UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

RENAN DE MATOS CASACA

ANÁLISE DO PROTOCOLO TCP FAST OPEN

CHAPECÓ
2018

RENAN DE MATOS CASACA

ANÁLISE DO PROTOCOLO TCP FAST OPEN

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.
Orientador: Prof. Dr. Marco Aurélio Spohn

CHAPECÓ
2018

Bibliotecas da Universidade Federal da Fronteira Sul - UFFS

Casaca, Renan de Matos
Análise do Protocolo TCP Fast Open / Renan de Matos Casaca. -- 2018.
59 f.:il.

Orientador: Dr. Marco Aurélio Spohn.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal da Fronteira Sul, Curso de Ciência da Computação, Chapecó, SC , 2018.

1. Handshake. 2. Protocolo de Rede. 3. TCP. 4. TCP Fast Open. 5. Internet. I. Spohn, Marco Aurélio, orient. II. Universidade Federal da Fronteira Sul. III. Título.

Elaborada pelo sistema de Geração Automática de Ficha de Identificação da Obra pela UFFS com os dados fornecidos pelo(a) autor(a).

RENAN DE MATOS CASACA

ANÁLISE DO PROTOCOLO TCP FAST OPEN

Trabalho de conclusão de curso de graduação apresentado como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

Orientador: Prof. Marco Aurélio Spohn

Aprovado em: 07\12\2018

BANCA EXAMINADORA:

Marco A. Spohn

Marco Aurélio Spohn - UFFS

Dr. Emílio Wuerges - UFFS

Dr. Claunir Pavan - UFFS

RESUMO

A importância e o uso da internet como ferramenta de trabalho, estudo, pesquisa, informação, comunicação e entretenimento é cada vez mais crescente e frequente no mundo atual. Dessa forma faz-se necessário a otimização de sua estrutura para permitir um uso eficiente de recursos tecnológicos que são responsáveis pela gerência e funcionamento adequado da internet. O protocolo TCP é um desses mecanismos, sendo responsável pelo controle e pela comunicação confiável e ordenada da maioria dos dados trafegados na rede. No entanto, o estabelecimento de conexões através do protocolo TCP exige o envio de várias mensagens de controle que acabam consumindo uma quantidade considerável de recursos, tornando-o ineficiente e causando atraso de transmissão entre cliente e servidor em conexões de curta duração. Sabendo que muitas aplicações da internet fazem uso intenso de conexões TCP, desenvolveram-se novos mecanismos ao longo do tempo para que o protocolo atue de forma eficiente a fim de proporcionar uma experiência satisfatória aos usuários. Esse trabalho propõe o estudo do TCP Fast Open, um novo mecanismo que se empenha em um uso mais inteligente e eficiente dos recursos da rede em conexões de curta duração, ao mesmo tempo, que mantém as propriedades do protocolo TCP. Foram utilizadas ferramentas de rede, como o Wireshark, para analisar o funcionamento do protocolo de acordo com suas especificações. Adicionalmente testamos seu desempenho em cenários específicos, identificando suas vantagens e quantificando seus ganhos em relação ao protocolo convencional. O TCP Fast Open apresentou resultados de desempenho significativamente melhores para a maioria dos cenários e, através de seu uso, foi possível reduzir o tempo médio de carregamento das páginas, resultando em uma experiência mais agravável para o usuário. Além disso, separamos em categorias as páginas mais populares no Brasil e no mundo, distinguindo quais aplicações possuem ganho mais significativo.

Palavras-chave: Análise. Protocolo de rede. TCP. TCP Fast Open. Internet. Handshake.

ABSTRACT

The importance and usage of the internet as a tool for work, study, research, information, communication and entertainment is increasingly frequent at today's world. Thus, it is necessary the optimization of its structure to allow an efficient use of technological resources that are responsible for the proper operation and management of the internet. The TCP protocol is one of such mechanisms, it is responsible to control a reliable and ordered communication of most data through the internet. However, the establishment of connections over TCP demands the dispatch of several control messages that end up consuming a considerable amount of resources, making it inefficient and causing transmission delays between client and server on short-term connections. Considering that many internet applications make intense use of TCP connections, there were developed new mechanisms through the time so that the protocol act efficiently and provide a satisfactory user experience. This work proposes the study of the TCP Fast Open, a new mechanism which strives to make a smarter and more efficient use of network resources on short-term connections and, at the same time, it keeps the same properties of the TCP protocol. We applied network tools, like Wireshark, to analyze the protocol operation according to its specifications. Additionally, we tested its performance in specific scenarios, identifying and quantifying its advantages over the conventional protocol. The TCP Fast open presented performance results significantly betters on most scenarios and, throughout its application, it was possible to reduce the average page loading time, resulting in a more pleasant user experience. Beyond that, we divided the most popular pages of Brazil and World in categories, distinguishing which application are benefited more with the use of the TCP Fast Open protocol as an alternative.

Keywords: Analisys. Network protocol. TCP. TCP Fast Open. Internet. Handshake.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

TCP	*Transmission Control Protocol*
TFO	*TCP Fast Open*
IP	*Internet Protocol*
RFC	*Request for Comments*
MAC	*Message code authentication*
OSI	*Open Systems Interconnection*
RTT	*Round-Trip Time*
ACK	*Acknowledge*
SYN	*Synchronize*
MSS	*Maximum segment size*
DoS	*Denial of Service*
NAT	*Network Address Translation*
IETF	*Internet Engineering Task Force*
IESG	*Internet Engineering Steering Group*
ISO	*International Organization for Standardization*
HTTP	*HyperText Transfer Protocol*
HTML	*Hypertext Markup Language*
URL	*Uniform Resource Locator*
WWW	*World Wide Web*
BCP	*Best Current Practice*
UDP	*User Datagram Protocol*
FTP	*File Transfer Protocol*
FIN	*Finalize*
PSH	*Push*
RST	*Reset*
MTU	*Maximum Transfer Unit*
VM	*Virtual Machine*
SSH	*Secure Shell*
CSS	*Cascading Style Sheets*
JS	*Java Script*
CLI	*Command Line Interface*
PLT	*Page Load Time*

SUMÁRIO

1 INTRODUÇÃO

A popularização e disseminação da internet como um meio eficiente e econômico de comunicação tem contribuído muito para o crescimento em grande escala no número de páginas web e no surgimento e difusão das mais diversas aplicações.

Com o crescimento no número de páginas e aplicações web assim como a ampliação nas possibilidade de uso da rede, cresceu consideravelmente a quantidade de usuários que usufruem desses serviços. Isso exigiu a ampliação das infraestruturas de rede e tornou mais relevante o desenvolvimento de novas técnicas para que o seu uso seja o mais eficiente possível (RADHAKRISHNAN et al., 2011).

No contexto de transferência de páginas web, o atraso (latência) e a velocidade na transferência dos dados na comunicação entre cliente (usuário) e servidor (provedor de serviço) é um fator importante para determinar a satisfação dos usuários ao fazer uso dos mais diversos recursos que são providos pela internet (RADHAKRISHNAN et al., 2011).

Aliado a necessidade de atender a satisfação dos usuários e manter a internet uma rede funcionalmente eficiente, novas abordagens surgem de tempos em tempos no que diz respeito a utilização de ferramentas capazes de gerenciar a comunicação, como os protocolos.

A internet que utilizamos faz uso de uma pilha de protocolos conhecida como pilha TCP/IP e vem se adaptando a crescente demanda por uma rede rápida e fluente (KUROSE; ROSS, 2006). Nessa pilha muitos protocolos são implementados, sendo o TCP um protocolo largamente utilizado para comunicação do par cliente-servidor em aplicações onde a confiabilidade e integridade dos dados transmitidos seja essencial. (TANENBAUM, 2003)

Protocolos de rede desempenham um papel fundamental na internet, pois, ditam as regras de comunicação entre os usuários desse serviço. A popularização da internet tem contribuído para o crescimento em grande escala no número de páginas web e, esse crescimento não é escalado pelos protocolos de rede utilizados (RADHAKRISHNAN et al., 2011).

O TCP é o protocolo mais utilizado para atender tais requisições de comunicação na internet (KUROSE; ROSS, 2006) e, por isso é fundamental que ele opere de forma eficiente.

Ao longo dos anos, protocolos como o TCP vem se adaptando as demandas e necessidades encontradas na rede afim de manter o bom funcionamento da internet num aspecto geral. Portanto, o estudo de novas técnicas que implementem melhorias significativas em ganho de performance são sempre bem-vindas.

Uma abordagem largamente utilizada pelo TCP, para melhorar seu desempenho, é a de utilizar conexões persistentes, fazendo com que um cliente tenha melhor aproveitamento de uma conexão já aberta com o servidor (RADHAKRISHNAN et al., 2011). Nem sempre, porém, essa técnica leva a resultados satisfatórios devido à características específicas no comportamento da rede, como a existência de poucos objetos (blocos de dados) a serem transmitidos pelas páginas (TOUCH; HEIDEMANN; OBRACZKA, 1998).

As páginas web podem conter uma quantidade variada de objetos web (como imagens e blocos de texto) que possuem diferentes tamanhos. Hoje, o tamanho médio transferido pela rede por página é de 300KB, mas a maioria dos objetos existentes nessas páginas são relativamente pequenos, com um tamanho médio de 2,4KB (RAMACHANDRAN, 2010). Considerando uma preponderância na existência de objetos web pequenos em grande parte das páginas, faz-se necessário o planejamento de novas soluções para integrar o protocolo TCP de modo a aumentar a eficiência da rede.

O TCP Fast Open, também chamado TFO, é um protocolo variante do TCP, resultado desses esforços que buscam tornar o uso da rede mais eficiente nessas situações. Ele foi proposto inicialmente em 2011 (RADHAKRISHNAN et al., 2011), mais tarde definido pela RFC 7413 em 2014 (CHENG et al., 2014) e encontra-se com o status de experimental pois ainda não foi aprovado pela IETF (*Internet Engineering Task Force*) representado pela IESG (*Internet Engineering Steering Group*) que é o órgão responsável por aprovar RFCs (*Request for Comments* - detalhamentos dos protocolos) como padrões da internet.

A ideia por trás do TFO é a de permitir a transferência de dados entre cliente e servidor enquanto o TCP ainda estiver na fase de *handshake* (estabelecimento de conexão), o que garantiria um ganho de desempenho (redução de atraso) na transmissão de objetos, especialmente os de tamanho pequeno e nas conexões de curta duração. No entando, poucos experimentos foram realizados para verificar sua real efetividade em situações reais, o que restringiu a sua adotação e uso em muitas aplicações.

Uma vez que uma RFC experimental pode se tornar um padrão da internet posteriormente, se torna importante que novos experimentos e testes sob tais ferramentas sejam feitas, com o intuito de verificar o bom funcionamento e as vantagens do que está sendo proposto.

O tema deste trabalho é o estudo e análise desse mecanismo, o TCP Fast Open. Esse protocolo propõe sistemáticas, porém simples, mudanças na implementação do TCP convencional com o objetivo de aumentar o desempenho em conexões de curta duração. Queremos verificar

e testar essas prerrogativas em um ambiente controlado, assemelhando-se a outras pesquisas já realizadas até então.

A escolha desse protocolo como objeto de estudo se deve a fato de que, em RADHAKRISHNAN et al. (2011) publicado em 2011, apresentam-se dados relevantes sobre o comportamento de conexões que usam o TCP e apresentam-se resultados inicias promissores quanto ao seu desempenho.

Nosso principal objetivo é analisar o funcionamento do protocolo TCP Fast Open no que diz respeito a suas especificações, através do estudo da RFC 7413 (CHENG et al., 2014), e testar ferramentas disponíveis em outras pesquisas para reproduzir resultados, analisando criticamente o comportamento do protocolo. Para tal, adaptamos a ferramenta de GARRITY; STATHATOS (2014) para cenários distintos, selecionando páginas web e variando valores como a largura da banda utilizada.

Com isso pretendemos identificar as situações em que o uso do TCP Fast Open traz vantagens sobre a implementação convencional, além de quantificar esses benefícios para que se possa ter uma ideia mais clara do seu impacto na eficiência da rede como um todo.

É de nosso conhecimento que, a satisfação de usuários ao utilizar um website é um fator importante e que, latência (atraso) e tempo de carregamento de uma página influenciam nessa experiência. Mesmo pequenas melhorias na latência ao consultar uma página web podem levar a um notável aumento no número de visitas em uma página (RADHAKRISHNAN et al., 2011).

Na primeira parte deste trabalho (Seção 2) damos uma breve introdução sobre a internet, explicamos sua estrutura, principais características e aplicações. Na Seção 3 abordamos sobre o protocolo mais utilizado na internet, o TCP. Isso será uma base para a seção seguinte (Seção 4) onde abordaremos mais detalhadamente sobre o TCP Fast Open.

A Seção 5 apresenta a metodologia de trabalho utilizada, com os detalhes sobre as ferramentas utilizadas para a análise de funcionamento e análise de desempenho. Na Seção 6 discutimos e apresentamos os resultados obtidos com os experimentos de acordo com as metodologias aplicadas. Por fim, na Seção 7 são discorridas as conclusões de nosso trabalho e comentado sobre trabalhos futuros.

2 A INTERNET E SUA ESTRUTURA

A internet é uma importante ferramenta nos dias atuais. De troca de mensagens instantâneas até automóveis conectados a rede, a internet oferece um amplo espectro de oportunidades para que seu potencial seja explorado. Segundo KUROSE; ROSS (2006) não podemos defini-la em apenas uma frase pois, ela é muito complexa e está sempre mudando, tanto no que se refere a seus componentes de *hardware* e *software* quanto aos serviços que oferece.

A rede com a qual normalmente nos referimos a internet, é uma rede pública mundial de computadores (KUROSE; ROSS, 2006), que interconecta milhões de equipamentos de computação em todo o mundo e engloba empresas públicas, privadas, acadêmicas e governamentais. Para que esse sistema global opere, ele utiliza um conjunto próprio de regras, executadas por sistemas finais e intermediários.

Em redes de computadores, definimos os sistemas finais como os componentes que fazem parte da periferia da rede, não apenas computadores de mesa, mas também uma variedade grande de equipamentos como telefones celulares, sistemas de automação industrial ou residencial, TVs e outros aparelhos eletrônicos (KUROSE; ROSS, 2006).

O que não está na periferia, está no núcleo da rede, intermediando os sistemas finais. Os sistemas intermediários são compostos essencialmente por enlaces de comunicação, como roteadores e *switches*, responsáveis por transmitir os dados entre a periferia e o núcleo da rede.

2.1 Protocolos

Nesta seção, vamos falar sobre o conceito de protocolo, como eles funcionam e sua importância no âmbito da ciência da computação, enfatizando sua aplicação para o ambiente de rede de computadores.

2.1.1 Protocolos de rede

Para que o sistema global que conhecemos como internet opere, é um necessário um conjunto de regras que conhecemos como protocolos. Um protocolo de rede é uma convenção ou conjunto de regras sobre como se dará a comunicação entre dois sistemas computacionais. Todas as atividades na internet que envolvem duas ou mais entidades remotas comunicantes são governadas por um protocolo (KUROSE; ROSS, 2006).

Na definição de TANENBAUM (2003) um protocolo é um acordo entre as partes que se comunicam, estabelecendo como se dará a comunicação.

Um protocolo define o formato e a ordem das mensagens trocadas entre duas ou mais entidades comunicantes, bem como as ações realizadas na transmissão e/ou no recebimento de uma mensagem, ou outro evento (KUROSE; ROSS, 2006). É importante entender que a internet, em geral faz uso intenso de protocolos, e por essa razão, é essencial estudar e garantir que os protocolos funcionem de forma correta e eficiente.

Existem diversos tipos de protocolos, sendo eles abertos ou proprietários, variando de acordo com o tipo de serviço a ser utilizado (KUROSE; ROSS, 2006). Os abertos são os protocolos padrões da internet (BRADNER, 1996), pelo fato de poderem se comunicar com outros protocolos que utilizem o mesmo padrão de protocolo. Isso significa que eles podem ser utilizados em diferentes plataformas, como Windows, Linux, MAC OS e outros.

Protocolos proprietários são limitados a um tipo de aplicação ou empresa, são feitos para ambientes específicos, pois se comunicam apenas com uma plataforma padrão.

2.1.2 Função dos protocolos de rede

Pelo fato de protocolos sofrerem essa divisão, é difícil generalizar todas as tarefas que um protocolo pode implementar, mas normalmente um protocolo de comunicação especifica uma ou mais das seguintes propriedades:

- Detecção da conexão física subjacente ou a existência de um nó;
- *Handshaking* (estabelecimento de conexão);
- Negociação de várias características de uma conexão;
- Como iniciar e finalizar uma mensagem;
- Como formatar uma mensagem;
- O que fazer com mensagens corrompidas ou mal formatadas;
- Como detectar perda inesperada de conexão e o que fazer em seguida;
- Estabelecimento de término de sessão ou conexão;

É através do protocolo, portanto, que o estabelecimento, tráfego e encerramento da conexão são manejados. Assim como componentes da troca de informações e variáveis associadas a comunicação (KUROSE; ROSS, 2006).

Uma das funções dos protocolos é dividir um bloco de informação que deve ser transmitida em segmentos menores (KUROSE; ROSS, 2006), que são chamados de pacotes. Dentro de cada pacote existem informações de endereçamento que informam sua origem e seu destino.

Por esta razão, os protocolos de rede podem desempenhar funções como endereçamento, numeração e sequenciação, estabelecimento de conexão, confirmação de recepção, controle de erro, retransmissão, conversão de código e controle de fluxo. Com isso, ele permite que seja especificado como as partes comunicantes devem se comportar em relação aos dados sem depender de conhecimentos minuciosos sobre o hardware específico da rede (KUROSE; ROSS, 2006).

Para prover uma estrutura para o projeto de protocolos de rede, projetistas de rede organizam protocolos, e o *hardware* e o *software* de rede que implementam protocolos em camadas (KUROSE; ROSS, 2006). Portanto, cada protocolo é pertencente a uma camada.

Uma camada provê seu serviço executando os eventos dentro da camada e utilizando os serviços da camada abaixo. As camadas de protocolos podem ser implementadas tanto em *software* como em *hardware*, assim como em uma combinação dos dois (TANENBAUM, 2003). Esses conceitos são importantes para que se entenda como os protocolos funcionam.

Como dito anteriormente, os protocolos podem ser simples e implementar poucas propriedades na comunição. Na prática, apenas os protocolos mais simples são utilizados sozinhos. Sistemas complexos de comunicação, geralmente, não utilizam apenas um protocolo, mas sim uma pilha de protocolos cooperativos (KUROSE; ROSS, 2006).

2.1.3 Pilha de protocolos

Uma pilha ou família de protocolos é uma combinação de protocolos que trabalham juntos, onde as diferentes tarefas que fazem uma comunicação são executadas por níveis especializados da pilha (KUROSE; ROSS, 2006).

No início, a internet não contava com um modelo que padronizasse suas aplicações. Por consequência, foi criado o modelo de redes conhecido como OSI (*Open Systems Interconnection*), referência da ISO (*International Organization for Standardization*). Esse modelo separa e descreve a comunicação em 7 camadas distintas, são elas: Aplicação, Apresentação, Ses-

são, Transporte, Rede, Enlace de Dados e Física. Porém, por ser um modelo complexo, não é implementado na internet.

Desta maneira, surge a pilha de protocolos ao qual nos interessa estudar, o TCP/IP, que é a pilha sobre a qual a internet e a maioria das redes comerciais funcionam (KUROSE; ROSS, 2006). Esse modelo é conhecido assim pois o TCP (*Transmission Control Protocol*); e o IP (*Internet Protocol*) foram os primeiros a serem definidos.

2.1.4 O modelo de referência TCP/IP

O modelo de camadas que conhecemos como TCP/IP também chamado de Modelo de Camadas Inter-redes, diferentemente do modelo OSI, contém quatro camadas: Física, Enlace, Rede, Transporte e Aplicação. Cada camada do modelo de referência TCP/IP correspondem a uma ou mais camadas no modelo de referência OSI.

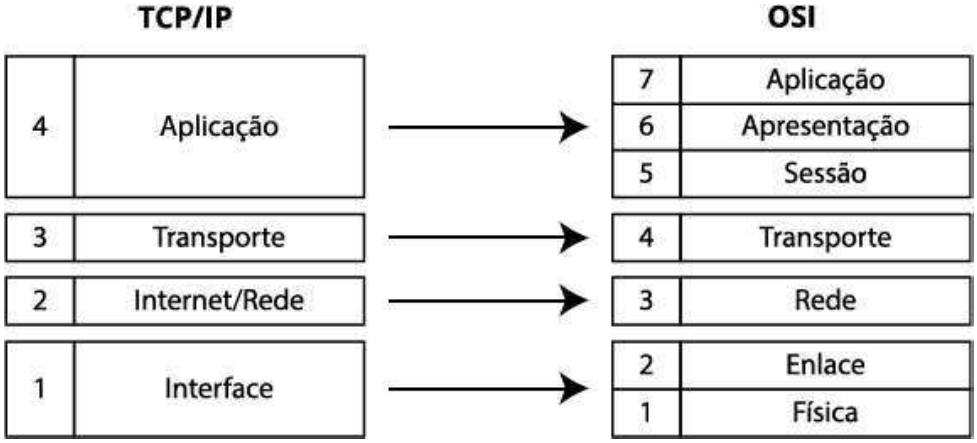

Figura 2.1 – Modelo de referência TCP e Modelo OSI

A camada física e de enlace corresponde ao *hardware* de rede básico, se limita apenas a transmissão física dos bits entre os comutadores e sistemas finais. A camada de rede especifica o formato dos pacotes enviados através da rede, assim como também os mecanismos usados para encaminhar os pacotes a partir de um computador por meio de um ou mais roteadores até um destino final.

A camada de transporte, assim como no modelo OSI, especifica como ocorre o transporte confiável dos dados. A última camada do modelo, que corresponde as camadas 6 e 7 do modelo OSI, é responsável por ditar como um aplicativo faz uso das camadas inferiores.

O TCP e o IP são os protocolos que formam o modelo de referência que conhecemos por TCP/IP. Como este primeiro é de suma importância para o entendimento deste trabalho,

será dedicado um capítulo especial para minuciar seu funcionamento. Portanto, será descrito a seguir sobre o segundo protocolo que forma este modelo.

2.1.5 IP - Internet Protocol

A internet é um conjunto de muitas redes interconectadas, e o elemento que mantém a internet unida é o protocolo da camada rede IP (TANENBAUM, 2003). O IP - *Internet Protocol* (Protocolo da Internet) foi projetado desde o início da internet e é definido na RFC 791 (CHENG et al., 2014). Ele tem o objetivo de interligar as redes, fornecendo mecanismos para transportar os datagramas (pacotes) de origem até o destino, mesmo que de forma insegura.

Na pilha de protocolos TCP/IP, o endereçamento é especificado pelo IP. Desta forma, o padrão IP especifica que a cada host é atribuído um número de 32 bits único (COMER, 2007). Esse endereço de 32 bits é comumente chamado de endereço IP.

Para nós, interessa saber que o IP é responsável por transportar datagramas (pacotes), da melhor forma possível, de uma origem para um destino, mesmo que de forma insegura, e que origem e destino podem estar na mesma rede ou haver várias redes entre elas.

Em resumo, os protocolos do modelo TCP/IP são largamente utilizados, apesar do modelo não ser praticamente aplicado. No caso do modelo OSI acontece justamente o contrário, mesmo com seus problemas, ele se mostrou excepecionalmente útil para discussão das redes de computadores. (TANENBAUM, 2003)

Como exposto anteriormente, para este trabalho estamos considerando o modelo de referência do TCP/IP justamente por trabalhar com protocolos que compõem a pilha de protocolos TCP/IP.

2.2 Páginas web e o HTTP

Uma característica que atrai a maioria dos usuários web é que ela funciona por demanda. Através da web os usuários podem acessar conteúdos específicos que sejam do seu gosto, diferentemente de mídias como rádio e televisão, onde a variedade de conteúdos é mais limitado.

Outra característica muito notável na web é que o conteúdo pode ser gerado por qualquer pessoa e acessado facilmente de qualquer lugar do mundo. Neste contexto, quando estamos falando de conteúdo, estamos falando basicamente de páginas web. Uma página web é tipicamente constituída de objetos web. Um objeto web é nada mais do que um arquivo, como uma

imagem JPEG, um arquivo HTML, um clipe de áudio, etc.

2.2.1 Descrição do HTTP

O HTTP - Protocolo de Transferência de Hipertexto (*HyperText Transfer Protocol*) é um protocolo da camada de aplicação da web, é o protocolo de transferência utilizado em toda a *World Wide Web*. Esse protocolo é definido na RFC 1945 (BERNERS-LEE; FIELDING; FRYSTYK, 2005) e na RFC 2616 (FIELDING et al., 1999) e assim como o TCP, é implementado no lado do cliente o no lado do servidor. Desta forma, o HTTP é responsável por definir a estrutura e como o cliente e servidor trocam mensagens entre si.

Embora o HTTP tenha sido projetado para utilização na web, ele foi criado de modo mais geral que o necessário, visando futuras aplicações orientadas a objeto. Por essa razão, são aceitas operações chamadas de métodos, diferentes de uma solicitação de uma página web (TANENBAUM, 2003).

2.2.2 Conexões HTTP

O HTTP entende que um cliente web é aquele que solicita um conteúdo, por exemplo, um navegador tentando acessar uma página web. Já o servidor é aquele que fica aguardando por essas solicitações.

É importante entendermos como o HTTP trabalha, pois ele usa o TCP como seu protocolo de transporte e este primeiro é o protocolo muito utilizado se tratando da *World Wide Web*. Usualmente, um navegador entra em contato com um servidor e é estabelecido uma conexão TCP na porta 80 do servidor.

O HTTP faz uso intenso do TCP em suas comunicações. A vantagem de usar o TCP é que nem os navegadores, nem os servidores têm de se preocupar com as mensagens perdidas e/ou duplicadas, assim como as mensagens de reconhecimento (confirmação), sendo que essas preocupações ficam por conta do TCP (TANENBAUM, 2003). Na implementação antiga do HTTP 1.0, depois que uma conexão fora estabelecida, uma única solicitação era enviada e uma única resposta era devolvida, e portanto, a conexão TCP era encerrada.

Se pensarmos que antigamente as páginas web eram basicamente um texto HTML, essa abordagem funciona bem. No entanto, com a evolução da internet, as páginas web passaram a ter vários objetos web, imagens, vídeos e ícones, tornando necessário que a conexão durasse mais tempo. A partir da versão do HTTP 1.1, os navegadores passam a usar o que hoje é

conhecido por conexões persistentes.

2.2.3 Conexões persistentes do HTTP

Em uma conexão persistente o servidor deixa a conexão TCP aberta após enviar uma resposta. A ideia por trás disso é que requisições e respostas subsequentes entre os mesmos cliente e servidor podem ser enviadas por meio da mesma conexão. (KUROSE; ROSS, 2006)

Desta forma, uma página web inteira, formada pelo arquivo-base HTML e todos os seus elementos (objetos web), pode ser transferida através de uma mesma conexão persistente. Consequentemente, se várias páginas são hospedadas por um mesmo servidor, elas podem ser transmitidas na mesma conexão persistente.

Normalmente, o servidor HTTP fecha uma conexão quando ela não é usada durante um certo tempo, um intervalo de pausa configurável (KUROSE; ROSS, 2006). Vimos anteriormente que a evolução das páginas web tornaram necessário a utilização de conexões persistentes a fim de otimizar a utilização da conexão para transferir vários objetos web. O que podemos perceber na prática é que muitas vezes essas conexões ficam ociosas por muito tempo e que essa talvez não seja a solução mais eficiente para esse problema.

2.3 Request For Comments

Quando falamos de sistemas computacionais, tudo precisa ser padronizado. Se tratando de protocolos de comunicação, é ainda mais importante que os padrões sejam obedecidos para que as mensagens entre as partes comunicantes sejam recebidas e interpretadas de forma correta.

No âmbito da internet, os documentos que padronizam e regularizam o comportamento de protocolos são as RFCs. As RFCs são documentos técnicos produzidos e mantidos pela IETF (*Internet Engineering TaskForce*), em português, Força Tarefa de Engenharia da Internet, e eles detalham todos os aspectos dos protocolos e serviços na internet. RFC é um acrônimo de *Request for Comments*, que significa em português, Requisições de Comentários.

A IETF é uma organização composta de técnicos, agências, fabricantes, fornecedores e pesquisadores da área de tecnologia, que tem a missão de identificar e propor soluções a questões e problemas relacionados à utilização da Internet.

A IESG (*Internet Engineering Steering Group*) é o corpo principal composto pela IETF que, juntamente com membros de outros segmentos da área de tecnologia, fornece a revisão

final para os documentos que definem padrões que serão utilizados na internet.

Existem diversos tipos de RFCs, alguns são documentos longos e complexos que esmiúçam vários detalhes de implementação e funcionamento de protocolos, com é o caso da RFC 793 (POSTEL et al., 1981), que fala sobre o TCP. Outras RFCs são apenas para título informativo, como é o caso da RFC 2026 [1], que define como se dá o processo de elaboração e aprovação de uma RFC.

As RFCs são sempre identificadas por um número, que é dado sequencialmente a medida em que novas RFCs são submetidas. Tecnicamente, a cada RFC é atribuído um status que diz respeito ao estado da padronização do protocolo. Os status podem ser:

- Informativo (*Informational*)
- Experimental (*Experimental*)
- Melhor Prática Atual (*Best Current Practice / BCP*)
- Trilha dos Padrões (*Standards Track*)

  Proposto (*Proposed Standard*)

  Rascunho (*Draft Standard*)

  Padrão da Internet (*Internet Standard*)
- Histórico (*Historic*)

Quando uma RFC precisa ser atualizado, usualmente, é criado um novo RFC com as alterações que foram feitas. Desta forma, quando um RFC é publicado ele não sofre mais alterações. Um RFC é classificado como "Informativo"quando não é um padrão da internet. Na realidade, ele não descreve um protocolo ou serviço em específico, apenas trás informações refetentes a um processo

No caso deste trabalho, estamos falando do TCP Fast Open, que encontra-se no status de experimental. Neste caso, isso significa que ainda não foi possível testar a real efetividade do protocolo ou que não foram feitos estudos suficientes para garantir que ele opere com segurança e eficiência antes de ser aprovado como um padrão para a internet.

[1] https://www.ietf.org/rfc/rfc2026.txt

Todos os RFCs podem ser consultados gratuitamente na Internet. Podemos acessar o repositório de RFCs no site rfc-editor.org [2], onde é possível buscar as RFCs por nome, palavra-chave, autor ou número clicando no link "Search RFCs".

[2] https://www.rfc-editor.org/

3 TCP - TRANSMISSION CONTROL PROTOCOL

Como o foco do nosso trabalho reside em fazer uma análise de um protocolo variante ao TCP convencional, vamos dedicar um capítulo para dissertar sobre o TCP. Nesse capítulo, consideramos como o TCP é estruturado, quais são suas características e como ele opera nos sistemas computacionais comunicantes.

O TCP (Protocolo de Controle de Transmissão) e o IP (Protocolo de Internet) são dois dos protocolos mais conhecidos na Internet (KUROSE; ROSS, 2006). Essa popularidade se deve muito ao fato de que o TCP é um protocolo confiável, característica fundamental para muitas aplicações que rodam na internet.

Esse protocolo foi projetado especificamente para oferecer um fluxo de bytes fim a fim confiável em uma inter-rede não confiável (TANENBAUM, 2003). Ao longo das últimas décadas desenvolveram-se muitas tecnologias de redes apresentando diferentes hardware e software. Com a evolução dos negócios e da tecnologia tornou-se necessário a interconexão de diferentes tipos de redes.

Para que isso possa ocorrer, é preciso que se estabeleçam conexões entre redes quase sempre incompatíveis. Muitas dessas interconexões são realizadas através de equipamentos conhecidos como gateways, que fazem a conversão de um ou mais protocolos de transmissão para compatibilizar as diversas arquiteturas. Um conjunto de redes interconectadas é chamado inter-rede ou internet.

Uma inter-rede se difere de uma única rede porque suas diversas partes podem ter topologias, larguras de banda, atrasos, tamanho de pacotes e outros parâmetros completamente diferentes (TANENBAUM, 2003). Por essa razão o TCP é um protocolo robusto pois, se adapta dinamicamente às propriedades dessas redes.

3.1 Características

A palavra confiável neste âmbito significa que o protocolo provê um serviço confiável de transmissão de dados (KUROSE; ROSS, 2006). Em outras palavras, o TCP garante que todos os dados enviados ao destino cheguem íntegros e na ordem correta. Aliado a isso está a segurança quanto à reposição de pacotes perdidos.

O TCP é um protocolo da camada de transporte que é dito orientado a conexão (KUROSE; ROSS, 2006). Isso significa que antes que um processo possa enviar dados para um

outro processo através do TCP, é necessário que haja uma "apresentação".

Na prática, isso significa enviar dados preliminares para que sejam estabelecidos os parâmetros para a transferência de dados entre os processos (KUROSE; ROSS, 2006). Essa apresentação é chamada *handshake*, que significa "aperto de mãos"e é usada também para iniciar muitas variáveis de estado que serão utilizadas durante a conexão.

Mais uma característica do TCP é prover um serviço *full-duplex* (KUROSE; ROSS, 2006), o que significa que um processo pode enviar dados para outro processo e, ao mesmo tempo receber dados deste mesmo processo. Na realidade, os processos comunicantes trocam dados somente entre si, por isso uma conexão TCP é sempre ponto a ponto (KUROSE; ROSS, 2006), de um único remetente para um único destinatário.

Outra característica importante do TCP é fato de possuir controle de congestionamento. Desta forma, ele implementa algumas funcionalidades para não permitir que a rede seja inundada com segmentos que um receptor não esteja apto a receber (KUROSE; ROSS, 2006). Uma dessas funcionalidades é o algoritmo *slow-start* ou partida-lenta que será explicado em detalhes posteriormente.

3.2 Vantagens e Desvantagens

Por todas as características apresentadas na seção anterior o TCP oferece muitas vantagens se comparado a outros protocolos de transmissão existentes. A principal vantagem para o uso do TCP em relação a qualquer outro protocolo de transporte é a confiabilidade e integridade dos dados durante o processo de transmissão. Quando pensamos em acessar uma página web através do navegador por exemplo, gostaríamos que todos os elementos fossem exibidos para nós. Não é do interesse do usuário que a página esteja incompleta ou que os dados exibidos não estejam corretos. Aplicações como essas requerem o serviço confiável de dados e é por esse motivo que o TCP é tão utilizado.

Além disso, o TCP possui controle de congestionamento, que não é tanto um serviço fornecido à aplicação, é mais um serviço dirigido a internet como um todo, para um bem geral (KUROSE; ROSS, 2006). Resumidamente, o que o controle de congestionamento faz é evitar que qualquer outra conexão TCP abarrote os enlaces e comutadores entre hospedeiros comunicantes com uma quantidade excessiva de tráfego.

Existem aplicações que não necessitam de confiabilidade de transmitir seus dados. As aplicações mais comuns que podemos pensar com essas características são os serviços de *stre-*

aming. Se pensarmos que esses serviços apenas transmitem os quadros das imagens que devem ser mostradas (afim de formar um vídeo), não existe real necessidade que todos esses dados cheguem até o destinatário.

Outras aplicações em que não é necessário garantir a integridade de todos os pacotes transmitidos são para jogos online. Nesses dois cenários, usa-se o protocolo UDP por ser mais simples e eficiente.

3.3 Aplicações

O TCP é um dos protocolos mais utilizados na internet e são incontáveis as aplicações que fazem o uso desse protocolo. Podemos pensar em um correio eletrônico, por exemplo, que faz uso do protocolo SMTP na camada de aplicação, mas que na camada subjacente faz uso do TCP.

Assim como páginas web que utilizam o protocolo HTTP, mas que na camada subjacente faz uso intenso do TCP para transferência de dados. O mesmo serve para transferência de arquivos (FTP) e acesso a terminal remoto (Telnet). Toda aplicação que precisar garantir a integridade da chegada e recebimento dos dados usará, tipicamente, o TCP para fazer isso.

3.4 Estrutura do segmento TCP

Nesta seção vamos abordar um pouco sobre como é estruturado um segmento TCP, para que mais tarde fique claro como e, porque esses campos do cabeçalho são manipulados. Como visto anteriormente, o TCP é um protocolo da camada de transporte orientado a conexão, isso significa que vários campos e flags (na realidade a maioria) do cabeçalho são utilizados para garantir a integridade e ordem dos dados que estão sendo transmitidos.

Um segmento TCP é formado por um cabeçalho, onde são armazenados metadados usados para manter as características intrínsecas do TCP, e um campo de dados, ondem são armazenados os bits que contém os dados a serem transmitidos. Um cabeçalho TCP tem seu tamanho, tipicamente, fixado em 20 bytes. A figura 3.1 mostra a estrutura de um segmento TCP, onde cada um dos campos tem um tamanho e funcionalidade específicas que são elucidadas adiante.

- Porta de Origem: informa o número da porta de origem;
- Porta de Destino: informa o número da porta de destino;

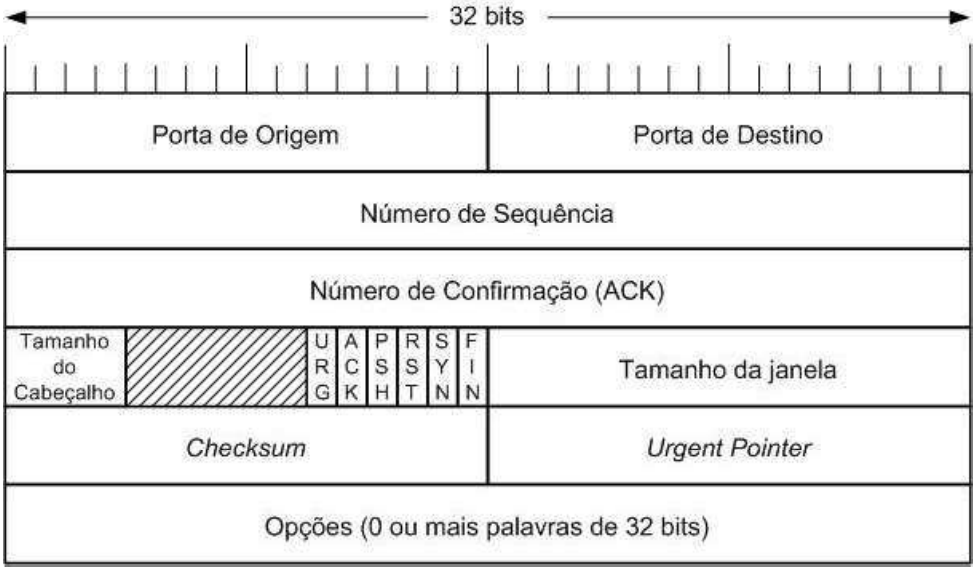

Figura 3.1 – Estrutura do segmento TCP

- Número de sequência e Número de confirmação (ACK): campo de 32 bits utilizado para implementar o serviço confiável de transferência de dados;
- Tamanho do cabeçalho: informa quantas palavras de 32 bits existem no cabeçalho TCP. Informação necessária pois o campo de Opções pode ter tamanho variável;
- Tamanho da janela (janela de recepção): serve para indicar a quantidade de dados que o destinatário está disposto a aceitar;
- Flag:
  - URG: usado para indicar um deslocamento de bytes a a partir do número de sequência atual;
  - ACK: serve para reconhecimento de um segmento;
  - RST, SYN, FIN: servem para negar, iniciar e finalizar um pedido de conexão, respectivamente;
  - PSH: indica dados com a flag PUSH, onde o receptor é solicitado a entregar os dados a aplicação imediatamente;
  - RST: é utilizado para reinicializar uma conexão que tenha sido identificada alguma falha pelo host ou por qualquer outra razão;
- *Checksum*: usado para aumentar a confiabilidade, conferindo o total de verificação do cabeçalho;

- Dados: cadeia de bytes com os dados do remetente;
- Opções: usado quando remetente e destinatário negociam o MSS e outras finalidades;

Anexado a esse cabeçalho está o campo de dados, que pode ter tamanho variável. Com isso se forma um segmento TCP. Dois fatores restringem o tamanho do segmento. Primeiro, cada segmento, incluindo o cabeçalho TCP, deve caber na carga útil do IP que é de 65515 bytes. Em segundo lugar, cada rede tem uma unidade máxima de transferência ou MTU (*Maximum Transfer Unit*) que também deve ser respeitada. Geralmente o valor da MTU tem 1500 bytes, que é o tamanho da carga útil para Ethernet (TANENBAUM, 2003).

3.5 Conexão TCP

Para que uma conexão TCP seja iniciada, um processo que esteja rodando no lado do cliente informa ao TCP cliente que deseja estabelecer uma conexão com um processo em um servidor (KUROSE; ROSS, 2006). A partir disso, o TCP cliente estabelece uma comunicação com o TCP servidor, seguindo as seguintes etapas:

- **Etapa 1.** O cliente envia um segmento TCP especial ao servidor através do TCP, chamado segmento SYN. Esse segmento não contém nenhum dado, é apenas um segmento TCP com a flag SYN do campo do cabeçalho TCP com bit igual a 1. Esse campo do cabeçalho informa que esse segmento é um requerimento de abertura de conexão. Além disso, o cliente escolhe um número aleatório como número de sequência inicial.
- **Etapa 2.** Se aceitar o pedido, o servidor responde com um segmento SYN-ACK. Esse segmento serve para informar ao cliente que o segmento anterior foi recebido e que o servidor confirma a abertura de conexão. O servidor também escolhe aleatoriamente seu número de sequência, esse servirá para que o cliente saiba qual o próximo pacote a ser esperarodo servidor.
- **Etapa 3.** Por fim, o cliente envia um segmento ACK até o servidor, para reconhecer que a conexão foi aceita pelo servidor. Aqui, o bit SYN volta a ter valor 0, uma vez que a conexão foi estabelecida.

A figura 3.2 mostra de forma simplista como se dá a troca de mensagens entre um cliente que solicita uma conexão e posteriormente solicita um arquivo do servidor.

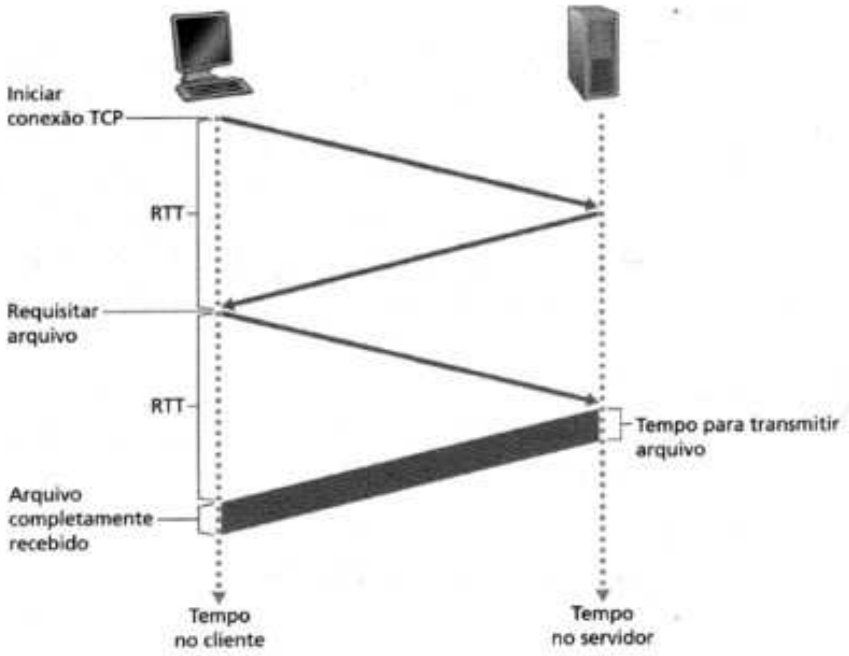

Figura 3.2 – Transferência da um arquivo pela rede

A imagem tem o intuito de mostrar um importante fator da comunicação entre o par-cliente servidor, o RTT. O *Round-Trip Time*, ou RTT como é chamado, é a demora (ou atraso) para que pacotes sejam enviados de um computador de origem (cliente) até um computador destino (servidor) (KUROSE; ROSS, 2006). Esse conceito será importante para justificar mais tarde a utilização do protocolo a ser estudado.

A figura 3.3 ilustra como é o processo de apresentação de três vias ou *handshake*, denominado assim pelo fato de que são necessárias três mensagens entre o cliente e o servidor para que a conexão seja estabelecida e haja a troca de dados entre essas entidades.

É importante lembrar que é na fase de *handshake* que são alocados os *buffers* de envio e recepção usados na comunicação, assim como outras variáveis necessárias para o TCP fazer o controle de fluxo e congestionamento (KUROSE; ROSS, 2006).

Uma vez estabelecida a conexão, o cliente pode enviar dados ao servidor e vice-versa. Aqui, o TCP direciona os dados recebidos para seus respectivos *buffers* de envio e recepção (KUROSE; ROSS, 2006). Como páginas web contém objetos maiores do que os tamanhos típicos de segmentos TCP, é comum que o TCP divida a mensagem que deve ser transmitida em vários segmentos.

A quantidade máxima de dados que pode ser colocada em um segmento TCP é limitada pelo tamanho máxido do segmento, ou MSS (*Maximum Segment Size*), como definido por

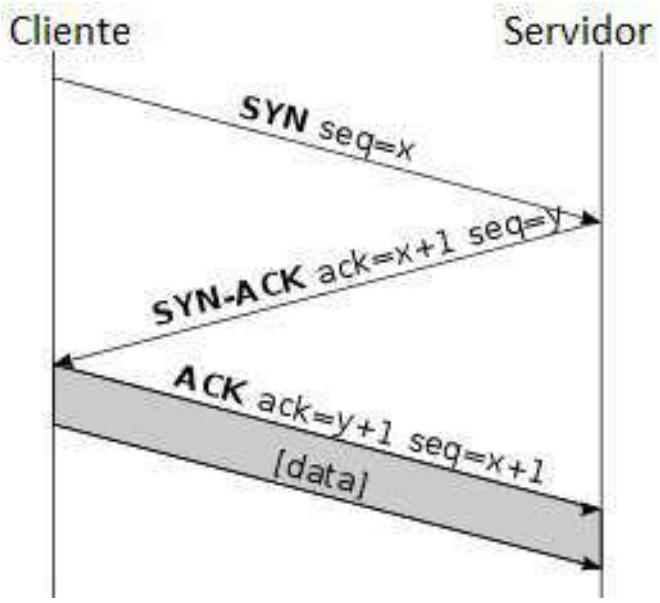

Figura 3.3 – Apresentação de três vias do TCP (handshake)

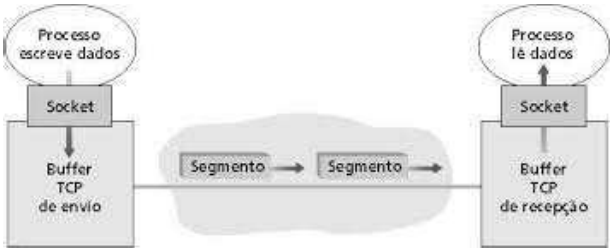

Figura 3.4 – Buffers TCP de envio e de recepção

KUROSE; ROSS (2006). É interessante notar que o MSS não limita o tamanho máximo do segmento TCP, mas sim a quantidade máxima de dados que pode ser colocada em cada segmento.

3.5.1 Algoritmo de partida lenta (*slow start*)

Para entendermos melhor como funciona o algoritmo de *slow start*, precisamos entender um pouco melhor como opera o controle de congestionamento do TCP.

O controle de congestionamento funcione de forma eficiente a estratégia adotada pelo TCP é de obrigar que cada remetente limite a taxa à qual envia tráfego para sua conexão. A ideia é que, a medida que o TCP remetente percebe que não está inundando a rede com pacotes que serão perdidos ou que há pouco congestionamento entre o destinatário, ele aumente a taxa à qual envia mensagens.

Como visto na seção anteior, cada lado da conexão TCP consiste de um buffer de envio, um buffer de recepção, além de diversas outras variáveis de controle. O que o mecanismo de controle de congestionamento faz é manipular uma variável conhecida como janela de conges-

tionamento em cada lado da conexão.

Essa variável determina qual a limitação à taxa à qual um remetente TCP pode enviar tráfego para dentro da rede. Especificamente, a quantidade de dados não reconhecidos em um hospedeiro não pode exceder o mínimo do tamanho da janela de congestionamento e da janela de recepção (KUROSE; ROSS, 2006), ou seja, deve respeitar a seguinte regra:

$$LastByteSent - LastByteAcked \leq min(CongWin, RcvWindow)$$

Onde *LastByteSent* representa o último byte enviado, *LastByteAcked* é o último byte que foi enviado e teve seu recebimento reconhecido, *min()* é uma função mínimo que retorna o menor valor entre seus parâmetros, *CongWin* representa a janela de congestionamento e finalmente *RcvWindow*, é o tamanho da janela de recepção.

Quando uma conexão TCP é iniciada, o valor da janela de congestionamento tipicamente é inicializada em 1 MSS (ALLMAN; FLOYD; PARTRIDGE, 2002), o que torna a taxa inicial de envio de aproximadamente MSS/RTT.

Exemplo: Se o tamanho máximo de um segmento (MSS) estiver em 500 bytes e tivermos um RTT de 100 milisegundos, a taxa de envio inicial se torna 500 bytes / 100 milisegundos = 40 kbps. Notamos que a largura de banda disponivel normalmente é superior ao valor da relação MSS/RTT e, por essa razão, seria um desperdício aumentar a taxa de transmissão apenas linearmente. A RFC 3390 (ALLMAN; FLOYD; PARTRIDGE, 2002) especifica um padrão para o TCP para incrementar o tamanho da janela de congestionamento de forma exponencial, duplicando o valor de *CongWin* a cada RTT.

O fator delimitante para a função que duplica o valor de *CongWin* é até que ocorra um evento de perda, ou seja, até que um segmento que foi enviado não tenha sido reconhecido. A partir dai, a taxa de aumento então se torna linearmente, por isso essa fase inicial é denomiada partida lenta (*slow start*).

Resumidamente o remetente gera o crescimento exponencial aumentando o valor de CongWin de 1 MSS toda vez que um segmento transmitido é reconhecido. O que na prática acontece é que o TCP envia o primeiro segmento para dentro da rede e espera por um reconhecimento. Se esse segmento for reconhecido antes de um evento de perda, o remetente TCP aumenta a janela de congestionamento em 1 MSS e envia dois segmentos de tamanho máximo. Se esses segmentos forem reconhecidos antes de eventos de perda, o processo se repete, resultando em uma janela de congestionamente de 4 MSS, e assim por diante.

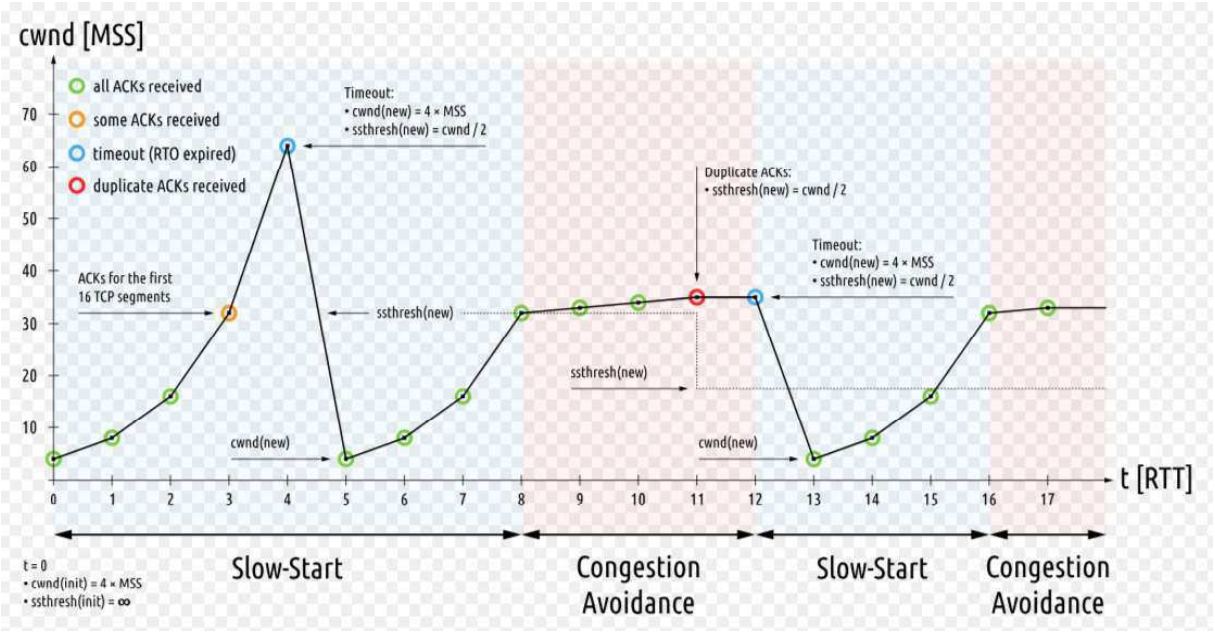

Figura 3.5 – Aumento da *CongWin* no algoritmo de slow-start (FLESHGRINDER, 2014)

Assim que é estabelecida a conexão, o remetente escreve bytes no *buffer* de envio do TCP. Tais bytes são agrupados em segmentos do tamanho do MSS. O TCP, tipicamente, tem o MSS é igual a janela de recepção, que se inicia em 1. Na relidade, o que o TCP faz é esperar por um reconhecimento do segmento que acabou de ser enviado (KUROSE; ROSS, 2006). Toda vez que esse reconhecimento é recebido (em forma de segmento ACK), o TCP incrementa o tamanho do MSS em 1 segmento.

A ideia do algoritmo é incrementar o tamanho da janela em 1 segmento afim de que a taxa que novos pacotes injetados na rede seja a mesma em que as confirmações (ACK) são enviadas pelo outro destino. Esse processo otimiza o quanto de dados é trocado entre o par cliente-servidor e evita que equipamentos no núcleo da rede sejam inundados com pacotes que serão descartados.

Após a fase de *handshake* a conexão segue normalmente com a troca de mensagens até que uma das partes solicite o encerramento da conexão. O funcionamento típico do TCP na troca de dados é mostrado na figura 4.4.

Como sabemos, os enlaces de comunicação no núcleo da rede podem estar congestionados durante a conexão. Isso pode fazer com que pacotes com dados ou segmentos especiais do TCP se percam na rede (KUROSE; ROSS, 2006). É parte de entender um protocolo saber quais atitudes ele toma quando se depara com esses cenários.

Para resolver esse problema, as implementações do TCP contam com temporizadores.

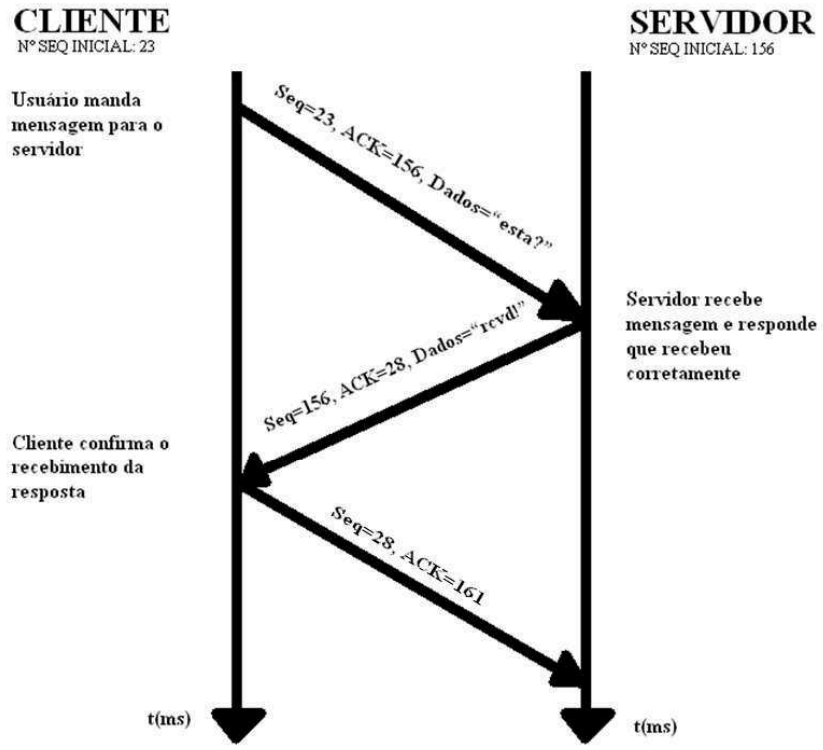

Figura 3.6 – Funcionamento do TCP após o *handshake*

Temporizadores são acionados quando um novo segmento é enviado. O TCP então, inicia um temporizador e aguarda por um reconhecimento do pacote enviado (KUROSE; ROSS, 2006). Quando o temporizador é esgotado, o TCP retransmite o último pacote não reconhecido e ajusta o próximo tempo de expiração do temporizador para o dobro do anterior.

3.5.2 Encerramento de conexão TCP

Como vimos anteriormente, o TCP faz uso de algumas flags em seu cabeçalho para fazer os controles necessários em uma conexão. Para encerrar uma conexão, qualquer um dos lados (cliente ou servidor) pode enviar um segmento com o bit FIN ativado, o que significa que há mais dados para serem transmitidos da sua parte.

Neste momento, o solicitante do fim de conexão não envia mais dados, mas isso não garante que os dados no outro sentido também tenham sido interrompidos. Assim como durante o *handshake*, é necessário que a outra parte reconheça que houve um pedido de encerramento de conexão e esta parte deve confirmar o fim da conexão, fazendo o reconhecimento do segmento com o bit FIN ativo recebido.

Esse processo é feito enviando um segmento ACK para reconhecer o segmento anterior-

mente recebido, da mesma maneira que é feita no *handshake*. Tipicamente, são quatro segmentos necessários para encerar uma conexão, um FIN e um ACK para cada lado. Apesar de que, o segmento de reconhecimento do receptor da solicitação pode enviar no mesmo segmento a flag FIN ativada, poupando um segmento no processo.

Assim que ambas as partes confirmarem e receberem a confirmação de que a conexão deve ser encerrada é que de fato ambos não trocam mais dados.

4 TCP FAST OPEN

Esse capítulo é dedicado a explicar detalhadamente o TCP Fast Open (TFO). A melhor forma de descrever o TCP Fast Open (CHENG et al., 2014) seria afirmar que ele é uma variante do TCP convencional, que permite a troca de dados entre cliente e servidor durante o *handshake* de forma segura. Evidentemente, para que isso aconteça ele deve garantir proteção tanto ao cliente quanto a servidor de ataques maliciosos.

O TCP Fast Open foi discutido em 2011, através de RADHAKRISHNAN et al. (2011) com a finalidade de propor um protocolo variante ao TCP que permitisse a transmissão de dados durante o *handshake*. Na realidade, como apontado em CHENG et al. (2014), é possível através do TCP transmitir dados durante os pacotes SYN. No entanto, vide sua RFC (ALLMAN; PAXSON; BLANTON, 2009), os dados encontrados nesses pacotes de reconhecimento (SYN) não podem ser entregues para a aplicação. A ideia então é que o TFO remove essa limitação e permite que os dados sejam entregues para as aplicações.

Para garantir a transmissão e entrega dos dados durante o *handshake*, o componente chave para o funcionamento do TFO é um *Cookie*, contendo um MAC (*Message Authentication Code*). Desta forma, um cliente que deseja utilizar o TFO deve solicitar um *Cookie* para o servidor através de uma conexão convencional utilizando o TCP, da mesma forma como descrito no capítulo anterior. Como detalhado em RADHAKRISHNAN et al. (2011), a solicitação de um *Cookie* segue da seguinte maneira:

- 1. O cliente envia um segmento SYN para o servidor com uma solicitação de Cookie TFO no campo de opções do TCP;
- 2. O servidor gera um *Cookie* encriptando o endereço IP do cliente sob uma chave secreta. O servidor então responde ao cliente com um pacote SYN-ACK que contém o *Cookie* gerado;
- 3. O cliente armazena o *Cookie* para futuras conexões TFO;

A figura 4.1 ilustra a solicitação de um *Cookie* para um servidor que esteja apto a receber conexões através do TFO:

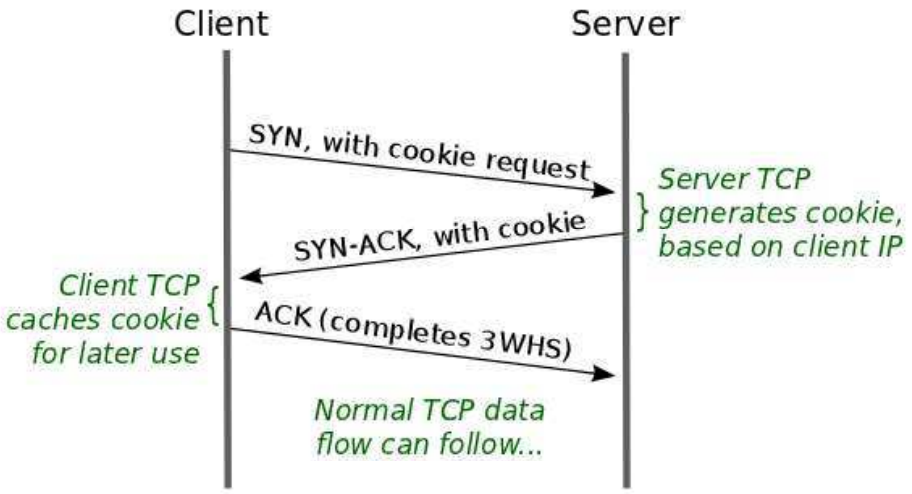

Figura 4.1 – Solicitação de um *Cookie* TFO

4.1 Conexão TCP Fast Open

A partir da solicitação de um *cookie*, um cliente pode, para conexões subsequentes, solicitar por uma conexão TFO em um servidor que esteja apto a receber tais conexões (RADHAKRISHNAN et al., 2011). Uma conexão utilizando TFO, segundo CHENG et al. (2014), acontece da seguinte maneira:

- 1. O cliente envia um segmento SYN, com dados e o *Cookie* armazenado para o servidor;
- 2. O servidor valida o *Cookie*:
  - a. Se o *Cookie* é válido, o servidor envia um segmento SYN-ACK que reconhece tanto o pedido de conexão (SYN) como também os dados. O servidor então envia os dados para aplicação;
  - b. Se não, o servidor descarta dos dados recebidos e envia um segmento SYN-ACK que reconhece apenas o número de sequência do cliente;
- 3. Se o servidor aceitar os dados provenientes do segmento SYN, ele pode enviar uma resposta antes do *handshake* ser finalizado. A quantidade máxima é definida pelo controle de congestionamento do TCP definida em RFC 5681 (ALLMAN; PAXSON; BLANTON, 2009);
- 4. O cliente envia um segmento ACK reconhecendo o SYN e os dados recebidos do servidor. Se os dados do cliente não foram reconhecidos, o cliente retransmite-os no segmento ACK;

- 5. O restante da conexão procede como se usasse o TCP convencional. O cliente pode repetir várias operações com TFO uma vez que possui um *Cookie* (que ainda não tenha sido expirado pelo servidor).

A figura 4.2 mostra como se dá a troca de mensagens entre o cliente e o servidor antes de proceder para uma conexão com TCP convencional.

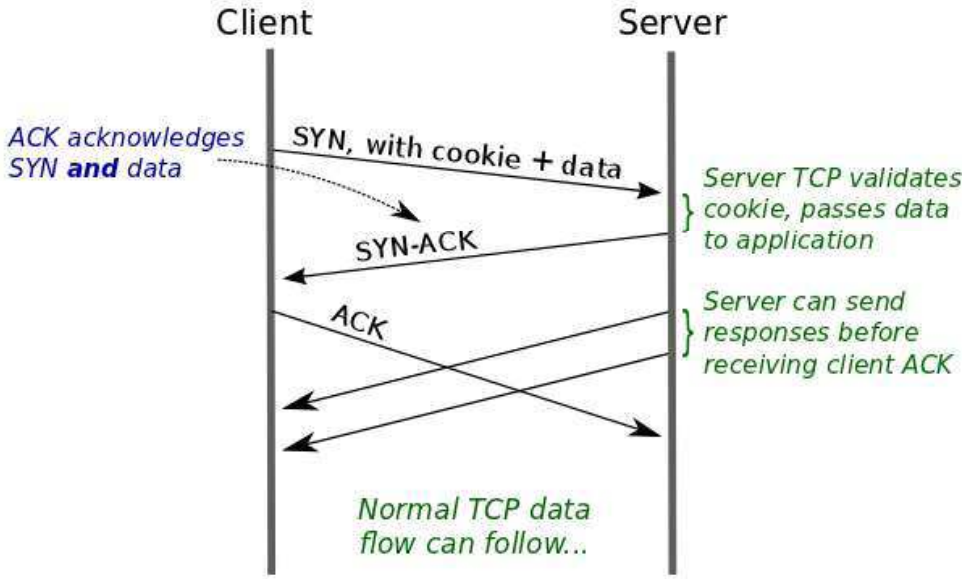

Figura 4.2 – Conexão entre cliente e servidor utlizando TFO

Perceba que, até o momento em que o *hanshake* foi concluído, tanto cliente como servidor trocaram dados entre si e, logo que os recebem podem passar os dados para a aplicação. Isso faz com que dados sejam entregues a aplicação mais cedo, poupando um RTT na conexão.

4.2 O Cookie TFO

Como dito anteriormente, para garantir a troca segura de dados durante o *handshake*, o coração do TFO faz uso de um *Cookie*. O *Cookie* nada mais é do que uma string de dados, para validar o IP do proprietário. O servidor encripta o endereço IP do cliente através de uma chave secreta e, gera um *Cookie* de comprimento de até 16 bytes (CHENG et al., 2014).

Em CHENG et al. (2014) é determinado que o *Cookie* tenha as seguintes propriedades:

- 1. O *Cookie* autentica o endereço IP da fonte contido no segmento SYN. O endereço IP deve ser IPv4 ou IPv6;

- 2. O *Cookie* só poderá ser gerado pelo servidor e por nenhuma outra entidade, nem o cliente;

- 3. A geração e validação do cliente são rápidas em relação ao processamento de segmentos SYN e SYN-ACK;
- 4. O servidor tem a opção de encriptar outras informações no *Cookie* e aceitar mais de um *Cookie* válido por cliente em um determinado tempo;
- 5. O *Cookie* expira depois de um determinado tempo. Isso pode ser feito pela mudança periódica da chave que gera o cookie ou incluindo um temporizador quando o cookie é gerado;

Esses itens tem o intuito de garantir a segurança no que se refere ao uso do TFO. Note que, sem o chave para geração do cookie, o cliente não poderá gerar um *Cookie* válido a não ser que quebre a cifra usada para encriptar seu endereço IP.

Outro fator interessante é a necessidade do servidor em expirar cookies depois de um determinado tempo, fazendo a rotação da chave secreta na criptografia (CHENG et al., 2014). Isso se deve ao fato de que, um cliente poderia fazer várias solicitações por *cookies* e guardá-los para fazer um ataque sincronizado contra o servidor, uma vez que todos os *cookies* seriam válidos.

4.2.1 Cookie do lado do cliente

Do lado do cliente, é necessário armazenar o *Cookie* para utilizá-lo em futuras conexões que façam uso do TFO. A documentação de CHENG et al. (2014) também sugere que, quando o cliente armazenar o *Cookie*, ele também armazene o MSS do servidor. Normalmente, essa informação MSS é enviada pelo servidor no segmento SYN-ACK, mas é vantagoso ter essa informação com antecedência.

Como visto anteriormente, o TCP tipicamente faz uso do algoritmo de partida lenta (KUROSE; ROSS, 2006) ao iniciar uma conexão no par cliente-servidor. Em ordem de acelerar o processo na troca de mensagens, é muito útil para o cliente saber a quantidade máxima de informação que o servidor está apto a receber (CHENG et al., 2014). Assim, o cliente pode colocar a quantidade certa de dados ao enviar o segmento ACK para o servidor, podendo aumentar a performace do protocolo.

Também cabe ao cliente, guardar informações quanto a respostas negativas nas tentativas de conexão com um servidor. Isso inclui falta de reconhecimento dos dados enviados pelo servidor, ou o não reconhecimentos de segmentos SYN-ACK. Essas informações são úteis para

que o cliente desabilite tentativas de usar o TFO para um determinado servidor do qual tenha recebido respostas negativas (CHENG et al., 2014).

4.2.2 Cookie do lado do servidor

O TFO é um mecanismo para melhorar a performace em conexões que utilizem o TCP para alguns cenários específicos, normalmente para conexões de curta duração, como explanado anteriormente. Por essa razão, a implementação do TFO deve ser completamente compatível com as implentações existente do TCP, tanto do lado do cliente como do lado do servidor (RADHAKRISHNAN et al., 2011).

Como descrito em CHENG et al. (2014), o servidor que faz uso de uma implementação do TFO deve sustentar portanto, duas características:

- O TFO deve estar desabilitado por padrão. Ele deve ser ativado especificamente por uma aplicação;
- Deve-se limitar o número de conexões ativas que utilizem TFO. Quando o limite for estourado, o servidor deve negar quaisquer pedidos de novas conexões TFO;

4.3 Considerações de segurança do TFO

Como mencionado anteriormente, o objetivo do TCP Fast Open é permitir, de forma segura, que dados sejam enviados com o hanndshake em andamento (RADHAKRISHNAN et al., 2011). O funcionamento descrito acima alude a algumas características específicas seguidas pelo protocolo para que isso ocorra, os próximos paráfragos falam sobre os problemas de segurança considerados quando o protocolo foi proposto.

Um dos problemas considerados por CHENG et al. (2014) é a exaustão de recursos do servidor. Se fosse permitida a troca de dados durante o *handshake* sem qualquer tipo de autenticação, um cliente poderia inundar o TCP do servidor com mensagens, fazendo-o processar todos os pacotes recebidos para redirecionar os dados para as aplicações corretas.

O *Cookie* de autenticação serve para mitigar esse problema, uma vez que sua validação por parte do servidor é pouco custosa. Enquanto que, se um cookie for inválido, os dados provenientes do segmento serão ignorados e a conexão volta a ser uma conexão TCP normal. Nesse segundo caso, o servidor pode usar técnicas já existentes para previnir que haja exaustão de recursos por parte do servidor.

Mesmo que um cliente obtenha vários *cookies* válidos para elaborar um ataque, assumindo que qualquer pode solicitar um cookie, o servidor pode se proteger de duas maneiras (CHENG et al., 2014). A primeira delas, como mencinado anteiormente, é fazer a rotação da chave que faz a criptografia do IP do cliente, fazendo com que apenas uma certa quantidade de *cookies* seja aceita para um determinado período de tempo.

A segunda forma, é configurar no servidor um limite de conexões ativas que utilizem o TFO (RADHAKRISHNAN et al., 2011). Com esse limite, pode-se imaginar que um cliente mau intencionado pode sobrecarregar o servidor com vários pedidos de conexão, fazendo com que o servidor desabilite temporariamente o TFO.

Isso de fato pode acontecer, mas o proponente do protocolo esclarece que isso não seria de interesse de um atacante, pois mesmo com o TFO desabilitado, a conexão retrocederia para uma conexão TCP normal e o serviço permaneceria intacto (RADHAKRISHNAN et al., 2011).

Outra situação considerada por CHENG et al. (2014) é quando um cliente compartilha o mesmo IP através de um NAT (*Network Address Translation*). O NAT é um protocolo que faz a tradução dos endereços IP e portas TCP da rede local para a Internet. Com isso, o pacote a ser enviado ou recebido na sua rede local, vai até o servidor onde seu IP é trocado pelo IP do servidor, a substituição do IP da rede local valida o envio do pacote na internet.

Por isso, um cliente por trás de um NAT pode obter cookies válidos e lançar ataques contra outros clientes que façam uso do mesmo IP posteriormente. Para solucionar isso, é sugerido que o servidor gere o cookie usando o IP e um timestamp. Isso pode ser feito concatenando o IP e o timestamp do TCP e encriptando-os a fim de formar o MAC. Essa abordagem permite que o servidor gere diferentes cookies para clientes que compartilhem o mesmo IP através do NAT.

5 METODOLOGIA DE PESQUISA

Esse trabalho é dividido em diferentes etapas. A primeira delas é verificar o funcionamento do TCP Fast Open. Isso é feito através do estudo na RFC 7413 (CHENG et al., 2014) que define o protocolo. Essa etapa tem o intuito de identificar os detalhes de implementação do TFO e analisar as características específicas necessárias para o protocolo funcionar de forma eficiente. Além disso, é importante indagar sobre as escolhas de estrutura que os autores fizeram para cada elemento do protocolo.

Uma das tarefas necessárias na primeira etapa é realizar a análise de tráfego entre um cliente e um servidor do qual estejam utilizando uma conexão TFO. O propósito desta análise é descobrir se o TFO realmente transmite os dados durante o handshake e se o servidor gera e passa o Cookie para o cliente. Para a analise dos pacotes, usamos o Wireshark que é um software muito popular nos ambientes de rede, no qual é possível identificar com detalhes os segmentos transmitidos. A próxima etapa é, a partir do estudo realizado e utilizando outros experimentos realidados e apresentados na seção 6.2, identificar o melhor cenário para se aplicar testes de performace.

Nesta etapa, são utilizados algumas ferramentas para configuração da ferramenta de testes. Precisamos selecionar uma quantidade razoável de páginas web, com diferentes objetos web, para aplicar em três cenários de atrasos(RTT) diferentes.

Na etapa seguinte, fazemos os testes de acordo com o cenário selecionado, para tal, uma das ferramentas necessárias é um emulador de redes. A configuração utilizada é a do Mininet. O propósito dos testes é avaliar os ganhos em relação ao tempo de carregamento das páginas, ou seja, essa é a métrica utilizada para validar o desempenho do protocolo em relação ao TCP convencional.

A partir disso, será feito uma análise comparativa, mostrando os resultados obtidos na terceira fase de testes, e comparando-os com a abordagem tradicional. Nessa etapa ainda, poderá se observar se outros fatores influenciam na ganha/perda de desempenho.

Por fim, será dissertado sobre os resultados obtidos, focando no comportamento do protocolo em relação à métrica escolhida que é o tempo de carregamento das páginas web. Além disso, será possível observar quais tipos de aplicações tendem a obter resultados mais significativos ao utilizar o TCP Fast Open. Por conseguinte, a análise também poderá identificar quais categorias de serviços não se observa ganhos significativos.

6 EXPERIMENTOS E RESULTADOS

Nesse capítulo vamos analisar de forma geral o protocolo TCP Fast Open, assim como testar o seu funcionamento de acordo com as especificações da RFC 7413 (CHENG et al., 2014). A primeira etapa desse capítulo trata da análise de funcionamento do protocolo de uma forma geral, utilizando um *sniffer* (analisador de pacotes) para observar o comportamento do servidor e cliente durante a comunicação. Posteriormente são selecionados os cenários para os testes de desempenho do protocolo, justificando suas escolhas. Por último, são analisados os resultados encontrados nas etapas anteriores, comparando o desempenho no tempo de carregamento das páginas selecionadas tanto para o TCP convencional quanto para o TCP Fast Open.

6.1 Análise de funcionamento

A primeira parte desse trabalho é analisar e entender o funcionamento do TFO da forma como ele foi proposto. Para isso, fez-se a leitura da RFC 7413 (CHENG et al., 2014) que define o protocolo e estudado suas especificações. Como elucidado na seção anterior, a principal característica que difere uma conexão TFO de uma conexão que utiliza o TCP convencional é o uso de um *Cookie* que é solicitado por parte do cliente e validado por parte do servidor, com o intuito de permitir a entrega segura dos dados transmitidos durante os pacotes SYN e ACK para as devidas aplicações.

A etapa de verificar o funcionamento do protocolo de acordo com suas especificações descritas na RFC 7413 (CHENG et al., 2014) é simples. Para sabermos se o protocolo atua como deveria, transmitindo dados durante o *handshake*, basta analisar os pacotes de reconhecimento (ACK) e de solicitação de abertura de conexão (SYN) enviados tanto pelo servidor como pelo cliente.

Como vimos anteriormente, na primeira conexão em que o TFO esteja habilitado por parte do servidor, não existe a troca de dados durante o *handshake*, e sim, apenas, uma solicitação de um *cookie* por parte do cliente, que é posteriormente enviado pelo servidor. Por esse motivo, também precisamos analisar se o *cookie* está sendo gerado e enviado corretamente. Para ambas essas situações, fazemos a análise de tráfego entre as partes comunicantes, a fim de confirmar o funcionamento.

6.1.1 Análise de tráfego

A análise de tráfego é um procedimento muito utilizado em redes de computadores pois, nele é possível identificar falhas e detalhes no funcionamento da rede analisada. Para esse trabalho, temos o objetivo de identificar duas situações:

- Primeiro:
  - Verificar a solicitação de um *Cookie* por parte do cliente;
  - Verificar se o servidor gera e envia o *Cookie* em resposta ao cliente;
- Segundo:
  - Verificar se há dados transmitidos do cliente para o servidor antes do fim do *handshake*;
  - Verificar se há dados transmitidos do servidor para o cliente antes do fim do *handshake*;

Dado esses dois objetivos, existem diversas maneiras de fazer essa análise. Existem diversas ferramentas que fazem a análise de tráfego assim como funções do próprio sistema operacional que podem auxiliar nessa tarefa. Em nosso trabalho, vamos utilizar o *Wireshark* para fazermos a análise dos pacotes.

O Wireshark é um sotfware de análise de tráfego, ou análise de protocolos de rede largamente utilizado. Ele foi idealizado pelo estudante de Ciência da Computação da University of Missouri - Kansas City, Gerald Combs, em meados de 1997. Sua versão 1.0 veio apenas em 2008, com um grande número de pessoas contribuindo para o projeto. Neste trabalho estamos utilizando o Wireshark Version 2.6.3 (Git v2.6.3 packaged as 2.6.3-1 ubuntu16.04.1). [3]

Como se trata de uma análise do conteúdo dos pacotes trocados entre cliente e servidor, para esta etapa do trabalho não nos interessa levar em conta valores como o RTT, ou qualquer tipo de atraso que se dá durante a comunicação. O TCP Fast Open é suportado no kernel do Linux 3.7+ (Ubuntu 14.04 LTS) e para habilitá-lo é necessário usar o comando da Figura 6.1.

O número 3 indica que o TFO deve ser habilitado tanto no lado do servidor quanto do lado do cliente. Outras configurações possíveis seriam 1 para habilitar apenas o cliente ou 2 que habilitaria essa opção apenas para o servidor. Agora, podemos usar uma simples aplicação que

[3] https://www.wireshark.org/docs/wsug$_h tml_c hunked/ChIntroHistory.html$

```
echo 3 > /proc/sys/net/ipv4/tcp_fastopen
```

Figura 6.1 – Habilitando o TFO

tenha habilitado o uso do TFO e verificarmos os pacotes transmitidos. Utilizamos um código[4] simples de um servidor e um cliente em *python* para este teste.

```
1  import socket
2
3  def listen():
4  connection = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5  connection.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
6  connection.setsockopt(socket.SOL_TCP, 23, 5)
7  connection.bind(('127.0.0.1', 12347))
8  connection.listen(10)
9  while True:
10 current_connection, adress = connection.accept()
11 while True:
12 data = current_connection.recv(2048)
13
14 if data:
15 current_connection.send(data)
16 print (data)
17 current_connection.close()
18 break
19 try:
20 listen()
21 except KeyboardInterrupt:
22 pass
```

Listing 6.1 – server.py

O código de *server.py* instância a função *listen()* que gerencia a conexão. Nessa função é criado um *socket* para a conexão, ligados pelo endereço local (locaslhost) e a porta 12347. O TCP Fast Open é habilitado na *linha 6*. O laço seguinte aceita a gerencia a conexão para receber uma mensagem do cliente conectado.

O código de *client.py* é ainda mais simples, foi necessário apenas criar um *socket* ligado ao endereço de destino (IP e porta). Da mesma forma como no servidor, precisamos habilitar o uso do TCP Fast Open na *linha 6*, após isso é enviado a mensagem para o servidor.

```
1  import socket
2
3  addr = ('127.0.0.1', 12347)
4  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5  s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
6  s.setsockopt(socket.SOL_TCP, 23, 5)
7  s.sendto("Hello!".encode(),536870912,addr)
8
9  print (s.recv(1000))
```

[4] https://superuser.blog/tcp-fast-open-python/

Listing 6.2 – client.py

Executando ambos esses códigos, podemos analisar através do Wireshark as mensagens (em forma de pacotes) trocadas entre o cliente o servidor. Configuramos o Wireshark para capturar apenas pacotes TCP com a porta 12347. Para o cenário acima, temos a seguinte captura:

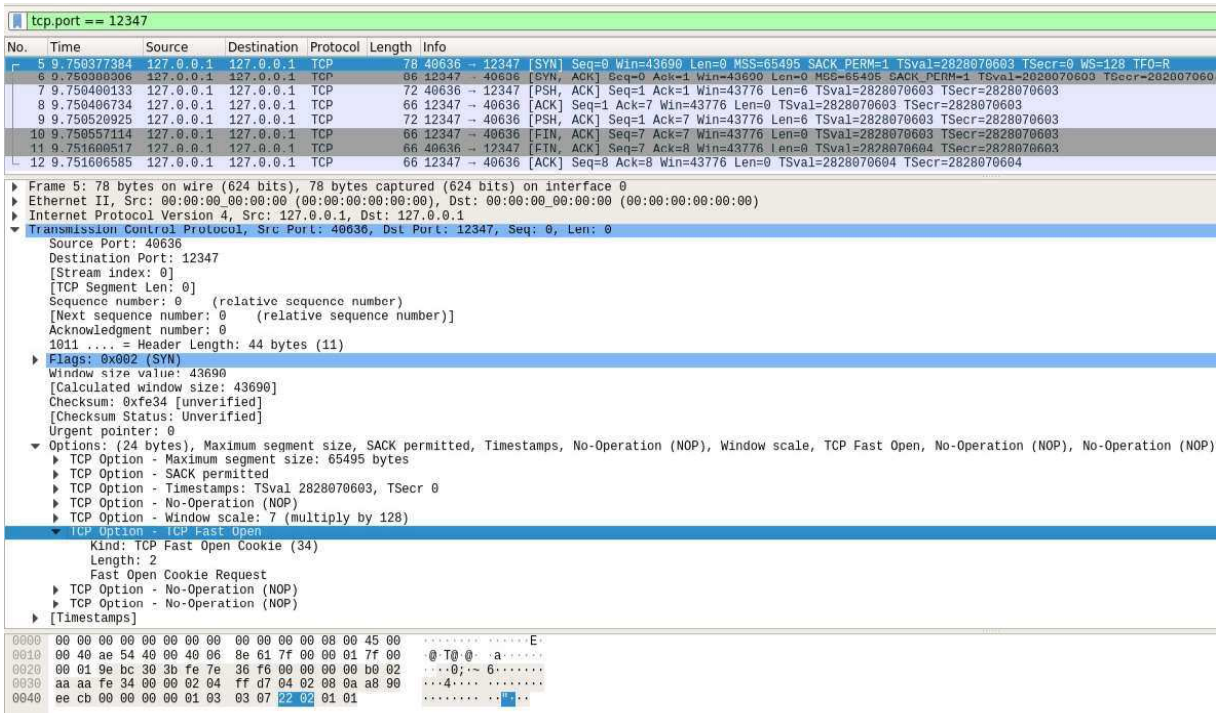

Figura 6.2 – Captura do pacote 5 durante a conexão TFO

Como observamos na figura 6.2 o pacote de número 5 é o primeiro segmento enviado do nosso programa cliente para o programa servidor. Esse pacote possui a flag SYN do cabeçalho do TCP ativada (valor 1), isso indica que este segmento se trata de uma solicitação de abertura de conexão TCP.

Ainda é possível observar na figura 6.2 que, como visto na seção anterior, por se tratar da primeira conexão utilizando o TFO, o cliente precisa solicitar um *cookie* para o servidor. Abrindo a aba que diz respeito ao protocolo TCP podemos visualizar com detalhes quais outros parâmetros estão sendo usados nessa conexão. Como observado nesta aba, o TCP utiliza o campo de opções do seu cabeçalho para solicitar um *cookie* para o TCP Fast Open, por isso esse campo do segmento tem comprimento de 2 bytes.

Agora vamos analisar a captura do pacote 6, que é o segmento enviado pelo servidor em resposta ao cliente por essa solicitação. A captura do pacote 6 é ilustrado na Figura 6.3.

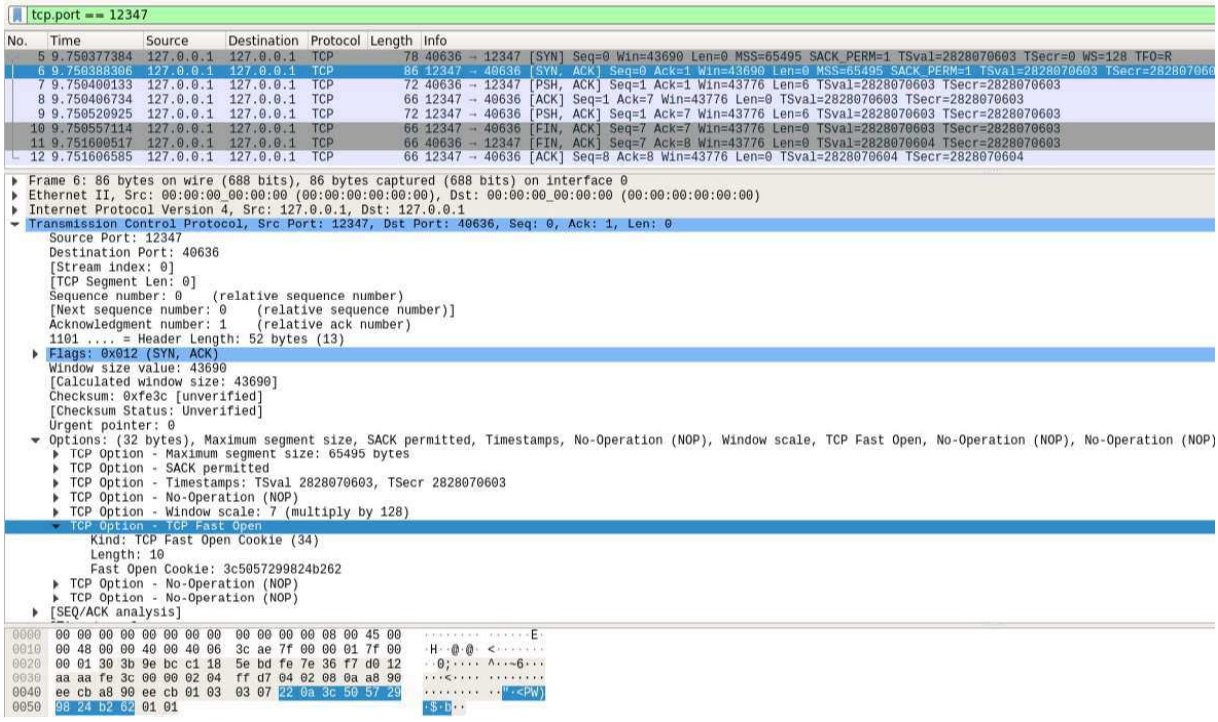

Figura 6.3 – Captura do pacote 6 (reconhecimento) durante a conexão TFO

Para o segmento de resposta (Figura 6.3), a flag SYN do cabeçalho TCP continua ativa, pois o cliente também precisará reconhecer esse segmento. Da mesma forma, a flag ACK passa também a ser ativada (valor 1), pois esse se trata de um segmento de reconhecimento. Na prática o que o servidor está dizendo é que recebeu a solicitação de abertura de conexão e que ela foi aceita.

Esse seria o procedimento normal em uma implementação com o TCP convencional. Porém, vimos na figura anterior que o cliente também fez uma solicitação por um *cookie* TFO e o servidor precisa lidar com essa solicitação. Nessa fase, o servidor então gerou um *cookie* encriptando o endereço IP do cliente sob uma chave secreta, por isso são utilizados 10 bytes para armazenar o *cookie* identificado por *Fast Open Cookie: 3c5057299824b262*.

O pacote 7 conseguinte é um segmento de reconhecimento (ACK) enviado para o servidor para reconhecer o segmento anteior, completando o *3-way handshake*. Aqui já podemos confirmar que o TCP do cliente e do servidor se comportam como deveriam, solicitando e gerando um *cookie* válido respectivamente.

Agora precisamos analisar como o cliente e servidor se comportam para conexões futuras, fazendo o uso do *cookie* para enviar dados durante o *handshake*. Para isso, deixamos o servidor rodando aguardando por novas conexões e re-executamos o código de *client.py* a fim de enviar outra mensagem para o servidor, mas desta vez utlizando o *cookie* recebido na

comunicação anteior. Nesta situação, temos a seguinte captura:

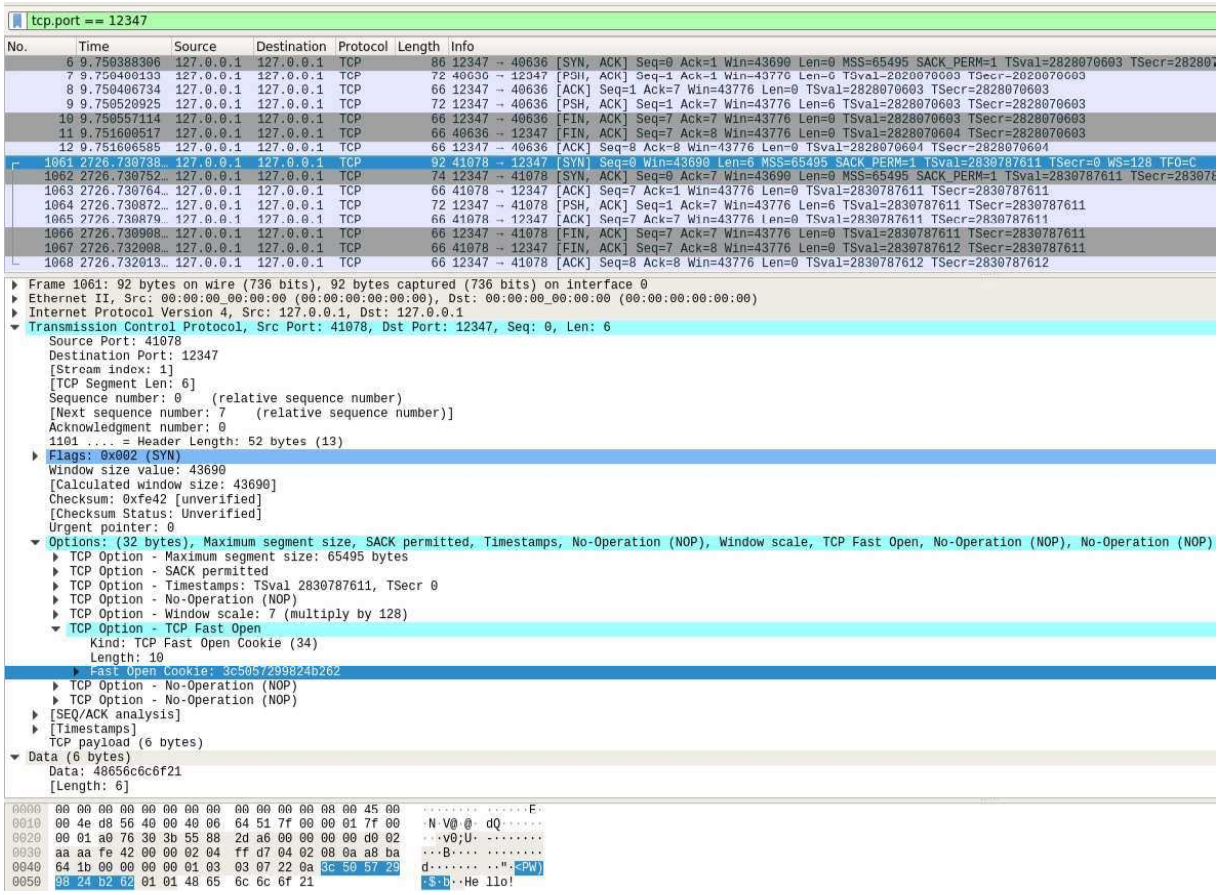

Figura 6.4 – Captura do pacote 1061 durante a conexão TFO

Na figura 6.4 podemos notar que esse segmento também é de uma solicitação de conexão TCP (flag SYN ativa). Mas desta vez o cliente já envia o *Cookie TFO* correspondente, assim como já o preenche com dados (campo *Data*) do segmento TCP. Vamos ver agora como o servidor lida com esse segmento:

Nesta última captura (Figura 6.5) vemos que o servidor reconhece a abertura de conexão (flag ACK ativa) e também reconhece os dados recebidos pelo cliente. O servidor poderia aqui também já enviar dados em resposta ao cliente como mostra o modelo da figura 4.2, mas isso irá depender do tipo da aplicação, neste exemplo o servidor não possui dados para enviar ao cliente.

6.2 Análise de desempenho

A segunda etapa do trabalho é a realização de testes de desempenho do protocolo. Estamos interessados em calcular o tempo de carregamento de páginas web comparando o desem-

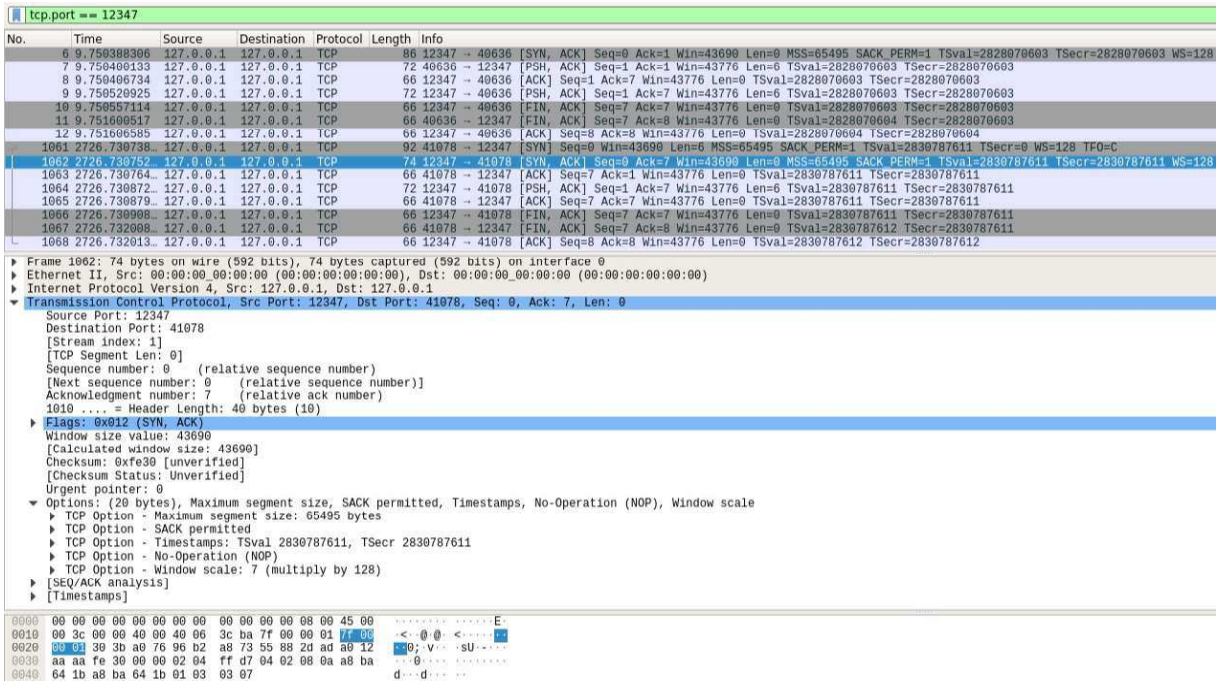

Figura 6.5 – Captura do pacote 1062 durante a conexão TFO

penho do TCP convencional com o TCP Fast Open. Para tal, utilizamos algumas ferramentas para auxiliar na captação dos recursos necessários bem como adaptamos outros experimentos para atender nossas necessidades.

Preliminarmente, precisamos selecionar quais páginas seriam utilizadas nos testes, portanto, consultamos quais os sites[5] mais visitados no ano de 2018. Também consultados quais os sites[6] brasileiros mais visitados, destes selecionamos 10 páginas além do site github.com. Os sites selecionados foram listados na Tabela 6.1.

Após a seleção das páginas web, precisamos adquirir o código-base dessas páginas assim como os objetos web que as compõem. Para cumprir essa desafio, usamos o *wget*, que é um programa que proporciona o download de dados da web. Existem vários parâmetros que podem ser usados para o download de recursos da web, no nosso caso usamos o seguinte comando de modo a conseguir os *assets* das páginas:

wget -E -H -k -K -p -e robots=off "URL"

No que diz respeito às configurações necessárias para rodar os testes, instanciamos uma VM através do Google Cloud Platform em um servidor remoto. O Google Cloud Platform é uma suíte de computação em nuvem oferecida pelo Google, funcionando com serviço em demanda,

[5] https://medium.com/@hotinsocialmedia/30-most-visited-websites-on-the-internet-in-2018-523be3aca0df
[6] https://www.similarweb.com/top-websites/brazil

Website	Categoria
baidu.com	Internet e Telecomunicações: Mecanismo de busca
facebook.com	Internet e Telecomunicações: Mídia social
github.com	Computação: Software
globo.com	Notícias e mídia
google.com	Internet e Telecomunicações: Mecanismo de busca
instagram.com	Internet e Telecomunicações: Mídia social
yahoo.com	Notícias e mídia
netflix.com	Entretenimento: TV e vídeo
youtube.com	Entretenimento: TV e vídeo
mercadolivre.com.br	Comércio
uol.com.br	Notícias e mídia

Tabela 6.1 – Sites selecionados para os testes e suas respectivas categorias

Detalhes da instância de VM	
Tipo de máquina	n1-standard-1 (1 vCPUj, 3,75 GB de memória)
OS	Ubuntu 14.04 LTS

Tabela 6.2 – Recursos da VM alocados

assim como encontramos em outros serviços, Amazon Web Services EC2 da Amazon, por exemplo. A Tabela 6.2 ilustra a configuração utilizada ao longo do experimento.

Além disso, configuramos também o *firewall* da VM para permitir o tráfego dos protocolos HTTP e HTTPS, uma vez que necessitamos usar esses protocolos ao acessar as páginas. O controle da VM, instalação de recursos e demais configurações foi feito através de SSH.

A partir de 2011, quando o protocolo foi proposto, alguns experimentos foram realizados a fim de certificar o seu funcionamento, comparando o desempenho do TCP convencional com o TFO, como encontramos em (KATTOUW; MEYERS, 2013). Desde então, novas pesquisas e experimentos foram sendo propostos ao longo dos anos, como o que encontramos em (YENDLURI; ECKERT, 2014), onde é feita uma reprodução dos resultados para os mesmos cenários do artigo original (RADHAKRISHNAN et al., 2011).

SRINIVASAN; VERMA (2015) também realizaram experimentos semelhantes aos trabalhos realizados nos anos anteriores e obtiveram um comportamente parecido com os demais. Devemos citar ainda, outras duas pesquisas na área realizada em 2016 e 2017, por TINDALL; THEIS (2016) e CHEN; ETO (2017) respectivamente, confirmando a tendência na importância de se realizar novos experimentos para essa área.

Esse trabalho usa como referência a publicação de GARRITY; STATHATOS (2014) como ferramenta para testes de desempenho. A partir de uma instância de VM em um servidor remoto, fizemos a instalação e configuração dos recursos necessários para rodar os testes, isso

inclui criação de diretórios para as páginas, extração dos objetos web das páginas (imagens, css, js, html, etc) através do *wget* e adaptação do código.

Como apontado anteriormente, consideramos importante manter algumas características realizadas nos experimentos originais, por essa razão, como encontramos em RADHAKRISHNAN et al. (2011), usamos os valores de RTT: 20ms, 100ms e 200ms. Diferentemente de GARRITY; STATHATOS (2014), a topologia de nossa rede através do Mininet foi alterada para 1Mbps para a largura de banda, como mostrado na Figura 6.6.

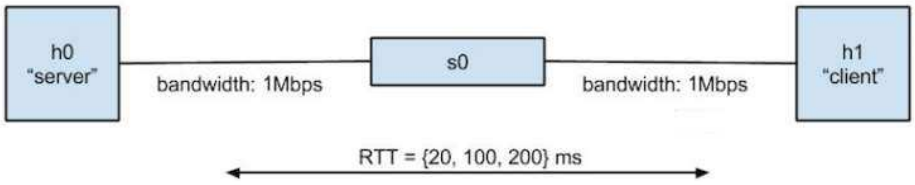

Figura 6.6 – Topologia da rede com Mininet

O Mininet é um emulador que nasceu com o objetivo de facilitar os experimentos no campo de redes de computadores, principalmente para auxiliar nas pesquisas das redes definidas por software e do OpenFlow. Ele é capaz de emular links, hosts, switches e controladores, utilizando processos que rodem em espaços de nomes da rede (*network namespaces*) e redes Ethernet virtuais. Para controlar e gerenciar todos os dispositivos emulados, o Mininet fornece uma CLI (*Command Line Interface*) conhecedora de toda a rede, ou seja, através de um único console, pode-se controlar todos os dispositivos emulados.

A partir das configurações listadas acima, adaptamos o código da ferramenta de teste para obedecer nossas necessidades, configurando os RTTs com valores 20, 100 e 200 e a largura de banda através do Mininet de 1Mbps, rodamos o experimento 10 vezes para cada website e fizemos um cálculo do tempo médio de carregamento das páginas tanto para o TCP convencional e para o TFO, obtento os resultados mostrados na Tabela 6.3.

A tabela 6.3 ilustra o comportamento de ambos os protocolos em funcionamento, sendo que para cada página são executados 3 testes com os respectivos RTTs para cada um dos protocolos. Podemos notar que com o aumento do RTT (tempo de ida e volta das mensagens) há um aumento natural também no tempo de carregamento da página. Se antes o servidor terminava o carregamento de toda a página em 3 RTTs, por exemplo, e cada RTT tem um valor maior (em tempo), isso implica que tempo de carregamento também será superior.

Podemos perceber também que, devido ao fato de que cada página possuir uma quanti-

Página	RTT(ms)	PLT: TCP (s)	PLT: TFO (s)	Melhoria (%)
baidu.com	20	0,073	0,07	4,28571
	100	0,323	0,31	4,19354
	200	0,68	0,61	11,47540
facebook.com	20	8,596	8,566	0,35022
	100	8,876	8,873	0,03381
	200	9,479	9,47	0,09503
github.com	20	1,576	1,261	24,98017
	100	1,877	1,81	3,70165
	200	4,029	2,99	34,74916
globo.com	20	6,419	6,159	4,22146
	100	7,226	7,206	0,27754
	200	7,605	7,26	4,75206
google.com	20	0,18	0,18	0
	100	0,5	0,5	0
	200	1,22	0,9	35,55555
instagram.com	20	0,831	0,526	57,98479
	100	0,86	0,859	0,11641
	200	1,473	1,4	5,21428
mercadolivre.com.br	20	5,034	5	0,68
	100	5,4	5,349	0,95344
	200	6,391	5,92	7,95608
netflix.com	20	1,122	1,122	0
	100	1,21	1,21	0
	200	2,881	2,193	31,37254
uol.com.br	20	5,202	4,377	18,84852
	100	6,067	5,78	4,96539
	200	7,638	6,36	20,09433
yahoo.com	20	4,271	4,249	0,51776
	100	5,588	4,716	18,49024
	200	5,806	5,78	0,44982
youtube.com	20	9,166	8,555	7,14202
	100	10,182	8,43	20,78291
	200	10,741	9,59	12,00208

Tabela 6.3 – Tempo médio de carregamento das páginas (PLT) com TCP Convencional e TCP Fast Open com RTTs de 20ms, 100ms e 200ms com largura de banda equivalente 1Mbps

Página	**Tamanho (*kbytes*)**
baidu.com	8
facebook.com	3136
github.com	7256
globo.com	3552
google.com	52
instagram.com	1924
mercadolivre.com.br	1348
netflix.com	1732
uol.com.br	7724
yahoo.com	2596
youtube.com	3764

Tabela 6.4 – Quantidade de *assets* por site

dade de recursos (*asstes*) diferentes, também é natural que algumas páginas demorem mais para serem carregadas. Se olharmos para os valores de PLT de baidu.com por exemplo, notamos que mesmo com o aumento do RTT a página continua sendo carregada em menos de 1 segundo. Este mesmo comportamento é notado para o google.com, sendo que o tempo de carregamento da página só é superior a 1 segundo com um RTT de 200ms e utilizando o TCP convencional. Sendo assim, a Tabela 6.4 informa a quantidade de recursos presentes em cada uma das páginas adquiridas pelo *wget*.

Através da Tabela 6.4 também podemos observar o comportamento contrário para aquelas páginas cujo a quantidade (em kbytes) de assets é maior. Citemos o globo.com, do qual mesmo como um RTT de 20ms o cliente leva cerca de 6 segundos para obter todos os recursos. O mesmo é observado no youtube.com, cujo o PLT é ainda mais significativo.

Notamos que, por ser uma variante ao protocolo TCP, o TFO mantém o mesmo padrão de comportamento no que diz respeito ao atraso RTT e o tamanho das páginas. A última coluna da Tabela 6.3 faz o comparativo no ganho do *Page Load Time*(Tempo de carregamento da página) entre TCP e o TFO. A primeira vista, notamos que o TFO obteve ganhos de desempenho na maioria dos casos de testes, variando entre 0% até 57,99%.

Seguindo as categorias da Tabela 6.4, exibimos os gráficos comparando o desempenho dos protocolos de acordo com cada categoria. A Figura 6.7 trás a comparação para as páginas de busca, por google.com e baidu.com se tratarem de páginas com poucos objetos web, 8kbytes e 52kbytes respectivamente, a diferença entre o TCP e o TFO é mais notada com um valor de RTT alto, justamente pelo fato do TFO poupar um RTT na comunicação e esse atraso poupado representar uma fatia maior de tempo.

Na Figura 6.8 temos a comparação para páginas de mídias sociais, notamos que o comportamento do protocolo TFO se mantém, tendo ganhos mais significativos para o instagram por ser um site com um número de objetos menor em relação ao facecook, reforçando a ideia de RAMACHANDRAN (2010) pela preponderância de páginas com poucos objetos web, tendo um ganho mais significativo com o TFO.

Nas Figuras 6.11, 6.9 e 6.10 encontramos outras comparações entre o TCP e o TFO mas para as caregorias de notícias, entreterimento e outros sites respectivamente. De forma geral, percebemos que o TFO apresenta ganhos significativos em relação ao PLT. Para o site do youtube por exemplo, os ganhos variaram de 7,14% até 20,78% na questão do desempenho, que podem ser significativos na satisfação dos usuários ao visitarem este site.

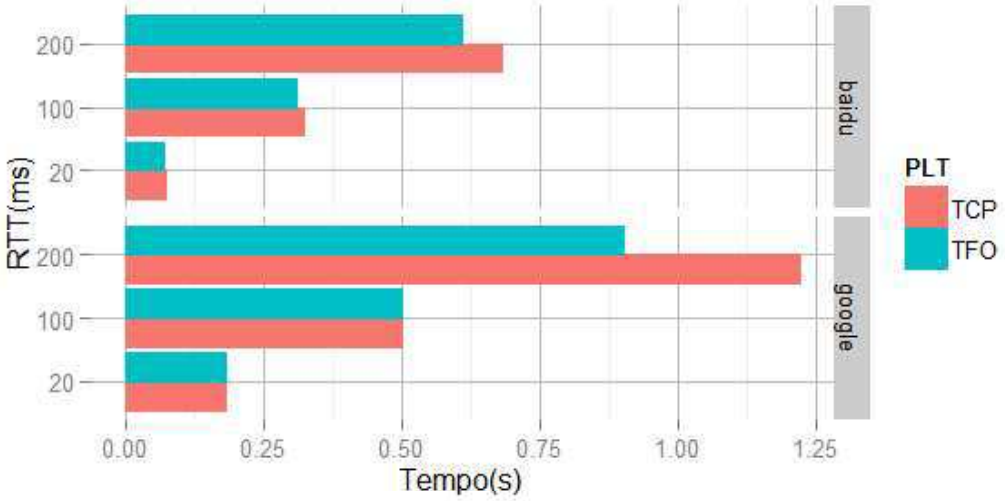

Figura 6.7 – Comparação de desempenho TCP *versus* TFO Internet em Telecomunicações: Mecanismo de busca

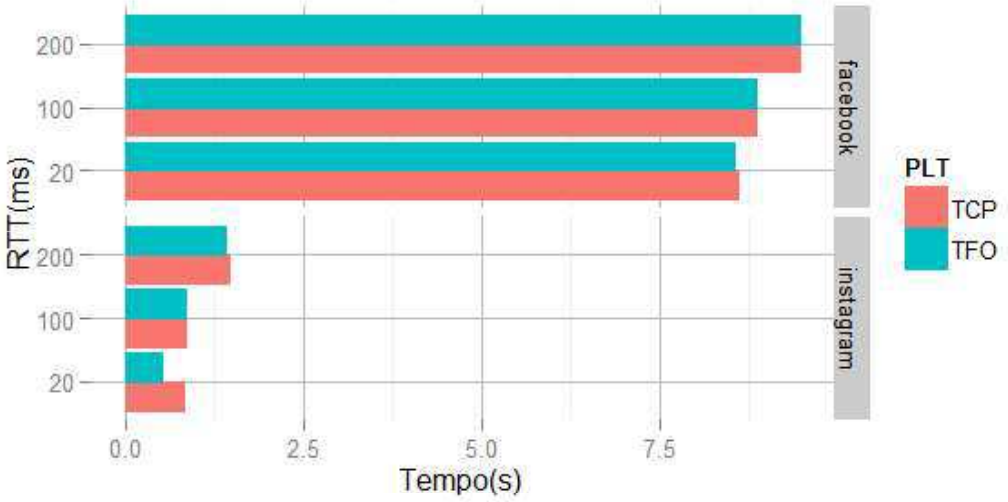

Figura 6.8 – Comparação de desempenho TCP *versus* TFO Internet em Telecomunicações: Mídia Social

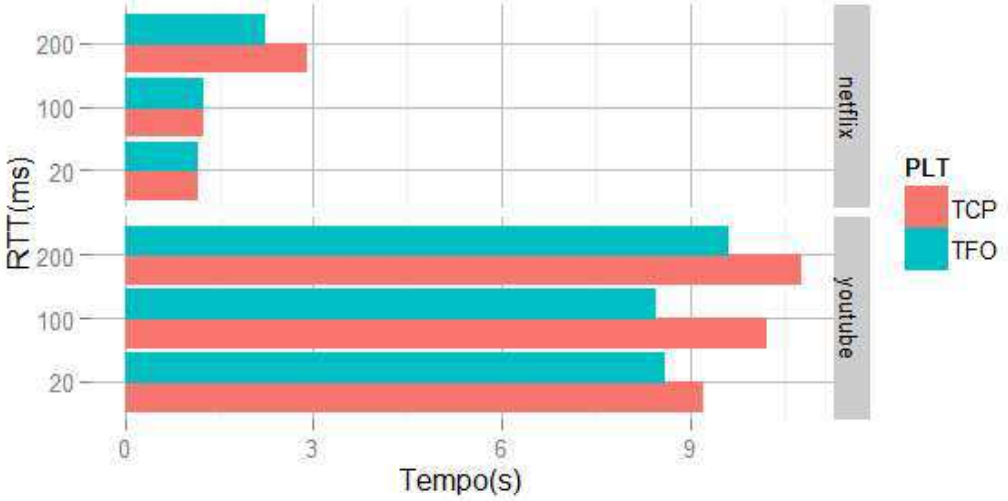

Figura 6.9 – Comparação de desempenho TCP *versus* TFO Internet em Entreterimento

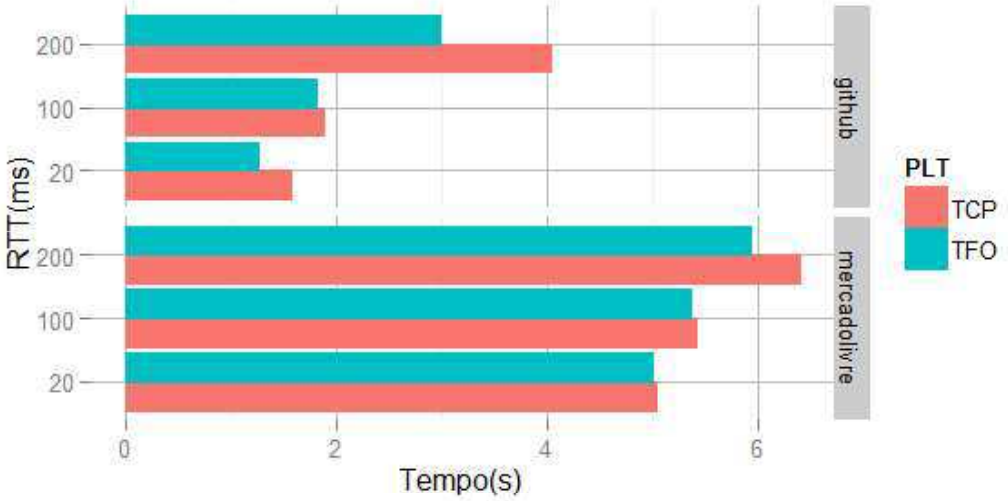

Figura 6.10 – Comparação de desempenho TCP *versus* TFO Internet em Outros sites

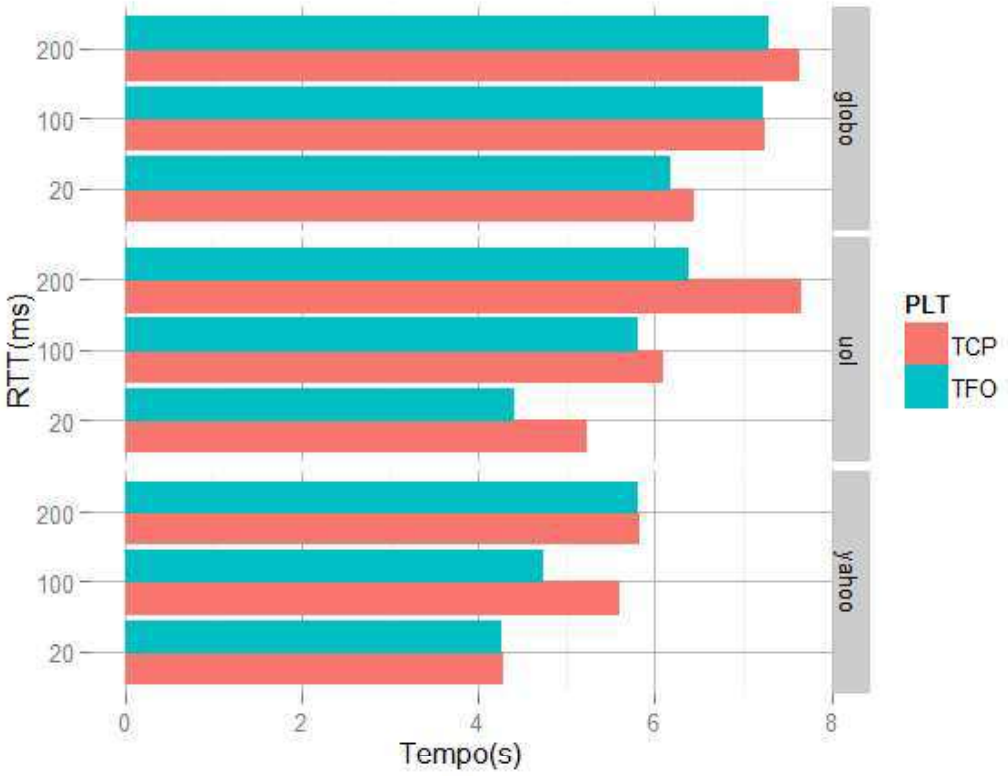

Figura 6.11 – Comparação de desempenho TCP *versus* TFO Internet em Notícias e mídia

7 CONCLUSÃO

A aplicação da internet que conhecemos como web, se popularizou por transformar drasticamente a maneira como as pessoas interagem dentro e fora de seus ambientes de trabalho (KUROSE; ROSS, 2006). Uma característica que atrai a maioria dos usuários web é o seu funcionamento por demanda. Através da web os usuários podem acessar conteúdos específicos que sejam do seu interesse de forma rápida e prática.

Quando acessamos uma página web a partir de um navegador, estamos solicitando ao servidor do domínio informado, um arquivo-base e diversos objetos web que são referenciados nesta página. O uso contínuo dos serviços prestados pela internet tornou-se habitual, escalando a quantidade de páginas e objetos que são requisitados diariamente.

O TCP é um protocolo muito utilizado nesses cenários, pois é o protocolo de transporte usado mais pelo HTTP, responsável pelo acesso das páginas web através dos navegadores. Garantir um bom funcionamento do TCP, assim como outras ferramentas que melhorem a experiência do usuário atraem cada vez mais estudos na área. O TCP Fast Open é um protocolo proposto de modo a atingir esses objetivos, melhorando a experiência do usuário com a diminuição no tempo de carregamento das páginas.

Esse trabalho propôs um estudo aprofundado deste protocolo, com o objetivo de averiguar o seu funcionamento e testar seu desempenho. A primeira etapa foi realizada através da análise de tráfego de um programa cliente e um programa servidor, onde pudemos observar a atuação do TFO. Verificamos que o protocolo atua conforme as especificações de CHENG et al. (2014), criando e validando um *cookie* para conexões futuras e enviando dados durante o *handshake*.

O desempenho do protocolo foi testado e seus resultados foram apresentados na Figura 6.3 onde atentamos que o TFO realmente obteve resultados significativos na redução do tempo de carregamento das páginas para praticamente todas as páginas testadas, corroborando com os resultados encontrados por RADHAKRISHNAN et al. (2011).

Apresentamos, finalmente, gráficos que ilustram o comportamento da Figura 6.3 trazendo de forma intuitiva os ganhos para cada categoria. Por esta razão, entendemos que os resultados encontrados neste trabalho captam de forma similar os resultados encontrados em outros experimentos, como citamos na seção 6.2.

Como podemos notar na etapa de análise de desempenho, o valor do atraso de comu-

nicação (RTT) variou entre 20ms, 100ms e 200ms, como encontrado em RADHAKRISHNAN et al. (2011), e é um importante fator que simula situações de atraso que podem acontecer em conexões sem o ambiente controlado. Esses atrasos podem ser atribuídos a questões geográficas (distância entre *host* e cliente), atrasos provocados por congestionamento da rede, entre outros.

O atraso RTT se torna um fator muito importante na experiência final do usuário pois é diretamente responsável pelo aumento no tempo de carregamento de páginas web, como percebemos através das Figuras 6.7, 6.8. 6.9, 6.10 e 6.11 sempre que houve aumento no valor do RTT, também houve aumento no valor final do *Page Load Time (PLT)*.

Infelizmente existem fatores que influenciam no valor final do RTT, mas que ficam limitados pelas soluções de tecnologias conhecidas na época, como a distância física entre as partes comunicantes. Por essa razão, o TCP Fast Open se foca em soluções que podem ser aplicadas independentes dessas limitações.

Um comportamento esperado também nesta etapa é observado por RADHAKRISHNAN et al. (2011). Observando seus resultados atentamos que sempre que houve um aumento no valor de RTT, consequentemente, houve aumento na porcentagem de ganho de desempenho do TFO em relação ao protocolo convencional. Esse comportamento faz todo sentido, pois como o tempo para transferência de todo conteúdo da página se torna maior, o RTT que é poupado na utilização do TFO representa uma fatia de tempo maior no total da comunicação.

Percebemos também que esse comportamento em específico não é, necessariamente, observado em todos os casos de testes apresentados por nosso trabalho. Adicionalmente, observamos comportamento semelhante ao nosso trabalho como o encontrado por YENDLURI; ECKERT (2014) onde em alguns casos o ganho com um RTT menor foi mais significativo em relação a um RTT mais alto.

Esse cenário foi ilustrado na Figura 6.3, observamos que o site instagram.com apresentou um ganho de 57,98% para um RTT de 20ms, e um ganho de apenas 5,21% para um RTT de 200ms. Vale ponderar que esse tipo de resultado pode acontecer, pois existem outros fatores que influenciam na comunicação, podemos atribuir a configuração da própria topologia da rede utilizada, que é diferente em cada um dos experimentos. Mesmo assim, na generalidade das situações, quando houver um aumento no tempo do atraso RTT também haverá um ganho mais significativo utilizando o TFO, comportamento este que também foi destacado por RADHAKRISHNAN et al. (2011).

Considerando que uma conexão TFO também é, necessariamente, uma conexão TCP e,

ao mesmo tempo é considerado um *upgrade* (melhoria) ou extensão do TCP. Dito isso, podemos observar que em nenhum caso dos experimentos realizados houve um tempo maior no PLT do TFO comparado ao TCP e que, no máximo, o TFO não apresentou nenhum ganho em alguns dos cenários testados. Podemos visualizar um exemplo dessa situação na Figura 6.3 onde para o site google.com, nos RTTs de valor 20ms e 100ms, o TFO teve o mesmo desempenho que o TCP.

Achamos importante acentuar que, assim como RADHAKRISHNAN et al. (2011), os testes de desempenho dos protocolos foram realizados somente em ambientes controlados, ou seja, não foram simuladas situações onde não existe a perda dos pacotes de reconhecimento e/ou que os *middleboxes* e *firewalls* estejam bloqueando, ou mesmo derrubando os pacotes.

O estudo de melhorias como o TCP Fast Open é muito importante antes que seu uso seja amplamente difundido. Para trabalhos futuros nessa área seria relevante a realização de experimentos considerando cenários reais. Assim seria possível identificar falhas ou situações que de outro modo não seriam notadas em ambientes totalmente simulados. Nosso trabalho em conjunto com o de outros autores como o de RADHAKRISHNAN et al. (2011) buscar contribuir nesse sentido ao apresentar dados relevantes que possam elevar o status do protocolo TFO para que se torne um padrão amplamente utilizado na internet.

REFERÊNCIAS

ALLMAN, M.; FLOYD, S.; PARTRIDGE, C. RFC 3390: increasing tcp's initial window. **Internet Eng. Task Force (IETF)-Request for Comments**, [S.l.], 2002.

ALLMAN, M.; PAXSON, V.; BLANTON, E. **TCP congestion control**. [S.l.: s.n.], 2009.

BERNERS-LEE, T.; FIELDING, R.; FRYSTYK, H. RFC 1945: hypertext transfer protocol—http/1.0, may 1996. **Status: INFORMATIONAL**, [S.l.], v.61, 2005.

BRADNER, S. **RFC 2026**: the internet standards process. [S.l.]: IETF Internet Draft, sec. 10, 1996.

CHEN, J.; ETO, N. CS244 '17 TCP FAST OPEN. **Online at https://reproducingnetworkresearch.wordpress.com/2017/06/05/cs244-17-tcp-fast-open/**, [S.l.], 2017.

CHENG, Y. et al. **Tcp fast open**. [S.l.: s.n.], 2014.

COMER, D. E. **Redes de Computadores E Internet 4 Ed.** [S.l.]: Bookman, 2007.

FIELDING, R. et al. RFC 2616. **Hypertext Transfer Protocol–HTTP/1.1**, [S.l.], v.2, n.1, p.2–2, 1999.

FLESHGRINDER. Illustration of TCP's Congestion Avoidance based on this image. **Online at https://reproducingnetworkresearch.wordpress.com/2016/05/30/cs244-16-tcp-fast-open/**, [S.l.], oct 2014.

GARRITY, L.; STATHATOS, S. CS244 '14: tcp fast open. **Online at https://reproducingnetworkresearch.wordpress.com/2014/06/03/cs244-14-tcp-fast-open-2/**, [S.l.], 2014.

KATTOUW, R.; MEYERS, M. CS244 '13: tcp fast open. **Online at https://reproducingnetworkresearch.wordpress.com/2013/03/12/cs244-13-tcp-fast-open/**, [S.l.], 2013.

KUROSE, J. F.; ROSS, K. W. Redes de Computadores e a Internet. **São Paulo: Person**, [S.l.], p.28, 2006.

POSTEL, J. et al. **Transmission control protocol RFC 793**. [S.l.]: September, 1981.

RADHAKRISHNAN, S. et al. TCP fast open. In: SEVENTH CONFERENCE ON EMERGING NETWORKING EXPERIMENTS AND TECHNOLOGIES. **Proceedings...** [S.l.: s.n.], 2011. p.21.

RAMACHANDRAN, S. Web metrics: size and number of resources. **Online at https://developers. google. com/speed/articles/web-metrics**, [S.l.], 2010.

SRINIVASAN, S.; VERMA, R. CS244'15- TCP FAST OPEN. **Online at https://reproducingnetworkresearch.wordpress.com/2015/05/31/cs24415-tcp-fast-open/**, [S.l.], 2015.

TANENBAUM, A. S. **Redes de computadoras**. [S.l.]: Pearson Educación, 2003.

TINDALL, N.; THEIS, E. CS244 '16: tcp fast open. **Online at https://reproducingnetworkresearch.wordpress.com/2016/05/30/cs244-16-tcp-fast-open/**, [S.l.], 2016.

TOUCH, J.; HEIDEMANN, J.; OBRACZKA, K. Analysis of HTTP performance. **URL: http://www. isi. edu/touch/pubs/http-perf96/>, ISI Research Report ISI/RR-98-463,(original report dated Aug. 1996), USC/Information Sciences Institute**, [S.l.], 1998.

YENDLURI, V.; ECKERT, A. CS244 '14: tcp fast open. **Online at https://reproducingnetworkresearch.wordpress.com/2014/06/03/cs244-14-tcp-fast-open/**, [S.l.], 2014.