**Universidade Estadual de Campinas**

**Faculdade de Engenharia Elétrica e de Computação**

****

****

**EA006 – Trabalho de Fim de Curso**

**Análise e simulação de tráfego de pacotes para jogos online**

**Aluno:** Lucca Bollani

**RA:** 158182

**E-mail:** l158182@dac.unicamp.br

**Orientador:** Prof. Christian Rothenberg

**E-mail:** chesteve@dca.fee.unicamp.br

Sumário

[**1.** **Introdução** 3](#_Toc117772311)

[**2.** **Objetivos** 4](#_Toc117772312)

[**3.** **Teoria** 4](#_Toc117772313)

[**3.1.** **Wireshark** 4](#_Toc117772314)

[**3.2.** **Jupyter** 5](#_Toc117772315)

[**3.3.** **Unity** 7](#_Toc117772316)

[**4.** **Metodologia** 9](#_Toc117772317)

[**4.1.** **Captura** 9](#_Toc117772318)

[**4.2.** **Análise** 12](#_Toc117772319)

[**4.2.1.** **Transformação da distribuição dos pacotes em relação ao tamanho** 13](#_Toc117772320)

[**4.2.2.** **Transformação da distribuição dos pacotes em relação ao tempo de envio** 15](#_Toc117772321)

[**4.3.** **Reprodução de tráfego** 17](#_Toc117772322)

[**5.** **Resultados** 18](#_Toc117772323)

[**6.** **Aplicação Prática** 19](#_Toc117772324)

[**7.** **Conclusão** 21](#_Toc117772325)

[**8.** **Referências** 21](#_Toc117772326)

**9. Anexo A - Descrição da estrutura de arquivos do projeto**........................................................22

1. **Introdução**

A utilização da internet para tráfego de informações relativas a jogos já se tornou um conceito extremamente comum, e do conhecimento de todos que navegaram na WEB diariamente. Em 1978, antes mesmo da popularização da internet, o jogo MUD (Multi-User Dungeon) foi lançado, e se tornou o primeiro jogo a trazer o universo em expansão de Dungeons and Dragons para dentro desse ambiente conectado. Cerca de 20 anos depois, com o crescente aumento de usuários da rede como forma de entretenimento, vimos os lançamentos de diversos jogos que revolucionaram a indústria, conquistando milhares de jogadores, e solidificando de vez o mercado de jogos online, como Lineage, Starcraft e Counter Strike.

Em 2018, apenas 40 anos após os primeiros protótipos de jogos online, temos a notícia de que um jogo completamente gratuito em sua essência, foi capaz de gerar U$2,4bi em seu segundo ano de lançamento, com mais de 350 milhões de jogadores registrados (Fortnite) [1].

Apesar da clara importância dos jogos online para o mercado, ainda hoje também se observa jogos das maiores e mais populares franquias do mundo perdendo usuários e credibilidade devido ao mal funcionamento de seus servidores, e diversos casos de jogos que falharam em entregar a experiência prometida em seus primeiros dias de lançamento, por não conseguirem lidar com o alto tráfego gerado pelos usuários.

Com o objetivo de entender e se preparar para esses problemas, pesquisadores começaram a analisar o tráfego de pacotes de diversos jogos, de acordo com seu tamanho e IDT (Inter-Departure Time), e propor modelos representativos do tráfego para que os desenvolvedores possam entender a carga esperada de um usuário comum em seu sistema, e então elaborar uma ferramenta onde seja possível que um número arbitrário de usuários seja simulado para testar um serviço.

Neste trabalho, é desenvolvido um procedimento de acordo com os modelos propostos previamente por pesquisadores, para simplificar e simular a geração de pacotes para jogos online, e avaliar o desempenho de um servidor dedicado.

1. **Objetivos**

Este trabalho tem como objetivo principal a produção de uma ferramenta capaz de simular o tráfego de rede enviado de um cliente para um servidor de jogos online, aplicando as propostas definidas previamente em [2]-[3].

Para que isso seja possível, também é necessário a compreensão de técnicas utilizadas na captura de pacotes a nível da camada de transporte, e familiarizar-se com estratégias de processamento de pacotes de rede.

Por fim, tem-se como objetivo fazer uso da ferramenta elaborada para de fato simular pacotes de um jogo especificado, e avaliar o real suporte de seu servidor quanto ao número de usuários simultâneos.

1. **Teoria**
   1. **Wireshark**

Originalmente criado por Gerald Combs em 1998, Wireshark (inicialmente chamado de Ethereal) surgiu da necessidade que tinha em analisar e otimizar o trafego de rede sendo gerado pelos contratantes do provedor de internet para o qual trabalhava na época. Nessa época, outros analisadores de trafego de rede já existiam no mercado, porém, todos eles eram ferramentas pagas de custo elevado e que não ofereciam suporte a plataformas baseadas em Linux, que eram os principais tipos de servidores utilizados pelo provedor em que trabalhava na época. Hoje, graças a seu modelo Open-Source, Wireshark tornou-se a ferramenta mais utilizada para analisar protocolos de rede em todo o mundo [4]-[5].

Alguma das funcionalidades dessa ferramenta são:

* Captura de pacotes em tempo real que transitam através de uma interface de rede.
* Criar e abrir arquivos com listas de pacotes de rede capturados.
* Apresentar os pacotes com uma lista extremamente detalhada de conteúdo e descrição de protocolo.
* Filtrar pacotes de acordo com regras definidas pelo usuário, durante e/ou após a captura.

Graças a essas funcionalidades e seu caráter Open-Source, Wireshark é a ferramenta ideal para realizar a captura do tráfego de jogos online. Sua capacidade de filtrar pacotes baseando-se em regras específicas e exportar o conteúdo capturado durante as seções, torna possível que qualquer tráfego de uma aplicação seja obtido num formato que pode ser facilmente analisado programaticamente, utilizando bibliotecas especificas em linguagens de programação.

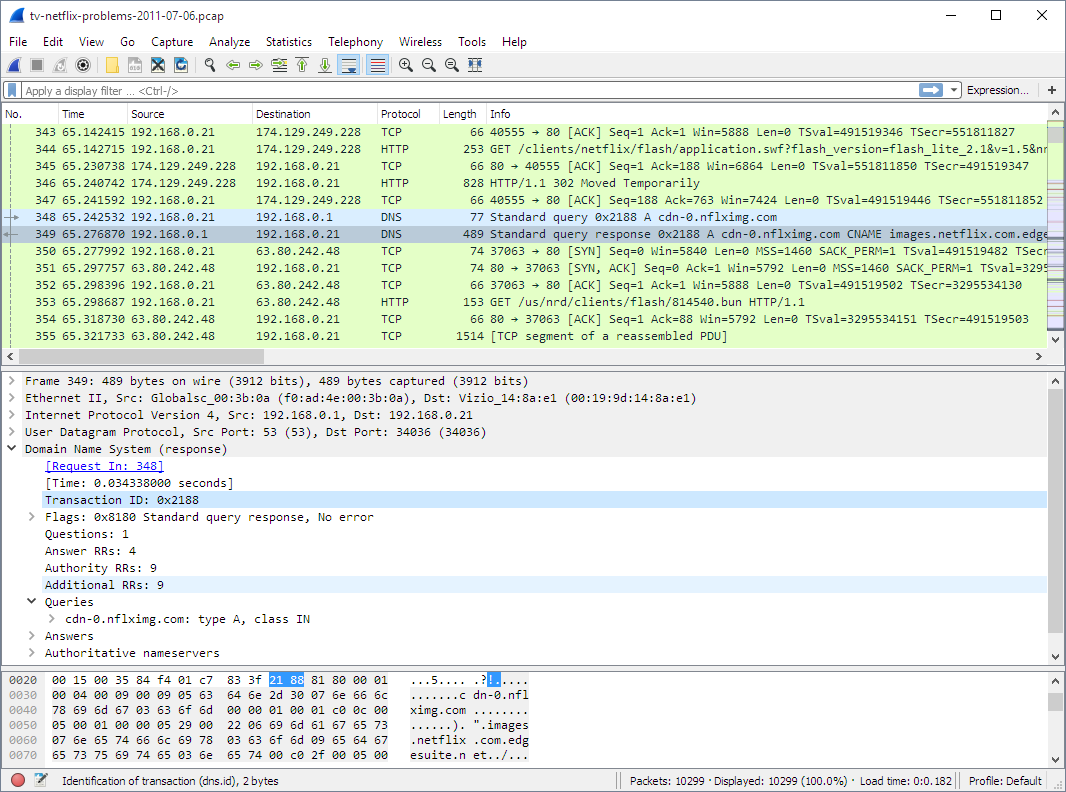


Figura 1: Interface de captura do Wireshark. [4]

* 1. **Jupyter**

Jupyter é um projeto Open-Source sem fins lucrativos criado em 2014 a partir do popular terminal interativo de Python, o IPython [8]. O objetivo do projeto foi criar uma ferramenta com interface flexível, destinada a facilitar o desenvolvimento de projetos e estudos em ciência de dados, computação científica e machine learning [6].

Essa ferramenta disponibiliza ao desenvolvedor a escolha de diversas versões de kernels de mais de 40 linguagens de programação diferentes, sendo as principais Python, R e Julia [7]. Jupyter funciona como um servidor local desses kernels, organizando seu conteúdo num formato de células com as quais o usuário pode interagir através de um navegador web. Nessas células, o desenvolvedor pode definir um bloco de código da linguagem de programação escolhida, um bloco de descrição de texto no formato Markdown, ou então texto simples. Esses blocos podem então ser manipulados através do documento, alterando sua ordenação, criação, cópia, e sendo executados livre e individualmente de acordo com a vontade do desenvolvedor.

Por sua grande flexibilidade, praticidade de visualização de dados e suporte a diversas linguagens comumente utilizadas na área, Jupyter se tornou uma ferramenta extremamente popular entre a comunidade de ciência de dados ao redor do mundo.

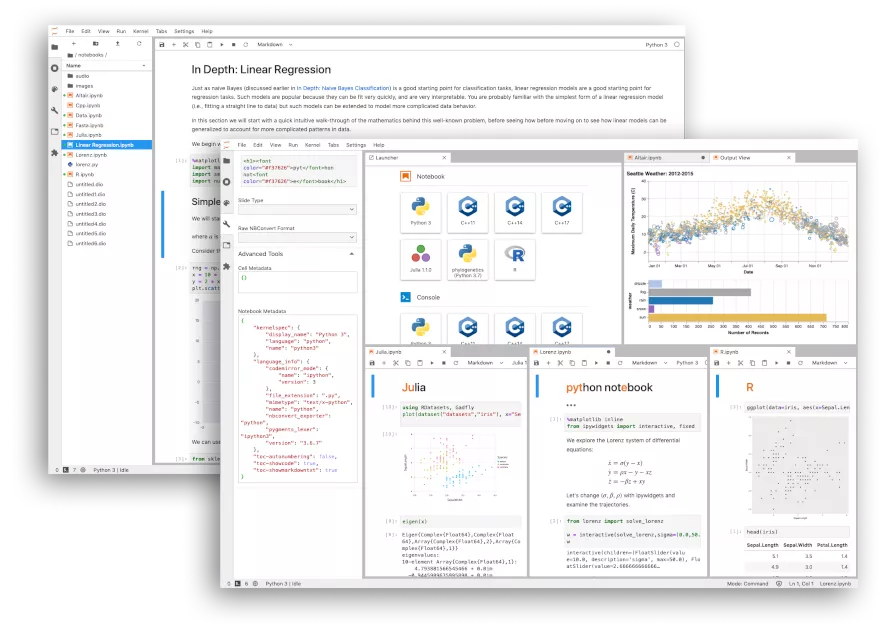


Figura 2: Principais interfaces Web disponíveis no Jupyter. [6]

Neste projeto, Jupyter foi utilizado com kernels da linguagem de programação Python, juntamente de algumas bibliotecas Open-Source para processamento de pacotes de rede, análise de dados, comunicação Web e outras:

* **DPKT**: Módulo rápido e eficiente para análise, processamento e criação de pacotes de rede para protocolos TCP/IP [9].
* **PANDAS**: Biblioteca de processamento de dados que usa uma estrutura de dataframes. Possuiu diversas funcionalidades de agrupamento, busca, filtro e reformatação de dados [10].
* **MATPLOTLIB**: Ferramenta de visualização com diversas opções de apresentação de gráficos estáticos e dinâmicos, com integração ao ambiente Jupyter [11].
* **THREADING**: Biblioteca Python para criação de threads para execução paralela.
* **SOCKET**: Biblioteca padrão de Python para comunicação com outros processos TCP/IP através de sockets.
  1. **Unity**

Unity é uma das principais plataformas para criação de jogos disponível no mercado, por ser uma plataforma intuitiva e bem otimizada, ganhou muita popularidade entre os amantes do desenvolvimento de jogos, e seus diversos tipos de planos de assinatura a tornam um ambiente comumente utilizado por estudantes, artistas interessados na área de animação, e até mesmo grandes empresas da indústria de games global.

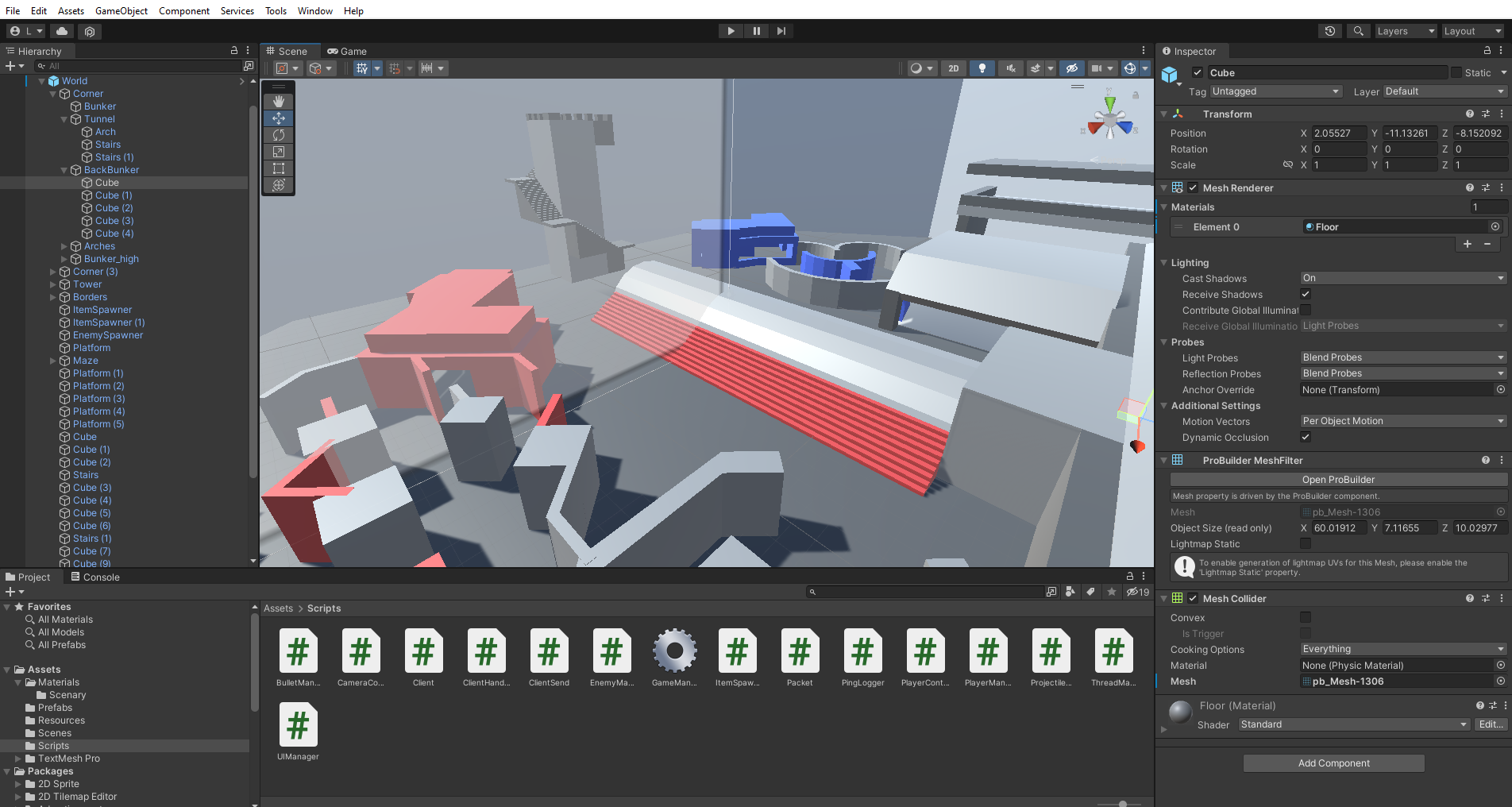
Graças a sua licença gratuita e suporte ao framework .NET com o uso da linguagem C#, Unity torna possível criar um cenário realista de jogos que utilizam de uma arquitetura cliente-servidor, onde todos os pacotes necessários para o jogo funcionar são conhecidos, e podem ser utilizados para a simulação de um cenário real, ou de testes de estresse, como será feito neste trabalho.

Figura 3: Ambiente de desenvolvimento Unity.

1. **Metodologia**
   1. **Captura**

A primeira etapa necessária para tornar possível a análise do trafego de jogos consiste na captura dos pacotes que são transmitidos entre o cliente e o servidor. Para isso, utilizaremos a ferramenta de captura gratuita Wireshark. Através do Wireshark, é possível escolher uma interface de rede local que será analisada, e a ferramenta imediatamente começa a capturar os pacotes que passam por essa interface a nível da camada de transporte.

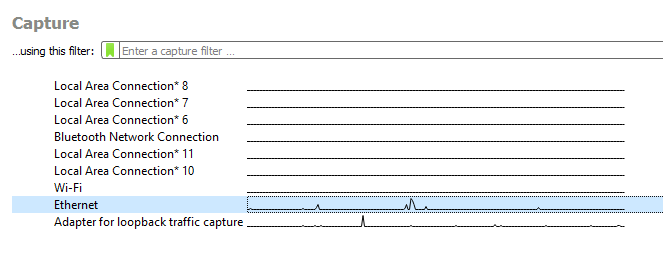


Figura 4: Interface inicial do Wireshark, onde é possível definir o filtro de captura e a interface Web a ser observada.

Mas apenas selecionar a interface correta não é uma boa solução, pois há muitos outros processos fazendo uso da mesma interface, portanto, o painel principal de captura do Wireshark rapidamente se vê repleto de pacotes que não têm relação alguma com a aplicação alvo de estudo. Para corrigir isso, é possível fazer uso de dois tipos de filtro disponibilizados pela ferramenta. O primeiro é chamado de filtro de display, e consiste em, após feita uma captura, o Wireshark irá apenas apresentar os pacotes cuja regra do filtro for verdadeira, tornando possível que então sejam exportados para um arquivo lista de pacotes. O segundo tipo de filtro disponibilizado é o chamado filtro de captura, que é um filtro informado previamente pelo usuário, e fará com que a ferramenta ignore completamente todos os pacotes que encontrar durante a captura, cuja regra do filtro é dada como falso.

Devido à grande quantidade de pacotes que desejamos analisar, para que possamos obter um resultado mais fiel à realidade, a utilização do segundo tipo de filtro se torna mais interessante, pois não sobrecarrega a memória da aplicação com pacotes desnecessários durante o período de captura.

Para determinar o filtro, é necessário saber qual tipo de regra se aplica apenas aos pacotes que queremos analisar, essas regras podem ter relação com o IP, Portas, Endereços MAC, conteúdo, e diversos outros atributos carregados pelos pacotes. Dado a falta de referências exatas de qual porta, IP, ou MAC do servidor onde a seção do jogo acontece, inicialmente é necessária que o filtro seja mais amplo, contendo todas as possibilidades que podem ser atribuídas à conexão. Para isso, será feito o uso da referência [12] que contém as possíveis portas de comunicação com um jogo atual de mercado, Valorant, da Riot Games.

O primeiro filtro de captura a ser utilizado é, portanto, o seguinte [13]:

*udp portrange 7000-8000 or tcp port 2099 or tcp port 5223 or tcp port 5222*

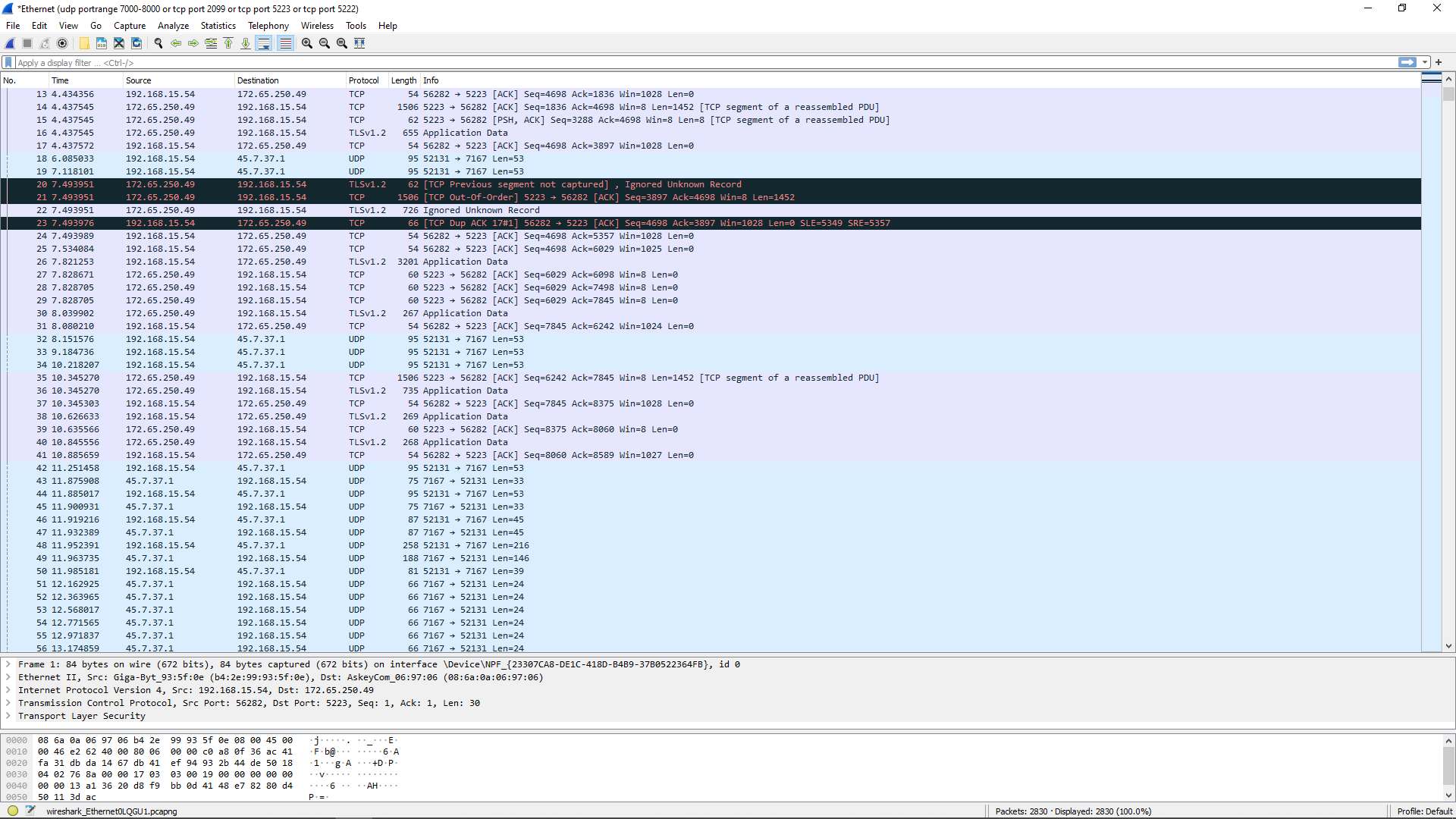


Figura 5: Resultados com o primeiro filtro de captura

Dessa forma, o excesso de pacotes que certamente não têm relação com portas utilizadas pelo provedor do jogo já são previamente filtrados da captura inicial, facilitando a identificação de informações mais precisas, como o IP e a porta ao qual de fato a seção do jogo foi atribuída. Ainda da mesma captura, é possível ver que inicialmente há uma pequena quantidade de pacotes do protocolo TCP e TLS trocados para alocar os jogadores na arena em que ocorrerá a partida, porém, imediatamente em seguida, majoritariamente são trocados pacotes do protocolo UDP, que constituem os comandos pressionados constantemente pelo jogador, e as posições dos outros jogadores/recursos da partida.

Com essa visão inicial de como se comporta o tráfego de informações do jogo, é possível ver que praticamente 100% do volume total enviado pelo cliente é constituído de pacotes do protocolo UDP. Como estamos apenas interessados nos pacotes que são enviados pelo cliente, o filtro que de fato será utilizado para captura precisa filtrar a porta exata da seção, o IP de destino, e o protocolo empregado:

*dst port 7302 and udp port 7302 and dst 45.7.37.1*

Dessa forma, verifica-se que os pacotes capturados são todos enviados do cliente para o servidor do jogo, através do protocolo udp. Nota-se que esse novo filtro deve ser refeito para cada partida, dado que a porta é alterada a cada seção. A período de captura durou por uma partida de aproximadamente 10 minutos.

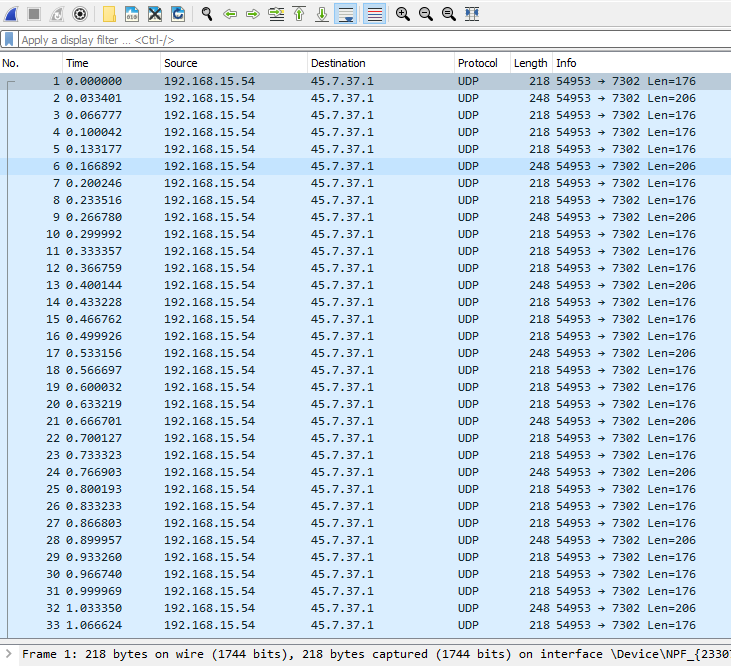


Figura 6: Resultado da captura com o segundo filtro

Essa captura pode então ser exportada como um arquivo no formato ***.pcap***, que será utilizado para a análise.

* 1. **Análise**

Para que seja possível a simulação de tráfego baseada no conteúdo capturado, é necessário primeiramente verificar como se comportam esses pacotes. Para isso, utilizou-se a ferramenta Jupyter, onde é possível processar os pacotes utilizando-se a linguagem de programação Python, e visualizar interativamente gráficos estatísticos sobre sua distribuição.

Inicialmente, foi feita a leitura do conteúdo exportado pelo Wireshark utilizando-se a biblioteca de processamento chamada DPKT. As informações obtidas foram então utilizadas para popular um dataframe contendo os dados mais relevantes para a proposta. Esses são:

* **SIZE**: Coluna contendo o tamanho do pacote em bytes.
* **IDT** (*inter-departure-time*): Coluna processada, representa o tempo decorrido entre o envio deste pacote, e o anterior.
* **PACKET**: Coluna contendo o próprio pacote, pode ser modificado para gerar novos pacotes simulados posteriormente.

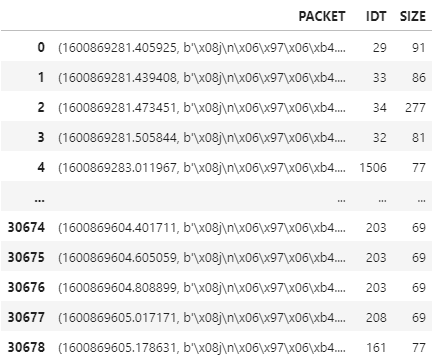


Figura 7: Dataframe gerado a partir do conteúdo capturado.

Com esse dataframe em mãos, é possível iniciar algumas investigações, como por exemplo, a verificação sobre uma possível correlação entre o tamanho dos pacotes, e seu tempo de envio. Isso é útil, pois pode indicar se existe uma função que modela claramente essa relação, facilitando a geração de tráfego simulado.

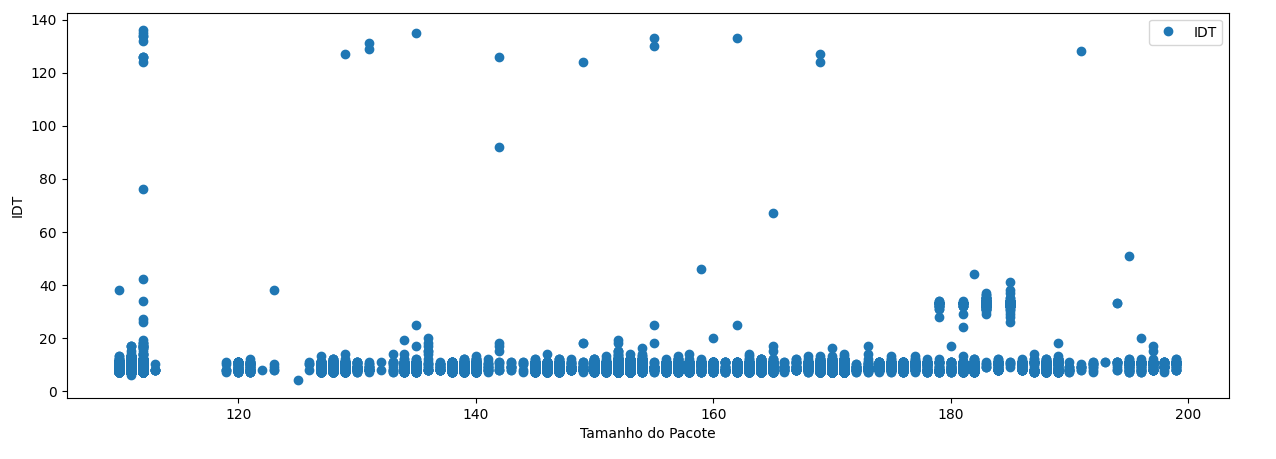


Figura 8: Relação IDT x Tamanho dos pacotes capturados

De uma análise visual do gráfico, é possível inferir que não há relação clara entre o tamanho dos pacotes, e seu tempo de envio, portanto, será descartada a opção de modelagem através de uma função que relaciona os dois parâmetros e será usada uma modelagem do tráfego de acordo com cada parâmetro individualmente.

* + 1. **Transformação da distribuição dos pacotes em relação ao tamanho**

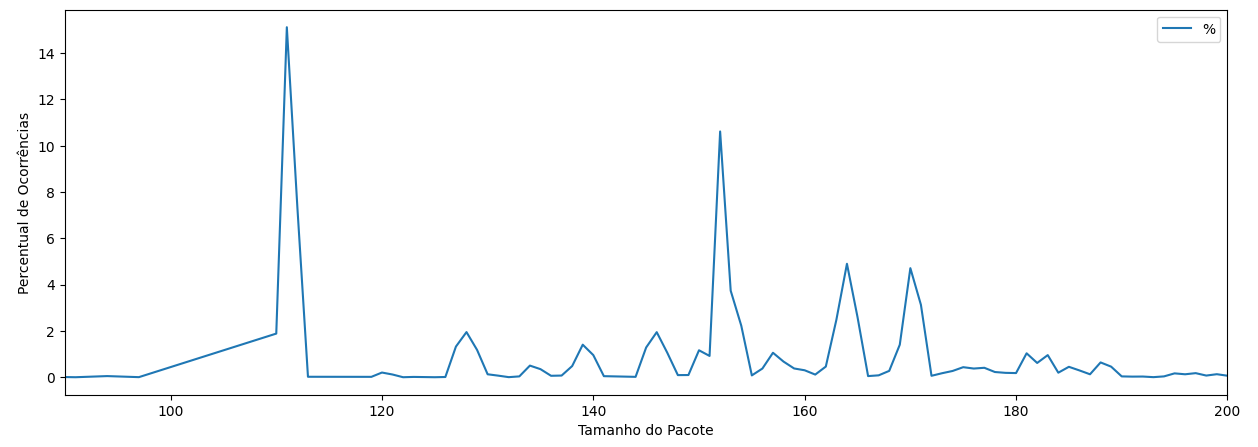
Para verificar como se comporta a distribuição dos pacotes em relação a seu tamanho, foi feita uma agregação em relação a esse parâmetro em cima do dataframe original. Verificando a taxa de frequência com que cada tamanho de pacote é produzido, é possível determinar quais são os tamanhos de pacote com maior ocorrência.

Figure 9: PDF (probability density function), representa o percentual de ocorrência de cada pacote em relação a seu tamanho.

É possível perceber que, apesar de existir uma grande variação na quantidade de tamanhos possíveis para cada pacote, alguns valores se revelam predominantes na distribuição de probabilidade. Para simplificar o processo de geração de pacotes que será feita, aplica-se sucessivos filtros de média móvel ponderada (WMA), para remover os pacotes menos significativos do conjunto [3].

Esse filtro é aplicado da seguinte maneira:

Cada ponto da média móvel ponderada, é calculado com uma soma ponderada composta por elementos de uma janela de vizinhos, de forma que quanto mais distante o vizinho, menor é o seu fator multiplicativo na média.

Por exemplo, se pegamos o ponto SIZE = 10 no dataframe de distribuição percentual, seu valor correspondente na média móvel ponderada central será dado por:

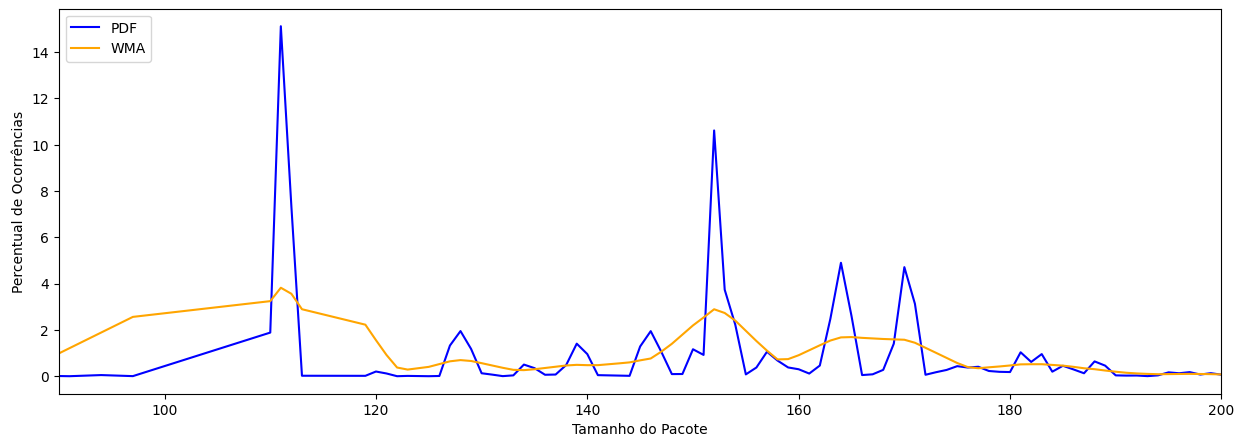
Assim, aplicando o procedimento para todo o dataframe, obtêm-se a seguinte curva em comparação com o original.

Figura 10: WMA (weighted moving average) em relação à PDF da qual foi produzida

Para a seleção dos pontos mais relevantes que serão utilizados, são escolhidos todos os picos de PDF que estejam acima da média. Esse novo conjunto de dados será denominado como dataset majoritário. Os outros pontos são então segregados, constituindo o dataset minoritário, e podendo passar por mais iterações do processo de seleção caso se deseje que o conjunto final de dados seja mais próximo dos dados coletados de fato.

Este processo de filtragem é importante, pois, como observado, existe uma variação muito grande de tipos e tamanho de pacotes que são trocados entre um cliente e um servidor durante a partida, e queremos simplificar ao máximo o conjunto de possíveis pacotes a serem simulados.

Para a formação do dataset majoritário, foi realizado duas iterações de seleção dos pacotes com mais ocorrência que a média, sendo a segunda feita sobre o dataset minoritário, para pegar novamente os pontos mais relevantes.

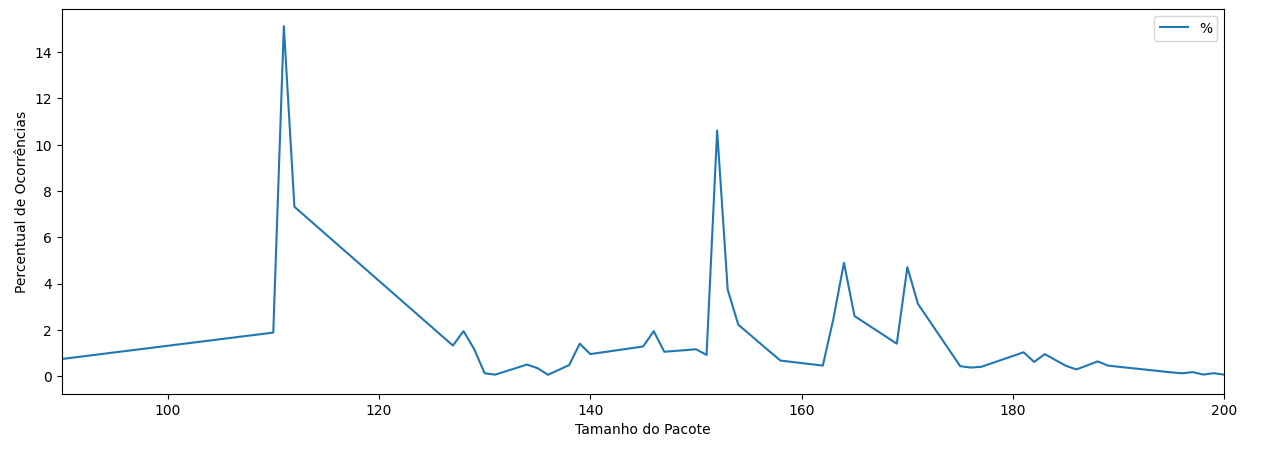
O dataset obtido contém apenas os pacotes com os principais tamanhos filtrados, e será utilizado na próxima etapa pelo gerador de tráfego simulado.

Figure 11: PDF do dataset majoritário usado para geração de pacotes por tamanho.

Com os pacotes devidamente filtrados em relação aos tamanhos mais relevantes, é então repetido o processo para obter os pacotes mais significativos quanto ao IDT.

* + 1. **Transformação da distribuição dos pacotes em relação ao tempo de envio**

Assim como na análise do conjunto de dados principal em relação ao Tamanho dos pacotes, é feita uma análise equivalente em relação ao tempo de envio (IDT). Para isso, é gerado o dataset de distribuição de probabilidade, e comparado com sua média móvel ponderada para determinar o dataset majoritário.

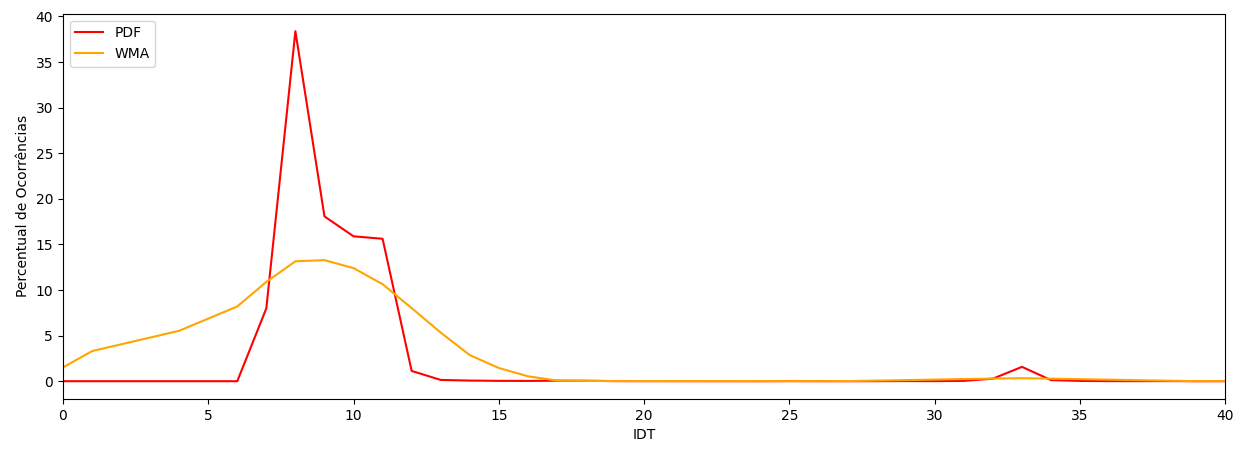


Figura 12: WMA em relação à PDF da qual foi produzida

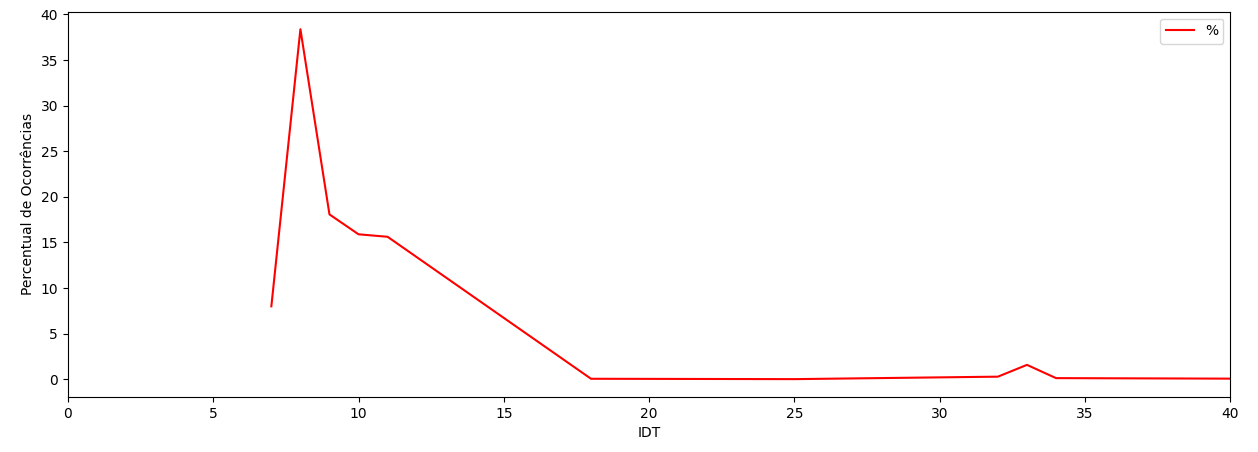
Novamente, são feitas duas iterações do procedimento, a primeira selecionando os elementos acima da média da distribuição de probabilidade (PDF), e a segunda aplicando novamente o processo sobre os pontos que ficaram abaixo da média (dataset minoritário).

Figura 13: PDF do dataset majoritário usado para geração de IDTs.

O dataset resultante contém apenas os tempos de envio mais relevantes, e será utilizado como base para a geração de IDTs durante a simulação.

* 1. **Reprodução de tráfego**

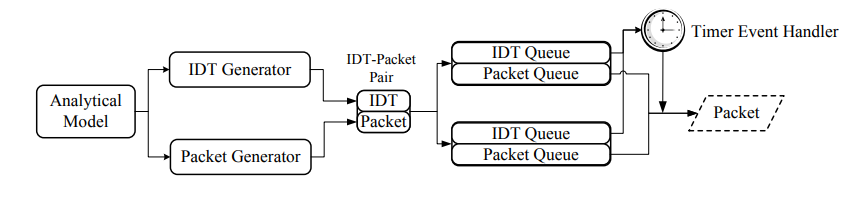
Com a distribuição desejada em mãos, tanto para o tamanho dos pacotes, quanto para o IDT, é possível agora gerar novos pacotes simulando inputs de jogadores. Para isso, é utilizado uma arquitetura definida em [2], que consiste em conectar dois geradores, um de IDTs e um de Pacotes, baseados na distribuição previamente definida nos datasets majoritários. Esse gerador funciona da seguinte forma:

Figura 14: Arquitetura do gerador de pacotes com tempo de partida. [2]

A partir do modelo analítico composto pelos dois datasets majoritários, são criados dois geradores que buscam elementos aleatórios dentro desses datasets, obtendo assim, um dos possíveis valores para IDT e um possível pacote entre os possíveis tamanhos. Em seguida, estes dois atributos são associados para criar um par IDT-PACOTE, e popular uma de duas filas, definidas como buffers. Quando um desses dois buffers está devidamente preenchido com seus pares, ele se torna um buffer disponível para consumo, isto é, pode ser usado para de fato simular um cliente enviando pacotes a determinados intervalos para algum servidor. A arquitetura de duplo buffer é útil, pois permite que o preenchimento de um buffer seja feito sempre daquele que não está sendo consumido.

Em paralelo ao gerador de pacotes, há também em execução o consumidor. Essa thread constantemente verifica se há algum buffer pronto para ser utilizado, e quando disponível, retira cada pacote individualmente da fila, aguardando um período de tempo especificado por seu par IDT. Dessa forma é possível simular pacotes sendo enviados com os períodos corretos de intervalo para o servidor.

Essa arquitetura pode ser usada de forma que cada jogador a ser simulado tenha suas próprias threads de produção e consumo de pacotes, ou então, pode ser usado um único gerador com buffer inflado, para que todos os jogadores simulados obtenham seus pacotes da mesma fila.

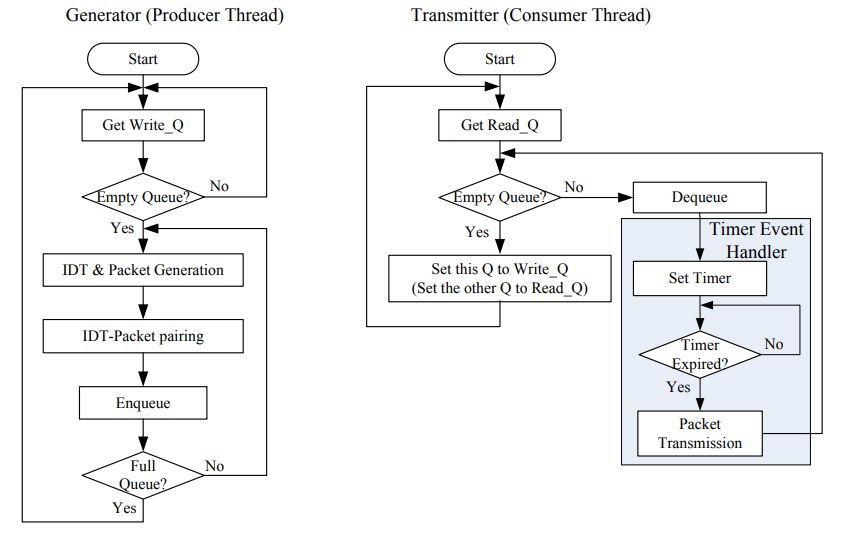


Figura 15: Arquitetura das threads geradora e consumidora de tuplas IDT-PACOTE. [2]

1. **Resultados**

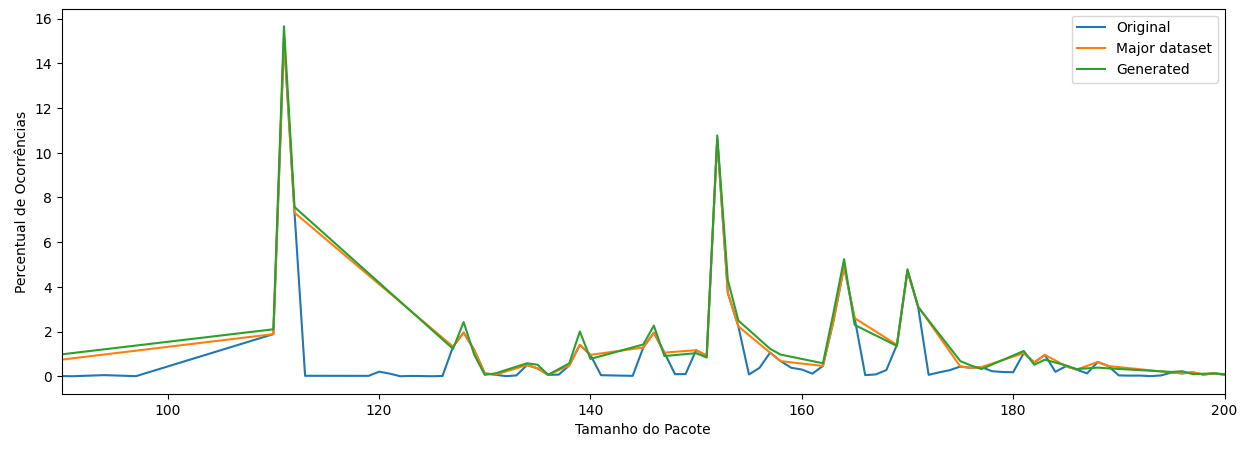
A ferramenta de geração de tráfego foi executada durante um período de 30 segundos, produzindo aproximadamente 3000 tuplas IDT-PACOTE. O resultado foi então processado em um dataset de distribuição de probabilidade e comparado com os dados originais e os datasets majoritários pós filtro.

Figura 16: Resultado comparativo entre PDFs Original, Majoritário e simulado, em relação ao tamanho dos pacotes.

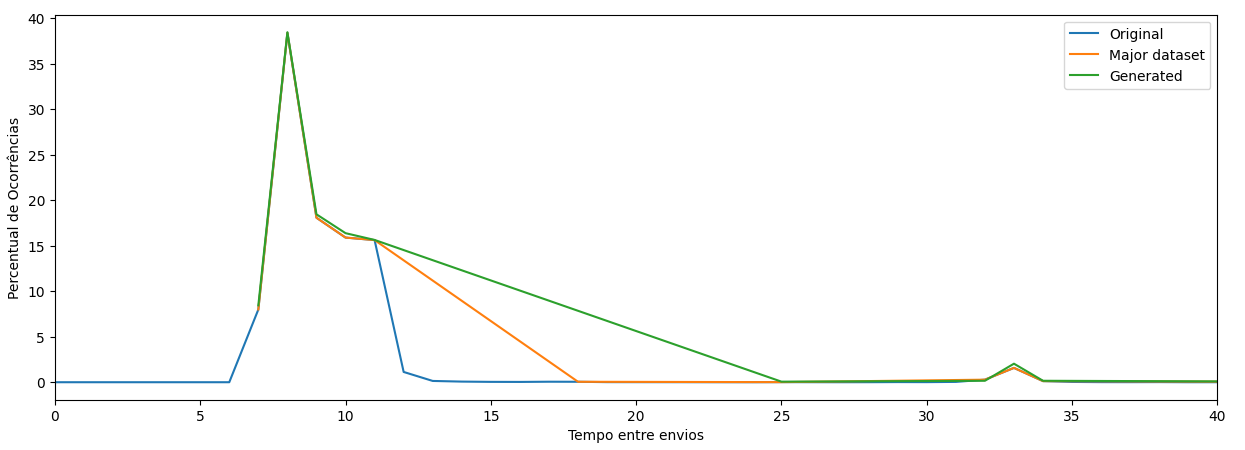


Figura 17: Resultado comparativo entre PDFs Original, Majoritário e simulado, em relação ao tempo de envio entre pacotes.

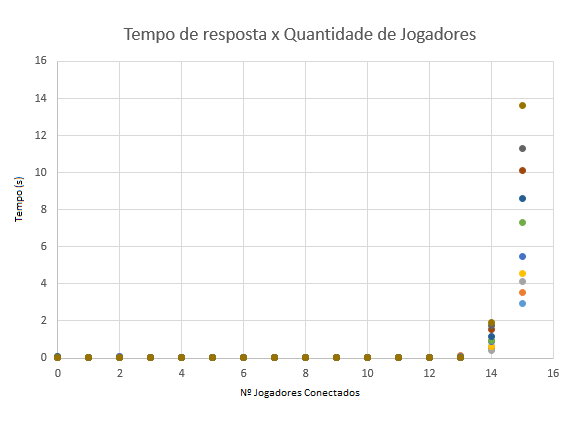
1. **Aplicação Prática**

Dado que o gerador é capaz de produzir um resultado próximo ao modelado no dataset majoritário, é possível fazer uso da ferramenta para de fato avaliar o desempenho de uma aplicação de jogos sem a necessidade de múltiplos usuários reais se conectarem. Porém esse teste requer alguns acessos especiais ao servidor de um jogo, pois não é possível simplesmente enviar pacotes falsos para um jogo disponível no mercado, dados que esses seriam tratados como pacotes inválidos, sem ter qualquer tipo de associação com um usuário real dentro de uma seção. Para isso, será utilizado um jogo ao qual há acesso ao código fonte e que utiliza um sistema cliente-servidor baseado no framework .NET juntamente com a plataforma Unity.

Para que seja possível utilizar a ferramenta num cenário prático, é necessário conhecer os procedimentos empregados na autenticação de um usuário do jogo escolhido. Com a disponibilidade ao código fonte de um jogo prototipado em mãos, é possível fazer a captura dos pacotes comunicados ao servidor (novamente utilizando Wireshark), e determinar corretamente quais fazem parte do handshake cliente-servidor. Isso permite que sejam feitas alterações nesses pacotes, para enganar o servidor a pensar que determinado usuário é de fato real, e, portanto, aceitar pacotes provenientes dele.

Assim, foi estabelecido uma rotina de autenticação falsa de usuários para o jogo escolhido, que é executada previamente, antes de qualquer pacote com comandos de jogo ser de fato gerado e enviado ao servidor.

Para analisar a responsividade do jogo em relação ao número de usuários conectados em simultâneo, foi realizado um teste de delay de resposta, onde foi medido o tempo entre o jogador principal enviar um comando de PULO para o servidor, e a resposta fazer de fato com que o avatar controlado iniciasse o movimento. Esse teste só é possível dada a característica do jogo escolhido ser servidor-autoritativo, isto é, todos os comandos enviados devem ser aprovados pelo servidor, que então atualiza a posição/atributos dos avatares.

Dez valores de delay foram coletados para cada número de usuários simulados na partida, obtendo o seguinte resultado:

Além desse teste, foi também avaliado um teste empírico, onde um jogador real se conectou ao servidor, e em seguida foram sendo adicionados bots individualmente em sucessão. Com isso notou-se que, com até 5 jogadores, o jogo mostrou-se totalmente responsivo, sem que o usuário fosse capaz de perceber qualquer alteração em relação a estar jogando sozinho. A partir de 5 a 10 jogadores, notou-se um crescente aumento no tempo de resposta entre um comando enviado e sua resposta, dependendo de quão interativo era o jogador principal. A partir de 14 Jogadores, observou-se um delay cumulativo, que tornou impossível de se jogar com qualquer resposta a tempo viável para os controles.

1. **Conclusão**

Dos resultados obtidos na geração de pacotes para o Jogo Valorant, foi possível observar que a ferramenta conseguiu reproduzir o comportamento do dataset após o filtro de maneira idêntica ao esperado, e que, apesar de não ser possível avaliar diretamente o servidor desse jogo, temos o indicativo de que a carga produzida por um usuário real seria bem simulada usando essa arquitetura.

Já com a análise do servidor do jogo protótipo, observou-se de fato a possibilidade de verificar um gargalo quanto a capacidade de processamento do servidor. Neste teste foi possível verificar que a partir de 14 jogadores ativos, a fila de tratamento de comandos do jogo deixa de ser capaz de entregar um resultado satisfatório, comprometendo a jogabilidade dos usuários.

Com este trabalho, foi possível executar uma série de procedimentos que constituem o ferramental de um simulador de tráfego online, exercendo os conhecimentos adquiridos na área de captura de tráfego na camada de transporte, processamento e ciência de dados, e comunicação web.

Com o resultado final, foi visto que as estratégias aplicadas, de fato se mostraram viáveis para a real determinação do número de usuários suportados por um servidor de jogos, sem a necessidade real de suas conexões.

Todos os documentos utilizados para a elaboração deste projeto encontram-se disponíveis em [14], e têm sua arquitetura descrita em Anexo – A (Descrição da estrutura de arquivos do projeto).

1. **Referências Bibliográficas**

[1] Superdata 2018 - Year in review: <https://adindex.ru/files2/access/2019_01/230617_SuperData%202018%20Year%20in%20Review.pdf>

[2] Shin K., Kim J., Sohn K., Park C., Choi S. (2010) Online Gaming Traffic Generator for Reproducing Gamer Behavior: <https://hal.inria.fr/hal-01055646/document>

[3] Shin, K., Kim, J., Sohn, K., Park, C., Choi, S.: Transformation Approach to Model Online Gaming Traffic: <https://onlinelibrary.wiley.com/doi/pdf/10.4218/etrij.11.1510.0087>

[4] Wireshark: <https://www.wireshark.org/>

[5] What is Wireshark, NetworkWorld: <https://www.networkworld.com/article/3663021/what-is-wireshark.html>

[6] Jupyter Project: <https://jupyter.org/>

[7] Jupyter Kernel options: <https://docs.jupyter.org/en/latest/projects/kernels.html>

[8] IPython Project: <https://ipython.org/>

[9] DPKT Python Library: <https://dpkt.readthedocs.io/en/latest/>

[10] Pandas Python Library: <https://pandas.pydata.org/>

[11] MatplotLib Python Library: <https://matplotlib.org/>

[12] Valorant Game port Forwarding Guide: <https://support-valorant.riotgames.com/hc/en-us/articles/4402306473619-How-to-Set-Up-Port-Forwarding>

[13] Wireshark Capture Filters: <https://wiki.wireshark.org/CaptureFilters>

[14] Github Project Repository: <https://github.com/LucBollani/EA006-TCC>

**ANEXO A – Descrição da estrutura de arquivos do projeto**

**.**

**├── TFC/**

**│ ├── client\_packets/**

**│ │ │** Diretório contendo capturas de pacotes pequenas, utilizadas

│ │ │ durante desenvolvimento do projeto

│ │ │

**│ │ ├── client\_emptyudp.pcap**

**│ │ │** Captura de pacote UDP sem dados, utilizado como base para a

**│ │ │** criação de novos pacotes

│ │ │

**│ │ ├── client\_player1.pcap**

**│ │ ├── client\_player2.pcap**

**│ │ ├── client\_player3.pcap**

**│ │ │** Captura de comportamento de jogador, usadas como teste

**│ │ │** para reproduzir exatamente os movimentos de um jogador num bot

│ │ │

**│ │ ├── client\_welcomereceived.pcap**

**│ │ │** Captura de pacote de handshake entre cliente e servidor,

**│ │ │** usado para autenticar um usuário no servidor do jogo

│ │ │

**│ │ └── readme.txt**

**│ │** Arquivo de descrição do conteúdo do diretório

│ │

**│ ├── FPSGAME\_client.pcap**

**│ │** Captura de tráfego do jogo personalizado, utilizada para

**│ │** análise, simulação e avaliação final da ferramenta produzida no

**│ │** projeto

│ │

**│ ├── NB 1 - VALORANT.ipynb**

**│ │** Notebook Jupyter contendo todos os códigos e resultados da

**│ │** análise do primeiro jogo tratado: Valorant

│ │

**│ ├── NB 2 - CUSTOM\_GAME.ipynb**

**│ │** Notebook Jupyter contendo todos os códigos e resultados da

**│ │** análise do segundo jogo tratado: Jogo Personalizado

│ │

**│ ├── Relatorio.docx**

**│ │** Relatório final apresentado para a disciplina

│ │

**│ ├── Valorant\_TDM\_client\_UDP.pcap**

**│ │** Captura de tráfego do jogo Valorant numa partida online,

**│ │** utilizada para análise e simulação da ferramenta produzida no

**│ │** projeto

│ │

**│ ├── img1.PNG**

**│ │** Logotipo da faculdade

│ │

**│ └── requirements.txt**

**│** Lista de bibliotecas python e suas versões utilizadas no projeto

│

**└── Readme.md**

Descrição inicial do projeto e conteúdo do repositório