Um Protocolo Simples e Eficiente para Atualização Consistente de Políticas em Redes Definidas por Software com Controle Distribuído*

Diogo M. F. Mattos[1], Otto Carlos M. B. Duarte[1] e Guy Pujolle[2]

[1]Grupo de Teleinformática e Automação
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro – RJ – Brasil

[2]Laboratoire d'Informatique de Paris 6
Sorbonne Universities, UPMC Univ Paris 06
Paris, France

Resumo. *Novas políticas são instaladas na rede a todo instante. Nas Redes Definidas por Software, os diversos controladores distribuídos têm que instalar as novas políticas de forma consistente, para não haver riscos de a rede passar por estados de configuração transitórios e inesperados, que comprometam a segurança ou mesmo a operação. Este artigo propõe um protocolo de consistência para serializar as atualizações de políticas e para compor as políticas evitando conflitos. As principais contribuições são: (i) um protocolo de consistência para serializar a atualização de políticas; (ii) uma interface de consenso para os controladores acordarem sobre a versão mais recente da configuração da rede; e (iii) um algoritmo para verificar se a nova política é uma atualização, um refinamento, ou se entra em conflito com outras políticas já instaladas. Através de verificação formal, mostra-se que o protocolo de consistência proposto garante uma ordem global para todas as atualizações de políticas e que o algoritmo proposto compõe corretamente todas as políticas. A simulação da proposta em uma topologia de rede real mostra que a atualização distribuída de políticas é consistente por pacote e apresenta uma baixa sobrecarga de mensagens de controle.*

Abstract. *New policies are constantly installed on the network. In Software Defined Networks, the various distributed controllers have to install the new policies consistently for assuring that there is no risks of network experiences transient and unexpected configuration states, which compromise the safety and the operation. In this paper, we propose a consistency protocol for serializing policy updates and for composing policies, avoiding conflicts. The main contributions are three-fold: (i) a consistency protocol for serializing policy updates; (ii) a consensus interface for enabling controllers to agree on the latest version of the network configuration; and (iii) an algorithm to verify that the new policy is an update, a refinement, or if it conflicts with other policies that have already been installed. Through formal verification, we show that the proposed consistency protocol ensures global order for all policy updates and that the proposed algorithm correctly composes all policies. The simulation of the proposal in a real network topology shows that the distributed policy update is per-consistent packet and it has a low overhead of control messages.*

*Este trabalho foi realizado com recursos do CNPq, CAPES e FAPERJ.

1. Introdução

O gerenciamento de redes envolve a definição contínua de políticas de rede que incluem a engenharia de tráfego e o encadeamento de funções de rede [Mattos et al. 2016a, Han et al. 2015, Reitblatt et al. 2012]. O paradigma de Redes Definidas por Software (*Software Defined Networking* - SDN) simplifica o gerenciamento, uma vez que separa o plano de controle, logicamente centralizado, do plano de dados distribuído [Levin et al. 2012]. Aplicações de rede, localizadas no plano de controle, acessam uma visão global consistente da rede. Isso permite definir políticas de alto nível que codificam o comportamento esperado da rede [Canini et al. 2015]. As políticas são traduzidas para o plano de dados, no qual se configura o encaminhamento e tratamento dos pacotes. Em SDN, as regras de encaminhamento são expressas por configurações de fluxo nas tabelas dos comutadores em execução no plano de dados.

O controle logicamente centralizado é o principal pilar do paradigma SDN, embora a realização do controlador de rede como um servidor centralizado implique desafios para a segurança, o desempenho e a escalabilidade da rede [Mattos et al. 2016a]. Portanto, o controle e a consequente manipulação das atualizações de política em SDN são um desafio de computação distribuída. O bom funcionamento da rede depende da coordenação e do tratamento consistente das atualizações de políticas que chegam simultaneamente aos controladores, além da composição da interação entre todas as políticas aplicadas na rede [Canini et al. 2015, Reitblatt et al. 2012].

Este artigo propõe um protocolo simples e eficiente para a serialização[1] da instalação de atualizações de políticas em Rede Definidas por Software com um plano de controle distribuído. A ideia principal é garantir o consenso entre os controladores quanto à aplicação de uma nova política sobre a rede. Quando atualizações de política chegam concomitantemente a diferentes controladores, esses devem concordar sobre a ordem de instalação de todas as atualizações requisitadas e, também, se a nova política gera conflito com as demais. Por isso, as principais contribuições deste artigo são três: (i) um protocolo simples de consistência, que serializa a instalação de atualizações de políticas simultâneas lançadas por diferentes controladores; (ii) uma interface de consenso abstrata, na qual os controladores acordam sobre a versão mais atual da configuração da rede; e (iii) um algoritmo simples para verificar se uma nova política é uma atualização, um refinamento, ou se está em conflito com outras políticas já instaladas na rede.

O restante do artigo está organizado da seguinte forma. Na Seção 2, apresentam-se os trabalhos relacionados. A Seção 3 discute os desafios da atualização das políticas em Redes Definidas por Software com um plano de controle distribuído. Na Seção 4, propõe-se o protocolo de consistência para atualizações de políticas em Redes Definidas por Software com controle distribuído. As simulações e os resultados são apresentados e discutidos na Seção 5. A Seção 6 conclui o artigo.

2. Trabalhos Relacionados

Reitblatt *et al.* propõem o esquema de atualização de duas fases (*Two-Phase Update*), em que aplicam o modelo de consistência por pacote para a atualização consistente

[1]No contexto deste artigo, serialização refere-se ao ordenamento de ações na rede sem que haja sobreposição temporal entre ações.

de regras nos comutadores em SDN com controle centralizado [Reitblatt et al. 2012]. Esse esquema de atualização de duas fases insere etiquetas nos pacotes quando entram na rede, indicando a marcação da versão mais recente da configuração. Durante o encaminhamento do pacote na rede, ele é sempre processado de acordo com a versão da configuração da rede marcada na etiqueta, o que evita a manipulação de pacotes por duas versões distintas da configuração da rede. No processo de atualização, a primeira fase acrescenta novas regras nos comutadores do núcleo da rede, que aplicam a nova política aos pacotes marcados com a nova etiqueta de versão da configuração. A segunda fase atua nas portas de entrada dos comutadores de borda. Nessa fase, o sistema atualiza as regras sobre os comutadores de ingresso para marcar os pacotes entrantes com a nova versão.

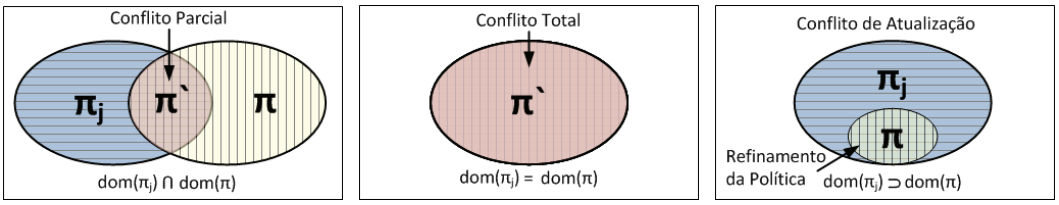
Canini *et al.* estendem a proposta de atualização de duas fases a uma Rede Definida por Software com controle distribuído [Canini et al. 2015]. Canini *et al.* introduzem o problema de Composição Consistente de Políticas (*Consistent Policy Composition* - CPC), no qual eles definem que para aceitar atualizações de políticas de uma forma consistente em uma rede com controle distribuído é necessário compor todas as políticas em uma configuração de rede única e sem conflitos. As políticas aceitas não podem entrar em conflito com as políticas já implantadas ou com outras instalações de políticas concomitantes. Os autores propõem uma interface transacional para a busca de conflitos entre as políticas e, após, a aceitação simples das políticas que não entrem em conflito com as demais. Embora Canini *et al.* resolvam o problema de Composição Consistente de Políticas e alcancem a complexidade ideal de etiquetas, a proposta assume a pré-existência de uma abstração consenso entre os controladores.

Mattos *et al.* propõem que a atualização das regras na rede seja feita na ordem inversa dos fluxos, garantindo assim a consistência e evitando a necessidade de marcação de pacotes com etiquetas de versão [Mattos et al. 2016b, Mattos e Duarte 2015]. McGeer propõe realizar o armazenamento temporário dos pacotes em trânsito no controlador durante a atualização da rede. Após a atualização, os pacotes são reinjetados na rede [McGeer 2012]. Katta *et al.* propõem realizar a atualização em rodadas. Em cada rodada, eles executam uma atualização de duas fases em um subconjunto de fluxos. A ideia principal é remover o antigo conjunto de regras após cada rodada, liberando a memória dos comutadores [Katta et al. 2013]. McClurg *et al.* propõem um algoritmo para procurar automaticamente uma ordem para atualizar a configuração de rede, em que sejam garantidas as propriedades invariantes da rede em todos os estados de configuração transitórios [McClurg et al. 2015]. Essas propostas, no entanto, focam em Redes Definidas por Software com um controle centralizado.

No caso de nós distribuídos, a ideia mais simples de consenso é decidir se uma transação deve ser confirmada ou não [Gray e Lamport 2006]. Um protocolo simples para alcançar um consenso é o protocolo *Two-Phase Commit*, ou efetivação em duas fases, que utiliza uma fase para propor uma transação e, depois de todos os nós confirmarem que concordam com a transação, uma segunda fase envia o comando de efetivação a todos os nós. No entanto, o *Two-Phase Commit* pode gerar uma situação de impasse, caso o nó que inicia o protocolo falhe. Uma solução para a resolução do impasse é o protocolo de efetivação de três fases (*Three-Phase Commit*) que adiciona uma fase extra entre a votação e a efetivação da transação, para facilitar a recuperação do estado da transação no caso de falha do nó iniciador.

3. O Problema de Composição Consistente de Políticas

O plano de controle logicamente centralizado consiste em uma abstração de uma visão rede global compartilhada por todos os controladores da rede [Mattos et al. 2016a]. Assim, todos os controladores têm acesso a uma interface de consenso para atualizar sua visão de rede global. Em um cenário de controle distribuído, diferentes controladores podem realizar requisições atualizações de políticas concomitantes na rede. A fim de aplicar de forma consistente as atualizações na rede, cada controlador tem que estar consciente sobre a ordem de instalação das atualizações e, também, se uma requisição de uma nova política entra em conflito com as demais. Desta forma, Canini *et al.* definem o problema da Composição Consistente de Política (*Consistent Policy Composition* - CPC) [Canini et al. 2015], que consiste em consolidar as políticas de rede que chegam aos controladores concomitantemente em uma configuração da rede única, consistente e global, na qual não haja conflito entre diferentes políticas.



(a) Conflito parcial entre π e π_j. (b) Conflito total entre π e π_j. (c) Conflito de atualização entre π e π_j.

Figura 1. Casos possíveis de conflitos entre a nova política π com uma política já instalada π_j no espaço de fluxos S: a) A solução para um conflito parcial é a criação de um subconjunto de políticas π'; b) A solução para um conflito total é a avaliação das duas políticas e a consequente instalação de uma nova política π' que considere os casos em conflito; c) π é o refinamento de uma política π_j ou uma atualização explícita. O conflito de atualização é trivialmente resolvido aplicando-se a política π.

Considerando-se o problema de atualização de políticas em um ambiente de controle distribuído, identificam-se duas propriedades que devem ser satisfeitas: a consistência e a composição. A instalação consistente de políticas consiste em agendar requisições de atualização de políticas e garantir que é possível estabelecer uma ordem global entre todos os pedidos programados. Em um segundo momento, considera-se o problema de composição de políticas. Assim, o problema composição lida com a aceitação, ou rejeição, de uma nova política, ou uma atualização de política, considerando as políticas que já foram aplicadas à rede. Portanto, dois subproblemas são definidos: serialização das requisições e composição de políticas.

O problema de serialização de requisições consiste em definir uma ordem global consistente entre todas as requisições concomitantes. Seja H a história da rede, isto é, o conjunto de todos os eventos que acontecem na rede. Uma relação parcial de ordem nos eventos da história H é definida como $<_H$. Uma requisição req precede outra requisição req' na história H, representada por $req <_H req'$, se a reposta de req aparece antes da chamada de req' em H [Canini et al. 2015, Mattos e Duarte 2015]. Se duas requisições não estão relacionadas em uma ordem de precedência, diz-se que ambas são requisições concomitantes. De forma similar, um evento de rede ev, como a chegada de um novo

pacote, precede uma requisição req em H, representado por $ev <_H req$, se ev ocorre antes da chamada de req em H. Ademais, o evento ev sucede req em H, $req <_H ev$, se ev ocorre após a resposta à req. Um evento ev é concorrente com a requisição req se $ev \not<_H req$ e $req \not<_H ev$. A história H é sequencial se não existe duas requisições concorrentes nem um evento concorrente com uma requisição.

Seja H_{c_i} a história local controlador c_i uma subsequência de H, que consiste de todos os eventos e requisições que ocorrem em c_i. Localmente, verifica-se que toda e qualquer história H_{c_i} é sequencial, já que um controlador aceita uma nova requisição se, e somente se, não existir uma requisição prévia sem reposta, nem um evento sendo tratado. A história H é consistente se a relação de precedência entre duas requisições na história local de um controlador é mantida em qualquer outra história local de outros controladores. Assim, uma requisição consistente mantém a propriedade

$$req <_{H_{c_i}} req' \rightarrow req <_{H_{c_j}} req', \tag{1}$$

$\forall \{req, req'\} \subset H_{c_i} | req <_H req'$ e $\forall c_j \in C$, onde C é o conjunto de todos os controladores da rede. Assim, considera-se que a ordem local de todos os controladores é consistente com a ordem global definida em H.

Embora a ordem global defina a serialização consistente das requisições, ainda é necessário definir corretamente a composição entre as novas políticas e as já aplicadas na rede. A fim de abordar a composição de políticas, considera-se que H deve respeitar as seguintes propriedades [Canini et al. 2015]:

- uma política é aceita com sucesso para ser adicionada à H se, e somente se, não entra em conflito com nenhuma outra política já aplicada à H;
- para cada evento de entrada de pacote ev na rede, o comportamento da rede é consistente com a composição de todas as políticas aplicadas com sucesso em H e que precedam o evento ev em H.

O conflito entre políticas surge quando duas políticas agem em conjuntos de fluxos de sobrepostos [Ferguson et al. 2012, Luo et al. 2015]. Políticas livres de conflitos são aquelas que possuem domínios completamente disjuntos[2]. Sejam π uma requisição de atualização de política, Π o conjunto de todas as políticas já instaladas na rede, e $dom(\pi)$ o domínio da política π, então as políticas não-conflitantes respeitam a propriedade

$$dom(\pi) \cap dom(\pi_j) = \emptyset, \forall\, \pi_j \in \Pi. \tag{2}$$

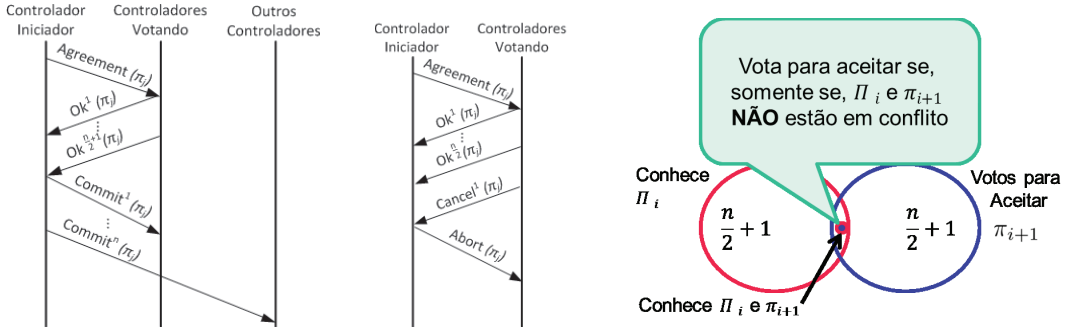
É possível identificar três tipos de conflito: conflito parcial, conflito total e conflito de atualização. O conflito parcial é quando o domínio de uma política sobrepõe o domínio de outra. Nesse caso, a composição das duas políticas pode levar à criação de regras de manipulação de pacotes para cada subconjunto, como mostrado na Figura 1(a). O conflito total ocorre quando duas políticas diferentes têm o mesmo domínio. Nesse caso, mostrado na Figura 1(b), a nova política substitui a anterior, ou a nova política é totalmente rejeitada. O terceiro caso é o conflito de atualização, Figura 1(c). De fato, o conflito de atualização

[2]O domínio de uma política é o conjunto de fluxos que a política afeta, o subespaço de fluxo (partição do *flowspace*) [Reitblatt et al. 2012, Ferguson et al. 2012].

ocorre quando uma nova versão da política anterior é lançada e, assim, a anterior deve ser explicitamente substituída por uma nova política com o mesmo domínio. Em um caso de instalação de uma política que é um refinamento da anterior, quando o domínio de nova política é subconjunto de uma política instalada anteriormente, a nova requisição deve ser tratada como um conflito de atualização.

4. O Protocolo de Consistência Proposto

A ideia principal da proposta é definir uma ordem global entre todas as requisições de atualização de política. A instalação de uma política na rede ocorre em três passos. O primeiro é o recebimento da requisição de atualização de política por um controlador. O segundo passo é o lançamento da política na rede, que consiste em votar a sua aplicabilidade e o número de ordem a ser assumido pela política. Se a atualização política é aceita por uma maioria de controladores, recebe, então, um número global de ordem. A partir de então, inicia-se o terceiro passo, em que a atualização de política é efetivada na rede seguindo o esquema de atualização de duas fases [Reitblatt et al. 2012]. A efetivação da atualização de política consiste na instalação da política nos comutadores, traduzindo a política de alto nível em regras de manipulação de pacotes. A proposta deste artigo concentra-se no primeiro e segundo passos.



(a) Instalação bem sucessidida da política π_j.

(b) Aborto da instalação da política π_j.

(c) Ilustação do algoritmo de verificação de conflitos.

Figura 2. O protocolo de consistência proposto. O controlador iniciador recebe uma requisição de atualização de políticas e começa a instalação da política π_j na rede. a) Todos $n/2+1$ controladores concordam em aceitar a instalação de π_j. O controlador iniciador envia as mensagens de `commit` a todos controladores na rede. b) Se ao menos um controlador discordar da instalação de π_j, o iniciador envia uma mensagem de `abort` para todos os controladores que estavam votando na aceitação da política. c) Algoritmo de verificação de conflitos garante que ao menos um nó dos votantes conhece todas as políticas já instaladas.

O protocolo de consistência proposto funciona da seguinte maneira. A requisição de atualização de política chega a um controlador[3]. Requisições podem chegar simultaneamente em diversos controladores. O controlador iniciador é aquele que recebe uma requisição, vinda de uma aplicação de controle, inicia a execução do protocolo e envia

[3]Uma requisição de atualização de política chega ao controlador através da *Northbound API* ou através da East/Westbound API. No primeiro caso, uma aplicação de controle gera a requisição a ser tratada pelo controlador. No segundo caso, o controlador recebe uma mensagem de `agreement` de outro controlador na rede.

mensagens de `agreement` para $n/2 + 1$ outros controladores, em que n é o número total de controladores na rede. A mensagem de `agreement` contém a requisição de atualização de política, assim como o domínio de atuação da política e o número de ordem proposto, dado pela ordem mais recente conhecida pelo iniciador acrescida de uma unidade. Quando um controlador recebe uma mensagem de `agreement`, duas mensagens de resposta são possíveis: `ok` e `cancel`. Se todos os $n/2 + 1$ controladores respondem com `ok`, o controlador iniciador então envia uma mensagem de `commit` para todos os controladores na rede, indicando que o processo de instalação pode ser realizado, como mostrado na Figura 2(a). Ao receber uma mensagem `commit`, o controlador verifica se o número de ordem na mensagem é compatível com a versão da próxima configuração de rede que ele espera receber. Em caso afirmativo, o controlador instala a nova política. No caso de a mensagem `commit` apresentar um número de ordem superior ao que os controlador estava esperando, o controlador sincroniza a sua base de dados de políticas instaladas como o controlador iniciador. No caso de o controlador iniciador receber uma mensagem `cancel` como resposta à sua mensagem de `agreement`, o controlador iniciador aborta a instalação da atualização de política e sincroniza sua base de políticas com o outro controlador que enviou a mensagem de `cancel`, como mostrado na Figura 2(b).

Execução do protocolo. Como mostrado na Figura 2(a), se $n/2+1$ controladores concordam em efetivar a atualização da política, a informação é propagada para todos os controladores e a atualização da política está instalada na rede muda de acordo com o esquema de atualização de duas fases (*Two-Phase Update*) [Reitblatt et al. 2012]. No entanto, falhas podem surgir durante a execução do protocolo proposto. Quatro casos de falha são tratados pelo protocolo: i) mais do que um controlador iniciam o protocolo ao mesmo tempo; ii) o controlador iniciador falha após o envio da mensagem de `agreement` para qualquer nó; iii) um nó falha depois de receber a mensagem de `agreement`; e iv) um nó identifica que o seu número da ordem da configuração global está desatualizado.

Se dois ou mais controladores iniciam o protocolo simultaneamente, apenas um deles é capaz de alcançar adequadamente $n/2 + 1$ mensagens de confirmação (`ok`). No caso de mais de dois iniciadores, pode acontecer de nenhum deles atingir o número necessário de votos e todos recebam uma mensagem de `cancel`, que indica a existência de outra transação em andamento. Nesse caso, cada controlador, que recebe a mensagem `cancel`, relança a sua requisição de atualização após a espera por um tempo aleatório.

Se o controlador iniciador falhar antes de enviar qualquer mensagem `agreement`, não há danos para o estado atual do protocolo, pois não existe qualquer transação iniciada. Por outro lado, se ele falhar depois enviar qualquer mensagem `agreement`, um controlador que recebeu a mensagem, a reenvia, acrescentando a sua própria assinatura na mensagem, e a marca como uma mensagem de recuperação. O novo controlador iniciador envia a mensagem `agreement` reassinada a um novo grupo de $n/2 + 1$ controladores. No caso de a atualização de política já haver sido instalada na rede, o novo iniciador apenas atualiza seu banco de dados de políticas, instala a política nos comutadores que controla e propaga a informação. Caso contrário, ele segue o protocolo como se fosse o iniciador da requisição.

O caso de uma falha de um controlador é tratado de duas formas diferentes. Primeiro, se um controlador falha após receber a mensagem de `agreement`, o controla-

dor iniciador aguarda sua resposta até que um tempo limite ou, se a falha interrompe a conexão TCP, o iniciador detecta imediatamente a perda de conexão. Em seguida, o iniciador envia uma nova mensagem `agreement` para outro controlador até que alcança $n/2 + 1$ votos positivos, mensagens de `ok`, ou pelo menos uma mensagem `cancel`. O segundo caso é quando o controlador não participa em qualquer acordo entre os controladores. Nesse caso, a falha é ignorada. Quando um controlador recupera-se de uma falha, ele atualiza seu banco de dados de políticas com qualquer outro controlador ativo na rede.

Se todos controladores seguem o protocolo sem falhas, esse caso não é viável. Contudo, um controlador pode falhar e não atualizar sua base de dados de políticas. Um controlador sabe que está desatualizado quando recebe uma mensagem com o número de ordem de política maior do que a que ele está esperando. A mensagem pode ser `agreement`, `cancel` ou `commit`. Quando recebe uma `agreement` ou `commit`, o controlador desatualizado pede ao controlador atualizado, que a enviou a mensagem, a base de dados mais recente das políticas da rede. No caso de uma mensagem de `cancel`, o controlador desatualizado aborta a etapa de efetivação da política e, em seguida, atualiza sua base de dados com o controlador que o enviou a mensagem com o número mais recente de ordem de políticas.

Vale ressaltar que o protocolo proposto é mais simples do que outros protocolos de consenso, como Paxos [Prisco et al. 2000] e Zab [Junqueira et al. 2011], já que descarta a etapa de votação de líderes e flexibiliza as restrições de durabilidade. A proposta assume alguns detalhes de implementação, como o uso de conexões TCP que mantém o estado da conexão de cada controlador e assegura a confiabilidade e a ordenação das mensagens. Assim, a proposta também garante uma maior disponibilidade do que o protocolo de efetivação de transação de duas fases (*Two-Phase Commit*), apesar de ainda alcançar a efetivação da transação em dois tempos de ida e volta (RTT). Destaca-se ainda que a proposta não assume qualquer mecanismo de sincronização ou qualquer interface de consenso entre os controladores. A solução de compromisso assumido nesta proposta é aumentar a complexidade em relação ao número de etiquetas de marcação de versões de configuração da rede em relação à simplicidade do consenso e ao desprezo ao monitoramento de quais marcações continuam válidas ou não na rede. Considera-se que os nós não possuem um limite superior sobre o número de ordem de política, apesar de considerá-lo um contador cíclico.

O algoritmo de composição de políticas. O algoritmo proposto funciona localmente como mostrado no Algoritmo 1. A ideia principal é de buscar qualquer tipo de conflito (total, parcial ou conflito de atualização) que possa aparecer entre a nova requisição de atualização de política π e o conjunto de políticas já definidas na rede Π, exemplificado na Figura 2(c). Dessa forma, assim que a requisição de atualização de política chega ao controlador, o algoritmo verifica localmente se o domínio da requisição, $dom(\pi)$, está em conflito com a união do domínio de todas as outras políticas já instaladas, $\cup\, dom(\pi_i), \forall \pi_i \in \Pi$. Se o algoritmo identifica um conflito, parcial ou total, o controlador recusa a requisição de atualização de política. Assim, se a requisição chegou ao controlador através de uma mensagem `agreement`, o controlador a recusa através de resposta com a mensagem `cancel`, sinalizando a existência de um conflito. Se a nova política a ser instalada é verificada e conclui-se que é livre de conflito, Expressão 2, ou é

uma atualização explícita, o algoritmo aceita a nova política e, então, a política é instalada, sem qualquer modificação em relação à sua proposição inicial. Vale ressaltar, que a proposta instala as políticas na rede sob uma abordagem de tudo-ou-nada (*all-or-nothing*), em que a política ou é totalmente aceita ou totalmente recusada. Não há a aceitação parcial de políticas. Esse comportamento é desejado, pois garante que não há a possibilidade de geração de estados intermediários e inconsistentes.

Algoritmo 1: Algoritmo de Composição de Atualização de Políticas. O algoritmo executa localmente em cada controlador. A saída do algoritmo é o voto do controlador, a favor ou contra, a instalação da política verificada.

Entradas: π_i (requisição de atualização de política)
Π (conjunto de todas as políticas já instaladas)
conflito := Falso
for $\pi_j \in \Pi$ **do**
  if $dom(\pi_i) \cap dom(\pi_j) \neq \emptyset$ ***e*** $isUpdate(\pi_i, \pi_j) = False$ **then**
    conflito := Verdadeiro
  end
end
Saída : conflito

O Algoritmo 1 verifica se os domínios de atuação das políticas se sobrepõem. Contudo, o algoritmo pode ser aplicado, sem perda de generalidade ou corretude, com a adoção de métodos mais complexos de identificação de conflitos e composição de regras de forma mais elaborada de políticas, como através do uso a linguagem `Pyretic` [Monsanto et al. 2013].

Prova de corretude. Para provar a corretude de funcionamento da proposta, é necessário provar que a instalação de políticas na rede respeitam duas propriedades: i) as políticas são serializadas e ii) as políticas instaladas não estão em conflito com qualquer outra política na rede. Assim, primeiro prova-se por contradição que a ordem global de instalação das políticas, definida entre os controladores, é a mesma que a ordem local de qualquer controlador na rede. Após, prova-se usando o mecanismo de indução que a composição de todas as políticas é consistente.

Teorema 1: A ordem local de instalação de políticas em qualquer controlador da rede é compatível com a ordem global.

Prova por contradição. Assume-se que a ordem local de instalação de políticas em um dos controladores da rede não é compatível com a ordem global de instalação. Assim, assume-se que existe o controlador c_i, em que a política π_2 precede a política π_1, $\pi_2 <_{Hc_i} \pi_1$, e, na ordem global, π_1 precede π_2, $\pi_1 <_H \pi_2$. Como qualquer outro controlador, diferente de c_i, é compatível com a ordem global, tem-se que existe um controlador c_j, em que a ordem local é $\pi_1 <_{Hc_j} \pi_2$. As c_i e c_j executam corretamente o protocolo proposto e não é possível de ocorrer reordenamento de mensagens na rede. Para instalar a política π_2 antes da instalação de π_1, o controlador c_i teve que obter no mínimo $n/2 + 1$ votos de outros controladores (mensagens `ok`), assim como os demais controladores tiveram que obter $n/2 + 1$ votos para instalar π_1 antes de π_2. De acordo com o protocolo, um controlador

não pode votar em ordens contraditórias. Assim, o único modo possível de haver duas ordens locais diferentes é através da votação das ordens por dois conjuntos disjuntos de controladores votantes. Nesse caso, seriam necessários $(n/2+1)+(n/2+1) = n+2$ votos. Como a rede apresenta apenas n controladores, esse é um cenário impossível e, então, prova-se que uma ordem local diferente da ordem global é uma contradição lógica.

Teorema 2: A composição das políticas é livre de conflitos.

Prova por indução. O mecanismo de indução é usado sobre o conjunto Π, que representa o conjunto de todas as políticas instaladas na rede.

`Caso base` ($dom(\Pi_0) = \emptyset$) **:** Nesse caso, o conjunto Π é vazio e, então, é trivialmente um conjunto de políticas não-conflitantes.

`Hipótese indutiva` ($dom(\Pi_i) = \cup_{k=1}^{i} dom(\pi_k)$) **:** Seja Π_i o conjunto de todas as políticas instaladas até a requisição π_i, para $i > 0$. Assim, o domínio de Π_i é definido como a união dos domínios de todas as políticas no conjunto. Por hipótese, considera-se que a composição de todas as políticas em Π_i é consistente.

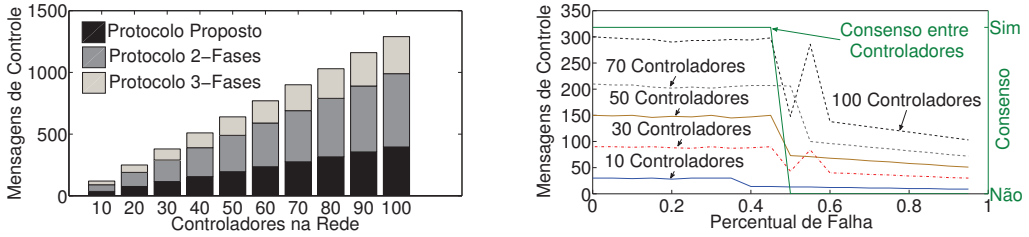
`Passo indutivo` ($dom(\Pi_{i+1}) = dom(\Pi_i) \cup dom(\pi_{i+1})$) **:** Considera-se que todas as políticas em Π_i são compostas de maneira consistente, como é previsto na hipótese indutiva. Assim, a prova consiste em mostrar que a inclusão da política π_{i+1} em Π_i não gera conflitos. Para tanto, usa-se o algoritmo proposto de verificação de conflitos. O algoritmo executa localmente e a resposta do algoritmo é se o controlador deve votar a favor ou contra a aceitação da requisição de atualização de política. Adicionar π_{i+1} ao conjunto Π_i só é possível se, e somente se, $n/2+1$ controladores votarem a favor, garantindo que não há conflitos entre a nova política π_{i+1} e todas as demais políticas em Π_i. De acordo com o Teorema 1, no mínimo um controlador, entre os $n/2+1$ controladores que votam, deve já ter instalado todas as políticas em Π_i e, portanto, está de acordo com a ordem global de políticas. Se todos os $n/2+1$ controladores aprovam a nova política significa que a nova política π_{i+1} não entra em conflito com nenhuma outra política instalada na rede, já que ao menos um controlador no grupo de controladores votantes conhece todas as políticas já aprovadas. Caso contrário, a política π_{i+1} é completamente rejeitada. Logo, prova-se o Teorema 2, mostrando que a composição de Π_i com π_{i+1} só é possível se não houver conflitos.

5. Resultados Experimentais

O protocolo de consistência proposto foi avaliado através da simulação do consenso entre os nós controladores de uma SDN com controle distribuído, aplicando-se uma extensão para o cenário distribuído do simulador SDN desenvolvido por Mattos *et al.* [Mattos et al. 2016b, Mattos e Duarte 2015].

Um protótipo do mecanismo foi implementado para avaliar a carga de mensagens trocada entre os nós. Nesse experimento, consideram-se, a critério de comparação, os protocolos de efetivação de transações de duas fases (*Two Phase Commit* – 2PC) e de três fases (*Three Phase Commit* – 3PC). A Figura 3(a) compara o número de mensagens enviadas pelos protocolos de efetivação de duas fases (`2PC`), efetivação de três fases (`3PC`) e o protocolo de consenso proposto (`Proposto`) para a instalação de uma política na rede. As topologias consideradas são malhas completas de 10 a 100 nós controladores. Os resultados evidenciam que a quantidade de mensagens enviadas pelo protocolo pro-

posto é menor que a enviada pelo protocolo de efetivação de duas fases. Ao se considerar 30 controladores, por exemplo, a redução no número de mensagens de controle chega a 25%, quando comparado com o protocolo de efetivação de duas fases e 50%, com o de três fases. Considerando n o número de nós que estão executando os protocolos, a análise do comportamento de cada protocolo revela que o número esperado de mensagens, em um cenário sem falhas, para o `2PC` é de $4 \times (n-1)$ mensagens e para o `3PC`, $6 \times (n-1)$ mensagens. O protocolo proposto, por sua vez, apresenta no máximo $3n$ mensagens.



(a) Mensagens para uma atualização. (b) Mensagens enviadas em cenários de falhas.

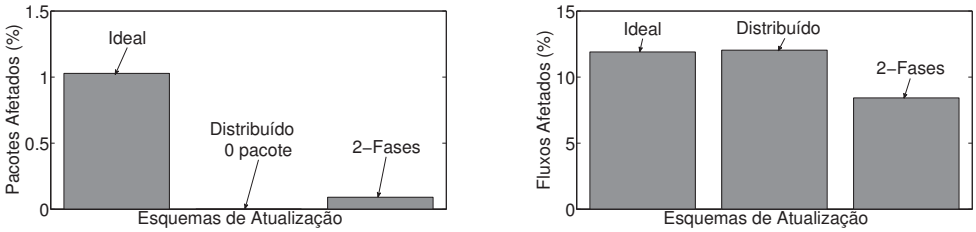
Figura 3. Comparação da carga de mensagens de controle gerada pelo protocolo proposto (`Proposto`) com os protocolos de efetivação de duas fases (`2PC`) e três fases (`3PC`). a) Mensagens de controle em função do número de controladores, a proposta reduz de 25% em relação `2PC` para um número de controladores maior que 30. b) Mensagens de controle em função do percentual de falha de nós da rede. A proposta mantém um baixo número de mensagens e converge mesmo até quando quase metade da rede falha. `Sim` indica o consenso pela aceitação da atualização e `Não`, o consenso pelo aborto da operação.

O experimento seguinte avalia a resiliência do protocolo proposto à ocorrência de falhas na rede. A Figura 3(b) mostra o número de mensagens enviado na rede quando há falhas nos nós. Os nós alcançam o consenso, mesmo quando o índice de falhas na rede é próximo a 50%, $n/2 - 1$ nós falham. Caso a maioria dos nós falhem, as requisições não são aceitas e, então, são abortadas pelo protocolo. A Figura 3(b) evidencia a baixa carga de mensagens na rede, mesmo quando as transações são abortadas. No caso em que a proposta aborta a efetivação das políticas na rede, o número de mensagens é reduzido e apresenta pouco impacto no funcionamento normal da rede. Quando ocorrem falhas, o protocolo proposto envia novas mensagens a nós aleatórios até exaurir a busca por nós ativos ou conseguir o número necessário de votos. Contudo, essa busca pode gerar a expiração do tempo limite de espera por uma reposta dos controladores ativos e que já responderam. Por essa razão, verifica-se a incidências de picos de envio de mensagens nos cenários em que as falhas na rede estão próximas a 50% dos nós controladores, Figura 3(b).

Na segunda etapa de avaliação da proposta, foi simulada uma SDN, baseada na topologia real de rede da RNP, no Brasil, com 31 nós[4]. Os parâmetros da simulação definem que a chegada de novos fluxos é uniformemente distribuída entre todos os nós da rede e o intervalo de chegada entre fluxos segue uma distribuição *log-normal* com média 7 ($\mu = 7$) e desvio padrão igual a 2 ($\sigma = 2$) [Mattos et al. 2016b, Mattos e Duarte 2015]. A chegada de fluxos acontece durante os 900 primeiros passos de simulação, cada fluxo

[4]O grafo da topologia da rede foi obtido em *The Internet Topology Zoo*, disponível em http://www.topology-zoo.org/.

é modelado com uma duração de 50 passos e o fim da simulação é determinado quando não há mais pacotes ou eventos a serem tratados.



(a) Percentual de pacotes em trânsito que são encaminhados por duas configurações distintas.

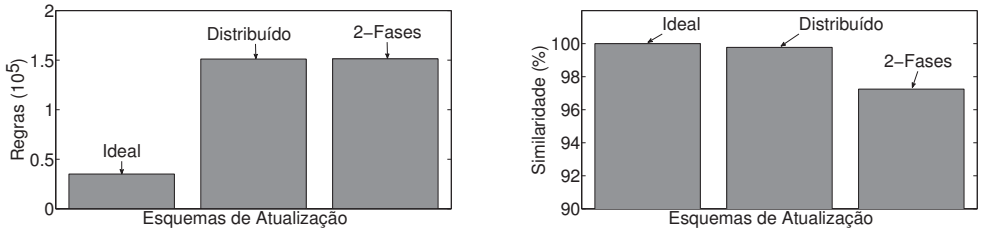
(b) Percentual de pacotes que são encaminhados por duas configurações distintas.

Figura 4. Impacto das atualizações nos pacotes e fluxos encaminhados na rede.

A simulação do esquema distribuído foi realizada definindo-se um controlador para cada nó da rede e todos os controladores executam o protocolo de consenso. O protocolo de consenso acorda quanto à versão da configuração da rede e a distribui entre os demais controladores. O esquema de atualização com controle distribuído baseado no protocolo de consenso (`Distribuído`) foi comparado com os esquemas centralizados de atualização de duas fases (`2-Fases`) e atualização ideal (`Ideal`). O ideal é factível somente em um cenário simulado, pois considera que todo o encaminhamento de pacote é interrompido durante o processo de atualização das regras de encaminhamento nos comutadores. Contudo, a atualização ideal é considerada como um esquema de atualização proporcional, ou seja, aquele em que o custo de instalação da atualização é proporcional às mudanças implementadas [Reitblatt et al. 2012]. Assim, ao comparar um esquema de atualização com a atualização ideal, verifica-se o quão próximo o esquema proposto está de uma atualização proporcional.

A Figura 4(a) compara o comportamento dos esquemas de atualização de política em relação aos pacotes encaminhados na rede. O esquema distribuído não encaminha nenhum pacote por mais de uma versão de configuração da rede, comportamento esperado para um esquema consistente por pacote. Contudo, durante a atualização de duas fases, verifica-se a ocorrência de um pequeno percentual de pacotes que é encaminhado por mais de uma configuração de rede. Isso ocorre porque o modelo de controlador considerado é o mais ingênuo possível, em que após a atualização ele sempre marca os pacotes com a nova configuração de rede sem guardar qualquer estado. Assim, pacotes de fluxos que ainda não foram instalados em comutadores intermediários, podem chegar a um controlador já atualizado, a partir de então, são encaminhados por uma nova configuração [Mattos et al. 2016b]. A atualização por controladores distribuídos age mais prontamente na rede do que a atualização de duas fases com controle centralizado, apresentando um resultado mais próximo ao ideal. O efeito é evidenciado pelo número de fluxos afetado pelas atualizações na rede, mostrado na Figura 4(b). O esquema distribuído atualiza 42% mais de fluxos do que o centralizado de atualização de duas fases.

O número total de regras instaladas nos comutadores da rede é evidenciado na Figura 5(a). Essa métrica indica o quanto da memória dos comutadores é usada por cada esquema de atualização. Como os esquemas de atualização distribuído e de atualização de duas fases centralizado instalam regras no núcleo da rede para garantir a consistência



(a) Número total de regras instaladas na rede. (b) Similaridade com o ideal.

Figura 5. Comparação do número de regras e do efeito no destino causado pelo uso dos esquemas de atualização. a) Número total de regras instaladas na rede. b) A similaridade das propostas em relação ao esquema ideal.

por pacote, o número de regras instalado por esses esquemas chega a ser 4x superior ao do ideal[5]. Por sua vez, a Figura 5(b) compara o resultado do encaminhamento no destino dos pacotes. A similaridade mede o quão próximo o encaminhamento dos pacotes em cada esquema de atualização está do ideal. Essa medida fornece uma estimativa da qualidade de cada esquema de atualização. Verifica-se que o esquema de duas fases já apresenta um desempenho muito próximo do ideal. Contudo, a proposta do esquema distribuído alcança um resultado ainda mais próximo do ideal devido à coordenação eficiente de ações entre os controladores com o uso do protocolo de consistência proposto.

6. Conclusão

Atualizações de políticas de forma consistente em uma Rede Definida por Software com plano de controle distribuído é um desafio, pois as requisições devem ser ordenadas globalmente e as políticas, compostas sem conflitos. Esse artigo propõe um protocolo de consistência para controladores distribuídos, em que o conflito entre políticas é verificado localmente e a ordem global de instalação é garantida através do acordo entre controladores. O artigo propõe ainda um algoritmo simples para a composição de políticas que se aproveita da interface de consenso provida pelo protocolo de consenso. O algoritmo é executado localmente e sua interação com o protocolo de consistência proposto assegura que todas as políticas aceitas são livres de conflitos. A prova de corretude do protocolo e do algoritmo propostos é realizada através de verificação formal. A simulação de um cenário de aplicação da proposta em uma topologia de rede real mostra que o número de mensagens do protocolo proposto é inferior ao das demais propostas, mesmo em cenários de falha, e que a proposta alcança atualizações consistentes em dois tempos de ida e volta. Os resultados mostram ainda que a proposta alcança o consenso e, consequentemente, atualizações consistentes sem a ocorrência de impasses, mesmo quando até $n/2 - 1$ controladores apresentam falhas. A simulação da aplicação do protocolo proposto em uma topologia de rede real mostra que o esquema distribuído de atualização de políticas aumenta em até 42% o número de fluxos que são tratados pela configuração mais recente da rede e mantém a garantia de que cada pacote em trânsito é consistentemente encaminhado por apenas uma configuração de rede.

[5]Há fluxos não expiraram na tabela de fluxos dos comutadores e são afetados por mais de uma atualização, gerando um aumento ainda maior no número de regras instaladas do que a instalação de uma regra a mais por fluxo em cada comutador.

Referências

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

[Ferguson et al. 2012] Ferguson, A. D., Guha, A., Liang, C., Fonseca, R. e Krishnamurthi, S. (2012). Hierarchical policies for software defined networks. Em *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN'12, páginas 37–42, New York, NY, USA. ACM.

[Gray e Lamport 2006] Gray, J. e Lamport, L. (2006). Consensus on transaction commit. *ACM Trans. Database Syst.*, 31(1):133–160.

[Han et al. 2015] Han, J. H., Mundkur, P., Rotsos, C., Antichi, G., Dave, N., Moore, A. e Neumann, P. (2015). Blueswitch: enabling provably consistent configuration of network switches. Em *Architectures for Networking and Communications Systems (ANCS), 2015 ACM/IEEE Symposium on*, páginas 17–27.

[Junqueira et al. 2011] Junqueira, F. P., Reed, B. C. e Serafini, M. (2011). Zab: High-performance broadcast for primary-backup systems. Em *IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN), 2011*, páginas 245–256.

[Katta et al. 2013] Katta, N. P., Rexford, J. e Walker, D. (2013). Incremental consistent updates. Em *ACM SIGCOMM - HotSDN'13*, Hong Kong, China. ACM.

[Levin et al. 2012] Levin, D., Wundsam, A., Heller, B., Handigol, N. e Feldmann, A. (2012). Logically centralized?: state distribution trade-offs in software defined networks. Em *Proceedings of the First workshop on Hot topics in software defined networks*, HotSDN'12, Helsinki, Finland. ACM.

[Luo et al. 2015] Luo, S., Yu, H. e Li, L. (2015). Consistency is not easy: How to use two-phase update for wildcard rules? *Communications Letters, IEEE*, 19(3):347–350.

[Mattos e Duarte 2015] Mattos, D. M. F. e Duarte, O. C. M. B. (2015). Atualizaçao reversa: Garantindo consistência de estados em redes definidas por software. Em *SBSeg 2015 - XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, Florianópolis - Brazil.

[Mattos et al. 2016a] Mattos, D. M. F., Duarte, O. C. M. B. e Pujolle, G. (2016a). A resilient distributed controller for software defined networking. Em *IEEE ICC 2016 - Next Generation Networking and Internet Symposium (ICC'16 - NGN)*, Kuala Lumpur, Malaysia.

[Mattos et al. 2016b] Mattos, D. M. F., Duarte, O. C. M. B. e Pujolle, G. (2016b). Reverse update: A consistent policy update scheme for software-defined networking. *IEEE Communications Letters*, 20(5):886–889.

[McClurg et al. 2015] McClurg, J., Hojjat, H., Cerny, P. e Foster, N. (2015). Efficient synthesis of network updates. Em *ACM SIGPLAN - PLDI*, Portland, USA. ACM.

[McGeer 2012] McGeer, R. (2012). A safe, efficient update protocol for openflow networks. Em *ACM SIGCOMM - HotSDN'12*, Helsinki, Finland. ACM.

[Monsanto et al. 2013] Monsanto, C., Reich, J., Foster, N., Rexford, J., Walker, D. et al. (2013). Composing software defined networks. Em *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI'13)*, páginas 1–13, Berkeley, CA, USA. USENIX Association.

[Prisco et al. 2000] Prisco, R. D., Lampson, B. e Lynch, N. (2000). Revisiting the Paxos algorithm. *Theoretical Computer Science*, 243(1-2):35 – 91.

[Reitblatt et al. 2012] Reitblatt, M., Foster, N., Rexford, J., Schlesinger, C. e Walker, D. (2012). Abstractions for network update. Em *Proceedings of the ACM SIGCOMM 2012*, páginas 323–334, New York, USA. ACM.