RESISTING: Um Novo Mecanismo de Fast-Reroute com Distribuição de Pacotes em Switches Programáveis P4

Daniel B. de Lima[1], **Francisco G. Vogt**[1], **Alan T. da Silva**[1], **Christian E. Rothenberg**[1]

[1] Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)

{d208925,f234632@,a265560}dac.unicamp.br, chesteve@dca.fee.unicamp.br

Abstract. *Fast-Reroute (FRR) mechanisms provide failure recovery in different types of network architectures, avoiding application flow performance degradation. Current FRR mechanisms for P4 programmable switches provide fast reroute approaches without considering flow load balancing support. Thus, existing methods do not deliver efficient use of bandwidth resources, resulting in overloaded network links under multiple failure recovery scenarios. This work proposes* `RESISTING` *as a new FRR-ECMP (Equal-Cost Multi-Path) mechanism for Programming Protocol-independent Packet Processor (P4) programmable switches, delivering packet flow load balancing and failure recovery. The proposed method is compared against Primitive for Reconfigurable Fast Reroute (PURR) – the state-of-the-art FRR mechanism in P4 – under one, two, and three links failure experiments. The results show that the proposed prototype does not incur packet losses during the experimental evaluation, while PURR presents losses under two or more simultaneous failures.*

Resumo. *Os mecanismos de Fast-Reroute (FRR) operam na recuperação de falhas em diversas arquiteturas de redes, prevenindo a degradação dos fluxos de pacotes das aplicações. Os trabalhos que suportam mecanismos de FRR na arquitetura de switches programáveis na linguagem Programming Protocol-independent Packet Processor (P4) aplicam uma abordagem de recuperação de falhas sem considerar um método de distribuição de fluxos de pacotes. Deste modo, essas soluções não contribuem para um uso eficiente dos recursos de banda, que pode causar sobrecarga nos links durante a recuperação de cenários drásticos de falhas. O presente trabalho propõe* `RESISTING` *como um novo mecanismo de FRR Equal-Cost Multi-Path (ECMP) para switches programáveis em P4, oferecendo balanceamento dos fluxos de pacotes durante a recuperação de falhas. O mecanismo de recuperação proposto é comparado ao Primitive for Reconfigurable Fast Reroute (PURR) – mecanismo considerado o estado da arte – durante os eventos de uma, duas e três falhas. Os resultados mostram que o método proposto não apresenta perda de fluxos de pacotes durante a avaliação experimental, ao passo que o PURR apresenta perdas a partir de duas falhas simultâneas.*

1. Introdução

À medida que a tecnologia de planos de dados programáveis avança, novas aplicações surgem para atender às demandas de serviços de instituições públicas e privadas. Atualmente, destaca-se a grande diversidade de aplicações na linguagem *Programming*

Protocol-Independent Packet Processors (P4) [Bosshart et al. 2014] disponíveis para operação de redes, tais como roteamento baseado em L2, L3, TAG, *Multiprotocol Label Switching* (MPLS), *Multicast* e outros tipos de serviços, incluindo *Network Address Translation* (NAT), Filtros, *Sflow*, *Equal Cost Multi Path* (ECMP) e *In-band Network Telemetry (INT)*. Entretanto, a construção de aplicações voltadas para tolerância à falhas é considerada um grande desafio na linguagem P4 [Sedar et al. 2018], visto que não existe uma funcionalidade de recuperação de falhas nativamente disponível nos hardwares utilizados [Chiesa et al. 2019].

Para lidar com tal limitação, os mecanismos *Fast-Reroute* (FRR) emergem como soluções eficazes para recuperação de falhas no plano de dados, já que promovem o re-roteamento rápido dos pacotes durante eventos de falhas. A estratégia de tolerância à falhas precisa lidar com variados cenários de infraestrutura de rede, dentre os quais destaca-se o data center. Neste cenário, o intenso volume de tráfego arbitrário exige mecanismos sofisticados de recuperação capazes de lidar com múltiplas falhas aleatórias sem causar degradação do tráfego entre servidores. O balanceamento de tráfego ECMP ganhou relevância neste panorama, sobretudo na infraestrutura de nuvem, ao promover a distribuição de fluxos de pacotes entre *links* com o mesmo custo. Tal habilidade otimiza o uso dos recursos de banda e aumenta a capacidade de comunicação entre servidores.

Os mecanismos FRR apresentados nos trabalhos [Merling et al. 2020, Chiesa et al. 2019] permitem a recuperação de falhas sem prejudicar a latência e a vazão dos fluxos de pacotes. No entanto, esses trabalhos adotaram uma estratégia primária de recuperação que não leva em consideração a quantidade de *links* operacionais no data center. Como resultado, todos os fluxos de pacotes afetados pela falha são redirecionados para um único *link backup*, o que pode causar uma sobrecarga adicional de tráfego [Zhang et al. 2008] e, consequentemente, reduzir a largura de banda efetiva entre servidores. O presente artigo propõe o RESISTING (*datacente**R** **E**qually co**S**t mult**I**path fa**S**t re-rou**TING***): um novo mecanismo de FRR integrado com ECMP para switches programáveis em P4 com o propósito de oferecer resiliência, robustez e distribuição dos fluxos de pacotes no data center durante os eventos de falhas. Nosso mecanismo implementado em P4 foi compilado para o switch *Barefoot Tofino* e utiliza switches virtuais *Behavioral Model Version 2* (BMv2).

Para avaliar a eficiência do mecanismo proposto, este trabalho realizou dois experimentos distintos. No primeiro, avaliou-se a quantidade de pacotes submetidos à recirculação e o impacto da recuperação de falhas na latência dos pacotes, em comparação com o cenário sem falhas. Em seguida, foi realizada uma avaliação de degradação do fluxo de pacotes, comparando a implementação proposta com o *Primitive for Reconfigurable Fast Reroute* (PURR) [Chiesa et al. 2019], mecanismo considerado estado da arte, durante eventos de uma, duas e três falhas simultâneas. Os resultados do primeiro experimento indicaram que, mesmo durante o aumento da taxa de transmissão e a aplicação de falhas, o mecanismo RESISTING apresentou uma taxa de $\approx 0\%$ de pacotes recirculados em relação à quantidade de pacotes gerados (e.g., de 31 pacotes recirculados para 91 milhões de pacotes, no pior caso), sugerindo que o aumento na taxa de transmissão não resultou em aumento proporcional na quantidade de pacotes recirculados. Além disso, durante o evento de cinco falhas simultâneas, não foi identificado impacto negativo na latência dos pacotes. No segundo experimento o mecanismo proposto foi capaz de se

recuperar de até três falhas simultâneas. Tal recuperação otimiza o uso dos recursos de banda em comparação com o PURR, que apresentou perdas de fluxos de pacotes a partir de duas falhas.

O restante do conteúdo está estruturado da seguinte forma: na Seção 2, são descritos alguns dos fundamentos teóricos utilizados. Os trabalhos relacionados são abordados na Seção 3. Na Seção 4, é apresentada a arquitetura FRR-ECMP. Na Seção 5, é avaliada a capacidade de recuperação e resiliência. Por fim, na Seção 6, são discutidas as conclusões e apontados os trabalhos futuros.

2. Fundamentação teórica

Esta seção apresenta o embasamento teórico relacionado a este artigo e aborda os seguintes tópicos relevantes: conceitos de recuperação e balanceamento de carga aplicados no contexto de data centers e a definição da topologia de rede e das possíveis falhas nos enlaces. Por fim, é realizada uma avaliação em um cenário de data center para demonstrar e comparar os mecanismos de recuperação PURR e RESISTING durante eventos de falhas.

O re-roteamento rápido FRR representa uma solução eficiente para a recuperação de falhas em redes de computadores, sem a necessidade de recorrer a elementos externos [Luz et al. 2022], tais como o plano de controle ou o controlador SDN (*Software Defined Networks*). Essa estratégia permite que a rede se recupere de falhas locais ou em dispositivos adjacentes em milissegundos ou até microssegundos [Chiesa et al. 2021]. No contexto dos data centers, uma estratégia utilizada consiste em combinar a funcionalidade do ECMP, responsável pelo balanceamento de carga, com a solução de FRR [Chiesa et al. 2021], visando mitigar falhas. É importante destacar que essa estratégia é viável porque o mecanismo ECMP, ao operar com múltiplos caminhos alternativos instalados no plano de dados, não depende de elementos externos para a recuperação de falhas. Na linguagem P4, o FRR e o ECMP são componentes separados [Merling et al. 2020, Sivaraman et al. 2015], exigindo esforço de desenvolvimento para sua integração no plano de dados. Discutimos mais sobre a integração desses componentes na Seção 4.

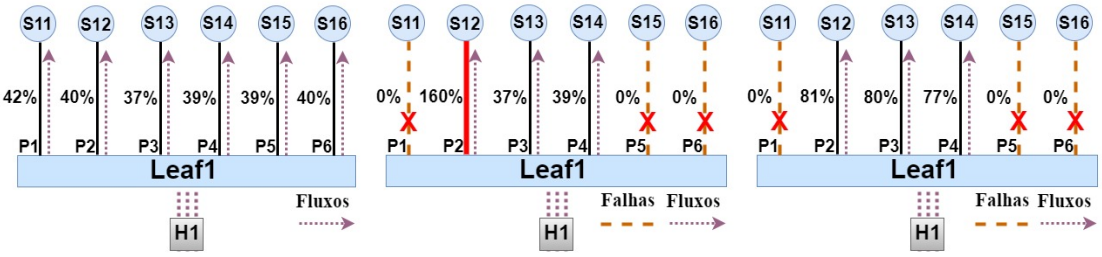
A definição do cenário de análise é fundamental, inclusive a topologia de rede e os pontos a serem analisados. Dentre as topologias discutidas na literatura ([Al-Fares et al. 2008, Greenberg et al. 2009, Alizadeh and Edsall 2013]), optou-se pela *Clos leaf-spine* devido aos resultados positivos na distribuição de tráfego ECMP com *uplinks* de alta capacidade (40Gbps e 100Gbps) durante os experimentos com *Flow Completion Time* (FCT) ([Alizadeh and Edsall 2013]). No presente trabalho, os experimentos foram conduzidos em um switch *leaf* com seis *uplinks*, permitindo a avaliação em uma escala reduzida da topologia *Clos*. Além disso, a recuperação sugerida atua no escopo da saída do tráfego, ou seja, do *host* $\rightarrow$ *leaf* $\rightarrow$ *spine*. Portanto, o tráfego de saída se torna o sentido mais relevante para avaliação. A Figura 1 ilustra um estudo de caso considerando um cenário que emprega a topologia *Clos leaf-spine*. Tal estudo utilizou o emulador Mininet, bem como o software BMv2, e propôs três situações distintas: a primeira, sem ocorrência de falhas, apenas realizando o balanceamento de carga; a segunda, aplicando o mecanismo de recuperação PURR, tido como estado da arte para lidar com as falhas; e a terceira, com o mecanismo FRR sugerido neste artigo atuando diante das falhas.

ECMP. A Figura 1a apresenta o switch *leaf* 1 conectado ao *host* h1, responsável por gerar

fluxos de pacotes para vários *hosts* na rede. Seis *spines* na camada de *core* suportam o *leaf* 1 e realizam o roteamento do tráfego entre os switches, permitindo a comunicação entre os *hosts* da rede. Para avaliação, consideramos apenas o tráfego de saída do *leaf* 1 para os *spines* e desconsideramos o tráfego de retorno. O *leaf* 1 utiliza o balanceamento de carga ECMP e distribui equilibradamente 2400 fluxos distintos de pacotes TCP (gerados pelo *host* h1) entre os seis *uplinks*, de modo que cada um apresenta uma taxa de ocupação abaixo de 50% - o que não impede a distribuição eficiente dos fluxos de pacotes.

PURR. A Figura 1b apresenta o mesmo contexto do cenário 1, porém com falhas simultâneas nas portas P1, P5 e P6 do *leaf* 1. Nesse caso, o mecanismo PURR é acionado, redirecionando os fluxos afetados para a próxima porta operacional, que é a P2, sem considerar que as portas P3 e P4 também estão operacionais. Isso causou uma sobrecarga na porta P2, que ultrapassa sua capacidade de 100% do *uplink* devido à adição de fluxos de pacotes.

RESISTING. A Figura 1c apresenta o mesmo contexto de rede descrito nos cenários anteriores, mas desta vez o switch *leaf* 1 utiliza o mecanismo proposto neste trabalho para lidar com as falhas nas portas P1, P5 e P6. O RESISTING distribui os fluxos de pacotes afetados pelas falhas entre as portas P2, P3 e P4 que estão operacionais no switch. Esta distribuição evita sobrecargas e potenciais riscos de degradação no fluxo de pacotes.



(a) Sem Falha (b) PURR: 3 Falhas (c) RESISTING: 3 Falhas

Figura 1: Utilização dos *uplinks* em diferentes cenários de falhas.

3. Trabalhos Relacionados

Nesta seção, discutimos três trabalhos da literatura relevantes para este artigo: 1) o IPFRR-LFA [Shand and Bryant 2010], que aborda a estratégia primária de FRR e o uso de ECMP como método de recuperação. Adaptamos o IPFRR (LFA)[1] para a linguagem P4 e avaliamos seu funcionamento durante uma falha no emulador Mininet, comparando-o com outros trabalhos. 2) O BIER FRR [Merling et al. 2020], que apresenta os mecanismos de recuperação IP-FRR e BIER-FRR, ambos disponíveis na linguagem P4 no repositório dos autores. 3) O PURR [Chiesa et al. 2019], considerado o mecanismo estado da arte. Adaptamos o PURR[2] para a linguagem P4 com base no código fonte P4 do trabalho [Merling et al. 2020], para fins de comparação nesta seção e nas avaliações experimentais das Seções 2 e 4. A Tabela 1 apresenta uma comparação entre os mecanismos de recuperação da literatura e o RESISTING.

A RFC 5714 [Shand and Bryant 2010], denominada "*IP Fast Reroute Framework*", apresenta um estudo abrangente sobre o nosso tema de pesquisa e descreve

[1]P4-LFA - `https://github.com/danielbl1000/P4-LFA`
[2]Simple_sw_PURR -`https://github.com/danielbl1000/P4-simple_switch_purr`

Trabalho	Recuperação	Falhas	Topologia	Plataforma	Balanceamento
IPFRR	Rota Backup	Uma Falha	Anel	BMv2	Não
BIER-FRR	Consulta Ternária	Uma Falha	Anel	BMv2	Não
PURR	Consulta Ternária	Múltiplas	Leaf-Spine	BMv2, Tofino, FPGA	Não
RESISTING	Recirculação temporária	Múltiplas	Leaf-Spine	BMv2, Tofino	ECMP

Tabela 1: Comparação entre o estado da arte e o mecanismo proposto.

dois mecanismos de recuperação relevantes: o LFA (*Loop-Free Alternates*) e o ECMP. Na estratégia do LFA, uma rota backup sem loop é pré-calculada no plano de controle e em seguida instalada no plano de dados para recuperação da falha no caminho principal. O ECMP é um mecanismo de balanceamento de carga que pode ser utilizado como um mecanismo de recuperação ao permitir que múltiplas rotas no plano de dados sejam usadas como backup na eventualidade de falha em um caminho principal. Esse conceito fundamental foi a base do desenvolvimento inicial do mecanismo FRR-ECMP proposto.

Os autores do trabalho [Merling et al. 2020] abordam o conceito do BIER (*Bit Indexed Explicit Replication*), uma nova arquitetura de roteamento multicast que consome menos recursos dos roteadores em relação ao protocolo PIM[3]. Além disso, o trabalho também é relevante por implementar o BIER em linguagem P4 com a integração de dois mecanismos de recuperação: um para proteção do tráfego *unicast* denominado IP-FRR e outro para proteção do tráfego *multicast* denominado BIER-FRR. Neste trabalho, o método de recuperação adotado consiste em uma estratégia primária que desvia os fluxos de pacotes do caminho principal afetado pela falha para um caminho backup, sem considerar outros caminhos operacionais para balanceamento de carga.

Os estudos apresentados em [Sedar et al. 2018, Chiesa et al. 2019] trouxeram a proposta de uma primitiva de FRR para plano de dados programável P4, denominada de (PURR), conforme mencionado em [Chiesa et al. 2019]. Esta primitiva apresenta uma notável capacidade de recuperação local, viabilizando o reroteamento rápido de uma ou várias falhas simultâneas, sem qualquer perda de pacotes ou aumento de latência. A solução proposta pelos autores realiza uma única busca ternária na tabela PURR, encontrando a próxima porta operacional de saída. O resultado dessa busca proporciona a indicação precisa da porta operacional subsequente, independentemente da quantidade de falhas simultâneas. Conforme ilustrado na Figura 1b, a estratégia do PURR consiste em desviar os pacotes do caminho principal afetado pela falha para um caminho backup, sem considerar os caminhos operacionais.

A Tabela 1 compara de forma sucinta as principais características dos trabalhos relacionados com o nosso mecanismo: 1) estratégia de recuperação; 2) quantidade de falhas recuperadas; 3) topologia de rede testada; 4) validação na plataforma física e/ou software; 5) suporte a distribuição de pacotes;

A estratégia de recuperação dos trabalhos listados na Tabela 1 apresenta vantagens em relação ao nosso trabalho, pois não causa aumento da latência, degradação do tráfego e *overhead*. Isso se deve ao fato de que nossa estratégia de recuperação exige um custo mínimo devido à recirculação temporária de pacotes. No entanto, em cenários de rede que exigem múltiplos *links* disponíveis para atender a um intenso volume de tráfego [Zhang et al. 2008], nossa estratégia de recuperação oferece a vantagem de distribuição de fluxo de pacotes durante os eventos de falhas, otimizando o uso dos

[3]Protocol Independent Multicast - `https://www.rfc-editor.org/rfc/rfc7761.html`

recursos de banda e prevenindo a sobrecarga dos *links*.

4. RESISTING: datacenteR Equally coSt multIpath faSt re-rouTING

Esta seção descreve os componentes essenciais da nossa arquitetura de recuperação ECMP, abordando os seguintes assuntos: o método de detecção de falhas adotado, o funcionamento do ECMP e, por fim, explora o conceito fundamental do FRR proposto.

4.1. Detecção de falha

A recuperação em uma arquitetura de rede é acionada quando há perda de sinal na porta física, interrupção no *link* ou falha no dispositivo de rede. Na arquitetura P4, são utilizados diferentes métodos para detectar falhas. No switch BMv2, um método sugerido pela comunidade de desenvolvimento P4[4] consiste em criar uma tabela ou registro [Luz et al. 2022] para armazenar o estado operacional (*up/down*) da porta. No switch Tofino, é possível utilizar o mesmo método ou um mecanismo de *trigger* que envia pacotes sinalizando a falha para o plano de controle ou para um elemento remoto. Essa abordagem possibilita iniciar o processo de recuperação ou notificar a indisponibilidade da porta. Para padronizar os protótipos BMv2 e Tofino, optamos por implantar a tabela `port_status` e realizar a simulação de falhas nas portas via plano de controle. Como trabalho futuro, pretende-se avaliar a possibilidade de integrar o mecanismo de *trigger* e o protocolo *Bidirectional Forwarding Detection* (BFD).

4.2. *Equal Cost Multi Path* (ECMP)

O ECMP é uma técnica de roteamento que encaminha fluxos de pacotes por meio de múltiplos caminhos com o mesmo custo de roteamento. Seu funcionamento começa no plano de controle, onde protocolos de roteamento ou aplicações de rede calculam rotas com o mesmo custo para alcançar as redes de destino. Em seguida, as rotas são instaladas no mecanismo de roteamento ECMP no plano de dados. No presente trabalho, o roteamento ECMP dos pacotes entre as camadas *leaf* e *spine* é baseado no endereço MAC (*Media Access Control*) de destino [Greenberg et al. 2009], o qual contém um *DST_ID* para identificar o switch *leaf* na rede Clos. Para tanto, a tabela ECMP busca a rota de destino *DST_ID* e requisita a *action*. Esta *action*, por sua vez, solicita a execução da função *hash* levando em consideração a quantidade máxima de *links* ECMP armazenada no registro. Então, a função *hash* recebe uma sequência de caracteres de tamanho variável como entrada e calcula um valor de tamanho fixo. Desta forma, a função *hash* recebe os campos do pacote como entrada e calcula a distribuição dos fluxos entre os *links* de saída [Al-Fares et al. 2008]. O ECMP não leva em consideração a carga dos fluxos de pacotes durante a distribuição. Para contornar essa limitação, é possível implementar outros métodos de balanceamento de carga em conjunto com o ECMP [Greenberg et al. 2009], de modo a garantir um desempenho adequado mesmo diante de fluxos gigantes.

Na Figura 2, é apresentada a abstração do mecanismo ECMP no *ingress pipeline* do protótipo BMv2. O plano de controle inicialmente instala a rota 2 e os *links* 0, 1, 2 e 3 no mecanismo ECMP, seguido pela instalação do valor 4 no registro `max_link`, que determina a quantidade máxima de *links* disponíveis no balanceamento de carga ECMP.

[4] `https://groups.google.com/a/lists.p4.org/g/p4-dev/c/n8TsgzHsYnQ/m/HJQzUUFNAAAJ`

Por fim, as portas de saída (P1, P2, P3 e P4) são instaladas nos respectivos registros de encaminhamento `Port_out_0-3` (ilustrado em um único registro). É importante destacar que o roteamento ECMP na topologia *Clos leaf-spine* proposta é baseado em *DST_ID/TAG*. Então, no exemplo citado, a rota 2 alcança o switch *leaf* 2.

Considerando que todas as regras estejam presentes no plano de dados, o switch recebe os pacotes pela interface de entrada e realiza a análise dos seguintes campos: IP de origem, IP de destino, protocolo, porta de origem TCP e porta de destino TCP. Esses campos determinam um fluxo, que é utilizado como parâmetro de entrada na função *hash*. Em seguida, a função calcula o *link* de destino de cada fluxo de pacotes. Neste caso, os *links* estão associados com os registros de encaminhamento `Port_out_(0-3)` da seguinte forma: ($L0 \rightarrow P1$), ($L1 \rightarrow P2$), ($L2 \rightarrow P3$) e ($L3 \rightarrow P4$). Dessa forma, o RESISTING realiza o balanceamento de carga ECMP na rede no cenário sem falhas.

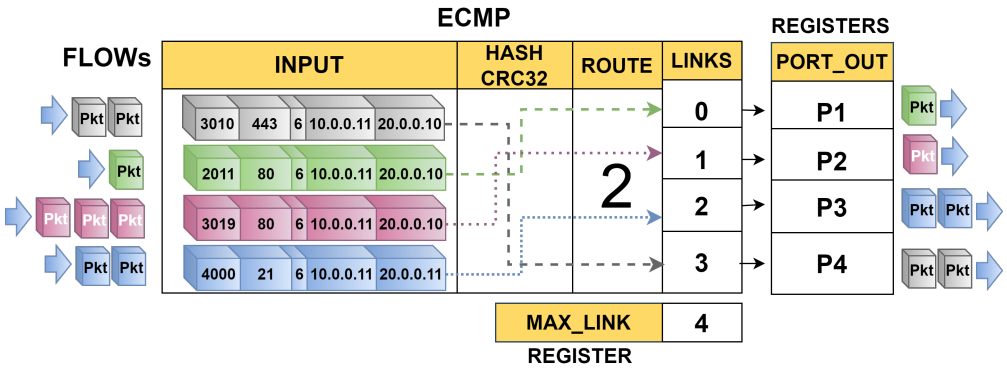


Figura 2: RESISTING: mecanismo ECMP

4.3. FRR-ECMP

Considerando a instalação de todas as regras no plano de dados da arquitetura proposta, os pacotes são submetidos a uma série de etapas nos *pipelines* durante o processamento. Isso inclui a análise do cabeçalho, verificação de integridade, *lookup* e outras operações necessárias para determinar o roteamento dos pacotes. A seguir, ilustramos na Figura 3 os principais passos da recuperação sugerida.

Na Figura 3a, é possível observar o ECMP encaminhando os pacotes 1, 2 e 3 pelo *link* 3 e pela porta 3 de saída antes da detecção da falha pela tabela `port_status`. Após a detecção da falha, a tabela `frr_recirculation` executa uma *action* que realiza uma série de operações para o início da recuperação. Primeiro, o *header* FRR é adicionado a cada pacote, permitindo que o primeiro pacote subsequente à falha, chamado *Pkt 1*, transporte informações de controle, portas e *links* do *ingress* para o *egress pipeline*, e o mesmo se aplica no sentido inverso. Os demais pacotes recebem o *header* FRR para serem recirculados e prevenir o descarte enquanto ocorre a recuperação da falha. Em seguida, o modo de atualização (`update`=1) é habilitado no *header* FRR, informando ao plano de dados que os pacotes a seguir são de recuperação e, portanto, novos componentes P4 devem ser envolvidos. A seguir, a posição do *link* ECMP relacionada com a falha é preenchida no campo `idx_port_down` do FRR, possibilitando que os componentes envolvidos na recuperação saibam qual falha precisa ser recuperada. Para habilitar a recirculação dos pacotes, a *action* utiliza as portas *loopback* reservadas para tal finalidade no caso do Tofino. No protótipo BMv2, a função

recirculate<headers>(hdr) requisita a recirculação. Finalmente, o registro dr_rg_first_failure é solicitado para executar uma operação de contador de pacotes, identificando e marcando no campo first_failure do FRR todos os pacotes envolvidos na falha. É importante ressaltar que, dentre todos os registros aplicados no protótipo do Tofino, o dr_rg_first_failure precisou ser do tipo *DirectRegister*. Esse tipo de registro permite somente um acesso de pacote por vez no registro e garante que o contador escreva um único valor por pacote, assegurando assim a correta identificação do primeiro pacote subsequente à falha.

Na Figura 3b, os três pacotes são submetidos à instrução condicional *IF*, responsável por identificar o *Pkt 1* dentre os demais pacotes e redirecioná-lo para a estrutura FRR de tabelas, *actions* e seis registros fport_out_0-5. Os demais pacotes, por sua vez, são encaminhados para a instrução *else* e recirculados. Os registros fport_out_0-5 armazenam a sequência idêntica de portas dos registros port_out_0-5 do *ingress pipeline* e, a partir da posição do *link* ECMP que sofreu a falha (representado pelo valor do campo idx_port_down no FRR), é gerada uma nova sequência de portas, excluindo a porta com falha. Em seguida, essa nova sequência é escrita nos campos responsáveis pelo transporte no *header* FRR, com o propósito de atualizar os registros port_out_0-5. Além disso, o campo loop (Lo=1) do FRR é habilitado, a fim de acionar a instrução que reinicia o contador do registro dr_rg_first_failure no *ingress pipeline*, indicando o encerramento da recuperação. Por fim, o *Pkt 1* é recirculado para o *ingress pipeline*, juntamente com os demais pacotes.

Na Figura 3c, o *Pkt 1* transporta a nova sequência de portas, permitindo que a estrutura atualize os registros seguindo a lógica de movimentar a porta do último registro para o anterior. O processo se repete até o registro responsável pela porta com falha, sendo que o último registro é preenchido com o valor 255. Neste exemplo, a nova sequência é: ($L1 \rightarrow P1$), ($L2 \rightarrow P2$), ($L3 \rightarrow P4$), ($L5 \rightarrow P6$) e ($L6 \rightarrow P255$). Em seguida, na tabela frr_recirculation, o *Pkt 1* reinicia o contador (x=0) do registro dr_rg_first_failure, permitindo que o RESISTING encerre a recuperação e libere o mecanismo para se recuperar de uma nova falha na sequência ou em outro evento. Após isso, o *Pkt 1* e os demais pacotes seguem para o *egress*. No protótipo BMv2, nesta etapa da recuperação, ocorre um passo adicional: a redução da quantidade máxima de *links* no registro max_link (Figura 2). Assim, o último *link* associado à porta 255 é excluído do cálculo da função *hash*. Atualmente, a função *hash* disponível no Tofino não permite a utilização de registros para controlar a quantidade máxima de *links* via plano de dados. Como solução, preenchemos a última porta com o valor 255 que, ao ser acionado, requisita uma tabela denominada default, responsável por encaminhar os fluxos para uma porta sempre operacional em nossa arquitetura.

Na Figura 3d, os três pacotes são submetidos à instrução condicional *IF*. Nesta etapa, o *Pkt 1* carrega o valor 0 no campo first_failure, o que faz com que ele não atenda mais aos critérios para acessar a estrutura FRR, sendo redirecionado para a instrução *else* juntamente com os demais pacotes. Nessa instrução, de forma semelhante ao processo de codificação do IPv4 no *header Ethernet*, é adicionado o valor 0x800 no campo ether_type do *header* FRR nos pacotes. Além disso, os campos loop e update são desabilitados. Por fim, os pacotes são recirculados pela segunda e última vez para passarem por um novo processo de ECMP, agora utilizando a porta 4 de saída.

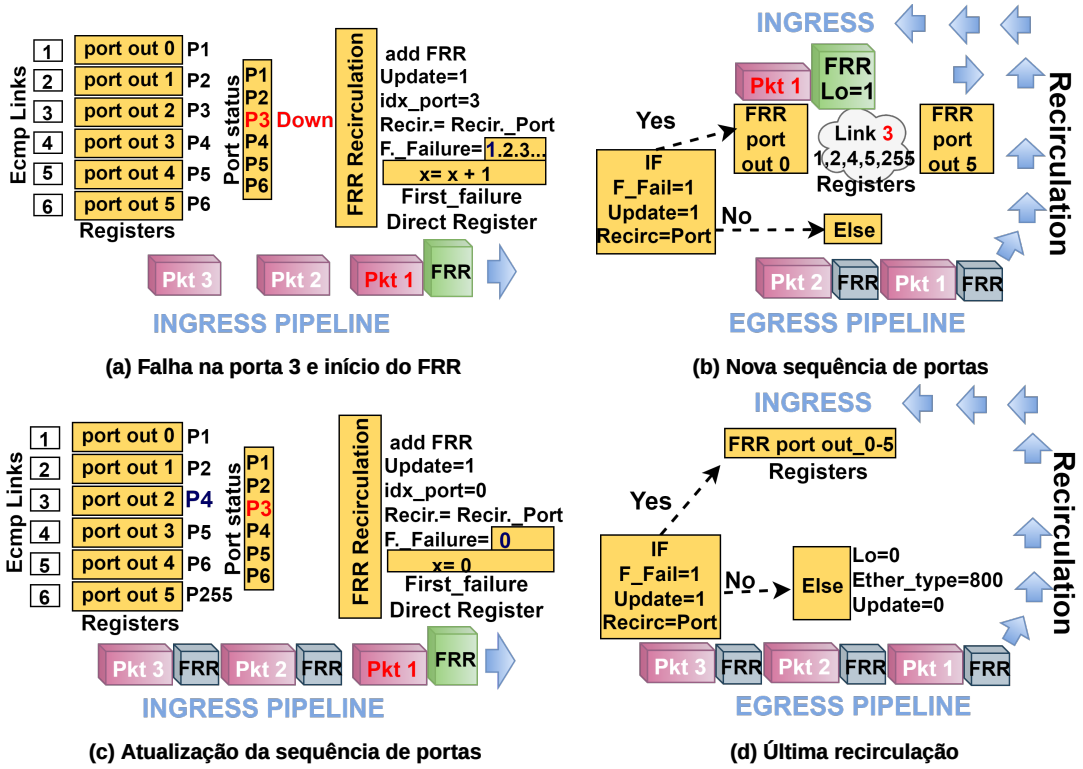


Figura 3: RESISTING: mecanismo de *Fast-reroute*

5. Avaliação experimental

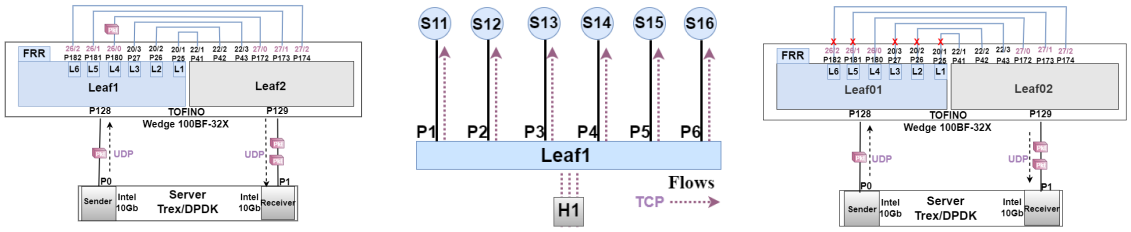
A seguir é apresentado um conjunto de experimentos que visam avaliar dois requisitos fundamentais para um mecanismo FRR-ECMP: o desempenho do processo de recuperação e a resiliência da rede. Os protótipos P4 utilizados nos experimentos estão disponíveis nas versões BMv2[5] e Tofino[6] e os logs dos experimentos estão disponíveis no repositório WGRS[7].

Cenário 01 - Hardware Tofino. A Figura 4a apresenta a topologia criada com um único switch físico emulando dois *leaves*. Para fins de compreensão, denominamos os *leaves* de *leaf* 1 e *leaf* 2. No entanto, é importante ressaltar que essa técnica não deve ser confundida com virtualização. O Tofino executa um único programa P4 com adaptações de regras e componentes para emular dois switches. Neste contexto, os *leaves* foram interconectados fisicamente por meio de seis cabos de 10 Gbps, representando os *uplinks*. O tráfego foi gerado pelo servidor *Trex/DPDK*, equipado com duas placas de rede de 10 Gbps que simulam um *host* em cada uma dessas placas. Os equipamentos utilizados neste cenário foram um *Tofino Wedge 100B-32X* com *Software Develpment Environment* (SDE) *Bf-Sde-9.9.0* e um servidor DPDK com processador *Intel Xeon* de 2,20 GHz, 62 GB de RAM e placa de rede *Intel Ethernet Connection X552 10GbE SFP*.

Cenário 02 - BMv2 Software. Na Figura 4b, é apresentada a topologia *Clos leaf-spine* instanciada no emulador Mininet utilizando o BMv2. Foram criados um *leaf*, seis *spines* e um *host*. Os equipamentos utilizados neste cenário foram um servidor *Intel Xeon* de 2,13 GHz, 25 GB de RAM, SO Linux (Ubuntu 5.4.0-6ubuntu1-16.04.11), Mininet v2.2.1,

[5]RESISTING BMv2 - `https://github.com/danielbl1000/P4-RESISTING-bmv2`
[6]RESISTING Tofino - `https://github.com/danielbl1000/P4-RESISTING-tofino`
[7]Logs - `https://github.com/danielbl1000/WGRS`

BMv2 e *target* `simple_switch1.13.0-33e221fd`.



(a) Cenário 01 (Tofino) **(b) Cenário 02 (BMv2)** **(c) Evento: 5 x Falhas**

Figura 4: Ambiente experimental

5.1. Recirculação de pacotes

Este experimento tem como objetivo avaliar a métrica da quantidade de pacotes submetidos a recirculação durante a ocorrência de uma, três e cinco falhas, a fim de identificar os elementos que possam influenciar a quantidade de pacotes recirculados durante a recuperação e mensurar o impacto da recirculação no fluxo de pacotes. Com esse objetivo, foram executadas avaliações contemplando o cenário 01 (Figura 4a). O experimento consiste em gerar no servidor *Trex/DPDK* a cada 10 segundos um fluxo de pacotes UDP de tamanho grande (1514 B), médio (814 B) e pequeno (114 B), da placa de rede P0 (h1) para a placa de rede P1 (h2), abrangendo quatro diferentes taxas de transmissão: 1) 100Mbps; 2) 1Gbps; 3) 5Gbps; e 4) 9Gbps, e aplicação de 1, 2 e 3 falhas simultâneas.

Para avaliar a quantidade de pacotes submetidos à recirculação durante a ocorrência de uma, três e cinco falhas, foram capturadas amostras para análise de cada fluxo de pacotes na ferramenta *Wireshark*. Além disso, é importante destacar que as falhas foram introduzidas por meio do plano de controle na tabela `port_status` do *leaf* 1, o que causou o acionamento do FRR-ECMP proposto. Por exemplo, o cenário da avaliação do fluxo UDP de 114 bytes com aplicação de 5 falhas enviou 95.339.027 pacotes UDP na taxa de 9 Gbps e apenas 31 pacotes foram recirculados durante o processo de recuperação. A Figura 5 apresenta os resultados consolidados dos experimentos no gráfico. Além disso, disponibilizamos a tabela e os dados[8] do experimento como contribuição adicional.

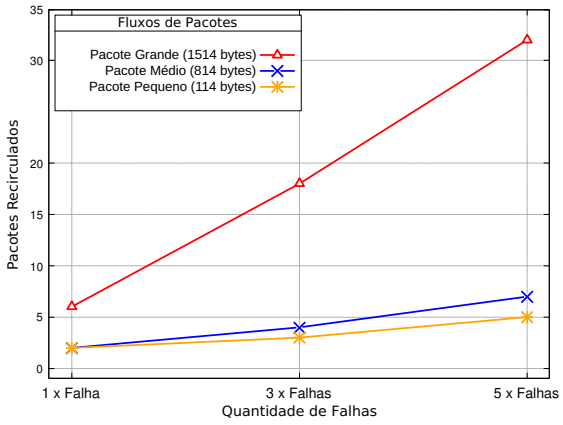


Figura 5: Quantidade de pacotes recirculados por falha.

Os resultados apresentados no gráfico da Figura 5 evidenciaram uma correlação entre o tamanho do pacote e o número de falhas aplicadas, o que resultou na mesma

[8]Ex01-1 - https://github.com/danielbl1000/P4-RESISTING-tofino/tree/main/ex01-1

quantidade de pacotes submetidos à recirculação. É importante destacar que o aumento progressivo na taxa de transmissão de 100Mbps para 1Gbps, 5Gbps e 9Gbps dos fluxos de pacotes durante os experimentos causou um aumento progressivo na quantidade de pacotes que foram gerados. No entanto, não tiveram efeito sobre a quantidade de pacotes recirculados, mantendo-se constante em todos os cenários testados. Por exemplo, na avaliação do pacote de 114 bytes, foram gerados 1 milhão de pacotes com taxa de 100Mbps e 95 milhões de pacotes na avaliação com taxa de 9Gbps. Apesar das diferentes taxas, a quantidade de pacotes recirculados foi a mesma em ambos.

Portanto, neste experimento, observou-se que a quantidade de pacotes recirculados foi baixa em relação ao total de pacotes enviados. Além disso, a quantidade de pacotes recirculados manteve-se estável mesmo com o aumento progressivo da taxa de transmissão, indicando que o RESISTING foi capaz de lidar com a carga adicional gerada sem aumentar a quantidade de pacotes que precisaram ser recirculados. Isto demonstra um promissor comportamento do mecanismo em lidar com situações de alta demanda de tráfego de rede sem comprometer a qualidade do serviço prestado em um cenário real.

5.1.1. Impacto da recirculação de pacotes na latência

Este experimento tem como objetivo avaliar a métrica do impacto da latência na arquitetura de recuperação sugerida. Para isso, a latência dos fluxos de pacotes UDP durante a recuperação de 5 falhas simultâneas é comparada em relação a latência dos fluxos de pacotes em um cenário sem falhas. Este experimento é considerado um complemento ao experimento anterior, no cenário 4a. No entanto, em razão das taxas acima de 1Gbps causarem atraso nos pacotes processados pelo *Wireshark*, foram gerados fluxos de pacotes somente nas taxas de 100Mbps e 1Gbps no experimento. Cada experimento de fluxo de pacotes UDP foi repetido cinco vezes, nos cenários com 5 falhas e sem falhas.

Nos gráficos da Figura 6a e da Figura 6b, é possível observar os resultados consolidados de latência dos tempos mínimo, máximo e médio em microssegundos do experimento. Em ambos os gráficos, a latência calculada durante os eventos de falhas e sem falhas é semelhante em quase todos os tamanhos de fluxos de pacotes, principalmente no tempo médio. A seção 4.3 explica que cada pacote recircula duas vezes por falha. Isto significa que 5 falhas demandam 10 recirculações do pacote no mecanismo sugerido. Os resultados dos cálculos de latência no cenário drástico com 5 falhas não apresentaram atrasos em relação ao cenário sem falha, o que sugere que o custo da recirculação do FRR na "visão" da aplicação não causaria atraso no fluxo de pacotes entre servidores.

5.2. Resiliência de rede

Neste experimento é avaliada a métrica de distribuição dos fluxos de pacotes da arquitetura de recuperação proposta em relação ao PURR nos cenários de uma, duas e três falhas simultâneas. Para calcular a perda dos fluxos de pacotes durante os eventos de falha, foi seguido o conceito de resiliência de rede abordado nos trabalhos [Omer et al. 2009, Figueira and Vasconcelos 2011], que definem a resiliência de rede com base na quantidade de fluxo/tráfego entregue após sofrer redução ou perda de fluxo de pacotes. Na Equação 1a, o valor entregue (VE) de fluxo expressa o nível de resiliência de rede para cada topologia de rede testada. É importante considerar que o termo "topologia de rede" no cálculo representa um cenário que calcula o fluxo/tráfego que foi entregue