Serviços Virtualizados de Rede Confiáveis: Uma Arquitetura para SFCs Tolerantes a Falhas e Intrusão

Giovanni Venâncio[1], Vinicius Fulber-Garcia[1], Eduardo A. P. Alchieri[2], Elias P. Duarte Jr.[1]

[1]Universidade Federal do Paraná (UFPR) - Curitiba - Paraná - Brasil

[2]Universidade de Brasília (UnB) - Brasília - Distrito Federal - Brasil

gvsouza@inf.ufpr.br, vinicius@inf.ufpr.br, alchieri@unb.br, elias@inf.ufpr.br

Resumo. *Serviços virtualizados de rede podem ser construídos através da combinação de múltiplas VNFs (Virtualized Network Functions) conectadas em uma ordem predefinida, denominada de SFC (Service Function Chain). A IETF define uma arquitetura padronizada para SFCs, baseada em elementos de classificação e de encaminhamento. Considerando que diversos serviços de rede implementam funcionalidades críticas para o funcionamento correto da rede, falhas em qualquer componente da SFC podem comprometer toda a infraestrutura, levando a prejuízos monetários ou até mesmo ao uso não autorizado do sistema. Neste contexto, este trabalho propõe a FIT-SFC (Fault- & Intrusion Tolerant SFC): uma arquitetura para suportar serviços virtuais seguros e altamente disponíveis. Enquanto grande parte dos trabalhos anteriores consideram somente falhas por parada, a FIT-SFC utiliza estratégias de replicação para tolerar falhas bizantinas de qualquer componente da arquitetura SFC, sendo ainda totalmente compatível com a arquitetura de referência da IETF. Um protótipo da arquitetura foi implementado como prova de conceito e resultados experimentais avaliam os custos para tolerar as falhas.*

1. Introdução

O paradigma de Virtualização de Funções de Rede (*Network Function Virtualization* - NFV) vem transformando a forma como as funções de rede são projetadas e gerenciadas [Mijumbi et al. 2015, Chiosi et al. 2012]. O paradigma NFV define uma alternativa concreta para a substituição de funções de rede tradicionalmente implementadas em hardware (*i.e., middleboxes*) por instâncias virtuais que executam em hardware de prateleira [Mijumbi et al. 2015]. Assim, funções de rede passam a ser implementadas diretamente em software, executadas através de diversas tecnologias de virtualização, como máquinas virtuais ou *containers*. Dada a sua natureza virtualizada, a tecnologia NFV representa um impacto significativo na forma como a rede é gerenciada, uma vez que simplifica e flexibiliza a implementação, operação e disponibilização de funções de rede [Han et al. 2015].

No paradigma NFV, uma instância virtual que executa uma função de rede passa a ser chamada de Função Virtualizada de Rede (*Virtualized Network Function* - VNF). Por outro lado, enquanto uma VNF executa uma funcionalidade específica, um serviço virtual completo de rede pode ser obtido através da composição encadeada de múltiplas VNFs, chamado de *Service Function Chain* (SFC) [Quinn et al. 2015]. Em uma SFC, VNFs são conectadas em uma ordem pré-definida através do qual o tráfego é encaminhado. A IETF (*Internet Engineering Task Force*) propôs uma arquitetura de referência para SFCs, permitindo que os serviços virtualizados sejam executados no núcleo da rede, considerando

inclusive diferentes tipos de topologias. A arquitetura SFC proposta pela IETF consiste principalmente de elementos de classificação e encaminhamento, permitindo a criação e execução de um serviço virtual totalmente funcional.

Ainda que a tecnologia NFV ofereça diversas vantagens sobre as alternativas em hardware, é inquestionável que serviços de rede baseados em virtualização são mais suscetíveis a falhas [Sharma et al. 2020, Cotroneo et al. 2019, Han et al. 2017]. Em particular, a indisponibilidade de serviços virtualizados que hospedam plataformas de comércio online, transições bancárias, *streaming* de vídeo e armazenamento online podem gerar prejuízos monetários significativos [Gunawi et al. 2016, Gill et al. 2011]. Ainda, a falha em serviços de segurança podem levar ao uso não autorizado do sistema [Pattaranantakul et al. 2018], afetando não somente a infraestrutura subjacente (rede NFV), como também os próprios serviços dos usuários. Portanto, é essencial garantir a alta disponibilidade de serviços de rede baseados em NFV, permitindo assim a sua adoção para usos comerciais em larga escala.

Apesar de serviços virtualizados de rede serem amplamente explorados na literatura, diversos aspectos relacionados a tolerância a falhas ainda são marginalmente abordados. Dentre as estratégias para a construção de serviços virtuais tolerantes a falhas, é possível citar aquelas que se dedicam a identificar, tolerar e, eventualmente, recuperar falhas por parada em instâncias de VNF [Wang et al. 2021, Ghaznavi et al. 2020, Khalid and Akella 2019, Kong et al. 2017]. No entanto, tais soluções não consideram falhas nos componentes da própria arquitetura SFC, o que pode comprometer o funcionamento de todo o serviço de rede. Por exemplo, a falha no componente de encaminhamento impede que um fluxo de pacotes seja encaminhado entre as VNFs da SFC.

Neste contexto, este trabalho propõe a FIT-SFC (*Fault- & Intrusion Tolerant* SFC): uma arquitetura tolerante a falhas para suportar serviços virtuais seguros e altamente disponíveis. A FIT-SFC utiliza estratégias de replicação para tolerar tanto falhas por parada quanto falhas por omissão ou intrusão (*i.e.,* falhas bizantinas). A arquitetura FIT-SFC foi projetada de maneira totalmente compatível com a arquitetura de referência proposta pela IETF, possibilitando a interoperabilidade da plataforma com diversos tipos de serviços virtualizados. A arquitetura proposta tolera falhas tanto em fluxos *stateless* quanto em fluxos *stateful*. Por sua vez, há dois tipos de consistência para fluxos *stateful*. No primeiro tipo, a FIT-SFC garante a consistência de mensagens de cada fluxo que processa. No segundo tipo, as múltiplas réplicas devem se manter consistentes tendo em vista todas as mensagens de múltiplos fluxos. Neste último caso é necessário utilizar o consenso bizantino para garantir a consistência.

Como prova de conceito, um protótipo da arquitetura FIT-SFC foi implementado, possibilitando a avaliação e a verificação do seu funcionamento. Para isso, três serviços virtualizados com diferentes características também foram implementados e executados sobre a FIT-SFC: (*i*) um serviço de DNS (*Domain Name System*); (*ii*) um serviço de DNS autenticado e; (*iii*) um serviço de balanceamento de carga (*Load Balancer* - LB). Resultados experimentais avaliam o impacto na latência sobre cada um destes serviços, demonstrando o custo necessário para tolerar falhas.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 faz uma breve descrição do paradigma NFV e da arquitetura SFC proposta pela IETF, além de des-

crever os principais trabalhos relacionados. A Seção 3 descreve em detalhes a arquitetura FIT-SFC, apresentando as estratégias de replicação adotadas para tolerar falhas bizantinas em qualquer componente da arquitetura SFC. A Seção 4 apresenta a implementação do protótipo e discute os resultados experimentais. Por fim, a Seção 5 conclui o trabalho.

2. Fundamentação e Trabalhos Relacionados

Esta seção apresenta os principais conceitos do paradigma NFV e detalha a arquitetura SFC definida pela IETF. Além disso, os trabalhos relacionados a tolerância a falhas de SFCs mais relevantes são discutidos.

2.1. Serviços Virtualizados e a Arquitetura SFC IETF

Enquanto VNFs executam funções específicas de rede, elas podem ser combinadas para criar serviços completos de rede, denominadas de SFCs [Halpern and Pignataro 2015]. Uma SFC consiste de uma composição de múltiplas VNFs, encadeadas em uma ordem específica através da qual o tráfego é encaminhado [Fulber-Garcia et al. 2020, Huff et al. 2018]. Com o objetivo de flexibilizar a criação e gerenciamento de serviços, além de facilitar a configuração e criação de serviços em diferentes topologias, a IETF propôs uma arquitetura padrão para SFCs. A arquitetura, ilustrada na Figura 1, consiste de elementos de classificação e de encaminhamento, descritos a seguir.

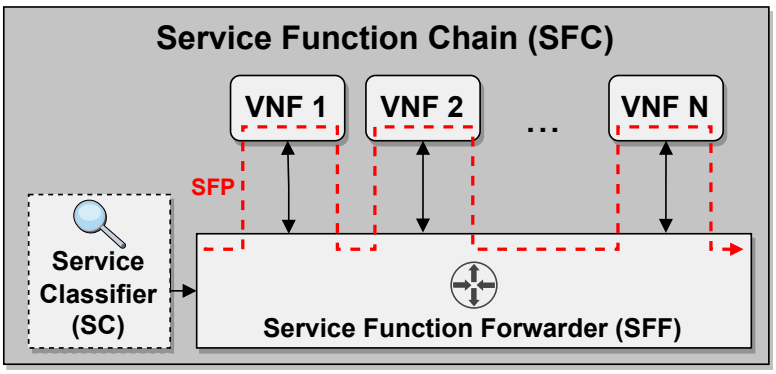

Figura 1. A arquitetura SFC da IETF.

O elemento de classificação, denominado de *Service Classifier* (SC), é responsável por receber o tráfego de rede e, através de um conjunto de regras, classificar e determinar para qual serviço (*i.e.,* SFC) o fluxo deverá ser enviado. As regras podem ser baseadas em diversos campos do pacote, por exemplo, endereços de origem e destino, protocolo (*e.g.,* UDP, TCP), entre outros. Após um pacote ser classificado, o SC é responsável por encapsular este pacote, incluindo informações para formar o NSH (*Network Service Header*) [Quinn et al. 2018]. Em particular, o NSH inclui informações que permitem o encaminhamento do pacote através do *Service Function Path* (SFP) – a ordem de VNFs da SFC através da qual o tráfego deverá percorrer.

Uma vez que o tráfego foi classificado e encapsulado pelo SC, os pacotes são enviados ao *Service Function Forwarder* (SFF), componente responsável pelo encaminhamento propriamente dito dos pacotes de acordo com a ordem definida pelo SFP. O SFF utiliza as informações de encapsulamento inseridas pelo SC para enviar o tráfego para a primeira VNF da SFC. Após o recebimento e processamento dos pacotes, a VNF

envia novamente o tráfego processado para a SFF. O SFF por sua vez recebe este tráfego, e realiza o encaminhamento para a próxima VNF de acordo com o SFP. Este processo é repetido até que a última VNF da SFP processe o tráfego. Por fim, ao receber o tráfego processado pela última VNF, o SFF remove o encapsulamento de cada pacote feito pelo SC e envia o tráfego para o destino final.

2.2. Trabalhos Relacionados

A estratégia apresentada em [Ghaznavi et al. 2020], denominada de FTC (*Fault Tolerant Chaining*), propõe um sistema que tolera falhas por parada em SFCs. Utilizando o próprio fluxo de processamento dos pacotes, o sistema FTC extrai e copia informações sobre o estado das VNFs. O FTC, no entanto, não utiliza nenhum tipo de réplica para as VNFs da SFC – cada instância de VNF na topologia de serviço age como uma réplica para a sua VNF sucessora. Tal característica torna o FTC um sistema não compatível com a arquitetura de referência SFC IETF, uma vez que demanda comunicação direta entre instâncias de VNF diferentes. Além disso, fica a cargo do operador da rede indicar quais operações causam mudança no estado de cada VNF, o que a torna uma abordagem propensa a erros.

A solução proposta em [Wang et al. 2021] assume a utilização de instâncias em modo *standby* para prover tolerância a falhas em serviços de rede virtualizados. Essa proposta emprega dois métodos de *Deep Reinforcement Learning* (DRL), cujo objetivo é decidir, de maneira eficiente, o mapeamento e a implantação de instâncias de VNF (tanto as ativas, quanto as *standby*) no substrato físico. Além disso, para SFCs que possuem VNFs *stateful*, um mecanismo extra é empregado para enviar constantemente o estado de uma instância de VNF ativa para as suas réplicas em *standby*. Vale ressaltar que essa estratégia de replicação de estado de VNFs *stateful* é bastante custosa em termos de mensagens trocadas no sistema. Por fim, esta solução também não é nativamente compatível com a arquitetura SFC IETF.

Em [Khalid and Akella 2019] um *framework* NFV é proposto para tolerar falhas de SFCs, denominado de CHC (*Correct, High performance Chains*). A estratégia adotada pelo CHC desacopla o processamento das VNFs do seu estado interno, que é armazenado posteriormente em um banco de dados distribuído. Em casos de falha, uma nova VNF deve ser instanciada e o seu estado recuperado do banco de dados. Apesar dos autores afirmarem que a solução introduz somente um pequeno atraso para cada pacote, a falha em alguma VNF da cadeia pode prejudicar significativamente a vazão e o tempo de indisponibilidade, uma vez que uma nova VNF deverá ser instanciada e o seu estado recuperado do banco de dados. Além disso, o CHC considera somente falhas por parada.

Em [Kulkarni et al. 2018] os autores propõe o REINFORCE, outro *framework* NFV para tolerar a falha de serviços virtualizados. O REINFORCE replica o estado interno de cada VNF para uma réplica dedicada através de *checkpoints*, aplicando o conceito de sincronia externa para maximizar a vazão do serviço. Assim como nos trabalhos anteriores, o *framework* considera somente falhas por parada, além de não ser compatível com a arquitetura IETF SFC, dificultando a sua integração em outros sistemas.

O sistema proposto em [Venâncio and Duarte Jr 2022] descreve o NHAM, uma arquitetura de alta disponibilidade para VNFs e SFCs. O NHAM utiliza estratégias baseadas em *checkpoints* de VNFs junto ao gerenciamento de *buffers* para permitir a falha de múltiplas VNFs de uma SFC. Além disso, o NHAM consegue garantir a recuperação

consistente de VNFs *stateful*, preservando o estado global do serviço. Apesar do NHAM ser totalmente compatível com a arquitetura de referência definida pela ETSI, ele não tolera falhas nos demais componentes da SFC (*i.e.,* SC e SFF). O NHAM também somente garante a alta disponibilidade de serviços que falham por parada.

Outro trabalho propõe Necklace [Esposito et al. 2020], uma arquitetura para a instanciação de SFCs robustas. O Necklace utiliza um algoritmo de consenso distribuído para garantir um bom desempenho do serviço e tolerar falhas de processos ou nos próprios enlaces de comunicação. A arquitetura, no entanto, tolera somente falhas não bizantinas. Já em [Kong et al. 2017], outra solução para garantir a alta disponibilidade de SFCs é proposta. Nesse caso, os autores consideram a utilização de *backups* tanto para os enlaces quanto para as instâncias relacionadas às VNFs de um serviço. Assim, através de um algoritmo heurístico, a solução define a quantidade de réplicas a serem criadas para cada VNF presente em uma topologia completa.

Após um exaustivo exame dos trabalhos relacionados da literatura, podemos afirmar que os trabalhos de tolerância a falhas de serviços de rede virtualizados consideram somente falhas por parada (*crash*) de VNFs e apenas o NHAM é proposto para a arquitetura de SFC da IETF. Desta forma, o presente trabalho pode ser entendido como o primeiro que *(i)* considera falhas não somente das VNFs, mas de qualquer componente da arquitetura SFC definida pela IETF e *(ii)* que tolera não somente falhas por parada, como também falhas por omissão e falhas bizantinas.

3. Uma Arquitetura SFC Tolerante a Falhas e Intrusão

Diversos serviços de rede fornecem funcionalidades críticas para manter uma infraestrutura de rede funcionando corretamente. Uma falha, por exemplo, de um serviço de segurança ou de engenharia de tráfego, pode comprometer toda a infraestrutura de rede. Esta seção propõe a FIT-SFC (*Fault- & Intrusion Tolerant* SFC), uma arquitetura tolerante a falhas para suportar serviços virtuais seguros e altamente disponíveis.

A arquitetura FIT-SFC foi projetada de forma a permitir que a arquitetura de referência SFC proposta pela IETF tolere falhas por parada (*i.e.,* falhas *crash*), falhas por omissão ou falhas bizantinas. Uma falha bizantina é uma falha arbitrária, normalmente utilizada para representar um componente que sofreu uma intrusão decorrente de um ataque. Neste trabalho, uma falha bizantina consiste do resultado de um ataque – uma função ou componente malicioso pode realizar modificações não autorizadas em pacotes que trafegam pela SFC, injetar pacotes na rede, ou ainda um componente pode tentar deixar as funções replicadas em estados inconsistentes.

Para evitar tais problemas, a estratégia utilizada pela FIT-SFC é baseada na replicação dos principais componentes da arquitetura SFC da IETF: SC, SFF e VNFs. Para permitir até f falhas bizantinas [Lamport et al. 2019], são necessárias $3f+1$ réplicas de cada componente. Além disso, a arquitetura proposta evita que pacotes sejam perdidos na presença de falhas, garantindo a consistência correta do serviço. O modelo de sistema adotado é parcialmente síncrono, com tempo de estabilização global (GST - *Global Stabilization Time*).

A FIT-SFC, cujo fluxo de execução é ilustrado na Figura 2, foi projetada de maneira totalmente compatível com a arquitetura de referência da IETF. A FIT-SFC também

consiste de uma sequência de VNFs, onde cada VNF precede e sucede um SFF. A SFC pode ser constituída por um único SFF, ou pode ser constituído de múltiplos SFFs compartilhados através de um ambiente distribuído e federado. O tráfego é recebido pelo SC através de uma determinada origem (*Traffic Source*). Dependendo da natureza do serviço virtual, o tráfego pode consistir de um único pacote contendo uma requisição, ou pode consistir de um fluxo de pacotes. Apesar da utilização de réplicas, os pacotes seguem o fluxo padrão definido pela arquitetura SFC da IETF (Seção 2.1).

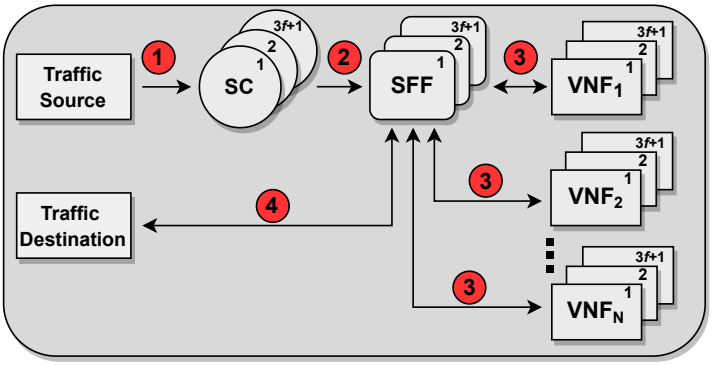

Figura 2. Fluxo de execução da FIT-SFC.

A seção seguinte (Subseção 3.1) descreve cenários com falhas bizantinas e discute a necessidade de se utilizar $3f + 1$ réplicas para cada componente. Já a Subseção 3.2 descreve o funcionamento da arquitetura, enquanto que a Subseção 3.3 discute como a consistência entre as múltiplas réplicas de VNFs é mantida de acordo com cada tipo de função.

3.1. Cenários com Falhas Bizantinas

Considerando que n é a quantidade total de réplicas de um componente do sistema, e f é a quantidade de réplicas com falhas bizantinas, é necessário que $n \geq 3f + 1$. Considere o elemento que recebe a saída do componente replicado e deve decidir pela mensagem correta a partir das mensagens vindas das réplicas. Como o modelo de falhas bizantinas inclui a omissão, deve ser possível tomar a decisão a partir de $n - f$ mensagens, pois f mensagens podem ser omitidas. Por outro lado, é possível que nenhuma omissão tenha acontecido e que entre as $n - f$ mensagens recebidas das réplicas, f tenham sido produzidas por componentes maliciosos. Portanto, dentro das $n - f$ mensagens é necessário ter uma maioria de mensagens vindas de réplicas corretas, para garantir a decisão correta. Isso só é possível se $(n - f) - f > f$, portanto $n > 3f$ ou $n \geq 3f + 1$.

A Figura 3 mostra, através de um exemplo, que $n \geq 2f + 1$ réplicas são insuficientes para garantir a tolerância a falhas bizantinas. Apenas uma das réplicas do primeiro componente é maliciosa, mas uma das réplicas corretas está lenta. O segundo componente deve tomar a decisão com $f = 2$ mensagens (uma poderia ter sido omitida) mas como há uma réplica maliciosa o segundo componente não chega a um quórum.

A Figura 4 traz outro exemplo mostrando que $n \geq 2f + 1$ réplicas são insuficientes. No caso, o cliente é malicioso e está em conluio com uma réplica do primeiro componente. Para as réplicas do primeiro componente, o cliente malicioso envia a mensagem #1 para uma réplica correta e a mensagem #2 para a outra réplica correta. A réplica

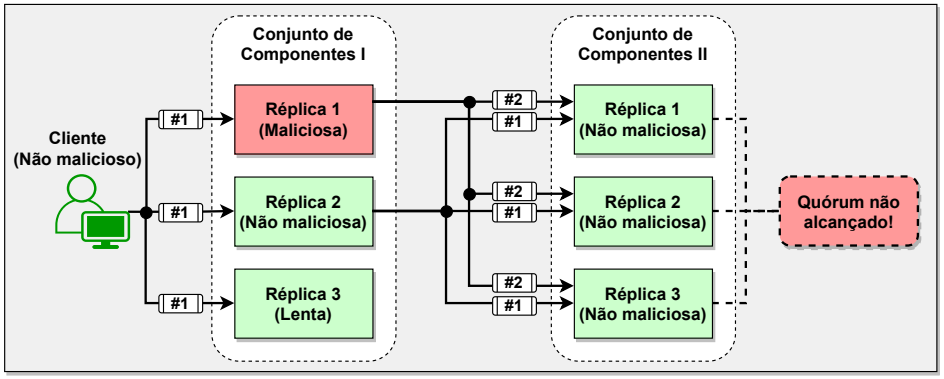

Figura 3. Cenário de impossibilidade de quórum (falhas bizantinas).

maliciosa conhece ambas as mensagens. As réplicas do segundo componente recebem as mensagens #1 e #2 das réplicas corretas do primeiro componente, mas a réplica maliciosa encaminha a mensagem #1 para uma réplica correta, a mensagem #2 para outra réplica correta, além de omitir a mensagem que seria enviada para a última réplica correta. Nesta situação, uma réplica correta obtém quórum para a mensagem #1, enquanto a outra réplica correta obtém quórum para a mensagem #2, ainda, a Réplica 3 não obtém quórum. Dessa forma, o número de réplicas deve ser de pelo menos $3f + 1$ para um quórum de pelo menos $2f + 1$ mensagens idênticas.

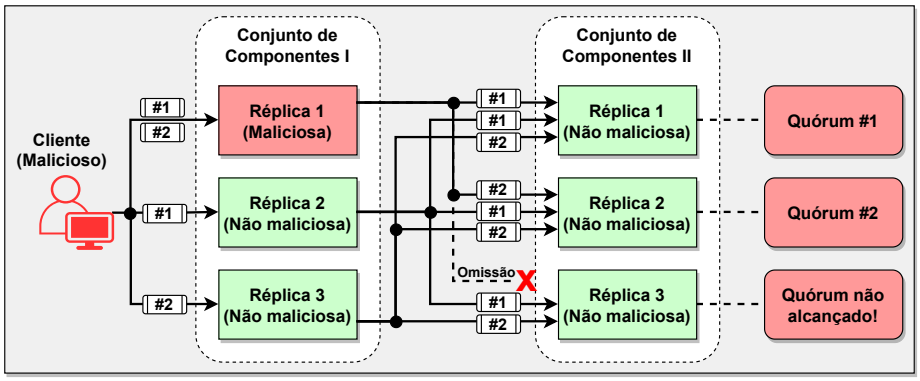

Figura 4. Cenário de conluio com cliente malicioso.

3.2. Funcionamento da Arquitetura FIT-SFC

Um cliente deve utilizar um *wrapper* para encaminhar requisições para um serviço FIT-SFC. O *wrapper* tem duas finalidades: *(i)* enviar uma cópia de cada pacote para cada uma das $3f+1$ réplicas do SC e; *(ii)* marcar cada pacote enviado com um *timestamp* local, que pode ser implementado como um contador local de pacotes que serão transmitidos para a SFC, cujo objetivo é permitir a identificação única de cada pacote.

O tráfego é então recebido por cada uma das réplicas do SC (denominada de *FIT-Classifier*), onde cada uma delas processa e encaminha uma única vez cada pacote recebido para cada uma das $3f + 1$ réplicas de SFF. Como pode ser observado na Figura 5, antes do SFF realizar o encaminhamento do tráfego entre as VNFs do SFP correspondente, é necessário considerar que podem haver até f instâncias de SC falhas ou que a

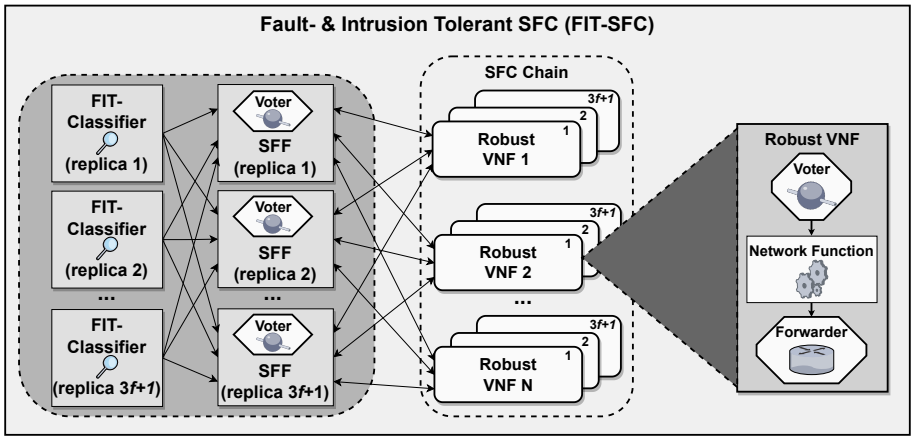

Figura 5. A arquitetura FIT-SFC.

origem do tráfego é maliciosa. Portanto, cada SFF deve realizar uma votação para selecionar o pacote correto a ser encaminhado para a VNF replicada. É importante observar que versões diferentes de pacotes podem ser recebidas devido a falhas bizantinas.

Para viabilizar esta funcionalidade, um elemento adicional é incorporado às réplicas de SFF, denominado de *Voter*. A partir do momento em que o *Voter* recebe $2f + 1$ cópias idênticas de um mesmo pacote (*i.e.,* um quórum), este é autorizado a processar e encaminhar o mesmo para o destino seguinte (*i.e.,* as VNFs). O número de cópias adotado pode ser entendido da seguinte forma: de maneira inicial, para existir uma decisão unânime entre todos os componentes de um determinado sistema é necessário o recebimento de uma quantidade mínima de mensagens provenientes de componentes corretos, sendo que esta deve superar o número de mensagens provenientes de (ou omitidas por) componentes falhos. Ou seja, ao receber $2f + 1$ cópias do pacote, é possível que até f cópias tenham sido produzidas por componentes bizantinos. Consequentemente, $f + 1$ cópias necessariamente correspondem ao pacote original correto. Sendo assim, a estratégia de replicação garante a tolerância de f falhas de cada tipo de componente dado um número de réplicas de cada componente igual a $3f + 1$. Entretanto, se não houver um quórum de $2f + 1$ pacotes coincidentes, o encaminhamento e processamento do tráfego é imediatamente interrompido.

Em seguida, cada réplica do SFF envia a sua cópia do pacote para cada uma das réplicas da VNF, de acordo com a ordem definida no SFP. Semelhante aos SFFs, cada réplica da VNF também possui um *Voter* que aguarda por $2f + 1$ cópias idênticas de um determinado pacote, antes de efetivamente realizar o processamento da função. Da mesma forma, se menos do que esse número de cópias for recebido pelo *Voter* das VNFs, o fluxo deve ser interrompido.

Após cada réplica de VNF completar a sua execução, o subcomponente *Forwarder* envia uma cópia de cada pacote de volta para cada réplica de SFF. Nesta etapa, cada réplica de SFF realiza uma nova votação para obter um quórum de $2f + 1$ pacotes, antes de encaminhar os pacotes para as próximas réplicas da VNF de acordo com o SFP. Este processo se repete até que o tráfego tenha sido processado pela última VNF replicada da SFC. Finalmente, cada réplica do SFF envia uma cópia do pacote para o destino final. Novamente, uma votação final é realizada e o pacote só é entregue no destino se houver

$2f + 1$ cópias idênticas.

3.3. Garantindo a Consistência de VNFs Replicadas

Ao considerar VNFs replicadas, a consistência entre as múltiplas réplicas deve ser mantida. Existem diferentes tipos de funções de rede, e o tráfego que essas funções processam podem ou não modificar o seu estado interno. Se o estado interno da VNF é desacoplado do seu processamento, dizemos que esta VNF é *stateless*. Por outro lado, se o estado interno da função de rede muda de acordo com os pacotes processados, dizemos que ela é *stateful*. Para garantir a consistência do serviço, cada VNF replicada executa um protocolo diferente de acordo com o tipo de consistência esperado. Em particular, este trabalho considera que os serviços de rede podem ser classificados de três formas diferentes: *(i)* serviços *stateless*; *(ii)* serviços *stateful* por fluxo e; *(iii)* serviços *stateful* global. Cada tipo de serviço é descrito a seguir.

Para os serviços *stateless*, uma vez que eles não possuem estado, basta realizar o processamento dos pacotes recebidos de acordo com a ordem de recebimento. Como estas funções não mantém estado, a consistência do serviço não é comprometida se somente parte das réplicas da VNF executam um determinado pacote.

Por outro lado, funções de rede *stateful* por fluxo possuem um estado para cada fluxo percorrendo o serviço, *i.e.,* necessita de ordem parcial. Portanto, os pacotes provenientes de um mesmo fluxo devem ser processados na mesma ordem por todas as réplicas de VNFs executando uma mesma função. Este cenário é tratado na arquitetura FIT-SFC através do *timestamp* local adicionado pelo *wrapper* dos clientes, o qual é utilizado para criar um ordenamento no processamento padrão para os pacotes de um fluxo.

Por fim, as funções de rede do tipo *stateful* global recebem múltiplos fluxos de diferentes origens, e o seu estado é definido a partir do processamento de todos os pacotes recebidos. Neste caso, cada réplica precisa executar cada pacote de cada fluxo na mesma ordem, *i.e.,* necessita de ordem total. A existência de um estado único demanda que todos os pacotes sejam processados na mesma ordem por todas as réplicas de VNFs que executam uma mesma função, sendo necessário adotar um mecanismo de consenso entre as mesmas. Um exemplo clássico deste tipo de serviço é um balanceador de cargas. Esse tipo de função recebe fluxos de diferentes origens e distribui a carga para diferentes servidores. Ao replicar este tipo de serviço, é essencial garantir que todas as réplicas processam e enviam os pacotes na mesma ordem para o mesmo conjunto de servidores.

Em particular, a arquitetura FIT-SFC implementa o algoritmo de consenso PBFT (*Practical Byzantine Fault Tolerance*) [Castro and Liskov 2002]. O consenso é executado somente para os serviços *stateful* global, garantindo a ordem total entre fluxos, além de garantir que todas as réplicas da VNF executam a mesma sequência de pacotes. O consenso é implementado junto ao componente *Voter* da VNF e a execução do consenso é realizada antes da etapa de votação. Dessa forma, a entrega dos pacotes para cada *Voter* é feita na mesma ordem.

4. Implementação e Avaliação Experimental

Esta seção apresenta uma avaliação do custo imposto pela arquitetura FIT-SFC sobre os serviços tolerantes a falhas e intrusões. Inicialmente, é apresentada a implementação de um protótipo, além dos três serviços utilizados na avaliação experimental. Em seguida,

são apresentados os resultados para o custo em termos da latência, comparando versões das SFCs construídas com e sem a FIT-SFC.

4.1. Implementação da Arquitetura FIT-SFC e dos Serviços Virtualizados

Como prova de conceito, um protótipo da arquitetura FIT-SFC foi desenvolvido[1]. O protótipo implementa todos os elementos operacionais da arquitetura SFC da IETF (descrita na Seção 2) utilizando a linguagem Python 3.9.1. Os elementos realizam todas as trocas de mensagens descritas na Seção 3, tolerando até f falhas bizantinas em cada componente da arquitetura.

Além da configuração de rotas e fluxos tradicionais da arquitetura SFC da IETF, os SCs da arquitetura FIT-SFC são configurados com informações sobre as réplicas de SFFs em execução, permitindo a troca de mensagens entre elas. Por sua vez, os SFFs também são configurados para identificar, através do endereço IP, cada réplica dos SCs e das VNFs dos serviços instanciados. Essa configuração é necessária uma vez que o SFF recebe cópias de mensagens de cada réplica do SC e de cada réplica de VNF dos serviços instanciados. Por fim, as VNFs também são configuradas para identificar todas as réplicas de SFFs, permitindo tanto o envio quanto o recebimento de tráfego entre estes elementos.

De forma a verificar o funcionamento correto da arquitetura FIT-SFC, além de avaliar o custo para tolerar as falhas e intrusões, três serviços de rede virtualizados foram implementados: um serviço de DNS, um serviço de DNS autenticado e um serviço de balanceamento de cargas. Os serviços são brevemente descritos a seguir.

DNS. O serviço de nomes (*Domain Name System* - DNS) implementado é baseado na especificação definida no RFC 1035 [Mockapetris 1987]. O serviço recebe como argumento um domínio e retorna ao usuário as informações correspondentes. Neste caso, a SFC consiste de uma única VNF de DNS. Uma vez que o tráfego processado por essa VNF não modifica o seu estado interno, este serviço é classificado como *stateless*.

DNS autenticado. O segundo serviço implementado é um DNS autenticado. As funcionalidades são as mesmas do serviço anterior, no entanto existe uma etapa adicional de autenticação. Neste caso, a SFC consiste de duas VNFs: uma VNF que realiza somente a autenticação, e outra que executa a funcionalidade tradicional de DNS. Este serviço é considerado *stateful por fluxo*, pois necessita armazenar informações de autenticação do usuário enquanto o fluxo está ativo.

Balanceador de cargas. O terceiro e último serviço implementado é um balanceador de cargas. O serviço implementa uma política *round-robin* em que cada requisição é distribuída de maneira sequencial e ciclicamente para um conjunto de N servidores. A SFC deste serviço consiste de uma única VNF, e o serviço é *stateful entre fluxos* (global), uma vez que cada pacote recebido altera o contador de pacotes de cada instância do balanceador de cargas. Dessa forma, para garantir a consistência entre as múltiplas réplicas da SFC, todas as instâncias do serviço devem processar os mesmos fluxos/pacotes na mesma ordem. Portanto, para este serviço, a etapa de consenso (descrita na Seção 3) é necessária.

4.2. Resultados Obtidos

Os cenários de testes da arquitetura FIT-SFC foram construídos utilizando o emulador NIEP [Tavares et al. 2018] e executados em uma máquina com processador Intel Core I7

[1]Disponível em `https://github.com/ViniGarcia/FIT-SFC`

@ 2.5GHz, 16GB RAM DDR4 e Ubuntu 16.04. Para cada experimento descrito nesta seção, dois cenários são comparados. O primeiro cenário executa os elementos operacionais da arquitetura FIT-SFC, porém utiliza apenas uma instância de cada elemento, *i.e.,* sem tolerância a falhas nem intrusões ($f = 0$). Já o segundo cenário consiste de quatro réplicas de cada elemento da arquitetura FIT-SFC, *i.e.,* com tolerância a falhas ($f = 1$). Todos os experimentos relatados foram executados 50 vezes e as médias dos resultados são apresentadas com um intervalo de confiança de 95%.

Os experimentos descritos a seguir avaliam, para cada serviço implementado, qual o impacto em termos de latência ao utilizar a arquitetura FIT-SFC. A latência total para cada cenário (*i.e.,* com e sem replicação) considera a progressão do tempo total de processamento de um pacote do momento da emissão do mesmo pelo cliente até o seu recebimento pelo servidor ao qual foi destinado, passando antes por todos os elementos operacionais da arquitetura FIT-SFC. Além disso, cada experimento determina o tempo consumido em cada etapa do fluxo de execução: *(i)* SC: tempo despendido entre o cliente e os SCs; *(ii)* SFF: tempo despendido entre os SCs e os SFFs; *(iii)* NF: tempo despedindo entre os SFFs e as funções de rede (*Network Function* - NF).

O primeiro experimento, ilustrado na Figura 6, avalia a latência para o serviço DNS. O tamanho de cada requisição é de 512 Bytes. Para o cenário sem replicação, o tempo total da requisição foi de 12.16ms (*i.e.,* Origem $\rightarrow$ SC $\rightarrow$ SFF $\rightarrow$ NF $\rightarrow$ SFF $\rightarrow$ Destino). Do tempo total, aproximadamente 65% foi dedicado a etapa SC, uma vez que o tráfego precisa ser classificado e cada pacote deve ser encapsulado com o NSH. Já as etapas de SFF e NF, ambas representam aproximadamente 17.5% do tempo total da requisição. O tempo para estas etapas é menor, visto que o SFF simplesmente encaminha pacotes (*i.e.,* não realiza processamento) e a NF atende uma requisição local.

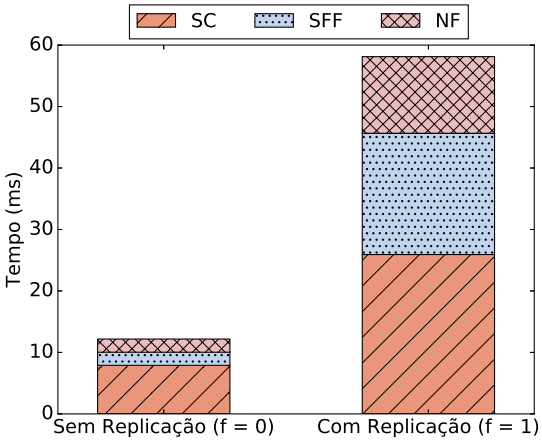

Figura 6. Comparação da latência do serviço de DNS.

Ao comparar com o cenário com tolerância a falhas ($f = 1$), o tempo total da requisição aumenta de 12.16ms para 58.1ms – aproximadamente 4.7 vezes maior. Esse impacto ocorre em decorrência das trocas de mensagens adicionais e das comparações que o componente *Voter* realiza para garantir que ao menos $2f + 1$ réplicas (nestes experimentos, 3 réplicas) receberam o mesmo pacote. Em particular, o tempo da etapa de SC aumenta de 7.89ms para 25.92ms, a etapa de SFF aumenta de 2.13ms para 19.72ms, enquanto que a etapa de NF aumenta de 2.143ms para 12.46ms.

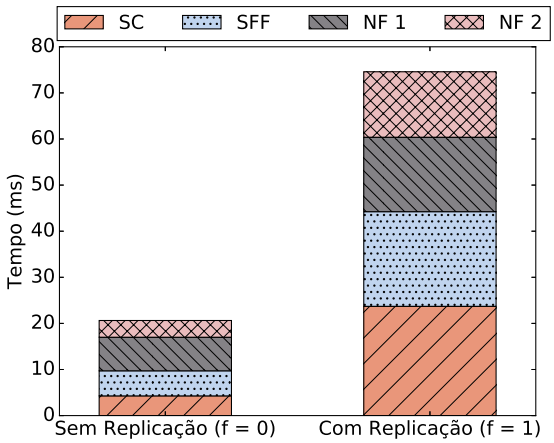

Figura 7. Comparação da latência do serviço de DNS autenticado.

O segundo experimento mede a latência para o serviço de DNS autenticado, ilustrado na Figura 7. O tamanho de cada requisição também é de 512 Bytes, além das informações relacionadas à autenticação. A diferença para o experimento anterior é que a SFC para o DNS autenticado é composta de duas VNFs. No cenário sem replicação, o tempo total de cada requisição é de 20.6ms – valor levemente maior em comparação com o experimento anterior devido ao passo extra de autenticação (*i.e.,* etapa NF 1) e ao encaminhamento adicional do SFF (*i.e.,* Origem $\rightarrow$ SC $\rightarrow$ SFF $\rightarrow$ NF 1 $\rightarrow$ SFF $\rightarrow$ NF 2 $\rightarrow$ SFF $\rightarrow$ Destino). Em comparação com o serviço anterior, o tempo da etapa de SFF aumenta linearmente, devido ao encaminhamento adicional com duas funções de rede.

No cenário com tolerância a falhas, o tempo total da requisição aumenta para 74.56ms – aproximadamente 3.6 vezes maior. Ao comparar com o serviço de DNS sem autenticação, houve um aumento de 16.46ms, tempo que corresponde ao processamento da função de autenticação (NF 1) e ao passo adicional entre as VNFs e o SFF.

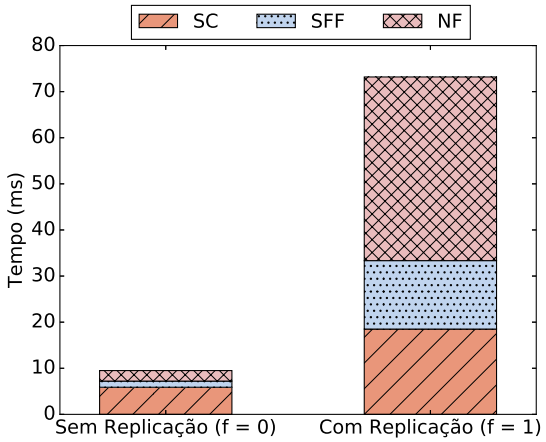

Figura 8. Comparação da latência do serviço de balanceamento de cargas.

Finalmente, o terceiro experimento avalia a latência do serviço de balanceamento de cargas, ilustrado na Figura 8. Para este experimento, o tamanho de cada requisição é de 1024 Bytes. No cenário sem replicação, o tempo total de cada requisição foi de

9.47ms, dos quais 61.9% foi destinado a etapa de classificação (SC), enquanto que 14.04% e 23.9% foram destinados as etapas de SFF e NF, respectivamente.

Comparando novamente com o cenário com replicação, o tempo total de cada requisição aumenta para 73.17ms – aproximadamente 7.7 vezes maior. A maior parte da sobrecarga imposta para esta função está no consenso, que deve ser executado para garantir a consistência entre as múltiplas réplicas do balanceador de cargas. Em particular, a etapa de NF para este cenário consome 54.4% do tempo, o que corresponde ao tempo de processamento da função, além do tempo de execução do algoritmo de consenso.

Naturalmente, o uso da arquitetura FIT-SFC tem um custo associado em termos de aumento da latência. Os componentes são replicados, aumentando a quantidade de pacotes redundantes transitando na rede e que precisam ser recebidos e processados.

5. Conclusão

Este trabalho apresenta a FIT-SFC, uma arquitetura baseada em replicação para prover tolerância a falhas e intrusão para serviços virtualizados de rede. A FIT-SFC permite que os próprios componentes internos da SFC sofram falhas por parada, omissão ou falhas por intrusão. Para tolerar f falhas, cada componente é configurado com $3f+1$ réplicas. Além disso, o componente *Voter* é incorporado nos SFFs e nas VNFs, responsável por comparar diferentes cópias de mensagens e, somente após receber um quórum de $2f + 1$ cópias idênticas, autorizar o processamento do pacote. Um protótipo foi desenvolvido e avaliado através de três serviços de rede: um DNS, um DNS autenticado e um balanceador de cargas. Resultados experimentais demonstram que apesar do custo imposto pela solução em termos de latência, serviços podem se tornar seguros e altamente disponíveis sem a necessidade de realizar extensas modificações no seu código fonte. Trabalhos futuros incluem a definição de estratégias para otimizar o tempo total de processamento, como a execução do tráfego em lotes.

Referências

Castro, M. and Liskov, B. (2002). Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461.

Chiosi, M., Clarke, D., Willis, P., Reid, A., Feger, J., Bugenhagen, M., Khan, W., Fargano, M., Cui, C., Deng, H., et al. (2012). Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action. In *SDN and OpenFlow World Congress*, pages 22–24.

Cotroneo, D., De Simone, L., Liguori, P., Natella, R., and Bidokhti, N. (2019). How bad can a bug get? an empirical analysis of software failures in the openstack cloud computing platform. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 200–211.

Esposito, F., Mushtaq, M., Berno, M., Davoli, G., Borsatti, D., Cerroni, W., and Rossi, M. (2020). Necklace: An architecture for distributed and robust service function chains with guarantees. *IEEE Transactions on Network and Service Management*, 18(1):152–166.

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

Ghaznavi, M. et al. (2020). Fault tolerant service function chaining. In *ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 198–210.

Gill, P., Jain, N., and Nagappan, N. (2011). Understanding network failures in data centers: measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM 2011 Conference*, pages 350–361.

Gunawi, H. S., Hao, M., Suminto, R. O., Laksono, A., Satria, A. D., Adityatama, J., and Eliazar, K. J. (2016). Why does the cloud stop computing? lessons from hundreds of service outages. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 1–16.

Halpern, J. and Pignataro, C. (2015). Service Function Chaining (SFC) Architecture. RFC 7665, IETF.

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

Han, B., Gopalakrishnan, V., Kathirvel, G., and Shaikh, A. (2017). On the resiliency of virtual network functions. *IEEE Communications Magazine*, 55(7):152–157.

Huff, A. et al. (2018). A holistic approach to define service chains using click-on-osv on different nfv platforms. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE.

Khalid, J. and Akella, A. (2019). Correctness and performance for stateful chained network functions. In *The 16th NSDI*, pages 501–516, Boston. USENIX Association.

Kong, J. et al. (2017). Guaranteed-availability network function virtualization with network protection and vnf replication. In *Global Communications Conference*, pages 1–6.

Kulkarni, S. G. et al. (2018). Reinforce: Achieving efficient failure resiliency for network function virtualization based services. In *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, pages 41–53, Heraklion. ACM.

Lamport, L., Shostak, R., and Pease, M. (2019). The byzantine generals problem. In *Concurrency: the works of leslie lamport*, pages 203–226. ACM.

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

Mockapetris, P. V. (1987). Rfc1035: Domain names-implementation and specification.

Pattaranantakul, M., He, R., Song, Q., Zhang, Z., and Meddahi, A. (2018). Nfv security survey: From use case driven threat analysis to state-of-the-art countermeasures. *IEEE Communications Surveys & Tutorials*, 20(4):3330–3368.

Quinn, P. et al. (2015). Problem Statement for Service Function Chaining - RFC 7498. Technical report, Internet Engineering Task Force.

Quinn, P. et al. (2018). Network Service Header (NSH) - RFC 8300. Technical report, Internet Engineering Task Force.

Sharma, S., Engelmann, A., Jukan, A., and Gumaste, A. (2020). Vnf availability and sfc sizing model for service provider networks. *IEEE Access*, 8:119768–119784.

Tavares, T. N. et al. (2018). Niep: Nfv infrastructure emulation platform. In *International Conference on Advanced Information Networking and Applications*, pages 173–180.

Venâncio, G. and Duarte Jr, E. P. (2022). NHAM: An nfv high availability architecture for building fault-tolerant stateful virtual functions and services. In *11th Latin-American Symposium on Dependable Computing (LADC)*. IEEE.

Wang, L. et al. (2021). Ddqp: A double deep q-learning approach to online fault-tolerant sfc placement. *IEEE Transactions on Network and Service Management*, 18(1):118–132.