Uma Função Virtualizada de Rede para a Sincronização Consistente do Plano de Controle em Redes SDN

Giovanni V. Souza[1], Rogério C. Turchetti[12], Edson T. Camargo[13], Elias P. Duarte Jr.[1]

[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

[3]Universidade Tecnológica Federal do Paraná (UTFPR) – Toledo – PR – Brasil

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

edson@utfpr.edu.br, elias@inf.ufpr.br

Abstract. *Switches of Software Defined Networks (SDN) are programmed by a centralized entity which is the controller. Centralized control has limited availability and scalability by definition. Distributed control strategies solve this problems, but achieving consistent synchronization across multiple SDN controllers is not a trivial endeavor. In this work we propose a VNF-Consensus, a VNF (Virtual Network Function) that implements the Paxos consensus algorithm to ensure consistency among controllers of a distributed control plane. Using this approach, controllers are decoupled from synchronization tasks and can perform their control plane activities without having to execute expensive synchronization tasks. Experimental results show the benefits of the proposed solution, in particular in terms of the optimization of resource usage and a reduction of the controller workload.*

Resumo. *A programação dos switches SDN (Software Defined Networks) nas Redes Definidas por Software é realizada por um controlador. Esse controle é centralizado, portanto tendo restrições em termos de disponibilidade e de escalabilidade. Para resolver esse problema pode-se utilizar estratégias que distribuem o plano de controle. Entretanto, implementar a sincronização entre os múltiplos controladores distribuídos não é uma tarefa trivial. O presente trabalho propõe uma solução para a sincronização consistente do plano de controle em redes SDN através da implementação de uma função virtualizada de rede (VNF – Virtual Network Function) denominada de VNF-Consensus. VNF-Consensus implementa o algoritmo Paxos e com sua utilização os controladores ficam desacoplados e podem executar em paralelo suas atividades no plano de controle. A implementação e os resultados experimentais mostram os diversos benefícios obtidos pelo uso da solução proposta, em especial a otimização no uso dos recursos computacionais e a redução na carga dos controladores.*

1. Introdução

Redes Definidas por Software (SDN – *Software Defined Networks*) têm melhorado a flexibilidade e facilitado o gerenciamento das redes de computadores. A estratégia adotada

pelas arquiteturas SDN é separar o plano de controle do plano de dados. O controle é, em geral, centralizado e responsável por tomar decisões que ocorrem no plano de dados. Nesta abordagem centralizada, o estado da rede é determinado por um controlador [Ho et al. 2016]. Considerando aspectos de gerenciamento, controle e visão global da topologia da rede, esta abordagem centralizada é, sem dúvidas, atraente. Por outro lado, a implementação do plano de controle centralizado tem, naturalmente, restrições em termos de disponibilidade, desempenho e de escalabilidade [Canini et al. 2015].

Neste sentido, há um consenso de que o plano de controle precisa ser distribuído [Schiff et al. 2016]. Além disso, para que a falha de um controlador possa ser mascarada é necessário aplicar técnicas de redundância no plano de controle [Koponen et al. 2010, Berde et al. 2014]. Entretanto, a sincronização de múltiplos controladores distribuídos não é uma tarefa trivial. Considere, por exemplo, o problema da instalação consistente de novas regras de encaminhamento de pacotes. Se regras conflitantes forem aplicadas estipulando rotas distintas haverá uma patologia na rede podendo, por exemplo, resultar em *loops* indesejados ou rotas contornando serviços de interesse.

Segundo [Canini et al. 2015] um dos principais problemas em aberto no contexto das redes SDN é justamente a necessidade de tornar o plano de controle robusto, sem interferir no desempenho global das funções oferecidas pela rede e, obviamente, preservar a correta operação do plano de dados. Entretanto, as soluções existentes para a construção de um plano de controle robusto, em geral, produzem ambientes em que deixam a cargo do controlador a função de coordenar as ações realizadas no plano de controle. Em outras palavras, são hospedados nos próprios controladores serviços que permitem *sincronizar* de maneira consistente as informações gerenciadas pelos vários controladores presentes na rede [Koponen et al. 2010, Canini et al. 2015, Ho et al. 2016]. Por outro lado, há soluções que procuram aliviar a carga dos controladores usando os próprios *switches* para sincronizar o plano de dados [Schiff et al. 2016, Dang et al. 2015]. Entretanto, estas são soluções que, em geral, implicam em alterações do funcionamento padrão, como exemplo, mudanças no protocolo *OpenFlow*.

Portanto, acredita-se que aumentar as múltiplas tarefas que os controladores já exercem na rede, não é a estratégia mais adequada, uma vez que a inclusão dessas funcionalidades extras implica no aumento da carga de trabalho. Além disso, segundo os autores em [Karakus and Durresi 2017] problemas quanto ao desempenho do plano de controle ainda é um assunto que precisa ser investigado tanto pela comunidade acadêmica quanto pela indústria. Sendo assim, o presente trabalho propõe uma solução para a sincronização do plano de controle em redes SDN, onde os controladores são coordenados por uma função virtualizada de rede (VNF – *Virtual Network Function*). A VNF proposta é denominada de *VNF-Consensus* e implementa o algoritmo de consenso Paxos [Lamport 1998] no ambiente de rede. Dessa forma, a *VNF-Consensus* consegue manter um plano de controle consistente pois sincroniza as ações entre todos os controladores SDN. Conceitualmente, uma VNF é a representação de dispositivos de redes em entidades virtuais que são executadas utilizando tecnologias de virtualização baseadas em software [ETSI 2016].

Para obter uma visão comum do estado da rede, cada controlador possui acesso a uma instância da *VNF-Consensus* através da qual poderá receber decisões e enviar dados para serem sincronizados. Dessa maneira, todas as decisões executadas pela *VNF-*

Consensus são sistematicamente executadas sem a atuação direta dos controladores. As vantagens dessa abordagem são: (i) os controladores ficam desacoplados e podem executar em paralelo suas atividades no plano de controle; (ii) o plano de controle é sincronizado sem aumentar a carga de trabalho dos controladores, pois a *VNF-Consensus* é executada como entidade externa aos controladores, não utilizando os mesmos recursos computacionais; (iii) como os controladores não participam diretamente nas decisões do Paxos, o consenso consegue garantir, independentemente do número de controladores operacionais, a conclusão de seus serviços; (iv) por fim, a solução proposta não exige nenhuma alteração no funcionamento padrão do protocolo *OpenFlow* ou nos *switches* SDN. Resultados experimentais mostram o desempenho da *VNF-Consensus* e seu impacto na utilização dos recursos, como também os benefícios obtidos por executar o serviço sem a participação direta dos controladores SDN.

O restante deste trabalho está organizado da seguinte forma. A seção 2 apresenta trabalhos relacionados que tratam da sincronização do plano de controle em redes SDN. Na seção 3 é apresentada a proposta focando no aspecto de implementação e no algoritmo de consenso. Os experimentos e as conclusões do trabalho são apresentados nas seções 4 e 5, respectivamente.

2. Sincronização dos Dados em Redes SDN

Nesta seção apresentamos os trabalhos relacionados que tratam da sincronização do plano de controle em redes SDN. A principal diferença desses trabalhos para a solução proposta é que no presente trabalho há um esforço em permitir a sincronização consistente no plano de controle através do uso de uma função virtualizada de rede.

Para garantir a consistência nas operações de rede, as ações executadas em diferentes controladores SDN precisam ser sincronizadas. Neste contexto, em [Schiff et al. 2016] os autores propõem um *framework* para a sincronização das informações no plano de dados implementadas dentro (*in-band*) dos *switches*. Segundo os autores, protocolos convencionais que executam suas funções fora dos *switches* possuem um alto custo para coordenação e sincronização das informações. Em contraste, soluções *in-band* permitem resolver problemas de acordo (*agreement*) através da troca de poucas mensagens. A solução é baseada no método 'Compare-And-Set' (CAS) abordagem utilizada para a consistência de memória. Esse método garante atomicidade nas transações evitando cenários inconsistentes. Em outras palavras, para evitar inconsistência nas regras instaladas nos *switches* pelos controladores, o comando *FlowMod* é modificado e usa *flags* para definir quais regras devem ser adicionadas ou removidas. Os autores apresentam uma prova de conceito na qual demonstram a eficiência da solução proposta. O mecanismo apresentado pelos autores é simples e pode ser implementado sem a necessidade de hardware ou protocolos extras. Por outro lado, para que os algoritmos propostos funcionem, há a necessidade de efetuar alterações nas *flags* do protocolo *OpenFlow*.

Em [Dang et al. 2015] os autores propõem mover a lógica do algoritmo de consenso Paxos para ser executado dentro da rede. Em particular, propõem que o algoritmo seja executado nos próprios *switches* SDN. Duas abordagens são propostas: a implementação do algoritmo Paxos na sua versão completa, isto é, sem otimizações, para ser executado nos *switches* SDN; e a execução de uma versão otimizada denominada de NetPaxos que evita a implementação de um coordenador (elemento do Paxos) entre os *switches*. Os

autores mostram que mover a lógica do consenso para dentro da rede reduz a complexidade das aplicações, reduz a latência das mensagens da aplicação e aumenta o *throughput* das transações. Entretanto, os *switches* precisam executar e assumir papéis que são atribuídos pelo algoritmo Paxos, essa abordagem dificulta sua execução pois exige mudanças no funcionamento padrão da rede, incluindo a alteração do *firmware* nos *switches* SDN.

Em [Ho et al. 2016] os autores propõem uma solução para a sincronização e consistência das informações que são gerenciadas pelos controladores em uma rede SDN. O objetivo é manter a consistência entre múltiplos controladores através da implementação de um algoritmo de consenso. Uma versão ao algoritmo Paxos denominada de FPC (*Fast Paxos-based Consensus*) é proposta. Segundo os autores, o algoritmo FPC reduz a complexidade se comparado ao algoritmo original proposto por Lamport (1998). Em especial, FPC não possui um líder pré-definido, ao invés disso o algoritmo permite que qualquer processo participante possa se tornar o líder (denominado de *chairman*). Uma vez que todos os processos (controladores) tenham realizado a atualização, o *chairman* retorna a seu papel de *listener* terminando a rodada do algoritmo. Os autores comparam o FPC com o algoritmo de consenso Raft [Ongaro and Ousterhout 2014] e concluem que o algoritmo proposto apresenta menor tempo para a execução do consenso.

Onix [Koponen et al. 2010] é uma plataforma que permite a implementação do plano de controle como um sistema distribuído mantendo uma visão global da rede. Onix é um sistema distribuído executado em um *cluster* de servidores físicos. No Onix um controlador armazena informações em uma estrutura de dados denominada de NIB (*Network Information Base*). A NIB representa a parte mais importante da plataforma, pois todo o estado da rede é sincronizado através de leituras e escritas controladas pela base de dados. Em especial, Onix representa um conjunto de APIs que fornecem escalabilidade e confiabilidade por replicar e distribuir as informações sincronizadas entre múltiplas instâncias na rede. Se uma informação for alterada, a mesma será propagada por todas as instâncias de NIBs garantindo a consistência através da coordenação de um algoritmo de consenso (Zookeeper [Hunt et al. 2010]). Dito pelos próprios autores, Onix não é uma inovação em si, pois derivam de diversos trabalhos propostos ao longo da história.

Em [Canini et al. 2015], os autores propõem um modelo formal para a comunicação entre o plano de dados e o plano de controle distribuído de uma rede SDN. Em particular, os autores desejam tratar o problema de inconsistência durante a atualização de regras que são instaladas em um ou mais *switches*. Estas regras são as políticas da rede e a solução deve, informalmente, garantir que um pacote que está trafegando na rede seja processado por exatamente uma única política, mesmo em situações em que ela (a política) eventualmente seja atualizada. Os autores apresentam um protocolo denominado de *ReuseTag* que usa uma abordagem de máquina de estados para implementar a ordem total das atualizações das políticas, a solução assume que o plano de dados é como se fosse uma estrutura de memória compartilhada onde os controladores são os processos que fazem leituras e escritas nessa memória. Dado um limite máximo de latência na rede e assumindo um número máximo de f processos/controladores falhos, o protocolo *ReuseTag* funciona corretamente. Por fim, são definidos ainda requisitos mínimos para a sincronização das ações entre os controladores demonstrando a complexidade dessa sincronização mesmo na presença de um algoritmo de consenso.

Tabela 1. Comparação dos trabalhos com relação a sincronização dos dados.

Trabalhos Relacionados	Algoritmo de Sincronização	Plano de Sincronização	Local de Sincronização dos Algoritmos
[Schiff et al. 2016]	Transações atômicas usando *compare-and-set* (CAS)	Plano de Dados	Switches SDN
[Dang et al. 2015]	Algoritmo de consenso *NetPaxos*	Plano de Dados	Switches SND
[Ho et al. 2016]	Algoritmo de consenso *Fast Paxos-based Consensus*	Plano de Controle	Controladores SND
[Koponen et al. 2010]	Algoritmo de consenso *Zookeeper*	Plano de Controle	Controladores SDN
[Canini et al. 2015]	Algoritmo *Policy Serialization* (PS) baseado em Rep. de Máquina de Estados	Plano de Controle	Controladores SDN
Plataforma ODL	Algoritmo de consenso Raft	Plano de Controle	Controladores SDN

O OpenDaylight[1] (ODL) é um controlador de rede SDN que oferece em sua arquitetura diversas funcionalidades, em especial a possibilidade de executar e de sincronizar múltiplos controladores SDN. Essa arquitetura é denominada de ODL *Clustering* e oferece um mecanismo de integração entre múltiplos processos e aplicações. Cada controlador possui seu próprio local de armazenamento e a replicação dos dados ocorre através de caches distribuídas aplicando o esquema Infinispan[2]. Toda a comunicação e notificação entre os controladores no ODL *Clustering* ocorre usando um grupo de ferramentas denominado de *Akka*. Essa comunicação ocorre somente entre os controladores da rede, as decisões e a consistência são garantidas pelo uso do protocolo de consenso Raft. O Raft é utilizado para coordenar as atualizações que são executadas entre os controladores SDN. Note que toda a sincronização das informações exige a execução de algoritmos que estão plugados diretamente em cada controlador. Como resultado há um esforço adicional executado pelos controladores a fim de obter a consistência das informações na rede.

A Tabela 1 apresenta os trabalhos relacionados nesta seção descrevendo os mecanismos de sincronização utilizados por cada solução. Vale ressaltar que a principal diferença no presente trabalho pode ser visualizada na coluna que especifica o local de sincronização. Note que a proposta neste trabalho executa a sincronização no plano de controle sem a participação direta das entidades da rede SDN, como exemplo os *switches* e os controladores. Mais detalhes da abordagem utilizada são apresentadas na próxima seção.

3. Um Plano de Controle Robusto Baseado em VNF

Nesta seção é apresentada uma caracterização do problema abordado: a consistência do plano de controle em redes SDN. Na sequência é detalhada a arquitetura e as particularidades de implementação, bem como, uma breve descrição do algoritmo de consenso Paxos e detalhes de como sua lógica é implementada e executada na *VNF-Consensus*.

3.1. Caracterização e Delimitação do Problema

Para garantir um plano de controle consistente em uma rede SDN, todos os eventos a serem atualizados precisam ser sincronizados entre os controladores envolvidos. Cada

[1]https://www.opendaylight.org/
[2]http://infinispan.org/

atualização gera um novo estado da rede. As mudanças de estados são causadas por eventos como, por exemplo: a criação de um novo fluxo de comunicação, a ocorrência de falhas em *links* ou *switches*, a instalação de regras para passar por serviços como filtros de pacotes ou novas rotas, entre outros. Em síntese, neste contexto é necessário utilizar algum mecanismo que sincronize cada evento evitando estados inconsistentes. Os autores de [Schiff et al. 2016] descrevem diversas patologias causadas na rede devido a esses estados inconsistentes.

As decisões realizadas no plano de controle implicam em mudanças no plano de dados, isto é, a criação de um novo fluxo de dados implicará na instalação de uma nova regra no *switch*. Portanto, é importante entender a comunicação entre os controladores e os *switches* SDN. Em [Tianzhu et al. 2016] os autores identificam dois tipos de comunicação (ver Figura 1): *Switch-to-Controller* e *Controller-to-Controller*.

A comunicação *Switch-to-Controller* suporta iterações entre os *switches* e o controlador usando o protocolo *OpenFlow*. Por exemplo, é nesse plano que um pacote *packet-in* (pacote gerado quando não há uma correspondência na tabela de fluxos do *switch*) é encaminhado do *switch* ao controlador. O controlador, por sua vez, retorna uma mensagem *flow-mod* para autorizar ou não a comunicação do pacote. Essas são mensagens utilizadas para gerenciar as tabelas de fluxos de cada *switch*. Considerando a execução de múltiplos controladores, a decisão tomada para a autorização do novo fluxo de pacotes deve ser sincronizada entre todos os controladores da rede. Caso contrário, estados inconsistentes no plano de controle poderão causar patologias na rede, como o descarte de pacotes, criação de *loops*, rotas contornando serviços específicos, entre outros. Além disso, a mensagem *flow-mod* precisa ser atualizada pelos *switches* no mesmo instante de tempo (no contexto do plano de dados), evitando estados inconsistentes. Entretanto, esta tarefa pode ser significativamente complicada em ambientes onde os atrasos não podem ser previstos [Zhou et al. 2014]. Portanto, vale ressaltar que neste trabalho abordamos a consistência no plano de controle, ao passo que a sincronização no plano de dados pode ocorrer em instantes diferentes de tempo. Em resumo, assumimos que nos estados transientes do plano de dados há a possibilidade de ocorrer períodos com estados inconsistentes da rede.

A comunicação *Controller-to-Controller* permite a sincronização do plano de controle. A consistência nesse plano exige que todos os controladores possuam a mesma visão do estado da rede. No plano de controle a consistência pode ser forte ou eventual [Tianzhu et al. 2016]. A consistência forte significa que as leituras dos dados em diferentes controladores produzem sempre o mesmo resultado, ao passo que, a consistência eventual significa que há períodos transientes em que as leituras em diferentes controladores podem produzir diferentes valores. No presente trabalho utilizamos o algoritmo de consenso Paxos que implementa a consistência forte.

3.2. VNF-Consensus

Em uma rede SDN todas as ações executadas por um conjunto de controladores precisam ser sincronizadas e coordenadas. A *VNF-Consensus* implementa um algoritmo de consenso que possibilita garantir a consistência do plano de controle em redes SDN. Em particular, a *VNF-Consensus* implementa, no ambiente de rede, o algoritmo de consenso Paxos descrito a seguir. Note que, embora sucinta, a descrição do algoritmo é importante para a compreensão da *VNF-Consensus* que será detalhada na sequência.

Paxos é um algoritmo de consenso tolerante a falhas proposto por Leslie Lamport [Lamport 1998]. Informalmente, um algoritmo de consenso [Santos et al. 2011] permite que um conjunto de processos entre em acordo sobre um determinado valor, sendo que inicialmente cada processo pode propor valores distintos. Dentre as propriedades que um algoritmo de consenso deve atender, destaca-se neste trabalho duas propriedades: segurança (*safety*) e progresso (*liveness*). A segurança garante que todos os processos sem falha decidem pelo mesmo valor. O progresso garante que o consenso termina com sucesso. No algoritmo Paxos os processos podem assumir três diferentes papéis denominados de: *proposers*, *acceptors* e *learners*.

Os *proposers* propõem um valor, os *acceptors* escolhem um valor e os *learners* aprendem o valor decidido. Um único processo pode assumir qualquer uma dessas funções e múltiplas funções simultaneamente. Paxos é ótimo em termos de resiliência [Lamport 2006]: para tolerar f falhas ele requer $2f + 1$ *acceptors* – isto é, para assegurar o progresso um quórum de $f + 1$ *acceptors* devem estar sem-falha. Para resolver o consenso, Paxos executa em rodadas. Tipicamente um *proposer* ou um *acceptor* atua como *coordenador* da rodada. O *coordenador* recebe propostas que tentam alcançar o consenso. Quando o quórum de *acceptors* aceita o mesmo número de rodada, o consenso termina. Neste momento os *learners* têm acesso ao valor decidido.

Paxos é utilizado neste trabalho para sincronizar os dados no plano de controle. Em especial, os controladores aprendem (*learners*) os valores decididos e fornecidos pela *VNF-Consensus*. Dessa maneira, o plano de controle mantém o estado da rede em todos os controladores. Paxos é baseado em um modelo de consistência forte. Portanto, a *VNF-Consensus* garante que os dados resultantes das leituras de diferentes controladores irão produzir sempre o mesmo resultado.

Para obter uma visão comum do estado da rede, cada controlador possui acesso a uma instância da *VNF-Consensus* através da qual poderá receber decisões e enviar dados para serem sincronizados. A Figura 1 apresenta a integração dos controladores SDN com a *VNF-Consensus*. Todas as decisões realizadas pela *VNF-Consensus* são sistematicamente executadas sem a atuação direta dos controladores. Assim, cada controlador é um cliente do algoritmo de consenso que irá aprender os resultados gerados pela *VNF-Consensus*.

Considere que o *host* x apresentado na Figura 1 deseja, pela primeira vez, fazer uma comunicação com outro dispositivo qualquer. O *host* x encaminha o pacote ao *switch OpenFlow* que, por sua vez, verifica se há regras deste destinatário em sua tabela de fluxos. Se nenhuma entrada existir na tabela de fluxos referente ao destinatário, o *switch* encaminha um *packet-in* ao controlador SDN. Antes do controlador decidir o que fazer com o pacote, ele encaminha uma cópia do mesmo para a *VNF-Consensus*. O consenso então é inicializado e, ao final, o resultado é encaminhado a todos os controladores SDN via comunicação REST (*REpresentational State Transfer*).

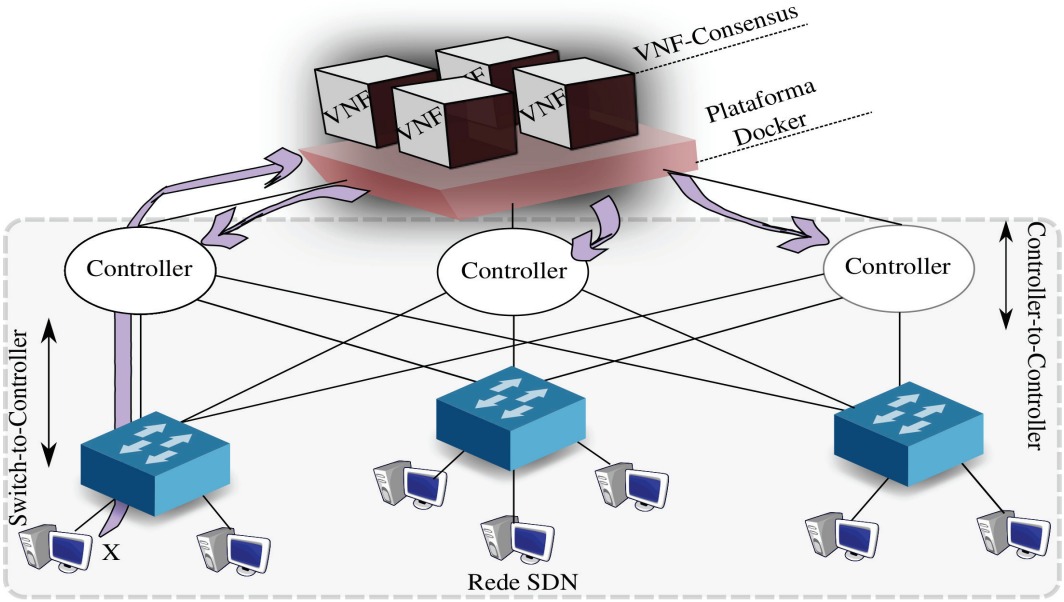


Figura 1. Integração da VNF na arquitetura do controlador SDN.

Observe que quando o controlador SDN recebe um novo fluxo, como no exemplo apresentado no parágrafo anterior, faz-se necessário executar um 'desvio' no fluxo dos pacotes com o objetivo de encaminhá-los à *VNF-Consensus*. Em outras palavras, cada pacote precisa ser classificado. A classificação permite definir qual pacote será encaminhado e para qual destino. Se for decidido que um pacote precisa ser encaminhado para a *VNF-Consensus*, então um fluxo de encaminhamento precisa ser criado.

Para executar essa tarefa é proposta uma ferramenta denominada de *Filter*. A *Filter* realiza a classificação e, quando necessário, o encaminhamento dos pacotes para a *VNF-Consensus*. O funcionamento e a integração desses componentes podem ser visualizados na arquitetura apresentada na Figura 2. Observe que as instâncias da *VNF-Consensus* são executadas em *containers* no ambiente Docker[3].

Note (na arquitetura apresentada na Figura 2), que a ferramenta *Filter* é composta por dois módulos denominados de *Classifier* e *Forwarder*. O *Classifier* possui acesso às regras que são armazenadas no repositório *Rules*. Essas regras permitem classificar os fluxos de entrada para determinar o destino de cada pacote. O processo ocorre de acordo com os passos mostrados na Figura 2 e detalhados a seguir: o controlador SDN encaminha todos os pacotes para a ferramenta *Filter* (passo 1), se houver alguma regra cadastrada no módulo *Classifier* que se encaixe ao pacote recebido (passo 2), o pacote é repassado para o módulo *Forwarder* (passo 4), caso contrário, o pacote será retornado ao controlador (passo 3). O *Forwarder*, por sua vez, é responsável por realizar o repasse dos pacotes, por exemplo, permitindo encaminhar o pacote à correspondente VNF (passo 5). Esse último passo também é identificado como SFP (*Service Function Path*)

[3]http://www.docker.org/

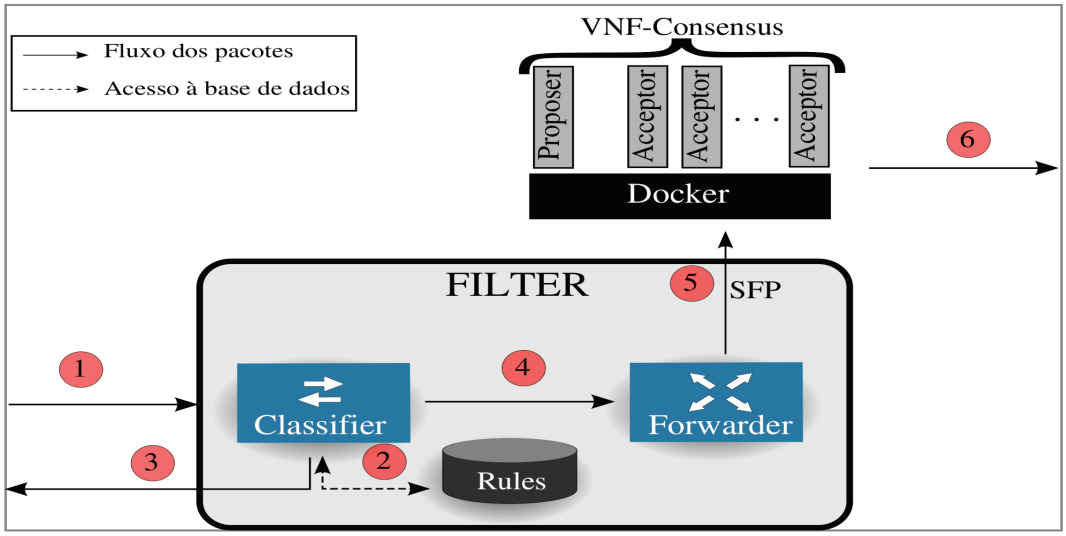


Figura 2. Arquitetura para a sincronização do plano de controle.

Quando um SFP é criado, o pacote contendo a regra a ser instalada é repassado para o coordenador da *VNF-Consensus*. Do ponto de vista do Paxos, neste cenário cada VNF executa funções distintas de acordo com as especificações do algoritmo. Isto é, na arquitetura apresentada na Figura 2 considere o *proposer* como sendo também o coordenador. É o coordenador quem recebe as regras vindas dos diversos controladores da rede. Ao receber uma regra, o coordenador então inicia uma nova instância de consenso. Uma instância é definida executando duas fases distintas do algoritmo Paxos. Na primeira fase, o coordenador seleciona um número único para a rodada e o envia em uma solicitação para os *acceptors*. Ao receber a solicitação com um número maior que qualquer outro recebido previamente para aquela instância, o *acceptor* responde ao coordenador prometendo que rejeitará qualquer solicitação futura com um número de rodada menor. Se o *acceptor* já aceitou uma regra para a instância, ele retorna essa regra ao coordenador junto com o número de rodada recebido quando a regra foi aceita. Quando o coordenador recebe respostas de um quórum de *acceptors*, a segunda fase do protocolo é inicializada.

Na segunda fase, o coordenador verifica os valores recebidos do quórum de *acceptors*. Se não houver *acceptors* que tenham aceitado uma regra, o coordenador executa a fase 2 com sua regra. Por outro lado, se algum *acceptor* retornou uma regra na primeira fase, o coordenador é obrigado a executar a segunda fase com a regra que possuir o maior número de rodada. Com a regra selecionada, o coordenador envia uma requisição aos *acceptors* com o número da rodada e a regra escolhida. Se os *acceptors* não prometeram participar de uma rodada com número maior, então eles respondem ao coordenador aceitando a regra proposta. Quando o coordenador recebe respostas de um quórum de *acceptors* há a garantia de que não há outro *proposer*, ou coordenador, com um número de proposta maior que a já aceita. A regra então é enviada aos *learners* e o consenso para aquela instância termina.

O coordenador envia o valor decidido aos *learners* que são controladores SDN que, dessa forma, aprendem o valor decidido pela *VNF-Consensus*.

Note que para cada papel definido no algoritmo Paxos um *container* é criado para

ser executado de forma independente entre os processos. Essa abordagem permite o isolamento total dos processos. A implementação em *containers* é motivada por sua baixa utilização dos recursos computacionais, rápida instanciação e fácil implementação de funções virtualizadas, possibilitando também uma alta densidade de processos virtualizados no mesmo *host* [Anderson et al. 2016]. A seguir são apresentados os experimentos que permitem comprovar os benefícios do serviço descrito nesta seção.

4. Avaliação da NFV-Consensus

Nesta seção são apresentados experimentos executados para avaliar o desempenho da *VNF-Consensus* em uma rede SDN. O ambiente para execução dos experimentos é implementado usando o protocolo *OpenFlow* e controladores *Ryu*[4]. O ambiente é hospedado em uma máquina física com as seguintes características: processador AMD FX-4300 CPU 3.8GHz com 4 núcleos e sistema operacional *Ubuntu* 16.04 com *kernel* 4.4.0-53. Para os experimentos apresentados nesta seção, a rede é virtualizada através da ferramenta *Mininet*[5].

A topologia consiste de 3 *switches*, 3 controladores remotos e 3 *hosts* que são utilizados para gerar pacotes para a sincronização do plano de controle. A *VNF-Consensus* é configurada com 1 *proposer*, 3 *acceptors* e 3 *learners*, estes últimos são os controladores SDN. Além disso, o fluxo de dados foi gerado a partir do *cbench*[6], uma ferramenta que tem como objetivo sobrecarregar o controlador através de múltiplos pacotes do tipo *packet-in*.

A seguir são apresentados três grupos de experimentos. No primeiro grupo o objetivo é avaliar e comparar o impacto causado para a sincronização do plano de controle. No segundo grupo comparamos o tempo para inicialização das instâncias do Paxos. Por fim, no terceiro grupo é avaliado e comparado o *throughput* para a execução do Paxos.

Custo para a sincronização do plano de controle

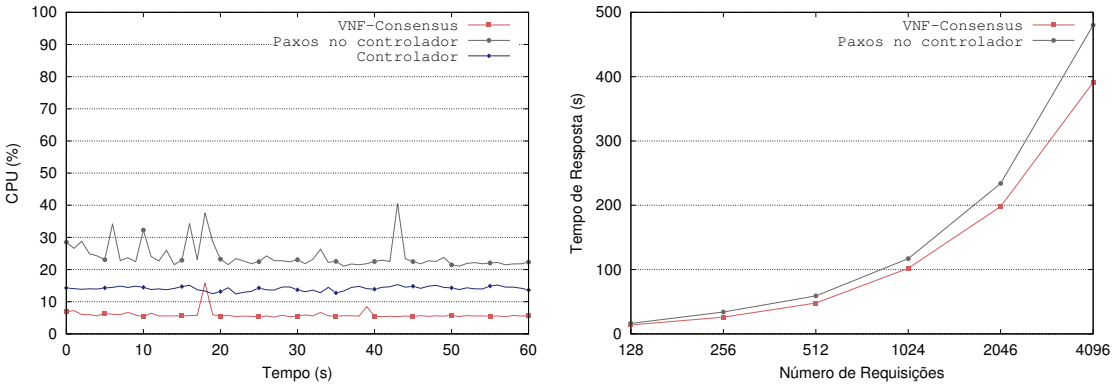
Quando a sincronização do plano de controle é deixada a cargo dos próprios controladores SDN, aumenta-se a quantidade de tarefas que os controladores precisam executar. Nestas condições, pode-se ter um impacto negativo no desempenho global da rede. Para medir tal impacto, a figura 3 apresenta os seguintes experimentos: a utilização de CPU e o tempo para um controlador instalar um conjunto de regras.

O objetivo do experimento apresentado na figura 3(a) é avaliar o custo para a sincronização do plano de controle em termos de utilização de CPU. Em particular, deseja-se medir a carga extra de trabalho causada pelas ações de sincronização realizadas pelos próprios controladores. Para este experimento, os controladores foram submetidos a um fluxo de dados gerado a partir dos *hosts* através da ferramenta *cbench*. Note que a *VNF-Consensus* apresenta em média 5,9% de utilização de CPU. Quando os controladores não sincronizam o plano de controle, a utilização média de CPU é de aproximadamente 14.1%. Porém, quando o Paxos é executado nos controladores há um impacto maior na utilização de CPU, atingindo aproximadamente 24,1%. Essa sobrecarga dos controladores é causada pelas ações de sincronização do plano de controle somada ao processamento

[4]https://osrg.github.io/ryu/

[5]http://mininet.org/

[6]https://github.com/mininet/oflops/tree/master/cbench



(a) Uso de CPU.

(b) Tempo necessário para instalar um conjunto de regras.

Figura 3. Comparação quanto ao impacto causado para a sincronização do plano de controle: execução dentro e fora dos controladores.

dos novos fluxos gerados na rede. Como resultado, há um impacto negativo no desempenho global da rede. Esse cenário pode ser melhor visualizado nos experimentos da figura 3(b).

No experimento da figura 3(b) o objetivo é verificar se os controladores apresentam algum impacto negativo no processamento normal da rede quando eles também precisam sincronizar o estado da rede. Para o experimento foi utilizado um *script* que cria 128 processos que produzem, de maneira contínua, requisições REST que instalam regras aleatórias no *switch*. Em paralelo, a ferramenta *cbench* envia um fluxo de dados fazendo com que o plano de controle seja constantemente sincronizado. Nestas condições, avaliamos o tempo necessário para instalar um conjunto de regras. Vale ressaltar que esse tempo é medido desde o início da requisição até o seu respectivo retorno, isto é, quando o *switch* instala a regra e retorna uma confirmação ao controlador.

Note que conforme aumenta-se o número de requisições, o tempo de resposta aumenta em ambos os casos. Entretanto, quando a sincronização é realizada pela *VNF-Consensus*, observa-se uma redução de até 18,5% no tempo para instalação de um conjunto de regras. Como mostrado na figura 3(b), essa diferença tende a crescer conforme o número de requisições aumenta. Claramente podemos notar os benefícios da sincronização do plano de controle via *VNF-Consensus*.

Tempo para inicializar uma instância do Paxos

Conforme visto na seção 3.2, o Paxos é ótimo em termos de resiliência, já que requer $2f + 1$ *acceptors* para tolerar f falhas. Para aumentar o grau de resiliência é necessário aumentar o número de participantes. Na abordagem onde os próprios controladores executam a sincronização através do Paxos, para aumentar o número de participantes é necessário aumentar o número de controladores. Essa abordagem impõe restrições, tendo em vista que a inclusão de novos controladores exige a inserção de uma infraestrutura subjacente (e.g., mecanismo de virtualização, armazenamento, etc.). Por outro lado, ao utilizar o Paxos como uma VNF, para aumentar a resiliência, basta criar novas instâncias

Tabela 2. Paxos instanciado via *container* na *VNF-Consensus* vs. Paxos instanciado via VM no controlador.

Sistema de Virtualização	Média (s)	Máximo (s)	Mínimo (s)	Desvio Padrão (s)
Paxos *VNF-Consensus*	1.73	2.04	1.58	0.15
Paxos controlador	21.02	23.09	20.18	1.02

da *VNF-Consensus*. Em outras palavras, bastaria criar um novo *container* e iniciar uma nova instância da *VNF-Consensus*.

Concretamente, neste experimento comparamos o tempo para iniciar uma nova instância da *VNF-Consensus* versus o tempo para iniciar um novo controlador SDN. Para avaliar tal custo, o experimento apresentado na tabela 2 mostra a diferença de tempo para iniciar uma instância da *VNF-Consensus* e iniciar um controlador SDN, ambos executando o Paxos. Vale ressaltar que os controladores estão hospedados em máquinas virtuais via *Virtualbox*[7]. Os dados apresentados são resultados de 10 execuções e a medição de tempo foi realizada através do comando *time* do próprio *Linux*.

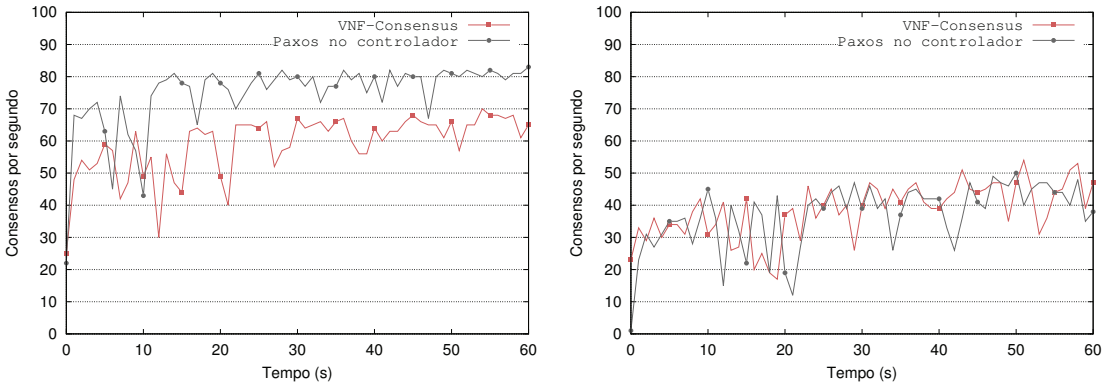
Na tabela 2 podemos observar que o tempo para inicializar uma nova instância via *container* é menor do que via máquina virtual, na comparação entre os valores médios há uma redução de aproximadamente 88% do tempo para instanciar o Paxos via *container*. Além disso, ao analisar os valores máximo, mínimo e desvio padrão podemos notar que a implementação via *container* apresenta baixa variação. Os resultados demonstram claramente a vantagem de executar a função de sincronização como uma VNF, ao invés de acrescentar esta funcionalidade ao controlador.

Análise do throughput do Paxos

Os experimentos apresentados na figura 4 tem como objetivo mostrar e comparar o *throughput* do Paxos, isto é, o número de execuções por segundo quando o consenso está sendo executado na *VNF-Consensus* e quando ele está hospedado nos controladores. No experimento da figura 4(a) o controlador foi submetido a um fluxo de dados com requisições contínuas de atualizações para forçar a sincronização do plano de controle. É possível notar que quando o consenso está sendo executado no controlador há um aumento no *throughput* se comparado à *VNF-Consensus*. Esse resultado é, de certa forma, previsível, uma vez que as instâncias da VNF estão localizadas fora dos controladores e, para toda execução do Paxos, há a necessidade de comunicação entre as VNFs e os controladores. Isso onera o tempo de execução do Paxos via *VNF-Consensus*.

No experimento da figura 4(b), além do fluxo de dados gerado pela ferramenta *cbench*, o controlador é sobrecarregado com requisições REST, provenientes do mesmo *script* utilizado no experimento da figura 3(b). As requisições em paralelo forçam os controladores a executarem consultas contínuas às tabelas de fluxos. Como resultado, note que ambos os casos apresentam uma redução no *throughput*. Porém, quando o Paxos está sendo executado nos controladores a redução foi de aproximadamente 53,3%, ao passo que na *VNF-Consensus* essa redução foi menor, isto é, de aproximado 38,8%. Po-

[7] www.virtualbox.org



(a) Os controladores não executam atividades em paralelo.

(b) Os controladores são forçados a executarem atividades em paralelo.

Figura 4. Comparação quanto ao throughput do Paxos.

demos concluir que apesar da *VNF-Consensus* sofrer com a sobrecarga dos controladores, o impacto maior ocorre quando Paxos está sendo executado nos controladores.

Por fim, ao analisarmos todos os resultados realizados neste trabalho podemos concluir que quando a sincronização do plano de controle é realizada pela *VNF-Consensus*, tem-se uma redução na carga dos controladores, o que tem impacto significativo na escalabilidade, na medida em que há um claro limite para a quantidade de tarefas que um controlador pode executar antes de se tornar um gargalo da rede.

5. Conclusão

Neste trabalho foi proposta uma solução para a sincronização do plano de controle em redes SDN, na qual um grupo de controladores distribuídos se mantêm consistentes com o auxílio de uma função virtualizada de rede denominada de *VNF-Consensus*. A *VNF-Consensus* implementa o algoritmo de consenso Paxos no ambiente de rede. Dessa forma, a *VNF-Consensus* consegue manter um plano de controle consistente pois sincroniza as ações entre todos os controladores envolvidos. Além disso, todas as decisões realizadas pela *VNF-Consensus* são executadas sem a atuação direta dos controladores.

Os resultados experimentais mostraram que quando os próprios controladores executam a sincronização do plano de controle, há um impacto negativo no desempenho global da rede. Por outro lado, com a utilização da *VNF-Consensus*, o plano de controle é sincronizado sem aumentar a carga de trabalho nos controladores. Por consequência, vimos que há melhorias no desempenho da rede. Outro benefício da *VNF-Consensus* refere-se a resiliência em que o Paxos consegue garantir, independentemente do número de controladores operacionais, a conclusão de seus serviços, isto é, a consistência dos controladores. Além disso, a implementação da *VNF-Consensus* em *containers*, como proposto neste trabalho, mostrou ter benefícios, em especial, a otimização no uso dos recursos computacionais, como também rápida instanciação dos serviços.

Para trabalho futuro planeja-se a implementação de um serviço para a sincronização do plano de dados de uma rede SDN, isto é, visando manter a consistência entre todos

os *switches* da rede.

Referências

[Anderson et al. 2016] Anderson, J., Hu, H., Agarwal, U., Lowery, C., Li, H., and Apon, A. (2016). Performance considerations of network functions virtualization using containers. In *International Conference on Computing, Networking and Communications (ICNC'16)*.

[Berde et al. 2014] Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., and Parulkar, G. (2014). ONOS: Towards an Open, Distributed SDN OS. In *3th Workshop on Hot Topics in Software Defined Networking (HotSDN)*.

[Canini et al. 2015] Canini, M., Kuznetsov, P., Levin, D., and Schmid, S. (2015). A distributed and robust SDN control plane for transactional network updates. In *IEEE Conference on Computer Communications (INFOCOM)*.

[Dang et al. 2015] Dang, H. T., Sciascia, D., Canini, M., Pedone, F., and Soulé, R. (2015). Netpaxos: Consensus at network speed. In *Symposium on Software Defined Networking ResearchSymposium on Software Defined Networking Research, (SOSR'15/SIGCOMM)*.

[ETSI 2016] ETSI (Available at http://www.etsi.org/technologies-clusters/technologies/nfv, Accessed on October 02, 2016). Etsi gs nfv 002: Architectural framework.

[Ho et al. 2016] Ho, C. C., Wang, K., and Hsu, Y. H. (2016). A fast consensus algorithm for multiple controllers in software-defined networks. In *18th International Conference on Advanced Communication Technology (ICACT)*.

[Hunt et al. 2010] Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. (2010). ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *USENIX Conference on USENIX Annual Technical Conference (USENIXATC)*.

[Karakus and Durresi 2017] Karakus, M. and Durresi, A. (2017). A survey: Control plane scalability issues and approaches in software-defined networking (SDN). *Computer Networks*, 112.

[Koponen et al. 2010] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., and Shenker, S. (2010). Onix: A Distributed Control Platform for Large-scale Production Networks. In *9th Conference on Operating Systems Design and Implementation (OSDI)*.

[Lamport 1998] Lamport, L. (1998). The Part-time Parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2).

[Lamport 2006] Lamport, L. (2006). Lower bounds for asynchronous consensus. *Distributed Computing*, 19(2).

[Ongaro and Ousterhout 2014] Ongaro, D. and Ousterhout, J. (2014). In Search of an Understandable Consensus Algorithm. In *USENIX Conference on USENIX Annual Technical Conference (USENIXATC)*.

[Santos et al. 2011] Santos, N., Schiper, A., Hutle, M., and Borran, F. (2011). Quantitative analysis of consensus algorithms. *IEEE Transactions on Dependable and Secure Computing*, 9.

[Schiff et al. 2016] Schiff, L., Schmid, S., and Kuznetsov, P. (2016). In-Band Synchronization for Distributed SDN Control Planes. *SIGCOMM Comput. Commun. Rev.*, 46(1).

[Tianzhu et al. 2016] Tianzhu, Z., Andrea, B., Samuele De, D., and Paolo, G. (2016). The role of inter-controller traffic for placement of distributed sdn controllers. In *Networking and Internet Architecture.*

[Zhou et al. 2014] Zhou, B., Wu, C., Hong, X., and Jiang, M. (2014). Programming network via distributed control in software-defined networks. In *2014 IEEE International Conference on Communications (ICC)*.