UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MARCELO PIZZUTTI

# Roteamento por Múltiplos Caminhos: Explorando a Operação Híbrida entre os Planos de Dados e de Controle

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Orientador: Prof. Dr. Alberto Egon Schaeffer-Filho

Porto Alegre
2019

**CIP — CATALOGAÇÃO NA PUBLICAÇÃO**

Pizzutti, Marcelo

Roteamento por Múltiplos Caminhos: Explorando a Operação Híbrida entre os Planos de Dados e de Controle / Marcelo Pizzutti. – Porto Alegre: PPGC da UFRGS, 2019.

81 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2019. Orientador: Alberto Egon Schaeffer-Filho.

1. Multipath routing. 2. Load balancing. 3. Flowlet. 4. Plano de controle. 5. Plano de dados. 6. Programabilidade P4. 7. Arquitetura híbrida. I. Schaeffer-Filho, Alberto Egon. II. Título.

## RESUMO

A programabilidade do plano de controle, habilitada sobretudo por SDN (*Software-Defined Networking*), trouxe novas possibilidades às práticas já consolidadas de gerenciamento e planejamento na área de redes de computadores. No entanto, estratégias que executam ações pontuais na manipulação dos fluxos de dados precisam de mecanismos mais dinâmicos com atuação além do plano de controle. Esse nível de gerenciamento pode ser efetivamente explorado por meio de interações mais robustas com o plano de dados, onde as decisões e respostas reativas podem ser executadas na velocidade do processamento de pacotes. A arquitetura P4 atinge o objetivo fornecendo diretivas que permitem definir o comportamento do *hardware* que executa as tarefas de encaminhamento de pacotes, trazendo a capacidade de programação para esse nível. Esta dissertação apresenta uma estratégia de roteamento *multipath* que utiliza uma arquitetura híbrida SDN e P4. No plano de dados, P4 e a abstração Flowlet são usadas para ativamente efetuar balanceamento de carga e roteamento através de caminhos assimétricos, além do monitoramento dos *links* ativos medindo a latência de cada rota paralela. No plano de controle, logicamente centralizado, a visão geral da rede é aproveitada para executar tarefas passivas mais elaboradas, que incluem o mapeamento de caminhos, a configuração de dispositivos e a atualização de rotas de forma assíncrona. Essa interação constitui a interface de comunicação entre os dispositivos de encaminhamento e o controlador e pode ser expandida para acomodar outras estratégias que atuam de forma centralizada para construir um entendimento dinâmico do ambiente e auxiliar no gerenciamento. Os resultados obtidos através da avaliação experimental apontam tempos favoráveis para transferências FCT (*Flow Completion Time*) nas estratégias que consideram o histórico de latência. Esses dados são coletados por meio do processo de medição dinâmica realizado pelos *switches* e enviados ao controlador que prossegue com a análise e a substituição de rotas conforme a demanda.



**Palavras-chave:** Multipath routing. load balancing. Flowlet. plano de controle. plano de dados. programabilidade P4. arquitetura híbrida.

**Multipath Routing: Exploring hybrid operation between data and control planes**

## ABSTRACT

The programmability of the control plane, enabled mainly by SDN (Software-Defined Networking), brought new possibilities to the already consolidated management and planning practices in the area of computer networks. However, strategies that perform timely actions in manipulating data streams need more dynamic mechanisms that act beyond the control plane. This level of management can be effectively exploited through more robust interactions with the data plane, where decisions and reactive responses can be executed at the speed of packet processing. The P4 architecture achieves this by providing directives that define the behavior of the hardware that performs packet forwarding tasks, bringing programming capability to this level. This dissertation presents a multipath routing strategy that uses an SDN and P4 hybrid architecture. In the data plane, P4 and Flowlet abstraction are used to actively perform load balancing and routing through asymmetric paths, as well as monitoring active links by measuring the latency of each parallel route. In the logically centralized control plane, the network overview is leveraged to perform more elaborate passive tasks, including path mapping, device configuration, and asynchronous route updating. This interaction constitutes the communication interface between the routing devices and the controller and can be expanded to accommodate other strategies that act centrally to build a dynamic understanding of the environment and assist in management. The results obtained through the experimental evaluation indicate favorable times for FCT (Flow Completion Time) transfers in the strategies that consider the latency history. This data is collected through the dynamic measurement process performed by the switches and sent to the controller which proceeds with the analysis and replacement of routes as required.

**Keywords:** Multipath routing, load balancing, Flowlet, control plane, data plane, programmability, P4, hybrid architecture.

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| API | Application Programming Interface |
| CAPEX | Capital Expenditure |
| EMA | Exponential Moving Average |
| FCT | Flow Completion Time |
| IP | Internet Protocol |
| ISP | Internet Service Provider |
| MPLS | Multi Protocol Label Switching |
| NFV | Network Function Virtualization |
| OPEX | Operational Expenditure |
| P4 | Programming Protocol-Independent Packet Processors |
| QoS | Quality of Service |
| RTT | Round Trip Time |
| SACK | Selective Acknowledgment |
| SDN | Software Defined Networking |
| TCP | Transmission Control Protocol |
| VLSM | Variable Length Subnet Masking |
| VoIP | Voice over Internet Protocol |
| WMA | Weighted Moving Average |

## LISTA DE FIGURAS

## LISTA DE TABELAS

**SUMÁRIO**

# 1 INTRODUÇÃO

A crescente demanda por conectividade e tolerância a falhas tem imposto diversos desafios para os profissionais envolvidos com as áreas responsáveis pela infraestrutura e gerenciamento de redes. Algumas tecnologias propostas na última década abordaram os conceitos de resiliência e de flexibilidade, centralizando o gerenciamento e segregando a administração em níveis. Tradicionalmente, a configuração dos ativos de rede está bastante relacionada com os equipamentos e com os recursos disponibilizados pelos fabricantes do *hardware*. Por isso, muito tempo é gasto no gerenciamento, o que inclui planejamento, execução e monitoramento, a fim de unificar a administração lógica por meio de configurações e permitir que políticas de gerenciamento de tráfego sejam aplicadas de forma uniforme.

Com a ampliação da Internet, algumas padronizações surgiram como forma de estruturar a configuração e tornar mais simples as mudanças em ativos de rede, tornando mais eficiente a execução de tarefas de gerenciamento. A abordagem introduzida pelas Redes Definidas por Software (SDN - *Software Defined Networking*) idealizou um conceito que desvincula os planos de controle e dados [Feamster, Rexford and Zegura 2014] e provê uma camada de abstração entre dispositivos físicos e lógicos. No plano de dados encontra-se a malha dos equipamentos de encaminhamento (*switches* e *routers*) e no plano de controle uma plataforma de software efetivamente responsável pelo planejamento e centralização de regras.

Outra iniciativa voltada à separação de funções de rede é a arquitetura NFV (*Network Functions Virtualization*) [Mijumbi et al. 2016]. Aqui se busca desacoplar a execução de funções de rede de um *hardware* específico, o que dinamiza a alocação de recursos, permitindo que os serviços sejam provisionados de forma escalável conforme a demanda. Essa segregação de funções de rede melhora o custo benefício da manutenção de operações por possibilitar que funções antes exclusivamente desempenhadas em equipamentos de rede sejam agora executadas em *hardware* de propósito geral. O ganho é observado tanto na simplificação e na agilidade do gerenciamento (OPEX - *operational expenditure*) quanto no menor custo de aquisição de ativos de hardware (CAPEX - *capital expenditure*).

Os conceitos sobre as SDN surgiram na década de 70 e após algum tempo o protocolo OpenFlow [McKeown et al. 2008] foi a implementação adotada pela indústria. No entanto, o *hardware* e o *software* embarcado (*firmware*) permanecem fortemente atrelados sendo difícil customizar as ações ou mesmo experimentar um novo protocolo, por

exemplo, pois isso requer uma nova versão do *software* para o equipamento disponibilizada pelo fabricante [Bosshart et al. 2014]. Outra limitação está relacionada às tarefas que necessitam de frequente interação entre controlador-*switch* para lidar com ações específicas sobre fluxos, devido aos atrasos de comunicação com o controlador. Ademais, a pluralidade de equipamentos e de configurações contribui para formar uma malha com características altamente heterogêneas.

Para proporcionar a definição de comportamento no plano de dados, o qual compreende os dispositivos que fazem o encaminhamento de pacotes, a arquitetura P4 possibilita definir a sequência lógica envolvida na interpretação e no processamentos dos fluxos de dados [Bosshart et al. 2014]. Tal recurso permite empregar com sucesso a programabilidade nos ativos de rede ao definir as rotinas e ações envolvidas de forma muito mais ampla e livre que aqueles presentes no OpenFlow. Em consonância com as SDN, a arquitetura P4 concebe a existência de uma unidade controladora conectada logicamente com todos os dispositivos de encaminhamento.

A arquitetura P4 apresenta uma proposta de inovação na maneira como desenvolvedores podem definir tarefas em dispositivos compatíveis para, por exemplo, testar um protocolo experimental. Similar à inovação trazida pelo OpenFlow [McKeown et al. 2008], a arquitetura P4 possibilita, com uma linguagem precisa, que a lógica não permaneça fixada pelo fabricante, mas que possa ser compilada e embarcada de forma independente.

## 1.1 Descrição do problema

As redes tradicionais atendem satisfatoriamente à comunicação na Internet por meio de uma série de protocolos de roteamento que atuam de maneira distribuída e empregam o melhor esforço para atingir o mínimo de qualidade. No entanto, pode-se observar que a prestação de serviços enfrenta uma série de desafios, alguns dos quais oriundos da multiplicidade de rotas, da diversidade de equipamentos, de configurações e até de imposições de políticas de roteamento entre provedores.

Em ambientes de crescente conectividade, técnicas que visam ao uso otimizado dos enlaces são importantes com a finalidade de propor inovações e buscar maior qualidade nos serviços de entrega de dados [Ferlin et al. 2016] [Irteza et al. 2017]. Como a comunicação usa naturalmente o princípio de compartilhar o meio, a subutilização de recursos é uma consequência inerente quando não se gerencia, por exemplo, a oportunidade

de ter disponíveis enlaces redundantes.

As técnicas que exploram a diversificação de rotas pelo uso de múltiplos caminhos (*multipath routing*), tornaram-se ferramentas importantes para atender às demandas por maiores taxas de transferência de dados e menores atrasos associados. O balanceamento de carga (*load balancing*) é complementar quando o objetivo é obter um uso otimizado dos *links* de conexão. *Multipath routing* [Qadir et al. 2015] tem a função de descobrir e estabelecer caminhos paralelos entre nós de origem e de destino, com objetivos relacionados principalmente à resiliência e ao desempenho. *Load balancing* [Prabhavat et al. 2012], por outro lado, está vinculado à técnica que infere a classificação e divisão dos fluxos a fim de obter o melhor uso dos canais sem causar excessivos efeitos adversos como reordenamento de pacotes ou retransmissões. Nesse sentido, *multipath routing* e *load balancing* [Qadir et al. 2015] promovem o uso racional de enlaces de rede, explorando caminhos paralelos entre nós de origem e de destino.

Algumas abordagens foram propostas com o objetivo de explorar a programabilidade dos planos de dados e de controle empregando múltiplos caminhos e balanceamento de carga [Qadir et al. 2015] [Domżał et al. 2015] [Wójcik et al. 2016]. Técnicas de balanceamento de carga que utilizam o protocolo OpenFlow apresentam maior granularidade na operação, também devido às características de seleção de fluxo e à necessidade de comunicação com o controlador, gerando troca de mensagens que estão sujeitas à latência do meio. Essa abordagem não permite que um alto nível de reatividade seja aplicado ao mesmo nível de processamento das regras de roteamento, residentes nos *switches* ou roteadores. Embora as estratégias básicas que operam sobre fluxos, que são definidos pelas informações dos cabeçalhos IP e TCP, apresentem bons resultados e um baixo nível de complexidade conseguindo satisfatoriamente atender à infraestrutura atual, a expansão da malha de comunicação requer mecanismos inovadores. HULA [Katta et al. 2016], por exemplo, implementa sua estratégia para obter balanceamento de carga diretamente no plano de dados ao utilizar a arquitetura P4. Seu uso é direcionado para topologias hierárquicas em ambiente *datacenter*.

Assim, considerando-se os avanços das tecnologias relacionadas ao plano de dados e o suporte à programabilidade que é agregado neste nível, a oportunidade percebida é a da releitura das abordagens que exploram a aplicação de roteamento por múltiplos caminhos empregando uma forma de operação híbrida entre o plano de controle e o plano de dados com foco na capacidade de resposta alcançada através de programabilidade. A oportunidade de interação e modificações mais dinâmicas que as abordagens que uti-

lizam OpenFlow, por exemplo, oferece a integração da visão unificada que o plano de controle obtém naturalmente com a capacidade de definir novas tarefas executadas sobre cada *frame* processado. Portanto, este trabalho busca esquematizar esta operação híbrida, mediante a exploração de mecanismos que atuem na análise de dados coletados dos dispositivos de encaminhamento (*switches)* para conseguir resultados diferenciados com a participação do controlador.

Com o objetivo de efetuar monitoramentos mais precisos e explorar múltiplas rotas de forma que seja possível sincronizar informações entre os planos, é preciso pré-definir caminhos e rotulá-los. Isso implica identificar unicamente caminhos e, ao mesmo tempo, efetuar um particionamento na topologia a fim de mitigar a geração de uma quantidade excessiva de rotas únicas.

As lacunas observadas constituem uma ocasião para unir algumas técnicas que podem ser desenvolvidas no plano de dados com maior precisão, como um método para realizar balanceamento de carga por múltiplas rotas, e um ambiente de interação com o controlador que agrega uma visão global do estado da topologia.

## 1.2 Objetivos

Este trabalho desenvolve uma estratégia que habilite roteamento usando múltiplos caminhos (*multipath routing*) e balanceamento de carga (*load balancing*) no plano de dados em *links* heterogêneos, em ambiente intra-domínio e em conjunto com o plano de controle. Agindo diretamente no plano de dados é possível construir arranjos lógicos que explorem técnicas destinadas à utilização de enlaces físicos com um nível de granularidade menor nas operações de decisão sobre fluxos. A troca de informações com o plano de controle tem por objetivo criar uma visão global que propõe mudanças de roteamento na medida em que se detecta um estado de congestionamento.

O objetivo, portanto, é propor uma arquitetura híbrida que opere de forma conjunta através de dois *control-loops* independentes: *(i)* um ciclo de controle *on-line*, ativo, localizado no plano de dados, responsável por ações imediatas situadas no nível de processamento de pacotes; e *(ii)* um ciclo *off-line*, reativo, residente no controlador, responsável por algumas tarefas, tais como a definição e análise do estado das rotas e alterações com visão global. Por meio dessa separação, tarefas ligadas à alta responsividade são posicionadas próximas ao fluxo de decisão nos dispositivos de encaminhamento, trazendo agilidade na execução de funções, tais como a divisão e o monitoramento de fluxos pela

observação indireta do comportamento da camada de transporte (TCP). Outras atividades podem ocorrer de forma assíncrona no plano de controle, que possui conhecimento mais abrangente, o que acaba por beneficiar decisões mais elaboradas.

Portanto, acredita-se que o gerenciamento de redes tradicionais que aplicam *multipath routing* e *load balancing* pode obter muitos benefícios através da integração possibilitada pela programabilidade dos planos de dados e de controle. As oportunidades que surgem a partir dessas aplicações podem ser direcionadas para aperfeiçoar soluções que até então permaneciam exclusivamente relacionadas ao processador (*hardware*) dos *switches*. Estas iniciativas habilitam o desenvolvimento de estratégias atualizáveis e integradas com o plano de controle de forma muito mais fácil.

## 1.3 Contribuições

Por meio deste trabalho, nossas contribuições pretendem ser:

*(i)* Uma proposta de organização dos componentes e de distribuição de funções entre os planos de dados e de controle para executar o roteamento por múltiplas rotas e balanceamento de carga utilizando a técnica Flowlet [Kandula et al. 2007];

*(ii)* A utilização de programabilidade no plano de dados para a execução de tarefas que demandam alta responsividade juntamente com o emprego de um componente controlador para interoperar com os *switches*;

*(iii)* A elaboração de uma estratégia destinada à descoberta e à seleção de rotas paralelas em topologias que apresentam multiplicidade de conexões e estejam inseridas em ambiente intra-domínio;

*(iv)* A aplicação de um mecanismo para monitoramento do estado dos *links* no plano de dados. Isto permite que no controlador atuem-se estratégias que selecionem rotas alternativas com base no histórico de medições.

Espera-se com este estudo contribuir com o desenvolvimento das atuais estratégias, abordando a programabilidade disponível para o plano de dados. A flexibilidade que pode ser alcançada na intersecção entre as camadas operacional e gerencial contribui para melhor utilizar os enlaces instalados e também para aplicar condutas que visem alcançar maior disponibilidade através da redundância disponível no nível lógico.

## 1.4 Estrutura da dissertação

Na sequência da dissertação, o Capítulo 2 apresenta os principais conceitos e tecnologias que influenciaram a estratégia apresentada, envolvendo ideias relacionadas aos planos de dados e de controle (SDN) e à arquitetura P4, além de conceitos de roteamento *multipath* e *load balancing*. No Capítulo 3, são relacionados os trabalhos que apresentam similaridades ou que nos auxiliaram na formulação das ideias aplicadas na proposta apresentada. No Capítulo 4, detalha-se a proposta para alcançar *multipath routing* e *load balancing* através de uma abordagem híbrida que atue de forma cooperativa e assíncrona entre os planos de dados e de controle. No Capítulo 5 apresenta-se uma avaliação experimental da proposta com foco no controlador, onde estratégias são alteradas para verificar o impacto no comportamento do conjunto. Por fim, no Capítulo 6, são expostas as conclusões, as considerações finais e possíveis direcionamentos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentadas as principais ideias que impulsionaram a aplicação da programabilidade em componentes de rede e também algumas das estratégias que permitem o uso de roteamento através de múltiplos caminhos. Estas abordagens visam melhorias no desempenho ou propõem estruturas que promovem uma melhor utilização dos recursos disponíveis.

## 2.1 Programabilidade em redes

O conceito de redes programáveis vem sendo modelado juntamente com a crescente utilização dos recursos de conectividade e dos avanços tecnológicos aplicados à arquitetura do *hardware* de equipamentos de rede [Kreutz et al. 2015]. O aumento do poder computacional, a consolidação de técnicas de programação e a ampliação das malhas de comunicação intensificaram a complexidade do gerenciamento dos ativos de rede. Na gênese desse conceito, encontram-se várias ideias que contribuíram para auxiliar as tarefas ligadas à engenharia de tráfego, especialmente direcionadas ao fomento de um ambiente dinâmico cujo comportamento pode ser modelado independentemente do *hardware*, de forma relativamente estável e escalável.

Muitas soluções abordam o conceito de controlador que atua administrativamente de forma centralizada estabelecendo um canal de comunicação com cada dispositivo e manipulando diversas funções nos ativos de redes, como roteadores e *switches*, reduzindo a necessidade de intervenção humana efetuada de forma isolada em cada componente da infraestrutura.

As ideais envolvidas com *Active networks* pertencem aos primeiros esforços para ampliar as funções de dispositivos de rede, tornando-os executadores de códigos [Tennenhouse et al. 1997] [Calvert et al. 1998]. Com a aplicação destas ideias, pretendia-se estender as triviais e necessárias funções desempenhadas pelos equipamentos, como *switches* e roteadores, envolvendo-os de forma ativa no processamento das funções de rede. A abordagem principal consistia em criar um canal de comunicação que possibilitasse, mesmo que indiretamente, a alteração dinâmica durante o funcionamento dos estados, das funções e dos serviços providos para o usuário através de equipamentos de rede de propósito mais geral. Um dos modelos foi o de inserir instruções executáveis em pacotes que, ao trafegarem pelos *links*, a cada salto programassem aspectos dos equipamentos.

Outra proposta foi a introdução da noção de uma interface paralela de comunicação que permitisse ao administrador o envio de instruções/códigos, de modo a separar as instâncias de processamento das de configuração/programação.

Inspirados na ideia de separação dos planos de dados e de controle, avanços no plano de controle ganharam notoriedade. ForCES (Forwarding and Control Element Separation) [Yang et al. 2004] e Linux Netlink [Khosravi et al. 2003] são exemplos de projetos que materializam interfaces para comunicação do plano de controle com os demais dispositivos. Porém, apesar do desenvolvimento de um conjunto de soluções, não houve inovações significativas implementadas e utilizadas pela indústria. A falta de objetividade e praticidade das propostas não motivou os fabricantes e os grandes provedores de serviços, potencialmente os maiores interessados nessas tecnologias.

### 2.1.1 SDN e OpenFlow

Redes definidas por *software* incorporaram princípios de redes programáveis e são fortemente caracterizadas pela separação entre as camadas operacional e de gerenciamento: plano de dados e plano de controle. No plano de dados encontra-se o *hardware* que abriga as funções de encaminhamento de pacotes, enquanto que no plano de controle reside a porção de inteligência que orquestra as funções relacionadas à configuração e ao roteamento. Desta separação deriva a ideia de administração centralizada, função desempenhada pelo controlador que fica posicionado na camada de gerenciamento e logicamente conectado a todos os componentes do substrato físico. Isto cria um mapeamento que possibilita unificar a administração.

A necessidade de manter a coerência, a harmonia e a eficiência das regras, somada à grande heterogeneidade de dispositivos e de interfaces de administração, impulsionaram a aplicação de programabilidade como ferramenta. O protocolo OpenFlow [McKeown et al. 2008] fundamentou sua importância como padrão na aplicabilidade alcançada e na adesão pelos fabricantes de *hardware* como interface de comunicação empregada no canal de comunicação entre os dois sistemas: um *software* controlador e um *firmware* (sistema embarcado) que recebe as regras do controlador.

Em OpenFlow, o controlador estabelece uma sessão TCP com cada *switch*, geralmente assegurada com criptografia, e cada *switch* mantém uma estrutura de dados que armazena as regras enviadas pelo controlador. As principais estruturas - tabelas - são: *Flow Table*, *Group Table* e *Meter Table*. Em cada *table* é possível criar critérios compos-

tos por um conjunto de cabeçalhos (*flow entries*) que serão confrontadas com os fluxos de dados recebidos por um *switch*. Cada regra é associada a uma ação pré-definida sobre o pacote. *Flow table*, por exemplo, seleciona de forma única uma entrada com base na prioridade e na expressão (*match*) sobre os cabeçalho dos protocolos. Para cada *flow entry* um conjunto de instruções pode ser atrelado e, quando nenhuma regra retornada na busca, é disparada uma ação padrão definida previamente pelo controlador. Tal ação é conhecida como entrada *table miss* e comumente resulta no envio do pacote ao controlador para ser analisado. Isso caracteriza uma configuração reativa, podendo impor alguma latência, a depender da comunicação com o controlador. De outra forma, também é possível que o controlador instale novas regras sem que um evento *table miss* aconteça, ou seja, agindo proativamente. Outro atributo importante nas regras é o de expiração, que define o tempo em segundos que uma entrada permanece na memória do equipamento, que fica responsável por fazer essa reciclagem. A ação de remoção também pode ser solicitada pelo controlador.

Cada *switch* deve instanciar pelo menos uma *Flow Table*, de número 0, e as outras são sequencialmente numeradas. Cada regra tem uma prioridade em sua tabela, contém os atributos que serão analisados e a instrução de processamento, além de poder ter um atributo de expiração (*timeout*), entre outros. As instruções são conjuntos de ações que podem alterar os cabeçalhos do pacote, o fluxo de processamento ou ainda executar operações sobre outras tabelas. A estrutura *Group Table* permite realizar ações sobre um grupo de pacotes e sobre um conjunto de portas fornecendo, por exemplo, uma abstração para implementar roteamento *multipath* ou *multicast*. A estrutura *Meter Table* armazena dados de métrica por fluxo, informações necessárias para operações de QoS (*Quality of Service*), por exemplo.

A API do OpenFlow, que define as diretivas que analisam os cabeçalhos, as regras e as ações sobre os fluxos é bem definida e fixa. Portanto, a composição das regras constitui a parte dinâmica das operações. Isso dificulta algumas atividades, como a implementação de tarefas personalizadas, por exemplo, a execução de validações e de operações em fluxos mantendo o estado entre pacotes. Tarefas que não requerem uma interação frequente com o controlador ou são assíncronas operam de forma satisfatória. As SDN também trouxeram alguns desafios adicionais, tais como a questão do controlador como ponto de falha [Zhang et al. 2018], contudo isso está relacionado a toda a arquitetura que utiliza esse paradigma.

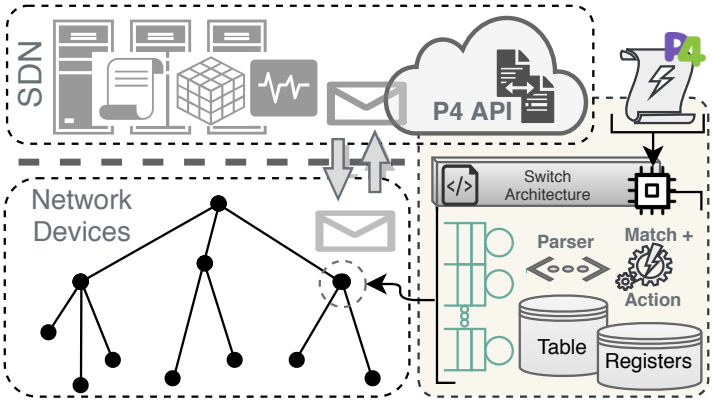
Com OpenFlow muitas habilidades de gerenciamento são significativamente me-

lhoradas, pois abordou-se diretamente o gerenciamento de rede relacionado ao monitoramento e configuração de equipamentos partindo da centralização do controlador, o que apresentou novas perspectivas de organização. Um sinal dessa evolução é o desenvolvimento de vários controladores, proprietários ou *open sources* [Kreutz et al. 2015] que corroboram a ideia da liberdade fornecida aos administradores de rede. Em redes tradicionais, onde um número certo de protocolos operacionalizam a Internet: IPv4, IPv6, TCP, UDP, BGP, OSPF, IS-IS etc, o tratamento desse conjunto mínimo já satisfez a necessidade de engenharia de tráfego, permitindo ao OpenFlow receber admirável visibilidade em redes programáveis. No entanto, o fator de dependência do lançamento de novo *firmware* pelo fabricante, para novas versões da especificação, limitou o aprimoramento de novas funcionalidades.

### 2.1.2 Programabilidade no plano de dados

Os avanços na área de *design* de *hardware* são um campo promissor para o desenvolvimento de plataformas que, como P4 [Bosshart et al. 2014] e Domino [Sivaraman et al. 2016], conseguem interagir com um *chip* de propósito mais geral a fim de ampliar as possibilidades de desenvolvimento de aplicações no processamento de fluxos de rede. Assim, *hardwares* específicos e com custo de produção elevado podem ser substituídos por arquiteturas genéricas e programáveis que, pela demanda, tendem a ter o custo de manufatura reduzido. A Figura 2.1 apresenta uma visão geral dos paradigmas envolvidos com a programabilidade e a nova interação adicionada pela arquitetura P4.

Figura 2.1: Visão geral da programabilidade em redes atuais.



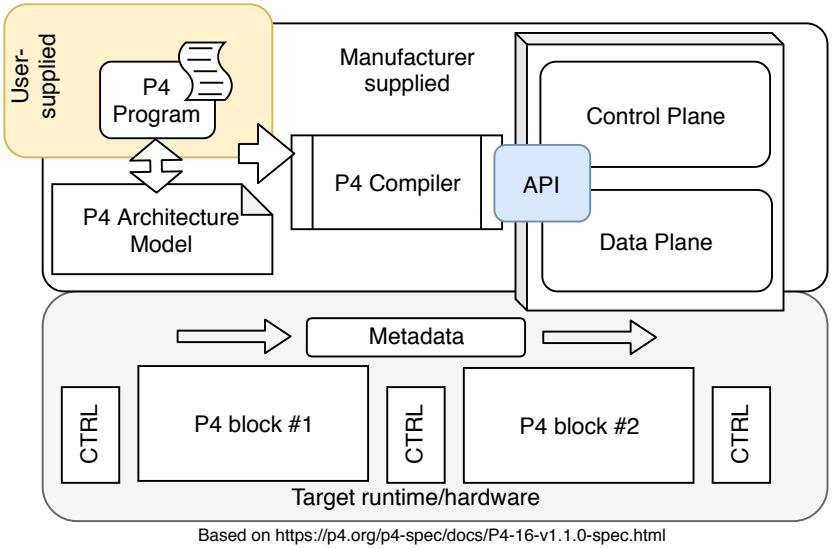

Fonte: O autor.

P4, nome advindo de *"Programming Protocol-independent Packet Processors"* [Bosshart et al. 2014], define uma linguagem que gera um artefato de *software* para

ser embarcado em equipamentos habilitados para executarem estas instruções codificadas. O artefato gerado, escrito na linguagem P4, simboliza um acordo que o equipamento deve atender e, portanto, resume um dos objetivos propostos pelo projeto: desenvolver aplicações para manipular pacotes com implementação independente do *hardware*. Esse novo tipo de interação com os dispositivos de encaminhamento pode efetivamente levar programabilidade ao plano de dados em escala muito maior que OpenFlow atingiu.

Um código P4 é compilado especificamente para uma plataforma e requer que o compilador seja compatível com a arquitetura do *hardware*. A Figura 2.2 apresenta uma visão geral dos elementos que compõem a arquitetura P4.

Figura 2.2: Visão geral dos componentes da arquitetura P4.



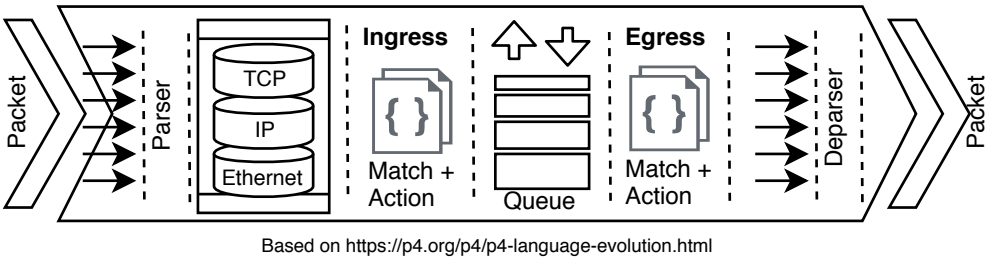

Fonte: O autor.

Uma API possibilita que um controlador altere os objetos definidos no plano de dados. Em relação ao OpenFlow, o *switch* se torna uma unidade muito mais versátil já que interpreta o artefato de *software* gerado pelo compilador P4 sem depender da intervenção do fabricante para que novas funcionalidades lógicas sejam embarcadas no dispositivo. Portanto, P4 permite programar o comportamento do comutador com um nível de detalhe muito maior. Em contraste ao OpenFlow, a dependência criada com relação ao desenvolvedor do dispositivo é relativamente menor, pois não está atrelada a funcionalidades específicas ou protocolos, mas ao compilador que atende à especificação da linguagem.

A linguagem P4 especifica a sintaxe, as regras semânticas, os blocos lógicos e suas interfaces e os tipos de dados. A arquitetura também define um conjunto de instruções e estruturas que possibilita ao desenvolvedor definir um manipulador de pacotes em *software* para ser executado no plano de dados. Os pacotes, ao ingressarem por uma porta,

são submetidos a uma sequência lógica (o *pipeline*) de blocos. Os cabeçalhos, as estruturas *statefull* internas e os metadados, que acompanham o pacote na sequência lógica, constituem as fontes de informações utilizadas nas decisões de processamento. Então é definida uma máquina de estados finita para leitura dos cabeçalhos dos pacotes (*Parser*) no *pipeline*, que são extraídos, processados e submetidos às estruturas lógicas definidas pelo usuário. Trata-se de uma espécie de cadeia, formada entre os blocos de controle e suas tabelas, que contêm chaves de pesquisa e ações correspondentes de análise ou de modificação de dados, semelhante com as *Flow Tables* do OpenFlow. O processamento dos pacotes acontece de forma paralela e, ao final, procede-se com a remontagem dos cabeçalhos (*Deparser)* e respectiva destinação de cada pacote (Figura 2.3).

Figura 2.3: Componentes de processamento P4.




Fonte: O autor.

O tratamento de funções externas à arquitetura central do P4 está previsto para ser implementado através de interfaces definidas pelo comando *extern*, acomodando tarefas complexas executadas no plano de dados e associadas a bibliotecas externas fornecidas pelo fabricante do equipamento ou por terceiros. Atividades complexas também podem ser executadas com o auxílio do plano de controle, de forma análoga ao que foi observado no OpenFlow. Um esforço para definir um padrão de arquitetura de *switch*[1] propõe um conjunto de componentes e algumas funções mínimas que tornam o código portável entre equipamentos de fabricantes diferentes.

## 2.2 Roteamento por múltiplos caminhos (*multipath routing*) e balanceamento de carga (*load balancing*)

Roteamento por múltiplos caminhos é incorporado ao projeto de rede para enumerar e usufruir de caminhos paralelos que podem ser considerados como rotas alternativas em uma topologia. A técnica traz muitos benefícios, não apenas em termos de redun-

[1]https://p4lang.github.io/p4-spec/docs/PSA.html

dância, mas também para aprimorar a taxa de transferência quando combinada com balanceamento de carga [Qadir et al. 2015]. A granularidade da divisão [Prabhavat et al. 2012] pode variar de pacote para pacote, o que na prática pode gerar diversos efeitos negativos; por fluxo, identificado pelas informações do cabeçalho TCP/IP; e pela divisão de um fluxo em subfluxos, como Flowlet [Kandula et al. 2007], que alcança maior flexibilidade ao dividir fragmentos que são roteados por caminhos diferentes independentemente. Outro aspecto relacionado à granularidade diz respeito às características observadas do trajeto. ECMP (*Equal-cost multi-path routing*) [Hopps 2000], por exemplo, na implementação mais comum, considera caminhos que têm como métrica o número de saltos (*hop-by-hop*). Já o WCMP (*Weighted Cost Multipathing for Improved Fairness in Data Centers)* [Zhou et al. 2014] apresenta uma proposta de distribuição do tráfego de maneira proporcional, respeitando a largura de banda disponível dinamicamente. No protocolo OSPF (*Open Shortest Path First*), observa-se que parâmetros como capacidade do *link* podem ser considerados na definição do custo da rota.

*Multipath routing* habilita o balanceamento de carga ao mesmo tempo que alcança resiliência através do uso de rotas redundantes de mesmo destino. No balanceamento de carga, a divisão por fluxo virtualmente fixa um caminho e tende a não gerar efeitos adversos, como reordenação de pacotes, o que pode desorientar os controles aplicados pelo protocolo TCP e induzir retransmissões e a redução da vazão de dados [Qadir et al. 2015] [Li et al. 2016] [He and Rexford 2008] [Domżał et al. 2015]. No entanto, a estratégia está sujeita à perda de desempenho devido a fluxos longos e que acabam por estabelecer congestionamento, não acontecendo a alternância entre rotas paralelas, o que acaba por criar gargalos. Flowlet [Kandula et al. 2007] está relacionado a esse problema: define que um fluxo seja dividido em subfluxos transmitidos independentemente por caminhos alternados. A definição do intervalo mínimo que permite alteração de rota é o fator envolvido na mitigação de possíveis efeitos adversos, como a reordenação de pacotes. Espera-se que isso acabe promovendo um equilíbrio no uso dos enlaces, já que a divisão aplicada sobre um fluxo permite o uso de caminhos paralelos de acordo com os intervalos de comunicação que ocorrem na transmissão.

Considerando-se as redes de computadores tradicionais, ou seja, que não empregam SDN e/ou NFV, *multipath routing* e *load balancing* podem ser analisados de acordo com a camada TCP/IP na qual operam. No nível de enlace, SPB (*Shortest Path Bridging*) [IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks–Amendment 20: Shortest Path

Bridging 2012] e TRILL (*Transparent Interconnection of Lots of Links*) [Perlman et al. 2011] habilitam múltiplos caminhos de custo igual em redes de escopo local, ambos calculando os caminhos mais curtos.

Avançando para a camada 3, inter-redes, na qual os mecanismos de roteamento agem de forma cooperativa e descentralizada, o paradigma de roteamento *hop-by-hop* é amplamente utilizado [Qadir et al. 2015]. Protocolos de roteamento bem estabelecidos, como o OSPF (*Open Shortest Path First*) e o BGP (*Border Gateway Protocol*), juntamente com o MPLS (*Multiprotocol Label Switching*), são empregados em escala global. Propostas [Wójcik et al. 2016] [Domżał et al. 2015] explorando recursos de *multipath routing* e *load balancing* são relatadas na literatura técnica, trazendo inovações ou ampliando funcionalidades para diversos cenários. Um destaque é dado à proposta MIRO [Xu and Rexford 2006], compatível com o protocolo BGP, e ao GMPLS (*Generalized Multi-Protocol Label Switching*) [Wójcik et al. 2016] [Mannie 2004], que possibilita ao MPLS (*Multi Protocol Label Switching*) estabelecer e administrar múltiplos caminhos. Essas propostas abordam a questão do balanceamento com a granularidade de fluxo pois atuam abaixo da camada de transporte TCP, que é efetivamente responsável pela conexão fim-a-fim. Nesse contexto, o MPTCP (*Multipath* TCP) [Ford et al. 2013] figura com contribuições importantes por ser compatível com o protocolo TCP regular e adicionar um nível de abstração que separa uma sessão TCP em subfluxos associando-os a caminhos diferentes, o que faz o balanceamento de carga com precisão muito maior. Para usufruir disto, o remetente ou o destinário devem possuir mais de uma ligação (*multihomed*) com a WAN (*Wide Area Network*). As propostas de *multipath routing* e *load balancing* que focam na camada de transporte também devem garantir a manutenção da justiça entre os fluxos (*fairness*) e prover procedimentos elaborados para o controle de congestionamento [Xu, Zhao and Muntean 2016].

Dentre as técnicas que promovem *multipath routing* utilizando a escala de divisão de subfluxos, a taxonomia apresentada em [Prabhavat et al. 2012] categoriza FLARE [Kandula et al. 2007], que emprega Flowlet, como um mecanismo adaptativo de balanceamento que analisa as condições do tráfego e da rede para operar. FLARE é um mecanismo de divisão de fluxos que implementa sua lógica para mitigar o reordenamento de pacotes. Sua unidade, o *flowlet*, é caracterizada como um conjunto (*burst*) de pacotes pertencentes a uma mesma janela. A segmentação dos fluxos está subjugada por um critério marcador que deve examinar a latência das rotas concorrentes de forma contínua, denominado *Minimum Time Before Switch-ability* (MTBS). FLARE tem três componentes: um estimador

de MTBS, um atribuidor de *Flowlet* por rota com base nos cabeçalhos e um algoritmo de *Token-counting* responsável pelo gerenciamento da carga de cada *link*.

A abordagem utilizada neste trabalho usa conceitos desta Seção para fazer a releitura de algumas técnicas e promover a interoperabilidade entre os planos de dados e de controle. Utiliza-se a abordagem Flowlet, aplicada em FLARE, mas de forma simplificada para operar na arquitetura P4 como elemento de programabilidade do plano de dados. Além disso, são definidos processos que atuam no plano de controle de maneira cooperativa e executam tarefas abordadas tradicionalmente por SDN.

# 3 TRABALHOS RELACIONADOS

Nesta capítulo serão apresentados trabalhos correlatos que envolvem redes programáveis, *multipath routing*, *load balancing* ou alguma técnica que vise ao melhor aproveitamento de recursos de rede. As estruturas e funcionalidades que diferenciam cada aplicação no seu contexto são organizadas na Tabela 3.1, que apresenta um resumo estruturado com a identificação do plano de atuação, do escopo e do modelo de gerenciamento identificado. O plano de ação e a estratégia correspondem aos recursos detectados de *software* ou de *hardware* que apresentam maior impacto na performance.

B4 [Jain et al. 2013] utiliza os princípios de SDN empregando OpenFlow em escala global. Propõe simultaneamente lidar com os protocolos de roteamentos tradicionais aliando-os às técnicas de engenharia de tráfego centralizadas que alocam largura de banda entre serviços concorrentes baseando-se na prioridade da aplicação para atender a demanda elástica de tráfego de dados. O projeto apresentou bons resultados usando *merchant switch silicon* integrado com OpenFlow e movendo funcionalidades do plano de dados para o processando centralizado. Com isso muitas possibilidades relacionadas ao gerenciamento e à tolerância a falhas puderam ser suportadas por técnicas conhecidas e empregadas em sistemas distribuídos. Uma implementação personalizada de ECMP *hashing* atua para executar o balanceamento de carga e promover uma abstração dos caminhos disponíveis na topologia.

CONGA [Alizadeh et al. 2014] foi projetado para componentes de alto rendimento e atua na topologia de *datacenters* utilizando encapsulamento VXLAN (*Virtual Extensible LAN*) na troca de informações. Através de administração centralizada, mantém-se atualizado do estado global de congestionamento da rede e utiliza a técnica *Flowlet* para atingir uma granularidade otimizada. CONGA analisa o estado de congestionamento pelo *feedback* e opera na escala de microssegundos. Cada pacote carrega uma métrica de congestionamento que é atualizada *hop-by-hop* e indica a utilização e congestionamento do caminho. O nó de origem efetua as decisões de balanceamento de carga com base nas métricas de congestionamento. O algoritmo DRE (*Discounting Rate Estimator*), similar ao EWMA (*Exponential Weighted Moving Average*), é empregado para determinar a carga do *link*. No projeto o balanceamento de carga é feito na camada de rede e mantém conhecimento do estado global de congestionamento para lidar com assimetrias.

Tabela 3.1: Tabela comparativa dos trabalhos relacionados

| **Arquitetura** | **Atributos/ Estratégia de Decisão** | **Plano de ação** | **Escopo/ Topologia** | **Plataforma** | **Gerenciamento** | **Algoritmo/ Técnica** |
|---|---|---|---|---|---|---|
| **B4 [Jain et al. 2013]** | Aplica divisão de fluxos sobre múltiplos caminhos conforme capacidade/demanda detectada. | dados e controle | global/ *datacenter* | OpenFlow e *merchant switch silicon* | centralizado | variante de ECMP *hashing* |
| **CONGA [Alizadeh et al. 2014]** | Utiliza Flowlet para conseguir alta granularidade e mede o estado de congestionamento usando o algoritmo DRE (*Discounting Rate Estimator*). | dados | *datacenter* | ASIC | distribuído | Flowlet |
| **HEDERA [Al-Fares et al. 2010]** | Detecta grandes fluxos nos *switches* de borda e escolhe o melhor caminho. | dados e controle | *datacenter* | OpenFlow | centralizado | *Global First Fit* e *Simulated Annealing* |
| **HULA [Katta et al. 2016]** | Cada *switch* somente determina o próximo melhor *hop*; utiliza testes períodicos para descoberta de conectividade. | dados | *datacenter* | P4 | distribuído | Flowlet |
| **MicroTE [Benson et al. 2011]** | Divide logicamente o tráfego entre previsível e imprevisível e apresenta uma heurística ECMP para esta classificação. | dados e controle | *datacenter* | OpenFlow | centralizado | heurística baseada em ECMP |
| **SWAN [Hong et al. 2013]** | Apresenta mecanismos para atualização consistente do planos de dados e implementa controle de fluxo nos *switches*. | dados e controle | *inter-datacenters* | OpenFlow | centralizado | *não identificado |
| **Presto [He et al. 2015]** | Realiza balanceamento de carga com granularidade fina e alta estabilidade através de sub-fluxos com unidade *flowcell* de 64KB. Apresenta mecanismos para lidar com falhas e trabalhar com *links* assimétricos. | dados (*soft-swithes*) e controle | *datacenter* | OpenFlow | centralizado | *flowcell* |
| **LetItFlow [Vanini et al. 2017]** | Aplica simplesmente Flowlet sobre fluxos e randomicamente seleciona o salto sucessivo para o grupo de pacotes. | dados | *datacenter* | P4 | distribuído | Flowlet |
| **nossa abordagem* | Utiliza dois *control-loops* integrados que trocam informações sobre o estado das rotas. A técnica Flowlet é aplicada no plano de dados. O plano de controle age passivamente modificando as rotas usadas no balanceamento de carga e no roteamento. | dados e controle | genérico | P4 e controlador | centralizado | Flowlet |

HEDERA [Al-Fares et al. 2010], aplicado em *datacenters*, monitora os fluxos ativos de forma global e centralizada. Apresenta e compara dois algoritmos, *Global First Fit* e *Simulated Annealing*, para estimar a largura ideal de banda compartilhada para cada fluxo e conseguir assim alocar o tráfego de forma balanceada. Os resultados apresentados também demonstram que com um gerenciamento global foram conseguidos resultados superiores às técnicas que usam ECMP para balanceamento de carga e atenuação da ocorrência de colisão de *hash*.

HULA [Katta et al. 2016] tem o projeto CONGA como mais próximo. Trouxe a abordagem de programabilidade com alto desempenho no plano de dados usando a linguagem P4. Cada *switch* atua de forma independente, empenhando-se para definir somente o próximo melhor salto (*hop*), o que racionaliza o uso de memória dos *switches*. Concebido para a topologia de um *datacenter*, estima o congestionamento através de um protocolo criado especificamente para comunicar o nível de utilização dos enlaces (*probes*). Essas *probes* são enviadas separadamente dos pacotes de dados. A informação é sumarizada e armazenada em cada *switch* que mantém estimativa de utilização de *link* por porta e usa *Flowlet* para efetuar balanceamento e mitigar o efeito de reordenamento.

MicroTE [Benson et al. 2011], em ambiente *datacenter*, age separando logicamente o tráfego previsível do imprevisível e emprega ECMP ponderado para esta última classificação a fim de mitigar o impacto de congestionamento. Utiliza OpenFlow e faz extensivo monitoramento para reagir em segundos, mas impondo baixa sobrecarga com mensagens de controle. O tráfego previsível é monitorado pelo controlador que o encaminha de forma otimizada, sendo o restante direcionado para rotas com pesos que representam a capacidade disponível. Para lidar com a escalabilidade utiliza heurísticas que efetuam a coleta de estatísticas de dados de monitoramento e buscam minimizar a re-computação de rotas.

SWAN [Hong et al. 2013] (*Software-driven WAN*) acomoda topologias inter-*datacenters* através de administração centralizada usando o OpenFlow. Baseado na demanda de serviço e na topologia de rede, decide quanto tráfego cada serviço pode enviar e procede com a configuração dos equipamentos. Aplica algoritmo para atualização consistente do planos de dados sem ocasionar congestionamentos temporários e reserva uma pequena largura de banda para esse objetivo. Também tem por objetivo utilizar de forma eficiente a memória limitada de *switches* mudando dinamicamente o conjunto de caminhos disponíveis conforme a demanda.

A proposta Presto [He et al. 2015], adaptado para ambiente de *datacenters*, é

implementado em *soft switches* e tem como foco o balanceamento de carga com gerenciamento centralizado e proativo. Emprega *weighted multipathing* nos *switches* das extremidades da conexão, mas sem modificar os *end-points*. Utiliza como unidade o *flowcell*, de 64 KBytes de tamanho, que escoam por caminhos pré-configurados para realizar *multipath routing*. Nos casos de reordenamento de pacotes, os *switches* atuam na correção de eventuais erros, falhas de transmissão ou efeitos gerados por *links* assimétricos.

LetItFlow [Vanini et al. 2017] emprega a forma mais simples do Flowlet fazendo balanceamento de carga. O processo ocorre naturalmente em topologias de árvores com múltiplas raízes. Cada grupo de *frames* de dados segue um caminho escolhido aleatoriamente, apostando na simplicidade mesmo com topologias assimétricas. LetItFlow apoia-se na habilidade do Flowlet para automaticamente definir o tamanho dos fragmentos com base nas condições detectadas de tráfego dos caminhos (via protocolo da camada de transporte), sem nenhum mecanismo explícito para detecção de congestionamento ou análise de retorno.

A proposta apresentada mantém semelhanças com B4, SWAN e HEDERA, uma vez que estas não têm o objetivo imediato de operar em ambiente *datacenter* e também precisa, em certa medida, da participação do controlador. No entanto, o mecanismo proposto é semelhante àquelas abordagens que apresentam alta responsividade no plano de dados, como CONGA e LetItFlow. A nossa estratégia prima pela simplicidade para melhor acomodar um fluxo de dados. CONGA, por exemplo, emprega uma técnica de detecção de congestionamento e aplica o algoritmo DRE para selecionar a porta que fornece o melhor equilíbrio.

Nossa abordagem é diferente na maneira como propõe a divisão de um fluxo, pois altera o parâmetro marcador usado para detectar novos Flowlets, que possibilita a alternância entre rotas, analisando o estado de congestionamento detectado na última rota usada. Acredita-se, assim, ser possível adicionar e remover rotas sem causar maiores problemas. A proposta mantém uma estimativa de congestionamento por destino e rota, ao contrário do HULA que busca o melhor próximo salto, e com aplicação em topologias de redes genéricas.

Este trabalho compartilha várias características com as propostas SDN apresentadas, mas é inovador no uso da programabilidade P4 do plano de dados, que possibilitou a utilização da técnica de balanceamento de carga ativo em cooperação com o plano de controle. Com uma abordagem que mantém a independência de funcionamento do plano de dados, espera-se que o mecanismo Flowlet atue com alta responsividade e limitada

visibilidade do estado geral dos *links* paralelos, enquanto ativamente detecta o estado do *link* utilizado, através da medição da latência. A presença do controlador atua para auxiliar de forma assíncrona e servir como uma interface na aplicação de estratégias que possuem conhecimento geral do estado dos enlaces.

# 4 ESTRATÉGIA HÍBRIDA DE ROTEAMENTO *MULTIPATH* INTEGRANDO OS PLANOS DE DADOS E DE CONTROLE

Nossa proposta aborda a perspectiva de roteamento *multipath* através de uma integração entre os planos de dados e de controle. Os *switches*, que estão situados no plano de dados, com sua capacidade de resposta oferecem maior velocidade de reação. Enquanto isso, as tarefas passivas, que se beneficiam de um conhecimento mais amplo da topologia e do estado geral da rede, são encaminhadas para o plano de controle. Portanto, a abordagem de roteamento *multipath* híbrida depende da operação conjunta de dois *control-loops* adaptativos: *(i)* um *control-loop on-line* para tomadas de decisões rápidas, executado nos *switches* e *(ii)* um *control-loop off-line* executado sob demanda pelo controlador para analisar a situação das rotas ativas e, se necessário, propor mudanças.

A estratégia de roteamento depende da programabilidade dos planos de dados e de controle usando respectivamente a linguagem P4 [Bosshart et al. 2014] e uma interface compatível para comunicação com os *switches*, semelhante à existente no OpenFlow [McKeown et al. 2008], que atualiza as regras e dados definidos na arquitetura P4. Essa separação entre os planos permite que os *switches* sejam auto-suficientes após a configuração inicial, ou na ocorrência de problemas de conexão com o controlador, que é responsável pela pré-computação e classificação de rotas alternativas na topologia e faz isso de forma assíncrona. A operação cooperativa consiste em atribuir tarefas que requerem alta capacidade de resposta, como decisões de encaminhamento e divisão de fluxos, ao plano de dados. Tarefas reativas, portanto que podem ser executadas de maneira assíncrona, como descoberta de rotas, são atribuídas ao plano de controle.

De forma simplificada, o controlador inicia sua operação e faz contato com todos os *switches* para realizar o mapeamento dos *links* e portas físicas envolvidas. Em seguida, no controlador, ocorre a descoberta de rotas, conforme será descrito na Seção 4.2.1. Então, o controlador insere um conjunto limitado de caminhos que permite o roteamento de subfluxos e a alternância entre rotas paralelas. Com a conclusão desta etapa, ocorre no plano de dados a operação ativa de divisão de fluxos, juntamente com o processo de medição dinâmica da latência. A operação de medição pode disparar um pedido de substituição de rota, mudando o conjunto de rotas ativas através da ação do controlador. Serão apresentados na Seção 4.1 as operações do plano de dados e na Seção 4.2 as estratégias que atuam para permitir a comunicação e sincronização de dados entre os planos.

## 4.1 Operações básicas do plano de dados

As atividades executadas no plano de dados, do ponto de vista da capacidade de resposta, são independentes do plano de controle e executam a divisão dos fluxos de tráfego de rede para obter o balanceamento de carga entre caminhos paralelos e assimétricos.

### 4.1.1 A abstração Flowlet

O princípio básico do Flowlet [Kandula et al. 2007] é a divisão de um fluxo de dados em segmentos, que são considerados grupos atômicos de pacotes. Um fluxo é definido pela tupla: IP de origem, IP de destino, versão de protocolo (4 ou 6), porta TCP de origem e porta TCP de destino. Cada subfluxo é criado pela detecção de uma lacuna de inatividade, o que permite o roteamento independente do conjunto de *frames*. O motivo dos intervalos de inatividade (falta de pacotes transmitidos por curtos períodos) é devido à dinâmica do protocolo TCP, do algoritmo de congestionamento usado, do *buffer* disponível a cada salto e também nos *endpoints*, bem como pelos estados de congestionamento agregados da rede.

A estratégia Flowlet pode operar com *links* paralelos e, conforme mostrado em [Vanini et al. 2017], a técnica de divisão de fluxo é eficaz mesmo quando são empregadas rotas assimétricas. Nossa abordagem utiliza essa abstração e adiciona um mecanismo simplificado para a análise do estado dos enlaces. Pretende-se através da constante medição do estado entre dois pontos da topologia conseguir calcular o menor intervalo que habilite alternância de rotas sem causar reordenamento de pacotes e atingir maior homogeneidade no uso de *links* paralelos.

### 4.1.2 Estimativa dinâmica da latência e operação de divisão sobre fluxo

A medição RTT (*Round Trip Time*) representa a medida de tempo calculada a partir de quando uma fonte envia um pacote (mensagem) e termina quando a confirmação do destinatário chega ao remetente. Esta informação é usada pela estratégia para determinar o momento de início e de término de um Flowlet. A correta determinação do intervalo permite a mudança de rota e, ao mesmo tempo, minimiza os efeitos relacionados ao reordenamento de pacotes.

A rota pela qual uma medição é realizada pode apresentar um caminho de ida e outro diferente de retorno. Para minimizar variações decorrentes da assimetria de rotas, na implementação experimental desenvolvida, o *frame* contendo a medição retorna pela mesma rota sem carga de dados. A metade da medida RTT é considerada como a medição e corresponde a uma aproximação da latência unidirecional.

A estimativa da medição detectada pela camada de transporte TCP, envolvida na conexão lógica fim-a-fim, é certamente a mais precisa. A técnica proposta para a estimativa de RTT calcula indiretamente esse valor, que pode sofrer algumas variações devido ao processamento que ocorre nas camadas superiores da pilha TCP/IP. Para fazer essa medição, mitigando oscilações provenientes de caminhos diferentes, é necessário que as rotas não sejam alteradas. Neste processo, os *switches* localizados próximo aos envolvidos na conexão fazem periodicamente a medição para cada caminho ativo. Um cabeçalho personalizado é usado para transportar o registro *timestamp* (Seção 4.1.4) e é incorporado em todos os pacotes, no entanto, a medição é realizada por amostragem. Para efeito de simplificação, a medida é realizada entre os *switches* de borda localizados próximo ao usuário, não sendo possível identificar qual fragmento de rota, compreendido entre dois *switches*, possui maior disponibilidade para o tráfego, tanto em volume (*throughput*) quanto em relação ao *delay* associado. Essas restrições podem englobar diversas situações como restrições de velocidade administrativa, por exemplo, QoS (*Quality of Service*).

A definição da palavra "latência" aplicada neste trabalho corresponde ao período de tempo, geralmente expresso em milissegundos, que um *frame* de dados leva para sair de um dispositivo de origem, ser processado por outro dispositivo de destino e retornar à origem. Esta definição também é aplicada de forma equivalente para a palavra *delay* no contexto deste texto.

O processo que envolve o cálculo da latência por RTT armazena em registradores o resultado do cálculo e tem como chave a rota e o destino (*switch*). Cada tupla mantém um valor correspondente ao intervalo de inatividade, que posteriormente é usado na detecção de um novo Flowlet. O valor é atualizado conforme os *frames* com dados são recebidos pelo *switch* origem.

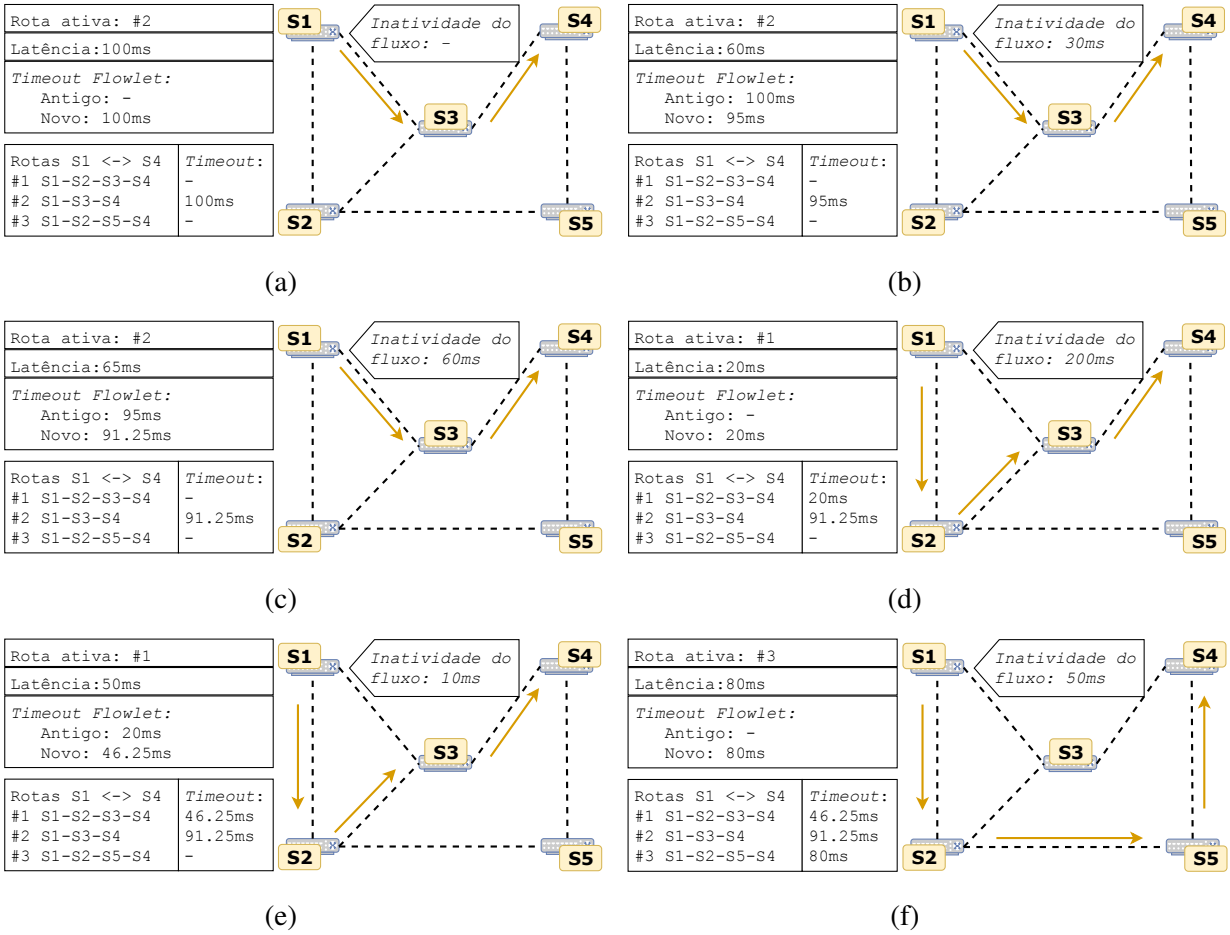
O esquema apresentado na Figura 4.1 representa de forma simplificada a detecção de um novo Flowlet. A Equação 4.1 apresenta a fórmula envolvida na definição do *timeout* necessário para o processo de alteração de rota. A condição geral associada ao parâmetro $\beta$ responde de maneira mais acentuada a medições ($\Delta$) superiores ao valor armazenado e,

caso contrário, decresce mais lentamente conforme a razão do parâmetro $\lambda$.

$$\beta = \begin{cases} \beta * \lambda + \Delta * (1 - \lambda), & \text{if } (\Delta > \beta) \\ \beta * (1 - \lambda) + \Delta * \lambda, & \text{senão} \end{cases} \tag{4.1}$$

onde: $\lambda$ : $\frac{1}{8}$, usado como fator de atenuação de $\beta$
$\Delta$ : última medição da latência
$\beta$ : valor usado para detectar novos Flowlets

Figura 4.1: Esquema ilustrativo da estimativa da latência




Fonte: O autor.

Na Figura 4.1a, tem início uma conexão que passa pelos nós de borda S1 e S4, a rota escolhida aleatoriamente é a #2 e a latência medida corresponde a 100ms. Em um segundo momento, representado pela Figura 4.1b, um período de inatividade de 30ms e, portanto, menor que o valor de *timeout* armazenado, habilita o uso da mesma rota para a comunicação. O valor que representa o intervalo necessário para criação de novos Flowlets é atualizado para 95ms após a medição de 60ms. Na Figura 4.1c, o intervalo de inatividade continua maior e é utilizada a rota #2. A latência medida de 65ms é aplicada

no cálculo que resulta na atualização do marcador relacionado à rota #2 para 91.25ms.

Na situação expressa na Figura 4.1d, a rota #1 passa a ser utilizada após um período superior de inatividade (200ms), quando comparado ao valor calculado correspondente a última rota utilizada (91.25ms). Essa mudança é resultado de um processo aleatório. Na Figura 4.1e, a rota #1 continua a ser utilizada e o valor da latência é atualizado, pois uma nova medição ocorreu (50ms). O valor aumentou, se comparado à última medição (20ms), atualizando o marcador para 46.25ms. Por fim, na Figura 4.1f, o intervalo de inatividade de 50ms foi superior ao valor de *timeout* armazenado para a última rota usada, que era de 46.25ms, e passou-se a utilizar uma rota aleatória.

### 4.1.3 Detecção do estado da conexão para substituição de rota

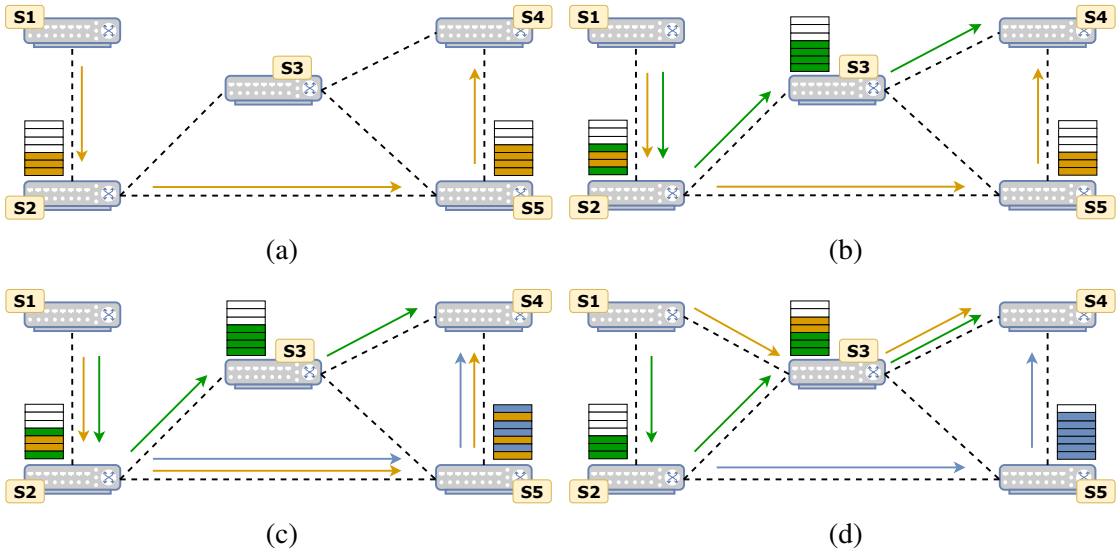
Depois de analisar o processo de alternância de rotas que acontece localmente nos *switches* e sobre um conjunto restrito de opções, será apresentado o processo relacionado a alterações sobre o conjunto de rotas ativas. Nesta seção será esquematizada a ideia relacionada à estratégia do mecanismo usado por um *switch* para solicitar alteração de rota.

Cada *switch* realiza a operação de balanceamento de carga sobre um conjunto limitado de rotas - de três, por exemplo - e mantém a estimava de latência de cada uma destas. Para este conjunto de rotas, denominadas ativas, outro registro mantém uma estimativa de média compartilhada. Essa média é associada ao destino e é usada no mecanismo de solicitação de substituição de rota. Um mecanismo acionador no dispositivo envia uma mensagem ao controlador sobre a situação do meio enviando também a latência medida quando determinada condição é atingida.

O esquema apresentado na Figura 4.2 busca abstrair a ideia envolvida com a definição da média compartilhada. Na Figura 4.2a, uma conexão estabelecida através de S1 até S4 (S1-S2-S5-S4) mantém uma estimativa regular de latência através da medida RTT. A ocupação dos *buffers* disponíveis nos *switches* tende a permanecer constante quando não existe concorrência. As cores e as pilhas representam os *frames* relacionados a cada conexão, em analogia aos espaços dos *buffers* ocupados.

Cada *switch* mantém uma média compartilhada entre os fluxos de mesmo destino (Figura 4.2b), que na ausência de congestionamento tende a apresentar pequenas oscilações. Quando ocorre um estado de congestionamento persistente devido à concorrência do *link* (Figura 4.2c), o tempo médio de permanência dos pacotes em trânsito aumenta,

Figura 4.2: Esquema ilustrativo da estratégia aplicado ao plano de dados para solicitação de nova rota.




Fonte: O autor.

resultando no acréscimo da medida RTT da rota S1-S2-S5-S4 (*frames* laranjas).

Quando a latência da rota analisada atinge um determinado valor limite, por exemplo, maior que 150% da média geral mantida por S1 com destino a S4, S1 envia ao controlador uma solicitação encapsulada em um cabeçalho inserido pelo P4 e contendo a informação dos *switches* envolvidos (origem e destino), o identificador da rota e o valor da latência atual, conforme:

```
header flow_t {
   bit<16> ether_type;
   bit<24> route_id;
   bit<9>  switch_id_1; // origem
   bit<9>  switch_id_2; // destino
   bit<32> latency;
   bit<6>  type; //informação ou requisição
}
```

Se existe a disponibilidade de nova rota para ativação no controlador e existindo a condição de alternância de rota, quando um novo Flowlet é detectado, o fluxo de S1 para S4 passa a ser encaminhado por um caminho que, possivelmente, tem maior chance de apresentar menor latência associada, por exemplo, S1-S3-S4 (Figura 4.2d).

Nos *switches*, a lógica do cálculo de manutenção da média compartilhada é ajustado para responder menos sensivelmente a oscilações bruscas e de curta duração e atuar de forma mais lenta se comparado à atualização do valor utilizado na detecção de novos Flowlets (mais detalhes serão apresentados no Capítulo 5). Desta forma, diversos parâmetros influenciam a dinâmica desses valores, especialmente a assimetria dos caminhos, o número de rotas concorrentes instanciadas nos *switches* e a quantidade de caminhos que

a topologia permite definir.

### 4.1.4 Organização lógica do plano de dados

No plano de dados cada *switch* pode representar logicamente uma função intermediária ou de extremidade. Os *switches* das extremidades estão conectados diretamente ao cliente e são responsáveis por monitorar a conexão, dividir e alternar rajadas de pacotes entre as rotas instaladas pelo controlador, além de inserir o cabeçalho Flowlet com o seguinte formato:

```
header flowlet_t {
   bit<16> flowlet_type; //tipo do próximo cabeçalho
   bit<2>  type;         //1: requisição, 2: resposta
   bit<9>  switch_id;    //identificador do dispositivo
   bit<13> id;           //identificador do flowlet
   bit<24> route_id;     //identificador da rota
   bit<32> flowlet_time; //timestamp
}
```

O cabeçalho Flowlet é inserido entre os cabeçalhos Ethernet e IP, conforme a Figura 4.3.

Figura 4.3: Localização do cabeçalho Flowlet.


Fonte: O autor.

Todas as decisões sobre os fluxos acontecem nos nós das extremidade, portanto os nós intermediários executam apenas as operações de encaminhamento entre porta de entrada e porta de saída de acordo com a leitura do identificado da rota (`route_id`). Esse identificador consta no cabeçalho e é utilizado como chave de pesquisa. Desta maneira, os comutadores intermediários não precisam executar nenhuma modificação de cabeçalho.

O controlador deve manter uma estrutura lógica de dados que armazene todas as interconexões entre os *switches*. Isso requer a troca de mensagens entre dispositivos e sua vizinhança, que são encaminhados ao controlador e permitem a descoberta de caminhos (Seção 4.2.1). Com a lista de caminhos determinada, o controlador rotula cada rota com uma etiqueta de identificação única e também define quais rotas serão inicialmente configuradas como ativas, ou seja, aquelas que participarão do processo de *multipath routing*. Em cada *switch*, para cada endereço IP/máscara de destino, um conjunto predefinido de rotas será usado no processo local de encaminhamento. Algumas rotas permanecerão de-

sativadas, de conhecimento do controlador e poderão ser substituídas nos *switches* pelo processo descrito na Seção 4.1.3.

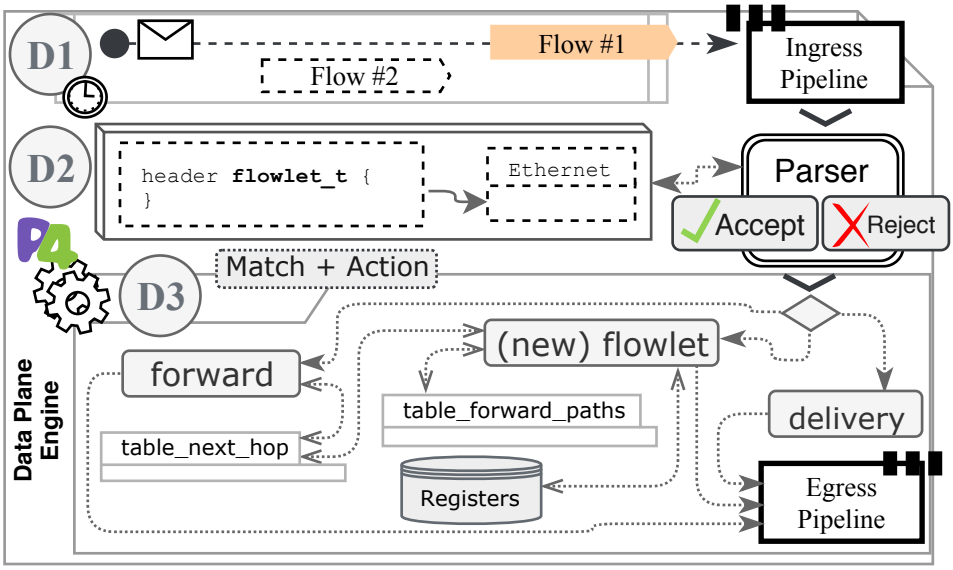
Para pré-configurar as rotas ativas a serem utilizadas, o controlador deve modificar os dados em duas tabelas dos *switches*, classificadas como estruturas P4 que mantém seu estado (*statefull*). A tabela `forward_paths` é indexada pelo IP/máscara do destinatário e armazena as rotas ativas. Por sua vez, a tabela `next_hops` é indexada pelo identificador de rota e retorna quais portas do *switch* estão envolvidas na rota. Para cada rota, o controlador (i) elenca todos os *switches* intermediários, (ii) preenche os dados sobre as informações de rota na tabela `next_hop` dos nós intermediários e, para as rotas que devem ser usadas para *multipath* (ativas), (iii) insere a informação na tabela `forward_paths`. Assim, cada *switch* em um caminho tem conhecimento da etiqueta de identificação de rota (`route_id`) presente no cabeçalho Flowlet e das portas de entrada e de saída (físicas ou lógicas) que servem bidirecionalmente ao tráfego de rede.

Como se percebe, há apenas um subconjunto de caminhos ativos para interconectar dois pontos da topologia. Cada *switch*, ao executar balanceamento de carga, seleciona um caminho paralelo e usa a técnica Flowlet para alternar entre as outras opções. Como será explicado na Seção 4.2, o papel do controlador é analisar os parâmetros recebidos para cada tupla <id_rota, switch_origem, switch_destino> e substituir, caso oportuno, alguma rota. O procedimento de substituição verifica na base de dados, que contém o histórico de medições, uma rota alternativa possivelmente com menor congestionamento. A estratégia usada para essa tarefa é personalizável para cada ambiente segundo as características desejadas, por exemplo: pode-se priorizar rotas com menor número de *hops*, com menor latência ou que utilizam um enlace específico.

Quando o controlador define as rotas e configura logicamente os caminhos, a abstração do Flowlet é usada como um mecanismo simples, eficiente e ativo para o balanceamento de carga. O processo de medição dinâmica via RTT é empregado sobre a rota ativa para constatar se um intervalo de inatividade é maior que o valor armazenado e, assim, acionar a utilização de um novo caminho. Portanto, não é necessário estimar a latência de todos os caminhos paralelos, o que permite adicionar e remover rotas dinamicamente. A medição entre os *switches* de borda foi escolhida para avaliar o estado geral de integridade da conexão devido à simplicidade e à vantagem de sumarizar problemas correlacionados ao desempenho. Certamente o uso combinado de outras métricas, como a carga cumulativa de utilização do enlace, pode agir de maneira complementar e aumentar a precisão, contudo adicionando alguma complexidade.

A Figura 4.4 mostra os principais componentes localizados no plano de dados. O rótulo D1 representa uma interface de rede que recebe vários fluxos de entrada, *frames Ethernet* sequenciados para processamento. Cada *frame* é então submetido a um analisador (D2) que extrai as informações contidas nos cabeçalhos. Os cabeçalhos envolvidos com o roteamento seguem para o próximo estágio (D3), resultando em uma ação que é executada com base nas informações já processadas, nas informações armazenadas nos registradores, nas tabelas e nos metadados. Os registradores mantém o estado entre pacotes, já as tabelas são estruturas que armazenam regras gerenciadas pelo controlador e os metadados carregam informações auxiliares que acompanham o *frame* durante o processamento. Para cada destino, um *switch* consulta o IP e conjunto com a máscara na tabela `forward_paths` usando a maior combinação possível (*Long Prefix Match*).

Figura 4.4: Componentes do plano de dados.




Fonte: O autor.

O estágio D3 (Figura 4.4), que segue a etapa *Parser*, constitui a principal estrutura de decisão sobre fluxos. Quando um pacote possui o cabeçalho Flowlet na sua estrutura, significa que ele está em trânsito (*forward*) ou seu destinatário está conectado com alguma porta do *switch*, portanto, acontece a entrega local (*delivery*). Quando um *frame* não contém o cabeçalho Flowlet (*(new) flowlet*) é necessário verificar se ele pertence a um novo grupo, consultar a rota pela qual será encaminhado e a porta correspondente de envio. Após a definição da porta pelo qual o *frame* será enviado, o fluxo de processamento segue para o bloco *Egress*. Esta estratégia aplica-se a um ambiente intra-domínio que apresenta características específicas, tais como o controle total sobre os nós envolvidos no roteamento e o suporte a topologias que apresentam conexões redundantes.

## 4.2 Cooperação entre os planos de dados e de controle

As técnicas aplicadas a *multipath routing*, dividindo os fluxos de dados, promovem o balanceamento do tráfego e as decisões envolvidas devem ser executadas de uma maneira a prevenir a formação de gargalos. Assim, ações menos sensíveis ao tempo, como mapear caminhos, configurar dispositivos e atualizar rotas, são executadas de forma assíncrona. As atividade reativas, desempenhadas pelo controlador, beneficiam-se da visão agrupada de todos os dispositivos envolvidos. As tarefas específicas que precisam de alta responsividade e que são relativamente independentes (ao menos quanto a alguns aspectos ou por um período de tempo) são delegadas ao plano de dados.

Os protocolos que descobrem rotas de maneira distribuída, como os protocolos de roteamento *link-state*, podem fornecer algum nível de redundância, mas com maior granularidade. Desta forma, delegar as atividades de definição de rotas ao controlador pode trazer o benefício de maior liberdade para poder aplicar diferentes estratégias. Esta abordagem, que é muito próxima do que o OpenFlow oferece, é aqui complementada por uma estrutura que acopla programação ao plano de dados: a arquitetura P4. Nesta abordagem, o plano de controle participa passivamente das tarefas de descoberta das rotas e da otimização na seleção de rotas ativas, procedimentos que demandaria muito mais complexidade se executados de forma distribuída.

Analogamente ao que um protocolo de roteamento distribuído inicialmente executa, uma das etapas para tornar a rede funcional é a descoberta de topologia. Esta atua de forma coordenada entre os planos através da troca de mensagens encapsuladas em um cabeçalho criado para esse fim:

```
header discovery_t {
   bit<14> id;
   bit<9>  port1;
   bit<9>  port2;
}
```

O controlador envia uma mensagem, através do canal de comunicação dedicado, e cada dispositivo reencaminha por meio de *broadcast* o pacote para todas as portas físicas. Ao receber esse pacote, o *switch* destinatário preenche o campo com a identificação da porta recebida, encaminhando em seguida a mensagem ao controlador. Este procede com a montagem lógica, refletindo as interligações físicas detectadas.

Com a centralização do processo, o controlador também participa do gerenciamento das traduções dos endereços físicos (camada de rede) para endereços da camada de inter-redes. Projeta-se que cada *switch* do domínio atue como *gateway* e seja respon-

sável pelas tarefas relacionadas ao protocolo ARP (*Address Resolution Protocol*) de seu ramo, que compreende os *hosts* conectados ao nó. Para tal, quando uma requisição ARP de um *gateway* é recebida, mas o endereço correspondente não se encontra nos registros locais, aciona-se o controlador para a tradução. O resultado fica armazenado no *switch* para evitar que sejam repetidas as mesmas solicitações.

O mapeamento dos dispositivos é organizado em uma estrutura de grafo, o que permite ao controlador simular caminhos sem envolver qualquer troca de mensagens entre os *switches*. Conforme será detalhado nas próximas seções, isso possibilita procurar rotas de forma centralizada e disponibilizá-las em um repositório. Para cada caminho ativado, o controlador configura todos os *switches* envolvidos com uma etiqueta identificadora de rota e também configura as portas pelas quais o tráfego deve ser roteado. Em seguida, o controlador informa às unidades quais rotas estão ativas e, após essa intervenção, os *switches* podem rotear o tráfego através de múltiplos caminhos e com divisão de subfluxos. Posteriormente, os ajustes nas rotas ativas podem acontecer de forma reativa, quando o controlador recebe uma solicitação enviada pelo *switch*, analisa as alternativas e, se apropriado, promove alguma substituição.

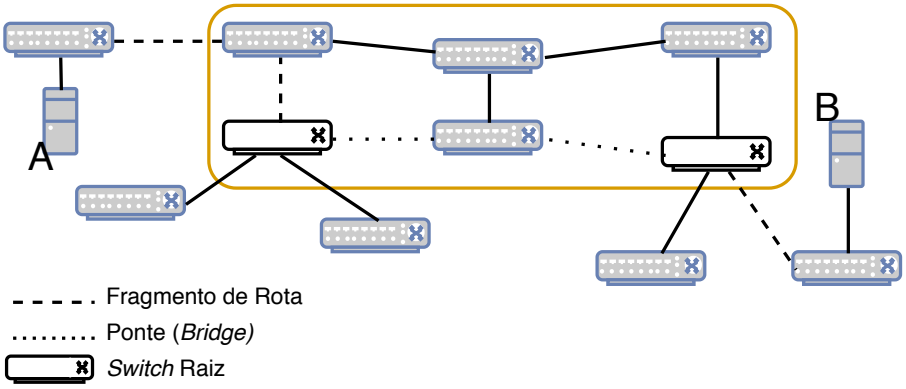
### 4.2.1 Descoberta de rotas

Como mencionado, a primeira tarefa executada pelo controlador é o mapeamento lógico das conexões entre os dispositivos de roteamento (*switches*), mapeando as portas que conectam os *links*. Prossegue-se então com a descoberta de novos caminhos, processo composto por cinco tarefas principais: (i) seleção das raízes da topologia, (ii) descoberta de rotas a partir dos vértices de menor grau até atingir uma ou mais raízes, (iii) combinação dos caminhos para criar um subgrafo que conecta as raízes, (iv) outras combinações para geração de rotas extras. Por fim, ocorre a designação de um identificador único para cada caminho criado.

As *Raízes* são nós agregados que concentram um valor acima da média de caminhos passantes. Cada caminho que começa a partir de outro *switch* e alcança uma raiz é entendido como um *fragmento de rota*. Um caminho que conecta duas raízes é denominado *ponte*. A ideia envolvida no desenvolvimento da sistemática é agregar um conjunto de rotas de forma mais flexível do que aquele hierárquico, utilizando porções que identificam um mínimo denominador comum de um conjunto de caminhos. A Figura 4.5 representa uma topologia simples a fim de demonstrar uma conexão estabelecida entre A

e B, utilizando dois fragmentos de rotas e uma ponte interligando os *switches* com função de raiz. As possibilidades de rotas que podem ser geradas entre os *switches* raízes representa uma porção do identificador da rota, enquanto que os dois fragmentos representam outra parte do identificador. Pela aplicação de uma máscara que considera apenas parte do identificador da rota, como será melhor explicado na Seção 4.2.2, é possível reduzir a quantidade de regras de encaminhamento consideradas pelos *switches* no processo de comutação.

Figura 4.5: Simulação do processo de descoberta de rotas.




Fonte: O autor.

O processo de adição ou de remoção de *switches* da topologia deve ser acompanhado por uma nova etapa de descoberta de rotas, mapeamentos de interconexão e aplicação de configurações pelo controlador nos *switches*. Uma possível estratégia é fazer o processo em dois estágios: configuração e ativação. Após a nova configuração implantada, com o auxílio de um marcador de tempo (*timestamp*) universal, pode-se de forma sincronizada proceder com a ativação do novo conjunto de configurações.

#### *4.2.1.1 Seleção de raízes*

O pseudo-código que faz a seleção das raízes é apresentado no Algoritmo 1. Existem dois estágios para selecionar os nós "raízes", que não podem ser adjacentes. Primeiramente, os vértices são classificados em ordem decrescente de grau e os que possuem um grau que satisfaz a condição indicada na linha 7 são incluídos como raízes. Com a relação de raízes parciais, é realizada uma pesquisa de amplitude baseada no algoritmo que será apresentado na Seção 4.2.1.2 (linha 9). Esse procedimento gera um conjunto de caminhos (fragmentos de rotas) que possuem apenas uma raiz cada. Nos caminhos descobertos, é contabilizada a ocorrência de cada vértice pertencente ao grafo (linhas 13-20). Em seguida, os vértices que apresentam contagem acima da média geral dos graus são

incluídos ao conjunto de raízes criado na primeira fase (linhas 21-25). Para cada nó, se a quantidade de ocorrências de caminhos passantes exceder a condição mostrada na linha 22, ele será selecionado como uma nova raiz, e então adicionado ao conjunto existente. Neste último passo, após considerar as raízes existentes, como restrição adicional e forma de aumentar a capilaridade dos fragmentos de rotas gerados é possível limitar o conjunto de nós com função de raiz que podem ser adicionados ao conjunto de raízes estabelecidas para não exceder, por exemplo, 25% do total.

**Algoritmo 1** Algoritmo para seleção de raízes.

**Require:** A connected graph $G(V, E)$
1: **procedure** GET_ROOTS
2: $roots \leftarrow \emptyset$
3: $N \leftarrow max\ V\ degree$
4: $M \leftarrow count\ V$ > $avg(degrees)$
5: $qtde \leftarrow M$ * ($N \div size$(G))
6: $D \leftarrow$ {v ∈ V}, sorted by $deg$(v) in descending order
7: $roots \leftarrow \{\{D_i\} \mid len(roots) < qtde;\ D_i \notin \forall(roots.adj)\}$
8: **for** each $\{i \in D;\ i \notin roots\}$ **do**
9: $path\ discovery$
10: **end for**
11: $count \leftarrow \emptyset$
12: $sum \leftarrow 0$
13: **for** each $\{i \in all\_paths\}$ **do**
14: **for** each $\{v \in V\}$ **do**
15: **if** $i \in v$ **then**
16: $count[i] + +$
17: **end if**
18: $sum \leftarrow sum + count[i]$
19: **end for**
20: **end for**
21: **for** each $\{i \in count \mid i \notin \forall(roots.adj)\ \}$ **do**
22: **if** $count[i] > (sum \div size(all\_paths))$ **then**
23: $roots \leftarrow roots \cup i$
24: **end if**
25: **end for**
26: **return** $roots$
27: **end procedure**

Depois de selecionadas as raízes, a próxima etapa é a descoberta de caminhos que têm como origem cada vértice que não é raiz e que terminam em um nó raiz. Nesse processo, cada vértice idealmente deve ter caminhos que alcancem pelo menos dois nós raízes diferentes. Assim, se uma raiz ficar indisponível, cada um dos nós não raiz ainda poderá rotear o tráfego de forma alternativa.

#### *4.2.1.2 Algoritmo para descoberta de rotas*

O Algoritmo 2 apresenta a lógica encarregada de criar fragmentos de rotas. A função (linha 3) recebe como parâmetro os seguintes elementos: um grafo não direcionado, o nó de origem, o nível que cresce conforme a função vai sendo recursivamente acionada, o nó – posição – atual, o caminho descoberto e as raízes atingidas pelos fragmentos que partiram do nó origem.

A função é acionada recursivamente e explora os caminhos até encontrar pelo menos 2 raízes (linha 26). Para cada nó adjacente (linhas 24-34), aqueles não visitados são adicionados em uma pilha (linha 31) e, caso o número mínimo de raízes não seja atingido, passam a ser explorados . Assim, o objetivo é partir das extremidades – dos nós de menor grau – e descobrir fragmentos de caminhos que levem a raízes. Para explorar rotas diferentes observa-se o segundo elemento de cada caminho (linha 14 e linha 26). Posteriores combinações usarão os fragmentos descobertos que não formam ciclos e buscam ser os caminhos mais curtos que conectam os vértices às raízes. As rotas entre raízes também são elencadas utilizando uma lógica semelhante.

#### *4.2.1.3 Expansão das raízes*

O procedimento de expansão das raízes combina pontes menores para que se formem pontes com mais de duas raízes. Desta maneira é possível definir rotas que interconectem nós distantes e, ao mesmo tempo, atribuir uma identidade através de um identificador de ponte.

#### *4.2.1.4 Combinação de fragmentos de rotas*

As demais combinações que utilizam fragmentos de rotas têm por objetivo criar caminhos que não envolvam nenhuma ponte e contenham, no máximo, um nó raiz envolvido. Os fragmentos tem início em nós não raízes, portanto, a criação de rotas transversais pode gerar caminhos menores entre pontos colaterais. Os principais tipos de combinação efetuadas são dois:

- Junção de dois fragmentos de rotas com um nó em comum e que não apresentem nenhum nó raiz envolvido;
- Junção de dois fragmentos com um nó raiz em comum.

Cada uma das combinações não pode apresentar elementos repetidos e é otimizada

**Algoritmo 2** Algoritmo para descoberta de caminhos.

**Require:** A connected graph $G(V, E)$

$roots \leftarrow \{v', v'', ...\}, v \in V$
$all\_paths \leftarrow \emptyset$
**procedure** DISCOVERY($graph, origin, level, node, visited, path, visited\_roots$) ▷ Graph is the $G(V, E)$
    $level \leftarrow level + 1$ ▷ Level is the deeph
    $next \leftarrow \emptyset$
    **if** $node \notin visited$ **then**
        $visited \leftarrow visited \cup node$
        **if** $length(path) > 1$ **then**
            **if** $node \in graph.adj(path.last)$ $and$ $node \notin path$ **then**
                $path \leftarrow path \cup node$
                **if** $node \in roots$ **then**
                $all\_path \leftarrow all\_path \cup path$
                $visited\_roots[origin] \leftarrow visited\_roots[origin] \cup node$
                $visited\_roots[origin][second] \leftarrow visited\_roots[origin][second] \cup node$
                **return**
                **end if**
            **else**
                **return**
            **end if**
        **else**
            $path \leftarrow path \cup node$
        **end if**
    **end if**
    **for all** $x \in graph.adj(node)$ **do**
        **if** $x \notin visited$ **then**
            **if** $(len(visited\_roots[x][second]) > 1)$ $and$ $(len(visited\_roots[x] > 2)$ **then**
                **if** $(graph.degree(x) < graph.avg)$ **then**
                $visited \leftarrow visited \cup node$
                **end if**
            **else**
                $next \leftarrow next \cup x$
            **end if**
        **end if**
    **end for**
    **if** $next = \emptyset$ **then**
        $all\_path \leftarrow all\_path \cup path$
    **end if**
**end procedure**

para selecionar, preferencialmente, os caminhos mais curtos.

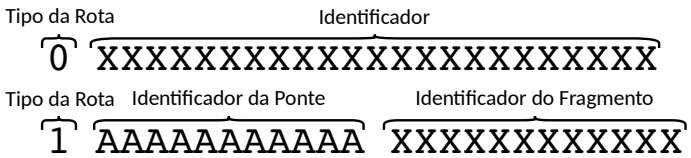
### 4.2.2 Tipos de rotas

Com uma lista selecionada de fragmentos de rota e de pontes, descobertos através do Algoritmo 2, ocorrem combinações ocorrem a fim de encontrar caminhos únicos e sem ciclos. As rotas chamadas *pontes* interconectam duas raízes e são usadas para agregar caminhos e reduzir o número de identificadores. Na identificação de um caminho, são previstos dois tipos de rotas para alcançar dois pontos na topologia: aquelas que não contém nenhuma ponte e aquelas que contém uma.

Aumentar o número de raízes na fase de descoberta aumenta o nível de fragmentação e, por conseguinte, pode resultar em maior quantidade de pontes necessárias para interligar todos os pontos da topologia. Isso pode colaborar na dinâmica da agregação de caminhos comuns.

O `route_id` usado no cabeçalho do Flowlet (Seção 4.1.4) é a etiqueta que identifica uma única rota, por isso denominada de Identificador Único do Caminho (UIP) (Figura 4.6). O UIP adotado é composto de 24 *bits*, sendo o primeiro *bit* mais significativo responsável por informar se a rota contém alguma ponte. Se o *bit* mais significativo for definido como 1, o UIP é formado por uma rota que combinada dois fragmentos e uma ponte para definir um caminho. A razão para usar "pontes" é otimizar o armazenamento de informações, já que os *switches* intermediários só precisam tomar decisões sobre uma parte do UIP, sem que seja necessário conhecer todas as combinações possíveis.

Figura 4.6: Formato do UIP em representação binária.




Fonte: O autor.

Em relação à limitação de memória presente nos *switches* para armazenar metadados de roteamento, o uso da máscara ternária (conceito utilizado pela UIP) contribui significativamente para atenuar a quantidade de regras necessárias A premissa de fixar caminhos na topologia garante que as rotas possam ser medidas sem variações. A agregação de rotas pode ser explorada pelo controlador de forma diferente, por exemplo: pode-se usar um UIP que contenha uma ponte para agregar dois ramos distintos, cada ramo com

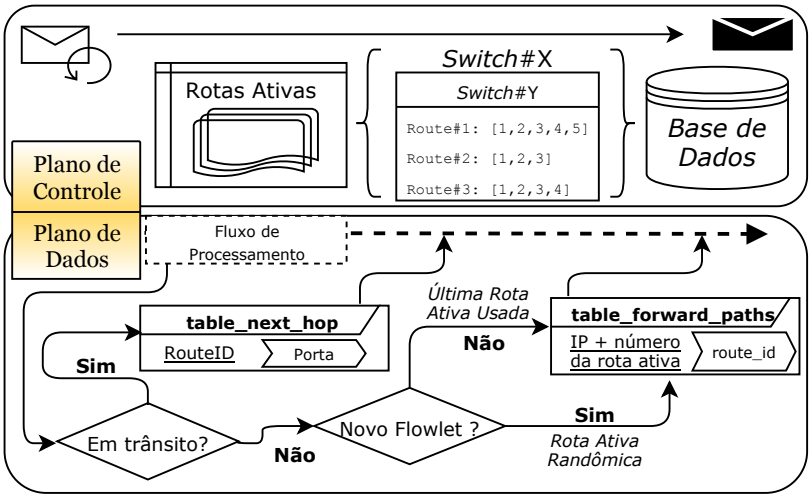
um identificador próprio. Além disso, o tamanho do campo `route_id` do cabeçalho também pode ser estendido para acomodar uma quantidade maior de dispositivos ou para empregar uma subdivisão diferente.

A organização hierárquica utilizada pelo protocolo IP continua a ser empregada para agrupar ramos da topologia. A introdução do conceito UIP é um complemento que busca tornar mais simples a fixação de rotas e permite utilizar indexação ternária com maior capacidade de fragmentação quando comparado à utilização de máscara de tamanho variável empregado pelo IP (VLSM - *Variable-Length Subnet Mask*).

### 4.2.3 Sincronização de dados entre os planos de dados e de controle

Para integrar as atividades de monitoramento executadas em conjunto com o plano de dados, o controlador instala rotas que permanecem sujeitas a atualizações. A Figura 4.7 apresenta uma visão geral e a localização lógica das principais estruturas envolvidas. No plano de dados, uma sequência de verificações decide se cada pacote (*frame)* está em trânsito, pertence a um Flowlet ou pode ser enviado por um caminho novo, armazenado localmente. As informações sobre os caminhos ativos são sincronizadas com a base de dados localizada no controlador, que armazena os valores históricos de medições de latência de cada caminho usado pelo *switch*. O UIP (ID de rota) é usado nesta troca de informações como chave de identificação.

Figura 4.7: Esquema da arquitetura abordando a integração dos planos de dados e de controle.




Fonte: O autor.

Uma rota é composta por um grupo de comutadores que lê o cabeçalho Flowlet

inserido, consulta a porta de saída em uma tabela interna e prossegue com a transferência, como visto na Seção 4.1.4. Portanto, uma rota ativa, partindo de um nó origem, pode atender a mais de um destino, pois cada *switch* envolvido no caminho verifica se o destinatário está conectado a uma porta local para então determinar a entrega. Dois *hosts* que desejam trocar informações podem usar uma rota mais ampla, o que reduz a quantidade de combinações geradas.

Um dispositivo de encaminhamento recebe um conjunto limitado de rotas e realiza a alternância entre elas conforme sejam detectados novos Flowlets. Cada *switch*, em sua operação normal, faz periodicamente medições de latência, operação que é controlada e disparada pelo plano de dados e usada para determinar o período que permite ao mecanismo Flowlet alternar entre rotas. Essa medição, conforme visto na Seção 4.1.3, também é usada para avaliar se uma substituição de rota é oportuna. O cálculo usado para decidir se uma nova rota deve ser instalada pelo controlador é realizado inteiramente no plano de dados, mediante a consideração dos parâmetros médios da rota analisada e das rotas paralelas (mais detalhes serão apresentados no Capítulo 5). Através desta ação, espera-se que ocorra a substituição das rotas de pior desempenho gradativamente para atingir a uniformidade entre rotas concorrentes, pois isso evita grandes oscilações que poderiam gerar falsas interpretações pelo algoritmo que coordena o controle de congestionamento da camada de transporte.

O controlador armazena os dados das últimas $n$ amostras na base de dados (Figura 4.7) para cada caminho ativo, com base no *switch* de origem e no *switch* de destino. Isso permite o uso de diferentes estratégias reativas para sugerir uma substituição de rota. Para alcançar equilíbrio entre caminhos paralelos, diferentes *políticas de atualização de rotas* podem ser implementadas com essas informações, dependendo do contexto.

Entre as rotas descobertas, aquelas que estão em desuso permanecem disponíveis em um banco de dados no controlador e, conforme seja necessário, são instaladas. Isso permite que um grupo de caminhos em uma topologia de múltiplas escolhas seja habilitado sob demanda para aliviar os estados de congestionamento detectados.

No Capítulo 5 serão avaliadas soluções que implementam as políticas de atualização de rotas com base na Média Móvel Exponencial (EMA - *Exponential Moving Average*) e na Média Móvel Ponderada (WMA - *Weighted Moving Average*) das medições de latência. No entanto, outras políticas de decisão podem ser implementadas para acomodar necessidades diferenciadas.

# 5 AVALIAÇÃO

Neste capítulo, serão apresentados os resultados das avaliações conduzidas para verificar o comportamento de algumas estratégias adaptativas propostas para este trabalho. A Seção 5.1 descreve as topologias e cargas de trabalho usadas. A Seção 5.2 apresenta alguns detalhes sobre a implementação e soluções de codificação adotadas. A Seção 5.3 discute as métricas consideradas na avaliação e a Seção 5.4 apresenta a análise dos resultados obtidos.

## 5.1 Topologia e carga de trabalho

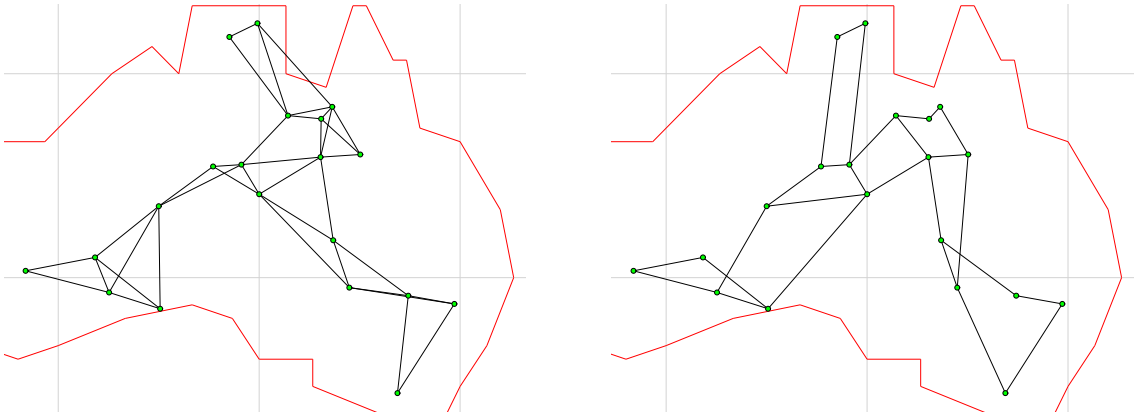
As topologias de rede adotadas em nossa avaliação foram geradas com base no estudo desenvolvido por [Kamiyama et al. 2010], que analisou o impacto que a formação topológica exerce em um cenário que envolve entrega de dados, mais especificamente a entrega de vídeo sob demanda. O estudo é feito a partir da análise de 23 Provedores de Serviços de Internet (ISPs) comerciais, sendo que uma classificação de topologia é definida com base nas características das conexões. Com base neste estudo, duas topologias foram selecionadas para serem avaliadas nos experimentos: *Hub and Spokes* (H&S) e *Ladder* (Figura 5.1).

A principal distinção entre essas formações topológicas é que *H&S* apresenta nós com função de agregação, enquanto *Ladder* não depende de nós concentradores e possui ciclos que são usados para interconectar pontos distantes. Para a topologia *Ladder*, o grau médio dos nós é menor que 3, enquanto para o *H&S* o parâmetro $R$ — definido como o número de nós com grau maior que o grau médio dividido pelo número total de nós — deve ser maior que 0,25. Para criar as topologias, foi utilizado o utilitário iGen [1]. Para cada classificação um arranjo foi gerado contendo 20 *switches* cada.

Cada *host* que é conectado a um *switch* funcionou como um cliente ou como um servidor. Nas análises foram fixados 10 elementos para desempenharem a função de cliente e o restante (outros 10) como servidor. Para cada transferência, uma nova conexão TCP é estabelecida e um número fixo de transferências em cada avaliação é executado com um intervalo de 2 segundos entre as conexões. Para a realização de balanceamento de carga através da divisão em Flowlet, um número máximo de 3 rotas paralelas para cada *switch* destinatário da topologia foi definido. A sequência de conexões e o tamanho

[1] http://igen.sourceforge.net/

Figura 5.1: Topologias avaliadas.



(a) H&S (b) Ladder

Fonte: O autor.

de arquivo transferido foi fixada para melhor avaliar o experimento, e cada cliente estabeleceu 100 requisições para somente um servidor. Desta forma, 10 pares de *switches* cliente-servidor foram criados e utilizados nas avaliações.

A carga de trabalho utilizada é inspirada na avaliação realizada em [Alizadeh et al. 2010] [Vanini et al. 2017]. A Tabela 5.1 apresenta os diferentes tamanhos de arquivos das transferências, realizadas aleatoriamente do lado do cliente e respeitando a distribuição apresentada.

Tabela 5.1: Carga de trabalho usada nos experimentos.

| **Tamanho** | **Distribuição** |
|---|---|
| 100K | 0.5 |
| 1M | 0.3 |
| 10M | 0.1 |
| 100M | 0.1 |

Nos experimentos, foi utilizado o emulador Mininet[2] para a definição da estrutura de rede e o controle de tráfego nativo foi desativado. A fila (*queue*) implementada nos *switches* e associada a cada interface virtual conectada aos *links* foi limitada para processar 2500pps (*packets per second*) e o seu tamanho ajustado para 250 pacotes. A política *default* de descarte de pacotes implementada pelos *soft switches* P4 não foi alterada.

No plano de dados o cálculo envolvido na detecção dos Flowlets corresponde a Equação 4.1. Para o ambiente experimental, as medições ocorrem com um intervalo mínimo de 500ms para cada rota e *switch*.

[2]mininet.org

O cálculo da média da latência, usado para informar ao controlador a situação detectada do *link*, ocorre juntamente com a medição Flowlet sobre as rotas paralelas e conforme a Equação 5.1. A média cresce e decresce de forma mais lenta e mantendo por mais tempo o histórico que é manipulado pelos fluxos ativos relacionados ao *switch* de destino. O parâmetro $\lambda$ permite fazer essa atenuação mantendo uma relação com $\Delta$, que corresponde a última medição da latência.

$$\gamma = (\gamma * (1 - \lambda)) + (\Delta * \lambda) \tag{5.1}$$

onde: $\lambda$ : $\frac{1}{16}$, usado como fator de atenuação
$\gamma$ : média da latência das rotas paralelas
$\Delta$ : última medição da latência

Para cada rota utilizada, um *switch* origem também mantém uma média relacionando a rota e ao *switch* de destino. A Equação 5.2 apresenta a definição da média relacionada ao caminho ativo que é utilizada na verificação para substituição de rota. O cálculo segue a mesma lógica da Equação 5.1 atribuindo menor peso a medições recentes.

$$\eta = (\eta * (1 - \lambda)) + (\Delta * \lambda) \tag{5.2}$$

onde: $\lambda$ : $\frac{1}{8}$, usado como fator de atenuação
$\eta$ : média da latência da rota
$\Delta$ : última medição da latência

A condição avaliada para disparar um alerta ao controlador solicitando alteração de rota é acionada conforme a Equação 5.3. O campo *type* do cabeçalho de requisição tem a função de informar o tipo do pacote, com valor 0 para notificar a latência quando $\eta$ for maior que a média $\gamma$ e o fator $\mu$, e o valor 1 para solicitar substituição de uma rota quando o valor ultrapassar o cálculo envolvendo o fator $\nu$. O objetivo de informar ao controlador o valor é popular os vetores com os dados que serão usados pelas estratégias de escolha de rota (políticas de atualização).

$$\begin{cases} type = 0, & \text{if } (\eta > (\gamma + \gamma * \mu)) \\ type = 1, & \text{if } (\eta > (\gamma + \gamma * \nu)) \end{cases} \tag{5.3}$$

onde: $\gamma$ : média da latência das rotas paralelas
$\eta$ : média da latência da rota
$\mu$ : $\frac{1}{4}$, notificação da latência
$\nu$ : $\frac{1}{2}$, solicitação de alteração

Na avaliação experimental conduzida, três diferentes *políticas de atualização* são aplicadas para executar a seleção de rotas: *(i)* aleatória, denominada NAIVE, *(ii)* Média Móvel Ponderada (WMA) e *(iii)* Média Móvel Exponencial (EMA).

A Equação 5.4 expressa uma estratégia na qual a média das medições de latência é calculada com base em pesos que diminuem para valores mais antigos. É assumido que o armazenamento de histórico está limitado às últimas 5 amostras para cada rota ativa.

$$y = \sum_{i=1}^{n} i; \quad WMA = \sum_{i=1}^{n} x_i \times \left(\frac{i}{y}\right) \tag{5.4}$$

onde: $n$ : 5

Por outro lado, a Equação 5.5 expressa uma estratégia na qual a média das medições de latência é calculada com base em uma Média Móvel Exponencial, considerando um parâmetro de suavização $\lambda$. Também é assumido que o armazenamento de histórico está limitado às últimas 5 amostras para cada rota ativa.

$$x_1 = \bar{x}; \quad EMA_i = (\lambda \times x_i) + (1 - \lambda) * EMA_{i-1} \tag{5.5}$$

onde: $\lambda$ : 0.30
$i$ : 2, . . . , 5

## 5.2 Implementação

A implementação do controlador foi realizada utilizando-se a linguagem de programação Python na versão 2.7. A comunicação do controlador com os *soft switches* foi mantida utilizando-se as bibliotecas desenvolvidas pelo projeto P4, que também são utilizadas pelo utilitário *simple_switch_CLI* e desenvolvidas usando a linguagem Python. A comunicação com os processos ocorreu através do protocolo de comunicação Thrift[3].

No plano de dados, a aplicação foi escrita utilizando-se a arquitetura *v1model*[4], provida pela equipe envolvida com o projeto P4. Somente os construtores nativos foram utilizados. Desenvolveu-se grande parte da lógica sobretudo através de registradores, que são estruturas que mantém o estado entre *frames* processados. A indexação nessas estruturas utiliza *hash*, usado também para identificar cada *frame* que pertence a um Flowlet pelo cálculo sobre os cabeçalhos IP e TCP. Os *hashes* também são empregados na escolha randômica de rotas, pois é possível definir um valor mínimo e máximo limitando o retorno, que torna-se aleatório. Em conjunto com as tabelas manipuladas pelo controlador, que contém as rotas ativas, e com o identificador da última rota usada, que é armazenado nos registros, é possível executar a lógica Flowlet verificando o último valor de *timeout* e, quando necessário, efetuar a alternância de rota.

Para detectar a latência entre dois *switches* mantendo a solicitação e a resposta trafegando pela mesma rota, a estratégia utilizada para atender um pedido de medição faz um clone do pacote, remove o *payload* e reencaminha-o pela mesmo caminho recebido. O custo envolvido é o da geração de um *frame* extra utilizado apenas para medir a latência. No entanto, se houvesse uma forma de sincronização do relógio interno dos diversos *switches*, seria possível devolver o valor por qualquer outra via e ainda atingir uma precisão maior pelo fato de conseguir medir somente o percurso de ida.

O compilador *p4c*[5] foi usado na geração do artefato executado com o utilitário *bmv2* (*Behavior Model*). Para a interação com o plano de controle, a primeira porta dos *switches* foi reservada e conectada logicamente ao controlador, permitindo a troca de *frames* diretamente.

[3]https://github.com/p4lang/thrift
[4]https://github.com/p4lang/p4c/blob/master/p4include/v1model.p4
[5]https://github.com/p4lang/p4c

## 5.3 Métricas

A métrica FCT (*Flow Completion Time*) [Dukkipati and McKeown 2006] foi utilizada na avaliação como principal parâmetro para compreender a estratégia de roteamento adaptável. A FCT está focada no desempenho que é percebido pelo usuário e indiretamente avalia o estado geral da conexão. A contabilização de Flowlets, resultado da divisão sobre fluxos, também é apresentada como forma de análise das possibilidades de utilização de rotas alternativas de acordo com a topologia verificada. A contagem dos pacotes que são explicitamente descartados é efetuada pelos *switches* virtuais P4, conforme apresentado na Seção 5.4.3. Além disso, a distribuição cumulativa de tráfego de dados para todas as interfaces dos *switches* é usada para verificar o uso de rotas na topologia. Os valores correspondentes aos dados transmitidos foram normalizados para identificar o comportamento entre os arranjos e os possíveis efeitos das políticas de atualização de rotas.

## 5.4 Resultados

As avaliações foram realizadas no mesmo ambiente, utilizando o sistema operacional Ubuntu Linux Xenial virtualizado com QEMU/KVM com 16 GBytes de memória RAM e acesso ao processador do *host*, CPU Intel (R) Xeon (R) E5-2620. O emulador Mininet versão 2.2 foi empregado para definir *hosts* virtuais e *switches*. A MTU (*Maximum Transmission Unit*) foi configurada para 1450 bytes nas interfaces de usuário a fim de evitar fragmentação pela inserção do cabeçalho Flowlet. A versão do simulador do bmv2 [6] 1.13 foi adotada para interpretar o código P4.

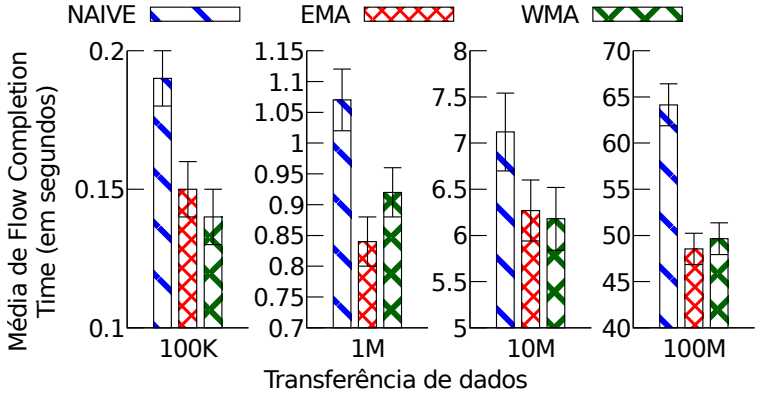
### 5.4.1 *Flow Completion Time*

A métrica *Flow Completion Time* analisa de forma indireta o estado geral da rede por meio da medição temporal das transferências, o que de fato é perceptível ao usuário em termos gerais. Quanto menor o valor (em segundos), maior é a fluidez dos enlaces. Essa métrica acaba por abstrair diversas situações como pacotes descartados, retransmitidos ou mesmo a situação de concorrência entre fluxos.

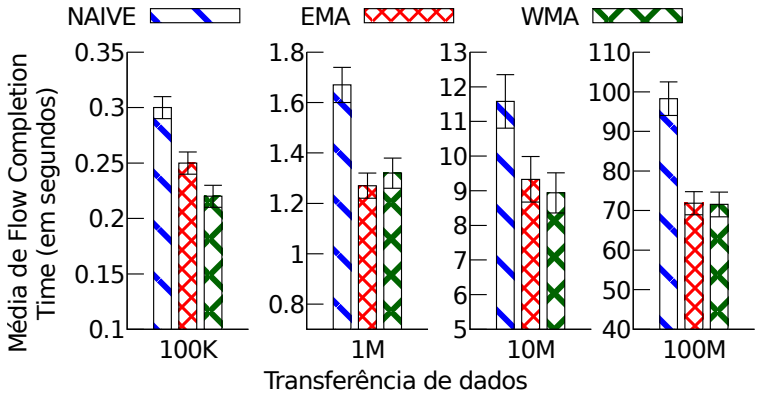
[6] https://github.com/p4lang/behavioral-model

Os resultados obtidos na avaliação empregando a métrica FCT (*Flow Completion Time*) (Figura 5.2) apresentam um comportamento favorável às estratégias EMA e WMA, especialmente para arquivos maiores (100M). O intervalo definido para as medições de latência e os limites que permitiram a intervenção do controlador provavelmente privilegiaram ações sobre transferências com maior duração.

Figura 5.2: Medições FCT da topologia.




(a) H&S




(b) Ladder

Fonte: O autor.

Ao observar o desempenho de EMA e WMA nos dois arranjos, não é possível definir uma das duas estratégias como melhor devido a sobreposição do intervalo de erro padrão. Em transferências menores de 10M, a diferença relativa à estratégia NAIVE não apresenta considerável vantagem.

Quanto às diferentes topologias, nota-se que Ladder impõe maiores tempos nas transferências pois apresenta menor quantidades de nós concentradores e, assim, menor proporção de conexões quando comparado à dinâmica da topologia *H&S* (Figura 5.1a).

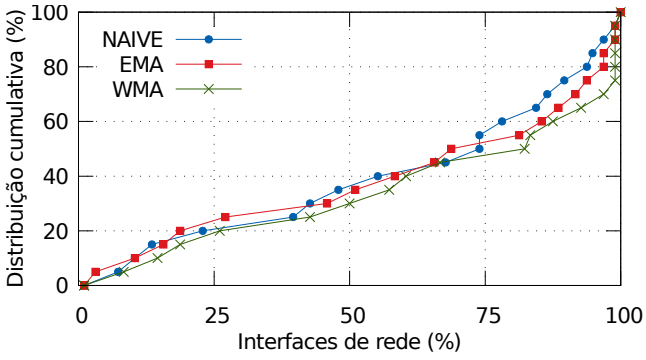
Os melhores resultados observados nas estratégias EMA e WMA devem-se também ao fato de que, com tais estratégias, priorizam-se os caminhos com menor quantidade de saltos (*hops*), o que naturalmente irá introduzir uma latência agregada menor. Já na

estratégia NAIVE, o caminho simplesmente é escolhido aleatoriamente, sem qualquer priorização ou verificação nesse sentido.

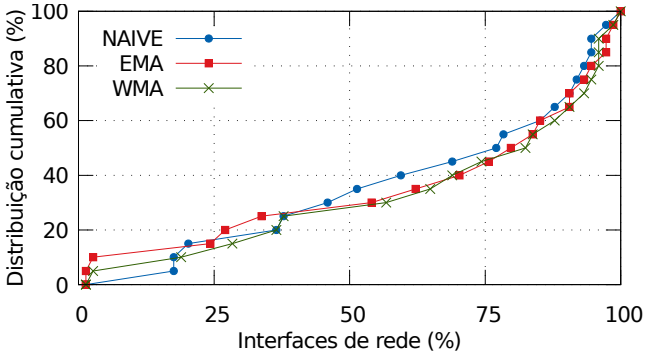
### 5.4.2 Distribuição de carga nos *links*

Em nossas avaliações, a soma da quantidade de dados transmitida por cada interface na topologia foi utilizada para analisar o comportamento dinâmico dos fluxos e está intimamente relacionada com o arranjo e a quantidade de enlaces. *H&S*, por exemplo, comparado a *Ladder*, tem maior potencial de combinação de caminhos, mas a desvantagem de que algumas rotas podem sobrecarregar-se, devido ao emprego de enlaces que conectam nós com função de agregação. *Ladder* possibilita menor quantidade de combinações de rotas, mas maior concorrência na utilização dos *links* para interconectar nós distantes.

Figura 5.3: Distribuição cumulativa das transferências de dados (normalizada %).




(a) H&S




(b) Ladder

Fonte: O autor.

De forma geral, ambas as topologias apresentam uma distribuição com cerca de 25% das interfaces envolvidas em 60% do volume transmitido. A Figura 5.3 apresenta os dados obtidos após normalização. A distribuição cumulativa referente à topologia *H&S*

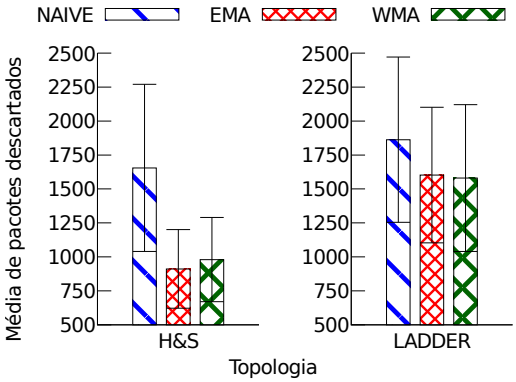
(Figura 5.3a) apresentou um uso levemente diferenciado dos enlaces pelas estratégias, especialmente naquelas interfaces envolvidas com o maior tráfego observado.

Em *Ladder* (Figura 5.3b) nota-se oscilações da estratégia NAIVE com maior distribuição no uso das interfaces. Este efeito possivelmente ocorreu devido às escolhas aleatória de rotas maiores.

### 5.4.3 Pacotes descartados

A quantidade de pacotes descartados pelos *switches*, excedentes somente na fila das interfaces envolvidas nos nós intermediários de uma conexão (Seção 4.1.4), é apresentada pela média entre todos os dispositivos da topologia (Figura 5.4). A contabilização foi realizada com os pacotes explicitamente descartados pelos *switches* P4, que implementam uma única fila entre os blocos de entrada e de saída. Pela diferença entre os pacotes recepcionados e os pacotes enviados, foi possível visualizar a diferença de *frames* excedentes descartados.

Figura 5.4: Descarte de pacotes contabilizado pelo *switches* P4.



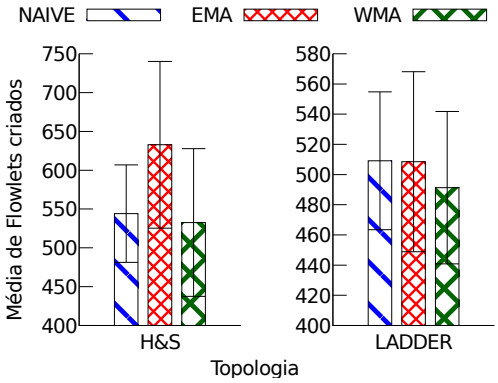

Fonte: O autor.

Buscou-se averiguar, de forma global, se alguma das estratégias apresentou comportamento díspar que acabou por interferir no controle de congestionamento exercido pelo protocolo TCP. Nesse sentido, a estratégia NAIVE apresentou o perfil com maior quantidade de pacotes descartados que se encontravam em trânsito. Em *H&S*, EMA e WMA apresentaram os menores valores, porém, pela sobreposição do erro padrão associado, não é possível definir se efetivamente EMA ou WMA apresentaram benefícios relacionados a este aspecto.

### 5.4.4 Flowlets criados

Esta avaliação contempla a quantificação de Flowlets criados durante o experimento. A detecção de um novo Flowlet depende de um intervalo de tempo mínimo, considerado um período de inatividade. Esse período ocorre naturalmente devido à dinâmica de atuação do protocolo TCP e, mais especificamente, pela interpretação da situação atribuída ao algoritmo de controle de congestionamento. A Figura 5.5 apresenta a média correspondente à contabilização de Flowlets efetuada somente nos *switchs* conectados a um *host* que atuou como servidor. A contabilização inclui as alterações de rotas detectadas por *timeout* de inatividade e as impostas pelo controlador, situação em que uma substituição de rota é atendida.

Figura 5.5: Contabilização de Flowlets criados pelos *switches* P4.




Fonte: O autor.

A topologia *H&S* apresentou maior disparidade entre as estratégias, com maior quantidade de Flowlets criados para EMA. Com a interposição dos intervalos de erro padrão, não é seguro concluir as causas envolvidas, mas parece razoável inferir que nas situações que a medição dinâmica da latência apresenta menor variação, o mecanismo de divisão de fluxo consegue proporcionar mais oportunidades de segmentação. Nos experimentos, mesmo *H&S* apresentando um valor geral menor na principal métrica avaliada, *Flow Completion Time*, a diferença média na criação de Flowlets entre as topologias não apresentou significativa diferença.

## 5.5 Considerações e limitações

Nos experimentos foi utilizado como simulador para o protótipo P4 o utilitário *bmv* (*Behavioral Model*) que tem primariamente a função de emular o comportamento de

um *switch* processador de pacotes, sem compromisso em ser uma ferramenta que consiga reproduzir o comportamento de um ambiente real de rede. O emulador Mininet, utilizado em conjunto com o utilitário *bmv* (versão 2) para promover os enlaces necessários, também menciona[7] que certa instabilidade é adicionada ao ambiente quando pacotes são trocados entre o espaço do *kernel* e o do usuário, inclusive sofrendo variações decorrentes do escalonamento de processo do Sistema Operacional. Portanto, apesar de os resultados obtidos demonstrarem alguns benefícios que o controlador adiciona ao ambiente, em uma topologia que emprega equipamentos físicos especializados as latências associadas a cada salto (*hop*) certamente seriam muito mais estáveis.

Referente ao mecanismo empregado na medição dinâmica da latência, a arquitetura P4 não provê uma forma de sincronizar o relógio interno (virtual) de cada *switch*, por exemplo, através de *epoch Unix* via protocolo NTP (*Network Time Protocol*). Com um recurso de sincronização incorporado à plataforma, seria possível medir com maior precisão e de forma direcional a latência detectada. As equações associadas ao monitoramento da latência, apresentadas na Seção 5, ilustram o princípio proposto da medição dinâmica, podendo sempre ser ajustadas em vista do aumento da acurácia em ambientes reais.

Outro fator que pode interferir no funcionamento normal do processo de estimativa da latência é a perda constante de *frames*, fato gerado pelo descarte ou por falhas na comunicação física que causam interferências nas transferências. A ideia envolvida no processo de monitoramento não abrange necessariamente a verificação da integridade dos enlaces, mas poderá, em trabalhos futuros, incluir a comunicação ao controlador de métrica relacionada ao monitoramento de pacotes descartados.

[7] http://mininet.org/walkthrough/#other-switch-types

# 6 CONCLUSÃO

No contexto de evolução das tecnologias atualmente empregadas tanto na administração quanto nos equipamentos e enlaces físicos, a programabilidade do plano de dados, especialmente habilitada pela arquitetura P4, deverá proporcionar a implementação de novos e aprimorados serviços de Internet. Assim, acredita-se que as atuais propostas – que aplicam a segregação da administração dos planos de dados e de controle, facilitadas pelo protocolo OpenFlow – precisam evoluir ainda mais para atender aos requisitos de tarefas que exigem alta capacidade de resposta, como no caso do balanceamento de carga. Tais tarefas são executadas com maior agilidade quando realizadas nos equipamentos que processam o fluxo de dados (*switches* e roteadores).

## 6.1 Contribuições

Este trabalho explora o uso de programabilidade nos planos de dados e de controle para oferecer uma estratégia híbrida que interconecte ambas camadas e suporte estratégias que promovam a utilização de múltiplos caminhos (*multipath routing*) e o balanceamento de carga (*load balancing*).

Esta proposta alcança a cooperação entre os planos de dados e de controle propondo a disposição lógica de componentes que interagem para fornecer algo mais flexível que o atual cenário em SDN. O plano de dados, através da programabilidade da arquitetura P4, comporta atividades que acompanham o fluxo de dados, classificadas como tarefas que necessitam de alta responsividade. Tais tarefas incluem a medição do estado do *link*, a divisão de fluxos pela técnica Flowlet e o respectivo roteamento por múltiplos caminhos. No plano de controle, diversas atividades desempenham tarefas reativas através da visão construída pelo controlador. Essas atividades incluem o mapeamento de interconexões e a descoberta de rotas através de algoritmos desenvolvidos para formar caminhos únicos e com a maior quantidade de nós envolvidos. A dinâmica iterativa entre os planos permite a substituição de rotas conforme políticas executadas pelo controlador, a partir dos dados coletados no plano de dados.

Os resultados obtidos nas avaliações apontam para alguns benefícios que uma estrutura híbrida pode proporcionar. A arquitetura/linguagem P4 ainda está amadurecendo e, mesmo em ambientes que empregam avaliação por simulação, pode-se verificar que as ações indiretas do controlador conseguem ser aplicadas como instrumento de otimização.

As possíveis melhorias devem-se à visão amplificada que o controlador adiciona ao ambiente através do acoplamento de diferentes políticas de atualização de rotas. No entanto, a dependência do controlador pode introduzir fragilidades ao ambientes, situação parcialmente mitigada ao garantir o funcionamento autônomo dos *switches* em tarefas triviais de encaminhamento.

Observa-se que a aplicação de estratégias que exploram as características de múltiplos caminhos encontra afinidade quando posicionada junto ao plano de dados. Contudo, muitos desafios ainda permanecem devido à falta de suporte que os equipamentos tradicionais oferecem. Com a proposta de trazer programabilidade para o plano de dados, muitas tarefas, hoje executadas por protocolos tradicionais de forma distribuída, podem ser engajadas para atuarem de forma coordenada e atingirem resultados diferenciados sob o comando de um agente controlador. Nesse estudo busca-se explorar algumas ideias, porém muito esforço também deve ser empregado na direção de compatibilizar os dispositivos legados ou que não apresentam nenhum tipo de disponibilidade para programabilidade. Considerando-se o protocolo OpenFlow, é possível inferir que a convivência com o novo paradigma proposto pela arquitetura P4 tem maior potencial e apresenta novas possibilidades de interação e criação de infraestruturas híbridas com a presença de administração centralizada. Espera-se, também, que essas inovações tornem-se incentivo para que provedores de serviços utilizem-se de plataformas diferenciadas para promover serviços com mais qualidade, rapidez e dinamicidade e, assim, atender às demandas emergentes relacionadas aos recursos de telecomunicação.

A questão que envolve o desempenho observado, especialmente na ferramenta de emulação *Behaviour Model*, certamente está distante do que se espera verificar em um *hardware* compatível com a arquitetura P4. Assim, acredita-se que diante dos desafios que surgiram na avaliação, os aspectos quantitativos necessários para comparar o desempenho com outras propostas somente conseguem ser validados através de avaliações realizadas em *hardware*.

## 6.2 Trabalhos futuros

A solução proposta visa à simplicidade na implementação da estratégia, porém algumas condições são limitantes para o adequado funcionamento, como é o caso da execução em uma topologia que apresenta excessiva perda de pacotes. Uma possível otimização, usando recursos de programabilidade no plano de dados, é analisar o com-

portamento das solicitação SACK (*Selective Acknowledgment*) do protocolo TCP. Essa informação é carregada no campo *options* do cabeçalho TCP e tem a função de solicitar explicitamente fragmentos que não foram recebidos ou apresentam um atraso maior que o esperado. Conforme é indicado pelo controle do protocolo da camada de transporte, trata-se de um indício de pacote descartado ou condição de *link* com problema físico de conectividade. Através da análise dos cabeçalhos torna-se possível identificar e comunicar ao controlador a situação. A quantidade de pacotes descartados devido à restrição do *buffer* das interfaces dos *switches* P4, que é de conhecimento no plano de dados, também pode ser uma informação útil para ser repassada ao plano de controle.

A detecção do estado dos *links*, neste trabalho, ocorre entre os *switches* que estão mais próximos dos usuários, pois acredita-se que, na utilização intra-domínio, é possível conseguir bons resultados com este nível de granularidade. Porém, acrescentando-se alguma complexidade de gerenciamento, essa informação pode ser segmentada a cada *hop*. Com o auxílio do controlador e utilizando medição a cada salto, uma maior granularidade permitiria identificar gargalos com a precisão necessária para ações pontuais de priorização de tráfego. Contudo, as políticas de priorização de rotas se tornariam mais elaboradas e um algoritmo para análise do fluxo, como Ford-Fulkerson, seria mais indicado para resolver o problema.

Nos testes realizados, o tamanho da fila relacionada a cada interface permaneceu constante e nenhuma técnica de priorização de pacotes foi aplicada. Com o avanço da arquitetura P4, parece ser possível priorizar fluxos e alterar dinamicamente o tamanho e a política de priorização de filas e de pacotes. Em um ambiente controlado (intra-domínio), *buffers* muito grandes auxiliam na transferência e evitam perdas de pacotes devido à flutuação do tráfego, porém adicionam demasiada latência nas conexões sensíveis à percepção de resposta, como no caso do tráfego VoIP (*Voice over Internet Protocol*).

Sobre os mecanismos situados no plano de controle, apresentados na Seção 4.2, os algoritmos envolvidos no processo de descoberta de rotas foram definidos sobretudo para apresentar uma noção inicial que suportasse o particionamento da topologia. Certamente esses códigos podem ser otimizados a fim de reduzir a complexidade envolvida na pesquisa de rotas, no entanto, mantendo o princípio de endereçamento apresentado na Seção 4.2.2.

# REFERÊNCIAS

AL-FARES, M. et al. Hedera: Dynamic flow scheduling for data center networks. In: **Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation**. Berkeley, CA, USA: USENIX Association, 2010. (NSDI'10), p. 19–19. Available from Internet: <http://dl.acm.org/citation.cfm?id=1855711.1855730>.

ALIZADEH, M. et al. Conga: Distributed congestion-aware load balancing for datacenters. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 44, n. 4, p. 503–514, aug. 2014. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/2740070.2626316>.

ALIZADEH, M. et al. Conga: Distributed congestion-aware load balancing for datacenters. In: **Proceedings of the 2014 ACM Conference on SIGCOMM**. New York, NY, USA: ACM, 2014. (SIGCOMM '14), p. 503–514. ISBN 978-1-4503-2836-4. Available from Internet: <http://doi.acm.org/10.1145/2619239.2626316>.

ALIZADEH, M. et al. Data center tcp (dctcp). In: **Proceedings of the ACM SIGCOMM 2010 Conference**. New York, NY, USA: ACM, 2010. (SIGCOMM '10), p. 63–74. ISBN 978-1-4503-0201-2. Available from Internet: <http://doi.acm.org/10.1145/1851182.1851192>.

BENSON, T. et al. Microte: Fine grained traffic engineering for data centers. In: **Proceedings of the Seventh Conference on Emerging Networking Experiments and Technologies**. New York, NY, USA: ACM, 2011. (CoNEXT '11), p. 8:1–8:12. ISBN 978-1-4503-1041-3. Available from Internet: <http://doi.acm.org/10.1145/2079296.2079304>.

BOSSHART, P. et al. P4: Programming protocol-independent packet processors. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 44, n. 3, p. 87–95, jul. 2014. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/2656877.2656890>.

CALVERT, K. L. et al. Directions in active networks. **IEEE Communications Magazine**, v. 36, n. 10, p. 72–78, Oct 1998. ISSN 0163-6804.

DOMżAŁ, J. et al. A survey on methods to provide multipath transmission in wired packet networks. **Computer Networks**, v. 77, p. 18 – 41, 2015. ISSN 1389-1286. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S1389128614004435>.

DUKKIPATI, N.; MCKEOWN, N. Why flow-completion time is the right metric for congestion control. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 36, n. 1, p. 59–62, jan. 2006. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/1111322.1111336>.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The road to sdn: An intellectual history of programmable networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 44, n. 2, p. 87–98, abr. 2014. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/2602204.2602219>.

FERLIN, S. et al. Revisiting congestion control for multipath tcp with shared bottleneck detection. In: **IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications**. [S.l.: s.n.], 2016. p. 1–9.

FORD, A. et al. **TCP Extensions for Multipath Operation with Multiple Addresses**. RFC Editor, 2013. RFC 6824. (Request for Comments, 6824). Available from Internet: <https://rfc-editor.org/rfc/rfc6824.txt>.

HE, J.; REXFORD, J. Toward internet-wide multipath routing. **IEEE Network**, v. 22, n. 2, p. 16–21, March 2008. ISSN 0890-8044.

HE, K. et al. Presto: Edge-based load balancing for fast datacenter networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 45, n. 4, p. 465–478, aug. 2015. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/2829988.2787507>.

HONG, C.-Y. et al. Achieving high utilization with software-driven wan. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 43, n. 4, p. 15–26, aug. 2013. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/2534169.2486012>.

HOPPS, C. **Analysis of an Equal-Cost Multi-Path Algorithm**. United States: RFC Editor, 2000.

IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks–Amendment 20: Shortest Path Bridging. **IEEE Std 802.1aq-2012 (Amendment to IEEE Std 802.1Q-2011 as amended by IEEE Std 802.1Qbe-2011, IEEE Std 802.1Qbc-2011, IEEE Std 802.1Qbb-2011, IEEE Std 802.1Qaz-2011, and IEEE Std 802.1Qbf-2011)**, p. 1–340, June 2012.

IRTEZA, S. M. et al. Load balancing over symmetric virtual topologies. In: **IEEE INFOCOM 2017 - IEEE Conference on Computer Communications**. [S.l.: s.n.], 2017. p. 1–9.

JAIN, S. et al. B4: Experience with a globally-deployed software defined wan. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 43, n. 4, p. 3–14, aug. 2013. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/2534169.2486019>.

KAMIYAMA, N. et al. Impact of topology on parallel video streaming. In: **2010 IEEE Network Operations and Management Symposium - NOMS 2010**. [S.l.: s.n.], 2010. p. 607–614. ISSN 1542-1201.

KANDULA, S. et al. Dynamic load balancing without packet reordering. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 37, n. 2, p. 51–62, mar. 2007. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/1232919.1232925>.

KATTA, N. et al. Hula: Scalable load balancing using programmable data planes. In: **Proceedings of the Symposium on SDN Research**. New York, NY, USA: ACM, 2016. (SOSR '16), p. 10:1–10:12. ISBN 978-1-4503-4211-7. Available from Internet: <http://doi.acm.org/10.1145/2890955.2890968>.

KHOSRAVI, H. M. et al. **Linux Netlink as an IP Services Protocol**. RFC Editor, 2003. RFC 3549. (Request for Comments, 3549). Available from Internet: <https://rfc-editor.org/rfc/rfc3549.txt>.

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. **Proceedings of the IEEE**, v. 103, n. 1, p. 14–76, Jan 2015. ISSN 0018-9219.

LI, M. et al. Multipath transmission for the internet: A survey. **IEEE Communications Surveys Tutorials**, v. 18, n. 4, p. 2887–2925, Fourthquarter 2016. ISSN 1553-877X.

MANNIE, E. **Generalized Multi-Protocol Label Switching (GMPLS) Architecture**. RFC Editor, 2004. RFC 3945. (Request for Comments, 3945). Available from Internet: <https://rfc-editor.org/rfc/rfc3945.txt>.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008. ISSN 0146-4833. Available from Internet: <http://doi.acm.org/10.1145/1355734.1355746>.

MIJUMBI, R. et al. Network function virtualization: State-of-the-art and research challenges. **IEEE Communications Surveys Tutorials**, v. 18, n. 1, p. 236–262, Firstquarter 2016. ISSN 1553-877X.

PERLMAN, R. et al. **Routing Bridges (RBridges): Base Protocol Specification**. [S.l.], 2011. Available from Internet: <https://doi.org/10.17487/rfc6325>.

PRABHAVAT, S. et al. On load distribution over multipath networks. **IEEE Communications Surveys Tutorials**, v. 14, n. 3, p. 662–680, Third 2012. ISSN 1553-877X.

QADIR, J. et al. Exploiting the power of multiplicity: A holistic survey of network-layer multipath. **IEEE Communications Surveys Tutorials**, v. 17, n. 4, p. 2176–2213, Fourthquarter 2015. ISSN 1553-877X.

SIVARAMAN, A. et al. Packet transactions: High-level programming for line-rate switches. In: **Proceedings of the 2016 ACM SIGCOMM Conference**. New York, NY, USA: ACM, 2016. (SIGCOMM '16), p. 15–28. ISBN 978-1-4503-4193-6. Available from Internet: <http://doi.acm.org/10.1145/2934872.2934900>.

TENNENHOUSE, D. L. et al. A survey of active network research. **IEEE Communications Magazine**, v. 35, n. 1, p. 80–86, Jan 1997. ISSN 0163-6804.

VANINI, E. et al. Let it flow: Resilient asymmetric load balancing with flowlet switching. In: **14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)**. Boston, MA: USENIX Association, 2017. p. 407–420. ISBN 978-1-931971-37-9. Available from Internet: <https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/vanini>.

WÓJCIK, R. et al. A survey on methods to provide interdomain multipath transmissions. **Computer Networks**, v. 108, p. 233 – 259, 2016. ISSN 1389-1286. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S138912861630281X>.

XU, C.; ZHAO, J.; MUNTEAN, G. M. Congestion control design for multipath transport protocols: A survey. **IEEE Communications Surveys Tutorials**, v. 18, n. 4, p. 2948–2969, Fourthquarter 2016. ISSN 1553-877X.

XU, W.; REXFORD, J. Miro: Multi-path interdomain routing. In: **Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications**. New York, NY, USA: ACM, 2006. (SIGCOMM '06), p. 171–182. ISBN 1-59593-308-5. Available from Internet: <http://doi.acm.org/10.1145/1159913.1159934>.

YANG, L. et al. **Forwarding and Control Element Separation (ForCES) Framework**. RFC Editor, 2004. RFC 3746. (Request for Comments, 3746). Available from Internet: <https://rfc-editor.org/rfc/rfc3746.txt>.

ZHANG, Y. et al. A survey on software defined networking with multiple controllers. **J. Netw. Comput. Appl.**, Academic Press Ltd., London, UK, UK, v. 103, n. C, p. 101–118, feb. 2018. ISSN 1084-8045. Available from Internet: <https://doi.org/10.1016/j.jnca.2017.11.015>.

ZHOU, J. et al. Wcmp: Weighted cost multipathing for improved fairness in data centers. In: **Proceedings of the Ninth European Conference on Computer Systems**. New York, NY, USA: ACM, 2014. (EuroSys '14), p. 5:1–5:14. ISBN 978-1-4503-2704-6. Available from Internet: <http://doi.acm.org/10.1145/2592798.2592803>.

# ANEXO A — ARTIGO GLOBECOM 2018

O artigo apresenta uma estratégia de utilização de múltiplos caminhos pela aplicação de programabilidade no plano de dados através da arquitetura P4. O uso de recursos de infra-estrutura de rede de uma maneira eficiente é uma necessidade que requer o emprego de técnicas que aproveitem o potencial de roteamento aliado à alta responsividade. Roteamento por múltiplos caminhos tem se projetado como boa prática para conseguir balanceamento de carga e algum nível de resiliência. No entanto, a fim de alcançar alta eficiência, as estratégias precisam de alta capacidade de resposta aos eventos de rede e posicionadas no plano de processamento do fluxo. No artigo é apresentada uma abordagem que usa a abstração Flowlet para gerenciar a divisão e manipulação de fluxos de rede e, através de um monitoramento simplificado localizado no plano de dados, realizar a detecção dinâmica do estado de conexão para efetuar balanceamento de carga com granularidade de subfluxo entre *links* paralelos.

- **Título:**
  An Efficient Multipath Mechanism based on the Flowlet Abstraction and P4
- **Conferência**
  IEEE Global Communications Conference (GLOBECOM)
- **Tipo**
  Trilha principal
- **Qualis**
  A1
- **Data**
  9 a 13 de Dezembro de 2018
- **Realizado em:**
  Abu Dhabi, United Arab Emirates

# An Efficient Multipath Mechanism based on the Flowlet Abstraction and P4

Marcelo Pizzutti, Alberto Egon Schaeffer-Filho
Institute of Informatics
Federal University of Rio Grande do Sul (UFRGS)
Porto Alegre, RS, Brazil
{marcelo.pizzutti, alberto}@inf.ufrgs.br

***Abstract*—The use of network infrastructure resources in an efficient manner is a necessity that requires the employment of a range of sophisticated techniques. Multipath is already designed as good practice to achieve load balancing and some level of resiliency. However, in order to achieve high efficiency, strategies need great responsiveness to network events. In this paper we present an approach that relies on the Flowlet abstraction to manage the division and monitoring of network flows. Through a simple idea, we add a function that dynamically monitors and acts in the data plane to find the minimum possible interval that allows the alternation of a flow between routes without causing adverse effects. We performed the evaluation using P4 as an engine that acts on the data plane.**

***Index Terms*—multipath, load balance, Flowlet**

## I. Introduction

The increasing demand for connectivity and fault tolerance has imposed several challenges on the areas that promote infrastructure and network management. Some technologies proposed in the last decade have addressed the concept of resilience and flexibility, centralizing management and segregating administration into levels. Software Defined Networking (SDN) decouples control and data planes [1] and provides an abstraction layer between physical and logical devices. In the data plane lies the routing equipment (switches and routers) and in the control plane a software platform, effectively responsible for the decisions.

The P4 programming language [2] came with the proposal of effectively programming the behavior of the devices, to define how packets are interpreted in the data plane and to enable, among other things, the interpretation of experimental protocols without relying on third parties, who manufactured the equipment. Similarly to what OpenFlow [3] does in the control plane, P4 increases the freedom of planning and programming in the data plane.

Quality of service in traditional networks can be affected by parameters of proprietary switches, traffic engineering mechanisms and inter-domain routing policies. Techniques that exploit the diversification of routes, namely multipath, have become important tools to meet the demands for higher transfer rates and lower associated delays. Multipath and load balancing are complementary when the goal is to achieve optimized use of the connection links, particularly in datacenters. Multipath explores the idea of defining multiple paths between source and destination nodes, assigning a given flow to each route with goals primarily related to resilience and performance. Load balancing [4], on the other hand, is linked to the technique that infers the classification and division of streams in order to achieve the best use of the channels without unnecessarily burdening processing and memory or causing adverse effects like retransmissions.

A number of approaches have been proposed with the goal of exploring control and data plane programmability for multipath routing and load balancing. Techniques for load balancing based on OpenFlow, which is the *de facto* SDN standard, present higher latency compared to solutions in the data plane. This approach does not allow a high level of reactivity to be applied to the same level of processing of routing rules.

While basic strategies satisfactorily support today's infrastructure, the expansion of the technology requires innovative mechanisms. HULA, for instance, [5] implements its strategy to achieve load balancing directly in the data plane using the promising P4 language.

This work presents a strategy that enables multipath and load balancing in the data plane in heterogeneous links and intra-domain environments. We argue that a load balancing approach that operates directly at the data plane is capable of increasing performance of management decisions when compared to control plane solutions. The concepts of programmability offered by the P4 architecture were used to evaluate the strategy that seeks to maintain a broader view of the one adopted by the HULA approach, focused on choosing the best next hop. Among the techniques applied in the data plane are the monitoring and measurement of latency dynamically used to split flows. An evaluation is conducted in order to analyze performance and stability requirements compared to the MPTCP [6] and LetItFlow [7].

The remainder of this paper is organized as follows: In Section II we present the main concepts underlying the proposed strategy. In the Section III we describe the elements that cooperatively act on the data plane. Section IV evaluates the proposed strategy and shows a comparison applying a metric focused on the experience observed by the user. In Section V we present the related work and finally in Section VI we outline the concluding remarks.

978-1-5386-4727-1/18/$31.00 

## II. Background

In this section we present an explanation about the ideas that have driven the application of programmability in network components and some of the strategies that allow using multipath routing.

### A. Network Programmability

The concept of programmable networks has evolved to make the management and configuration of network assets more flexible, enabling faster and safer deployment. The increase in computing power, the consolidation of programming techniques and the expansion of communication networks have accelerated the adoption of hardware devices with greater processing capacity.

Software Defined Networking (SDN) has incorporated the principles of programmable networks and is strongly characterized by the separation of the operational and management layers: data plane and control plane. In the data plane there are pieces of hardware that host the packet forwarding function, while in the control plane lies the portion of intelligence that orchestrates the interconnection and routing functions. The separation derived the idea of centralized administration, performed by the controller that is positioned in the control plane and logically connected to all the components of the physical substrate, creating a mapping that makes it possible to unify the administration.

OpenFlow [3] is the interface used in the communication channel between the controller software and the forwarding devices, which receive the rules of the controller and store them in tables in memory. Despite the limitations, many management skills are significantly improved with OpenFlow, but are still tightly bound to the firmware which is usually a proprietary product. More elaborate tasks, such as subflow handling or packet inspection are not easily supported by OpenFlow. A major factor linked to the current state of OpenFlow is that each update of the OpenFlow protocol version requires a new specification.

For a highly responsive approach, P4 [2], a name derived from "Independent Programming Protocol Packet Processors", defines a compiled and strongly typed language that generates a software artifact to be sent to an enabled device. The code written in P4 defines the behavior that the network equipment must comply with, and allows the development of applications that handle packets with implementation independent from the hardware in which they will execute. The language is a specification, a means of accessing the resources offered by hardware and that depend on the compiler provided by the manufacturer. In contrast to OpenFlow, the dependency on device manufacturers is relatively minor because it is not tied to specific functionalities or protocols, but to the language architecture. Manufacturers are also expected to provide standardized libraries with specific external functions, such as a specific hash algorithm. P4 defines the syntax and semantics, the data types, a set of instructions and structures that enable the developer to define a packet handler in software to run in the data plane.

### B. Multipath Routing

Multipath enables load balancing and resiliency through the use of redundant same-destination routes. Additional links can be active or used for backup, which may end up underutilizing the resources. In its simplest form, load balancing can use parallel links to transfer different flows, usually identified based on the fields of the IP and TCP headers, through independent but unvarying routes. A smarter approach is one that can split a single flow into subflows that can be independently routed. In this way, the use of multiple paths can generate undesirable effects because of the disparity of latency and throughput that may exist. The Equal-Cost Multi-Path (ECMP) routing, for example, is only viable on routes that have the same number of hops. The main problems arising from the use of multipath on parallel routes are observed by the TCP protocol, as packet reordering [8] [9] [10] [11].

In SDN, multipath can have the cooperation of a central controller for discovery and configuration of routes. In traditional multipath approaches these tasks are generally performed in a distributed form. At the local link level, Shortest Path Bridging (SPB) and TRILL (Transparent Batch Linking) make this work. In inter-networks there are also proposals that interact with the traditional routing protocols, as MIRO [12], compatible with the BGP protocol, and GMPLS (Generalized Multi-Protocol Label Switching) [13]. In the transport layer MPTCP (Multipath TCP) [6] figures with important contributions being compatible with the regular TCP protocol.

## III. Multipath Strategy on the Data Plane

### A. The Flowlet Abstraction

The basic principle of the Flowlet [14] is to divide a stream of data - flow - into bursts, as subflows, that are atomic groups of segments. The flow concept is defined by the tuple: source IP, destination IP, protocol version (4 or 6), the source TCP port, and the destination TCP port. The information used as a marker to divide the flow is based on the detection of an interval between a group of packets - *i.e.* a period of inactivity. These gaps are caused by the dynamics of the TCP protocol, the control of the sliding flow window, the available buffer of endpoints and network congestion states. After the controller, or other strategy, defines the parallel paths, even if composed of heterogeneous routes, the bursts of packets are switched between paths respecting the minimum interval that mitigates the reordering of packets. This forms its basic operation.

Existing solutions that depend on the *Flowlet* abstraction, such as HULA [5] and LetItFlow [7], treat this marker as a fixed parameter. CONGA [15] does not mention any dynamic change of the detection interval. In this way the estimation of gap is not aware of any possible congestion. FLARE [14], which introduced the concept of Flowlet, mentions that this measurement can be performed on all parallel paths. Our proposal relies on the dynamic measurement and considers only the last route used to make the decision on whether the current burst should use a new path. Thus, it is not necessary

to estimate the latency of all parallel routes, which allows adding and removing routes dynamically.

### B. Data Plane Multipath Mechanism

The proposed technique operates on the data plane and relies on the use of the Flowlet abstraction [14] to provide a simple and efficient mechanism for multipath routing. It applies the Flowlet technique by choosing the next path, which is composed of a sequence of switches, at random, respecting only the congestion state of the last route for which some data of flow was sent. This behavior is intended to avoid any kind of packet reordering and does not require the knowledge of all available parallel links and allows dynamic addition of paths. As shown in [7], the Flowlet division technique is effective even when asymmetric routes are considered, as opposed to the classic Equal-cost multi-path routing (ECMP).

Our multipath approach assumes the existence of an SDN controller to initially define the routes and logically configure the paths. Afterwards, the entire strategy works in the data plane and is based on P4. Each switch can logically play an intermediate or endpoint role. Endpoint switches are responsible for monitoring the connection, splitting and switching bursts of packets between the routes installed by the controller, as well as inserting the Flowlet header with the following formatting:

```
header flowlet_t {
      bit<16> flowlet_type;
      bit<2>  type;
      bit<8>  switch_id;
      bit<4>  priority;
      bit<10> route_id;
      bit<32> flowlet_time;
}
```

All the decisions over the flows happen at the endpoints, so the intermediate nodes only perform the routing operation between the input port and the output port, thus reducing processing and memory usage. Therefore, in intermediate nodes it is not necessary to perform any header change operation.

Figure 1 shows the main components located in the data plane. Label D1 represents an interface that receives multiple incoming flows in a processing pipeline. Each packet is then submitted to a parser (D2) that extracts the information contained in the headers. Only the headers involved with routing go to the next stage (D3), resulting in an action that is selected based on information from the headers, from the registers (which are statefull) and from the tables, which are populated by the control plane. Flow splitting operations happen through dynamic latency measurement between endpoit switches, without the knowledge of the applications involved.

Our technique focuses on an intra-domain environment, such as a datacenter, which presents specific characteristics, such as: total control over nodes involved in routing, a topology that presents redundant connections, allowing the formation of loops, and different levels of congestion.

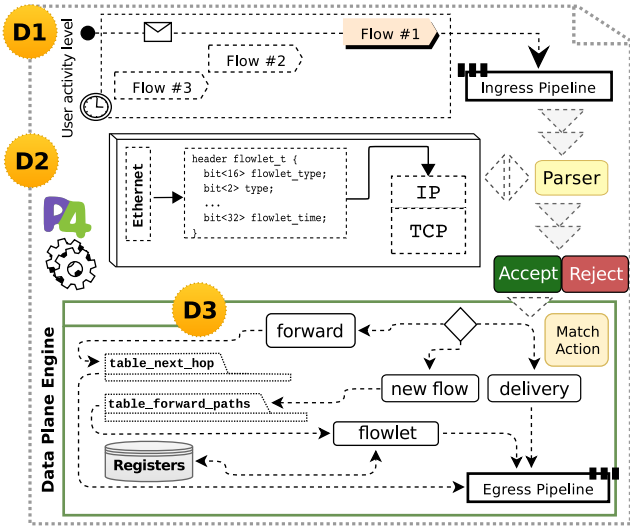



Figure 1: Data plane components.

### C. Basic Operation

*1) Dynamic RTT estimation:* RTT (round trip time) theoretically represents the time measure starting when a source sends a packet (message) and ends when the acknowledgment of the destination arrives at the sender. The RTT measure is the information needed to divide a flow into Flowlets and participates in the calculation used to detect a new group of packets, an interval that allows the change of route and, at the same time, minimizes the effects related to the reordering of packets.

The RTT view obtained by the TCP transport layer protocol is more accurate than that obtained in any other way, but it is still possible to estimate the minimum interval that permits to safely oscillate between parallel routes. To do this, we need to fix the routes and measure the RTT value for each link, from end to end. The header used to carry the RTT timestamp (see III-B) is embedded in all packets, however the measurement is performed by sampling. Code 1 presents a fragment of P4 code involved with the strategy that obtains the current state of each route.

```
if (header.flowlet.isValid()) {
   ...
   if (header.flowlet.type == 2) {
      ...
      if (current_flowlet_rtt < interval) {
         interval = interval - (interval>>6);
      } else {
         interval = current_flowlet_rtt;
      }
      ...
   }
}
```

Code 1: Component of the P4 program: RTT estimation.

*2) Control of ACK path:* Another strategy that does not interfere in the operation of the TCP protocol is the monitoring of the packets that carry the ACK flag. The idea basically consists of storing in a P4 register the last route by which a packet (belonging to a flow) has been received. Thus, if a

frame of size zero is observed and the ACK flag is on, the packet with acknowledgment will return through the same route that it arrived. This type of behavior is commonly observed in unidirectional streams, such as a file download. Code 2 presents a fragment of P4 code involved in Flowlet basic decision.

```
if (meta.tcp_payload == 0  &&
    meta.last_route_id > 0)  {
      meta.route_id = meta.last_route_id;
} else {
   if (meta.flow_ipg > interval || meta.last_route_id == 0
      || interval > avg_all_paths_switch * 3) {
      update_flowlet_id();
      table_ecmp_group.apply();
      table_forward_paths.apply();
      ...
   } else {
      meta.route_id = meta.last_route_id;
   }
}
```

Code 2: Component of the P4 program: Flowlet decision.

## IV. Evaluation

In this section we present an evaluation of the proposed multipath strategy. Firstly we will describe the topology and workload used. Then, the main technical characteristics involved in the assessment scenarios.

### A. Topology and Workload

The topology adopted is based on the one used in [15] [7] and is aimed at data center environments. It consists of four switches, two spine switches directly connected to two leaf switches. Each leaf is connected to 32 servers, as shown in Figure 2. The dashed link represents an off-line link, which characterizes an asymmetric environment.

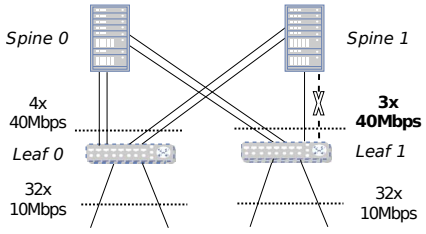



Figure 2: Topology used in the experiments.

In our experiments, the bandwidth of the links was reduced to 40Mbps between the main switches and 10Mbps on the links that connect to the servers. This maintains a 2:1 subscription which is typical in an actual datacenter. The experiments evaluate three strategies: MPTCP, LetItFlow and our approach. All evaluations are conducted in the same setup with identical configuration, operating system Ubuntu Linux Xenial virtualized with QEMU/KVM, with 8GBytes of RAM and full access to host processor, Intel (R) Core (TM) i7-3770 CPU. The Mininet emulator version 2.2 was used to define virtual hosts and switches.

Each host operates as client and as server and for each request a new connection (3 way handshake) is established. Servers accept multiple connections (multi-thread) and clients can establish exactly 20 connections with random servers for each evaluation. To manipulate the global load level, a time parameter, which is a delay in seconds between 5 and 15 is configured among one request and another.

The workload used is inspired by the evaluation performed in [7]: Enterprise and Web Search. Table 1 summarizes workloads that have different file size transfers and are selected randomly from the client side, respecting the distribution presented.

Table I: Workload used

| **Workload A** | | **Workload B** | |
|---|---|---|---|
| **File size** | **Distribution** | **File size** | **Distribution** |
| 1K | 0.4 | 10K | 0.2 |
| 10K | 0.4 | 100K | 0.4 |
| 1M | 0.1 | 1M | 0.2 |
| 10M | 0.1 | 10M | 0.2 |

### B. Metrics

The metric used for evaluation is the Flow Completion Time (FCT) [16]. FCT is focused on performance that is analyzed by the user and is related to congestion.

### C. Experimental Evaluation

The experiments were conducted in three environments: 1) MPTCP, 2) P4 (our approach), and 3) LetItFlow. Environment 1 runs the MPTCP distribution version 0.93[1]. The MPTCP manager paths were configured for the *ndiffports* parameter and the maximum number of subflows limited to 8. The rules of static routing were defined using native distribution balancing, assigning equal weights to the parallel connections. This gives a very simple bandwidth granularity based on source IP. In all environments the Mininet emulator was used and the traffic control configured for the Hierarchical Token Bucket (HTB) algorithm. The Maximum Transmission Unit (MTU) was set to 1490 bytes. In environments 2 and 3, the Mininet emulator was used in conjunction with bmv2[2] simulator version 1.11. For the prototype in P4 the routes were preconfigured, because this configuration is inserted in the context of the control plane. In LetItFlow we reproduced the environment described in [7] with the gap detection parameter set to 50ms, as observed in FLARE [14].

### D. Analysis of Results

Our measurements showed that LetItFlow and MPTCP have strong and weak points. By the topology used, several parameters are favorable to LetItFlow, especially the absence of loops and the reduced number of nodes. Figure 3 shows the set of analyses performed observing the topology and workload already presented.

The evaluation performed with MPTCP cannot necessarily be compared to the results obtained with the P4 simulator, due to the maturity level of the solutions. Some considerations can

[1]http://multipath-tcp.org/

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

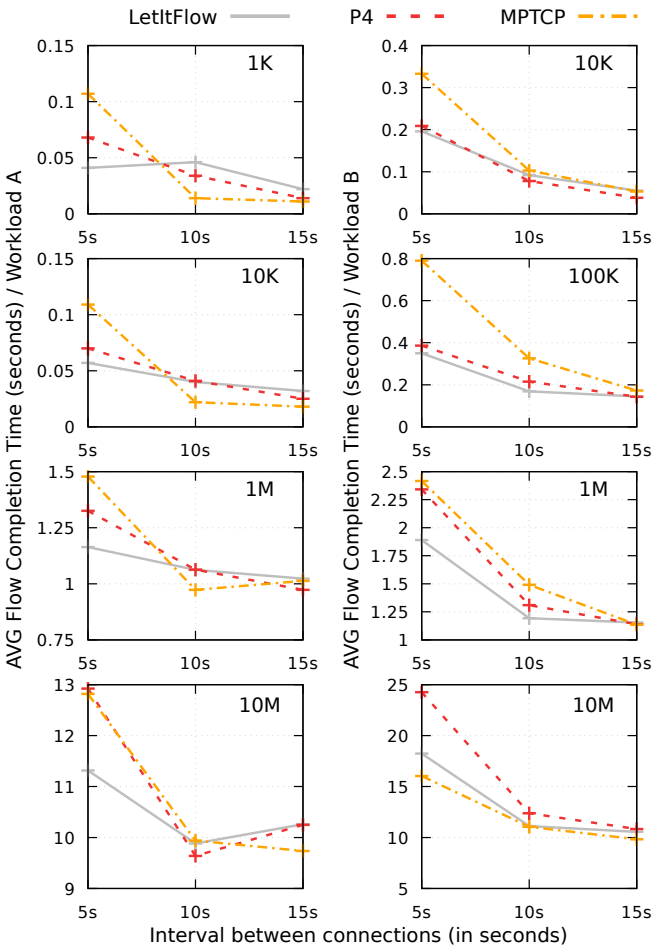



Figure 3: Analysis of flow completion time (FCT).

be observed in the analyses conducted with MPTCP that help us to understand the behavior of LetItFlow. A high level of congestion penalizes all types of traffic and a lower value of FCT (Flow Completion Time) had a consistently inverse relation. Through the results obtained and considering LeIt-Flow as ideal for the topology used, our approach presented similar results with both workloads. Small transfers of 1K achieved close results in the three approaches and long flows did not show significant variation among all transfer sizes at the highest intervals (15 seconds).

Our approach was stable and similar to LetItFlow, which allows us to infer that the strategy must achieve good results in mesh-format topologies, which use links with significantly different speeds and latencies without imposing a high processing load residing in the data plane.

## V. Related Work

In this section we present related work that involve programmable networks, multipath and load balancing. Table II has the purpose of structuring the functionality that differentiate each approach in its context. In the analysis, the operation plane, the scope and the management model are important to contextualize the requirements sought by the projects, including the strategies to be actuated.

B4 [17] received a lot of attention because it employs OpenFlow on a global scale and achieves very positive results. The efficiency achieved is the result of a customized hardware and hybrid devices integrated with OpenFlow. To perform load balancing, a variant of the algorithm ECMP *hashing* acts at the level of the flow.

CONGA [15] was designed for datacenters and uses the VXLAN encapsulation in the exchange of information. It has centralized administration and remains conscious of the global state of network congestion. It uses the technique of *Flowlet* and measures congestion using the Discounting Rate Estimator (DRE) algorithm.

HEDERA [18], also applied in datacenters, monitors the active flows centrally. It compares two algorithms, *Global First Fit* and *Simulated Annealing* to estimate the bandwidth of the shared ideal that each flow can allocate fulfilling the requirements of justice. Thus, it achieves results that are superior to techniques that use ECMP to mitigate the occurrence of collision of the hash.

HULA [5] focuses on the approach to programmability in the dataplane using P4 as a programming language. Each switch acts independently, striving to set only the next best hop. It has designed for the topology of a datacenter and estimates the congestion through its own messages. It maintains the estimate of the use of link per port and uses Flowlet to perform load balance.

SWAN [19] accommodates inter-datacenters topologies with centralized administration using OpenFlow. It applies the algorithm to consistently update the data plane and perform efficient use of the limited memory of forwarding tables.

LetItFlow [7] presents the most naive form of Flowlet which makes load balancing. The process occurs naturally in topologies of trees with multiple roots. Each chunk of data follows a randomly chosen path, betting on simplicity even with asymmetric topologies.

Our P4 approach has similarities with B4, SWAN and HEDERA since they do not have the direct purpose of operating in a datacenter environment, and also require to some extent the participation of the controller. However, our proposed mechanism is also similar to those approaches that focus on high responsivity in the data plane, such as CONGA and LetItFlow. CONGA also employs a technique for congestion detection. Our approach differs from the way load balancing occurs. With the DRE algorithm, CONGA selects the port that provides the best balance. We do not propose to do balancing by choosing the most favorable link, but changing a marker parameter used to detect new Flowlets and therefore diversifying the route. We believe this approach allows adding and removing routes without causing problems. Finally, our approach maintains an estimate of congestion by destination and route, unlike HULA that seeks the best next hop, maintains path usage estimation and assumes that the network topology has the notion of upstream and downstream switches.

Table II: Comparative table of architectures

| Architecture | Plan of action | Scope/ Topology | Platform | Management | Algorithm/ Technique | Attribute/ strategy for decision |
|---|---|---|---|---|---|---|
| **B4** [17] | data and control | global/ *datacenter* | OpenFlow e *merchant switch silicon* | centralized | variant of ECMP *hashing* | Applies the division of flows in multiple paths according to link capacity/demand. |
| **CONGA** [15] | data | *datacenter* | ASIC | distributed | Flowlet | Uses Flowlet to achieve high granularity and measures congestion using the Discounting Rate Estimator (DRE) algorithm. |
| **HEDERA** [18] | data and control | *datacenter* | OpenFlow | centralized | *Global First Fit* e *Simulated Annealing* | Detects large flows in border switches and chooses the best path. |
| **HULA** [5] | data | *datacenter* | P4 | distributed | Flowlet | Each switch only determines the next best hop, and uses probes to discover the best path to forward. |
| **SWAN** [19] | data and control | *inter-datacenters* | OpenFlow | centralized | *not identified | Presents mechanisms to update consistent data plane and implements flow control in the sender switches. |
| **LetItFlow** [7] | data | *datacenter* | P4 | distributed | Flowlet | Simply applies the splitting and forwarding of packets by randomly choosing each next hop. |

## VI. Conclusions

In this paper, we presented an adaptative multipath mechanism based on the Flowlet abstraction and P4. We compared and analyzed our proposal with respect to MPTCP and LetItFlow. With this analysis, we can see that the naive strategy LetItFlow presents satisfactory results in oriented topologies. Some dynamic controls may not be needed in datacenter topologies, however, strategies that identify flow classes can be applied to free certain paths that suffer from bottlenecks. This type of treatment requires unified orchestration and administration, a role played by a controller. In this sense, our strategy allows a coupling with the control plane, which makes the planning and management of routes (multipath), while load balancing using Flowlet happens entirely in the data plane. In order to deal with heterogeneous routes, we also propose dynamically detecting the state of the connection links, which allows better handling of asymmetric topologies.

## Acknowledgments

This work has been partially supported by grants NSF CNS-1740911 and RNP/CTIC (P4Sec). Alberto Schaeffer-Filho would like to thank CNPq for research grants ref. 311088/2015-5 and 407899/2016-2.

## References

[1] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014.

[2] P. Bosshart, D. Daly *et al.*, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[3] N. McKeown, T. Anderson *et al.*, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[4] S. Prabhavat, H. Nishiyama *et al.*, "On load distribution over multipath networks," *IEEE Communications Surveys Tutorials*, vol. 14, no. 3, pp. 662–680, Third 2012.

[5] N. Katta, M. Hira *et al.*, "Hula: Scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16. New York, NY, USA: ACM, 2016, pp. 10:1–10:12.

[6] A. Ford, C. Raiciu *et al.*, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 6824, Jan. 2013.

[7] E. Vanini, R. Pan *et al.*, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 407–420.

[8] J. Qadir, A. Ali *et al.*, "Exploiting the power of multiplicity: A holistic survey of network-layer multipath," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2176–2213, Fourthquarter 2015.

[9] M. Li, A. Lukyanenko *et al.*, "Multipath transmission for the internet: A survey," *IEEE Communications Surveys Tutorials*, vol. 18, no. 4, pp. 2887–2925, Fourthquarter 2016.

[10] J. He and J. Rexford, "Toward internet-wide multipath routing," *IEEE Network*, vol. 22, no. 2, pp. 16–21, March 2008.

[11] J. Domżał, Z. Duliński *et al.*, "A survey on methods to provide multipath transmission in wired packet networks," *Computer Networks*, vol. 77, pp. 18 – 41, 2015.

[12] W. Xu and J. Rexford, "Miro: Multi-path interdomain routing," in *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '06. New York, NY, USA: ACM, 2006, pp. 171–182.

[13] E. Mannie, "Generalized Multi-Protocol Label Switching (GMPLS) Architecture," RFC 3945, Nov. 2004.

[14] S. Kandula, D. Katabi *et al.*, "Dynamic load balancing without packet reordering," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 2, pp. 51–62, Mar. 2007.

[15] M. Alizadeh, T. Edsall *et al.*, "Conga: Distributed congestion-aware load balancing for datacenters," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 503–514.

[16] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 59–62, Jan. 2006.

[17] S. Jain, A. Kumar *et al.*, "B4: Experience with a globally-deployed software defined wan," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, Aug. 2013.

[18] M. Al-Fares, S. Radhakrishnan *et al.*, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 19–19.

[19] C.-Y. Hong, S. Kandula *et al.*, "Achieving high utilization with software-driven wan," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 15–26, Aug. 2013.

# ANEXO B — ARTIGO INFOCOM 2019

A programabilidade empregada no plano de controle foi potencializada pelas iniciativas SDN/OpenFlow que trouxeram um novo impulso direcionado para a gestão conjunta ao plano de controle e utilização de práticas de planejamento na área de redes de computadores. No entanto, aplicações que exigem alta responsividade e demandam mecanismos mais dinâmicos que executam além do plano de controle somente conseguem ser efetivamente exploradas via interações específicas com o plano de dados, onde as decisões e ações reativas podem ser executadas na linha de processamento de pacotes. P4 alcança esse objetivo fornecendo controle sobre o *hardware* que executa tarefas de encaminhamento de pacotes, trazendo programabilidade para o plano de dados. O artigo apresenta uma abordagem híbrida para roteamento por múltiplas rotas que integra cooperativamente atividades situadas nos planos de dados e de controle. No plano de dados a programabilidade alcançada pela arquitetura P4 age ativamente fazendo a detecção do nível de congestionamento das rotas e tratando de tarefas ligadas à manipulação de subfluxos de dados, enquanto no plano de controle tarefas passivas agregam-se à visão global obtida pelo controlador, o que permite acoplar, por exemplo, diferentes estratégias de definição de caminhos e de priorização de rotas.

- **Título:**
  Adaptive Multipath Routing based on Hybrid Data and Control Plane Operation
- **Conferência**
  IEEE International Conference on Computer Communications (INFOCOM)
- **Tipo**
  Trilha principal
- **Qualis**
  A1
- **Data**
  29 de Abril a 2 de Maio de 2019
- **Realizado em:**
  Paris, France

# Adaptive Multipath Routing based on Hybrid Data and Control Plane Operation

Marcelo Pizzutti, Alberto E. Schaeffer-Filho
Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS)
Porto Alegre, RS, Brazil
{marcelo.pizzutti, alberto}@inf.ufrgs.br

***Abstract*—Control plane programmability, primarily enabled by SDN/OpenFlow, has brought new impetus to already consolidated management and planning practices in the area of computer networks. However, applications that require high responsiveness and demand more dynamic mechanisms executing beyond the control plane can only be effectively exploited through interactions with the data plane, where decisions and reactive responses can be made at line rate. P4 achieves this goal by providing directives that allow a certain level of control over the hardware that performs packet forwarding tasks, bringing programmability to the data plane. In this paper, we present a multipath routing strategy that takes full advantage of a hybrid SDN/OpenFlow and P4 architecture. In the data plane, P4 and the Flowlet abstraction are used to actively perform load balancing and routing over multiple asymmetric paths, in addition to monitor active links. In the logically centralized control plane, the overall view of the network is leveraged to perform more elaborate passive tasks, which include mapping paths, configuring devices and updating routes asynchronously. Experimental evaluations are conducted in which we analyze different strategies for choosing the routes that provide the best results based on RTT values received from the switches.**

***Index Terms*—multipath, load balance, Flowlet, control plane, data plane, programmability, P4, hybrid architecture**

## I. Introduction

Network management is traditionally closely related to the hardware components used for packet forwarding and routing tasks. Maintaining the network infrastructure demands not only efforts toward the administration of the physical equipment, but also efforts employed in the management of the logical part, which basically comprises planning, configuration, and scheduling tasks. However, dynamic and emerging demands have more elaborate requirements for resiliency and flexibility [1] [2].

Software Defined Networking (SDN) [3] focuses on abstracting certain functions and unlinking the data plane from the control plane, and promoting programmability in the control plane. Despite the ideas surrounding SDN have been around since the 70's, the Openflow protocol [4] is the realization of the SDN concept that has been effectively adopted by the industry. However, hardware and embedded software (firmware), where OpenFlow resides, remain tightly coupled, making it difficult to customize actions or experiment with new protocols, for example, because this would require the release of new firmware for the hardware [5]. In tasks that require frequent controller-switch interaction this architecture does not perform well because it adds a series of delays that occur naturally in the communication.

To address these constraints, the P4 language [5] brings programmability to the data plane. This allows more control over the forwarding devices and enables programmers to specify how switches should process packets and to define customized routines. Because they are executed within the processing pipeline, policies supported by the P4 architecture can execute at line rate. In line with SDN, there is a logical communication interface that allows a central unit to interact with each P4 switch. Figure 1 presents an overview of the current paradigms that involve network programmability, with the new interaction that the P4 architecture adds to SDN. A P4 code is embedded in a switch and an API allows the controller to manipulate its internal structures.

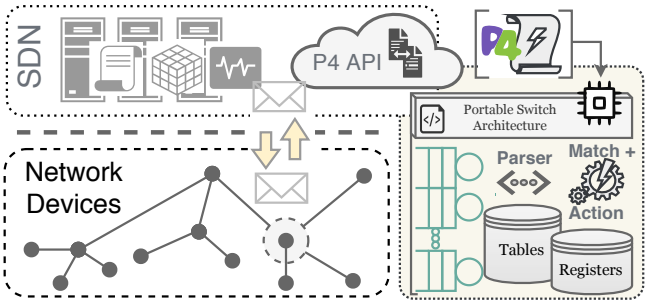



Fig. 1: Overview of programmability in networks.

In an environment of increasing connectivity, techniques that optimize the use of multiple links are essential to meet quality of service requirements [6] [7]. Because communication naturally uses the principle of sharing the medium, traffic restrictions or bandwidth reservation may represent underutilization of resources. Therefore the use of a multipath strategy enables load balancing and softens delays associated with the oscillation that appears spontaneously in the use of a link and may eventually generate congestion. In this sense, multipath and load balancing [8] promote the rational use of network links, exploring multiple paths between source and destination nodes. Load balancing acts cooperatively to define the strategy used to direct the packets along the routes, leading to optimized use of the communication channels without causing adverse effects such as retransmissions.

In this paper, we advocate that traditional network management in general, and multipath routing and load balancing in specific, can capitalize on the benefits of network programmability. The opportunities that arise with the application of programmability in the data plane can be directed to the improvement of solutions that up to now remain exclusively in the field of the physical substrate, the switches, as it opens up the possibility of developing new hybrid strategies that integrate with the control plane. With this is mind, this work proposes an architecture that relies on specific data and control plane functionality to offer an efficient multipath routing mechanism. Our proposed hybrid multipath strategy relies on the joint operation of two adaptive feedback control-loops: *(i)* an online control-loop for fast decision-making at the switches themselves, and *(ii)* an offline control-loop that is periodically executed by the controller to analyze routes and make global changes. Through this separation, tasks linked to high responsiveness are positioned in the processing pipeline of the forwarding devices, bringing agility in the execution of functions for division and monitoring of data flows. Other activities can occur asynchronously in the control plane, which has a broad vision that favors any more elaborate decisions.

Our contributions in this work go beyond the question that involves the paradigm of centralization or distribution of roles. In particular, we:

*(i)* Propose the organization of components and the distribution of functions to execute multipath and load balance through the Flowlet [9] technique;
*(ii)* Exploit the recent (and evolving) P4 architecture for data plane programmability and the notion of a controller to interoperate with the control plane;
*(iii)* Employ link state monitoring mechanisms implemented entirely in the data plane and routing strategies that seek to simplify the dynamics of the programmability aspect.

The remaining of this paper is organized as follows. Section II presents the main concepts and technologies that influenced our thinking. In Section III we detail our proposal to reach multipath and load balance with a hybrid strategy that relies on programmability both in the control and in the data planes. Section IV describes the experimental evaluation of our strategy. Section V discusses the related work and finally Section VI concludes the paper and presents future work.

## II. Background

In this section, we present the background that is necessary to understand the main aspects of our approach, with focus on the multipath and load balance application goal.

### A. Network Programmability

The SDN paradigm addresses network management difficulties that are essentially related to monitoring and configuration of network equipment, which brings an increase in scalability, consistency, and reliability. Arguably, SDN gave rise to additional challenges such as the issue of the controller as a point of failure [10]. However, the benefits associated this set of technologies are undeniable.

Through OpenFlow [4], any application can interact with the switches that comply with the protocol. Packet forwarding is determined by rules inserted in a switch's flow table by the controller. These flow tables can be populated by the controller in a proactive manner. Subsequently, when the switch receives a packet, if a match on the packet headers is found in the flow table, an action (e.g., *drop* or *forward* to port) is triggered. Otherwise, when there is no matching rule in the flow table, a default action can be configured, for instance, to send the packet to be inspected.

Both the OpenFlow API, match options and types of actions on the flows are fixed in the firmware installed on the switch, so the tables and configured actions are the only dynamic portion. This makes it difficult, for example, to implement custom tasks like to perform validations and operations on statefull flows. Of course, the adoption of OpenFlow primitives greatly facilitates the management of the network infrastructure. Tasks that do not require frequent interaction with the controller, or that are asynchronous, operate satisfactorily. However, there is still dependency on proprietary firmware and new versions of the specification for feature enhancement.

The advances in the design of hardware solutions for processing packets are a promising field for the development of abstractions, such as P4 [5] and Domino [11]. This new type of interaction with the forwarding devices can effectively bring programmability to the data plane. The P4 architecture (Independent Programming Protocol Packet Processors) specifies the syntax, semantic rules, logical blocks and their interfaces. Blocks are scopes of code defined by the programmer that form a logical sequence (*pipeline*) through which the packets are submitted when received.

The P4 code is compiled specifically for a platform and requires the compiler to be compatible with the hardware architecture. External libraries with hardware-specific routines can extend its functionality. An API is generated and allows a controller to change the objects defined in the data plane. Compared to OpenFlow, the switch becomes a much more versatile unit that interprets the software artifact generated through the P4 compiler, and does not depend on manufacturer intervention for the addition of new functionality constructed with the language. Therefore P4 allows programming the behavior of the switch with a much higher level of detail.

### B. Multipath Routing

Multipath makes it possible to enumerate and use the parallel paths of a network topology. This technique brings many benefits, not only in terms of redundancy, but also performance enhancement when coupled with load balancing [8]. The granularity can vary from packet to packet, which in practice can generate numerous negative effects; by stream, identified by information in the TCP/IP header; and by subflow tagging, such as Flowlet [9], which achieves greater flexibility to divide the fragments that will be routed through different paths. Another aspect concerns the characteristics of the path. ECMP, for example, which acts in a hop-by-hop basis and is the most common implementation of multipath,

works with the concept of paths that have the same metrics (hops) and operates with flows.

For load balancing, splitting by stream is straightforward and does not generate adverse effects such as packet reordering, which may disorient the controls applied by the TCP protocol and induce retransmissions and the reduction of throughput [8] [12] [13] [14]. However, this technique is subject to loss of performance due to long established flows, and does not allow the alternation of routes, creating bottlenecks. Flowlet [9] addresses precisely this issue: it enables a flow to be divided into subflows that are transmitted independently through differentiated paths minimizing the possible adverse effects, such as packet reordering.

Multipath and load balance can also be analyzed according to the TCP/IP layer in which they operate. At the link level, Shortest Path Bridging (SPB) and TRILL (Transparent Interconnection of Lots of Links) enable multiple equal cost pathways in local scope networks, both calculating the best paths through the network. Advancing to layer 3, inter-networks, where the routing mechanisms act cooperatively and in a decentralized way, the hop-by-hop routing paradigm dominates [8]. Well-established routing protocols, such as Open Shortest Path First (OSPF) and BGP (Border Gateway Protocol), along with MPLS (Multiprotocol Label Switching), operate broadly on a global scale. Proposals exploring multipath and load balance capabilities are reported in the technical literature [15] [14], bringing innovations or extending features: MIRO [16] is compatible with the BGP protocol, and GMPLS (Generalized Multi-Protocol Label Switching) [17] enables MPLS to establish and manage multiple paths. These proposals address the issue of flow-balancing because they act below the TCP transport layer, which is effectively responsible for the end-to-end connection. In this context, MPTCP (Multipath TCP) [18] figures with important contributions by being compatible with the regular TCP protocol and adding an abstraction level that separates a TCP session into sub-flows associated with disparate paths, allowing load balancing.

## III. Hybrid Multipath Strategy using the Data and Control Planes

Our proposal addresses the multipath perspective through an integration between the data and control planes. High responsiveness is supported by the data plane, while passive tasks that require a broader knowledge of the topology and overall state of the network are forwarded to the control plane. As such, our proposed hybrid multipath strategy relies on the joint operation of two adaptive feedback control-loops: *(i)* an online control-loop for fast decision-making at the switches themselves, and *(ii)* an offline control-loop that is periodically executed by the controller to analyze routes and make global changes.

This hybrid multipath routing strategy relies on data and control plane programmability, using respectively the P4 language [5] and a switch-compatible interface similar to what exists in OpenFlow [4], to update rules. This separation between planes becomes very clear and makes the switches self-sufficient after the initial bootstrap in the occurrence of connection problems with the controller, which is responsible for asynchronously pre-computing and ranking alternative routes in the topology. The cooperative operation consists of assigning tasks that require high responsiveness, such as forwarding and splitting of flows, to the data plane and performing less time sensitive tasks, e.g., route discovery, asynchronously in the control plane.

### A. Data Plane Basic Operation

Activities that execute in the data plane, from the point of view of responsiveness, are independent from the control plane and are based on splitting network traffic flows to achieve load balancing between parallel and asymmetrical paths. The portion of the multipath strategy that executes in the data plane is based on our previous work [19], and relies on P4 and on the Flowlet abstraction.

*1) The Flowlet Abstraction:* The basic principle of Flowlet [9] is to divide a flow (stream of data) into segments that are considered atomic groups of packets. The flow concept is defined by the tuple: source IP, destination IP, protocol version (4 or 6), the source TCP port, and the destination TCP port. Each subflow is defined by the detection of a gap of activity that will allow the independent routing of each group of packets. The reason for these inactivity intervals (lack of transmitted packets for short periods) can be due to the dynamics of the TCP protocol, the control of the sliding flow window, the available buffer at endpoints and network congestion states.

The Flowlet strategy can operate with parallel links and in our approach it allows the dynamic addition of paths in heterogeneous networks. As shown in [20], the Flowlet division technique is effective even when asymmetric routes are considered, in contrast to the classic Equal-cost multipath routing (ECMP).

*2) Data Plane Multipath Mechanism:* On the data plane each switch can logically play an *intermediate* or *endpoint* role. Endpoint switches are responsible for monitoring the connection, splitting and switching bursts of packets between the routes installed by the controller as well as inserting the Flowlet header with the following format:

```
header flowlet_t {
   bit<16> flowlet_type; //type of the upper header
   bit<2>  type;         //1: request, 2: response
   bit<9>  switch_id;    //identifier of the device
   bit<13> id;           //unique request control
   bit<24> route_id;     //identifier of the path
   bit<32> flowlet_time; //timestamp
}
```

All the decisions over the flows happen at the endpoints, so the intermediate nodes only perform the forwarding operation between the input port and the output port, so reducing processing and memory usage. Thus, intermediate switches do not need to perform any header modification.

The controller must keep a logical data structure that stores all the interconnections between the switches. This requires the exchange of messages between the switches and their neighbors, which are forwarded to the controller and enable

the discovery of paths (Section III-B1). With the list of paths determined, the controller labels each route with a unique identification tag, and also defines which routes will initially be configured as active, that is, which will participate in the multipath process. On each switch, for each IP address/destination mask, a pre-defined set of routes will be enabled.

To pre-configure the active routes that will be used the controller must modify data in two tables that exist on the switches, which are statefull structures of the P4 language. The `forward_paths` table is indexed by IP/mask and is used to store active routes. The other, `next_hops`, is indexed by the route identifier and returns which switch ports are path-related. For each route, the controller (i) analyzes all intermediate switches, (ii) populates the data about route information in the `next_hop` table of the intermediate nodes and, for those routes that are to be used for multipath (the ones that are active), (iii) inserts the information in the table `forward_paths`. Thus, each switch in a path is aware of the route identification tag (i.e., `route_id` in the Flowlet header) and the input and output (physical or logical) ports that serve bidirectionally the network traffic.

Given that at any point in time there is only a subset of active paths, not all possible routes are used to allow a switch to communicate with the rest of the network. Each switch, when performing load balancing, chooses one among a fixed number of parallel paths and uses the Flowlet technique to alternate between its own local options. As will be explained in Section III-B, the role of the controller is to analyze the feedback for each triple <route, source_switch, destination_switch>, and swap those for better alternatives that may be available in the database managed by the controller. Routes that offer worse conditions according to specific metrics (e.g., RTT) are gradually replaced.

Once the controller defines the routes and logically configures the paths, the Flowlet abstraction is used as a simple, efficient and active mechanism for load balancing. Our mechanism relies on the dynamic measurement of RTT and considers the recently used routes to make the decision on whether a new burst should use an alternative route or not. Thus, it is not necessary to estimate the RTT latency of all parallel routes, which allows adding and removing routes dynamically and simplifies the process. The RTT metric was chosen to evaluate the overall health state of the connection due to the simplicity and power of representing performance related problems. Certainly the combined use of other metrics, such as the cumulative load of the end-to-end network path, can act in a complementary way and increase accuracy, but adding some complexity.

Figure 2 shows the main components located in the data plane. Label D1 represents a switch interface that receives multiple incoming flows in a processing pipeline. Each packet is then submitted to a parser (D2) that extracts the information contained in the headers. Only the headers involved with routing go to the next stage (D3), resulting in an action that is selected based on information from the headers, from the registers (which are statefull) and from the tables, which store rules added by the controller. For each destination, the switch queries the target IP in the `forward_paths` table using the *long prefix match*.

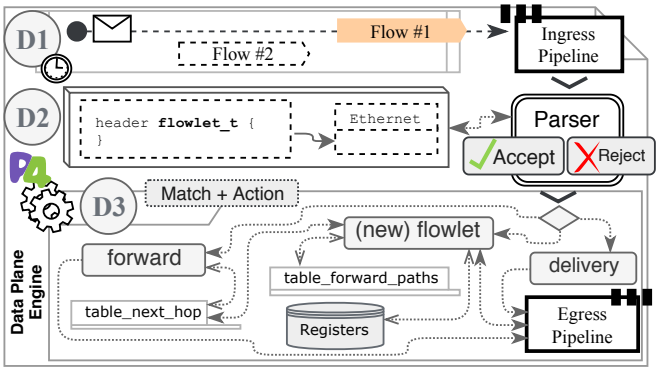



Fig. 2: Data plane components.

Our technique focuses on an intra-domain environment which presents specific characteristics, such as: total control over nodes involved in routing and supporting topologies that present redundant connections allowing the formation of loops. Applicability in traditional WAN networks can be partially achieved by reallocating the Flowlet header to above the IP header.

*3) Dynamic RTT Estimation:* RTT (round trip time) theoretically represents the time measure starting when a source sends a packet (message) and ends when the acknowledgement of the destination arrives at the sender. This information is used to estimate a secure value that characterize a timeout of communication. The correct determination of the interval allows the change of route and, at the same time, minimizes the effects related to the reordering of packets.

The estimation of the measurement detected by the TCP transport layer, involved in the end-to-end logical connection, is certainly quite accurate. Our proposed RTT estimation technique indirectly calculates this value, which can suffer some variations due to processing that happens in the upper layers of the TCP/IP stack. To calculate this metric, it is necessary that the switches periodically measure the delay between each two host/switch along the paths and that the switches through which a path passes do not change. The header used to carry the RTT timestamp (Section III-A2) is embedded in all packets, however the measurement is performed by sampling.

### *B. Data and Control Plane Cooperation*

Techniques applied to multipath, making division over data flows, naturally promote traffic balancing and the decisions involved should be executed in a way that does not interfere and does not impose processing bottlenecks. However, less time sensitive actions, which include mapping the available paths, configuring devices and updating routes, may demand knowledge of the topology and of the overall state of the network, and should preferably be performed asynchronously.

Protocols that discover routes in a distributed way, such as link-state routing protocols, can deliver some level of redundancy, but with greater granularity. In this way, centralizing information in the role of the controller can bring the benefit of applying different strategies with greater freedom. This view, which is very close to what OpenFlow offers, can be complemented by an approach that brings programmability to the data plane, such as the P4 architecture.

In our approach the control plane participates passively in the multipath and load balance tasks by integrating the route discovery operations and optimizing the selection of active routes. Analogously to what happens in a traditional network, the first phase is the topology discovery that acts in a coordinated way between the planes through the exchange of messages encapsulated in a header created for this purpose. Data is organized into a graph structure, thus allowing the controller to simulate path finding without involving any message exchange with the switches. This makes it possible to calculate the paths in a centralized way and make them available in a repository. For each path, the controller configures all the switches involved, which consists in informing the route identifier tag and the physical or logical ports by which the traffic must be switched. Then, the controller informs the P4 units which routes are active and, after this intervention, the switches are able to route traffic through multiple paths independently.

Subsequently, adjustments in the active routes take place. The role of the controller is to receive traps periodically sent by the switches, to analyze the alternatives and, if appropriate, to promote the replacement of some route. Next, we discuss the main aspects of our approach in more details.

*1) Path Discovery:* The first task performed by the controller is to make the logical mapping of connections between the routing devices (switches) and proceed with the discovery of paths, mapping the neighborhoods and ports that connect the links. With this mapping it is possible to construct a logical structure in the form of a graph. For the discovery of paths, five main tasks are performed: (i) root selection, (ii) discovery of routes beginning at the vertices with the lowest degrees and ending at a root, (iii) combination of the possible paths by creating a subgraph that connects the roots, (iv) make the other combinations and (v) assignment of a unique identifier to each created path. *Roots* are aggregate nodes which concentrate an amount above the average of transit paths. Each path that starts from any one switch and reaches a root is understood as a *route fragment*, and every path that connects two roots is understood as a *bridge*.

The pseudo-code that makes the selection of the roots is presented in Algorithm 1. There are two stages to select the root nodes, which can not be adjacent. First, the nodes are sorted in descending order of degree, and the ones with higher degree that satisfy the condition indicated in line 7 are included as root. With the partial roots, a breadth-first search (BFS) is performed (line 9). This will generate a set of paths (route fragments) that have only one root each. On the discovered paths, the occurrence of each vertex belonging to the graph is counted (lines 13-20). Then, results above the general average of the degrees of the graph will be added to the set of roots listed in the first phase (lines 21-25). For each node, if the amount of occurrences in the paths exceeds the condition shown in the line 22, it is selected as a new root, and added to the existing ones. As an additional constraint, the number of root nodes must be higher than two and should not exceed 20% of the number of elements in the graph.

The introduction of new nodes in the graph suggests that recalculation can be performed. The strategy to reach the goal may include a time marker for expiration of the paths, but this will not be addressed in this work.

**Algorithm 1** Root selection heuristic

**Require:** A connected graph $G(V, E)$
1: **procedure** GET_ROOTS
2: $roots \leftarrow \emptyset$
3: $N \leftarrow max\ V\ degree$
4: $M \leftarrow count\ V > avg(degrees)$
5: $qtde \leftarrow M$ * ($N \div size$(G))
6: $D \leftarrow$ {v ∈ V}, sorted by $deg$(v) in descending order
7: $roots \leftarrow \{\{D_i\} \mid D_i <$ *qtde*; $D_i \notin \forall(roots.adj)\}$
8: **for** each $\{i \in D;\ i \notin roots\}$ **do**
9: $BFS\ path\ discovery$
10: **end for**
11: $count \leftarrow \emptyset$
12: $sum \leftarrow 0$
13: **for** each $\{i \in all\_paths\}$ **do**
14: **for** each $\{v \in V\}$ **do**
15: **if** $i \in v$ **then**
16: $count[i] + +$
17: **end if**
18: $sum \leftarrow sum + count[i]$
19: **end for**
20: **end for**
21: **for** each $\{i \in count \mid i \notin \forall(roots.adj)\ \}$ **do**
22: **if** $count[i] > (sum \div size(all\_paths))$ **then**
23: $roots \leftarrow roots \cup i$
24: **end if**
25: **end for**
26: **return** $roots$
27: **end procedure**

After the selection of the roots, the next step is the discovery of paths that have as origin each non-root vertex of the graph and that end at a root node. In this process, such non-root vertices must have paths reaching at least two different root nodes. Thus, if a root becomes unavailable, each non-root node should still be able to route traffic through an alternative root. With a selected list of route fragments, which were discovered by the BFS heuristic, multiple combinations occur to find single and loop-free paths. Routes called *bridges* are used to aggregate paths and reduce the number of identifiers. So, to reach any two points of the mesh there are two types of paths: paths that do not pass through any bridge or paths that contain one or more bridges. Since each route fragment may contain only one root, increasing the number of roots increases the level of fragmentation.

The `route_id` used in the Flowlet header (Section III-A2) is a tag that identifies a route, which we called the Unique

Identifier of the Path (UIP) (Figure 3). In our implementation, the UIP is made up of 24 bits and the first most significant bit tells if the route contains any bridge. If the most significant bit is set to 1 the UIP is formed by one bridge route combined with another path. The reason for using "bridges" is to optimize the storage of information, since the intermediate switches only need to take decisions on a portion of the UIP and there is no need to know all the possible combinations.

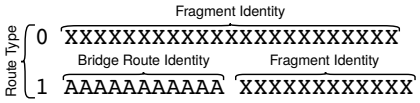



Fig. 3: UIP format in binary representation.

Regarding the memory limitations imposed on the switches for storing routing metadata, the use of the ternary mask (the concept used by UIP) contributes significantly to attenuating the amount of rules required, including related to paths without bridges. The aggregation of routes can be explored by the controller, for example, using a UIP containing a bridge to aggregate common paths of an end to end route. Further, the size of the `route_id` field of the header also can be extended to accommodate a larger amount of switches or to work with a different UIP format.

*2) Election of Active Routes and Link Monitoring:* By integrating the monitoring activities that are performed in conjunction with the data plane, the active routes are installed by the controller and are subject to update. Figure 4 shows an overview and the logical location of the main structures involved. In the data plane a sequence of checks decides whether each packet is in transit, belongs to a Flowlet or can originate another Flowlet. Information about the active paths is synchronized with the data base located on the controller, which also stores the RTT historical values of each path used by the switch. The UIP is used in this exchange of information as identifier.

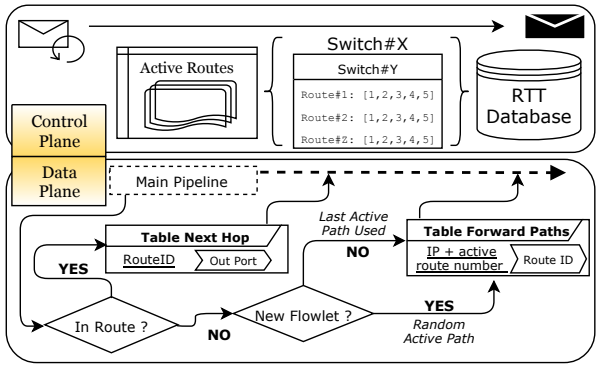



Fig. 4: Architecture for multipath routing considering data and control plane cooperation.

A route is composed of a group of switches that read the inserted Flowlet header, query the output port in an internal table and proceed with the transfer, as seen in the Section III-A2. So an active route can serve more than one destination since each switch involved with a route checks whether the destination is attached to a local port and determines whether delivery should be performed. Therefore, two hosts that wish to exchange information can use a broader route, which reduces the amount of paths generated.

A forwarding device receives a fixed set of routes and alternates between them as new Flowlets are detected. There is no selection criterion for the first active routes since the worst performing ones will be replaced. The calculation used to decide whether a new route should be installed is made entirely in the data plane considering the average RTT of the last route used and the RTT of the current route. If the difference is greater than a threshold specified by a configuration policy (e.g., 200%), the switch will request a new route to the controller. With this behavior, uniformity between competing routes is taken into account, because it avoids large oscillations that can eventually generate misinterpretations by the algorithm that coordinates TCP congestion control.

*3) Updating Routes on Switches:* When switches request a new route, they encapsulate the request in a custom packet header inserted using P4 that is sent to the controller. The controller stores data from the last *n* samples for each active path, which allows the use of different reactive strategies to choose a new route. This can possibly achieve a balance between the parallel paths. As such, different *route update policies* can be implemented with this information, depending on the context.

Among all the routes listed, the ones that are not in use and available in the controller RTT database can be activated on the devices as needed. This allows a group of paths in a multiple-choice topology to be enabled on demand to alleviate congestion states. In Section IV we evaluate our solution considering route update policies based on the Exponential Moving Average (EMA) and on the Weighted Moving Average (WMA) of RTT measurements. However, other decision policies can be easily implemented within our strategy to accomodate different needs.

The process of adding or removing switches must be accompanied by a new route discovery process, interconnection mappings and route configurations in the devices. One possible strategy is to use a two-stage deployment, i.e., configuration and activation, using a timestamp marker to synchronize the activation.

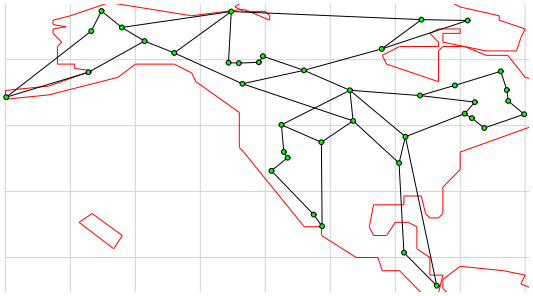
## IV. Evaluation

In this section we present an evaluation of the proposed adaptive multipath strategy based on hybrid data and control plane operation. Section IV-A describes the topologies and workloads used. Section IV-B discusses the metrics we considered in our evaluation, and Section IV-C presents the evaluation results.

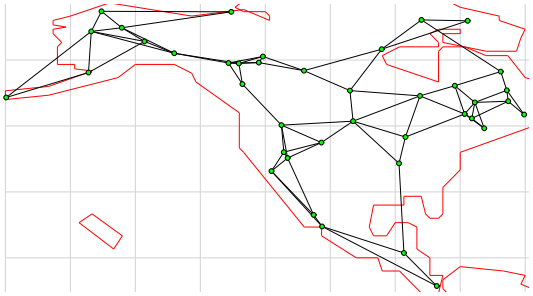
### A. Topology and Workload

The network topologies adopted in our evaluation use the classification presented on the study developed by Kamiyama

et al. [21], which analyzed the impact that the topological formation exerts in a scenario that involves data delivery, more specifically, video on demand. A study is presented along with 23 commercial Internet Service Providers (ISPs) and a topology classification is defined based on the characteristics of the formed graph. We selected two arrangements to be used in our experiments: *Hub and Spokes* (H&S) and *Ladder*.



(a) Ladder topology.



(b) Hub and Spokes topology.

Fig. 5: Topologies used in the experiments.

The main distinction between these topological formations is that a Hub and Spokes network presents nodes with aggregation function, while Ladder does not rely on concentrator nodes and has loops used to interconnect distant points. To create the topologies we used iGen[1]. The topologies are composed of 40 switches with interfaces that connect switches limited to a rate of 100Mbits. Each switch has a connected node running a client/server application with a link limitation of also 100Mbits with the switch interface.

To better understand the experiments, only 20 clients are randomly selected to be active at any moment, and we ensure that between 5 and 6 connections are established simultaneously. For every request a new TCP connection (3-way handshake) is established. The workload used is inspired by the evaluation performed in [22] [20]. Table I shows the different file size transfers that are performed randomly from the client side, respecting the distribution presented. Servers can accept multiple connections and clients execute the workload presented in Table I with random servers for each evaluation, with a 2 seconds interval between connection requests.

[1]http://igen.sourceforge.net/

| File size | Distribution |
|---|---|
| 100K | 0.5 |
| 1M | 0.3 |
| 10M | 0.1 |
| 100M | 0.1 |

TABLE I: Workload used in the experiments.

In the data plane, the minimum value used as a parameter ($\beta$) for the detection of a new Flowlet is defined according to Equation 1. The condition associated with $\gamma$ has the purpose of forcing the change of route in case a flow presents latency much higher than that found in the parallel alternatives.

$$\beta = \begin{cases} \beta * \lambda + \Delta * (1 - \lambda), & \text{if } (\Delta > \beta) \vee (\Delta > \gamma * 3) \\ \beta * (1 - \lambda) + \Delta * \lambda, & \text{otherwise} \end{cases} \quad (1)$$

where: $\lambda$ = 0.25, which is used to smooth the oscillation of $\beta$
$\Delta$ = last RTT measurement
$\beta$ = stored timeout used to detect new Flowlets
$\gamma$ = average RTT of all paralell paths

Next, we define the condition for triggering the trap sent by the data plane to the controller for requesting a route change. More specifically, this is triggered when the measured RTT exceeds 150% of the arithmetic mean between the new RTT measurement and the previously stored RTT over the parallel paths between a source and destination pair.

In our evaluation, three different *route update policies* are assessed to perform the selection of routes: *(i)* Naive (random) route update, *(ii)* Weighted Moving Average of RTT measurements, and *(iii)* Exponential Moving Average of RTT measurements. Equation 2 expresses a strategy in which the average of RTT measurements is calculated based on weights that decrease for older values. We assume the RTT history storage is limited to the last 5 samples for each active route.

$$y = \sum_{i=1}^{n} i; \quad WMA = \sum_{i=1}^{n} x_i \times \left(\frac{i}{y}\right) \quad (2)$$

where: $n = 5$

On the other hand, Equation 3 expresses a strategy in which the average of RTT measurements is calculated based on an Exponential Moving Average, considering a smoothing parameter $\lambda$. We also assume the RTT history storage is limited to the last 5 samples for each active route.

$$x_1 = \bar{x}; \quad EMA_i = (\lambda \times x_i) + (1 - \lambda) * EMA_{i-1} \quad (3)$$

where: $\lambda = 0.30$
$i = 2, \ldots, 5$

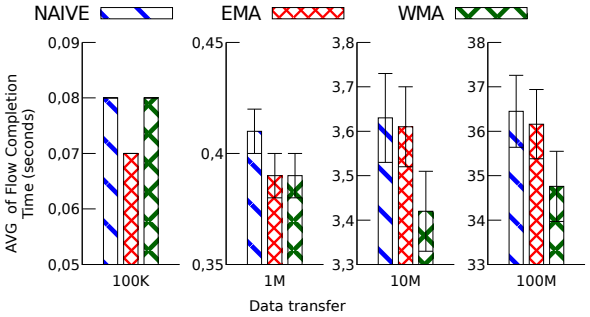
### B. Metrics

We used the Flow Completion Time (FCT) [23] as the main metric for evaluating our adaptive routing strategy. Furthermore, the cumulative frequency of data traffic for all switch

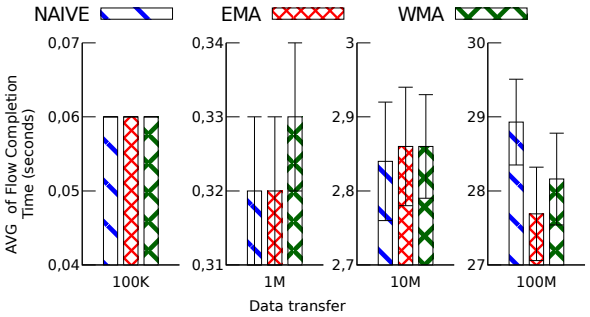
interfaces is used to evaluate the usage of alternative routes in the topology. The values corresponding to the transmitted data were normalized in order to identify if the route update policy enforced by the controller is able to use the available links in a balanced way.

### C. Experimental Results

The evaluations were conducted on the three route update policies enforced by the controller to make route changes on the switches. All evaluations were performed in the same setup with identical configuration, operating system Ubuntu Linux Xenial virtualized with QEMU/KVM, with 16 GBytes of RAM and access to the host processor, Intel(R) Xeon(R) CPU E5-2670. The Mininet emulator version 2.2 was used to define virtual hosts and switches. The Maximum Transmission Unit (MTU) was set to 1400 bytes for user interfaces. The bmv2[2] simulator version 1.12 was used to interpret the P4 code. No limitations were considered regarding the link that connects the controller to the switches.




(a) Ladder topology.

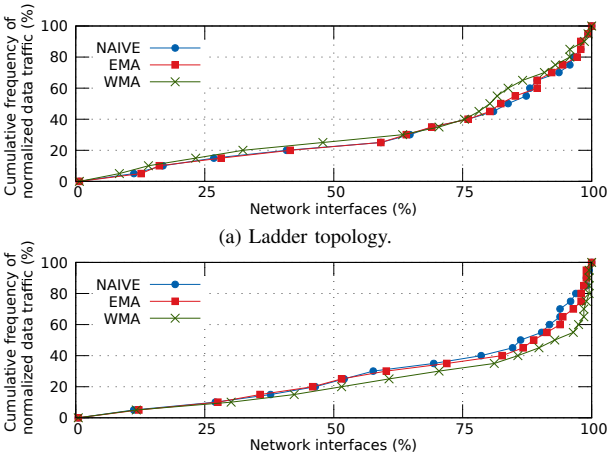



(b) Hub and Spokes topology.

Fig. 6: Flow Completion Time metric.

The results obtained for the Flow Completion Time metric are presented in Figure 6a and 6b. We can observe that in both topologies, especially considering large file transfers, both WMA and EMA present lower FCT in comparison to the naive policy. Ideally, a longer history of RTTs should be used to illustrate the different temporal characteristics of the WMA and EMA route update policies, but in our experiments in the H&S topology mainly the EMA update policy performed better, whereas in the Ladder topology the WMA performed

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




(a) Ladder topology.




(b) Hub and Spokes topology.

Fig. 7: Data traffic sent by nodes.

better than the other policies (except for the 100k small file transfer).

In the second comparison (Figures 7a and 7b) we analyzed the behavior of the cumulative frequency of data traffic per interface (NIC). This can indicate how well alternative routes are used for data transfer. It is clear that the paths are explored more extensively in the Ladder topology, and in this particular case the WMA route update policy performed better.

Despite the limitations found in simulation environments, we can conclude that there is a wide range of applications that can benefit from our hybrid strategy for multipath routing, such as real-time applications or applications that are subject to the seasonality of an event.

## V. Related Work

In this section we present related work that involve the use of cooperation between data plane and control plane, including programmable networks, multipath and load balancing.

B4 [1] employs OpenFlow on a global scale and achieves good results using merchant switch silicon integrated with OpenFlow. It combines existing routing protocols and traffic engineering services to accommodate elastic traffic demand. A custom implementation of ECMP *hashing* acts at the level of the flow to perform load balancing. B4 is also a hybrid strategy for centralized traffic engineering that relies on the separation between the data and control planes. HEDERA [24] monitors active flows and seeks to obtain a global view of routes and traffic to estimate the demand of flows in datacenters. It applies dynamic flow scheduling for decisions over large flows and monitors the ideal bandwidth sharing.

SWAN [25] works in inter-datacenters topologies with centralized administration using OpenFlow. It attempts to consistently update the data plane and perform efficient use of the limited memory of forwarding tables using a small amount of capacity on links to apply updates on the data plane. It allocates resources centrally, improving the fairness

and supporting flexible network-wide sharing. Finally, MicroTE [26] logically separates the predictable traffic from the unpredictable one and employs weighted ECMP with a load-sensitive traffic engineering approach in datacenters. It uses OpenFlow and relies on a centralized controller for decision-making, while generating low overhead with control messages.

Our work shares several characteristics with the research above, but it is novel in its use of data plane programmability (P4), which made possible to use the active load balance Flowlet technique in cooperation with the control plane. With a simplified approach to detect link state by reading RTT from links, the controller can act asynchronously and serve as an interface for the application of various strategies.

## VI. Conclusions

This paper exploits the use of programmability to offer a hybrid strategy for multipath routing, which operates in the data and control planes. We advocate that this separation, facilitated by the OpenFlow protocol, needs to evolve even more to meet requirements of applications that demand high responsiveness, as is the case for load balance. Network programmability, in our view, will bring more freedom to network operators and promote the implementation of new and enhanced Internet services.

The results obtained in our evaluation demonstrate the effectiveness that the interaction proposed by a hybrid strategy can provide. The P4 language is still maturing, and even in an environment that employs a simulation evaluation, we can observe many benefits, especially due to a broader view of what is occurring in the network. Our proposed approach was evaluated using route update policies based on the Exponential Moving Average (EMA) and on the Weighted Moving Average (WMA) of RTT measurements, in addition to a random/naive update policy. As future work, we plan to evaluate the approach using alternative traffic metrics and route update policies.

## Acknowledgments

Alberto E. Schaeffer-Filho would like to thank CNPq for research grants ref. 311088/2015-5 and 407899/2016-2. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and also by NSF CNS-1740911 and RNP/CTIC (P4Sec) grants.

## References

[1] S. Jain, A. Kumar *et al.*, "B4: Experience with a globally-deployed software defined wan," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, Aug. 2013.

[2] C.-Y. Hong, S. Kandula *et al.*, "Achieving high utilization with software-driven wan," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 15–26.

[3] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014.

[4] N. McKeown, T. Anderson *et al.*, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[5] P. Bosshart, D. Daly *et al.*, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[6] S. Ferlin, O. Alay *et al.*, "Revisiting congestion control for multipath tcp with shared bottleneck detection," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.

[7] S. M. Irteza, H. M. Bashir *et al.*, "Load balancing over symmetric virtual topologies," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[8] J. Qadir, A. Ali *et al.*, "Exploiting the power of multiplicity: A holistic survey of network-layer multipath," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2176–2213, Fourthquarter 2015.

[9] S. Kandula, D. Katabi *et al.*, "Dynamic load balancing without packet reordering," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 2, pp. 51–62, Mar. 2007.

[10] Y. Zhang, L. Cui *et al.*, "A survey on software defined networking with multiple controllers," *J. Netw. Comput. Appl.*, vol. 103, no. C, pp. 101–118, Feb. 2018.

[11] A. Sivaraman, A. Cheung *et al.*, "Packet transactions: High-level programming for line-rate switches," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 15–28.

[12] M. Li, A. Lukyanenko *et al.*, "Multipath transmission for the internet: A survey," *IEEE Communications Surveys Tutorials*, vol. 18, no. 4, pp. 2887–2925, Fourthquarter 2016.

[13] J. He and J. Rexford, "Toward internet-wide multipath routing," *IEEE Network*, vol. 22, no. 2, pp. 16–21, March 2008.

[14] J. Domżał, Z. Duliński *et al.*, "A survey on methods to provide multipath transmission in wired packet networks," *Computer Networks*, vol. 77, pp. 18 – 41, 2015.

[15] R. Wójcik, J. Domżał *et al.*, "A survey on methods to provide interdomain multipath transmissions," *Computer Networks*, vol. 108, pp. 233 – 259, 2016.

[16] W. Xu and J. Rexford, "Miro: Multi-path interdomain routing," in *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '06. New York, NY, USA: ACM, 2006, pp. 171–182.

[17] E. Mannie, "Generalized Multi-Protocol Label Switching (GMPLS) Architecture," RFC 3945, Nov. 2004.

[18] A. Ford, C. Raiciu *et al.*, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 6824, Jan. 2013.

[19] M. Pizzutti and A. Schaeffer-Filho, "An efficient multipath mechanism based on the flowlet abstraction and P4," in *IEEE Global Communications Conference (GLOBECOM 2018)*. IEEE, 2018.

[20] E. Vanini, R. Pan *et al.*, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 407–420.

[21] N. Kamiyama, R. Kawahara *et al.*, "Impact of topology on parallel video streaming," in *2010 IEEE Network Operations and Management Symposium - NOMS 2010*, April 2010, pp. 607–614.

[22] M. Alizadeh, T. Edsall *et al.*, "Conga: Distributed congestion-aware load balancing for datacenters," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 503–514.

[23] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 59–62, Jan. 2006.

[24] M. Al-Fares, S. Radhakrishnan *et al.*, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 19–19.

[25] C.-Y. Hong, S. Kandula *et al.*, "Achieving high utilization with software-driven wan," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 15–26, Aug. 2013.

[26] T. Benson, A. Anand *et al.*, "Microte: Fine grained traffic engineering for data centers," in *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '11. New York, NY, USA: ACM, 2011, pp. 8:1–8:12.