Pós-Graduação em Ciência da Computação

Assis Tiago de Oliveira Filho

UMA ANÁLISE EXPERIMENTAL DE DESEMPENHO DO PROTOCOLO QUIC

Universidade Federal de Pernambuco
posgraduação@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

Recife
2020

Assis Tiago de Oliveira Filho

UMA ANÁLISE EXPERIMENTAL DE DESEMPENHO DO PROTOCOLO QUIC

Este trabalho foi apresentado à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade de Pernambuco como requisito parcial para obtenção do grau de Mestre Profissional em Ciência da Computação.

Área de Concentração: Redes de Computadores.

Orientador: Prof. Dr**.** Djamel Fawzi Hadj Sadok.

Recife
2020

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

O48a Oliveira Filho, Assis Tiago de
Uma análise experimental de desempenho do protocolo QUIC / Assis Tiago de Oliveira Filho. – 2020.
101 f.: il., fig., tab.

Orientador: Djamel Fawzi Hadj Sadok.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2020.
Inclui referências e apêndices.

1. Redes de computadores. 2. Análise de desempenho. I. Sadok, Djamel Fawzi Hadj (orientador). II. Título.

004.6 CDD (23. ed.) UFPE - CCEN 2020 - 75

Assis Tiago de Oliveira Filho

UMA ANÁLISE EXPERIMENTAL DE DESEMPENHO DO PROTOCOLO QUIC

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 10 de fevereiro de 2020.

BANCA EXAMINADORA

_____

Prof. Dr. Nelson Souto Rosa
Centro de Informática / UFPE

_____

Prof. Dr. Glauco Estácio Gonçalves
Departamento de Estatística e Informática/UFRPE

_____

Prof. Dr. Djamel Fawzi Hadj Sadok
Centro de Informática / UFPE
(**Orientador**)

Dedico este trabalho a toda minha família e amigos, especialmente aos meus pais, Assis e Sônia, à minha irmã Elaine, e à minha namorada, Lorraine.

AGRADECIMENTOS

Primeiramente, agradeço a toda minha família, e destaco, principalmente, os meus pais, Assis Tiago de Oliveira e Sônia Oliveira da Silva, minha irmã Elaine Oliveira, e minha namorada Lorraine Morais por sempre incentivarem-me a correr atrás dos meus objetivos, obrigado pelo imenso apoio.

Agradeço a todos os meus professores, que contribuíram imensamente para minha formação acadêmica e influenciaram na construção de meu caráter.

Reconheço a Universidade Federal de Pernambuco por propiciar conhecimento cientifico e humano por meio dos docentes e infraestrutura.

Agradeço também aos grandes amigos e companheiros que pude conhecer e adquirir na Universidade, muito obrigado pelo convívio e compartilhamento de conhecimentos, tornando a experiência acadêmica mais prazerosa.

Por fim, porém não menos importante, não posso deixar de agradecer também a toda equipe que compõe o Grupo de Pesquisa em Redes e Telecomunicações (GPRT-UFPE) que foram fundamentais para consolidação dos meus conhecimentos. E, em especial aos professores Djamel Sadok, que também tem seu peso por ter me orientado durante todo o período do mestrado, e Judith Kelner por terem me incentivado a iniciar minha carreira como pesquisador.

Sou privilegiado por ser rodeado de pessoas maravilhosas.

RESUMO

O protocolo *Quick UDP Internet Connection* (QUIC) foi apresentado pelo Google inicialmente em 2012 como uma solução para alguns dos problemas estruturais da Internet evidenciados principalmente pelas limitações do protocolo *Transmission Control Protocol* (TCP) frente às novas necessidades das aplicações e serviços Web. Diante desse cenário, mesmo que esse protocolo venha sendo considerada progressivamente favorável, algumas questões ainda estão em aberto, como por exemplo à avaliação do seu efetivo ganho em relação ao modelo já estabelecido (TCP + HTTP/2) utilizado na atualidade para navegação Web. Sendo assim, este trabalho utilizou o método de pesquisa considerando o estudo de caso. Ficando ao fim deste evidenciado, através da análise dos resultados que o QUIC apresenta melhor desempenho em relação ao TCP apenas casos específicos, como por exemplo em redes com grande quantidade de atraso e perda de pacotes. Possibilitando, portanto, a visualização, de forma evidente, que a proposta do QUIC ainda não é suficientemente robusta e que necessita se desenvolver para ser efetivamente aplicada.

Palavras-chave: QUIC. TCP. Protocolos de transporte. Análise de Desempenho.

ABSTRACT

The Quick UDP Internet Connection (QUIC) protocol was initially presented by Google in 2012 as a solution to some of the structural problems of the Internet, evidenced mainly by the limitations of the Transmission Control Protocol (TCP) in view of the new needs of Web applications and services. In view of this scenario, even if the protocol is considered progressively favorable, some questions are still open, such as, for example, evaluation of its effective gain in relation to the already established model (TCP + HTTP / 2) currently used for Web browsing. Therefore, this work used the research method considering the case study. At the end of this, through the analysis of the results that the QUIC presents better performance in relation to TCP only specific cases, as for example in networks with a large amount of delay and packet loss. Therefore, allowing the visualization, evidently, that the QUIC proposal is not yet robust enough and that it needs to develop in order to be effectively applied.

Keywords: QUIC. TCP. Transport Protocols. Performance Analysis. Internet.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

ABNT	Associação Brasileira de Normas Técnicas
BBR	*Bottleneck Bandwidth and Round-trip propagation time*
CSS	*Cascading Style Sheets*
FEC	*Forward Erasure Correction*
GIF	*Graphics Interchange Format*
HOL	*Head-of-line blocking*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IEEE	*Institute of Electrical and Electronics Engineers*
IETF	*Internet Engineering Task Force*
IP	*Internet Protocol*
JPEG	*Joint Photographics Experts Group*
MTU	*Maximum Transfer Unit*
OSI	*Open Systems Interconnection*
QUIC	*Quick UDP Internet Connections*
PNG	*Portable Network Graphics*
RFC	*Request for Comments*
RLC	*Sliding Window Random Linear Code*
RTT	*Round Trip Time*
SPDY	Abreviatura da palavra speedy
TCP	*Transmission Control Protocol*
TLS	*Transport Layer Security*
UDP	*User Datagram Protocol*
WWW	*World Wide Web*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo descreve as principais motivações para realização deste trabalho, incluindo sua justificativa, questão da pesquisa, objetivos de pesquisa gerais e específicos desejados e, também apresenta como está estruturado o restante da dissertação.

1.1 JUSTIFICATIVA

No meio social, grande parte das atividades exercidas pelas pessoas passam de alguma forma por uma ou diversas redes de computadores. Sendo perceptível o fato de que a tecnologia de comunicação transcende todos os níveis sociais, e que ela tem como seu principal expoente a Internet.

Com a ampla difusão da Internet e de seus serviços, determinada principalmente pela propiciação de inúmeras facilidades para a sociedade, grande parte dessas comodidades tornaram-se essenciais e até mesmo imprescindíveis.

Em função dessa grande popularização, progressivamente, serviços e aplicações estão se tornando mais complexos, exigindo bem mais da infraestrutura tradicional de protocolos pertencentes ao modelo *Transmission Control Protocol*[1]*/Internet Protocol*[2] *(*TCP/IP), e consequentemente evidenciando a necessidade de melhorias deste. A navegação Web, que é uma das principais atividades possibilitadas pela Internet, é afetada criticamente.

Um argumento em favor desta afirmação é que as páginas Web da atualidade são significativamente mais complexas e distintas do que aquelas encontradas nos primeiros anos da Web. Por exemplo, o tamanho das páginas da Internet aumentou cerca de cinco vezes entre 2016 e 2020 [1].

Dessa forma, propiciar que a Internet se desenvolva, de forma a facilitar cada vez mais o dia a dia das pessoas, tornou-se tema recorrente entre os pesquisadores. Exemplo a isso foi que, a fim de garantir o progresso da Internet, foram elaboradas

[1] Protocolo de Controle de Transmissão.
[2] Protocolo de Internet.

diversas propostas buscando a melhoria do contexto atual. De modo que, um dos projetos de tecnologia, no campo de redes de computadores, é o protocolo *Quick UDP Internet Connections*[3] *(QUIC)*, desenvolvido pelo Google em 2012.

Desde do seu surgimento, o protocolo QUIC vem sendo proposto como alternativa para modificar o modelo convencional de redes obtendo destaque através de alguns estudos [2] [3] [4] [5], e portanto tem sido empregado progressivamente no cenário de redes, principalmente, pelo Google [6].

Diante do expoto, este trabalho propõe uma abordagem prática de implementação e avaliação do protocolo QUIC, com auxílio de outras tecnologias, objetivando transparecer questões relacionadas a avaliação de desempenho do mesmo em diferentes cenários.

1.2 PROBLEMA E QUESTÃO DE PESQUISA

Embora, a proposta do protocolo QUIC tenha apresentado uma melhoria significativa, principalmente pelo estudo mais relevante realizado em 2015 pelo Google [2], ainda é indispensável examinar o comportamento do QUIC com mais profundidade e clareza, visto que diversas informações fundamentais para validação de qualquer estudo, como a configuração dos elementos, ou mesmo os parâmetros da rede utilizados, não foram apresentadas.

Além disso, os outros poucos trabalhos existentes sobre a tecnologia às vezes ofecerem resultados conflitantes ou mesmo incompletos, dificultando o esclarecimento sobre os reais benefícios e limitações do QUIC. Um exemplo dessa conflutância seria nos trabalhos de Carlucci [3] e Magyesi [7], que divergem no desempenho do QUIC com a métrica de Tempo de carregamento da páginas.

É importante ressaltar, que no presente, redes de computadores, através do seu principal expoente, a Internet, tornou-se parte essencial da infraestrutura social, assim como o sistema de distruiição da energia elétrica [8]. E, diante desse cenário, o QUIC propõe a substituição de um importante protocolo que viabiliza a tecnologia de comunicação [1], o TCP, fazendo-se importante a validação dessa proposta.

É essencial o desenvolvimento de estudos que buscam analisar o comportamento dessa proposta por meio de experimentos acessíveis, buscando

[3] Conexões rápidas de Internet à UDP.

esclarecer, e, eventualmente enaltecer algum comportamento não esperado, evitando por exemplo outro colapso dessa ferramenta, como o que aconteceu nos primórdios da Internet [9].

Dessa forma, despertou-se a necessidade de elaborar a presente dissertação, abordando o seguinte objetivo geral:

Avaliar o desempenho do protocolo QUIC através do desenvolvimento de um experimento consistente, buscando esclarecer e identificar aspectos comportamentais ainda não sabidos da proposta.

Além disso, essa dissertação também abrangeu os objetivos específicos de comparar a avaliação de desempenho do QUIC com o protocolo TCP, verificar o impacto dos algoritmos de controle de congestionamento Cubic e BBR no QUIC, e avaliar o compartilhamento de recursos da rede com os algoritmos de congestionamento com o QUIC e TCP.

1.3 ESTRUTURA DA DISSERTAÇÃO

A estrutura da dissertação é formada a partir da apresentação da instrodução, seguida de sua contextualização por meio da fundamentação teórica e trabalhos relacionados, e aprofundamento com o estudo de caso, finalizando-a com a conclusão, onde pude sintetizar tudo o que abordei no decorrer do trabalho, bem como tecer algumas considerações finais, além de propor sugestões de estudos futuros. Assim, o presente trabalho de dissertação estruturou-se em seis capítulos, como se segue, observe:

- **Capítulo 1 – Introdução:** O primeiro capítulo consiste da introdução e sua temática, composto ainda da questão problema, objetivos da pesquisa, justificativa, metodologia, e este item de estrutura da dissertação.
- **Capítulo 2 – Fundamentação Teórica:** O segundo capítulo retrata a fundamentação teórica. Este refere-se à uma breve exposição histórica sobre redes de computadores, uma formalidade sobre protocolo QUIC, além de explanar sobre elementos relacionados com este trabalho, como os algoritmos de controle de

congestionamento *Bottleneck Bandwidth and Round-trip propagation time*[4] (BBR*), Cubic*, além de outros.

- **Capítulo 3 – Trabalhos Relacionados:** O terceiro capítulo apresenta uma revisão do estado da arte sobre o tema principal, considerando artigos e dissertações que foram examinados na pesquisa.
- **Capítulo 4 – Método de Pesquisa:** O quarto capítulo detalha informações referentes ao estudo de caso desenvolvido e a metodologia utilizada.
- **Capítulo 5 – Análise dos Resultados**: No quinto capítulo são descritos os aspectos relevantes constatados no estudo de caso.
- **Capítulo 6 – Conclusão e Trabalhos Futuros:** Neste expõe-se as conclusões dos tópicos estudados, e apresentação de considerações finais, seguido de sugestões de estudos futuros.

[4] Gargalo de largura de banda e tempo de propagação de ida e volta.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta e teórica sobre Redes de Computadores, assim como alguns conceitos relacionados com o tema principal do trabalho, o Protocolo QUIC, evidenciando aspectos julgados relevantes, além de demonstrar os elementos que compõem e normatizam essa proposta.

2.1 MODELO TCP/IP

Nesta seção será apresentada uma descrição sobre do modelo TCP/IP e alguns dos protocolos julgados relevantes para a plena compreensão da pesquisa e do tema principal deste trabalho, o protocolo QUIC.

O TCP/IP é um agrupamento de protocolos de comunicação hierárquicos e relativamente independentes, geralmente utilizados de maneira conjunta, alocados em camadas interativas e com seus respectivos propósitos bem delimatados [10]. O conjunto de protocolos TCP/IP pode ser definido, tendo cinco camadas (Figura 1): física, link de dados, rede, transporte e aplicação. É importante frisar que não existe um consenso universal entre os autores e também na academia em relação a descrição do modelo TCP/IP.

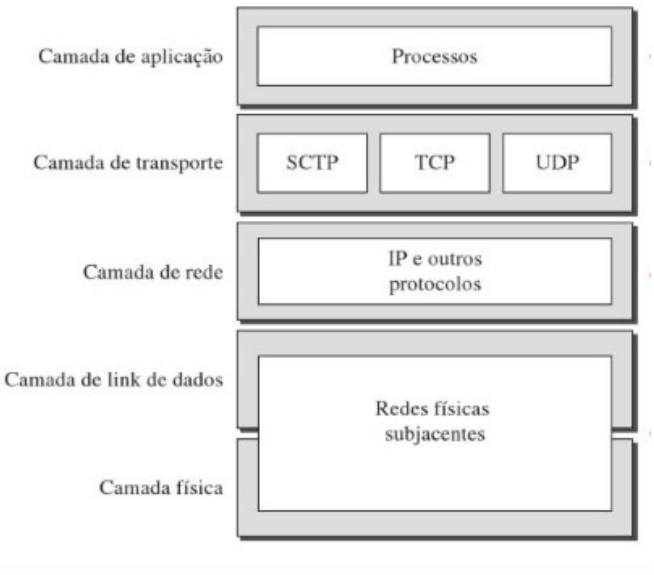

Figura 1 – Modelo TCP/IP [10].

Dois protocolos bastante relevantes neste modelo de referência são o *Internet Protocol (IP) e Transmission Control Protocol* (TCP), que podem ser esclarecidos da seguinte maneira:

Pretendendo construir o conceito teórico da dissertação com mais clareza e objetividade, foi adotado a contextualização das tecnologias relacionadas bem como do protocolo QUIC utilizando a versão IPv4.

2.1.1 Internet Protocol

O *Internet Protocol*[5] (IP) é a proposta utilizada para estabelecer normas e formatos de endereços e encaminhamento de dados possibilitando a interconexão de quaisquer dispositivos. O IP pertence à terceira camada do modelo TCP/IP e inicialmente foi definido no RFC 791 [11].

No IP, os dados são denominados datagramas, onde um datagrama IP é formado por um cabeçalho IP, que indica endereços IP de origem e do destino, além de outros campos, como se observa na Figura 2. O cabeçalho IP pode ter comprimento total de até 20 bytes e é formado por: *versão, IHL, Tipo de Serviço, Tamanho Total, Identificação, Flags, Fragment Offset, Tempo de sobrevida, Protocolo, Checksum, Endereço de Origem, Endereço de Destino, Opções e Padding.*

[5] Protocolo de Internet.

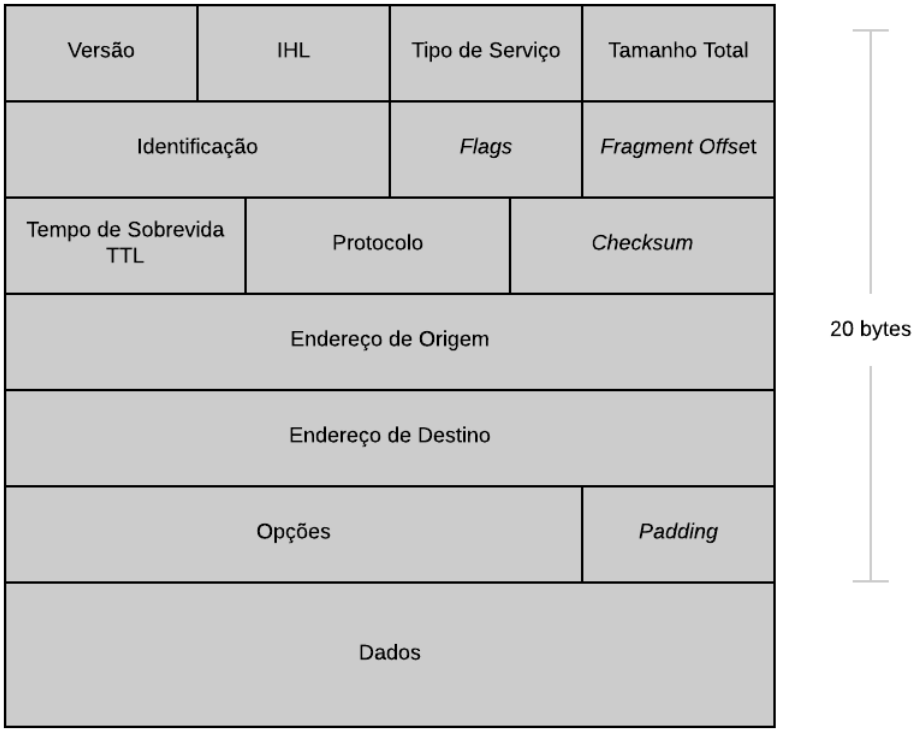

Figura 2 – Datagrama Ip.

O tamanho total do datagrama pode ser definido até 64Kbytes, efetivamente não é maior que 1.500 bytes visando evitar fragmentação do datagrama na rede física.

O IP não dispõe de nenhum mecanismo verificador para tratamento de erros em relação ao envio desses datagramas, possuindo apenas um serviço não confiável, sem garantias, de transmissão. Seu funcionamento segue o modelo de serviço de melhor esforço (*best-effort*[6]), delimitando seu propósito apenas em oferecer a transmissão de mensagens até seu destino final da maneira mais adequada possível, com por exemplo escolhendo o menor caminho [10].

2.1.2 Transmission Control Protocol

O *Transmission Control Protocol*[7] (TCP) é um dos mais importantes protocolos do modelo TCP/IP, pertencendo à quarta camada deste modelo. Foi definido originalmente no RFC 793 em 1981 [12]. Até hoje, o TCP continua sendo adaptado e

[6] Melhor esforço possível.
[7] Protocolo de Controle de Transmissão.

alterado para novos cenários como mobilidade, comunicações sem fio, comunicação via satélite, *high delay bandwidth product* [8] , comunicação multi-ponto, etc.

Diferente do protocolo IP, que viabiliza a comunicação entre *hosts* arbitrários da melhor forma possível. O TCP tem a finalidade de garantir que as mensagens dos processos arbitrários de diferentes máquinas que utilizem o protocolo se comuniquem entre si independente de características da rede e do meio físico.

É através do TCP, que aspectos como confiabilidade e disponibilidade podem ser alcançados nas comunicações entre os diferentes dispositivos da rede. Essas características tornam o protocolo uma das ferramentas mais utilizadas [1].

No TCP, a transmissão de dados é fragmentada e formatada em unidades menores chamada de segmentos, formados por cabeçalho, que podem ter até 20 bytes de tamanho, e dados. Estes segmentos viabilizam o sequenciamento das informações, possibilitando uma eventual reordenação na receptação das informações [10]. A Figura 3 ilustra como é feita a alocação de um segmento TCP em um datagrama IP.

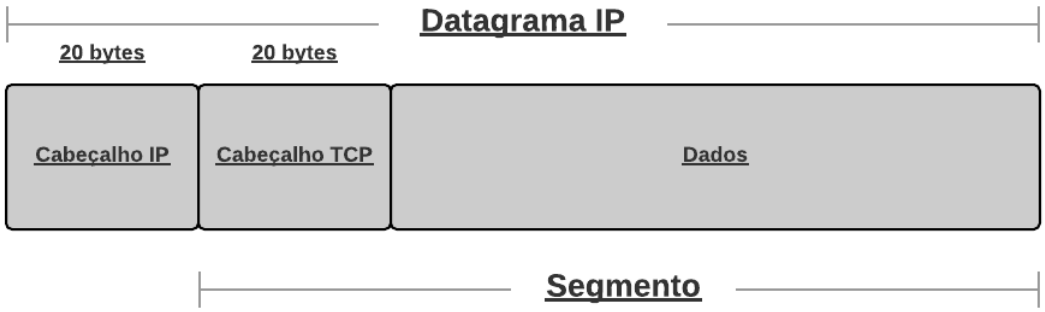

Figura 3 – Seguimento TCP

Os campos de cabeçalhos são formados por Porta *de Origem, Porta de Destino, Número de Sequência, Número de Confirmação, Comprimento do Cabeçalho,* Campo *Reservado, Flags (NS, CW, ECE, URG, ACK, PSH, RST, SYN, FIN), Checksum, Ponteiro Urgente e Opções*. É possível observar uma ilustração na Figura 4, assim como também uma descrição [10] [13] concisa de cada campo.

[8] Aplicações com alta vazão.

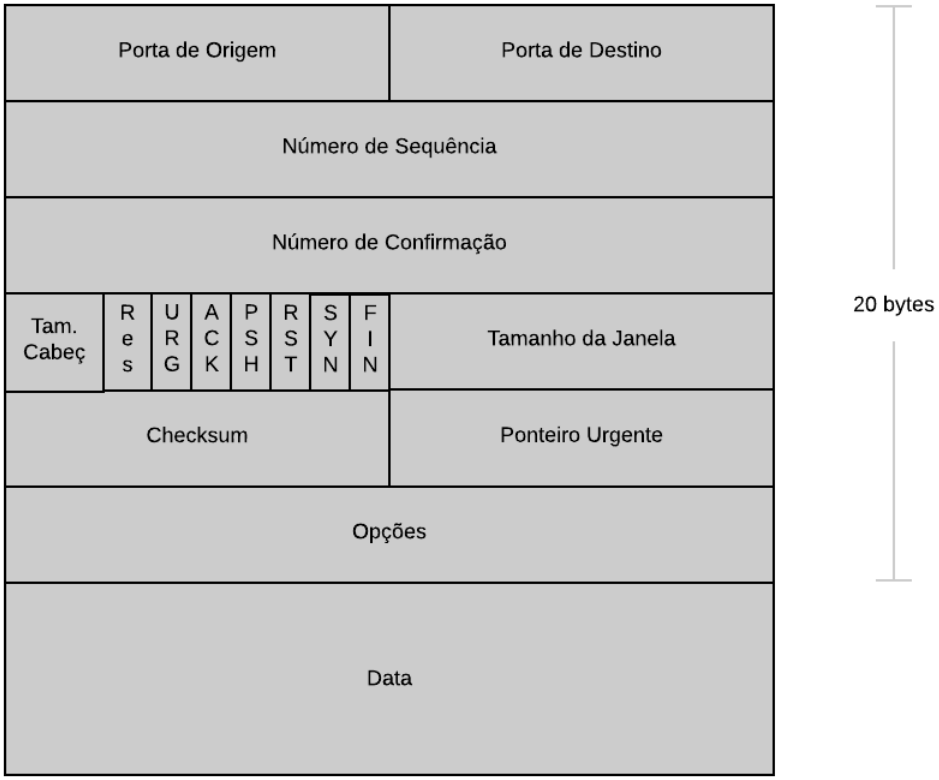

Figura 4 – Campos do Cabeçalhos TCP.

- **Porta de Origem**: Indica o processo (aplicação) que enviou o respectivo dado.
- **Porta de Destino**: Indica o software ou processo que vai receber o respectivo dado.
- **Número de Sequência**: Indica o número de identificação único de um segmento possibilitando identificar se esse segmento foi produto de uma re-transmissão.
- **Número de Confirmação**: Utilizado em conjunto com a flag ACK. Caso a flag esteja ativada, indicará o valor do próximo número da sequência que o destinatário da mensagem deseja receber, propiciando dessa maneira a confirmação de entrega de mensagens.
- **Tamanho do Cabeçalho**: Indica o tamanho do cabeçalho do TCP.
- **Reservado**: Espaço alocado de 6 bits para um eventual uso futuro.
- **Tamanho da Janela**: Informa quantos *bytes* podem ser enviados sem receber confirmação.
- ***Flags**:* bits de controle

- **URG**: Sinaliza a presença de dados críticos (urgentes).
- **ACK**: Sinaliza que o presente segmento é de confirmação.
- **PSH**: Sinaliza a iminência de entrega de dados para a aplicação.
- **RST**: Reconfigura uma conexão com mau funcionamento.
- **SYN**: Sinaliza o desejo de estabelecimento de uma conexão.
- **FIN**: Sinaliza o desejo de encerramento de uma conexão.

- ***Checksum****:* Propicia a verificação de erros dos dados transmitidos assim como também dos cabeçalhos.
- ***Ponteiro Urgente****:* Sinaliza informações referentes ao posicionamento das informações críticas no segmento.
- ***Opções*** (Campo Opcional)*:* Busca oferecer recursos não presentes no cabeçalho padrão.

O protocolo TCP é ponto-a-ponto e orientado a conexão, isto é, uma conexão precisa ser estabelecida entre dois elementos, normalmente um cliente e um servidor, que desejam trocar informações para viabilizar o início da transmissão de dados. O estabelecimento da conexão só é atingido através de um procedimento entre os participantes pretendentes denominado de *Three Way Handshake*[9] (SYN, SYN-ACK, ACK) (Figura 5). Esse procedimento aumenta a latência da configuração de uma conexão, mas é justamente o mecanismo que viabiliza alguns dos benefícios citados anteriormente. Além disso, é importante ressaltar que a transferência de dados ocorre de forma simultânea em conexões com uso do protocolo, configurando um sistema *Full-Duplex*.

[9] *Handshake* de Três vias.

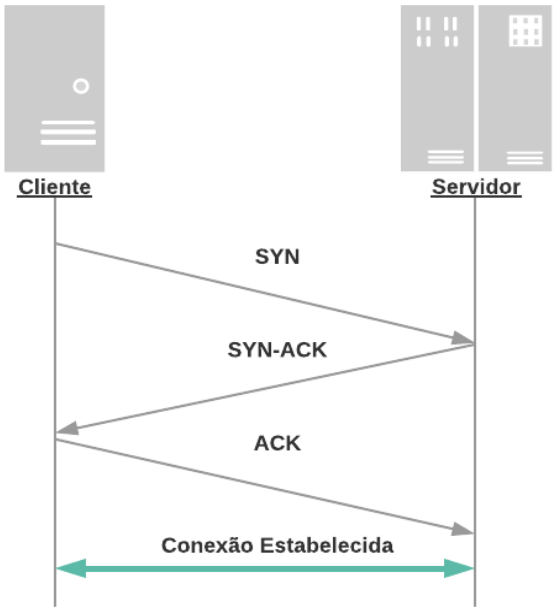

Figura 5 – Three Way Handshake.

Um outro importante recurso oferecido pelo TCP é o da janela deslizante (Figura 6). Uma janela refere-se à quantidade de bytes que podem ser transmistidos sem a confirmação de recebimento. Esse mecanismo surgiu principalmente para evitar a necessidade de cada pacote transmitido requerer uma confirmação (ACK) de recebimento (reconhecimento positivo com retransmissão, um dos métodos que proporciona o aspecto de confiabilidade do TCP), consequentemente subutilizando a banda da rede.

Além de possibilitar a otimização do uso da largura de banda da rede, a janela deslizante também possui o controle de fluxo transmitidos, por instrumento do ajuste dinâmico de seu tamanho, auxiliado por uma variável informativa denominada *advertised window*[10] (RWND). É por meio da janela de recebimento que os dispositivos informam quantos pacotes são habilitados a receber em determinado momento, promovendo apenas o envio do que pode ser processado. A Figura 6 exemplifica o funcionamento do mecanismo da janela deslizante, onde observa-se que o emissor enviou até o sexto segmento sem ser necessário uma confirmação (ACK).

[10] Janela de Recebimento, anunciação ou rececpção.

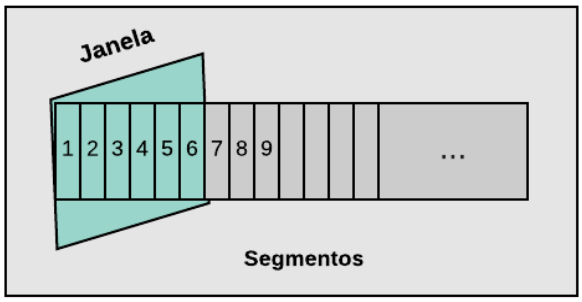

Figura 6 – Janela deslizante.

Aplicações que não exigem confiabilidade na entrega de seus pacotes, isto é, sem qualquer tipo de garantia no envio ou recebimento de pacotes, não confiável, podem utilizar outro protocolo, como o *User Datagram Protocol*[11] (UDP).

2.1.2.1 Controle de Congestionamento

Mesmo com todos os mecanismos preventivos inseridos inicialmente no protocolo TCP, a Internet, ainda em seu estado inicial, começou a apresentar um mau funcionamento que ficou conhecido como *colapsos de congestionamento* [9]. Naquela ocasião, seus utilizadores puderam observar uma queda de desempenho acentuada da rede.

O problema de congestionamento em uma rede de transmissão de dados, como foi observado no relato, ocorre quando se configura um cenário onde existe um agrupamento excessivo de pacotes na rede, diminuindo seu desempenho. Esse congestionamento pode ser acarretado por diversos motivos, como insuficiência de processamento ou memória dos roteadores que constituem a rede, ou mesmo, quando é utilizado o mecanismo do *buffer* nos dispositivos de forma exagerada [14].

Assim sendo, alguns mecanismos foram desenvolvidos e tornaram-se a solução deste problema relatado, visando garantir o pleno funcionamento da rede e impedir eventuais cenários de congestionamentos. No protocolo TCP, as propostas de controle de congestionamento utilizadas tipicamente são [15] [16]: *Slow Start*[12]*,*

[11] Protocolo de datagrama de usuário.
[12] Partida lenta.

Congestion Avoidance[13]*, Fast Retransmit*[14] *e Fast Recovery*[15]. É importante ressaltar que apesar de serem mecanismos independentes entre si, são geralmente utilizados em conjunto.

2.1.2.2 Slow Start e Congestion Avoidance

Os algoritmos de *Slow Start* e *Congestion Avoidance* foram propostos a princípio por Van Jacobson em 1988 [9]. Em um cenário onde a implementação do TCP não dispunha ainda de mecanismos para prevenção de congestionamentos, como já narrado.

Os dois algoritmos têm como objetivo final manter o balanceamento da rede atuando de forma preventiva. O *Slow Start*, propõe aumentar progressivamente a taxa de transmissão de tráfego na rede. Já o *Congestion Avoidance* atua em contextos em que o somatório de fluxos independentes na rede se sobrepõe as capacidades dos dispositivos que viabilizam a rede, como roteadores. O funcionamento dos algoritmos se baseia na adição de mais uma janela de controle de fluxo, chamada de janela de *congestion window* [16](CWND), e um tamanho de inicialização (*start threshold size* ou *ssthresh*), além do uso de RWND, conservado no agente transmissor.

Como a janela deslizante, cuja atuação já foi retratada, a janela de congestionamento também exerce o gerenciamento do fluxo transmitido, mas, levando em consideração uma pespectiva mais abrangente, o estado atual da rede e de seus componentes. Dessa forma, é possível prever acúmulo de pacotes em comutadores ou roteadores, elementos que recebem vários fluxos simultâneos, e são apenas um meio de possibilitar a conexão entre transmissores e receptores entre redes distintas. Já o *ssthresh* é utilizado para definir qual dos dois algoritmos será utilizado momentaneamente, onde seu tamanho inicial é de 65535 *bytes* [15].

Buscando mostrar o uso da CWND e *ssthresh* para a escolha de um dos algoritmos de citados, visto que esses são as ferramentas capazes de indicar a capacidade imediata da rede, é possível observar concisamente como é o funcionamento para definir como crescerá a CWND da seguinte maneira:

---

[13] Prevenção de congestionamentos.
[14] Transmissão rápida.
[15] Recuperação rápida.
[16] Janela de congestionamento.

I. Se ocorrer recebimento *Timeout* ACK ou *ACK* duplicado:
   a. *Ssthresh = CWND/2;*
   b. *CWND =1;*
   c. *Reinicialização da CWND.*

II. Se NÃO ocorrer o recebimento de *Timeout ACK* ou *ACK* duplicado e CWND < ssthresh:
   a. Use *Slow Start* (crescimento exponencial da CWND).

III. Se NÃO ocorrer o recebimento de *Timeout ACK* ou *ACK* duplicado e CWND = ssthresh:
   a. Use *Slow Start* ou *Congestion Avoidance.*

IV. Se NÃO ocorrer o recebimento de *Timeout ACK* ou *ACK* duplicado e CWND > ssthresh:
   a. Use *Congestion Avoidance* (crescimento linear da CWND).

É possível observar o crescimento da janela de congestionamento em relação a cada algoritmo, assim como os requisitos para tal crescimento. Onde a condição *CWND<sshtresh*, indica o crescimento exponencial na taxa de transmissão, com o uso *Slow Start*. E em um cenário *CWND>sshtresh*, ocorre o emprego do algoritmo de *Congestion Avoidance*, com um crescimento linear da taxa de transmissão, incrementando a CWND em um segmento a cada *Round Trip Time*[17] *(*RTT).

2.1.2.3 FAST RETRANSMIT E FAST RECOVERY

Outros algoritmos de controle de congestionamento que podem ser usados dependendo da implementação utilizada do TCP, em conjunto com os apresentados anteriormente são o *Fast Retransmit* e *Fast Recovery*.

O algoritmo de *Fast Retransmit* oferece a possibilidade de retransmissão de pacotes que foram perdidos, com três ou mais ACKs duplicados (esse comportamento no TCP significa que é desejado o número do segmento esperado, além de indicar qual pacote recebido está fora de ordem) em sequência, ao invés de aguardar o *timeout* de retransmissão padrão [15]. Já o *Fast Recovery* possibilita que após a

[17] Tempod de ida e volta.

retransmissão dos pacotes que estavam ausentes, não seja necessário o uso da *Slow start* na prevenção de congestionamento [15].

2.1.3 Implementações do TCP

Ao longo dos anos, diversas mudanças foram implantadas nos algoritmos de congestionamento. Além disso, diferentes combinações foram sugeridas no TCP, buscando um melhor desempenho. Dessa forma, diferentes versões do TCP foram surgindo e se destacando. Na presente seção, serão discutidas algumas dessas implementações, consideradas relevantes para bom entendimento do presente trabalho: Tahoe, Reno, Cubic e BBR.

2.1.3.1 TAHOE

A primeira elaboração do TCP a incluir políticas de controle de congestionamento foi denominada de TCP Tahoe [17] [19], nome dado em homenagem ao *Lake Tahoe*, localizado na costa sul do estado da Califórnia, dos Estados Unidos da América. Essa implementação utiliza de maneira padrão os algoritmos de *Slow Start, Congestion Avoidance* e *Fast Retransmit* atuando em caso de *timeouts* e em ACKs duplicados. Apesar de realizar as funções para qual foi implantando, sua concisa implementação não possibilita ao TCP maior desempenho se comparado com outras propostas.

2.1.3.2 Reno

A implementação do TCP Reno [17] surgiu posteriormente ao do TCP Tahoe, com a proposta de melhorar o desempenho em redes com o alto produto atraso por largura de banda. O Reno possui uma implementação semelhante a do seu precedente, considerando *timeout* de ACKs e ACKs duplicados (que pode indicar uma perda de segmento, reordenação de pacotes na rede ou reodernamento dos pacotes) e alocação justa de banda entre conexões [18], mas se diferencia em relação ao seu funcionamento em respostas a ACKs duplicados. São observados também algumas alterações no algoritmo *Congestion Avoidance* e à adição dos algoritmos *Fast Recovery* [15] [17] e AIMD (*Additive Increase Multiplicative Decrease*) [19] [18]. Assim,

seu funcionamento na ocasião em que ocorre ACKs duplicados pode ser narrado da seguinte maneira:

I. Se ocorrer recebimento de três ACKs duplicados:
   a. *Não uso do algoritmo de Slow Start;*
   b. *ssthresh = (CWND / 2);*
   c. Retransmissão do segmento ausente com o *Fast Retransmit*;

II. *Artificial Inflation*[18] com *Fast Recovery:*
   a. *CWND = sshresh + 3 * MSS;*
   b. A cada ACK duplicado do segmento que ocasionou o *Fast Retrasmit*, CWND = CWND + 1;

III. Após às ocorrências de novos ACKs recebidos com sucesso com uso do Fast Recovery:
   a. *CWND = ssthresh* (com o valor do momento da perda);
   b. *Encerramento do uso do algoritmo Fast Recovery;*
   c. *Execução do Congestion Avoidance* com AIMD, com aumento da CWND maneira linear, mas diminuindo exponencialmente.

Esse incremento estabelece por exemplo uma melhoria de desempenho do Reno em analogia ao Tahoe em redes com grandes taxas de perdas. Posteriormente, outras variações provenientes do Reno surgiram, sempre fundamentalmente buscando melhorar a performance do TCP, como o NewReno.

2.1.3.3 Cubic

Em 2008, foi apresentado um novo algoritmo de controle de congestionamento denominado TCP Cubic [20], buscando melhorar a escalabilidade e estabilidade do TCP em relação às redes rápidas e de longas distâncias, consequência dos avanços estruturais da Internet. Assim como o Reno e outra proposta de implementação chamada *Binary Increase Congestion Controle*[19] (BIC) [21], o Cubic também é um protocolo baseado nas perdas de pacotes para ajustar o comportamento da CWND.

[18] Inflação Artificial.
[19] Controle de Congestionamento de Aumento Binário.

Porém, se destacando em desempenho e escalabilidade em relação ao Reno, e sendo menos agressivo, por consequência mais justo que o BIC [20].

O algoritmo Cubic define como ocorrerá o crescimento da janela de congestionamento, onde é possível observar a seguir, conforme definido na Equação 1:

$$CWND = C \cdot (T - K)^3 + CWND\text{máx}$$

Equação 1 – Função cúbica utilizada no Cubic**.**

Onde,

- *C* é uma variável escalar auxiliar da função Cubic;
- *T* é tempo da redução da janela de congestionamento;
- *CWNDmáx* é tamanho da janela antes da redução;
- CWND é o valor que definirá como ocorrerá o crescimento da janela;
- *K* é período necessário para função conseguir chegar em C*WNDmáx* sem a ocorrências de eventos de perdas. O cálculo da variável *K* é mostrado Equação 2:

$$K = \frac{\sqrt[3]{CWNDmáx \cdot \beta}}{C}$$

Equação 2 – Período necessário para atingir CWNDmáx.

- β é fator de redução multiplicativo.

Por possuir melhor desempenho em relação as propostas anteriores, atualmente o Cubic é utilizado como protocolo de controle de congestionamento padrão TCP em sistemas Linux, após a versão do Kernel Linux 2.6.18 [20], e no Microsoft Windows Server 2016 1709 [22] .

2.1.3.4 BBR

BBR (Bottleneck Bandwidth and Round-trip propagation time), foi desenvolvido pelo Google em 2016 [23], propondo ser os mecanismos utilizado no controle de congestionamento.

Com uma abordagem diferente das outras propostas citadas anteriormente, baseadas principalmente na perda pela indicação de ACKs recebidas no remetente, o modelo sugerido pelo Google tem um comportamento que pode ser considerado híbrido para definir o estado de congestionamento de uma rede. Utilizando o *periodically estimates the available bandwidth*[20] (BWBTL), referente a estimativa da taxa máxima de transferência para ao destinatário indicada pelos ACKs recebidos pelo emissor (semelhantes aos algoritmos de congestionamento baseados em perda) e o *propagation round-trip time*[21] (RTTP), mensurado pelo menor valor das medições RTTs feitas periodicamente. Apesar de ser um padrão incomum de estimativa e inferência de congestionamento de uma rede, outras sugestões como o TCP Vegas [24] e TCP *New* Vegas [25] são igualmente baseadas no tempo de ida e volta obtidos, inclusive influenciando em vários aspectos de funcionamento do BBR [23].

O algoritmo de BBR contém quatro fases: *Startup*, *Drain*, *ProbeBandwidth* e *ProbeRTT* [23].

1. *Startup:*
    a. *Slow start* (New Reno);
    b. Para cada pacote enviado é calculada uma amostra de RTT;
    c. RTTP = RTTmin (nos últimos dez segundos);
    d. Análise da taxa de entrega para cada ACK recebido. Caso o crescimento da taxa de entrega fique estagnado por três RTTs consecutivos (indicador de descoberta do gargalo da largura de banda da rede pelo BBR), o mesmo vai para fase de *Drain*.
2. *Drain:*
    a. Para cada pacote enviado é calculado uma amostra de RTT;
    b. RTTP = RTTmin (nos últimos dez segundos);
    c. Redução da taxa de entrega para uma parcela do produto da estimativa periódica da largura de banda disponível (BWBTL) e Tempo de progagação de ida e volta (RTTP);
    d. O valor bandiwdth delay product[22] *(BDP)* é encontrado através da equação BDP = (RTTP x BWBTL);
    e. CWND = BDP.

[20] Estimativa periódica da largura de banda disponível.
[21] Tempo de propagação de ida e volta.
[22] Atrasdo de largura de banda.

3. *ProbeBandwidth:*
    a. Fase de maior duração no fluxo, ativada periodicamente durante todo período de transmissão;
    b. Propositalmente a taxa de entrega é alterada para um valor maior que o valor BDP;
    c. Volta para a fase *Drain*, com valores reduzidos de taxa de entrega para uma parcela do BDP.
4. *ProbeRTT:*
    a. Fase que é ativada caso o valor do RTTP fique inalterado por dez segundos durante a transmissão;
    b. Interrupção do *ProbeBandwidth.*
    c. Ocorre uma remedição do RTTP.

Uma das formas do Google validar o BBR foi através da exposição de resultados, sem muito destalhes da infraesturura testada, de utilização em suas plataformas, como o *Youtube*, onde foi indicado uma melhoria nas taxas de transferências em média de 4%, e cerca de 14% em alguns países [26]. Mesmo com essa comprovação de aperfeiçoamento que o algoritmo pôde propiciar para o cenário atual de redes, alguns estudos indicaram desequilíbrio e injustiça (destruição dos recursos disponíveis da rede sem igualdade) do BBR entre fluxos concorrentes, com outras propostas de controle de congestionamente, principalmente o Cubic [27] [28].

2.1.4 User Datagram Protocol

O *User Datagram Protocol*[23] (UDP) é um relevante do modelo TCP/IP, pertencendo à quarta camada deste modelo. O mesmo foi definido originalmente no RFC 768 [29].

O *UDP* tem o mesmo propósito de envio de dados, assim como o TCP, mas não é orientado a conexão (*não ocorre o Three Way Handshake*) e também dispõe de menos funcionalidades, como o não uso de mecanismos de controle de congestionamento ou gerência mais robusta da conexão. Isto o torna um protocolo com latência menor, acelerando o processo de envio [10]. Essa ausência de

[23] Protocolo de Datagramas de utilizador.

mecanismos do protocolo indica que as aplicações que o utilizam, deverão implementar essas funcionalidades.

Utilizando o UDP, os pacotes são denominados de datagramas de usuário, formado por cabeçalhos que possuem oito *bytes* [10]. A Figura 7 ilustra a alocação dos cabeçalhos UDP nos datagramas IP.

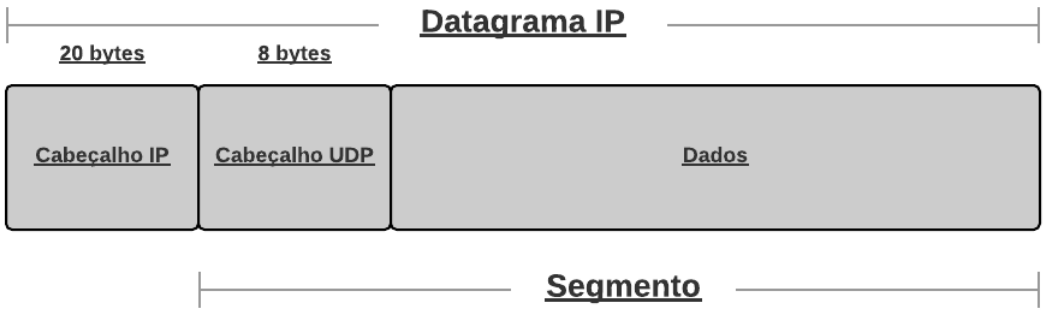

Figura 7 – Segmento UDP.

No UDP, os campos de cabeçalhos são formados pelos campos *Porta de Origem* (Opcional), *Porta de Destino*, utilizados para identificação do processo, *Comprimento*, onde é especificado o tamanho do Cabeçalho mais os dados carregados, e *Checksum* (Opcional), que pode ser usado para verificação de eventuais erros do cabeçalho ou mesmo de dados transmitidos. Na Figura 8, é possível observar os campos de cabeçalho contidos no UDP.

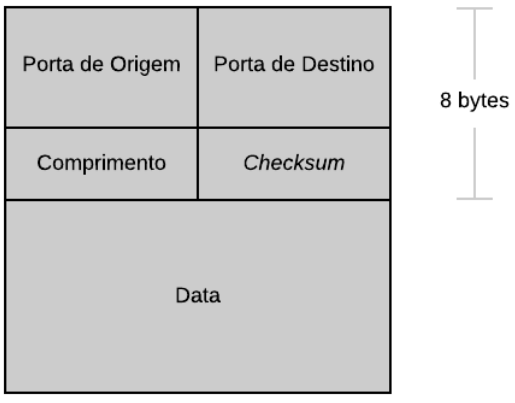

Figura 8 – Campos de cabeçalho UDP.

2.1.5 Hypertext Transfer Protocol

Nas seções anteriores, definiu-se o conceito de alguns protocolos importantes na estrutura tradicional da Internet buscando contextualizar esse documento. Na presente seção, também será discutido outra tecnologia que é considerável para o pleno funcionamento da *Web*, o protocolo HTTP, além de ser extremamente relacionado com o tema principal do trabalho, o protocolo QUIC.

O HTTP tem como fundamental funcionalidade a viabilização ao acesso de dados na *Web*, consequentemente levando-o a ser o protocolo mais utilizado da Internet, considerando os protocolos da camada de aplicação [30]. Atualmente, diversos serviços da *Web* são propiciados através do mesmo.

Como já contextualizado anteriormente, no desenvolvimento da *Web*, foi observado a necessidade de um protocolo que possibilitasse a padronização no acesso aos dados entre os dispositivos contidos naquela conjuntura, essa comunicação foi definida como sendo através de mensagens. Essas mensagens são similares e podem ser classificadas em relação ao seu progenitor e sua funcionalidade: mensagens de solicitação e mensagens de reposta (Figura 9).

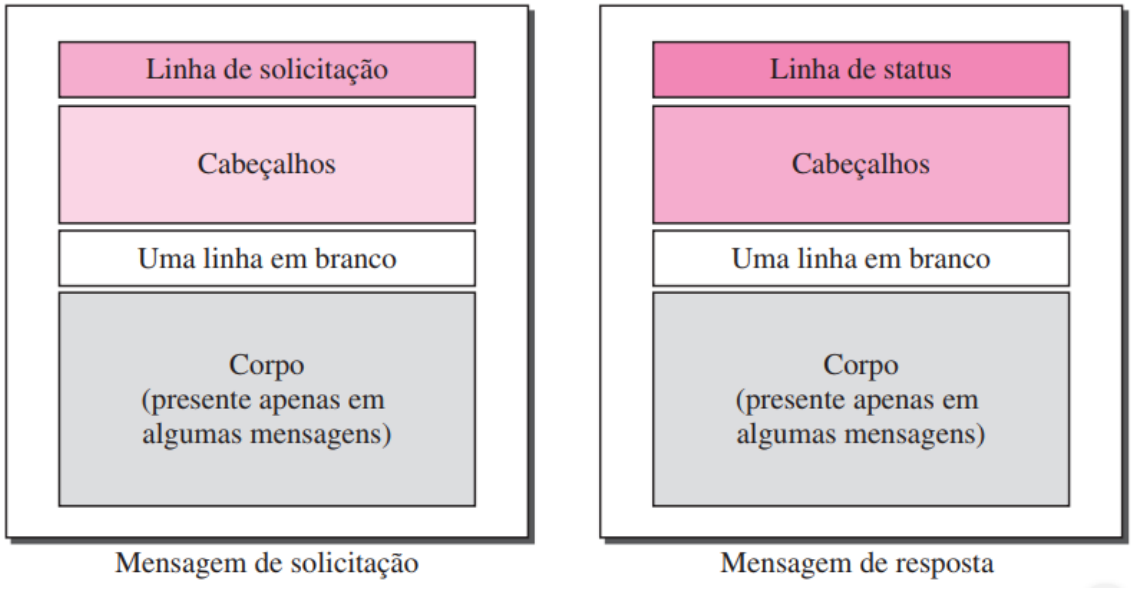

Figura 9 – Mensagens de soclitação e de reposta [10].

Além disso, as mensagens HTTP são constituídas da seguinte maneira: uma linha inicial, que expressa o tipo da mensagem, algumas linhas reservadas de cabeçalho, o corpo da mensagem, que é opcional em casos epecificos, e uma linha

em branco obrigatória de separação entre as linhas de cabeçalho e o corpo da mensagem [31].

Por meio dos serviços do TCP e do IP, o protocolo HTTP torna viável a transferência de arquivos e outras informações através de mensagens encapsuladas enviadas entre cliente e o servidor, onde essas mensagens são iniciadas mediante uma solicitação do cliente HTTP (*browser*) e atendidas por meio de um servidor HTTP [10], como por exelmplo *Apache2* [32]. A Figura 10 ilustra como é realizada a transação convencional entre um cliente um servidor.

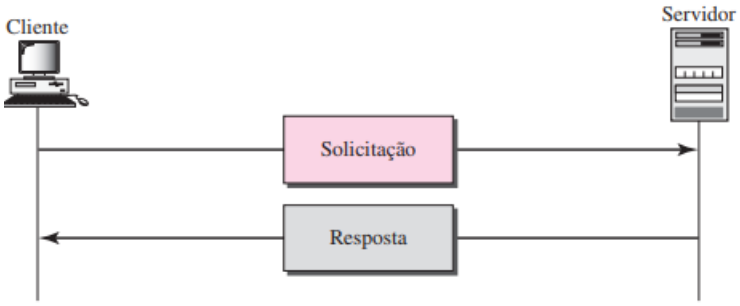

Figura 10 – Transação HTTP [10].

Uma outra caracteristica significativa do protocolo, considerando seu padrão de conexão na versão 1.0, é ser *stateless*[24], ou seja, o HTTP considera cada requisição como uma transação independente, sem qualquer relação com outras conexões anteriores ou posteriores. Esta individualidade garante que a comunicação entre os elementos ocorra sempre com uma requisição seguida de uma resposta independente [10].

Em um cenário posterior ao da formalização da versão 1.0 do HTTP, onde as páginas *Web* começavam a apresentar mais conteúdo (.html, .gif, .css, .jpeg, .png, etc) [1], esse modelo de conexão utilizado pelo HTTP/1.0, proporcionava a necessidade de criação de diversas conexões para um acesso de página comum, ocasionando o uso de recursos das máquinas além de gerar mais *overhead*[25] em uma rede, em razão do grande número de conexões necessárias. Essa postura foi um dos motivos que ensejou o surgimento e implantação da versão HTTP/1.1 [31], que propôs um modelo diferente de conexão através do uso de *persistente connection*[26] e

[24] Sem estado.
[25] Sobrecarga.
[26] Conexão persistente.

pipelining, possibilitando múltiplas requisições e repostas não correspondentes por conexão [33]. Além dessa mudança citada, a variante 1.1 do HTTP, proporcionou outras melhorias relacionadas a aspectos de confiabilidade, performance e segurança do protocolo [31], deixando mais adequado com o contexto da *Web* até certo tempo atrás.

Assim como ocorreu no HTTP/1.0, com o passar do tempo, mais uma vez, o HTTP, agora em sua revisão 1.1, também começou a mostrar sinais de incoerência com as transformações cada vez mais rápidas das particuliaridades dos serviços *Web*. Um indicador desse contexto foi em relação ao tamanho das páginas *Web*, que dentro de um intervalo de apenas três anos (2012 e 2015), dobrou [1]. Evidenciando, dessa forma, a necessidade de uma outra remodulação.

Em 2009, uma das sugestões de reforma do HTTP surgiu através do Google, que à época apresentou uma modificação experimental deste, denominada SPDY (abreviatura da palavra *speedy*[27]). O SPDY foi elaborado essencialmente visando reduzir a latência de carregamentos de páginas Web, mediante o uso de basicamente três funcionalidades: multiplexação, priorização e compressão [34], obtendo bons resultados de desempenho em relação a seu ascendente [35].

Através dos bons resultados demonstrados pela proposta, o SPDY por consequência se consolidou com uma das principais premissas do HTTP/2 [36].

O HTTP/2 surgiu em 2015 [37], e atualmente é a versão do HTTP mais recente e empregada desse protocolo, sendo responsável por cerca de 58% (cinquenta e oito por cento) das requisições HTTP, tanto em dispositivos móveis, como em Desktops [1].

2.2 PESPECTIVA ATUAL DE REDES DE COMPUTADORES

Na pesquisa realizada anualmente, publicada em fevereiro de 2019 pelo Facebook, é apontado que cerca de 3,8 bilhões (três bilhões e oitocentos milhões) da população mundial tem acesso à internet e que ainda, o mesmo número de pessoas não tem acesso [38]. Mesmo que a metade da população mundial ainda não utilize esta ferramenta, constata-se que a tecnologia atualmente se tornou parte essencial da infraestrutura social, assim como a energia elétrica [8].

[27] Rápido.

Mesmo com a popularização da Internet, Rothenberg ressalta que o desenvolvimento da Internet, sua tecnologia, representada fundamentalmente pela arquitetura em camadas e protocolos do modelo TCP/IP não teve uma evolução satisfatória:

> Apesar da evolução formidável da Internet, em termos de penetração e de aplicações, sua tecnologia, representada pela arquitetura em camadas e pelos protocolos do modelo TCP/IP, não evoluiu suficientemente nos últimos vinte anos [39].

Principalmente através da popularização da Internet, serviços e aplicações estão se tornando cada vez mais complexas, exigindo bem mais da pilha de protocolos pertencentes ao modelo TCP/IP, e consequentemente requerendo melhorias. Um exemplo dessa afirmação são as características das páginas *Web* atualmente, cada vez mais dinâmicas e complexas, diferente do contexto onde foram idealizadas, onde as páginas eram simples e estáticas [1].

Proporcionamente a essa popularização da *Web*, existe a reivindicação de baixa latência das conexões, sem a renúncia de aspectos como segurança, integridade ou confiabilidade.

De uma maneira conceitual, quando é realizado um acesso a uma eventual página *Web*, através de um *browser* (Cliente), a estrutura básica do agrupamento dos protocolos utilizados é semelhante a Figura 11.

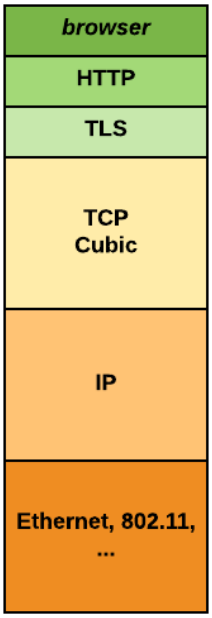

Figura 11 – Conjunto tradicional de protocolos.

Nessa concisa demonstração da Figura 11, é possível observar a utilização dos protocolos contidos na Camada de Aplicação, o HTTP e o *Transport Layer Security*[28] (TLS). Ao longo do tempo com a mudança e exigência dos conteúdos na *Web* já indicados anteriormente, esses protocolos sofreram mudanças. O protocolo HTTP, por exemplo, recebeu atualizações e consequente melhorias, saindo de sua versão inicial, obtendo versões 1.0, 1.1 e 2. Assim como o TLS, que tem a finalidade de proporcionar segurança das informações nas comunicações sobre uma rede de computadores, ganhando versões 1.1, 1.2 e 1.3.

Essas melhorias foram possíveis, pois as tecnologias relacionadas à camada citada, obrigatoriamente tem como característica o alto nível de abstração (nível de usuário), ou seja, não interferindo diretamente na infraestrutura de rede geral. Já os outros protocolos que são associados às camadas mais inferiores, como o TCP, possíveis propostas de mudanças são mais difíceis de serem implementadas, pelo fato de que as validações das mesmas geralmente são custosas, sendo necessário diversos testes. Além disso, mesmo que essa eventual proposta seja uma vez validada, a implantação da mudança também acarraterá um grande custo, porque exige alteração de componentes mais sensíveis na infraestrutura tal como kernel de

[28] Segurança da camada de transporte.

sistemas operacionais, software de NAT e *firewall* (no caso do TCP) ou mesmo *firmware* de roteadores (no caso do IP).

Outro ponto que é importante evidenciar, é em referência à configuração de comunicação a um eventual servidor *Web*. Utilizando a pilha de protocolos demonstrados na Figura 11 seria necessário entre dois ou três RTTs, obrigatoriamente um para o *handshake* TCP, e um RTT ou dois, pela utilização do protocolo de criptografia TLS (1.2) e HTTP 1.1 (Figura 12). O novo TLS 1.3 adota novas técnicas de autenticação operando sobre um RTT somente, assim diminuindo o tempo de estabelecimento de conexão requerido pelo TCP.

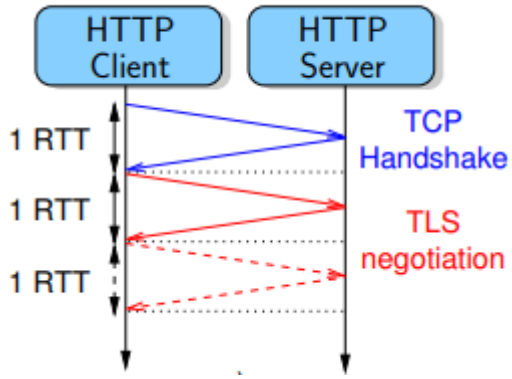

Figura 12 – Configuração de latência [3].

Essa invividualidade do TCP no estabelecimento das conexões, apesar das vantagens já indicadas na **Seção 2.12**, pode diminuir muito o desempenho, e influenciar negativamente na latência, em ambientes onde fatores como *loss*[29] ou *delay*[30] são encontrados, que são comuns em determinados cenários da Internet, indo justamente de encontro com uma das exigências atuais, a baixa latência.

Portanto, para suprir a exigência das novas possibilidades ferramentais atuais, existe necessidade de evolução da Internet, consequentemente dos protocolos que permitem seu funcionamento. Diversas iniciativas de aperfeiçoamento já foram sugeridas, como o próprio BBR, comentado na **Seção 2.1.3.4,** e o Quick *UDP Internet Connections* (QUIC) que passará a ser abordado no próxima seção deste trabalho.

[29] Perda.
[30] Atraso.

2.3 QUICK UDP INTERNET CONNECTIONS

O paradigma proposto pelo *Quick UDP Internet Connections*[31] (QUIC) é uma nova forma de encarar a estrutura da pilha de protocolos TCP/IP tradicional, sugerindo substituir alguns dos principais protocolos do modelo, a fim de proporcionar melhoria no cenário atual.

O QUIC foi desenvolvido pelo Google (no ano de 2012), propondo suprir as dificuldades reconhecidas pelos pesquisadores e desenvolvedores da empresa em relação ao desempenho e funcionalidade do TCP, com a conjuntura atual da Internet.

2.3.1 Proposta

Na página de documentação do *browser* Chromium, um projeto de código aberto do Google, que auxilia o *browser Google Chrome* a fundamentar seu código fonte, a definição da proposta do QUIC é iniciada dando ênfase a seu *design*:

> O QUIC é muito semelhante ao TCP + TLS + HTTP/2, mas implementado no UDP. Como o TCP é implementado nos kernels do sistema operacional e no firmware das *middlebox*, fazer mudanças significativas no TCP é quase impossível. No entanto, como o QUIC é construído sobre o UDP, ele não é impactado por essas limitações [40].

Assim, a validação e emprego do protocolo nas redes usuais se tornou uma tarefa viável já que os *firewalls* reconhecem tráfego QUIC como UDP. Além disso, é também indicado que o QUIC oferece diversas vantagens em relação ao modelo mais convencional (**Seção 2.2**), como a redução da latência no estabelecimento das conexões, controle de congestionamento aprimorado, multiplexação, correções de erros e migração de conexão [40] [41].

Desse modo, o esboço conceitual ilustrado na seção anterior (Figura 11) passaria a ser como o esquema arquitetônico apresentado na Figura 13.

[31] Conexões rápidas de Internet à UDP

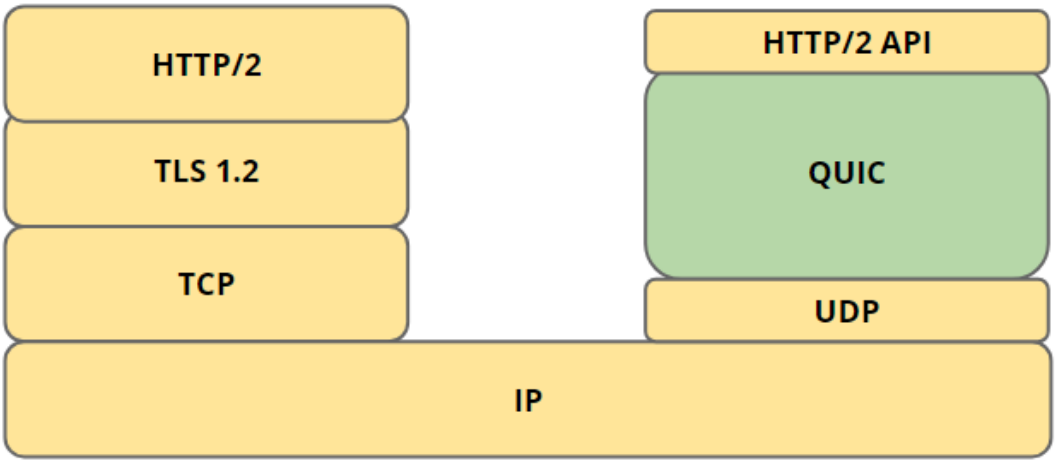

Figura 13 – Funcionamento do QUIC em relação o modelo habitual [41].

Como apontado na **Seção 2.1.4**, o UDP não é um protocolo que implementa certas funcionalidades que auxiliam as conexões, como o controle de congestionamento. Em contrapartida, na implementação do QUIC, esses mecanismos são implantados justamente pela própia proposta.

2.3.1.1 Estabelecimento de conexão e Segurança

Um dos fatores que influenciam significamente na diminuição da latência e na implementação de segurança das conexões utilizando o protocolo QUIC é como o mesmo efetua o estabelecimento de conexões.

Como exemplificado na **Seção 2.2**, tradicionalmente a maioria dos serviços da *Web* empregam a pilha TCP + TLS + HTTP, sendo necessário no mínimo dois RTTs para o estabecimento de uma conexão com êxito. No QUIC, ocorre a condensação das informações necessárias para a viabilização do estabelecimento de uma conexão segura, mesclando os parâmetros relevantes de configuração para o estabelecimento da conexão com os dados necessários para funcionamento do protocolo de criptografia (TLS 1.3), reduzindo a necessidade de RTTs para um. Na Figura 14, é possível observar uma ilustração comparativa a respeito do QUIC e TCP em relação ao estabelecimento de novas conexões.

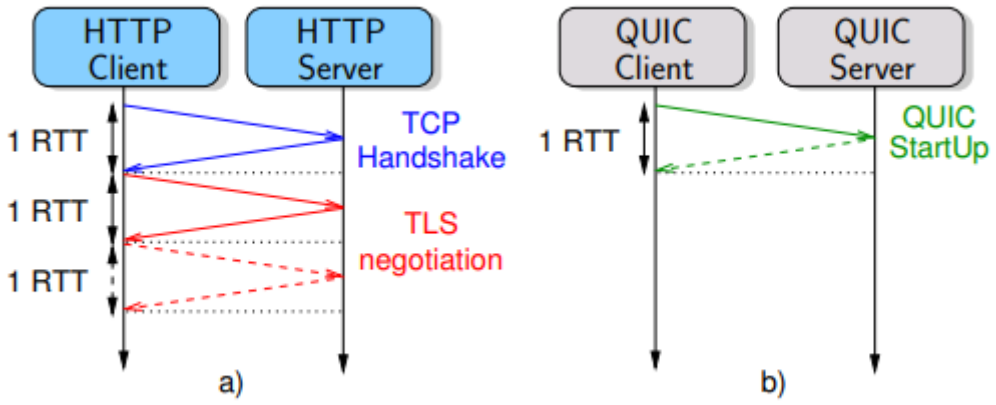

Figura 14 – Latência de inicialização [3].

Outra particuliaridade do QUIC é que caso algum cliente queira restabelecer comunicação com um servidor familiar (que eventualmente já tenha consumido determinado serviço, não será necessário ocorrer um novo *handshake* para configurar aquela conexão, reduzindo o tempo de inicialização da conexão para geralmente 0-RTT (Figura 15), isso é possível através do algoritmo QUIC-*Crypto*, contido no própria própia proposta do Google. Na prática, o algoritmo QUIC-*Crypto* utiliza um *cookie* criptográfico armazenado em *cache* no cliente, que contém dados de identificação dos servidores já conhecidos [42].

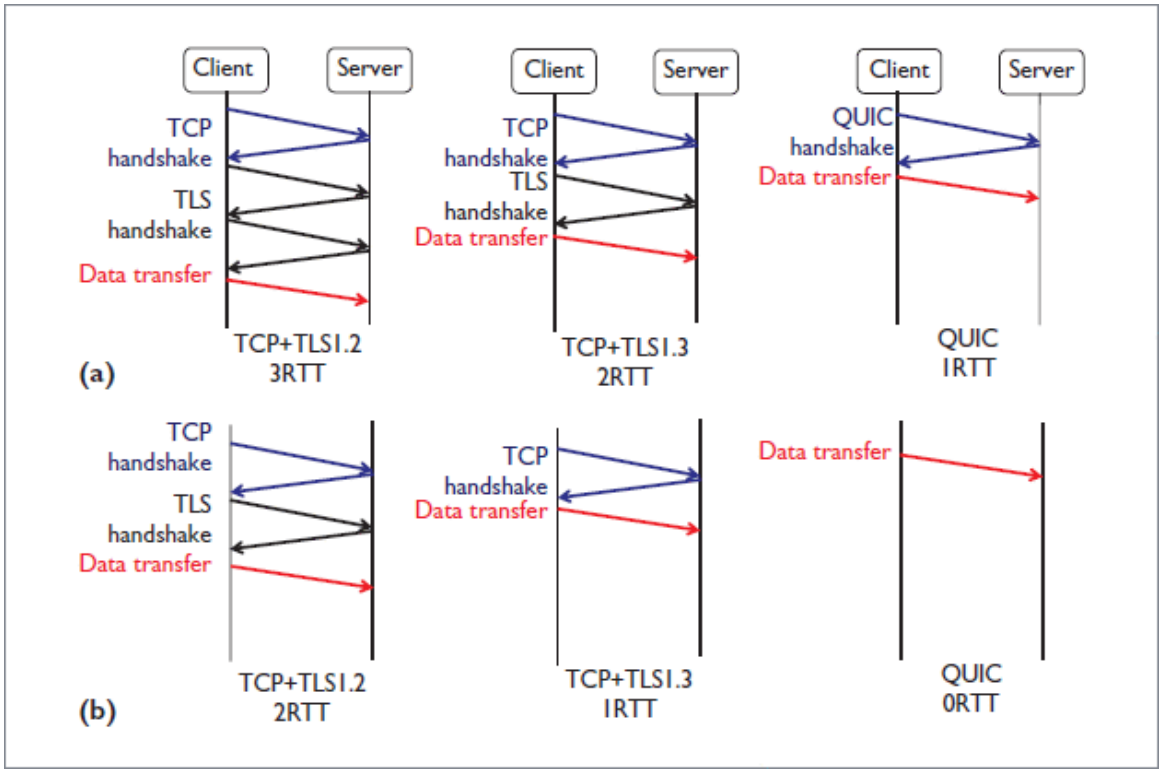

Figura 15 – Tempo de ida e volta do handshake (RTT) de diferentes protocolos. (a) Estabelecimento de conexão inicial. (b) Conexões subsequentes [43].

Este modelo de estabelecimento de conexão implementado no QUIC indica ser, e, consequentemente, é um dos fatores para diminuição da latência na rede.

De acordo com o esboço de RFC responsável pela formalização da segurança do protocolo QUIC, atualmente o QUIC é formalizado com a versão do *Transport Layer Security* 1.3 [44].

2.3.1.2 Multiplexação

O artifício de multiplexação do protocolo QUIC, assim como no HTTP/2, foi um dos mecanismos herdados da proposta de modificação do HTTP/1.1, o SPDY. Apesar de compartilharem da mesma origem, a implementação desse recurso diverge entre as tecnologias.

O HTTP/1.1, mesmo com a adição de técnicas que sugeriram melhoria de desempenho, como *a pipelining* (**Seção 2.1.5**), a obtenção dos objetos de uma aplicação *Web* era feita pelo cliente sempre gradualmente e sequencialmente, ocasionando eventualmente o surgimento de gargalos, como quando o número limite de solicitações paralelas é alcançado pelo cliente e posteriores solicitações

necessitam aguardar o término dessas, fenômeno denominado de *Head-of-line blocking*[32] (HOL) (nível de aplicação).

No HTTP/2 tradicional com o TCP, através da implementação do modelo de requisição por multiplexação de fluxos indepedentes, que utiliza uma única conexão TCP para consumir determinado servidor, o problema do HOL a nível de aplicação foi solucionado. Porém, em detrimento do uso do TCP, outro modelo do HOL foi realçado (nível de transporte), devido as características de controle de congestionamento do TCP, que quando identifica alguma perda de pacote em determinado fluxo, faz com que todos os outros fluxos seguintes aguardem a retransmissão daquela informação perdida, invialibilizando a entrega fora de ordem [45].

Como solução para o problema exposto, o QUIC propõe o emprego do UDP para substituição do TCP como protocolo de transporte, visto que esse último não possuí recursos mais robustos de gerenciamento de conexões, como o controle de congestionamento, permitindo por exemplo a entrega de pacotes sem aguardar retransmissões (Figura 16).

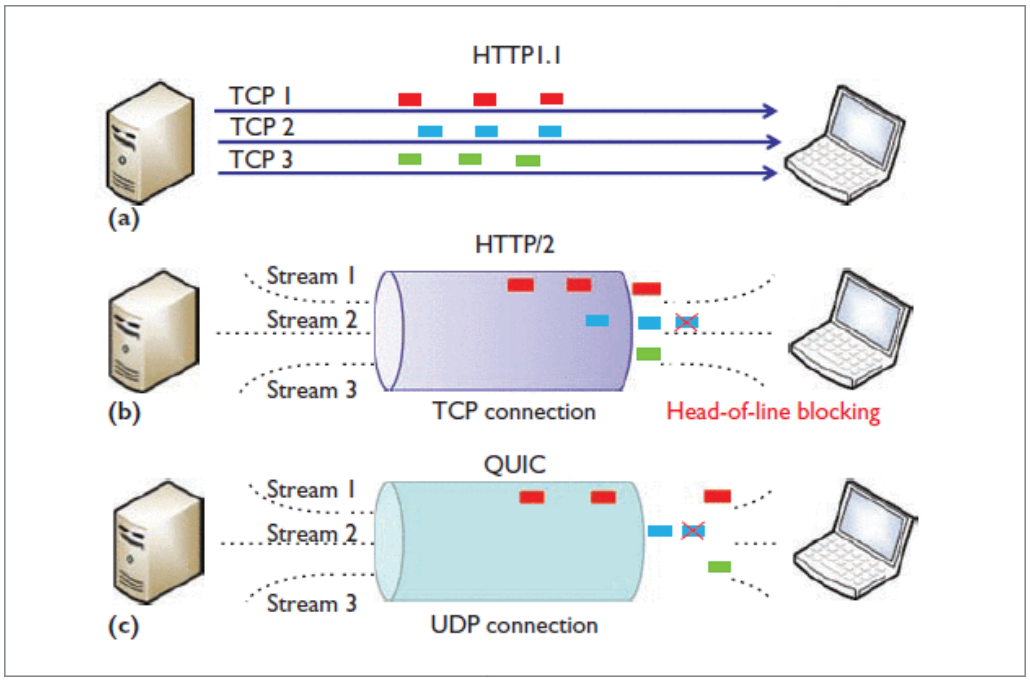

Figura 16 – Comparação de multiplexação. Isso envolveu o envio de vários fluxos de dados por uma única conexão de transporte usando (a) HTTP/1.1, (b) HTTP/2, e (c) Quic [43].

[32] Bloqueio de linha de cabeçalhos.

2.3.1.3 Correção de erros e Controle de congestionamento

Como já enfatizado, o UDP não dispõe de mecanismos para contornar eventuais problemas em conexões. Dessa forma, coube ao QUIC, implementar recursos otimizados para gerenciar essa questão.

Em relação à perda de pacotes, o protocolo QUIC dispõe de dois módulos denominados *Sliding Window Random Linear Code* (RLC) e *Forward Erasure Correction* (FEC) que auxiliam na recuperação e tratamento em contextos com perdas de pacotes [46].

Já no que diz respeito ao controle de congestionamento, vale ressaltar que para distribuir os recursos de rede de maneira igualitária e justa entre fluxos, em conjunto com o tráfego TCP, por exemplo, o QUIC adota mecanismos de controle de congestionamento similares ao TCP, empregando o algoritmo de congestionamento Cubic (**Seção 2.1.3.3**) como padrão, oferecendo uma interface conectável e flexível que suporta o uso de outros algoritmos com a mesma funcionalidade, como o NewReno [43] e BBR [47].

Apesar da implementação referente ao controle de congestionamento QUIC + Cubic ser bastante parecida com o modelo tradicional TCP Cubic, como acima mencionado, algumas modificações foram acrescentadas ao QUIC, como por exemplo a adição do recurso de *Early Retransmit*[33] (antecipa as retransmissões de pacotes) e alterações no funcionamento do *Fast Retransmit*, detecção baseada em um temporizador, entre outras, buscando antes de tudo tornar a proposta mais otimizada em analogia a seu semelhante [48] .

[33] Retransmissão antecipada.

3 TRABALHOS RELACIONADOS

Este capítulo descreve a fundamentação teórica do estudo e também os principais conceitos relacionados com o presente trabalho, revisando o estado da arte.

A abordagem bibliográfica sobre o protocolo QUIC, assim como de outros componentes considerados na dissertação, fundamentando-se em trabalhos acadêmicos, artigos e livros, refere-se principalmente a seus conceitos, inovações e em novas perspectivas de como a tecnologia poderia ser aplicada.

No trabalho de Carlucci [3], são retratados dois experimentos efetuados referentes ao uso do protocolo QUIC e TCP. Inicialmente é produzida uma análise comparativa do comportamento do algoritmo de controle de congestionamento Cubic com os referentes protocolos em questão e, subsequentemente, é observado o impacto causado do emprego do QUIC em relação ao tempo de carregamento da página em paralelo com as propostas do SPDY e HTTP/1.1. Os dois experimentos foram conduzidos em diferentes condições de rede e reveleram melhor desempenho do QUIC em relação ao TCP Cubic em redes *under-buffered* [49], ou seja, redes onde os roteadores tem *buffer* insuficiente, incompatível com as características dos elementos da rede, não dispondo de um tamanho maior ou igual a RTT x C, sendo RTT o tempo médio de ida e volta de um fluxo passando pelo *link* e C a taxa de dados do *link*, mesmo aumentando a perda de pacotes. É também mostrado melhor comportamento do QUIC no tempo geral de recuperação de paginação *Web* em relação ao SPDY e HTTP/1.1.

Outros estudos que enfatizam o tempo de carregamento das páginas podem ser encontrados em Megyesi [7] e Nepomuceno [50]. Onde se destacam no primeiro, o fraco desempenho do QUIC em ambientes com grande largura de banda e no *download* de grandes quantidades de dados, e no segundo, um desempenho pior do protocolo em relação ao TCP nos 100 (cem) *sites* mais acessados do mundo.

Já em Wang [51], é apresentado uma implementação do protocolo QUIC no Kernel Linux e feita uma comparação de desempenho com o TCP. Nesta pesquisa, foi adotado o padrão 802.11. Já a implantação dos elementos inseridos no contexto

da pesquisa foi propiciada pelo emprego da virtualização, por meio de máquinas virtuais. Questões relacionadas a performance de rede como, *Throughput*, atraso e perda foram evidenciadas e constataram que o QUIC superava o TCP.

Wang [47], análisa os protocolos QUIC e BBR no cenário de Internet via satélite, onde aspectos de rede como perda de pacotes e atraso são altos. A avaliação foi conduzida preliminarmente em emulação de rede dedicados, mostrando que o QUIC com BBR tornam a velocidade de transmissão mais estável além de reduzir seu atraso.

Srivastava [4] aborda o desempenho do controle de congestionamento padrão do QUIC e TCP Cubic em três conjuntos de experimentos, observando aspectos de rede como atraso *e* perda. Os resultados indicaram injustiça do QUIC contra vários fluxos concorrentes TCP no compartilhamento de recursos da rede, além que em cenários com maiores perdas de pacotes, o TCP tem rendimento menor que o QUIC.

No trabalho de Camarinha [5], é feita uma avaliação do QUIC através do uso de simulação, com caráter determinístico, usando a ferramenta *NS3* e o módulo nsQUIC no experimento. Nesse estudo, inicialmente foram analisados praticamente todos os protocolos mais conhecidos de congestionamento TCP, com exceção do BBR, em comparação com o QUIC. Evidenciando a superioridade do TCP Cubic e QUIC em relação às outras propostas, e posteriormente também evidenciando melhor performance do QUIC em todos os cenários desenvolvidos já sendo comparado diretamente com o Cubic, principalmente em relação à métrica Tempo de transmissão.

A Tabela 1 apresenta uma breve descrição dos trabalhos relacionados buscando estabelecer uma apresentação concisa das métricas utilizadas em cada pesquisa.

TRABALHOS RELACIONADOS	VERSÃO DO HTTP COM TCP	TCP TLS?	VERSÃO DO TLS	VERSÃO DO QUIC	ALGOR. CONGE. QUIC	MÉTRICAS
Carlucci [3]	HTTP/1.1 e SPDY	✓	TLS 1.2	Q021	Cubic	*Goodput*, Utilização do Canal, Razão de perda e Tempo de Carregamento da página
Megyesi [7]	HTTP/1.1 e SPDY	✓	-	-	Cubic	Tempo de carregamento da página
Nepomuceno [50]	HTTP/1.1	✓	-	-	Cubic	Tempo de Carregamento da página
Wang [51]	X	-	-	-	Cubic	*Throughput*
Wang [47]	HTTP/2	✓	TLS 1.2	Q039	Cubic e BBR	*Throughput*, *Goodput* e Justiça
Srivastava [4]	HTTP/1.1	X	X	Q036	Cubic	Throughput e Justiça
Camarinha [5] **(Simulação)**	HTTP/2	✓	TLS 1.2	Q041	Cubic	Justiça e Tempo Total de Transmissão
Este Trabalho	HTTP/2	✓	TLS 1.3	Q046	Cubic e BBR	*Throughput*, *Overhead*, Justiça e Tempo Total de Transmissão

Tabela 1 – Concisa demonstração dos trabalhos relacionados e suas respectivas métricas.

Apenas algumas fontes de conhecimento apresentaram uma análise no que concerne o comportamento do controle de congestionamento no protocolo QUIC em relação ao TCP [3] [47] [4] [5], nenhuma faz o emprego simultâneo dos elementos propostos no estudo de caso idealizado. Assim, esta realidade reforça a originalidade e o valor dessa pesquisa.

4 MÉTODO DE PESQUISA

Neste capítulo é apresentado a metodologia empregada nesta pesquisa. Incialmente é indicada a abordagem de pesquisa, seguida de uma apresentação detalhada de como foi realizada a coleta e tratamento de dados. Posteriormente, são apresentadas as das ferramentas utilizadas para a análise. Por fim, são descritos aspectos referentes à validade e as limitações do estudo.

4.1 METODOLOGIA

A metodologia utilizada neste estudo considerou o estudo de caso aplicando a pesquisa exploratória. A metodologia abrangeu as seguintes etapas:

a) Levantamento de trabalhos;
b) Revisão conceitual sobre os principais conceitos de redes de computadores;
c) Estudo teórico do protocolo QUIC, além de alguns protocolos do modelo TCP/IP;
d) Desenvolvimento de um estudo de caso, usando um cenário de aprendizado, com o objetivo de apresentar e análise sobre as tecnologias estudadas.

Foi aplicado o estudo de caso para esta pesquisa exploratória por ser o método mais adequado às particularidades desse trabalho.

Para a execução dos experimentos, o ambiente foi configurado de forma que representasse da melhor maneira possível à Internet. Por meio de pesquisas, foram estabelecidas várias configurações que influenciam e representam a Internet do mundo real. Nas próximas subseções, é apresentado a configuração dos experimentos, além das métricas e fatores utilizados.

4.2 TESTBED E EXPERIMENTOS

Na presente seção são retratadas as informações, mecanismos, além das regras gerais referentes ao desenvolvimento e configuração do ambiente desenvolvido no estudo.

4.2.1 Organização do experimento

O *testbed* consistiu de seis máquinas, possuindo as mesmas configurações de *hardware* e *software*, com exceção dos dispositivos *Router1* e *Router2* (Figura 17), que possuem duas interfaces de rede (NICs) extras, conforme apresentado na Tabela 2.

Configuração	Máquina nativa
CPU	1x Intel Core i3-2120 @ 3.30Ghz
Memória RAM	8 GB
Disco Rígido	WDC WD10EZEX-00R
NIC	Intel 82541PI Gigabit
NIC Extras	Realtek RTL 8111/8168/8411 Gigabit

Tabela 2 – Hardwares do testbed.

As especificações das versões dos principais *softwares* utilizados são apresentadas na Tabela 3.

Configuração	Máquina nativa
Sistema Operacional	Ubuntu 18.04.3 LTS
Kernel	4.15.0-74-generic
Wget	1.99.2
Apache	2.4.41
QUIC	Q046

Tabela 3 – Softwares do testbed.

O ambiente consistiu de seis máquinas, denominadas *Server1, Server2, Router1, Router2, Client1 e Client2*, organinzadas com a topologia *dumbbell*[34]. Esse esquema físico costuma ser bastante empregado em pesquisas onde se deseja observar os impactos que diferentes condições de rede desencadeiam em mecanismos [4] [5], como nos algoritmos de congestionamento, que são influenciados

[34] Haltere.

diretamente por esses fatores. A topologia foi composta de dois elementos principais: comumente roteadores, que viabilizam a transmissão de dados entre os *n* nós atrelados a eles. O ambiente desenvolvido está representado na Figura 17.

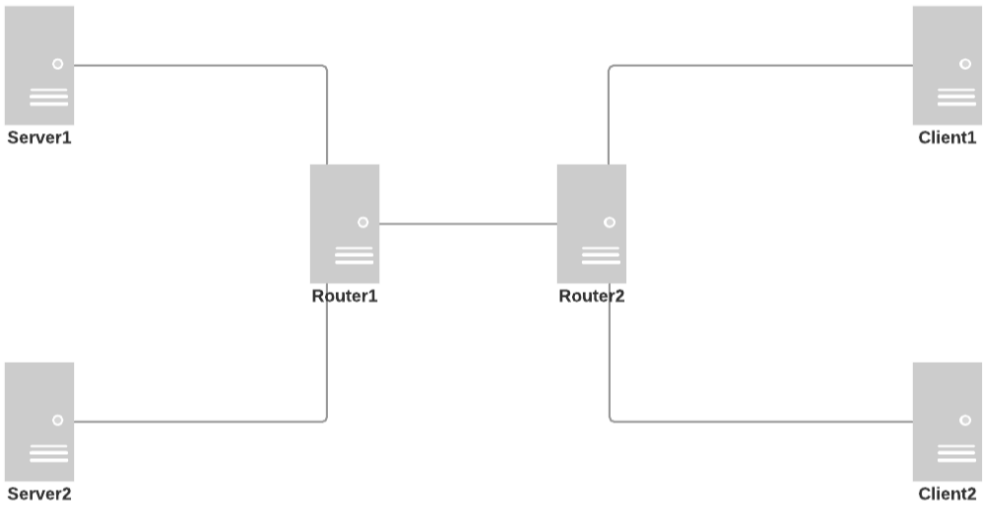

Figura 17 – Topologia dumbell do testbed.

4.2.2 Configuração de rede

Esta seção esclarece as configurações de rede de cada elemento do *testbed* (Figura 18).

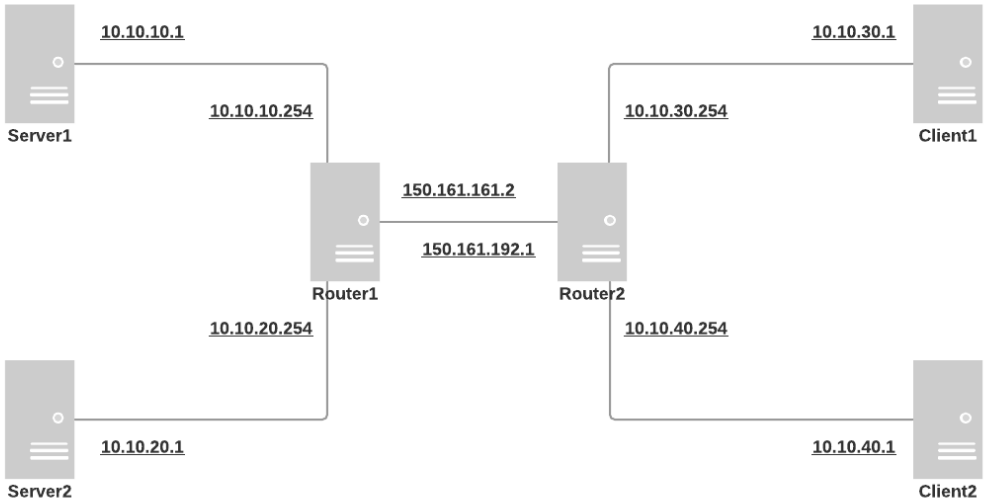

Figura 18 – Topologia dumbell com configuração de rede.

As interfaces de comunicação do dispositivo *Route1* consistiu de três configurações, em que os endereços das placas de rede correspondiam a 150.161.192.2/24, 10.10.10.254/24 e 10.10.20.254/24. No *Route2* os endereços das placas de rede correspondiam a 150.161.192.1/24, 10.10.30.254/24 e 10.10.40.254/24.

Os *hosts Server1 e Server2,* que desempenharam o papel de servidor na plataforma experimental foram atribuídos com os endereços 10.10.10.1/24 e 10.10.20.1/24, na devida ordem.

Já nos elementos Client1 e Client2, foram definidos com os endereços de rede 10.10.30.1/24 e 10.10.40.1/24, respectivamente.

O código de configuração correspondente do arquivo */etc/network/interfaces* de cada dispositivo do *testbed* pode ser observado no Apêndice D.

Foi também realizado um ajuste adicional buscando viabilizar a conexão entre todos os *hosts* na topologia *dumbell*. Em ambos os roteadores foi essencial a ativação da variável do kernel *ip_foward*.

Além disso, todas as interfaces de redes foram normatizadas para o *Maximum Transmission Unit* [35](MTU) padrão de 1500, que é o valor utilizada comumente nas intercomunicações, e o módulo IPV6 foi desabilitado das interfaces de rede de cada dispositivo buscando não interferir na coleta de dados das transmissões.

4.2.3 Cénarios do Experimento

A partir da topologia *dumbell*, foram desenvolvidos dois cenários para análise. Os cenários foram configurados de maneira semelhante levando em consideração a estrutura de protocolos, mas distinguem-se em outros aspectos, como sua finalidade.

Na camada de aplicação, foram usados o HTTP/2 e o TLS 1.3 combinados com os protocolos TCP e QUIC. A Figura 19 resume as configurações do protocolo durante a análise. Os protocolos são combinados desta forma, considerando a comparação entre a estrutura que o QUIC propõe (QUIC e HTTP/2) com o que é mais utilizada na Internet atualmente (TCP e HTTP/2).

[35] Unidade Máxima de Transmissão.

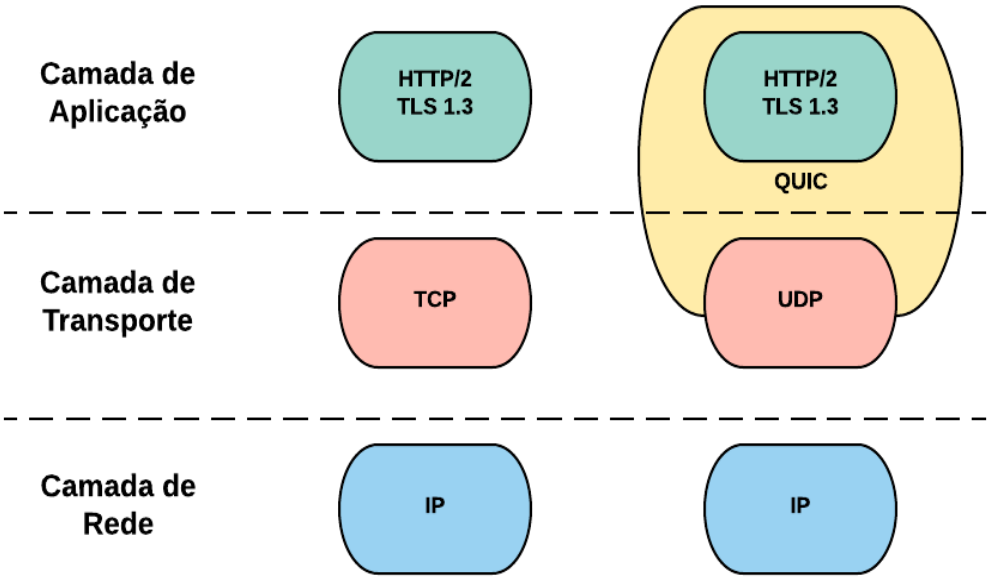

Figura 19 – Cenário dos protocolos.

A configuração completa dos cenários, assim como seu objetivo, pode ser observada da seguinte maneira:

1. **Cenário Base***:* Esta configuração possui *Client1 (wget2 ou Cliente QUIC)* consumindo serviço do *Server1 (Apache ou QUIC Server)*, alternando o uso dos protocolos TCP ou QUIC em conjunto com os algoritmos de congestionamento BBR ou Cubic. A finalidade deste cenário foi isolar o comportamento dos protocolos de transporte com cada possível grupamento de parâmetros destacados na **Seção 4.3**. As Figura 20 e Figura 21 ilustram o fluxo no cenário Base com o protocolo QUIC e TCP.

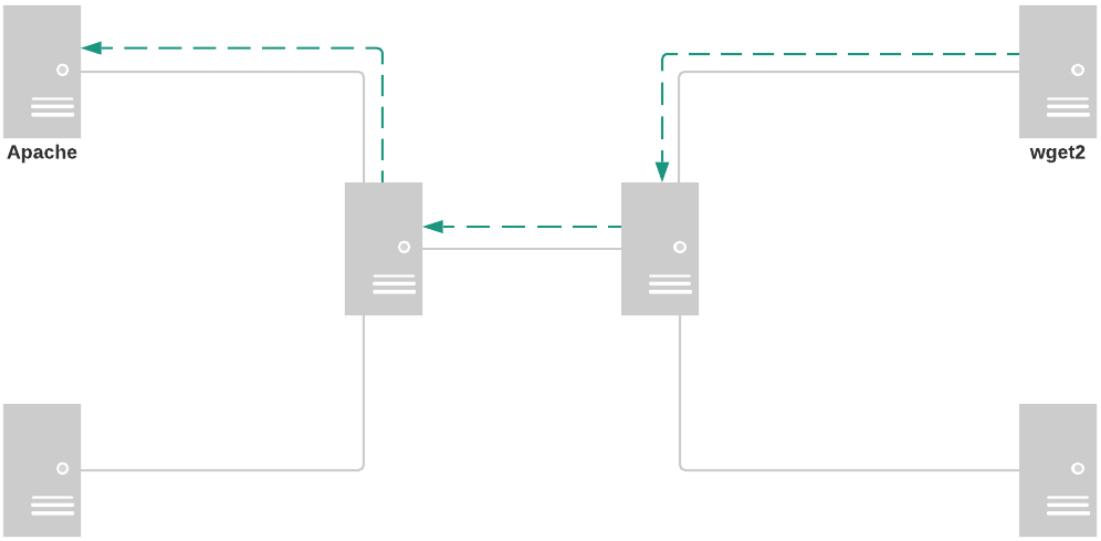

Figura 20 – Cenário Base com uso do TCP.

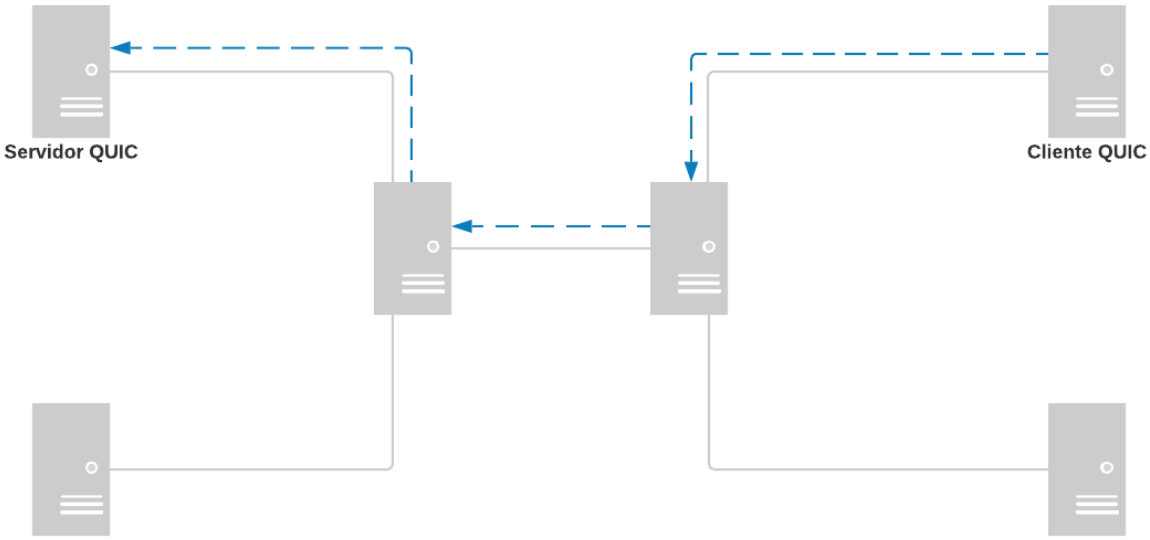

Figura 21 – Cenário Base com uso do QUIC.

2. ***Cenário 2****:* Este cenário incluí os dispositivos *Client1* e *Client2 (wget2 ou Cliente QUIC)* usando simultaneamente os serviços dos elementos *Server1* e *Server2 (Apache ou QUIC Server),* respectivamente, neste uso, dois fluxos concomitantemente configurados com o mesmo protocolo de transporte (QUIC ou TCP) alternavam o uso dos algoritmos de congestionamento BBR ou Cubic. O propósito desse cenário foi observar a atuação de cada implementação de controle de congestionamento em relação a seu semelhante, ou não, em

conjunto com cada protocolo de transporte, também em cenários configurados com os diversos fatores definidos em **Seção 4.3**. As Figura 22 e Figura 23 ilustram os fluxos do cenário 2 com o protocolo TCP e QUIC.

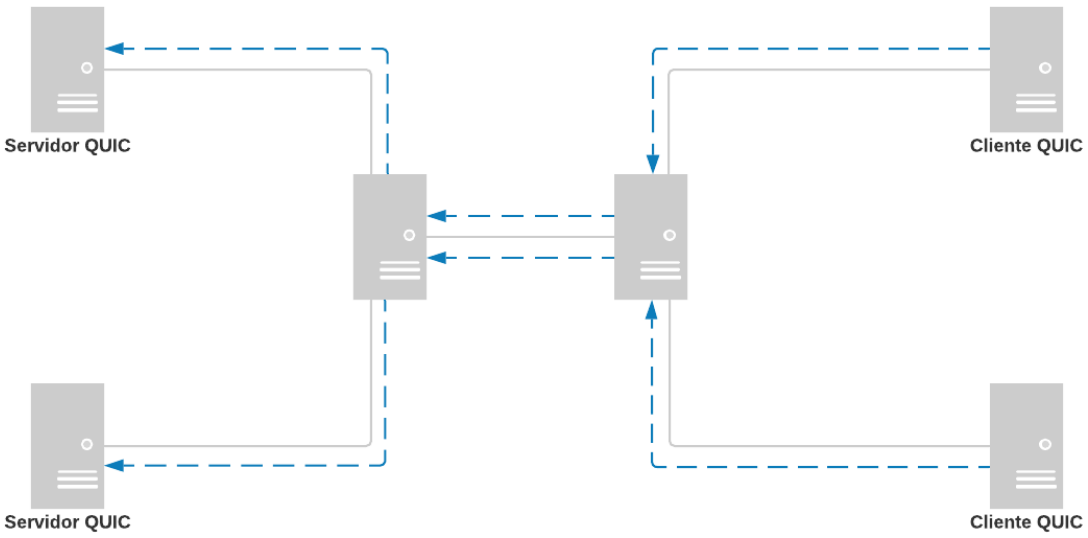

Figura 22 – Cenário 2 com uso do QUIC.

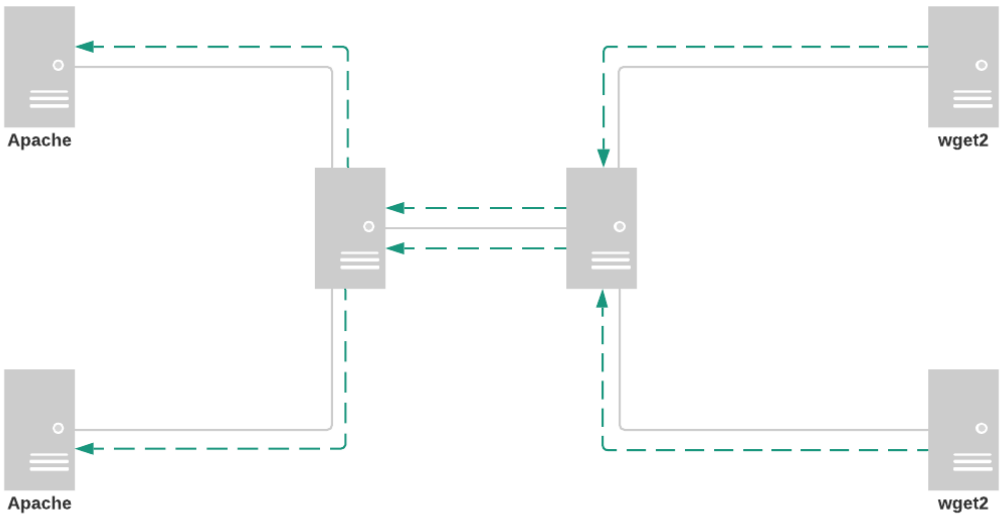

Figura 23 – Cenário 2 com uso do TCP.

4.2.4 Coleta e validação de dados

Para realização do monitoramento dos dados foi usado o *tcpdump*. O tcpdump [52] possibilita a captura e descrição do conteúdo dos pacotes de uma interface de rede com o emprego da biblioteca *libpcap*.

Nesse estudo, todos os cenários foram analisados com as informações obtidas com o uso da ferramenta de monitoramento apresentada. Uma simples explanação do fluxo de transmissão e monitoramento pode ser observado na Figura 24.

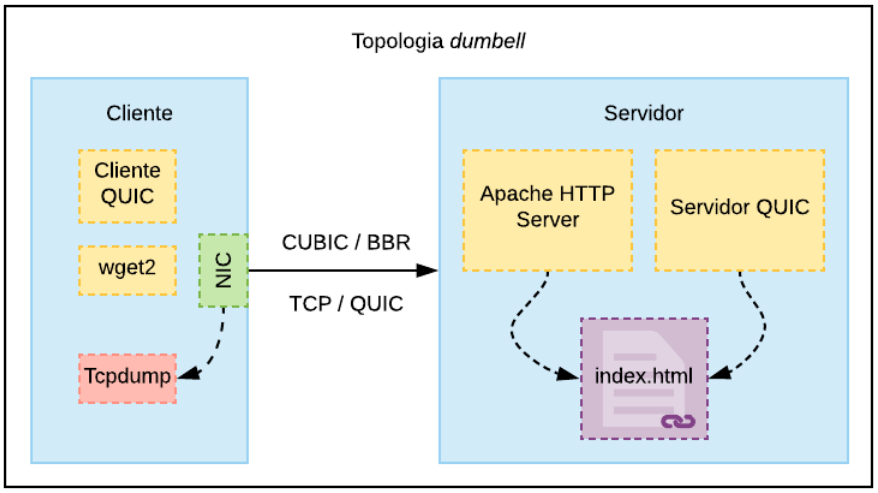

Figura 24 – Modelo de monitoramento do experimento.

Ambos os cenários foram produzidos utilizando os arquivos de teste de tamanho estático (*index.html*), de *2MB* e *32MB*, decorrido dos *headers*[36] da página de exemplo www.example.org e o comando Linux *dd* (disk dump). Este último propiciou o ajuste do tamanho da página utilizada visto que a página base não o tinha inicialmente.

4.3 MÉTRICAS E FATORES

Por meio de pesquisa, levando em consideração principalmente os trabalhos relacionados, foram usados quatro métricas de desempenho:

[36] Cabeçalhos.

1. ***Throughput***[37]: Indica a relação de bits total recebido (com o *overhead*) no(s) destinitário(s) em um tempo.
2. ***Overhead*** [38]: Informações extras em uma transmissão recebidas pelo(s) destinario(s).
3. **Justiça**: Indica se a largura de banda distribuída foi compartilhada com equidade entre os fluxos, considerando fluxos entre algoritmos de congestionamento semelhantes e entre algoritmos de congestionamento distintos. O índice de justiça entre os fluxos foi calculado de acordo com o proposto por Raj Jain [53]. Onde, dado um conjunto de taxas de recebimento *(x1, x2,..., xn),* o índice de justiça pode ser descrito da seguinte maneira na Equação 3:

$$f(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} {x_i}^2}$$

Equação 3 – Equação do Índice de justiça de Raj Jain [53].

O valor retornado da equação indicará o índice de justiça entre os fluxos simultâneos, e, deverá estar sempre entre 0 e 1. Expressando igualdade no compartilhamento dos recursos de rede quanto mais próximo o valor retornado for de 1, e uma distribuição injusta caso seja mais próximo de 0.

4. **Tempo Total de Transmissão**: Tempo decorrido no inicio da transferência até o término em cada transmissão.

Diversos fatores de rede foram utilizados no estudo, a inserção dessas configurações foi possível através de algumas ferramentas auxiliares, a *Network Emulator (netem)*, *Token Bucket Filter (tbf)* e tc. No Linux, a emulação das propriedades de rede foi feita pelo *netem*, já o controle de taxa pela aplicação *tbf,* ambos orquestrados pela ferramenta de linha de comando *t*c [54]. Todas essas ferramentas são disponibilizadas nativamente pela distrubuição Linux utilizada no experimento. Os fatores usados, assim como as métricas de desempenho consideradas, foram determinadas levando em consideração principalmente alguns dos trabalhos apresentados na **Seção 3**, e podem ser visualizadas na Tabela 4. Todas

[37] Taxa de transferência.
[38] Sobrecarga.

as configurações de largura de banda, atraso e perda foram configuradas através das ferramentas citadas anteriormente, e, variadas, apenas entre *Router1 e Router2*, diferentemente dos outros enlaces, justamente configurando o *bottleneck*[39].

Fatores	Configuração
Protocolos	TCP e QUIC
Controle de Congestionamento	Cubic e BBR
Largura de Banda (Mbits)	5, 40 e 100
Atraso (ms)	10, 50 e 100
Taxa de perda de pacotes (%)	0.0, 1.0, e 5.0

Tabela 4 – Parâmetros do experimento.

Para configurar o ambiente com protocolo QUIC foram usados os binários do Servidor e Cliente QUIC obtidos do Chromium [55], assim como Carlucci [3] e Srivastava [4] utilizaram em seus trabalhos.

Já o protocolo TCP foi escolhido a aplicação servidor HTTP Apache. O Apache dispõe de diversos recursos, entre eles, depedendo da versão do mesmo, o suporte ao HTTP/2 [32] [56]. Em relação ao cliente TCP, foi essencial a escolha de uma aplicação que suportasse o protocolo HTTP/2, dessa forma, foi definido a versão mais recente de outra tradicional aplicação do cenário de redes, o wget2 [57]. As instruções para a construção desses elementos (Cliente e Servidor do QUIC e TCP) no Apêndice A.

Outra ferramenta fundamental para o desenvolvimento do estudo foi a *sysctl* [58]. A *sysctl* permite alterar diversos parâmetros do Kernel do Linux em tempo de execução, dessa forma, possibilitou a alteração do algoritmo de congestionamento em cada configuração que abrangia o protocolo TCP.

Na manipulação dos protocolos de congestionamento nos cenários envolvendo o QUIC, como a implementação desses recursos ocorre a nível de usuário, foi necessário indicar esse desejo na construção dos binários do cliente e servidor.

[39] Gargalo.

5 ANÁLISE DOS RESULTADOS

Neste capítulo é feita a análise dos dados coletados durante a pesquisa com a intenção de verificar o comportamento das tecnologias.

5.1 RESULTADOS

Cada experimento foi replicado dez vezes e foi usado a ilustração gráfica para a apresentação dos resultados. Nos experimentos Base, a representação gráfica dos valores obtidos foi feita através da barra de erro, com informações como limite superior e limite inferior. Já no Experimento 2 foi usada uma outra abordagem gráfica mais tradicional.

5.1.1 Experimento Base

5.1.1.1 Throughput

As Figura 25 a Figura 31 apresentam a métrica *Throughput* (Mbps) no cenário denominado Base com a variação do fator perda de pacote (%).

Na Figura 25 observa-se que o QUIC tem melhor desempenho no *Throughput* em relação ao TCP em transferências mais duradouras (*index.html* de 32MB) e cenários configurados com atraso e perda de pacotes.

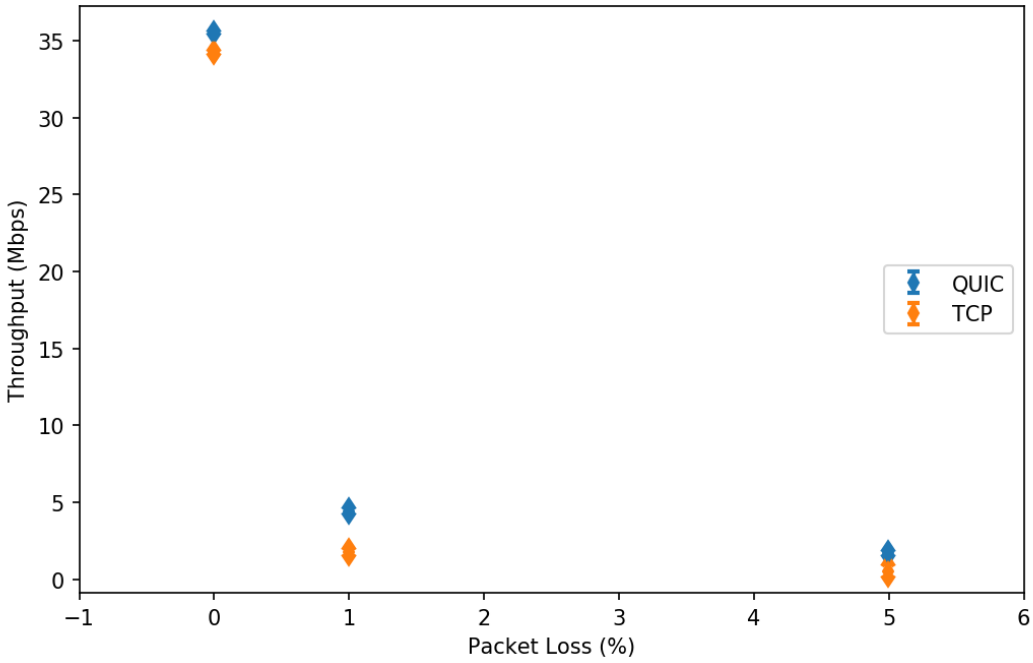

Figura 25 – Comparação de Throughput conforme a variação do fator perda, com Largura de Banda fixa de 40Mbps e atraso de 100ms, utilizando o algoritmo Cubic.

Outro ponto interessante que pode ser notado nas Figura 26 e Figura 27, foi diferença de desempenho do uso do controle de congestionamento BBR em relação ao algoritmo Cubic no QUIC.

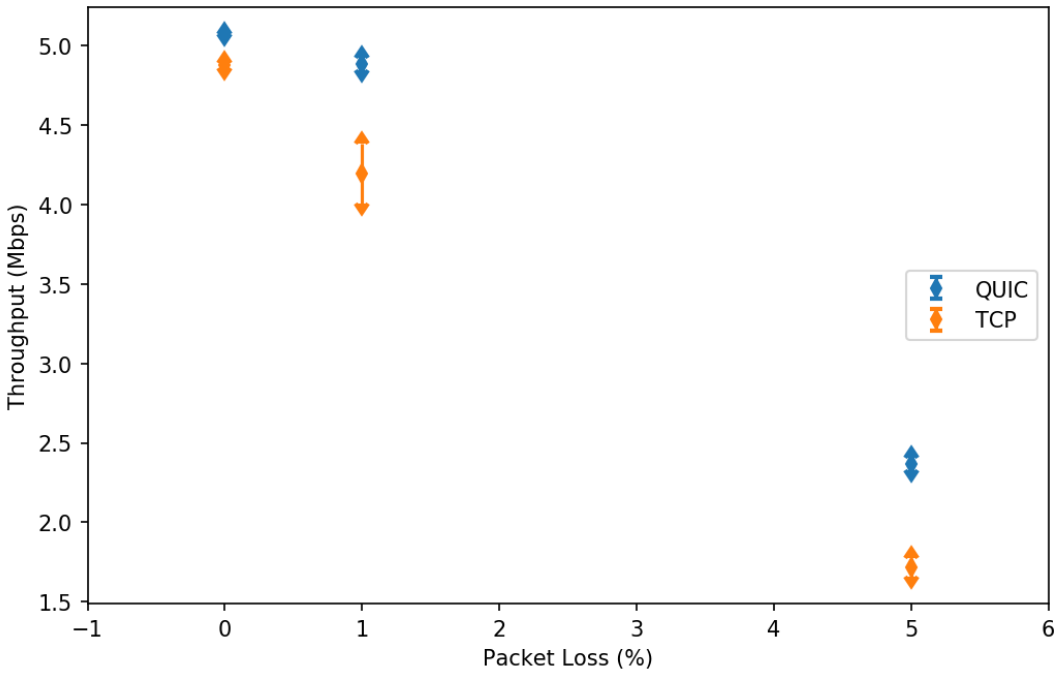

Figura 26 – Comparação de Throughput conforme a variação do fator perda, com Largura de Banda fixa de 5Mbps e atraso de 50ms, utilizando o algoritmo Cubic.

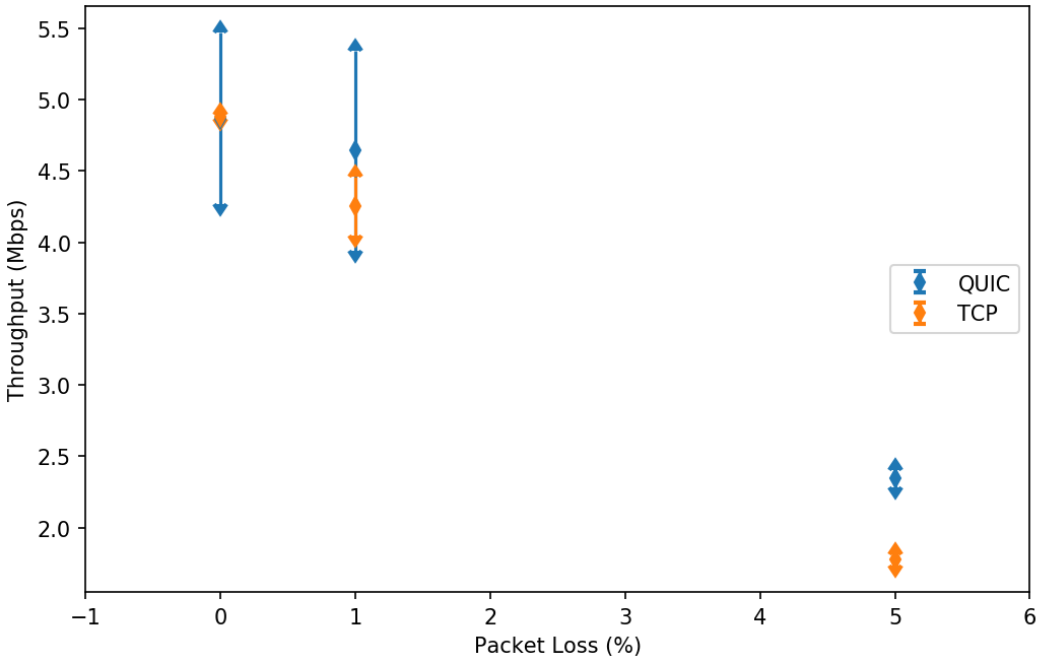

Figura 27 – Comparação de Throughput conforme a variação do fator perda, com Largura de Banda fixa de 5Mbps e atraso de 50ms, utilizando o algoritmo BBR.

Em comunicações mais curtas, em detrimento da transferência de uma quantidade de dados menor (*index.html* de 2MB), os resultados obtidos (Figura 28 e Figura 29) apresentaram bastante semelhança de desempenho entre os protocolos QUIC e TCP, com ambos os algoritmos de congestiomento. Por esse motivo e pelo tamanho da página de 2MB não refletir mais a características das páginas Web no presente [1], os próximos resultados são dos experimentos com o uso do index.html de 32MB.

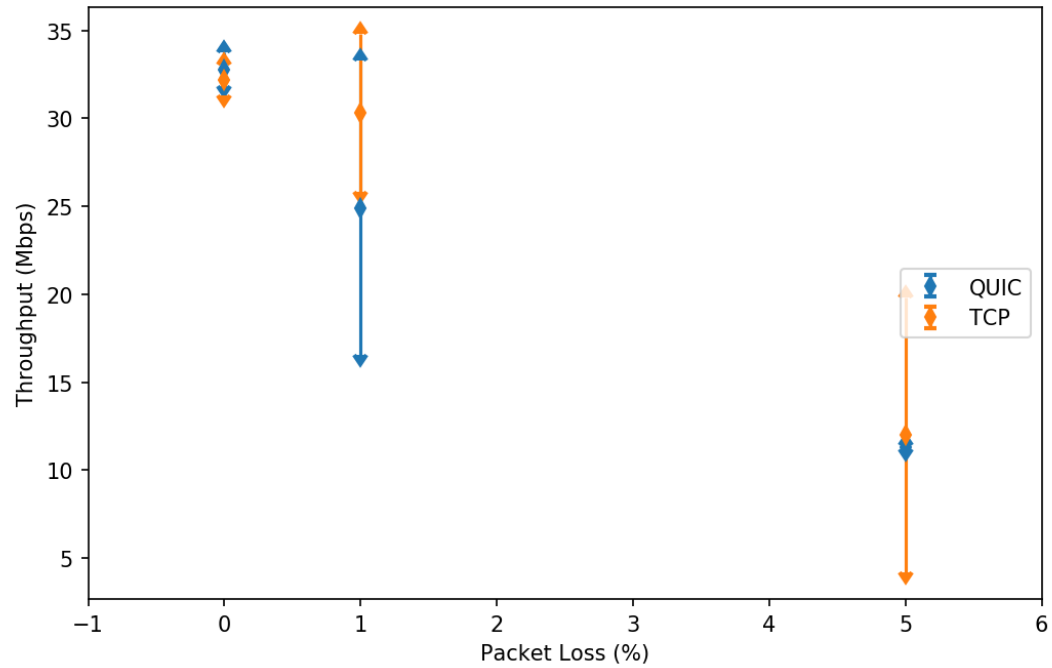

Figura 28 – Comparação de Throughput conforme a variação do fator perda, com Largura de Banda fixa de 40Mbps e atraso de 10ms, utilizando o algoritmo Cubic.

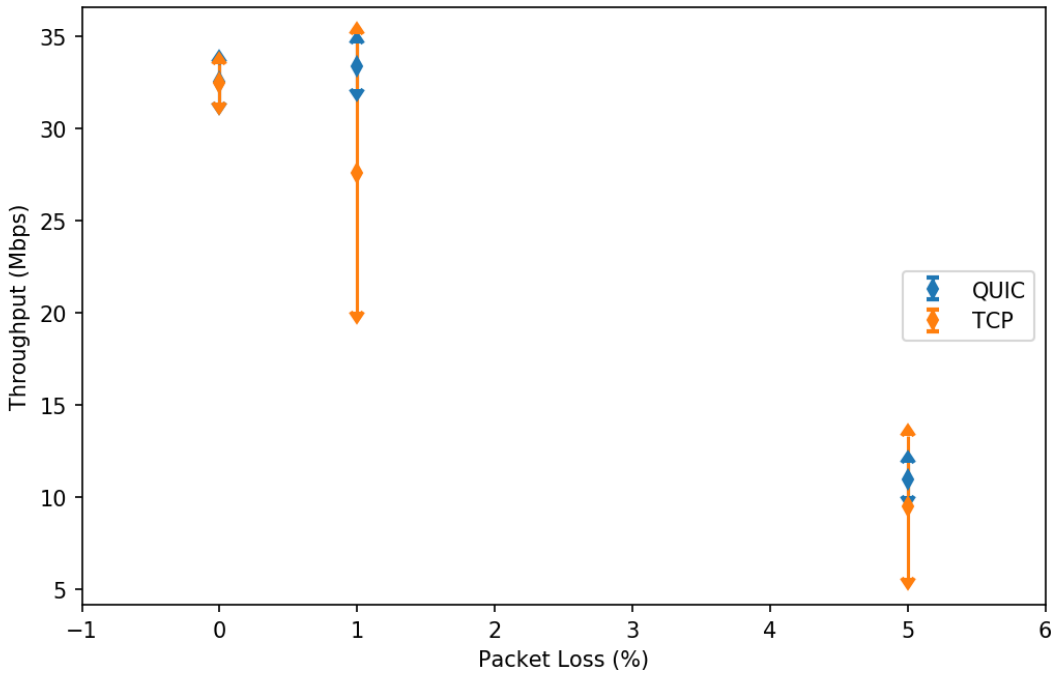

Figura 29 – Comparação de Throughput conforme a variação do fator perda, com Largura de Banda fixa de 40Mbps e atraso de 10ms, utilizando o algoritmo BBR.

Em um cenário de largura de banda maior (100Mbps), Figura 30. Observa-se um comportamento não esperado, considerando que o QUIC deveria ser o protocolo mais rápido. O uso do TCP com o Cubic ou o uso do TCP com o BBR tem um desempenho melhor do que o QUIC e QUIC BBR. O QUIC não conseguiu utilizar os recursos disponíveis com tanta eficiência como o TCP no cenário condicionado por

0% de perda e apenas 10 ms de atraso, ou seja, um contexto classificado bom em redes. Além disso, com a variação do fator de perda (%), ocorre semelhança de desempenho entre os dois protocolos.

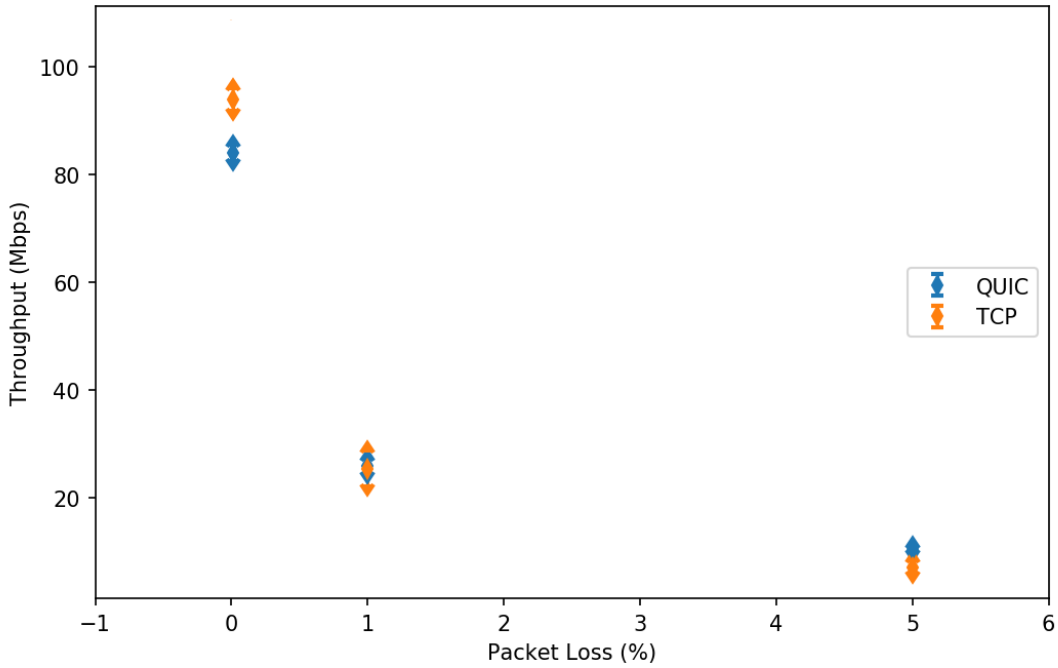

Figura 30 – Comparação de Throughput conforme a variação do fator perda, com Largura de Banda fixa de 100Mbps e atraso de 10ms, utilizando o algoritmo Cubic.

Conforme ocorre a inserção de perda (Figura 31), e também atraso, o QUIC acaba equilibrando e em alguns casos, como tipificado com 50 ms de atraso, ganhando em relação ao *Throughput* alcançado do TCP (Figura 0.1).

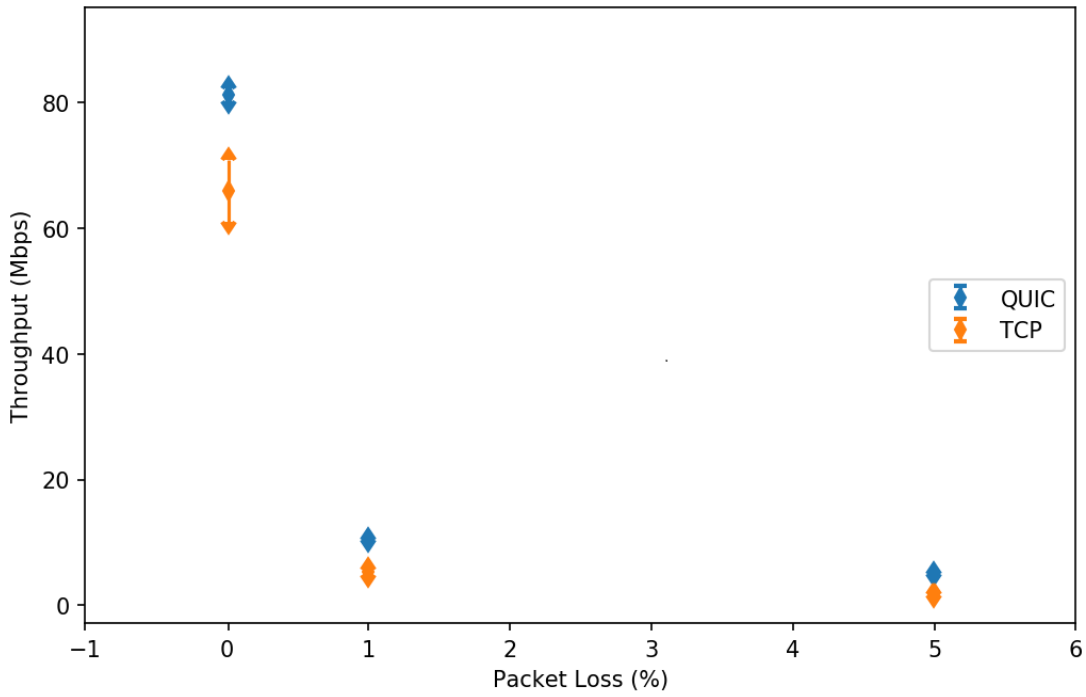

Figura 31 – Comparação de Throughput conforme a variação do fator perda, com Largura de Banda fixa de 100Mbps e atraso de 50ms, utilizando o algoritmo Cubic.

5.1.1.2 Overhead

As Figura 32 a Figura 35 apresentam o *Overhead* (MB) no cenário denominado Base. Nos cenários considerados (com o arquivo index.html de 32MB e com a devida alternância dos algoritmos de congestionamento) o QUIC tem um *ovehead* maior que o TCP na grande maioria dos cenários. É importante destacar que as variações dos três fatores, largura de banda, atraso e perdas de pacotes não interferiram consideravelmente no tamanho do *ovehead* independente do protocolo de transporte ou algoritmo de congestionamento utilizado.

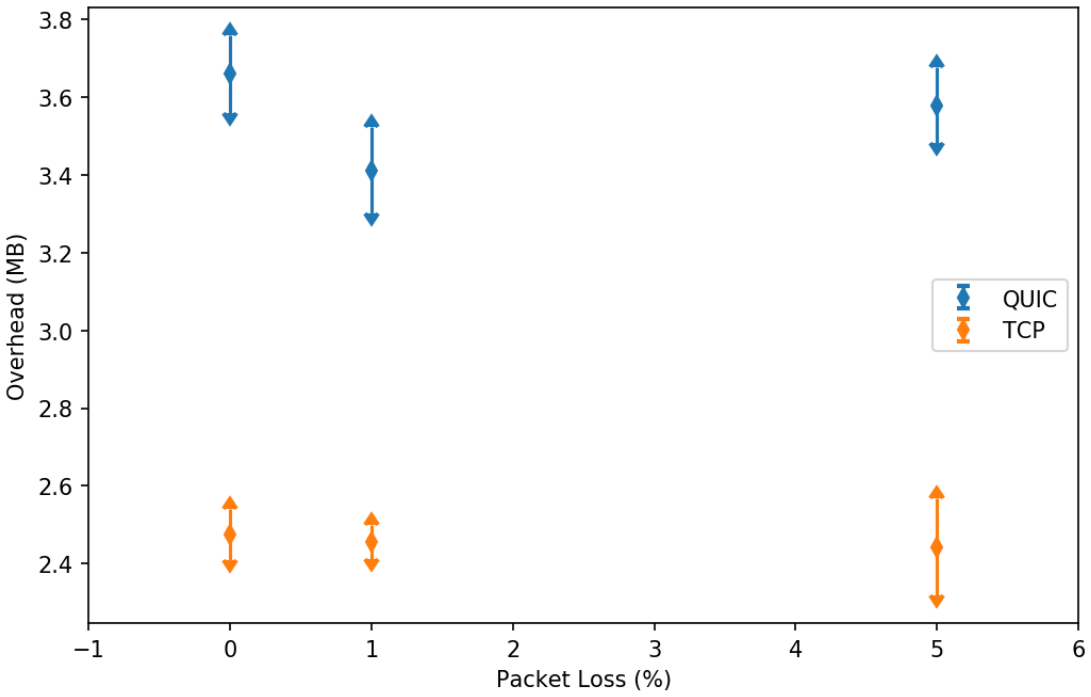

Figura 32 – Comparação do Overhead conforme a variação do fator perda, com Largura de Banda fixa de 5Mbps e atraso de 10ms, utilizando o algoritmo BBR.

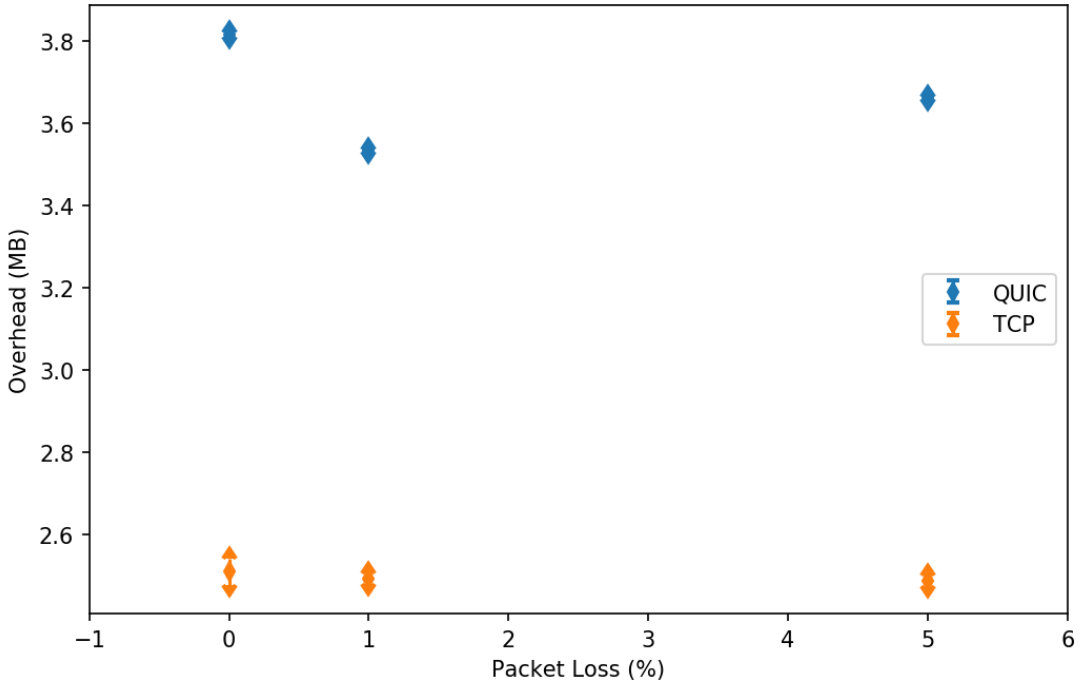

Figura 33 – Comparação do Overhead conforme a variação do fator perda, com Largura de Banda fixa de 5Mbps e atraso de 50ms, utilizando o algoritmo Cubic.

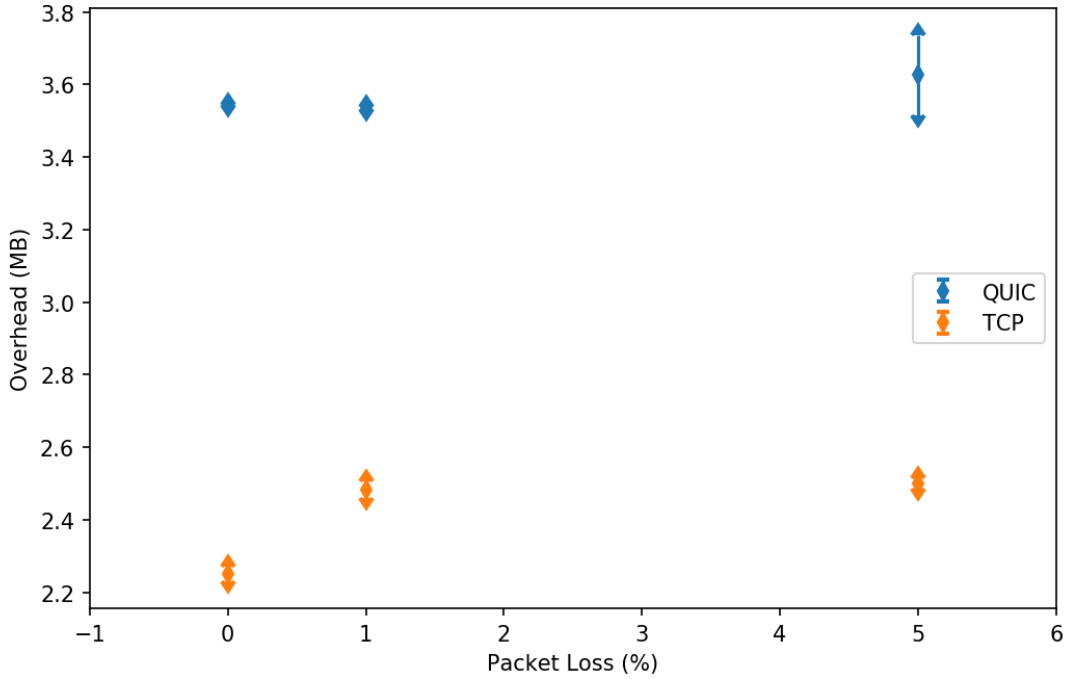

Figura 34 – Comparação do Overhead conforme a variação do fator perda, com Largura de Banda fixa de 40Mbps e atraso de 100ms, utilizando o algoritmo Cubic.

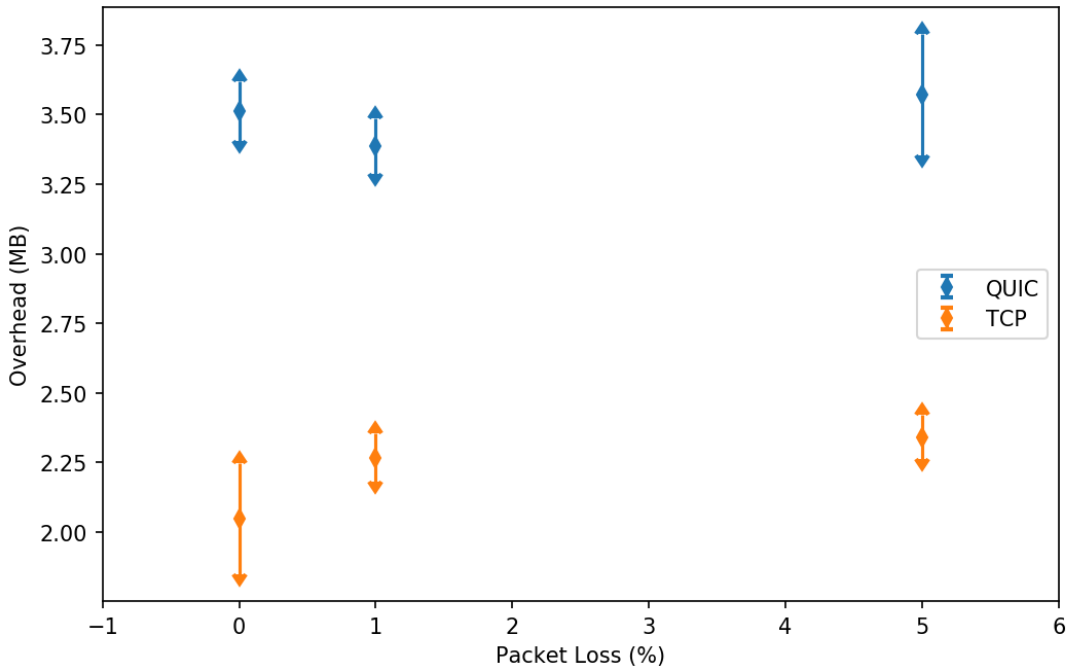

Figura 35 – Comparação do Overhead conforme a variação do fator perda, com Largura de Banda fixa de 40Mbps e atraso de 100ms, utilizando o algoritmo Cubic.

5.1.1.3 Tempo Total de Transmissão

Essa métrica foi vista como de caráter crucial para análise dos valores obtidos das métricas *Throughput* e *overhead* em relação ao QUIC proporcionaram um ganho efetivo de velocidade, como sugere o nome do protocolo.

Nas Figura 36, Figura 37 e Figura 38 são apresentados a métrica Tempo Total de transmissão (s) nos cenários denominado Base. Foram fixados e variados os mesmos fatores empregados nas exposições gráficas das outras métricas.

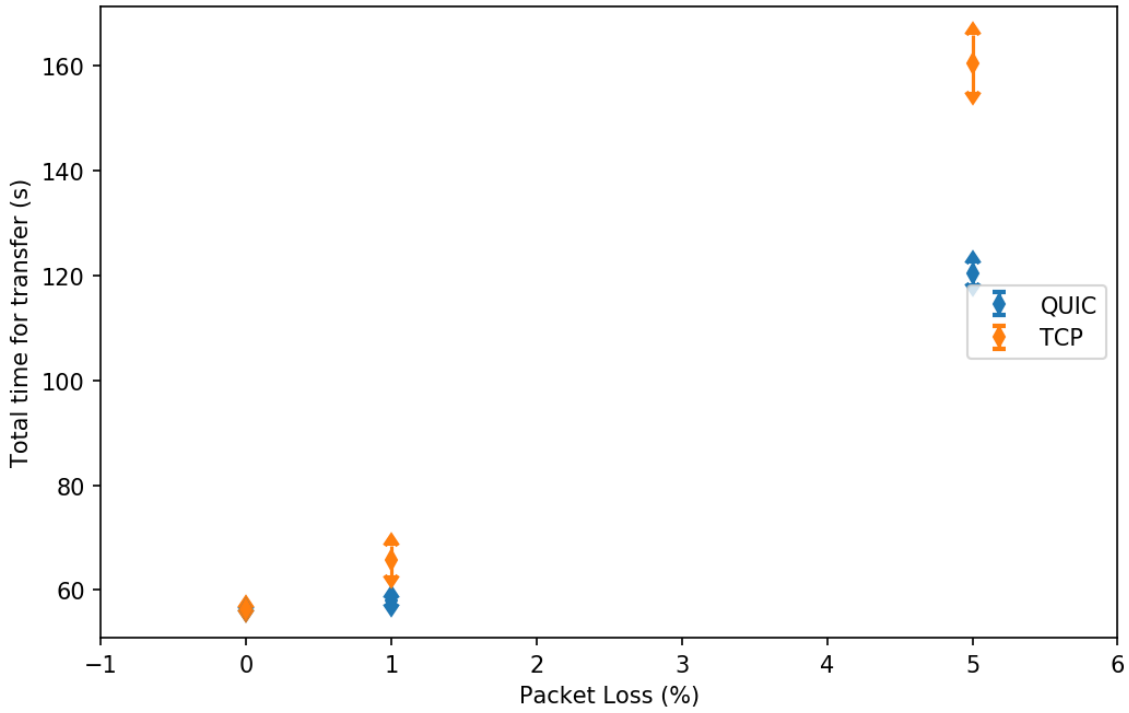

Figura 36 – Comparação do Overhead conforme a variação do fator perda, com Largura de Banda fixa de 5Mbps e atraso de 50ms, utilizando o algoritmo Cubic.

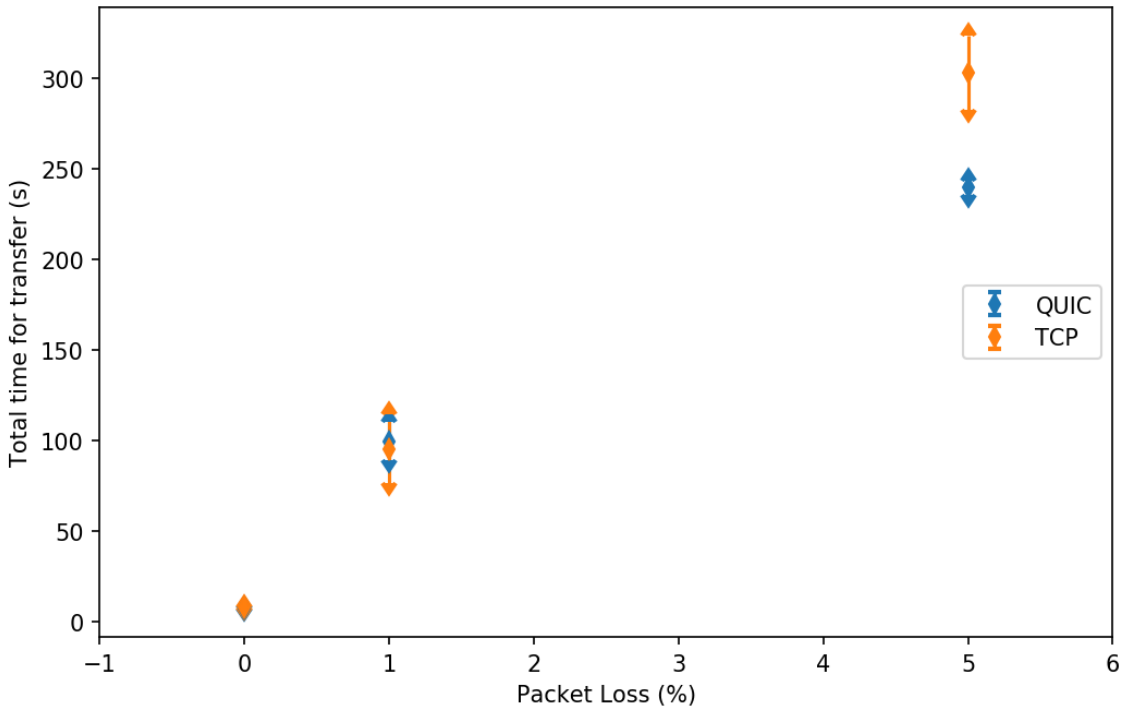

Figura 37 – Comparação do Overhead conforme a variação do fator perda, com Largura de Banda fixa de 40Mbps e atraso de 100ms, utilizando o algoritmo Cubic.

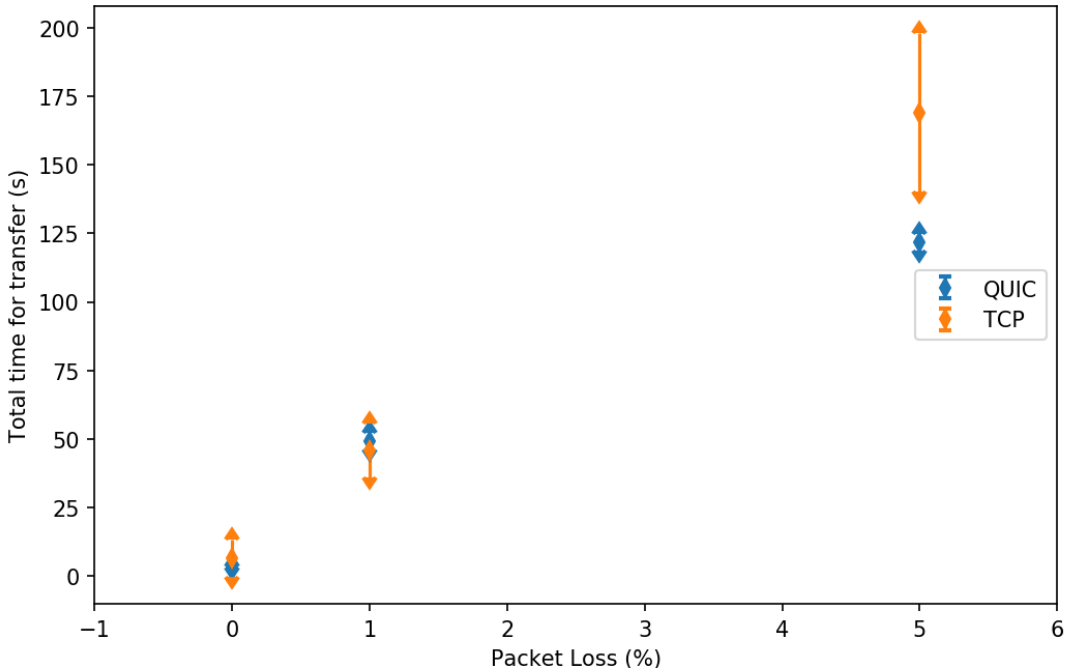

Figura 38 – Comparação do Overhead conforme a variação do fator perda, com Largura de Banda fixa de 100Mbps e atraso de 50ms, utilizando o algoritmo Cubic.

A partir dos resultados apresentado a cima é possível observar que o QUIC proporciona apenas um efetivo ganho de taxa nas transmissões apenas em cenários com atraso e perda de pacotes acentuados, independente da largura de banda disponível.

5.1.2 Experimento *2*

As próximas Figuras apresentam o Índice de justiça no cenário 2. Foi fixado o fator de perda de pacotes (%) em cada gráfico. Foi utilizado apenas o arquivo *index.html* de 32MB. A escolha do uso desse arquivo se deu, pois, o index.html de 32MB acarreta um maior tempo nas transferências e consequentemente, um maior tempo de convivio entre os fluxos, dando mais oportunidade de atuação para os algoritmos de congestionamento e assim mais precisão aos resultados.

5.1.2.1 Justiça

O QUIC tem desempenho melhor e bastante estável nos Índices de justiça calculados através da equação apresentada na **Seção 4.3** em todos os cenários e fatores empregados. Destacando-se principalmente em relação ao TCP, em ambientes onde ambas as métricas, atraso e perda, são expressivas, com é possível observar nas Figura 39, Figura 40, Figura 41 e Figura 42. Nesse contexto, foi notado também a pouca diferença no Índice de justiça do uso BBR em relação ao Cubic no QUIC.

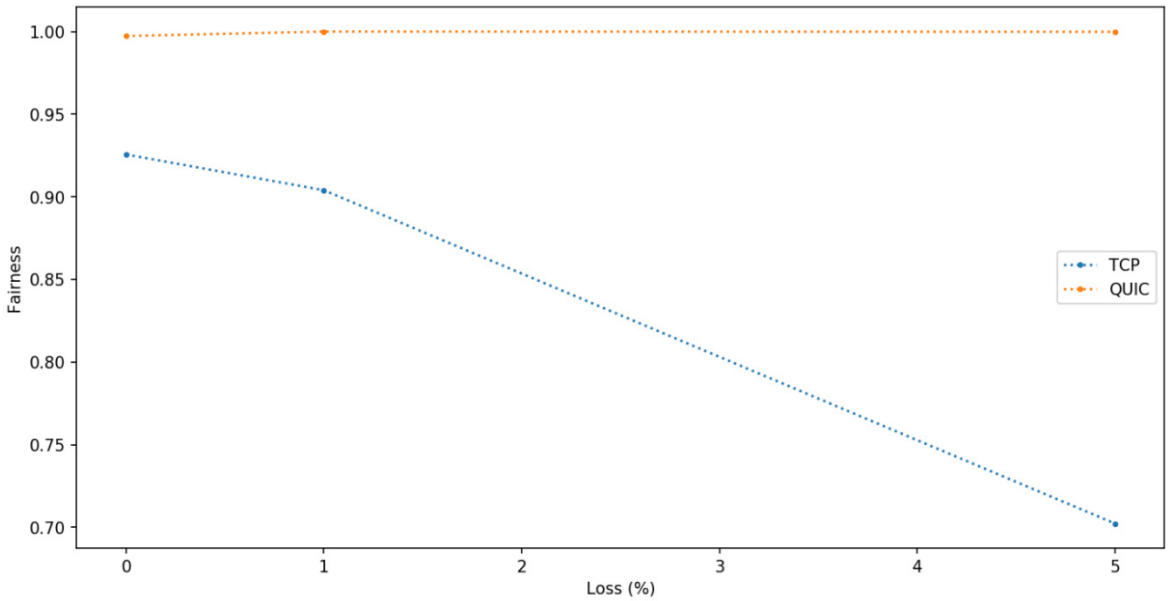

Figura 39 – Índice de justiça conforme a variação do fator perda, com Largura de Banda fixa de 40Mbps e atraso de 10ms, utilizando o algoritmo BBR em ambos os fluxos.

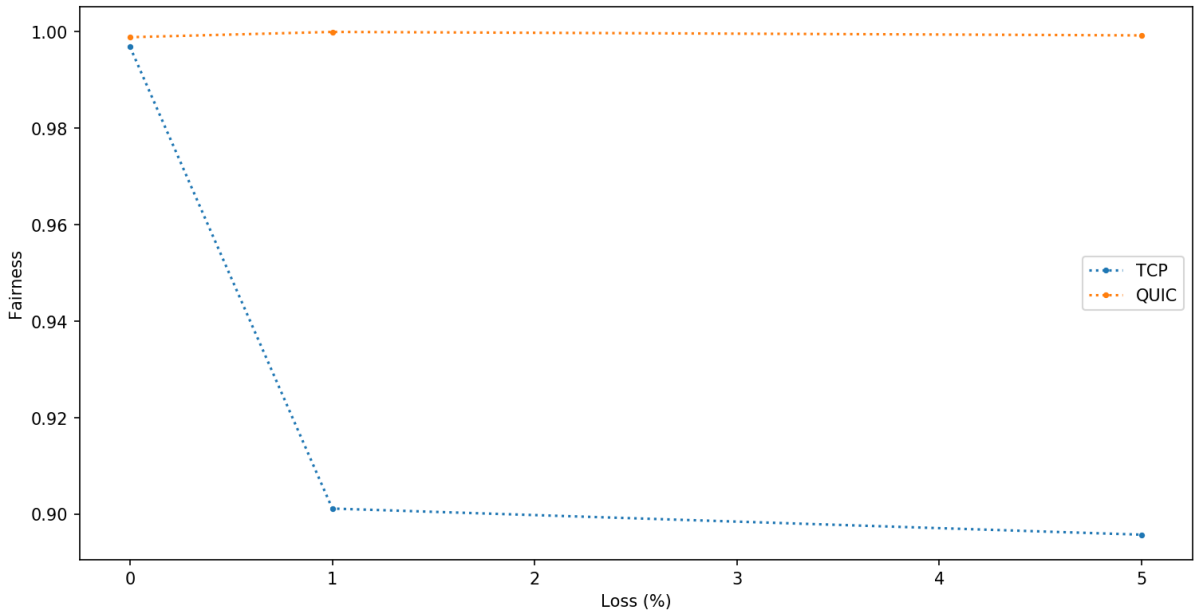

Figura 40 – Índice de justiça conforme a variação do fator perda, com Largura de Banda fixa de 40Mbps e atraso 10ms, utilizando o algoritmo Cubic em ambos os fluxos.

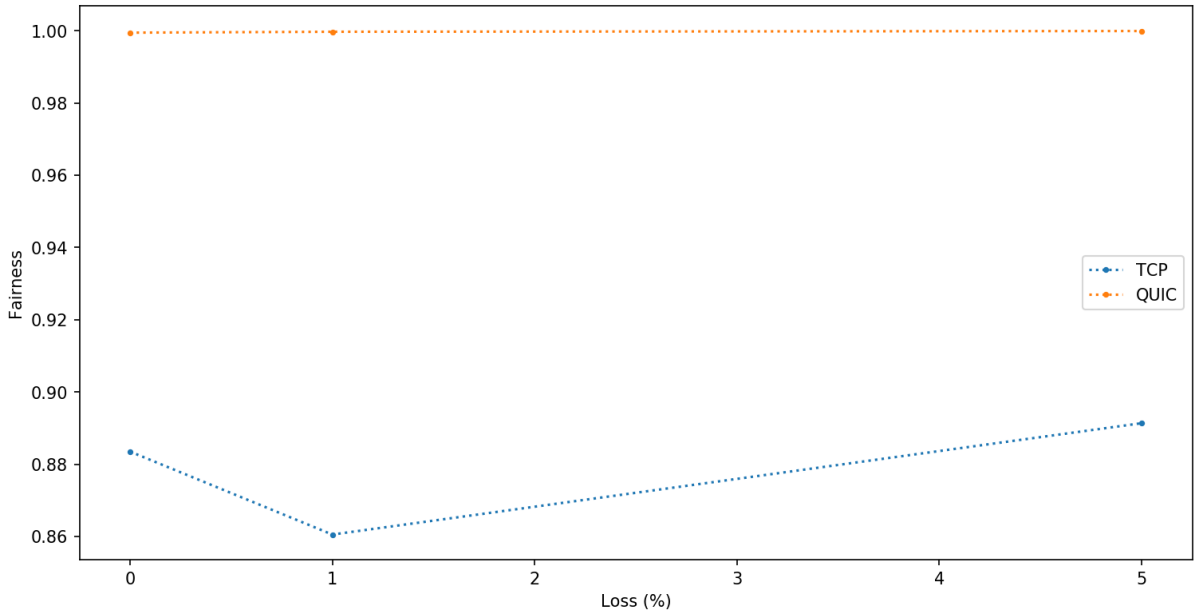

Figura 41 – Índice de justiça conforme a variação do fator perda, com Largura de Banda fixa de 100Mbps e atraso 10ms, utilizando o algoritmo Cubic e BBR em cada um dos fluxos simultaneamente.

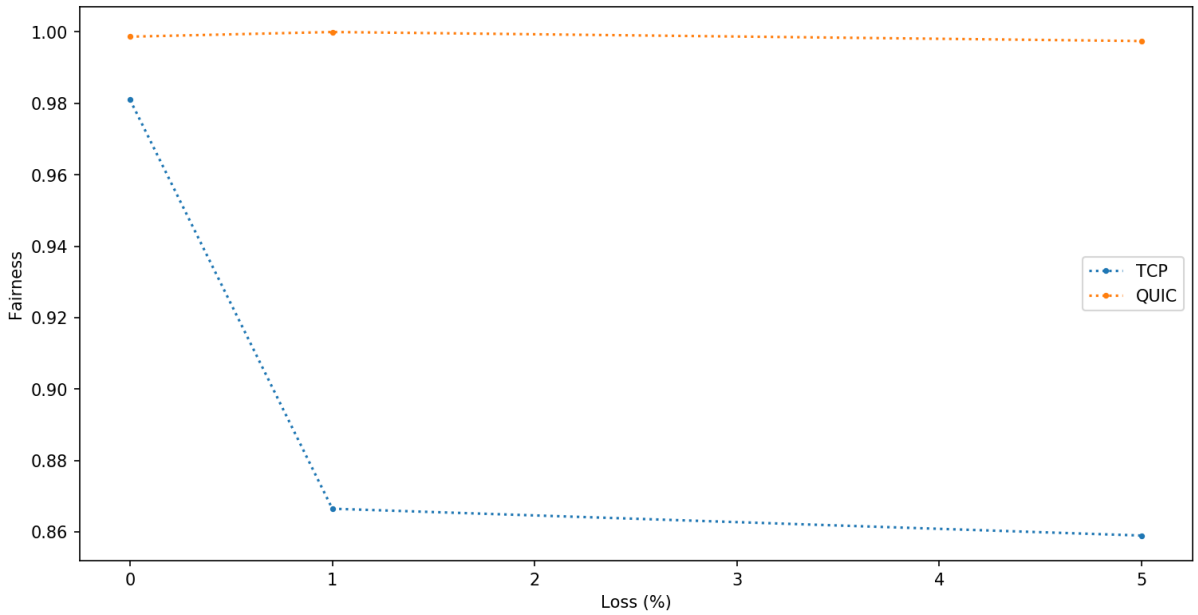

Figura 42 – Índice de justiça conforme a variação do fator perda, com Largura de Banda fixa de 40Mbps e atraso 50ms, utilizando o algoritmo Cubic e BBR em cada um dos fluxos simultaneamente.

5.2 DISCUSSÃO DOS RESULTADOS

Era esperado que o protocolo QUIC efetivamente proporcionasse melhor perfomance em relação ao TCP em todos os cenários possíveis, principalmente devido a divulgação praticada em torno dele especialmente pela empresa criadora da proposta, porém, não foi isso que pôde ser observado. Apesar de proporcionar relativamente uma melhor *Throughput* que o TCP, em boa parte dos cenários principalmente com certa quantidade de atraso (50 e 100) e perda de pacotes (1% e 5%), o custo de *overhead* do protocolo é alto e não propicia um concreto ganho no tempo total das transmissões em boa parcela dos cenários. O QUIC oferece apenas de fato uma melhoria de perfomance, levando em consideração o tempo total de transmissão, principalmente em cenários com grande atraso e perda, possivelmente pelo TCP ser afetado pelo problema do *Head-of-line blocking* (HOL), assim como foi apresentado na pesquisa de Srivastava [4].

Boa parte dos cenários indicaram equidade entre os protocolos analisando o tempo total de transmissão, com exceção dos casos demonstrados acima. Fazendo com que a performance do TCP não seja profundamente afetada e não necessite realmente ser substuído por um protocolo complexo como o QUIC. Essa conclusão diverge parcialmente do trabalho Camarinha [5] na análise de mesma métrical.

Apresentando superioridade do QUIC em todos os cenários possíveis, e não só os indicados no presente trabalho.

O uso do algoritmo de congestionamento BBR não influencia no desempenho significativo do QUIC, levando pequena desvantagem de performance em relação à proposta padrão do QUIC com o Cubic. Esse resultado diverge do trabalho de Wang [47], que mostra justamente o QUIC com o uso do BBR levando vantagem de desempenho em relação a proposta padrão do QUIC, com o Cubic. Porém, esse comportamento obtido evidenciado por esse trabalho era esperado, pois alguns desenvolvedores do Google indicaram que o QUIC BBR não era tão maduro quanto a versão padrão, que foi originalmente desenvolvida para o TCP [59].

Um dos pontos extramementes positivos do QUIC observado foi em relação ao índice de justiça alcançado pelo protocolo. O QUIC proporciona uma partilha dos recursos da rede mais igualitária, e isso pode ser explicado basicamente devido às modificações e adições de mecanismos em sua implementação, como o *Early Retransmit.* Esses resultados estão de acordo com Camarinha [5], que apresenta também o índice de justiça de Raj Jain obtido pelo QUIC. Talvez, se essas alternativas fossem inseridas no TCP o mesmo poderia usufruir do mesmo ou ainda melhorar seu comportamento.

6 CONCLUSÃO E TRABALHOS FUTUROS

O último capítulo deste trabalho apresenta as considerações finais, os principais resultados obtidos, algumas possibilidades para realização de trabalhos futuros e por fim a conclusão final do trabalho.

6.1 RESULTADOS OBTIDOS

Os principais resultados são apresentados abaixo a seguir:

- O uso do algoritmo de congestionamento BBR não influencia no desempenho do QUIC, levando pequena desvantagem de performance em relação à proposta padrão do QUIC com o Cubic.
- Os fluxos de ambos os algoritmos de congestionamento oferecem mais justiça na distribuição dos recursos da rede utilizando o QUIC em todos os cenários.
- Sob as melhores condições de rede, com grande largura de banda, baixo atraso e sem perda, o TCP oferece uma melhor performance em relação ao QUIC.
- Por ter um grande *overhead*, quase sempre o melhor *Throughput* obtido do QUIC não significa transferências mais rápidas em comparação ao uso do TCP.
- Percebe-se, portanto, que o QUIC tem bom desempenho no tempo total de transmissão em relação ao TCP em cenários classificados como não ideias em rede, com alto atraso e percentual de perdas.

De modo que, com sua competência atual, o QUIC pode ser uma alternativa de protocolo de transporte em conjunto com o TCP e UDP. Sendo empregado justamente em redes com alto atraso e com grande porcentual de perda de pacotes.

6.2 TRABALHOS FUTUROS

Buscando aperfeiçoar a análise de desempenho do QUIC e consequentemente esclarecer mais sobre a tecnologia, diversas pesquisas ainda podem ser feitas. Sendo

estas algumas sugestões que podem ser desenvolvidas pelos pesquisadores interessados ao tema, vejamos:

- Analisar o protocolo QUIC com outras propostas de controle de congestionamento suportadas, como o *NewReno* e o BBRv2.
- Adicionar mais fatores e métricas no estudo, exemplo mais porcentual de perdas de pacotes ou atraso.
- Empregar páginas de teste dinâmicas e de tamanho variáveis.
- Desenvolver mais cenários de estudo, como a adição de elementos na topologia *dumbell* buscando esclarecer mais comportamentos do protocolo.
- Após a consolidação das RFCs referentes ao protocolo, desenvolver ou utilizar alguma aplicação que retrate fielmente a formalização e assim desenvolver outros estudos.
- Aplicar o QUIC em redes *Wireless*.

6.3 LIMITAÇÕES DO ESTUDO

A principal limitação deste trabalho foi :

- Apesar da versão do QUIC usada no trabalho, a Q046, ser a mais recente disponibilizada pelo Google para os pesquisadores e a comunidade, essa não é a versão atualizada do protocolo QUIC.

6.4 CONCLUSÕES

O paradigma apresentado pelo Google em 2012, o QUIC, utimamente, vem sendo uma proposta que busca contornar alguns dos problemas decorrentes da grande popularização da Internet, como a alta latência.

Nesse contexto, pesquisadores, e, geralmente profissionais da empresa criadora, tem analisado e empregado progressivamente esse mecanismo. Contudo, há algumas questões que ainda estão em aberto, como a avaliação de desempenho dos elementos que compõem ou interagem diretamente com a tecnologia abordada.

O presente trabalho propôs a avaliação de desempenho de um componente que integra esse protocolo, o algoritmo de controle de congestionamento.

Através do desenvolvimento de um estudo de caso concreto, foi possível produzir um *testbed* e dessa forma realizar a análise de desempenho das tecnologias estudadas no trabalho.

REFERÊNCIAS

[1] "**Report : State of the Web**," httparchive, [Online]. Available: https://httparchive.org/reports/state-of-the-web. [Acesso em 7 Abril 2019].

[2] A. Wilk, R. Hamilton e I. Swett, "**A QUIC update on Google's experimental transport**," Google Inc., 17 Abril 2015. [Online]. Available: https://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html. [Acesso em 27 Dezembro 2019].

[3] G. Carlucci, L. D. Cicco e S. Mascolo, "**HTTP over UDP: an experimental investigation of QUIC**," em *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, Salamanca, 2015.

[4] A. Srivastava, "Worcester Polytechnic Institue - Compute Science - **Performance Evaluation of QUIC Protocol under Network Congestion**," Abril 2017. [Online]. Available: http://web.cs.wpi.edu/~claypool/ms/quic/quic-thesis.pdf. [Acesso em 09 Setembro 2019].

[5] D. d. A. M. Camarinha, "**Análise de desempenho do nnsQUIC: um Módulo para Simulação do protocolo QUIC**," Julho 2018. [Online]. Available: https://www.teses.usp.br/teses/disponiveis/45/45134/tde-16102018-181616/publico/defesa.pdf. [Acesso em 31 12 2019].

[6] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. R. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. L. Kulik, P. Westin e R. Tenneti, "**The QUIC Transport Protocol: Design and Internet-Scale Deployment**," em *SIGCOMM '17: Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, New York, 2017.

[7] P. Megyesi, Z. Krämer e S. Molnár, "**How quick is QUIC?**," em *2016 IEEE International Conference on Communications (ICC)*, Kuala Lumpur, 2016.

[8] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka e T. Turletti, "**A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks**," *IEEE Communications Surveys & Tutorials,* vol. 16, nº 3, pp. 1617-1634, 2014.

[9] V. Jacobson, "**Congestion avoidance and control**," *SIGCOMM '88 Symposium proceedings on Communications architectures and protocols,* vol. 18, nº 4, pp. 314-329, 1988.

[10] A. Forouzan, **Comunicação de Dados e Redes de Computadores** - 4ª Edição, São Paulo: McGraw-Hill, 2008.

[11] W. Boulevard, “**RFC: 791 Internet Protocol**,” IETF, Setembro 1981. [Online]. Available: https://tools.ietf.org/html/rfc791. [Acesso em 20 Maio 2019].

[12] W. Boulevard, “**RFC: 793 Transmission Control Protocol**,” IETF, Setembro 1981. [Online]. Available: https://tools.ietf.org/html/rfc793. [Acesso em 10 Abril 2019].

[13] A. S. Tanenbaum e D. J. Wetherall, **Redes de Computadores - 5ª Edição**, São Paulo: Pearson Prentice Hall, 2011.

[14] J. Nagle, “**On Packet Switches with Infinite Storage**,” *IEEE Transactions on Communications,* vol. 35, nº 4, pp. 435-438, 1987.

[15] W. Stevens, “**RFC: 2001 TCP Slow Start, Congestion Avoidance**,” IETF, Janeiro 1997. [Online]. Available: https://tools.ietf.org/html/rfc2001. [Acesso em 20 Novembro 2019].

[16] M. Allman e V. Paxson, “**RFC: 5681 TCP Congestion Control**,” IETF, Setembro 2009. [Online]. Available: https://tools.ietf.org/html/rfc5681. [Acesso em 15 Novembro 2019].

[17] V. Jacobson, “**Modified TCP Congestion Avoidance Algorithm**,” 10 Novembro 1995. [Online]. Available: ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt. [Acesso em 10 Novembro 2019].

[18] G. Hasegawa, M. Murata e H. Miyahara, “**Fairness and stability of congestion control mechanisms of TCP**,” *Telecommunication Systems,* vol. 15, nº 1-2, pp. 167-184, 2000.

[19] D. M. Chiu e R. Jain, “**Analysis of the increase and decrease algorithms for congestion avoidance in computer networks**,” *Computer Networks and ISDN Systems,* vol. 17, nº 1, pp. 1-14, 1989.

[20] S. Ha, I. Rhee e L. Xu, “**CUBIC: a new TCP-friendly high-speed TCP variant**,” *ACM SIGOPS Operating Systems Review,* vol. 42, nº 5, pp. 64-74, 2008.

[21] L. Xu, K. Harfoush e I. Rhee, “**Binary increase congestion control (BIC) for fast long-distance networks**,” *Proceedings - IEEE INFOCOM,* vol. 4, nº 4, pp. 2514- 2524, 2004.

[22] P. Balasubramanian, “**Updates on Windows TCP**,” 15 Novembro 2017. [Online]. Available: https://datatracker.ietf.org/meeting/100/materials/slides-100-tcpm-updates-on-windows-tcp. [Acesso em 24 Novembro 2019].

[23] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh e V. Jacobson, “**BBR: Congestion-Based Congestion Control**,” *Queue,* vol. 14, nº 5, pp. 50:20--50:53, 2016.

[24] L. Brakmo e L. Peterson, “**TCP Vegas: end to end congestion avoidance on a global Internet**,” *IEEE Journal on Selected Areas in Communications ,* vol. 13, nº 8, pp. 1465 - 1480, 1995.

[25] J. Sing e B. Soh, “**TCP New Vegas: Improving the Performance of TCP Vegas Over High Latency Links**,” *Fourth IEEE International Symposium on Network Computing and Applications,* pp. 73-82, 27-29 Julho 2005.

[26] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, V. Jacobson e A. Vahdat, “**TCP BBR congestion control comes to GCP – your Internet just got faster**,” Google Inc., 20 Julho 2017. [Online]. Available: https://cloud.google.com/blog/products/gcp/tcp-bbr-congestion-control-comes-to-gcp-your-internet-just-got-faster. [Acesso em 28 Novembro 2019].

[27] G. Huston, “**TCP and BBR**,” 16 Maio 2018. [Online]. Available: https://ripe76.ripe.net/presentations/10-2018-05-15-bbr.pdf. [Acesso em 25 Novembro 2019].

[28] M. Hock, R. Bless e M. Zitterbart, “**Experimental evaluation of BBR congestion control**,” em *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, Toronto, 2017.

[29] J. Postel, “**RFC: 768 User Datagram Protocol**,” IETF, 28 Agosto 1980. [Online]. Available: https://tools.ietf.org/html/rfc768. [Acesso em 2018 Abril 20].

[30] SANDVINE, “**Resources - All of our latest materials available at your fingertips**,” Outubro 2018. [Online]. Available: https://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report.pdf. [Acesso em 11 Abril 2019].

[31] R. Fielding, U. Irvine, J. Gettys, Compaq/W3C, J. Mogul, Compaq, H. Frystyk, W3C/MIT, L. Masinter, Xerox, P. Leach, Microsoft, T. Berners-Lee e W3C/MIT, “**RFC: 2616 Hypertext Transfer Protocol -- HTTP/1.1**,” IETF, Junho 1999. [Online]. Available: https://tools.ietf.org/html/rfc2616. [Acesso em 20 Outubro 2019].

[32] “**APACHE HTTP SERVER PROJECT**,” The Apache Software Foundation, [Online]. Available: https://httpd.apache.org/ABOUT_APACHE.html. [Acesso em 10 Outubro 2019].

[33] E. R. Fielding, Adobe, E. J. Reschke e greenbytes, “**RFC: 7230 Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing**,” IETF, Junho 2014. [Online]. Available: https://tools.ietf.org/html/rfc7230#section-6.3.2. [Acesso em 20 Dezembro 2019].

[34] "**SPDY: An experimental protocol for a faster web**," Google Inc., 2009. [Online]. Available: https://dev.chromium.org/spdy/spdy-whitepaper. [Acesso em 15 Outubro 2019].

[35] G. Mineki, S. Uemura e T. Hasegawa, "**SPDY accelerator for improving Web access speed**," em *2013 15th International Conference on Advanced Communications Technology (ICACT)*, PyeongChang, 2013.

[36] D. Synodinos, "**HTTP 2.0 First Draft Published**," InfoQ, 30 Novembro 2012. [Online]. Available: https://www.infoq.com/news/2012/11/http20-first-draft/. [Acesso em 21 Novembro 2019].

[37] M. Belshe, BitGo, R. Peon, I. Google, E. M. Thomson e Mozilla, "**RFC: 7540 Hypertext Transfer Protocol Version 2 (HTTP/2)**," IETF, Maio 2015. [Online]. Available: https://tools.ietf.org/html/rfc7540. [Acesso em 21 Outubro 2019].

[38] R. Pepper, "**Reduzindo as barreiras digitais: Índice de Internet Inclusiva de 2019**," Facebook, 25 Fevereiro 2019. [Online]. Available: https://br.newsroom.fb.com/news/2019/02/reduzindo-as-barreiras-digitais-indice-de-internet-inclusiva-de-2019/. [Acesso em 7 Abril 2019].

[39] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador e Maurício, "**OpenFlow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes**," *Cad. CPqD Tecnologia,* vol. 7, nº 1, pp. 65-76, 2011.

[40] "**QUIC, a multiplexed stream transport over UDP**," Google, [Online]. Available: https://www.chromium.org/quic. [Acesso em 20 Junho 2019].

[41] J. Iyengar, "**QUIC, a multiplexed stream transport over UDP**," [Online]. Available: https://docs.google.com/presentation/d/15e1bLKYeN56GL1oTJSF9OZiUsI-rcxisLo9dEyDkWQs/edit#slide=id.g99041b54d_0_0. [Acesso em 20 Julho 2019].

[42] A. Langley e W.-T. Chang, "**QUIC Crypto**," Google Inc., 2016. [Online]. Available: https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/edit. [Acesso em 20 Dezembro 2019].

[43] Y. Cui, T. Li, C. Liu, X. Wang e M. Kühlewind, "**Innovating Transport with QUIC: Design Approaches and Research Challenges**," *IEEE Internet Computing,* vol. 21, nº 2, pp. 72 - 76, 2017.

[44] E. M. Thomson, Mozilla, E. S. Turner e sn3rd, "**Using TLS to Secure QUIC**," IETF, Abril 2019. [Online]. Available: https://tools.ietf.org/html/draft-ietf-quic-tls-20. [Acesso em 25 Dezembro 2019].

[45] I. Grigorik, "**Making the web faster with HTTP 2.0**," *Commun. ACM,* vol. 56, nº 12, pp. 42-49, 2013.

[46] E. V. Roca, INRIA, F. Michel, UCLouvain, I. Swett, Google, M.-J. Montpetit e T. Video, “**Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) draft-roca-nwcrg-rlc-fec-scheme-for-quic-02**,” IETF, 3 Novembro 2019. [Online]. Available: https://tools.ietf.org/html/draft-roca-nwcrg-rlc-fec-scheme-for-quic-02. [Acesso em 24 Dezembro 2019].

[47] Y. Wang, K. Zhao, W. Li, J. Fraire e Z. Sun, “**Performance Evaluation of QUIC with BBR in Satellite Internet,”** ***2018 6th IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE),*** pp. 195-199, 11-13 Dezembro 2018.

[48] E. J. Iyengar, Fastly, E. I. Swett e Google, “**QUIC Loss Detection and Congestion Control draft-ietf-quic-recovery-16**,” IETF, 23 Outubro 2018. [Online]. Available: https://tools.ietf.org/html/draft-ietf-quic-recovery-16#section-4.2.1. [Acesso em 25 Dezembro 2019].

[49] G. Appenzeller e N. M. Isaac Keslassy, “**Sizing router buffers**,” em *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communication*, Portland, 2004.

[50] K. Nepomuceno, I. N. d. Oliveira, R. R. Aschoff, D. Bezerra, M. S. Ito, W. Melo, D. Sadok e G. Szabó, “**QUIC and TCP: A Performance Evaluation**,” em *2018 IEEE Symposium on Computers and Communications (ISCC)*, Natal, 2018.

[51] P. Wang, C. Bianco, J. Riihijärvi e M. Petrova, “**Implementation and Performance Evaluation of the QUIC Protocol in Linux Kerne**l,” *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems,* pp. 227-234, 28-2 Outubro-Novembro 2018.

[52] V. Jacobson, C. Leres, Steven McCanne, a. o. t. L. B. N. Laboratory e tcpdump.org, “**Manpage of TCPDUMP**,” tcpdump.org, 2 Abril 2019. [Online]. Available: https://www.tcpdump.org/manpages/tcpdump.1.html. [Acesso em 20 Dezembro 2019].

[53] R. Jain, **Art of Computer Systems Performance Analysis Techniques For Experimental Design**, Wiley, 1991.

[54] “**networking:netem**,” Linux Foundation Wiki., 14 Setembro 2018. [Online]. Available: https://wiki.linuxfoundation.org/networking/netem. [Acesso em 10 Novembro 2019].

[55] Google, “**depot_tools(7) Manual Page**,” Google Inc., 03 Maio 2019. [Online]. Available: https://commondatastorage.googleapis.com/chrome-infra-docs/flat/depot_tools/docs/html/depot_tools.html. [Acesso em 29 Julho 2019].

[56] “**May 2019 Web Server Survey**,” Netcraft, 10 Maio 2019. [Online]. Available: https://news.netcraft.com/archives/2019/05/10/may-2019-web-server-survey.html. [Acesso em 20 Dezembro 2019].

[57] T. Rühsen, “**Wget2 Introduction**,” 6 Novembro 2019. [Online]. Available: https://gitlab.com/gnuwget/wget2. [Acesso em 26 Dezembro 2019].

[58] G. Staikos, “**sysctl(8) - Linux man page**,” die.net, [Online]. Available: https://linux.die.net/man/8/sysctl. [Acesso em 10 Dezembro 2019].

[59] “**How to select BBR as the congestion control in QUIC code?**,” Google Inc., 13 Janeiro 2017. [Online]. Available: https://groups.google.com/a/chromium.org/forum/#!topic/proto-quic/g27OSwdBeqM. [Acesso em 15 Dezembro 2019].

APÊNDICE A – CONSTRUÇÃO DOS ELEMENTOS DO TESTBED

Servidor QUIC

Incialmente foi necessário obter o suíte de scripts denominado *depot_tools*. Essa ferramenta tem a finalidade de gerenciamento de versionamento das aplicações do navegador Chromium. Além disso, foi configurado o binário do *depot_tools* na variável de ambiente, *$PATH*, dos sistemas operacionas das respectivas máquinas do cenário de experimento.

```
$  git clone https://chromium.googlesource.com/chromium/tools/depot_tools.git

$ export PATH="$PATH: ${HOME} /depot_tools"
```

Depois de configurar o *depot_tools*, se tornou viável adquirir o código fonte do Chormium, assim como todas as depêndencias adicionais necessárias para uma eventual compilação do navegador ou de ferramentas relacionadas.

```
$ mkdir ~/chromium && cd ~/chromium && fetch --nohooks chromium && cd src && \
./src/build/install-build-deps.sh
```

Após os passos relatados a cima, a compilação da binário do servidor QUIC pôde ser concluída com a utilização do sistema de meta-construção GN e compilação Ninja .

```
$ gclient runhooks && gn gen out/Default && autoninja -C out/Default quic_server
net_unittests
```

O último passo para execução do servidor foi a necessidade de gerar um certificado e chave privada no formato *pkcs8*. O certificado foi obtido através de um *script* denominado *generate-certs.sh,* localizado no repositório do Chormium. Além da

criação do certificado válido do servidor e chave pública, esse *script* teve a funcionalidade de gerar um *Certification Authority*[40] (CA). Mais adiante, esse certificado foi registrado e validado na raiz do sistema pela ferramenta *libnns3-tools*.

```
$ cd /home/usr/chromium/src/net/tools/quic/certs && /generate-certs.sh

$ sudo apt-get install libnss3-tools && cd /home/usr/chromim/src/ && certutil -d \
sql:$HOME/.pki/nssdb -A -t "C,," -n QUIC-CA -i net/tools/quic/certs/out/2048-sha256-
root.pem
```

Servidor TCP

A instalação da ferramenta foi concebida através da aplicação de administração de pacotes Linux, *Advanced Packaging Tool* (APT).

```
$ sudo apt-get install apache2
```

Assim como no servidor QUIC, foi preciso gerar um certificado autoassinado, buscando para segurança das conexões do Apache, através da aplicação *OpenSSL.* Além disso, foi preciso configurar a aplicação para usar esse recurso.

```
$ openssl genrsa -out server.key 1024 && openssl req -new -key server.key -out \
server.csr && openssl x509 -req -days 365 -in server.csr -signkey server.key -out \
server.crt && sudo a2enmod ssl && sudo a2ensite default-ssl.conf
```

O Apache não vem configurado nativamente com o HTTP/2. Por isso, foram feitas algumas modificações. Inicialmente foi necessário ativar os módulos, *http2, proxy_fcgi, setenvif* e mpm_event além da configuração *php7,0-fpm*, e desativar os módulos *mpm_prefork* e *php7.0*, visto que este não faz o Apache funcionar apropriadamente com o HTTP/2. Além disso, foi acrescentado a linha "*Protocols h2 http/1.1*" no arquivo de configuração do Apache (default-ssl.conf).

[40] Autoridade de certificação.

```
$ sudo apt-get install php7.0-fpm && sudo a2enmod proxy_fcgi setenvif && sudo a2enconf php7.0-fpm && sudo a2dismod php7.0 && sudo a2dismod mpm_prefork && sudo a2enmod mpm_event.
```

Cliente QUIC

O cliente QUIC também foi obtido através do projeto Chormium, sendo necessário a execução dos mesmos estágios preliminares do seu Servidor QUIC, mas divergindo deste através do ajuste do comando *ninja* com o interesse de sua sua compilação.

O binário referente ao uso da implementação do cliente QUIC padrão, ou seja, com o QUIC Cubic foi feito com o seguinte comando:

```
$ gn gen out/Default && ninja -C out/Default quic_client
```

A construção do Cliente QUIC BBR se deu através da modificação da variável booleana de *FLAGS_quic_reloadable_flag_quic_default_to_bbr* para *True*, localizada em um dos arquivos utilizado pelo *Ninja* para construção do Cliente, o */home/usr/chromium/src/net/quic/quic_flags_list.h*. A construção de um novo servidor QUIC não é necessária, visto que quando o cliente envia essa opção de conexão, TBBR, o servidor o imita.

```
$ gn gen out/Default2 && ninja -C out/Default2 quic_client
```

Cliente TCP

O wget2 foi adquirido por meio de compilação de código fonte, visto que a versão oferecida pelo repositório do Ubuntu *bionic* no período do estudo apresentava *bugs*.

```
$ wget https://gnuwget.gitlab.io/wget2/wget2-latest.tar.gz  && tar xf wget2-latest.tar.gz && cd /home/usr/wget2-1.99.2/ && ./configure && make && make check && sudo make install.
```

APÊNDICE B – PRINCIPAIS CÓDIGOS DO TESTBED

```
#!/bin/bash

#base

base='BW'

#dev_r1 -> r2

dev='eno1'

#dev client H1

devH1='eno1'

#dev client H2

devH2='enp0s25'



#IPs

h1r1='10.10.10.1'

h2r1='10.10.20.1'

h1r2='10.10.30.1'

h2r2='10.10.40.1'

r1='150.161.192.2'

r2='150.161.192.1'

N=10


for n in $(seq 1 $N)

do

echo "Starting test #$n ..."

 for delay in '10' '50' '100'
```

```
do
 for loss in '0' '1' '5'
 do
  for algor1 in bbr cubic
  do
   for bw in '100' '40' '5'
   do
    echo "Configuring netem in router with following commands:"
    ./traficControl.sh $dev $loss $delay $bw
    algor2=$algor1
    sleep 1
    echo "Running tests for enviroment: packetloss=$loss, delay=$delay, bandwidth=$bw, $algor1 vs $algor2"
     for transport in 'tcp' 'quic'
     do
     echo "Capture started"
     cat startPCAP.sh | ssh ${h1r2} bash -s - $devH1 $transport &
     pid1=$!
     sleep 1
     if [ $transport == 'tcp' ]
     then
            echo "Configuring congestion protocols"
            cat cfg_cong_ctl | ssh ${h1r1} sudo bash -s - $algor1
            cat cfg_cong_ctl | ssh ${h1r2} sudo bash -s - $algor1
      echo "Generating test file..."
```

```
            arquivo1=${base}_${n}_${transport}_h1s_${delay}_${loss}_${bw}_${algor1}_vrs_${al
gor2}

       echo "Files test created."

       ssh -tt ${h1r2} "sudo wget2 -O ./tmp/index.html https://${h1r1}/index.html --http2-only"

       echo -e "\n"

      else

       if [ $algor1 == 'bbr' ]

       then

        echo "Generating test file..."

               arquivo1=${base}_${n}_${transport}_h1s_${delay}_${loss}_${bw}_${algor1}_vrs_
${algor2}

        echo "Files test created."

        ssh -tt ${h1r2} "sudo /home/usr/chromium/src/out/Default2/quic_client --host=${h1r1} --
port=6121 https://www.example.org/ > ./tmp/download"

        echo -e "\n"

       else

        echo "Generating test file..."

               arquivo1=${base}_${n}_${transport}_h1s_${delay}_${loss}_${bw}_${algor1}_vrs_
${algor2}

        echo "Files test created."

        ssh -tt ${h1r2} "sudo /home/usr/chromium/src/out/Default/quic_client --host=${h1r1} --
port=6121 https://www.example.org/ > ./tmp/download "

        echo -e "\n"

       fi

      fi



      echo "Analyzing pcap file..."

      ssh -tt ${h1r2} "sudo killall tcpdump"
```

```
      ssh -tt ${h1r2} "sudo tcpdump -r /tmp/test.pcap -tttttnnqv > ./raw/${arquivo1} 2>/dev/null"

      sleep 1

      ssh -tt ${h1r2} "sudo rm -f /tmp/test.pcap"

      echo "Clean history cache"

      ssh -tt ${h1r2} "sudo rm -f ./tmp/download"

      ssh -tt ${h1r2} "sudo rm -f ./tmp/index.html"

      echo "Test finished protocol=$transport, packetloss=$loss, delay=$delay, bandwidth=$bw,
$algor1 vs $algor2, output in $arquivo1"

      sleep 1

      echo -e "\n"

      done

     done

    done

   done

  done

done
```

```
#!/bin/bash

#2Flow

base='BW'

#dev_r1 -> r2

dev='eno1'
```

```
#dev client H1

devH1='eno1'

#dev client H2

devH2='enp0s25'


#IPs

h1r1='10.10.10.1'

h2r1='10.10.20.1'

h1r2='10.10.30.1'

h2r2='10.10.40.1'

r1='150.161.192.2'

r2='150.161.192.1'

N=10


for n in $(seq 1 $N)
 do
 echo "Starting test #$n ..."
  for delay in '10' '50' '100'
  do
   for loss in '0' '1' '5'
    do
    for algor1 in bbr cubic
    do
     for algor2 in bbr cubic
     do
```

```
    for bw in '100' '40' '5'
    do
     echo "Configuring netem in router with following commands:"
     ./traficControl.sh $dev $loss $delay $bw
     sleep 1
     echo "Running tests for enviroment: packetloss=$loss, delay=$delay, bandwidth=$bw,
$algor1 vs $algor2"
      for transport in 'tcp' 'quic'
      do
       echo "Capture started"
       cat startPCAP.sh | ssh ${h1r2} bash -s - $devH1 $transport &
       pid1=$!
       cat startPCAP.sh | ssh ${h2r2} bash -s - $devH2 $transport &
       pid2=$!
            sleep 1
       if [ $transport == 'tcp' ]
       then
        echo  "Configuring congestion protocols"
        cat cfg_cong_ctl | ssh ${h1r1} sudo bash -s - $algor1
        cat cfg_cong_ctl | ssh ${h1r2} sudo bash -s - $algor1
        echo -e "\n"
        cat cfg_cong_ctl | ssh ${h2r1} sudo bash -s - $algor2
        cat cfg_cong_ctl | ssh ${h2r2} sudo bash -s - $algor2
        echo -e "\n"
        echo "Generating test file..."
```

```
                arquivo1=${base}_${n}_${transport}_h1s_${delay}_${loss}_${bw}_${algor1}_vrs
_${algor2}

        echo "Generating test file 2..."

                arquivo2=${base}_${n}_${transport}_h2s_${delay}_${loss}_${bw}_${algor2}_vrs
_${algor1}

        echo "Files test created."

        echo "Starting test..."

        #-P = dirty screen

        #-t 0, by default pssh use 60s, set 0 disable timeout. Fix error Time out, killed by signal
9

        parallel-ssh -H "${h1r2} ${h2r2}" -t 0 -I < sourceTcp.sh

      else#QUIC

       sleep 1

             echo "Generating test file..."

                arquivo1=${base}_${n}_${transport}_h1s_${delay}_${loss}_${bw}_${algor1}_vrs
_${algor2}

             echo "Generating test file 2..."

                arquivo2=${base}_${n}_${transport}_h2s_${delay}_${loss}_${bw}_${algor2}_vrs
_${algor1}

              echo "Files test created."

         echo "Starting test..."

              if [ $algor1 == 'bbr' ] && [ $algor2 == "bbr" ]

          then

           parallel-ssh -H "${h1r2} ${h2r2}" -t 0 -I < sourceQuicBbrBbr.sh

          elif [ $algor1 == 'bbr' ] && [ $algor2 == "cubic" ]

          then

           parallel-ssh -H "${h1r2} ${h2r2}" -t 0 -I < sourceQuicBbrCubic.sh

          elif [ $algor1 == 'cubic' ] && [ $algor2 == "bbr" ]
```

```
          then
          parallel-ssh -H "${h1r2} ${h2r2}" -t 0 -I < sourceQuicCubicBbr.sh
          else
          parallel-ssh -H "${h1r2} ${h2r2}" -t 0 -I < sourceQuicCubicCubic.sh
          fi
        fi
        echo "Kill TCPdump..."
        parallel-ssh -H "${h1r2} ${h2r2}" -P sudo killall tcpdump
        echo "Analyzing pcap file..."
        ssh -tt ${h1r2} "tcpdump -r /tmp/test.pcap -tttttnnqv > ./raw/${arquivo1}
2>/dev/null"
        ssh -tt ${h2r2} "tcpdump -r /tmp/test.pcap -tttttnnqv > ./raw/${arquivo2}
2>/dev/null"
        sleep 1
        echo "Remove old pcap file..."
        parallel-ssh -H "${h1r2} ${h2r2}" -P sudo rm -f /tmp/test.pcap

       echo "Remove cache file..."
       parallel-ssh -H "${h1r2} ${h2r2}" -P sudo rm -f ./tmp/download
       parallel-ssh -H "${h1r2} ${h2r2}" -P sudo rm -f ./tmp/index.html
       echo "Test finished for: protocol=$transport, packetloss=$loss, delay=$delay,
bandwidth=$bw, in output files $arquivo1 and $arquivo2."
      sleep 1
     done
    done
   done
```

```
  done
 done
done
done


done
```

APÊNDICE C – CÓDIGOS AUXILIARES DO TESTBED

```
# File startPCAP.sh

sudo touch /tmp/test.pcap

if [ $2 == 'tcp' ]

then

        sudo tcpdump -i $1 port 443 -w /tmp/test.pcap

else

        sudo tcpdump -i $1 port 6121 -w /tmp/test.pcap

fi
```

```
# File traficControl.sh

dev=$1

loss=$2

delay=$3

bw=$4


sudo tc qdisc del dev $dev root

sudo tc qdisc add dev $dev root handle 1: netem delay ${delay}ms loss ${loss}%

sudo tc qdisc add dev $dev parent 1: handle 2: tbf rate ${bw}mbit mtu 1500

echo "sudo tc qdisc del dev $dev root"

echo "sudo tc qdisc add dev $dev root handle 1: netem delay ${delay}ms loss ${loss}%"

echo "sudo tc qdisc add dev $dev parent 1: handle 2: tbf rate ${bw}mbit mtu 1500"
```

```
# File sourceQuicCubicCubic.s
```

```
# The files source sourceQuicBbrCubic.sh, sourceQuicBbrBbbr.sh sourceQuicCubicBbbr.sh are very similar, only with the correct QUIC client setting.

#!/bin/bash

h1r1='10.10.10.1'

h2r1='10.10.20.1'

if [ "$HOSTNAME" = h1r2 ]; then

    sudo /home/usr/chromium/src/out/Default/quic_client --host=${h1r1} --port=6121 https://www.example.org/ > ./tmp/download

else

    sudo /home/usr/chromium/src/out/Default/quic_client --host=${h2r1} --port=6121 https://www.example.org/ > ./tmp/download

fi

exit 0
```

```
# File sourceTCP.sh

h1r1='10.10.10.1'

h2r1='10.10.20.1'

if [ "$HOSTNAME" = h1r2 ]; then

    sudo wget2 -O ./tmp/index.html https://${h1r1}/index.html --http2-only

else

    sudo wget2 -O ./tmp/index.html https://${h2r1}/index.html --http2-only

fi
```

```
exit 0

done
```

APÊNDICE D – CONFIGURAÇÃO DE REDE DO EXPERIMENTO

```
# Router2 <--> Router1
auto eno1
iface eno1 inet static
        address 150.161.192.2
        netmask 255.255.255.0
        gateway 150.161.192.1
        up ip route add 10.10.30.0/24 via 150.161.192.1
        up ip route add 10.10.40.0/24 via 150.161.192.1


#Server1 <-->Router1
auto enp1s0
iface enp1s0 inet static
        address 10.10.10.254
        netmask 255.255.255.0
        network 10.10.10.0
        broadcast 10.10.10.255


#Server2 <-->Router1
auto enp2s0
iface enp2s0 inet static
        address 10.10.20.254
        netmask 255.255.255.0
        network 10.10.20.0
```

```
        broadcast 10.10.20.255


#Router1 <--> Router2
auto enp0s25
iface enp0s25 inet static
        address 150.161.192.1
        netmask 255.255.255.0
        network 150.161.192.0
        broadcast 150.161.192.255
        up ip route add 10.10.10.0/24 via 150.161.192.2
        up ip route add 10.10.20.0/24 via 150.161.192.2
#Client1 <-->Router1
auto enp2s0
iface enp2s0 inet static
        address 10.10.30.254
        netmask 255.255.255.0
        network 10.10.30.0
        broadcast 10.10.30.255



# Client2 <-->Router1
auto enp3s0
iface enp3s0 inet static
        address 10.10.40.254
```

```
        netmask 255.255.255.0
        network 10.10.40.0
        broadcast 10.10.40.255
```

```
# Router1 <--> Server1
auto enp0s25
iface enp0s25 inet static
        address 10.10.10.1
        gateway 10.10.10.254
        netmask 255.255.255.0
        network 10.10.10.0
         broadcast 10.10.10.255
```

```
# Router1 <--> Server2
auto eno1
iface eno1 inet static
        address 10.10.20.1
        gateway 10.10.20.254
        netmask 255.255.255.0
        network 10.10.20.0
         broadcast 10.10.20.255
```

```
# Router1 <--> Client1
auto eno1
iface eno1 inet static
```

```
        address 10.10.30.1
        gateway 10.10.30.254
        netmask 255.255.255.0
        network 10.10.30.0
         broadcast 10.10.30.255
```

```
# Router1 <--> Client2
auto enp0s25
iface enp0s25 inet static
        address 10.10.40.1
        gateway 10.10.40.254
        netmask 255.255.255.0
        network 10.10.40.0
         broadcast 10.10.40.255
```