Segurança em Redes Definidas por Software:
Autenticação, Controle de Acesso e Consistência com
Plano de Controle Eficientemente Distribuído

Diogo Menezes Ferrazani Mattos

PEE / COPPE / UFRJ

Tese submetida para a obtenção do título de
Doutor em Engenheira Elétrica
ao Programa de Engenharia Elétrica/COPPE/UFRJ

SEGURANÇA EM REDES DEFINIDAS POR SOFTWARE: AUTENTICAÇÃO, CONTROLE DE ACESSO E CONSISTÊNCIA COM PLANO DE CONTROLE EFICIENTEMENTE DISTRIBUÍDO

Diogo Menezes Ferrazani Mattos

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia Elétrica.

Orientadores: Otto Carlos Muniz Bandeira Duarte
Guy Pujolle

Rio de Janeiro
Janeiro de 2017

SEGURANÇA EM REDES DEFINIDAS POR SOFTWARE: AUTENTICAÇÃO, CONTROLE DE ACESSO E CONSISTÊNCIA COM PLANO DE CONTROLE EFICIENTEMENTE DISTRIBUÍDO

Diogo Menezes Ferrazani Mattos

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

Prof. Otto Carlos Muniz Bandeira Duarte, Dr.Ing.

Prof. Artur Ziviani, Dr.

Prof. Eduardo Coelho Cerqueira, D.Sc.

Prof. Igor Monteiro Moraes, D.Sc.

Prof. Ítalo Fernando Scotá Cunha, Dr.

Prof. Miguel Elias Mitre Campista, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
JANEIRO DE 2017

Mattos, Diogo Menezes Ferrazani

Segurança em Redes Definidas por Software: Autenticação, Controle de Acesso e Consistência com Plano de Controle Eficientemente Distribuído/Diogo Menezes Ferrazani Mattos. – Rio de Janeiro: UFRJ/COPPE, 2017.

XII, 106 p.: il.; 29, 7cm.

Orientadores: Otto Carlos Muniz Bandeira Duarte
Guy Pujolle

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2017.

Referências Bibliográficas: p. 96 – 106.

1. Virtualização de Redes. 2. Redes Definidas por Software. 3. Computação em Nuvens. 4. Consistência. 5. Internet do Futuro. 6. Qualidade de Serviço. 7. Xen. 8. OpenFlow. I. Duarte, Otto Carlos Muniz Bandeira *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

À minha família.

Agradecimentos

Agradeço a Deus pela oportunidade de finalizar mais um ciclo de formação e obter o grau de doutor. Agradeço à minha família que sempre esteve ao meu lado, por todo carinho e compreensão. Em especial, agradeço aos meus pais, pelo apoio que me dão em todos os momentos e por sempre me motivarem a seguir em frente.

Agradeço a todos os amigos que fiz no Grupo de Teleinformática e Automação, pois sempre contribuíram positivamente para a conclusão deste trabalho. Agradeço a Lyno Ferraz, Martin Andreoni e Dianne Medeiros pela amizade e companheirismo durante o período do doutorado e, em especial, pela participação ativa na minha experiência do doutorado no exterior, sempre me motivando e ajudando em todo o possível. Agradeço também a todos os amigos do Laboratoire d'Informatique de Paris 6 (LIP6/UPMC) que me acolheram, contribuíram para minha formação e tornaram minha estadia no exterior uma experiência única e muito prazerosa.

Agradeço, ainda, a todos os professores que participaram da minha formação. Em especial, agradeço aos meus orientadores, professor Otto Carlos Duarte, por todos os conselhos, dedicação e principalmente paciência, durante a orientação desde a graduação, e o professor Guy Pujolle, pela recepção no laboratório em Paris e por todos os conselhos e oportunidades concedidas durante o período do estágio no exterior. Gostaria de agradecer também aos professores Luís Henrique, Miguel, Aloysio e Pedro do GTA/UFRJ, ao professor Igor da UFF e aos professores Marcelo Amorim, Stefano Secci e Thi-Mai-Trang Nguyen da UPMC.

Agradeço aos professores Artur Ziviani, Eduardo Coelho Cerqueira, Igor Monteiro Moraes, Ítalo Fernando Scotá Cunha e Miguel Elias Mitre Campista pelas sugestões na qualificação e pela participação na banca examinadora desta tese.

Agradeço aos funcionários do Programa de Engenharia Elétrica da COPPE/UFRJ, Maurício, Daniele e Roberto pela presteza no atendimento na secretaria do Programa.

Agradeço a todos que participaram de forma direta ou indireta da minha formação profissional. Por fim, agradeço a CNPq, CAPES, FAPERJ, FINEP, FUNTTEL e UOL pelo financiamento deste trabalho.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

SEGURANÇA EM REDES DEFINIDAS POR SOFTWARE: AUTENTICAÇÃO, CONTROLE DE ACESSO E CONSISTÊNCIA COM PLANO DE CONTROLE EFICIENTEMENTE DISTRIBUÍDO

Diogo Menezes Ferrazani Mattos

Janeiro/2017

Orientadores: Otto Carlos Muniz Bandeira Duarte
Guy Pujolle

Programa: Engenharia Elétrica

A distribuição do controle em Redes Definidas por Software melhora a segurança, o desempenho e a escalabilidade da rede. O controle distribuído introduz desafios de consistência na visão global da rede. Este trabalho apresenta as principais ameaças de segurança às redes definidas por *software*, propõe um mecanismo de autenticação e controle de acesso de estações finais baseado na credencial da estação, propõe uma eficiente arquitetura de distribuição de controle com otimização da localização de controladores na rede e, também, propõe esquemas consistentes para o processo de atualização de políticas em redes com controle centralizado ou distribuído. Um protótipo do mecanismo de autenticação foi implementado, sobre o controlador POX, e os resultados da avaliação mostram que a proposta bloqueia o acesso de estações não autorizadas, mesmo no cenário em que uma estação autenticada tem a sua permissão de acesso revogada. O protótipo de controlador distribuído foi implementado e a otimização da resiliência foi analisada em diversas topologias reais. Os resultados da avaliação da localização de controladores mostram que a conectividade é mantida, mesmo em cenários em que há alta taxa de falhas em nós da rede. Comprova-se ainda que a otimização da localização reduz a latência média entre controladores. Um simulador de Redes Definidas por Software foi desenvolvido e validado. A simulação de um cenário de uma aplicação de políticas de segurança, em uma topologia real de rede, mostra que a sobrecarga gerada pelos esquemas propostos é muito baixa. Através de verificação formal, mostra-se que o protocolo de consistência proposto garante uma ordem global para todas as atualizações de políticas e que compõe corretamente todas as políticas instaladas na rede.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

SECURITY ON SOFTWARE DEFINED NETWORKING: AUTHENTICATION, ACCESS CONTROL AND CONSISTENCY ON AN EFFICIENTLY DISTRIBUTED CONTROL PLANE

Diogo Menezes Ferrazani Mattos

January/2017

Advisors: Otto Carlos Muniz Bandeira Duarte
Guy Pujolle

Department: Electrical Engineering

The control distribution in Software-Defined Networks improves the security, performance and scalability of the network. However, distributed control introduces consistency challenges into the network global view. In this work, we present the main security threats to the software-defined networks, we propose a host authentication and access control mechanism based on host credentials, we propose an efficient controller-distribution architecture, and we also propose consistent schemes for the policy updates on centralized or on distributed control networks. An authentication-mechanism prototype was implemented over POX, an OpenFlow controller. The results of the authentication prototype show that the proposal blocks access to unauthorized hosts, even in the scenario where a host has its access permission revoked. The distributed controller prototype was implemented and the placement heuristic was analyzed in several real topologies. The results of the controller placement evaluation show that network connectivity is maintained in scenarios where there is a high rate of network node failure. It is also proven that placement optimization reduces the average latency between controllers. A simulator for Software-Defined Networking has been developed and validated. The simulations of security policy enforcement in real network topologies show that the overhead generated by the proposed schemes is very low. Through formal verification, it is shown that the proposed consistency protocol guarantees a global order for all policy updates and that correctly composes all policies installed on the network.

Sumário

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução

Prover segurança em redes é uma necessidade crescente em redes empresariais, em redes de centro de dados para computação em nuvem e em redes que constituem infraestruturas críticas como as redes elétricas inteligentes [1]. As principais dificuldades para garantir um alto nível de segurança nas redes [2–5] são a variedade de equipamentos de redes, como comutadores, roteadores, *middleboxes* [6], entre outros; e o fato de as estações finais que se conectam à rede nem sempre serem confiáveis e, até mesmo, poderem apresentar diversas vulnerabilidades. Assim, a implantação de políticas de segurança requer do operador da rede configurações manuais dos equipamentos de acordo com o padrão e as funcionalidades de cada um. Além disso, as estações finais também devem ser autenticadas para garantir o acesso à rede somente às estações que apresentam credenciais válidas e autorizadas.

O paradigma de Redes Definidas por *Software* (*Software Defined Networks* – SDN) desacopla o controle do encaminhamento de dados, oferecendo alta programabilidade do controle e uma visão global da rede. A adoção desta tecnologia permite desenvolver, de forma logicamente centralizada, políticas integradas de segurança [7] e, assim, facilitar a solução de problemas complexos de segurança em rede [8]. O OpenFlow [9] é a implementação de maior sucesso de uma rede definida por *software*. O controlador OpenFlow, implementando em *software*, executa as funções do plano de controle. O plano de encaminhamento é executado por comutadores de alto desempenho compatíveis com o OpenFlow. Contudo, esse novo paradigma apresenta algumas limitações quanto à segurança da rede, pois um componente com comportamento malicioso pode comprometer o funcionamento de toda a rede, realizando, por exemplo, um ataque de negação de serviço no controlador. Dessa forma, a distribuição do controle da rede e o controle de acesso são necessários para garantir maior segurança. Tanto a autenticação das estações que têm acesso à rede, quanto o nível de privilégio atribuído a cada estação são também essenciais para garantir a segurança da rede definida por *software*.

As Redes Definidas por *Software* advogam pela centralização lógica do plano de

controle da rede [10, 11]. A centralização do controle e a visão global consistente da rede permitem que operadores inovem e implementem novos serviços no núcleo da rede de forma ágil. No entanto, a centralização física do controle em Redes Definidas por *Software* implica desafios para a segurança, o desempenho e a escalabilidade da rede [10]. A escalabilidade é prejudicada tanto no diâmetro da rede, comutadores podem não estar próximos de um controlador, assim como no número de comutadores na rede, já que o número de controladores pode não ser suficiente para responder em tempo hábil às requisições de todos comutadores. A centralização física do controle gera um ponto único de falha. O controlador pode ser alvo de ataques de negação de serviço (*Denial of Service* - DoS) [12, 13]. Os ataques de DoS podem exaurir tanto recursos de processamento, exigindo o atendimento a uma quantidade anômala de requisições pelo controlador, como também pode ocorrer através da poluição (*jamming*) do canal entre controlador e comutadores [14]. Outro ponto frágil da proposta de SDN com controle fisicamente centralizado é a resiliência da rede, dado que em um cenário de falha do controlador há a perda do controle de toda a rede, o que pode causar interrupção dos serviços ou comportamentos inesperados.

A distribuição do controle e a replicação de controladores são as principais técnicas para aumentar a resiliência das Redes Definidas por *Software* [15]. No entanto, a adoção de controladores que agem como um sistema distribuído [16, 17] ou a simples replicação de controladores [18] não é suficiente para garantir a segurança, o desempenho e a escalabilidade, pois dependem ainda do mapeamento otimizado entre comutadores e controladores, criando domínios de controle. Outro desafio que desponta da distribuição do controle é manutenção da consistência da visão global unificada [7, 19]. A visão global inconsistente acarreta em perda de desempenho e erros na execução de aplicações de controle da rede [19]. Assim, a distribuição do controle em SDN pode ser entendida sob duas vertentes. A primeira é a provisão de um controlador distribuído com visão global da rede consistente [16, 17]. A segunda é a otimização da localização e da quantidade de controladores necessários para uma determinada topologia de rede [20, 21].

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 [22]. 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 [23]. 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 [22]. Logo, ao ocorrer uma atualização, cada pacote é tratado pela configuração anterior à atualização ou pela configuração posterior.

1.1 Objetivos

O objetivo deste trabalho é discutir os problemas de segurança, distribuição de controle e consistência em Redes Definidas por Software e, então, propor soluções. A segurança da rede é definida em relação a três desafios fundamentais em redes definidas por *software*: vulnerabilidade a ataques de negação de serviço, ausência de confiança entre componentes e a vulnerabilidade de componentes. O controle logicamente centralizado em SDN garante a facilidade de programação de aplicações de rede e a serialização correta das ações na rede. Contudo, a centralização física do controle representa um desafio para a segurança, a resiliência e o desempenho da rede. Assim, é necessário que o controle das redes definidas por software seja fisicamente distribuído em diferentes instâncias de controladores, enquanto também apresente uma interface única e consistente de controle logicamente centralizado. A consistência em SDN é composta por duas vertentes principais: a consistência de tratamento do fluxo durante a sua existência e a consistência das políticas de encaminhamento ou tratamento de fluxos entre controladores. Este trabalho formaliza o problema de consistência em SDN com base na literatura existente e propõe um mecanismo de autenticação e controle de acesso para SDN, assim como também propõe um controlador distribuído com consistência forte de políticas entre controladores. A ideia central é usar o teorema do CAP (Consistência, Disponibilidade e Tolerância a Partições) para balizar a proposta do controlador distribuído e garantir consistência local e disponibilidade, mesmo em cenários em que haja partições da rede [24].

1.1.1 Mecanismo de Autenticação e Controle de Acesso para Redes Definidas por Software

Esse trabalho primeiramente apresenta a proposta e o desenvolvimento do mecanismo de autenticação e controle de acesso para redes definidas por *software*, o AuthFlow. O mecanismo AuthFlow apresenta duas contribuições principais: (i) a

autenticação das estações finais diretamente na camada de enlace e (ii) a associação das credenciais de acesso de uma estação aos fluxos pertencentes a essa estação. A autenticação da estação final na camada de enlace é realizada através do padrão IEEE 801.X que garante que as informações de autenticação sejam trocadas de forma padronizada entre a estação e o autenticador e, portanto, não requer qualquer alteração nas estações finais. O mecanismo de autenticação encapsula as mensagens no formato *Extensible Authentication Protocol* (EAP), o que permite a adoção de diferentes métodos de autenticação. A autenticação do mecanismo AuthFlow, direto na camada de enlace, tem como vantagem prover uma baixa sobrecarga de controle quando comparada a uma autenticação na camada rede, ou na camada de aplicação, que dependem da atribuição de um IP à estação que está se autenticando e dependem da troca de informações da aplicação para a autenticação.

As principais propostas para prover segurança às redes definidas por *software* buscam desenvolver módulos de segurança no controlador que facilitam o desenvolvimento de novas aplicações seguras [25, 26]. Por outro lado, outras propostas de autenticação de estações finais em SDN consideram que a autenticação deve ser feita somente após a estação receber um endereço IP temporário e ser redirecionada a um sítio *Web*, onde deve apresentar suas credenciais [2, 27]. Contudo, essas propostas estão sujeitas a ataques de falsificação de endereço, além de introduzirem uma maior sobrecarga de controle quando comparadas ao AuthFlow. Outras propostas descrevem algumas ameaças de segurança das redes definidas por *software* e indicam possíveis direções para solucionar essas ameaças [3, 20]. Considerando as principais ameaças às redes definidas por *software*, um protótipo do mecanismo AuthFlow foi desenvolvido e avaliado no ambiente de experimentação *Future Internet Testbed with Security* (FITS) [28][1]. A eficácia do mecanismo de controle de acesso proposto é evidenciada nos experimentos que mostram o bloqueio de estações ao tentar usar a rede, tanto no caso em que a estação não está autenticada, quanto no caso de a estação ter sua autenticação revogada. Os resultados da avaliação do protótipo mostram ainda que as estações finais têm visões diferentes da rede, liberando ou bloqueando acesso a determinados serviços, dependendo do nível de privilégio que cada estação tem de acordo com a sua credencial de acesso.

1.1.2 Distribuição do Controle e Resiliência para Redes Definidas por Sofware

Em seguida, este trabalho propõe uma arquitetura eficiente para a distribuição de controladores em uma Rede Definida por *Software*. Para tanto, o trabalho aborda

[1] O FITS é uma rede de testes interuniversitária desenvolvida a partir da parceria de instituições brasileiras e europeias. Maiores informações em http://www.gta.ufrj.br/fits/.

a distribuição do controle sob a ótica das duas vertentes. Primeiro, desenvolve-se um controlador distribuído para SDN. A proposta consiste em criar zonas de controle independentes com um controlador responsável. Os controladores das zonas se reportam a um controlador designado responsável por manter a consistência da visão global conhecida pelos controladores. Na arquitetura proposta, todos os controladores exercem o controle plano da rede, isto é, não há uma hierarquia em que controladores locais se reportem a controladores globais. O controle plano depende somente da propagação das informações de atualização da visão global através dos controladores. Na arquitetura proposta, adota-se a ideia de um controlador designado que recebe as atualizações e repassa aos demais, reduzindo a sobrecarga da propagação de informações. Após, modela-se o problema da localização de controladores, considerando a minimização da latência entre controlador e comutadores e, também, a maximização da conectividade entre controladores. A minimização da latência é um critério que privilegia diminuir o tempo de configuração de um novo fluxo. A maximização da conectividade entre controladores é importante para garantir a resiliência da rede [29]. Um protótipo do controlador proposto foi implementado. A otimização da localização de controladores foi analisada em diferentes topologias e a proposta cria um mecanismo de solução para a falha do controlador designado, que foi denominado de controladores de salvaguarda (*backup*). No caso de queda de um controlador designado, outro controlador assume. A lista de controladores de salvaguarda é ordenada de acordo com a média das latências dos controladores de zona em relação aos demais. O objetivo de se manter a lista ordenada é garantir que sempre o controlador com menor latência está pronto para assumir o papel de controlador designado e manter a resiliência da rede.

A distribuição do controle entre diferentes nós físicos, mantendo a visão global de rede centralizada, é um dos principais desafios em SDN [16, 17]. Algumas propostas apontam os problemas das Redes Definidas por *Software* e definem soluções iniciais para a distribuição de controle e para a resiliência da rede [18, 20]. Outras propostas buscam criar novos controladores distribuídos do ponto de vista de implementação de um sistema operacional distribuído para controlar uma SDN, mas não otimizam a localização dos controladores [16, 17, 30, 31]. Há ainda estudos que focam na otimização da localização dos controladores seguindo diferentes heurísticas e objetivos [15, 21]. Essas propostas não visam criar uma arquitetura única que considere desde a concepção do controlador até a localização otimizada dos controladores. Por sua vez, este trabalho propõe a arquitetura de controle distribuído com a localização otimizada e mecanismos para garantir a resiliência da rede. A ideia principal de a arquitetura considerar tanto a distribuição quanto a localização é que a resiliência seja pensada já na fase de projeto do controlador distribuído e não somente como uma restrição da localização otimizada de instâncias de controladores. Assim,

considera-se a interdependência entre a arquitetura do controlador distribuído e a sua localização.

1.1.3 Atualização de Políticas Consistente em Redes Definidas por Sofware

O tratamento consistente dos fluxos em uma rede definida por *software* requer que os pacotes de um fluxo sempre sejam processados pela visão global mais recente da rede. Para tanto, ao se fazer uma atualização nas políticas de encaminhamento e tratamento de pacotes na rede, essas atualizações devem ser orquestradas de forma a garantir que nenhum pacote possa ser processado por configurações mais antigas da rede, mesmo que durante um curto transiente de instalação das novas políticas. Os estados intermediários de transição entre configurações da rede podem causar laços, perda de conectividade, vulnerabilidades ou, até mesmo, parada no funcionamento da rede. As principais propostas para a atualização de políticas em SDN baseiam-se nas ideias de atualização atômica [32] ou atualização em duas fases [22, 33]. 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 da 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 coringas [34, 35]. Já a proposta deste trabalho reduz a sobrecarga, quando comparada à Atualização de Duas Fases, e provê garantias de consistência, quando comparada à atualização atômica.

Este trabalho 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 trabalho 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 [22], verifica-se que que a proposta deste trabalho apresenta menor sobrecarga de configuração e, ainda, age mais imediatamente para a implantação da atualização consistente sobre os pacotes em trânsito.

Em um cenário com o controle distribuído da rede, o desafio de se garantir a consistência no tratamento dos fluxos é ainda maior, pois atualizações das políticas que definem a visão global da rede podem ser emitidas concomitantemente por diferentes controladores. Assim, antes de se instalar uma nova política na rede, há a necessidade de se definir uma ordem entre as políticas que chegam aos controladores, garantir que haja um acordo entre os controladores sobre quais políticas devem ser instaladas e, só então, proceder a instalação consistente da política na rede. Este trabalho propõe, então, um protocolo simples e eficiente para a serialização[2] 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 e, assim, permitir que a nova política seja instalada de forma consistente. 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.

1.2 Organização do Texto

O restante do texto deste trabalho é organizado em seis capítulos. O Capítulo 2 explica os principais conceitos e vulnerabilidades de Redes Definidas por Software, apresenta e discute os principais desafios na distribuição do controle em SDN e, em especial, discute o problema de quantos controladores e qual o posicionamento dos controladores em uma dada rede. A proposta do mecanismo de autenticação e controle de acesso é apresentada e avaliada no Capítulo 3. O Capítulo 4 aborda o problema da consistência entre controladores e apresenta a proposta de controlador distribuído e, na sequência, avalia o controlador distribuído proposto e as heurísticas de localização de controladores em topologias reais de rede. O Capítulo 5 formaliza

[2]No contexto desta tese, serialização refere-se ao ordenamento de ações na rede sem que haja sobreposição temporal entre ações.

o problema da consistência abordando as duas vertentes, consistência de fluxo e entre controladores. O Capítulo 6 discute o problema da consistência de fluxos e propõe dois esquemas consistentes para atualização de políticas em redes definidas por *software*, a Atualização Reversa e o protocolo de consistência entre controladores. O Capítulo 7 conclui o trabalho e discute os trabalhos futuros.

Capítulo 2

Segurança e Distribuição de Controle em Redes Definidas por Software

O paradigma de Redes Definidas por Software (*Software Defined Networking* - SDN) [36] se baseia na separação física das funções de controle, em um plano de controle, das funções de encaminhamento de quadros, em um plano de encaminhamento. A ideia chave da separação é prover maior flexibilidade às funções de controle enquanto o *hardware* especializado para comutar quadros a alta velocidade permanece inalterado. Logo, a SDN oferece uma alta programabilidade das funções de controle da rede em um comutador com alto desempenho de encaminhamento de quadros. O operador pode definir de maneira simples os fluxos e as ações sobre os fluxos através de uma interface de programação de aplicação [37].

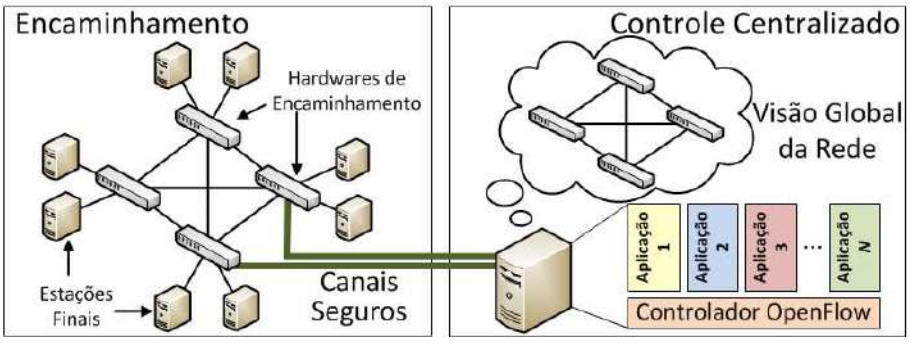


Figura 2.1: Rede Definida por *Software* separa as funções de encaminhamento e controle. O *hardware* de encaminhamento é controlado por aplicações centralizadas que têm uma visão global da rede.

A Figura 2.1 ilustra uma rede definida por *software* com controle centralizado, separado dos elementos comutadores responsáveis pelo encaminhamento de pacotes. O controle da rede é executado por um *software* de propósito geral, denominado

controlador de rede, sobre o qual se desenvolvem aplicações com propósitos específicos para o controle da rede. O controlador se comunica com os comutadores e, então, possui uma visão unificada de todo o estado da rede. Assim, uma das principais vantagens da abordagem SDN é a formação de uma visão global, unificada, do controle da rede facilitando a tomada de decisões sobre sua operação. A visão global centralizada torna a programação da rede mais fácil e simplifica a representação de problemas [37]. O OpenFlow define que os elementos de encaminhamento ofereçam uma interface de programação de aplicação (*Application Programming Interface* - API) que permita um nó controlador centralizado estender as ações de controle e de acesso sobre a tabela utilizada pelos componentes de encaminhamento para determinar o próximo destino de cada pacote encaminhado.

2.1 Ameaças de Segurança em Redes Definidas por *Software*

As Redes Definidas por *Software* desacoplam os planos de controle e de encaminhamento. A arquitetura original SDN considera um plano de controle centralizado, no qual são implementadas aplicações de rede sobre um sistema operacional de rede [10, 38]. No entanto, a centralização do controle implica questões para a segurança, a escalabilidade e o desempenho da rede [20]. Nesta seção, discutem-se as principais iniciativas para prover distribuição do controle em SDN [16, 17, 30], assim como outras abordagens que aproveitam a localidade de ações na rede para diminuir a carga de controle [7, 19, 31].

A segurança de redes definidas por *software*, em especial a segurança de redes OpenFlow, é um tema bastante discutido atualmente. Há propostas para o desenvolvimento de aplicações de segurança sobre a infraestrutura de rede OpenFlow, como também há outras que visam garantir a segurança da própria infraestrutura. Contudo, garantir a autenticação, o controle de acesso, a escalabilidade, o baixo tempo de resposta, a confidencialidade e a disponibilidade em SDN continuam a serem tópicos importantes de pesquisa [3].

A visão global centralizada das redes definidas por *software* permite que a lógica de controle das aplicações de segurança seja mais completa e integrada do que as atuais [26] e, portanto, simplifica o tratamento de problemas complexos de segurança em rede. As aplicações de segurança usando Redes Definidas por Software se valem do controlador centralizado para implementarem lógicas de definição de fluxos baseadas em estados e, também, segurança baseada em fluxos como, por exemplo, algoritmos de detecção de intrusão ou de anomalias. Contudo, a criação de aplicações de segurança em SDN é um desafio, pois a própria segurança de uma rede

definida por *software* ainda é questionável [3]. Os principais desafios de segurança de uma rede definida por *software* se dividem em três categorias: negação de serviço, ausência de confiança entre componentes e vulnerabilidades de componentes.

A Negação de Serviço pode ocorrer tanto no plano de dados quanto no plano de controle. No plano de dados, uma estação maliciosa que gere fluxos falsos pode exaurir tanto os recursos de banda, quanto os recursos de memória, ou tabela de fluxos, dos comutadores da rede. A negação de serviço no plano de controle pode ser alcançada em dois pontos distintos da rede: no controlador e na comunicação do controlador com os comutadores. É possível exaurir a capacidade de processamento do controlador de rede ao se enviar uma grande quantidade de pacotes com diferentes cabeçalhos. Isto acontece, pois todo pacote é analisado por um comutador e todos os pacotes com cabeçalhos que não correspondem a nenhum dos fluxos já definidos devem ser enviados pelo comutador ao controlador de rede para que se defina um novo fluxo. Assim, em um cenário em que um comutador envia uma quantidade atípica de novos cabeçalhos de pacotes para o controlador, este pode ter seus recursos de processamento exauridos e não ser capaz de responder a pedidos de novos fluxos em tempo hábil. Da mesma forma, a negação de serviço pode ser obtida quando o enlace de conexão entre o controlador e os comutadores na rede é intencionalmente congestionado. Caso não haja um canal de controle em separado, redundância ou banda suficiente no enlace que conecta os comutadores ao controlador, um comutador malicioso pode gerar tráfego suficiente para sobrecarregar esse enlace e, consequentemente, impedir a comunicação do controlador com os demais comutadores. A autenticação de estações e dos comutadores, usando *Secure Socket Layer* (SSL) e infraestrutura de chaves públicas (*Public Key Infrastructure* – PKI), é capaz de evitar esse tipo de ameaça, pois somente nós autorizados teriam acesso à rede e, no caso de identificação de um comportamento malicioso, a autenticação do nó pode ser revogada e o nó, expulso da rede.

A Ausência de Confiança entre Componentes da Rede compromete uma Rede Definida por *Software*, pois as aplicações executadas sobre o controlador podem ter comportamentos maliciosos. Nesse caso, o controlador deve ser capaz de identificar quais são as aplicações confiáveis e quais são maliciosas. Assim, uma possível medida para aumentar a segurança nas aplicações executadas em uma SDN é o uso de mecanismos de atestação de aplicações e o estabelecimento de cadeias de confiança por atestação. Algumas propostas para prover segurança em SDN consideram o uso de um núcleo de segurança no próprio controlador para garantir a execução segura de aplicações, sem que uma aplicação interfira em outras ou execute ações proibidas na rede [25, 26]. Por outro lado, a ausência de confiança também

afeta o registro de ações que ocorreram na rede (*log*), já que o registro de ações, quando é feito, não apresenta nenhuma garantia de que a ação ocorreu e de que a aplicação que o registrou não tinha comportamento malicioso. Uma possível solução é a adoção de registro de ações por aplicação, assinados por cada aplicação, que por sua vez sejam atestadas e assinadas por desenvolvedores.

A Vulnerabilidade de Componentes não é um desafio de segurança exclusivo desse novo paradigma de rede, embora seja mais crítico em redes definidas por software, pois uma vulnerabilidade em um nó controlador torna toda a rede vulnerável. Assim, são três as possíveis fontes de vulnerabilidades: comutadores, controlador e estações de gerenciamento. Uma vulnerabilidade em um comutador pode permitir que um atacante, que ganhe acesso a um comutador, exerça um ataque contra o plano de controle, a exemplo da falsificação de mensagens de outros comutadores para exaurir os recursos do controlador. Uma vulnerabilidade que libere o acesso ao controlador permite que um atacante altere o plano de controle ou, até mesmo, execute uma nova aplicação de controle da rede. Uma vulnerabilidade que garanta ao atacante acesso à uma estação de gerenciamento permite que o atacante exerça configurações no plano de controle diferentes das corretas. Medidas de prevenção a este tipo de ataque são a atestação das aplicações de controle, o uso de protocolos de certificação dupla entre estações de gerenciamento e aplicações e, por fim, a replicação das aplicações de controle para a tolerância a falhas e a intrusão.

Entre alguns dos principais desafios em segurança para as redes definidas por *software*, é possível destacar três características necessárias a essas redes: escalabilidade, responsividade e disponibilidade [2]. Para prover tais características definir a melhor quantidade e a localização dos controladores é essencial [20]. Nesse sentido, a localização e a quantidade de controladores replicados necessários a uma rede devem respeitar os requisitos de segurança, escala, disponibilidade e tempo de resposta da rede.

A autenticação, autorização e controle de acesso são primitivas essenciais em uma Rede Definida por *Software*. Essas primitivas, somadas à atestação e replicação de controladores, são a base de uma rede segura em que componentes maliciosos, sejam por vulnerabilidades em *software*, sejam pelo comportamento nocivo à rede, podem ser identificados e isolados do funcionamento normal da rede [2, 26, 39].

2.2 O Problema da Distribuição e Localização de Controladores

Aplicações tradicionais de rede com controle distribuído, como protocolos de roteamento (BGP, OSPF, IS-IS), são par-a-par, pois cada equipamento comunica-se com seus pares, informando e recebendo eventos, para tomar decisões autônomas baseadas em conhecimento local da visão global da rede, mesmo que a visão local seja inconsistente com a visão global. Já nas Redes Definidas por *Software*, na qual o controle e o encaminhamento são desacoplados, ocorre uma mudança no modelo de controle. O controle em SDN segue o modelo cliente-servidor. Os equipamentos de encaminhamento de pacotes agem como clientes de um servidor de controle, o controlador da rede [20]. A mudança no modelo como ocorre a comunicação dos eventos em SDN impõe uma mudança na alocação e localização dos controladores e, portanto, ainda são desafios em aberto.

O bom desempenho do plano de controle é fundamental, pois o tempo de resposta de um controlador a um evento na rede determina a capacidade da rede de reagir e, no caso de um controlador reativo a chegada de novos fluxos, determina o tempo de instalação de novos fluxos. Assim, a distribuição do controle pode diminuir o tempo de reação e o tempo de instalação de novos fluxos, já que tende a diminuir a latência entre os comutadores e o controlador mais próximo na rede. Ademais, a arquitetura com controle centralizado é proposta por ser um plano de controle mais simples, capaz de melhorar a convergência, flexibilidade e agilizar a adoção de novos serviços no núcleo da rede. Contudo, a arquitetura centralizada apresenta limitações quanto à latência, à escalabilidade e à disponibilidade do plano de dados.

Uma vez que por necessidade de projeto ou de desempenho se decide por uma arquitetura com controlador SDN distribuído, coloca-se o problema da determinação do número de controladores e suas localizações. O problema da distribuição e da localização de controladores em SDN é definido por Heller *et al.* como duas características do controle da rede: i) o número de controladores necessários e ii) o lugar de posicionamento desses controladores [20]. No entanto, essas duas características não exaurem as possibilidades de otimização do controle da rede. Elas apenas norteiam a busca por soluções de controle otimizadas. Em adição a esses dois pontos, deve-se considerar uma função objetivo. A função objetivo pode maximizar a distribuição de controladores para tolerar falhas, ao passo que pode também minimizar a latência para aumentar o desempenho e reduzir o tempo de configuração de fluxos.

Além do problema de otimização da localização e da quantidade de controladores, há também o desafio da distribuição do estado na rede de controle. Para tanto, considera-se a arquitetura de rede mostrada na Figura 2.2. A Figura 2.2 apresenta os pontos de troca de estado da rede em uma SDN. A ideia é que a rede é divida

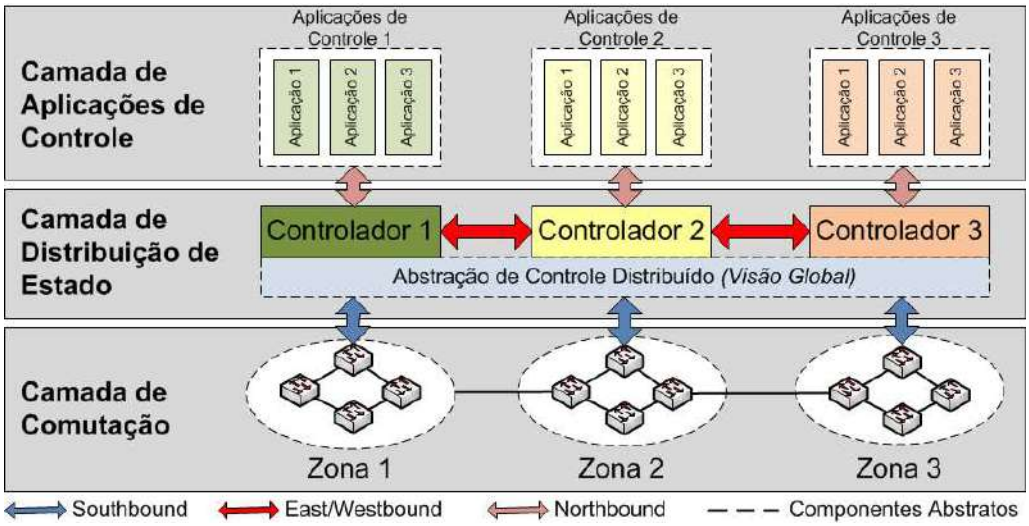


Figura 2.2: Arquitetura de uma Rede Definida por *Software* com controle distribuído com três camadas: comutação, distribuição do controle e aplicações. Cada ponto de troca de dados tem uma API (*Southbound, Northbound* e *East/Westbound*) indicando os fluxos de dados de controle na rede.

em três camada lógicas [19]. A camada mais baixa, a camada de comutação, é composta pelos elementos de encaminhamento de pacote. Esses elementos armazenam as tabelas de encaminhamento e, no caso de redes OpenFlow [10], armazenam também informações relativas aos fluxos e aos contadores de fluxo. A camada de comutação pode ser agrupada em domínios de controle distintos, seja por questões de escalabilidade, seja por questões administrativas.

A camada intermediária, denominada camada de distribuição de estado de rede, consiste na comunicação entre controladores, divulgando os eventos e o estado local de cada controlador. As propostas ONIX [17] e ONOS [16] implementam tal camada de forma semelhante a um *middleware* de sistemas distribuídos. Já o HyperFlow a implementa como um canal de eventos sob o modelo de Publicador/Assinante. Este trabalho propõe a realização da camada de distribuição de estado através da ideia de controlador designado que é responsável por manter uma cópia atualizada da visão global da rede. O modelo de controlador designado é inspirado pela ideia do roteador designado no protocolo de roteamento OSPF (*Open Shortest Path First*). A principal vantagem de se adotar o modelo do controlador designado é a simplificação da propagação de informação entre controladores, dado que as atualizações são encaminhadas ao designado que redistribui a todos os demais controladores. Esse modelo é mais simples que a abordagem de *middleware*, pois elimina a necessidade de gerenciar os estados distribuídos e garante à arquitetura proposta consistência e disponibilidade no plano de controle, enquanto os controladores por zona garantem resiliência e tolerância a partições. A camada de distribuição de estado é normalmente referenciada como Sistema Operacional de Rede ("*Network Operating Sys-*

tem") [7], pois é essa camada que faz a mediação entre as aplicações de rede e a camada de comutação, permitindo a abstração da camada de comutação para as aplicações. Por fim, a terceira camada lógica é a camada das aplicações de controle. Cada aplicação se comunica com as demais através da camada de distribuição de estado. As aplicações podem executar em nós distintos ou mesmo em um único nó.

Na Figura 2.2 é possível ressaltar três pontos sensíveis de troca de informação e distribuição do estado. O primeiro é entre os comutadores e os controladores (sistemas operacionais de rede). A implementação mais comum do conceito de SDN, padroniza esse ponto de troca de informação através da API OpenFlow [10]. Esse ponto de troca é comumente chamado de *southbound API*. O segundo ponto de troca de informação é entre controladores na camada de distribuição de estado. Nesse caso ocorre a comunicação horizontal entre controladores. A comunicação horizontal entre controladores é chamada de *East/Westbound API*. Por fim, há ainda o ponto de troca de informação entre o controlador, representado na camada de distribuição de estado, e as aplicações. Esse último ponto é chamado de *Northbound API* [38].

Vale ressaltar que o entendimento da arquitetura SDN em três camadas lógicas é geral para o controle centralizado ou distribuído. A diferença do controle distribuído encontra-se na camada de distribuição de estado. Assim, o problema de distribuição e localização de controladores pode ser entendido sob duas óticas. A primeira é a otimização do número de controladores e a localização deles e a segunda é a implantação de uma camada de distribuição de estados na rede que seja eficiente.

2.3 Conclusão do Capítulo

Esse capítulo apresentou os principais vetores de ameaça às Redes Definidas por *Software* e discutiu a distribuição de controle em SDN. Ambos os desafios estão também presentes nas redes tradicionais, contudo em Redes Definidas por *Software* tornam-se mais evidentes, devido à centralização do controle e à dependência da comunicação entre controlador e comutadores para o correto funcionamento da rede. Nos próximos capítulos discutem-se os trabalhos relacionados à provisão de segurança em SDN, sob diferentes aspectos da rede, assim como trabalhos relacionados à distribuição do controle. Baseando nos conceitos abordados nesse capítulo, este trabalho desenvolve as propostas de um mecanismo de autenticação e controle de acesso para SDN e uma arquitetura de controle distribuído com consistência forte.

Capítulo 3

O Mecanismo AuthFlow para Autenticação e Controle de Acesso em SDN

A ideia principal do mecanismo AuthFlow é realizar a autenticação usando protocolos da camada de enlace, fazendo o mapeamento da identidade usada na autenticação em fluxos criados por uma dada estação autenticada. Para tanto, o mecanismo proposto usa o padrão IEEE 802.1X e o *Extensible Authentication Protocol* (EAP). O EAP encapsula as trocas de mensagens de autenticação entre a estação suplicante[1] e um servidor de autenticação RADIUS. O autenticador empregado no mecanismo AuthFlow é um processo que se comunica como uma aplicação OpenFlow, que executa sobre o controlador POX. A aplicação controla o acesso da estação suplicante, aceitando ou bloqueando seu tráfego de rede, dependendo do resultado da autenticação entre o suplicante e o autenticador.

3.1 O Mecanismo AuthFlow

O mecanismo AuthFlow adota o padrão IEEE 802.1X, que especifica uma forma de autenticação diretamente na camada de enlace, por ser um padrão bastante adotado e não requerer modificações nas estações finais para a autenticação na rede, já que esse padrão já é implementado nos principais sistemas operacionais. Assim, quando uma estação compatível com o padrão IEEE 802.1X inicia, ela também inicia a fase de autenticação através do envio da mensagem de `início` do IEEE 802.1X para o endereço reservado MAC *multicast* (`01:80:C2:00:00:03`), com o tipo Ethernet definido em `0x888E`. Dessa forma, o procedimento de autenticação de uma estação

[1]Os nomes suplicante, autenticador e servidor de autenticação são definidos pelo padrão IEEE 802.1X.

não depende de nenhum conhecimento prévio acerca da rede, nem mesmo da tradução de um endereço IP em um endereço MAC, já que os quadros endereçados para o endereço *multicast* do protocolo são diretamente encaminhados para o autenticador. Esse procedimento evita que a estação receba um IP temporário, pois toda a autenticação ocorre através dos quadros *ethernet*, antes mesmo de a estação receber um IP definitivo dependendo do resultado da autenticação. Esse procedimento evita a falsificação de endereços de estações já autenticadas e, além disso, diminui a sobrecarga de autenticação, já que evita uma atribuição desnecessária de um IP temporário. O uso do padrão IEEE 802.1X facilita muito o processo de autenticação.

A seguir, discutem-se a arquitetura e o funcionamento do mecanismo AuthFlow. O estudo de caso considerado é usar o mecanismo proposto para a autenticação de roteadores virtuais em uma infraestrutura de rede virtual híbrida Xen e OpenFlow. Contudo, a proposta não se limita a esse estudo de caso e pode ser usada, sem qualquer alteração, na autenticação de estações finais em uma rede OpenFlow. No estudo de caso considerado, as estações componentes da rede OpenFlow são máquinas virtuais que se comportam tanto como estações finais quanto como roteadores.

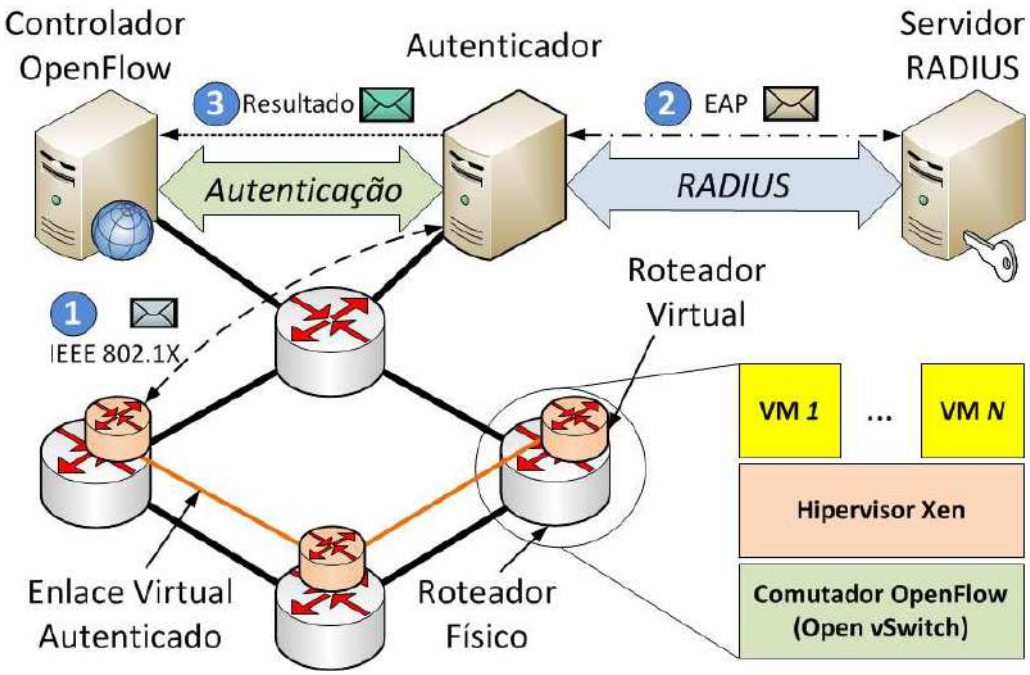


Figura 3.1: A arquitetura AuthFlow composta por três entidades: o controlador OpenFlow, o Autenticador e o Servidor RADIUS. Para uma máquina virtual (VM) para se autenticar na rede, (1) envia um pacote IEEE 802.1X com a autenticação EAP. (2) O Autenticador usa o conteúdo EAP do pacote IEEE 802.1X para autenticar o roteador virtual com o servidor RADIUS e (3) comunica o resultado da operação ao controlador OpenFlow.

A fim de realizar a integração do padrão IEEE 802.1X com a estrutura de uma Rede Definida por Software, propõe-se a arquitetura mostrada na Figura 3.1. A

arquitetura do mecanismo AuthFlow é composta de máquinas físicas hospedando máquinas virtuais, comutadores OpenFlow por *software*, um controlador POX, um autenticador e um servidor de autenticação RADIUS. As máquinas físicas e virtuais agem como roteadores e, então, são denominadas respectivamente roteadores físicos e roteadores virtuais. Os roteadores físicos são os nós com o sistema de virtualização Xen que hospedam os roteadores virtuais. O encaminhamento dos pacotes entre os roteadores físicos e virtuais é realizado por um comutador por *software* compatível com a API OpenFlow. No caso do mecanismo proposto, o comutador OpenFlow é um Open vSwitch[2] instanciado em cada roteador físico. O modelo de virtualização adotado no mecanismo proposto é o modelo híbrido Xen e OpenFlow usado no sistema XenFlow [40–42]. O controlador POX executa uma aplicação para manipular o encaminhamento de todos os pacotes, em especial, os pacotes[3] do padrão IEEE 802.1X. Os pacotes IEEE 802.1X são encaminhados diretamente para o autenticador, já que o controlador insere regras *a priori* na rede para o tratamento desses pacotes. Autenticador é um cliente RADIUS que implementa o padrão IEEE 802.1X e repassa o conteúdo EAP para o RADIUS. O autenticador foi desenvolvido como uma versão adaptada do `hostapd`[4], que é um autenticador usado em redes sem fio. O `hostapd` foi modificado para informar à aplicação POX sobre a autenticação das estações ou, no estudo de caso apresentado, redes virtuais. Assim, ao realizar a autenticação de uma rede, o `hostapd` envia uma confirmação de autenticação para o POX através de um canal seguro, criptografado e autenticado usando o esquema de distribuição de chaves públicas (*Public Key Infrastructure* – PKI) e o padrão SSL 3.0 (*Secure Socket Layer*). O servidor de autenticação é um servidor RADIUS que extrai as informações de autenticação do encapsuladas pelo EAP e valida as credenciais apresentadas pelos roteadores virtuais. Como o EAP permite o uso de diversos métodos de autenticação diferentes, o método adotado foi o MSCHAP v2 [43] que autentica o roteador virtual em uma base de dados usando como credenciais nome do usuário e senha. Para tanto, foi usada uma base de dados *Lightweight Directory Access Protocol* (LDAP). Contudo, o mecanismo de autenticação e a base de dados a serem usados não são essenciais para a descrição do mecanismo proposto, pois não interagem diretamente com a rede OpenFlow.

O mecanismo de autenticação AuthFlow funciona da seguinte maneira. Um roteador virtual envia um pedido de autenticação, no padrão IEEE 802.1X, e o controlador POX redireciona para o Autenticador. Em seguida, o Autenticador responde e a estação suplicante envia suas credenciais. O Autenticador verifica as credenciais de estação suplicante com o servidor RADIUS, executando o método de

[2]http://www.openvswitch.org/.

[3]A nomenclatura de pacote foi usada, pois é mais genérica e a API OpenFlow tem acesso às camada de enlace Ethernet até a camada de transporte.

[4]http://hostap.epitest.fi/hostapd/.

autenticação definido no EAP. Se as credenciais estão corretas, o Autenticador envia uma mensagem de `sucesso` para a estação suplicante e envia uma mensagem de autorização e confirmação de autenticação para o POX, através do canal seguro SSL, identificando a estação suplicante. Após o estágio de autenticação, o controlador POX permite que a estação suplicante acesse aos recursos da rede. Em caso de revogação das credenciais da estação suplicante, o Autenticador comunica ao POX, que imediatamente suspende o acesso da estação à rede.

O controle de acesso do mecanismo proposto funciona através da liberação e bloqueio de enlaces. Assim, ao iniciar uma rede OpenFlow que empregue o AuthFlow, todos os enlaces da rede estão bloqueados para a comunicação, inclusive os enlaces que interconectam os nós comutadores OpenFlow, ou seja, os enlaces pertencentes ao núcleo da rede. Nesse caso, esses enlaces não necessitam realizar o processo de autenticação para ter o seu tráfego liberado. Para tanto, o mecanismo AuthFlow realiza a descoberta da topologia do núcleo da rede através de pacotes *Link Layer Discovery Protocol* (LLDP) encaminhados enlace a enlace. Como o controlador gera e verifica cada pacote LLDP transmitido no núcleo da rede, o controlador é capaz de identificar quais enlaces estão entre comutadores OpenFlow e quais são enlaces finais ligados a estações finais. Os pacotes LLDP gerados pelo controlador são marcados unicamente para evitar ataques de repetição (*replay*) ou de falsificação (*spoofing*). Assim, somente os pacotes LLDP gerados e recebidos pelo controlador validam a autenticação de um enlace do núcleo da rede.

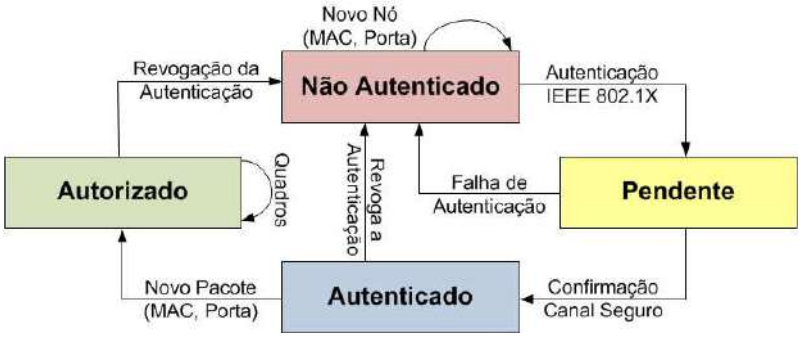


Figura 3.2: Diagrama de estados do controle de acesso do mecanismo AuthFlow. i) Estado Não autenticado, quando a estação não iniciou o processo de autenticação; ii) estado Pendente, enquanto o processo de autenticação foi finalizado; iii) estado Autenticado, quando a estação já finalizou a autenticação com sucesso; iv)estado Autorizado, quando a estação está autorizada a usar a rede.

O controle de acesso no mecanismo AuthFlow é executado de acordo com o diagrama de estados apresentado na Figura 3.2. Na figura, um nó da rede é sempre representado por uma tupla `(MAC, porta)`. Essa tupla identifica que uma estação com um dado endereço de camada de enlace (`Medium Access Control` – MAC) está conectada na `porta` do comutador. A Figura 3.2 sintetiza o processo de autentica-

ção, mostrando que a autenticação ocorre no sentido de controlar o acesso de um MAC através de uma porta do comutador. Dessa forma, um nó, ao ingressar na rede, está inicialmente no estado `não autenticado` e, assim, todo tráfego gerado ou destinado a esse nó é bloqueado, exceto pelo tráfego com tipo Ethernet `0x888E` especificando o padrão IEEE 802.1X. O controle de acesso é pela definição de dois fluxos padrões: i) encaminhar todo o tráfego de autenticação para o Autenticador, através de fluxo *multicast*; ii) descartar pacotes não autenticados. Esses fluxos são criados ao chegar um novo pacote de um endereço MAC não autenticado. Essa estratégia de descarte de pacotes evita a negação de serviço distribuída (`Distributed Denail of Service` - DDoS). A comunicação no sentido do Autenticador para a estação suplicante, o tráfego é encaminhado em fluxos *unicast*, já que o Autenticador aprende o endereço MAC da estação suplicante após o recebimento do primeiro pacote IEEE 802.1X. Assim que a estação inicia o procedimento de autenticação, enviando a mensagem de `início`, a estação é movida para o estado de `pendente`, estado em que todo o tráfego da estação continua bloqueado, mas a estação está no aguardo de uma confirmação do Autenticador para o POX de que sua autenticação foi bem sucedida e quais as credenciais foram usadas na autenticação. Assim que há a confirmação de que a autenticação foi bem sucedida, o POX move a estação para o estado `autenticado`. Nesse estado, o POX confere as permissões de acesso a recursos da rede que a estação possui, de acordo com suas credenciais, e libera o acesso da estação à rede. No entanto, quando há tráfego para a estação, o POX confere se o trafego para a estação está de acordo com as políticas referentes às credenciais usadas pela estação para acessar a rede. É importante ressaltar que no modelo de controle de acesso adotado, verifica-se se o tráfego que chega ou que parte de uma estação está de acordo com as políticas de uso da credencial associada à estação. Caso as políticas estejam de acordo com o uso da rede, a estação é movida para estado `autorizado` e acessa os recursos da rede de acordo com seus privilégios.

O AuthFlow interage com um Provedor de Identidade e fornece ao controlador a primitiva de identificação dos fluxos de acordo com a credencial do usuário usada na autenticação na rede. No estudo de caso desse trabalho, o Provedor de Identidade é representado pelo servidor LDAP. O controle de acesso proposto, baseado na identidade do usuário e nos fluxos gerados pelo usuário, permite monitorar e controlar, o conjunto de serviços que o usuário tem acesso. Tendo em vista o controle de acesso empregado, a liberação ou bloqueio do tráfego de uma estação final é realizado de acordo com as credenciais apresentadas pela estação. A autenticação da tupla `(MAC, porta)` está relacionada, assim, com a identidade do usuário da estação. Dessa forma, é possível fazer o mapeamento dos fluxos de uma dada estação para a identidade de seu usuário. A tupla de autenticação é gerenciada pelo controlador e está associada ao comutador de acesso de uma estação à rede. O

mapeamento ocorre da seguinte forma. Se um fluxo OpenFlow apresenta entre suas características o endereço MAC de origem (`dl_src`) e a porta de entrada no comutador (`in_port`) iguais aos que estão na tupla de autenticação, a identidade, validada pela credencial de autenticação usada pela estação, é atribuída ao fluxo. Assim, a decisão de encaminhamento desse fluxo pode tomar como parâmetro, também, a identidade do usuário da estação e, portanto, o controle de acesso à rede pode ser refinado. A identidade do usuário passa a ser mais um componente na identificação e definição dos fluxos na rede. De forma análoga, se um fluxo OpenFlow apresenta entre suas características o endereço MAC de destino (`dl_dst`) e a porta de saída do comutador (`output`) iguais aos que estão na tupla de autenticação, a credencial de autenticação da estação é também atribuída ao fluxo. Assim, o mecanismo AuthFlow também controla os fluxos destinados a uma estação de acordo com a sua identidade. Portanto, no AuthFlow, as políticas de controle de acesso podem definir regras tanto de saída quanto de entrada de pacotes para as estações finais de acordo com sua identidade.

3.2 Resultados Experimentais

O protótipo do mecanismo AuthFlow foi implementado em uma ilha do *Future Internet Testbed with Security* (FITS) [28]. O protótipo utiliza o hipervisor Xen 4.1.4 para prover os domínios virtuais que agem como estações finais acessando uma rede OpenFlow que, por sua vez, é implementada através do comutador programável Open vSwitch 1.2.2. O Open vSwitch [44] é configurado para ser controlado pelo POX[5], o controlador OpenFlow utilizado. A aplicação que realiza o controle de acesso das estações finais à rede e o controle do encaminhamento de pacotes na rede OpenFlow foi desenvolvida em Python. O autenticador usado no protótipo é uma versão modificada do `hostapd`, para criar o canal seguro e informar ao controlador POX quando há uma autenticação ou perda da autenticação de uma estação final. O servidor RADIUS empregado no protótipo é o FreeRADIUS v2.1.12[6]. Como prova de conceito, o método de autenticação testado no protótipo foi o EAP-MSCHAP v2 [43].

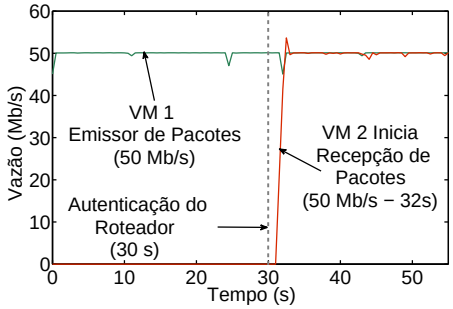
As ferramentas `Iperf`, `Nmap` e `Tcpdump`[7] foram usadas para realizar as medidas de avaliação de desempenho do protótipo. Quatro computadores pessoais compõem o cenário dos experimentos. Todos executam o protótipo do mecanismo AuthFlow. Nos computadores pessoais foram instanciadas quatro máquinas virtuais que agem como roteadores, enviam e recebem pacotes, dependendo de cada experimento. Todos os computadores possuem processadores Intel Core 2 Quad 2.4 GHz, 3 GB de

[5]O controlador POX utilizado nos experimentos é uma adaptação do controlador usado na rede de testes FITS para dar suporte ao mecanismo AuthFlow.

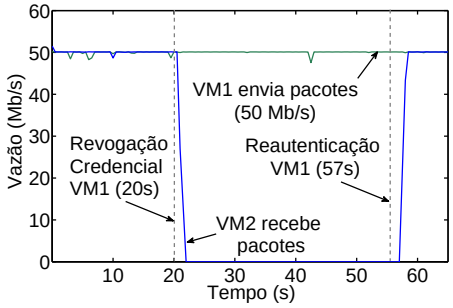
[6]http://freeradius.org/.

[7]http://iperf.sourceforge.com/, http://www.nmap.org/ e http://www.tcpdump.org/.

memória RAM e executam o Debian Linux 3.2.0-4-amd64. Cada computador possui, no mínimo, 2 interfaces de rede sendo que todas são configuradas para funcionarem a 100 Mb/s, para garantir homogeneidade, uma vez que havia também interfaces de 1 Gb/s. As máquinas virtuais são configuradas com uma CPU virtual, 128 MB de memória RAM e executa o Debian Linux 3.2.0-4-amd64. As máquinas virtuais executam os protocolos de roteamento através da plataforma XORP [45], a fim de avaliar o comportamento dos protocolos de roteamento no cenário testado, assim como avaliar o conhecimento da rede que cada máquina virtual possui após a sua autenticação.



(a) Liberação de tráfego pelo roteador situado entre as máquinas virtuais 1 e 2 (VM1 e VM2) após a sua autenticação que ocorre no instante igual a 30 s.



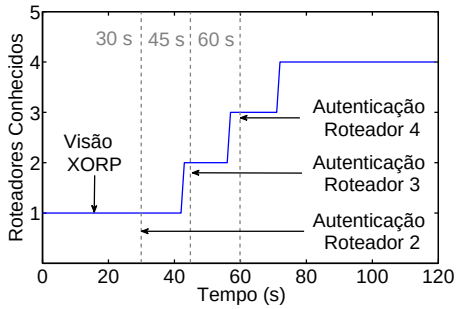
(b) Bloqueio do tráfego quando a autenticação da máquina virtual 2 é revogada no instante igual a em 20 s e liberação do tráfego quando a autenticação é restabelecida ao fim do teste.

Figura 3.3: Bloqueio da função encaminhamento de tráfego por falta de autenticação e por revogação de credencial. A máquina virtual 1 (VM1) envia pacotes para a máquina virtual 2 (VM2). a) Autenticação do roteador entre VM1 e VM2. b) Revogação da autenticação da VM1.

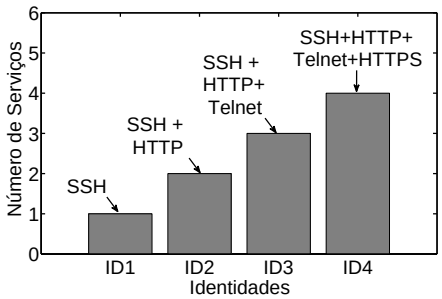
O primeiro experimento avalia a eficácia do mecanismo AuthFlow em bloquear os tráfegos não autorizados. O encaminhamento de pacotes de um fluxo só é liberado após a autenticação, caso contrário, os pacotes são descartados. O cenário é simples, a máquina virtual 1 (VM1) envia pacotes destinados à máquina virtual 2 (VM2) através de um roteador virtual. Assume-se que as máquinas virtuais 1 e 2 foram previamente autenticadas na rede e o roteador virtual que as interconecta não está autenticado. A VM1 gera um fluxo de pacotes UDP de tamanho 1472 B de conteúdo a uma taxa constante de 50 Mb/s. Como o roteador não está autenticado, o fluxo não chega à VM2. Após 30 s, o roteador se autentica na rede, como mostrado na Figura 3.3(a), e o fluxo UDP é recebido pela VM2. A Figura 3.3(a) evidencia que há um atraso, da ordem de 2 s a 2,5 s entre o início do processo de autenticação do roteador e a efetiva liberação do acesso à rede. Esse atraso é devido ao processo de autenticação do padrão IEEE 802.1X somado ao tempo de instanciação do fluxo OpenFlow. Esse atraso ocorre somente no momento em que o roteador, entre VM1 e VM2, ingressa na rede. Após a autenticação da estação na rede, os novos fluxos

sofrem apenas o atraso da instalação das regras nos comutadores que é da ordem de 30 ms [40]. Os pacotes seguintes, após a instalação das regras nas tabelas de fluxo, são encaminhados de acordo com as regras instaladas e, portanto, não sofrem atrasos provocados pela operação do OpenFlow, apenas os atrasos inerentes ao encaminhamento normal dos pacotes.

O segundo experimento evidencia a eficácia do mecanismo de revogação de credencial do AuthFlow, mostrado na Figura 3.3(b). O cenário consiste de uma máquina virtual, VM1, que se comunica diretamente com outra máquina virtual, VM2, não há roteadores entre elas. Mais uma vez assume-se que inicialmente ambas as máquinas virtuais estão autenticadas. Após 20 s, a autenticação da VM2 é revogada e, então, o acesso à rede da VM 2 é bloqueado, tanto para o envio quanto para a recepção de pacotes. Observa-se que o atraso para o bloqueio da atividade da VM2 na rede é menor que 1 s. Após 57 s, a autenticação da VM2 é restabelecida e a VM2 volta a receber os pacotes. O procedimento de restabelecimento da autenticação ocorre com um atraso de aproximadamente 2 s, assim como a autenticação de uma nova estação. Deve ser ressaltada a efetividade do mecanismo proposto AuthFlow correspondente à ação de liberação e de bloqueio de tráfego através do encaminhamento e do descarte de pacotes, respectivamente, associada ao procedimento de autenticação e revogação de credenciais. Assim, o AuthFlow se constitui em um forte aliado na defesa contra ataques de negação de serviço devido a sua efetividade na ação de liberar e bloquear fluxos em redes definidas por *software*, condicionadas ao processo de autenticação.



(a) Quantidade de vizinhos que um roteador virtual conhece, base de dados do protocolo de roteamento (`Visão XOPR`), de acordo com o número de outros roteadores autenticados.



(b) Número de serviços providos pela rede, de acordo com a credencial usada pela estação final ao se autenticar na rede.

Figura 3.4: Visão da rede a partir de (a) um roteador virtual e de (b) uma estação final. Cada identidade usada só tem acesso a um determinado conjunto de serviços na rede.

Os experimentos seguintes demonstram a visão da rede para as estações autenticadas, ou seja, quais estações da rede uma estação autenticada alcança e quais os serviços da rede essa estação acessa. A Figura 3.4(a) mostra a visão da rede segundo um roteador virtual executando o protocolo de roteamento de estado de enlace

OSPF (*Open Shortest Path First*). A topologia considerada é um anel, conectando quatro roteadores virtuais. No início do experimento, somente o roteador observado está autenticado na rede. Após 30 s, autentica-se o segundo roteador. Em 45 s, autentica-se o terceiro e, finalmente em 60 s, autentica-se o quarto roteador. Vale ressaltar que o atraso entre a autenticação e a descoberta de cada novo roteador, indicado na Figura 3.4(a), é devido ao tratamento de pacotes de *brodcast*/*multicast* adotado na rede OpenFlow. Para evitar a sobrecarga na rede, a cada inundação de pacotes, é inserida uma regra nas tabelas de fluxos para realizar o descarte de pacotes com a mesma característica do pacote inundado por 5 s.

O quarto experimento procura demonstrar uma das principais vantagens oferecidas pelo mecanismo AuthFlow, que consiste no uso da credencial de autenticação como forma de realizar o encaminhamento de fluxos. A ideia chave é a autenticação prover uma "identificação" dos fluxos correspondentes aos serviços que são autorizado para a estação. Assim, a autenticação pelo mecanismo AuthFlow possibilita a liberação dos fluxos correspondentes aos serviços que foram liberados, bloqueando todos os demais fluxos. O experimento consiste em uma estação solicitante, autenticada com uma das quatro identidades possíveis (ID1, ID2, ID3 ou ID4), acessar outra estação fornecedora de serviços na rede. Cada identidade permite o acesso a um determinado número de serviços na rede (um, dois, três ou quatro serviços, respectivamente). Para tanto, a estação solicitante executa uma varredura de portas (`nmap`) na estação fornecedora de serviços. No cenário de testes, a estação solicitante é a mesma, para as quatro identidades, mantendo o mesmo endereço IP e MAC durante todo o experimento. A única modificação no cenário de testes é a autenticação da estação solicitante com outra identidade a cada teste. A Figura 3.4(b) mostra os serviços que a estação solicitante consegue acessar na estação fornecedora de serviços na rede. Assim, é possível observar que, ao estar autenticada com uma identidade, a estação só consegue acessar os serviços liberados para aquela identidade. Vale ressaltar que a varredura de portas retorna que as portas que não têm serviços liberados são filtradas, o que mostra que o bloqueio dos demais serviços é realizado pelo descarte dos pacotes `SYN`, o que, de fato, ocorre pelas regras instaladas pelo controlador POX ao verificar que uma estação não tem o devido nível de privilégio para acessar um serviço.

O último experimento compara a sobrecarga de autenticação, na perspectiva da estação final, entre dois mecanismos: um portal de autenticação web (*Captive Portal*) e através do IEEE 802.1X usado no AuthFlow. O experimento executa dez autenticações na rede, tanto no portal de autenticação quanto no IEEE 802.1X. Os resultados são mostrados como a média das dez rodadas, como 95% de intervalo de confiança. A Figura 3.5 mostra que o um portal de autenticação com uma página web

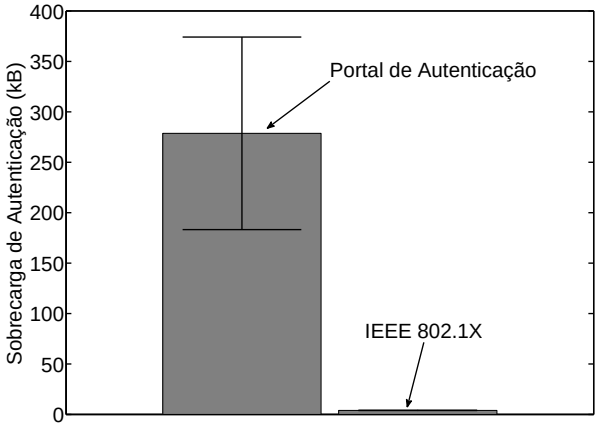


Figura 3.5: Comparação entre a sobrecarga da autenticação de um portal de autenticação web (*captive portal*) e da autenticação com o AuthFlow através do padrão IEEE 802.1X.

simples (`Portal de Autenticação`)[8], como proposto pelos trabalhos Resonance [2] and Ethane [27], pode introduzir até 300 kB de sobrecarga de autenticação. A grande variância na sobrecarga nessa abordagem de autenticação web deve-se ao uso de cache local pelo navegador web. O IEEE 802.1X (`IEEE 802.1X`), como proposto pelo AuthFlow, introduz uma pequena sobrecarga de 4 kB. Assim, a autenticação provida pelo AuthFlow introduz menor sobrecarga que as demais abordagens de autenticação em SDN.

3.3 Trabalhos Relacionados

Kreutz *et al.* apresentam uma classificação dos principais vetores de ataque a uma rede definida por *software* e possíveis contramedidas para se proteger de ataques contra a resiliência e confiabilidade da rede [3].

Visando garantir a confidencialidade e a disponibilidade de redes definidas por *software*, Mattos e Duarte propuseram o sistema QFlow [42, 46] que se baseia em um sistema híbrido Xen e OpenFlow para prover o isolamento de recursos e de comunicação entre redes virtuais sobre uma infraestrutura SDN. O sistema adota o encaminhamento de pacotes por filas para garantir a reserva de banda para cada rede virtual e marca os pacotes de cada rede com um marcador de VLAN, para multiplexar a rede virtual que um pacote pertence.

A rede UPV/EHU [47], uma rede OpenFlow europeia de testes, adota uma proposta de autenticação baseada no padrão IEEE 802.1X. Assim, não há a necessidade

[8]A mérito de simplicidade, avaliou-se a página padrão do *software* NoCat Captive Portal. Disponível em http://nocat.net.

de um nó adquirir um endereço IP para depois se autenticar. A estação inicia sua autenticação de acordo com o padrão IEEE 802.1X diretamente na camada de enlace assim que entra na rede, autenticando o seu endereço MAC na porta do comutador em que está conectado. Esse procedimento impede que um nó use um endereço MAC falsificado. O modelo de autenticação adotado baseia-se no encapsulamento EAP (*Extensible Authentication Protocol*), assim, qualquer que seja o método de autenticação escolhido, se for compatível com EAP, é trivialmente suportado pelo autenticador. A autenticação em redes definidas por *software* usando o padrão IEEE 802.1X tem a vantagem de ocorrer diretamente na Camada 2. Contudo, a proposta da rede UPV/EHU não considera o uso das credenciais de autenticação do nó na rede para a definição de novos fluxos.

Guenane *et al.* propõem um mecanismo de autenticação de redes virtuais usando EAP-TLS (*Extensible Authentication Protocol - Transport Layer Security*) implementado em cartões inteligentes (*smart cards*) [48]. A proposta consiste em garantir o acesso de máquinas virtuais e de clientes das redes virtuais a usuários que possuem cartões inteligentes. Estes cartões inteligentes implementam o protocolo TLS e encapsulam as mensagens em EAP. As mensagens encapsuladas EAP são enviadas para um servidor RADIUS que autentica os componentes da rede virtual, assim como os clientes da rede virtual, através da autenticação mútua provida pelos certificados, assinados por uma Autoridade Certificadora, apresentados durante a negociação TLS. No entanto, essa proposta não define como seria o mecanismo de controle de acesso dos nós à rede e como a autenticação é usada para autorizar o acesso do cliente aos recursos da rede. Uma limitação da proposta é a necessidade de uso de um cartão inteligente, o que limita o cenário de aplicação da proposta.

Resonance [2] e Ethane [27] são outras propostas que visam a autenticação de nós em uma rede definida por *software*. Ambas defendem que a autenticação do nó na rede deve ser feita através de portal *Web* em que o usuário deve apresentar as suas credenciais. Essa abordagem apresenta uma restrição básica que é a necessidade de o nó ter um navegador *Web* instalado. Esse requisito é bem limitante, quando se consideram ambientes formados por redes virtuais compostas por máquinas virtuais extremamente leves que não possuem nem interface gráfica. Outra desvantagem desse método de autenticação é a limitação ao modelo de autenticação por usuário e senha.

Hong *et al.* defendem que novos vetores de ataque podem prejudicar as Redes Definidas por *Software* de maneira fundamental [49]. Os autores apresentam modelos de atacantes que visam essencialmente comprometer os serviços de gerenciamento de topologia em redes OpenFlow existentes. Assim, Hong *et al.* propõem dois cenários de ataques de envenenamento na topologia de rede OpenFlow. A ideia central de ambos os ataques é manipular os pacotes LLDP (*Link-Layer Discovery Protocol*),

usados na descoberta de topologia da rede, para fingir uma localização falsa para uma dada estação e para manipular a visão global sobre a topologia de comutação da rede. Nesses cenários, definem-se três novos ataques que afetam o gerenciamento de topologia em redes OpenFlow: o reenvio de pacotes LLDP, a injeção de pacotes LLDP falsos e exploração de vulnerabilidades dos serviços de descoberta de topologia. Vale ressaltar que os ataques descritos no trabalho dependem da exploração de vulnerabilidades em comutadores ou em estações finais legítimas em uma SDN.

Porras *et al.* defendem que os avanços no protocolo OpenFlow não visam a evolução do controle da rede em suportar diversas aplicações em uma mesma rede [50]. Assim, os autores identificam os desafios de segurança envolvidos na gestão múltiplas aplicações dentro de uma única rede e, também, os desafios de se projetar um extenso conjunto de características que permitiria uma rede OpenFlow executar em um ambiente de computação sensível, com requisitos de segurança rigorosos. Os principais desafios de segurança em redes OpenFlow destacados são: a coexistência de aplicações; os conflitos entre regras de diferentes aplicações no momento de instanciação de fluxos; um esquema de permissão entre aplicações; a auditoria de aplicações; a separação de privilégios. Os autores, então, propõem uma implementação de referência de um controlador com segurança avançada, chamada SE-Floodlight. O controlador proposto introduz a ideia de papéis de aplicações seguras e a noção de um modelo de permissão específica do OpenFlow, para cobrir todas as trocas de mensagens entre aplicativos OpenFlow e o plano de dados. A proposta ainda conta com um algoritmo de verificação de conflito entre regras e um sistema de auditoria para a camada de aplicação.

As propostas FRESCO [26] e FortNOX [25] definem um conjunto de primitivas de segurança para redes OpenFlow. A proposta FortNOX defende a criação de um núcleo seguro de execução de aplicações sobre um controlador da rede OpenFlow. Esse núcleo seguro impede que uma aplicação execute ações que interfiram nas políticas de controle de outra aplicação. A proposta FortNOX defende o fatiamento da rede entre aplicações sobre um mesmo controlador, o que gera um controle mais fino dos privilégios e do domínio de controle de cada aplicação do que o previsto pelo FlowVisor [51]. A proposta do FlowVisor, por sua vez, fatia a rede entre diversos controladores, contudo, não prevê uma política de segurança entre controladores para que as ações de um controlador não afetem aos demais. Seguindo a ideia do núcleo seguro de execução de aplicações, a proposta FRESCO define um conjunto de primitivas e uma linguagem modular para o desenvolvimento de aplicações de segurança para a rede OpenFlow.

A maior parte das ameaças parte de estações legítimas da rede, vítimas de programas maliciosos ou mau uso por parte dos usuários legítimos [13, 52, 53]. Hand, Ton e Keller propõem o conceito de segurança ativa para a proteção de uma Rede

Definida por *Software* [54]. A ideia chave é prover uma interface de programação centralizada para controlar a detecção de ataques na rede, para coletar dados que permitam a investigação do ataque e para reagir aos ataques. Para tanto, os autores defendem a instalação de componentes ativos de segurança que são elementos capazes de monitorar a rede e lançar contramedidas. O funcionamento dos elementos de segurança é definido pela interface de programação proposta. Entre os componentes ativos de segurança pode-se citar *middlebox* de detecção de intrusão, ferramentas para estações finais capazes gerar cópias do conteúdo de memória e armazená-lo em local seguro em caso de identificação de anomalias e, também, elementos na rede capazes de lançar contra-ataques de negação de serviço em caso de identificação de um nó que esteja atacando a rede. No entanto, a proposta de segurança ativa viola a privacidade das estações da rede e gera uma grande sobrecarga de controle.

Seguindo uma ideia semelhante de monitoramento de redes através de uma infraestrutura de Rede Definida por Software, Shin e Gu propõem a ideia CloudWatcher [55]. A proposta consiste em um arcabouço de controle da rede em que o administrador seja capaz de definir através de uma semântica própria quais fluxos da rede devem ser inspecionados por dispositivos de segurança e qual dispositivo é responsável por inspecionar cada tipo de fluxo procurando por padrões previamente definidos. A proposta otimiza o roteamento dos fluxos para os dispositivos de segurança. Contudo, a proposta não considera a consistência da definição e da atualização de políticas de segurança.

As propostas de segurança em SDN relacionam-se com o mecanismo de autenticação proposto nesse trabalho, o AuthFlow, no sentido de que o AuthFlow pode ser usada como um módulo seguro do controlador OpenFlow, por exemplo, para permitir o uso de uma nova primitiva de segurança, a primitiva de autenticação e definição de fluxos baseada na identidade da estação. Com a primitiva de autenticação é possível identificar a quem pertence cada conjunto de fluxos definidos na rede.

3.4 Conclusão do Capítulo

A segurança de redes empresariais, principalmente, depende de mecanismos de controle de acesso e de autenticação eficientes. Com a crescente adoção da tecnologia de redes definidas por *software* (SDN) em redes empresariais, o desafio de prover segurança às SDN tornou-se ainda mais fundamental. Esse capítulo propõe o AuthFlow, um mecanismo de autenticação e controle o acesso à infraestrutura de um rede definida por *software* OpenFlow, baseado no padrão IEEE 802.1X e no servidor de autenticação RADIUS. O mecanismo AuthFlow proposto implementa a autenticação através de uma base dados LDAP com RADIUS. A proposta, no entanto, é

extensível a outros métodos de autenticação, como o EAP-TLS, que autentica os nós com base em certificados. Os resultados mostram que o mecanismo de autenticação proposto impede que estações não autorizadas acessem recursos da rede, mesmo quando já autenticadas e, após, perdem seus privilégios. Os resultados mostram ainda que o mecanismo proposto é mais eficiente que as demais propostas, já que introduz menor sobrecarga de controle, e permite a definição de políticas de controle de acesso por fluxo de acordo com as credenciais de acesso de cada estação. O Authflow, diferentemente das demais propostas de autenticação e controle de acesso, provê a autenticação já na camada de enlace e provê a primitiva de identificação dos fluxos para o controlador. A autenticação em Camada 2 garante a aplicabilidade do AuthFlow para dipositivos mais simples, como no cenário de Internet das Coisas (*Internet of Things* - IoT), e para funções de redes virtualizadas (*Virtualized Network Functions* - VNFs). A primitiva de identificação dos fluxos permite que o controlador da rede defina regras de acesso, em alto nível, por usuário e não mais por subredes ou endereços físicos. Assim, as regras de acesso são desvinculadas dos dispositivos do usuário e passam a serem vinculadas à identificação do usuário na rede.

Capítulo 4

O Plano de Controle Resiliente e Distribuído em SDN

A ideia central do controlador distribuído proposto é dividir o controle da Rede Definida por *Software* em zonas de controle e gerar uma visão global consolidada e consistente da rede entre todos os controladores de zona.

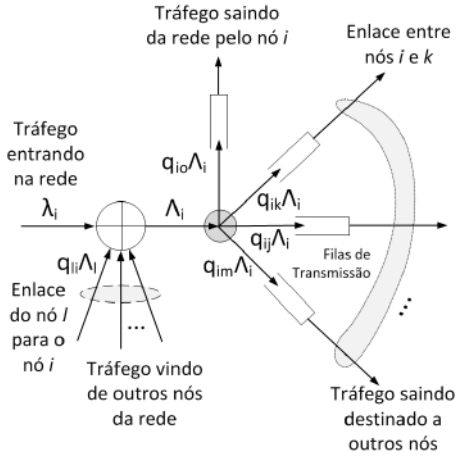
Propõe-se uma camada de distribuição do controle entre as instâncias de controladores que é responsável por manter a consistência. Essa camada distribui a visão global da rede entre todos controladores de zona. É importante ressaltar que a proposta divide o controle em zonas, mas diferentemente de outras propostas, não introduz hierarquias de controle. Assim, todo controlador tem a visão global e toma ações que podem afetar o estado global da rede.

Este capítulo propõe um modelo teórico de desempenho de um controlador em uma rede definida por *software* e, a partir do modelo, calcula-se a restrição do número de comutadores suportado por um dado controlador, assim como o número mínimo de controladores para uma rede a fim de se estabelecer um tempo médio de resposta. Em seguida, apresenta-se a proposta de controlador distribuído e as heurísticas de localização de controladores na rede.

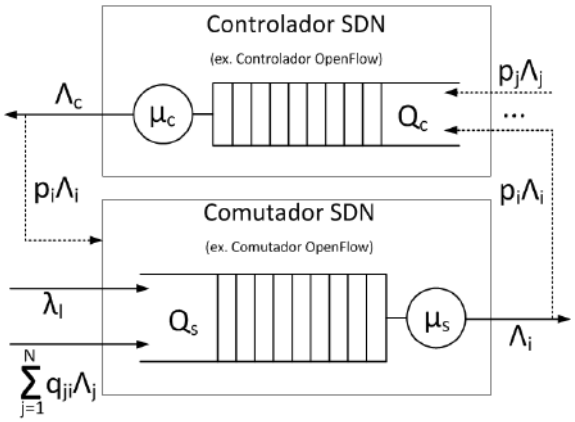
4.1 O Modelo Teórico de Desempenho de um Controlador

A implantação do plano de controle de uma Rede Definida por *Software* deve considerar quantos controladores são necessários para manter a disponibilidade, o baixo tempo de resposta e a robustez do plano de controle [20]. Nesse sentido, uma proposta deste trabalho é um modelo para o controlador SDN em que modela-se o tempo de tratamento de cada pacote em função do número de comutadores que um determinado controlador é responsável. O modelo proposto é simples e baseado

em teoria de filas. Assim, assume-se que o tráfego que chega a cada comutador da rede segue uma distribuição de Poisson para a frequência de chegada de novos pacotes e a taxa de tratamento de eventos em um controlador segue uma distribuição exponencial [56].



(a) Exemplo do nó i em uma Rede Definida por *Software*. A carga de entrada de pacotes na rede pelo nó i é dada por λ_i. O nó recebe ainda pacotes de outros nós.



(b) Esquema de filas de espera entre o controlador SDN e um comutador. A carga no controlador é a soma da carga de todos os comutadores que ele controla.

Figura 4.1: Modelo de uma Rede Definida por *Software* baseado em teoria de filas. a) Avaliação da carga em comutador da rede. b) Relação entre a carga dos comutadores e controlador que os controla.

A Figura 4.1 exemplifica o modelo de comutadores e controladores SDN adotado neste trabalho. Considera-se que cada comutador, representado pelo no i, recebe uma carga de tráfego que chega à rede através desse comutador, representada por λ_i, assim como também é responsável por encaminhar uma parcela do tráfego vindo de outros comutadores, dada por $q_{li}\Lambda_l$, que representa o percentual,q_{li}, do tráfego do nó l, Λ_l, que é encaminhado pelo nó i. Nesse cenário, a Figura 4.1(a) apresenta o modelo do nó i em relação a uma rede de filas. Vale ressaltar que o modelo da Figura 4.1(a) assume uma rede que pode conter ciclos[1]. A ponderação de qual parcela de tráfego segue para cada enlace é dada por q_{ij}, em que $q_{io} + \sum_{j=1}^{N} q_{ij} = 1$ e q_{io} é a parcela do tráfego que sai da rede pelo nó i.

Tendo em vista o funcionamento da rede de filas formada pelos comutadores em uma rede SDN, pelo Teorema de Bruke, tem-se que a saída de cada comutador também segue a distribuição de Poisson, assim como a entrada. Logo, a entrada em qualquer comutador da rede pode ser considerada uma distribuição de Poisson. Dessa forma, temos que a saída de cada comutador é dado por:

[1]A hipótese de rede acíclica é válida mesmo quando a rede física apresenta ciclos, mas executa o protocolo de *Spanning Tree*, pois o encaminhamento de pacotes é restrito à árvore de cobertura.

$$\Lambda_i = \lambda_i + \sum_{j=1}^{N} q_{ji} * \Lambda_j, \tag{4.1}$$

em que N representa o número total de nós na rede. Para os nós que não são vizinhos do nó i, q_{ji} assume o valor 0. Com base na Equação 4.1 pode-se modelar o funcionamento de um comutador genérico SDN, controlado por um controlador SDN, como mostrado na Figura 4.1(b). A Figura 4.1(b) mostra a carga de tráfego que chega ao controlador SDN. A chegada de pacotes no controlador é modelada como uma porção do tráfego do comutador que não é tratado diretamente pelo comutador devido à inexistência de fluxos pré-instalados [57]. A proporção de pacotes que segue para o controlador em um comutador i é dada por p_i.

Para o controlador, considera-se um modelo de filas M/M/1, em que a chegada de pacotes é dada por Poisson, representada por Λ_c, o tempo de serviço segue uma distribuição exponencial com média μ_c, considera-se somente um controlador por domínio de controle e, à mérito de simplicidade, considera-se que o controlador tem capacidade infinita de armazenar as requisições não tratadas. A política de filas usada é a FIFO (*First In First Out*), em que a primeira requisição a ser tratada é a primeira que chega ao controlador. Assim, tendo em vista que a carga no controlador pode ser representada pela soma das cargas dos comutadores que são controlados por ele, tem-se que:

$$\Lambda_c = \sum_{i=1}^{N} p_i * \left(\lambda_i + \sum_{j=1}^{N} q_{ji} * \Lambda_j \right), \tag{4.2}$$

em que Λ_c é a carga total de pacotes que chegam ao controlador, que por hipótese, segue uma distribuição de Poisson, dado que as chegadas de pacotes nos comutadores da rede também a sigam. Assim, pode-se estimar o tempo médio de serviço do controlador de acordo com modelo. O tempo médio de serviço é dado por:

$$\tau_{serv} = \frac{1}{\mu_c - \sum_{i=1}^{N} p_i * \left(\lambda_i + \sum_{j=1}^{N} q_{ji} * \Lambda_j \right)}, \tag{4.3}$$

em que μ_c representa a taxa de eventos de fluxos por segundo que o controlador consegue atender. O tempo de atendimento é modelado seguindo uma distribuição exponencial negativa.

A partir do modelo dado pela Equação 4.3, pode-se assumir algumas hipóteses para simplificar a um modelo de aplicação mais direta que leve a um tempo médio de serviço por pacote no controlador SDN:

- a taxa de pacotes de cada nó j que é considerada no modelo do controlador, Λ_j, pode ser simplificada pela taxa de pacotes que entram na rede em j, dado por

λ_j, já que todos os nós da rede são considerados no somatório a ponderação pode ser feita por qji;

- a distribuição dos clientes é uniforme na rede, assim como a entrada de pacotes em cada nó i apresenta as mesmas características do que nos demais nós. Portanto, considera-se $\lambda_i = \lambda, \forall i \in N$;
- a probabilidade de um pacote não encontrar um fluxo já instalado em um comutador, p_i, é igual em todos os comutadores, sendo simplificada por uma probabilidade constante p;
- a probabilidade de um pacote ser encaminhado para outro comutador, q_{ij}, assim como a probabilidade de um pacote deixar a rede, q_{io}, podem ser aproximadas para um valor constante q.

Assim, a Equação 4.3 pode ser simplificada para:

$$\hat{\tau}_{serv} = \frac{1}{\mu_c - p\lambda N * (1 + qN)}. \tag{4.4}$$

Vale ressaltar que no termo $(1 + qN)$ o valor 1 representa os pacotes que vão ao controlador no comutador de entrada na rede, enquanto o valor qk representa os pacotes que vão ao controlador nos demais comutadores da rede. Dessa forma, em uma SDN proativa, em que já na primeira vez que o pacote chega ao controlador, esse toma a decisão de instalar as regras do fluxo em todos os comutadores na rede, qN vale 0.

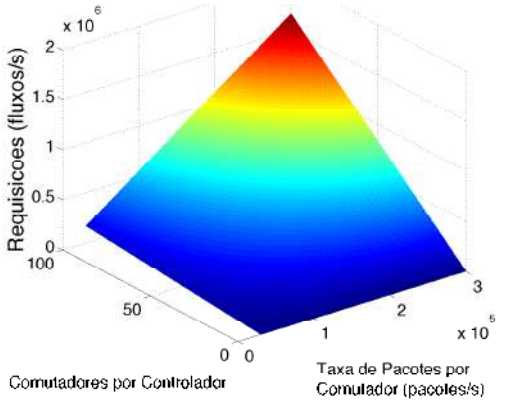
Outro ponto importante é que, dada a carga do controlador, pode-se calcular a probabilidade de haver δ requisições na fila do controlador. A probabilidade é dada por:

$$P_\delta = \left(\frac{\Lambda_c}{\mu_c}\right)^\delta \left(1 - \frac{\Lambda_c}{\mu_c}\right) \approx \hat{P}_\delta = \left(\frac{p\lambda N * (1 + qN)}{\mu_c}\right)^\delta \left(1 - \frac{p\lambda N * (1 + qN)}{\mu_c}\right) \tag{4.5}$$

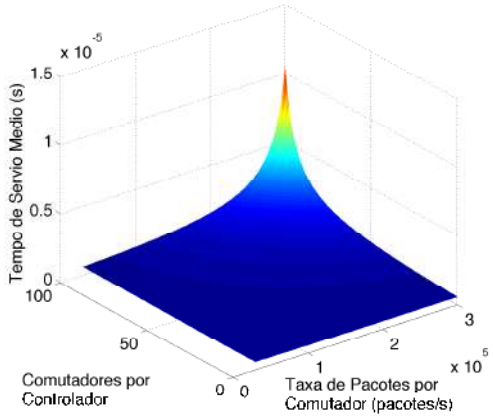
A partir do modelo de controlador dado pelas Equações 4.4 e 4.5, verifica-se que é possível estabelecer um limite de N comutadores para cada controlador a fim de se manter o sistema em equilíbrio ($\mu_c = \Lambda_c$).

A Figura 4.2 mostra a aplicação do modelo proposto em uma rede com até 100 comutadores sendo controlada por um nó controlador. A aplicação do modelo considera um controlador de alto desempenho, com capacidade de responder até $1,75 * 10^6$ $fops$[2] [58]. A taxa de chegada de fluxos por comutador varia de 0,03 a

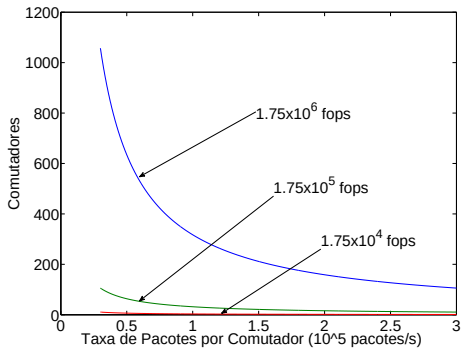
[2]A unidade `fops` é definida como o número eventos de fluxos por segundo que um controlador é capaz de tratar.



(a) Carga agregada de requisições no controlador.



(b) Tempo médio de serviço no para cada requisição no controlador.



(c) Número de comutadores por controlador para manter o sistema em equilíbrio.

Figura 4.2: Aplicação do modelo de controlador SDN em uma rede com um controlador com taxa de serviço $\mu_c = 1,75 * 10^6$ `fops`, número de comutadores variando de 1 a 100 e taxa de chegada de pacotes por comutador de até 0,3 `Mpps`.

0,3 milhões de pacotes por segundo, de acordo com os resultados de desempenho máximo de comutadores [59]. A probabilidade de um pacote ser encaminhado para o controlador foi estimada no inverso do número de pacotes por fluxo. A estimativa foi realizada com base nos dados reportados pela CAIDA (*Center for Applied Internet Data Analysis*) para o ponto de medidas passivas `chicago (dirA)` no período do ano de 2014[3]. Com base nesses dados, definiu-se a probabilidade $p = 0,0276$. Da mesma forma, os dados validam as taxas de chegada de pacote por comutador na medida em que a taxa de chegada de pacotes média em `chicago (dirA)`, um ponto de troca tráfego, para o período avaliado, é 0, 374 milhões de pacote por segundo.

Por fim, a probabilidade q de o tráfego de outro comutador ser mandado para o controlador em um ponto mais a frente na rede foi aproximada para o inverso do número de nós da rede, ou seja, $q = 1/N$, pois, por hipótese, os clientes estão

[3]Dados disponíveis em http://www.caida.org/data/passive/trace_stats/.

distribuídos uniformemente na rede e, assim, cada comutador receberia $1/N$ do tráfego de cada um dos outro $N-1$ comutadores. Vale ressaltar que essa simplificação implica não conhecer a topologia da rede, nem conhecer o caminho dos pacotes na rede e, portanto, é uma generalização do modelo para o cenário em que apenas há o conhecimento da taxa de serviço do controlador e do número de comutadores que o controlador atende na rede. Assim, aplicando-a nas Equações 4.2 e 4.4, tem-se que o termo qN tende a 1.

Tendo em vista o cenário avaliado, a Figura 4.2(a) mostra que a carga do controlador, com 100 comutadores e cada um gerando tráfego de 300 mil pacotes por segundo, é próxima do limite suportado pelo controlador. O limite de 100 comutadores é evidenciado pela Figura 4.2(c), na qual valores acima da curva tendem tirar o sistema da situação de equilíbrio. Outro ponto importante é verificar que próximo da saturação do sistema, o tempo total médio de tratamento do pacote pelo controlador[4] é de aproximadamente 10 ms, mostrado na Figura 4.2(b).

4.2 O Controlador Distribuído

A Figura 4.3 mostra a arquitetura do controlador proposto. Um dos controladores de zona é selecionado como o Controlador Designado. O Controlador Designado é um controlador de zona como os demais, mas que, a partir do momento da sua seleção como controlador designado, ele passa a ter a função adicional de ser o responsável por construir, manter e divulgar a visão global da rede. Nesse contexto, a visão global da rede é considerada como qualquer informação das aplicações de controle que sejam de interesse de mais de um controlador de zona ou que um controlador de zona deseja disponibilizar para os demais controladores.

O Controlador Designado armazena e disponibiliza um repositório de objetos de visão global da rede. Um objeto de visão global da rede é uma estrutura de dados que qualquer controlador registra na camada de distribuição do controle. Quando o objeto é registrado na camada de distribuição do controle, ele torna-se acessível aos demais controladores, tanto para leitura, quanto para a atualização. O repositório do controlador designado armazena todos os objetos registrados.

O registro de objetos na camada de distribuição do controle é importante para abstrair o controle distribuído. Ao acessar um objeto que está registrado no controlador designado, o controlador de zona pode operá-lo como se fosse local. O controle da consistência do objeto é realizado pelo controlador designado. Para tanto, um objeto registrado só pode ser atualizado através de um único acesso por vez. As aplicações que executam nos controladores de zona podem realizar o controle local

[4] A análise não considera a latência de envio do pacote entre comutador e controlador. Tal característica é inerente a cada topologia de rede.

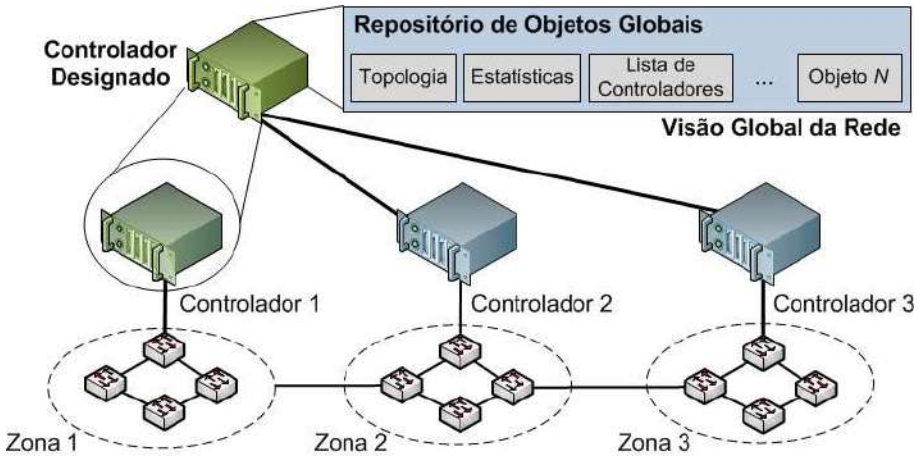


Figura 4.3: Arquitetura do controlador distribuído composta de diversos controladores de zona, possuindo um deles as funções adicionais de controlador designado. O controlador designado, Controlador 1 da figura acima, mantém o Repositório de Objetos Globais e é o responsável pela criação, manutenção e a divulgação do estado global da rede.

sem que haja a necessidade de se reportarem para o controlador designado, dado que para o controle local não há a necessidade de atualizar a visão global da rede.

No caso de aplicações que atualizam a visão global da rede, como a aplicação de descoberta de topologia, sempre que necessário, cada controlador de zona acessa o objeto referente àquela informação e o atualiza. A atualização dos objetos globais é mediada por um semáforo que impede o acesso concomitante de múltiplos controladores àquele objeto e as operações são atômicas. No caso da descoberta de topologia, o objeto compartilhado é uma coleção de enlaces conhecidos entre os nós e o conjunto dos nós conhecidos da rede. Além da descoberta de topologia, também se pode citar o cálculo da árvore de cobertura. Vale notar que, dada a visão global consistente e única da rede, o cálculo da árvore de cobertura pode ser realizado localmente. Cada controlador de zona só é responsável por ativar e desativar a inundação de portas de comutadores da sua zona, sem interferir nos demais [19].

Para prover resiliência ao sistema de controladores distribuídos proposto, todo controlador de zona é capaz de assumir o papel de Controlador Designado. Assim, todos controladores de zona possuem uma lista ordenada de controladores indicando a ordem do próximo controlador de zona que deve assumir o papel do controlador designado no caso do atual controlador designado vir a falhar. Essa lista é distribuída e mantida ordenada através de um objeto registrado no repositório do controlador designado ativo. Os controladores de zona, de tempos em tempos, consultam a lista ordenada e mantém uma cópia local do estado global da rede, pois no caso de falha do Controlador Designado, o estado global da rede é mantido. A ordenação da lista se dá através da ordem crescente média $L(C_k)$ da latência de um controlador a todos os outros. Ou seja, a cada controlador C_k é associado um valor:

$$L(C_k) = \frac{\sum_{\forall i \in N} Latencia(C_k, C_i)}{|N|}, \tag{4.6}$$

onde i é um nó da rede, C_i é o controlador do nó i, N o conjunto de todos os nós da rede e a lista é ordenada em função do valor $L(C_k)$ de cada controlador. A $Latencia(C_k, C_i)$ na Equação 4.6 é calculada *a prioprí*, de acordo com a topologia da rede, no momento da localização dos controladores. O controlador designado apenas mantém esse valor atualizado de acordo com a monitoração da rede. A ordenação da lista tem como principal objetivo garantir que todos os controladores conheçam a mesma ordem de controladores que devem assumir o papel de designado no caso de falha do controlador designado vigente. Portanto, a periodicidade de atualização da lista não é crítico.

4.3 Localização dos Controladores

A localização dos controladores é definida como um problema de otimização com múltiplos objetivos. A localização dos controladores é realizada no momento de inicialização da rede. Para atingir a máxima resiliência da rede é importante que a localização dos controladores considere que em um cenário de falha, a rede continue conexa e que, caso haja partição da rede, cada partição possua um controlador em seu interior. É importante que para o funcionamento adequado da rede, o tempo de configuração de um novo fluxo seja o mínimo possível [20, 21]. Assim, o tempo de comunicação entre o controlador e o conjunto de comutadores controlados deve ser o mínimo possível. Portanto, os objetivos da otimização da localização de controladores são dados por:

$$min \prod_{\forall i \in N} \left(PercentualRede(C_i) * Prob(C_i)\right)^{y_i}, \tag{4.7}$$

onde y_i identifica se o nó i é controlador ($y_i = 1$) ou não ($y_i = 0$), $PercentualRede(C_i)$ indica o percentual da rede em número de nós que fica sem controlador caso haja uma falha em C_i e N representa o conjunto de todos os nós na rede. A equação

$$min \sum_{\forall i \in N} \frac{\sum_{\forall k \in N} y_i * y_{i,k} * latencia(C_k, C_i)}{|N|} \tag{4.8}$$

minimiza a latência, em que $y_{i,k}$ representa que o controlador i controla o nó k e y_i representa se o nó de índice i é um controlador ou não. As equações valem para $i, k \in N$, onde N é o conjunto de todos os nós da rede.

A interpretação da Equação 4.7 é minimizar o percentual da rede, representado pela função $0 < PercentualRede(C_i) \leq 1$, que fica sem controle caso o controlador

C_i falhe. A função $Prob(C_i)$ modela a probabilidade de falha do nó C_i. No entanto, a falha do nó C_i só é prejudicial para o controle da rede caso C_i seja um controlador. Assim, cada termo do produtório é elevado a y_i que determina se o nó de índice i é controlador ou não. No caso de ser controlador, o termo é considerado. No caso contrário, o termo é elevado a 0, resultando no valor 1, que não interfere no produtório. Assim, ao minimizar a Equação 4.7, procura-se uma solução que mantenha a rede conectada e com controle, mesmo em cenários de falha. A Equação 4.8 busca uma solução de localização que minimize a latência média entre todos os controladores e os comutadores que ele controla.

4.4 Heurística de Localização

A instalação de controladores na rede de modo a otimizar a resiliência é um problema comprovadamente de solução não polinomial (NP-Difícil ou *NP-Hard*) [60]. Assim, neste trabalho desenvolve-se uma heurística para otimizar a resilência da rede ao passo em que limita o número de controladores ativos na rede. As Equações 4.7 e 4.8 definem a ideia central da estratégia de localização proposta. No entanto, para aplicar a estratégia proposta, desenvolve-se uma heurística de cálculo da localização de controladores otimizada. A heurística se baseia no método de otimização de Arrefecimento Simulado[5] [61].

O problema da localização foi modelado em Arrefecimento Simulado da seguinte forma. A solução do problema é um vetor com Y posições, onde Y é o número de controladores que se deseja alocar na rede[6]. A cada iteração é gerada uma nova solução candidata que pode ser aceita caso tenha um custo menor do que a anterior. Se a solução não for melhor que a anterior, ela tem uma probabilidade associada a ela. Sorteia-se um número aleatório e verifica-se se o número é menor que a probabilidade de aceitação da solução. Em caso positivo, a solução é aceita mesmo tendo um custo maior do que a anterior. Esse comportamento é necessário para evitar que a otimização convirja para um mínimo local.

Dado o vetor solução retornado pela meta-heurística de Arrefecimento Simulado, dois algoritmos são aplicados sobre ele. O Algoritmo 1 mostra o mapeamento dos comutadores nos controladores do vetor solução. Esse algoritmo define as zonas de controle. O Algoritmo 2 mostra o cálculo da função objetivo que possui como resultado quantos nós da rede permanecem conectados mesmo em um cenário com

[5]O método de otimização de Arrefecimento Simulado foi escolhido como meta-heurística, pois esse método tem a convergência para o mínimo global comprovada, em tempo indeterminado, mesmo no cenário de decaimento linear de temperatura. Para tanto, a perturbação usada para a geração de novas soluções segue distribuição Cauchy.

[6]A heurística proposta considera que o número de controladores é decidido em função da relação de compromisso entre o atraso gerado por uma localização de controladores na rede e a resiliência provida por essa solução.

```
G = Grafo(topologia_da_rede)
controladoresArray = arranjo de controladores calculados
mapArray = arranjo [tamanho(G)] de inteiro
mapArray[k] = 0 ∀k ∈ G
for i ∈ tamanho(mapArray) do
    for j ∈ tamanho(controladoresArray) do
        if Distancia(controladoresArray[j], G[i]) <
        Distancia(mapArray[i], G[i]) then
            mapArray[i] = controladoresArray[j]
        end
    end
end
return mapArray
```

Algoritmo 1: Mapeamento dos comutadores em controladores pertencentes ao vetor solução. A função *Distancia* é definida como a latência entre os nós.

probabilidade de falha de nós.

O Algoritmo 2 recebe como entrada o grafo G, contendo toda a topologia da rede, um vetor de probabilidade *prob*, no qual cada posição i representa a probabilidade de falha do nó i da rede, e o vetor de mapeamento de comutadores em controladores. A saída do algoritmo é o inverso do percentual de nós da rede que permanecem conectados e com controle, mesmo no cenário de falhas representado pelo vetor *prob*. É importante ressaltar que o valor retornado é o inverso do percentual de nós que permanecem controlados, pois o problema de localização foi desenvolvido e modelado como um problema de minimização. Sendo assim, a minimização do inverso é a maximização dos nós que permanecem controlados mesmo no cenário de falhas.

4.5 Avaliação do Sistema Proposto

A proposta foi avaliada em duas etapas. A primeira etapa foi a implementação de um protótipo do controlador distribuído. O protótipo foi submetido a experimentos para verificar a consistência da visão global da rede e o ganho de desempenho ao distribuir o controle e submeter o sistema a altas taxas de requisição de fluxos/segundo. A segunda etapa de avaliação consiste na otimização da localização de controladores em diferentes topologias reais e, posteriormente, a localização otimizada encontrada é avaliada quanto à latência média entre controladores e comutadores e quanto à resiliência provida à rede. A heurística de otimização proposta é comparada com heurísticas com diferentes funções objetivo: menor latência entre controlador e comutadores, centroides da rede, menor número de saltos entre controlador e comutadores.

```
G = Grafo(topologia_da_rede)
mapArray = mapeamento de cada comutador em um controlador
prob = array com a probabilidade de falha de cada nó em G
for k ∈ G do
    aleatorio = random()
    if aleatorio ≤ prob[k] then
        G.remove(k)
    end
end
componentesConexas = conjunto_componentes_conexas(G)
soma = 0 %soma de todos os nos controlados em componentes conexas
for componente ∈ componentesConexas do
    for no ∈ componente do
        if mapArray[no] ∈ component then
            soma+ = 1
        end
    end
end
return tamanho(G)/soma %inverso do percentual da rede com controle
```

Algoritmo 2: Função objetivo da otimização correspondente à medida da partição da rede em cenário de falha.

O protótipo do controlador proposto, seguindo o modelo de Controlador Designado, considera uma rede com comutadores OpenFlow e implementa o controle distribuído sobre os controladores POX[7]. Os comutadores por software são computadores pessoais que executam o Open vSwitch controlados pelos controladores POX. O repositório de objetos globais foi implementado usando-se uma versão adaptada do módulo para a programação distribuída em Python DOPY[8]. Assim, o Controlador Designado escolhido, inicia um servidor em que os demais controladores se conectam a ele para manter a consistência do repositório de objetos globais. Vale ressaltar que para realizar os experimentos com o protótipo, as aplicações `forwarding.l2_learning` e `openflow.spanning_tree` do controlador POX foram adaptadas para registrar e consultar o repositório de objetos globais ao invés de manterem a visão de cada controlador somente local[9].

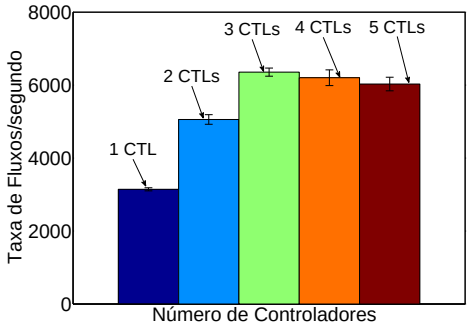
O protótipo do controlador foi executado como diversos processos em um computador pessoal Intel i7-2600 @ 3.40 GHz, com 16 GB de RAM, executando Debian Linux. Os comutadores OpenFlow são quatro computadores pessoais Core 2 Duo @ 2.40 GHz, com 3 GB de RAM, executando Debian Linux e o comutador por software com suporte a OpenFlow, Open vSwitch [10].

[7] http://www.noxrepo.org/pox

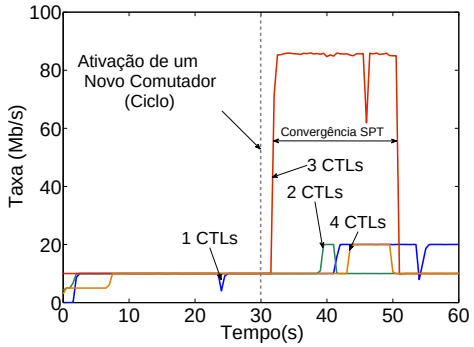
[8] http://www.mindhog.net/ mmuller/projects/dopy/.

[9] A implementação distribuída pode ser obtida através do contato com o autor.

[10] http://openvswitch.org/.



(a) Execução do `cbench` para 1, 2, 3, 4 e 5 controladores (CTLs). Taxa de fluxos/s aumenta até 3 controladores quando a capacidade de processamento requerida é maior do que a disponível na máquina de testes.



(b) Ativação de um novo comutador na rede, em 30 s, gerando um ciclo. Durante a convergência do algoritmo de árvore de cobertura (SPT) há duplicação de pacotes na rede.

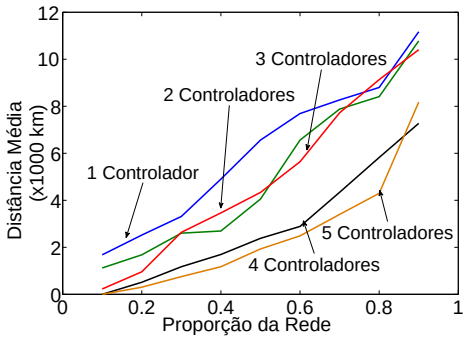
Figura 4.4: Avaliação do protótipo de controlador distribuído a) quanto ao número agregado de fluxos atendidos em função do número de controladores na rede e b) reação do controle distribuído à entrada de um novo comutador na rede. A duplicação de pacotes ocorre durante o tempo de convergência da árvore de cobertura no novo cenário.

A Figura 4.4 mostra a avaliação do controlador distribuído proposto. O primeiro experimento visa avaliar o número de fluxos/segundo que o plano de controle consegue atender. Para avaliar o plano de controle, foi usada a ferramenta `cbench`[11]. `Cbench` emula diversos comutadores conectados a um controlador e gera eventos de `packet_in` para medir a capacidade do controlador a reagir a esses eventos. Os resultados mostrados são médias com um intervalo de confiança de 95%. A Figura 4.4(a) evidencia que ao adicionar mais controladores ao plano de controle, a taxa de fluxos/segundo aumenta. Com três controladores, a taxa/fluxos por segundo atendida é quase o dobro de quando há somente um controlador na rede. Contudo, com 4 ou 5 controladores a taxa de fluxos atendidos é menor do que com três controladores, pois há concorrência entre os processos de controle e geração de fluxo e, portanto, há um gargalo de processamento, já que com 4 controladores os 8 núcleos (`cores`) do computador que hospeda o teste são usados para gerar fluxos e reagir aos eventos de `packet_in`. O segundo experimento verifica a reação do plano de controle à ativação de um novo comutador, fechando uma topologia em anel com os 4 comutadores. O comutador é ativado aos 30 s. Percebe-se que após a ativação há a duplicação de pacotes devido à inconsistência na árvore de cobertura. No pior caso, com três controladores, a convergência da árvore de cobertura chega a demorar 20 s, e converge para que todos os comutadores tenham acesso à visão global.

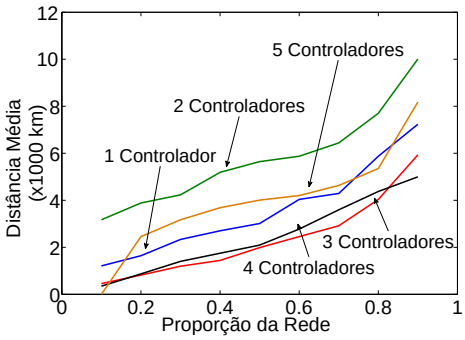
A avaliação da heurística de localização proposta é realizada através da simulação de falhas de nós sobre três topologias reais de rede: topologia da rede da RNP no

[11] http://www.openflowhub.org/display/floodlightcontroller/Cbench.

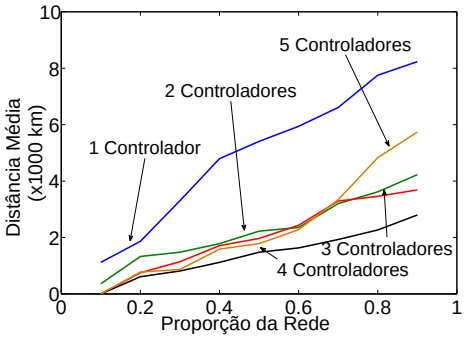
Brasil, da rede GEANT na Europa e da rede MPLS da AT&T nos Estados Unidos. Os grafos das topologias usadas foram obtidos no *The Internet Topology Zoo*[12]. A rede da RNP conta com 31 nós, a GEANT, com 40 nós e a AT&T conta com 25 nós. A otimização considera que cada nó na rede é capaz de suportar um controlador.



(a) Topologia da RNP no Brasil.



(b) Topologia da GEANT na UE.



(c) Topologia da AT&T nos EUA.

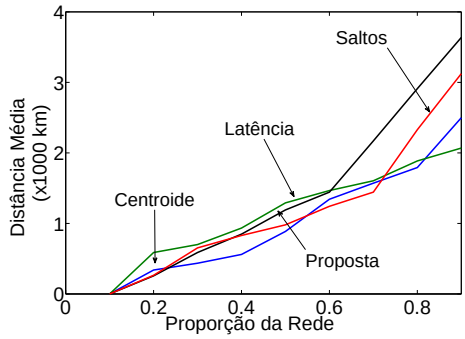
Figura 4.5: Latência média entre comutadores e o controlador a que foi mapeado em função do percentual da rede. Heurística proposta de localização considera maior resiliência e menor latência média. Os resultados mostram para cada topologia o uso de até cinco controladores.

Como o conjunto de dados disponibilizados não apresenta informação sobre latência entre os nós, a avaliação usa a distância física entre os nós como medida indireta da latência mínima entre os nós. Tal procedimento é aplicado de maneira semelhante por Heller *et al.* [20]. A Figura 4.5 mostra o comportamento da latência em função do percentual de nós na rede. A partir dessa figura, pode-se observar que ao adicionar o quinto controlador, a redução da latência não é tão significativa. No caso da topologia da AT&T, mostrada na Figura 4.5(c), com 5 controladores ocorre o processo inverso e há um aumento da latência. Portanto, os próximos resultados, por mérito de clareza, consideram somente os cenários com 4 controladores.

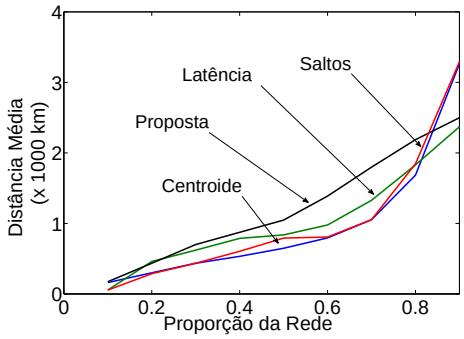
Os experimentos seguintes comparam a heurística proposta com três outras: menor latência entre controlador e comutadores (`Latência`), centroides da rede

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

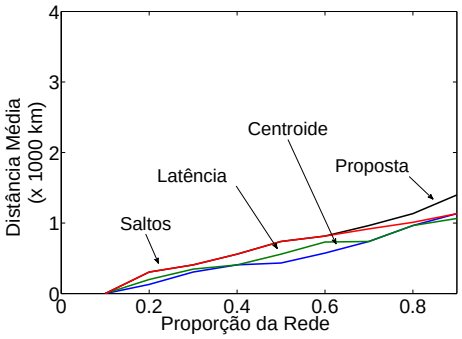
(`Centroide`), menor número de saltos entre controlador e comutadores (`Saltos`). A heurística de menor latência é inspirada na proposta de [20]. A heurística da escolha dos centroides da rede é baseada na estratégia de [62]. Por fim, a heurística de escolha dos controladores com o menor número de saltos até os comutadores é uma proposta simplista de comparação com as demais. As heurísticas foram usadas como função objetivo para a otimização do Arrefecimento Simulado.



(a) Topologia da RNP no Brasil.



(b) Topologia da GEANT na UE.

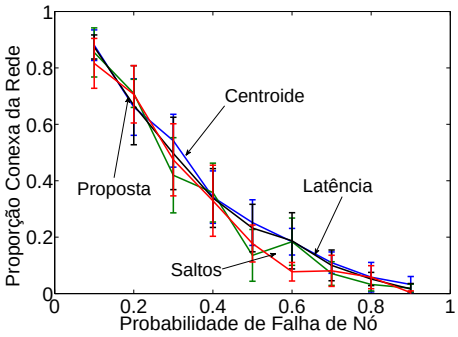


(c) Topologia da AT&T nos EUA.

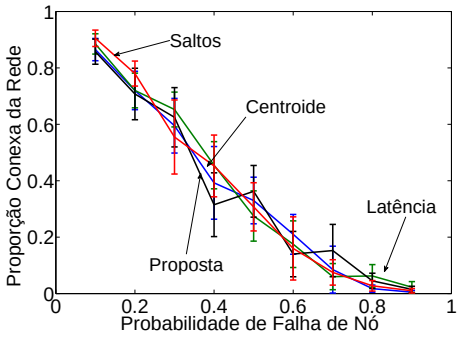
Figura 4.6: Latência média entre comutadores e o comutador a que foi mapeado em função do percentual da rede. A comparação é feita entre a `Proposta`, a menor latência (`Latência`), os centroides da rede (`Centroide`) e o menor número de saltos (`Saltos`). Todas as heurísticas consideram a localização de 4 controladores.

A Figura 4.6 mostra as quatro heurísticas comparadas quanto à latência média entre os controladores e os comutadores. Nas três topologias, evidencia-se que a heurística proposta apresenta um desempenho similar às demais. Vale observar, que a heurística do menor número de saltos entre comutador e controlador (`Saltos`), embora seja simplista, apresenta baixa latência média. Esse resultado é compatível com os cenários, pois os nós das topologias consideradas tendem conectarem diretamente com nós próximos.

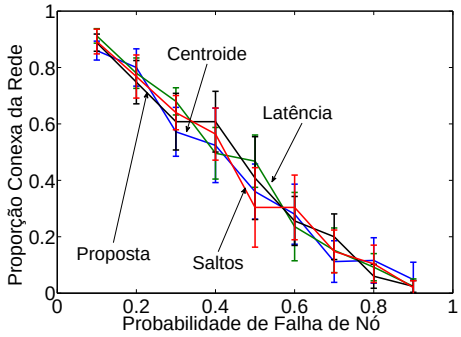
O último experimento avalia a resiliência da rede, quando a localização dos controladores é realizada através das quatro heurísticas avaliadas. Para tanto, foi desenvolvido um simulador de falhas na topologia. Os resultados da Figura 4.7 mostram a



(a) Topologia da RNP no Brasil.



(b) Topologia da GEANT na UE.



(c) Topologia da AT&T nos EUA.

Figura 4.7: Proporção da rede que continua conexa e controlada em função da probabilidade de falha dos nós. Comparação entre a `Proposta`, a menor latência (`Latência`), os centroides da rede (`Centroide`) e o menor número de saltos (`Saltos`). Todas as heurísticas consideram a localização de 4 controladores.

proporção da rede que continua conexa e controlada em função da probabilidade de falha dos nós da rede. Uma partição da rede é considerada funcional, caso a partição contenha algum nó no qual foi implantado um controlador. Essa suposição é feita, pois os comutadores dessa partição têm uma lista de controladores de salva-guarda que contem todos os controladores da rede, ordenados[13].

Os resultados das Figura 4.7(a) e 4.7(b) demonstram que a heurística proposta tende a manter uma maior proporção da rede conexa quando a probabilidade de falhas dos nós é até 0.5. Contudo, o ganho é limitado quando comparado com outras heurísticas, pois as topologias consideradas não apresentam redundância capaz de suportar falhas de múltiplos nós. Assim, mesmo com uma probabilidade pequena de falha de nós, as partições geradas na rede ficam sem controlador e, portanto, são consideradas como falhas também. Evidencia-se também que a heurística simples de alocação de controladores em nós com menor número médio de saltos até os demais (`Saltos`) apresenta bons resultados de resiliência. A heurística simples `Saltos`, com

[13]Considera-se o uso do OpenFlow 1.2, ou superior, para a definição de papéis entre controladores, permitindo a configuração de controladores de salvaguarda.

menor tempo de convergência, é então uma boa opção em topologias com baixo grau de redundância.

4.6 Trabalhos Relacionados

Esta seção foca nas diferentes propostas de como se realizar o controle fisicamente distribuído em Redes Definidas por *Software*. Levin *et al.* argumentam que a distribuição do controle físico, enquanto o controle lógico mantém-se centralizado, prejudica o desempenho da rede, quando as aplicações de controle centralizado são agnósticas quanto à distribuição do estado [7]. Os autores identificam a existência de duas relações de compromisso. A primeira é entre o desempenho das aplicações de controle e a sobrecarga de distribuição de estados entre controladores. A segunda relação é entre a complexidade da lógica das aplicações de controle e a robustez contra inconsistência. Por sua vez, Schmid e Suomela defendem que há aplicações que podem ser executadas por controladores locais, sem que necessitem de uma visão global consolidada [19]. Schmid e Suomela identificam dois tipos de planos de controle, o plano de controle horizontal e o plano de controle hierárquico. No plano de controle horizontal cada controlador é responsável por uma área disjunta dos demais e a estrutura se organiza a partir de restrições administrativas. Já no plano de controle hierárquico, os controladores se organizam verticalmente, convergindo para uma visão global da rede conforme se sobe na hierarquia. Localidade refere-se às operações que podem ser resolvidas localmente em reposta a eventos que aconteçam na sua vizinhança de um salto. Os autores defendem que em uma SDN com controle hierárquico, certas operações, como a verificação de uma árvore de cobertura, podem ser resolvidas localmente e, portanto, podem ser tratadas por controladores locais, mais próximos dos comutadores.

A presença de um controlador fisicamente centralizado não está intrinsicamente ligada à arquitetura SDN. Portanto, há propostas de controladores fisicamente distribuídos, mas que mantêm a visão global da rede. A proposta ONIX age como um *middleware* para sistemas distribuídos, propagando as informações para os controladores físicos, e fornece uma API (*Application Programming Interface*) para as aplicações se comunicarem entre as diferentes réplicas e controlarem a rede [17]. Baseado nos mesmos preceitos do controlador ONIX, há também a proposta ONOS [16]. O controlador ONOS baseia-se no controlador Floodlight[14] e distribui o estado da rede com o registro distribuído Zookeeper [63]. Outro controlador distribuído é o HyperFlow [30]. Esse, por sua vez, cria um canal de propagação de eventos para sincronizar todos os controladores da rede. Os eventos no HyperFlow são divulga-

[14]Floodlight é um controlador OpenFlow disponível em: http://www.projectfloodlight.org/floodlight/.

dos através do modelo Publicador/Assinante (*Publish/Subscriber*). A cada evento de rede em um controlador, o evento e os dados relativos a ele são publicados em uma classe de eventos no canal de sincronização. As demais aplicações, inclusive em outros controladores, que dependam de uma classe de eventos para funcionarem, subscrevem tal classe e, a partir de então, recebem todas as atualizações que forem publicadas nessa classe de eventos. Tais propostas se baseiam na distribuição do estado entre todos os controladores, mas não preveem uma política de localização e otimização do número de controladores na rede.

Yeganeh *et al.* argumentam que a escalabilidade do controle de Redes Definidas por *Software* pode ser alcançada através de controladores multiprocessados ou através de diversos controladores distribuídos [64]. A proposta Kandoo é um controlador SDN hierárquico em que o escopo das aplicações é dividido em local ou global [31]. Aplicações locais e globais coexistem no controle da rede. As aplicações locais são as que podem operar somente com o estado local de cada comutador e, portanto, são implantadas nos controladores mais próximos do plano de dados. Mais próximas dos comutadores, as aplicações podem tratar mais rapidamente as requisições mais frequentes e evitam a sobrecarga no restante da rede de controle. Por sua vez, as aplicações globais são executadas por controladores no topo da hierarquia. Essas aplicações necessitam da visão global da rede para operar. Os controladores raízes também agem como mediadores na comunicação de controladores locais. Contudo, uma desvantagem do Kandoo é que no momento da programação das aplicações, deve-se definir se ela se comportará como local ou global. A consistência da visão global do Kandoo é fraca e pode estar desatualizada dado o processamento local das aplicações.

A otimização da localização e o número de controladores para atender uma Rede Definida por *Software* também é um desafio de pesquisa atual. Müller *et al.* propõem uma estratégia de localização de controladores, denominada *Survivor*, que considera a diversidade de caminhos até o controlador, a capacidade dos controladores e os mecanismos de recuperação de falhas [15]. A estratégia se baseia em um modelo de Programação Linear Inteira para encontrar a localização ótima dos controladores. O modelo considera que a localização ótima é a que maximiza a conectividade entre controladores e comutadores da rede. Paralelamente, Zhang *et al.* também propõem uma estratégia de localização de controladores para aumentar a resiliência da rede [62]. A ideia central de Zhang *et al.* consiste em minimizar a probabilidade de falha em cada partição de controle da rede. Para tanto, os autores usam um algoritmo de *min-cut* para minimizar duas funções. A primeira minimiza a distância de todos os nós até seu controlador, resultando em uma minimização intra-clusters. A segunda minimiza a conectividade inter-cluster, reduzindo o número de enlaces passíveis de falha entre controlador e comutadores. Bari *et al.* também propõem um

esquema de otimização da localização de controladores [21]. Assim como Müller *et al.*, Bari *et al.* se baseiam em Programação Linear Inteira para otimizar a localização. Contudo, esta proposta realiza o provisionamento dinâmico de controladores, instanciando e desligando controladores conforme haja necessidade. Para tanto, a proposta monitora a rede continuamente e reage conforme haja mudança no perfil de tráfego. No entanto, o provisionamento dinâmico não considera a resiliência da rede, mas sim a capacidade dos controladores e, portanto, em caso de falha, a realocação de controladores só ocorre no próximo intervalo de monitoramento. Por sua vez, Ros e Ruiz defendem que uma rede definida por *software* confiável instala controladores de acordo com uma restrição de confiabilidade de rede. Ros e Ruiz propõem uma heurística para calcular o estimador de confiabilidade de rede [60].

A proposta ElastiCon cria uma arquitetura de controle distribuído para Redes Definidas por Software em que o número de controladores disponíveis aumenta e diminui de acordo com a carga de uso da rede [65]. A proposta se baseia em uma abordagem de duas fases. A primeira fase faz o balanceamento de cargas periodicamente otimizando o mapeamento entre controladores e comutadores, durante a execução da rede. No segundo momento, é verificado se algum controlador teve o seu limiar máximo de carga extrapolado. Em caso positivo, um novo controlador é adicionado ao conjunto de controladores ativos. De forma análoga, se um controlador atinge uma carga menor que o limiar mínimo de uso, um controlador é desligado e retirado do conjunto de controladores ativos. A distribuição do controle é realizada por um repositório comum de dados, acessível a todas às réplicas de controladores. A proposta apresenta ainda um protocolo de migração de comutador entre controladores. Contudo, o protocolo não é resiliente à falha do controlador ou comutador e depende que todas as mensagens sejam recebidas em ordem.

Bari *et al.* também propõem um provisionamento dinâmico de controladores em uma Rede Definida por Software [21]. Os autores definem o problema do provisionamento dinâmico de controladores que consiste em adaptar o número de controladores disponíveis de acordo com a carga da rede. Contudo, diferentemente da proposta ElastiCon, Bari *et al.* focam na modelagem do problema de otimização, propondo um modelo de Programação Linear Inteira que considera tanto o mapeamento entre controladores e comutadores, como também a otimização da carga nos controladores, em uma única etapa. Por fim, Bari *et al.* propõem uma heurística para a solução do problema modelado, enquanto ElastiCon foca na implementação do controlador distribuído e na proposta de um protocolo de migração de comutadores entre controladores.

Diferente de redes tradicionais, em que uma falha no plano de controle pode ser corrigida através da convergência dos protocolos de roteamento, em uma Rede Definida por Software, a recuperação de falhas demanda uma lógica própria para

cada aplicação [66]. Nesse sentido, as SDNs carecem de um mecanismo automático de recuperação de falhas. Assim, Kuźniar *et al.* propõem que ao se detectar uma falha, um novo controlador é iniciado, para garantir a resiliência da rede, mesmo no cenário de falha do controlador [66]. Contudo, o novo controlador passa a controlar apenas um ambiente emulado, o qual é constituído por uma replicação da topologia real da rede a exceção somente dos elementos que falharam. Após a convergência do novo plano de controle no ambiente emulado, o novo controlador assume o controle da rede real, com a visão global convergida [66]. Essa abordagem, no entanto, depende do forte sincronismo entre o ambiente emulado e a rede real. Ademais, o mecanismo proposto não apresenta garantias de consistência entre o controle do ambiente emulado e real, assim como não garante a consistência das políticas de encaminhamento.

No caso de uma falha em um controlador, a resiliência de Redes Definidas por Software é comprometida, pois o monitoramento e a recuperação de falhas dependem do envolvimento direto do controlador no tratamento de mensagens de monitoramento [67]. Em geral, o monitoramento é realizado através de mensagens LLDP enviadas na rede e tratadas pelo controlador logicamente centralizado. Uma alternativa proposta por Kempf *et al.* é implementar uma função de monitoramento diretamente nos comutadores OpenFlow, que passam a emitir mensagens de monitoramento sem que haja a interferência do controlador [67], gerando menos sobrecarga sobre controlador. A proposta se baseia no uso do OpenFlow em redes de transporte, redes em que o tráfego é agregado e o tempo necessário para respostas a falhas de no máximo 50 ms e, portanto, leva a uma sobrecarga estimada de 100 mensagens de monitoramento por segundo para cada túnel na rede. O funcionamento proposto é que o próprio comutador envie as mensagens de prova em cada túnel na rede e, no caso de a extremidade do túnel parar de receber provas periodicamente, reportar a falha para o controlador que aplica uma rota alternativa de encaminhamento para o fluxo.

Jarschel *et al.* propõem um modelo de avaliação de desempenho para redes OpenFlow baseado em teoria de filas [57]. O modelo proposto considera uma rede de um controlador e um comutador. O modelo simples é usado para avaliar o impacto da probabilidade de um pacote ser enviado ao controlador no atraso total do encaminhamento de pacotes. Por sua vez, Azodolmolky *et al.* propõem um modelo baseado em cálculos de rede (*network calculus*) [58]. A ideia central é avaliar o atraso de pior caso no encaminhamento de pacotes, ou seja, o atraso quando a fila do controlador está cheia. No entanto, esse trabalho considera somente um comutador como o plano de dados da rede SDN. Chilwan *et al.* estendem o modelo de teoria de filas proposto por Jarschel *et al.*, pois consideram um Modelo de Jackson para realizar a retroalimentação entre o tráfego do controlador e do comutador [56].

O modelo proposto por Chilwan *et al.* é mais acurado que o anterior, pois não considera que o tráfego injetado por um controlador no comutador possa voltar a ser tratado pelo controlador. Chilwan *et al.* considera ainda um cenário em que mais de um comutador gera carga de requisições no controlador. Todavia, os modelos propostos não consideram como o tempo de resposta do controlador à chegada de novas requisições pode ser afetado pelo número de comutadores que esse controlador está encarregado.

Pries *et al.* avaliam a escalabilidade e o uso de OpenFlow em centro de dados [68]. Para tanto, os autores consideram modelos de tráfego para centros de dados e, a partir desses modelos, simulam uma rede OpenFlow considerando os parâmetros obtidos de implementações reais. Os resultados obtidos indicam que, de acordo com o modelo de tráfego de cada centro de dados, o uso de controladores distribuídos é necessário para garantir um tempo de tratamento dos pacotes que satisfaça as restrições de atraso em cada cenário avaliado. Por sua vez, a proposta OFLOPS desenvolve um arcabouço de avaliação para comutadores OpenFlow [59]. A ideia principal do arcabouço é medir o desempenho de comutadores OpenFlow através da medição de quantas operações de fluxo por segundo o comutador é capaz de fazer. A medição se baseia em diversos processos concorrentes que geram pacotes, capturam pacotes, traduzem pacotes OpenFlow em eventos, realizam controle SNMP e fazem o gerenciamento de tempo.

A proposta de controle distribuído desse trabalho é uma arquitetura com controle horizontal, cuja localização maximiza o número de caminhos dos comutadores ao controlador, mas também minimize a latência entre controlador e comutadores de uma partição de controle. Tais funções objetivo são importantes, pois a maximização da quantidade de caminhos entre controlador e comutadores garante resiliência a falhas de enlaces, enquanto a minimização da latência influi diretamente no tempo de configuração de novos fluxos. Assim, a proposta deste trabalho age tanto na resiliência quanto na eficiência da rede, enquanto outras propostas atacam apenas uma das vertentes.

4.7 Conclusão do Capítulo

As Redes Definidas por *Software* (SDN) utilizam a centralização lógica do plano de controle para criar uma abstração da visão global da rede que facilita o a gerência, a implantação e o controle de novos serviços no núcleo da rede. Esse capítulo propõe uma arquitetura de controladores distribuídos para SDN que mantém a visão global da rede, ao passo que cria zonas de controle com controle distribuído. A visão global da rede é alcançada através da ideia de Controlador Designado, que é um controlador de zona que assume o papel de manter a consistência da visão global da rede. O

controlador designado mantém um repositório de objetos globais que é acessível a todos controladores. O capítulo também propõe duas heurísticas de otimização da localização dos controladores: uma baseada na maximização da resiliência da rede e, outra, mais simples, baseada na minimização do número de saltos entre controlador e comutadores. Um protótipo do controlador proposto foi implementado e avaliado. Os resultados mostraram que a visão global é mantida no cenário de controle fisicamente distribuído e todos os controladores a acessam e convergem no cálculo de uma árvore de cobertura comum. As heurísticas de otimização foram avaliadas e os resultados mostraram que a heurística proposta aumenta a resiliência da rede, em especial quando a taxa de falhas é de até 0.5, mas para topologia com menor redundância, a heurística mais simples, o menor número médio de saltos, apresenta desempenho comparável às demais.

Capítulo 5

Modelo de Redes Definidas por Software e o Problema de Consistência

Neste capítulo apresenta-se um modelo de Redes Definidas por *Software*, levando-se em consideração o problema da consistência na atualização de políticas. O problema de consistência é apresentado sob duas vertentes. A primeira considera a consistência do tratamento do fluxo enquanto ativo na rede, no qual deve ser garantido que o mesmo estado da rede é mantido para todo pacote durante o trânsito deles na rede da origem ao destino, chamada de consistência por pacote. A segunda vertente da consistência é representada pela consistência entre controladores da rede. Nesse segundo caso, objetiva-se que o estado da rede conhecido por todos controladores seja o mesmo. Assim, para formalizar o problema e entender as limitações de cada vertente, este capítulo apresenta um modelo formal simplificado de Redes Definidas por Software proposto por Reitblatt *et al.* [22] e estendido por Canini *et al.* [33, 69].

A formalização teórica apresentada neste capítulo baseia o desenvolvimento do simulador para Redes Definidas por /textitSoftware proposto neste trabalho. O simulador desenvolvido implementa as componentes do modelo teórico como objetos na linguagem Python e adota a ideia de simulação de eventos discretos para calcular cada passo de simulação.

5.1 Modelagem para 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 no modelo formal de Redes Definidas por Software adotado nesse capítulo se baseia na aplicada por Reitblatt et al. [22], 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 dada por $[x_1, \cdots, x_n]$, em que $[\,]$ representa uma lista vazia e $xs_1 + + xs_2$ designa a concatenação das listas xs_1 e xs_2.

No modelo formal são definidas algumas estruturas básicas. A primeira estrutura é o pacote, representado por pk, que é a unidade de transmissão de dados na rede, formado por uma sequência de `bits`. A próxima estrutura é a porta, p, que 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 ou destroem pacotes encaminhados.

O plano de dados considerado nesse capítulo 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[1] [33, 69].

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 [22]. Portanto, o processamento dos pacotes é modelado como a composição de duas funções mais simples, encaminhar um pacote no comutador e mover

[1] 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.

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 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 e a de fila de portas. Um traço t é uma lista de pacotes localizados que mantém o registro da sequência dos saltos que o pacote atravessa na rede. A função Q define uma fila de porta, ou seja, que 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 [22].

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:

$$override(S, u) = S' sendo\ S'(p, pk) = \begin{cases} u(p, pk)\ se\ (p, pk) \in dom(u) \\ S(p, pk),\ caso\ contrário. \end{cases} \tag{5.1}$$

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 $N \xrightarrow{us} \star N'$ 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 Pr da rede se todos os traços t em Q aparecem no conjunto Pr:

$$\forall t \in Q \rightarrow t \in Pr, \tag{5.2}$$

onde Pr 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.

5.2 Atualização do Plano de Controle

Uma propriedade que deve ser observada em Redes Definidas por *Software* é a consistência das atualizações no plano de controle. A consistência na atualização é uma propriedade independente da distribuição de controladores no plano de controle e 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 [32]. Contudo, a atualização atômica não é suficiente para garantir a consistência da rede, 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 $\sim$ nos traços. A equivalência $\sim$ considera que dois traços são equivalentes se os pacotes nos dois traços são equivalentes. A ideia de atualização consistente, portanto, prevê que os traços gerados pela rede no momento de uma atualização sejam equivalentes a traços ou pela configuração inicial ou pela configuração final.

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

- estado inicial Q,
- execuções $(Q, C_1) \xrightarrow{us} \star(Q', C_2)$,
- 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 Pr 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 $\sim$, 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 [22].

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 $C_2 = (S[u_1, \cdots, u_k], T)$ a nova configuração da rede e seja $us = [u_1, \cdots, u_k]$ 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), apareçam no domínio de uma das funções de atualização $[u_1, \cdots, u_k]$,

então us é uma atualização única de C_1 para C_2. Nesse caso, a atualização é $\sim$-consistente por pacote [22].

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). O potencial das atualizações únicas é aumentado quando se combinam atualizações únicas com atualizações não observáveis.

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 $C_2 = (S[u_1, \cdots, u_k], T)$ 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 [22].

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 composição $us_1 + +us_2$ 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 em Duas Fases. A atualização em 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*. De maneira formal, 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 em 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 $us = [u_1^i, \cdots, u_m^i, u_1^e, \cdots, u_m^e]$ 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.* [22, 33, 69].

Contudo, a implementação na prática da atualização em duas fases exige a marcação dos pacotes com a versão da configuração. Reitblatt *et al.*, Canini *et al.* e Pereśíni *et al.* argumentam que uma implementação válida para a marcação de versão é o uso de marcadores de VLAN [22, 33, 69]. A marcação dos pacotes, no entanto, gera uma sobrecarga de controle para a verificação dos marcadores válidos e exige a mudança 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.

5.3 Consistência do Plano de Controle Distribuído

Ao contrário do cenário em que o plano de controle é centralizado e as atualizações de políticas e regras de encaminhamento são realizadas por uma única entidade (controlador), no caso do plano de controle distribuído, as atualizações podem ter múltiplos autores. Em especial, políticas atualizadas concomitantemente por múltiplos atores podem apresentar sobreposições que afetam a sua ordenação em série e, portanto, podem gerar inconsistências no controle da rede. Nesta seção, considera-se a abstração de Composição Consistente de Políticas (*Consistent Policy Composition* - CPC). A abstração CPC é definida por Canini *et al.* como uma interface capaz de aceitar requisições de atualização de políticas para a rede e assegurar que as atualizações de políticas só afetam o tráfego da rede se for possível definir uma composição sequencial das requisições. No caso de não haver uma ordenação sequencial possível, as requisições de atualização são abortadas sem interferir no tráfego [33]. A ideia básica é que os pacotes atravessem a rede sob exatamente uma única visão global consistente, mesmo no período em que as políticas estão sendo atualizadas. Vale ressaltar que o conflito entre políticas também deve ser resolvido para que as políticas sejam compostas em série.

5.3.1 Modelo do Plano de Controle Distribuído

O plano de controle distribuído é modelado de forma análoga à de Canini *et al.* [33, 69]. Considera-se o plano de controle distribuído como um conjunto $p_1, \cdots, p_n$ de controladores, onde $n \geq 2$ *controladores*. Os controladores estão sujeitos a falhas e, considera-se que pelo menos um controlador se mantém correto,

ou seja, sem falhar. O modelo assume também que os controladores são capazes de se comunicarem entre eles, de maneira confiável e assíncrona, com troca de mensagens, não necessariamente ordenadas. O modelo de plano de controle distribuído proposto por Canini *et al* considera ainda que os controladores acessam uma abstração de consenso que os permitem implementar uma máquina de estados replicada e tolerante a falhas, que provê a implementação da serialização das operações [69]. Embora essa premissa seja forte, a premissa simplifica o modelo e deixa o desafio de prover tal abstração de consenso para a implementação do plano de controle.

O modelo de rede considerado para o controle distribuído é semelhante ao descrito anteriormente. Contudo, novos conceitos e premissas são adicionadas ao modelo anterior. Assim, assume-se que uma porta suporta a execução atômica de operações de leitura, escrita e modificação de regra. De maneira formal, uma porta i suporta a operação $update(i, g)$, em que g é uma função definida em conjuntos de regras. A operação realiza uma leitura atômica no estado da porta i e, dependendo do estado, usa a função (regra) g para atualizar a porta e retorna uma resposta.

Outro conceito a ser adicionado ao modelo de controle centralizado é o de política de rede. Uma política de rede, π, é definida por: um domínio, $dom(\pi) \subseteq \Pi$, um nível de prioridade, $pr(\pi) \in \mathbb{N}$, e um caminho único e uma sequência de portas conectadas sem laços, para cada porta de entrada i que se aplica aos pacotes no domínio $dom(\pi)$. Para cada porta de entrada i e cada pacote $pk \in dom(\pi)$ que chega na porta i, π especifica uma sequência de portas distintas $i_1, \cdots, i_s$ que pk deve seguir, em que $i_1 = i$: $(i_j, i_{j+1}) \in L$, $j = 1, \cdots, s-1$ e $i_s \in \{World,\ Drop\}$, que assegura que um pacote encaminhado eventualmente sai da rede ou é descartado.

Duas políticas π e π' são independentes se $dom(\pi) \cap dom(\pi') = \varnothing$. Duas políticas estão em conflito se π e π' não são independentes e $pr(\pi) = pr(\pi')$. Caso $pr(\pi) > pr(\pi')$, π tem prioridade sobre a política π'. Um conjunto de políticas, U, é não-conflitante se não há quaisquer duas políticas em U que estejam em conflito. A prioridade é usada para estabelecer a ordem entre as políticas que não estão em conflito, mas têm domínios sobrepostos, isto é, um pacote $pk \in dom(\pi) \cap dom(\pi')$, em que $pr(\pi) > pr(\pi')$, é processado pela política π. Assim, as políticas não-conflitantes em um conjunto U podem ser compostas. A composição de políticas consiste em que um pacote, que chega a uma porta, seja processado segundo a política de maior prioridade $\pi \in U$, tal que $pk \in dom(\pi)$. Tal funcionamento é o adotado pelo OpenFlow quando há mais de um fluxo ao qual o pacote se adequa [9].

O tráfego no sistema é modelado usando eventos de $inject$, para a entrada de um novo pacote na rede, e $forward$, para o encaminhamento do pacote na rede, definidos como:

- $inject(pk, j)$ adiciona um novo pacote pk na porta j, através da inclusão de pk no fim da fila de Q_j, ou pela substituição de Q_j por $Q_j + +pk$;

- $forward(pk, j, pk', k),\ j \in P$ representa que o primeiro pacote em Q_j é processado de acordo com S_j. Sendo $Q_j = pk + +Q'$, após $forward(pk, j, pk', k)$, Q_j assume o valor de Q' e $Q_k = Q_k + +pk'$, em que $r(pk) = (pk', k)$ e r é a regra de maior prioridade em S_j que pode ser aplicada a pk.

Cada controlador p_i apresenta um *algoritmo*, ou seja, uma máquina de estados que: aceita a chamada de operações de alto nível; acessa portas para realizar operações de leitura, modificação e escrita; comunica com outros controladores e responde a chamadas de alto nível. O algoritmo distribuído gera uma sequência de *execuções* composta de acesso a portas, chamadas, respostas e eventos de encaminhamento de pacotes. Dada uma execução de um algoritmo, uma *história* é a sequência de eventos observáveis externamente, como eventos $inject$ e $forward$, assim como chamadas e respostas de operações entre controladores.

Assume-se um escalonador justo e canais de comunicação confiáveis entre os controladores. Assim, em um tempo infinito de execução, todos os pacotes em uma fila são tratados por um evento de $forward$ e todas as mensagens entre controladores são eventualmente recebidas.

Define-se que um *problema* é um conjunto $\mathcal{P}$ de histórias. Um algoritmo resolve um problema $\mathcal{P}$ se a história de todas as suas execuções estiver em $\mathcal{P}$. Um algoritmo $f - resiliente$ resolve $\mathcal{P}$ se a história de todas as execuções $f - resilientes$ estiver em $\mathcal{P}$. Uma execução f-resiliente é quando até f controladores chegam ao fim da computação com passos finitos. Uma solução $(n-1)$-resiliente é chamada de livre de espera (*wait-free*).

Em uma história H, todos os pacotes inseridos na rede geram um traço, uma sequência de pacotes localizados. Cada evento $ev = inject(pk, j)$ em E resulta em um (pk, j) como um primeiro elemento do traço, $forward(pk, j, pk_1, j_1)$ adiciona (pk_1, j_1) ao traço e cada próximo evento $forward(pk_k, j_k, pk_{k+1}, j_{k+1})$ adiciona o elemento (pk_{k+1}, j_{k+1}) ao traço, a menos que $j_k \in \{Drop, World\}$, pois esse caso indica a terminação do traço. Seja $\rho_{ev,H}$ o traço correspondente ao evento de entrada de um pacote na rede, $ev = inject(pk, j)$, em uma história H, o traço $\rho = (pk_1, i_i), \cdots, (pk_n, i_n)$ é consistente com a política π se $pk_1 \in dom(\pi)$ e $(i_1, \cdots, i_n) \in \pi$.

O plano de controle requer a informação de quais políticas estão ativas e quais já perderam a validade. Políticas ativas, nesse caso, referem-se a políticas que ainda possuam pacotes em trânsito. Para tanto, assume-se o modelo do oráculo, em que cada controlador pode consultar o oráculo quais marcações de políticas, *tags*, ainda estão em uso por pacotes em qualquer que seja a fila. As premissas[2] sobre o modelo do oráculo são as mesmas de Canini *et al.* e são mínimas dado que o oráculo pode

[2]Canini *et al.* assumem que o oráculo tem a visão global da rede atualizada e consistente durante toda a execução [69].

ter interações assíncronas [69]. Na prática, as marcações disponíveis podem ser simplesmente estimadas assumindo um limite superior para o tempo de trânsito de um pacote na rede.

5.3.2 Problema de Composição Consistente de Políticas

A abstração de Composição Consistente de Políticas (*Consistent Policy Composition* – CPC), aceita requisições concorrentes de atualização e assegura que as requisições afetem o tráfego em uma composição sequencial das políticas. Todos os controladores p_i aceitam requisições, $apply_i(\pi)$ em que π é uma política, e retorna ack_i, no caso de a requisição ser efetivada com sucesso, ou $nack_i$, no caso de a requisição ser abortada [69].

Uma relação de ordem parcial sobre os eventos da história H é definida como $<_H$. Uma requisição req precede a requisição req' na história H, representado por $req <_H req'$, se a resposta de req aparece antes da chamada de req' em H. Se uma requisição não precede a outra, diz-se que as requisições são concorrentes. De forma similar, um evento ev de entrada de pacote na rede precede uma requisição req em H, representado por $ev <_H req$, se ev aparece antes da invocação de req em H. Paralelamente, ev sucede req em H, $req <_H ev$, se ev ocorre após a resposta de req. Dois eventos ev e ev' de entrada de pacotes na rede, através de uma mesma porta, estão relacionados por $ev <_H ev'$ se ev precede ev' em H. Um evento ev é concorrente com uma requisição req se $ev \not<_H req$ e $req \not<_H ev$. A história H é sequencial se em H não existe duas requisições concorrentes, nem um evento de entrada de pacote concorrente com uma requisição.

Seja $H|p_i$ a história local do controlador p_i, uma subsequência de H composta por todos os eventos de p_i. Assume-se que todos os controladores são bem-formados, que a história local $H|p_i$ é sequencial, ou seja, nenhum controlador aceita uma nova requisição caso não haja a resposta da anterior. Uma requisição realizada por p_i é completa em H se a requisição é sucedida por sua resposta (ack_i ou $nack_i$) em $H|p_i$. Caso contrário, a requisição é incompleta. Uma história é completa se todas as requisições são completas em H. A completação de uma história H é uma história completa H', equivalente a H exceto pelo fato de que as requisições incompletas em H são completadas com um ack, caso a requisição já tenha afetado algum pacote, ou com um $nack$, caso nenhum pacote tenha sido afetado pela requisição, inserido após sua chamada. Duas histórias H e H' são equivalentes se H e H' têm o mesmo conjunto de eventos, $H|p_i = H'|p_i, \forall i$ e para todos os eventos ev de entrada de pacote em H e H': $\rho_{ev,H} = \rho_{ev,H'}, \forall ev \in \{H \cup H'\}$.

Uma história sequencial completa H é legal se duas propriedades são satisfeitas:

- uma política é aplicada com sucesso em H se e somente se não está em conflito

com nenhuma outra política aplicada previamente em H;

- para cada evento de entrada de pacote na rede $ev = inject(pk, j)$ em H, o traço $\rho_{ev,H}$ é consistente com a composição de todas as políticas aplicadas com sucesso que precedam o evento ev em H.

Definição: Histórias compostas sequencialmente. Uma história completa é capaz de ser composta sequencialmente se existe uma história sequencial legal S, tal que H e S sejam equivalentes e $<_H \subseteq <_S$. Isso implica que o tráfego em H é processado como se as requisições fossem efetivadas automaticamente e todo pacote que entra na rede é processado instantaneamente (consistência por pacote [22]). A propriedade de legalidade requer que somente requisições bem sucedidas afetem os pacotes. A história equivalente sequencial S deve respeitar a ordem em que as requisições não-concorrentes foram efetivadas e a ordem em que os pacotes chegaram em H.

Definição: Composição Consistente de Políticas. Um algoritmo resolve o problema de Composição Consistente de Políticas se para cada uma de suas histórias H, existe uma completação H', tal que respeita duas propriedades:

- **Consistência:** H' é sequencialmente possível de ser composto.
- **Terminação:** Se H é infinito, então todo controlador p_i correto que efetiva a requisição $apply(\pi)$, eventualmente retorna uma resposta (ack_i ou $nack_i$) em H.

Vale ressaltar que para uma história infinita H, as propriedades de consistência e de terminação implicam que uma requisição incompleta causa somente o aborto de requisições conflitantes por um determinado período de tempo, pois eventualmente a requisição será abortada ou efetivada em uma completação de H e, no caso de ela ser abortada, as requisições seguintes não serão mais afetadas.

Canini *et al.* apresentam dois algoritmos para resolver o problema de composição consistente de políticas [69]. A solução mais simples de Canini *et al.* é o algoritmo livre de espera FixTag, que ordena as políticas em uma dada porta de entrada de pacotes na rede e segue tal ordem para compor as políticas. A ideia do algoritmo FixTag é numerar as políticas para serem instaladas, obedecendo um ordenamento geral da rede. Portanto, a complexidade desse algoritmo é linear com o número de políticas a serem instaladas. Outra solução proposta por Canini *et al.* é o algoritmo ótimo ReuseTag, f-resiliente, com complexidade $f + 2$, em que f designa o número de controladores que podem falhar durante o processo de efetivação de políticas.

A ideia básica do FixTag é codificar cada possível caminho da rede em uma única marcação, *tag*. Assim, seja τ_k a marcação do k-ésimo caminho, a regra $r_{\tau_k} =$

(pk, i_{x+1}) é instalada para encaminhar qualquer pacote marcado com τ_k para a próxima porta no caminho $x + 1$. Ao receber uma nova requisição de mudança de política π e antes de instalar qualquer regra, um controlador p_i envia uma mensagem para todos os demais controladores informando-os das regras que ele irá adicionar na porta de entrada. Todos os controladores ao receberem a mensagem, a retransmitem para garantir confiabilidade à transmissão, e começam a instalar as regras em prol de p_i. Esse procedimento garante que toda atualização de políticas eventualmente será completada. Assim, sejam $i_1, \cdots, i_s$ as portas de ingresso de pacotes na rede, π^j o caminho especificado pela política π para as portas de ingresso $i_j, talque j = 1, \cdots, s$. Para instalar π, o algoritmo FixTag procura adicionar uma regra para cada porta de ingresso i_j. A regra marca todos os pacotes que combinem com o domínio da política com a marcação (tag) que representa o caminho π^j. No entanto, como diferentes políticas de diferentes controladores podem estar em conflito, todos os controladores atualizam suas portas de ingresso em uma ordem pré-determinada. A ordem de atualização, predeterminada e globalmente conhecida, garante que as políticas sejam efetivadas em série e, no caso de um conflito, esse é logo determinado no início do processo de instalação da política. Assim, os conflitos são descobertos ainda nas portas de menor ordem e a instalação da política sem conflitos é garantida pelo modelo de efetivação de tudo-ou-nada, evitando instalações parciais de políticas.

Vale ressaltar que o algoritmo FixTag não requer nenhum retorno da rede sobre quando os pacotes chegam ou deixam a rede, dado que o algoritmo simplesmente marca os pacotes na extremidade da rede e, internamente, os pacotes só são encaminhados de acordo com a marcação. Contudo, a composição de políticas prevista no algoritmo FixTag resume-se a serialização da efetivação das políticas, dado que a ordem de efetivação predeterminada no algoritmo já escalona as políticas concorrentes para serem efetivadas de maneira sequencial. Portanto, o algoritmo FixTag depende de um agendamento prévio da ordem de efetivação de políticas, o que pode limitar o seu funcionamento e o desempenho ao aumentar o número de políticas recebidas.

Canini *et al.* provam o funcionamento do algoritmo FixTag a partir do teorema que propõe que o FixTag resolve o problema de CPC sem gerar esperas indeterminadas, sem confiar em um oráculo ou em objetos de consenso [69]. Contudo, a complexidade do algoritmo é alta, dado que a complexidade de mensagens, ou seja, número de mensagens enviadas, é alta, já que o algoritmo realiza duas inundações na rede para cada atualização de política. A complexidade de marcações usadas é exponencial em relação ao tamanho da rede, dado que marca todos os caminhos possíveis.

O outro algoritmo para realizar a composição consistente de políticas proposto por Canini *et al.* é o ReuseTag, que permite que os controladores utilizem dinamica-

mente $f+2$ marcações através da coordenação entre controladores. Em um cenário em que não há falhas ($f = 0$), o protocolo usa somente duas marcações, usando somente um bit para armazenar a marcação.

A máquina de estados do protocolo é replicada nos nós, o que impõe uma ordem global nas políticas de atualização e assegura o uso coordenado e o reuso das marcações do protocolo. A máquina de estados usada no protocolo exporta para cada controlador p_i duas operações: $push(i, \pi)$, que coloca a política π na fila de políticas a instalar; e $pull(i)$ que retorna $\perp$ ou uma tupla (π, tag), em que π é a política a ser tirada da pilha e instalada com a $tag \in \{0, \cdots, f+1\}$.

Um controlador p_i chama $push(i, \pi)$ para colocar a política π na fila de políticas a serem instaladas; o controlador p_i chama $pull(i)$ para buscar na fila de políticas a próxima a ser instalada. A invocação de *pull* retorna $\perp$ se todas as políticas inseridas na fila já foram instaladas e há uma marcação, *tag*, disponível, caso contrário, retorna (π, tag), informando p_i que a política π deve ser marcada com *tag*.

Seja S uma execução sequencial da máquina de estados, seja $\pi_1, \cdots, \pi_n$ a sequência de políticas propostas em S como argumentos de $push()$ na ordem em que aparecem. Se S contém exatamente k operações $pull(i)$ não triviais, ou seja, que não retornam $\perp$, então é dito que p_i realiza k operações de *pull* não triviais em S. Se S contém $pull(i)$ que retorna $(\pi, t) \neq \perp$, seguido por um $pull(i)$, então π está instalado em S, pois se π não é confirmada como efetivada ou instalada, (π, t) permanece na fila de políticas a serem instaladas.

A marcação τ_k está bloqueada ao fim de uma história finita S, se S contém $pull(i)$ que retorna (π_{k+1}, τ_{k+1}), mas não contém um $pull(i)$ subsequente. Nesse caso, p_i bloqueia a marcação τ_k ao fim de S. Um controlador instalando a política π_{k+1} bloqueia a marcação associada à política anterior π_k, ou da política inicialmente instalada no caso de $k = 0$. Assim, Canini *et al.* especificam a máquina de estados sequencial do algoritmo ReuseTag com base nos seguintes requisitos de S [69]:

- **Não-trivialidade:** Se p_i realiza k operações de *pull* não triviais, então a operação $pull(i)$ subsequente retorna $\perp$ se e somente se a operação de *pull* é precedida por no máximo k operações de *push* ou no mínimo $f+1$ mais políticas estão bloqueadas em S. Em outras palavras, a k-ésima operação de *pull* de p_i deve retornar alguma política se no mínimo k políticas foram previamente colocadas na fila ($push$) e no máximo f políticas estão bloqueadas.

- **Acordo:** Para todo $k > 0$, existe $\tau_k \in \{0, \cdots, f+1\}$ tal que se os controladores p_i e p_j realizam k operações de *pull* não triviais, então $\pi_{i,k} = \pi_{j,k} = \pi_k$ e $\tau_{i,k} = \tau_{j,k} = \tau_k$ para algum τ_k. Assim, os controladores computam a mesma ordem em que as políticas propostas devem ser instaladas, com a mesma marcação de sequência.

- **Validade da Marcação:** Para todo k, τ_k é valor mínimo em $\{0, \cdots, f+1\} - \{\tau_{k-1}\}$ que não está bloqueado em $\{0, \cdots, n-1\}$ quando a primeira operação de $pull(i)$ que retorna (π_k, τ_k) é realizada. As marcações são escolhidas de maneira determinística baseadas nas marcações que não estão bloqueadas no momento. Pela propriedade de Não-Trivialidade, no máximo f políticas estão bloqueadas e $\{0, \cdots, f+1\} - \{\tau_{k-1}\}$ não é vazio.

Para instalar a política $\tilde{\pi}$, o controlador p_i insere $\tilde{\pi}$ na fila de políticas, chamando $push(i, \tilde{\pi})$. Em paralelo, para instalar as suas políticas e as de outros controladores, o controlador chama a operação de $pull(i)$ até um valor não trivial (diferente de $\perp$) ser retornado (π_k, τ_k), onde k é o número de operações *pull* não triviais realizadas até o momento por p_i. O controlador verifica se π_k não está em conflito com políticas previamente instaladas. Caso esteja em conflito, p_i espera até que o tráfego na rede tenha somente a marcação τ_{k-1}, ou seja, a marcação τ_{k-2} não está mais sendo usada. Para determinar o se a marcação está em uso ou não, p_i usa o oráculo, que é um objeto global de consenso.

O controlador tenta instalar π_k em todas as portas internas e, após, nas portas de entrada, uma a uma, em uma ordem pré-definida, adotando a estratégia de atualização de duas fases [22]. A atualização de uma porta interna p é realizada usando uma operação atômica que adiciona a regra associada com π_k, marcada com τ_k, ao conjunto de regras atualmente instaladas em p. A atualização na porta de ingresso dos pacotes na rede p simplesmente substitui a regra atualmente instalada pela nova regra que marca o tráfego de entrada com a marcação τ_k, o que acontece com sucesso se e somente se a porta, no momento de instalação da nova política, apresenta a política com a marcação τ_{k-1}; caso contrário, a porta não é modificada. Uma vez que todas as portas foram atualizadas, as regras antigas são removidas, uma a uma, das portas internas.

Um controlador bloqueando a marcação τ_k pode estar envolvido na instalação de τ_{k+1} e, então, a marcação τ_k não pode ser reusada para outra política que não seja π_k. Caso o bloqueio não seja respeitado, o controlador mais lento pode acordar e atualizar a porta com uma regra desatualizada. No entanto, como um controlador mais lento ou controlador com falha pode bloquear no máximo uma marcação, então eventualmente deve haver, no mínimo, uma marcação disponível em $\{0, \cdots, f+1\} - \{\tau_{k-1}\}$ quando o primeiro controlador realizar a k-ésima operação de *pull* não trivial.

5.4 Conclusão do Capítulo

O objetivo desse capítulo é verificar a consistência no plano de controle de Redes Definidas por Software, tanto no cenário de controle centralizado, como no cenário

de controle distribuído. Para tanto, esse capítulo apresentou um modelo formal de consistência em Redes Definidas por Software, proposto por Reitblatt *et al.* [22], e estendido para o cenário de múltiplos controladores distribuídos por Canini *et al.* [33, 69].

A proposta de consistência de Reitblatt *et al.*, denominada Atualização de Duas Fases, foca na consistência da atualização das políticas na rede em um cenário com apenas um controlador e, portanto, a consistência por pacote é definida e alcançada baseada em como os pacotes são tratados na rede. A Atualização de Duas Fases divide a rede entre portas de entrada e portas internas. Prova-se que ao realizar a atualização nas portas internas e, depois, nas portas de entrada, a consistência por pacote é garantida. Contudo, a Atualização de Duas Fases exige que pacotes sejam marcados ao ingressarem na rede, o que gera uma complexidade de gerenciamento de marcações livres e necessita que os pacotes sejam modificados para a adição ou modificação da marcação.

Por fim, o capítulo apresenta dois esquemas para a atualização de políticas em Redes Definidas por Software com controle distribuído. Para a instalação consistente de políticas adota-se a atualização de duas fases. Contudo, para definir as marcações e a ordem em que as políticas concorrentes são instaladas, Canini *et al.* propõem os algoritmos FixTag e ReuseTag. A composição de políticas no algoritmo FixTag realiza a serialização da efetivação das políticas, através da ordem de efetivação predeterminada no algoritmo, que escalona as políticas concorrentes para serem efetivadas de maneira sequencial. O algoritmo FixTag depende de um agendamento prévio da ordem de efetivação de políticas, o que pode limitar o seu funcionamento e o desempenho ao aumentar o número de políticas recebidas. Por sua vez, o algoritmo ReuseTag se vale de um objeto global de consenso para gerenciar as marcações disponíveis e gerenciar uma pilha global de operações a serem realizadas.

O modelo de redes definidas por *software* discutido nesse capítulo é usado como base para a construção de um simulador de redes e, também, como formalização teórica das propostas de atualização consistente apresentadas nesse trabalho.

Capítulo 6

Os Esquemas de Atualizações Consistentes para Redes Definidas por Software

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 [70, 71]. Este capítulo propõe dois esquemas distintos para lidar com a atualização de políticas em uma rede definida por *software*. O primeiro esquema, chamado Atualização reversa, foca na efetivação das atualizações de políticas em uma rede com controle centralizado. O esquema proposto não depende da marcação de pacotes com a versão da configuração da rede, tal como os esquemas de atualização consistentes apresentados na literatura [22, 23, 33, 69]. A seguir, discute-se um esquema de atualização de políticas para o cenário em que o controle da rede é distribuído. Nesse cenário, propõe-se um protocolo de consistência para serializar as atualizações de políticas e para compor as políticas, evitando conflitos. A formalização dos esquemas propostos se baseia no modelo de rede discutido no Capítulo 5.

6.1 O Esquema Proposto de Atualização Reversa para Plano de Controle Centralizado

A proposta deste capítulo, 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 Open-

Flow [72]. 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 o mecanismo de atualização baseado no algoritmo de migração de fluxos: 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:

$$\forall (p, pk) \in dom(u_i), \nexists (p, pk) \in dom(u_j), \text{para } i < j \leq k, \tag{6.1}$$

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 inverso do destino para a origem. Assim, para todos:

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

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 1: 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. Pela definição de Atualização Reversa, se $(p, pk) \in Dom(us_i)$, então $(p, pk) \notin dom(u_{i+1})$, logo, basta verificar que:

- 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 [22], 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, em um cenário em que não há a o controle distribuído da rede.

```
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
```

Algoritmo 3: Algoritmo de Atualização Reversa. As tabelas dos comutadores são atualizadas na sequência inversa do sentido do fluxo de pacotes.

Uma implementação do esquema de Atualização Reversa, proposto e provado ser consistente, é formalizado no Algoritmo 3. 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.

Vale ressaltar que a definição de uma Atualização Reversa está restrita a atualizações de políticas que atuam em caminhos, anterior e novo, disjuntos e sem laços na rede. Além disso, a composição de caminhos novos e antigos deve ser livre de laços. Levando essas restrições em consideração, a Atualização Reversa se atende cenários de atualizações de caminho e de ação nos caminhos de fluxo. Ao se considerar os comutadores que processam pacotes através de várias tabelas, a Atualização Reversa permanece mantém o mesmo comportamento, já que o fluxo de processamento, *pipeline*, age como um percurso de processamento sem laço em cada comutador. O esquema atua em cada comutador no caminho inverso, atualizando cada tabela na ordem inversa do *pipeline* de processamento de pacotes. Outra consideração importante é que, como entradas de fluxo curinga têm granularidades diferentes, atualizações de políticas podem incorrer na definição de políticas sobrepostas. A atualização de políticas sobrepostas é um desafio complexo [34]. A composição de políticas é abordada na Seção 6.2.

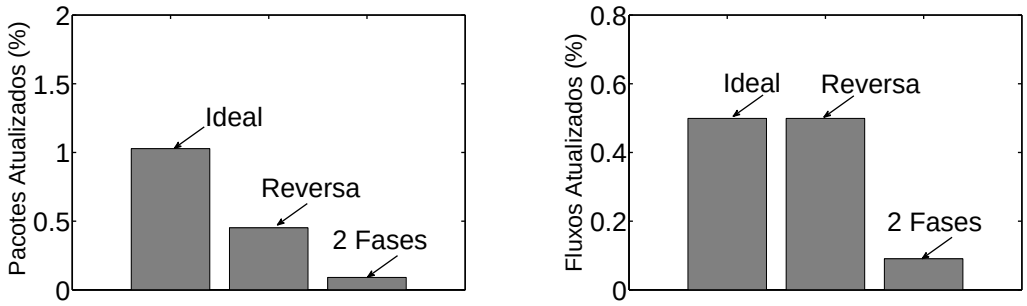
6.1.1 Resultados Experimentais

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 capítulo e proposto por [22]. 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*[1]. 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 ($\sigma = 2$) [5, 73]. 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 [22] (`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 [32]. Contudo, a atualização atômica pode

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

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. A Atualização Ideal é consistente por pacote, considerando-se o conceito de consistência relaxado, já que permite que um pacote seja tratado por mais de uma configuração de rede, desde que seja sempre mais atual que a anterior. 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.



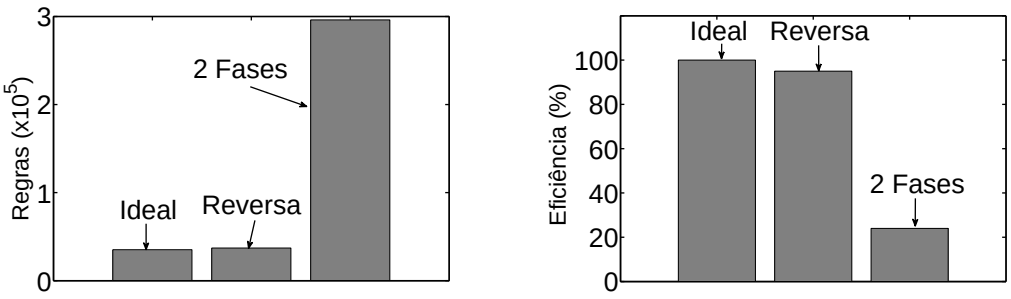
(a) Percentual de pacotes que são encaminhados por duas configurações distintas de rede.

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

Figura 6.1: 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 inesperado. 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 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 [22]. Com o relaxamento do conceito proposto nesse capítulo, 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 mais próximo da Atualização Ideal, mostrado na Figura 6.1(a), indicando um menor tempo de reação à atualização quando compa-

rada 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 6.1(b), enquanto, por causa do 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) Similaridade comparada de cada esquema de atualização em relação à Ideal.

Figura 6.2: O mecanismo proposto de Atualização Reversa em relação ao mecanismo de Atualização de Duas Fases: a) acarreta menor sobrecarga de regras geradas e b) apresenta maior similarida em relação à Atualização Ideal.

Outro ponto importante da avaliação é o número de regras instaladas nos comutadores. A Figura 6.2(a) compara o número de regras instaladas por cada esquema durante toda a execução da simulação. O número de regras pode ser interpretado como o número de entradas na tabela TCAM (*Ternary Content Addressable Memory*) dos comutadores na rede. É possível perceber que o número de regras instaladas pela Atualização de Duas Fases é praticamente oito vezes superior ao 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. As regras adicionais instaladas pela Atualização de Duas Fases agem tanto nos fluxos ativos quanto nos fluxos inativos que ainda estão nas tabelas dos comutadores. Quanto maior for o tempo para a expiração dos fluxos nas tabelas, maior é o número de novas regras a cada atualização, pois é gerada uma nova versão para cada fluxo ainda presente na tabela de fluxos dos comutadores. Por fim, avaliou-se a similaridade de cada esquema de atualização em relação ao Ideal. A similaridade é definida neste capítulo 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 similaridade de 94%, enquanto a Atualização de Duas Fases apresenta 24%, mostrado na Figura 6.2(b). A baixa similaridade da Atualização de Duas Fases em

relação ao Ideal é consequência de a Atualização de Duas Fases adotar um conceito de consistência restritivo que assume a impossibilidade de se realizar atualizações atômicas.

6.2 O Esquema de Consistência e Acordo em Plano de Controle Distribuído Proposto

O plano de controle logicamente centralizado consiste em uma abstração de uma visão rede global compartilhada por todos os controladores da rede [74]. 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) [75], 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.

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 [75, 76]. 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

$$req <_{H_{c_i}} req' \rightarrow req <_{H_{c_j}} req', \tag{6.2}$$

$\forall \{req, req'\} \subset H_{c_i} | req <_H req'$ 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 [75]:

- 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 apresentam regras que agem em conjuntos de fluxos sobrepostos [34, 77]. 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

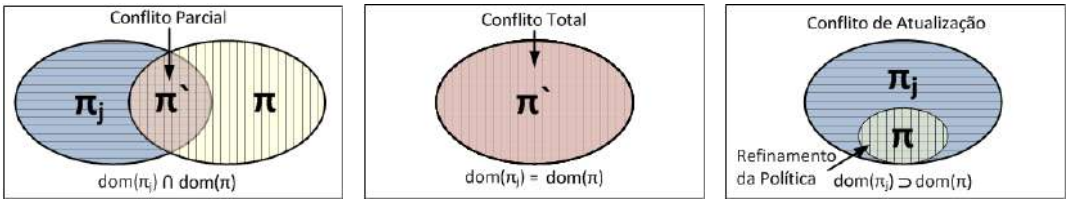
$$dom(\pi) \cap dom(\pi_j) = \emptyset, \forall\ \pi_j \in \Pi. \tag{6.3}$$

É possível identificar três tipos de conflitos: conflito parcial, conflito total e conflito de atualização. O conflito parcial é quando o domínio de uma política a

[2]O domínio de uma política é o conjunto de fluxos que a política afeta, o subespaço de fluxo (partição do *flowspace*) [22, 77].

ser instalada sobrepõe ao domínio da política atual. Nesse caso, a composição das duas políticas pode levar à criação de regras de manipulação de pacotes para cada subconjunto, como mostrado na Figura 6.3(a). O conflito total ocorre quando duas políticas diferentes têm o mesmo domínio. Nesse caso, mostrado na Figura 6.3(b), a nova política substitui a anterior, ou a nova política é totalmente rejeitada. O terceiro caso é o conflito de atualização, Figura 6.3(c). De fato, o conflito de atualização 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.

Além de aceitar consistentemente atualizações de políticas e de compô-las, o procedimento de atualização de política deve aplicar as atualizações de forma consistente por pacote [22, 75]. Assim, um pacote em trânsito não pode ser tratado por mais do que uma configuração de rede e os eventos de entrada de pacote na rede são discretos e serializados com as demais requisições.



(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 6.3: Casos possíveis de conflitos entre as regras da nova política π com as regras de uma política já instalada π_j no espaço de fluxos S: a) Conflito parcial, cuja solução é a criação de um subconjunto de políticas π'; b) Conflito total, cuja solução é a avaliação das duas políticas e a consequente instalação de uma nova política π' que considere os casos em conflito; c) Conflito de Atualização, em que 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.

6.2.1 O Protocolo de Consistência e o Algoritmo de Composição de Políticas

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 [22]. No caso de controladores distribuídos, o uso da atualização de duas fases simplifica o procedimento de instalação da nova política na rede em relação à atualização reversa, pois não necessita da sinalização entre os controladores para indicar quando um controlador já atualizou sua parte do caminho do pacote na rede para, então, o próximo poder realizar a atualização em seu domínio de controle. 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 capítulo concentra-se no primeiro e segundo passos.

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 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 6.4(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 6.4(b).

Execução do protocolo. Como mostrado na Figura 6.4(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

[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.

acordo com o esquema de atualização de duas fases (*Two-Phase Update*) [22]. 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.

*Caso 1: Mais de um controlador iniciador.*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.

Caso 2: Falha no controlador iniciador. 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.

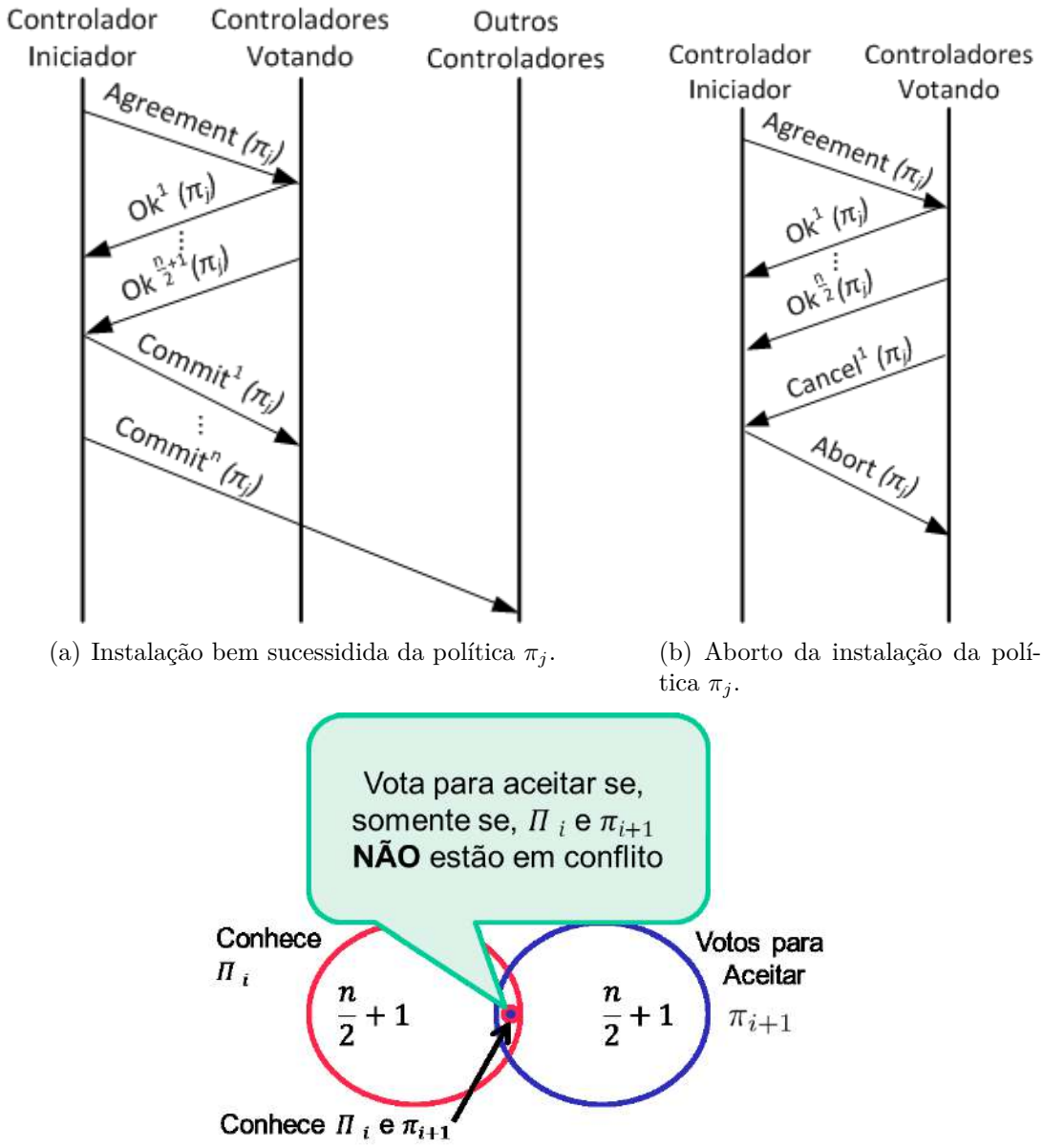
Caso 3: Um controlador qualquer falha. O caso de uma falha de um controlador é tratado de duas formas diferentes. Primeiro, se um controlador falha após receber a mensagem de `agreement`, o controlador iniciador aguarda sua resposta até que um tempo limite ou, se a falha interrompe a conexão TCP, o iniciador 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.

Caso 4: Um controlador está desatualizado. 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 [78] e Zab [79], 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 4. 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 6.4(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 6.3, 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



(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 6.4: 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.

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.

O Algoritmo 4 verifica se os domínios de atuação das políticas se sobrepõem. Contudo, o algoritmo pode ser aplicado, sem perda de generalidade ou corretude, com

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

Algoritmo 4: 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.

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` [80].

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 2: 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, $\pi_1 <_H \pi_2$. Escolhe-se então um controlador qualquer, c_j, que é compatível com a ordem global, em que a ordem local é $\pi_1 <_{Hc_j} \pi_2$. Como 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 o controlador c_j obteve $n/2 + 1$ votos para instalar π_1 antes de π_2. De acordo com o protocolo, um controlador não pode votar em duas 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.

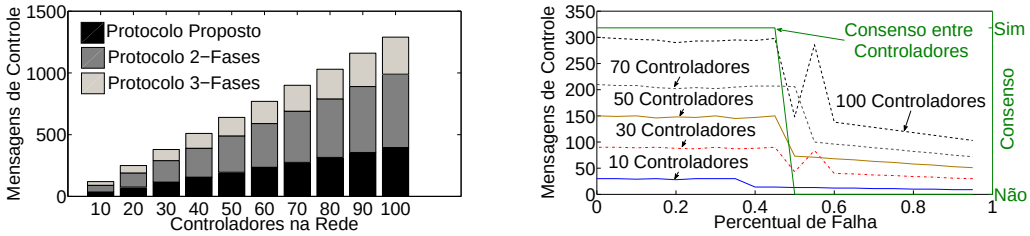
Teorema 3: A composição das políticas é livre de conflitos.
Prova por indução. O mecanismo de indução é usado sobre o conjunto Π, que representa o conjunto de todas as políticas instaladas na rede.
Caso base ($dom(\Pi_0) = \emptyset$): Nesse caso, o conjunto Π é vazio e, então, é trivialmente um conjunto 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 2, 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 3, mostrando que a composição de Π_i com π_{i+1} só é possível se não houver conflitos.

6.2.2 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.* [76, 81].

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 6.5(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 proposto 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) Número de mensagens para instalar uma atualização.

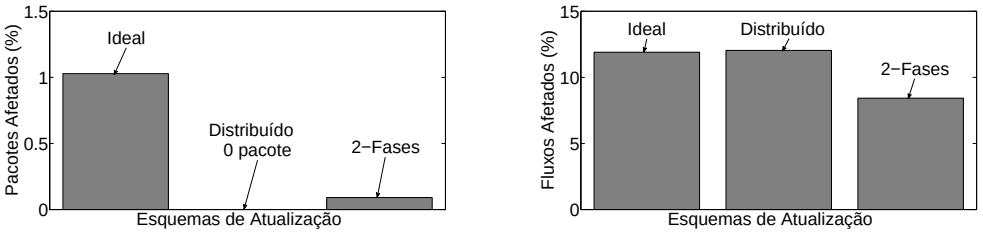
(b) Mensagens enviadas em cenários de falhas.

Figura 6.5: 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 6.5(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 6.5(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 resposta 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 6.5(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 ($\sigma = 2$) [76, 81]. A chegada de fluxos acontece durante os 900 primeiros passos de simulação, cada fluxo 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.

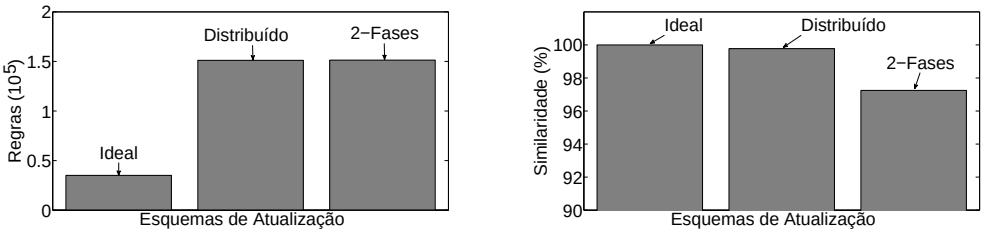
Figura 6.6: Impacto das atualizações nos pacotes e fluxos encaminhados na rede.

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 [22]. 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 6.6(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

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

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 [81]. 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 afetados pelas atualizações na rede, mostrado na Figura 6.6(b). O esquema distribuído atualiza 42% mais fluxos do que o centralizado de atualização de duas fases.



(a) Número total de regras instaladas na rede.

(b) Similaridade com o ideal.

Figura 6.7: 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.

O número total de regras instaladas nos comutadores da rede é evidenciado na Figura 6.7(a). Essa métrica indica o quanto da memória dos comutadores é usado por cada esquema de atualização. A métrica é dada pelo número de entradas que cada esquema de atualização insere nas tabelas de fluxos dos comutadores. Como os esquemas de atualização distribuído e de atualização de duas fases centralizado instalam regras no núcleo da rede para garantir a consistência por pacote, o número de regras instalado por esses esquemas chega a ser 4x superior ao do ideal[5]. Por sua vez, a Figura 6.7(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

[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.

à coordenação eficiente de ações entre os controladores com o uso do protocolo de consistência proposto.

6.3 Trabalhos Relacionados

As seções anteriores focam no controle de acesso e na distribuição do controle. Nesta seção o foco é a consistência do estado global do plano de controle e na consistência do tratamento dos pacotes em Redes Definidas por *Software* em uma implementação com controladores distribuídos. Vale ressaltar que os controladores são responsáveis por alterar os estados dos fluxos nos comutadores do plano de dados e todos pacotes que se encontram no caminho da origem para o destino devem seguir regras e políticas consistentes. O teorema CAP (*Consistency, Availability, and Partition tolerance* - Consistência, Disponibilidade e Tolerância a Partições) propõe uma relação de compromisso entre a linearização, a disponibilidade e a tolerância a partições em sistemas distribuídos. O teorema define, e é provado, que em um sistema distribuído é impossível prover simultaneamente consistência, disponibilidade e tolerância a partições [24]. Vale ressaltar que o teorema considera um sistema distribuído de propósito geral. Panda *et al.* interpretam o teorema CAP (*CAP Theorem*) sob a perspectiva de Redes Definidas por Software[82]. Nesse sentido, os autores identificam a validade do teorema no cenário de SDN e avaliam possíveis medidas que asseguram consistência, disponibilidade e tolerância à partição em uma SDN. A abordagem de Panda *et al.*, no entanto, garante a disponibilidade e a tolerância à partição na rede de controladores, considerada uma rede separada da rede de dados. A consistência na definição de políticas de encaminhamento é garantida ao se considerar, por hipótese, que as políticas de encaminhamento são estáticas. Assim, mesmo quando há partições na rede de controle, não há o problema de o conjunto de políticas de cada controlador estar desatualizado, pois, ao se assumir políticas estáticas, não há mudanças de política.

Uma das primeira propostas para realizar a divisão do controle em uma rede OpenFlow foi o FlowVisor [51]. O FlowVisor é uma camada de virtualização do plano de controle da rede OpenFlow. O FlowVisor usa o protocolo OpenFlow para controlar o rede física subjacente e fornece uma abstração para os controladores de que a rede é dedicada a um único controlador, particionanado os recursos da rede de acordo com cinco dimensões primárias: isolamento de banda, descoberta de topologia, engenharia de tráfego, monitoramento de CPU (*Central Processing Unit*) e controle das tabelas de encaminhamento. O FlowVisor define uma partição de rede como um conjunto de fluxos executando sobre uma topologia de comutadores. A topologia conhecida pelo controlador é limitada pelo FlowVisor, que intercepta a comunicação do controlador com a rede OpenFlow, de forma semelhante a um

proxy transparente. O isolamento de banda é garantido por um valor mínimo de banda que cada fluxo de uma partição tem acesso. A divisão das tabelas de fluxos em todos os comutadores é realizada controlando quantas entradas cada partição da rede está usando nas tabelas de fluxo em um dado instante. Cada partição da rede é representa, nesse contexto, por um controlador OpenFlow.

A proposta OF.CPP argumenta que o tratamento 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 [32]. Assim, em OF.CPP, os erros mais comuns às SDNs são identificados: instalação de laços na rede, amplificação de regras e tratamento inconsistente de pacotes. Nesse sentido, os autores defendem a ideia de usar conceitos de transações, tais como as de banco de dados, para o tratamento dos pacotes na rede. O principal conceito defendido é o de ACID (Atomicidade, Consistência, Isolamento e Durabilidade). O acrônimo ACID define que a instalação de um fluxo deve ser: Atômica, Consistente, Isolada e Durável. Contudo, a proposta OF.CPP relaxa os requisitos e propõe um mecanismo para gerar registros (*logs*) de transações e guardar estados anteriores no controlador. Com isso, quando há identificação de que alguma transação gerou uma inconsistência no estado global da rede, o estado global da rede é revertido e garante-se que a rede volta a um estado consistente. A consistência é garantida pelo seguinte princípio. Todas as transações são registradas. No momento da efetivação da transação, essa é efetivada se todas as demais transações são de somente leitura ou se não há conflito entre leitura e escrita de dados no controlador, ou seja, conflitos entre uma transação que realiza uma escrita de dados enquanto outra transação está lendo os dados modificados. Vale ressaltar, que os recursos observados para a verificação de conflitos são estruturas de dados do próprio controlador.

Seguindo a mesma linha de garantir a consistência no plano de controle de 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 [22]. 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 de duas fases. A primeira fase consiste em atualizar todos os comutadores da rede, com exceção dos comutadores de ingresso do pacote na rede. Nesses comutadores, novos fluxos são gerados com as características dos pacotes, porém com o marcador da versão atualizado para o valor mais novo. Portanto, esses fluxos novos são adicionados às tabelas dos comutadores em paralelo aos fluxos com o marcador antigo. 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 da rede. Na implementação dos autores, o marcador de versão usado foi a etiqueta de VLAN, dada a simplicidade de implementação no OpenFlow. O mecanismo de atualização em duas fases é provado que funciona de acordo com o modelo de rede OpenFlow definido.

Por sua vez, a proposta NICE (*No bugs In Controller Execution*) define uma ferramenta para a busca de erros em aplicações OpenFlow [23]. 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 realizar a asserção do código da aplicação. A aplicação OpenFlow é tratada pelo NICE como um modelo de transições que podem ser disparadas a cada estado da rede. O estado da rede é definido como o conjunto de variáveis conhecido pelo controlador. O funcionamento do NICE é semelhante ao de outros verificadores de modelo, tal como o SPIN para Promela, porém com algumas otimizações específicas para o uso em redes OpenFlow. O NICE é capaz de realizar a verificação de mais estados que o SPIN sem que haja a explosão do uso de memória ou de processamento. Algumas otimizações são o armazenamento de apenas os *hashes* dos estados já visitados, em detrimento de armazenar todas as variáveis conhecidas no estado, e a verificação de quais estados são recorrentes para evitar recolocá-los na lista de estados a visitar. Contudo, mesmo com as otimizações propostas, a verificação de consistência é difícil para redes grandes. O experimento realizado mostra uma rede de apenas dois comutadores e duas estações realizando no máximo 5 `pings`. Aplicando o verificador NICE, os autores mostram que existem erros de quebra de propriedades importantes mesmo em aplicações OpenFlow conhecidas.

A consistência na atualização de políticas em SDN também é abordada por Canini *et al.* [83]. Os autores argumentam que a atualização de políticas concomitante 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 de atualização de políticas transacional 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. Para realizar a consistência por pacote do estado da rede, para os pacotes em trânsito, Canini *et al.* adotam ideia semelhante à de Reibaltt *et al.*, marcando os fluxos com identificadores de VLAN para a versão da configuração em uso [22].

Canini *et al.* estendem a proposta de atualização de duas fases a uma Rede Definida por Software com controle distribuído [33]. 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.

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 [84]. 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 [85]. 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 [86]. A proposta da Atualização Reversa, por sua vez, define o conceito relaxado de consistência por pacote e realiza a atualização das regras na rede na ordem inversa dos fluxos, garantindo assim a consistência e evitando a necessidade de marcação de pacotes com etiquetas de versão [76, 81]. Essas propostas focam em Redes Definidas por Software com um controle centralizado.

Fergunson *et al.* propõem que as políticas em uma SDN sejam definidas de maneira hierárquica [77]. A ideia de políticas hierárquicas é adequada para cenários em

que múltiplas entidades (usuários ou administradores de rede) atualizam as políticas de encaminhamento da rede. Na proposta, a hierarquia facilita a resolução de conflitos e permite a delegação de autoridade sobre fluxos. Fergunson *et al.* definem uma semântica para avaliar as políticas e inseri-las nas árvores de políticas. Assim, são propostos dois grupos de funções. O primeiro grupo de funções verifica a qual política mais específica um pacote a ser encaminhado combina. O segundo grupo de funções é formado por operadores de resolução de conflitos entre políticas na árvore. A proposta de políticas hierárquicas se assemelha ao FlowVisor [51]. Contudo, a ideia central do FlowVisor é compartilhar uma rede OpenFlow entre distintos controladores, cada um controlando um sub-espaço de fluxos bem definido, enquanto as políticas hierárquicas permitem a divisão do controle em mais alto nível, através da declaração de políticas, inclusive superpostas, e não somente de regras de encaminhamento.

No caso de nós distribuídos, a ideia mais simples de consenso é decidir se uma transação deve ser confirmada ou não [87]. 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.

Para a consistência de fluxos, esse trabalho propôs 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 foca em controladores centralizados e 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. Para controladores distribuídos, este trabalho propôs um protocolo simples e eficiente para a serialização da instalação de atualizações de políticas em Rede Definidas por Software. A ideia principal é garantir o consenso entre os controladores quanto à aplicação de uma nova política sobre a rede e se a nova política pode ser composta com as demais já instaladas.

6.4 Conclusão do Capítulo

Atualizações de políticas de forma consistente em uma Rede Definida por *Software* é um desafio, pois deve-se garantir que um pacote em trânsito na rede sempre

encontre uma visão global mais recente do que às demais pela qual já foi processado. Considerando a complexidade da Atualização de Duas Fases, esse capítulo propõe a Atualização Reversa. O esquema proposto considera que o pacote possui um caminho na rede entre a origem e o destino, formado por uma sequência de comutadores, onde cada comutador é um salto de encaminhamento entre a origem e o destino do pacote, e a atualização de políticas é realizada no sentido inverso do caminho dos pacotes na rede. Prova-se que se a atualização de políticas quando é realizada no sentido do caminho inverso em que os pacotes atravessam a rede, as propriedades dos fluxos são mantidas e que a atualização é consistente por pacote. Vale ressaltar que esse esquema possui a vantagem de não exigir a marcação de pacotes. 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 94% de similaridade quando comparada à Atualização Ideal.

Em um plano de controle distribuído, a atualização consistente de políticas é ainda mais complexa, pois as requisições de atualização devem ser ordenadas globalmente e as políticas, compostas sem conflitos. Esse capítulo propôs ainda 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 capítulo 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.

Capítulo 7

Conclusão

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 [74]. O paradigma de Redes Definidas por *Software* simplifica o gerenciamento, uma vez que separa o plano de controle, logicamente centralizado, do plano de dados distribuído. 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 [33]. As políticas são traduzidas para o plano de dados, no qual configura-se o encaminhamento e tratamento dos pacotes. Em SDN, as regras de encaminhamento são expressas por configurações de fluxo em tabela dos comutadores em execução no plano de dados. Contudo, mesmo com a crescente adoção de redes definidas por software (SDN) por redes empresariais, prover segurança às SDN ainda é um desafio fundamental.

Esse trabalho apresentou os principais vetores de ameaça e discutiu a segurança e a distribuição de controle em SDN [14, 88]. Os desafios são presentes mesmo nas redes tradicionais, contudo em Redes Definidas por Software tornam-se mais evidentes, devido à centralização do controle e à dependência da comunicação entre controlador e comutadores para o correto funcionamento da rede. Para prover segurança às SDNs, esse trabalho propôs AuthFlow [14, 88], um mecanismo de autenticação e controle de acesso para Redes Definidas por Software que permite a definição de fluxos baseada na identidade de um usuário. AuthFlow é um mecanismo de autenticação e controle o acesso à infraestrutura de um rede definida por *software* OpenFlow, baseado no padrão IEEE 802.1X e no servidor de autenticação RADIUS. O mecanismo AuthFlow proposto implementa a autenticação através de uma base dados LDAP com RADIUS. A proposta, no entanto, é extensível a outros métodos de autenticação, como o EAP-TLS, que autentica os nós com base em certificados. Os resultados mostram que o mecanismo de autenticação proposto impede que estações não autorizadas acessem recursos da rede, mesmo quando já autenticadas e, após, perdem seus privilégios. O controle de acesso proposto, base-

ado na identidade do usuário e nos fluxos gerados pelo usuário, permite monitorar e controlar, o conjunto de serviços que o usuário tem acesso e a qualidade de serviço. Os resultados mostram que o AuthFlow é efetivo no bloqueio de fluxos para estações não autenticadas, assim como provê uma visão da rede, para estações autenticadas, de acordo com a identidade de cada estação. Os resultados mostram ainda que o mecanismo proposto é mais eficiente que as outras propostas citadas, já que introduz menor sobrecarga de controle, e permite a definição de políticas de controle de acesso por fluxo de acordo com as credenciais de acesso de cada estação. A autenticação em Camada 2 garante a aplicabilidade do AuthFlow para dipositivos mais simples, como no cenário de Internet das Coisas (*Internet of Things* - IoT), e para funções de redes virtualizadas (*Virtualized Network Functions* - VNFs). A primitiva de identificação dos fluxos permite que o controlador da rede defina regras de acesso, em alto nível, por usuário e não mais por subredes ou endereços físicos. Assim, as regras de acesso são desvinculadas dos dispositivos do usuário e passam a serem vinculadas à identificação do usuário na rede.

Como o controle logicamente centralizado é o principal pilar do paradigma SDN, a realização do controlador de rede como um servidor centralizado implica desafios para a segurança, o desempenho e a escalabilidade da rede [74]. A centralização lógica do plano de controle cria uma abstração da visão global da rede que facilita o desenvolvimento e a implantação de novos serviços no núcleo da rede. Consequentemente, o controle e a manipulação das atualizações de política em SDN de forma distribuída passam a ser um desafio, pois fogem às implementações padrões de redes definidas por *software* e impõem novas restrições sobre a consistência da visão global de rede. 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. Assim, o conceito de consistência em SDN foi apresentado sob duas vertentes principais: a consistência de no tratamento do fluxo durante a sua existência e a consistência das políticas de encaminhamento ou tratamento de fluxos entre controladores. Este trabalho formalizou o problema de consistência em SDN com base na literatura existente e propôs um controlador distribuído com consistência forte de políticas entre controladores. A ideia central foi usar o teorema do CAP (*Consistency, Availability and Partition Tolerance* - Consistência, Disponibilidade e Tolerância a Partições) para balizar a proposta do controlador distribuído e garantir consistência local e disponibilidade, mesmo em cenários em que haja partições da rede.

Esse trabalho propôs um modelo generalizado em teoria de filas para derivar uma expressão analítica do desempenho de um controlador SDN. Com base nesse modelo, estima-se o número de comutadores que um controlador é capaz de atender,

dada uma restrição de atraso máximo. O trabalho propôs ainda uma arquitetura distribuída de controladores para SDN que mantém a visão global da rede e cria zonas de controle a fim de manter uma visão global consistente [74, 89, 90]. Para tanto, o trabalho introduz a ideia de Controlador Designado, que é um controlador de zona que assume o papel de manter a consistência da visão global da rede, um repositório de objetos globais que são acessíveis a todos controladores. O modelo de controlador proposto segue o teorema CAP (Consistência, Disponibilidade e Tolerância a Partições) ao definir um controlador sempre disponível que atualiza uma visão global compartilhada por todos os controladores de maneira consistente e, para garantir a tolerância a partições, o controle de zona, quando a rede é particionada, é autônomo para manter o funcionamento da partição. Foram propostas duas heurísticas de otimização da localização dos controladores: uma baseada na maximização da resiliência da rede e, outra, mais simples, baseada na minimização do número de saltos entre controlador e comutadores. Os resultados da avaliação das heurísticas mostram que a heurística mais complexa aumenta a resiliência da rede, em especial quando a taxa de falhas é de até 0.5 e, para topologias menos redundantes, a heurística de menor número de saltos apresenta resiliência comparável às demais. O protótipo do controlador proposto foi avaliado e os resultados mostraram que a visão global é mantida no cenário de controle fisicamente distribuído e todos os controladores a acessam e convergem no cálculo de uma árvore de cobertura comum.

Por fim, vale ainda ressaltar que esse trabalho apresentou a modelagem de Redes Definidas por Software proposta na literatura. Com base nessa modelagem, verifica-se que é possível exercer a atualização consistente de políticas em Redes Definidas por Software, mesmo com controle distribuído. Para esse fim, o procedimento da literatura é a marcação de pacotes ao entrarem na rede, associando a marcação à versão da configuração em uso na rede. Internamente, os pacotes são sempre encaminhados com a configuração vigente no momento de ingresso dos pacotes na rede. Esse trabalho, por sua vez, propôs um novo esquema de atualização de políticas em Redes Definidas por *Software*, que prova-se ser consistente e não depende de marcação de pacotes, mas somente da ordem como as atualizações são realizadas na rede [76, 81]. O esquema proposto, chamado Atualização Reversa, 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 uma eficiência de 94%, quando comparada à Atualização Ideal.

Esse trabalho propôs ainda 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 algoritmo para a composição de políticas é simples e se aproveita da interface de consenso provida pelo protocolo de consistência para garantir que políticas conflitantes não sejam instaladas. Aplicações que executam sobre um controlador distribuído fazem requisições de instalação de políticas na rede sem o conhecimento prévio de quais políticas já foram instaladas. A interface de consenso permite que os controladores distribuídos acordem se as requisições de política estão em conflito ou não e, a partir de então, informem as aplicações sobre a instalação das políticas. O algoritmo de composição de políticas 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 número de mensagens de outros protocolos de consenso, 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.

Dessa forma, as propostas discutidas nessa tese apresentam contribuição na área de Segurança em Redes Definidas por *Software*. Como trabalhos futuros, visa-se a implementação e consequentes experimentação dos esquemas de atualização consistente de políticas em ambientes reais.

Outra análise futura possível é o uso do AuthFlow para o provimento de qualidade de serviço na rede de acordo com a identidade do usuário. Assim, os fluxos da rede seriam direcionados a filas de encaminhamento com garantias de qualidade de serviço de acordo com a identidade do usuário que usa a rede. Com os limites de banda associados a cada fila de encaminhamento, cada identidade definirá o nível de qualidade de serviço que a estação deverá acessar.

A distribuição de controle em uma Rede Definida por Software deve ser agnóstica quanto ao sistema operacional de rede usado, assim como deve ser capaz de

distribuir o controle entre diferentes sistemas operacionais de rede. Um dos trabalhos futuros para prosseguir com a pesquisa iniciada nessa tese é a implementação de um protocolo para o plano de controle distribuído, de forma que seja agnóstico ao sistema operacional de rede e capaz de representar tipos abstratos de dados em uma visão global compartilhada por todos os controladores.

Referências Bibliográficas

[1] GUIMARÃES, P. H. V., PIEDRAHITA, A. M., ANDREONI LOPEZ, M. E., et al. "Comunicação em Redes Elétricas Inteligentes: eficiência, confiabilidade, segurança e escalabilidade". In: *SBRC 2013 - Minicursos*, maio 2013.

[2] NAYAK, A. K., REIMERS, A., FEAMSTER, N., et al. "Resonance: Dynamic Access Control for Enterprise Networks". In: *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, WREN '09, pp. 11–18, New York, NY, USA, 2009. ACM. ISBN: 978-1-60558-443-0.

[3] KREUTZ, D., RAMOS, F. M., VERISSIMO, P. "Towards Secure and Dependable Software-defined Networks". In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN'13, pp. 55–60, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2178-5.

[4] MATTOS, D. M. F., FERRAZ, L. H. G., DUARTE, O. C. M. B. "Virtual Machine Migration". In: da Fonseca, N. L. S., Boutaba, R. (Eds.), *Cloud Services, Networking and Management*, Wiley-IEEE Press, Hoboken, EUA, abr. 2015.

[5] FERRAZ, L. H. G., MATTOS, D. M. F., DUARTE, O. C. M. B. "A two-phase multipathing scheme based on genetic algorithm for data center networking". In: *Global Communications Conference (GLOBECOM), 2014 IEEE*, pp. 2270–2275, dez. 2014.

[6] ANDREONI LOPEZ, M., MATTOS, D. M. F., DUARTE, O. C. M. B. "Evaluating Allocation Heuristics for an Efficient Virtual Network Function Chaining". In: *2016 7th International Conference on the Network of the Future (NOF)*, pp. 1–4, nov. 2016.

[7] LEVIN, D., WUNDSAM, A., HELLER, B., et al. "Logically centralized?: state distribution trade-offs in software defined networks". In: *Proceedings of the*

First workshop on Hot topics in software defined networks, HotSDN'12, Helsinki, Finland, 2012. ACM.

[8] NOBRE, J., ROSARIO, D., BOTH, C., et al. "Toward software-defined battlefield networking", *IEEE Communications Magazine*, v. 54, n. 10, pp. 152–157, out. 2016. ISSN: 0163-6804. doi: 10.1109/MCOM.2016.7588285.

[9] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., et al. "OpenFlow: Enabling Innovation in Campus Networks", *SIGCOMM Comput. Commun. Rev.*, v. 38, n. 2, pp. 69–74, mar. 2008. ISSN: 0146-4833.

[10] KOBAYASHI, M., SEETHARAMAN, S., PARULKAR, G., et al. "Maturing of OpenFlow and Software-defined Networking Through Deployments", *Comput. Netw.*, v. 61, pp. 151–175, mar. 2014. ISSN: 1389-1286.

[11] MORAES, I. M., MATTOS, D. M., FERRAZ, L. H. G., et al. "FITS: A flexible virtual network testbed architecture", *Computer Networks*, v. 63, n. 0, pp. 221 – 237, 2014. ISSN: 1389-1286. Special issue on Future Internet Testbeds Part {II}.

[12] WEI, L., FUNG, C. "FlowRanger: A Request Prioritizing Algorithm for Controller DoS Attacks in Software Defined Networks". In: *IEEE ICC 2015 - Next Generation Networking Symposium (ICC'15 - NGN)*, London, United Kingdom, jun. 2015.

[13] ANDREONI LOPEZ, M., DUARTE, O. C. M. B. "Providing Elasticity to Intrusion Detection Systems in Virtualized Software Defined Networks". In: *IEEE ICC 2015 - Communication and Information Systems Security Symposium (ICC'15 - CISS)*, London, United Kingdom, jun. 2015.

[14] MATTOS, D. M. F., DUARTE, O. C. M. B. "AuthFlow: Um Mecanismo de Autenticação e Controle de Acesso para Redes Definidas por Software". In: *XXXII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2014*, maio 2014.

[15] MULLER, L. F., OLIVEIRA, R. R., LUIZELLI, M. C., et al. "Survivor: an Enhanced Controller Placement Strategy for Improving SDN Survivability". In: *Global Communications Conference (GLOBECOM), 2014 IEEE*, Austin, Texas, USA, dez. 2014.

[16] BERDE, P., GEROLA, M., HART, J., et al. "ONOS: Towards an Open, Distributed SDN OS". In: *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN'14, pp. 1–6, New York, NY, USA, 2014. ACM. ISBN: 978-1-4503-2989-7.

[17] KOPONEN, T., CASADO, M., GUDE, N., et al. “Onix: A Distributed Control Platform for Large-scale Production Networks”. In: *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI’10, pp. 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[18] KREUTZ, D., RAMOS, F. M., VERISSIMO, P. “Towards Secure and Dependable Software-defined Networks”. In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN’13, pp. 55–60, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2178-5.

[19] SCHMID, S., SUOMELA, J. “Exploiting Locality in Distributed SDN Control”. In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN’13, pp. 121–126, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2178-5.

[20] HELLER, B., SHERWOOD, R., MCKEOWN, N. “The Controller Placement Problem”. In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN’12, pp. 7–12, New York, NY, USA, 2012. ACM. ISBN: 978-1-4503-1477-0.

[21] BARI, M., ROY, A., CHOWDHURY, S., et al. “Dynamic Controller Provisioning in Software Defined Networks”. In: *9th International Conference on Network and Service Management (CNSM), 2013*, pp. 18–25, out. 2013.

[22] REITBLATT, M., FOSTER, N., REXFORD, J., et al. “Abstractions for Network Update”. In: *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM’12, pp. 323–334, New York, NY, USA, 2012. ACM. ISBN: 978-1-4503-1419-0.

[23] CANINI, M., VENZANO, D., PEREŠÍNI, P., et al. “A NICE Way to Test Openflow Applications”. In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI’12, pp. 10–10, Berkeley, CA, USA, 2012. USENIX Association. Disponível em: <`http://dl.acm.org/citation.cfm?id=2228298.2228312`>.

[24] GILBERT, S., LYNCH, N. “Brewer’s Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services”, *SIGACT News*, v. 33, n. 2, pp. 51–59, jun. 2002. ISSN: 0163-5700.

[25] PORRAS, P., SHIN, S., YEGNESWARAN, V., et al. “A Security Enforcement Kernel for OpenFlow Networks”. In: *Proceedings of the First Workshop*

on Hot Topics in Software Defined Networks, HotSDN'12, pp. 121–126, New York, NY, USA, 2012. ACM. ISBN: 978-1-4503-1477-0.

[26] SHIN, S., PORRAS, P., YEGNESWARAN, V., et al. "FRESCO: Modular composable security services for software-defined networks". In: *Proceedings of Network and Distributed Security Symposium*, 2013.

[27] CASADO, M., FREEDMAN, M., PETTIT, J., et al. "Ethane: Taking control of the enterprise", *ACM SIGCOMM Computer Communication Review*, v. 37, n. 4, pp. 1–12, 2007.

[28] GUIMARÃES, P. H., FERRAZ, L., TORRES, J. V., et al. "Experimenting Content-Centric Networks in the Future Internet Testbed Environment". In: *IEEE International Conference on Communications 2013: IEEE ICC'13 - Workshop on Cloud Convergence: challenges for future infrastructures and services (WCC 2013) (ICC'13 - IEEE ICC'13 - Workshop WCC)*, pp. 1398–1402, Budapest, Hungary, jun. 2013.

[29] PETRY, T., DA FONTE LOPES DA SILVA, R., BARCELLOS, M. "Off the Wire Control: Improving the Control Plane Resilience through Cellular Networks". In: *IEEE ICC 2015 - Next Generation Networking Symposium (ICC'15 - NGN)*, London, United Kingdom, jun. 2015.

[30] TOOTOONCHIAN, A., GANJALI, Y. "HyperFlow: A Distributed Control Plane for OpenFlow". In: *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, pp. 3–3, Berkeley, CA, USA, 2010. USENIX Association.

[31] HASSAS YEGANEH, S., GANJALI, Y. "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications". In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN'12, pp. 19–24, New York, NY, USA, 2012. ACM. ISBN: 978-1-4503-1477-0.

[32] PEREŠÍNI, P., KUZNIAR, M., VASIĆ, N., et al. "OF.CPP: Consistent Packet Processing for Openflow". In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN'13, pp. 97–102, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2178-5.

[33] CANINI, M., KUZNETSOV, P., LEVIN, D., et al. *The case for reliable software transactional networking.* Relatório Técnico CKLS-CRSTN-13, Internet Network Architectures - Department of Telecommunication Systems - Technische Universität Berlin, 2013. Disponível em: <`http://www.net.t-labs.tu-berlin.de/papers/CKLS-CRSTN-13.pdf`>.

[34] LUO, S., YU, H., LI, L. "Consistency is Not Easy: How to Use Two-Phase Update for Wildcard Rules?" *Communications Letters, IEEE*, v. 19, n. 3, pp. 347–350, mar. 2015.

[35] FOGEL, A., FUNG, S., PEDROSA, L., et al. "A general approach to network configuration analysis". In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI'15)*, Berkeley, CA, USA, 2015. USENIX Association.

[36] CASADO, M., KOPONEN, T., SHENKER, S., et al. "Fabric: a retrospective on evolving SDN". In: *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN'12, pp. 85–90, New York, NY, USA, 2012. ACM. ISBN: 978-1-4503-1477-0.

[37] GUEDES, D., VIEIRA, L., VIEIRA, M., et al. "Redes Definidas por Software: uma abordagem sistêmica para o desenvolvimento de pesquisas em Redes de Computadores", *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC 2012*, pp. 160–210, maio 2012.

[38] KREUTZ, D., RAMOS, F. M. V., VERÍSSIMO, P., et al. "Software-Defined Networking: A Comprehensive Survey", *Proceedings of the IEEE*, v. 103, n. 1, pp. 63, 2015.

[39] NAGAHAMA, F. Y., GRANVILLE, L., FARIAS, F., et al. "IPSFlow – Uma Proposta de Sistema de Prevenção de Intrusão Baseado no Framework OpenFlow". In: *II Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF) do Simpósio Brasileiro de Redes de Computadores-SBRC 2012*, Ouro Preto, MG - Brasil, maio 2012.

[40] MATTOS, D. M. F., DUARTE, O. C. M. B. "XenFlow: Seamless migration primitive and quality of service for virtual networks". In: *Global Communications Conference (GLOBECOM), 2014 IEEE*, pp. 2326–2331, Dec 2014.

[41] MATTOS, D. M. F., FERNANDES, N. C., DUARTE, O. C. M. B. "XenFlow: Um Sistema de Processamento de Fluxos Robusto e Eficiente para Migração em Redes Virtuais". In: *XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2011*, maio 2011.

[42] MATTOS, D., FERRAZ, L., DUARTE, O. C. M. B. "Um Mecanismo para Isolamento Seguro de Redes Virtuais Usando a Abordagem Híbrida Xen e OpenFlow". In: *SBSeg 2013 - XIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, Manaus - Brazil, nov. 2013.

[43] ZORN, G. “Microsoft PPP CHAP Extensions, Version 2”. RFC 2759 (Informational), jan. 2000.

[44] PFAFF, B., PETTIT, J., AMIDON, K., et al. “Extending Networking into the Virtualization Layer”. In: *Eight ACM Workshop on Hot Topics in Networks (HotNets-VIII),HOTNETS'09*, New York, USA, out. 2009.

[45] HANDLEY, M., HODSON, O., KOHLER, E. “XORP: An open platform for network research”, *ACM SIGCOMM Computer Communication Review*, v. 33, n. 1, pp. 53–57, 2003.

[46] MATTOS, D. M. F., DUARTE, O. C. M. B. “QFlow: Um Sistema com Garantia de Isolamento e Oferta de Qualidade de Serviço para Redes Virtualizadas”. In: *XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2012*, abr. 2012.

[47] MATIAS, J., JACOB, E., TOLEDO, N., et al. “Towards Neutrality in Access Networks: A NANDO Deployment with OpenFlow”. In: *ACCESS 2011, The Second International Conference on Access Networks*, pp. 7–12, Luxembourg City, Luxembourg, jun. 2011.

[48] GUENANE, F., SAMET, N., PUJOLLE, G., et al. “A strong authentication for virtual networks using EAP-TLS smart cards”. In: *Global Information Infrastructure and Networking Symposium (GIIS), 2012*, pp. 1–6, 2012.

[49] HONG, S., XU, L., WANG, H., et al. “Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures”. In: *Proceedings of the 2015 Network and Distributed System Security Symposium (NDSS)*, San Diego, California, 2015. Internet Society.

[50] PORRAS, P., CHEUNG, S., FONG, M., et al. “Securing the Software-Defined Network Control Layer”. In: *Proceedings of the 2015 Network and Distributed System Security Symposium (NDSS)*, San Diego, California, 2015. Internet Society.

[51] SHERWOOD, R., GIBB, G., YAP, K., et al. *FlowVisor: A network virtualization layer*. Relatório técnico, Tech. Rep. OPENFLOW-TR-2009-01, OpenFlow Consortium, 2009.

[52] LOPEZ, M. A., MATTOS, D. M. F., FERRAZ, L. H. G., et al. “Localização Eficiente de Sensores Colaborativos para Detecção e Prevenção de Intrusão em Ambientes Virtualizados”. In: *Workshop de Gerência e Operação de Redes e Serviços (WGRS 2015) do SBRC'2015*, maio 2015.

[53] LOPEZ, M. A., MATTOS, D. M. F., DUARTE, O. C. M. B. "An elastic intrusion detection system for software networks", *Annals of Telecommunications*, v. 71, n. 11, pp. 595–605, 2016.

[54] HAND, R., TON, M., KELLER, E. "Active Security". In: *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, HotNets-XII, pp. 17:1–17:7, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2596-7.

[55] SHIN, S., GU, G. "CloudWatcher: Network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)". In: *20th IEEE International Conference on Network Protocols (ICNP), 2012*, pp. 1–6, out. 2012.

[56] CHILWAN, A., MAHMOOD, K., ØSTERBØ, O. N., et al. "On Modeling Controller-Switch Interaction in OpenFlow based SDNs", *International Journal of Computer Networks & Communications*, v. 6, n. 6, pp. 135, 2014.

[57] JARSCHEL, M., OECHSNER, S., SCHLOSSER, D., et al. "Modeling and Performance Evaluation of an OpenFlow Architecture". In: *Proceedings of the 23rd International Teletraffic Congress*, ITC'11, pp. 1–7, San Francisco, California, 2011. International Teletraffic Congress. ISBN: 978-0-9836283-0-9.

[58] AZODOLMOLKY, S., NEJABATI, R., PAZOUKI, M., et al. "An analytical model for software defined networking: A network calculus-based approach". In: *2013 IEEE Global Communications Conference (GLOBECOM)*, pp. 1397–1402, dez. 2013.

[59] ROTSOS, C., SARRAR, N., UHLIG, S., et al. "OFLOPS: An Open Framework for OpenFlow Switch Evaluation". In: Taft, N., Ricciato, F. (Eds.), *Passive and Active Measurement*, v. 7192, *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 85–95, 2012. ISBN: 978-3-642-28536-3.

[60] ROS, F. J., RUIZ, P. M. "On reliable controller placements in Software-Defined Networks", *Computer Communications*, v. 77, pp. 41 – 51, 2016. ISSN: 0140-3664.

[61] CARDOSO, L. P., FERRAZ, L. H. G., DUARTE, O. C. M. B. "Migração de máquinas virtuais para economia de energia". In: *Workshop de Gerência e Operação de Redes e Serviços (WGRS 2014) do SBRC'2014*, maio 2014.

[62] ZHANG, Y., BEHESHTI, N., TATIPAMULA, M. "On Resilience of Split-Architecture Networks". In: *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pp. 1–6, dez. 2011.

[63] HUNT, P., KONAR, M., JUNQUEIRA, F. P., et al. "ZooKeeper: Wait-free Coordination for Internet-scale Systems". In: *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pp. 11–11, Berkeley, CA, USA, 2010. USENIX Association.

[64] YEGANEH, S., TOOTOONCHIAN, A., GANJALI, Y. "On scalability of software-defined networking", *Communications Magazine, IEEE*, v. 51, n. 2, pp. 136–141, fev. 2013. ISSN: 0163-6804.

[65] DIXIT, A., HAO, F., MUKHERJEE, S., et al. "Towards an Elastic Distributed SDN Controller". In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN'13, pp. 7–12, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2178-5.

[66] KUŹNIAR, M., PEREŠÍNI, P., VASIĆ, N., et al. "Automatic Failure Recovery for Software-defined Networks". In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN'13, pp. 159–160, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2178-5.

[67] KEMPF, J., BELLAGAMBA, E., KERN, A., et al. "Scalable fault management for OpenFlow". In: *Communications (ICC), 2012 IEEE International Conference on*, pp. 6606–6610, jun. 2012.

[68] PRIES, R., JARSCHEL, M., GOLL, S. "On the usability of OpenFlow in data center environments". In: *2012 IEEE International Conference on Communications (ICC)*, pp. 5533–5537, jun. 2012.

[69] CANINI, M., KUZNETSOV, P., LEVIN, D., et al. *A Distributed SDN Control Plane for Consistent Policy Updates*. Relatório técnico, Internet Network Architectures - Department of Telecommunication Systems - Technische Universität Berlin, 2013. Disponível em: <`http://arxiv.org/abs/1305.7429`>.

[70] DUARTE, R., VIEIRA, A. B., CUNHA, I. F. S., et al. "Impact of provider failures on the traffic at a university campus". In: *2015 IFIP Networking Conference (IFIP Networking)*, pp. 1–9, maio 2015. doi: 10.1109/IFIPNetworking.2015.7145327.

[71] HAN, J. H., MUNDKUR, P., ROTSOS, C., et al. "Blueswitch: enabling provably consistent configuration of network switches". In: *Architectures for Networking and Communications Systems (ANCS), 2015 ACM/IEEE Symposium on*, pp. 17–27, maio 2015.

[72] PISA, P., FERNANDES, N., CARVALHO, H., et al. "OpenFlow and Xen-Based Virtual Network Migration". In: Pont, A., Pujolle, G., Raghavan, S. (Eds.), *Communications: Wireless in Developing Countries and Networks of the Future*, v. 327, *IFIP Advances in Information and Communication Technology*, Springer Boston, pp. 170–181, 2010.

[73] FERRAZ, L. H. G., LAUFER, R., MATTOS, D. M., et al. "A high-performance Two-Phase Multipath scheme for data-center networks", *Computer Networks*, v. 112, pp. 36 – 51, 2017. ISSN: 1389-1286. doi: http://dx.doi.org/10.1016/j.comnet.2016.09.025.

[74] MATTOS, D. M. F., DUARTE, O. C. M. B., PUJOLLE, G. "A Resilient Distributed Controller for Software Defined Networking". In: *IEEE ICC 2016 - Next Generation Networking and Internet Symposium (ICC'16 - NGN)*, Kuala Lumpur, Malaysia, maio 2016.

[75] CANINI, M., KUZNETSOV, P., LEVIN, D., et al. "A distributed and robust SDN control plane for transactional network updates". In: *The IEEE INFOCOM 2015*, abr. 2015.

[76] MATTOS, D. M. F., DUARTE, O. C. M. B. "Atualização Reversa: Garantindo Consistência de Estados em Redes Definidas por Software". In: *SBSeg 2015 - XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, Florianópolis, SC - Brasil, nov. 2015.

[77] FERGUSON, A. D., GUHA, A., LIANG, C., et al. "Hierarchical Policies for Software Defined Networks". In: *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN'12, pp. 37–42, New York, NY, USA, 2012. ACM. ISBN: 978-1-4503-1477-0.

[78] PRISCO, R. D., LAMPSON, B., LYNCH, N. "Revisiting the Paxos algorithm", *Theoretical Computer Science*, v. 243, n. 1-2, pp. 35 – 91, 2000. ISSN: 0304-3975.

[79] JUNQUEIRA, F. P., REED, B. C., SERAFINI, M. "Zab: High-performance broadcast for primary-backup systems". In: *IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN), 2011*, pp. 245–256, jun. 2011.

[80] MONSANTO, C., REICH, J., FOSTER, N., et al. “Composing Software Defined Networks.” In: *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI’13)*, pp. 1–13, Berkeley, CA, USA, 2013. USENIX Association.

[81] MATTOS, D. M. F., DUARTE, O. C. M. B., PUJOLLE, G. “Reverse Update: A Consistent Policy Update Scheme for Software-Defined Networking”, *IEEE Communications Letters*, v. 20, n. 5, pp. 886–889, maio 2016. ISSN: 1089-7798.

[82] PANDA, A., SCOTT, C., GHODSI, A., et al. “CAP for Networks”. In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN’13, pp. 91–96, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2178-5.

[83] CANINI, M., KUZNETSOV, P., LEVIN, D., et al. “Software Transactional Networking: Concurrent and Consistent Policy Composition”. In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN’13, pp. 1–6, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2178-5.

[84] MCGEER, R. “A Safe, Efficient Update Protocol for Openflow Networks”. In: *ACM SIGCOMM - HotSDN’12*, Helsinki, Finland, 2012. ACM.

[85] KATTA, N. P., REXFORD, J., WALKER, D. “Incremental Consistent Updates”. In: *ACM SIGCOMM - HotSDN’13*, Hong Kong, China, 2013. ACM.

[86] MCCLURG, J., HOJJAT, H., CERNY, P., et al. “Efficient Synthesis of Network Updates”. In: *ACM SIGPLAN - PLDI*, Portland, USA, jun. 2015. ACM.

[87] GRAY, J., LAMPORT, L. “Consensus on Transaction Commit”, *ACM Trans. Database Syst.*, v. 31, n. 1, pp. 133–160, mar. 2006. ISSN: 0362-5915.

[88] MATTOS, D. M. F., DUARTE, O. C. M. B. “AuthFlow: authentication and access control mechanism for software defined networking”, *Annals of Telecommunications*, v. 71, n. 11, pp. 607–615, 2016. ISSN: 1958-9395. doi: 10.1007/s12243-016-0505-z. Disponível em: <`http://dx.doi.org/10.1007/s12243-016-0505-z`>.

[89] MATTOS, D. M. F., ANDREONI LOPEZ, M., FERRAZ, L. H. G., et al. “Controlador Resiliente com Distribuição Eficiente para Redes Definidas por Software”. In: *XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC’2015*, maio 2015.

[90] MATTOS, D. M. F., DUARTE, O. C. M. B., PUJOLLE, G. “Profiling Software Defined Networks for Dynamic Distributed-Controller Provisioning”. In: *2016 7th International Conference on the Network of the Future (NOF)*, pp. 1–4, nov. 2016.