# Implementação de uma Função Virtualizada de Rede para Detecção de Falhas

**Rogério C. Turchetti**[1,2], **Elias P. Duarte Jr.**[2]

[1] Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19.081 – 81.531-980 – Curitiba – PR – Brasil

[2]CTISM - Universidade Federal de Santa Maria (UFSM)
Avenida Roraima, 1000 – 97105-900 – Santa Maria – RS – Brasil

turchetti@redes.ufsm.br, elias@inf.ufpr.br

***Abstract.*** *Network Function Virtualization (NFV) is an emerging technology that uses software virtualization techniques to implement network functions that are usually deployed on specific hardware and software devices. With NFV technology it is possible to design, deploy, and manage network functions in a fraction of the time it often takes to do the same in non-virtualized settings. NFV improves the flexibility and reduces the time for the development, deployment and management of new functions. In this paper, we propose a NFV to detect process and link failures, called NFV-FD. NFV-FD relies on an OpenFlow controller from which information about the network is obtained. With this information NFV-FD keeps track of the state of both processes and links. NFV-FD was implemented and experimental results are reported for the amount of resources required by the virtual function, as well as the quality of the failure detection and notification service.*

***Resumo.*** *Network Function Virtualization (NFV) é uma tecnologia emergente que implementa, utilizando técnicas de virtualização, funções de rede, antes vinculadas a dispositivos de hardware e software específicos. A tecnologia de NFV possibilita projetar, desenvolver e gerenciar funções diversas de rede, oferecendo maior flexibilidade dos serviços e reduzindo o tempo para o desenvolvimento de novas funções. Neste trabalho, propõem-se uma NFV para detectar falhas em processos e enlaces da rede, denominada NFV-FD. O mecanismo para monitoramento dos processos executado pela NFV-FD é auxiliado por um controlador OpenFlow que disponibiliza informações sobre uma rede SDN. A partir destas informações, a NFV-FD realiza o monitoramento dos processos e enlaces de comunicação. A NFV-FD foi implementada e são apresentados resultados experimentais do uso de recursos e da qualidade para a detecção e notificação de falhas.*

## 1. Introdução

Por décadas as funcionalidades das arquiteturas de redes de computadores têm sido fornecidas por dispositivos de rede implementados em hardware e software (*middleboxes*) para prover serviços como, roteamento, filtros de dados, gerenciamento, confiabilidade, segurança, entre outros [Xilouris et al. 2014]. Se por um lado *middleboxes* introduzem benefícios em termos de prover funcionalidades, por outro, eles constituem uma importante

fração de despesas operacionais (*OPerational EXpenditures - OPEX*) como a manutenção de equipamentos e despesas de capital (*CAPital EXpenditures - CAPEX*) como a aquisição de novos equipamentos de redes [Cotroneo et al. 2014]. Além disso, *middleboxes* possuem limite de flexibilidade, são ineficientes em termos de custo de energia, difíceis de gerenciamento e de solucionar problemas [Sherry et al. 2012].

Funções Virtualizadas de Rede (NFV - *Network Function Virtualization*) surgem para 'amenizar' estes problemas. Em geral, as NFVs exploram tecnologias de virtualização para transformar equipamentos de redes em entidades virtuais [ETSI 2015], podendo ser implementadas em conjunto com outras tecnologias emergentes, como a execução em redes SDN (*Software Defined Networks*) [Han et al. 2015]. Indústria, academia e organismos de padronização têm mostrado interesse em projetos para NFV [Bondan et al. 2014], pois é possível estabelecer novos mecanismos para operar e desenvolver serviços de rede [Ferrer Riera et al. 2014]. O ETSI (*European Telecommunications Standards Institute*) criou um grupo de pesquisa em NFV focando seus esforços para desenvolver NFVs usando plataforma de servidores comerciais [Xilouris et al. 2014]. O ETSI tem, como uma motivação a novas pesquisas, apresentado estudos de caso e possíveis temas para pesquisas em NFV, os autores relatam em [ETSI 2015] que métodos para recuperação e detecção de falhas ainda precisam ser investigados.

Neste sentido, este trabalho objetiva propor e implementar uma NFV para dar suporte à tolerância a falhas em aplicações distribuídas. O serviço proposto executa as funções de um detector de falhas, que tem como objetivo descobrir os estados dos processos através de seu monitoramento. A função para monitoramento dos processos é denominada de NFV-FD (*FD - Failure Detector*), e utiliza os benefícios de uma rede SDN através do compartilhamento de informações fornecidas por um controlador da rede. O compartilhamento destas informações auxilia na obtenção dos atributos dos processos a serem monitorados, bem como, permite determinar o próprio estado destes processos. Além disso, são também obtidas informações de monitoramento dos estados dos enlaces de comunicação, a partir dos próprios dispositivos de rede (*switches*).

A NFV-FD trabalha na formação de uma visão que indica quais processos estão suspeitos de terem falhado. Esta visão dos estados pode ser consultada por uma aplicação distribuída responsável por tomar decisões. Em outras palavras, a NFV-FD possibilita a execução de algoritmos distribuídos tolerantes a falhas. Como exemplo deste tipo de aplicação pode-se citar um serviço para difusão confiável [Guerraoui and Rodrigues 2006]. Este algoritmo é amplamente conhecido na literatura, sendo utilizado neste trabalho como estudo de caso, isto é, ele realiza a difusão confiável com base nas informações obtidas pela NFV-FD. O trabalho ainda apresenta e detalha questões sobre a implementação de uma NFV. O desempenho do serviço de detecção de falhas e o impacto da NFV na utilização dos recursos para seu processamento são também avaliados neste artigo.

O trabalho está organizado da seguinte forma. A seção 2 apresenta uma breve descrição sobre o conceito e trabalhos realizados em NFV. Na seção 3 é apresentado o modelo de sistema, juntamente com as definições consideradas neste trabalho. Para contextualizar a metodologia para a detecção de falhas, a seção 4 detalha a arquitetura e as características de funcionamento da NFV-FD. Os experimentos e as conclusões do trabalho são apresentados nas seções 5 e 6, respectivamente.

## 2. *Network Function Virtualization*

Essencialmente, uma NFV implementa funções de rede através de técnicas de virtualização de software para executá-las em hardware de prateleira. Este cenário pode ser visto na Figura 1, onde dispositivos de rede são migrados para uma solução baseada em NFV. As funções virtualizadas podem ser instanciadas/executadas por demanda quando necessárias e destruídas quando estiverem ociosas. Dessa forma, uma NFV pode estabelecer novos mecanismos para operar e desenvolver funções de rede, com possibilidades de economia de energia e redução de espaço físico [Han et al. 2015].

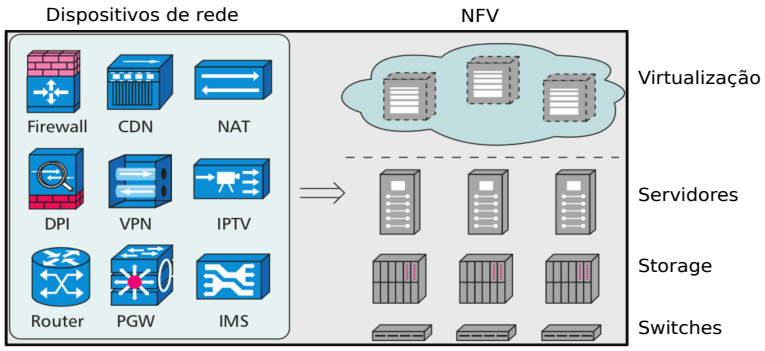



**Figura 1. De Dispositivos de Rede Baseados em Hardware para Soluções Baseadas em NFV [Han et al. 2015]**

Segundo os autores em [Haleplidis et al. 2014] os projetos em NFV estão concentrados em 3 principais objetivos. Primeiro, reduzir os custos de aquisição de equipamentos físicos, dedicados e dispendiosos (OPEX e CAPEX). Segundo, prover novos serviços aos usuários em um rápido ciclo de desenvolvimento. Terceiro, reduzir o custo global necessário para o gerenciamento de um grande número de dispositivos.

A possibilidade de desenvolver serviços através da implementação e virtualização de funções de rede, tem feito crescer a pesquisa por novos projetos em NFV. Há várias soluções para a implementação de uma NFV, disponíveis na literatura que consideram sua execução em uma rede OpenFlow [Batalle et al. 2013] [Fukushima et al. 2014] [Vilalta et al. 2015]. OpenFlow é o protocolo mais comum para a construção de redes SDN, possibilitando a separação entre o plano de controle e o plano de dados, definindo regras para a comunicação entre os dispositivos da rede, através de um controlador[1].

Em [Batalle et al. 2013] os autores propõem uma função virtualizada para o encaminhamento de pacotes. A NFV separa os fluxos dos pacotes do protocolo IPv4 e IPv6, oferecendo um roteamento entre domínios utilizando uma rede OpenFlow. Resumidamente, a ideia é criar um módulo dentro do controlador responsável por selecionar os pacotes que devem ser processados pela NFV. Os autores conseguem reduzir o fluxo de entrada nos *switches* transferindo funcionalidades (seleção dos fluxos) para uma máquina externa que implementa a função virtualizada. Segundo os autores, esta foi a primeira validação de implementação de uma NFV, mas que ainda carece de resultados mais robustos.

[1]O controlador é responsável por gerenciar o encaminhamento dos pacotes nos *switches*. Um *switch* é responsável por encaminhar os pacotes aos destinatários com base em regras instaladas pelo controlador.

No trabalho proposto em [Cerrato et al. 2014], os autores utilizam uma abordagem diferente, eles consideram diversas funções virtualizadas em um servidor. A ideia é propor um algoritmo para mover os dados entre funções virtualizadas de forma eficiente e isolada. A proposta trabalha em uma abordagem mestre/escravo[2], onde o mestre decide quais funções devem processar um dado pacote, e os escravos são as funções virtualizadas que processam os pacotes. A ideia é uma espécie de escalonamento de tarefas, onde o mestre controla o caminho dos pacotes entre as diversas funções.

Preocupados com o gerenciamento em ambientes com diversas NFVs, no trabalho [Fukushima et al. 2014] é proposto um *framework* que permite a um operador de rede descrever as NFs (*Network Functions*) virtualizadas através de suas características e atributos. O *framework* captura estas informações, cria relações entre as variáveis de diversas NFs e a consistência é sistematicamente validada. Como exemplo, considere duas NFs: uma para NAT (*Network Address Translation*) e outra para prevenção de intrusão IPS (*Intrusion Prevention System*). Os pacotes que passam pela função NAT dependerão do endereço da origem (atributo) para serem processados, ao passo que a função de IPS deverá processar todos os pacotes. Considerando a existência de uma DMZ (*DeMilitarized Zone*), os pacotes originados desta rede não devem ser processados pela função NAT , ao passo que os pacotes da rede privada devem ter os endereços traduzidos. Através do *framework* a ideia é criar estas regras para que as funções virtualizadas sejam configuradas. A proposta pode ser considerada relevante, mas ainda há problemas a serem resolvidos, como exemplo, a proposta não suporta configurações dinâmicas e o *script*, gerado pelo *framework*, é executado no controlador OpenFlow sobrecarregando as tarefas deste controlador.

Em [Vilalta et al. 2015] NFVs são utilizadas através de uma arquitetura em nuvem, para disponibilizar serviços executados por um PCE (*Path Computation Element*). PCE é um elemento responsável pelo mecanismo de cálculo de rota para criação de caminhos em uma rede de transporte. O objetivo dos autores é escalar o serviço oferecido por estes elementos. Para isso, um servidor PCE dedicado é removido e suas funcionalidades são virtualizadas (vPCE) e transportadas para um sistema em nuvem. Os resultados da implementação do PCE como uma NFV, demonstraram que o vPCE foi capaz de garantir um tempo médio de processamento, mesmo perante a picos de solicitações para o cômputo dos caminhos.

## 3. Modelo de Sistema

Neste trabalho considera-se um sistema distribuído assíncrono de topologia arbitrária composto por um conjunto $\Pi$ de $n$ processos/hosts, incrementado com detector de falhas [Chandra and Toueg 1996]. Tanto os processos como os enlaces falham por colapso (*crash*), ou seja, deixam de executar suas tarefas prematuramente. O monitoramento dos processos é realizado por um detector de falhas que pode cometer enganos. O monitoramento dos processos, pelo detector de falhas, resulta nos seguintes estados: suspeito (*S: suspect*), não falho (*T: trust*) ou inatingível (*U: unreachable*). O estado *S* é definido quando uma resposta de uma requisição de vida não é obtida dentro de um intervalo específico de tempo. O estado *S* relata a percepção momentânea do detector de falhas referente ao estado de um determinado processo. Entretanto, se a mensagem chegar dentro do inter-

[2]Na qual o mestre é um *switch* virtual e os escravos são funções virtualizadas

valo de tempo, o respectivo processo é considerado no estado *T*. O estado *U* ocorre quando é reportada falha em canais de comunicação que, consequentemente, deixam processos inacessíveis.

Assume-se que falhas não afetam a comunicação entre o detector de falhas e o controlador da rede. Além disso, o canal não cria, não perde e não altera mensagens. A rede é também formada por um controlador que nunca falha.

## 4. Proposta de uma NFV para Detecção de Falhas

Nesta seção são apresentados conceitos sobre detectores de falhas (seção 4.1) e seu funcionamento como uma NFV (seção 4.2). A arquitetura em que a NFV-FD está sendo proposta é apresentada na seção 4.3, por fim são apresentados os mecanismos para detecção de falhas nos processos (seção 4.4) e enlaces de comunicação (seção 4.5).

### 4.1. Detectores de Falhas

Em sistemas distribuídos tolerantes a falhas, são frequentes as situações em que os processos precisam entrar em um acordo referente a uma decisão a ser tomada. Como exemplo, pode-se citar transações atômicas em banco de dados, difusão confiável, *membership* de grupo, eleição de líder, entre outras [Charron-Bost et al. 2010]. A realização do acordo distribuído pode não ter uma solução trivial [Borran et al. 2012]. Considerando sistemas síncronos ou isentos de falhas, as propriedades do algoritmo podem ser garantidas e o consenso pode ser resolvido trivialmente. Entretanto, o principal impasse para resolvê-lo ocorre em ambientes assíncronos sujeitos a falhas [Fischer et al. 1985]. Neste contexto, uma alternativa é utilizar um modelo parcialmente síncrono formado por um assíncrono incrementado com detectores de falhas.

Detectores de falhas foram propostos por Chandra e Toueg [Chandra and Toueg 1996] e sua abstração possibilita encapsular o indeterminismo permitindo garantir as propriedades do algoritmo de consenso, mesmo perante falhas por colapso. Sendo assim, um detector de falhas tem por objetivo determinar os estados dos processos. Os estados indicam se um processo está falho ou não falho de acordo com o monitoramento que ocorre por troca de mensagens e respeitando um limite de tempo (*timeout*). Dois algoritmos tradicionais denominados de *Pull* e *Push* foram propostos para monitoramento dos processos [Felber et al. 1999]. Resumidamente, no algoritmo *Push* os processos monitorados enviam mensagens de *heartbeat* periodicamente ao processo monitor, ao passo que, no algoritmo *Pull* o processo monitor é responsável por, periodicamente, consultar os estados dos processos monitorados. O serviço para detecção de falhas proposto neste trabalho implementa o algoritmo *Pull* e será detalhado na Seção 4.4.

### 4.2. Detectores de Falhas como uma NFV

Ao longo dos anos, diversas propostas de algoritmos de detecção de falhas surgiram. Consequentemente, suas funcionalidades foram sendo melhoradas em vários aspectos, dentre os quais pode-se destacar a sua utilização como serviço independente da aplicação [Felber et al. 1998]. A utilização dos detectores de falhas como serviços vão desde sua implementação como *middleware* [Zia et al. 2009], até serviços mais próximo do ambiente de rede como a implementação via protocolo SNMP (*Simple Management Network Protocol*) [Moraes and Duarte Jr. 2011].

Neste contexto, a proposta do presente trabalho é dispor o serviço para detecção de falhas via uma função de rede virtualizada. A virtualização do serviço para detecção de falhas é denominada de NFV-FD. A NFV-FD é implementada em uma rede OpenFlow [Openflow 2014], onde toda a lógica para a troca de informações dos dispositivos na rede é realizada por regras instaladas por um controlador. Para realizar a detecção dos estados dos processos nesta tipo de rede, a NFV-FD se comunica com o controlador e extrai informações relevantes para o processo de monitoramento. A arquitetura apresentada na próxima seção detalha a integração destes componentes no ambiente.

### 4.3. Arquitetura

A arquitetura apresentada na Figura 2 é uma integração do mecanismo para monitoramento dos processos proposto neste trabalho, com o controlador OpenFlow (na Figura 2: **Controller**). O controlador é composto por módulos (**Module Applications**) que possibilitam a integração de novos parâmetros de funcionalidades ao ambiente em execução. O **Controller** separa as funções pertinentes ao controle dos dispositivos na rede OpenFlow, como a criação de regras de comunicação, gerenciamento de topologia e de dispositivos, interface para acesso via Web, entre outras. O controlador possui ainda uma API para a interface **REST**[3], essa interface facilita a configuração em rotinas internas aos módulos.

Inicialmente, para a integração do controlador com a NFV-FD, um módulo denominado de FDMod é definido. O FDMod trabalha como um filtro de pacotes analisando informações do cabeçalho. Sua função é auxiliar na obtenção dos atributos dos processos que serão monitorados pela NFV-FD. A obtenção destes atributos é realizada com base em regras pré-definidas. As informações são extraídas do cabeçalho do pacote; um exemplo de atributos é o endereço IP e porta do processo a ser monitorado.

A NFV-FD sempre que recebe informações de atributos pertencentes a um novo processo, inicia o mecanismo de monitoramento do respectivo processo. O recebimento das informações dos atributos dos processos é feito com base em regras que podem ser instaladas por uma **Network Application**[4]. Uma regra deve descrever as características de comunicação da aplicação distribuída. Regras podem ser criadas de acordo com o protocolo e porta de comunicação da aplicação distribuída, como exemplo, considere as informações de um pacote capturado pelo FDMod:

*[in_port=1,dl_dst=01:00:5e:00:00:01,dl_src=00:00:00:00:00:01,nw_dst=230.0.0.1,nw_src=10.0.0.1,nw_proto=17,nw_tos=0,tp_dst=4446,tp_src=4446]*

Uma regra é criada com base nos seguintes campos: o endereço de destino (*nw_dst=230.0.0.1*), o protocolo UDP (*nw_proto=17*) e um endereço da camada de transporte (*tp_dst=4446*). O FDMod encaminha o pacote para a NFV-FD executar sua tarefa, por exemplo iniciar o monitoramento do seguinte processo: *nw_src=10.0.0.1* porta *tp_dst=4446*.

Em outra palavras, todas as mensagens recebidas pela **API OpenFlow** (ilustrada na arquitetura) são repassadas ao FDMod. O FDMod filtra estes pacotes para repassar à NFV-FD somente os pacotes que se encaixarem nas regras, ou ainda, mensagens de eventos notificados pelos *switches* que são descritos na seção 4.5.

[3] *REpresentational State Transfer*

[4] Aplicação que utiliza a interface REST para configurar as regras

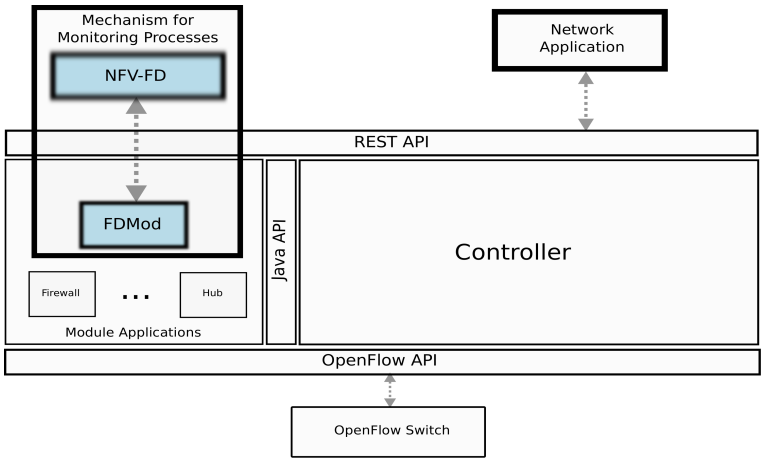



**Figura 2. Integração da NFV-FD na Arquitetura do Controlador OpenFlow**

### 4.4. Mecanismo para Detecção de Falhas em Processos

O monitoramento dos processos realizado pela NFV-FD ocorre através de requisição de mensagens de monitoramento (*liveness request*), que são periodicamente enviadas aos processos monitorados. O período de requisição é indicado pela variável $\Delta_i$. A cada intervalo de tempo $\Delta_i$, a NFV-FD envia uma mensagem `AreYouAlive?` e aguarda um período de tempo por uma resposta do tipo `YesIAm!`. Se a mensagem chegar antes do intervalo de *timeout* ($\Delta_{to}$) ter expirado, o estado para o respectivo processo é atualizado para *T*. Entretanto, se nenhuma resposta for recebida ou se a mensagem recebida não condizer com a respectiva requisição esperada, o processo monitorado terá seu estado atualizado para *S*.

Determinar um valor adequado para o *timeout* não é uma tarefa trivial, pois o tempo de comunicação pode variar de acordo com influências de atrasos de processamento ou sobrecarga nos canais de comunicação. Escolher um *timeout* adaptativo é uma alternativa, pois ele é baseado em estimativas que visam corrigir seu valor de acordo com o comportamento do ambiente ou a necessidade da aplicação. Nesse trabalho é utilizada uma estratégia adaptativa que faz estimativas para o tempo de chegada da próxima mensagem ($EA$) acrescida de uma margem de segurança ($\alpha$), como proposto em [Bertier et al. 2003]. $EA$ é derivada de uma média dos últimos $n$ dados de chegada. O próximo instante de chegada ($EA_{k+1}$) pode ser estimado da seguinte forma:

$$EA_{k+1} \approx \frac{1}{n}\left(\sum_{i=1}^{n} A_{(i)} - \Delta_{is_i}\right) + (k+1)\Delta_i \quad (1)$$

Na expressão (1) $s_1$, $s_2$,...,$s_n$ é o número de sequência das mensagens e $A_1$, $A_2$,...,$A_n$ os instantes para as respostas das mensagens de *liveness request* recebidas pela NFV-FD. O próximo *timeout* expira em:

$$\tau_{(k+1)} = EA_{(k+1)} + \alpha_{(k+1)} \quad (2)$$

Na expressão (2), $\alpha_{(k+1)}$ é uma margem de segurança que apresenta um comportamento probabilístico. Ela é adaptável a cada mensagem recebida de acordo com a carga na rede. A NFV-FD implementa a margem de segurança utilizando o mesmo cálculo do protocolo TCP (*Transmission Control Protocol*) e proposto em [Jacobson 1988]. A expressão apresentada a seguir possibilita corrigir o *timeout* de acordo com o comportamento do sistema:

$$\begin{aligned} Diferenca &= Tempo\_msg_k - Tempo\_msg_{k-1} \\ Media &= \phi * Media + (1 - \phi) * Diferenca \\ Desvio &= \phi * Desvio + (1 - \phi) * |Media - Diferenca| \\ \alpha_{k+1} &= Media + \beta * Desvio \end{aligned} \tag{3}$$

De acordo com a expressão (3), é possível observar que o cálculo de $\alpha$ está relacionado à periodicidade das mensagens de *liveness request* transmitidas com sua respectiva resposta ($Diferenca$), bem como às variações de tempo durante o tráfego das mensagens entre a NFV-FD e os processos monitorados ($Media$ e $Desvio$). Com o cálculo da expressão (2) tenta-se respeitar os limites entre os parâmetros de periodicidade e de *timeout*. Em geral, os valores utilizados para as variáveis $\phi$ e $\beta$, são próximos dos propostos por Jacobson, sendo 0.9 para $\phi$ e 4 para $\beta$.

### 4.5. Mecanismo de Detecção de Falhas em Enlaces de Comunicação

Entre as diversas vantagens em utilizar a NFV-FD em uma rede OpenFlow, pode-se destacar a possibilidade de detectar falhas em enlaces de comunicação. Para executar esta função, a NFV-FD faz uso de um mecanismo de descoberta de topologia que é executado pelos próprios *switches* OpenFlow.

O mecanismo de descoberta de topologia é realizado por um protocolo denominado de *Link Layer Discovery Protocol* (LLDP) [IEE 2009]. O protocolo OpenFlow define este mecanismo de descoberta através de pacotes do tipo *packet in* e *packet out*. Pacotes do tipo *packet out* são transmitidos entre *switches* OpenFlow periodicamente através do protocolo LLDP. Quando um *switch* OpenFlow recebe um pacote *packet out*, pacotes LLDP são transmitidos para todas as portas de saída. Quando o correspondente *switch* OpenFlow recebe o pacote LLDP, ele envia um *packet in* para o controlador comunicando o *status* de um enlace [Sharma et al. 2011].

De acordo com [Openflow 2014] há três situações para o controlador ter o *status* de um enlace:

- **Link up (Switch –> Controller)**: Quando um *switch* OpenFlow detecta um novo *switch*, ele encaminha uma mensagem para o controlador comunicando a nova conexão.
- **Link down (Switch –> Controller)**: Quando um *switch* OpenFlow fica sem receber mensagens LLDP de outro *switch* por um período de tempo (*timeout*), ele encaminha uma nova mensagem para o controlador comunicando a perda do enlace.
- **Get Links (Controller –> Switch)**: O controlador pergunta para o *switch* sobre o estado dos enlaces, é mais comum ocorrer esta situação quando um controlador se conecta pela primeira vez ao *switch*.

Para repassar os eventos detectados pelos *switches* para a NFV-FD, o FDMod tem acesso as informações da topologia da rede. O FDMod busca estas informações através do controlador OpenFlow. O controlador recebe informações dos eventos listados anteriormente, através de uma função denominada de *Link Discovery*. Por exemplo, quando ocorrem eventos do tipo **Link up** ou **Link down** o controlador recebe a mensagem e repassa para o FDMod.

O FDMod por sua vez, repassa informações sobre o evento ocorrido para a NFV-FD. Por exemplo, *'port s2-eth2 changed: DOWN'* indica que a porta 2 do *switch* 2 está sem conexão. Se houver processos que estão sendo monitorados nesta porta, do referido *switch*, a NFV-FD rotula o estado do processo como *U*, indicando que o processo está inacessível.

Vale ressaltar que a utilização de uma rede SDN possibilita a integração de protocolos ou dispositivos que, em geral, necessitam de interfaces específicas de comunicação. Exemplo disso é a comunicação com o protocolo LLDP. Neste contexto, acredita-se que outros benefícios podem ser explorados como a utilização de outras estratégias de detecção de falhas em enlaces mais efetivas. Um exemplo é o protocolo *Bidirectional Forwarding Detection* (BFD) que em [van Adrichem et al. 2014] foi estendido para implementar *fast recovery* em redes SDN.

## 5. Avaliação da NFV-FD

Para validar a proposta apresentada neste trabalho foi implementado um algoritmo de difusão confiável baseado no serviço de detecção de falhas. O algoritmo implementado é descrito na seção 5.1. Na seção 5.2 a utilização dos recursos computacionais e o desempenho dos serviços oferecidos pela NFV-FD, são avaliados.

### 5.1. Algoritmo de Difusão Confiável

Uma definição formal de difusão confiável faz uso das seguintes primitivas: *broadcast(m)* e *deliver(m)*, onde $m$ é uma mensagem. A primitiva *broadcast(m)* significa que $m$ será transmitida para todos os processos pertencentes ao conjunto de processos $\Pi$ (ver Seção 3), a primitiva *deliver(m)* significa que a mensagem $m$ pode ser entregue a este conjunto de processos. Estas primitivas são apresentadas no Algoritmo 1, os detalhes de seu funcionamento são descritos na sequência.

O Algoritmo 1 está dividido por 4 eventos distintos. Primeiramente, o algoritmo executa o evento *_Init* inicializando as seguintes variáveis: *delivered* (registro das mensagens já entregues), *correct* (estados dos processos atualizados pela NFV-FD) e *from[$p_i$]* (armazena cópia da mensagem original enviada pelo emissor). Quando uma mensagem é transmitida por difusão confiável, o evento *_Broadcast* é invocado. O evento *_Deliver* é executado para entregar uma mensagem $m$ transmitida por algum processo do conjunto de processos $\Pi$. A mensagem $m$ é entregue e o estado do processo emissor ($p_i$) é verificado. Se $p_i \notin correct$, uma nova difusão confiável é inicializada, sendo encaminhanda a mensagem original já gravada em *from[$p_i$]*. Finalmente, o evento *_Crash* é invocado quando um processo $p_i$ está falho e a mensagem $m$, transmitida por $p_i$, ainda não foi entregue. Uma cópia de $m$ é novamente transmitida por difusão confiável para futura entrega.

Em síntese, dois eventos que são dependentes da NFV-FD, forçam um processo a retransmitir uma mensagem:

**Algorithm 1:** Algoritmo para Difusão Confiável[Guerraoui and Rodrigues 2006]

```
upon event ⟨ _Init ⟩ do
  delivered := ∅;
  correct := Π;
  forall p_i ∈ Π do
    from[p_i] := ∅;

upon event ⟨ _Broadcast | m⟩ do
  trigger ⟨ broadcast | [Data, self, m] ⟩

upon event ⟨ _Deliver | p_i, [Data, s_m, m]⟩ do
  if (m ∉ delivered) then
      delivered := delivered ∪ {m}
      trigger ⟨ deliver | s_m, m ⟩;
      from[p_i] := from[p_i] ∪ {(s_m, m)}
          /* (Caso 2:  redistribui m) */
      if (p_i ∉ correct) then
          trigger ⟨ broadcast | [Data, s_m, m] ⟩;

upon event ⟨ _Crash | p_i ⟩ do
  correct := correct ∪ \ {p_i}
      /* (Caso 1:  redistribui m) */
  forall (s_m, m) ∈ from[p_i] do
      trigger ⟨ broadcast | [Data, s_m, m] ⟩;
```

- Caso 1: quando um processo percebe a falha do emissor antes de entregar $m$.
- Caso 2: quando um processo entrega $m$ e percebe que o emissor falhou.

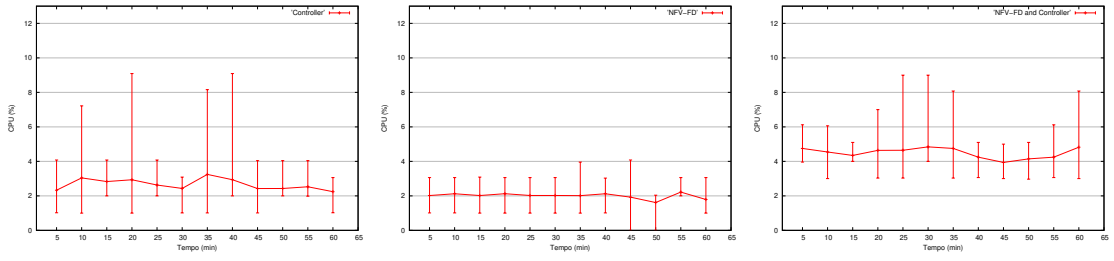
## 5.2. Experimentos

O ambiente para execução dos experimentos é implementado em uma rede OpenFlow com o controlador Floodlight [Floodlight 2014] sendo virtualizado através da ferramenta Mininet. O ambiente é hospedado em uma máquina física com as seguintes características: processador Intel Core i5 CPU 2.50GHz com 4 núcleos e sistema operacional Mint 17 com kernel 3.13.0-24. Para os experimentos da Figura 3, a rede é virtualizada com uma topologia simples utilizando 1 *switch*, 1 controlador remoto e 4 hosts executando o algoritmo de difusão confiável. A NFV-FD é configurada para monitorar os hosts com periodicidade de $\Delta_i$=1000ms. Os hosts que executam a difusão confiável são utilizados para gerar fluxo na rede e, portanto, também configurados para transmitirem mensagens a cada 1000ms.

Segundo [Batalle et al. 2013] uma implementação integrada ao controlador traz benefícios, pois assegura completa visão da rede. Neste sentido, o experimento da Figura 3 tem por objetivo avaliar o impacto que a NFV causa quando hospedada no controlador. Este experimento verifica a utilização da CPU mostrando medidas para o controlador sem a NFV-FD, a NVF-FD executada separadamente e, por fim, a NFV-FD e o controlador executados na mesma CPU. Cada experimento é executado no período de uma hora. Os dados apresentados nos gráficos da Figura 3 são dados médios de 5 execuções, sendo também apresentados valores mínimos e máximos observados.

No gráfico 3(a) a média de utilização da CPU é de 2,67%, este valor representa basicamente o processamento para o gerenciamento dos pacotes transmitidos por difusão confiável pelos 4 hosts. O custo de execução da NFV-FD separada é apresentado no

gráfico 3(b) e demonstrou uma utilização média de 2%. Finalmente, o resultado que buscávamos para os nossos experimentos, a utilização da NFV-FD no controlador. Este experimento é apresentado no gráfico 3(c). Neste experimento a utilização média da CPU é de 4,49%, praticamente dobrando a carga no controlador.



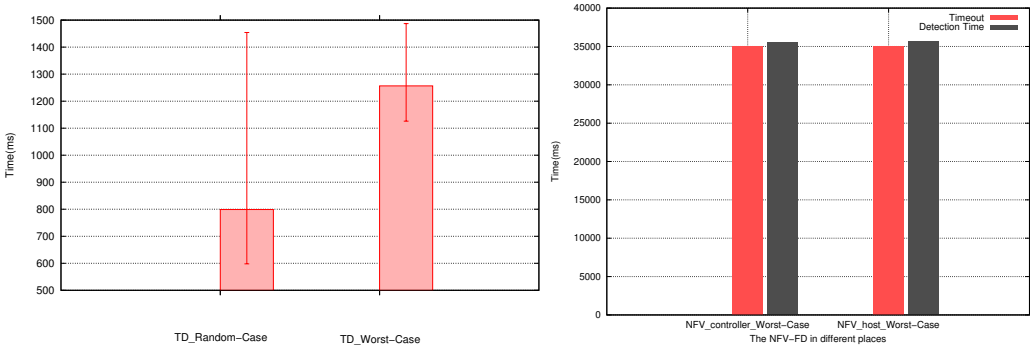

(a) Uso da CPU, Somente Controlador

(b) Uso da CPU, Somente NFV-FD

(c) Uso da CPU, NFV-FD no Controlador

**Figura 3. Impacto da NFV-FD Quanto ao Uso da CPU**

Para os experimentos da Figura 4, a rede foi virtualizada com uma topologia contendo 3 *switches*, 1 controlador remoto e 4 hosts executando o algoritmo de difusão confiável.




(a) Falha no Host: TD Computado Até Todos os Hosts da Difusão Confiável Serem Notificados

(b) Falha no Enlace: TD Computado Até a NFV-FD Ser Notificada

**Figura 4. Análise da Qualidade do Serviço para o Tempo de Detecção**

A Figura 4 apresenta o tempo de detecção ($T_D$) considerando falhas ocorridas em host e em enlace de comunicação. Para todos os casos, o tempo apresentado para detecção são valores médios, onde 10 falhas são forçadas no ambiente. A falha é implementada pelo bloqueio das mensagens de vida[5] forçando o estouro do *timeout*. O pior caso (*Worst-Case*) para o $T_D$ é computado considerando-se que a falha ocorre concomitantemente com a resposta de uma requisição de vida.

[5]Mensagens do protocolo LLDP (falha em enlace) e por mensagens da NFV-FD (falha em host)

Para o gráfico da figura 4(a) o tempo de detecção é computado no início da falha de um host até todos os demais hosts da rede (3 hosts) serem notificados da falha. A notificação é verificada somente quando o algoritmo de difusão confiável é informado. Considerando o pior caso, o tempo de detecção médio é de 1256ms, enquanto que o máximo observado é de 1454,23ms. Tendo em vista que o *timeout* possui um cômputo adaptativo, o valor máximo de $T_D$ pode ser causado por atrasos nas mensagens de *liveness request* ou pelo próprio processamento necessário para executar o algoritmo de difusão confiável.

Para o caso em que as falhas ocorrem em instantes aleatórios (*Random-Case*), é possível observar que há uma grande variação no $T_D$, onde os valores médios se concentram aproximadamente em 800ms. O valor mínimo observado neste experimento é de 598ms. Este melhor resultado ocorreu, provavelmente, no caso em que a falha foi executada instantes antes do $\Delta_{to}$ expirar, aliado a isso, ocorreu a notificação dos demais hosts. Considerando a detecção e notificação de uma falha por um único host, o melhor tempo observado pelo algoritmo de difusão confiável foi de 22.68ms. Ressalta-se que este experimento foi beneficiado pela utilização de uma rede SDN, pois a implementação deste tipo de cenário em um ambiente não virtualizado, não é uma tarefa trivial, uma vez que exige-se uma forte sincronização dos relógios.

Para o gráfico da figura 4(b), o objetivo é avaliar o tempo de detecção de uma falha no enlace de comunicação. Este evento ocorre através da notificação realizada pelo protocolo LLDP para a NFV-FD. Neste sentido, avalia-se o tempo de detecção com a NFV-FD em diferentes locais: no controlador e no host. Pode-se observar que o tempo de detecção do gráfico 4(b) é elevado se comparado ao gráfico da figura 4(a). Isso se deve ao fato de que manteve-se o valor padrão para a variável `LINK_TIMEOUT` sendo de 35 segundos no controlador *Floodlight*. Verificamos que o tempo de detecção é ligeiramente menor quando a NFV-FD está junto ao controlador, se comparado com a NFV-FD localizada no host. A diferença média observada foi de 120ms, o que corresponde ao tempo de comunicação entre o controlador e o host. Por outro lado, vimos no experimento anterior que este tempo superior no tempo de detecção, pode ser compensado com a redução no uso dos recursos computacionais do controlador.

Ressalta-se que o experimento para a detecção de falhas em enlaces reportado pelo protocolo LLDP, foi avaliado para demonstrar a sua viabilidade e os possíveis benefícios no uso de redes SDN. Uma vez que, estamos cientes de que a redução no tempo de detecção pode ser obtida diminuindo o valor padrão do `LINK_TIMEOUT`.

## 6. Conclusão

Neste trabalho foi apresentada uma NFV para detecção de falhas de processos e enlaces de uma rede OpenFlow denominada NFV-FD. Para avaliar a NFV-FD, um algoritmo para difusão confiável foi implementado e utilizado nos experimentos. O tempo de detecção de falhas e o uso dos recursos computacionais foram avaliados. Os experimentos mostraram que a implementação de uma NFV junto ao controlador causa um impacto relevante no uso do processador, sendo portanto uma boa estratégia de projeto concentrá-la fora do controlador, mesmo com o ligeiro aumento no tempo de detecção de falhas demonstrado nos experimentos. Como trabalhos futuros ainda pretende-se aperfeiçoar funcionalidades da NFV proposta neste trabalho, como exemplo, tornar o valor do *timeout* mais adequado

a variações no ambiente, como também investigar a utilização de outros protocolos para detecção de falhas em enlaces.

## Referências


(2009). IEEE standard for local and metropolitan area networks– station and media access control connectivity discovery. *IEEE Std 802.1AB-2009 (Revision of IEEE Std 802.1AB-2005)*, pages 1–204.

Batalle, J., Ferrer Riera, J., Escalona, E., and Garcia-Espin, J. (2013). On the implementation of nfv over an openflow infrastructure: Routing function virtualization. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–6.

Bertier, M., Marin, O., and Sens, P. (2003). Performance analysis of a hierarchical failure detector. In *DSN*.

Bondan, L., Dos Santos, C., and Zambenedetti Granville, L. (2014). Management requirements for clickOS-based network function virtualization. In *Network and Service Management (CNSM), 2014 10th International Conference on*, pages 447–450.

Borran, F., Hutle, M., Santos, N., and Schiper, A. (2012). Quantitative analysis of consensus algorithms. *IEEE Trans. Dependable Sec. Comput.*, 9(2).

Cerrato, I., Marchetto, G., Risso, F., Sisto, R., and Virgilio, M. (2014). An efficient data exchange algorithm for chained network functions. In *High Performance Switching and Routing (HPSR), 2014 IEEE 15th International Conference on*, pages 98–105.

Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2).

Charron-Bost, B., Pedone, F., and Schiper, A. (2010). *Replication: Theory and Practice*. Springer.

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

ETSI (disponível em http://www.etsi.org/technologies-clusters/technologies/nfv, acessado em 16 mar. 2015). Etsi gs nfv 002: "architectural framework".

Felber, P., Défago, X., Guerraoui, R., and Oser, P. (1999). Failure detectors as first class objects. In *DOA*.

Felber, P., Guerraoui, R., and Schiper, A. (1998). The implementation of a CORBA object group service. *Theory and Practice of Object Systems*, 4(2):93–105.

Ferrer Riera, J., Escalona, E., Batalle, J., Grasa, E., and Garcia-Espin, J. (2014). Virtual network function scheduling: Concept and challenges. In *Smart Communications in Network Technologies (SaCoNeT), 2014 International Conference on*, pages 1–5.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2).

Floodlight (acessado em 8 out. 2014). http://www.projectfloodlight.org/.

Fukushima, M., Yoshida, Y., Tagami, A., Yamamoto, S., and Nakao, A. (2014). Toy block networking: Easily deploying diverse network functions in programmable networks. In *Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International*, pages 61–66.

Guerraoui, R. and Rodrigues, L. (2006). *Introduction to Reliable Distributed Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Haleplidis, E., Denazis, S., Koufopavlou, O., Lopez, D., Joachimpillai, D., Martin, J., Salim, J., and Pentikousis, K. (2014). Forces applicability to sdn-enhanced nfv. In *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, pages 43–48.

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

Jacobson, V. (1988). Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM 88.

Moraes, D. M. and Duarte Jr., E. P. (2011). A failure detection service for internet-based multi-as distributed systems. In *ICPADS*. IEEE.

Openflow (acessado em 19 nov. 2014). http://archive.openflow.org/wk/index.php/Openflow_1.X_Discussion.

Sharma, S., Staessens, D., Colle, D., Pickavet, M., and Demeester, P. (2011). Enabling fast failure recovery in openflow networks. In *Design of Reliable Communication Networks (DRCN), 2011 8th International Workshop on the*, pages 164–171.

Sherry, J., Hasan, S., Scott, C., Krishnamurthy, A., Ratnasamy, S., and Sekar, V. (2012). Making middleboxes someone else's problem: Network processing as a cloud service. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 13–24. ACM.

van Adrichem, N. L. M., van Asten, B. J., and Kuipers, F. A. (2014). Fast recovery in software-defined networks. In *Third European Workshop on Software Defined Networks, EWSDN 2014, Budapest, Hungary, September 1-3, 2014*, pages 61–66.

Vilalta, R., Munoz, R., Mayoral, A., Casellas, R., Martinez, R., Lopez, V., and Lopez, D. (2015). Transport network function virtualization. *Lightwave Technology, Journal of*, 33(8):1557–1564.

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

Zia, H. A., Sridhar, N., and Sastry, S. (2009). Failure detectors for wireless sensor-actuator systems. *Ad Hoc Networks*, 7(5):1001 – 1013.