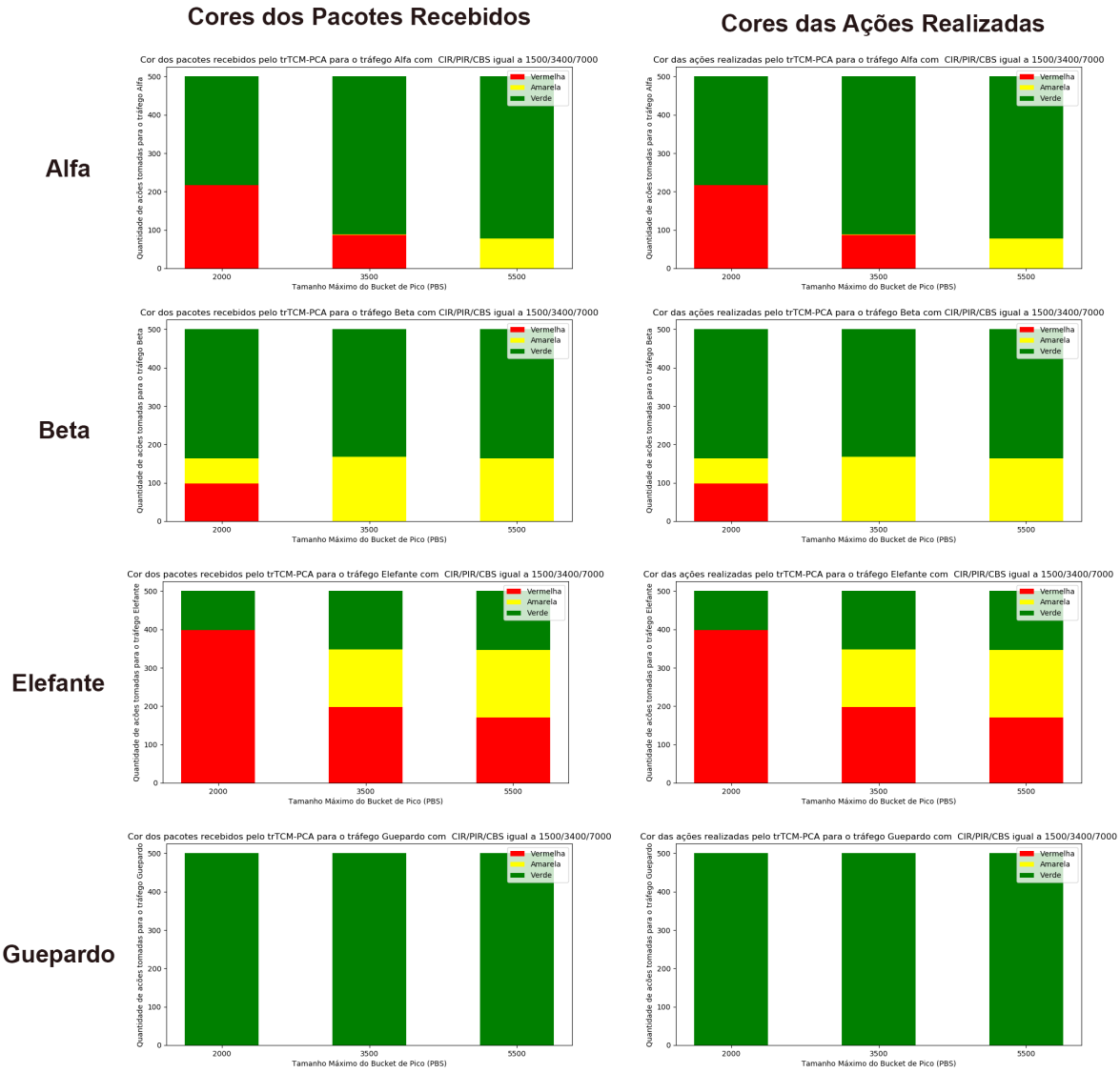


Figura 5.25: Comparação entre as cores recebidas e as cores das ações realizadas para o trTCM-PCA, tendo o valor de PBS alterado, e CIR, PIR e CBS iguais a 1500, 3400 e 7000, respectivamente.

para esse experimento não era correto. Desta forma, o trTCM-PCA verifica esse aumento de amarelos na rede e pode alterar a coloração do mesmo para vermelho, para atender os requisitos da rede. As colorações dos pacotes ao serem recebidos e a coloração das ações realizadas pelo trTCM-PCA, alterado o valor de PBS, e com CIR, PIR e CBS iguais a 2700, 2300 e 7000 *tokens*, respectivamente, podem ser visualizadas na Figura 5.26. Ao analisar a figura, é possível verificar que existe uma diminuição na quantidade de amarelos para o tráfego Beta, sendo alterados 12,6% e 8,3% dos pacotes amarelos para a ação vermelha, quando EBS é igual 3500 e 5500 *tokens*, respectivamente. O mesmo ocorre para o tráfego Elefante, sendo alterados 50,3% e 57,9% dos pacotes amarelos para a ação vermelha. Ou seja, para esses tráfegos o *marker* coloriu mais pacotes com a cor amarela, ultrapassando os limites da rede. Para o tráfego Guepardo, não houve alterações na coloração, sendo recebidos e transmitidos todos os pacotes na cor verde e sendo realizada a ação verde.

Já para o tráfego Alfa, é interessante verificar que, apesar de ter uma certa quantidade de amarelos, o número de ações verdes, amarelas e vermelhas respeitam a coloração, ou seja, o trTCM-PCA não altera a coloração de nenhum pacote, recebendo e transmitindo, quando o

EBS é igual a 2000 *tokens*, 60,4% de pacotes verdes com ações verdes e 39,6% de pacotes vermelhos com ações vermelhas. Já quando o EBS é igual a 3500 ou 5500 *tokens*, são recebidos e transmitidos 84,6% de pacotes verdes com ações verdes e 15,4% de pacotes amarelos com ações amarelas. Vale ressaltar, que mesmo assinalando outros parâmetros ao marcador, o mesmo resultado é atingido para o Alfa, isso se deve a fato de que o trTCM-PCA não pode aumentar a prioridade (cor) do pacote, sendo assim, ele apenas aceita as colorações do *marker*, mesmo que os *buckets* tenham *tokens* suficientes para a realização da ação superior.

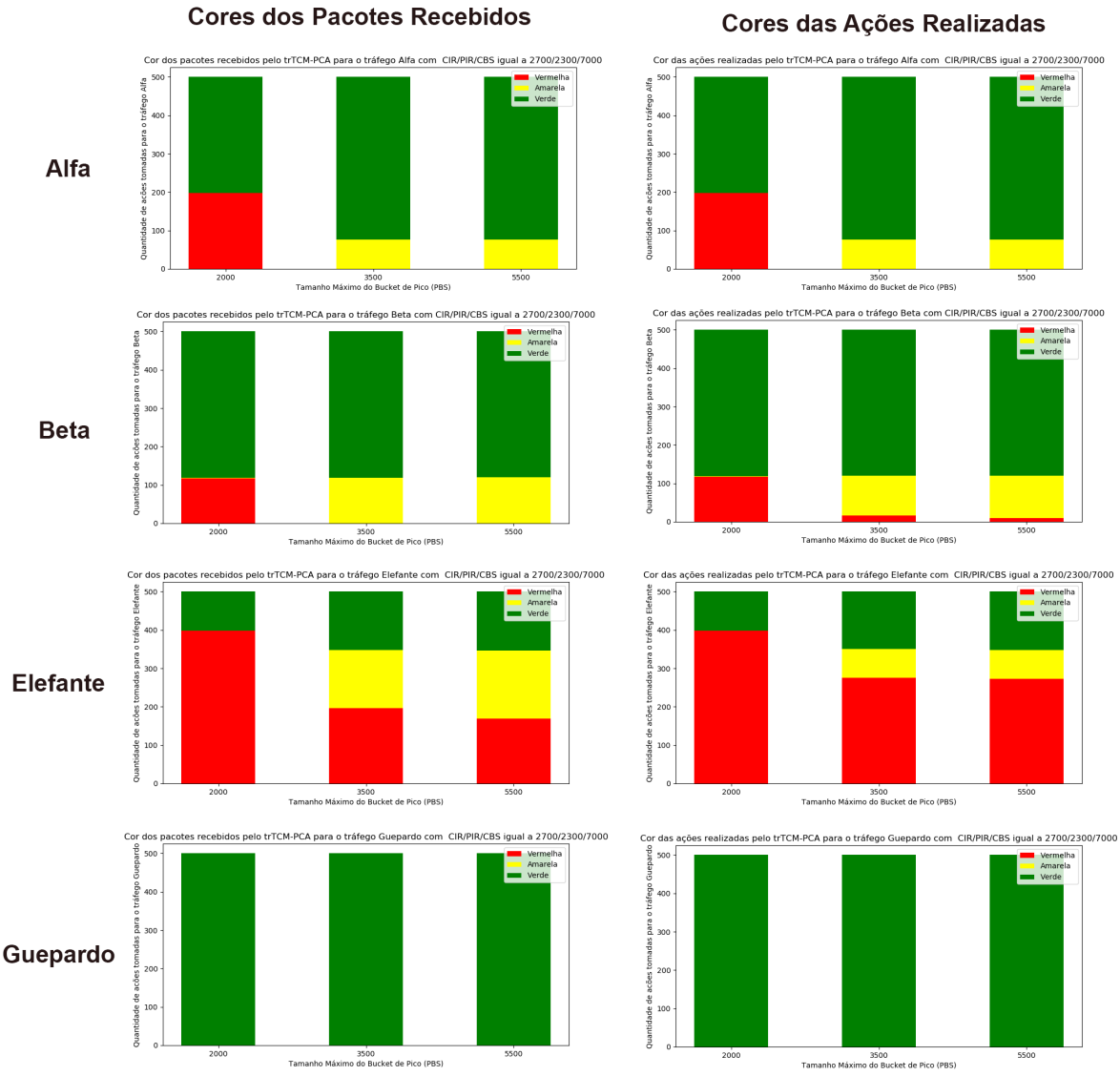


Figura 5.26: Comparação entre as cores recebidas e as cores das ações realizadas para o trTCM-PCA, tendo o valor de PBS alterado, e CIR, PIR e CBS iguais a 2700, 2300 e 7000, respectivamente.

Após análise dos experimentos para ambos *Color-Aware Policers*, srTCM-CA e trTCM-CA, verificam-se que os mecanismos de CAP são mecanismos de confiança, pois atestam, utilizando parâmetros próprios, a veracidade da coloração dos pacotes recebidos por eles. Os mecanismos também conseguem readequar os tráfegos para estarem nos requisitos da rede, alterando a coloração de pacotes, se necessário, de verdes para amarelos, verdes para vermelhos e amarelos para vermelhos. Entretanto, se a falha do mecanismo de *marking*, que está realizando as colorações nos pacotes, causar aumento da quantidade de pacotes vermelhos, os CAP, mesmo

com condições de transmissão, aceitam a coloração, sendo assim, no exemplo, tais pacotes são descartados desnecessariamente na rede.

5.3 *SHAPERS*

Os mecanismos de *shaping* visam ajustar o fluxo de tráfego a um limite estabelecido, utilizando uma taxa máxima definida. Diferentemente dos mecanismos de *policing*, nos *shapers* são utilizadas estruturas de fila de pacotes. Desta forma, nestes mecanismos o descarte de pacotes pode ser evitado através da realização de uma transmissão posterior, quando as condições necessárias para à mesma forem atingidas.

A seguir são mostrados os resultados experimentais obtidos para os dois principais mecanismos de *shaping*: o *Single Rate Token Bucket Shaper* (srTBS) e o *Leaky Bucket* (LB).

5.3.1 *Single Rate Token Bucket Shaper*

No srTBS, pacotes recebidos que não estiverem em conformidade são adicionados a uma fila de espera, se a mesma não tiver atingido a capacidade máxima, para poderem ser transmitidos posteriormente quando as condições para a transmissão forem atingidas, ao contrário do srTBP, o *policer* equivalente. Dessa forma, o srTBS pode gerar um atraso no tráfego de pacotes, visto que pacotes não conformes são armazenados na fila e esperam até que sua transmissão possa ser realizada. Enquanto isso, pacotes recebidos depois, mas que estejam em conformidade podem ser transmitidos imediatamente, ou seja, antes dos pacotes que estão na fila.

Os experimentos realizados no mecanismo srTBS foram feitos alterando dois parâmetros: a rajada (*burst*) e a taxa de reposição de *tokens* ao *bucket* (*rate*), assim como no TBP. Baseando-se em cada perfil de tráfego e a quantidade de *bytes* transmitidos por segundo pelos mesmos, foram escolhidos os valores para ambos os parâmetros. Após análise, foram selecionados os seguintes valores para a taxa de reposição de *tokens* ao *bucket*: 1500, 2700 e 3900 *tokens*. E os seguintes valores para a rajada: 2700, 4000 e 7000 *tokens*. O tamanho da fila, que armazena pacotes não conformes, foi escolhido para ser 10% da quantidade de pacotes enviados pelo cliente, assim, a quantidade de pacotes que podem ser armazenados na fila é igual a 50.

Os resultados experimentais obtidos na execução do mecanismo srTBS na rede, para os diferentes valores da rajada e com a taxa de reposição de *tokens* ao *bucket* igual a 1500, podem ser visualizados na Figura 5.27. Ao analisar os resultados, é possível verificar que os mesmos são semelhantes aos obtidos para o srTBP com mesma taxa de reposição de *tokens* ao *bucket*, apresentados na Figura 5.3, havendo um aumento de, em média, 24,7%, 7,7% e 47,3% no número de pacotes transmitidos para os tráfegos Alfa, Beta e Elefante, respectivamente, por conta da existência da fila de espera. Para o tráfego Guepardo, manteve-se a transmissão de 100% dos pacotes recebidos. Entretanto, apesar do aumento no número de transmissões, assim como no srTBP, no srTBS a taxa de reposição de *tokens* ao *bucket* com valor de 1500 é menor que a quantidade de *bytes* transmitidos por segundo pelo tráfego Alfa, Beta e Elefante, fazendo com que ao aumentar o valor da rajada, haja mudanças ínfimas nos resultados, com desvios padrão de 2,9, 7,4 e 16,8, respectivamente, na quantidade de pacotes transmitidos pelo srTBS.

O srTBS realiza mais transmissões que o srTBP, com os mesmos valores para os parâmetros, visto que o srTBS tem a fila de espera. Entretanto, o pacote adicionado na fila tem que esperar o mecanismo ter condições para que a transmissão do mesmo seja realizada, ou seja, o *bucket* deve ter *tokens* suficientes para transmitir o pacote. Dependendo do tráfego, a disponibilidade de *tokens* para a transmissão de pacotes da fila pode ser pequena, visto que os pacotes que chegam ao mecanismo também consomem *tokens*, e apenas os *tokens* que restam após as transmissões dos mesmos são utilizados pelos pacotes da fila.

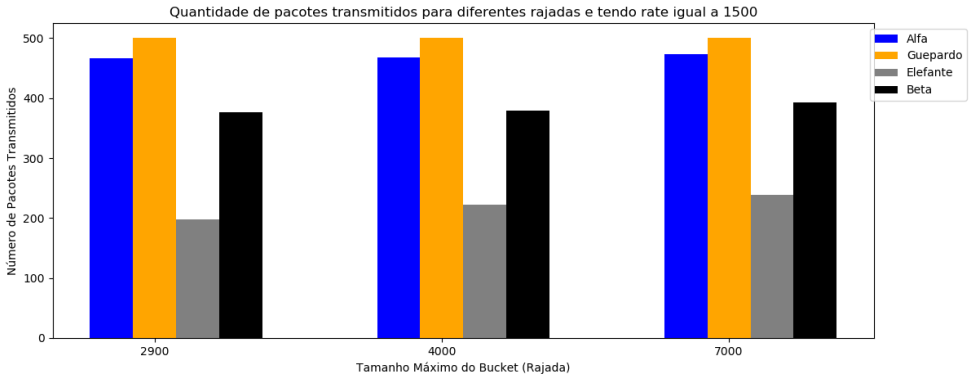


Figura 5.27: Transmissões realizadas pelo srTBS alternando a rajada e *rate* igual a 1500 para os diferentes tipos de tráfego.

Os tempos de espera médios para os pacotes da fila na execução do srTBS, alterando a rajada e com taxa de reposição de *tokens* ao *bucket* igual a 1500, podem ser visualizados na Figura 5.28. Ao analisar os resultados, é possível verificar que os pacotes do tráfego Elefante, de tamanho 1048 *bytes*, consomem o *bucket* em grande parte, fazendo com que não restem *tokens* para a transmissão da fila. Desta forma, o tempo de espera na fila para o tráfego Elefante chega a 58 segundos quando o valor da rajada é 2900 *tokens*, mas esse tempo diminui ao se aumentar a rajada, sendo de 49,2 segundos quando a rajada é 4000 *tokens* e 47 segundos quando é 7000 *tokens*.

Já para o tráfego Alfa, durante os 2 segundos de não transmissão, os *tokens* do *bucket* são direcionados para o consumo da fila, já que não são recebidos novos pacotes, mas *tokens* ainda são adicionados ao *bucket*. Entretanto, a taxa de reposição de *tokens* ao *bucket* ainda é menor que a taxa de *bytes* do tráfego, fazendo com que os pacotes esperem 15,7 , 15,6 e 15,1 segundos, quando a rajada tem valor igual a 2900, 4000 e 7000 *tokens*, respectivamente.

O tráfego Guepardo não adicionou pacotes a fila, portanto todos os pacotes foram transmitidos assim que recebidos pelo mecanismo, não tendo tempo de espera, ou seja, 0 segundos. Já o tráfego Beta tem o tempo de espera do pacote na fila de 7,7, 7,5 e 7,5 segundos, quando a rajada tem valor igual a 2900, 4000 e 7000 *tokens*, respectivamente. O fato de haver pouca variação nos tempos de espera obtidos para o tráfego Alfa e Beta, para os diferentes valores da rajada ocorre, pois, são consumidos mais *tokens* do *bucket* do que adicionados ao mesmo, isso faz com que a quantidade de *tokens* diminua para valores menores que o tamanho de 1 pacote do tráfego. Assim, a quantidade de *tokens* adicionados sempre é consumida a cada segundo, gerando um padrão de transmissões e descartes, e o aumento do valor da rajada apenas retarda o aparecimento desse padrão.

Na Figura 5.29 são apresentados os resultados experimentais obtidos na execução do mecanismo srTBS na rede para os diferentes valores de rajada e com a taxa de reposição de *tokens* ao *bucket* igual a 2700 *tokens*. Para os tráfegos Alfa, Beta e Guepardo, é atingido 100% de transmissão dos pacotes enviados pelo cliente, independentemente dos valores da rajada. Isso ocorre, pois, a taxa de reposição de *tokens* no *bucket* é igual a 2700 *tokens*, no caso do Guepardo, o valor é superior à quantidade de *bytes* transmitidos por segundo no fluxo, fazendo com que o *bucket* sempre tenha disponibilidade de *tokens* para a transmissão do pacote.

Para o tráfego Beta, ao ter a rajada igual a 2900 *tokens*, quando 3480 *bytes* são recebidos, os pacotes que excederem o *bucket* são adicionados a fila, sendo transmitidos quando o mecanismo receber os 1480 *bytes*, pois haverá disponibilidade de *tokens* no *bucket*, transmitindo 100% dos pacotes recebidos. Quando a rajada é igual 4000 ou 7000 *tokens*, nenhum pacote é adicionado na

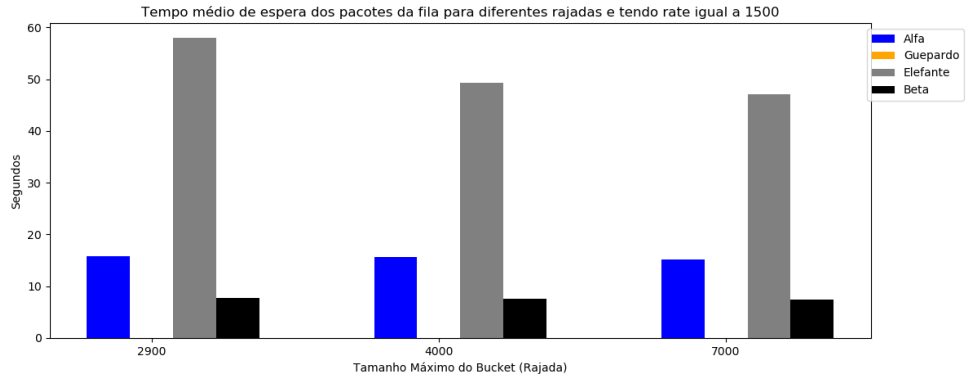


Figura 5.28: Tempo de espera médio dos pacotes da fila na execução do srTBS, alterando a rajada e taxa de reposição de *tokens* ao *bucket* igual a 1500.

fila, visto que o *bucket* sempre tem disponibilidade de *tokens* para o fluxo, já que quando 1480 *bytes* são recebidos, a quantidade de *bytes* é menor que a quantidade de *tokens* já adicionados, e o restante dos *tokens*, 1220 *tokens*, somados aos 2700 adicionados, são suficientes para os 3480 *bytes* seguintes.

Para o tráfego Alfa, quando a rajada é igual a 2900 e 4000 *tokens*, o valor é inferior à quantidade de *bytes* transmitidos pelo tráfego em 1 segundo, fazendo com que pacotes sejam adicionados à fila, mas os mesmos são transmitidos no período de 2 segundos de não recebimento de pacotes. Quando a rajada é de 7000 *tokens*, o valor é superior à quantidade de *bytes* transmitidos pelo tráfego em 1 segundo, dessa forma, no período de 2 segundos de não recebimento de pacotes, o *bucket* atinge sua capacidade máxima, tendo disponibilidade de *tokens* para transmitir os pacotes sem precisar adicioná-los a fila.

Já para o tráfego Elefante, obtêm-se 50,6%, 67,6% e 73,8% de pacotes transmitidos, quando a rajada é igual a 2900, 4000 e 7000 *tokens*, respectivamente. Isso ocorre já que a quantidade de *bytes* transmitidos por segundo no fluxo é superior à taxa de reposição de *tokens* ao *bucket*, fazendo com que sejam consumidos mais *tokens* que adicionados. Assim, o *bucket* acaba com a quantidade inferior de *tokens* disponíveis ao tamanho de um pacote do tráfego, tendo que adicionar pacotes recebidos na fila. A fila tem tamanho finito e quando cheia causa o descarte de pacotes. O aumento da rajada faz com que o *bucket* demore mais pacotes antes de ter que adicioná-los à fila de espera e até mesmo descartá-los, ou seja, aumenta-se o número de transmissões realizadas pelo srTBS inicialmente.

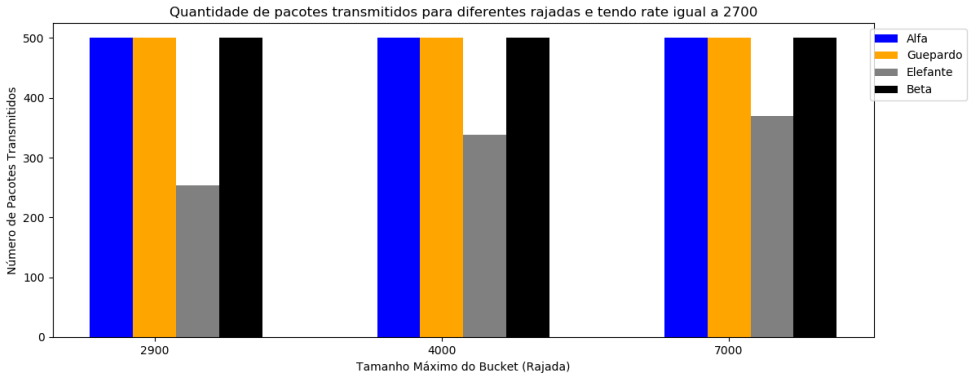


Figura 5.29: Transmissões realizadas pelo srTBS alternando a rajada e *rate* igual a 2700 para os diferentes tipos de tráfego.

Já na Figura 5.30 mostra-se o tempo de espera médio para um pacote da fila ser transmitido pelo srTBS, para os diferentes valores de rajada e com a taxa de reposição de *tokens* no *bucket* igual a 2700 *tokens*. Analisando os resultados tem-se que o tempo dos pacotes na fila de espera, para os diferentes perfis de tráfegos, diminuíram significativamente se comparados ao resultado obtido na Figura 5.28, quando a taxa de reposição de *tokens* no *bucket* era igual a 1500. Para o tráfego Alfa, houve uma diminuição do tempo de espera dos pacotes na fila de 91%, 92,9% e 100%, para os respectivos valores de rajada: 2900, 4000 e 7000 *tokens*. Para o Beta, obteve-se uma diminuição do tempo de espera dos pacotes na fila de 97,4%, 100% e 100%, para os respectivos valores de rajada: 2900, 4000 e 7000 *tokens*. Para o tráfego Guepardo, manteve-se o tempo de 0 segundos na fila, visto que nenhum pacote foi adicionado à mesma. Já para o Elefante, houve uma diminuição do tempo de espera dos pacotes na fila de 40%, 59,1% e 60%, para os respectivos valores de rajada: 2900, 4000 e 7000 *tokens*. A diminuição se deve ao fato da taxa de reposição de *tokens* ser igual a 2700, aumentando a disponibilidade de *tokens* no *bucket* tanto para os pacotes da fila de espera, quanto para os que estão sendo recebidos.

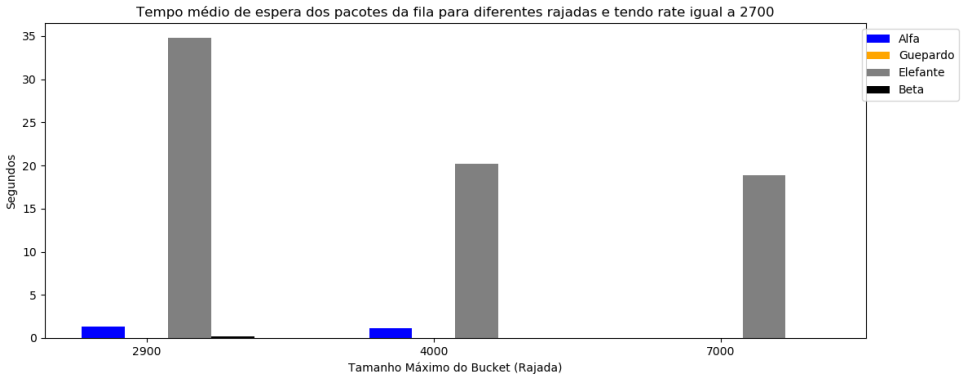


Figura 5.30: Tempo de espera médio dos pacotes da fila na execução do srTBS, alterando a rajada e taxa de reposição de *tokens* ao *bucket* igual a 2700.

Destarte, após análise de todos os experimentos em conjunto é possível verificar os impactos gerados pelos dois parâmetros, a rajada e a taxa de reposição de *tokens* ao *bucket*, no comportamento do srTBS. Quanto aos valores das rajadas, percebe-se que aumentar a rajada auxilia para que a fila não atinja sua capacidade máxima rapidamente, diminuindo o descarte de pacotes. Entretanto, se forem muito pequenos em comparação à quantidade de *bytes* transmitidos por segundo pelo tráfego, independente do valor da taxa de reposição de *tokens* ao *bucket*, a rajada limitaria a rede em seu valor, fazendo com que pacotes que excederem esse limite sejam adicionados a fila, e se necessário, descartados. Por outro lado, o aumento taxa de reposição de *tokens* ao *bucket* mostrou diminuir o tempo de espera para a transmissão de pacotes da fila de espera. Fato que ocorre, visto que o aumento da taxa de reposição de *tokens* ao *bucket* faz com que menos pacotes sejam adicionados na fila; além de acumularem mais *tokens*, no *bucket*, aumentando as oportunidades de haver *tokens* suficientes para o srTBS realizar a transmissão de pacotes da fila. Desta forma, o mecanismo realiza o descarte de pacotes apenas se o limite estabelecido pelos parâmetros forem ultrapassados, fazendo com que a fila de espera atinja sua capacidade máxima.

5.3.2 *Leaky Bucket*

Os experimentos com o LB foram realizados alterando a taxa constante (*rate*), a qual indica a quantidade de pacotes que serão consumidos do *bucket*, que neste mecanismo é uma

estrutura de fila. Os valores para a taxa foram escolhidos baseando-se na quantidade de pacotes transmitidos em 1 segundo por cada perfil de tráfego: Alfa, Beta, Elefante e Guepardo. Desta forma, após análise foram selecionados os valores de taxa 3, 7, 13 e 25 pacotes. A profundidade do *bucket* foi escolhida para ser 10% da quantidade de pacotes transmitidos pelo cliente, assim, a quantidade de pacotes que podem ser armazenados no *bucket* é igual a 50. Fato importante é que o LB não considera o tamanho do pacote, mas sim a quantidade de pacotes trafegando pela rede. Os resultados experimentais obtidos na execução do mecanismo LB na rede podem ser visualizados na Figura 5.31.

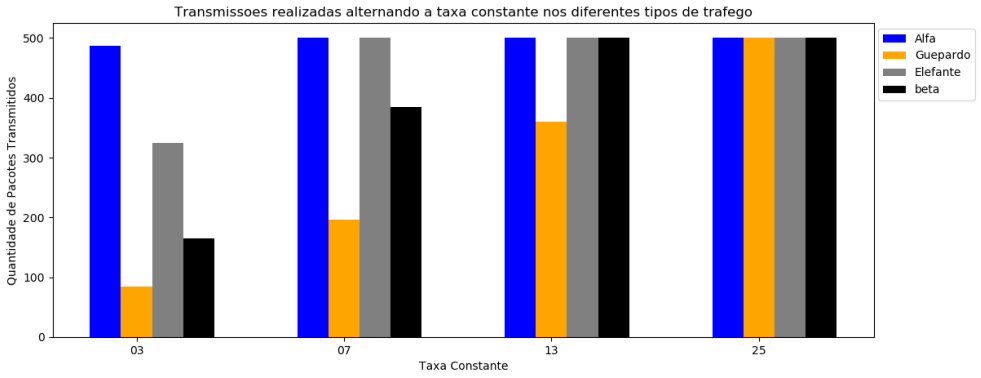


Figura 5.31: Transmissões realizadas pelo LB alternando a taxa constante para os diferentes tipos de tráfego.

Ao analisar os resultados é possível verificar que para o tráfego Alfa, quando a taxa constante é igual a 3, atinge-se 97,4% de transmissões dos pacotes recebidos, enquanto para as outras taxas constantes, 7, 13 e 25, são obtidas taxas de 100% de transmissões. Isso se deve ao fato de que o tráfego Alfa tem 2 segundos de não transmissão após transmitir 10 pacotes. Assim, nesse intervalo não há o recebimento de pacotes, fazendo com que não haja adição de novos pacotes ao *bucket*. Entretanto, o LB continua a consumir o *bucket* a cada intervalo de 1 segundo, diminuindo a quantidade de pacotes presentes no *bucket*. Apesar disso, é possível afirmar que mesmo com os 2 segundos de intervalo, para a taxa constante igual a 3, a fila está crescendo de tamanho, pois mais pacotes são recebidos do que transmitidos, sendo 10 recebidos e 9 transmitidos a cada 3 segundos. Assim, o descarte de pacotes tende a crescer com o aumento na quantidade de pacotes recebidos.

Para o tráfego Guepardo, que apresenta pacotes de tamanho pequeno, são obtidas as seguintes taxas de transmissão: 16,8%, 39,2%, 72% e 100%, quando a taxa constante é igual a 3, 7, 13 e 25, respectivamente. A pequena taxa de transmissão, de apenas 16,2%, quando a taxa constante é igual a 3, se deve ao fato do tráfego Guepardo enviar 20 pacotes por segundo, ou seja, valor superior à taxa constante, fazendo com que o *bucket* atinja a capacidade máxima de pacotes, descartando novos pacotes recebidos. Assim, aumentar a taxa constante faz com que a taxa de transmissão dos pacotes do fluxo também aumente, sendo que quando a taxa constante é maior que o número de pacotes enviados por segundo no tráfego, atinge-se 100% de transmissões.

Para o tráfego Elefante, não se atinge 100% de transmissões apenas quando a taxa constante é igual a 3, tendo 64,8% de pacotes transmitidos pelo LB. Isso ocorre já que são enviados 5 pacotes por segundo, e quando a taxa constante é igual a 3, não é possível transmitir todos os pacotes, adicionando os restantes na fila. Entretanto, a fila é finita e, quando cheia, o LB realiza o descarte de novos pacotes, por não ser possível armazená-los. Já para o tráfego Beta, ocorre a mesma situação, mas são 10 pacotes enviados por segundo, assim, não se atinge 100% de transmissões quando a taxa constante é igual a 3 e 7, atingindo 33% e 77% de transmissões,

respectivamente. Isso também se deve ao fato da quantidade de pacotes transmitidos por segundo pelo fluxo é maior que a taxa constante.

Para os experimentos de *throughput* máximo foram escolhidas as taxas 500, 700, 900 e 1100 pacotes. Os resultados obtidos são apresentados na Figura 5.32. Após análise, é possível verificar que para a taxa igual a 500, atinge-se transmissão de em média, 5,6 Mbps, para 700 pacotes, atinge-se, em média, 8,4 Mbps, e para 900 e 1100 pacotes, atinge-se 10 Mbps, sendo transmitidos todos os pacotes neste último caso. Os resultados obtidos para o tráfego de 10 Mbps mostram que o comportamento do LB para esse fluxo é semelhante ao encontrado nos experimentos realizados para os tráfegos Alfa, Beta, Elefante e Guepardo. Fato que mostra uma uniformidade no comportamento do mecanismo.

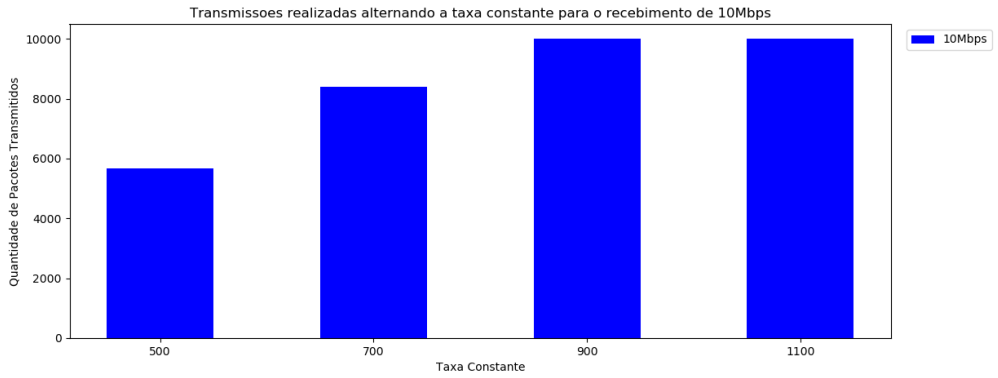


Figura 5.32: Transmissões realizadas pelo LB para o tráfego de 10Mbps.

Destarte, é explícito como a quantidade de transmissões realizadas pelo LB está diretamente ligada à taxa constante. Por exemplo, com a taxa constante em 25 pacotes, ou seja, maior que a quantidade de pacotes enviados por segundo em todos os perfis de tráfego, em todos eles foram transmitidos 100% dos pacotes. Ou seja, se o valor da taxa constante for maior que o número de pacotes recebidos em 1 segundo, todos serão transmitidos, caso contrário o fluxo é limitado pela taxa constante, havendo, se necessário, o descarte de pacotes.

6 CONCLUSÃO

A engenharia de tráfego tem um papel importante no gerenciamento de redes de computadores, visto que conta com diversos mecanismos que auxiliam a manter a qualidade de serviço, administrando o tráfego de dados que circula nas mesmas. Este trabalho propõe o NFV-TE, um *framework* parametrizável para geração de funções de rede voltadas para engenharia de tráfego. Essas funções são projetadas para executarem no contexto do paradigma NFV, com tecnologias de virtualização. O NFV-TE consegue gerar mecanismos de *shaping* e *policing* que são parametrizáveis em termos das taxas máximas permitidas. Os mecanismos podem causar atraso na entrega dos pacotes, bem como o descarte dos mesmos quando necessário. Sua configuração é realizada através de um arquivo JSON que contém os parâmetros desejados para a NF que será gerada, a qual pode ser executada em servidores de prateleira utilizando tecnologias de virtualização.

Os mecanismos de *policing* implementados no NFV-TE foram o *Single Rate Token Bucket Policer* (srTBP), o *Single Rate Three Color Marker Policer* (srTCM-*Policer*), o *Two Rate Three Color Marker Policer* (trTCM-*Policer*) e os *Color-Aware Policers*. Já os mecanismos de *shaping* foram o *Single Rate Token Bucket Shaper* e o *Leaky Bucket*.

Um conjunto de experimentos foi realizado com todos os mecanismos disponíveis no NFV-TE, sendo geradas as funções de rede com diferentes parâmetros e executadas em ambientes virtualizados e controlados utilizando diferentes perfis de fluxos de dados: Alfa, Beta, Elefante e Guepardo. Com os resultados experimentais obtidos, foi possível observar como os mecanismos implementados de engenharia de tráfego atuam nos fluxos da rede, demonstrando o comportamento dos *shapers* e *policers* no processamento de pacotes, assim fazendo com que o tráfego de rede seja limitado pelos valores estabelecidos pelos parâmetros de cada mecanismo executado. Desta forma, ao aumentar o limite estabelecido para um mecanismo, a quantidade de pacotes transmitidos tipicamente aumenta. É interessante verificar as diferenças obtidas no LB, sendo o único mecanismo a realizar o descarte de pacotes para o tráfego Guepardo. Fato esse que ocorre já que o LB é o único dentre os mecanismos que classifica os pacotes com base na quantidade de pacotes recebidos por segundo, e não pela quantidade de *bytes* recebidos por segundo.

Como trabalhos futuros, sugere-se verificar o comportamento dos mecanismos implementados pelo NFV-TE com diferentes combinações de fluxos de dados em simultâneo. Além disso, também é objetivo futuro executar os mecanismos de engenharia de tráfego em ambientes não controlados, como através das suas implantações em redes de produção. Destaca-se também a possibilidade de extensão do *framework* NFV-TE, podendo esta ser feita através da adição de novos *policers* e *shapers*, bem como outros tipos de mecanismos de engenharia de tráfego, como os *schedulers*. Por fim, é também trabalho futuro tornar o NFV-TE dinâmico, no sentido de poder ser reconfigurado por um sistema de gerenciamento de tráfego baseado em políticas que determinam a necessidade de *policers* e *shapers* segundo as condições observadas na rede.

REFERÊNCIAS

Cotroneo, D., De Simone, L., Iannillo, A. K., Lanzaro, A., Natella, R., Fan, J. e Ping, W. (2014). Network function virtualization: Challenges and directions for reliability assurance. Em *International Symposium on Software Reliability Engineering Workshops*, páginas 37–42. IEEE.

Cui, C., Deng, H., Telekom, D., Michel, U. e Damker, H. (2012). Network functions virtualisation.

da Cruz Marcuzzo, L., Garcia, V. F., Cunha, V., Corujo, D., Barraca, J. P., Aguiar, R. L., Schaeffer-Filho, A. E., Granville, L. Z. e dos Santos, C. R. P. (2017). Click-on-osv: A platform for running click-based middleboxes. Em *IFIP/IEEE Symposium on Integrated Network and Service Management*, páginas 885–886.

de Souza, G. V. e Duarte Jr, E. P. (2020). Uma arquitetura de alta disponibilidade para funções virtualizadas de rede. Em *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, páginas 407–420. SBC.

Dong, L. e Clemm, A. (2021). High-precision end-to-end latency guarantees using packet wash. Em *IFIP/IEEE International Symposium on Integrated Network Management*, páginas 259–267. IEEE.

ETSI (2020). Osm. `https://osm.etsi.org/`. Acessado em 27/10/2021.

ETSI, N. (2014). Network functions virtualisation (nfv); terminology for main concepts in nfv. *Group Specification, Dec*.

Evans, J. W. e Filsfils, C. (2010). *Deploying IP and MPLS QoS for multiservice networks: theory and practice*. Elsevier.

Flauzino, J., Fulber-Garcia, V., Huff, A., Venâncio, G. e Duarte Jr, E. P. (2021). Gerência e orquestração de funções e serviços de rede virtualizados em nuvem cloudstack. Em *Workshop de Gerência e Operação de Redes e Serviços*, páginas 82–95. SBC.

FRAUNHOFER-FOKUS e TU-BERLIN (2017). Open baton. `https://openbaton.github.io/`. Acessado em 27/10/2021.

Fulber-Garcia, V., Luizelli, M. C., dos Santos, C. R. P. e Duarte, E. P. (2020). Cusco: a customizable solution for nfv composition. Em *International Conference on Advanced Information Networking and Applications*, páginas 204–216. Springer.

Garcia, V. F., de Freitas Gaiardo, G., da Cruz Marcuzzo, L., Nunes, R. C. e dos Santos, C. R. P. (2018). Demons: A ddos mitigation nfv solution. Em *International Conference on Advanced Information Networking and Applications*, páginas 769–776. IEEE.

Garcia, V. F., Marcuzzo, L. d. C., Huff, A., Bondan, L., Nobre, J. C., Schaeffer-Filho, A., dos Santos, C. R. P., Granville, L. Z. e Duarte, E. P. (2019). On the design of a flexible architecture for virtualized network function platforms. Em *IEEE Global Communications Conference*, páginas 1–6.

Hamdan, M., Mohammed, B., Humayun, U., Abdelaziz, A., Khan, S., Ali, M. A., Imran, M. e Marsono, M. N. (2020). Flow-aware elephant flow detection for software-defined networks. *IEEE Access*, 8:72585–72597.

Han, B., Gopalakrishnan, V., Ji, L. e Lee, S. (2015). Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97.

Heinanen, J. e Guerin, R. (1999a). Rfc2697: a single rate three color marker.

Heinanen, J. e Guerin, R. (1999b). Rfc2698: A two rate three color marker.

Kuribayashi, S.-i. (2021). Dynamic shaping method using sdn and nfv paradigms.

Luizelli, M. C., Bays, L. R., Buriol, L. S., Barcellos, M. P. e Gaspary, L. P. (2015). Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions. Em *IFIP/IEEE International Symposium on Integrated Network Management*, páginas 98–106.

Maji, S., Veeraraghavan, M., Buchanan, M., Alali, F., Ros-Giral, J. e Commike, A. (2017). A high-speed cheetah flow identification network function (cfinf). Em *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, páginas 1–7.

Martins, J., Ahmed, M., Raiciu, C., Olteanu, V., Honda, M., Bifulco, R. e Huici, F. (2014). Clickos and the art of network function virtualization. Em *USENIX Symposium on Networked Systems Design and Implementation*, páginas 459–473, Seattle, WA. USENIX Association.

Mijumbi, R., Serrat, J., Gorricho, J.-L., Bouten, N., De Turck, F. e Boutaba, R. (2015). Network function virtualization: State-of-the-art and research challenges. *IEEE Communications surveys & tutorials*, 18(1):236–262.

Nucci, A. e Papagiannaki, K. (2009). *Design, measurement and management of large-scale IP networks: Bridging the gap between theory and practice*. Cambridge University Press.

OPENSTACK (2016). Tacker. `https://wiki.openstack.org/wiki/Tacker`. Acessado em 27/10/2021.

Rosa, R., Siqueira, M., Rothenberg, C. E., Barea, E. e Marcondes, C. (2014). Network function virtualization: Perspectivas, realidades e desafios. *Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*.

Sarvotham, S., Riedi, R. e Baraniuk, R. (2001). Connection-level analysis and modeling of network traffic. IMW '01, página 99–103, New York, NY, USA. Association for Computing Machinery.

Sekar, V., Egi, N., Ratnasamy, S., Reiter, M. K. e Shi, G. (2012). Design and implementation of a consolidated middlebox architecture. Em *9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, páginas 323–336.

Sherry, J., Hasan, S., Scott, C., Krishnamurthy, A., Ratnasamy, S. e Sekar, V. (2012). Making middleboxes someone else's problem: Network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24.