# Implementação Inicial da RFC 6897 para Auxílio no Tratamento de Fluxos Elefantes

**Alan C. Silva**[1], **Simone Ferlin**[2], **Fábio L. Verdi**[1]

[1]LERIS – Universidade Federal de São Carlos (UFSCar)
Sorocaba – SP – Brasil

[2]Simula Research Laboratory, 1364 Fornebu, Norway

{alansilva,verdi}@ufscar.br, ferlin@simula.no

***Abstract.*** *The Multipath TCP (MPTCP) protocol allows applications to better explore the network resources available to multi-connected devices such as mobile phones or multi-homed systems. Here, two main advantages are envisioned: bandwidth aggregation, and the ability to maintain the connection, if one of the network path fails. To extend these capabilities to the application, RFC 6897 defines an API to better control each of MPTCP's subflows, so that these can be added or removed as needed. This paper presents an API implementation as defined in RFC 6897 to give applications the capability to control MPTCP's subflows. To test the API and validate our implementation, we build experiments with elephant flows in datacenter networks.*

***Resumo.*** *O protocolo Multipath TCP (MPTCP) permite que as aplicações possam explorar melhor os recursos de rede disponíveis para dispositivos multi-conectados como os telefones móveis ou sistemas multi-homed. Aqui, duas principais vantagens são previstas: agregação de banda e a habilidade de manter a conexão estabelecida se houver falha em um dos caminhos de rede. Para estender essas capacidades para a aplicação, a RFC 6897 define uma API para permitir um melhor controle de cada subfluxo MPTCP, de modo que esses possam ser adicionados ou removidos conforme necessário. Este artigo apresenta uma implementação da API conforme definida na RFC 6897 para dar as aplicações a capacidade de controlar os fluxos MPTCP. Para testar a API e validar nossa implementação, nós construimos experimentos usando fluxos elefantes em redes de datacenter.*

## 1. Introdução

O protocolo de controle de transmissão *(TCP)* é um protocolo-chave no paradigma atual da Internet, pois ele permite criar serviços onde os fluxos de bytes que passam são confiáveis e por isso acaba sendo utilizado por diversas aplicações que demandam confiabilidade na transmissao de dados nos mais variados tipos de dispositivos, desde smartphones até servidores em redes de datacenters.

Mesmo pertencendo a uma das partes fundamentais da Internet que conhecemos hoje, o protocolo *TCP* continua em constante processo de evolução. Com o surgimento de uma Internet e serviços mais complexos, uma das evoluções mais recentes que acompanhou esta mudanca é o *Multipath TCP (MPTCP)*. [Raiciu et al. 2012, Raiciu et al. 2013].

O *MPTCP* altera uma das premissas básicas da especificação original do protocolo *TCP* que determina que uma conexão *TCP* será sempre identificada através da tupla de 4 elementos que consiste nos endereços *IP* de origem e destino e das portas de origem e destino. Todos os pacotes que são enviados por uma conexão *TCP* sempre serão formados por essa tupla de 4 elementos. O *TCP* como conhecemos não consegue explorar hoje todas os recursos oferecidos pelo hardware ou pelas tecnologias atuais. Por exemplo, um *host multi-homed* ou um *smartphone* que normalmente possuem a capacidade de trabalhar com multiplas interfaces ou multiplos caminhos. Nestes casos, uma conexão *TCP* não poderá se beneficiar das múltiplas interfaces e caminhos caso não exista um mecanismo que ofereça suporte para isso.

O *MPTCP* [Raiciu et al. 2013] resolve esta questão permitindo que os pacotes pertencentes a uma determinada conexão sejam transmitidos por diferentes interfaces e endereços através dos subfluxos criados para cada conexão, que, inicialmente se aproveitam da estrutura de uma conexão *TCP* original. Estes subfluxos potencialmente serão enviados individualmente pelas interfaces e caminhos disponíveis, tarefa que convém ao agendador *(scheduler)* e ao mecamismo de controle de congestionamento *(congestion control)* decidirem.

O *MPTCP* ganhou reconhecimento comercial e foi adaptado aos sistemas operacionais recentes da *Apple* [Hesmans et al. 2015b], permitindo a sua utilização em grande escala. Desde 2013, todos os *tablets* e *smartphones* da *Apple* utilizam o *MPTCP* na aplicação de reconhecimento de voz chamada *Siri*, explorando um dos aspectos do protocolo cujo objetivo é manter a conectividade quando há falha em um dos caminhos possíveis na rede. Em julho de 2015, a maior companhia de telefonia sul-coreana, a *Korean Telecom* em parceria com a *Samsung* anunciou a implantação em escala comercial do protocolo *MPTCP* em diversos *smartphones* equipados com *Android*, combinando as interfaces *LTE* com *wifi* disponíveis, aumentando significativamente o ganho na taxa de utilização de banda desses dispositivos [Oh and Lee 2015].

O desenvolvimento do protocolo *MPTCP* foi motivado por alguns cenários que necessitam enviar dados utilizando a mesma conexão através de interfaces e endereços diferentes, como por exemplo, *smartphones* equipados com interface celular e *wifi* [Paasch et al. 2012], ou redes de *datacenters* [Raiciu et al. 2011] cuja malha topológica oferece múltiplos caminhos para os diferentes subfluxos, que seguem possivelmente para um mesmo destino.

O *MPTCP* foi desenvolvido tendo três principais objetivos de compatibilidade em mente [Ford et al. 2011]:

1. O protocolo deve ser utilizado pelas aplicações aproveitando a *API* de *socket* já existente. Atualmente, o *Linux* [Paasch et al. 2013] possui uma implementação do protocolo *MPTCP* que resolve em parte essa situação.
2. O protocolo deve ser compatível com a rede onde está sendo implantado. Para alcançar este objetivo, o protocolo possui diversos mecanismos que possibilitam a sua comunicação com *middleboxes* existentes na rede [Raiciu et al. 2012, Raiciu et al. 2013].
3. O protocolo deve manter a equidade entre os usuários da rede. Para alcançar esse objetivo, vários esquemas de controle de congestionamento foram propostos e implementados.

Este artigo tem como objetivo rever o primeiro item da lista de compatibilidades que é requerido para o *MPTCP* operar de forma compatível na Internet, ou seja, implementar a *API* proposta pela *RFC 6897*. Esta *API* oferece para as aplicações a capacidade de controlar o protocolo *MPTCP* podendo, por exemplo, ligar ou desligar o suporte ao protocolo e ser capaz de adicionar ou remover um ou mais subfluxos. Hoje, a única forma de controlar o número de subfluxos que são fixos e criados no momento de abertura da conexáo é via comando *sysctl*.

A criação da *API* permitirá que todas as aplicações que estão rodando no sistema possam ajustar de forma individual a quantidade de subfluxos necessários, servindo como motivação para o trabalho.

Assumindo que a aplicação necessita de um serviço de transmissão de bytes, precisamos de uma interface *(API)* que possa providenciar a adaptação de forma transparente para permitir o controle do protocolo *MPTCP* com a finalidade de se obter melhor utilização. Para validar nossa implementação, usamos a *API* para o tratamento de fluxos elefantes em redes de datacenters.

O artigo está organizado da seguinte forma: na Seção 2 serão apresentados os trabalhos relacionados. A descrição do protocolo *MPTCP* é apresentada na Seção 3. Uma visão mais detalhada do funcionamento da *RFC 6897* será apresentada na Seção 4. Os métodos utilizados para o desenvolvimento e implementação assim como o caso de uso serão apresentados na Seção 5.

## 2. Trabalhos Relacionados

Os gerenciadores de caminhos (*path managers*) possuem a função de decidir como os subfluxos serão estabelecidos e atualmente há dois disponíveis: *Fullmesh* e *ndiffports*. No caso do *ndiffports*, apenas o cliente é responsável pela criação de subfluxos. O servidor nunca cria subfluxos, pois o cliente pode estar atrás de um *firewall* ou *NAT* que bloqueia a tentativa de conexão [Eardley 2013]. Em *fullmesh* é possível a criação do subfluxo a partir do servidor, caso ele conheça o IP de conexão e trabalhar como passivo, esperando um *ADD* por parte do cliente, porém, o modo comum de operação é o ativo (onde o cliente cria o subfluxo).

O gerenciador de caminhos *fullmesh* escuta os eventos a partir das interfaces de rede subadjacentes e cria um subfluxo para o servidor para cada interface ativa. Estes subfluxos são criados imediatamente após a criação da conexão ou quando a interface é ativada a partir da criação do primeiro subfluxo, o subfluxo principal. Isto permite, por exemplo, que os *smartphones* possam reagir no caso de uma perda de conectividade [Paasch et al. 2012].

O gerenciador de caminhos *ndiffports* cria `N` subfluxos em uma mesma interface de forma imediata após o estabelecimento da conexão. Este gerenciador de caminhos foi desenvolvido tendo o foco em *datacenters* onde são permitidos a utilização de diferentes caminhos que possam ser balanceados com *Equal Cost Multipath (ECMP)* [Raiciu et al. 2011].

Diversos pesquisadores têm explorado como o protocolo *MPTCP* deve gerenciar os fluxos e as interfaces disponíveis. Paasch et Al. [Paasch et al. 2012] avaliam como os dispositivos *wireless* se adaptam a perdas de conectividade. Este artigo propõe três modos

de operação para o protocolo *MPTCP* em *smartphones*: *single path*, *backup* e *full-mptcp*. Bocassi et al. [Boccassi et al. 2013] propõem o gerenciador de caminhos chamado *Binder* que explora o roteamento de origem livre e utiliza o *MPTCP* para agregar caminhos diferentes em redes de malha sem fio. Lim et al. [Lim et al. 2014b] propõem uma extensão para o *MPTCP* que permite adaptar a utilização dos subfluxos baseado na informação extraída a partir da camada de *MAC*. Essa extensão foi validada experimentalmente, porém, a implementação não possui muitos detalhes.

Lim et al. [Lim et al. 2014a] também propõem o *eMPTCP* que retarda o estabelecimento dos subfluxos em *smartphones* que estão utilizando a interface *LTE*. Porém, quando o smartphone troca de interface, por exemplo, *(LTE - Wifi)*, o artigo propõe a reinicialização da estimativa do *RTT* do subfluxo utilizado na interface *LTE* para forçar a utilização do mesmo na nova interface selecionada. Essa solução agiliza a utilização do subfluxo *LTE*, mas não é elegante para solucionar o problema do gerenciamento de subfluxos.

Schmidt et al. [Schmidt et al. 2013] propõem a utilização do conceito de *socket intents* que permite as aplicações informarem a conexão sobre o seu conhecimento da comunicação estabelecida, podendo gerar informaçao que serve como previsão para melhoria da conexão. Estes *intents* incluem informação sobre o tipo de transferência *(query, bulk, stream)* ou a informação sobre o fluxo (número de bytes, duração).

Hesmans et al. [Hesmans et al. 2015a] definem a criação de um gerenciador de caminhos que utiliza o conceito de *socket intents*, extraindo as informações dos *intents* e usando as mesmas como parâmetros, permitindo que aplicações possam manipular o protocolo *MPTCP*.

Porém, mesmo que a *RFC 6897* [Scharf and Ford 2013] proponha a criação de algumas extensões na interface *(API)* básica de *socket* para permitir que aplicações possam adicionar ou remover subfluxos em uma conexão *MPTCP*, atualmente nenhuma implementação do *MPTCP* faz uso desta extensão. Esta fato serve de motivação para a implementação que é mostrada nesse artigo, dado a necessidade existente demonstrada através do caso de uso escolhido.

Por fim, é importante definirmos o que é fluxo elefante. De acordo com a definição encontrada em [Greenberg et al. 2009], um fluxo é considerado elefante quando ele atinge 100MB. Muito embora existam outras definições na literatura [Curtis et al. 2011, Casado 2013], esta é a mais utilizada. Atualmente, existem algumas soluções que tratam o problema dos fluxos elefantes. Elas se baseiam na ideia de detecção e tratamento no centro da rede usando controladores que fazem o escalonamento através de múltiplos caminhos conforme *Al Fares et. al.* [Al-Fares et al. 2010].

## 3. Multipath TCP

O *MPTCP* [Raiciu et al. 2013] permite aos hosts trocarem pacotes que pertencem a uma determinada conexão entre várias interfaces ou caminhos na rede. Para que esta tarefa possa ser executada, cada conexão *MPTCP* é composta por várias conexões *TCP* que são denominadas subfluxos [Raiciu et al. 2013].

O *MPTCP* é uma extensão do protocolo *TCP* definida pelo *IETF* em [Ford et al. 2009] dando a possibilidade de uma conexão *TCP*, que por padrão é *single*

*path*, poder funcionar como uma conexão de múltiplos caminhos, suportando assim uma maior diversidade para os dados de uma mesma conexão.

Uma das principais vantagens do *MPTCP* em relação a outras soluções, como por exemplo, o *SCTP* [Paasch and Bonaventure 2014], é poder usar a mesma estrutura do protocolo *TCP* para trafegar informação, fazendo com que o *MPTCP* seja transparente para aplicações que utilizam o protocolo. Isso é possível pois o MPTCP se aproveita da mesma *API* de *socket* do protocolo *TCP* como mostra a Figura 1.

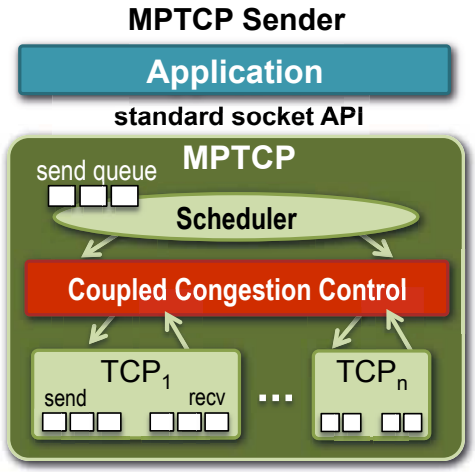


**Figura 1. Pilha TCP com suporte ao MPTCP.**

O *MPTCP* se aproveita do processo de *three-way handshake* do *TCP* para efetuar a negociação necessária que permite seu uso. Basicamente, ele possui 3 fases que são:

- *Estabelecer uma nova conexão MPTCP;*
- *Adicionar subfluxos em uma conexão MPTCP;*
- *Transmitir dados através da conexão MPTCP.*

Para estabelecer uma nova conexão *MPTCP*, o host de origem envia um pacote *SYN* com a *flag* de *MP_CAPABLE* sinalizada mais uma chave aleatória que será utilizada para o cálculo do *hash* que servirá de identificador para a conexão. Nesse caso, se o *host* de destino suportar o protocolo *MPTCP*, ele devolve a resposta com um *SYN+ACK* contendo o *MP_CAPABLE* sinalizado mais uma chave aleatória escolhida pelo servidor. O terceiro *ACK* do processo de *three-way handshake* também vai incluir a *flag* de *MP_CAPABLE* sinalizada e as chaves que serão utilizadas como identificador, estabelecendo assim a conexão, conforme a Figura 2.

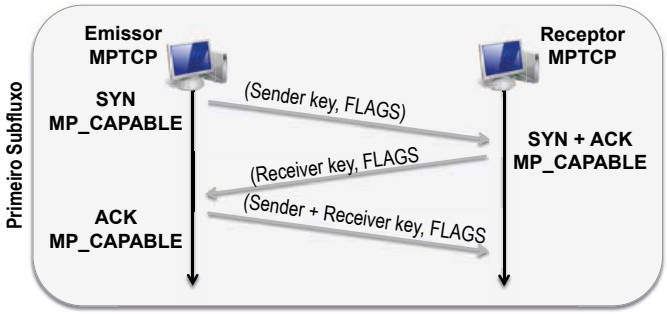


**Figura 2. Criação de uma conexão MPTCP (MP_CAPABLE).**

Caso seja necessária a criação de um novo subfluxo em uma conexão *MPTCP* existente, é efetuado um novo processo de *three-way handshake*, onde o *host* de origem envia o primeiro *SYN* com o campo *MP_JOIN* que possui o *token* da conexão *MPTCP* existente. Este *token* é o resultado da chave que foi gerada no estabelecimento da conexão.

O host de destino recebe o pacote e calcula o *HMAC* a partir do *token*, devolvendo um *SYN+ACK* com o resultado do *HMAC*. O host de origem recebe e valida o *HMAC* devolvendo um *ACK* com o *HMAC* e assim estabelecendo a validação do subfluxo, conforme a Figura 3.

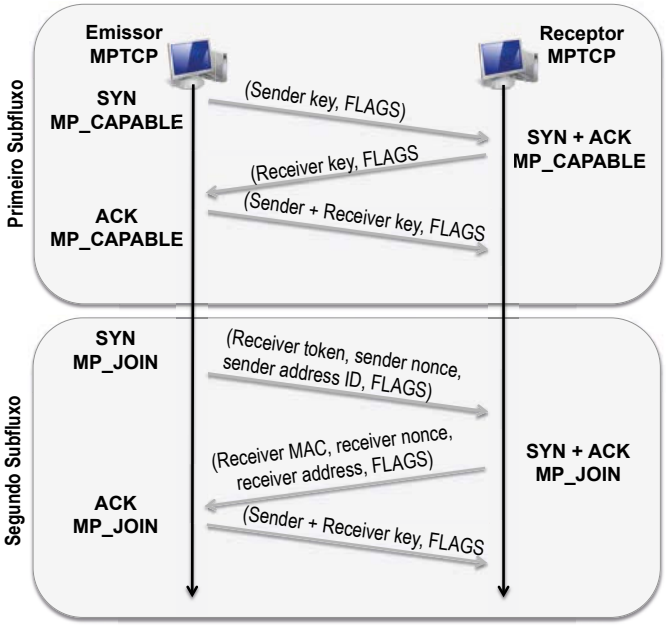


**Figura 3. Criação de um novo subfluxo em uma conexão existente (MP_JOIN).**

A proposta desse projeto é aproveitar o recurso de subfluxos do *MPTCP* justamente para quebrar os fluxos grandes em subfluxos menores, permitindo assim, o espalhamento desses subfluxos através dos vários caminhos oferecidos e explorados pelo protocolo *MPTCP* por padrão.

## 4. RFC 6897

Conforme citado anteriormente, o protocolo *MPTCP* adiciona a capacidade de se utilizar caminhos múltiplos em uma conexão *TCP*. As motivações para isso incluem o aumento da taxa de transferência e a resiliência no caso de uma possível falha de rede.

O texto encontrado na *RFC 6897* descreve algumas questões sobre a compatibilidade do protocolo *MPTCP* com aplicações que não suportam o mesmo, e sugere uma interface básica de aplicação que estende a interface *TCP* já existente para permitir esse suporte. A idéia é apresentar quais são os efeitos que o protocolo *MPTCP* exerce sobre as aplicações, como por exemplo, a questão de desempenho comparando-o ao *TCP* e a interoperabilidade entre *MPTCP* e as aplicações.

Assumindo que a aplicação não suporta transporte de dados por caminhos múltiplos sem que haja a necessidade de alteração nas mesmas, se faz necessário especificar uma *API* básica para permitir que estas aplicações sejam *MPTCP-aware*, ou seja, suportem o protocolo.

A motivação para a criação de uma *API* que suporte o protocolo *MPTCP* consiste no fato de permitir que aplicações possam utilizar uma interface já conhecida. Por isso, a implementação é efetuada através da extensão da interface de *sockets* que já é utilizada como padrão na implementação de aplicações que precisam interagir entre si utilizando o protocolo *TCP*. Isso permite a mesma facilidade de implementação, pois quem já utiliza a interface de *sockets TCP* poderá utilizar a *interface MPTCP* da mesma forma.

A *API padrão* já consolidada para aplicações *TCP* é a interface via *socket*. A partir disso, o documento propõe uma maneira abstrata de estender esta interface de forma que possa suportar o *MPTCP* através de operações que obtenham *(get)* e definam *(set)* valores específicos do *MPTCP* através de uma opção de socket *socket option* gerada no nível do protocolo *TCP*. Isso permite que aplicações, linguagens de programação e bibliotecas possam decidir como utilizar essas informações de forma transparente, ou seja, sem nenhuma alteração de código.

Uma *API MPTCP* básica consiste em um conjunto de novos valores que serão associados a um *socket MPTCP*. Estes valores podem ser utilizados para alterar propriedades em uma conexão *MPTCP* ou recuperar uma informação.

Os valores podem ser acessados através das chamadas de sistema já utilizadas na interface de *socket* como *setsockopt()* ou *getsockopt()*, através de símbolos específicos criados especialmente para alterar ou recuperar dados correspondentes ao protocolo *MPTCP* que serão acessados por parâmetros passados para essas chamadas.

As opções que são especificadas na Seção 5.3.1 da *RFC 6897* correspondem a uma versão básica da *API* e devem ser implementadas para uma utlização aceitável por outras aplicações:

- *TCP_MULTIPATH_ENABLE*: Habilita ou desabilita o *MPTCP*;
- *TCP_MULTIPATH_ADD*: Conecta o protocolo *MPTCP* em um conjunto de endereços locais definidos ou adiciona um conjunto de novos endereços locais em uma conexão *MPTCP* existente. Na prática esta opção permite que novos subfluxos sejam adicionados à uma conexão *MPTCP*;
- *TCP_MULTIPATH_REMOVE*: Remove um endereço local de uma conexão

*MPTCP*. Na prática, esta opção permite que subfluxos existentes sejam removidos de uma conexão *MPTCP*;
- *TCP_MULTIPATH_SUBFLOWS*: Recupera os pares de endereços atualmente utilizados por subfluxos *MPTCP*;
- *TCP_MULTIPATH_CONNID*: Retorna o identificador de conexão local da conexão *MPTCP* atual.

A Tabela 1 mostra o conjunto de funções que serão implementadas na *API* de *socket* para permitir a manipulação do *MPTCP* através das aplicações.

**Tabela 1. Operações da API MPTCP**

| Nome | Get | Set | Tipo de Dados |
|---|---|---|---|
| TCP_MULTIPATH_ENABLE | x | x | Booleano |
| TCP_MULTIPATH_ADD | | x | Lista de Endereços/Portas |
| TCP_MULTIPATH_REMOVE | | x | Lista de Endereços/Portas |
| TCP_MULTIPATH_SUBFLOWS | x | | Lista de Pares de Endereços/Portas |
| TCP_MULTIPATH_CONNID | x | | Inteiro |

## 5. Implementação e Avaliação

A *RFC 6897* define como estender a *API* de *sockets* para agregar funções que consigam interoperabilizar com o protocolo *MPTCP*. Lembrando que a *API* de *socket* é implementada diretamente em nível de *kernel* e, por isso, é necessário que as funções da *API* também sejam implementadas no mesmo nível.

Para isso, adicionamos as chamadas de interface para o protocolo *MPTCP* dentro da interface de *sockets* que são representadas pelas funções *do_tcp_setsockopt*, no caso das funções de alteração de dados, e *do_tcp_getsockopt*, no caso das funções de recuperação de dados. Tais funções são definidas e implementadas no arquivo *tcp.c* localizado em *net/ipv4* dentro da árvore de código do *kernel*.

Abaixo temos um trecho de código, esqueleto em C, da implementação da função que adiciona subfluxos, ilustrando o início do desenvolvimento em modo de *kernel*.

```
static int do_tcp_setsockopt(struct sock *sk, int level,
       int optname, char __user *optval, unsigned int optlen)
{
...
switch (optname) {
     case TCP_MULTIPATH_ADD: {
         //Implementação da função que adiciona subfluxos.
        }
    }
    ...
}
```

A característica padrão de operação do protocolo *MPTCP* é a de estabelecer um número fixo de subfluxos que são criados no momento de abertura da conexão. Atualmente, a única forma de alterar essa operação é por meio de variável de controle do sistema referente ao protocolo e que pode ser definida através do comando *sysctl*. Entretanto, esta alteração passa a valer para

todas as aplicações que estão rodando no sistema, não permitindo, portanto, que cada aplicação, individualmente, ajuste a quantidade de subfluxos necessários.

Outro ponto importante na operação do *MPTCP*, e também relevante para a API que está sendo implementada, é a questão dos *path managers*; já que são eles que definem como o protocolo deve se comportar em relação à criação de subfluxos. A implementação da API feita neste trabalho utiliza o *path manager ndiffports* uma vez que seu modo de operação permite abrir mais de um subfluxo por par de endereços.

As duas operações que devem ser inicialmente implementadas para permitir que uma aplicação execute as funções básicas descritas no protocolo são as de adicionar (*TCP_MULTIPATH_ADD*) e remover subfluxos *TCP_MULTIPATH_REMOVE*.

Estas operações permitem adicionar e remover um subfluxo por vez. Entretanto, neste trabalho, fizemos uma pequena extensão para que uma aplicação possa criar `N` subfluxos em uma única requisição. Com isso, adicionamos à opção *TCP_MULTIPATH_ADD* mais um campo na criação de subfluxos, podendo definir o número de subfluxos a serem criados. Esse campo foi denominado *num_subflows*, o que permite criar a chamada na interface conforme exemplo:

```
setsockopt(fd, SOL_TCP, TCP_MULTIPATH_ADD, num_subflows);
```

Para um teste inicial destas operações, utilizamos 2 máquinas virtuais *VMs* com *VMWare*, ambas rodando em um *Ubuntu Linux Server 14.04* com kernel versão 3.14.33 e, finalmente, com suporte ao protocolo *MPTCP* em sua versão de implementação 0.89. Uma máquina foi utilizada como servidor e a outra como cliente e através de duas aplicações escritas em *Python* simulamos o cliente e o servidor 4. Para fins de testes, criamos 4 subfluxos em uma mesma conexão *MPTCP*.

Atualmente, temos uma interface simples desenvolvida na linguagem *Python* para efetuar a chamada da *API* em *user mode* e um conjunto de funções desenvolvidas em linguagem C estendendo a *API de Sockets* em *Kernel Mode* no *Linux*. As funções disponíveis são: *TCP_MULTIPATH_ENABLE*, *TCP_MULTIPATH_CONNID* e *TCP_MULTIPATH_ADD*.

## 5.1. Caso de uso da RFC 6897: tratamento de fluxos elefantes

Como caso de uso, desenvolvemos um módulo que, junto ao *end-host*, tem a capacidade de verificar se um determinado fluxo de saída contém características de um fluxo elefante. Quando um fluxo elefante é detectado, a aplicação solicita a criação de subfluxos conforme a necessidade. A Figura 4 ilustra uma visão completa e genérica da solução. A seguir, temos uma descrição da arquitetura do módulo e a definição do que é um fluxo elefante para o mesmo.


**Figura 4. Visão da Arquitetura do Módulo.**

A arquitetura do módulo basicamente consiste nas seguintes fases:

- *Fase de detecção do fluxo elefante*: Nessa fase, todo o tráfego é monitorado através de um serviço que será executado no *end-host*, permitindo assim a detecção de um possível fluxo elefante;

- *Tratamento do fluxo elefante*: Após o fluxo elefante ser detectado, o módulo notifica o *kernel MPTCP* transformando esse fluxo elefante em vários fluxos camundongos;
- *Encaminhamento*: Conforme o número de subfluxos que foram estabelecidos, os fluxos camundongos que foram gerados são encaminhados. Eles chegam até o seu destino através do espalhamento que é feito utilizando os subfluxos criados.

A arquitetura se baseia em um algoritmo que define todo processo de atuação no *end-host*, passando pela detecção, tratamento e o encaminhamento final. A validação do algoritmo é determinada através de valores considerados mínimo e máximo para atuação do mesmo. Esses valores são utilizados como parâmetros limites de atuação para a detecção de um fluxo elefante, que hoje é definido a partir de 100MB até o limite de 300MB, abrindo um novo subfluxo a cada 50MB até atingir o limite máximo de 4 subfluxos, baseando se na definição de fluxo elefante encontrada em [Curtis et al. 2011].

O algoritmo consiste em um método contínuo que fica monitorando todos os fluxos *TCP* até que o valores estabelecidos como limite sejam atingidos. Quando o valor definido para o limite é alcançado, o módulo de controle de conexões *MPTCP* é ativado, recebendo a conexão sinalizada e a transformando em diversos fluxos camundongos. Esses fluxos camundongos são distribuídos através dos subfluxos criados e múltiplos caminhos existentes na rede conforme observado na Figura 5.

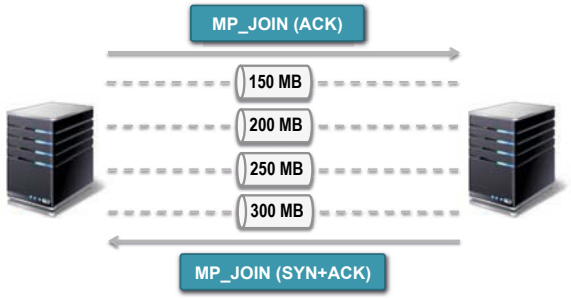


**Figura 5. Visão de Criação e Controle de Subfluxos pelo Módulo.**

O módulo trabalha em modo de serviço *(daemon UNIX)* que consiste na comunicação direta colocando a interface de rede em modo promíscuo, permitindo com isso que o repasse de fluxos e gerenciamento de chamadas ao protocolo *MPTCP* seja estabelecido.

As fases de tratamento e encaminhamento dependem de um *kernel* do *linux* que tenha suporte ao *MPTCP*. Importante lembrar que para o funcionamento do módulo se faz necessário o controle de conexões e criação de subfluxos através de rotinas que serão controladas pela aplicação, justificando a necessidade de implementação da API.

## 6. Conclusão

O *Multipath TCP (MPTCP)* foi desenvolvido para que fluxos *TCP* tradicionais usem com maior eficiência as diferentes interfaces de rede dos dispositivos, assim como os diferentes caminhos disponíveis em uma topologia de rede.

Entretanto, todo potencial fornecido pelo *MPTCP* só pode ser explorado se as aplicações fizerem uso do protocolo de maneira consciente (*MPTCP-awareness*), ideia esta especificada na *RFC 6897* que propõe uma *API* para que um controle fino dos subfluxos *MPTCP* seja realizado pelas aplicações que desejem usar tal protocolo.

Este trabalho, portanto, apresenta uma implementação desta *RFC* em nível de *kernel* para que aplicações façam uso do protocolo *MPTCP* conforme suas necessidades, definindo quando adicionar e remover fluxos assim como quando ligar e desligar o protocolo. Com esta implementação, os desenvolvedores e usuários do protocolo *MPTCP* podem encontrar caminhos para estender as suas aplicações de forma a utilizar os recursos existentes no protocolo com a finalidade de se obter os melhores resultados em sua utilização.

Atualmente, com os testes que foram realizados, conseguimos criar até 4 subfluxos simultâneos por conexão *MPTCP*. Dando continuidade ao trabalho, os próximos passos são: desenvolvimento das funções de *TCP_MULTIPATH_REMOVE* e *TCP_MULTIPATH_SUBFLOWS*, realização de avaliações experimentais da solução e discussão com o WG do *MPTCP*. Por fim, pretendemos disponibilizar a implementação para a comunidade.

## Referências

Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. (2010). Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, volume 10, pages 19–33.

Boccassi, L., Fayed, M. M., and Marina, M. K. (2013). Binder: a system to aggregate multiple internet gateways in community networks. In *Proceedings of the 2013 ACM MobiCom workshop on Lowest cost denominator networking for universal access*, pages 3–8. ACM.

Casado, M. (2013). Of mice and elephants.

Curtis, A. R., Kim, W., and Yalagandula, P. (2011). Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *INFOCOM, 2011 Proceedings IEEE*, pages 1629–1637. IEEE.

Eardley, P. (2013). Survey of mptcp implementations.

Ford, A., Raiciu, C., Handley, M., Barre, S., Iyengar, J., et al. (2011). Architectural guidelines for multipath tcp development. *IETF, Informational RFC*, 6182:2070–1721.

Ford, A., Raiciu, C., Handley, M., Bonaventure, O., et al. (2009). Tcp extensions for multipath operation with multiple addresses. *IETF MPTCP proposal-http://tools. ietf. org/id/draft-ford-mptcp-multiaddressed-03. txt*.

Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. (2009). Vl2: a scalable and flexible data center network. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 51–62. ACM.

Hesmans, B., Detal, G., Bauduin, R., Bonaventure, O., et al. (2015a). Smapp: Towards smart multipath tcp-enabled applications. In *CoNEXT'15*.

Hesmans, B., Tran-Viet, H., Sadre, R., and Bonaventure, O. (2015b). A first look at real multipath tcp traffic. In *Traffic Monitoring and Analysis*, pages 233–246. Springer.

Lim, Y.-s., Chen, Y.-C., Nahum, E. M., Towsley, D., and Gibbens, R. J. (2014a). How green is multipath tcp for mobile devices? In *Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges*, pages 3–8. ACM.

Lim, Y.-s., Chen, Y.-C., Nahum, E. M., Towsley, D., and Lee, K.-W. (2014b). Cross-layer path management in multi-path transport protocol for mobile devices. In *INFOCOM, 2014 Proceedings IEEE*, pages 1815–1823. IEEE.

Oh, B.-H. and Lee, J. (2015). Constraint-based proactive scheduling for mptcp in wireless networks. *Computer Networks*, 91:548–563.

Paasch, C., Barre, S., et al. (2013). Multipath tcp in the linux kernel.

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

Paasch, C., Detal, G., Duchene, F., Raiciu, C., and Bonaventure, O. (2012). Exploring mobile/wifi handover with multipath tcp. In *Proceedings of the 2012 ACM SIGCOMM workshop on Cellular networks: operations, challenges, and future design*, pages 31–36. ACM.

Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., and Handley, M. (2011). Improving datacenter performance and robustness with multipath tcp. *ACM SIGCOMM Computer Communication Review*, 41(4):266–277.

Raiciu, C., Handley, M., and Bonaventure, O. (2013). Tcp extensions for multipath operation with multiple addresses. Technical report, RFC 6824, Jan.

Raiciu, C., Paasch, C., Barre, S., Ford, A., Honda, M., Duchene, F., Bonaventure, O., and Handley, M. (2012). How hard can it be? designing and implementing a deployable multipath tcp. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 29–29. USENIX Association.

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

Schmidt, P. S., Enghardt, T., Khalili, R., and Feldmann, A. (2013). Socket intents: Leveraging application awareness for multi-access connectivity. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 295–300. ACM.