GERADOR DE TRÁFEGO HTTP PARA ESTUDOS EM AMBIENTES DE SIMULAÇÃO

Jordan Guimarães Mattos[1], **Saulo Henrique da Mata**[2]

Resumo: As redes de computadores têm se destacado cada vez mais como um setor fundamental em diversos aspectos da sociedade contemporânea e, com sua popularização, a Internet tornou-se parte do cotidiano das pessoas, na qual, muitos processos de conversão de serviços para o espaço digital foram acelerados, ocorrendo mudança de escala no uso das redes, como, o aumento do tráfego e mudanças drásticas nos serviços oferecidos ao usuário. A navegação web já foi considerada o tráfego dominante na Internet e as aplicações de vídeo têm projeção de representar 82% de todo o tráfego da Internet em 2022. Estas estatísticas posicionam o tráfego HTTP como um importante elemento no cenário de geradores de tráfego. Dessa forma, o projeto tem como objetivo desenvolver um gerador de tráfego para o protocolo HTTP, que permita tanto o estudo do protocolo em questão, como a utilização em estudos de outros aspectos de uma rede de comunicação, atuando como tráfego de entrada para outros tipos de estudos como, por exemplo, a alocação de recursos.
Palavras-chave: Gerador de tráfego. Ns-3. Simulação de redes. Tráfego HTTP.

HTTP Traffic Generator for Studies in Simulation Environments

Abstract: Computer networks have increasingly stood out as a fundamental sector in various aspects of society. With its popularization the Internet has become part of people's daily lives, in which many services' conversions to the digital space were accelerated, evident by the rising use of these networks, such as the increase in traffic and drastic changes in the services offered to the users. Web browsing was once considered the dominant Internet traffic and video applications are projected to reach over 82% of all Internet traffic in 2022. These statistics places HTTP traffic as an important element in the traffic generator scenario. Thus, the project aims to develop a traffic generator for the HTTP protocol, which allows both the study of the protocol in question, and its uses in studies of other aspects of communication network, acting as input traffic for other types of studies such as resources allocation.
Keywords: Traffic generator. Ns-3. Network simulation. HTTP traffic.

[1]Estudante de Ciência da Computação, IFTM, Campus Ituiutaba, jordan.mattos@estudante.iftm.edu.br
[2]Professor Orientador, IFTM, Campus Ituiutaba, saulodamata@iftm.edu.br

1 INTRODUÇÃO

As redes de computadores têm se destacado cada vez mais como um setor fundamental em diversos aspectos da sociedade contemporânea. Seja no trabalho, no estudo, ou entretenimento, a conexão com a Internet se tornou parte do cotidiano das pessoas. Com a pandemia causada pelo Covid-19, esta constatação se tornou ainda mais avassaladora e muitos processos de conversão de serviços para o espaço digital foram acelerados (LAVADO, 2020) (JOHNSON, 2021).

Esta mudança de escala no uso das redes, seja no aumento do tráfego, bem como nas diferentes formas que as aplicações oferecem serviços para os usuários, traz consigo grandes desafios para a infraestrutura de rede e os atuais protocolos de comunicação.

As redes de comunicação por si só são ambientes complexos, frutos de um grande esforço de engenharia, controle e padrões estabelecidos. Nesta dinâmica de constante crescimento e oferta de novas aplicações, a inovação, revisão e proposição de novas soluções para os desafios das comunicações é um fator primordial para o sucesso deste sistema. Por outro lado, a concepção de elementos inovadores em um sistema tão grande e complexo pode ser algo impraticável quando consideramos aspectos de custos e tempo investidos nas pesquisas.

Neste contexto, os estudos realizados a partir de simuladores de rede oferecem uma boa contrapartida entre custos, tempo e resultados obtidos. O desenvolvimento de modelos matemáticos e simuladores para sistemas reais coloca-se como um importante campo de pesquisa. Simuladores desenvolvidos pela comunidade de pesquisadores, com a premissa do código aberto, apresentam características muito atrativas para o desenvolvimento de novas pesquisas, aprimoramento de protocolos e melhoria das atuais redes de comunicação.

Nesse sentido, o presente projeto de pesquisa propõe o estudo para a concepção de um gerador de tráfego para fluxos de dados característicos do protocolo HTTP (*Hyper-Text Transfer Protocol*).

2 REFERENCIAL TEÓRICO

Geradores de tráfego são um assunto recorrente na área de redes de computadores (CHENG; ÇETINKAYA; STERBENZ, 2013). Dada a complexidade das redes, o ambiente de simulação torna-se uma alternativa muito atrativa para os pesquisadores da área. Ao realizar uma proposição de inovação em algum aspecto da rede, é comum que os pesquisadores demandem um

gerador de tráfego adequado para a realização dos testes de avaliação da proposta.

O estudo do funcionamento do protocolo HTTP é bem amparado por diversos autores, sendo um dos mais clássicos (KUROSE; ROSS, 2013). De acordo com o modelo OSI/ISO, o HTTP é um protocolo da camada de aplicação que utiliza o protocolo TCP (*Transmission Control Protocol*) na camada de transporte. Também conhecido como protocolo web, o HTTP é o protocolo utilizado pelos usuários para acessar as páginas dos inúmeros sites disponíveis na Internet.

Para além da abordagem didática dos livros, o protocolo HTTP é detalhado nos documentos RFCs (*Request For Comments*), publicados pela Força Tarefa de Engenharia da Internet, o IETF (*Internet Engineering Task Force*) (IETF, 2014). A partir dos RFCs, o desenvolvedor pode conhecer os detalhes do comportamento do protocolo, assegurando compatibilidade com outros protocolos e elementos da rede.

A dinâmica do protocolo HTTP também pode ser entendida a partir dos aspectos de implementação deste protocolo e sua relação com os conceitos de programação para redes (do inglês, *Network Programming*). Nesta vertente, podemos destacar autores como (RHODES; GOERZEN, 2014) para a programação com Python e (HALL, 2019) para a programação em linguagem C.

2.1 MODELOS MATEMÁTICOS

Para realizar a implementação de um módulo gerador de tráfego, é necessário estabelecer um modelo matemático que descreva, de forma simplificada, o comportamento do tráfego em um sistema real. Neste contexto, podemos destacar dois cenários de comportamento para o protocolo HTTP: (i) tráfego de páginas web e (ii) tráfego de vídeo.

Páginas web são compostas por um objeto principal (*main object*) e diversos objetos secundários (*inline objects*). Quando é utilizado para a transmissão de páginas web, o protocolo HTTP apresenta um tráfego caracterizado por rajadas (*burst*), isto é, uma aplicação do tipo Liga/Desliga (*ON/OFF*), onde os objetos são enviados (intervalo ON) e existe um tempo de leitura do usuário para aquele conteúdo enviado (intervalo OFF). A Figura 1 ilustra um esquema desse modelo de tráfego.

Inicialmente, o usuário faz a requisição de uma página web, por meio de um cliente web, geralmente um navegador. Passa-se então a uma fase em que o usuário visualiza os itens enquanto eles estão sendo transferidos do servidor web para o cliente (HTTP ON). Ao término

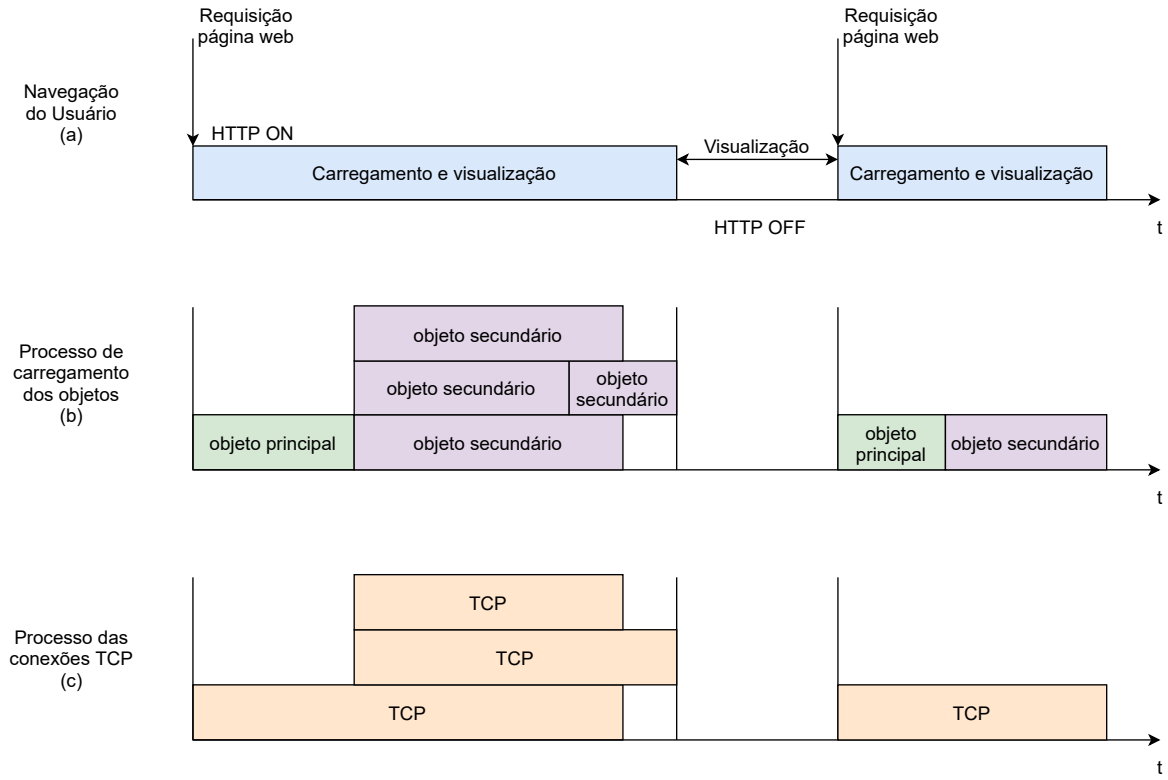


Figura 1: Processos HTTP (Adaptado de (PRIES; MAGYARI; TRAN-GIA, 2012)).

da transmissão (HTTP OFF), o usuário continua visualizando o conteúdo, até que realiza uma nova requisição para outros elementos web.

Do ponto de vista do processo de carregamento dos objetos, a requisição da página web se inicia com o objeto principal, seguido dos respectivos elementos secundários que compõe a página web. Por fim, sob a ótica do protocolo TCP, podem ser abertas sessões paralelas para acelerar as transferências dos diversos objetos secundários.

Este comportamento em rajadas é descrito em diversos trabalhos na literatura, confirmando ainda a propriedade de escala desse efeito, ou seja, o comportamento se mantêm em um curto ou longo intervalo de análise, o que é denominado auto-similaridade (*self-similarity*) (CROVELLA; BESTAVROS, 1997).

Na literatura, este comportamento pode ser modelado sob diferentes perspectivas (CHENG; ÇETINKAYA; STERBENZ, 2013): baseado na página (*page-based*), baseado no comportamento (*behavior-based*) e baseado na conexão (*conection-based*).

Modelos baseados na página, se preocupam apenas com a estrutura da página, sem considerar aspectos como atrasos dos servidores ou latência das requisições. Já os modelos baseados em comportamento, consideram o tráfego como uma aplicação ON/OFF. Por fim, o modelo baseado em conexões, modela o comportamento do tráfego HTTP considerando as características

das sessões TCP, tais como taxa de estabelecimento de conexões, tamanho das requisições e respostas, intervalo entre os objetos, dentre outros.

Em (CHENG; ÇETINKAYA; STERBENZ, 2013), Cheng, Çetinkaya e Sterbenz afirmam que o modelo baseado em conexões é a melhor alternativa. Por outro lado, o modelo utilizado por estes autores é baseado em medições realizadas no ano de 1997, e que já não correspondem a realidade da web atual.

Pries, Magyari e Tran-Gia apresentam em (PRIES; MAGYARI; TRAN-GIA, 2012) um modelo baseado em comportamento que, além de considerar o tráfego como uma aplicação ON/OFF, também define um intervalo de leitura para o usuário. Além disso, os autores analisaram as característica da estrutura de 1 milhão de páginas web ranqueadas como as mais acessadas no mundo. A partir dessas características, um modelo que detalha o tamanho dos objetos principais, tamanho dos objetos secundários e tempo de leitura do usuário foi consolidado. Este modelo corrige os problemas do modelo baseado em conexões, citados pelos autores em (CHENG; ÇETINKAYA; STERBENZ, 2013), enquanto se aproxima do modelo baseado em conexões, apresentando dados mais atualizados, coletados em meados do ano de 2010.

Vale ressaltar que mesmo o modelo apresentado em (PRIES; MAGYARI; TRAN-GIA, 2012), já apresenta uma defasagem de mais de 10 anos para a web atual. Contudo, não foram encontrados trabalhos com a mesma relevância e contemporaneidade que pudessem substituir o modelo proposto por Pries, Magyari e Tran-Gia.

Acredita-se que essa lacuna de trabalhos foi ocasionada pela evolução do tráfego HTTP nos últimos anos. Passando de um tráfego caracterizado por transmissões de páginas web em rajadas, hoje temos um tráfego HTTP dominado pelo conteúdo de vídeo, que por sua vez é caracterizado por uma alta demanda de recursos da rede e fluxo contínuo de dados em uma dada taxa de transmissão. Somado a este fato, é possível identificar também um maior uso da rede em dispositivos móveis (Cisco, 2018) (G1, 2016). Portanto, a relevância desse novo comportamento pode ser observada na lacuna de trabalhos avaliando apenas o trafego web clássico e a pluralidade de trabalhos que consideram a transmissão de vídeo e dispositivos móveis.

Apesar da abundância de trabalhos, percebe-se que existem muitas diferenças nas abordagens e focos das pesquisas, priorizando aspectos como a análise do funcionamento do *streaming* de vídeo (RAMOS-MUNOZ et al., 2014) (RAGIMOVA; LOGINOV; KHOROV, 2019), modelos baseados em dispositivos móveis (FANG; LIU; LEI, 2016), comportamento do usuário na navegação (TSOMPANIDIS; ZAHRAN; SREENAN, 2014) (ZHAO et al., 2013), bem como

nas estratégias para otimizar a transmissão do HTTP a partir do DASH (*Dynamic Adaptive Streaming over HTTP*) (WALDMANN; MILLER; WOLISZ, 2017).

Considerando-se este contexto de diferentes abordagens e a necessidade de estabelecer um modelo para a transmissão de vídeo via HTTP no gerador de tráfego proposto, acredita-se que o modelo recomendado por Navarro-Ortiz et al. em (NAVARRO-ORTIZ et al., 2020), que tem como base os valores estabelecidos pelo YouTube e apresentados em (Youtube, 2021), seja o mais adequado neste momento, dada a relevância e extensão da pesquisa realizada por esses autores.

2.2 AMBIENTES DE SIMULAÇÃO

Geradores de tráfego ganham maior relevância quando estão integrados em ambientes de simulação, uma vez que podem se beneficiar do uso contínuo por parte da comunidade de pesquisadores, recebendo melhorias com o passar do tempo.

Existem muitas propostas de sistemas para simulações em Redes de Computadores. Em consonância com os objetivos do projeto, foi possível elencar quatro características desejáveis para a escolha do ambiente de simulação (MATA, 2017):

- **Código aberto (Open Source):** a maioria dos softwares de código aberto são gratuitos. Simuladores comerciais geralmente apresentam um alto custo pela licença de uso. Além disso, os softwares de código aberto permitem customizações e ajustes de acordo com a necessidade.

- **Modelos confiáveis:** o ambiente de simulação deve apresentar modelos confiáveis que asseguram a validade dos resultados.

- **Implementação eficiente:** para possibilitar a simulação de um grande número de cenários, o ambiente de simulação deve apresentar implementação moderna para fazer uso eficiente dos recursos de hardware.

- **Boa documentação:** um software bem documentado é essencial para facilitar o uso, customizações e contribuições da comunidade de usuários.

Considerando as características elencadas acima, o ambiente de simulação Network Simulator 3 (ns-3) (NS-3 Consortium, 2021b) se destaca por cumprir todos os requisitos.

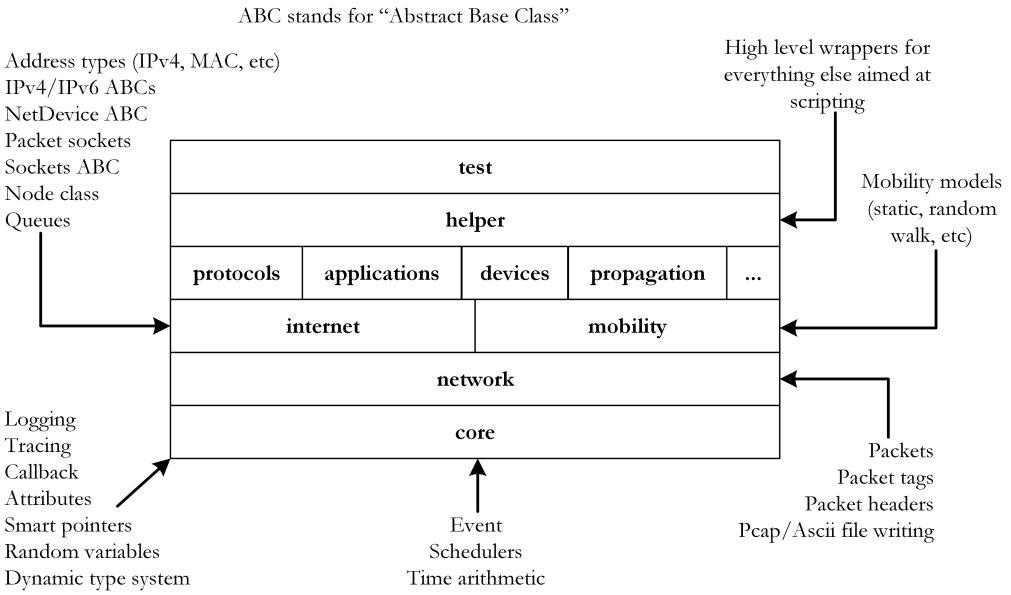


Figura 2: Estrutura modular de organização do ns-3 (Adaptado de (NS-3 Consortium, 2021b)).

O ns-3 é um simulador de redes do tipo *discrete-event*, desenvolvido com o objetivo de atender demandas educacionais e de pesquisa. É um programa gratuito, distribuído sob a licença GNU GPLv2, publicamente disponível no site dos desenvolvedores. É um simulador de propósito geral, ou seja, possibilita o estudo de diversos aspectos de uma rede. Com uma estrutura modular, o ns-3 permite que funcionalidades e melhorias sejam integradas ao seu núcleo por meio de módulos. O núcleo e os módulos do simulador são implementados na linguagem C++. A Figura 2 apresenta uma representação dessa estrutura modular. O núcleo (*core*) do simulador é responsável por prover funcionalidades que são comuns a todos os tipos de redes, tais como, eventos, aritmética de tempo e variáveis aleatórias. O módulo de rede (*network*) é responsável por implementar a estrutura dos pacotes. Logo acima, diversos módulos são posicionados com funcionalidades específicas, dentre eles, os módulos de aplicações onde se encontram os geradores de tráfego. O módulo *helper* facilita o processo de construção dos *scripts* de simulação. Estes *scripts* podem ser escritos em C++ ou Python e especificam os parâmetros de simulação (topologia, interface aérea, tempo de simulação, etc).

No que concerne geradores de tráfego, o ns-3 apresenta, por padrão uma série de aplicações com características específicas endereçadas a diferentes propósitos dos usuários. Para o HTTP, o ambiente disponibiliza um módulo baseado em especificações publicadas em 2001 pelo 3GPP2 (3th Generation Partneship Project) publicadas em (3GPP2-TSGC5, 2001). O módulo é interessante como guia de implementação para o gerador proposto neste projeto. Por outro lado,

observa-se que o modelo matemático é baseado em dados com 20 anos de defasagem para a web atual.

Em (CHENG; ÇETINKAYA; STERBENZ, 2013), Cheng, Çetinkaya e Sterbenz apresentam uma proposta de módulo gerador de tráfego HTTP, baseado em conexões, para o ns-3. Este é o único trabalho encontrado com proposta similar ao projeto aqui delineado. Este módulo descrito em (CHENG; ÇETINKAYA; STERBENZ, 2013) apesar de apresentar uma implementação interessante, baseia-se em um modelo matemático com dados coletados em 1997 e não está integrado ao ns-3.

Dado o contexto atual da literatura do assunto, observa-se que exite um espaço de contribuição para um módulo com modelos matemáticos mais atualizados e funcionalidades que permitam o estudo do protocolo HTTP.

3 MATERIAL E MÉTODOS

Para o desenvolvimento do gerador proposto se faz necessário o uso intenso de ferramentas para o desenvolvimento de programas computacionais.

Com relação ao hardware, utilizou-se um notebook com processador AMD Ryzen 5 e 8GB de memória RAM. No aspecto do armazenamento, o projeto não demandou um grande volume de dados e, portanto, o uso de um SSD com espaço de pelo menos 100GB é o bastante.

Para o software, os simuladores de rede costumam ser executados em ambientes Linux e Unix. A execução em ambientes Windows, ainda que possível, apresenta maiores dificuldades e fontes de erros. Portanto, tanto para o desenvolvimento quanto para os testes e avaliação de desempenho da proposta, utilizou-se o sistema operacional Linux Ubuntu 20.04 LTS com virtualização via Virtual Box.

Nos aspectos de desenvolvimento do gerador e, considerando-se um ambiente de simulação como o ns-3, as linguagens de programação C++ e Python serão as protagonistas no desenvolvimento do projeto. Os artefatos produzidos no decorrer do trabalho foram armazenados na plataforma de disponibilização de código GitLab, já empregada pelos desenvolvedores do ns-3.

4 GERADOR DE TRÁFEGO HTTP

O protocolo HTTP, definido nos RFCs 1945 e 2616 é implementado em dois programas: cliente e servidor.

A interação entre estes programas, que estão sendo executados em diferentes dispositivos da rede ocorre por meio da troca de mensagens HTTP. Estas mensagens tem uma estrutura estabelecida e permitem a requisição, por parte do cliente, de recursos hospedados no servidor, conforme tratado na Figura 1.

Quando um usuário solicita um recurso em um servidor web, este recurso pode ser identificado por uma URL (*Uniform Resource Location*). Em outras palavras, o cliente envia uma mensagem de requisição ao servidor com indicando, por meio de uma URL, qual recurso deseja receber.

O servidor por sua vez, responde essa solicitação com uma mensagem de resposta e o recurso solicitado, quando este é localizado, é claro. Tais mensagens de requisição e repostas são padronizadas pelos RFCs 1945, 2616 e 7540 e formam o cabeçalho HTTP (KUROSE; ROSS, 2013).

Este cabeçalho HTTP poderia ser abstraído em uma implementação de gerador de tráfego HTTP para ambientes de simulação. Contudo, considerando que a presente proposta também tem como objetivo fomentar o estudo do protocolo, procedeu-se a implementação de uma versão simplificada deste cabeçalho.

4.1 CABEÇALHO HTTP

O trecho abaixo ilustra uma mensagem HTTP de requisição típica (adaptado de (KUROSE; ROSS, 2013)).

```
GET /umaurl/algumrecurso.html HTTP/1.1
Host: www.algumsite.com
Connection: close
User-agent: Mozilla/5.0
```

A primeira linha desta mensagem é denominada linha de requisição (*request line*). As linhas seguintes são as linhas de cabeçalho.

Na linha de requisição podemos identificar três campos: o método (GET, POST, PUT), a URL e a versão do HTTP. Já as linhas de cabeçalho são formadas pelo nome do campo, acrescido de dois pontos (:), espaço e o valor para aquele campo.

A mensagem de requisição do cabeçalho HTTP implementada no módulo gerador proposto, por ora, apresenta apenas a linha de requisição. Essa simplificação atende as demandas do gerador de tráfego, enquanto abre espaço para futuras melhorias.

Para a mensagem de resposta, o trecho abaixo ilustra uma situação comum (adaptado de (KUROSE; ROSS, 2013)):

```
HTTP/1.1 200 OK
Connection: close
Date: Sun, 12 Dec 2021 22:18:05 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Fri, 10 Dec 2021 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(data data data data ...)
```

A estrutura da mensagem de reposta é composta por três partes: linha de estado (*status line*), linhas de cabeçalho e corpo da entidade.

A linha de estado apresenta a versão do HTTP, o código de estado do recurso e a frase de estado correspondente. As linhas de cabeçalho apresentam informações sobre a conexão, servidor e detalhes do recurso solicitado. Por fim, o corpo da entidade traz o recurso propriamente dito.

A mensagem de resposta implementada no gerador de tráfego proposto apresenta a linha de estado, o corpo da entidade e possibilita o uso de quantas linhas de cabeçalho a aplicação servidora desejar enviar ao cliente.

4.1.1 Implementação no ns-3

O ns-3 apresenta uma estrutura de diretórios com organização pré-definida. Os cabeçalhos de protocolos se encontram no diretório `src/internet/`. Escrito em C++, o código de implementação apresenta as estruturas necessárias para a montagem do cabeçalho por parte da aplicação definidos em arquivos com extensão `.h` e `.cc`.

Por ser um projeto de grande escala, desenvolvido por pesquisadores e desenvolvedores do mundo inteiro, o consórcio que gerencia o desenvolvimento do simulador estabeleceu regras para a escrita do código, conforme apresentado em (NS-3 Consortium, 2021a). Neste sentido, o ns-3 utiliza o Doxygen como ferramenta para gerar a documentação a partir dos comentários inseridos ao longo do código fonte. A Figura 3 apresenta um trecho de código de um gerador

```
69 class ThreeGppHttpServer : public Application
70 {
71 public:
72   /**
73    * Creates a new instance of HTTP server application.
74    *
75    * After creation, the application must be further configured through
76    * attributes. To avoid having to do this process manually, please use
77    * ThreeGppHttpServerHelper.
78    *
79    * Upon creation, the application randomly determines the MTU size that it
80    * will use (either 536 or 1460 bytes). The chosen size will be used while
81    * creating the listener socket.
82    */
83   ThreeGppHttpServer ();
```

Figura 3: Comentários no código fonte.).

Figura 4: Documentação da API da classe gerada via doxygen.

de tráfego já presente no ns-3. Repare que antes da declaração do construtor da classe, existe um comentário explicativo. Na Figura 4 tem-se um trecho da documentação da API do ns-3 gerada com o Doxygen a partir dos comentários inseridos no código fonte e disponibilizada na web (NS-3 Consortium, 2021c).

Para o cabeçalho HTTP desenvolvido no projeto, seguiu-se a mesma estratégia, conforme ilustrado na Figura 5.

Outro aspecto importante da implementação é a abordagem realista estabelecida pelo ns-3 para o cabeçalho. Os valores dos campos do cabeçalho são "serializados"e "deserializados"no processo de implementação do código conforme ilustrado na Figura 6. Esse mecanismo resulta em um cabeçalho que pode ser registrado em arquivos de captura de pacotes de rede, como por exemplo o tradicional formato `.pcap`. Ao serem lidos por uma aplicação adequada como o Wireshark, estes arquivos `.pcap` revelam a estratégia realista da implementação. A Figura 7 apresenta duas capturas: uma realizada via simulação no ns-3 e a outra em uma rede real. É

```
89  std::string GetVersion (void) const;
90
91  /**
92   * \brief Set the Status Code field of the response message.
93   * \param statusCode the string with the status code (200, 301, 404) of the response message.
94   */
95  void SetStatusCode (std::string statusCode);
96
97  /**
98   * \brief Get the status code field.
99   * \return the status code field.
100  */
101 std::string GetStatusCode (void) const;
102
103 /**
104  * \brief Set the Phrase field of the response message.
105  * \param phrase the string with the phrase (OK, NOT FOUND) of the response message.
106  */
107 void SetPhrase (std::string phrase);
```

Figura 5: Comentários no código do cabeçalho para futura documentação.

possível observar que a implementação do cabeçalho está em consonância com os requisitos estabelecidos pelo ns-3 e organizações de padronização para redes reais. A implementação da mensagem de requisição no cabeçalho proposto apresenta apenas a linha de requisição. Contudo, novas versões poderão apresentar os outros campos e possibilitar novas análises.

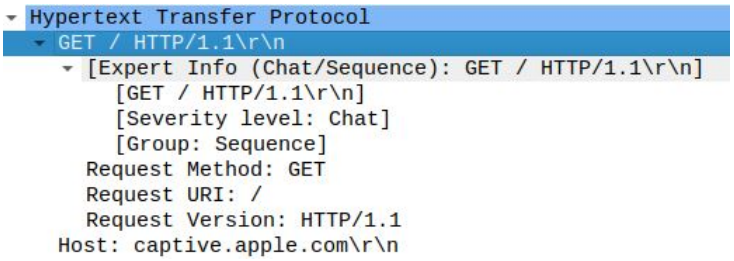
```
void
HttpHeader::Serialize (Buffer::Iterator start) const
{
  NS_LOG_FUNCTION_NOARGS ();

  if(m_request)
    {
      std::string requestLine = m_method + " " + m_url + " " + m_version + "\r\n\r\n";
      char tmpBuffer [requestLine.length() + 1];
      strcpy (tmpBuffer, requestLine.c_str());
      start.Write ((uint8_t *)tmpBuffer, strlen(tmpBuffer) + 1);
    }
```

(a) Serialização.

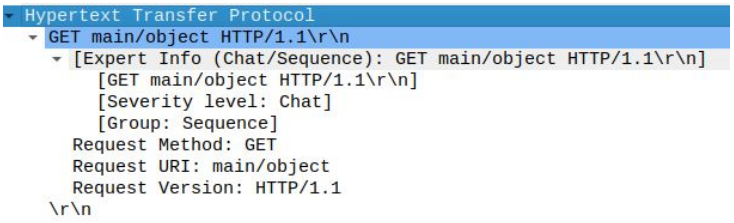
```
uint32_t
HttpHeader::Deserialize (Buffer::Iterator start)
{
  NS_LOG_FUNCTION_NOARGS ();
  Buffer::Iterator i = start;
  uint8_t c;
  uint32_t len = 0;

  do
    {
      c  = i.ReadU8 ();
      len++;
    } while (c != 0);


  char tmpBuffer [len];
  start.Read ((uint8_t*)tmpBuffer, len);
```

(b) Deserialização.

Figura 6: Métodos de serialização e deserialização.



(a) Cabeçalho HTTP real.



(b) Cabeçalho HTTP simulado.

Figura 7: Comparação de capturas de pacotes em rede real e simulada.

4.2 APLICAÇÕES CLIENTE E SERVIDOR

O cabeçalho HTTP fomenta maiores estudos em torno do funcionamento do protocolo HTTP. Contudo, o núcleo principal do gerador de tráfego proposto são as aplicações cliente e servidor.

No ns-3, as aplicações estão localizadas no diretório `/src/applications/`. Dentro deste diretório é possível identificar algumas outras pastas. Dentre elas podemos destacar `model`, `helper` e `examples`.

No diretório `model` encontram-se as implementações dos modelos das aplicações do ns-3. Já no diretório `helper` estão programas auxiliares que facilitam o uso das aplicações nos scripts de simulação. Por fim, a pasta `examples` apresenta exemplos de utilização das aplicações, que facilitam o uso por parte de novos usuários.

O gerador proposto segue o comportamento descrito anteriormente para aplicações baseadas no HTTP, isto é, um cliente solicita um dado recurso que está armazenado em um servidor remoto. O servidor por sua vez, encontra o recurso e envia para o cliente solicitante.

Do lado da aplicação cliente, o gerador de tráfego implementa uma aplicação simples. Ao receber, via script de simulação, as informações de endereço IP e porta de trabalho do servidor, o cliente cria um socket TCP e inicia o processo de estabelecimento da sessão TCP. Se o servidor está operante, este recebe o pedido de sessão e realiza os procedimentos do tradicional *three-way handshake*. Aqui vale ressaltar as vantagens da implementação do gerador de tráfego dentro de um ambiente de simulação como o ns-3. Toda a pilha de protocolos já está estabelecida,

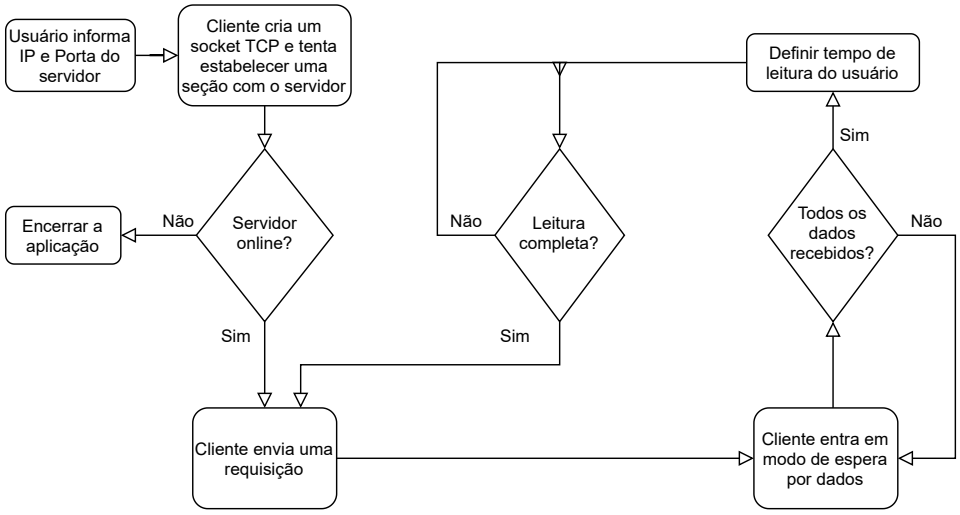


Figura 8: Fluxograma do funcionamento do cliente HTTP.

facilitando o trabalho dos desenvolvedores das aplicações e dando maior realismo à simulação.

Com a sessão TCP estabelecida, o cliente envia, então, uma mensagem HTTP de requisição, passando a url `main/object` e indicando que deseja receber o objeto principal de uma página web genérica. Em seguida, o cliente entra em modo de espera, aguardando o envio dos dados. Ao receber todos os dados, a aplicação cliente utiliza o modelo matemático apresentado em (PRIES; MAGYARI; TRAN-GIA, 2012) para calcular o tempo de leitura do usuário, antes de realizar uma nova solicitação por outro recurso no servidor. A Figura 8 ilustra esse fluxo de funcionamento.

Do lado da aplicação servidora, o programa recebe a porta de trabalho via script de simulação, inicia um socket TCP e fica aguardando solicitações de criação de sessão.

Ao receber e aceitar uma sessão TCP, o servidor HTTP passa para o modo de espera por requisições web. Ao receber a requisição, a aplicação estabelece os parâmetros das respostas, tais como, tamanho do objeto principal, número e tamanho dos objetos *inline*, com base no modelo matemático apresentado em (PRIES; MAGYARI; TRAN-GIA, 2012). Assim que todos os dados solicitados são entregues, o servidor volta ao modo de espera por novas requisições. A Figura 9 ilustra esse processo.

5 RESULTADOS E DISCUSSÃO

Para testar o gerador de tráfego proposto, bastar realizar a criação de um script de simulação e utilizar as estruturas criadas e armazenadas no diretório `/src/application/helper/` para facilitar a utilização do gerador de tráfego.

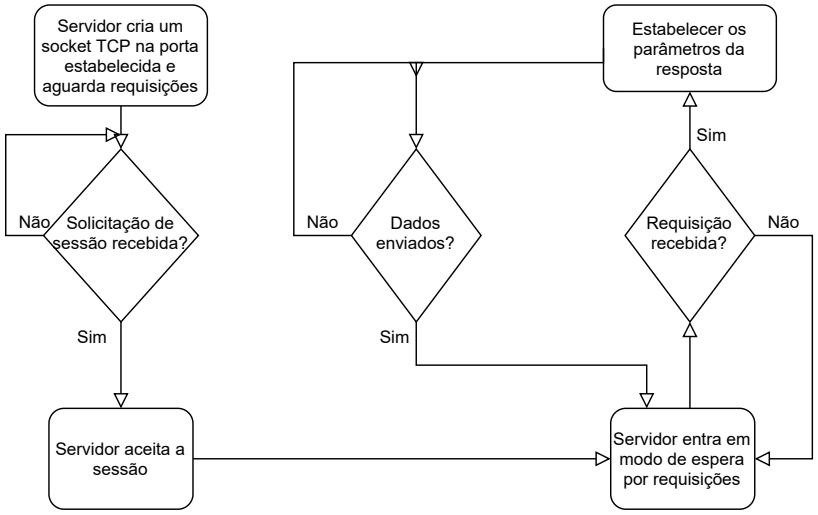


Figura 9: Fluxograma do funcionamento do servidor HTTP.

A Figura 10 apresenta um script simples para teste do gerador de tráfego. Nas linhas 34 e 35 dois nós são criados para assumirem o papel de cliente e servidor na simulação. Em seguida, nas linhas 38 e 39 uma instância da pilha de protocolos da Internet é criada e instalada nos dois nós da rede. Os dois nós são então conectados por um enlace ponto a ponto, com os parâmetros dessa conexão definidos nas linhas 42 a 46.

Em seguida, uma sub-rede é criada e o endereçamento IP dos nós da rede são estabelecidos e atribuídos (linhas 48 a 51). Com nós criados, conectados e com endereçamento adequado, precisamos instalar as aplicações. Isso ocorre nas linhas 58 a 62.

Com a topologia da simulação estabelecida, passa-se à definição do tempo de simulação. Serão 10 segundos (linha 70), considerando a realidade simulada. O tempo real pode variar bastante dependendo da complexidade da rede. Neste caso a simulação é simples e é executada quase que instantaneamente. É preciso ficar atento ao agendamento do início das aplicações. Repare que a aplicação do servidor é iniciada primeiro, com um segundo de simulação (linha 64). Já o cliente se inicia aos dois segundos de simulação. Dessa forma, quando o cliente for iniciado, o servidor já estará apto a receber as solicitações.

O último trecho do script apresenta estruturas para capturas de pacotes, e a execução da simulação propriamente dita.

Ao executar o script de simulação recebemos uma saída no console, com os principais acontecimentos da simulação, conforme apresentado na Figura 11. O trecho da saída ilustrado mostra o passo a passo das trocas de mensagens entre os elementos do gerador de tráfego. O comportamento do gerador correspondeu ao esperado, com diferentes tamanhos de páginas e

```
int
main (int argc, char *argv[])
{
  // Habilitando os logs do script e das aplicacoes
  LogComponentEnable ("SimulacaoHTTP", LOG_LEVEL_INFO);
  LogComponentEnable ("HttpClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("HttpServerApplication", LOG_LEVEL_INFO);

  NS_LOG_INFO ("Criando os nós...");
  NodeContainer n;
  n.Create (2);

  InternetStackHelper internetStack;
  internetStack.Install (n);
```

(a)

```
  NS_LOG_INFO ("Criando a ligação ponto a ponto...");
  PointToPointHelper p2p;
  p2p.SetDeviceAttribute ("DataRate", DataRateValue (DataRate ("1Gb/s")));
  p2p.SetDeviceAttribute ("Mtu", UintegerValue (1500));
  p2p.SetChannelAttribute ("Delay", TimeValue (Seconds (0.001)));
  NetDeviceContainer c = p2p.Install (n);

  NS_LOG_INFO ("Atribuindo os Endereços IP...");
  Ipv4AddressHelper ipv4;
  ipv4.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer i = ipv4.Assign (c);

```

(b)

```
  NS_LOG_INFO ("Criando as Aplicações...");
  uint16_t httpServerPort = 80;
  ApplicationContainer httpServerApps;
  ApplicationContainer httpClientApps;

  HttpServerHelper httpServer (httpServerPort);
  httpServerApps.Add (httpServer.Install (n.Get (0)));

  HttpClientHelper httpClient (i.GetAddress (0), httpServerPort);
  httpClientApps.Add (httpClient.Install (n.Get (1)));

  httpServerApps.Start (Seconds(1.0));
  httpServerApps.Stop (Seconds(10.0));

  httpClientApps.Start (Seconds(2.0));
  httpClientApps.Stop (Seconds(10.0));
```

(c)

```
  Simulator::Stop(Seconds(10.0));

  NS_LOG_INFO ("Habilitando a Captura dos Pacotes...");
  p2p.EnablePcapAll ("simulacao-http");

  NS_LOG_INFO("Começando a Simulação...\n");
  Simulator::Run();
  Simulator::Destroy();
  NS_LOG_INFO("\nSimulação Completa.");
  return 0;
}
```

(d)

Figura 10: Script de simulação para teste do gerador de tráfego.

números de objetos *inline*, de acordo com o modelo matemático estabelecido anteriormente.

```
jordan@ubuntu-ns3:~/ns-3/ns-3-dev$ ./waf --run simulacao-http
Waf: Entering directory `/home/jordan/ns-3/ns-3-dev/build'
Waf: Leaving directory `/home/jordan/ns-3/ns-3-dev/build'
Build commands will be stored in build/compile_commands.json
'build' finished successfully (0.486s)
Criando os nós...
Criando a ligação ponto a ponto...
Atribuindo os Endereços IP...
Criando as Aplicações...
Habilitando a Captura dos Pacotes...
Começando a Simulação...

HttpServer >> Request for connection from 10.1.1.2 received.
HttpClient (10.1.1.2) >> Server accepted connection request!
HttpClient (10.1.1.2) >> Sending request for main/object to server (10.1.1.1).
HttpServer >> Connection with Client (10.1.1.2) successfully established!
HttpServer >> Client requesting a main/object
HttpServer >> Sending response to client. Main Object Size (16815 bytes). NumOfI
nlineObjects (4).
HttpClient (10.1.1.2) >> main/object: 445 bytes of 16815 received.
HttpClient (10.1.1.2) >> main/object: 981 bytes of 16815 received.
HttpClient (10.1.1.2) >> main/object: 1517 bytes of 16815 received.
HttpClient (10.1.1.2) >> main/object: 2053 bytes of 16815 received.
HttpClient (10.1.1.2) >> main/object: 2589 bytes of 16815 received.
```

Figura 11: Saída da execução do script de simulação.

5.1 Trabalhos Futuros

Após a implementação bem-sucedida do modelo de tráfego web clássico de (PRIES; MAGYARI; TRAN-GIA, 2012), foram inicializados estudos para a implementação de um modelo para transmissão de vídeo streaming, com base nos modelos citados por (NAVARRO-ORTIZ et al., 2020).

Durante essa fase do desenvolvimento, percebeu-se que para uma efetiva implementação desse modelo, seria necessário utilizar arquivos de vídeo no formato especial YUV, que por sua vez possibilitam a codificação (*enconding*) de acordo com os parâmetros elencados em (NAVARRO-ORTIZ et al., 2020) e (Youtube, 2021). Esse pré -processamento é necessário para a posterior simulação da transmissão dos pacotes de streaming de vídeo a partir de módulos específicos para esta tarefa como o Evalvid (GERCOM, 2021). Diante deste cenário, não houve tempo hábil para a implementação dessa etapa do projeto.

Outro tarefa futura é a revisão do código fonte e comentários para submissão do código para uma eventual inclusão na distribuição oficial do simulador ns-3.

6 CONCLUSÃO

Considerando a importância das redes de comunicações no cenário atual, bem como a relevância do protocolo HTTP na composição do tráfego destas redes, o presente trabalho propôs a concepção de um gerador de tráfego HTTP para estudos em ambientes de simulação.

A partir de modelos matemáticos criados pela comunidade de pesquisadores da área e utilizando-se da estrutura e pujante documentação do ambiente de simulação ns-3, foi possível endereçar os principais pontos da proposta, com a criação de um cabeçalho HTTP e a implementação de duas aplicações, de acordo com a arquitetura cliente-servidor.

Os resultados obtidos comprovam a hipótese inicial e evidenciam os aspectos promissores de uma iniciativa como esta. Futuros desenvolvimentos são necessários para aprimorar a proposta e fomentar estudos cada vez mais aprofundados de diversos aspectos das redes de computadores.

REFERÊNCIAS

3GPP2-TSGC5. *HTTP, FTP and TCP models for 1xEV-DV simulations*. 2001.

CHENG, Y.; ÇETINKAYA, E. K.; STERBENZ, J. P. G. Transactional Traffic Generator Implementation in ns-3. In: *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*. [S.l.: s.n.], 2013. p. 182–189.

Cisco. *Cisco Predicts More IP Traffic in the Next Five Years Than in the History of the Internet*. 2018. Acesso em: 11 de Junho de 2021. Disponível em: <https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1955935>.

CROVELLA, M.; BESTAVROS, A. Self-similarity in World Wide Web traffic: evidence and possible causes. *IEEE/ACM Transactions on Networking*, v. 5, n. 6, p. 835–846, 1997. ISSN 10636692. Disponível em: <http://ieeexplore.ieee.org/document/650143/>.

FANG, C.; LIU, J.; LEI, Z. Fine-Grained HTTP Web Traffic Analysis Based on Large-Scale Mobile Datasets. *IEEE Access*, IEEE, v. 4, p. 4364–4373, 2016. ISSN 2169-3536. Disponível em: <http://ieeexplore.ieee.org/document/7529079/>.

G1. *Pela 1ª vez, internet em smartphones e tablets supera uso no computador*. 2016. Acesso em: 12 de dezembro de 2021. Disponível em: <http://g1.globo.com/tecnologia/noticia/2016/11/pela-1-vez-internet-em-smartphones-e-tablets-supera-uso-no-computador.html>.

GERCOM. *Módulo Evalvid para o ns-3*. 2021. Acesso em: 14 de dezembro de 2021. Disponível em: <https://github.com/gercom/evalvid-ns3>.

HALL, B. *Beej's Guide to Network Programming: Using Internet Sockets*. Autopublicado, 2019. ISBN 978-1705309902. Disponível em: <http://beej.us/guide/bgnet/>.

IETF. *RFCs 7230 a 7237 - Hypertext Transfer Protocol (HTTP/1.1)*. [S.l.], 2014. Acesso em: 12 de dezembro de 2021. Disponível em: <https://datatracker.ietf.org/wg/httpbis/documents/>.

JOHNSON, J. *Internet usage worldwide - Statistics & Facts*. 2021. Acesso em: 22 de Maio de 2021. Disponível em: <https://www.statista.com/topics/1145/internet-usage-worldwide/>.

KUROSE, J. F.; ROSS, K. W. *Redes De Computadores E A Internet: Uma Abordagem Top Down*. [S.l.]: Pearson, 2013. ISBN 978-8581436777.

LAVADO, T. *Com maior uso da internet durante pandemia, número de reclamações aumenta; especialistas apontam problemas mais comuns*. 2020. Acesso em: 24 de Maio de 2021. Disponível em: <https://g1.globo.com/economia/tecnologia/noticia/2020/06/11/com-maior-uso-da-internet-durante-pandemia-numero-de-reclamacoes-aumenta-especialistas-apontam-problemas-mais-comuns.ghtml>.

MATA, S. H. da. *A New Genetic Algorithm Based Scheduling Algorithm for the LTE Uplink*. Tese (Doutorado) — Universidade Federal de Uberlândia, Uberlândia, Brasil, 2017. Disponível em: <https://repositorio.ufu.br/bitstream/123456789/19062/3/NewGeneticAlgorithm.pdf>.

NAVARRO-ORTIZ, J. et al. A Survey on 5G Usage Scenarios and Traffic Models. *IEEE Communications Surveys & Tutorials*, p. 905–929, 2020. ISSN 1553-877X. Disponível em: <https://ieeexplore.ieee.org/document/8985528/>.

NS-3 Consortium. *Contributing to ns-3*. 2021. Acesso em: 13 de dezembro de 2021. Disponível em: <https://www.nsnam.org/docs/contributing/html/index.html>.

NS-3 Consortium. *ns-3 | a discrete-event network simulator for internet systems*. 2021. Acesso em: 12 de dezembro de 2021. Disponível em: <https://www.nsnam.org>.

NS-3 Consortium. *ns-3 API Documentation*. 2021. Acesso em: 14 de dezembro de 2021. Disponível em: <https://www.nsnam.org/docs/release/3.35/doxygen/index.html>.

PRIES, R.; MAGYARI, Z.; TRAN-GIA, P. An HTTP web traffic model based on the top one million visited web pages. In: *Proceedings of the 8th Euro-NF Conference on Next Generation Internet NGI 2012*. IEEE, 2012. p. 133–139. ISBN 978-1-4673-1634-7. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6252145http://ieeexplore.ieee.org/document/6252145/>.

RAGIMOVA, K.; LOGINOV, V.; KHOROV, E. Analysis of YouTube DASH Traffic. *2019 IEEE International Black Sea Conference on Communications and Networking, BlackSeaCom 2019*, IEEE, p. 1–5, 2019.

RAMOS-MUNOZ, J. et al. Characteristics of mobile youtube traffic. *IEEE Wireless Communications*, v. 21, n. 1, p. 18–25, feb 2014. ISSN 1536-1284. Disponível em: <http://ieeexplore.ieee.org/document/6757893/>.

RHODES, B.; GOERZEN, J. *Foundations of Python Network Programming*. [S.l.]: Apress, 2014. ISBN 978-1430258544.

TSOMPANIDIS, I.; ZAHRAN, A. H.; SREENAN, C. J. Mobile network traffic: A user behaviour model. In: *2014 7th IFIP Wireless and Mobile Networking Conference (WMNC)*. IEEE, 2014. p. 1–8. ISBN 978-1-4799-3060-9. Disponível em: <https://ieeexplore.ieee.org/document/6878862>.

WALDMANN, S.; MILLER, K.; WOLISZ, A. Traffic model for HTTP-based adaptive streaming. In: *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2017. p. 683–688. ISBN 978-1-5386-2784-6. Disponível em: <http://ieeexplore.ieee.org/document/8116459/>.

Youtube. *Recommended upload encoding settings - YouTube Help*. 2021. Acesso em: 12 de dezembro de 2021. Disponível em: <https://support.google.com/youtube/answer/1722171?hl=en>.

ZHAO, G.-f. et al. Generating Web Traffic based on User Behavioral Model. *Proceedings of the 5th International Conference on Evolving Internet (INTERNET 2013)*, v. 11, n. c, p. 10–15, 2013.