InFaRR: Um algoritmo para reroteamento rápido em planos de dados programáveis

Gustavo V. Luz[1], **André L. B. Rocha**[1], **Leandro C. de Almeida**[1], **Fábio L. Verdi**[1]

[1]Departamento de Computação (Dcomp) – Universidade Federal de São Carlos (UFSCar)
Sorocaba – SP – Brasil.

{gustavo.luz,andrebeltrami,leandro.almeida}@estudante.ufscar.br

verdi@ufscar.br

Abstract. *InFaRR (In-network Fast ReRouting) is an algorithm for fast rerouting Implemented in P4, it has four features not jointly found in other recovery mechanisms: loop prevention, pushback, learning and return to the main route. Tests in a Fat-Tree topology with failures in different scenarios showed positive results when compared to state-of-the-art algorithms in the literature. In scenarios with one failure, InFaRR presented less time variation in packet delay when pushback was used, resulting in fewer hops when bypassing the failure. In scenarios with multiple failures, InFaRR successfully rerouted where the others algorithms looped. The unique mechanism for returning to the main route innovated because of the possibility of verifying remote links in a topology without using signaling or announcements about the state of the network.*

Resumo. *O InFaRR (In-network Fast ReRouting) é um algoritmo para reroteamento rápido implementado em P4, apresenta quatro características não encontradas, de maneira conjunta, em outros mecanismos de recuperação: prevenção de loop, pushback, aprendizagem e retorno à rota principal. Os testes em uma topologia Fat-Tree com falhas em diferentes cenários apresentaram resultados positivos quando comparado aos algoritmos do estado da arte da literatura. Nos cenários com uma falha, o InFaRR apresentou menor variação de tempo no atraso dos pacotes quando o pushback foi utilizado, proporcionando menor número de saltos ao contornar a falha. Nos cenários com múltiplas falhas, o InFaRR realizou com sucesso o reroteamento onde os outros algoritmos entraram em loop. O mecanismo único para retorno à rota principal inovou diante da possibilidade de verificação de enlaces remotos em uma estrutura livre de sinalização ou anúncios sobre o estado da rede.*

1. Introdução

Nos diferentes mecanismos de recuperação em redes de computadores destacam-se soluções de reroteamento rápido que são caracterizadas pela capacidade local do comutador discernir pelo uso da política de recuperação que foi previamente configurada. O reroteamento rápido, portanto, é uma ação tomada sem a participação do plano de controle e sem a dependência da sinalização de outros elementos externos como, por exemplo, anúncios de novas rotas, *heartbbeat* ou do estouro do limite de tempo de uma sessão do protocolo de roteamento dinâmico [Kiranmai et al. 2014].

As soluções de reroteamento sustentadas pelo plano de controle centralizado em Redes Definidas por Software (do inglês, *Software Defined Networking* - SDN) apresentam um mecanismo de recuperação mais lento quando comparado ao reroteamento rápido no plano de dados [Kiranmai et al. 2014]. O plano de controle centralizado tem um atraso para iniciar a recuperação, pois é preciso detectar uma remota, contrário a proposta de reroteamento rápido [Sgambelluri et al. 2013]. No entanto, os mecanismos de recuperação executados no plano de dados provêm a agilidade e velocidade de recuperação durante períodos de falhas, pois não há a necessidade de consultar o plano de controle ou dependência de algum tipo de sinalização de agentes externos [Chiesa et al. 2020].

Neste contexto, o InFaRR (*In-network Fast Rerouting*) se beneficia do plano de dados programável e propõe um algoritmo de reroteamento rápido capaz de contornar múltiplas falhas. O InFaRR foi implementado na linguagem P4 (*Programming Protocol-Independent Packet Processors* [Bosshart et al. 2014]) e comparado com as principais soluções encontradas na literatura atualmente. O InFaRR possui quatro características não encontradas, de maneira conjunta, em outros mecanismos: prevenção de *loops* na rede, *pushback*, mecanismo de aprendizagem e retorno à rota principal.

O controle de *loop* proposto no InFaRR age antes do tradicional TTL (*Time-To-Live*) evitando que pacotes fiquem circulando na rede. O mecanismo de *pushback*, também encontrado em [Liu et al. 2013], devolve os pacotes ao comutador anterior em caso onde não há rotas alternativas a partir do ponto da falha. O mecanismo de aprendizagem habilita a descoberta de quais rotas estão com falhas, evitando assim o envio de tráfego através delas. Finalmente, o retorno à rota principal é um mecanismo adicionado ao InFaRR capaz de detectar se a rota original se recuperou da falha e então retomar o envio dos fluxos afetados através de sua rota primária.

Avaliou-se o InFaRR utilizando uma topologia de rede *datacenter Fat-Tree* com k=4 em ambiente emulado no Mininet. Diferentes cenários de falhas foram exercitados comparando o algoritmo proposto com trabalhos correlatos e minimamente adaptados para o funcionamento em P4 [Chiesa et al. 2019, Sedar et al. 2018]. A comparação dos algoritmos foi realizada através da avaliação do sucesso a recuperação diante das falhas, ocorrência de perda de pacotes durante o processo de recuperação, tempo de transmissão dos pacotes e número de saltos na rede após a recuperação da falha.

Portanto, as principais contribuições deste trabalho são: 1) projeto e implementação de um algoritmo de reroteamento rápido para redes programáveis com a capacidade de recuperação em caso de múltiplas falhas; 2) execução de uma prova de conceito em ambiente virtual sob a ótica de uma topologia *Fat-Tree* para análise, interpretação e comparação com o estado da arte e; 3) disponibilização de um *dataset* coletado durante a prova de conceito para fins de replicação do estudo e futuras comparações. Este trabalho está organizado conforme descrito a seguir. A Seção 2 apresenta a fundamentação teórica. Os principais trabalhos relacionados que representam o estado da arte sobre o tema são apresentados na Seção 3. A Seção 4 apresenta as características do algoritmo InFaRR e detalhes de funcionamento. Na Seção 5, apresentamos os principais resultados quantitativos em relação aos diferentes cenários avaliados. A Seção 6, discute as conclusões e as propostas para trabalhos futuros.

2. Fundamentação Teórica

Dividimos esta seção em quatro subseções: classes de serviços em redes de computadores, métodos de recuperação, processo de recuperação e topologia *Fat-Tree*.

2.1. Qualidade de Serviços em redes de computadores

O Setor de Normatização das Telecomunicações (*Telecommunication Standardization Sector*, ITU-T) definiu parâmetros relacionados ao desempenho da rede IP que quantificam os níveis requeridos de Qualidade de Serviços (QoS) para as diferentes classes de serviços existentes, sendo eles: Atraso máximo de Transferência de Pacotes IP (*IP Packet Transfer Delay*, IPTD), Variação de atraso de Pacotes IP (*IP Delay Variation*, IPDV) e Taxa de perda de Pacotes IP (*IP Packet Loss Ratio*, IPLR) e Taxa de Erro de pacote (*IP packet Error Ratio*, IPER) [Chodorek 2002, ITU-T 2002].

As classes de serviços definidas como tempo real ou interativas são sensíveis ao tempo de transmissão (IPTD) e requerem caminhos de transmissão fim-a-fim inferior a meio segundo (400 ms), exigem uma variação de latência (IPDV) inferior a 50 ms e não permitem perdas de pacotes para um bom funcionamento [Stankiewicz et al. 2011]. Portanto, um mecanismo de recuperação com o intuito de atender estes requisitos descritos deve prover um reroteamento dentro do limite de IPDV e o novo caminho de contingência não deve ultrapassar o limite do IPTD requerido.

2.2. Métodos de Recuperação

Os métodos de recuperação podem ser classificados por diversos critérios [Cholda et al. 2007]. Três destes critérios possuem extrema relevância para este trabalho. O primeiro está associado ao método de configuração do caminho *backup* que pode ser pré-configurado ou reativo. O método pré-configurado possibilita uma recuperação mais rápida pois a tabela de roteamento secundária já esta pré-definida para ser utilizada quando necessário no plano de dados. No método reativo, um processo de recuperação é iniciado no plano de controle logo quando uma falha é encontrada. Todo o processamento de reconstrução da tabela de roteamento é feito pelo plano de controle e distribuído para o plano de dados.

O segundo critério está associado ao escopo da recuperação que pode ser global, local ou de segmento. Na política global é oferecida uma proteção a todos os elementos do caminho (fim-a-fim). A política local assegura que existe uma forma de contornar pontualmente o enlace com problema. Neste cenário, o tráfego sofrerá um ajuste de encaminhamento somente a partir do ponto onde a falha foi encontrada. Por fim, a política de recuperação por segmento oferece proteção a segmentos específicos envolvendo comutadores e enlaces e proverá contorno pontual ao trecho com problemas [Cholda et al. 2007].

O último critério trata as características relacionadas ao domínio de operação e recuperação, sendo este item diretamente associado as fronteiras rede. O atributo domínio de operação pode ser observado na autonomia de roteamento interno de cada *pod* nas redes *Fat-Tree*, Assim o domínio de recuperação é independente quando o processo de reroteamento ocorre exclusivamente no mesmo *pod* e dependente se existe dependência de comutadores de outros *pods*.

O InFaRR adota um método de recuperação pré-configurado que o classifica como um algoritmo de reroteamento rápido. O escopo de recuperação adotado no InFaRR é

por segmento. O domínio de operação será interno ao *pod* trazendo independência ao processo de reroteamento, evitando que, no momento da recuperação da falha, pacotes trafeguem por equipamentos desnecessários.

2.3. Processo de Recuperação

Inicialmente é importante definir o conceito básico referente aos tipos de tabela de roteamento existentes, em que a "tabela de roteamento principal" refere-se ao processo primário utilizado pelo comutador para definir a porta de saída para encaminhamento dos pacotes. Enquanto que a "tabela de roteamento secundário" ou *backup* será utilizada somente se a porta sugerida pela tabela de roteamento principal estiver com problemas.

A Figura 1 descreve as etapas existentes relacionadas ao processo de detecção e recuperação de falhas. Na etapa inicial $E0$, têm-se o funcionamento da rede com tráfego enviado através de um plano de encaminhamento ótimo determinado pelo plano de controle. A detecção da falha, que acontece na etapa $E1$, define o momento em que a porta foi para o estado de *down*, dando início ao processo de recuperação na etapa $E2$. A etapa $E3$, a rede passa a funcionar por um caminho de alternativo até que o mecanismo de controle possa restabelecer a conectividade, que acontece na etapa $E4$. O gatilho para mudança estado para *up* da etapa anterior, inicia o processo de atualização das tabelas de roteamento, etapa $E5$, retornando ao plano de encaminhamento inicial em $E6$.

O mecanismo de controle que ocorre nas etapas $E1$ e $E4$ não são escopo deste trabalho. Tais etapas estão associadas aos mecanismos de sinalização no meio físico, perda de pacotes ou outros fatores que levam a penalização da porta com estado *down*; ou admitem uma porta com estado *up*. Todavia, a mudança do estado da porta para *down* ou *up* é o gatilho para início do algoritmo de recuperação. Para fins de avaliação neste artigo, o estado da porta foi imposto arbitrariamente na simulação de falhas e respectivo processo de recuperação.

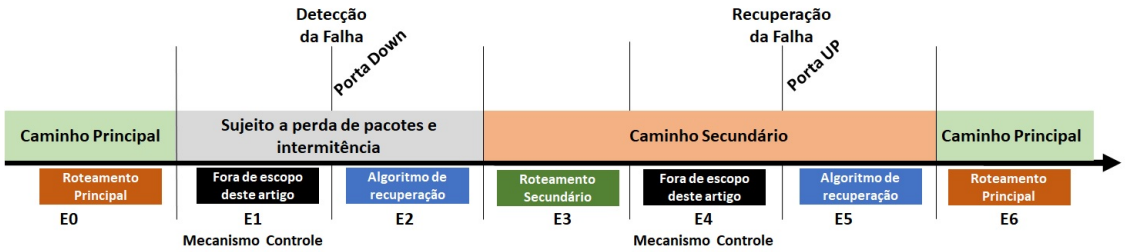


Figura 1. Etapas do processo de detecção e recuperação de falhas.

Este trabalho tem entre seus objetivos reduzir o tempo de recuperação, que é a somada da duração das etapas $E1$ e $E2$. Neste intervalo, de $E1$ a $E2$, conjectura-se que pode ocorrer perda de pacotes, portanto quanto menor for este tempo, mais rápido será o processo de convergência [Cholda et al. 2007]. Visto que a etapa $E1$ não é escopo deste trabalho, observa-se que é importante avaliar o duração da etapa $E2$, tendo em vista que ele pode sofrer influência direta do algoritmo de convergência utilizado. Realizar as medições relacionadas ao desempenho da rede durante o funcionamento normal no momento $E0$ e compará-las com o funcionamento no momento de contingência $E3$, possibilitará avaliar a eficácia do algoritmo de contingência. Conjectura-se que mesmo no

processo de convergência, as medições de desempenho permaneçam dentro dos limites apresentados na Seção 2.1.

As condições de rede no momento de contingência serão equivalentes ou piores se comparadas com o estado de funcionamento normal, tomando como premissa que o plano de controle ofereceu a rota ótima para o funcionamento normal na etapa $E0$. Portanto, as condições de rede serão equivalentes caso a estrutura de contingência possua as mesmas condições de utilização e latência de rede, encontradas em topologias com proteção dedicada (1+1). Em cenários de proteção com recursos compartilhados, o plano de encaminhamento contingente estará sujeito a utilização de trechos com maior ocupação, perda de pacotes, maior latência ou caminhos com maior número de saltos que impactarão diretamente os parâmetros de desempenho do fluxo contingenciado. Para efeito de estudo, consideraremos que as conexões que farão parte do processo de redundância serão dedicadas e apresentarão as mesmas condições de rede (latência e utilização) de forma que não será necessário considerar nenhum processo de priorização ou descarte de pacotes neste estudo.

2.4. Proteções físicas em redes *Fat-Tree*

As redes *Fat-Tree* possuem três camadas denominadas Núcleo, Agregação e Topo de Rack (em inglês, *CORE, Aggregation e Top of Rack* - ToR) as quais possuem funções distintas. O número de portas presentes nos equipamentos dimensiona o tamanho da rede e sua capacidade de tolerância a falhas. Seja *k* o número de portas presentes em um equipamento do Núcleo, *k* será o número de conexões existentes para cada um dos diferentes dispositivos de Agregação. O conjunto de equipamentos de Agregação e de topo de rede interconectados são denominados *pod*s [Leiserson 1985].

As redes *Fat-Tree* possuem tolerância a falhas de *k-1* nos enlaces entre os *pod*s e os equipamentos do Núcleo. Os equipamentos posicionados no Nũcleo possuem como função prover roteamento entre os *pods*. Cada dispositivo do Núcleo possui um enlace a pelo menos um equipamento de Agregação de um *pod* diferente, conforme Figura 3. Para concepção deste estudo, estaremos utilizando o modelo da *Standard Fat-Tree* [Liu et al. 2013] com k=4, no qual os equipamentos do Núcleo *S1CORE* e *S2CORE* possuem enlaces diretos com os equipamentos de Agregação *A1P1, A1P2, A1P3 e A1P4*, enquanto os equipamentos *S3CORE* e *S4CORE* possuem enlace diretos com *A2P1*, *A2P2*, *A3P2* e *A4P2*, conforme Figura 3. A variação de K não reflete no aumento do numero de camadas na topologia de rede, reflete apenas no número de conexões e equipamentos.

3. Trabalhos relacionados

Os protocolos de roteamento dinâmico estão entre as ferramentas utilizadas para prover tolerância a falhas. Com isso, quando uma conexão entre equipamentos apresenta falhas, a rede pode convergir para outro caminho. Protocolos tradicionais como o *Border Gateway Protocol* ou *Open Shortest Part First* podem levar dezenas de segundos para convergirem, impactando em fluxos sensíveis a atraso e perda de pacotes [Chiesa et al. 2020].

Neste artigo foram selecionados os três principais trabalhos da literatura considerando o estado-da-arte no contexto de recuperação de falhas em redes programáveis. Tais trabalhos foram adaptados para a linguagem P4 e utilizados para fins de comparação com o proposto neste estudo. Foram considerados um algoritmo tradicional com funcionamento no plano de controle e outros dois algoritmos de reroteamento rápido.

Em uma arquitetura com o plano de controle centralizado é possível oferecer um escopo de recuperação global, mecanismo de configuração reativo e domínio de operação independente. Uma das atribuições do plano de controle é a constante avaliação das condições a cada um dos equipamentos de rede (*pooling* de gerenciamento) e sempre que necessário atualizar as tabelas de roteamento dos comutadores com a melhor opção possível de encaminhamento (otimizado) [ONF 2013] [Sgambelluri et al. 2013].

No trabalho [Nikolaevskiy 2016] intitulado de "*Scalability and Resiliency of Static Routing*", apresenta um amplo estudo sobre a viabilidade da implementação de tolerância a falhas através do uso de roteamento estático e suas variações. Para o desenvolvimento deste estudo, foi abordado a implementação mais básica de roteamento estático apresentado, que consiste em prover uma tabela de roteamento principal e uma tabela de roteamento secundária a ser utilizada na ocorrência de falhas; ambas previamente configuradas pelo plano de controle. Esta abordagem oferece escopo de recuperação local e quando implementada no comutador de Núcleo necessita de um domínio de convergência de outro *pod* para encontrar um caminho alternativo até seu destino [Chiesa et al. 2019].

O trabalho de [Sedar et al. 2018] implementa no plano de dados programável sem a necessidade do uso de cabeçalhos adicionais. O algoritmo Rotor uma vez que a porta de saída esteja com problemas o pacote será submetido para próxima porta ativa disponível. O escopo de recuperação é local e sua implementação no dispositivo de Núcleo requer que o pacote seja roteado por outro *pod*. O método de configuração foi definido previamente, visto que não existem consultas ao plano de controle. Muito embora não exista uma tabela de roteamento *backup* pré-configurada, o algoritmo assume a responsabilidade de descobrir um caminho alternativo até seu destino [Leiserson 1985]. A Tabela 1 resume as principais características analisadas dos trabalhos relacionados em relação ao InFaRR.

Algoritmos	**Plano de Controle**	**Estático**	**Rotor**	**InFaRR**
Tipo de funcionamento	Reroteamento	Reroteamento rápido	Reroteamento rápido	Reroteamento rápido
Camada de funcionamento	Plano de controle	Plano de dados	Plano de dados	Plano de dados
Métodos de configuração	Reativo	Pré-configurado	Pré-configurado	Pré-configurado
Escopo de recuperação	Global	Local	Local	Segmento
Domínio de recuperação	Independente	Dependente	Dependente	Independente
Plano de encaminhamento	Otimizado	Não	Não	Otimizado
Pooling de gerenciamento	Sim	Não	Não	Não

Tabela 1. Resumo algoritmos selecionados.

4. *In-network Fast ReRouting* - InFaRR

O InFaRR é um algoritmo que apresenta quatro características essenciais não encontradas, de maneira conjunta, em outros estudos: 1) *pushback*; 2) prevenção de *loop* na rede; 3) aprendizagem; e 4) retorno à rota principal. As funcionalidades do reroteamento rápido tem início assim que o comutador detecta ser inviável o encaminhamento de pacotes através da porta indicada pela tabela de roteamento principal. Ao perceber que a porta de saída indicada esta no estado de *down* ou em *loop*, o InFaRR realiza o processo de roteamento a partir de uma tabela de secundária. Caso a porta de saída também esteja inviável para uso, o InFaRR adota o mecanismo de *pushback*, devolvendo o pacote para o comutador antecessor.

O mecanismo de *pushback* ao devolver o pacote causa um *loop* no comutador antecessor. Para resolver este problema adotamos um mecanismo de prevenção de *loop*

que verifica se a porta de entrada do pacote é a mesma de saída, o que inviabilizaria o encaminhamento. O mecanismo de aprendizagem quando detectado o *loop* armazena a porta de saída como inativa para este fluxo. Desta forma o mecanismo de aprendizagem possibilita que os pacotes subsequentes deste fluxo não sejam encaminhados por esta porta.

O InFaRR também possui um mecanismo de retorno à rota principal, que é executado quando o reroteamento acontece devido ao mecanismo de aprendizado. Neste caso, o algoritmo duplica um pacote de dados e o envia para a rota principal (além de enviar o pacote original via rota secundária). Caso o pacote duplicado não retorne dentro do período de um *Round Trip Time* (RTT), o InFaRR assume que a rota principal foi restabelecida. A partir deste momento, os pacotes poderão ser enviados pela rota principal. Caso o pacote retorne, conclui-se que a falha ainda persiste. Embora seja viável em redes programáveis o ajuste adaptativo da janela de espera nesta implementação definimos que a espera de 300ms.

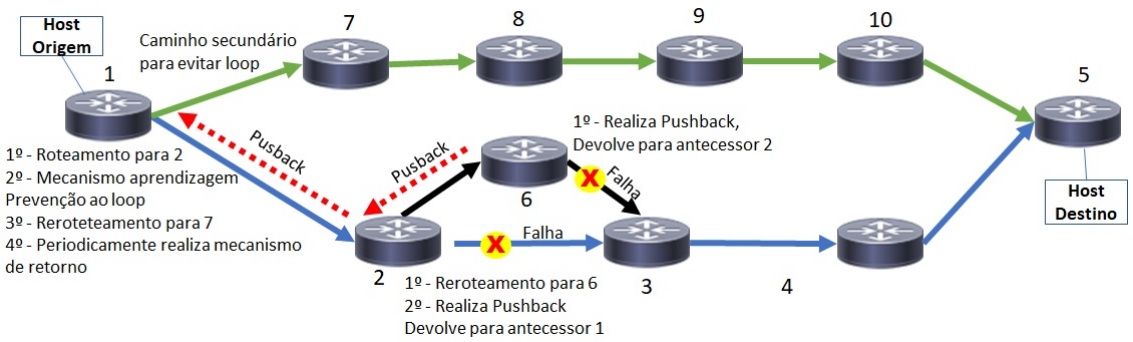


Figura 2. Características de *pushback* e aprendizado.

A Figura 2 representa um cenário com duas falhas e a usaremos para ilustrar o mecanismo de *pushback* nos comutadores 6 e 2, prevenção de *loop* nos comutadores 2 e 1, mecanismo de aprendizagem no comutador 1 e mecanismo de retorno à rota principal no comutador 1. O *host* de origem envia pacotes através do caminho principal destacado em azul que possui falhas. Neste caso, o InFaRR deve ajustar o roteamento para que as falhas sejam contornadas, através do caminho secundário, destacado em verde.

No estágio inicial o comutador 2 detecta a falha no enlace 2-3 e realiza o reroteamento via comutador 6. Porém, há também uma falha no enlace 6-3 e, neste caso, o InFaRR aciona o *pushback* devolvendo o pacote para o seu antecessor (comutador 2). É observado que o comutador 2 não possui opções viáveis de encaminhamento pois possui uma porta *down* e outra em *loop*. Neste caso, o mecanismo de *pushback* encaminha o pacote ao seu antecessor (comutador 1).

O comutador 1, ao receber de volta o pacote encaminhado pelo comutador 2, também detecta o *loop*, disparando assim o reroteamento rápido via comutador 7. Neste momento, o comutador 1 utiliza o mecanismo de aprendizagem para armazenar a informação de que o caminho através do comutador 2 está indisponível, ou seja, os pacotes subsequentes deste fluxo deverão ser encaminhados via comutador 7. Após isso, o comutador 1 inicia um mecanismo de consulta (*probing*) para verificar a disponibilidade de se encaminhar pacotes via o comutador 2. Neste caso, os pacotes originais (encami-

nhados para o comutador 7) deverão ser duplicados, tendo suas cópias enviadas para a interface de saída que o conecta ao comutador 2. Caso as falhas não tenham sido corrigidas no comutador 2, o pacote de cópia retornará ao comutador 1 conforme processo de *pushback* já descrito. Caso o pacote de cópia não retorne, assume-se que as falhas foram recuperadas e esta rota volta a ser a principal para o processo de roteamento.

Observar-se uma leve diferença entre a execução do InFaRR no comutador 2 e no comutador 1. No comutador 2, não ocorreu atuação do mecanismo de aprendizagem, visto que o mecanismo de *pushback* enviou o pacote de volta ao comutador 1. Neste caso, o comutador 1 se beneficiou do mecanismo de aprendizagem, visto que a rota secundária estava operacional. Sendo assim, observa-se que o mecanismo de aprendizagem só será utilizado quando a rota secundária estiver operacional, o que não ocorre com o comutador 2 para este cenário (forçado ao mecanismo de *pushback*). O mecanismos propostos pelo InFaRR funciona de forma autônoma em cada comutador e podem ocorrer simultaneamente em diferentes comutadores da rede quando necessário.

Algoritmo 1: Pseudocódigo - InFaRR

```
1  hashFluxo ← hash(cabecalhoIP, TCP);                          ▷ gera hash
2  if (porta_antecessor[var_hashfluxo]= vazio) then
3  |  porta_antecessor[hashFluxo] ← porta_entrada
4  end
5  Consulta Tabela_Roteamento_Principal                         ▷ retorna porta_saída;
6  if (porta_entrada = porta_saida) then
7  |  fluxo_ativo[hashFluxo, porta_saida] ← False              ▷ Antiloop
8  end
9  if (porta_saida = Down) ou (fluxo_ativo[hashFluxo,porta_saida] = False) then
10 |  Consulta Tabela_Roteamento_Secundário                    ▷ retorna porta_saída
11 |  if (porta_entrada = porta_saida) then
12 |  |  fluxo_ativo[hashFluxo, porta_saida] ← False          ▷ Antiloop
13 |  end
14 |  if (porta_saida= Down) ou (fluxo_ativo[hashFluxo,porta_saida] = False) then
15 |  |  porta_saida ← porta_antecessor[hashFluxo]            ▷ Pushback
16 |  end
17 end
18 Encaminha pacote através da porta de saída
```

O pseudocódigo do InFaRR está descrito no Algoritmo 1. Inicialmente, o algoritmo gera um *hash* (linha 1) com base no cabeçalho TCP/IP[1]. O valor do *hash* computado é utilizado para indexação de vetores a fim de armazenar as informações associadas por fluxo. A primeira informação a ser armazenada por fluxo é a porta de saída, que dá acesso ao comutador antecessor (linhas 2-4), informação essencial para o mecanismo de *pushback*. Na linha 5, ocorre a consulta à tabela de roteamento principal. Nas linhas 6-8 é verificado se aconteceu algum *pushback*, ou seja, a porta de entrada é igual a porta de saída. Neste caso, o *hash* e a porta de saída são usados como índice no vetor fluxo_ativo para armazenar o estado como inativo (valor *False*). Posteriormente, na linha 9, o InFaRR realiza duas verificações: 1) se a porta de saída está fisicamente *up/down*[2]; 2) se a porta

[1]Neste trabalho, a operação de *hash* utiliza os endereços IP de origem e destino, portas de origem e destino. Qualquer outra combinação de campos também pode ser utilizada.

[2]O Mecanismo de controle sobre o estado físico *up/down* das portas não é escopo deste trabalho.

de saída está em *loop* (vetor fluxo_ativo). Se a porta estiver em estado *up* e não estiver em *loop*, o algoritmo envia o pacote. Caso contrário, o reroteamento é iniciado a partir da tabela de roteamento secundária (linha 10) e as duas verificações são realizadas novamente antes do encaminhamento do pacote (linhas 11-16). Entretanto, caso a porta de saída ainda esteja indisponível (estado de *down* ou em *loop*), o mecanismo de *pushback* (linha 15) é disparado. Finalmente o comutador encaminha o pacote para porta de saída (linha 18). As seguintes estruturas de vetores foram definidas:

- *vetores_auxiliares_pushbak*: armazenam informações para realização do mecanismo de *pushback*, cada posição ocupando 96 bits de memória;
- *vetores_auxiliares_aprendizagem_loop*: armazenam informações utilizadas pelo mecanismo de aprendizagem e prevenção ao *loop* para o fluxo, cada posição ocupando 48 bits de memória;
- *vetores_auxiliares_recuperação*: armazenam informações utilizadas pelo mecanismo de retorno à rota principal, cada posição ocupando 96 bits de memória;

As estruturas de vetores alocadas para cada fluxo totalizam 288 bits (36 bytes). Assim, foram alocados 1,41 Mbytes de memória para o tratamento de 40960 fluxos diferentes. A escalabilidade dos números de fluxos tratados está diretamente associada a capacidade de memória disponível. Nesta implementação, não realizamos nenhum processo de otimização, agrupamento e seleção de quais fluxos seriam protegidos pelo algoritmo InFaRR, seja através dos hosts envolvidos (endereços IPs), tipo de serviço utilizado (portas TCP) ou qualquer outro mecanismo de classificação de QoS.

Uma característica tipicamente encontrada em *datacenters* se refere ao fato de que o tráfego de um domínio de recuperação utilize outros *pods* durante o processo de recuperação [Liu et al. 2013]. O InFaRR não possui esta característica, visto que os comutadores do Núcleo estão configurados para executarem o mecanismo de *pushback* quando necessitarem recorrer ao reroteamento. Essa estratégia possibilita a recuperação em cenários com múltiplas falhas com topologia *Fat-tree*.

Para exemplificar esta característica, a Figura 3 descreve a comunicação entre os sistemas finais H1P1 e H1P3, na qual as linhas em vermelho tracejadas indicam que o mecanismo de *pushback* se repete quatro vezes. O pacote quando processado por S1CORE encontra um enlace com falhas até A1P3 e recorre ao mecanismo de *pushback* devolvendo o pacote para A1P1(1). Uma vez que o caminho principal de A1P1 não alcança o destino, realiza-se o reroteamento através da tabela de roteamento secundário. Como S2CORE não possui conexão ativa para A1P3 ocorre um novo *pushback* (2). Como o roteamento principal e secundário de A1P1 estão inativos, ocorre um *pushback* para T1P1 (3). O comutador T1P1 ao receber a devolução do pacote faz o reroteamento para A2P1 que roteia para s3CORE, diante a impossibilidade de conexão direta com A2P3, ocorre o *pushback* (4), finalmente o reroteamento para s4CORE possibilita o acesso ao POD3 através A2P3. Neste caso, o mecanismo de aprendizagem proposto pelo InFaRR atuará nos comutadores T1P1 e A2P1 para que o caminho alternativo (linhas em roxo) seja utilizado pelos pacotes subsequentes limitando assim o esforço de busca de um caminho alcançável ao destino possibilitando a otimização do encaminhamento dos pacotes.

O InFaRR utiliza uma tabela de roteamento principal e outra tabela secundária para prover redundância, desta forma o aumento do numero de enlaces no comutador não é explorado diretamente pelo algoritmo. Na ocorrência de uma falha todo o fluxo de

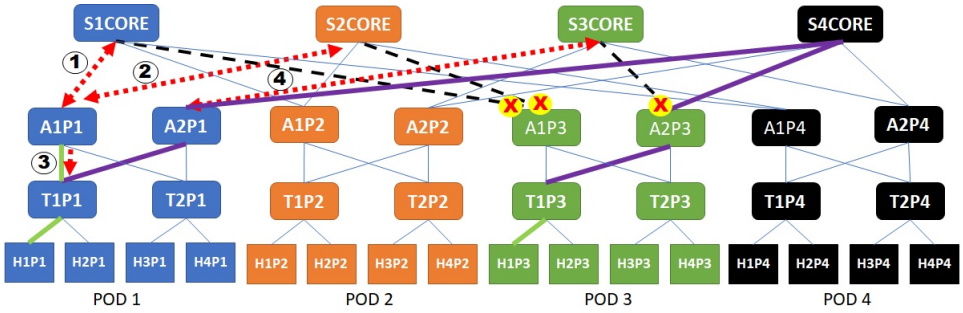


Figura 3. Processo de Recuperação InFaRR - Cenário com 3 falhas.

dados converge de imediato para o enlace secundário ou para o mecanismo de *pushback* permanecendo assim até a recuperação do enlace primário ou a intervenção do Plano de Controle que pode ajustar as tabelas de roteamento sempre que necessário. O Plano de Controle pode promover o enlace secundário para primário e eleger um enlace para fazer parte da tabela de roteamento secundária. Na topologia de rede *Fat-tree* o InFaRR habilita tolerância a falhas de até 3 falhas de conexões a cada POD para K=3, o crescimento de k não habilita maior capacidade de tolerância a falhas, já que neste caso o limitador para o processo de recuperação é o número de camadas.

5. Experimentação e avaliação

O algoritmo proposto foi implementado em linguagem P4 e avaliado em ambiente Mininet com suporte ao *software switch* Bmv2[3]. Os códigos fontes dos algoritmos apresentados neste artigo podem ser encontrados em repositório público[4].

O experimento foi executado contemplando cenários de roteamento existentes entre os *hosts* de *pods* distintos com variações nos tamanhos dos pacotes. Cada execução foi realizada de maneira independente para cada um dos quatro algoritmos mencionados na Seção 3. Todos os algoritmos foram submetidos aos seguintes cenários:

1. **Cenário com 1 falha[5]:** Simulou uma falha no enlace principal ao *pod* de destino.
2. **Cenário com 2 falhas:** Simulou uma falha no enlace principal e outra no enlace secundário ao *pod* de destino.
3. **Cenário com 3 falhas:** Simulou falhas em três enlaces sequenciais no processo de roteamento (principal, secundário e terciário) ao *pod* de destino.

Em cada execução, o *host* de origem gerou um fluxo de pacotes através de uma aplicação escrita em linguagem *Python*. Os pacotes recebidos no *host* de destino foram utilizados para a avaliar a operação dos algoritmos. Neste caso, foi possível obter as seguintes medições: 1) perda de pacotes durante o processo de recuperação; 2) IPTD; e 3) número de saltos para reroteamento. A perda de pacotes foi obtida através da diferença entre o número de pacotes enviados e recebidos. O IPDT (ver Seção 2.1) foi computado através da diferença entre o instante de tempo (*timestamp*) em que pacote entra na rede, com o instante de tempo (*timestamp*) em que o pacote sai da rede. Já o número de saltos foi medido através do campo *Time to Live* (TTL) do cabeçalho IPv4.

[3]https://github.com/p4lang/behavioral-model.
[4]https://github.com/gvenancioluz/SBRC2022.
[5]As falhas sempre ocorrem entre os equipamentos de Núcleo e Agregação do *pod* de destino.

Os algoritmos do estado da arte foram implementados, com algumas adaptações, em linguagem P4 para fins de comparação com a proposta deste estudo. O algoritmo de roteamento estático [Nikolaevskiy 2016] utilizou uma tabela de roteamento principal e uma tabela de roteamento secundária. Já o algoritmo Rotor [Sedar et al. 2018] teve como restrição o envio de pacotes pela porta de entrada.

O algoritmo que simulou o plano de controle centralizado realizou *pooling* em cada um dos comutadores P4 com o objetivo de detectar a ocorrência de falhas na rede [ONF 2013]. Neste caso, observou-se que o tempo médio para executas as consultas em todos os comutadores foi superior a 3 segundos, similar a outros trabalhos encontrados na literatura [Chiesa et al. 2019]. Sendo assim, para fins de padronização e coleta dos dados, fixamos as consultas em um intervalo de 5 segundos.

Algoritmos /Cenário de Testes	**Plano de Controle**	**Estático**	**Rotor**	**InFaRR**
Cenário sem falhas				
Número de saltos - Caminho Normal ($E0$)	5	5	5	5
Cenário 1 Falha				
Número de saltos - Processo de Recuperação ($E2$)	Perde Pacotes	7	7	7
Número de saltos - Caminho redundante ($E3$)	5	7	7	5
Atende critérios ITU	Não	Sim	Sim	Sim
Cenário 2 Falhas				
Número de saltos - Processo de Recuperação ($E2$)	Perde Pacotes	Não Recupera	Não Recupera	11
Número de saltos - Caminho redundante ($E3$)	5	Não Recupera	Não Recupera	5
Atende critérios ITU	Não	Não	Não	Sim
Cenário 3 Falhas				
Número de saltos - Processo de Recuperação ($E2$)	Perde Pacotes	Não Recupera	Não Recupera	13
Número de saltos - Caminho redundante ($E3$)	5	Não Recupera	Não Recupera	5
Atende critérios ITU	Não	Não	Não	Sim

Tabela 2. Avaliação consolidada dos algoritmos estudados.

A Tabela 2 consolida os resultados obtidos para todas as execuções. Como descrito conceitualmente na Seção 3, o algoritmo que utiliza o plano de controle apresenta um atraso que ocasiona perda de pacotes, inviabilizando sua utilização para fluxos sensíveis a perda de pacotes. A Figura 5(b) demonstra a ocorrência da perda de pacotes durante o processo de recuperação. Observa-se ainda que, independente do cenário de falhas e do tamanho dos pacotes, ocorreram falhas e não existem diferença estatística para a perda de pacotes diante os diferentes cenários testados. O algoritmo InFaRR não apresentou perda de pacotes durante o processo de recuperação em nenhum cenário.

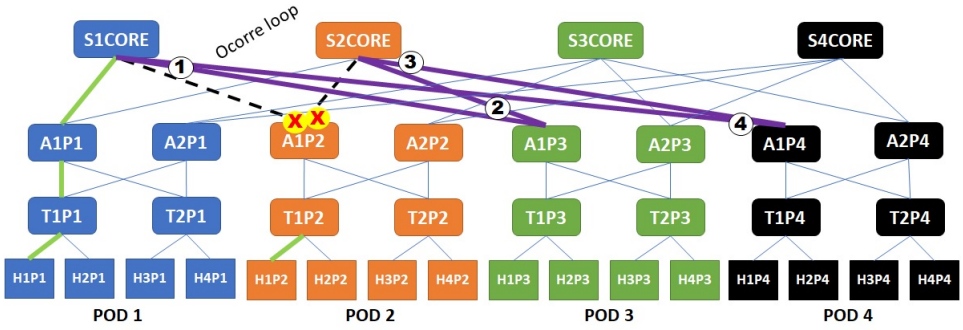
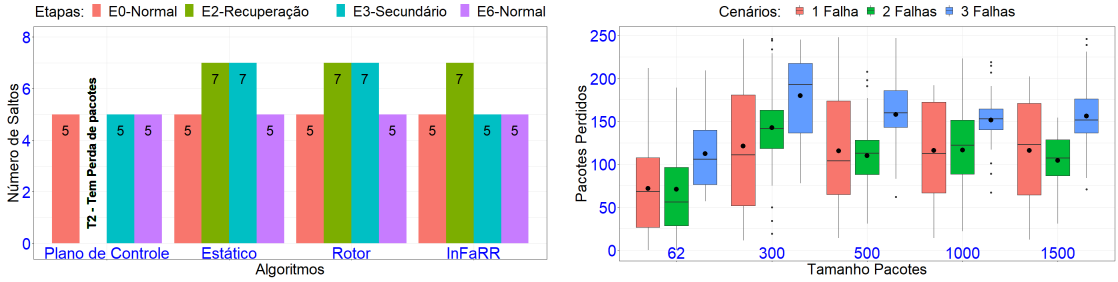


Figura 4. Cenário 2 falhas - Algoritmos Estático e Rotor.

Os algoritmos de roteamento Estático e Rotor não concluíram a recuperação durante o cenário com 2 e 3 falhas devido ao problema já descrito sobre as redes *standard Fat-Tree* que podem ocasionar *loop* [Liu et al. 2013]. Por outro lado, se diferenciando e indo além do que os demais, o InFaRR conseguiu concluir o processo de recuperação dentro dos parâmetros requeridos, conforme descrito na Seção 2.1, em todos os cenários avaliados. A Figura 4 ilustra a conexão entre H1P1 e H1P2 e detalha o cenário com 2 falhas. Neste caso, a utilização do caminho alternativo (rota *backup*), conforme sequência de enlaces numerados, acarreta em um *loop* no dispositivo S1CORE. A opção pelo mecanismo de *pushback* como reroteamento nos comutadores do Núcleo no InFaRR previne a ocorrência do *loop* conforme descrito na Seção 4 e possibilita a recuperação.

A Figura 5(a) apresenta o número de saltos observados durante as etapas do processo de recuperação ilustrado na Figura 1. As barras em vermelho demonstram a etapa sem falhas ($E0$). Em relação a etapa de recuperação $E2$, observa-se que algoritmos (Estático, Rotor e InFaRR) obtiveram 7 saltos. O comutador com o algoritmo plano de controle por possuir método de recuperação reativo não pode encaminha pacotes, até que uma atualização do plano de controle seja encaminhado a ele. Neste caso, conforme indicado na Figura 5(b), existe perda de pacotes ocasionado pelo intervalo de *pooling* de gerenciamento. A barra em azul representa o funcionamento no caminho redundante, etapa $E3$, no qual o InFaRR possui 5 saltos enquanto os algoritmos Estático e Rotor possuem 7. A barra em roxo, etapa $E6$, representa o retorno à rota principal.



(a) Número de saltos por etapas. Cenário 1 falha

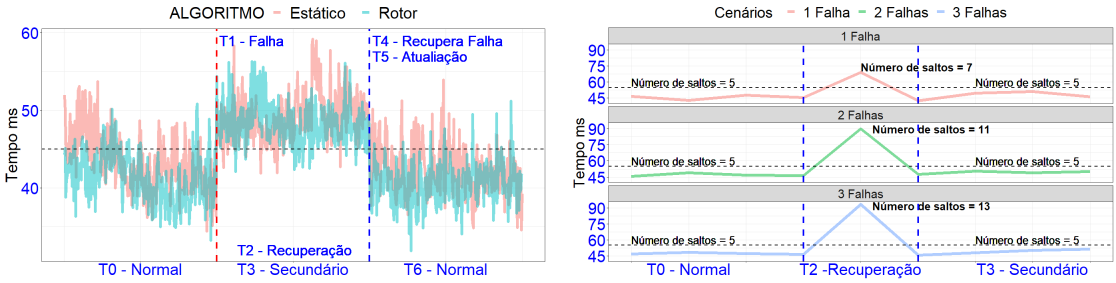
(b) Perda de pacotes - Plano de controle.

Figura 5. Resultados Obtidos.

A Figura 6(a) apresenta os resultados do cenário com 1 falha para os algoritmos Estático e Rotor. O início do reroteamento ocorre na linha vermelha, etapa $E1$, e finaliza na linha azul, etapa $E4$. As etapa de recuperação $E2$ e funcionamento pelo encaminhamento secundário (etapa $E3$) ocorrem simultaneamente, visto que todos os pacotes passaram pela etapa $E2$ e possuem um aumento no IPDT e número de saltos. A Figura 6(b) apresenta o tempo do IPTD durante a etapa $E2$, especificamente para o algoritmo InFaRR nos diferentes cenários de falhas. A etapa $E2$ consiste em buscar um caminho alternativo até seu destino e, graças ao mecanismo de aprendizagem, nesta etapa ocorre um aumento no número de saltos e do IPDT. É importante destacar que no InFaRR a etapa $E2$ é executada somente por um pacote sendo que durante a etapa $E3$ os demais pacotes terão o mesmo número de saltos e IPDT equivalentes a etapa inicial $E0$.

6. Conclusões

O InFaRR é um algoritmo de reroteamento rápido em redes programáveis desenvolvido na linguagem P4 com o intuito de atender as necessidades de recuperação de múltiplas



(a) IPTD - Algoritmo Estático e Rotor.

(b) IPTD - Detalhe do Processo de Recuperação do algoritmo InFaRR.

Figura 6. IPDT - Reroteamento Rápido.

falhas simultâneas, apresentando-se viável após a análise dos resultados em ambiente virtual sobre uma estrutura de redes hierárquicas *Fat-Tree*.

O processo de recuperação apresentou resultados que atendem os requisitos de QoS de IPTD e IPLR com base na Tabela 2. Se comparado aos algoritmos adaptados da literatura (Plano de controle, Estático e Rotor), o InFaRR obteve melhor resultado por se recuperar corretamente em todos os cenários de falhas avaliados. A programabilidade existente no plano de dados viabilizou a codificação do InFaRR que permitiu a detecção do *loop* e a otimização do domínio de recuperação, no qual não se fez necessário o roteamento do tráfego em um *pod* externo. Esta característica possibilitou a redução do número de saltos durante a etapa $E3$ - caminho secundário.

Os testes realizados em ambiente controlado fundamenta a realização de estudos mais aprofundados. Questões sobre a implementação em ambientes físicos possibilitará a observação do comportamento do InFaRR diante da velocidade de processamento e consumo de memória em redes com diferentes tipos de classes de tráfegos e flutuações na carga da rede.

A inclusão de funcionalidades atribuídas a etapa $E1$ - Mecanismo de Controle que não fizeram parte do escopo deste trabalho, poderão ser avaliadas em trabalhos futuros de forma a viabilizar o reroteamento rápido diante da avaliação da alta utilização da rede, descartes de pacotes, monitoração da retransmissão de pacotes ou o não atendimento de requisitos de qualidade de serviços como IPTD e IPDV. Há também necessidade de formalizacão do algoritmo, potencialmente usando Máquinas de Estados Finitos. Por fim, o InFaRR deverá ser avaliado em outras topologias, de forma a verificar sua viabilidade e desempenho em outros contextos topológicos.

Agradecimentos
Os autores agradecem o apoio financeiro da FAPESP através do processo $2021/14297-1$.

Referências

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

Chiesa, M., Kamisiński, A., Rak, J., Retvari, G., and Schmid, S. (2020). Fast Recovery Mechanisms in the Data Plane. *Ieee Cmst*, pages 1–46.

Chiesa, M., Sedar, R., Antichi, G., Borokhovich, M., Kamisiński, A., Nikolaidis, G., and Schmid, S. (2019). Purr: A primitive for reconfigurable fast reroute: Hope for the best and program for the worst. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, CoNEXT '19, page 1–14, New York, NY, USA. Association for Computing Machinery.

Chodorek, R. (2002). Qos measurement and evaluation of telecommunications quality of service [book review]. *Communications Magazine, IEEE*, 40:30–32.

Cholda, P., Mykkeltveit, A., Helvik, B. E., Wittner, O. J., and Jajszczyk, A. (2007). A survey of resilience differentiation frameworks in communication networks. *IEEE Communications Surveys and Tutorials*, 9(4):32–55.

ITU-T (2002). Recommendation Y.1541: network performance objectives for IP-based services. *ITU-T*, pages 7–9.

Kiranmai, L., Research Scholar, M., Professor, A., and Kumar, D. (2014). IP Fast Rerouting framework with Backup Topology. *International Journal of Computer Engineering In Research Trends*, 1(2):96–103.

Leiserson, C. E. (1985). Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901.

Liu, V., Halperin, D., Krishnamurthy, A., and Anderson, T. (2013). F10: A fault-tolerant engineered network. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, page 399–412, USA. USENIX Association.

Nikolaevskiy, I. (2016). *Scalability and Resiliency of Static Routing*. Doctoral thesis, School of Science.

ONF (2013). OpenFlow Switch Specification 1.4.0. *Current*, 0:1–3205.

Sedar, R., Borokhovich, M., Chiesa, M., Antichi, G., and Schmid, S. (2018). Supporting emerging applications with low-latency failover in P4. *NEAT 2018 - Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies, Part of SIGCOMM 2018*, pages 52–57.

Sgambelluri, A., Giorgetti, A., Cugini, F., Paolucci, F., and Castoldi, P. (2013). OpenFlow-based segment protection in Ethernet networks. *Journal of Optical Communications and Networking*, 5(9):1066–1075.

Stankiewicz, R., Cholda, P., and Jajszczyk, A. (2011). QoX: What is it really? *IEEE Communications Magazine*, 49(4):148–158.