NFV-TE: Uma Ferramenta para a Geração Automática de Funções Virtuais Rede para Engenharia de Tráfego

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

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

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

Abstract. *The unprecedented growth of computer and telecommunication networks has led to an extraordinary increase in network traffic. Traffic engineering consists of mechanisms related to monitoring, characterizing, modeling and controlling network traffic, in order to guarantee network performance. This work describes NFV-TE, a tool for generating virtual network functions for traffic engineering. The NFV (Network Function Virtualization) paradigm brings flexibility to networks by allowing the implementation of core functions on a software plane. After the network operator enters a set of features 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. This paper describes various policers and shapers generated by NFV-TE, describing their execution in different scenarios and under different network traffic profiles.*

Resumo. *A expansão sem precedentes das redes levou a um extraordinário aumento do tráfego. A engenharia de tráfego consiste de mecanismos relacionados ao monitoramento, caracterização, modelagem e controle do tráfego, visando garantir o bom desempenho da rede. Este trabalho descreve a ferramenta NFV-TE para geração de funções virtuais de rede para engenharia de tráfego. O paradigma NFV (Network Function Virtualization) traz grande flexibilidade às redes ao permitir a implementação de funções do núcleo em um plano de software. A partir de dados fornecidos na entrada pelo operador de rede, o NFV-TE faz a consolidação em um arquivo JSON, e gerada a função virtual de rede correspondente. O sistema foi projetado para apresentar grande facilidade para inclusão e configuração de funções. O artigo descreve diversos policers e shapers gerados 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 consiste em mecanismos relacionados ao monitoramento, caracterização, modelagem e controle do tráfego visando garantir o bom desempenho da rede como um todo, além de manter a utilização dos recursos computacionais em níveis desejáveis e com previsibilidade [Nucci and Papagiannaki 2009]. Os mecanismos permitem classificar e tratar fluxos em diferentes categorias através de políticas específicas, seu emprego viabiliza uma rede mais robusta e eficiente. Dentre os principais mecanismos existentes estão os *policers* e os *shapers* [Evans and Filsfils 2010]. *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, o que flexibiliza a gerência e operação da rede [Fulber-Garcia et al. 2019, Fulber-Garcia et al. 2020]. O objetivo é disponibilizar uma alternativa às estratégias convencionais. Chega a ser surpreendente como, apesar do grande potencial de uso de NFV no contexto de engenharia de tráfego, um número muito reduzido de esforços foi realizado neste sentido.

O trabalho propõe o *framework* NFV-TE (NFV-*Traffic Engineering*)[1], uma ferramenta para a geração automática de VNFs de engenharia de tráfego. O operador entra com parâmetros do mecanismo desejado, que são 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[2] e o potencial impacto da engenharia de tráfego em uma rede de computadores, bem como a viabilidade de sua implementação através de VNFs, experimentos são apresentados em que as VNFs geradas pelo NFV-TE são configuradas com diferentes parâmetros e 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 VNFs geradas demonstrando o potencial da NFV-TE.

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 trouxeram diversos desafios para sua operação e gerência. Uma demanda extremamente alta traz o

[1]Documentação, manual e código-fonte disponíveis em `https://github.com/jvmoreira/multiservice-networks`

[2]Vídeo de demonstração disponível em `https://www.youtube.com/watch?v=ju_UcMM2z0Y`

desafio de manter o desempenho da rede em níveis aceitáveis, o que não é tarefa trivial [Nucci and Papagiannaki 2009]. Neste cenário, a engenharia de tráfego se torna atividade essencial do gerenciamento, buscando assegurar o uso eficiente de recursos e o desempenho da rede como um todo. Em termos de monitoramento e controle de tráfego, o objetivo é mitigar os fatores que impactam a eficácia do tráfego, evitando congestionamentos, minimizando atrasos e perda de pacotes, e maximizando a vazão. A engenharia de tráfego também visa garantir da rede.

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 [Evans and Filsfils 2010], descartando pacotes excedentes. 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.

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 busca garantir limites de latência ponto-a-ponto com alta precisão, reduzindo o descarte de pacotes quando a rede se encontra congestionada. A ideia é reduzir o tamanho dos pacotes, 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 objetivo é reduzir o tempo de permanência (*dwell time*) dos pacotes nos roteadores ao ter o risco do mesmo sofrer atraso.

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 composto por cinco módulos principais, que são VNFs: 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 *policer* limita o tráfego de cada fluxo de rede baseando-se em suas prioridades. O módulo gerenciador é responsável pelo ciclo de vida de todos os módulos.

O NFV-TE diferente da estratégia de Lavagem de Pacotes não realiza modificações nos pacotes que trafegam pela rede, o que facilita sua implantação em redes de produção, sem que seja necessário adaptar dispositivos e softwares para a sua utilização. Além disso, o NFV-TE oferece diferentes mecanismos de *policing* e *shaping*, principalmente os clássicos, sendo 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 para ser usado em qualquer cenário.

3. A Ferramenta NVF-TE

Esta seção apresenta a ferramenta NFV-TE para geração de funções virtuais de rede para mecanismos de engenharia de tráfego. A ferramenta foi implementada utilizando a linguagem *Python 3*, caracterizada por sua flexibilidade e ampla portabilidade. 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 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"que pode ser executado em dispositivos com suporte à linguagem. Adaptações no código fonte da função gerada podem ser necessárias para viabilizar a sua execução em uma plataforma de VNF específica [Fulber-Garcia et al. 2019]. A seguir são descritos diversos *policers* e *shapers* gerados pela ferramenta.

3.1. *Policers*

Como mencionado anteriormente, os *policers* permitem assegurar que o tráfego passando pela rede não exceda taxas máximas acordadas. 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 de 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*". O Artigo descreve funções de rede geradas 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. Estes *policers* são descritos a seguir.

O *policer* srTBP atua verificando o tamanho de cada pacote, transmitindo-os apenas caso haja *tokens* suficientes no *bucket*. 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 executada.

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.

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 geradas as versões *Color-Aware* do srTCM e do trTCM e ambas foram construídas de forma similar.

3.2. *Shapers*

Como mencionado anteriormente, os *shapers*, assim como os *policers*, 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. 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*.

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.

4. Experimentos

Esta seção descreve experimentos executados com o objetivo de avaliar e validar as funções de rede geradas pelo *framework* NFV-TE quando submetidos a diferentes perfis de tráfego. O ambiente para realização dos experimentos 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], parametrizados como mostra a Tabela 1.

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

	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.

Alfa, Beta, Elefante e Guepardo será sempre de 500 pacotes. Já para os experimentos com tráfego de 10Mbps seão transmitidos 10 mil pacotes.

O primeiro experimento permite validar alguns dos principais tipos de funções de rede geradas pelo 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 de acordo com 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.

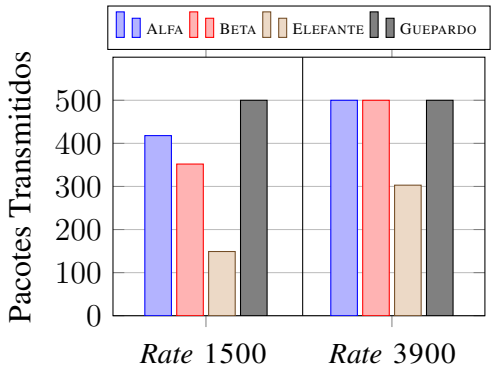

Figura 1. srTBP com tamanho do *Bucket* igual a 4000.

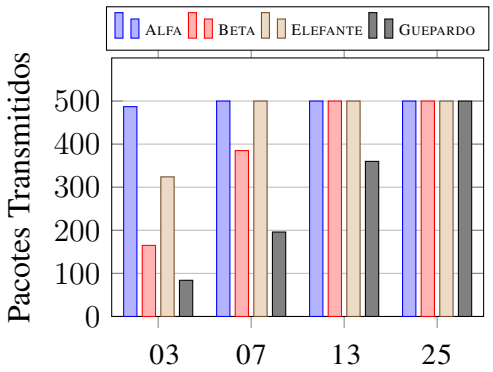

Figura 2. LB alterando a taxa constante.

Na Figura 1 observa-se que para o tráfego Guepardo, o mecanismo de srTBP não executou descartes de pacotes, encaminhando a totalidade destes 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.

A Figura 2 mostra 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 verifica-se 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 (todos os pacotes recebidos são transmitidos).

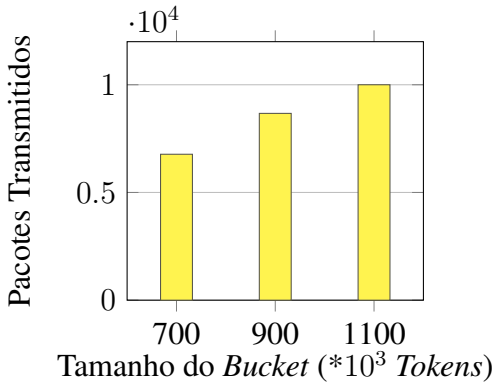

Figura 3. srTBP (10Mbps)

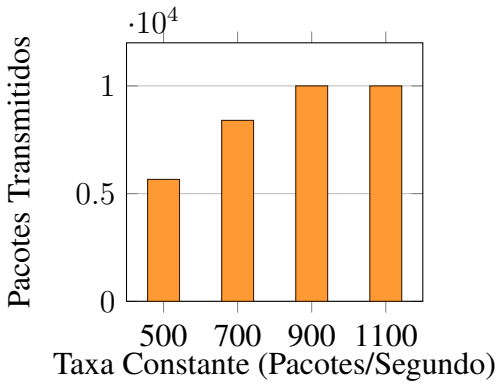

Figura 4. LB (10Mbps)

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.

5. Conclusão

A engenharia de tráfego é parte essencial do gerenciamento e operação de redes de computadores, garantindo seu desempenho e permitindo manter os níveis necessários de qualidade de serviço. Este trabalho propõe o NFV-TE, uma ferramenta para a geração de funções de rede voltadas para engenharia de tráfego, desenvolvido no paradigma NFV. No trabalho são apresentados resultados experimentais para diversos *shapers* e *policers* gerados. Como principal trabalho futuro, nosso objetivo é disponibilizar o NFV-TE através

de *marketplaces* [Bondan et al. 2019] para ser usado em ambiente de produção. É objetivo também incluir outros mecanismos de engenharia de tráfego, como os *schedulers*. 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. Por fim, a aplicação da estratégia de geração automática de código para funções virtuais de rede também deve ser avaliada para outras áreas de aplicação no paradigma NFV.

Para a demonstração da ferramenta o ambiente de experimentação presente na Seção 4 será utilizado. A demonstração consistirá na apresentação e utilização da interface do NFV-TE para configurar dois mecanismos de engenharia de tráfego: o srTBP e o LB. Posteriormente, as respectivas NFs geradas pelo *framework* serão executadas, sendo os resultados exibidos ao público e discutidos.

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.

Dong, L. and Clemm, A. (2021). High-precision end-to-end latency guarantees using packet wash. In *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.

Fulber-Garcia, V., Duarte Jr, 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., Gaiardo, G. d. F., 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., 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. (2019). On the design of a flexible architecture for virtualized network function platforms. In *2019 IEEE Global Communications Conference (GLOBECOM)*, 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.

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 *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–7.

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.

Sarvotham, S., Riedi, R., and Baraniuk, R. (2001). Connection-level analysis and modeling of network traffic. IMW '01, page 99–103, New York, NY, USA. ACM.