Atualização Reversa: Garantindo Consistência de Estados em Redes Definidas por Software *

Diogo Menezes Ferrazani Mattos e Otto Carlos Muniz Bandeira Duarte

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

Resumo. *A aplicação de novas políticas de segurança e a atualização de políticas já existentes em uma rede são causas comuns de instabilidade no funcionamento da rede, levando à interrupção de serviços e, até mesmo, a estados transitórios vulneráveis da configuração da rede. Durante o processo de instalação de políticas de segurança, a rede pode passar por estados inconsistentes que podem levá-la a comportamentos inesperados. Este artigo propõe um esquema consistente para o processo de atualização de políticas em Redes Definidas por Software, chamado de Atualização Reversa. O artigo prova a consistência da proposta através de um modelo formal. Assim, as principais contribuições deste artigo são: (i) a generalização do conceito de consistência por pacote no plano de dados de uma Rede Definida por Software e (ii) o esquema consistente proposto de atualização de políticas. Um simulador de Redes Definidas por Software foi desenvolvido e validado. A simulação de um cenário de uma aplicação de política de segurança, em uma topologia real de rede, mostra que a sobrecarga gerada pelo esquema proposto é muito baixa. Além disso, os resultados comprovam que o processo de Atualização Reversa proposto é bem mais rápido que a Atualização de Duas Fases proposta na literatura.*

Abstract. *Applying new security policies and updating existing policies on a network are common causes of instability in the operation of the network, leading to service disruptions and even vulnerable transitory states of the network configuration. During the process of security policy installation, the network may experience inconsistent state that can lead it to unexpected behaviors. In this paper, we propose a consistence scheme for policy updating on Software Defined Networking, the Reverse Update. We prove through a formal model that the proposal achieves consistent updates. Thus, the main contributions of the paper are: (i) the generalization of the concept of per-packet-consistency in the data plane of a Software Defined Network; and (ii) the policy update scheme, proved to be consistent. A Software Defined Network simulator was developed and validated. The simulation of an application scenario of security policy updates, in a real network topology, shows that the overhead generated by the proposed scheme is low. In addition, the results show that the proposed Reverse Update scheme is faster than the Two Phase Update proposed in the literature.*

1. Introdução

O gerenciamento e os esquemas de segurança em redes de comunicação requerem frequentes ações como, por exemplo, a atualização de rotas de tráfegos, a implantação

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

de novas políticas de segurança[1], entre outras [Monsanto et al. 2013, Kreutz et al. 2013]. As Redes Definidas por Software (*Software Defined Networking* – SDN) permitem que aplicações realizem essas operações de controle da rede através da instalação de regras de processamento de pacotes diretamente nos comutadores. Para tanto, o paradigma de SDN define a separação entre as operações de controle, logicamente centralizadas no controlador, e as operações de processamento e encaminhamento de pacotes, realizadas pelos comutadores [Mattos et al. 2015, Levin et al. 2012]. Assim, a operação mais fundamental do controlador de uma SDN é a implantação correta e consistente das políticas de processamento e encaminhamento de pacotes nos comutadores [Canini et al. 2015].

A mudança da configuração de uma Rede Definida por Software pode levar a instabilidades da rede, como a interrupção do funcionamento, a degradação do desempenho e, até mesmo, a estados vulneráveis de segurança [Reitblatt et al. 2012]. O desafio de manter a consistência das políticas durante as atualizações é presente mesmo quando os estados iniciais e finais da configuração da rede são consistentes e corretos, pois não há garantias que os estados intermediários, que ocorrem durante o processo de atualização, sejam consistentes. No cenário de SDN, a transição entre configurações da rede deve ocorrer em uma sequência de instalações e desinstalações de regras, comutador a comutador, que garanta que a rede apresente o comportamento esperado, mesmo durante o transiente de ocorrência da atualização. A hipótese de que cada aplicação em uma SDN deva ser responsável pelo seu próprio processo de atualização de políticas não é suficiente, pois as aplicações de controle da rede estão comumente sujeitas a erros quando tratam das atualizações de políticas [Canini et al. 2012]. Assim, o conceito de consistência por pacote define que um pacote atravessando a rede é processado somente por uma única configuração global consistente da rede e, portanto, nunca é processado por uma mistura de configurações [Reitblatt et al. 2012]. Logo, ao ocorrer uma atualização, cada pacote é tratado pela configuração anterior à atualização ou pela configuração posterior.

As principais propostas para a atualização de políticas em SDN baseiam-se nas ideias de atualização atômica [Perešíni et al. 2013] ou atualização em duas fases [Reitblatt et al. 2012, Canini et al. 2015]. A atualização atômica considera que todos os nós da rede são atualizados simultaneamente em uma operação atômica. No entanto, a operação de atualização dos comutadores em uma rede SDN não pode ser realizada de forma atômica e, consequentemente, os pacotes em trânsito, que estão atravessando a rede no momento da implantação atualização, podem ser processados por configurações da rede anteriores ou posteriores à atualização, sem garantias de consistência para estes pacotes em trânsito. Para assegurar a consistência foi proposta a Atualização de Duas Fases, que se baseia na marcação de uma etiqueta de versão da configuração nos pacotes e, consequentemente, no processamento dos pacotes segundo a versão que carregam. Assim, esse esquema de atualização consistente depende de um sistema de identificação de configurações no qual as regras da nova configuração sejam designadas com número de versão diferente das anteriores e, portanto, dependem da instalação das novas regras nos comutadores. Além disso, os comutadores passam a ter diferentes regras nas tabelas para o mesmo fluxo, que diferem pela versão, o que pode gerar uma grande sobrecarga nas tabelas de fluxos em função da implantação de regras mais elaboradas com campos

[1]Neste artigo, políticas de segurança referem-se às diretrizes de utilização e funcionamento da rede implementadas através de regras de processamento e encaminhamento de pacotes em uma Rede Definida por Software.

coringas [Luo et al. 2015, Fogel et al. 2015]. Já a proposta deste artigo reduz a sobrecarga, quando comparada à Atualização de Duas Fases, e provê garantias de consistência, quando comparara à atualização atômica.

Este artigo propõe a Atualização Reversa, um esquema de atualização de políticas de processamento, encaminhamento e segurança em Redes Definidas por Software. A ideia central do esquema proposto se baseia no relaxamento do conceito de consistência por pacote para executar a atualização das políticas no caminho inverso do fluxo na rede. A Atualização Reversa não introduz diferentes versões de regras nos comutadores, apenas atualiza as regras anteriores. A prova de consistência do processo de Atualização Reversa é realizada através de um modelo formal de Redes Definidas por Software. O modelo de SDN utilizado neste artigo serviu também como base para o desenvolvimento de um simulador de SDN. A simulação da implantação da Atualização Reversa em uma topologia real de rede mostrou que a sobrecarga de configuração imposta pela Atualização Reversa é próxima à de um esquema de atualização ideal. Quando a Atualização Reversa é comparada à Atualização de Duas Fases [Reitblatt et al. 2012], verifica-se que que a proposta deste artigo apresenta menor sobrecarga de configuração e, ainda, é mais imediata na implantação da atualização consistente sobre os pacotes em trânsito.

O restante do artigo está organizado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados. O modelo formal de Redes Definidas por Software é apresentado na Seção 3. O esquema de Atualização Reversa é proposto e provado na Seção 4. A Seção 5 discute o simulador desenvolvido e apresenta os resultados da simulação. A Seção 6 conclui o trabalho.

2. Trabalhos Relacionados

A proposta OF.CPP (*Consistent Packet Processing for OpenFlow*) [Perešíni et al. 2013] argumenta que o processamento de pacotes em uma SDN é passível de erros (*bugs*), pois a tomada de decisão e a implantação de regras no encaminhamento de pacotes não são atômicas, ou seja, não é uma ação única na rede e pode ser interrompida e retomada com o controlador tendo uma visão global diferente da inicial. Assim, Perešín *et al* identificam os erros mais comuns que podem ocorrer em SDNs: instalação de laços na rede, amplificação de regras e processamento inconsistente de pacotes. Nesse sentido, os autores defendem a ideia de usar conceitos de transações, tais como as de banco de dados, para o processamento dos pacotes na rede. O principal conceito defendido é o de ACID (Atomicidade, Consistência, Isolamento e Durabilidade). A consistência defendida pela proposta foca na resolução de conflitos entre transações que alteram o estado do controlador. No entanto, a resolução de conflitos entre transações não garante a consistência no tratamento dos pacotes em trânsito na rede durante a atualização, como é defendido pelo conceito de consistência por pacote.

Seguindo a linha de garantir a consistência no tratamento dos pacotes em uma SDN, Reiblatt *et al.* propõem um modelo abstrato para uma rede OpenFlow no qual é possível verificar se as propriedade de um fluxo são mantidas através dos procedimentos de atualização de políticas na rede [Reitblatt et al. 2012]. Os autores ainda implementam um protótipo do mecanismo de atualização consistente da rede que é capaz de manter as propriedades de cada fluxo. Por fim, realizam experimentos com o protótipo. A ideia central da proposta é que para manter um fluxo consistente na rede, mesmo após atualização

de políticas, cada visão global da rede deve ser associada a um número de versão. Dessa forma, ao realizar uma atualização das políticas da rede, uma nova versão do plano de controle é colocada em funcionamento e, para tanto, os pacotes devem ser marcados para diferenciar os pacotes da nova versão em relação aos pacotes da versão anterior que ainda estão em trânsito na rede. Assim, o mecanismo proposto para realizar a atualização consistente da rede é baseado em uma atualização em duas fases. A primeira fase consiste em atualizar todos os comutadores da rede, com exceção dos comutadores de ingresso do pacote na rede. Na segunda fase, são atualizados os fluxos nos comutadores de ingresso do pacote na rede. Nos comutadores de ingresso, os fluxos antigos são modificados apenas para marcarem os pacotes entrantes com o marcador da nova versão da configuração.Tal esquema de atualização consistente é proposto também para funcionar diretamente no *hardware* do comutador [Han et al. 2015]. Por outro lado, Luo *et al.* argumentam que a Atualização de Duas Fases pode gerar um crescimento exponencial no número de regras nos comutadores devido ao uso de regras coringas [Luo et al. 2015].

A proposta NICE (*No bugs In Controller Execution* [Canini et al. 2012]) define uma ferramenta para a busca de erros em aplicações OpenFlow. Diferentemente das outras propostas citadas, a proposta NICE visa verificar se há estados não alcançáveis em um programa OpenFlow ou se há violações de propriedades desejáveis. Para tanto, NICE é definido como um verificador de modelos que pode ser usado diretamente sobre a aplicação OpenFlow, já que o NICE é um conjunto de códigos em Python para verificar a assertiva do código da aplicação.

Canini *et al.* [Canini et al. 2013] argumentam que a atualização de políticas concomitantes pode levar a rede a estados inconsistentes, mesmo que seja somente durante o transiente entre a aplicação e a efetivação da política na rede. Assim, Canini *et al.* propõem uma interface transacional de atualização de políticas com semântica de "tudo ou nada". A interface proposta aplica a atualização na rede caso não haja conflitos com outras políticas. Caso contrário, a atualização é revertida e não modifica o estado da rede. A principal diferença entre a proposta e outras que tratam da consistência de políticas em SDN é que a transação de atualização da rede permite a identificação distribuída de conflitos na rede, em contraste à detecção de conflitos em um nó centralizado. A proposta aborda a composição de políticas de uma forma simples, se a política não pode ser trivialmente aceita em um comutador da rede, a instalação dessa política é totalmente rejeitada, o que mantém a consistência global do estado da rede.

Este artigo propõe o esquema de Atualização Reversa que garante a consistência da atualização de políticas ao efetivar as regras nos comutadores de uma SDN no sentido do destino para a origem dos fluxos. Esta proposta é simples e eficiente, pois evita a enorme sobrecarga de marcações de todos os pacotes com as versões das atualizações, como acontece no esquema de Atualização de Duas Fases. As seções que seguem mostram o modelo formal de Redes Definidas por Software, baseados nos trabalhos de [Reitblatt et al. 2012] e [Canini et al. 2015], e a prova da consistência da proposta.

3. O Modelo de Redes Definidas por Software

A ideia do modelo é descrever de forma detalhada a execução da rede. Assim, considera-se uma sequência de eventos observáveis us, que modificam o estado da rede N, inicial, para o estado N' após uma execução. Os eventos observáveis que compõem

us são mensagens entre controlador e comutadores que alterem o estado da rede. Como estado da rede entende-se qualquer alteração da visão global da rede ou alterações nas regras de encaminhamento de fluxos nos comutadores. Assim, a execução da rede é representada pela relação $N \xrightarrow{us} \star N'$.

A notação usada se baseia na aplicada por Reitblatt et al. [Reitblatt et al. 2012], onde $T_1 \rightarrow T_2$ denota uma função total que recebe argumentos do tipo T_1 e gera resultados do tipo T_2 e $T_1 \times T_2$ denota o conjunto de pares de elementos dos tipos T_1 e T_2. As notações simples de tuplas, por exemplo (x_1, x_2), são usadas para mostrar pares de elementos. A notação de listas com n elementos de x_1 a x_n é representada por $[x_1, \ldots, x_n]$. A notação $[\,]$ representa uma lista vazia e $xs_1 + + xs_2$ designa a concatenação das listas xs_1 e xs_2.

As estruturas básicas do modelo são: o pacote e a porta. O pacote, representado por pk, que é a unidade de transmissão de dados na rede, formado por uma sequência de `bits`. A porta, p, representa o lugar da rede onde um pacote pode estar esperando para ser processado. O modelo considera dois tipos distintos de portas. O primeiro tipo é o de portas ordinárias, que correspondem a portas dos comutadores da rede, numeradas de 1 a k. O segundo tipo de portas são as portas especiais, $Drop$ e $World$. A porta $Drop$ designa a ação de descarte de um pacote que é encaminhado para ela, enquanto a porta $World$ encaminha, ou recebe, o pacote para fora do domínio da rede considerada. Vale ressaltar que as portas especiais permitem a entrada e a saída de pacotes da rede, enquanto as portas ordinárias não criam nem destroem pacotes encaminhados.

O plano de dados considerado nesse artigo é um conjunto P de portas p da rede, assim como os enlaces são um conjunto $L \subseteq P \times P$ de enlaces direcionados. A porta de ingresso de um pacote na rede é aquela que não possui enlaces de entrada, então $\nexists j \in P : (j, i) \in L$. As demais portas são chamadas de portas internas. Todas as portas internas são conectas à porta $Drop$ e, portanto, podem descartar pacotes. Um subconjunto das portas internas é conectado à porta $World$ e, então, podem encaminhar pacotes para fora dos domínios da rede. As portas $World$ e $Drop$ não possuem enlaces de saída, então $\forall i \in \{World, Drop\}, \nexists j \in P : (i, j) \in L$. A carga de trabalho do plano de dados é representada por um conjunto Π de pacotes[2] [Canini et al. 2015].

Reitblatt *et al.* modelam a rede como um processador de pacotes capaz de encaminhar pacotes e, em alguns casos, capaz de modificar o conteúdo dos pacotes a cada salto [Reitblatt et al. 2012]. Portanto, o processamento dos pacotes é modelado como a Composição de duas funções mais simples, encaminhar um pacote no comutador e mover um pacote de ponto-a-ponto em um enlace. A execução da rede é representada por uma função do comutador S que recebe como entrada lp que representa um pacote localizado, ou seja, a tupla do pacote e a porta do comutador em que o pacote está, e retorna uma lista de pacotes localizados. O retorno indica a localização futura do pacote na rede. Para realizar a ação de descarte de pacote, a função do comutador encaminha o pacote para a porta $Drop$. A função de Topologia, denotada por T, mapeia a porta de um comutador em uma porta de outro comutador, caso ambas sejam ligadas por um enlace.

Outras definições importantes são a de traço, denotado por t, e a de fila de portas, denotada por Q. Um traço é uma lista de pacotes localizados que mantém o registro da

[2]A nomenclatura de pacotes é usada para a comunicação no plano de dados, enquanto o termo mensagens refere-se à comunicação do plano de controle.

sequência dos saltos de comutadores que o pacote atravessa na rede. Uma fila de porta mapeia portas em listas de pares de pacotes e traço. A fila de porta armazena os pacotes esperando para serem processados em cada porta na rede com o histórico de cada pacote. Assim, há outra função que reescreve a fila de portas, a função $override(Q, p{\rightarrow}l)$ que produz uma nova fila de porta Q' que mapeia p em l e para as demais portas mantém o mesmo mapeamento de Q.

A configuração da rede é representada por C que é composta por uma função de comutador S e uma função de topologia T. O estado da rede é representado por N que é um par (Q, C), contendo a fila de porta Q e a configuração C.

Reitblatt *et al.* também definem dois tipos de transições para a rede: transição por processamento de pacote e transição por atualização de política. Na transição por processamento de pacote, um pacote é retirado da fila de uma porta, processado usando uma função de comutador S sobre uma função de topologia T e os novos pacotes gerados, já que ao processar um pacote pode-se gerar mais de um pacote na saída, são inseridos nas filas das portas dos comutadores correspondentes a saída da função de topologia [Reitblatt et al. 2012]. O estado da rede é caracterizado por uma fila de porta Q_i e uma função de comutador S_i associados a cada porta i. A fila de porta Q_i é uma sequência de pacotes que estão esperando para serem processados pela porta i. A função de comutador é um mapeamento $S_i : \Pi \rightarrow \Pi \times P$, uma função de pacotes em tuplas de pacote e porta, que define a maneira como os pacotes na fila de porta Q_i são processados. Quando um pacote pk é retirado da fila Q_i, um *pacote localizado*, a tupla (pk', j), é computado e o pacote pk' é colocado na fila Q_j.

Representa-se a função de comutador na porta i, S_i, como uma coleção de regras. Uma regra, r, é um mapeamento parcial $r : \Pi \rightarrow \Pi \times P$ que para cada pacote pk no domínio $dom(r)$ gera um novo pacote localizado $r(pk) = (pk', j)$, que coloca pk' na fila Q_j, desde que $(i, j) \in L$. Assume-se que só parte do pacote pk pode ser modificada por uma regra, em especial o campo do cabeçalho que identifica a qual regra o pacote pertence, referenciado como *tag*.

Por sua vez, a transição de atualização da rede corresponde à mudança da função de um comutador. Enquanto a função S define o funcionamento de todos os comutadores distribuídos em rede, uma atualização u age somente em uma parte dos comutadores da rede. Assim, a função $override(S, u)$ gera uma nova função S' que modifica somente como u age sobre os pacotes localizados que estejam no domínio da atualização u. Nos demais pacotes, S' atua como S. Após a atualização u, há a mudança do estado da rede de $(Q, (S, T))$ para $(Q, (S', T))$. Assim, tem-se que a nova função de comutador é dada por:

$$override(S, u) = S' sendo\ S'(p, pk) = \begin{cases} u(p, pk)\ se\ (p, pk) \in dom(u) \\ S(p, pk),\ \text{caso contrário.} \end{cases} \quad (1)$$

Vale lembrar que o modelo considera que um número arbitrário de passos pode ser executado sobre a rede, partindo de um estado inicial, em que todas as filas das portas estão vazias. A execução da rede representada por $N \stackrel{us}{\rightarrow} \star N'$ considera que todas as atualizações contidas em us são aplicadas na rede através da concatenação das transições

de atualização, em ordem. A execução considera ainda um observador onisciente. Durante a execução da rede, um traço t é válido se e somente se existe um estado inicial Q, tal que $(Q, C) \rightarrow \star(Q', C)$ e t aparece em Q', ou seja, há uma mudança nas filas da rede, mantendo a mesma configuração da rede, e o traço t é um encaminhamento possível e, portanto, está na história do novo estado das filas de porta Q'. Com base neste modelo de SDN proposto por Reitblatt *et al.*, é possível descrever propriedades da rede, como encaminhamento sem laços e a marcação correta de VLANs, em relação ao traço t de um pacote. Contudo, propriedades relativas ao tempo de processamento dos pacotes, como a garantia de qualidade de serviço ou o controle de congestionamento da rede, não são possíveis de se descrever com esse modelo. Sendo assim, o modelo é adequado para a formalização e verificação da propriedade de consistência no controle e atualização de políticas em uma Rede Definida por Software. Para tanto, considera-se que Q satisfaz uma propriedade P da rede se todos os traços t em Q aparecem no conjunto P:

$$\forall t \in Q \rightarrow t \in P, \tag{2}$$

onde P é uma propriedade da rede. Dessa forma, para verificar a consistência do controle em uma Rede Definida por Software, verificam-se duas propriedades da rede. A primeira propriedade é a consistência no tratamento de um fluxo enquanto o fluxo é ativo na rede. A segunda é a consistência da atualização entre controladores.

A consistência na atualização é uma propriedade que deve ser garantida em todos os cenários. A dificuldade em manter a consistência nas atualizações está no fato de que o controle de uma rede é um exemplo de programação concorrente e, portanto, torna-se mais difícil, pois se deve considerar a interação entre todos os eventos que acontecem na rede e as combinações de eventos. Assim, uma das formas de se pensar a atualização da rede é a atualização atômica, ou seja, a atualização das regras de encaminhamento de pacotes deve ocorrer sem que haja sua interrupção ou sem que ocorra a execução de outras atualizações concorrentes [Perešíni et al. 2013]. Contudo, a atualização atômica não é possível, pois durante o processo de atualização a rede pode passar por estados inconsistentes e os pacotes em trânsito podem ser processados em uma parte inicial do caminho na rede pela configuração antiga e, em outra parte, pela nova configuração já atualizada e voltar a ser processado pela configuração antiga no final do caminho do pacote na rede.

Na definição de consistência por pacote apresentada por Reitblatt *et al.*, introduz-se a relação de equivalência, denotado por $\sim$, nos traços. Define-se dois traços equivalentes como aqueles traços em que todos os pacotes, em ambos os traços, apresentam as mesmas características. A ideia de atualização consistente, portanto, prevê que os traços gerados pela rede no momento de uma atualização sejam equivalentes a traços gerados pela configuração inicial antes da atualização ou a traços gerado pela configuração final depois da atualização.

Definição: Atualização $\sim$-consistente por pacote. $\sim$ é a relação de equivalência de traços. Uma sequência de atualização é consistente se e somente se para todos:

- estado inicial Q,
- execuções $(Q, C_1) \stackrel{us}{\rightarrow} \star(Q', C_2)$,
- e traços $t \in Q'$,

existe

- um estado inicial Q_i,
- e uma execução $(Q_i, C_1) \rightarrow \star(Q'', C_1)$ ou uma execução $(Q_i, C_2) \rightarrow \star(Q'', C_2)$,

em que Q'' contém t', em que $t' \sim t$. Um aspecto importante dessa definição é que uma atualização consistente por pacotes preserva as propriedades do traço. Isso ocorre, pois como o traço t apresenta a propriedade P e é equivalente ao traço t' que pertence a Q'', seja em C_1 seja em C_2, as propriedade de t também estão presentes em t'. Assim, para todas as relações de equivalências de traço, denotadas por $\sim$, se us é uma sequência de atualização $\sim$-consistente de C_1 para C_2, us também preserva as propriedades de C_1 para C_2 e, portanto, é uma atualização consistente de C_1 para C_2 [Reitblatt et al. 2012].

Reitblatt *et al.* consideram ainda duas outras definições importantes: atualização única e atualização não observável.

Definição: Atualização Única. Seja $C_1 = (S, T)$ a configuração inicial da rede, seja $C_2 = (S[u_1, \ldots, u_k], T)$ a nova configuração da rede e seja $us = [u_1, \ldots, u_k]$ uma sequência de atualizações, em que os domínios das funções de atualização u_1 a u_k sejam mutuamente excludentes. Se para todos:

- estados iniciais Q;
- execuções $(Q, C_1) \xrightarrow{us} \star(Q', C_2)$

e não existe um traço t em Q' tal que

- t contenha elementos de traço distintos (p_1, pk_1) e (p_2, pk_2);
- e ambos, (p_1, pk_1) e (p_2, pk_2), Pertençam ao domínio de uma das funções de atualização $[u_1, \ldots, u_k]$,

então us é uma atualização única de C_1 para C_2. Nesse caso, a atualização é $\sim$-consistente por pacote [Reitblatt et al. 2012].

Vale ressaltar que uma atualização única pode ser entendida como uma atualização que não gera traços com estados intermediários na rede, isto é, os pacotes só são tratados ou pelo estado inicial de configuração (C_1) ou pelo estado final (C_2).

Definição: Atualização Não-Observável é uma atualização que não altera o conjunto de traços gerados pela rede. Assim, seja $C_1 = (S, T)$ a configuração inicial da rede, $us = [u_1, .., u_k]$ uma sequência de atualizações e $C_2 = (S[u_1, \ldots, u_k], T)$ a configuração final da rede. Se, para todos

- estados iniciais Q;
- execuções $(Q, C_1) \xrightarrow{us} \star(Q', C_2)$;
- traços $t \in Q'$,

existe

- um estado inicial Q_i;
- e uma execução $(Q_i, C_1) \rightarrow \star(Q'', C_1)$;

tal que o traço t pertence a Q'', então us é uma sequência de atualização não-observável de C_1 para C_2. Vale ressaltar que uma sequência de atualização us é consistente por pacotes se for uma atualização não observável [Reitblatt et al. 2012].

Dado que uma sequência de atualização pode ser um exemplo de uma atualização única ou de uma atualização não-observável, Reitblatt *et al.* provam ainda que se us_1 é uma atualização não-observável de C_1 para C_2 e us_2 é uma atualização única de C_2 para C_3, então a concateção $us_1 + +us_2$ é uma atualização consistente por pacote de C_1 para C_3.

Baseado na ideia de que uma sequência de atualização pode ser considerada como a concatenação de duas atualizações, que por si só são consistentes, Reitblatt *et al.* definem o conceito de Atualização de Duas Fases. A Atualização de Duas Fases cria o conceito de versão da configuração da rede. Assim, a versão passa a ser uma propriedade do traço da rede. A diferenciação de um traço de uma versão para o traço de outra versão é através de uma marca de versão (*version tag*). Assim, a configuração C é uma versão-n da configuração da rede se $C = (S, T)$ e S modifica os pacotes processados em qualquer porta de ingresso de pacotes na rede, $p_i n$, alterando os pacotes que passam por $p_i n$ para terem o campo de versão alterado para n. O marcador de versão não é alterado em nenhum outro ponto da rede, senão a porta de ingresso. Duas configurações C e C' coincidem internamente na versão n sempre que $C = (S, T)$ e $C' = (S', T)$, para todas as portas internas p e todos os pacotes pk com versão configurada para n, tem-se que $S(p, pk) = S'(p, pk)$. Uma atualização u é dita um refinamento de S, se para todos os pacotes localizados lp, tem-se que $u(lp) = S(lp)$.

Definição: Atualização de Duas Fases. Seja $C_1 = (S, T)$ a configuração da versão 1 e $C_2 = (S', T)$ a configuração da versão 2. Assume-se que C_1 e C_2 são coincidentes internamente para pacotes da versão 1. Seja $us = [u_1^i, \ldots, u_m^i, u_1^e, \ldots, u_m^e]$ uma sequência de atualização que:

- $S' = override(S, us)$,
- cada u_j^i e u_k^e é um refinamento de S',
- p é uma porta interna, para cada (p, pk) no domínio de u_j^i,
- e p é uma porta de ingresso, para cada (p, pk) no domínio de u_k^e,

então, us é uma Atualização de Duas Fases de C_1 para C_2. Pela composição das duas atualizações, uma única e a outra não observável, percebe-se também que us é também uma atualização consistente por pacote. A prova completa é mostrada por Reitblatt *et al.* [Reitblatt et al. 2012, Canini et al. 2015].

Contudo, a implementação na prática da Atualização de Duas Fases exige a marcação dos pacotes com a versão da configuração. Reitblatt *et al.*, Canini *et al.* e Pereŝíni *et al.* argumentam que uma implementação válida para a marcação de versão é o uso de marcadores de VLAN [Reitblatt et al. 2012, Canini et al. 2015, Mattos et al. 2013]. A marcação dos pacotes, no entanto, introduz um atraso, gera uma sobrecarga de controle para a verificação dos marcadores válidos e exige a marcação dos pacotes. Assim, um pacote com um dado marcador de VLAN não pode ser encaminhado na rede, caso o mecanismo de consistência de Atualização de Duas Fases seja implantado, já que a semântica de VLAN é sobrecarregada para indicar a versão da configuração.

4. O Esquema Proposto de Atualização Reversa

A proposta deste artigo, a Atualização Reversa, é um esquema de atualização de políticas em Redes Definidas por Software que garante a consistência da aplicação das

políticas. Sua vantagem em relação ao mecanismo de Atualização de Duas Fases é a baixa sobrecarga requerida, pois não depende da marcação de pacotes.

Pisa *et al.* propõem um algoritmo para migração de fluxos em uma rede OpenFlow [Pisa et al. 2010]. O algoritmo proposto baseia-se em atualizar o caminho de um fluxo na rede, refazendo a configuração do fluxo no sentido contrário do fluxo. Este mesmo mecanismo pode ser usado para a atualização da configuração da rede, garantindo a consistência da configuração. Logo, este trabalho propõe uma nova forma de atualização: a Atualização Reversa. Seguem a definição formal da nova forma de Atualização Reversa proposta e a prova de que ela é consistente por pacotes.

Definição: Atualização Reversa. Sejam $dom(u_i)$ o domínio da atualização u_i e $us = [u_1, \ldots, u_k]$ uma sequência de atualizações de comutadores, ordenada para ser efetuada no sentido inverso do caminho do pacote na rede. Assim, define-se que a ordem das atualizações é dada por:

$\forall (p, pk) \in dom(u_i),\ \nexists (p, pk) \in dom(u_j)$, para $i < j \leq k$,

de forma que um pacote que segue no sentido da origem para o destino nunca possa passar por um comutador que ainda esteja em um estado de configuração anterior, pois as atualizações estão sendo efetuadas no sentido invertido do destino para a origem. Assim, para todo:

- estados iniciais Q;
- execuções $(Q, C_1) \xrightarrow{us} \star(Q', C_2)$;

os pacotes que são processados por C_i, não são mais processados por C_j, se $j < i$.

Tendo em vista que o mecanismo proposto de atualização reversa gera uma atualização consistente por pacote, tem-se o seguinte teorema.

Teorema: Se uma sequência de atualização us é uma Atualização Reversa, então us é consistente por pacote.

Prova: A prova desse teorema segue o mecanismo de indução em us. A princípio, considera-se a sequência de atualização $us = [u_1, \ldots, u_i, \ldots, u_k]$. A indução é feita sobre k.
Caso Base para $k = 1$: No caso base, $us = u_1$, pois só há uma atualização a ser realizada. Considerando que essa atualização, por si só, é uma atualização consistente por pacote, tem-se que para toda execução $(Q, C_1) \rightarrow \star(Q', C_2)$, que gera o traço t, ocorre $(Q, C_1) \rightarrow \star(Q'', C_1)$ ou $(Q, C_2) \rightarrow \star(Q'', C_2)$, em que Q'' contém o traço t' que é $\sim$-consistente com o traço t. Portanto, de acordo com a definição de atualização consistente por pacote, o caso base é consistente por pacote.

Hipótese Indutiva para $k = i$: Assume-se que a sequência de atualização $us = [u_1, \ldots, u_i]$ é uma Atualização Reversa e que é consistente por pacote.
Passo Indutivo para $k = i + 1$: Para $k = i + 1$, assume-se que $us = [u_1, \ldots, u_i, u_{i+1}]$. Assim, pode-se definir que $us_i = [u_1, \ldots, u_i]$. Considera-se também que:

$Dom(us_i) = \bigcup_{j \leq i} dom(u_j)$ e $us = us_i + + u_{i+1}$.
Por hipótese, us_i é uma Atualização Reversa consistente por pacote. Então, basta provar

que a concatenação com u_{i+1} é também uma Atualização Reversa que mantém a consistência por pacote. Para tanto, verifica-se que:

- Se $(p, pk) \in Dom(us_i)$, então $(p, pk) \notin dom(u_{i+1})$, pela definição de Atualização Reversa,
- t contém (p_1, pk_1) e (p_2, pk_2),
- Não existe $(p, pk) \in Dom(us_i)$ e, ao mesmo tempo, $(p, pk) \in dom(u_{i+1})$,

então, para todo:

- estado inicial Q_i,
- execução $(Q_i, C_i) \overset{u_{i+1}}{\rightarrow} \star(Q_{i+1}, C_{i+1})$

a execução é uma Atualização Única e, portanto, é consistente por pacote. Pelo teorema da Concatenção [Reitblatt et al. 2012], como us_i é consistente e a atualização u_{i+1} também é consistente, a sequência de atualização us é uma atualização consistente por pacote, dado que é uma Atualização Reversa.

Dessa forma, prova-se que não é necessária a marcação de pacotes para se garantir a consistência por pacote para atualização de políticas na rede, no caso de se utilizar o procedimento de Atualização Reversa. Contudo, a atualização de políticas foi verificada em um cenário em que não há a definição de controle distribuído. Assim, na próxima seção, verifica-se a consistência do plano de controle entre controladores distribuídos.

Algoritmo 1: Algoritmo de Atualização Reversa. As tabelas dos comutadores são atualizadas na sequência inversa do sentido do fluxo de pacotes.

```
G = Grafo(Topologia da rede)
Fluxo = Conjunto(Fluxos da rede)
Controlador.atualizaPoliticas()
for fluxo ∈ Fluxos do
    src = fluxo.origem
    dst = fluxo.destino
    caminho = calcula_caminho(G, src, dst)
    caminho.reverso()
    for comutador ∈ caminho do
        comutador.atualiza_regra(fluxo)
    end
end
```

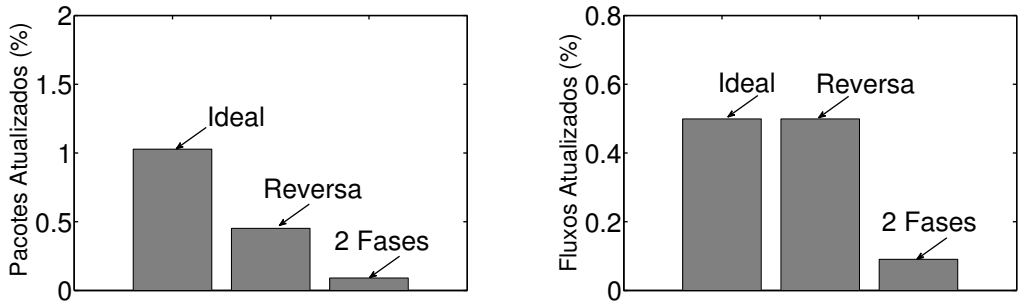
Uma implementação do esquema de Atualização Reversa, proposto e provado ser consistente, é formalizado no Algoritmo 1. Para cada fluxo na rede, verifica-se o caminho do fluxo na rede, reverte o caminho e, nessa ordem, aplica-se a atualização da regra em cada comutador do caminho revertido.

5. Resultados Experimentais

A avaliação do esquema proposto de Atualização Reversa foi realizada através da simulação de uma Rede Definida por Software. Para tanto, desenvolveu-se um simulador de eventos discretos que considera o modelo de redes apresentado nesse artigo e

proposto por [Reitblatt et al. 2012]. O simulador foi escrito em linguagem Python e implementa os elementos do modelo de SDN através de classes e seus relacionamentos. A avaliação do esquema proposto considera a topologia real de rede da RNP, no Brasil, com 31 nós. O grafo da topologia usada foi obtido no *The Internet Topology Zoo*[3]. A chegada de novos fluxos no simulador segue o modelo de intervalo de chegada entre fluxos dado por uma distribuição *log-normal* com média 7 ($\mu = 7$) e desvio padrão igual a 2 ($\sigma = 2$) [Ferraz et al. 2014]. A chegada de novos fluxos acontece durante 900 passos de simulação e a simulação só acaba quando as filas de porta de todos os comutadores já estão vazias. Cada fluxo é modelo para durar por 50 passos de simulação e os nós de origens e destino de cada fluxo são aleatórios.

Simulação da Atualização Reversa (`Reversa`) foi comparada com a Atualização de Duas Fases [Reitblatt et al. 2012] (`2 Fases`) e com uma Atualização Ideal (`Ideal`). A Atualização Ideal é somente factível através de simulação. A ideia da Atualização Ideal é que durante a atualização das políticas e regras de encaminhamento, nenhum pacote é encaminhado na rede e, portanto, não há estados inconsistentes. A Atualização Ideal é semelhante a uma atualização atômica [Perešíni et al. 2013]. Contudo, a atualização atômica pode incorrer na inconsistência de pacotes que estão atravessando a rede durante a sua execução. Essa hipótese não ocorre na simulação da Atualização Ideal, pois durante a execução da atualização, todo o encaminhamento e processamento de pacotes da rede são suspensos. A Atualização Ideal é usada como base para comparação dos demais esquemas de atualização de políticas. Vale ressaltar que na simulação de todos os esquemas de atualização foram realizados eventos de atualização de políticas a cada 300 passos de simulação e a chegada de novos fluxos foi a mesma para todos os esquemas.



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

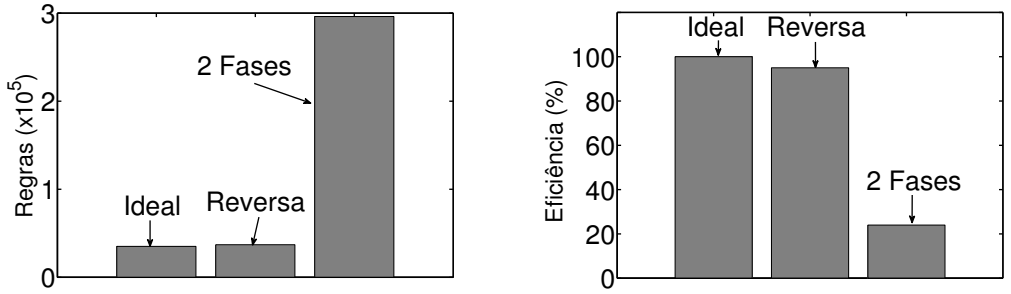
(b) Percentual de fluxos que são afetados por atualizações da rede.

Figura 1. Efeito das atualizações nos pacotes e fluxos encaminhados na rede. A Atualização de Duas Fases é a que deixa de atualizar o maior número de fluxos devido ao atraso para a implantação da segunda fase.

O primeiro experimento avalia o percentual de pacotes que são processados por duas configurações diferentes da rede durante a sua travessia. O esquema de Atualização de Duas Fases apresenta alguns pacotes que ainda são processados por duas diferentes configurações da rede, o que é um comportamento inexperado. Tal comportamento se explica pelo fato de que, quando um pacote é o primeiro do fluxo, o pacote é encaminhado para o controlador a cada comutador em que não há uma entrada na tabela de

[3]http://www.topology-zoo.org/.

fluxos correspondente ao novo fluxo. Sendo assim, quando o primeiro pacote de um fluxo chega ao controlador já atualizado, a nova entrada na tabela de fluxos é calculada baseada na configuração da rede atualizada. Esse comportamento é refletido pelo número de pacotes tratados por duas configurações de rede, mesmo no caso da Atualização de Duas Fases [Reitblatt et al. 2012]. Com o relaxamento do conceito proposto nesse artigo, basta garantir que o pacote, após ser encaminhado por uma configuração mais atual, não volte a ser encaminhado por uma configuração anterior. Assim, percebe-se que a Atualização Reversa possui um percentual de pacotes afetados pela atualização muito próxima da Atualização Ideal, mostrado na Figura 1(a), indicando um menor tempo de reação à atualização quando comparada com a Atualização de Duas Fases. Avaliou-se, também, o número de fluxos que são afetados pelas atualizações. Percebe-se que os esquemas de Atualização Reversa e Atualização Ideal apresentaram o mesmo número de fluxos atualizados, Figura 1(b), enquanto, por causa o atraso na atualização introduzido pela Atualização de Duas Fases, o número de fluxos que são tratados pela configuração mais atual é menor que dos outros dois esquemas.



(a) Número de Regras na rede após a simulação.

(b) Eficiência comparada de cada esquema de atualização.

Figura 2. A Atualização Reversa introduz menor sobrecarga de regras geradas que a Atualização de Duas Fases. A Atualização Reversa apresenta maior eficiência que a Atualização de Duas Fases.

Outro ponto importante da avaliação é o número de regras instaladas nos comutadores. A Figura 2(a) compara o número de regras instaladas por cada esquema durante toda a execução da simulação. É possível perceber que o número de regras instaladas pela Atualização de Duas Fases é praticamente o dobro dos outros esquemas de atualização. Isso ocorre devido à adição de novas regras nas portas do núcleo da rede, enquanto a Atualização Reversa apenas atualiza as regras já existentes, assim como a Atualização Ideal. Por fim, avaliou-se a eficiência de cada esquema de atualização. A eficiência é definida neste artigo como o percentual de pacotes encaminhados pelos esquemas de atualização que estão em correspondência àqueles encaminhados pela Atualização Ideal, durante a ocorrência de atualizações. Nesse sentido, verifica-se que a Atualização Reversa alcança uma eficiência de 94%, enquanto a Atualização de Duas Fases apresenta eficiência de 24%, mostrado na Figura 2(b).

6. Conclusão

Esse artigo propôs a Atualização Reversa. O esquema proposto efetua a atualização de políticas dos comutadores de uma Rede Definida por Software no sentido

inverso do caminho dos pacotes na rede. Prova-se que se a atualização de políticas é consistente por pacote e que as propriedades dos fluxos são mantidas, porque a atualização é realizada no sentido inverso do caminho em que os pacotes atravessam a rede. Vale ainda ressaltar que esse esquema é bem simples e possui a vantagem de não exigir a marcação de pacotes o que reduz enormemente a sobrecarga de processamento e o reduz o número de regras instaladas na rede. A simulação da aplicação do esquema em uma SDN mostrou que a sobrecarga de configuração é próxima à ideal e que a Atualização Reversa atua de maneira mais imediata que a Atualização de Duas Fases, apresentando um eficiência de 94%, quando comparada à Atualização Ideal.

7. Referências

[Canini et al. 2013] Canini, M., Kuznetsov, P., Levin, D. e Schmid, S. (2013). Software transactional networking: Concurrent and consistent policy composition. Em *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN'13, páginas 1–6, New York, NY, USA. ACM.

[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 34th Annual IEEE International Conference on Computer Communications (INFOCOM 2015)*.

[Canini et al. 2012] Canini, M., Venzano, D., Perešíni, P., Kostić, D. e Rexford, J. (2012). A nice way to test openflow applications. Em *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, páginas 10–10, Berkeley, CA, USA. USENIX Association.

[Ferraz et al. 2014] Ferraz, L. H. G., Mattos, D. M. F. e Duarte, O. C. M. B. (2014). A two-phase multipathing scheme based on genetic algorithm for data center networking. Em *Global Communications Conference (GLOBECOM), 2014 IEEE*, páginas 2270–2275.

[Fogel et al. 2015] Fogel, A., Fung, S., Pedrosa, L., Walraed-Sullivan, M., Govindan, R., Mahajan, R. e Millstein, T. (2015). A general approach to network configuration analysis. Em *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI'15)*, Berkeley, CA, USA. USENIX Association.

[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.

[Kreutz et al. 2013] Kreutz, D., Ramos, F. M. e Verissimo, P. (2013). Towards secure and dependable software-defined networks. Em *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN'13, páginas 55–60, New York, NY, USA. 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 et al. 2013] Mattos, D., Ferraz, L. e Duarte, O. C. M. B. (2013). Um mecanismo para isolamento seguro de redes virtuais usando a abordagem híbrida xen e openflow. Em *SBSeg 2013 - XIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, Manaus - Brazil.

[Mattos et al. 2015] Mattos, D. M. F., Andreoni Lopez, M., Ferraz, L. H. G. e Duarte, O. C. M. B. (2015). Controlador resiliente com distribuição eficiente para redes definidas por software. Em *XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2015*.

[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.

[Perešíni et al. 2013] Perešíni, P., Kuzniar, M., Vasić, N., Canini, M. e Kostiū, D. (2013). Of.cpp: Consistent packet processing for openflow. Em *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN'13, páginas 97–102, New York, NY, USA. ACM.

[Pisa et al. 2010] Pisa, P., Fernandes, N., Carvalho, H., Moreira, M., Campista, M., Costa, L. e Duarte, O. (2010). Openflow and xen-based virtual network migration. Em Pont, A., Pujolle, G. e Raghavan, S., editors, *Communications: Wireless in Developing Countries and Networks of the Future*, volume 327 of *IFIP Advances in Information and Communication Technology*, páginas 170–181. Springer Boston.

[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 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM'12, páginas 323–334, New York, NY, USA. ACM.