Marcus Sandri

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

Sorocaba, SP

30 de Março de 2015

Marcus Sandri

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

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCCS) da Universidade Federal de São Carlos como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação. Área de concentração: Arquiteturas Distribuídas de Software.

Universidade Federal de São Carlos – UFSCar

Centro de Ciências em Gestão e Tecnologia – CCGT

Programa de Pós-Graduação em Ciência da Computação – PPGCCS

Orientador: Prof. Dr. Fábio L. Verdi

Sorocaba, SP

30 de Março de 2015

Marcus Sandri
MultiFlow: Uma Solução para Distribuição de Subfluxos MPTCP em Redes OpenFlow/ Marcus Sandri. – Sorocaba, SP, 30 de Março de 2015-
62 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Fábio L. Verdi

Dissertação (Mestrado) – Universidade Federal de São Carlos – UFSCar
Centro de Ciências em Gestão e Tecnologia – CCGT
Programa de Pós-Graduação em Ciência da Computação – PPGCCS, 30 de Março de 2015.
1. MultiFlow. 2. SDN 3. OpenFlow. 4. Multipath-TCP 5. MPTCP

CDU 02:141:005.7

Marcus Sandri

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

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCCS) da Universidade Federal de São Carlos como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação. Área de concentração: Arquiteturas Distribuídas de Software.

Prof. Dr. Fábio L. Verdi
Orientador
(UFSCar)

Yeda R. Venturini
Universidade Federal de São Carlos
(UFSCar)

Christian E. Rothenberg
Universidade Estadual de Campinas
(UNICAMP)

Sorocaba, SP
30 de Março de 2015

Dedico a todos que me apoiaram ao longo do mestrado.

Agradecimentos

Agradeço,

ao Prof. Dr. Lucio A. Rocha por suas contribuições e orientações nas experimentações, na análise dos resultados e na escrita dos artigos.

ao meu orientador Prof. Dr. Fábio L. Verdi pelas oportunidades ao longo do mestrado, pela sua dedicação e entusiasmo nos resultados e no desenvolvimento deste trabalho.

ao Dr. Murphy McCauley, por ter desenvolvido a plataforma do qual utilizei para desenvolver este trabalho. Também, gostaria de agradecer pelos conselhos no desenvolvimento do MulfiFlow.

ao Alan Silva pelas incontáveis parcerias, ajudas e conselhos.

aos colegas de laboratório: Marcelo Frate, André Beltrami e Felipe Novais.

"After years lagging behind server automation, I was hugely optimistic when networking companies finally considered bringing modern autonomous automation methods to network equipment. When Software Defined Networking was proposed, its proponents were touting all the right words: separation of control from delivery, intelligence at the edge, abstraction, etc. But then they built OpenFlow, a Rube Goldberg machine with SDN to operate it. Don't worry though, it's open source so you can see all the moving parts!"

Mark Burgess

Resumo

Esta dissertação apresenta uma solução para distribuir subfluxos Multipath-TCP (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 Equal-Cost Multipath (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, é desenvolvido o MultiFlow, um módulo para o controlador POX a fim de garantir que subfluxos pertencentes a uma mesma conexão MPTCP possam ser encaminhados em caminhos disjuntos, em uma rede OpenFlow. MultiFlow é validado em experimentos de desempenho onde são analisados taxa de transferência *(throughput)* e resiliência em experimentos comparativos com os protocolos Spanning-Tree (STP) e ECMP. Para isso, utilizamos o Mininet: Um emulador de rede OpenFlow que permite a criação de diferentes topologias para experimentação.

Palavras-chaves: MultiFlow, MPTCP, Multipath-TCP, OpenFlow, SDN.

Abstract

This Master's thesis shows a solution for splitting MPTCP subflows in an Openflow network. MPTCP is a network protocol designed to branch a single TCP connection into many subflows. The main idea is to forward subflows th- rough disjointed paths. Commonly, ECMP protocol is adopted together to split flows through distinct paths. Nevertheless, there are many issues that shows that ECMP is not pareto-optimal, such as: ECMP can easily set two subflows from the same TCP connection on the same path and/or set a distinct forward and back forward route to the same subflow. To solve these issues, it is designed MultiFlow, a module which uses a controller for guarantee multipath routing by setting subflows from the same MPTCP connection so that such subflows are forwarded through distinct paths. MultiFlow is evaluated in experimentation where is analyzed throughput and resilience comparing it with Spanning-Tree (STP) and ECMP. The experiments were done by using Mininet: An OpenFlow emulator for experimenting with a set of topologies.

Key-words: MultiFlow, MPTCP, Multipath-TCP, OpenFlow, SDN.

Lista de ilustrações

Figura 1 – Arquitetura SDN (SDN...,). 26
Figura 2 – Arquitetura de um *switch* convencional Ethernet(NOTE, 2007). 28
Figura 3 – Exemplo de mensagens OpenFlow 28
Figura 4 – Arcabouço de um protocolo *multipath* (LIAO et al., 2010). 30
Figura 5 – Cabeçalho TCP (LIAO et al., 2010). 32
Figura 6 – Pilha do TCP com MPTCP (PAASCH; KHALILI; BONAVENTURE, 2013). 33
Figura 7 – Diagrama de Sequência MPTCP. 34
Figura 8 – *Fat-Tree* com roteamento IP. 36
Figura 9 – Arquitetura do MultiFlow. 41
Figura 10 – MultiFlow Dentro da Arquitetura do POX. 42
Figura 11 – Virtualização da Topologia com MultiFlow. 45
Figura 12 – Topologia experimental. 47
Figura 13 – MPTCP roteado pelo Multiflow. 48
Figura 14 – Topologia da rede OpenFlow: 3 Servidores e 3 Clientes, ambos MPTCP. 49
Figura 15 – TCP Reno vs. MPTCP com Multiflow. 50
Figura 16 – Analise assintótica. 50
Figura 17 – Setup time. 51
Figura 18 – Topologias utilizadas nos experimentos 52
Figura 19 – MPTCP com 2 subfluxos roteados por ECMP e MultiFlow. 52
Figura 20 – Resultados para topologia em *looping*. 53

Lista de abreviaturas e siglas

ACI	Application Centric Infrastructure
CERN	European Organization for Nuclear Research
DPI	Deep Packet Inspection
DSN	Data Sequence Number
DSS	Data Sequence Signal
ECMP	Equal-Cost Multipath
EW-MPTCP	Equaly Weighted MPTCP
IP	Internet Protocol
MB	MegaBytes
Mbps	Megabits por segundo
MPTCP	Multipath-TCP
POX	Controlador OpenFlow em Python
RTT	Round Trip Time
SDN	Software Definied Networks
SF	Subfluxo MPTCP
STP	Spanning-Tree Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network
WCMP	Weighted-Cost Multipath

Sumário

Introdução **21**

1 DISPOSIÇÃO DA DISSERTAÇÃO **23**
1.1 Motivações para esta Dissertação **23**
1.2 Objetivo desta Dissertação **23**
1.3 Desenvolvimento de Atividades **24**
1.3.1 Trabalhos Publicados 24
1.3.2 Projetos Desenvolvidos Parcialmente Relacionados a esta Dissertação . . . 24
1.4 Organização da Dissertação **24**

2 CONCEITOS BÁSICOS E TRABALHOS RELACIONADOS **25**
2.1 SDN e o Protocolo OpenFlow **25**
2.1.1 OpenFlow 27
2.1.2 Controladores 29
2.2 Evolução dos Protocolos de Transporte **29**
2.2.1 Camada de Transporte 30
2.2.2 Transmission Control Protocol (TCP) 32
2.2.3 Multipath-TCP 33
2.3 Evolução dos Protocolos de Roteamento **35**
2.3.1 Roteamento IP 35
2.3.2 Roteamento com ECMP 37
2.4 Trabalhos Relacionados **38**

3 ROTEAMENTO COM MPTCP E OPENFLOW **41**
3.1 MultiFlow **41**
3.1.1 Módulo MultiFlow 41

4 RESULTADOS E ANÁLISE **47**
4.1 MultiFlow vs. Spanning-Tree **47**
4.1.1 Análise do *Throughput* 47
4.1.2 Testes em Gargalo Compartilhado *(Shared Bottleneck)* 48
4.1.3 Curva Assintótica e Setup Time 50
4.2 MultiFlow vs. ECMP **51**

Conclusão **55**

Referências 57

APÊNDICE A – ESTATÍSTICA DETALHADA DOS RESULTADOS EXPERIMENTAIS 61

A.1 Qtd. de amostras 61

A.2 Desvio Padrão dos gráficos 61

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., 2011) 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; 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 qualquer 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

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

é aumentada. O aumento dos subfluxos teoricamente otimiza o *multipath*, porém existem algumas questões que devem ser consideradas: (A) O MPTCP apresenta-se de modo agressivo[2] 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; (B) 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 (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. O OpenFlow permite a programação de *switches* na rede permitindo o desenvolvimento de novas aplicações. Nesta dissertação, desenvolvemos um módulo denominado MultiFlow que garante que os subfluxos das mesmas conexões MPTCP sejam roteados por rotas totalmente disjuntas.

[2] interfere no *throughput* de outras

1 Disposição da Dissertação

1.1 Motivações para esta Dissertação

- Atualmente, é comum encontrar servidores de *datacenter single-homed*. Há a possibilidade de transformar *hosts single-homed* em *hosts multihoming*, porém isso implica no uso de mais portas, consequentemente implica no uso de mais *switches*, o que por fim implica num custo maior em um *datacenter*.

- O advento do MPTCP reacendeu a discussão entre custo e benefício de utilizar *hosts multihoming*, porém pouco se comenta das vantagens que o MPTCP agrega ao utilizá-lo em *hosts single-homed*. Essa discussão, incide diretamente nos algoritmos utilizados na rede para espalhar os subfluxos do MPTCP ao longo da rede, como o ECMP.

- O ECMP é utilizado para espalhamento do tráfego em *datacenters* e espalha o tráfego baseado em pacote ou fluxo. O ECMP ao distribuir por pacote, pode alocar pacotes de um mesmo fluxo em caminhos distintos o que acarreta em entrega desordenada. Quando distribui por fluxo, em uma rede não balanceada, pode acarretar em colocar fluxos em um mesmo caminho, gerando um balanceamento de tráfego deficiente.

- Como o ECMP é estatístico, o aumento do número de subfluxos nos *hosts* MPTCP pode aumentar a probabilidade de distribuir um subfluxo em um caminho disjunto dos demais. Entretanto, aumentar o número de subfluxos implica em aumentar o gargalo na rede, gerando um comportamento agressivo em relação as conexões TCP convencionais.

- Existem soluções que utilizam um controlador OpenFlow para controlar as rotas e se comunicam com *hosts* MPTCP para regular a quantidade de subfluxos a serem abertas. Porém, além de ser uma solução feita exclusivamente em um simulador, esta não funciona em ambientes reais.

1.2 Objetivo desta Dissertação

Melhorar o roteamento do protocolo MPTCP para *hosts single-homed* em *datacenters* utilizando OpenFlow. Para isso, identificamos os subfluxos do MPTCP observando um *token* a fim de separar estes subfluxos em rotas disjuntas.

1.3 Desenvolvimento de Atividades

1.3.1 Trabalhos Publicados

- SANDRI, M. et al. "On the Benefits of Using Multipath TCP and Openflow in Shared Bottlenecks". *The 29th IEEE International Conference on Advanced Information Networking and Applications (AINA'15).* Gwangju – Korea, March 2015.
- SANDRI, M. SILVA, A. e VERDI, F. L. "MultiFlow: Uma Solução para Distribuição de Subfluxos MPTCP em Redes OpenFlow". *XIII Workshop em Clouds e Aplicações (WCGA2015) – SBRC'15* Vitória – Brasil, Maio 2015.

1.3.2 Projetos Desenvolvidos Parcialmente Relacionados a esta Dissertação

- "Testbed OpenFlow Intercampi". *UFSCar São Carlos – Sorocaba*
  2013 – 2013.
- "The First Nornet Core in Latin America". *UFSCar Sorocaba*
  2014

1.4 Organização da Dissertação

No Capítulo 2 apresentamos os conceitos básicos e os trabalhos relacionados. No Capítulo 3 apresentamos o funcionamento do Multiflow. Por fim, no Capítulo 4, apresentamos os experimentos realizados e a discussão dos resultados obtidos.

2 Conceitos Básicos e Trabalhos Relacionados

Neste capítulo, apresentamos os conceitos necessários para a compreensão desta dissertação. Primeiramente, apresentaremos o modelo de roteamento em uma rede OpenFlow, posteriormente apresentaremos os protocolos MPTCP e ECMP. Após a apresentação dos conceitos básicos, discutiremos alguns trabalhos relacionados a esta dissertação.

2.1 SDN e o Protocolo OpenFlow

A evolução das redes de computadores comumente são relacionadas a servidores, *hosts* e dispositivos de rede (roteadores, *switches*, etc). Ao longo dos anos, esta evolução mostrou-se animadora para pesquisas relacionadas a servidores e *hosts*, como o desenvolvimento de SOs cada vez mais abertos e gratuitos, novos protocolos de rede transparentes e melhor segurança de dados. Porém, para dispositivos de rede tal evolução foi discreta. É comum até os dias atuais, que poucos dispositivos de rede sejam adeptos ao *software* livre e/ou *software* de fonte aberta *(open-source)*. Também é comum que protocolos desenvolvidos em SO livres tenham dificuldades em serem validados em ambientes reais, de modo que estes dependam exclusivamente da aceitação de fabricantes de dispositivos de rede. Como consequência, pesquisas em redes tornaram-se distantes do ambiente real e a necessidade de padronização e acessibilidade ao código tonaram-se algo cada vez mais aclamado nos ambientes de pesquisa e empresariais.

Não de hoje, a necessidade de controle dos dispositivos surgiu devido aos modelos convencionais (Rede IP) serem um ecossistema heterogêneo, do qual muitas vezes passam subitamente de um ambiente amigável para um ambiente inseguro, intolerante a falhas e a dispositivos de redes de fabricantes diferentes (BURGESS, 2000). Consequentemente, essa necessidade levou ao desenvolvimento de um modelo de centralização da rede. Inicialmente, modelos de gerenciamento adaptados a rede IP como SNMP e OSI suavizaram os efeitos da ossificação da rede IP. Porém, a condição de gerência desses modelos não permite que a rede IP torne-se escalável a longo prazo, nem que esta seja capaz de suportar protocolos novos, sem a necessidade de investir em novos equipamentos e/ou *softwares* proprietários.

Inicialmente discutido em 1993, o conceito de infraestrutura centrada em aplicação (ACI) foi apresentado no CERN como parte de uma possível solução para o avanço e solução de problemas das redes de telecomunicações e de computadores. O resultado foi o desenvolvimento de um sistema de gerenciamento de configuração centralizado (CFEngine), que também servia em dispositivos de rede embarcados (BURGESS, 2000). Dois anos após

o surgimento teórico de ACI no CERN em 1993, o conceito de ACI passa a ser objeto de estudo práticos com o adendo de utilizar virtualização, desde então, surge o que se chama de SDN. De 1995 até 1998, diversas tentativas de demonstrar SDN em um experimento prático totalmente funcional foram feitas, como o GeoPlex na AT&T e o WebSprocket da Sun Microsystems (WEBSPROCKET. . . , 2001). Por fim, somente em 2001 é apresentado o primeiro estudo de caso real sobre SDN, no qual utiliza-se a plataforma WebSprocket (CALYAM, 2002).

Após cinco anos do sucesso do primeiro teste real, em 2006 inicia-se um movimento para pesquisas em SDN na Universidade de Stanford (MCKEOWN; GIROD, 2006; BURKE, 2012). O documento publicado em (MCKEOWN; GIROD, 2006), incita a urgência em desenvolver tecnologias emergentes de redes, devido a seu possível potencial para a indústria e academia. Finalmente em 2008, diversas instituições acadêmicas[1] reuniram-se para a criação de uma arquitetura para SDN para desenvolvimento prático e científico, como mostra a Fig. 1.

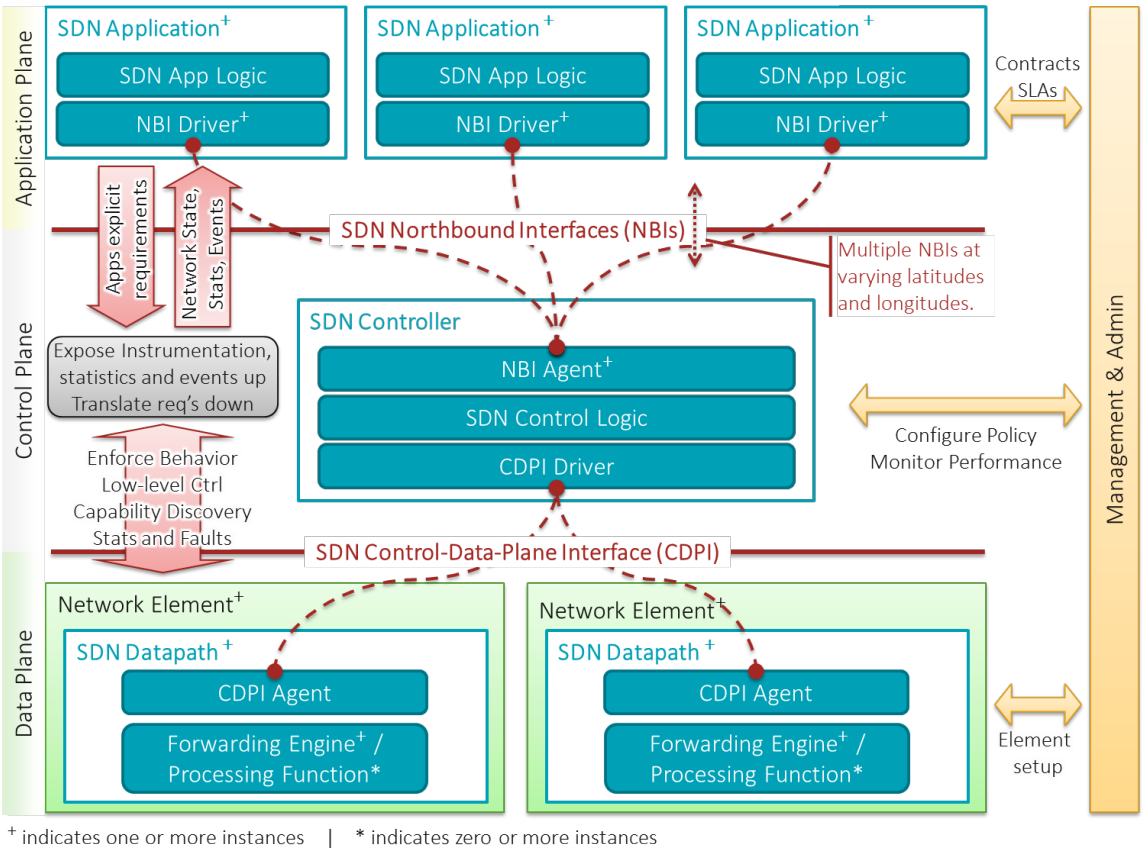


Figura 1 – Arquitetura SDN (SDN. . . ,).

Como consequência deste consórcio, cria-se um protocolo para interoperar entre as estruturas de comunicação da arquitetura SDN. O protocolo chamado OpenFlow, é o principal ator dentro da arquitetura SDN (BOUCADAIR; JACQUENET, 2014). Devido a

[1] Stanford, UC Berkeley, Washington, Princeton e MIT.

tamanha importância deste protocolo, convencionalmente adota-se a terminologia Rede OpenFlow ao se referir a uma rede com arquitetura SDN. Precisamente, o OpenFlow é responsável pela comunicação entre o plano de dados *(Data Plane)* e plano de controle *(Control Plane)* (MCKEOWN et al., 2008).

No plano de dados, um módulo do protocolo OpenFlow é instalado dentro de um ou mais *switches*. Os processos de comutação e roteamento, antes processados localmente nos *switches* na rede IP, são agora virtualizados para o plano de controle via protocolo OpenFlow, do qual tem suas informações processadas em um "controlador" *(Application Plane)*. No controlador OpenFlow é possível enviar comandos de volta aos *switches*, de forma a gerenciar seu roteamento (MCKEOWN et al., 2008).

A Subseção 2.1.1 apresenta em detalhes o funcionamento do OpenFlow.

2.1.1 OpenFlow

Como mostrado anteriormente, a arquitetura SDN prevê a separação do plano de controle, em relação ao plano de dados. Ao virtualizar o plano de controle, o OpenFlow comunica-se entre *switches* e controlador. Para isso, no *switch*, é embarcado um módulo OpenFlow e em um computador remoto, uma aplicação (controlador) opera em modo servidor, do qual o *switch* OpenFlow irá se autenticar.

Em um *switch* convencional, existem dois modos de operação que são semelhantes aos planos de dados e controle em SDN.

Inicialmente, um *switch* convencional possui dois modos de operação quando precisa encaminhar um pacote: Comutação por processamento rápido *(fast switching)* e por processamento de *switch (process switching)*(NOTE, 2007; George Varghese, 2005).

O *process switching* acontece quando um primeiro pacote de uma origem qualquer chega em um *switch* e este não possui pré-informação armazenada em alguma das memórias dos Processadores de Interface Versátil (VIP). Então, o pacote é encaminhado para o Processador de Comutação & Roteamento (RSP), no qual é processado. Após processar e encaminhar o primeiro pacote, os demais pacotes da mesma origem serão processados somente nos VIPs. Isso acontece devido ao RSP registrar nas memórias DRAM dos VIPs uma regra com uma estrutura de campos específicos do pacote, ex.: MAC Origem e MAC Destino. Os VIPs então processam os demais pacotes, o que é definido como o *fast switching*. Como o plano de controle é responsável por processar protocolos de roteamento e o plano de dados atua como um auxiliar para processamento rápido, podemos definir que o *fast switching* ocorre no plano de dados, assim como o *process switching* ocorre no plano de controle (George Varghese, 2005).

O RSP é um sistema embarcado[2] utilizado para processar pacotes e protocolos.

[2] Arquitetura programável e dedicada a executar uma tarefa.

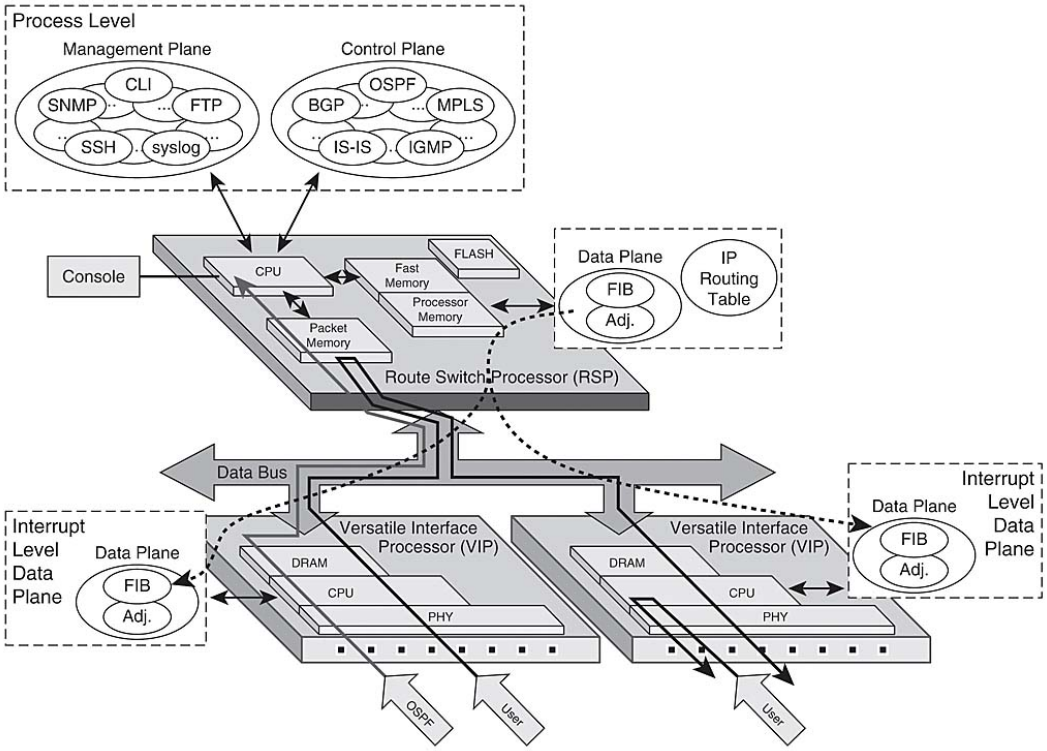


Figura 2 – Arquitetura de um *switch* convencional Ethernet(NOTE, 2007).

No RSP alguns de seus componentes são desenvolvidos com FPGA, o que permite aos dispositivos serem programados. O OpenFlow utiliza deste atributo para que seja possível instalar seu módulo que fará a comunicação com o controlador. Com isso, todos os pacotes que seriam processados como *process switching* são enviados para o controlador, de forma a virtualizar o plano de controle do *switch*. A Fig. 3 mostra os campos utilizados para identificar e aplicar regras em fluxos que o controlador envia ao *switch* com o OpenFlow v.1.1. Consideramos a versão 1.1, pois nas versões posteriores (1.3 e 1.4) foram acrescentados mais campos que não contribuiriam diretamente para este trabalho.

OpenFlow-enabled Network Device

Flow Table comparable to an instruction set

MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:..	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Figura 3 – Exemplo de mensagens OpenFlow .

Note que na Fig. 3 há uma tupla com o seguinte conteúdo: MAC origem/destino,

IP origem/destino e TCP porta destino. Do outro lado da figura, há um campo definido como *"Action"*. Ambos os campos correspondem a uma estrutura utilizada pelo OpenFlow chamada de *Match/Action*. O *match* corresponde a tupla de 5 campos descrita anteriormente e o *action* a alguma ação tomada pelo controlador. Isso acontece devido ao *Match/Action* atuarem como uma estrutura condicional: "Se existe algum fluxo com os seguintes campos *(match)*", "encaminhar para porta *x (action)*". Adicionalmente, o OpenFlow também sinaliza de qual *switch* a mensagem pertence, possibilitando virtualizar não apenas um *switch*, mas uma malha, do qual se limita em grande parte apenas pelo controlador utilizado.

Quando as mensagens chegam no controlador OpenFlow, este permite que seja possível programar as estruturas de *Match/Action*, podendo criar outras combinações de tuplas ou até inspecionando o pacote inteiro (DPI), como será feito ao longo desta dissertação para tratar o protocolo MPTCP. Na Subseção 2.1.2, descreveremos em maiores detalhes o funcionamento de um controlador OpenFlow.

2.1.2 Controladores

Controladores OpenFlow são interfaces programáveis que **se comunicam** com o protocolo OpenFlow. Este por sua vez, comunica-se com os *switches* e vice-versa. No controlador é possível programar as estruturas de *Match/Action*. Diversos controladores foram desenvolvidos, baseados em diferentes linguagens de programação. Entre os principais projetos estão: POX, NOX, Floodlight (MUNTANER, 2012).

O controlador POX é um controlador OpenFlow baseado na linguagem de programação Python, do qual permite maior abstração da linguagem, mas possui um desempenho inferior aos demais controladores citados. O NOX utiliza a linguagem de programação C++, o que oferece o melhor desempenho entre os controladores, porém possui o menor nível de abstração. O Floodlight tenta ser um controlador completo, juntando as aplicações OpenFlow com aplicações de monitoramento, como o sFlow.

Embora Floodlight e NOX ofereçam um melhor desempenho do que o POX, a abstração permite a facilidade de implementação quando há um novo protocolo, tal como o MPTCP. Ao longo desta dissertação, utilizaremos o controlador POX em nossos experimentos e desenvolvimento dos algoritmos.

2.2 Evolução dos Protocolos de Transporte

Protocolos de transporte objetivam a transferência de arquivos entre hospedeiros e servidores. Nesta Seção, descrevemos a evolução dos protocolos de transportes. Em especial, descrevemos os protocolos MPTCP.

2.2.1 Camada de Transporte

A camada de transporte é responsável pela transferência de dados entre um ou mais *hosts* – servidores. Dentre os tópicos de pesquisa em protocolos de transporte, destacam-se os protocolos que possuem a capacidade de derivar um fluxo de uma aplicação em diversos subfluxos entre diversas interfaces. Isso se deve a frequência com que dipositivos em geral utilizam ao mesmo tempo múltiplas tecnologias de dados, como telefones celulares que utilizam *Bluetooth, IEEE 802.11, LTE*, etc. Também é comum em servidores o uso de duas ou mais interfaces *Ethernet*. Uma outra possibilidade é aproveitar tais protocolos em *hosts* com apenas uma interface. Com isso, os múltiplos subfluxos alocados em apenas uma interface possibilitam um roteamento por caminhos disjuntos *(multipath routing)* ao longo da rede.

Ao transmitir subfluxos entre diversas interfaces, ou transmiti-los em uma só, surgem dois métodos de distribuição dos dados entre os subfluxos utilizados: (A) O protocolo pode duplicar os pacotes nos diversos subfluxos *(link backup)*. Com isso, caso haja alguma perda de pacote ao longo de um caminho, os pacotes perdidos são recuperados nos demais subfluxos. (B) O protocolo *multipath* pode balancear a carga de uma aplicação nos diversos subfluxos *(Concurrent Multipath Transfer)*. Caso haja diversas interfaces, o *throughput* final passará a ser incrementado pela banda das interfaces utilizadas. A Fig. 4 demonstra os processos comuns em um arcabouço de um protocolo *multipath*.

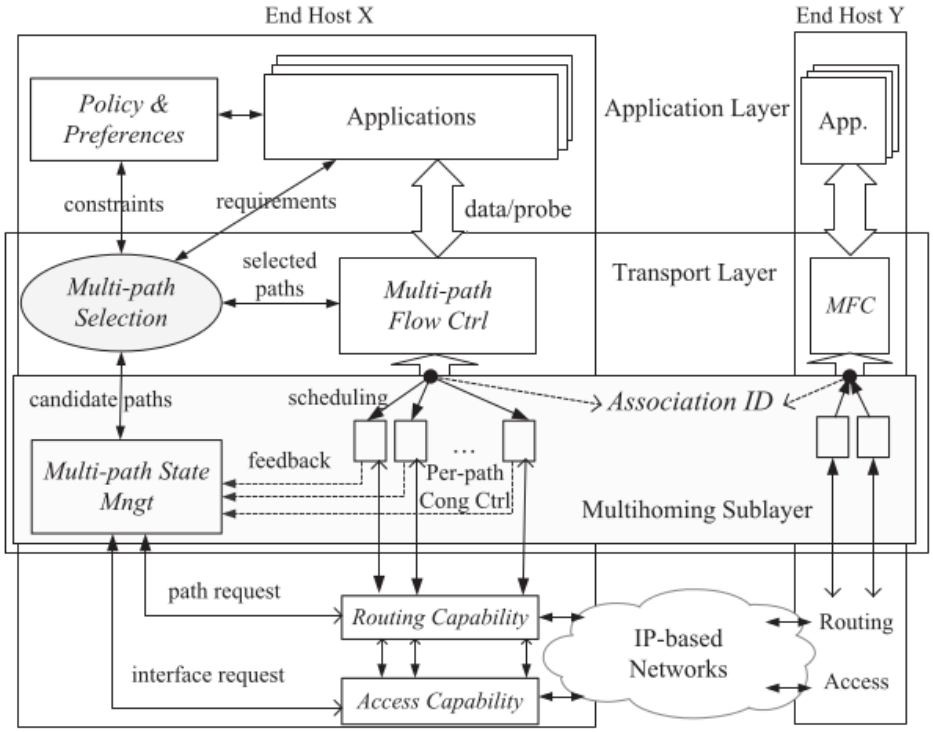


Figura 4 – Arcabouço de um protocolo *multipath* (LIAO et al., 2010).

A camada de aplicação encapsula os dados e os repassam para a camada de transporte. Esta por sua vez, possui a implementação de algum protocolo *multipath*. O processo *Multipath Flow Ctrl* é responsável por gerenciar todos os subfluxos. Este pertencente a camada de transporte e intermedia o fluxo encapsulado da aplicação com a distribuição dos dados no subfluxos. Os subfluxos são criados a partir de um escalonador e cada subfluxo passa a ser uma conexão com seu próprio controle de congestionamento. Em alguns modos de operação, como os que usam balancemaneto de carga entre os subfluxos, cada subfluxo retorna um *feedback* notificando o *Multipath Flow Ctrl* quanto ao tamanho da janela e retransmissões. Com esse *feedback*, o *Multipath Flow Ctrl* controla para qual subfluxo irá enviar mais dados. Por fim, os subfluxos devem ser roteados na camada de rede, sem que haja problemas em trechos intermediários na rede. Os subfluxos ao serem transmitidos, são marcados com um identificador de conexão *token* para que no receptor *(End Host Y)* seja possível remontar os pacotes quebrados nos diversos subfluxos.

Diversos protocolos *multipath* foram propostos para serem instituídos como um protocolo sucessor dos atuais TCP e UDP (ARYE et al., 2012; GRINNEMO; BRUNSTROM; CHENG, 2014; PAASCH et al., 2012; RAICIU et al., 2011). Surgem então, duas linhas de pensamento criadas para implantar um protocolo *multipath* consolidado e funcional: (1) Desenvolver um novo protocolo, com um novo paradigma e pilha própria. (2) Desenvolver a extensão de um protocolo para *multipath*, no caso o protocolo TCP.

Da linha de pensamento (1), surge o SCTP(Stevens, R. and Fenner B. and Rudoff, A., 2009). O SCTP é um protocolo criado para substituir os protocolos da camada de transporte: TCP e UDP. O SCTP junta ambos os modos de operação (datagrama e *stream*) em um único protocolo, de modo a fazer com que o desenvolvedor possa escolher qual a melhor opção para a aplicação que irá utilizá-lo. No modo *stream* do SCTP, o protocolo replica um fluxo SCTP de um servidor fim-a-fim, e o transmite por diversas interfaces disponíveis *(multihomming)*. Caso haja perda de pacote no *link* principal, o pacote é substituído por uma réplica de um outro fluxo. A manipulação na aplicação e a necessidade de utilizar um novo protocolo nos servidores, clientes e dispositivos de segurança, acabou por tornar o SCTP um protocolo em constante evolução, mas de difícil aplicabilidade. A versão mais atual do SCTP (CMT-SCTP) permite que os fluxos de dados possam ter a carga de pacotes distribuídas e não apenas replicadas, como em sua versão anterior. Para que isso aconteça, o CMT-SCTP utiliza um modo de transmissão chamado Transferência Concorrente por Multicaminho (CMT).

Da linha de pensamento (2), surge o *Multipath*-TCP (MPTCP). O MPTCP é uma extensão para *multipath* do qual utiliza a pilha TCP. Segue o mesmo padrão de comunicação do CMT-SCTP, tal que distribui aleatoriamente os dados dentro de seus subfluxos. Por usar uma sucinta implementação na pilha TCP, o MPTCP usa os mesmos campos dentro do cabeçalho TCP para sinalização. Especificamente, utiliza um trecho

inutilizado do campo *options* para sinalização. Devido a ser uma extensão do TCP, o MPTCP mostrou-se mais transparente do que o SCTP, o que o levou a ser um protocolo largamente desenvolvido e utilizado em diversos ambientes de redes. Pela aplicabilidade do MPTCP em redes diversas, decidimos utilizá-lo como protocolo *multipath* em nossos experimentos. Na Subseções 2.2.2 e 2.2.3, descrevemos em detalhes o funcionamento do MPTCP.

2.2.2 Transmission Control Protocol (TCP)

Atualmente, diversas aplicações utilizam o protocolo TCP. Isso se deve a este ter um conjunto de mecanismos que adapta-se intergralmente a problemas encontrados em diversos cenários, são estes: falha de comunicação no *link*, do qual o TCP persiste em re-enviar os dados; congestionamento em um meio compartilhado, tendo seu papel definido com o controle de congestionamento, adaptando as janelas de transmissão para enviar os dados; e por fim, estabelecimento da conexão com o *Three-Way Handshake* que resulta também em melhor segurança. Os campos que permitem que o TCP seja um protocolo largamente aceitável em diferentes cenários é mostrado na Fig. 5.

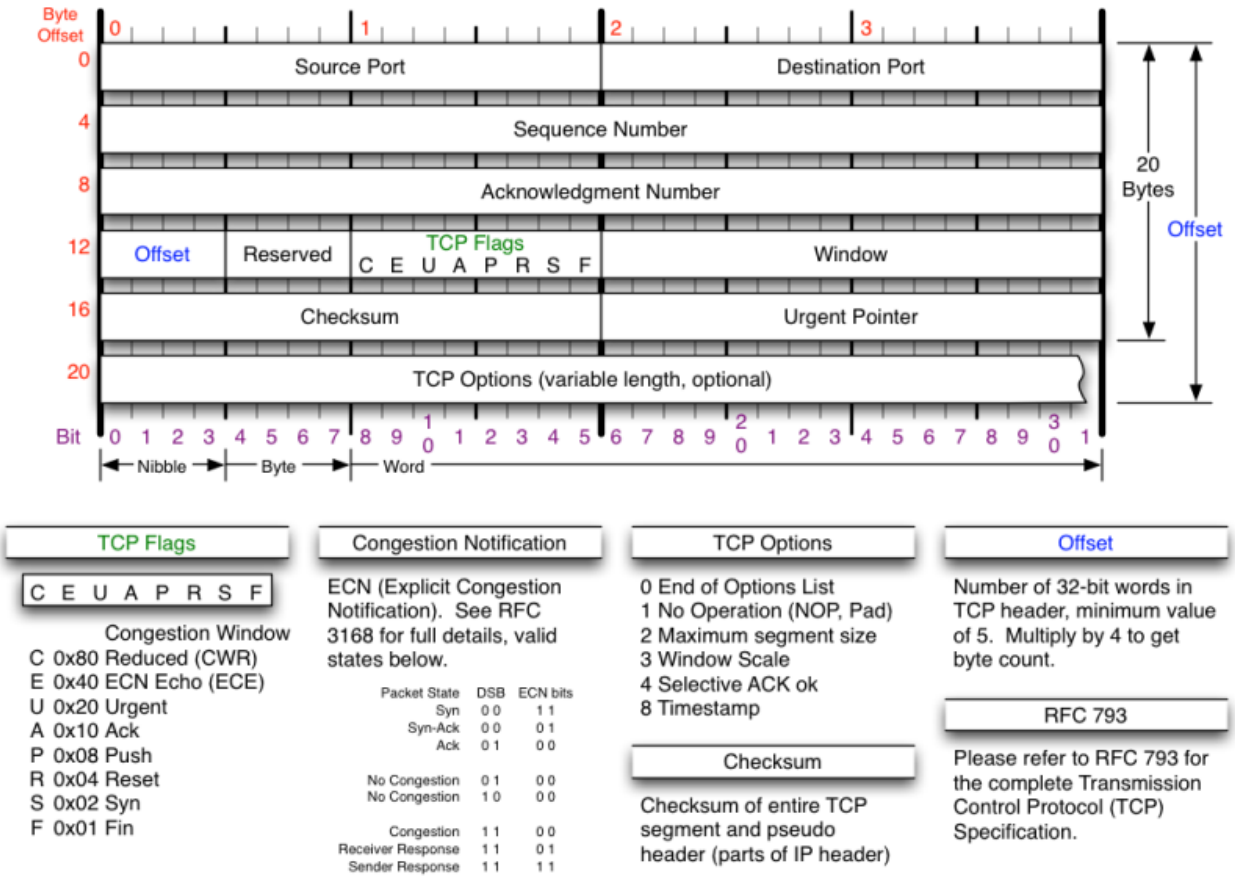


Figura 5 – Cabeçalho TCP (LIAO et al., 2010).

Os campos porta origem *Source Port* e porta destino *Destination Port* servem como

portas de comunicação entre um par cliente-servidor. Os campos Número de Sequência *Sequence Number* e *Acknowledge Number* são utilizados para controle. Os segmentos 0 a 8 *(Byte Offset)* possuem todos os seus campos de *bits* utilizados. No Segmento 12, o campo *reserved* é um dos poucos campos do cabeçalho TCP que não estão em uso. O *Offset* especifica o tamanho de uso do cabeçalho, enquanto que o campo de Janelamento *Window* especifica o tamanho da janela de recepção. O campo *TCP Flags* é utilizado para sinalização do TCP. No segmento 16, *Checksum* é utilizado para verificação de erros e para a segurança (caso o pacote tenha sido manipulado), e o ponteiro urgente *Urgent Pointer* é utilizado para priorizar a entrega de um ou alguns pacotes específicos, geralmente em conjunto com a *flag* PSH. Por fim, o último campo *Options* é responsável pela sinalização adicional descrita na figura. Entretanto, o campo *Options* não é totalmente utilizado.

O campo *Options* do TCP possui um espaço vazio, do qual é preenchido pelo MPTCP. Na subseção 2.2.3 detalharemos como o MPTCP utiliza este campo para fazer a sua sinalização.

2.2.3 Multipath-TCP

Como abordado brevemente nas subseções anteriores, o 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 utiliza as interfaces disponíveis para distribuição do tráfego. Isso acontece devido a criação de subfluxos (BARRÉ; PAASCH; BONAVENTURE, 2011).

Para criar subfluxos, o MPTCP utiliza uma implementação da pilha TCP, como é mostrado na Fig. 6.

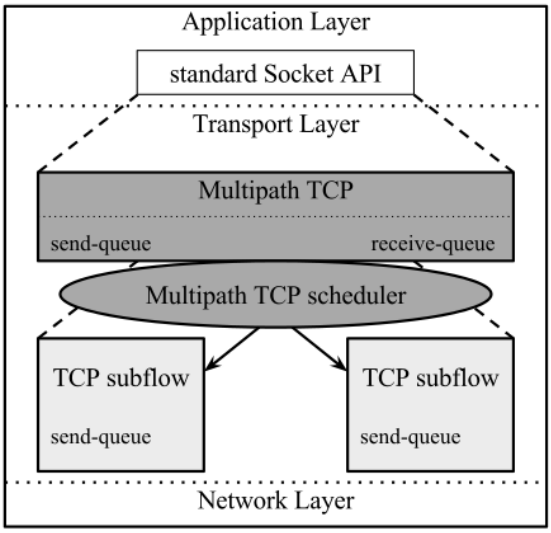


Figura 6 – Pilha do TCP com MPTCP (PAASCH; KHALILI; BONAVENTURE, 2013).

A camada de aplicação faz interface direta com a camada de transporte via

socket API. Entretanto, atualmente somente a camada de transporte faz interface com a subcamada do MPTCP. Nesta subcamada, a camada é virtualizada e adiciona-se um gerenciador de subfluxos. Ocorre então, que para um fluxo TCP que a aplicação utiliza, a subcamada do MPTCP virtualiza e deriva um único fluxo em diversos subfluxos. A partir de então, os dados são distribuídos entre os subfluxos utilizando o algoritmo de escalonamento Round Robin. Cada subfluxo, corresponde a uma conexão TCP comum, muito embora esta possua o cabeçalho do TCP incrementado com o MPTCP.

A transmissão de um pacote MPTCP ocorre inicialmente com uma requisição de capacidade de transmissão *multipath*, iniciada pelo lado do emissor. Caso o receptor possua MPTCP habilitado, ambos negociam chaves criptográficas para autenticação. Após a autenticação, negocia-se quantos subfluxos serão abertos, previamente selecionado pelo administrador do sistema. Com isso, a sinalização do MPTCP cria um identificador *(token)* para cada conexão e este *token* será carregado nos subfluxos de cada conexão para identificá-los no momento da remontagem no receptor. A Fig. 7 demonstra o diagrama de sequência de autenticação e inicialização dos subfluxos do MPTCP.

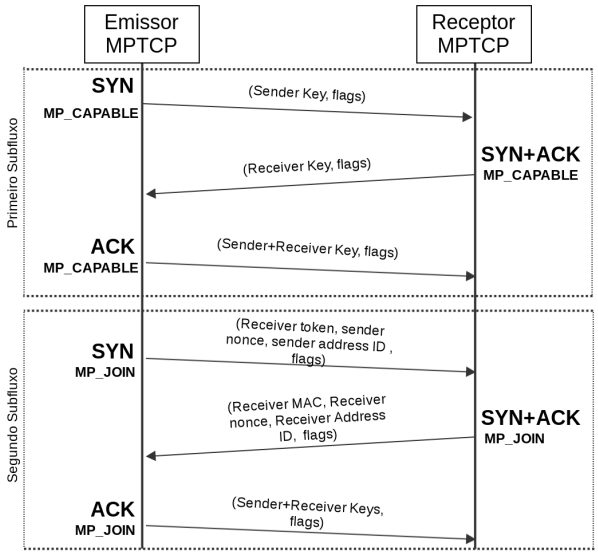


Figura 7 – Diagrama de Sequência MPTCP.

Uma sessão MPTCP começa quando se acrescenta ao cabeçalho do MPTCP a 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 à 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; BONAVENTURE, 2014), como é mostrado na Fig. 7.

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, permitirá a criação da nova conexão TCP e adicioná-lo à sessão MPTCP (PAASCH; KHALILI; BONAVENTURE, 2013).

Depois de uma sessão do qual há mais de um subfluxo, cabe aos sistemas em cada extremidade para decidir 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 é uma conexão TCP normal, com seus próprios números de seqüência, mas a sessão como um todo tem um número de seqüência em 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., 2011; 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* criptografico da chave do receptor que foi trocado no *handshake* a inicial do MP_CAPABLE. Depois de ter o *hash* criptográfico, 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).

2.3 Evolução dos Protocolos de Roteamento

Protocolos de roteamento são protocolos responsáveis por rotear protocolos como TCP, UDP, SCTP, etc. Nesta Seção, descrevemos alguns protocolos de roteamento que estão diretamente ou parcialmente relacionados a este trabalho.

2.3.1 Roteamento IP

Roteamento IP engloba a camada inferior do modelo TCP: Interface Física. Na camada de interface física, é feito o encaminhamento pelo endereço MAC e subsequentemente transmitida a informação. Na camada de *internet*, o modelo de encaminhamento é definido pelo endereçamento IP e é chamado de roteamento. Isso acontece devido a rede IP utilizar técnicas de troca de tabelas de endereçamento e roteamento que permitem

calcular distâncias e melhores caminhos, fazendo com que pacotes possam ser roteados ao invés de simplesmente serem encaminhados.

A desvantagem de utilizar o roteamento *(best effort)* surge necessariamente quando se há a necessidade de balancemanento de carga. Como exemplo do MPTCP, se um par inicia uma conexão com 5 subfluxos, todos estes serão potencialmente roteados pela mesma rota. Quando usado IPs diferentes por subfluxo (no caso de haver diversas interfaces ou IPs associados a somente uma) é possível que a rede IP roteie os subfluxos entre as demais rotas, porém associar muitos IPs a um dispositivo não é claramente viável, de modo que isso possa limitar uso de uma quantidade muito grande de máquinas na rede e/ou prejudicar o funcionamento e segurança de aplicações que não usam MPTCP. Portanto, o uso de diversos endereços IPs só é recomendado quando se usam diferentes interfaces conectadas a diferentes ISPs para que seja possível a agregação de banda e o *multipath*.

Para que o MPTCP ou qualquer protocolo que utilize subfluxos seja corretamente roteado em uma conexão fim-a-fim é necessário um protocolo de roteamento capaz de garantir a distribuição destes subfluxos entre rotas diferentes, utilizando um método de roteamento diferente. A Fig. 8 demonstra como o MPTCP é roteado em uma topologia *Fat-Tree*.

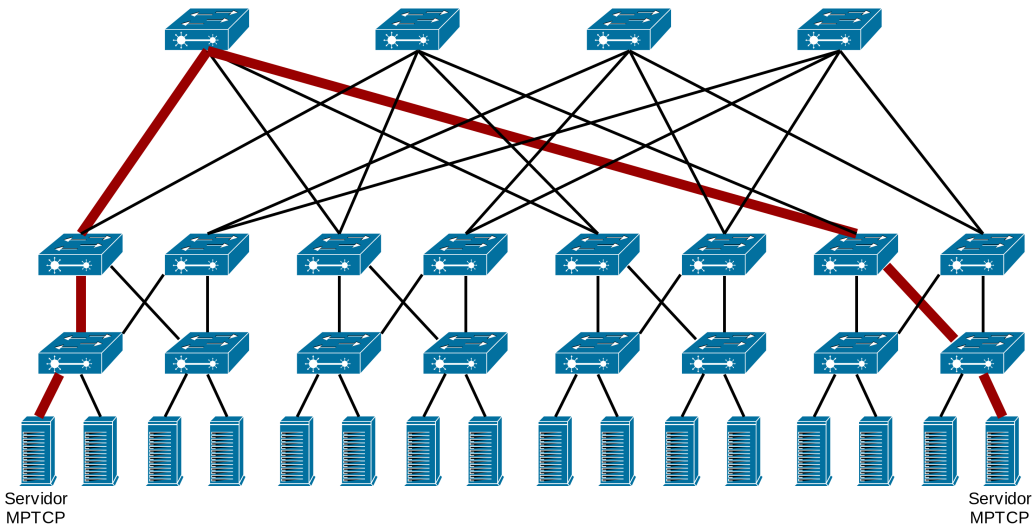


Figura 8 – *Fat-Tree* com roteamento IP.

Um par de servidores MPTCP tem seus subfluxos encaminhados por um mesmo caminho até o destino, embora haja diversas alternativas de caminhos ao longo da rede. Devido aos subfluxos não possuirem IPs diferentes, a rede IP não os distingue dos demais fluxos de uma mesma origem e então os roteia por um mesmo caminho, não importando a quantidade de subfluxos.

Como será visto a seguir, o protocolo ECMP faz a distribuiçao de fluxos baseado nas camadas de rede e transporte, isto é, o ECMP utiliza além do IP, portas e tipo de protocolo de controle para avaliar por qual caminho será roteado numa topologia que possui múltiplos caminhos entre origem e destino *(looping)*.

2.3.2 Roteamento com 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.

2.4 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. (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 tráfego (TE). Entretanto, o trabalho de (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 (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; 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 subfluxos 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.

Finalização do Capítulo

Neste capítulo, apresentamos os conceitos básicos relacionados ao entendimento da dissertação. Foram apresentados: A arquitetura de uma rede OpenFlow, o protocolo MPTCP e suas características de funcionamento e o protocolo de roteamento ECMP com suas funcionalidades. Finalizamos o capítulo com os trabalhos relacionados à dissertação.

3 Roteamento com MPTCP e OpenFlow

3.1 MultiFlow

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

3.1.1 Módulo MultiFlow

O Componente MultiFlow é um componente desenvolvido para melhorar o *throughput* em redes de datacenters *OpenFlow-enabled*. Uma rede OpenFlow é dividida em dois planos operacionais, conforme é mostrado na Fig. 9:

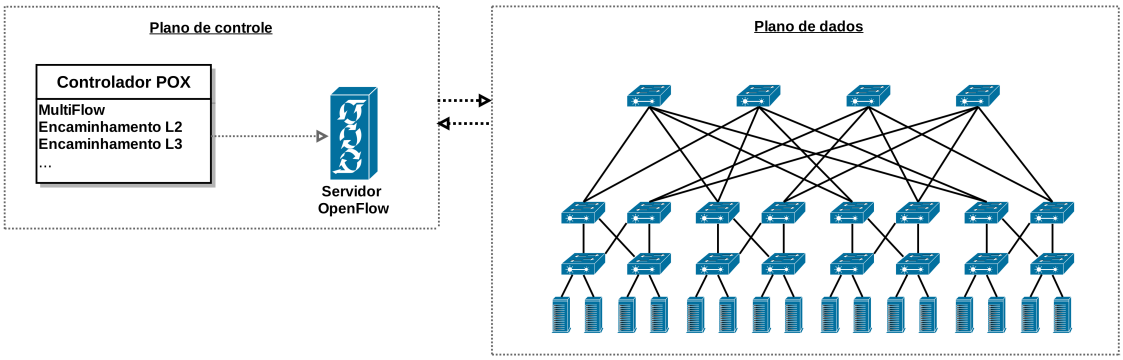


Figura 9 – Arquitetura do MultiFlow.

O plano de dados é o local no qual os pacotes estão sendo encaminhados entre os *switches* utilizando 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 programar *match* e *rules* para uma malha de *switches*.

Um pacote MPTCP convencionalmente utiliza o cabeçalho TCP para encapsular mensagens de controle. Então, quando uma aplicação se comunica com um servidor, mensagens de controle são trocadas para tentar estabelecer uma conexão MPTCP. O MultiFlow identifica tais mensagens de controle inspecionando o *token* que identifica

subfluxos da mesma conexão MPTCP. Além disso, como os pacotes são criados como uma extensão do TCP, é possível criar regras OpenFlow utilizando os números de portas TCP (tupla de 5 campos). Com isso, é verificado se o pacote pertence à mesma conexão MPTCP e posteriormente definem-se as regras para espalhá-las em caminhos disjuntos.

Neste trabalho, projetamos um componente chamado MultiFlow para roteamento de subfluxos em caminhos disjuntos. O MultiFlow é um componente projetado em um controlador POX. O controlador POX é um controlador OpenFlow desenvolvido em linguagem de programação Python. No POX é utilizado OpenFlow 1.1 para estabelecer a comunicação entre *switches* e controlador. A Fig. 10 mostra o módulo do MultiFlow na arquitetura do controlador POX.

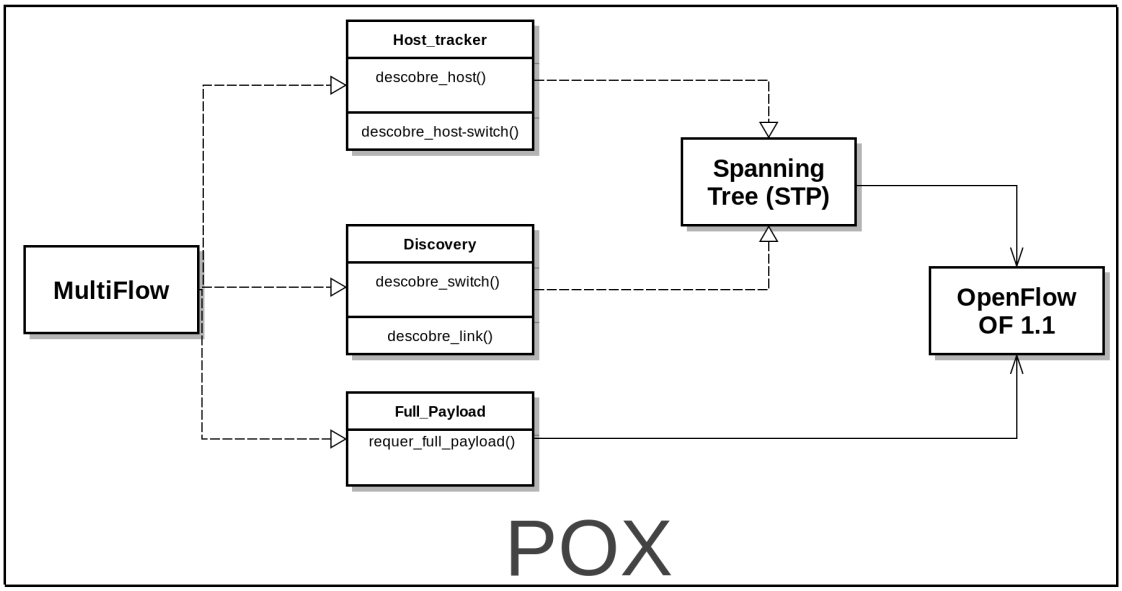


Figura 10 – MultiFlow Dentro da Arquitetura do POX.

O MultiFlow começa escutando o componente *Discovery*. Isto é necessário porque o MultiFlow utiliza este componente para montar a topologia de um modo pró-ativo, isto é, antes de qualquer fluxo ser processado no controlador. 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 componente *Host_tracker* para detectar qual porta um determinado *host* está conectado em cada *switch*. Em seguida, é possível preparar o MultiFlow para iniciar o processamento do protocolo MPTCP. O componente *Discovery* também é utilizado para descobrir todos os *switches* e suas respectivas portas na rede, do qual usa-se o Protocolo *Spanning-Tree* (STP). Novamente, o componente *Host_tracker* detecta a disponibilidade dos *hosts*.

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 é recebido no

controlador e é responsável pela distribuição dos MP_JOINS em caminhos disjuntos. Em suma, o algoritmo dos MP_JOINS 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 do qual 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
```

Algoritmo 1: Inspeção e encaminhamento do MP_CAPABLE .

Inicialmente, separa-se o MP_CAPABLE e MP_JOIN um do outro. 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 para o componente MultiFlow e são encaminhados utilizando qualquer outro protocolo da rede. Isto significa que toda *Flow_entry* no componente MultiFlow é um cabeçalho do TCP.

A fim de identificar uma mensagem MP_CAPABLE, o MultiFlow filtra o SYN do cabeçalho TCP (linha 1). Em seguida, identifica se o TCP SYN tem o cabeçalho MPTCP populado com MPTCP _ID (linha 2), observando se o campo *kind* possui o valor 30 (decimal) como o valor designado pelo IANA para o MPTCP. Em caso positivo, 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, MultiFlow adiciona 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
```

Algoritmo 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* de cada mensagem MP_JOIN (linha 3) e o associa 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 tal topologia individual. A Fig. 11 mostra a ideia com duas conexões diferentes MPTCP (MPTCP1 e MPTCP2) cada uma com dois subfluxos: SF1.1 , SF 1.2, SF2.1 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 seguida, suponha que o SF1.1 é recebido no controlador. Uma vez que este é o primeiro subfluxo, o MultiFlow usa a topologia original como mostrado na Fig. 11 (a) e encontra o caminho mais curto entre o cliente e o servidor MPTCP (linha 4). Suponha que o caminho escolhido para o encaminhamento do SF1.1 é dado pelos *switches* 1, 2 e 5. O MultiFlow define a rota para tal subfluxo utilizando regras OpenFlow (linhas 5 e 6) e corta a rota utilizada (linha 7), resultando na topologia mostrada na Fig. 11 (b). Essa nova topologia é então armazenada associando-a com o *token* que identifica esta conexão MPTCP (linha 8). Agora, suponha que há um segundo subfluxo da mesma conexão MPTCP: SF1.2. O

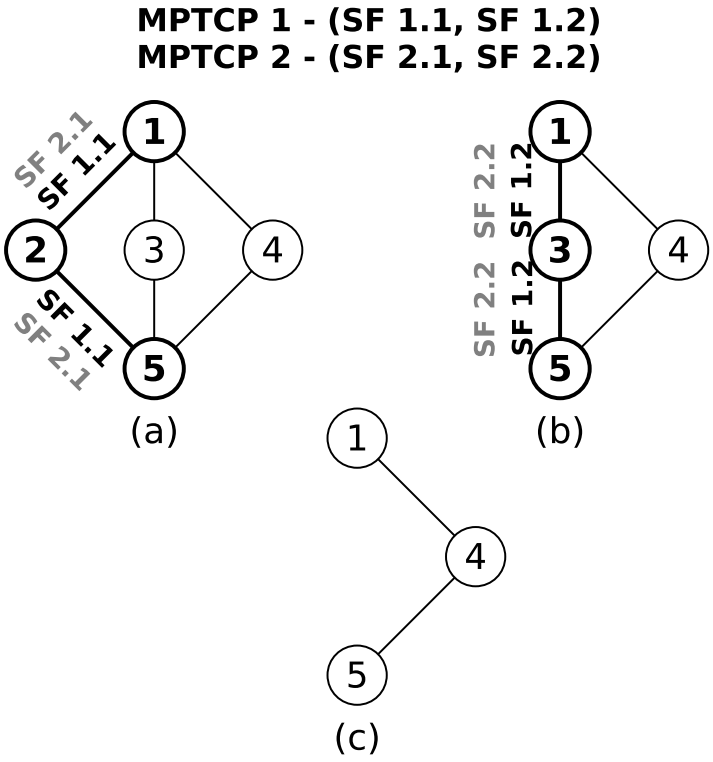


Figura 11 – Virtualização da Topologia com MultiFlow.

MultiFlow verifica se já existe subfluxos desta conexão MPTCP na rede (linha 2) e utiliza a topologia podada da Fig. 11 (b) (linha 11) para encontrar um novo caminho disjunto para SF1.2 (linha 12). Novamente, o MultiFlow corta apenas o percurso utilizado, resultando na nova topologia da Fig. 11 (c). Ao fazer isso recursivamente, o Multiflow garante que subfluxos da mesma conexão MPTCP utilizem mútiplos 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 (Fig. 11 (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 é aplicado o algoritmo explicado acima, 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. Pode-se interpretar 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. 11 é possível observar que a topologia original possui três caminhos disjuntos, o que significa que apenas três subfluxos em cada conexão MPTCP podem ser encaminhadas 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, como demonstramos na avaliação feita na próxima seção, o *throughput* 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.

Finalização do Capítulo

Neste capítulo apresentamos a metodologia que possibilita que o MPTCP seja roteado em uma rede OpenFlow. Para isso, criamos um módulo no controlador OpenFlow POX para inspecionar e distribuir os subfluxos MPTCP em caminhos disjuntos, ao longo da rede.

4 Resultados e Análise

A arquitetura MultiFlow foi implementada e testada para validar a abordagem deste trabalho. O MultiFlow foi desenvolvido e testado com OpenFlow 1.1 e MPTCP V.88. 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*. Para gerar tráfego e validar o desempenho, utilizamos o *iperf*. Quando o *iperf* é utilizado com o *kernel* MPTCP, ele passa a utilizar o protocolo MPTCP automaticamente, caso utilize um teste cliente-servidor TCP.

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. No Apêndice A, encontram-se detalhes sobre as estatísticas dos resultados experimentais.

4.1 MultiFlow vs. Spanning-Tree

4.1.1 Análise do *Throughput*

Nesta Subseção, analisamos o *Throughput* quando utilizamos o MultiFlow. Em nossos experimentos, utilizamos a topologia mostrada na Fig. 12, que possui três caminhos disjuntos compostos por *switches* OpenFlow. Nós utilizamos um par MPTCP (cliente - setvidor) conectado aos *switches* 1 e 5, respectivamente. Há também dois *hosts* na rede responsáveis pelo envio de um tráfego UDP de 7Mbps, utilizando a rota 1, 3 e 5.

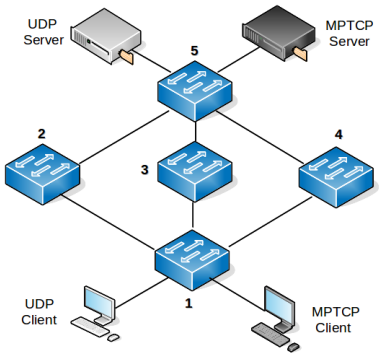
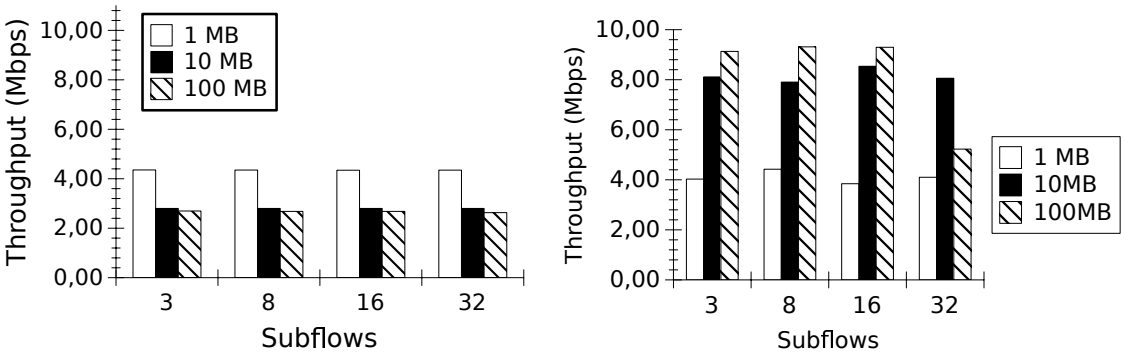


Figura 12 – Topologia experimental.

Os *hosts* irão transferir arquivos de 1MB, 10MB e 100MB. Para cada arquivo, há apenas uma conexão MPTCP com 3, 8, 16 e 32 subfluxos. A idéia nesse experimento

é comparar o MPTCP puro usando o STP (sem distribuir os subfluxos por caminhos disjuntos) versus MPTCP usando o MultiFlow (distribuindo fluxos por caminhos disjuntos).

Fig. 13 (a) mostra o cenário em que todos os subfluxos MPTCP são enviados usando a mesma rota (*switches* 1, 3 e 5). Note que essa rota é a mesma rota do tráfego UDP. É possível observar também que os resultados da Fig. 13 (a) mostram que ao aumentar o número de subfluxos, o *throughput* da conexão MPTCP não aumenta. Esse comportamento é comum, tal que não há uso de caminhos diferentes para os subfluxos. Entretanto, observamos a Fig. 13 (b) que mostra os resultados quando o MPTCP é roteado com o MultiFlow. Com o roteamento do MultiFlow, subfluxos de uma mesma conexão MPTCP são roteados utilizando três caminhos disjuntos e o *throughput* aumenta para as transmissões de arquivos com 10MB e 100MB. Isso occorre devido ao *congestion control* aumentar a janela nos caminhos com melhor *throughput* (rota 1, 2, 5 e rota 1, 4, 5).



(a) Subfluxos MPTCP utilizando a mesma rota. (b) Multiflow usando caminhos disjuntos.

Figura 13 – MPTCP roteado pelo Multiflow.

Também é possível observar que devido ao aumento da quantidade de subfluxos, não há ganho de *throughput*. Além disso, quando há 32 subfluxos, o *throughput* decai devido a sobrecarga na gerência de múltiplos subfluxos. Então, a conclusão com esse teste, divide-se em dois blocos: primeiramente, roteamento de subfluxos MPTCP utilizando rotas diferentes aumenta o *throughput*; segundo, aumentando a quantidade de subfluxos sem aumentar a quantidade de caminhos disjuntos não ajuda a aumentar o *throughput* e pode até piorar os resultados finais. Então, baseado nestes resultados, seria plausível concluir que criar subfluxos acima da quantidade de rotas disjuntas disponíveis só prejudicaria o desempenho.

4.1.2 Testes em Gargalo Compartilhado *(Shared Bottleneck)*

Nessa Subseção, serão validados os benefícios do MultiFlow quando múltiplos pares MPTCP estão presentes e se comunicam na rede. O propósito deste experimento é comparar conexões TCP com o MultiFlow. Utilizamos a topologia da Fig. 14 que faz uso

de três caminhos disjuntos e possui 3 pares MPTCP (cliente – servidor) conectados aos *switches* 1 e 5 respectivamente. Neste cenário não há tráfego UDP.

Para este experimento, novamente será transferido arquivos de 1MB, 10MB e 100MB, porém ao invés de utilizarmos 3, 8, 16 e 32 subfluxos, o Multiflow irá permitir somente 3 subfluxos por cada conexão MPTCP, devido a conclusão na Subseção 4.1.1 mostrado na Fig. 13 (b).

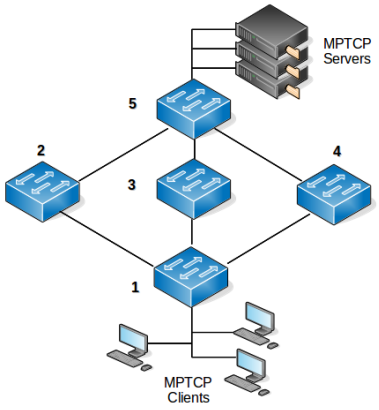
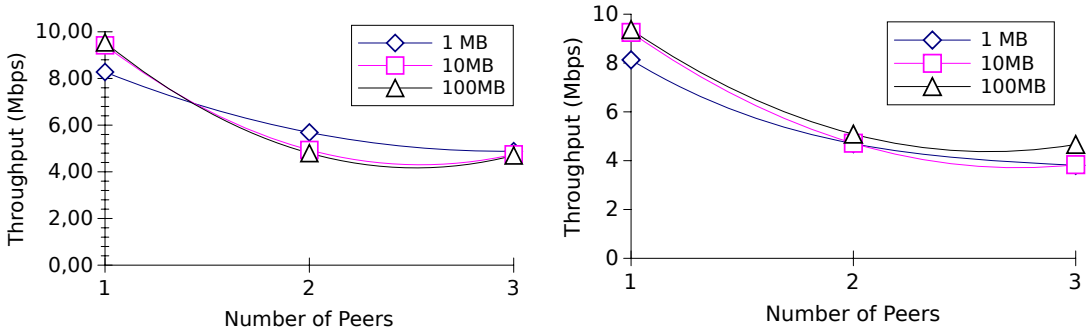


Figura 14 – Topologia da rede OpenFlow: 3 Servidores e 3 Clientes, ambos MPTCP.

A Fig. 15 (a) mostra os resultados para o envio do tráfego TCP (sem MPTCP) utilizando a rota composta pelos *switches* 1, 3 e 5. Note que quando a quantidade de conexões TCP aumenta, o *throughput* diminui devido aos fluxos compartilharem a mesma rota. A Fig. 15 (b) mostra o cenário com o MultiFlow. Nesse caso, cada par abre uma conexão MPTCP com três subfluxos, totalizando 9 subfluxos trafegando pela rede. Os subfluxos são enviados usando três caminhos disjutos dados pelo algoritmo do MultiFlow que separa subfluxos de uma mesma conexão MPTCP em caminhos disjuntos.

A primeira vantagem acontece quando se observa o *throughput* para 100MB. Neste caso, o *throughput* é maior quando é utilizado o MultiFlow, sendo uma relação de 4,67Mbps contra 4.06Mbps do STP. A segunda vantagem em usar o MultiFlow neste cenário é a tolerância a falhas. No cenário onde não é utilizado MultiFlow, se um *link* ou dispositivo falhar na rota 1, 3 e 5, todas as conexões TCP irão ser interrompidas.

Conclui-se nesse experimento que o MultiFlow é útil para fluxos longos, também conhecidos como fluxos elefantes. Para fluxos curtos, o desempenho em termos de *throughput* é alterado.



(a) Três conexões TCP usando o mesmo caminho (sem MultiFlow).

(b) Três conexões MPTCP cada uma com três subfluxos (com MultiFlow).

Figura 15 – TCP Reno vs. MPTCP com Multiflow.

4.1.3 Curva Assintótica e Setup Time

Para validar a complexidade do algoritmo do MultiFlow e seu impacto na rede, analisamos o comportamento assintótico e *setup time* comparando-o com o STP

A Fig. 16 mostra o melhor, pior e médio casos em termos de tempo de execução do algoritmo do MultiFlow. O tempo considerado é o tempo necessário somente para inspecionar o pacote MPTCP. O tempo para aplicar as regras OpenFlow na rede são desconsiderados neste teste.

Na execução do teste, utilizamos um arquivo de captura de pacotes (pcap). Neste arquivo há um conjunto de 14 pacotes para cada mensagem MP_CAPABLE e MP_JOIN. Também há 14 mensagens TCP sem MPTCP.

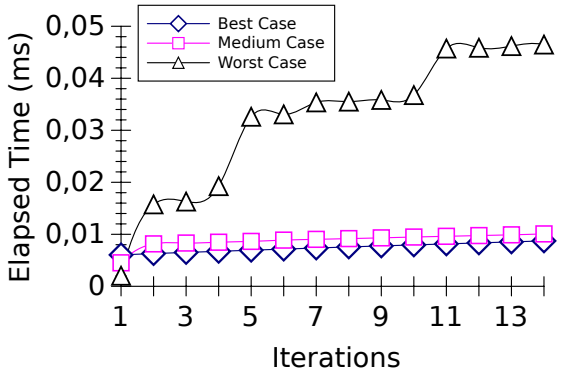


Figura 16 – Analise assintótica.

Observa-se que o melhor caso representa o cenário em que o pacote TCP não é uma mensagem MPTCP e então o MultiFlow não faz nada com o pacote. O pior caso é quando o pacote possui uma mensagem JOIN. Neste cenário, o MultiFlow precisa recorrer a topologia virtual correspondente para que a conexão MPTCP (utilizando o *token*) encontre a nova rota, delete esta nova rota utilizada e grave essa nova topologia virtual para uso

futuro. O caso médio acontece quando o pacote possui uma mensagem MP_CAPABLE e neste caso o MultiFlow precisa somente encontrar uma rota para enviar a mensagem.

O *setup time* foi calculado com o RTT das mensagens CAPABLE e JOIN, utilizando MultiFlow vs. STP. Utilizamos o STP nativo do controlador POX. O STP cria proativamente todas as rotas de todos os pares de *hosts* na rede. Então, o resultado esperado é que o *setup time* para STP seja baixo quando comparado ao *setup time* para o MultiFlow. Entretanto, como de conhecimento, o STP é uma solução extrememente simples para encaminhamento L2 e não leva em conta qualquer característica que o MultiFlow considera. A Fig. 17 demonstra os números.

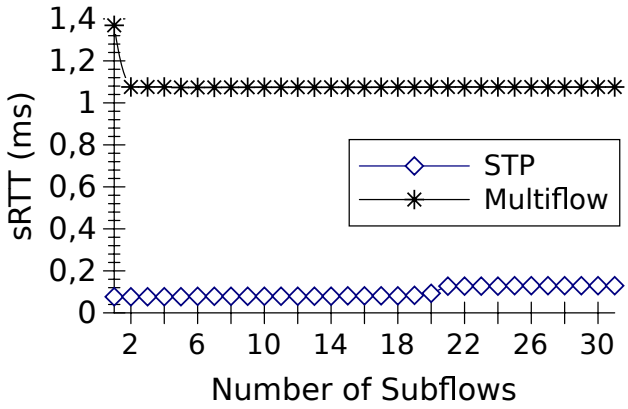


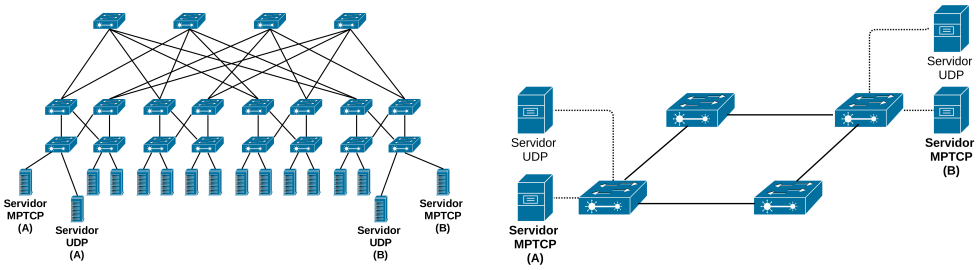
Figura 17 – Setup time.

4.2 MultiFlow vs. ECMP

Nesta seção, comparamos o MultiFlow com o ECMP e demonstramos o *throughput* do MPTCP quando usado em conjunto com o MultiFlow versus ECMP. Para utilizar o ECMP(CHIESA et al., 2014), utilizamos um módulo do POX chamado RiplPOX (HELLER, 2014) que emula o ECMP com os dois modos de operação apresentados no Capítulo 2 Nos experimentos, utilizamos a topologia *FatTree* utilizada em *datacenters* e que é apresentada na Fig. 18 (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 9Mpbs o que faz com que haja um tráfego de 9 Mbps, ocupando um dos caminhos da topologia.

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.

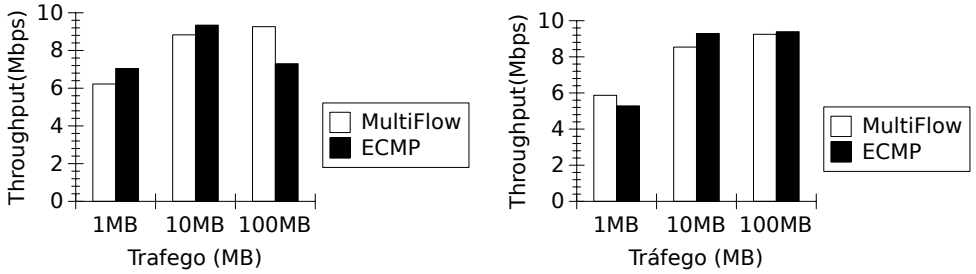


(a) Topologia FatTree (b) Topologia em *looping*.

Figura 18 – Topologias utilizadas nos experimentos

A Fig. 19 (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. 19 (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 19 – MPTCP com 2 subfluxos roteados por ECMP e MultiFlow.

Os resultados mostrados na Fig. 19 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 18 (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. 20, 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.

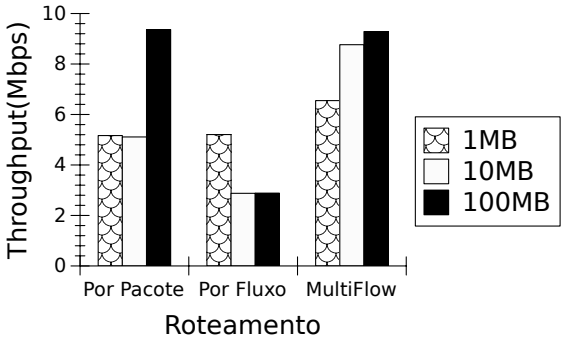


Figura 20 – Resultados para topologia em *looping*.

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 desempenho 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.

Conclusão e Trabalhos Futuros

Nesta dissertação, desenvolvemos um módulo chamado MultiFlow para otimizar a distribuição de subfluxos MPTCP em *hosts single-homed*. Experimentamos diferentes cenários, analisando o MPTCP em topologias de *datacenters OpenFlow-enabled*.

O primeiro beneficio da separação dos subfluxos em caminhos disjuntos é a resiliência. O detalhe é que ao contrário do ECMP ou de qualquer outro algoritmo que venha a ser utilizado como protocolo de roteamento é que o MultiFlow é capaz de forma passiva, ajustar a quantidade de subfluxos conforme a quantidade de caminhos disjuntos, resolvendo indiretamente um dos problemas mais estudados na literatura do MPTCP: ajustar a quantidade de subfluxos de uma conexão MPTCP. Por hora, o ajuste que obtivemos baseia-se no fato de que a criação de um número de subfluxos maior do que a quantidade de caminhos disjuntos na rede não ajuda a aumentar o *throughput*.

Um resultado visível obtido pelo MultiFlow é o aumento do *throughput* para fluxos longos (fluxos elefantes). Notamos que o benefício do MultiFlow varia conforme cada cenário utilizando tanto o STP quanto ECMP. Porém, é unânime que em todos os cenários o MultiFlow mostrou-se eficiente e mais estável para fluxos elefantes.

Diferentemente das outras soluções apresentadas como soluções concorrentes ao MultiFlow, o MultiFlow é independente de topologia e como foi aplicado no emulador OpenFlow Mininet, prevê-se que o MultiFlow possa ser aplicado sem dificuldades em uma rede real. Futuramente, pretendemos utilizar o *testbed* NorNet (GRAN; DREIBHOLZ; KVALBEIN, 2014) para testar o comportamento do MultiFlow em um cenário real.

Como continuidade do trabalho, prevê-se também que o MultiFlow possa ser avaliado com outros tipos de protocolos multipath, tais como o SCTP. Além disso, prevê-se a validação do ajuste de quantidade de subfluxos do MultiFlow, quando comparado a um ajuste feito pela própria API do MPTCP (SCHARF; FORD., 2013) (atualmente em desenvolvimento por um aluno de Mestrado). A ideia principal desta API é permitir que cada aplicação decida quantos subfluxos deverão ser criados em função de suas demandas e tamanhos de fluxos. Assim, por exemplo, fluxos considerados elefantes poderiam ser quebrados enquanto que fluxos curtos (ratos) não seriam divididos. Ao usarmos este tipo de ajuste dinâmico, espera-se um melhor aproveitamento da capacidade que o MPTCP oferece para cada aplicação.

Referências

ARYE, M. et al. A Formally-verified Migration Protocol for Mobile, Multi-homed Hosts. In: *Network Protocols (ICNP), 2012 20th IEEE International Conference on*. [S.l.: s.n.], 2012. p. 1–12. Citado na página 31.

BARRÉ, S. et al. Experimenting with Multipath TCP. *ACM SIGCOMM Computer Communication Review*, ACM, v. 41, n. 4, p. 443–444, 2011. Citado 2 vezes nas páginas 21 e 35.

BARRÉ, S.; PAASCH, C.; BONAVENTURE, O. Multipath TCP: From Theory to Practice. In: *NETWORKING 2011*. [S.l.]: Springer, 2011. p. 444–457. Citado na página 33.

BOUCADAIR, M.; JACQUENET, C. *Software-Defined Networking: A Perspective from within a Service Provider Environment*. 2014. RFC 7149. Citado na página 26.

BURGESS, M. *Principles of Network and System Administration*. 2nd. ed. [S.l.]: John Wiley & Sons, Ltd., 2000. Citado na página 25.

II; Robert M. Burke. *System for regulating access to and distributing content in a network*. 2012. US 8122128. Disponível em: <http://www.patentlens.net/patentlens/patent/US_8122128/>. Citado na página 26.

CALYAM, A. P. *PERFORMANCE MEASUREMENT AND ANALYSIS OF H.323 VIDEOCONFERENCE TRAFFIC*. 117 p. Dissertação (Mestrado) — The Ohio State University, 2002. Citado na página 26.

CHIESA, M. et al. Traffic Engineering with Equal-Cost-Multipath: An Algorithmic Perspective. In: *INFOCOM, 2014 Proceedings IEEE*. [S.l.: s.n.], 2014. p. 1590–1598. Citado 2 vezes nas páginas 21 e 51.

DUAN, J. et al. Responsive Multipath TCP in SDN-based Datacenters. *The IEEE International Conference on Communications (ICC'15)*, 2015. Citado na página 38.

FORD, A. et al. TCP Extensions for Multipath Operation with Multiple Addresses. *IETF RFC*, 2013. Citado na página 35.

GRAN, E. G.; DREIBHOLZ, T.; KVALBEIN, A. NorNet Core – A Multi-Homed Research Testbed. *Computer Networks, Special Issue on Future Internet Testbeds*, v. 61, p. 75–87, mar. 2014. ISSN 1389-1286. ISSN 1389-1286. Disponível em: <https://www.simula.no/sites/www.simula.no/files/publications/Simula.simula.2236.pdf>. Citado na página 55.

GRINNEMO, K.-J.; BRUNSTROM, A.; CHENG, J. Using Concurrent Multipath Transfer to Improve the SCTP Startup Behavior for PSTN Signaling Traffic. In: *Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on*. [S.l.: s.n.], 2014. p. 772–778. Citado na página 31.

HELLER, B. *RiplPOX: ECMP for POX Controller*. 2014. <https://github.com/brandonheller/riplpox>. Accessed: 01-04-2015. Citado na página 51.

LIAO, J. et al. Introducing multipath selection for concurrent multipath transfer in the future internet. In: ELSEVIER. *Computer Networks*. [S.l.], 2010. p. 1024–1035. Citado 3 vezes nas páginas 15, 30 e 32.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. In: *ACM SIGCOMM Computer Communication Review, 2008*. [S.l.: s.n.], 2008. p. 69–74. Citado na página 27.

MCKEOWN, N.; GIROD, B. *Clean-Slate Design for the Internet*: A research program at stanford university. [S.l.], 2006. Citado na página 26.

MING, L. et al. MPTCP Incast in Data Center Networks. *Communications, China*, v. 11, n. 4, p. 25–37, April 2014. ISSN 1673-5447. Citado na página 38.

MUNTANER, G. R. de T. *Evaluation of OpenFlow Controllers*: A research program at stanford university. [S.l.], 2012. Citado na página 29.

NOTE, C. T. *Performance Tuning Basics:*: Switching paths. [S.l.], 2007. Disponível em: <http://www.cisco.com/c/en/us/support/docs/ios-nx-os-software/ios-software-releases-121-mainline/12809-tuning.html#switchports>. Citado 3 vezes nas páginas 15, 27 e 28.

PAASCH, C.; BONAVENTURE, O. multipath tcp. *Communications of the ACM*, ACM, v. 57, n. 4, p. 51–57, 2014. Citado na página 35.

PAASCH, C. et al. Exploring mobile/WiFi Handover with Multipath TCP. *In Proceedings of the 2012 ACM SIGCOMM workshop on Cellular networks: operations, challenges, and future design (CellNet '12). ACM, New York, NY, USA, 31-36.*, 2012. Citado na página 31.

PAASCH, C.; KHALILI, R.; BONAVENTURE, O. On the Benefits of Applying Experimental Design to Improve Multipath TCP. In: ACM. *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. [S.l.], 2013. p. 393–398. Citado 3 vezes nas páginas 15, 33 e 35.

POL, R. Van-der et al. Multipathing with MPTCP and OpenFlow. *High Performance Computing, Networking, Storage and Analysis (SCC 2012)*, 2012. Citado 3 vezes nas páginas 21, 22 e 38.

RAICIU, C. et al. Improving Datacenter Performance and Robustness with Multipath TCP. *In Proceedings of the ACM SIGCOMM 2011 conference (SIGCOMM '11)*, 2011. Citado 2 vezes nas páginas 31 e 38.

SANDRI, M. et al. 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)*. [S.l.: s.n.], 2015. Citado na página 39.

SCHARF, M.; FORD., A. *Multipath TCP (MPTCP) Application Interface Considerations*. 2013. RFC 6897. Citado 2 vezes nas páginas 38 e 55.

SDN Architecture: Overview Transparent. <http://upload.wikimedia.org/wikipedia/commons/e/e6/SDN-architecture-overview-transparent.png>. Accessed: 01-04-2015. Citado 2 vezes nas páginas 15 e 26.

SONKOLY, B. et al. SDN Based Testbeds for Evaluating and Promoting Multipath TCP. In: *Communications (ICC), 2014 IEEE International Conference on*. [S.l.: s.n.], 2014. p. 3044–3050. Citado na página 38.

Stevens, R. and Fenner B. and Rudoff, A. *Network Programming*: The sockets networking api. [S.l.]: Addison-Wesley, 2009. Citado na página 31.

Varghese, G. *Network Algorithmics*: An interdisciplinary approach to design fast networked devices. [S.l.]: Morgan Kaufmann, 2005. Citado na página 27.

WEBSPROCKET Selected By Supranet Consortium to Enable the Internet With Smart Packet Technology;: Platform unifies supranet management through java and oracle. 2001. Disponível em: <http://www.thefreelibrary.com/Websprocket+Selected+By+Supranet+Consortium+to+Enable+the+Internet...-a071935216>. Citado na página 26.

WISCHIK, D. Multipath: A new control architecture for the internet: Technical perspective. *Communications of the ACM*, ACM, v. 54, n. 1, p. 108–108, 2011. Citado na página 35.

ZHOU, J. et al. WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers. *In Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14)*, 2014. Citado 2 vezes nas páginas 21 e 38.

APÊNDICE A – Estatística Detalhada dos Resultados Experimentais

A.1 Qtd. de amostras

Para cada experimento, utilizamos a quantidade de 30 amostras. A Seção A.2 demonstra a aproximação das amostras.

A.2 Desvio Padrão dos gráficos

Fig. 13(a)	3-sub	8-sub	16-sub	32-sub
1 MB	0,02	0,01	0,01	0,02
10 MB	0,00	0,00	0,00	0,00
100 MB	0,03	0,00	0,00	0,05

Fig. 13(b)	3-sub	8-sub	16-sub	32-sub
1 MB	0,35	0,03	0,97	0,37
10 MB	0,16	0,40	0,61	0,25
100 MB	0,41	0,02	0,20	0,98

Fig. 15(a)	1 par	2 pares	3 pares
1 MB	0	0,78	0,23
10 MB	0,00	0,05	0,00
100 MB	0,00	0,00	0,26

Fig. 15(b)	1 par	2 pares	3 pares
1 MB	0	0,00	0,16
10 MB	0,09	0,05	0,02
100 MB	0,09	0,00	0,05

Fig. 19(a)	ECMP	MultiFlow
1 MB	0	0,61
10 MB	0,30	0,20
100 MB	0,21	0,09

Fig. 19(b)	ECMP	MultiFlow
1 MB	0	0,61
10 MB	0,09	0,45
100 MB	0,02	0,03

Fig. 20	Hash	Random
MultiFlow		
1 MB	0	0,03
0,61		
10 MB	0,09	0,60
0,45		
100 MB	0,02	0,56
0,03		