Abordagem de Composição de Programas P4 em Redes Programáveis

Ricardo Parizotto, Lucas Castanheira, Alberto Schaeffer-Filho

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{rparizotto,lbcastanheira,alberto}@inf.ufrgs.br

Abstract. *Software Defined Networks (SDN) and emerging programmable data planes enable more flexibility for the operation of networks. Such technologies are capable of allowing network operators to reconfigure networks on both control and data planes dynamically. Such an ability to reconfigure and program the network on demand offers several benefits, in particular making it possible to improve network security mechanisms by using programmability. However, in addition to promoting a higher degree of flexibility, data plane programmability raises concerns with respect to bugs that can create inconsistencies in the network's most basic function, the forwarding of data, disrupting previously defined policies. In this work we present a framework to install functions on programmable data planes in a reliable manner, ensuring that the installation of such functions preserves basic forwarding properties. For this, we employ program composition techniques to merge knowingly correct modular functions into a single, aggregated data plane program, ensuring that the resulting program is correct after the merge. To show the correctness of our method, we present a case study with a firewall and a processing/monitoring module.*

Resumo. *Redes Definidas por Software (SDN) e o surgimento de planos de dados programáveis permitem maior flexibilidade para a operação de redes. Essas tecnologias permitem que os administradores de rede reconfigurem os planos de dados e de controle. A capacidade de reconfigurar e programar a rede sob demanda oferece vários benefícios, em particular possibilitando melhorar os mecanismos de segurança de rede usando a capacidade de programação. No entanto, além de promover um grau maior de flexibilidade, a programação do plano de dados levanta preocupações em relação a erros que podem criar inconsistências na função mais básica da rede, o encaminhamento de dados, interrompendo políticas previamente definidas. Neste trabalho apresentamos um framework para instalar funções em planos de dados programáveis de maneira confiável, garantindo que a instalação de tais funções preserve as propriedades básicas de encaminhamento. Para isso, empregamos técnicas de composição de programas para mesclar funções modulares em um único plano de dados agregado, garantindo que o programa resultante seja correto após a mesclagem. Para mostrar a corretude de nosso método, apresentamos um estudo de caso com um firewall e um módulo de monitoramento.*

1. Introdução

Paradigmas de redes definidas por software (SDN) possibilitam o desacoplamento do hardware (e.g. roteadores) e dos programas que nele executam (e.g. algoritmos de ro-

teamento) [Feamster et al. 2014]. Essa estratégia facilita a operação das redes, uma vez que promove programabilidade no plano de controle da rede. Recentemente, a programabilidade foi estendida também para o plano de dados. Com o intuito de possibilitar a programabilidade no hardware que reside no plano de dados, foi criada a linguagem P4 [Bosshart et al. 2014], que permite aos administradores da rede definirem o comportamento dos dispositivos de encaminhamento. Uma vantagem decorrente disso é a facilidade na criação e na implantação de novos protocolos de rede totalmente customizáveis, sem depender da indústria para que uma nova funcionalidade seja adicionada ao comportamento do plano de dados.

Tal habilidade para reconfigurar e programar a rede possui várias aplicações, que geralmente abrangem mecanismos governados pela dinâmica e pelas mudanças frequentes de políticas de rede. Isso pode envolver, por exemplo, a implantação de funções de rede adicionais e a reescrita da funcionalidade dos *switches* P4 de maneira que possam suportar mais do que apenas um serviço. Para que isso ocorra, é necessário o desenvolvimento de técnicas abrangentes que permitam que a reconfiguração da rede ocorra de maneira rápida e sem corromper propriedades básicas de operação. Alguns trabalhos recentes propõem a utilização de P4 para configurar funções virtualizadas no próprio plano de dados [Hancock and van der Merwe 2016, Zhang et al. 2017]. Tais propostas, porém, pecam de duas maneiras: em escalabilidade, devido ao uso excessivo de tabelas de controle e primitivas de recirculação de pacotes, atrasando o processamento e encaminhamento dos pacotes; e não fornecendo o isolamento necessário para as funções [Dimitropoulos et al. 2018]. Diante disso, entendemos que são necessárias abstrações e estratégias que permitam que os administradores de rede possam implantar novas funcionalidades nos seus dispositivos programáveis, sem que isso impacte negativamente no desempenho das funções de rede.

Neste trabalho, propomos uma estratégia baseada em P4 que permite que um administrador de rede possa estender o comportamento de *switches* programáveis. A nossa estratégia é composta de duas etapas: (1) a composição de programas, que deverá possibilitar que o administrador de rede componha funções modulares a um programa base, de modo que a composição resulte em um novo programa com as funcionalidades tanto do programa base como da extensão; (2) o isolamento lógico dos programas, que evita que as regras de *match+action* sejam sobrepostas e permite que a ordem em que os programas são executados seja alterada dinamicamente. Agregado à estratégia de composição, isso garante que as funções atuem de maneira isolada no processamento dos pacotes. A estratégia proposta depende de um programa base com construtores bem definidos, que permitem que as funções compostas possam ser gerenciadas de maneira dinâmica. O operador de rede poderá, então, compor o programa base com a configuração desejada e, enquanto está em funcionamento, decidir qual a ordem em que as funções serão processadas por um tipo de tráfego específico.

A estrutura desse artigo esta dividida da seguinte maneira: na Seção 2 apresentamos um background sobre programabilidade no plano de dados e as restrições no modelo de encaminhamento. Na Seção 3, apresentamos a estratégia de composição de programas, seguida pela estratégia de agregação de fluxos de controle. Na Seção 4 apresentamos um estudo de caso e avaliação de nossa estratégia. Por fim, apresentamos uma visão geral dos trabalhos relacionados e as conclusões.

2. Background e Motivação

Nesta seção, revisamos P4 e programabilidade no plano de dados, seguido por uma descrição das características principais que devem ser consideradas quando novas funcionalidades são implantadas ao plano de dados. Apresentamos também as principais restrições operacionais do modelo de encaminhamento, que dizem respeito às entradas das tabelas de *match+action* e de *parsers* de cabeçalhos de pacotes.

2.1. Abstração da linguagem P4

P4 é uma linguagem de especificação de plano de dados que permite a configuração e programação de dispositivos de encaminhamento [Garcia et al. 2018, Bosshart et al. 2014]. A sua abstração, apresentada na Figura 1, divide o comportamento do plano de dados em um *parser* de cabeçalhos de pacotes, um conjunto de tabelas de *match+action* e fluxos de controle. O *parser* é uma máquina de estados que descreve como ler os cabeçalhos de um pacote para as variáveis internas. Depois que um pacote chega a um estado final da máquina de estados do parser, o pacote é processado pelos construtores definidos no fluxo de controle. No fluxo de controle são definidas exclusivamente as estruturas das tabelas, ações e a ordem em que elas são executadas durante o processamento dos pacotes.

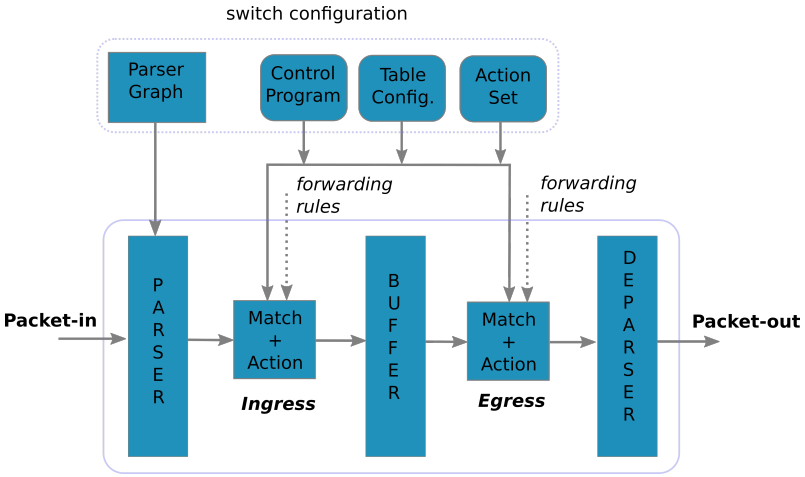

Figura 1. A abstração da linguagem P4, adaptado de Bosshart et al. [Bosshart et al. 2014]

A abstração P4 divide o modelo de encaminhamento em dois estágios sequenciais, (1) configuração do *hardware*, feita de maneira estática no programa P4 e (2) população das regras, feita de maneira dinâmica pelo controlador. Na configuração, temos que escrever o programa P4 que vai rodar no *switch* (incluindo *parser*, estágios do *match+action* e *deparser*). Tal configuração é toda feita dentro de um fonte P4, que é compilado para uma arquitetura especifica e carregado no *switch*. O estágio de população das regras acontece logo após a configuração e se manterá durante todo o *runtime* do *switch*. No estágio de população, o controlador pode alterar as regras do *switch* livremente. Essa fase é realizada pelo controlador e pelos programas que nele executam, cada um atualizando as regras que lhes é pertinente.

2.2. Restrições no modelo do plano de dados

Geralmente, programas são criados para executar no controlador, atualizando regras durante a fase de população. Para que evitemos *bugs* nesses programas, as atualizações de regras por ele geradas têm que respeitar algumas propriedades básicas, tais como: alcançabilidade e boa formação de pacotes. Na sequência, falaremos sobre cada uma delas.

Propriedades fim-a-fim Sempre que uma nova entrada é inserida em uma tabela do modelo do plano de dados, ela deve preservar propriedades (ou políticas) definidas previamente, tais como

"pacotes do switch A devem chegar ao switch B".

Ou propriedades de *safety*, como

"o fluxo i deve ser processado pelo switch X antes que ele alcance seu destino".

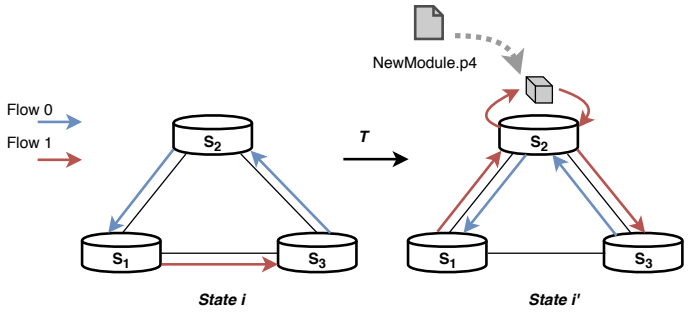

Figura 2. Transição do estado de rede

Mudanças de políticas e de roteamento no plano de dados podem ser modeladas como transições entre estados das tabelas de *match+action*. Na Figura 2, apresentamos um cenário que descreve uma transição entre dois estados de rede. No estado i, o Fluxo 1 é roteado através do caminho $(S1, S3)$. Uma transição T de um estado i para o estado i' é realizada implantando um módulo adicional ao *switch* $S2$ e mudando seu comportamento de encaminhamento para o novo módulo interceptar pacotes do Fluxo 1. Então, depois de atualizar $S1$, o mesmo fluxo é roteado por $(S1, S2, S3)$, respectivamente.

Atualizar o plano de dados não é uma operação atômica, porque *switches* não são dispositivos sincronizados. Por isso, a ordem em que cada *switch* aplica mudanças é um fator importante para alcançar uma transição do estado da rede sem inconsistências. Por exemplo, no cenário da Figura 2, se, durante a transição T o *switch* S_1 atualizar seu comportamento de encaminhamento antes de S_2, os pacotes do fluxo 1 irão enfrentar um 'buraco negro' quando alcançarem S_2 (ou os pacotes serão enfileirados) [Reitblatt et al. 2012][Katta et al. 2013] [Jin et al. 2014] [Nguyen et al. 2017].

Propriedades do switch O nível de inconsistência torna-se ainda maior com programabilidade no plano de dados, que permite que esse tipo de *bug* possa ocorrer dentro do *pipeline* de tabelas do *switch*, devido a possibilidade de mudanças da configuração de

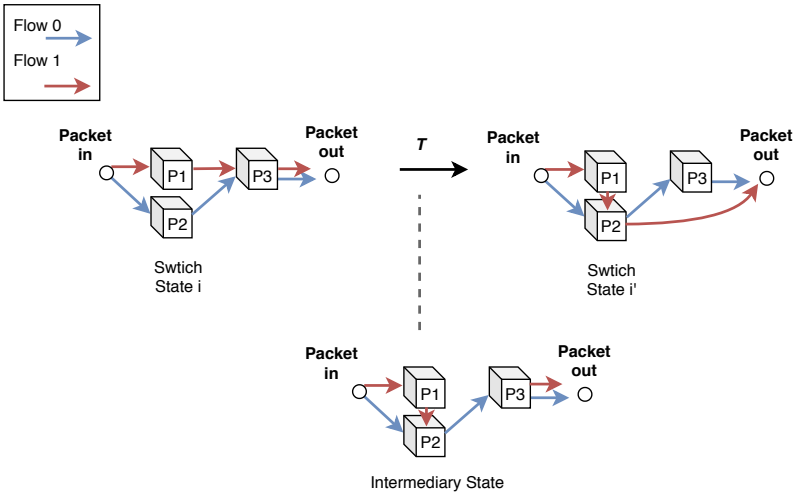

Figura 3. Transição do estado interno de um switch

tabelas e parsers de pacotes [Freire et al. 2018][Liu et al. 2018]. Ocasionalmente, se um novo módulo inserido não possuir as instruções corretas para decodificar os cabeçalhos dos pacotes, os pacotes vão ser processados de uma maneira indesejada. Um exemplo disso, é que pacotes chegariam ao pipeline sem nenhum valor instanciado ou seriam eliminados ainda no parser [Lopes et al. 2016].

A ordem em que as atualizações são gerenciadas dentro do pipeline do switch também estão sujeitas a falhas de configuração. A Figura 3 ilustra uma transição entre dois diferentes estados internos de um switch. No estado i, o Fluxo 1 é roteado pelos módulos (P_1, P_3). Uma transição do estado i ao i' é realizada mudando a sequência de módulos que processam o Fluxo 1. Depois da transição, o mesmo fluxo é roteado pelos módulos (P_1, P_2), respectivamente. Se o módulo P_1 atualizar seu comportamento de encaminhamento antes de P_2, irá formar um estado intermediário inconsistente, onde pacotes do Fluxo 1 irão enfrentar um 'buraco negro' quando chegarem ao módulo P_2.

Neste trabalho, propomos uma estratégia para compor programas P4 em redes programáveis. Para isso, apresentamos um *framework* capaz de unir características de diferentes programas em um único programa agregado, que contempla a funcionalidade de ambos e pode ser gerenciado dinamicamente.

3. Abordagem de Composição de Programas

Recentemente, diversas aplicações de redes voltaram à discussão, o que têm motivado a criação de vários mecanismos para o plano de dados. Porém, a maioria dos trabalhos trata os programas para plano de dados como monolíticos e com uma única funcionalidade específica. Neste trabalho, apresentamos uma estratégia para compor mais do que um programa em um switch P4. Nossa estratégia se utiliza de técnicas de composição de máquinas de estado para mostrar como realizar a extensão de parsers e deparsers de pacotes. Nessa seção, também apresentamos uma arquitetura base, que possui primitivas para composição de fluxos de controle de vários programas.

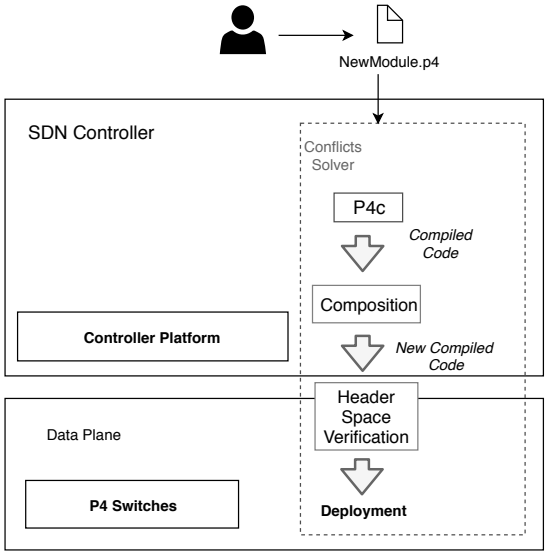

Figura 4. Arquitetura do mecanismo de composição

Visão geral da estratégia Dado um novo programa S, utilizamos uma técnica de composição de máquinas de estado para criar uma nova especificação com um conjunto de extensões do programa S. A composição é a união do conjunto de estados terminais, não-terminais, transições e as definições e instâncias de cabeçalhos definidos em cada programa. A estratégia proposta, ilustrada na Figura 4, compõe a configuração dos programas do plano de dados ainda no plano de controle, pelo administrador de rede. As funções devem ser inseridas conforme as necessidades do operador. Assim que a composição é finalizada, uma etapa de verificação checa o espaço de cabeçalhos para evitar loops e não determismo no código final gerado. Por fim, o resultado é compilado e pode ser implantado no switch. A implantação é realizada de maneira estática, isto é, exige a reinicialização do switch. Depois que a implantação é finalizada, o operador pode gerenciar dinamicamente a ordem em que as funções inseridas processam os pacotes. Isso se dá pela atualização das regras de *match+action* das tabelas que fazem parte da composição.

A composição de programas é realizada estendendo o código *host* (Figura 5, passo 1). *Parsers* e *Deparsers* são unidos utilizando composição de máquinas de estados (Figura 5, passo 2). O novo fluxo de controle é posicionado no começo do *pipeline* do programa *Host*, reescrevendo seu código fonte (Figura 5, passo 3). A seguir, essas etapas serão descritas em maiores detalhes.

3.1. Extensão de parsers de pacotes

Após a leitura do programa *Host*, o seu *parser* de pacotes é estendido para incluir as funcionalidades do *parser* de pacotes pertencente à extensão [Zheng et al. 2018]. O resultado da composição é uma nova máquina de estados que une os estados equivalentes (i.e. estruturas de cabeçalhos) e integra as transições que não estão no parser do programa a ser estendido. A Figura 5, Passo 2, apresenta a composição entre esses dois parsers. No exemplo da figura, o *parser* do programa *Host* é estendido para suportar a leitura do cabeçalho *sec* após decodificar o cabeçalho do IPv4. Depois que a composição de *parser* de pacotes é finalizada, o processo segue para a composição do fluxo de controle (passo 3). Essa estratégia permite que o administrador de rede altere dinamicamente a ordem

das funções processadas no plano de dados, simplesmente adicionando novas regras de *match+action* ao *switch*.

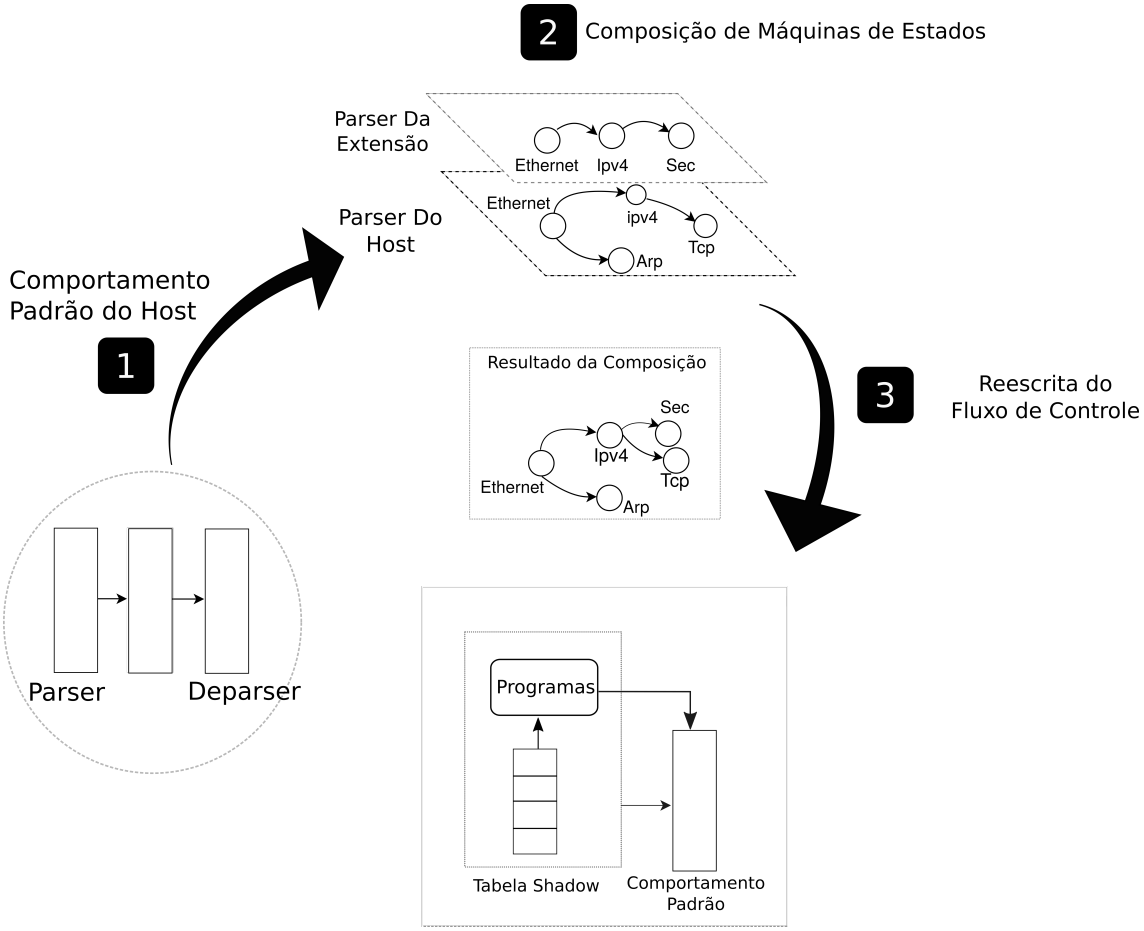

Figura 5. Visão geral da estratégia de composição de programas P4

3.2. Agregação de Fluxos de Controle

A composição de fluxos de controle permite incluir ações adicionais, definições de tabelas ou, até mesmo, ramificações adicionais ao programa *Host*. Uma maneira simples para compor fluxos de controle é unir a especificação de tabelas com o mesmo nome e tipos de atributos de *match*. Porém, isso cria a possibilidade de diferentes aplicações do plano de controle inserirem regras conflitantes, permitindo que, eventualmente, pacotes da mesma política sejam roteados por mais do que um caminho. Como uma consequência disso, propriedades básicas de processamento de pacotes podem ser violadas, como por exemplo, a propriedade de coerência de pacotes (ou de fluxos), apresentada em [Reitblatt et al. 2012]. A adição de um módulo que gera regras conflitantes requer que as aplicações do plano de controle sejam alteradas, o que, tecnicamente atrasaria o processo de desenvolvimento. Nessa seção, apresentamos o mecanismo que compõe os módulos do fluxo de controle, isolando os fluxos de controle específicos de cada aplicação.

Impacto de isolamento Com o objetivo de prevenir o problema de inserção de regras conflitantes, isolamos as tabelas e ações de cada novo módulo inserido. Para isso, é

necessário renomear tabelas e ações que tenham nomes ambíguos (para evitar loops no pipeline). Por isso, resolvemos conflitos de nomes e fixamos o novo módulo no início do pipeline do programa *host*. Essa estratégia isola as regras de *match+action* de cada aplicação, tornando o comportamento do programa *host* independente do processamento das funcionalidades inseridas. O isolamento garante que as regras instaladas não irão corromper as regras específicas das aplicações que gerenciam as outras tabelas do programa, preservando, assim, a ordem de execução das funções.

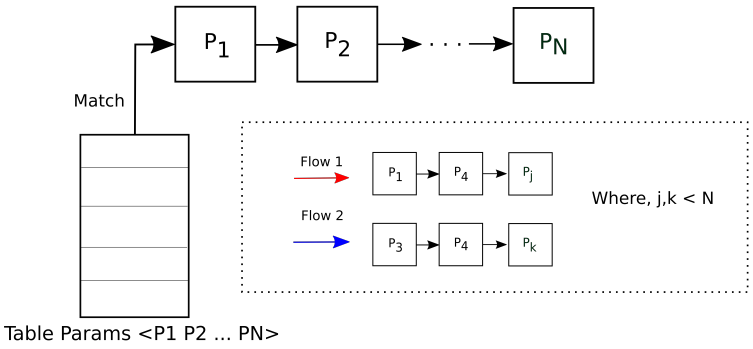

Figura 6. Encadeamento de programas no plano de dados

Encadeamento de Programas Para compor vários programas em nosso *framework*, criamos a abstração de encadeamento de funções. Essas possuem sua ordem de execução controlada a partir de regras adicionadas de uma aplicação do plano de controle. Para isso, utilizamos uma tabela, que chamamos de *'Shadow'*, a qual funciona como um grande catálogo de ponteiros para funções. Inicialmente, a tabela shadow é posicionada no inicio do pipeline de tabelas e intercepta todos os pacotes, mapeando um conjunto de fluxos para um conjunto sequencial de programas $P_1, P_2, ..., P_N$. Dessa maneira, a ordem de execução das funções pode ser diferente para cada parâmetro de *match* da tabela e alterada de maneira dinâmica pelo operador de rede. Quando o operador deseja alterar as funções executadas, ele apenas atualiza o conteúdo da tabela *Shadow*, alterando os parâmetros que dizem respeito à ordem de execução dos programas. A Figura 6 apresenta a estrutura da tabela *Shadow*. No exemplo da figura, Fluxo 1 é mapeado para ser processado apenas por P_1, P_4 e P_j, respectivamente. Enquanto o Fluxo 2 vai ser processado por P_3, P_4 e P_k em caso de *match* na tabela.

3.3. Composição de Deparsers

A composição de *deparser* é um processo mais simples. Porque o próprio *deparser* possui uma estrutura mais simples. A composição do *deparser* se dá apenas pela adição da primitiva que emite os cabeçalhos adicionais da extensão. Isto, é, os cabeçalhos que foram incluídos durante a extensão do *parser*, agora, devem ser emitidos na ordem correta.

4. Estudo de caso e Avaliação

Para validar nossa estratégia, desejamos mostrar o impacto que o mecanismo de composição traz para virtualização de vários programas P4. Para isso, construímos um

cenário de estudo de caso utilizando módulos de monitoramento e de controle de acesso (firewall) no próprio plano de dados. Adicionalmente, medimos o impacto de nossa estratégia de composição no processamento e encaminhamento de pacotes. Isso tudo, conforme a quantidade de funções (ou programas) inseridos aumenta. Como exemplo simples, apresentamos nessa seção a composição de um módulo de monitoramento a um *switch* simples, com encaminhamento de camada 2. Depois, mostramos como compor o firewall a esse mesmo programa.

Módulo de processamento/Monitoramento O módulo de processamento e monitoramento adiciona capacidades de armazenamento ao pipeline do programa. Isso permite que parte do processamento de funções de segurança aconteça no próprio plano de dados. Esse módulo armazena métricas sobre os fluxos (isto é, pacotes identificados pelo endereço IPv4 de destino e origem) e dispara um alerta para o plano de controle quando um determinado limite é atingido. O funcionamento deste módulo é baseado em estratégias de detecção de *Heavy Hitter* [Sivaraman et al. 2017] para identificar os fluxos que ultrapassam o limite. A composição do módulo de monitoramento ao programa *host*, estende os cabeçalhos do programa principal com as definições de cabeçalho IPv4 e suas respectivas definições. O fluxo de controle é composto estendendo as ações da tabela *shadow*, de maneira que elas incluam uma nova ação, a qual executa as operações do programa de monitoramento. Por fim, o *deparser* é estendido pela emissão do conteúdo do cabeçalho IPv4.

Firewall com Estado Propomos um firewall que inspeciona os cabeçalhos de camada 3. Ele funciona provendo uma interface para drop e reescrita de tipos específicos de pacotes conforme eles são interceptados. O firewall armazena o estado de novas conexões TCP no próprio switch (SYN & ACK = 1) e somente permite que a conexão seja emitida quando estabelecida. Ao contrário do que ocorre com alguns firewalls para redes definidas por software [Hu et al. 2014], em nossa proposta não há necessidade de tratar sobreposições de regras de *match+action*. Isso ocorre pois uma nova tabela é criada para o firewall e isolada das tabelas de funcionamento padrão do switch pela estratégia de composição. A composição do firewall incorpora ao *parser* de cabeçalhos o estado referente ao TCP. A definição da ação que reescreve os cabeçalhos é adicionada e o *deparser* começa a produzir cabeçalhos TCP.

4.1. Overhead de desempenho

Para avaliar nosso *framework*, utilizamos o *switch* de software bmv2 [1], em conjunto com o emulador mininet. Nós executamos os experimentos em um Intel(R) Core(TM) i3-6006U CPU @ 2.00GHz. O objetivo é mostrar o impacto da utilização de nossa estratégia para o atraso e a vazão dos fluxos processados pelo programa resultante da composição.

Para avaliar o atraso que a composição de novos programas traz para o comportamento usual do *switch*, nós configuramos um experimento que realiza 100 requisições e medimos o tempo em que uma requisição leva pra ser processada. Comparamos o atraso em um cenário utilizando o programa *host* contendo os módulos enunciados acima composto com o programa sem extensões e utilizamos tabelas 'Shadow' com 1024 entradas. Na Figura 7, identificamos como 'Shadow' o atraso gerado quando os pacotes são interceptados e combinam com alguma regra da tabela *Shadow*. Identificamos na figura

[1] https://github.com/p4lang/behavioral-model

como ‘P4’, o atraso gerado quando os fluxos são interceptados pela tabela *Shadow*, mas não combinam com nenhuma regra, consequentemente sendo processados apenas pelas funções usuais do switch (i.e. encaminhamendo de nível 2).

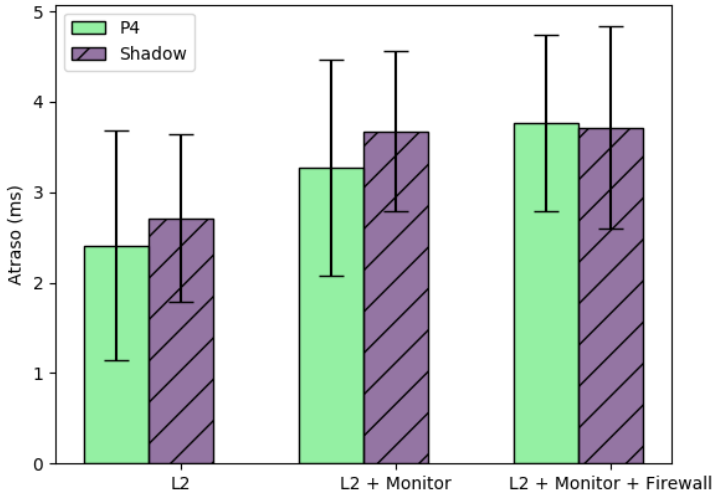

Figura 7. Desempenho e *Overhead* da Estratégia de Composição

Como pode ser visto na Figura 7, a utilização do mecanismo de composição não parece demonstrar sacrifício de performance. A variação de desempenho entre os dois cenários é limitada a uma percentagem relativamente pequena, sem ultrapassar a diferença de mais do que 0.4 ms entre o experimento utilizando a tabela *Shadow* e o experimento com P4 nativo. Isso mostra que a utilização ta tabela *Shadow* para isolar novos programas não degrada significativamente o desempenho das funcionalidades usuais do switch. Vemos como uma possibilidade de trabalho futuro comparar nossa solução com outras abordagens que proponham a composição de funções para plano de dados programáveis.

A Figura 8 apresenta o impacto que as novas funcionalidades trazem para a vazão, em Mbits por segundo. Para cada programa resultante da composição, avaliamos tanto a vazão pelo caminho usual do comportamento do switch (identificado como P4); e a vazão quando os fluxos são encaminhados e processados pelos módulos estendidos. É possível observar que a vazão reduz conforme os módulos que leem contéudos de cabeçalhos de camada mais alta são adicionados. A composição do *firewall* agregado ao módulo de monitoramento reduz pela metade a vazão quando é interceptado pela tabela *Shadow* (de 22Mbit/s para cerca de 12Mbit/s). Isso se deve tanto pela necessidade de decodificar mais bytes do cabeçalho, tanto pelo tempo de processando dos bytes no fluxo de controle. De qualquer maneira, é um preço aceitável a se pagar quando se deseja uma rede mais segura.

5. Trabalhos Relacionados

Programabilidade no plano de dados tem sido tipicamente empregada na virtualizacão de serviços que tradicionalmente eram engessados a *middleboxes* fechados ou ao circuito integrado dos *switches*. Nessa seção, apresentamos estudos já desenvolvidos sobre virtualização de funções do plano de dados e sobre estratégias de segurança e monitora-

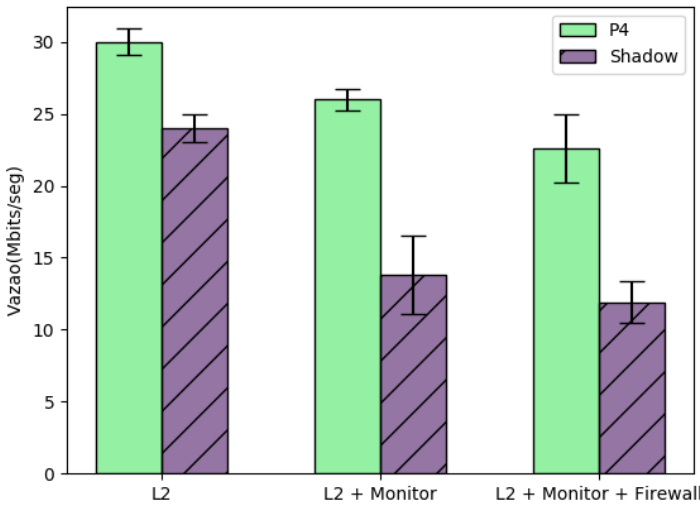

Figura 8. Impacto da composição dos módulos na vazão dos fluxos no plano de dados

mento. O esclarecimento sobre o que já foi produzido sobre o assunto ajudará a compreender a contribuição de nossa proposta para essas áreas.

Virtualização de Plano de Dados Em [Hancock and van der Merwe 2016], os autores propõem o Hyper4, um hypervisor para programas P4, cujo design permite a virtualização de vários programas P4. Dessa maneira, o Hyper4 possibilita que o administrador da rede altere dinamicamente a ordem lógica dos programas. Para isso, todavia, faz-se necessário um conjunto amplo de tabelas e primitivas de recirculação que permitam a execução de vários parsers. Em [Zhou and Bi 2017], os autores utilizam um número reduzido de tabelas, mas ainda exigem que os pacotes recirculem para novos programas serem inseridos. Em [Dimitropoulos et al. 2018], os autores propõem a virtualização de programas sem exigir recirculação de pacotes. Porém, eles não proveem isolamento entre as funções inseridas. Em [Zhang et al. 2017], também encontramos uma proposta de hypervisor utilizando P4, porém, empregando um número muito reduzido de tabelas para realizar o ordenamento topológico de maneira dinâmica. Diferentemente, nossa proposta utiliza apenas uma tabela adicional para suportar vários programas e não recorre à primitiva de recirculação.

Segurança no Plano de Dados Como qualquer outro paradigma, redes definidas por software necessitam de mecanismos para proteger seu funcionamento. Em [Hu et al. 2014] os autores propõem um firewall para redes SDN que executa em *switches* e permite resoluções efetivas de políticas de violação de firewall em redes OpenFlow. Para evitar a inserção de regras conflitantes que violem as políticas de segurança, os autores propõem uma camada para o plano de controle que resolve ambiguidades entre regras a serem inseridas. Em [Sonchack et al. 2016], os autores apresentam o OFX, sistema que permite a disposição de funções de segurança em switches, mas cuja estratégia não é adequada para processadores de pacotes genéricos. Em nosso trabalho, propomos um

mecanismo que permite implantar funções de segurança utilizando P4. Argumentamos que nosso design evita regras conflitantes entre diferentes aplicações/serviços e pode ser implantado em processadores genéricos de pacotes.

Monitoramento Trabalhos que permitem realizar o monitoramento de pacotes no plano de dados, são divididos entre aqueles que tentam fornecer abstrações para identificar e armazenar informações sobre fluxos de pacotes (heavy hitter e fluxos elefantes) [Sivaraman et al. 2017] e aqueles que fornecem mecanismos eficientes para telemetria e agregação da informação do plano de dados [Kim et al. 2015, Van Tu et al. 2017, Marques and Gaspary 2018]. Embora nosso trabalho seja quase ortogonal ao que propõem esses pesquisadores, acreditamos fornecer elementos que complementam seus estudos. Ao passo que aqueles não demonstraram a implantação de suas funcionalidades em um plano de dados de programável, entendemos que nossa proposta preenche essa necessidade.

6. Conclusões

Neste trabalho apresentamos uma estratégia de composição de programas P4 para estender a funcionalidade de dispositivos de planos de dados programáveis. A estratégia é dividida em uma etapa de composição da máquina de estados de *parser* de pacotes e em uma outra etapa complementar, em que as ações e os construtores do fluxo de controle são estendidos em uma arquitetura modular e que permite configuração dinâmica. Nós apresentamos um estudo de caso, mostrando o funcionamento do mecanismo para dois programas modulares: um módulo de monitoramento que utiliza técnicas de *heavy hitter* e um firewall que armazena o estado de conexões TCP. Os resultados das avaliações realizadas mostram que é possível compor programas para o plano de dados programável utilizando nossa estratégia sem impactar significativamente no atraso e vazão de processamento dos pacotes. Nós atribuímos isso ao uso muito reduzido de recursos, incluindo tabelas e lógica de controle.

Embora nossa estratégia garanta uma boa utilização dos recursos do switch ao compor módulos distintos, principalmente por causa da utilização de apenas uma tabela adicional, ela ainda enfrenta várias limitações. Em particular, a estratégia introduz alguns *overheads* ao plano de controle, exigindo que o desenvolvedor de um módulo seja responsável pela corretude do direcionamento dos pacotes.

Atualizações Consistentes Devido a essa limitação, existe a necessidade de uma outra etapa de verificação durante o funcionamento da rede para garantir que um pacote não passe por duas configurações distintas enquanto é processado. Isso ocorre pois os módulos internos podem possuir suas próprias tabelas e elas podem ser atualizadas de maneira que corrompa a configuração de direcionamento gerada pela tabela Shadow.

Regras Sobrepostas Embora a tabelha Shadow possa facilitar o direcionamento dos fluxos, a inserção de regras sobrepostas na tabela pode gerar direções conflitantes dentro do switch. Nós vemos como trabalho futuro o desenvolvimento de um mecanismo que filtre e resolva as sobreposições.

Operadores de Composição Esse trabalho focou em apresentar uma estratégia para compor e direcionar os fluxos pelos modulos compostos. Como parte de um trabalho em andamento nós investigamos a utilização de operadores de composição para facilitar ao administrador de redes a etapa de arranjo dos móudulos conforme suas necessidades.

Em trabalhos futuros, pretendemos utilizar a estratégia de composição em um *hypervisor* para planos de dados, onde as extensões possam ser adicionadas e removidas de maneira automática pelo operador de rede. Também desejamos construir uma interface adaptável para o plano de controle, eliminando a necessidade de reescrever as aplicações do plano de controle ao inserir um novo módulo no plano de dados. Também são necessários mecanismos que garantam que as atualizações dinâmicas não permitam que um fluxo passe por mais do que uma configuração enquanto o plano de dados é atualizado.

Referências

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. (2014). P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95.

Dimitropoulos, X. A., Dainotti, A., Vanbever, L., and Benson, T., editors (2018). *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2018, Heraklion, Greece, December 04-07, 2018*. ACM.

Feamster, N., Rexford, J., and Zegura, E. (2014). The road to sdn: An intellectual history of programmable networks. *SIGCOMM Comput. Commun. Rev.*, 44(2):87–98.

Freire, L., Neves, M., Leal, L., Levchenko, K., Schaeffer-Filho, A., and Barcellos, M. (2018). Uncovering bugs in p4 programs with assertion-based verification. In *Proceedings of the Symposium on SDN Research*, page 4. ACM.

Garcia, L. F. U., Villaça, R. S., Ribeiro, M. R. N., Martins, R. F. T., Verdi, F. L., and Marcondes, C. (2018). Minicurso introdução à linguagem p4 - teoria e prática. In *XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, Campos do Jordão, Brasil. SBC.

Hancock, D. and van der Merwe, J. (2016). Hyper4: Using p4 to virtualize the programmable data plane. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, pages 35–49, New York, NY, USA. ACM.

Hu, H., Han, W., Ahn, G.-J., and Zhao, Z. (2014). Flowguard: building robust firewalls for software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 97–102. ACM.

Jin, X., Liu, H. H., Gandhi, R., Kandula, S., Mahajan, R., Zhang, M., Rexford, J., and Wattenhofer, R. (2014). Dynamic scheduling of network updates. *SIGCOMM Comput. Commun. Rev.*, 44(4):539–550.

Katta, N. P., Rexford, J., and Walker, D. (2013). Incremental consistent updates. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 49–54. ACM.

Kim, C., Sivaraman, A., Katta, N., Bas, A., Dixit, A., and Wobker, L. J. (2015). In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM*.

Liu, J., Hallahan, W., Schlesinger, C., Sharif, M., Lee, J., Soulé, R., Wang, H., Caşcaval, C., McKeown, N., and Foster, N. (2018). p4v: Practical verification for programmable data planes.

Lopes, N., Bjørner, N., McKeown, N., Rybalchenko, A., Talayco, D., and Varghese, G. (2016). Automatically verifying reachability and well-formedness in p4 networks. Technical report, Technical Report.

Marques, J. A. and Gaspary, L. P. (2018). Explorando estratégias de orquestração de telemetria em planos de dados programáveis. In *Simpósio Brasileiro de Redes de Computadores (SBRC)*, volume 36.

Nguyen, T. D., Chiesa, M., and Canini, M. (2017). Decentralized consistent updates in sdn. In *Proceedings of the Symposium on SDN Research*, SOSR '17, pages 21–33, New York, NY, USA. ACM.

Reitblatt, M., Foster, N., Rexford, J., Schlesinger, C., and Walker, D. (2012). Abstractions for network update. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 323–334, New York, NY, USA. ACM.

Sivaraman, V., Narayana, S., Rottenstreich, O., Muthukrishnan, S., and Rexford, J. (2017). Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*, SOSR '17, pages 164–176, New York, NY, USA. ACM.

Sonchack, J., Smith, J. M., Aviv, A. J., and Keller, E. (2016). Enabling practical software-defined networking security applications with ofx. In *NDSS*, volume 16, pages 1–15.

Van Tu, N., Hyun, J., and Hong, J. W.-K. (2017). Towards onos-based sdn monitoring using in-band network telemetry. In *Network Operations and Management Symposium (APNOMS), 2017 19th Asia-Pacific*, pages 76–81. IEEE.

Zhang, C., Bi, J., Zhou, Y., Dogar, A. B., and Wu, J. (2017). Mpvisor: A modular programmable data plane hypervisor. In *Proceedings of the Symposium on SDN Research*, SOSR '17, pages 179–180, New York, NY, USA. ACM.

Zheng, P., Benson, T., and Hu, C. (2018). Shadowp4: Building and testing modular programs. In *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, pages 150–152. ACM.

Zhou, Y. and Bi, J. (2017). Clickp4: Towards modular programming of p4. In *Proceedings of the SIGCOMM Posters and Demos*, SIGCOMM Posters and Demos '17, pages 100–102, New York, NY, USA. ACM.