NFV-TE: Geração Automática de Funções Virtualizadas para Engenharia de Tráfego de Rede

Felipe Ribeiro Quiles, João Vitor Moreira, Vinicius Fulber-Garcia, Elias P. Duarte Jr.

Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19018 Curitiba 81531-990 PR

{frquiles, jvm17, vfgarcia, elias}@inf.ufpr.br

Abstract. *This work presents NFV-TE, a traffic engineering framework based on NFV technology (Network Function Virtualization). The NFV paradigm brings flexibility to networks by allowing the implementation of core functions on a software plane. After the network operator enters a set of characteristics of the desired traffic engineering mechanism, NFV-TE validates and consolidates the data on a JSON configuration file, from which the corresponding virtual network function is generated. NFV-TE is an extensible solution, designed to allow the addition of arbitrary functions, which can be easily instantiated and configured. This paper describes the various policers and shapers that are already present in NFV-TE, describing their execution in different scenarios and under different network traffic profiles.*

Resumo. *Este trabalho apresenta o NFV-TE, um framework para engenharia de tráfego baseado em tecnologia NFV (Network Function Virtualization). O paradigma NFV traz grande flexibilidade às redes ao permitir a implementação de funções do núcleo em um plano de software. O NFV-TE é um framework extensível e parametrizável, que permite a um operador de rede especificar o mecanismo de engenharia de tráfego, consolidado como um arquivo JSON, a partir do qual é gerada a função virtual de rede correspondente. O sistema foi projetado para apresentar grande facilidade para inclusão e configuração de funções. O artigo descreve os diversos policers e shapers já presentes no NFV-TE, destacando a facilidade de configuração e demonstrando sua execução em diversos cenários e para diferentes perfis de tráfego de rede.*

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 [Nucci and Papagiannaki 2009, Cavalca et al. 2014]. Entre seus principais papéis está garantir o bom desempenho da rede, enquanto permite manter a utilização dos recursos computacionais em níveis desejáveis e com previsibilidade. O emprego de mecanismos de engenharia de tráfego viabiliza uma rede mais robusta e eficiente. Dentre os principais mecanismos existentes estão os *policers* e os *shapers* [Evans and Filsfils 2010, Daniel-Simion and Dan-Horia 2011]. *Policers* têm o propósito principal de eliminar os picos no fluxo de dados, limitando os mesmos a uma taxa máxima predefinida. Já os *shapers* suavizam o perfil do tráfego de rede, atrasando os pacotes ao utilizar filas consumidas de acordo com taxas estabelecidas.

Este trabalho descreve esforços para trazer os benefícios do paradigma NFV para a engenharia de tráfego. A utilização de tecnologias de virtualização, torna possível implementar mecanismos de engenharia de tráfego em software, como funções virtualizadas de rede (*Virtualized Network Functions* - VNF) executadas em hardware de propósito geral, flexibilizando a gerência e operação da rede [Fulber-Garcia et al. 2019b, Fulber-Garcia et al. 2020, Flauzino et al. 2021, Venancio et al. 2021].

O trabalho propõe o *framework* NFV-TE (NFV-*Traffic Engineering*)[1], um *framework* parametrizável para geração de NFs de engenharia de tráfego. O operador entra com parâmetros do mecanismo desejado, consolidados em um arquivo JSON, a partir do qual é gerada a função virtual de rede correspondente. O código gerado é Python 3, linguagem escolhida devido à sua portabilidade e flexibilidade. O NFV-TE apresenta facilidade para instanciação e configuração de funções. O artigo descreve os diversos *policers* e *shapers* já presentes no NFV-TE. Os mecanismos de *policing* implementados 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* implementados são: *Single Rate Token Bucket Shaper* e *Leaky Bucket*.

Para demonstrar o uso do NFV-TE e o potencial impacto da engenharia de tráfego em uma rede de computadores, bem como a viabilidade de sua implementação através de VNFs, experimentos foram realizados utilizando VNFs geradas pelo NFV-TE com diferentes parâmetros. As VNFs são submetidas a quatro perfis típicos e distintos 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 NFs geradas pelo NFV-TE, sendo estes apresentados e discutidos no decorrer deste trabalho.

O restante do artigo está organizado da seguinte forma. A Seção 2 apresenta uma visão geral sobre engenharia de tráfego e apresenta trabalhos relacionados. A Seção 3 discorre sobre como o paradigma NFV é utilizado para a engenharia de tráfego; descrevendo a implementação e funcionamento do NFV-TE. A Seção 4 detalha experimentos realizados com as NFs geradas pelo NFV-TE, apresentando a análise de cada resultado obtido. Por fim, a conclusão é apresentada na Seção 5, junto às propostas de trabalhos futuros.

2. Engenharia de Tráfego: Visão Geral e Trabalhos Relacionados

O aumento na oferta de serviços e a crescente popularização das redes, em particular a Internet, trouxeram diversos desafios para a operação e gerência das redes. Uma demanda extremamente alta traz o desafio de manter o desempenho da rede em níveis aceitáveis, o que não é tarefa trivial [Pedreno-Manresa et al. 2017]. Neste cenário, a engenharia de tráfego se torna atividade essencial do gerenciamento, buscando assegurar a qualidade de serviço para aplicações individualmente e o desempenho da rede. Em última instância, a engenharia de tráfego pode ser compreendida como a responsável por manipular e aperfeiçoar os níveis de utilização das redes. Nesse sentido, a determinação adequada de estratégias de tratamento de tráfego é fundamental para alcançar desempenho globalmente satisfatório, com utilização razoável dos recursos das redes.

[1]Disponibilizado em https://github.com/jvmoreira/multiservice-networks

Há diversos mecanismos na engenharia de tráfego, entretanto, dois dos mais relevantes são os *policers* e *shapers*. Em geral, o objetivo de um *policer* é assegurar que o tráfego passando pela rede não exceda taxas máximas acordadas, descartando pacotes excedentes [Flach et al. 2016]. Por outro lado, os *shapers* também definem taxas máximas, buscando suavizar os perfis de tráfego, porém, ao contrário dos *policers* que diretamente descartam pacotes, os *shapers* utilizam filas para postergar a transmissão de pacotes para momentos futuros em que as condições apropriadas forem atingidas [Marcon et al. 2011].

Trabalhos relacionados ao uso de NFV para engenharia de tráfego incluem a estratégia de Lavagem de Pacotes (*Packet Wash*) de Dong e Clemm [Dong and Clemm 2021], que objetiva garantir limites de 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 redução do tamanho dos pacotes, descartando seletivamente pedaços da carga útil do mesmo, sem haver interrupção da entrega. 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.

Um sistema para mitigar ataques distribuídos de negação de serviço (DDoS — *Distributed Denial of Service*) é proposto por Garcia e co-autores [Fulber-Garcia et al. 2018]. O sistema, chamado DeMONS, utiliza os conceitos de NFV em conjunto com alocação dinâmica e um mecanismo de reputação no processamento de tráfego da rede. O DeMONS é uma solução híbrida composta por cinco módulos principais, sendo um dele um *policer*. Fluxos de rede são marcados com uma reputação entre 0 e 1. Os fluxos com reputação 0 são bloqueados no *firewall*. Já fluxos marcados com reputação maior que 0 são alocados em diferentes túneis, de baixa e alta prioridade. O *policer*, em particular, limita o tráfego de cada fluxo de rede no canal de baixa prioridade baseando-se em suas prioridades. Os resultados obtidos no trabalho demonstram a viabilidade do DeMONS em mitigar eficientemente ataques DDoS.

O NFV-TE diferente da estratégia de Lavagem de Pacotes não realiza modificações nos pacotes a trafegar pela rede, facilitando sua implantação em redes de produção, sem ser necessário adaptar dispositivos e softwares para a sua utilização. Além disso, o NFV-TE oferece diferentes mecanismos de *policing* e *shaping*, principalmente os clássicos, utilizáveis de forma holística em serviços de rede. Fato que o difere se comparado ao DeMONS, o qual utiliza um *policer* específico para o mitigador, ou seja, não é um mecanismo genérico que pode ser usado em qualquer cenário.

3. O *Framework* NVF-TE

O desenvolvimento do *framework* foi feita utilizando a linguagem de programação *Python 3*. O Python 3 foi usado não apenas para programar o NFV-TE, mas também é a linguagem das funções de rede geradas. As funções virtuais de rede para engenharia de tráfego são especificadas de maneira a facilitar os esforços dos operadores de redes. Uma *interface* gráfica simples permite que sejam fornecidos os dados utilizados para gerar um arquivo *JSON* com parâmetros do mecanismo desejado. Este arquivo inclui campos essenciais, como a categoria da função de engenharia de tráfego (atualmente, *policing* ou

shaping); o nome da função de rede desejada, além de um conjunto de parâmetros específicos de cada categoria para a configuração da NF. Nesse arquivo, devem ser definidos também os nomes das *interfaces* de rede do cliente e do servidor dos quais os pacotes serão enviados e recebidos.

Após a entrada dos dados, a subsequente validação dos parâmetros, e a criação do arquivo JSON, é feita a geração de código-fonte da função de rede designada utilizando variáveis com os valores correspondentes aos parâmetros definidos e gerando como resultado um arquivo com a extensão ".py". Leves adaptações no código-fonte da função gerada podem ser necessárias para viabilizar a sua execução em qualquer plataforma de VNF [Marcuzzo et al. 2017, Fulber-Garcia et al. 2019a, Fulber-Garcia et al. 2019b]. As funções de rede e seus parâmetros são descritos nas Seções 3.1 e 3.2.

3.1. *Policers*

Os *policers* são implementados utilizando dois conceitos principais: o *token* que representa um *byte* de um pacote e o *bucket* que é uma estrutura utilizada para armazenar os *tokens* até uma quantidade máxima, chamada rajada (do inglês, *burst*). Os *tokens* vão sendo consumidos a cada pacote processado, havendo reposição de acordo com um critério definido para cada *policer*.

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

O *policer* srTBP atua verificando o tamanho de cada pacote, transmitindo-os apenas caso haja *tokens* suficientes no *bucket*. Os parâmetros de configuração necessários para a geração de um srTBP são: "*bucket_size*"que corresponde ao tamanho inicial do *bucket*; "*bucket_max_size*"que corresponde ao tamanho máximo do *bucket*; "*interval*"indicando o valor em segundos do intervalo entre cada reposição de *tokens*; e "*rate*"que indica à quantidade de *tokens* que são adicionados a cada intervalo.

O funcionamento do srTCM-*Policer* 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 realizada.

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.

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 realizada.

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 indicam o número máximo de *tokens* para os *buckets* P e C, em ordem.

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 descrevem a quantidade de *tokens* que são adicionados aos *buckets* C e P, respectivamente, a cada intervalo.

3.2. *Shapers*

Ao contrário dos *policers* que diretamente descartam pacotes, os *shapers* utilizam filas para postergar a transmissão de pacotes para momentos futuros em que as condições apropriadas forem atingidas. 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* (srTBS) e *Leaky Bucket* (LB).

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, para ser consumido à medida que novos *tokens* são adicionados ao *bucket*.

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 indica o tamanho máximo da fila de pacotes.

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.

Os parâmetros de configuração necessários para o *Leaky Bucket* são: "*bucket_max_size*"que indica o 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.

4. Experimentos

O ambiente para a execução dos experimentos relacionados às funções do NFV-TE é composto por três máquinas virtuais, todas executando o sistema Linux Ubuntu 18.4, CPU Intel i7-8750H e 4GB de RAM. A primeira VM assume a função de cliente, ou seja, envia um fluxo de pacotes para um servidor. A segunda VM assume a função de servidor, que recebe e processa os pacotes enviados pelos clientes. Por fim, situada entre o cliente e servidor, uma VM que executa a VNF gerada pelo *framework* NFV-TE, realizando o *policing* ou *shaping* do tráfego de pacotes entre o cliente e o servidor.

Para verificar o funcionamento dos mecanismos de *policing* e *shaping*, foram definidos quatro perfis de tráfegos para testes, sendo eles: Alfa [Sarvotham et al. 2001], Beta [Sarvotham et al. 2001], Elefante [Hamdan et al. 2020] e Guepardo [Maji et al. 2017]. O Alfa tem como característica o envio de 10 pacotes por segundo e uma pausa de 2 segundos após o envio dos mesmos. O Beta tem o envio de 10 pacotes por segundo, mas o tamanho do pacote se altera após o envio dos mesmos. O Elefante tem um fluxo contínuo de pacotes maiores, mas em menor quantidade em relação aos demais perfis. Já o Guepardo tem maior envio de pacotes e são menores em quantidade de Bytes se comparados aos outros. A Tabela 1 apresenta informações sobres os perfis 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 do Pacote (Bytes)	**Pacotes por Segundo**	**Intervalo entre Envios (s)**	**Bytes por Segundo**
Alfa	348	10 ou 0*	0,1 (+2 a cada 10 pacotes)	3480 ou 0*
Beta	148 ou 348 (fluxos alternados)	10	0,1	1480 ou 3480 (fluxos alternados)
Elefante	1048	5	0,2	5240
Guepardo	68	20	0,05	1360
10 Mbps	1250	1000	0,001	1250000

*Serão transmitidos 0 pacotes (0 bytes) se o tráfego estiver no hiato de 2s sem transmissão

Tabela 1. Os diferentes perfis de tráfego utilizados.

Os experimentos utilizam intervalos de tempo de 1 segundo tanto para a adição de *tokens* aos *buckets* ou para o consumo do *bucket*. Os *buckets* são inicializados com sua capacidade máxima de *tokens*. Além disso, o número de pacotes transmitidos pelos tráfegos Alfa, Beta, Elefante e Guepardo será sempre de 500 pacotes. Já para os experimentos com tráfego de 10Mbps são transmitidos 10 mil pacotes.

O primeiro experimento permite validar o comportamento operacional de alguns dos principais tipos de funções de rede geradas pelo *framework* NFV-TE quando submetidos a diferentes cenários de tráfego. Para fazer isso, consideramos um tamanho máximo de *bucket* padrão de 4000 *tokens* e alternamos a configuração do *rate* entre 1500 e 3900 *tokens* por segundo tanto para o *policer* srTBP. Já para o *shaper* LB alternamos a taxa constante nos valores de 3, 7, 13 e 25 pacotes. Após instanciadas no ambiente de testes, essas funções processaram tráfego conforme os perfis previamente declarados. Para cada cenário considerado, as quantidades de pacotes transmitidos com sucesso entre o cliente e o servidor, considerando as configurações preestabelecidas, são apresentadas, respectivamente, nas Figuras 1 e 2.

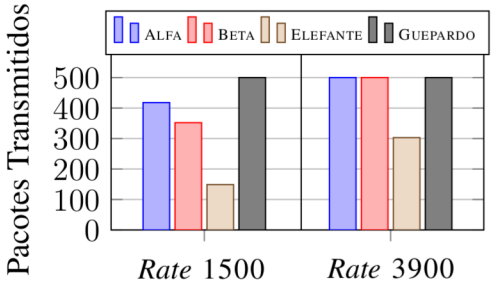

Figura 1. srTBP (*Bucket* 4000)

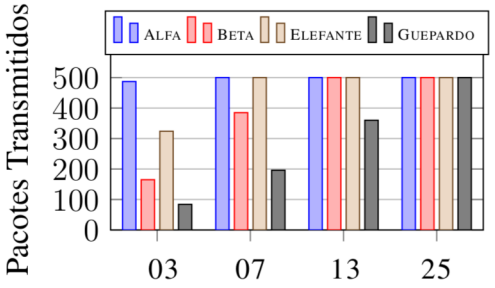

Figura 2. LB (Taxa Constante)

Na Figura 1 é possível verificar que para o tráfego Guepardo, o mecanismo de srTBP não executou descartes de pacotes, encaminhando a totalidade dos mesmos do cliente ao servidor, independente do valor do *rate*. Por outro lado, o srTBP com *rate* igual à 1500 *tokens*, executou transmissões de 86,6%, 70,4% e 29,8% da totalidade dos pacotes, para os tráfegos, respectivamente, Alfa, Beta e Elefante. Entretanto, ao aumentar o *rate* para 3900, o srTBP realizou a transmissão de 100% dos pacotes para os tráfegos Alfa e Beta, e atingiu 60,6% de pacotes transmitidos para o tráfego Elefante. Portanto, aumentar a quantidade de *tokens* que são adicionados ao *bucket* a cada intervalo de tempo gera um aumento no número de transmissões realizadas pelo srTBP para os diferentes perfis de tráfegos.

Na Figura 2 é apresentado que o LB para o tráfego Guepardo transmite 16,8%, 39,2%, 72% e 100%, quando a taxa constante é igual a 3, 7, 13 e 25, respectivamente. Fato que ocorre devido à alta quantidade de pacotes enviadas por segundo pelo tráfego. Para o tráfego Alfa e Elefante, quando a taxa constante é igual a 3, atinge-se 97,4% e 64,8% de transmissões dos pacotes recebidos, respectivamente, aumentando a taxa constante, ambos os tráfegos atingem os 100% de transmissão. Já para o tráfego Beta, não se atinge 100% de transmissões quando a taxa constante é igual a 3 e 7, atingindo 33% e 77% de transmissões, respectivamente. Isto se deve ao fato de que o tráfego Beta transmite 10 pacotes por segundo, valor inferior a essas taxas constantes.

Além da validação operacional das funções de rede através dos testes com diferentes perfis de tráfego, um segundo teste onde o cliente envia 10 Mbps de tráfego realizado como um experimento focado em um maior *throughput*. Para o srTBP, os experimentos de *throughput* utilizaram os valores de rajada iguais a 700 mil, 900 mil e 1,1 milhões de *tokens*, sendo a taxa de reposição de *tokens* ao *bucket* igual a 1,1 milhões de *tokens* por segundo. Os resultados obtidos são apresentados na Figura 3.

Ao analisar o experimento com o srTBP 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 uma taxa de transmissão de 8,6 Mbps, em média. Por fim, para a rajada igual a 1,1 milhão, mesmo valor que a reposição de *tokens* ao *bucket*, é obtido uma taxa de transmissão de 10 Mbps, ou seja, todos os pacotes recebidos são transmitidos.

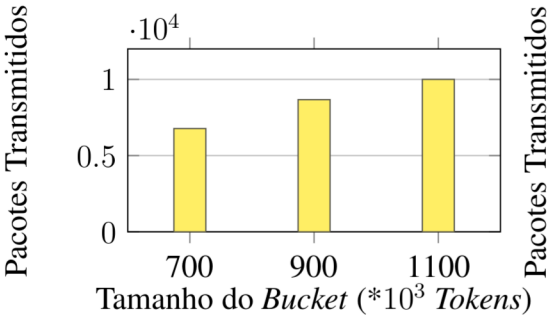

Figura 3. srTBP (10Mbps)

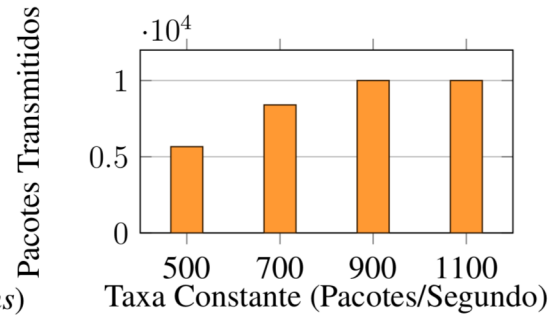

Figura 4. LB (10Mbps)

Já os experimentos de *throughput* para o LB utilizaram-se taxas de 500, 700, 900 e 1100 pacotes por segundo. Os resultados obtidos são apresentados na Figura 4. 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.

Através dos resultados experimentais apresentados, é possível observar como os mecanismos implementados de engenharia de tráfego atuam nos fluxos da rede, validando desta forma o comportamento dos *shapers* e *policers* no processamento de pacotes.

5. Conclusão

Este trabalho apresentou o NFV-TE, um *framework* parametrizável para geração de funções de rede voltadas para engenharia de tráfego, desenvolvido no contexto do paradigma NFV. O NFV-TE é extensível, gerando funções virtuais de rede para mecanismos de engenharia de tráfego como *shapers* e *policers* de forma parametrizável. Avaliamos as funções geradas pelo NFV-TE através de um conjunto de experimentos, demonstrando que as mesmas apresentam o funcionamento esperado.

Como trabalhos futuros nosso objetivo é disponibilizar o NFV-TE através de *marketplaces* [Xilouris et al. 2014, Bondan et al. 2019] para ser usado em ambiente de produção. Também, é objetivo incluir outros mecanismos de engenharia de tráfego, como os *schedulers*. Além disso, busca-se 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. Por fim, a aplicação da estratégia de geração automática de código para funções virtuais de rede também deve ser avaliada no contexto de outras áreas de aplicação no paradigma NFV, incluindo rastreamento de pacotes IP [Hilgenstieler et al. 2007], detecção de falhas [Turchetti and Duarte 2017] e emulação e implantação de serviços de rede virtualizados [Tavares et al. 2018, Fulber-Garcia et al. 2021].

Referências

Bondan, L., Franco, M. F., Marcuzzo, L., Venancio, G., Santos, R. L., Pfitscher, R. J., Scheid, E. J., Stiller, B., De Turck, F., Duarte, E. P., et al. (2019). Fende: marketplace-based distribution, execution, and life cycle management of vnfs. *IEEE Communications Magazine*, 57(1):13–19.

Cavalca, U., Mesquita, C., Pereira, A. C., and Carrano, E. G. (2014). A methodology for traffic shaping optimization in next generation networks. *International Journal of Computer Information Systems and Industrial Management Applications*, 6:474–483.

Daniel-Simion, D. and Dan-Horia, G. (2011). Traffic shaping and traffic policing impacts on aggregate traffic behaviour in high speed networks. In *IEEE International Symposium on Applied Computational Intelligence and Informatics*, pages 465–467. IEEE.

Dong, L. and Clemm, A. (2021). High-precision end-to-end latency guarantees using packet wash. In *IFIP/IEEE Int. Symp. Integrated Network Management*, pages 259–267. IEEE.

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

Flach, T., Papageorge, P., Terzis, A., Pedrosa, L., Cheng, Y., Karim, T., Katz-Bassett, E., and Govindan, R. (2016). An internet-wide analysis of traffic policing. In *ACM SIGCOMM Conference*, pages 468–482. ACM.

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

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

Fulber-Garcia, V., Duarte, E. P., Huff, A., and dos Santos, C. R. (2020). Network service topology: Formalization, taxonomy and the custom specification model. *Computer Networks*, 178:107337.

Fulber-Garcia, V., Huff, A., Marcuzzo, L. d. C., Luizelli, M. C., Schaeffer-Filho, A. E., Granville, L. Z., dos Santos, C. R., and Duarte, E. P. (2021). Customizable deployment of nfv services. *Journal of Network and Systems Management*, 29(3):1–27.

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

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

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

Hilgenstieler, E., Duarte, E. P., Mansfield-Keeni, G., and Shiratori, N. (2007). Improving the precision and efficiency of log-based ip packet traceback. In *IEEE Global Telecommunications Conference*, pages 1823–1827. IEEE.

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

Marcon, M., Dischinger, M., Gummadi, K. P., and Vahdat, A. (2011). The local and global effects of traffic shaping in the internet. In *International Conference on Communication Systems and Networks*, pages 1–10. IEEE.

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

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

Pedreno-Manresa, J.-J., Khodashenas, P. S., Siddiqui, M. S., and Pavon-Marino, P. (2017). Dynamic qos/qoe assurance in realistic nfv-enabled 5g access networks. In *International Conference on Transparent Optical Networks*, pages 1–4. IEEE.

Sarvotham, S., Riedi, R., and Baraniuk, R. (2001). Connection-level analysis and modeling of network traffic. In *ACM SIGCOMM Workshop on Internet Measurement*, page 99–103. ACM.

Tavares, T. N., da Cruz Marcuzzo, L., Fulber-Garcia, V., de Souza, G. V., Franco, M. F., Bondan, L., De Turck, F., Granville, L., Duarte, E. P., dos Santos, C. R. P., et al. (2018). Niep: Nfv infrastructure emulation platform. In *IEEE International Conference on Advanced Information Networking and Applications*, pages 173–180. IEEE.

Turchetti, R. C. and Duarte, E. P. (2017). Nfv-fd: Implementation of a failure detector using network virtualization technology. *International Journal of Network Management*, 27(6):e1988.

Venancio, G., Fulber-Garcia, V., da Cruz Marcuzzo, L., Tavares, T. N., Franco, M. F., Bondan, L., Schaeffer-Filho, A. E., Paula dos Santos, C. R., Granville, L. Z., and P. Duarte, E. P. (2021). Beyond vnfm: Filling the gaps of the etsi vnf manager to fully support vnf life cycle operations. *International Journal of Network Management*, 31(5):e2068.

Xilouris, G., Trouva, E., Lobillo, F., Soares, J. M., Carapinha, J., McGrath, M. J., Gardikis, G., Paglierani, P., Pallis, E., Zuccaro, L., et al. (2014). T-nova: A marketplace for virtualized network functions. In *European Conference on Networks and Communications*, pages 1–5. IEEE.