UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO - BACHARELADO

Arthur Medeiros de Assis Brasil

PROJETO E DESENVOLVIMENTO DE UM PROCESSADOR NSH PARA SERVIÇOS ENCADEADOS DE REDE

Santa Maria, RS
2018

Arthur Medeiros de Assis Brasil

PROJETO E DESENVOLVIMENTO DE UM PROCESSADOR NSH PARA SERVIÇOS ENCADEADOS DE REDE

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação - Bacharelado da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de **Bacharel em Ciência da Computação**

Orientador: Prof. Dr. Carlos Raniery Paula dos Santos

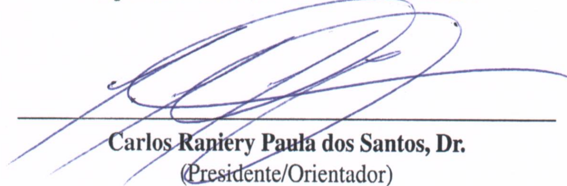
453
Santa Maria, RS
2018

Arthur Medeiros de Assis Brasil

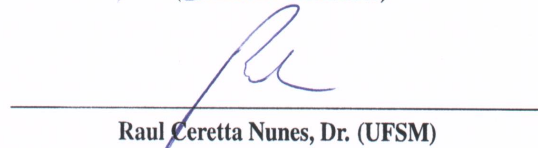
PROJETO E DESENVOLVIMENTO DE UM PROCESSADOR NSH PARA SERVIÇOS ENCADEADOS DE REDE

Trabalho de Conclusão de Curso apresentado ao Curso de Ciência da Computação - Bacharelado da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de **Bacharel em Ciência da Computação**

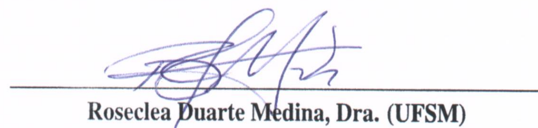
Aprovado em 18 de Dezembro de 2018:



Carlos Raniery Paula dos Santos, Dr.
(Presidente/Orientador)



Raul Ceretta Nunes, Dr. (UFSM)



Roseclea Duarte Medina, Dra. (UFSM)

Santa Maria, RS

2018

// RESUMO

PROJETO E DESENVOLVIMENTO DE UM PROCESSADOR NSH PARA SERVIÇOS ENCADEADOS DE REDE

AUTOR: ARTHUR MEDEIROS DE ASSIS BRASIL
ORIENTADOR: CARLOS RANIERY PAULA DOS SANTOS

Uma das maiores dificuldades da indústria de telecomunicações é a dependência de recursos físicos próprios para a execução de seus serviços. A Virtualização de Funções de Rede (NFV - *Network Functions Virtualization*) surgiu para solucionar este problema. A ideia de NFV é desacoplar os equipamentos físicos de rede das funções que eles executam. Com o advento de NFV, as instâncias de serviços de rede são executadas em várias nuvens para fins de desempenho e balanceamento de carga. Nesse contexto, é definido o Encadeamento de Serviços de Rede (SFC - *Service Function Chaining*), que é um mecanismo que permite que várias funções de rede sejam interconectadas para formar um serviço, permitindo que as operadoras se beneficiem da infraestrutura definida por software virtualizado. Para suprir as necessidades do SFC, surge o *Network Service Header* (NSH), um protocolo de plano de danos que atua como um encapsulamento para SFCs. Assim, este trabalho tem por objetivo o desenvolvimento de um Processador NSH para Serviços Encadeados de Rede. Para isto, foi realizado um levantamento dos requisitos necessários para a implantação deste processador, seguido pela escolha de ferramentas que atendessem estes requisitos de forma efetiva. Por fim, como não há implementações disponíveis de Processadores NSH até o momento deste trabalho, o processador desenvolvido foi avaliado e julgado qualitativamente.

Palavras-chave: NFV. Serviços encadeados de rede. NSH.

ABSTRACT

PROJECT AND DEVELOPMENT OF AN NSH PROCESSOR FOR SERVICE FUNCTION CHAINING

AUTHOR: ARTHUR MEDEIROS DE ASSIS BRASIL
ADVISOR: CARLOS RANIERY PAULA DOS SANTOS

One of the major difficulties of the telecommunications industry is the dependence of physical resources for the execution of its services. Network Function Virtualization (NFV) has come up to solve this problem. The idea of NFV is to decouple physical network equipment from the functions they perform. With the advent of NFV, network service instances run on multiple clouds for performance and load balancing purposes. In this context, Service Function Chaining (SFC) is defined, which is a mechanism that allows multiple network functions to be interconnected to form a service, allowing operators to benefit from the infrastructure defined by virtualized software. To meet the needs of the SFC, the Network Service Header (NSH) emerge, a data-plane protocol that acts as an encapsulation for SFCs. Thus, this work aims at the development of an NSH Processor for Service Function Chaining. For this, a survey was made of the necessary requirements for the implantation followed by the choice of tools that met these requirements effectively. Finally, since there are no available implementations of NSH Processors up to the time of this work, the developed processor was evaluated and judged qualitatively.

Keywords: NFV. Service Function Chaining. NSH.

LISTA DE FIGURAS

Figura 2.1 – Arquitetura NFV 12
Figura 2.2 – Arquitetura SFC 14
Figura 2.3 – Cabeçalho Base 15
Figura 2.4 – Cabeçalho de Contexto de Comprimento Variável 18
Figura 3.1 – Arquitetura SFC com Processador NSH 22
Figura 3.2 – Interfaces do Processador NSH 23
Figura 3.3 – Cabeçalho Base com a *flag* PSH 24
Figura 4.1 – Diagrama de sequência: somente a SF 29
Figura 4.2 – Diagrama de sequência: SF e Processador NSH 30
Figura 4.3 – Diagrama de sequência: SF e Processador NSH com leitura de metadados . . 30
Figura 4.4 – Avaliação da vazão dos cenários 31
Figura 4.5 – Avaliação do atraso dos cenários 32

LISTA DE ABREVIATURAS E SIGLAS

BSS	*Business Support Systems*
CPU	*Central Processing Unit*
DHCP	*Dynamic Host Configuration Protocol*
EMS	*Element Management System*
ETSI	*European Telecommunications Standards Institute*
FIFO	*First In, First Out*
GB	*Gigabyte*
GS	*Group Specification*
IETF	*Internet Engineering Task Force*
IP	*Internet Protocol*
MANO	*Management and Orchestration*
MHz	*Megahertz*
MPLS	*Multiprotocol Label Switching*
NFV	*Network Functions Virtualization*
NFVI	*Network Functions Virtualization Infraestructure*
NSH	*Network Service Header*
OAM	*Operation, Administration and Maintenance*
OPNFV	*Open Platform for Network Functions Virtualization*
OSS	*Operation Support Systems*
RAM	*Random Access Memory*
RFC	*Request for Comments*
RGW	*Residential Gateways*
SF	*Service Function*
SFC	*Service Function Chaining*
SFF	*Service Function Forwarder*
SFP	*Service Function Path*
SI	*Service Index*
SPI	*Service Path Identifier*
TTL	*Time to Live*
UDP	*User Datagram Protocol*
VM	*Virtual Machine*
VNF	*Virtual Network Function*

SUMÁRIO

1 INTRODUÇÃO 8
2 FUNDAMENTAÇÃO TEÓRICA 10
2.1 ARQUITETURA NFV 10
2.2 *SERVICE FUNCTION CHAIN* 13
2.2.1 Arquitetura SFC 13
2.3 *NETWORK SERVICE HEADER* 15
2.3.1 *Network Service Header* (NSH) 15
2.4 TRABALHOS RELACIONADOS 19
3 SOLUÇÃO PROPOSTA 21
3.1 REQUISITOS 21
3.2 ARQUITETURA 22
3.3 IMPLEMENTAÇÃO 25
4 AVALIAÇÃO DE DESEMPENHO 29
4.1 AMBIENTE DE TESTE E METODOLOGIA 29
4.2 RESULTADOS OBTIDOS 31
4.3 DISCUSSÃO 32
5 CONCLUSÃO 34
REFERÊNCIAS 36

1 INTRODUÇÃO

Uma das maiores dificuldades da indústria de telecomunicações é a dependência de recursos físicos próprios para a execução de seus serviços. O custo destes recursos, o gasto de energia e espaço, o trabalho necessário para a instalação e o treinamento de pessoal especializado tornam o lançamento de um novo serviço um processo custoso e complexo, limitando a capacidade das operadoras em acompanhar as constantes demandas dos usuários por novas e sofisticadas aplicações.

Como solução deste problema, surgiu a Virtualização de Funções de Rede (NFV - *Network Functions Virtualization*). A ideia de NFV (2013) é desacoplar os equipamentos físicos de rede das funções que eles executam. Assim, essas funções podem ser instanciadas sem a necessidade de instalação de um novo equipamento. Por exemplo, operadores de rede podem executar um *firewall* de código aberto em uma máquina virtual rodando em um servidor genérico (*e.g.,* plataforma x86). Dessa forma, um determinado serviço pode ser decomposto em um conjunto de funções de rede virtual (VNFs - *Virtual Network Functions*), que podem ser implementadas em software e executadas em diferentes servidores físicos. As VNFs também podem ser realocadas e instanciadas em diferentes locais da rede sem necessariamente exigir a compra e instalação de novo hardware.

Com o advento de NFV, os serviços de rede podem ser executados em diversas infraestruturas para fins de desempenho e balanceamento de carga. A interconexão dessas instâncias para formar um serviço completo de rede de ponta a ponta é uma tarefa complexa, demorada e cara. O Encadeamento de Serviços de Rede (SFC - *Service Function Chaining*) (2015) é um mecanismo que permite que várias funções de rede sejam interconectadas para formar serviços avançados. Desta forma, SFC surge como um facilitador para NFV, que fornece uma solução flexível e econômica para os provedores de rede.

A transição para plataformas virtuais exige um modelo ágil para criar e entregar serviços. Em especial, três tarefas são necessárias: a migração de funções; a capacidade de vincular facilmente políticas do serviço à informações granulares; e a capacidade de direcionar o tráfego para as funções necessárias. Nesse contexto, foi proposto o *Network Service Header* (NSH) (2018), um protocolo de plano de danos que atua como um encapsulamento para SFCs. O NSH é projetado para encapsular um pacote ou quadro original e, por sua vez, ser encapsulado por um encapsulamento de transporte externo.

Por se tratar de um padrão definido muito recentemente, não foram encontradas implementações disponíveis que executem todas as tarefas exigidas pela especificação do NSH. Desta forma, este trabalho visa realizar o projeto e desenvolvimento de um Processador NSH para Serviços Encadeados de Rede. O Processador NSH desenvolvido executará todas as ações referentes a NSH que até então são realizadas pelas próprias funções de rede. O uso do NSH e do Processador NSH proverá: (*i*) encadeamento de serviços, (*ii*) transporte de metadados, (*iii*) padrão para encadeamento, (*iv*) independência de encapsulamento de transporte e (*v*) suporte à funções de rede agnósticas.

Este trabalho está organizado da seguinte forma: o Capítulo 2 apresenta uma fundamentação teórica dos conceitos de NFV, SFC e NSH, demonstrando suas arquiteturas, elementos e funções. Além disso, realiza uma revisão bibliográfica sobre NSH. No Capítulo 3, é descrita a proposta principal deste trabalho, o Processador NSH, citando suas principais características e atribuições. Além disso, serão detalhadas as tecnologias usadas e a forma como foram utilizadas. No Capítulo 4, será avaliado o desempenho do processador em alguns cenários, tendo como métricas a vazão e o atraso. Por fim, no Capítulo 5, será discutido os benefícios trazidos pela proposta desenvolvida, o conhecimento adquirido, considerações finais sobre o trabalho e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A primeira seção deste capítulo apresentará uma revisão sobre NFV e sua arquitetura segundo o ETSI. Na segunda seção será apresentado o conceito de *Service Function Chain* (SFC) bem como os elementos que o compõe conforme o padrão criado pela IETF. Na terceira, será abordado o *Network Service Header* (NSH), que é o encapsulamento SFC discutido pela IETF na criação do padrão para SFC. Por fim, na quarta seção, serão discutidos alguns trabalhos relacionados a este que foram anteriormente desenvolvidos.

2.1 ARQUITETURA NFV

O *European Telecommunications Standards Institute* (ETSI) definiu uma arquitetura para serviços e funções de rede baseados em NFV com o objetivo de torná-la padrão para futuras implementações. Essa arquitetura possui 3 principais blocos: *Virtual Network Function* (VNF), NFV *Infraestructure* (NFVI) e NFV *Management and Orchestration* (NFV MANO).

- ***Virtual Network Function*** **(VNF)**: Uma VNF é uma implementação de uma função de rede usando recursos virtuais, como por exemplo, uma *Virtual Machine* (VM). Exemplos de funções de rede são servidores DHCP, *firewalls, Residential Gateways (RGW)*, etc. Uma VNF pode ser implementada em uma única VM, bem como ter seus componentes internos implementados em diferentes VMs. O comportamento e interfaces de gerenciamento devem ser os mesmos encontrados em uma função de rede convencional.

- **NFV *Infraestructure* (NFVI)**: NFVI é o conjunto de recursos físicos e virtuais que possibilitam a implementação, gerenciamento e execução de VNFs. Os recursos de hardware podem ser CPU, armazenamento e rede. Os recursos de *software*, como as VMs, são abstrações dos recursos físicos. Essa abstração pode ser alcançada com o uso de *hypervisors*, que coordena a alocação dos recursos físicos para a implementação das VMs.

- **NFV *Management and Orchestration* (NFV MANO)**: O NFV MANO fornece as funcionalidades necessárias para o provisionamento e configuração da infraestrutura de VNFs. Assim, controla o ciclo de vida dos recursos físicos e virtuais que fazem parte da infraestrutura, bem como das próprias VNFs. Em resumo, o NFV MANO se preocupa com todas as tarefas de gerenciamento específicas da virtualização que possibilitam a existên-

cia das VNFs. Ainda, define interfaces que podem fazer a conexão do ambiente virtual com sistemas tradicionais de gerenciamento de rede, permitindo tanto o gerenciamento das VNFs, como das funções executando em equipamentos legados.

Além destes, na arquitetura de uma NFV existem outros elementos que são necessários para a virtualização e funcionamento do ambiente virtualizado. Esses elementos fazem parte dos blocos principais descritos acima ou se relacionam diretamente com eles. Os elementos são: *Element Management System* (EMS); recursos físicos e virtualizados; camada de virtualização; gerente de infraestrutura virtualizada; orquestrador; descrição de serviço, VNFs e infraestrutura; gerente de VNF; e sistemas de suporte à operação e negócios ou *Operation and Business Support Systems* (OSS/BSS). A figura 2.1 mostra esses elementos e a relação entre eles.

- ***Element Management System* (EMS)**: O *Element Management System* (EMS) realiza a funcionalidade de gerenciamento de uma ou mais VNFs. Ele que faz a comunicação entre uma ou mais VNFs e o gerente de VNF;

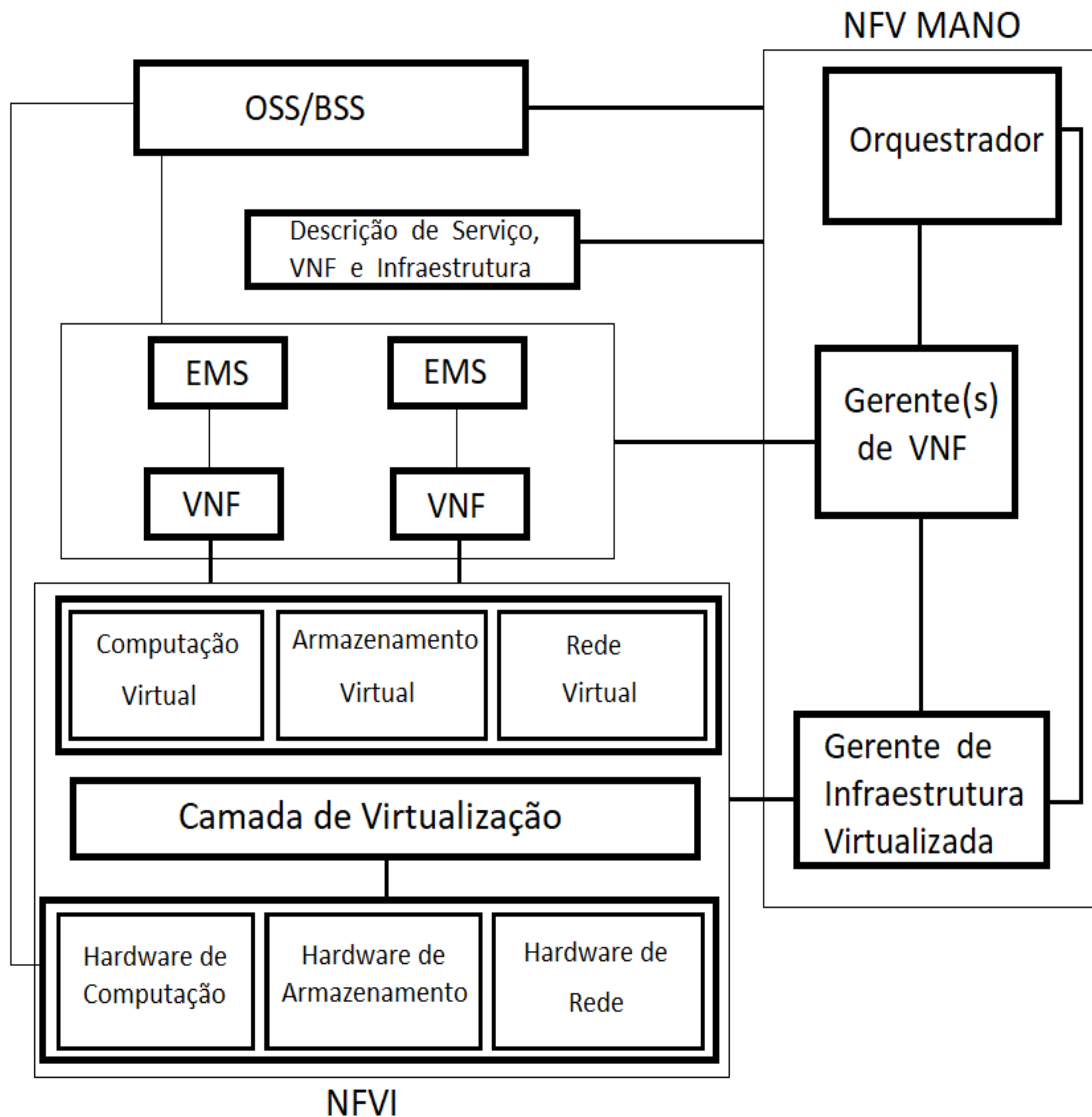
- **Recursos Físicos**: Estão situados na NFVI e incluem armazenamento, processamento e rede para a conexão com as VNFs através da camada de virtualização;

- **Camada de Virtualização e Recursos Virtualizados**: A camada de virtualização tem a função de abstrair o hardware para que as VNFs executem independentemente dos recursos físicos utilizados na NFVI. Ela permite que VNFs sejam executadas em diferentes recursos físicos. Geralmente, consegue-se essa abstração por meio de *hypervisors* e máquinas virtuais, mas não existe uma solução única e obrigatória para a implementação da camada de virtualização;

- **Gerente de Infraestrutura Virtualizada**: Faz o controle e gerenciamento das funcionalidades que possibilitam a conexão das VNFs com os recursos físicos e suas virtualizações, ou seja, é o gerente da NFVI. Algumas tarefas do gerente de infraestrutura virtualizada são, por exemplo, alocação de novas máquinas virtuais, melhorar a eficiência de energia, descobrir falhas e analisar a causa raiz, coletar informações para monitoramento e otimização. Podem existir vários gerentes de infraestrutura virtualizada em uma única implementação de NFV;

- **Orquestrador**: Realiza a orquestração e gerenciamento da infraestrutura e recursos virtuais na NFV. Algumas funções do orquestrador são: integrar novos pacotes de serviços

Figura 2.1: Arquitetura NFV



Fonte: Adaptado de ETSI GS NFV 002 (2013).

de rede e VNFs, gestão dos recursos de toda NFV, validação e autorização de solicitações de recursos da NFVI;

- **Descrição de Serviço, VNF e Infraestrutura**: Fornece informações sobre o modelo de implementação das VNFs, informações sobre a infraestrutura da NFV e informações relacionadas ao serviço;

- **Gerente de VNF**: O gerente de VNF coordena todo o ciclo de vida de um VNF, desde a instanciação até a paralisação. Podem existir um gerente de VNF para cada VNF ou um

gerente pode controlar várias VNFs;

- ***Operation and Business Support Systems*** **(OSS/BSS)**: É responsável pelo atendimento e garantia de serviços, faturamento, registro de problemas, entre outras funções. Possui foco nas questões de negócio da NFV.

2.2 *SERVICE FUNCTION CHAIN*

Segundo a RFC 7665, definida pelo *Internet Engineering Task Force* (IETF) (2015), *Service Function Chain* (SFC) é um conjunto ordenado de *Service Functions* (SFs) e restrições de ordenação que devem ser aplicadas a pacotes e/ou quadros e/ou fluxos selecionados como resultado de classificação. Vale ressaltar que, para o IETF, *Service Function* significa o mesmo que função de rede ou *Network Function* (NF), para o ETSI.

A principal vantagem do SFC é automatizar a configuração das conexões da rede virtual para os diferentes tráfegos que passam pelo serviço de rede. A IETF também definiu uma arquitetura para os componentes de um SFC. Essa arquitetura é detalhada na subseção seguinte.

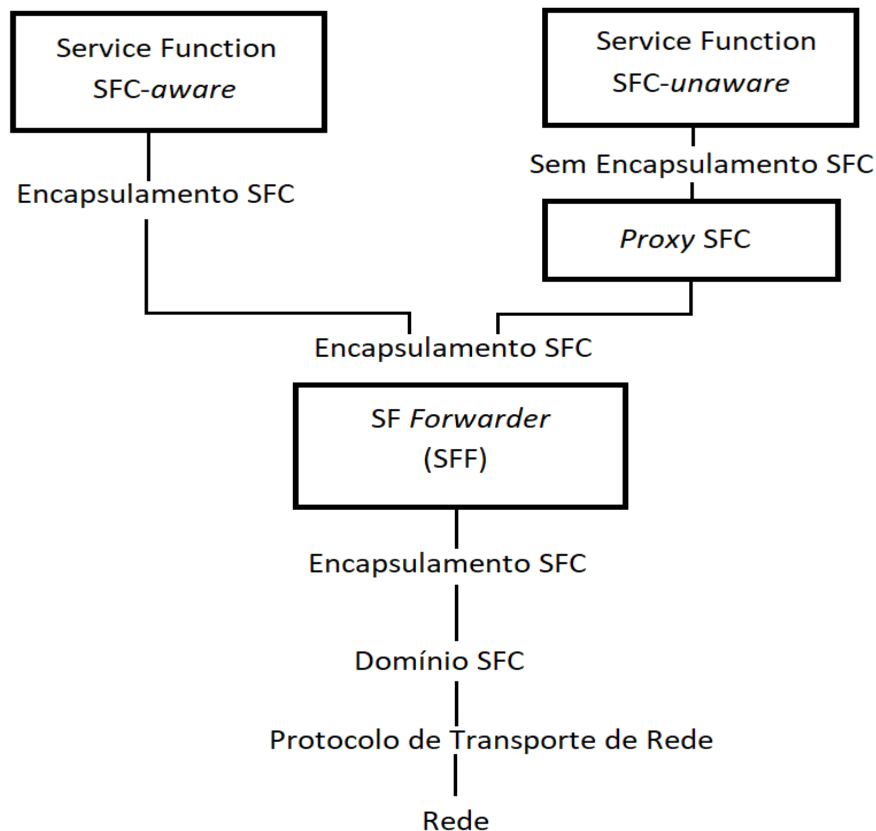
2.2.1 Arquitetura SFC

A arquitetura SFC é formada por elementos que são componentes lógicos. Esses componentes são classificadores, *Service Function Forwarders* (SFFs), *Service Functions* (SFs) e *proxys* SFC. Eles se comunicam através do encapsulamento SFC. A figura 2.2 mostra esses componentes e como se relacionam.

- **Encapsulamento SFC**: O encapsulamento SFC possibilita a troca de metadados entre as funções de rede dentro do SFC. Ele também torna possível a seleção do caminho da função de rede. O encapsulamento SFC deve ser independente de transporte. Assim, qualquer protocolo de transporte pode ser utilizado para transportar os dados com o encapsulamento SFC;

- ***Service Function*** **(SF)**: Os SFs enviam e recebem dados para um ou mais SFFs. Os SFs que reconhecem SFC (SFC-*aware*) recebem esse tráfego com o encapsulamento SFC. Os SFs que não reconhecem (SFC-*unaware*), fazem uso do *proxy* SFC para participar da arquitetura. O *proxy* SFC atua como um *gateway* entre o encapsulamento SFC e a SF que é *SFC-unaware*;

Figura 2.2: Arquitetura SFC



Fonte: Adaptado de RFC 7665 (2015).

- ***Service Function Forwarders* (SFFs)**: A função do SFF é redirecionar pacotes ou *frames* da rede para um ou mais SFs que estiverem associados ao mesmo SFF usando as informações passadas pelo encapsulamento SFC. Os pacotes ou *frames* ocasionalmente voltam para o mesmo SFF, que tem a função de reencaminhá-lo para a rede. O SFF também pode encaminhar os pacotes para um classificador ou para outro SFF, caso seja necessário, segundo as informações do encapsulamento SFC;

- ***Proxy* SFC**: O *proxy* SFC possibilita que SFs que são SFC-*unaware* façam parte da arquitetura de um SFC. Ele se situa entre um ou mais SFs e o SFF ao qual estão associados. A função do *proxy* SFC é remover o encapsulamento SFC e entregar os pacotes para as SFs que são SFC-*unaware*, usando um circuito de conexão local. Ele também recebe os pacotes dos SFs, reinsere o encapsulamento SFC e devolve para o SFF dar prosseguimento ao caminho do pacote. O SFF não deve perceber quando encaminha o pacote para um *proxy*

SFC ou um SF que é SFC-*aware*;

- **Classificador**: Classificação de um dado é a imposição do encapsulamento SFC e a definição do caminho que aquele dado terá pelo SFC. Assim como o nome sugere, o classificador realiza a classificação do tráfego. A primeira classificação ocorre na entrada do tráfego no SFC. A arquitetura SFC suporta reclassificação, o que significa que um dado pode sofrer alterações no seu caminho ou metadados ou ambos durante o trajeto pelo SFC.

2.3 *NETWORK SERVICE HEADER*

O *Network Service Header* (NSH) é colocado em um pacote ou *frame* para definir o caminho que aquela função de rede terá pelo SFC. Ele também permite a inserção de metadados nos pacotes ou *frames*. Portanto, NSH é o encapsulamento SFC descrito na Arquitetura SFC definida pela IETF (2015).

O classificador é o elemento responsável por inserir o NSH, enquanto que o último SFF é responsável por removê-lo. Como explicado anteriormente, o NSH deve ser independente de transporte.

2.3.1 *Network Service Header* (NSH)

Segundo o RFC 8300 (2018), o *Network Service Header* (NSH) é composto por um Cabeçalho Base de 4 bytes, um Cabeçalho de Caminho de Serviço de 4 bytes e Cabeçalhos de Contexto opcionais.

O Cabeçalho Base fornece informações sobre o protocolo do *payload*, ou seja, da parte principal dos dados, excluindo-se cabeçalhos e metadados. Além disso, possui informações sobre o próprio NSH, como versão, comprimento e tipo de metadados. É obrigatório em qualquer implementação de NSH e é composto por sete campos, como mostra a figura 2.3.

Figura 2.3: Cabeçalho Base

Versão	O	N	TTL	Comprimento	N	Tipo de Metadados	Próximo Protocolo

Fonte: Adaptado de RFC 8300 (2018).

- **Versão**: O campo de versão garante a compatibilidade com futuras versões do NSH. Caso um SFF receba um pacote com cabeçalho NSH e não entenda a versão, o pacote deve ser descartado pelo SFF;

- **Bit O**: O campo Bit O define se o pacote é ou não um pacote OAM (*Operations, Administration and Maintenance*). Caso seja, o bit deve ser 1, caso contrário, 0. Pacotes OAM são utilizados por funções dentro do SFC que atuam para a manutenção, gerência e operabilidade daquele SFC. Essas funções devem marcar os pacotes utilizados, definindo o valor do Bit O como 1. Esse campo não deve ser modificado durante o caminho do pacote pelo SFC;

- **TTL**: É usado para prevenção de *loop*. O valor inicial do campo deve ser configurável dentro de um SFC. Cada SFF deve decrementar em 1 o valor do campo TTL antes de encaminhar o pacote para o próximo destino. Caso o campo tenha o valor 0 após o decremento, o pacote deve ser descartado;

- **Comprimento**: Nesse campo é colocado o comprimento total do NSH, incluindo o Cabeçalho Base, o Cabeçalho de Caminho de Serviço e os Cabeçalhos de Contexto. O valor do campo multiplicado por 4 bytes significa o comprimento total do NSH;

- **Bits Não Atribuídos**: Esse campo possui bits que não estão atribuídos a algum significado e estão livres para uso futuro. Todos os bits do campo devem ter o valor 0 e devem ser ignorados por elementos que suportam NSH. Na figura 2.3, está sinalizado como "N";

- **Tipo de Metadados**: Indica o formato dos metadados do pacote com NSH. Define como será o cabeçalho NSH além do Cabeçalho Base e do Cabeçalho de Caminho de Serviço, que são obrigatórios. Pode conter os valores 0x0, 0x1, 0x2 e 0xF (representados em hexadecimal). O pacote com valor 0x0 no campo deve ser descartado, pois é um valor reservado. O pacote com valor 0x1 indica que o formato do cabeçalho é um Cabeçalho de Contexto de Comprimento Fixo. O valor 0x2 no campo não define nenhum cabeçalho adicional por obrigação, mas pode conter vários Cabeçalhos de Contexto de Comprimento Variável. E, por fim, o valor 0xF é reservado para testes, logo todos os pacotes que não estiverem configurados para fazerem parte de testes ou experimentos e tiverem o valor 0xF no campo de tipo de metadados devem ser descartados;

- **Próximo Protocolo**: Indica o tipo de protocolo do dado encapsulado. Esse campo possui

8 bits e alguns dos valores e protocolos suportados são: IPv4 com o valor 0x1, IPv6 com valor 0x2, *Ethernet* com valor 0x3, NSH com valor 0x4, MPLS com valor 0x5 e os valores 0xFE e 0xFF são reservados para experimentos. Pacotes com o valor do campo Próximo Protocolo não suportado devem ser descartados.

O Cabeçalho de Caminho de Serviço proporciona a identificação dos caminhos e a posição do dado dentro de um caminho de serviço. Ele possui dois campos: o Identificador de Caminho de Serviço ou *Service Path Identifier* (SPI) e o Índice de Serviço ou *Service Index* (SI).

O *Service Path Identifier* (SPI) possui 24 bits e é usado para identificar um caminho de função de rede. Os nós participantes devem usar o SPI ao definir o caminho da função de rede.

Já o *Service Index* (SI) possui 8 bits e fornece a localização da função de rede dentro do percurso pelo SFC. O valor do SI deve ser decrementado em 1 pela função de rede após executar seu serviço ou pelo *proxy* SFC, caso a função de rede seja SFC-*unaware*. O classificador inicial deve definir o valor do SI para 255 por padrão, mas esse valor inicial pode ser alterado levando em conta o comprimento do caminho da função de rede pelo SFC. Também é o classificador inicial quem envia o pacote para o primeiro SFF indicado no caminho

O SI é usado juntamente com o *Service Path Identifier* para definir o caminho da função de rede e definir o próximo SFF ou função de rede que o pacote passará. O SI também pode ajudar a identificar problemas no caminho de uma função de rede, pois ele, assim como o TTL, é útil na prevenção de *loop*, através do decremento realizado por cada função de rede em que o pacote passa. Quando um SFF que não é o SFF final segundo o caminho definido pelos SI e SPI recebe um pacote NSH com valor do campo SI igual a 0, esse SFF deve descartar o pacote.

Os Cabeçalhos de Contexto são utilizados para transportar metadados e podem possuir comprimento fixo ou variável. O padrão definido pela IETF propõe que sejam usados um Cabeçalho de Contexto de comprimento fixo ou zero ou mais Cabeçalhos de Contexto de comprimento variável por pacote NSH.

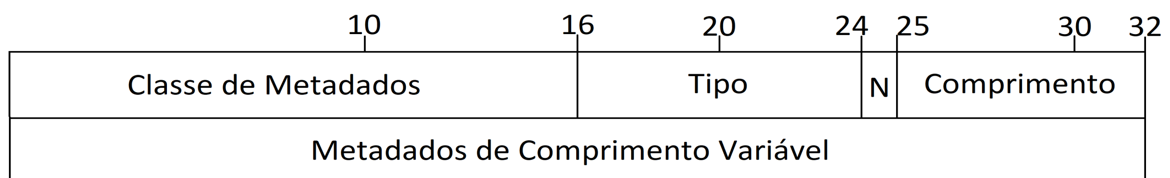
O Cabeçalho de Contexto de comprimento fixo deve estar situado obrigatoriamente após o Cabeçalho de Caminho de Serviço e possui sempre 16 bytes. A estrutura desse cabeçalho de contexto varia de acordo com a finalidade da função de rede em que o pacote NSH está transitando, podendo ser completamente diferentes de aplicação para aplicação. O valor do Cabeçalho de Contexto de Comprimento Fixo deve ser 0 quando o pacote não carrega metadados.

As funções de rede ou *proxys* SFC que não reconhecem o significado ou o formato do

Cabeçalho de Contexto de Comprimento Fixo de um determinado pacote NSH devem descartá-lo, impossibilitando, assim, que ignorem metadados que não podem processar.

Poderão existir zero ou mais Cabeçalhos de Contexto de comprimento variável em um pacote NSH quando o campo Tipo de Metadados do Cabeçalho Base tiver o valor 2. Quem define o comprimento do conjunto de Cabeçalhos de Contexto de Comprimento Variável é o campo Comprimento, também do Cabeçalho Base. Se o campo Comprimento possuir o valor 2, significa que não existem cabeçalhos adicionais, se possuir o valor 4, significa que existem 8 bytes de Cabeçalhos de Contexto de Comprimento Variável, e assim por diante. O comprimento dos Cabeçalhos de Contexto de Comprimento Variável deve ser múltiplo de 4 bytes, e, quando não é múltiplo, é feito um preenchimento. A estrutura do Cabeçalho de Contexto de Comprimento Variável é mostrada na figura 2.4.

Figura 2.4: Cabeçalho de Contexto de Comprimento Variável



Fonte: Adaptado de RFC 8300 (2018).

- **Classe de Metadados**: Define o escopo do campo Tipo para fornecer um *namespace* hierárquico. Possui 16 bits de comprimento e 3 valores atribuídos até o momento: 0x0000 que é a classe de metadados base definida pela IETF; 0xFFF6 até 0xFFFE estão reservados para experimentos e 0xFFFF foi definido como uma valor reservado e não deve ser utilizado;

- **Tipo**: Informa o tipo dos metadados do pacote. A definição dos tipos fica a cargo do proprietário da classe de metadados;

- **Bit Não Atribuído**: Esse bit ainda não está atribuído a algum significado e está disponível para uso futuro. Esse bit deve ser ignorado pelas implementações que suportam NSH. Na figura 2.4 está indicado como "N";

- **Comprimento**: Indica o comprimento dos metadados de comprimento variável. Caso o comprimento dos metadados não seja múltiplo de 4 bytes, o remetente deve preencher

com bits até o tamanho dos metadados ficar múltiplo de 4 bytes. Esses bits de preenchimento devem ser ignorados pelo receptor do pacote. Um valor 0 no campo Comprimento indica um cabeçalho de contexto sem o campo de metadados de comprimento variável.

2.4 TRABALHOS RELACIONADOS

Não existe uma única solução possível para a implementação de SFCs, cada solução tem suas características, proporcionando vantagens e desvantagens. Nos últimos anos, diversos trabalhos tentaram implementar SFCs adotando a visão da IETF e diversos propuseram novas soluções em relação à arquitetura de encadeamento, encapsulamento e implementação das funções de rede.

No trabalho realizado por Sanz et al. (2018), os autores desenvolvem funções virtuais de segurança de rede e avaliam seus desempenhos. Neste artigo, é feito o encadeamento de funções de rede conforme o padrão proposto pela IETF (2015) e também é utilizado o protocolo de cabeçalho de serviço de rede (NSH) na plataforma aberta OPNFV. A plataforma OPNFV é compatível com a arquitetura NFV do ETSI e toda função virtual implementada através dela é NSH-*aware*. No trabalho, todo o processamento NSH, que é realizado pelas VNFs, se resume a desencapsular, encapsular e decrescer o valor do campo *Service Index* em 1. A aplicação utilizada para o encapsulamento e desencapsulamento do NSH foi uma ferramenta escrita em *Python*, a *vxlan_tool*. Essa ferramenta opera de forma sequencial em apenas um núcleo de processamento, mas foi modificada pelos autores para operar paralelamente em vários núcleos. Segundo os autores, a ferramenta *vxlan_tool* é o principal fator limitante da plataforma OPNFV, mas essa limitação deve ser eliminada com a implementação da ferramenta em *kernel*, em versões futuras.

Segundo Kulkarni et al. (2017), o cabeçalho NSH pode ser modificado para alcançar uma melhor escalabilidade e eficiência no encadeamento de funções. No artigo, é proposto que o *Service Function Path* (SFP), que é definido nos campos SPI e SI do Cabeçalho de Caminho de Serviço, tenha uma nova semântica. A ideia é representar o tipo da função de rede ao invés da instância da função de rede e permitir que a rede escolha dinamicamente a melhor instância com base no tipo de função de rede e nas informações de dados de contexto. Dessa forma, caso ocorra uma mudança na topologia, não será necessário recriar todos os SPIs e notificar os SFFs em relação à mudança, para que atualizem os seus SPIs. O ponto negativo da proposta é a perda da visibilidade do caminho de ponta a ponta, pois a lista de funções de rede para

um SPI não pode ser definida estaticamente, já que um mesmo SPI pode resultar em diferentes caminhos. Como trabalhos futuros, os autores planejam criar um protótipo que implemente a solução proposta.

3 SOLUÇÃO PROPOSTA

Como foi apresentado anteriormente, não foram encontradas implementações disponíveis de Processadores NSH que executem todas as tarefas exigidas pelo padrão NSH (2018). Com o objetivo de solucionar este problema, foi desenvolvido um Processador NSH para Serviços Encadeados de Rede. Neste capítulo, serão apresentados os requisitos do Processador NSH, sua arquitetura interna e modelo de dados, e por fim, detalhes de funcionamento e implementação do mesmo.

3.1 REQUISITOS

É importante destacar que o Processador NSH pode ser utilizado tanto com funções de rede SFC-*unaware* como também com funções SFC-*aware*. O Processador NSH apresenta diferentes requisitos para estes dois tipos de funções, sendo muito mais exigido quando processa pacotes para uma função SFC-*unaware*.

A única tarefa executada tanto para funções de rede SFC-*aware* como para funções SFC-*unaware* é a tarefa de descartar pacotes. O descarte deve ocorrer quando a função de rede não conhecer o formato ou a semântica do Cabeçalho de Contexto do NSH, ou seja, metadados que não podem ser processados pela função de rede não devem ser ignorados. Ainda, devem ser descartados pacotes cujo Bit O estiver marcado como 1 e a função de rede não suportar procedimentos OAM, isto é, procedimentos de gerência e manutenção. O restante dos requisitos do Processador NSH somente são necessários quando são processados pacotes para funções SFC-*unaware*.

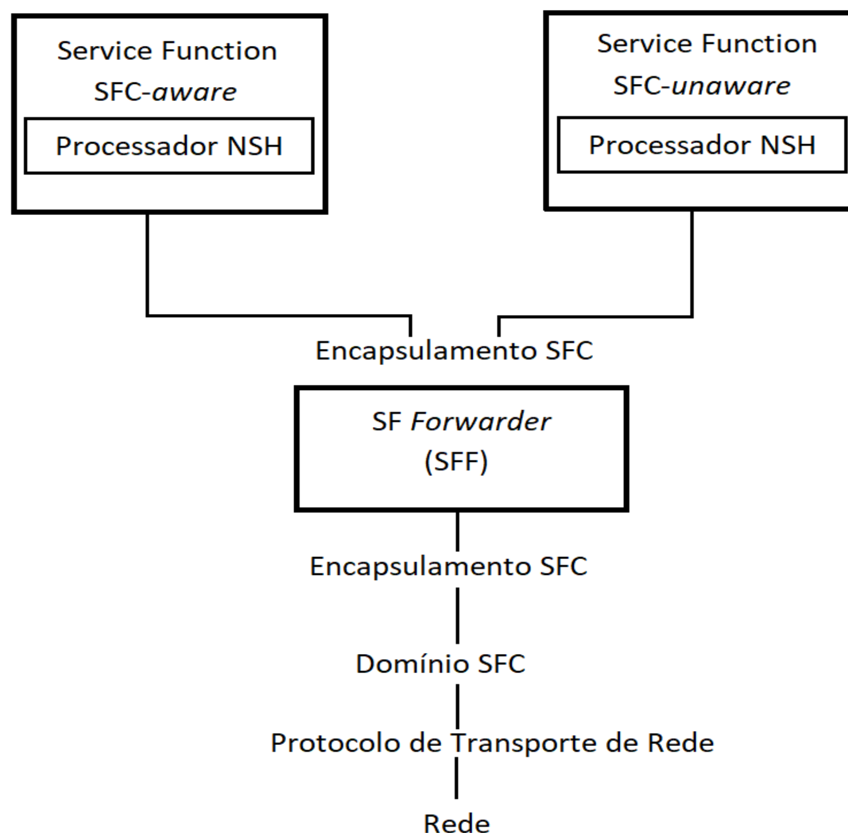
Uma das atribuições mais básicas do Processador NSH é desencapsular o cabeçalho NSH do quadro ou pacote durante o seu caminho pela SFC. Isto permite que a função de rede possa focar apenas nos dados úteis do pacote e, assim, o Processador NSH facilita a implementação de novas funções de rede no encadeamento, já que permite que funções de rede mais genéricas façam parte da arquitetura. Além disso, sempre após uma função de rede executar seu serviço, o pacote retorna para o Processador NSH, que reencapsula o pacote, decrementa o valor do *Service Index* em 1 e retorna o pacote para o SF *Forwarder* que encaminhou o pacote para a função.

Embora o uso do Processador NSH assemelhe-se ao do *proxy* SFC, ele possui o diferen-

cial de permitir que funções de rede SFC-*unaware* também façam uso dos metadados inseridos nos pacotes da arquitetura, permitindo a leitura e gravação de metadados. Esta função é importante pois permite a aplicação de políticas de serviço e fornece contexto de rede com uma maior precisão. Por exemplo, um função de rede pode mudar suas decisões locais com base nos metadados recebidos ou, ainda, forçar, após a inserção de metadados no pacote, que esse pacote seja reclassificado e tenha seu caminho pela SFC modificado. Em SFCs que não possuem o compartilhamento de metadados pelas funções de rede, as classificações de pacotes e decisões de caminho ocorrem apenas na entrada e na saída do pacote pelo encadeamento, o que faz com que políticas de serviço sejam aplicadas com maior granularidade.

3.2 ARQUITETURA

Figura 3.1: Arquitetura SFC com Processador NSH



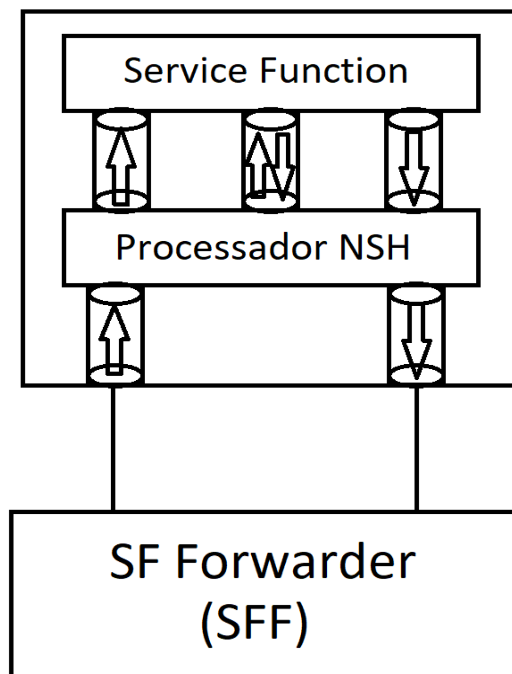
Fonte: acervo pessoal.

Tendo como base a arquitetura SFC proposta pela IETF (2015), o Processador NSH

situa-se dentro de uma *Service Function* (SF). O Processador NSH executa todas as tarefas de uma SF referentes a NSH, portanto, o uso do *proxy* SFC não é necessário para aquelas SFs que forem SFC-*unaware*. A figura 3.1 demonstra a arquitetura do SFC com o Processador NSH já inserido.

O Processador NSH trata o pacote assim que ele chega à máquina, após o tratamento, o pacote é enviado para a SF, que comunica-se, quando julga necessário, com o Processador NSH para a inserção ou leitura de metadados. Em seguida, o pacote retorna para o Processador NSH, que faz as ações necessárias e encaminha o pacote para o SF *Forwarder* para que ele tenha prosseguimento no seu caminho pelo SFC. É importante, então, que existam cinco interfaces para a comunicação do Processador NSH com a SF e com o SF *Forwarder*: uma interface de entrada, três interfaces entre o Processador NSH e a SF (uma para a entrada na SF, uma para a saída e uma para a troca de metadados), e uma interface de saída. A figura 3.2 mostra esta estrutura e o fluxo de dados com mais detalhes.

Figura 3.2: Interfaces do Processador NSH



Fonte: acervo pessoal.

Para funções de rede SFC-*unaware*, o Processador NSH precisa desencapsular o cabeçalho NSH antes de enviar o pacote para a SF. O NSH é guardado e reinserido no mesmo pacote após o serviço ser realizado pela função. Caso a SF não implemente modelo FIFO (*First*

In, First Out), existe a possibilidade de que o Processador NSH não reinsira o NSH no pacote que foi desencapsulado, ou que insira o NSH de um outro pacote erroneamente. Para que haja garantia de que os pacotes não perderão o NSH ou terão o seu NSH trocado, o Processador NSH exige que as SFs implementem o modelo FIFO. A implementação deste modelo pelas SFs também garante que as funções possam ler e gravar metadados nos pacotes corretos.

A tarefa de ler e gravar metadados é feita, por padrão, seguindo um modelo *pull*, ou seja, a função lê e grava metadados, que estarão no Cabeçalho de Contexto do NSH, quando for necessário. Para permitir o uso do modelo *push*, foi feita uma pequena modificação no Cabeçalho Base do NSH sugerido pela IETF (2018). Foi adicionada a *flag* PSH, e, quando essa *flag* está ativa, o Processador NSH envia os metadados para a função assim que recebe o pacote. A SF toma conhecimento, então, que o próximo pacote que for recebido possui metadados que não podem ser ignorados e que devem ser considerados para o processamento do pacote. É importante salientar que embora a *flag* PSH não esteja prevista e padronizada, ela não quebra o modelo, pois utiliza de um dos bits não atribuídos do NSH, e funções de rede que não entendem seu uso podem simplesmente ignorá-la. O Cabeçalho Base com a nova *flag* PSH está detalhado na figura 3.3.

Figura 3.3: Cabeçalho Base com a *flag* PSH

Fonte: acervo pessoal.

Tal como explicado anteriormente, existe uma interface de controle utilizada exclusivamente para a troca de metadados entre a função de rede e o Processador NSH. Para a SF realizar a leitura dos metadados, ela deve enviar um pacote UDP com a porta de origem definida com o valor 8001 e a porta de destino com o valor 8002 para a interface de controle. O processador então recebe esse pacote, muda o valor da porta de origem para 8002, adiciona os metadados ao final do pacote e devolve para a interface de controle. Para a gravação de metadados, a SF envia um pacote UDP com a porta de origem com valor 8001 e porta de destino com valor 8003 e, ao final do pacote, adiciona os metadados que deseja salvar. É importante destacar que a função sobrescreve os metadados que estavam no NSH, portanto, é interessante que a função faça a leitura antes de realizar alguma escrita. Os números das portas foram definidos apenas para a

implementação do protótipo do Processador NSH, o que significa que podem ser flexibilizados em versões futuras.

Segundo o RFC 8300 (2018), o qual define o padrão NSH, as implementações NSH precisam obrigatoriamente suportar tipos de metadados de valor 1 e 2 (onde o comprimento for 2). Porém, implementações NSH devem, não obrigatoriamente, suportar tipos de metadados de valor 2 com comprimento maior que 2. Portanto, para este projeto, o Processador NSH suportará somente os tipos de metadados 1, com comprimento 6 e 2, com comprimento 2.

3.3 IMPLEMENTAÇÃO

Esta seção descreve detalhes de implementação e funcionamento do Processador NSH, comentando sobre as ferramentas utilizadas e demonstrando trechos de código. Foi utilizada a linguagem de programação *Python* na versão 2.7 juntamente com a ferramenta *Scapy* (2010) para a implementação da solução proposta.

Segundo sua documentação, a ferramenta *Scapy* é um programa em *Python* que permite ao usuário enviar, farejar, dissecar e forjar pacotes de rede. Em outras palavras, *Scapy* é um poderoso programa interativo de manipulação de pacotes. Ele é capaz de forjar ou decodificar pacotes de um grande número de protocolos, enviá-los na rede, capturá-los, corresponder solicitações e respostas e muito mais. O *Scapy* pode lidar facilmente com tarefas mais clássicas, como varredura, *tracerouting*, sondagem, testes de unidade, ataques ou descoberta de rede.

Scapy não possui suporte ao protocolo NSH, mas possibilita que novos protocolos sejam adicionados aos programas que o usam. Para que o Processador NSH tenha conhecimento e possa alterar os valores dos campos do NSH, foi criada uma classe que define o protocolo NSH. O código da classe é demonstrado no *Listing* 3.1.

Listing 3.1: Código da classe que define o protocolo NSH

```
from scapy.all import *

class NSH(Packet):
        name = "NSH"
        fields_desc = [BitField ("Version", 1, 2),
                        BitField ("O_bit", 0, 1),
                        BitField ("PSH", 0, 1),
                        BitField ("TTL", 63, 6),
                        BitField ("Length", 0x6, 6),
                        BitField ("Unassigned", 0, 4),
                        BitField ("MD_Type", 0x1, 4),
                        ByteField("Next_Protocol", 0x01),
                        ThreeBytesField("SPI", 1),
                        ByteField("SI", 255),
                        ConditionalField(BitField("Context_Header", 0, 128),
                        lambda pkt:pkt.MD_Type == 0x1 and pkt.Length==0x6)
                        ]
```

Nesta classe, é definido o nome do protocolo através da variável *name* e todos os campos e seus valores padrão, através da lista *fields_desc*. Como o Processador NSH terá suporte apenas aos tipos de metadados 1 e 2 com comprimento 2, o campo *Context_Header* nem sempre existirá.

Foram padronizados, no RFC 8300, alguns valores para o campo Próximo Protocolo do Cabeçalho Base do NSH e o valor 0x894F para o campo tipo, do protocolo *Ethernet*, que representa o protocolo NSH. Como *Scapy* não possui suporte ao NSH, além de criar a classe que define o protocolo NSH, também é necessário determinar o comportamento desejado pelo programa. Por exemplo, sempre que um pacote com o protocolo *Ethernet* possuir o campo de tipo com valor 0x894F, representado em hexadecimal, espera-se que o próximo protocolo seja NSH.

Assim que um pacote chega a uma das interfaces do Processador NSH ele é capturado e, dependendo da interface em que o pacote é recebido, ele terá diferentes tratamentos. Caso ele seja recebido na interface de entrada do Processador NSH, é testado se o pacote possui o protocolo NSH, caso não possua, ele é simplesmente encaminhado para a SF, caso possua, é necessário realizar alguns procedimentos. Primeiro, é testado se o Bit O está setado e se a função possui suporte a procedimentos OAM e logo após são checados os campos Tipo de Metadados e Comprimento. Caso o pacote passe pelos testes e não seja descartado, o Processador NSH analisa se a função que está conectada a ele é SFC-*aware* ou SFC-*unaware*. Se for SFC-*aware*, ele encaminha o pacote para a função e deixa a cargo dela o decréscimo do *Service Index* e a leitura e gravação de metadados. Caso seja SFC-*unaware*, o Processador NSH realiza o desencapsulamento do cabeçalho NSH e observa se a flag PSH está ativa. Se estiver, envia os metadados do pacote para a SF logo antes de enviar o pacote sem o NSH, se não estiver, envia somente o pacote com o NSH removido para a SF. O *Listing* 3.2 mostra o trecho de código que executa estes procedimentos.

Listing 3.2: Código da função que trata pacotes recebidos na interface de entrada do Processador NSH

```
def fromSFF(self, pkt):
    if(NSH in pkt):
        if pkt[NSH].O_bit == 1 and not self.supportOAM:
            return
        if pkt[NSH].MD_Type != 1 and pkt[NSH].MD_Type != 2 :
            return
        elif pkt[NSH].MD_Type == 1 and pkt[NSH].Length != 0x6:
            return
        elif pkt[NSH].MD_Type == 2 and pkt[NSH].Length != 0x2:
            return

        processedPkt = None
        if not self.isAware:
```

```
            counter = 0
            while True:
                layer = copy.deepcopy(pkt.getlayer(counter))
                if(layer != None):
                    if(layer.name != "NSH"):
                        if(counter == 0):
                            layer.remove_payload()
                            processedPkt = layer
                        else:
                            layer.remove_payload()
                            processedPkt = processedPkt/layer
                    else:
                        self.nshList.append(counter)
                        layer.remove_payload()
                        self.nshList.append(layer)
                else:
                    break
                counter += 1

            if pkt[NSH].PSH == 1 and pkt[NSH].MD_Type == 1:
                psh=Ether()/IP()/UDP(sport=8002, dport=8004)/Raw(load=pkt[NSH].Context_Header)
                self.nshpcontrol.send(raw(psh))
        else:
            processedPkt = pkt
            self.nshList.append(-1)
    else:
        processedPkt = pkt
        self.nshList.append(-1)
    self.nshptosf.send(raw(processedPkt))
```

Durante o processamento do pacote, uma SF SFC-*unaware* pode desejar ler os possíveis metadados inseridos no NSH que foi desencapsulado. Para isso, como explicado anteriormente, ela deve enviar um pacote UDP com a porta de origem definida com o valor 8001 e a porta de destino com o valor 8002 para a interface de controle. O Processador NSH então envia o mesmo pacote para a interface de controle com os metadados inseridos ao final e a porta de origem modificada para o valor 8002. Para a SF escrever metadados, também como explicado anteriormente, é necessário que ela envie para a interface de controle um pacote UDP com a porta de origem com valor 8001 e porta de destino com valor 8003 e, ao final do pacote, os metadados que deseja gravar. Assim que recebe um pacote com essas características, o Processador NSH sobrescreve os metadados que estavam no NSH e insere os metadados desejados pela SF. O código que executa as ações de leitura e escrita de metadados é exibido no *Listing* 3.3.

Listing 3.3: Código da função que trata pacotes recebidos na interface de controle do Processador NSH

```
def fromControl(self, pkt):
    if(len(self.nshList) > 0 and self.nshList[0] > -1) and (UDP in pkt):
        if(pkt[UDP].sport == 8001 and pkt[UDP].dport == 8002):
            pkt[UDP].sport = 8002
            self.nshpcontrol.send(raw(pkt/Raw(load=self.nshList[1].Context_Header)))

        elif(pkt[UDP].sport == 8001 and pkt[UDP].dport == 8003):
            self.nshList[1].Context_Header = pkt.getlayer(3)
```

O Processador NSH também pode receber pacotes da interface de saída da SF. Quando

isto acontecer, ele analisa a lista com os NSHs que foram desencapsulados para julgar se o pacote recebido possuía ou não um cabeçalho NSH. Caso não possuísse NSH ou a SF que enviou é SFC-*aware*, o pacote é encaminhado diretamente para a interface de saída. Caso possuísse um cabeçalho NSH e a SF que enviou o pacote é SFC-*unaware*, o Processador NSH reinsere o NSH no pacote, diminui o valor do *Service Index* em 1 e envia o pacote pela interface de saída. O *Listing* 3.4 demonstra o código executado quando o Processador NSH recebe um pacote da SF.

Listing 3.4: Código da função que trata pacotes recebidos pela interface de saída da SF

```
def fromSF(self, pkt):
    processedPkt = None
    if(len(self.nshList) > 0 and self.nshList[0] > -1):
        counter = 0
        while True:
            layer = copy.deepcopy(pkt.getlayer(counter))

            if(layer != None or counter == self.nshList[0]):
                if(counter == self.nshList[0]):
                    if(counter == 0):
                        processedPkt = self.nshList[1]
                    else:
                        processedPkt = processedPkt/self.nshList[1]
                    if(layer != None):
                        layer.remove_payload()
                        processedPkt = processedPkt/layer

                else:
                    layer.remove_payload()
                    if(counter == 0):
                        processedPkt = layer
                    else:
                        processedPkt = processedPkt/layer

            else:
                break
            counter += 1
        self.nshList.pop(0)
        self.nshList.pop(0)
    else:
        if(len(self.nshList) > 0):
            self.nshList.pop(0)
        processedPkt = pkt

    processedPkt = raw(processedPkt)
    processedPkt = Ether(processedPkt)
    if(NSH in processedPkt) and not self.isAware:
                if(processedPkt[NSH].SI > 0):
            processedPkt[NSH].SI -= 1

    self.nshpout.send(raw(processedPkt))
```

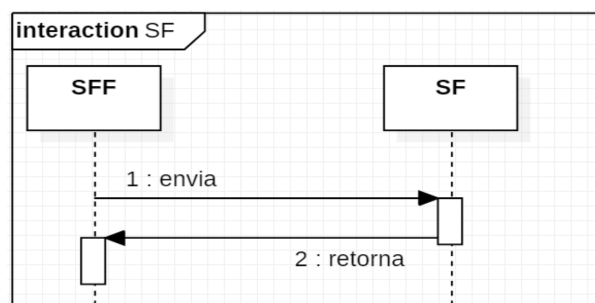
4 AVALIAÇÃO DE DESEMPENHO

A validação da solução proposta por este trabalho é constituída pela avaliação da mesma através das métricas de vazão e atraso. Assim, este capítulo tem como objetivo apresentar e discutir os resultados obtidos na avaliação. Na primeira seção será apresentado o ambiente de teste e metodologia, comentando sobre as características da máquina usada, função de rede utilizada, métricas, etc. Logo após, na segunda seção, serão exibidos os resultados obtidos e, por fim, na terceira seção deste capítulo, serão discutidos os resultados.

4.1 AMBIENTE DE TESTE E METODOLOGIA

O cenário utilizado para a execução da avaliação foi composto por uma máquina de sistema operacional Debian, processador Intel Core i7 de frequência 3.4 GHz e 8 núcleos, e 7.7 GB de memória RAM. A função de rede executada para a avaliação apenas encaminha pacotes entre duas interfaces de rede, sem realizar nenhum processamento mais complexo. Foi escolhida tal função para que haja o menor atraso possível pela função de rede e o melhor desempenho teórico possa ser observado. Esta função foi desenvolvida em *Python* e fez uso da ferramenta *Scapy* assim como o Processador NSH.

Figura 4.1: Diagrama de sequência: somente a SF

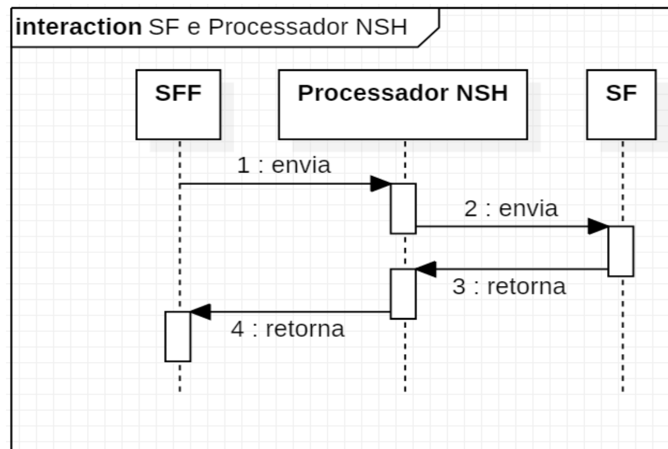


Fonte: acervo pessoal.

As métricas usadas para a avaliação foram a vazão e atraso do Processador NSH. Para a vazão, foram considerados três cenários: um com somente a função de rede; um com o Processador NSH e a função de rede sem a requisição de metadados; e o último com o Processador NSH e a função de rede fazendo leitura de metadados. As figuras 4.1, 4.2 e 4.3 apresentam os diagramas de sequência que exemplificam o processo de teste nos três cenários descritos. Além

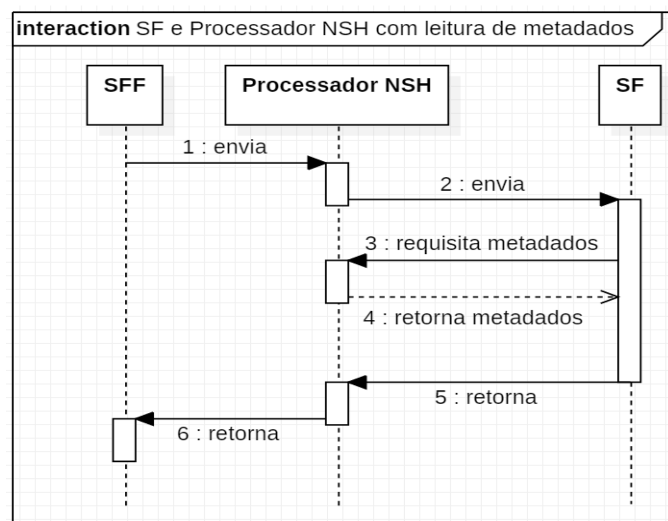
disso, para o teste de vazão, o tamanho dos pacotes variam de 64 bytes até 1500 bytes e não terão metadados, ou seja, o cenário que faz a leitura de metadados não exigirá um processamento complexo. Ao final de cada bateria de testes foi calculada a média da vazão dos pacotes e um intervalo de confiança de 95% foi encontrado.

Figura 4.2: Diagrama de sequência: SF e Processador NSH



Fonte: acervo pessoal.

Figura 4.3: Diagrama de sequência: SF e Processador NSH com leitura de metadados



Fonte: acervo pessoal.

Para o atraso, serão quatro cenários: somente a função de rede; função de rede e Processador NSH com pacotes sem NSH; função de rede SFC-*unaware* e Processador NSH com pacotes com NSH; e função de rede SFC-*unaware* requisitando a leitura de metadados e Pro-

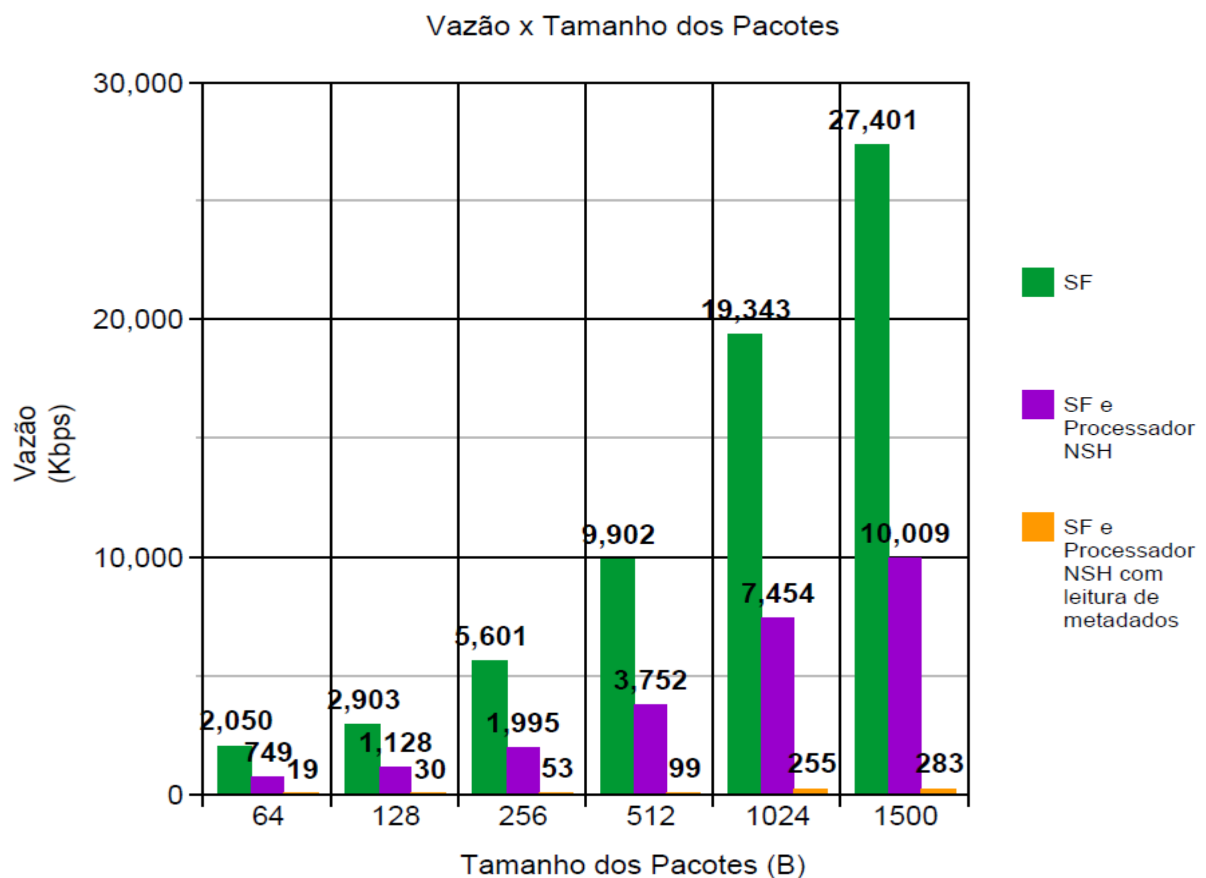
cessador NSH com pacotes com NSH. Assim, serão destacados o tempo de encapsulamento, desencapsulamento e o tempo entre a requisição da leitura de metadados e recebimento da resposta.

Como ferramentas para a avalição, foram utilizados o *Iperf* (2003) e *Wireshark* (2014). O *Iperf* foi utilizado para saturar os enlaces com pacotes UDP nos testes de vazão e a ferramenta *Wireshark* foi usada para o cálculo dos atrasos e vazão.

4.2 RESULTADOS OBTIDOS

Esta seção tem como objetivo apresentar e discutir brevemente os resultados obtidos com os experimentos realizados.

Figura 4.4: Avaliação da vazão dos cenários



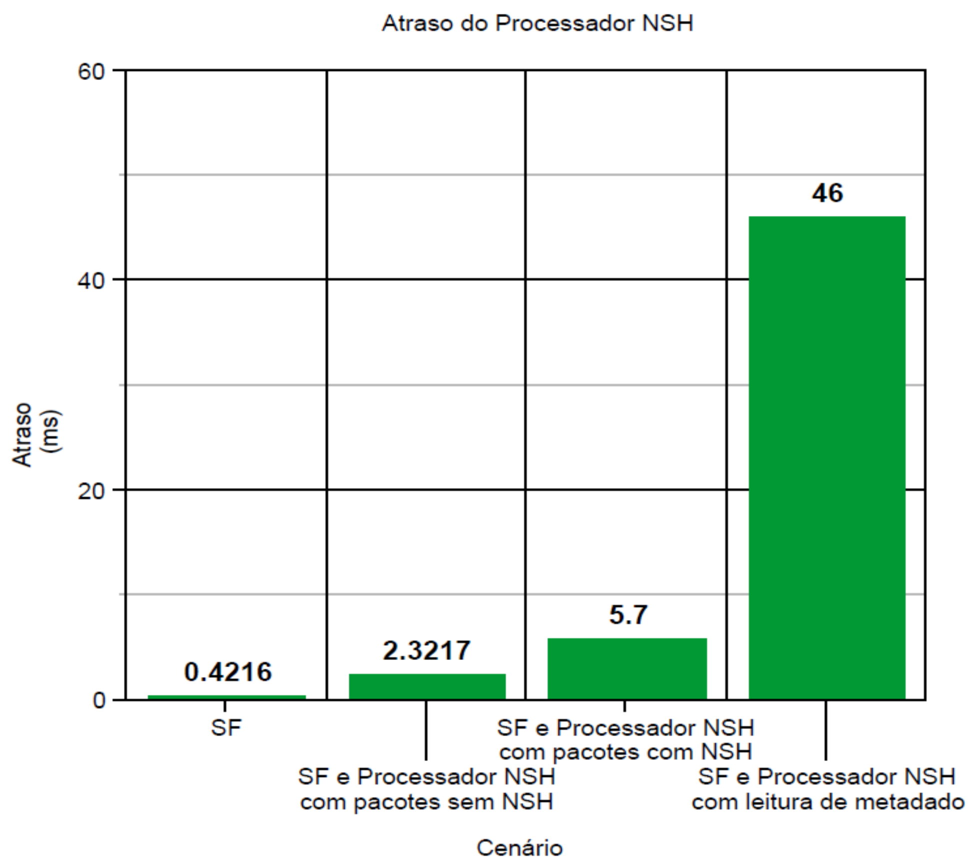
Fonte: acervo pessoal.

A figura 4.4 demonstra o gráfico de vazão dos cenários testados. Nota-se a queda brusca de vazão do cenário em que é feita a leitura de metadados em relação aos outros. O processo

que gera maior perda de vazão é a leitura de metadados e a espera pela resposta que a SF executa nesse cenário, o que é apresentado como as mensagens 3 e 4, da figura 4.3. Como os intervalos de confiança não se intercalaram, eles não foram inseridos no gráfico para uma menor poluição visual.

A Figura 4.5 apresenta o atraso das plataformas testadas. Assim como acontece com a vazão, a maior perda de desempenho é ocasionada pela comunicação entre o Processador NSH e a SF, principalmente quando a SF espera pelo recebimento de metadados.

Figura 4.5: Avaliação do atraso dos cenários



Fonte: acervo pessoal.

4.3 DISCUSSÃO

Olhando para os gráficos, percebe-se que a vazão do Processador NSH foi baixa, porém, a função de rede também teve um desempenho ruim. Um dos maiores fatores limitantes é a ferramenta *Scapy*, que foi usada para o recebimento e alteração de pacotes. *Scapy* é uma ferramenta poderosa e simples de utilizar, mas deixa a desejar no quesito desempenho. Como

exemplo, percebe-se a queda brusca no desempenho quando a função de rede espera por uma resposta do Processador NSH. Isso acontece porque toda vez que a função de rede requisita metadados, ela cria um novo *socket* com a função *sniff* do *Scapy* ao invés de criar um único *socket* e reutilizá-lo. Ainda, durante a implementação do Processador NSH, o envio de pacotes também estava sendo feito com *Scapy*, assim como o recebimento. A simples alteração do código de envio para o uso de *sockets* gerou uma vazão várias vezes maior.

Em relação ao atraso, é notável o atraso proveniente do encapsulamento e desencapsulamento do cabeçalho NSH. Esta queda no desempenho em relação a pacotes sem NSH é inevitável, já que adiciona-se uma nova obrigação. Como não foram encontradas implementações de *proxy* SFC disponíveis, não foi possível realizar a comparação do Processador NSH com um *proxy* SFC, entretanto, espera-se uma diminuição do atraso em relação aos *proxys* SFC que situam-se em máquinas diferentes das funções de rede para as quais eles realizam suporte, visto que o Processador NSH situa-se sempre na mesma máquina da função de rede. Em relação aos *proxys* SFC que situam-se na mesma máquina da função de rede, espera-se que o desempenho do Processador NSH seja muito similar quando executadas as mesmas operações.

Embora a adição do Processador NSH na arquitetura resulte em um pior desempenho no atraso e vazão, ele provê suporte para funções genéricas que desejam fazer parte de uma Arquitetura SFC, o que faz o seu uso muito importante para SFCs que desejam adicionar *Service Functions* SFC-*unaware*. O Processador NSH pode realizar o processamento para funções de rede implementadas em diferentes linguagens sem qualquer alteração nas funções, o que faz com que ele seja adequado para cenários onde existem infraestruturas de diferentes fornecedores e os requisitos com relação a atraso e vazão não sejam tão restritos.

5 CONCLUSÃO

A área de Serviços Encadeados de Rede tem atraído um grande interesse de pesquisadores do mundo inteiro. Por ainda se encontrar em um estágio inicial, alguns conceitos e métodos não estão definidos claramente. O processamento do cabeçalho NSH é um destes, já que embora existam modelos e especificações apresentando uma visão de alto nível de como este deve ser implementado, ainda não existe uma implementação capaz de atender efetivamente os requisitos especificados.

Partindo desta ideia, foi realizado uma revisão bibliográfica sobre a arquitetura e a especificação de Serviços Encadeados de Rede como um todo, de forma a identificar os requisitos, e quais as deficiências existentes nas implementações atuais. Também foi estudado o conceito de NSH, possibilitando a identificação das ferramentas necessárias para uma implementação que atendesse a todos os requisitos previamente padronizados. Assim, o próximo passo foi analisar as ferramentas disponíveis e selecionar a mais adequada para a criação do Processador NSH. Com o instrumental definido, a implementação pôde ser iniciada. Por fim, foram realizados testes em cenários que assemelham-se aos reais para que o processador pudesse ser validado e seu desempenho avaliado. Nesta etapa, após avaliações iniciais não satisfatórias, foram efetuadas modificações na implementação, que geraram uma significativa melhora de desempenho. Como apresentado no capítulo anterior, verificou-se que embora o desempenho tenha sido pouco satisfatório, o Processador NSH possui capacidade de executar em diferentes infraestruturas, além de funcionalidades ainda não encontradas em outras implementações que envolvem NSH, o que torna seu uso pertinente para arquiteturas em que a vazão e atraso não são os requisitos mais importantes.

Com relação aos objetivos do trabalho, é possível dizer que estes foram atingidos satisfatoriamente, por apresentar um protótipo funcional de um Processador NSH que atende de forma eficaz uma grande quantidade de requisitos, além da pesquisa sobre NSH e implementações disponíveis atualmente.

Para trabalhos futuros, é importante que o processador suporte cabeçalhos NSH com o campo Tipo de Metadados de valor 2 e comprimento maior que 2. Também sugere-se a substituição da ferramenta *Scapy* para o recebimento de pacotes pelo uso de *sockets* ou outra ferramenta com melhor desempenho, o que deve gerar melhoras significativas na vazão e atraso. Além disso, é interessante a adição de um *buffer* no processador para guardar os pacotes e, as-

sim, fazer com que a função de rede processe um pacote por vez, o que garante que a requisição de metadados pela função retorne os metadados do pacote correto. Por fim, é sugerido que sejam utilizadas *Threads* para as tarefas de encapsulamento, desencapsulamento e leitura e gravação de metadados, dado que deve causar melhoras consideráveis no atraso gerado por estas tarefas.

REFERÊNCIAS

BIONDI, P. **the Scapy community**: scapy documentation, release 2.1. 1. 2010.

ETSI, N. GS NFV 002-V1. 1.1-Network Function Virtualisation (NFV)-Architectural Framework. **publishing October**, [S.l.], 2013.

GATES, M. et al. Iperf user docs. **Technical Documentation**, [S.l.], 2003.

HALPERN, J. M.; PIGNATARO, C. **Service Function Chaining (SFC) Architecture**. [S.l.]: RFC Editor, 2015. n.7665. (Request for Comments).

KULKARNI, S. et al. Neo-NSH: towards scalable and efficient dynamic service function chaining of elastic network functions. In: INNOVATIONS IN CLOUDS, INTERNET AND NETWORKS (ICIN), 2017 20TH CONFERENCE ON. **Anais...** [S.l.: s.n.], 2017. p.308–312.

LAMPING, U.; SHARPE, R.; WARNICKE, E. **Wireshark User's Guide for Wireshark 2.1**. 2014.

QUINN, P.; ELZUR, U.; PIGNATARO, C. **Network Service Header (NSH)**. [S.l.]: RFC Editor, 2018. n.8300. (Request for Comments).

SANZ, I. J.; MATTOS, D. M. F.; DUARTE, O. SFCPerf: an automatic performance evaluation framework for service function chaining. In: IEEE/IFIP NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM–NOMS. **Anais...** [S.l.: s.n.], 2018.