MultiFlow: Uma Solução para Distribuição de Subfluxos MPTCP em Redes OpenFlow

Marcus Sandri[1], **Alan Silva**[1], **Fabio L. Verdi**[1]

[1]Departamento de Computação – Universidade Federal de São Carlos, Sorocaba – SP

{marcus.sandri, alansilva, verdi}@ufscar.br

Abstract. *This paper presents a solution for distributing MPTCP subflows in OpenFlow networks. MPTCP is a network protocol designed to branch a single TCP connection into many subflows and then, forward through disjointed paths. Usually, MPTCP uses ECMP for enabling flow balancing into a multipath topology. Nevertheless, there are many issues that shos that ECMP is not pareto-optimal, such as: ECMP can easily set two subflows from the same MPTCP connection on the same path and/or set a distinct forward and back forward route to the same subflow. To solve these issues, we designed MultiFlow, an OpenFlow module which guarantees multipath routing by setting subflows from the same MPTCP connection through distinct paths. MultiFlow is evaluated in comparative experiments with ECMP, taking into account resilience and throughput.*

Resumo. *Neste artigo, é apresentada uma solução para distribuir subfluxos MPTCP em redes OpenFlow. MPTCP é um protocolo desenvolvido para derivar um fluxo TCP em diversos subfluxos e estes serem roteados por caminhos disjuntos ao longo da rede. Convencionalmente, adota-se em conjunto o protocolo ECMP, do qual distribui fluxos de todos os tipos de protocolos ao longo de uma rede com múltiplos caminhos. Entretanto, existem diversas questões que mostram que o ECMP não é um protocolo altamente eficiente. Dentre elas, o ECMP comumente pode alocar dois subfluxos de uma mesma conexão em um mesmo caminho e/ou distribuir um caminho de ida diferente do caminho de volta. A fim de solucionar estes problemas, desenvolvemos o MultiFlow, um módulo em OpenFlow que garante que subfluxos pertencentes a uma mesma conexão MPTCP possam ser encaminhados em caminhos disjuntos. Validamos o MultiFlow em experimentos comparativos com o ECMP, analisando a taxa de transferência (throughput) e resiliência.*

1. Introdução

Atualmente, pesquisas em redes de computadores e *internet* do futuro estão focadas na busca de uma melhor qualidade de transmissão de dados. Para a melhoria da transmissão e contingência dos dados, normalmente é adotada uma topologia com múltiplos caminhos. Aproveitando o mesmo escopo, os protocolos de camada 4 têm sido aprimorados para se adaptarem a topologias de múltiplos caminhos. Juntando a topologia com a evolução dos protocolos de camada 4 permitindo o suporte de utilização de múltiplos caminhos *multipath*, surgem novas técnicas de roteamento que podem ser exploradas e estudadas.

Recentemente, o *Multipath TCP (MPTCP)* foi proposto em [Barré et al. 2011a] como uma extensão para o TCP possibilitando o roteamento *multipath*. A ideia é separar um fluxo TCP em diversos subfluxos. Com a criação desses subfluxos, é possível obter uma melhor resiliência e taxa de transferência de dados na utilização de diversos caminhos. O MPTCP permite a configuração de regras de rede com a intenção de encaminhar os subfluxos pelas interfaces de rede ativas em seu dispositivo. Quando um dispositivo possui diversas interfaces e os subfluxos são encaminhados por elas, a taxa de transferência de dados aumenta de forma a corresponder a soma da taxa de transferência das interfaces individuais por onde trafegam os subfluxos. O objetivo é que cada interface se conecte com diferentes portas do switch. Porém, o desenvolvimento do MPTCP engloba também dispositivos que utilizam somente uma interface de rede *(single-homed)*.

Os cenários *single-homed* mais comuns são aqueles que possuem topologias grandes, como o *datacenter*. O *datacenter* possui múltiplos caminhos e normalmente utiliza um protocolo de roteamento chamado *ECMP (Equal-Cost-MultiPath)* [Zhou et al. 2014]. O *ECMP* utiliza um algoritmo de *Hash* sobre a tupla de 5-campos[1], deste modo o *datacenter* é capaz de encaminhar os subfluxos MPTCP por caminhos distintos [Chiesa et al. 2014, van der Pol et al. 2012]. No entanto, pelo menos duas questões devem ser consideradas:

1. Os fluxos entre a origem e o destino são encaminhados apenas pelo caminho mais curto;
2. O desdobramento de tráfego é usado em cenários muito específicos e não há nenhuma garantia de que subfluxos serão encaminhados sempre por diferentes caminhos desconexos, devido ao comportamento estatístico da divisão do tráfego.

Como o balanceamento de carga no MPTCP é feito entre os subfluxos, os problemas apresentados sobre o ECMP são resolvidos com a utilização dos subfluxos. Entretanto, como abordado anteriormente, para que haja o roteamento multipath eficiente, é necessário aumentar a quantidade de subfluxos do MPTCP. Quando aumenta-se a quantidade de subfluxos, em conjunto, a probabilidade destes ocuparem um ou mais caminhos disjuntos é aumentada. O aumento dos subfluxos teoricamente otimiza o multipath, porém existem algumas questões que devem ser consideradas:

1. O MPTCP apresenta-se de modo agressivo quando em conjunto com o TCP convencional, isto é, quanto maior o número de subfluxos em um caminho, mais agressivo é com a conexão TCP que também está neste mesmo caminho;
2. Aumentar o número de subfluxos em uma rede com diversos *hosts* MPTCP pode aumentar desnecessariamente o tráfego, impactando tanto em conexões TCP quanto em conexões MPTCP.

Como base, o cenário ideal para *hosts single-homed* é aquele onde os subfluxos MPTCP são distribuídos de forma eficiente nos caminhos totalmente disjuntos disponíveis, ou seja, a quantidade de subfluxos deve ser igual a quantidade de caminhos disjuntos disponíveis e todos os subfluxos da mesma conexão MPTCP devem ser roteados por rotas diferentes. Com essa premissa, desenvolvemos um algoritmo de roteamento para redes OpenFlow [van der Pol et al. 2012] capaz de resolver os problemas acima citados.

Diversos datacenters utilizam redes OpenFlow devido a possibilidade de desenvolver novos algoritmos de roteamento que não são possíveis nas redes convencionais.

[1](IP origem e destino do IP; porta origem e destino do protocolo L4 e tipo de protocolo)

O OpenFlow permite a programação de *switches* na rede permitindo o desenvolvimento de novas aplicações. Neste artigo, desenvolvemos um módulo denominado MultiFlow que garante que os subfluxos das mesmas conexões MPTCP sejam roteados por rotas totalmente disjuntas.

2. Trabalhos Relacionados

Os primeiros estudos sobre MPTCP demonstram a sua utilização em redes de *datacenters* [Raiciu et al. 2011]. Em Raiciu et al. [Raiciu et al. 2011] é mostrado que o MPTCP tem um bom desempenho quando utilizado em conjunto com várias interfaces de rede e o seu encaminhamento é feito utilizando o protocolo ECMP. No entanto, os autores não consideram qualquer questão sobre redes OpenFlow.

O primeiro trabalho que apresenta a utilização de MPTCP e Openflow juntos foi descrito na literatura por van der Pol et al. [van der Pol et al. 2012]. Os autores demonstraram um conjunto de experimentos utilizando MPTCP como um protocolo de transferência e OpenFlow para a criação de uma engenharia de trafego (TE). Entretanto, o trabalho de [van der Pol et al. 2012] não propõe um modo de roteamento por subfluxos MPTCP e sim um estudo de caso em um *testbed* intercontinental em que os autores utilizam protocolos de roteamento como ECMP e TRILL [van der Pol et al. 2012] para a distribuição de fluxos. O uso do OpenFlow está relacionado a descoberta da topologia de rede e aplicação de regras para que cada subfluxo seja roteado em uma VLAN distinta, para que posteriormente seja encaminhado pela *internet* convencional. Existe um grande interesse em integrar MPTCP com OpenFlow conforme observado através do crescimento de trabalhos existentes na literatura. Em [Sonkoly et al. 2014] é utilizado um modelo de testes para mostrar que uma rede OpenFlow é capaz de melhorar o MPTCP ao permitir a escolha do melhor caminho para o primeiro subfluxo. [Zhou et al. 2014] mostra a possibilidade de experimentação de novos algoritmos de roteamento com a implementação do protocolo WCMP *(Weighted-Cost-Multipath)*, porém demonstra que não há ganhos significativos quando comparado ao ECMP.

Os problemas do MPTCP com *single-homed* são abordados inicialmente em Ming et al. [Ming et al. 2014]. Os autores investigam os efeitos de gargalos em *datacenters* e propõem um novo algoritmo de controle de congestionamento: *Equaly-Weighted Congestion Control*. Neste algoritmo, é previsto um baixo custo computacional para otimizar o uso da largura de banda. EW-MPTCP melhora o desempenho de largura de banda quando usado em conjunto com uma conexão TCP. Em paralelo, novamente em [Zhou et al. 2014] é demonstrado que o desempenho do ECMP é eficiente quando a topologia é sem falhas, regular e balanceada, porém os cenários criados não condizem com os ambientes reais existentes em redes. A necessidade e controle da quantidade de subfluxos é demonstrada em [Duan et al. 2015]. No trabalho de [Duan et al. 2015], o controlador OpenFlow se comunica com os *hosts* MPTCP e aloca uma determinada quantidade de subfluxos aproveitando as melhores rotas possíveis. O autor não especifica qual o critério utilizado para encaminhar os subfluxos e também admite que a criação dos subfluxos funciona exclusivamente no simulador NS-3. Entretando, na RFC 6897 [Scharf and Ford. 2013], é sugerida a criação de uma API para que uma aplicação possa ser capaz de manipular os atributos da conexão MPTCP. Dentre as possibilidades de manipulação, está a possibilidade de adicionar e remover novos subfluxos. Em [Sandri et al. 2015], demonstramos o MultiFlow como uma solução para o ajuste de sub-

fluxos com a quantidade de rotas. Foi demonstrado o impacto do roteamento baseado nos *tokens* dos subfluxos MPTCP quando comparado ao tradicional protocolo STP. Entretanto, os resultados não foram comparados com o ECMP, tal como será realizado neste trabalho.

3. ECMP

O intuíto de balancear a carga consiste em dividir a quantidade de pacotes ou fluxos ao longo de uma topologia com múltiplos caminhos. Um ambiente comum no qual o balanceamento de carga é utilizado são os ambientes que possuem topologias com múltiplos caminhos, como em *datacenters*. Num *datacenter*, topologias como Fat-Tree, Clos e BCube adotam um algoritmo de roteamento para balanceamento de carga, aproveitando o benefício dos múltiplos caminhos existentes. O algoritmo de roteamento mais utilizado atualmente para balanceamento de carga é o ECMP *(Equal-Cost-Multipath)*.

O ECMP possui dois modos de operação:

1. Balanceamento baseado em fluxo - utiliza o *hash* na tupla de 5 campos de cada pacote e o encaminha para uma interface de saída. Todos os pacotes do mesmo fluxo TCP/IP serão encaminhados pela mesma interface. Este é o modo padrão de funcionamento do ECMP;
2. Balanceamento baseado em pacote - utiliza o modelo *round robin* para cada pacote, fazendo com que pacotes do mesmo fluxo sigam por rotas diferentes.

A diferença básica destes dois modos de operação do ECMP impacta diretamente nos protocolos, tais como TCP ou MPTCP. Quando utiliza-se o modo de balanceamento baseado em pacote, pacotes que pertencem ao mesmo fluxo TCP/IP são roteados por rotas diferentes o que pode causar reordenação nos hosts finais e consequentemente diminuir a taxa de transferência. Quando o balanceamento baseado em fluxos é utilizado, o problema da entrega desordenada não ocorre pois todos os pacotes de um mesmo fluxo TCP/IP seguem a mesma rota. O ECMP, mesmo usando o modelo baseado em fluxo que normalmente se apresenta como melhor solução, ainda sofre com as colisões que ocorrem um vez que soluções baseadas em *hashing* estatisticamente podem gerar as mesmas saídas quando aplicadas em diferentes entradas. No caso da utilização do ECMP para balancear fluxos MPTCP, isso pode significar alocar os subfluxos de uma mesma conexão MPTCP em uma mesma rota.

4. MultiFlow

Esta seção descreve o módulo de roteamento MPTCP com OpenFlow para melhorar a taxa de transferência em cenários de redes de datacenters *OpenFlow-enabled*, por meio do envio de subfluxos de uma mesma conexão MPTCP através de caminhos disjuntos. Começamos descrevendo as funcionalidades do MPTCP. Por fim, será explicado como o MPTCP pode ser utilizado em conjunto com a rede OpenFlow.

4.1. MPTCP

MPTCP é um conjunto de extensões desenvolvidas para o protocolo TCP, que permite que os usuários dividam o tráfego entre caminhos potencialmente disjuntos. O MPTCP descobre o número de caminhos disponíveis para um usuário, estabelece os caminhos

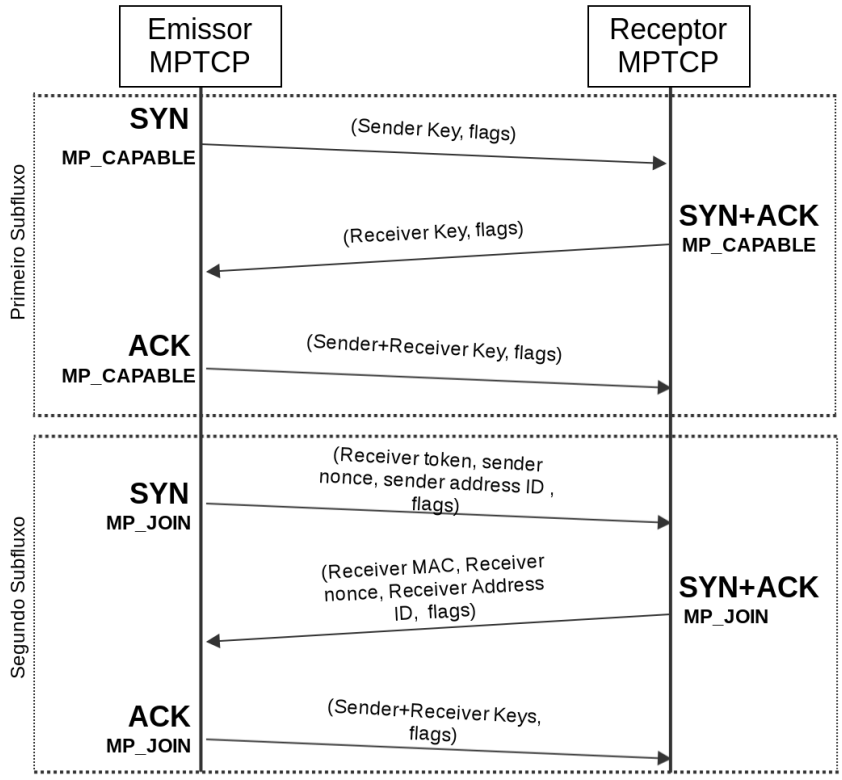


Figura 1. Diagrama de Sequência do MPTCP.

e distribui o tráfego entre esses caminhos. Isso acontece devido a criação de subfluxos [Barré et al. 2011b].

Como mostrado na Fig. 1, no MPTCP, cada subfluxo é uma conexão TCP convencional com seus próprios números de sequência. Os números de sequência de um subfluxo MPTCP são baseados no Número de Sequência de Dados (DSN), onde são adicionados números nos *bytes* do campo do MPTCP. Um único *buffer* de envio e de recepção MPTCP é compartilhado entre todos os subfluxos de uma mesma conexão MPTCP, enquanto cada subfluxo tem seu *buffer* de recepção para tratar a entrega desordenada de dados, caso esta venha a ocorrer [Diop et al. 2012].

Quando uma aplicação grava um fluxo de *bytes* para um *buffer* de envio MPTCP, o protocolo enumera cada *byte* com um DSN. Quando os *bytes* são enviados em seguida em um subfluxo particular, eles são encapsulados em TCP-PDUs com informações MPTCP colocadas no campo *options* do TCP. Quando um TCP-PDU é recebido em ordem em um subfluxo, a carga é entregue ao *buffer* de recepção MPTCP imediatamente. O número cum-ack do nível MPTCP, chamado DATA ACK, é incrementado quando os dados entregues também são enviados por ordem de entrada no nível do protocolo MPTCP [Alheid et al. 2014].

Uma aplicação consome dados em ordem do *buffer* de recepção do MPTCP. Atualmente, um remetente MPTCP apenas libera dados do *buffer* de envio MPTCP quando

eles usam o cum-ack pelos DATA ACK recebidos em qualquer subfluxo.

Os dados são recebidos muitas vezes fora de ordem em um *buffer* de recepção MPTCP por causa da perda ou devido aos RTTs assimétricos dos subfluxos. Em uma rede heterogênea quando diferentes subfluxos possuem características diferentes (ou seja, perda e RTT), a quantidade de dados fora da ordem que é recebido no lado do receptor MPTCP pode ser significativa [Paasch and Bonaventure 2014].

Basicamente, uma sessão MPTCP começa da mesma forma, com uma mudança: o iniciador acrescenta a nova opção MP_CAPABLE com chave remetente e *flags* para o pacote SYN. Se o *host* de recepção suporta MPTCP, ele irá adicionar essa opção ao seu SYNACK e responder com a chave do receptor e *flags*; os dois *hosts* também incluem chaves criptográficas nestes pacotes para uso posterior. O ACK final (que também deve levar a opção MP_CAPABLE) estabelece uma sessão de caminhos múltiplos utilizando a chave do remetente, chave receptor e *flags*.

Um novo subfluxo é adicionado na conexão MPTCP enviando um pacote SYN com o *token* receptor, *nonce* emissor, *address ID* do emissor e *flags* utilizando a opção MP_JOIN; o pacote também inclui informações sobre qual sessão MPTCP deve ser unida. Depois disso, o *host* receptor responde com SYN + ACK que inclui um MAC de origem, *nonce* do receptor, *address id* do receptor e *flags*. Os dados são transferidos com segmento ACK, e incluem a chave do remetente, chave do receptor e *flags* para habilitação da conexão [Paasch and Bonaventure 2014], como é mostrado na Fig. 1.

As chaves criptográficas anteriormente trocadas são usadas para prevenir ataques maliciosos e tais chaves serão utilizadas neste trabalho para a indexação de pacotes. Se o servidor de recebimento é passível de adicionar o subfluxo, vai permitir a criação de uma nova conexão TCP e adicionar o mesmo na sessão MPTCP já existente [Paasch et al. 2013].

Depois de uma sessão onde há mais de um subfluxo, cabe aos sistemas em cada extremidade decidirem como dividir o tráfego entre estes (embora seja possível marcar um subfluxo específico para uso somente quando qualquer outro já não funciona). Uma única janela de recepção se aplica para a sessão como um todo. Cada subfluxo se assemelha a uma conexão TCP normal, com seus próprios números de sequência, mas a sessão como um todo possui um número de seqüência separado; há outra opção TCP (DSS, ou *"Data Sequence Signal"*), que é usada para informar a outra extremidade sobre os dados em cada subfluxo. [Barré et al. 2011a, Wischik 2011].

Cada conexão MPTCP é identificada por um único *token* que é gerado pelo remetente ao receber a chave do receptor. O *token* é obtido pelo *hash* de criptografia de chave do receptor que foi trocado no *handshake* inicial do MP_CAPABLE. Depois de obtido o *hash* calculado, o *token* é gerado truncando os 32 bits mais significativos. Esse *token* é então incluído na opção MP_JOIN para identificar cada conexão MPTCP[Ford et al. 2013].

4.2. Fisiologia do MultiFlow

O Componente MultiFlow foi desenvolvido para melhorar a taxa de transferência em redes de datacenters que possuem suporte *OpenFlow*. Uma rede *OpenFlow* pode ser dividida em dois planos operacionais, conforme mostra a Fig. 2:

O plano de dados é o local onde os pacotes estão caminhando entre os *switches*,

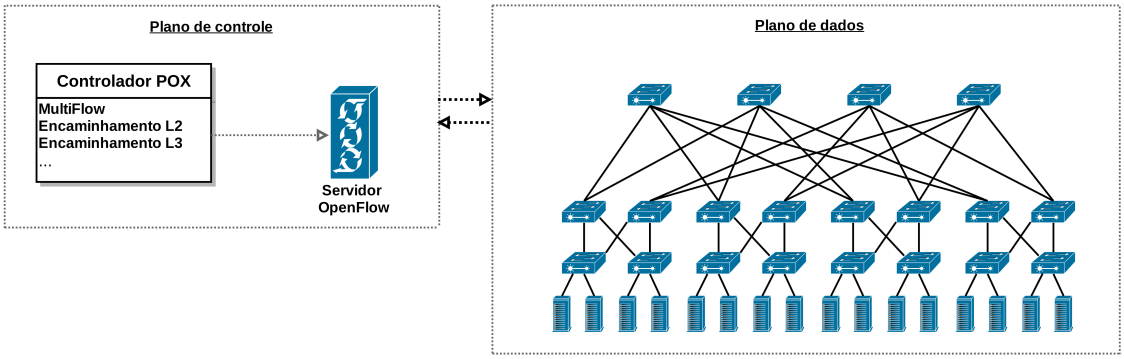


Figura 2. Arquitetura do MultiFlow.

utilizando uma estrutura de identificação do pacote e aplicação de regra *(match/rule)*. Em uma estrutura de *match/rule*, o *match* é usado para mapear um pacote com alguma regra. Quando um *match* não encontra nenhuma regra, o pacote é encaminhado para o plano de controle. No plano de controle, um controlador OpenFlow processa o pacote e sinaliza a estrutura *match/rule* para os *switches*. Como o controlador OpenFlow é virtualizado e programável, é possivel criar definições de *match* e *rules* para uma malha de *switches*.

Um pacote MPTCP por convenção, utiliza o cabeçalho TCP para encapsular mensagens de controle. Quando uma aplicação se comunica com um servidor, mensagens de controle são trocadas para tentar estabelecer uma conexão MPTCP. O MultiFlow identifica essas mensagens de controle inspecionando o *token*, o que permite que os subfluxos da mesma conexão MPTCP sejam identificados. Além disso, como os pacotes são criados a partir do *socket* TCP, é possível aproveitar a estrutura desse *socket* e criar regras OpenFlow utilizando os números de portas do pacote TCP (tupla de 5 campos). Isso permite que o pacote seja verificado para saber se ele pertence à mesma conexão MPTCP. Após a verificação, são definidas as regras que espalham as conexões através de caminhos disjuntos.

Neste trabalho, projetamos o MultiFlow para roteamento de subfluxos em caminhos disjuntos. O MultiFlow é um módulo projetado para o controlador POX. O POX é um controlador OpenFlow desenvolvido em linguagem de programação Python. Utiliza-se OpenFlow 1.1 para estabelecer a comunicação entre *switches* e controlador. A Fig. 3 mostra o MultiFlow na arquitetura do controlador POX.

O MultiFlow inicia escutando o módulo *Discovery*. Isto é necessário porque o MultiFlow utiliza o *Discovery* para montar a topologia e descobrir todos os *switches* e suas respectivas portas de rede, usando o Protocolo *Spanning-Tree* (STP). Em paralelo, o MultiFlow solicita que seja encaminhado o *payload* inteiro do protocolo TCP, quando os pacotes são do tipo MPTCP. Além disso, o controlador POX usa o módulo *Host_tracker* para detectar os *hosts* existentes e em quais *switches* estes estão conectados. Em seguida, o MultiFlow inicia a processamento do protocolo MPTCP e por fim envia as regras aos *switches*.

O MultiFlow divide-se em dois algoritmos principais: O primeiro algoritmo é responsável por encontrar o caminho mais curto para o MP_CAPABLE e enviar os *match/rules* para os *switches*. O segundo algoritmo é chamado quando um MP_JOIN

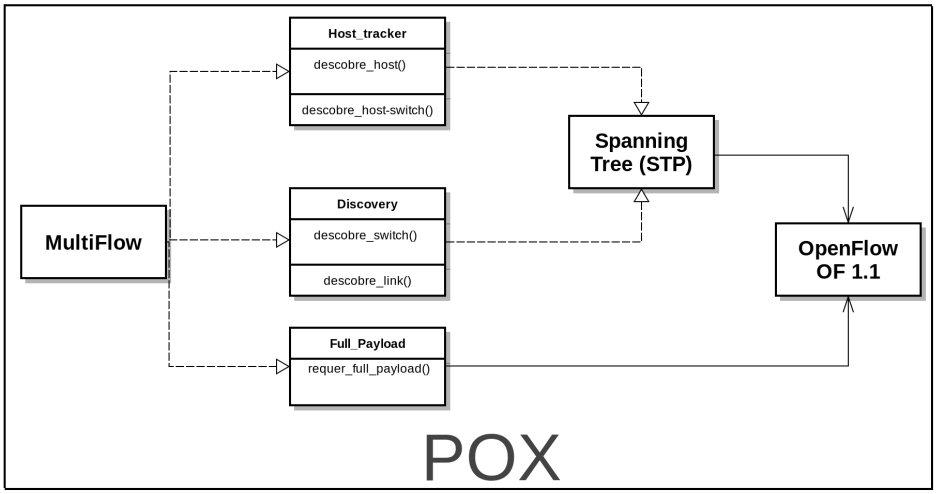


Figura 3. MultiFlow Dentro da Arquitetura do POX.

é recebido no controlador e é responsável pela distribuição dos MP_JOIN em caminhos disjuntos. Em suma, o algoritmo que efetua o MP_JOIN é o responsável por encaminhar efetivamente os subfluxos da mesma conexão MPTCP em caminhos disjuntos.

No Algoritmo 1, considera-se a tupla de 5 campos como um conjunto do IP de origem, IP de destino, porta de origem, porta de destino e tipo de protocolo. A *Flow_entry* é um novo pacote que não possui *match/rule* no *switch* e então, este é enviado para o controlador. As regras OpenFlow *(OpenFlow rules)* são mensagens com a estrutura de *match/rule* encaminhadas para serem registradas nos *switches*.

```
   input  : Flow_entry
   output: OpenFlow rules
1  if Flow_entry is tcp.syn then
2      if tcp.option == MPTCP_ID then
3          if MPTCP.option == MP_CAPABLE then
4              add_of.match(5-tuple)
5              add_of.action (default_route(match))
6          end
7      end
8  end
```

Algorithm 1: Inspeção e encaminhamento do MP_CAPABLE .

Inicialmente, separa-se o MP_CAPABLE e MP_JOIN. Isto é necessário porque apenas o MP_JOIN utiliza o *token* para identificar a conexão. Então, quando há mais de uma aplicação MPTCP que usa múltiplas conexões MPTCP provenientes do mesmo *host*, todos os MP_CAPABLE são encaminhados por um mesmo caminho. Por fim, o Algoritmo 1 detecta uma mensagem MP_CAPABLE e a encaminha para o *host* de destino.

O MultiFlow trata apenas de conexões MPTCP. Desta forma, os pacotes UDP, bem como os fluxos de TCP que não possuem cabeçalhos MPTCP, não são de interesse

e são encaminhados utilizando qualquer outro protocolo da rede. Isto significa que toda *Flow_entry* no MultiFlow é do tipo TCP.

Para identificar uma mensagem MP_CAPABLE, o MultiFlow filtra o SYN do cabeçalho TCP (linha 1). Em seguida, o algoritmo verifica se o TCP SYN tem o cabeçalho MPTCP populado com MPTCP_ID (linha 2), inspecionando se o campo *kind* possui o valor 30 (decimal) como o valor designado pelo IANA para o MPTCP. Caso a condição seja positiva, o MultiFlow inspeciona o campo *options* do MPTCP para identificar se o pacote é MP_CAPABLE ou MP_JOIN (linha 3). No caso de ser MP_CAPABLE, o MultiFlow calcula uma rota padrão para este MP_CAPABLE (linhas 4 e 5).

O Algoritmo 2 é responsável por identificar MP_JOINs e é considerado o núcleo do componente MultiFlow.

```
1  if MPTCP.option == MP_JOIN then
2      if hash.get(raw_token) == None then
3          token ← mp_join.raw_token
4          best_path ←dijkstra(S, D, Topology)
5          add_of.match(5-tuple)
6          add_of.action(best_path(match))
7          Topology ← del_path(best_path, Topology)
8          hash(token) ← Topology
9      end
10     else
11         newTopology ← hash.get(token)
12         best_path ←dijkstra(S, D, newTopology)
13         add_of.match(5-tuple)
14         add_of.action(best_path(match))
15         Topology ← del_path(best_path, Topology)
16         hash(token) ← Topology
17     end
18 end
```

Algorithm 2: Inspeção do MP_JOIN.

Quando um MP_JOIN é recebido (linha 1), o MultiFlow deve rotear este subfluxo utilizando uma rota diferente e disjunta dos subfluxos anteriores da mesma conexão MPTCP. Isto significa que o MultiFlow deve saber quais rotas já foram estabelecidas para a mesma conexão MPTCP. Para resolver isso, utiliza-se o *token* que está presente em cada mensagem MP_JOIN (linha 3) que é associado com uma topologia específica para cada conexão MPTCP. Em seguida, cada conexão MPTCP terá a sua própria topologia e todos os subfluxos da mesma conexão MPTCP (identificada por ter o mesmo *token*) utilizarão uma topologia individual. A Fig. 4 mostra a ideia com duas conexões diferentes MPTCP (MPTCP1 e MPTCP2) cada uma com dois subfluxos: SF 1.1 , SF 1.2, SF 2.1 e SF 2.2 onde SF significa subfluxo e os índices identificam a conexão MPTCP e o número do subfluxo, respectivamente.

Inicialmente, a topologia original é usada para cada nova conexão MPTCP. Em

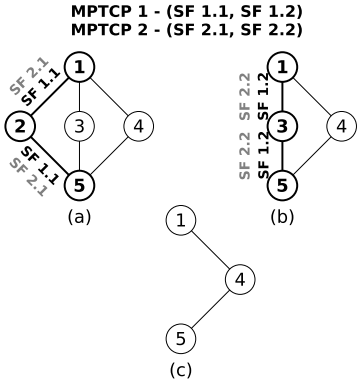


Figura 4. Virtualização da Topologia com MultiFlow.

seguida, suponha que o SF 1.1 é recebido no controlador. Uma vez que este é o primeiro subfluxo, o MultiFlow usa a topologia original como mostrado na Fig. 4 (a) e encontra o caminho mais curto entre o cliente e o servidor MPTCP (linha 4). O caminho escolhido para o encaminhamento SF 1.1 é dado pelos *switches* 1, 2 e 5. O MultiFlow define a rota para esse subfluxo utilizando regras OpenFlow (linhas 5 e 6) e exclui a rota utilizada (linha 7), resultando na topologia mostrada na Fig. 4 (b). Essa nova topologia é então armazenada e uma associação com o *token* é criada, servindo para identificar esta conexão MPTCP (linha 8). Agora, suponha que há um segundo subfluxo da mesma conexão MPTCP: SF 1.2. MultiFlow verifica se já existe subfluxos desta conexão MPTCP na rede (linha 2) e utiliza a topologia podada da Fig. 4 (b) (linha 11) para encontrar um novo caminho disjunto para SF 1.2 (linha 12). Novamente, o MultiFlow exclui apenas o percurso utilizado, resultando na nova topologia da Fig. 4 (c). Ao fazer isso sucessivamente, é criada a garantia de que subfluxos da mesma conexão MPTCP possam usar múltiplos caminhos disjuntos.

Agora, considere que SF 2.1 chegou no controlador, o que significa que o primeiro subfluxo de uma nova conexão MPTCP (MPTCP2) é recebido. Para este subfluxo, a topologia original usada é mostrada em Fig. 4 (a). Deste modo, o MultiFlow encontra o caminho mais curto, define as regras OpenFlow e exclui apenas a rota usada para que os próximos subfluxos desta conexão MPTCP não usem este mesmo caminho. Este algoritmo é repetido na medida em que cada novo subfluxo é recebido no controlador.

Observe que quando o algoritmo explicado acima é aplicado, subfluxos da mesma conexão MPTCP serão alocados em caminhos disjuntos (para resiliência e *throughput*) e subfluxos de diferentes conexões MPTCP são alocados possivelmente nos mesmos caminhos. Observe também que a ideia da exclusão das rotas utilizadas na topologia demonstra a ideia de ter uma topologia virtual que está sendo manipulada para cada conexão MPTCP. Com isso, é interpretado que o funcionamento do MultiFlow reside na criação de várias topologias lógicas diferentes sobre uma única topologia física.

Analisando mais a fundo o cenário da Fig. 4 é possível observar que a topologia original possui três caminhos disjuntos, o que significa que apenas três subfluxos em cada conexão MPTCP podem ser encaminhados de maneira disjunta. Se um quarto subfluxo é criado, nenhum caminho disjunto será encontrado. Para resolver isso, o MultiFlow

reinicia com a topologia original e aloca o quarto subfluxo em um dos caminhos mais curtos disponíveis. Isto irá configurar claramente subfluxos da mesma conexão MPTCP nos mesmos caminhos. No entanto, conforme demonstrado na avaliação que é feita na próxima seção, a taxa de transferência de uma mesma conexão MPTCP só aumenta se houver caminhos disjuntos suficientes para alocar os diferentes subfluxos em caminhos diferentes. Então, não é um ganho ter mais subfluxos do que a quantidade de caminhos desconexos.

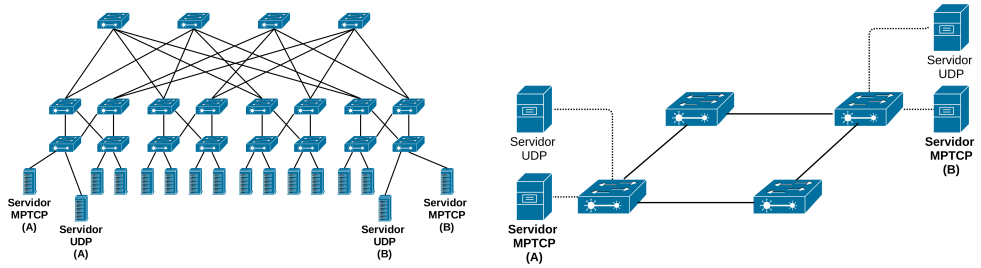
5. Resultados Experimentais

O MultiFlow foi implementado e testado para validar a abordagem deste trabalho em comparação com o tradicional ECMP. O MultiFlow foi desenvolvido e testado com OpenFlow 1.1 e MPTCP V.88. Para emular o ECMP, utilizamos uma implementação já existente chamada RiplPOX[2]. Utilizamos um *testbed* compilado com o *kernel* MPTCP no Ubuntu 13.10 e Mininet 2.1.0. O emulador Mininet utiliza o *kernel* da máquina física para a configuração do *kernel* no seus *hosts*. Todas as emulações foram feitas utilizando a seguinte configuração de *hardware*: um Computador Intel i7 com 8 núcleos de CPU (8 threads) e 8 GB de RAM. Nós restringimos a máquina do teste para utilizar somente nossa experimentação. Em nossos testes, utilizamos largura de banda de 10Mbps no Mininet para avaliação de todos os experimentos.

5.1. MultiFlow vs. ECMP

Nesta seção, é demonstrado a taxa de transferência do MPTCP quando usado em conjunto com o MultiFlow versus ECMP. Nos experimentos, utilizamos a topologia *FatTree* utilizada em *datacenters* e que é apresentada na Fig. 5 (a).

O experimento consiste em avaliar a taxa de transferência do MPTCP quando se utiliza MultiFlow e ECMP. Para isso, o Servidor MPTCP (A) irá se comunicar com o Servidor MPTCP (B). Há também um par de servidores na rede responsável pelo envio de um tráfego UDP intermitente com a janela de 9M o que faz com que haja um tráfego de 9 Mbps, ocupando um dos caminhos da topologia.



(a) Topologia FatTree (b) Topologia em *looping*.

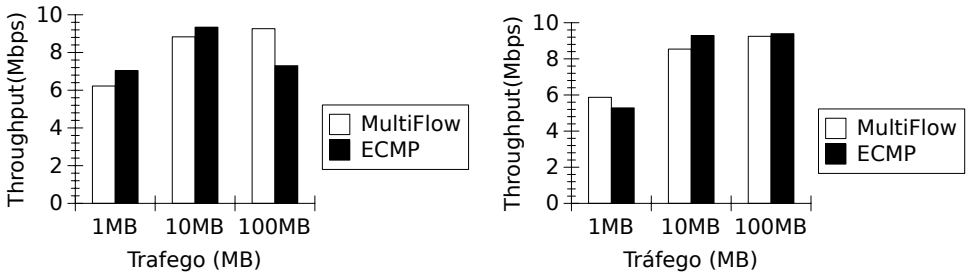
Figura 5. Topologias utilizadas nos experimentos

O par MPTCP irá transferir arquivos de 1MB, 10MB e 100MB. Como há dois caminhos disjuntos possíveis, o MultiFlow irá ajustar a quantidade de subfluxos para 2. Como critério de comparação, também utilizaremos 2 subfluxos nos testes com ECMP.

[2]https://github.com/MurphyMc/riplpox

A Fig. 6 (a) mostra o cenário em que é utilizado ECMP com distribuição por pacote. É possível observar que o MultiFlow ganha apenas no caso em que há 100MB de tráfego. Nos testes com 100 MB, o MultiFlow obteve uma média de 9,26 Mbps contra 7,29 Mbps do ECMP. Para o tráfego de 1 MB e 10 MB, o ECMP obteve uma ligeira vantagem.

Na Fig. 6 (b) há o cenário em que é utilizado ECMP com distribuição por fluxo. Neste cenário o MultiFlow obtem uma ligeira vantagem para 1MB, porém, perde nos testes com 10MB com 8,54 Mbps versus 9,29 Mbps do ECMP. No último caso de 100MB de tráfego, MultiFlow e ECMP estão praticamente empatados com 9,24 Mbps versus 9,39 Mbps, respectivamente.



(a) ECMP (*hash* por pacote) vs. MultiFlow (b) ECMP (*hash* por fluxo) vs. MultiFlow

Figura 6. MPTCP com 2 subfluxos roteados por ECMP e MultiFlow

Os resultados mostrados na Fig. 6 destacam que o Multiflow possui desempenho melhor em cenários com fluxos considerados elefantes (100MB) e que se comporta de maneira equivalente ao ECMP em outros cenários. Porém, o ponto de destaque aqui é que o Multiflow garante resiliência para as conexões MPTCP uma vez que os subfluxos das mesmas conexões não seguem pela mesma rota, característica que não está presente no ECMP.

Realizamos outro teste em uma topologia com apenas dois caminhos possíveis fim-a-fim como na 5 (b). Novamente, utilizamos um par de servidores MPTCP e um par de servidores UDP. Também utilizamos dois subfluxos para experimentação assim como foi utilizado no experimento com a FatTree. O par de servidores na rede responsável pelo envio de pacotes UDP envia um tráfego intermitente com a janela de 9M, fazendo com que haja um tráfego de 9 Mbps, ocupando um dos caminhos da topologia.

Na Fig. 7, são demonstrados os resultados finais.

Note que, diferentemente da topologia FatTree, o ECMP empata somente em um dos testes. Para os casos de 1MB e 10MB de tráfego, o MultiFlow teve um ganho significativo em comparação ao ECMP por fluxo e por pacote. Para os resultados de 100MB, o MultiFlow obteve êxito quando comparado ao ECMP por fluxo, porém empata com o ECMP por pacote.

Como avaliação final, podemos observar que o Multiflow possui bom desempenho em topologias com vários caminhos disjuntos fim-a-fim já que isto permite uma distribuição dos subfluxos das mesmas conexões MPTCP por rotas diferentes trazendo resiliência para estas conexões. O ECMP não requer que os caminhos sejam disjuntos e seu desem-

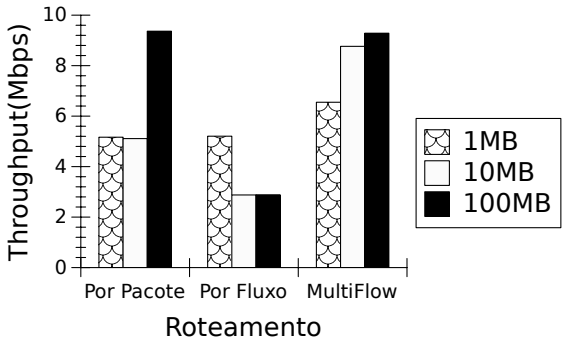


Figura 7. Resultados para topologia em *looping*.

penho foi, em alguns casos, melhor que o Multiflow, porém não garante nenhum tipo de tolerância a falhas na rede, requisito muitas vezes necessário para algumas aplicações.

6. Conclusão e Trabalhos Futuros

Neste artigo, demonstramos o MultiFlow, um módulo de roteamento utilizado para a solução na distribuição de subfluxos MPTCP em redes OpenFlow. O primeiro benefício da separação dos subfluxos em caminhos disjuntos é a resiliência. Um outro beneficio obtido pelo MultiFlow é o aumento da taxa de transferência de dados para fluxos longos (fluxos elefantes) em comparação a alguns modos de operação do ECMP.

Concluímos que a grande diferença entre o uso do ECMP (em ambos os modos de operação) e o MultiFlow, está na garantia de um melhor *throughput* para fluxos elefantes. Em todos os cenários, o pior caso para o MultiFlow foi o empate com o ECMP. Já no caso do ECMP, houve um empate no experimento com a topologia FatTree, utilizando *hash* por fluxo. Entretanto, quando testado em um cenário menor (topologia em *looping*), o mesmo modo de operação por fluxo obteve o pior desempenho para 10MB e 100MB.

Futuramente, pretendemos utilizar o MultiFlow em um *testbed* intercontinental utilizando a NorNet[3] para validar o uso do MultiFlow em cenários reais.

Referências

Alheid, A., Kaleshi, D., and Doufexi, A. (2014). An Analysis of the Impact of Out-of-Order Recovery Algorithms on MPTCP Throughput. In *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, pages 156–163.

Barré, S., Bonaventure, O., Raiciu, C., and Handley, M. (2011a). Experimenting with Multipath TCP. *ACM SIGCOMM Computer Communication Review*, 41(4):443–444.

Barré, S., Paasch, C., and Bonaventure, O. (2011b). Multipath TCP: From Theory to Practice. In *NETWORKING 2011*, pages 444–457. Springer.

Chiesa, M., Kindler, G., and Schapira, M. (2014). Traffic Engineering with Equal-Cost-Multipath: An Algorithmic Perspective. In *INFOCOM, 2014 Proceedings IEEE*, pages 1590–1598.

[3]Rede Experimental para MPTCP e SCTP.

Diop, C., Gomez-Montalvo, J., Dugue, G., Chassot, C., and Exposito, E. (2012). Towards a semantic and mptcp-based autonomic transport protocol for mobile and multimedia applications. In *Multimedia Computing and Systems (ICMCS), 2012 International Conference on*, pages 496–501. IEEE.

Duan, J., Wang, Z., and Wu;, C. (2015). Responsive Multipath TCP in SDN-based Datacenters. *The IEEE International Conference on Communications (ICC'15)*.

Ford, A., Raiciu, C., Handley, M., and Bonaventure, O. (2013). TCP Extensions for Multipath Operation with Multiple Addresses. *IETF RFC*.

Ming, L., Lukyanenko, A., Tarkoma, S., and Yla-Jaaski, A. (2014). MPTCP Incast in Data Center Networks. *Communications, China*, 11(4):25–37.

Paasch, C. and Bonaventure, O. (2014). multipath tcp. *Communications of the ACM*, 57(4):51–57.

Paasch, C., Khalili, R., and Bonaventure, O. (2013). On the Benefits of Applying Experimental Design to Improve Multipath TCP. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 393–398. ACM.

Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., and Handley., M. (2011). Improving Datacenter Performance and Robustness with Multipath TCP. *In Proceedings of the ACM SIGCOMM 2011 conference (SIGCOMM '11)*.

Sandri, M., Silva, A., Rocha, L., and Verdi, F. (2015). On the Benefits of Using Multipath TCP and Openflow in Shared Bottlenecks. In *The 29th IEEE International Conference on Advanced Information Networking and Applications (AINA'15)*.

Scharf, M. and Ford., A. (2013). Multipath tcp (mptcp) application interface considerations. RFC 6897.

Sonkoly, B., Nemeth, F., Csikor, L., Gulyas, L., and Gulyas, A. (2014). SDN Based Testbeds for Evaluating and Promoting Multipath TCP. In *Communications (ICC), 2014 IEEE International Conference on*, pages 3044–3050.

van der Pol, R., Boele, S., Dijkstra, F., Barczyk, A., van Malenstein, G., Chen, J., and Mambretti, J. (2012). Multipathing with MPTCP and OpenFlow. *High Performance Computing, Networking, Storage and Analysis (SCC 2012)*.

Wischik, D. (2011). Multipath: A new control architecture for the internet: Technical perspective. *Communications of the ACM*, 54(1):108–108.

Zhou, J., Tewari, M., Min Zhu, A. K., Poutievski, L., Singh, A., and Vahdat., A. (2014). WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers. *In Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14)*.