

# MC833 Laboratório de redes de computadores

## Relatório do projeto 2

### Analizando tráfego de rede com Wireshark e Mininet

Daniel de Sousa Cipriano RA: 233228

Maio 2025

## 1 Objetivo geral e ferramentas

O objetivo do projeto foi implementar e analisar uma simulação de rede simples, utilizando os softwares Mininet e Wireshark. O Mininet é um emulador que cria redes virtuais, utilizando recursos do kernel Linux, como namespaces de redes e virtualização baseada em processos. O Mininet suporta a criação de hosts, que são representados por processos de bash, rodando em um namespace de rede, cada um com sua própria interface de rede e seus próprios processos. Também é possível criar switchs virtuais com Mininet, baseados em Open vSwitch ou OpenFlow. Em um ambiente de rede Mininet, são gerados pares Ethernet virtuais, que representam links ligando os hosts emulados e os switchs emulados[1].

O Wireshark é um software "farejador" de rede, ele fornece uma interface gráfica para análise de tráfego, com filtros para interfaces específicas, protocolos específicos e detalhamento de camadas de rede.

## 2 Construção da rede virtual

Para implementação do projeto, foi utilizada a máquina virtual do Mininet, que contém todas as dependências de software do Mininet e roda em um sistema Linux baseado em Ubuntu[2]. A máquina virtual Mininet também inclui o Wireshark e um servidor SSH para acesso remoto. Utilizando o comando `sudo mn --topo single,4`, foi inicializado um ambiente de rede, com um switch e quatro hosts virtuais. O switch está na interface s1, enquanto os hosts estão nas interfaces h1, h2, h3, h4. Posteriormente, foi utilizado o emulador de terminal Xterm, que permite utilizar a linha de comando para rodar programas em modo gráfico. Com Xterm, é possível dividir um terminal para cada nó da rede virtual e executar o Wireshark dentro do contexto de processo específico.

### 3 Simulação de tráfego na rede e análise com Wireshark

Para simulação de tráfego na rede, foi utilizado o comando ping do Mininet, que gera tráfego direcionado. Foram executados os seguintes comandos:

- h1 ping -c 200 h3 que direciona 200 pacotes de ping do host h1 para o host h3.
- h2 ping -c 200 h4 que direciona 200 pacotes de ping do host h2 para o host h4.

Foram realizadas quatro análises utilizando Wireshark com filtro para pacotes ICMP(Internet Control Message Protocol). Uma análise rodou na interface de rede do host h1, outra na interface do host h2 e as duas últimas no switch s1. Os resultados das respectivas análises podem ser vistos nas Figuras 1, 2, 3 e 4. Podemos visualizar a troca de requests e replies ICMP entre os hosts, com exibição de informações como número de sequência do pacote, IP fonte e IP destino, TTL(Time to Live) que representa o número de saltos que podem ocorrer na transmissão antes do pacote ser descartado, dados do protocolo e etc.

### 4 Análise de dados em capturas com Scapy

Com o intuito de obter uma análise mais detalhada do tráfego capturado, foram salvos arquivos de capturas geradas pelas interfaces do host h1 e do host h2, ambos no formato pcap. A ferramenta utilizada para estudo foi a biblioteca Scapy do Python, que fornece um amplo conjunto de funcionalidades para análise de tráfego em redes. Com Scapy é possível configurar pacotes, capturar tráfego, enviar tráfego e processar capturas, adicionando e lendo uma grande variedade de camadas de rede, com acesso à atributos específicos para cada uma delas[3]. Juntamente com Scapy, foi utilizada a biblioteca matplotlib, que permite plotar gráficos de diversos tipos diferentes, o que auxiliou na visualização e interpretação dos resultados obtidos.

Para processar os pacotes capturados, foram implementados os scripts Python, `capture_handler.py`, `packet_analyzer.py`, `icmp_analyzer.py` e `ip_analyzer.py`. A lógica de construção segue o princípio de herança entre classes da programação orientada a objetos, com a classe abstrata `PacketAnalyzer` atuando como superclasse que processa pacotes do tipo `pcap` e define métodos para extração de dados e métricas importantes de rede. As subclasses `IcmpAnalyzer` e `IpAnalyzer` herdaram de `PacketAnalyzer` e implementam seus próprios métodos para análise específica de protocolo. O script `capture_handler.py` cria as instâncias necessárias e executa os métodos de análise. Uma visão geral do objetivo e local de implementação de cada classe pode ser vista a seguir:

- `PacketAnalyzer` em `packet_analyzer.py`. Essa classe implementa métodos gerais para análise de capturas, com foco em capturas com pacotes diversificados.
- `IcmpAnalyzer` em `icmp_analyzer.py`. Essa classe implementa métodos para análise detalhada de capturas com filtro para a camada de protocolo ICMP.
- `IpAnalyzer` em `ip_analyzer.py`. Essa classe implementa métodos para análise detalhada de capturas com filtro para a camada de protocolo IP.

A intenção dessa abordagem é permitir uma futura extensão para englobar análise de diversos protocolos de rede. Alguns métodos que podem possuir importância em mais de um contexto de protocolo, como o método que retorna o IPv4 de origem e destino na camada IP, foram definidos como estáticos para facilitar o uso entre subclasses de PacketAnalyzer. Foi também implementada a classe GraphPlotter em `graph_plotter.py`, que fornece métodos para plotagem de tipos variados de gráficos, utilizando matplotlib. As implementações das classes principais de análise, com comentários explicativos, podem ser vistas no Apêndice B.

## 5 Análise de resultados

Foram extraídos dados das capturas, para construção de gráficos e análise de métricas importantes de tráfego em redes. Primeiramente, foram obtidas métricas gerais da captura, como total de pacotes e camadas de protocolos de rede. Após isso, foram analisadas métricas específicas para o protocolo ICMP. Os resultados gerais para as duas capturas realizadas, podem ser vistos nas Figuras 5 e 6. Percebe-se que o padrão de tempo obtido para as medições, tende a transitar entre valores muito baixos. Isso ocorre pois a rede simulada pelo Mininet não possui latência física real, já que todas as ligações de rede são simuladas na mesma máquina. Ao gerar os gráficos de métricas temporais por número de sequência ICMP, foi visualizada em alguns momentos a ocorrência de um viés de borda, com os primeiros e últimos valores divergindo da média. Esse comportamento pode ocorrer devido à simulação rodar em ambiente virtualizado, com fatores como inicialização e encerramento de buffers e namespaces, cache ARP ainda não resolvido e outros efeitos colaterais de inicialização e encerramento de ambiente, causando aumento de latência. Então, para evitar inconsistências na análise, foram excluídos os dez primeiros e os dez últimos pacotes das capturas.

Foram analisados 380 pacotes para cada captura, totalizando 37240 bytes cada. O throughput medido, que representa a taxa real de transferência de dados que a rede virtual consegue efetuar, foi de 0,0015 Mbps. O IPv4 dos hosts foram confirmados como 10.0.0.1 para o host h1, 10.0.0.2 para o host h2, 10.0.0.3 para o host h3 e 10.0.0.4 para o host h4. Os pacotes das capturas possuem as camadas de protocolo Ethernet, IP, ICMP e Raw. Os gráficos de quantidade de pacotes por camada de protocolo para cada captura, podem ser vistos nas Figuras 7 e 8. A biblioteca Scapy fornece uma função que gera um diagrama visual e detalhado de todas as camadas de protocolos encontradas na captura, incluindo atributos de cada protocolo e a representação hexadecimal dos dados. Os diagramas de camadas de protocolo dos pacotes enviados pelos hosts h1, h3, h2 e h4 podem ser vistos respectivamente nas Figuras 9, 10, 11 e 12.

### 5.1 Análise de RTT

O RTT(Round Trip Time) representa o tempo entre a captação da requisição e o recebimento da resposta na interface base. A análise de RTT é importante para medir a latência de rede. RTT muito alto ou com flutuação muito grande, pode indicar falhas como alto congestionamento de rede, problemas de roteamento, problemas na infraestrutura da rede, etc. Em média, um RTT acima de 200 ms é considerado alto e prejudicial. Na análise de captura da interface do host h1, o RTT médio obtido foi de  $(0,057 \pm 0,001)$  ms, com desvio padrão de 0,018 ms. Enquanto que, na análise de captura da interface do host h2, o RTT médio obtido foi de  $(0,066 \pm 0,001)$  ms, com desvio padrão de 0,015 ms. O coeficiente de variação, que representa a porcentagem de desvio

padrão na média, foi de 32,03% na primeira captura e de 22,31% na segunda captura. Esse valor baixo indica latência normal de ping na rede. Os gráficos de RTT por número de sequência dos pacotes ICMP podem ser vistos nas Figuras 13 e 14. As frequências por faixas de RTT podem ser vistas nos histogramas de RTT nas Figuras 15 e 16.

## 5.2 Análise de intervalos entre chegadas de pacotes ICMP request

Analizar o tempo de chegada entre pacotes, é importante para assegurar o funcionamento correto do emissor e captar latência de rede. O intervalo médio obtido para a captura na interface em h1, foi de  $(1024,00 \pm 0,02)$  ms, com desvio padrão de 0,24 ms. Os mesmos resultados foram obtidos para a captura da interface em h2. O mesmo ocorreu para a medição do coeficiente de variação, com ambas as capturas marcando o valor de 0,02%. Essas métricas sinalizam alta homogeneidade de dados e perfeita correlação entre as capturas. Os gráficos de intervalos entre chegadas de pacotes ICMP request por número de sequência dos pacotes ICMP, podem ser vistos nas Figuras 17 e 18. As distribuições de intervalos para cada captura podem ser vistas nos histogramas das Figuras 19 e 20.

## 5.3 Análise de jitter

O jitter é uma métrica de rede que representa a variação em um conjunto de medidas temporais. As duas principais medições de jitter vem da variação na taxa de RTT e da variação no intervalo de chegada entre pacotes. A análise de variação do RTT é importante para identificar o impacto da latência, do congestionamento e das filas em roteadores na rede. Nesse caso, um valor normal de jitter que não cause falhas de desempenho, gira entorno de 20 ms. Enquanto que, a análise da variação no intervalo de chegada entre pacotes, é utilizada para inspecionar estabilidade de tráfego contínuo, importante para garantir o desempenho de transmissão em serviços como de VoIP e streaming. Para essas funcionalidades, um valor reduzido de jitter é ainda mais importante, devendo variar numa faixa de 10 ms para que não haja prejuízo ao cliente. O valor médio de jitter baseado em RTT obtido para a captura da interface em h1 foi de  $(0,015 \pm 0,001)$  ms, com desvio padrão de 0,015 ms. A mesma métrica calculada para a captura da interface em h2 foi de  $(0,013 \pm 0,001)$  ms, com desvio padrão de 0,016 ms. O valor médio de jitter baseado em intervalo de chegada entre pacotes para a captura da interface do host h1 foi de  $(0,33 \pm 0,02)$  ms, com desvio padrão de 0,26 ms. Essa mesma métrica de jitter obtida para a captura da interface do host h2 foi de  $(0,34 \pm 0,02)$  ms, com desvio padrão de 0,24 ms.

Em ambas as medições, o coeficiente de variação se mostrou relativamente alto, com resultados de 98,56% para o jitter baseado em RTT na primeira captura, 80,31% para o jitter baseado em intervalo de chegada entre pacotes na primeira captura, 126,75% para o jitter baseado em RTT na segunda captura e 69,96% para o jitter baseado em intervalo de chegada entre pacotes na segunda captura. Esse comportamento reflete uma dispersão grande nas medidas de jitter. Essas flutuações podem ser causadas devido à virtualização da simulação, já que as métricas temporais são baseadas no relógio do sistema e são sensíveis ao estado do kernel. Se o sistema hospedeiro estiver carregado ou rodando em segundo plano, threads de envio e resposta podem ser interrompidas, gerando atrasos irregulares, mesmo sem tráfego real de rede. Os gráficos dos dois tipos de jitter por número de sequência dos pacotes ICMP para as duas capturas, podem ser visualizados nas Figuras 21, 22, 23 e 24. Juntamente com os histogramas de jitter nas Figuras 25, 26, 27 e 28.

## **5.4 Análise de perda de pacotes**

Comparando as requisições ICMP com suas respectivas respostas esperadas, foi possível obter estatísticas acerca da perda de pacotes em cada captura. Todos os 380 pacotes analisados foram entregues ao respectivo destinatário em ambas as capturas, totalizando 190 pacotes ICMP request e 190 pacotes ICMP reply. Os gráficos com estatísticas de perda de pacotes podem ser vistos nas Figuras 29 e 30. Os gráficos que mostram a taxa de perda de pacotes podem ser vistos nas Figuras 31 e 32.

## **6 Conclusão**

A implementação do projeto permitiu obter contato com eficientes ferramentas de análise de rede, através da captura de pacotes com Wireshark e da construção de uma rede virtual com Mininet. Utilizando a biblioteca Scapy do Python, foi possível implementar um ambiente de classes relacionadas, seguindo paradigmas da programação orientada a objetos para criar uma lógica extensível com analisadores de pacotes para camadas específicas de protocolos de rede. Utilizando matplotlib e um script de plotagem em conjunto com métodos de extração de dados, foi possível obter estatísticas e gerar gráficos de variadas métricas de rede, o que possibilitou o entendimento da importância e melhor visualização real de como essas medidas são utilizadas em análises de redes.

# Apêndice

## A Figuras

No.	Time	Source	Destination	Protocol	Length	Info
383	195.547308812	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0622, seq=192/49152, ttl=64 (reply ...
384	195.547378012	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0622, seq=192/49152, ttl=64 (reques...
385	196.571313730	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0622, seq=193/49408, ttl=64 (reply ...
386	196.571382419	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0622, seq=193/49408, ttl=64 (reques...
387	197.595394952	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0622, seq=194/49664, ttl=64 (reply ...
388	197.595394952	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0622, seq=194/49664, ttl=64 (reques...
389	198.619113892	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0622, seq=195/49920, ttl=64 (reply ...
390	198.619180998	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0622, seq=195/49920, ttl=64 (reques...
391	199.643302045	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0622, seq=196/50176, ttl=64 (reply ...
392	199.643371476	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0622, seq=196/50176, ttl=64 (reques...
393	200.667000196	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0622, seq=197/50432, ttl=64 (reply ...
394	200.667051862	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0622, seq=197/50432, ttl=64 (reques...
395	201.691301650	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0622, seq=198/50688, ttl=64 (reply ...
396	201.691372073	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0622, seq=198/50688, ttl=64 (reques...
397	202.715123464	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0622, seq=199/50944, ttl=64 (reply ...
398	202.715185711	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0622, seq=199/50944, ttl=64 (reques...
399	203.738865896	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0622, seq=200/51200, ttl=64 (reply ...
400	203.738894870	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0622, seq=200/51200, ttl=64 (reques...

▶ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface h1-eth0, id 0  
 ▶ Ethernet II, Src: f6:9b:d4:80:4c:b4 (f6:9b:d4:80:4c:b4), Dst: f6:44:61:3f:a3:9a (f6:44:61:3f:a3:9a)  
 ▶ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.3  
 ▶ Internet Control Message Protocol

Figura 1: Captura de tráfego com Wireshark, na interface de rede do host h1

No.	Time	Source	Destination	Protocol	Length	Info
383	195.563194877	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x0686, seq=192/49152, ttl=64 (reply ...
384	195.563246855	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0686, seq=192/49152, ttl=64 (reques...
385	196.586949309	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x0686, seq=193/49408, ttl=64 (reply ...
386	196.587018509	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0686, seq=193/49408, ttl=64 (reques...
387	197.610997706	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x0686, seq=194/49664, ttl=64 (reply ...
388	197.611064552	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0686, seq=194/49664, ttl=64 (reques...
389	198.635088998	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x0686, seq=195/49920, ttl=64 (reply ...
390	198.635156004	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0686, seq=195/49920, ttl=64 (reques...
391	199.658964127	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x0686, seq=196/50176, ttl=64 (reply ...
392	199.659030613	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0686, seq=196/50176, ttl=64 (reques...
393	200.687134193	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x0686, seq=197/50432, ttl=64 (reply ...
394	200.687199075	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0686, seq=197/50432, ttl=64 (reques...
395	201.706960410	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x0686, seq=198/50688, ttl=64 (reply ...
396	201.707033928	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0686, seq=198/50688, ttl=64 (reques...
397	202.731224004	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x0686, seq=199/50944, ttl=64 (reply ...
398	202.731290029	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0686, seq=199/50944, ttl=64 (reques...
399	203.755244437	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x0686, seq=200/51200, ttl=64 (reply ...
400	203.755310241	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x0686, seq=200/51200, ttl=64 (reques...

▶ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface h2-eth0, id 0  
 ▶ Ethernet II, Src: 26:a2:ee:38:f5:ac (26:a2:ee:38:f5:ac), Dst: ae:lc:fd:5b:aa:7d (ae:lc:fd:5b:aa:7d)  
 ▶ Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.4  
 ▶ Internet Control Message Protocol

Figura 2: Captura de tráfego com Wireshark, na interface de rede do host h2

No.	Time	Source	Destination	Protocol	Length	Info
383	195.555637501	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x08e5, seq=192/49152, ttl=64 (reply ...
384	195.555688597	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08e5, seq=192/49152, ttl=64 (reques...
385	196.579692444	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x08e5, seq=193/49408, ttl=64 (reply ...
386	196.579740094	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08e5, seq=193/49408, ttl=64 (reques...
387	197.603635738	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x08e5, seq=194/49664, ttl=64 (reply ...
388	197.603686082	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08e5, seq=194/49664, ttl=64 (reques...
389	198.627725172	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x08e5, seq=195/49920, ttl=64 (reply ...
390	198.627774605	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08e5, seq=195/49920, ttl=64 (reques...
391	199.652009807	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x08e5, seq=196/50176, ttl=64 (reply ...
392	199.652060463	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08e5, seq=196/50176, ttl=64 (reques...
393	200.675953789	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x08e5, seq=197/50432, ttl=64 (reply ...
394	200.675993533	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08e5, seq=197/50432, ttl=64 (reques...
395	201.699713655	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x08e5, seq=198/50688, ttl=64 (reply ...
396	201.699763458	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08e5, seq=198/50688, ttl=64 (reques...
397	202.723651982	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x08e5, seq=199/50944, ttl=64 (reply ...
398	202.723691657	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08e5, seq=199/50944, ttl=64 (reques...
399	203.747695456	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x08e5, seq=200/51200, ttl=64 (reply ...
400	203.747746486	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x08e5, seq=200/51200, ttl=64 (reques...

▶ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface s1-eth1, id 0  
 ▶ Ethernet II, Src: 6a:32:a4:cf:a2:8d (6a:32:a4:cf:a2:8d), Dst: 8e:14:94:49:e3:d7 (8e:14:94:49:e3:d7)  
 ▶ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.3  
 ▶ Internet Control Message Protocol

Figura 3: Captura de tráfego com Wireshark, na interface de rede do switch s1, visualizando tráfego do host h1 para o host h3

No.	Time	Source	Destination	Protocol	Length	Info
383	195.562843587	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x08f4, seq=192/49152, ttl=64 (reply ...
384	195.562895735	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x08f4, seq=192/49152, ttl=64 (reques...
385	196.586673010	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x08f4, seq=193/49408, ttl=64 (reply ...
386	196.586758852	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x08f4, seq=193/49408, ttl=64 (reques...
387	197.610673458	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x08f4, seq=194/49664, ttl=64 (reply ...
388	197.610746105	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x08f4, seq=194/49664, ttl=64 (reques...
389	198.635025077	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x08f4, seq=195/49920, ttl=64 (reply ...
390	198.635078026	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x08f4, seq=195/49920, ttl=64 (reques...
391	199.659037673	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x08f4, seq=196/50176, ttl=64 (reply ...
392	199.659090462	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x08f4, seq=196/50176, ttl=64 (reques...
393	200.682952448	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x08f4, seq=197/50432, ttl=64 (reply ...
394	200.683005728	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x08f4, seq=197/50432, ttl=64 (reques...
395	201.706945502	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x08f4, seq=198/50688, ttl=64 (reply ...
396	201.707000225	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x08f4, seq=198/50688, ttl=64 (reques...
397	202.730941018	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x08f4, seq=199/50944, ttl=64 (reply ...
398	202.730994168	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x08f4, seq=199/50944, ttl=64 (reques...
399	203.754940028	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0x08f4, seq=200/51200, ttl=64 (reply ...
400	203.754994410	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0x08f4, seq=200/51200, ttl=64 (reques...

▶ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface s1-eth2, id 0  
 ▶ Ethernet II, Src: 6e:40:87:7a:ae:25 (6e:40:87:7a:ae:25), Dst: 02:9f:9c:4e:64:ea (02:9f:9c:4e:64:ea)  
 ▶ Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.4  
 ▶ Internet Control Message Protocol

Figura 4: Captura de tráfego com Wireshark, na interface de rede do switch s1, visualizando tráfego do host h2 para o host h4

```
Capture h1-h3
Source IPv4: 10.0.0.1
Destination IPv4: 10.0.0.3
Total packets: 380
Total bytes: 37240 bytes
Layers: ['Ethernet', 'IP', 'ICMP', 'Raw']
Throughput: 0.0015 Mbps

Mean ICMP RTT: 0.057 ms
ICMP RTT standard deviation: 0.018 ms
Maximum ICMP RTT: 0.114 ms
Minimum ICMP RTT: 0.024 ms
Standard error: 0.001 ms
Percentage of standard deviation from the mean: 32.03%

Mean ICMP packets arrival time interval: 1024.00 ms
Standard deviation of ICMP packets arrival time interval: 0.24 ms
Maximum ICMP packet arrival time interval: 1024.90 ms
Minimum ICMP request packet arrival time Interval: 1023.49 ms
Standard error: 0.02 ms
Percentage of standard deviation from the mean: 0.02%

ICMP RTT based jitter mean: 0.015 ms
ICMP RTT based jitter standard deviation: 0.015 ms
ICMP RTT based maximum jitter: 0.076 ms
ICMP RTT based minimum jitter: 0.000 ms
Standard error: 0.001 ms
Percentage of standard deviation from the mean: 98.56%

ICMP arrival time interval based jitter mean: 0.33 ms
ICMP arrival time interval based jitter standard deviation: 0.26 ms
ICMP arrival time interval based maximum jitter: 1.32 ms
ICMP arrival time interval based minimum jitter: 0.00 ms
Standard error: 0.02 ms
Percentage of standard deviation from the mean: 80.31%

ICMP sent packets: 190
ICMP received packets: 190
ICMP lost packets: 0
ICMP loss rate: 0.0%
```

Figura 5: Dados obtidos na captura com interface de rede do host h1

```
Capture h2-h4
Source IPv4: 10.0.0.2
Destination IPv4: 10.0.0.4
Total packets: 380
Total bytes: 37240 bytes
Layers: ['Ethernet', 'IP', 'ICMP', 'Raw']
Throughput: 0.0015 Mbps

Mean ICMP RTT: 0.066 ms
ICMP RTT standard deviation: 0.015 ms
Maximum ICMP RTT: 0.149 ms
Minimum ICMP RTT: 0.036 ms
Standard error: 0.001 ms
Percentage of standard deviation from the mean: 22.31%

Mean ICMP packets arrival time interval: 1024.00 ms
Standard deviation of ICMP packets arrival time interval: 0.24 ms
Maximum ICMP packet arrival time interval: 1024.50 ms
Minimum ICMP request packet arrival time Interval: 1023.50 ms
Standard error: 0.02 ms
Percentage of standard deviation from the mean: 0.02%

ICMP RTT based jitter mean: 0.013 ms
ICMP RTT based jitter standard deviation: 0.016 ms
ICMP RTT based maximum jitter: 0.099 ms
ICMP RTT based minimum jitter: 0.000 ms
Standard error: 0.001 ms
Percentage of standard deviation from the mean: 126.75%

ICMP arrival time interval based jitter mean: 0.34 ms
ICMP arrival time interval based jitter standard deviation: 0.24 ms
ICMP arrival time interval based maximum jitter: 0.99 ms
ICMP arrival time interval based minimum jitter: 0.00 ms
Standard error: 0.02 ms
Percentage of standard deviation from the mean: 69.96%

ICMP sent packets: 190
ICMP received packets: 190
ICMP lost packets: 0
ICMP loss rate: 0.0%
```

Figura 6: Dados obtidos na captura com interface de rede do host h2

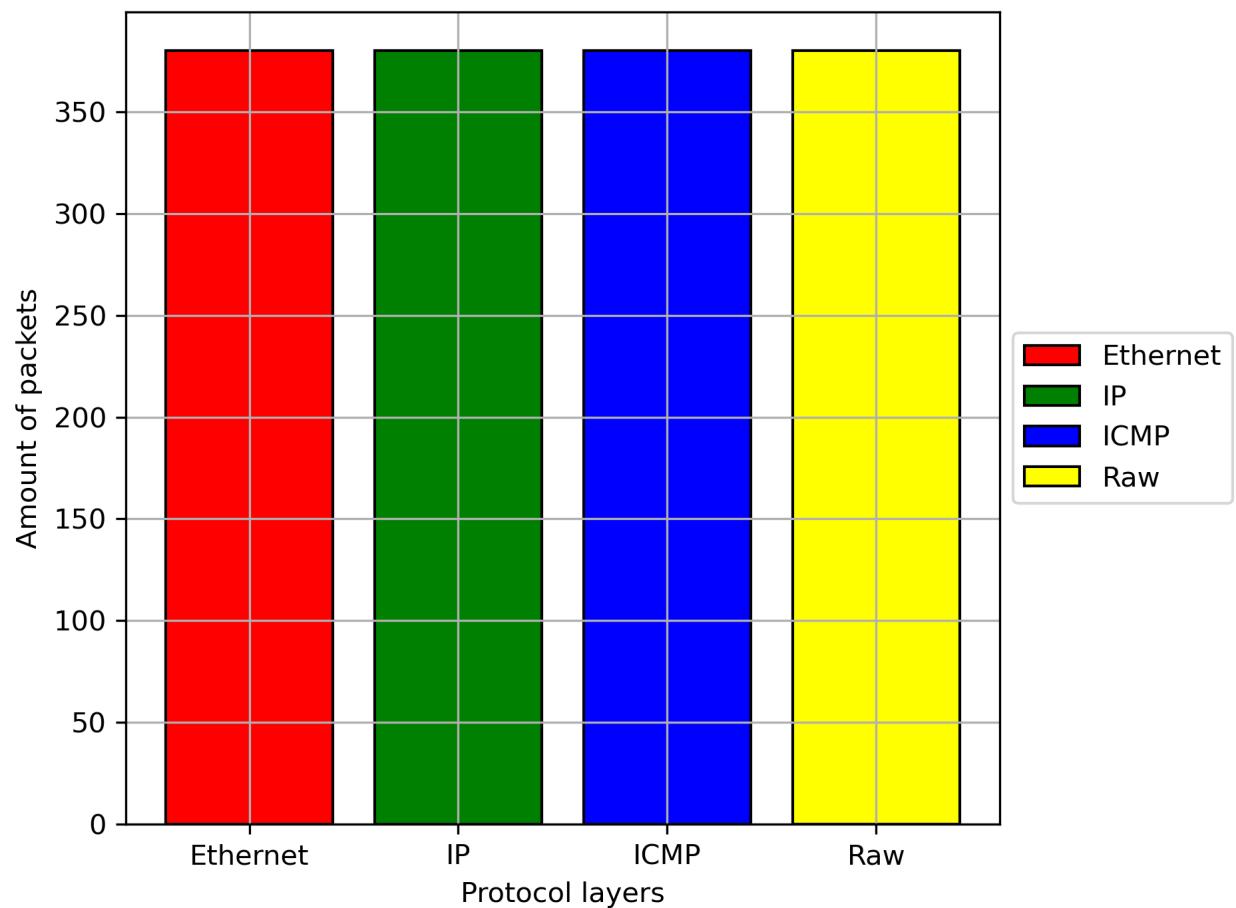


Figura 7: Gráfico de quantidade de pacotes por camada de protocolo, para a captura da interface de rede do host h1

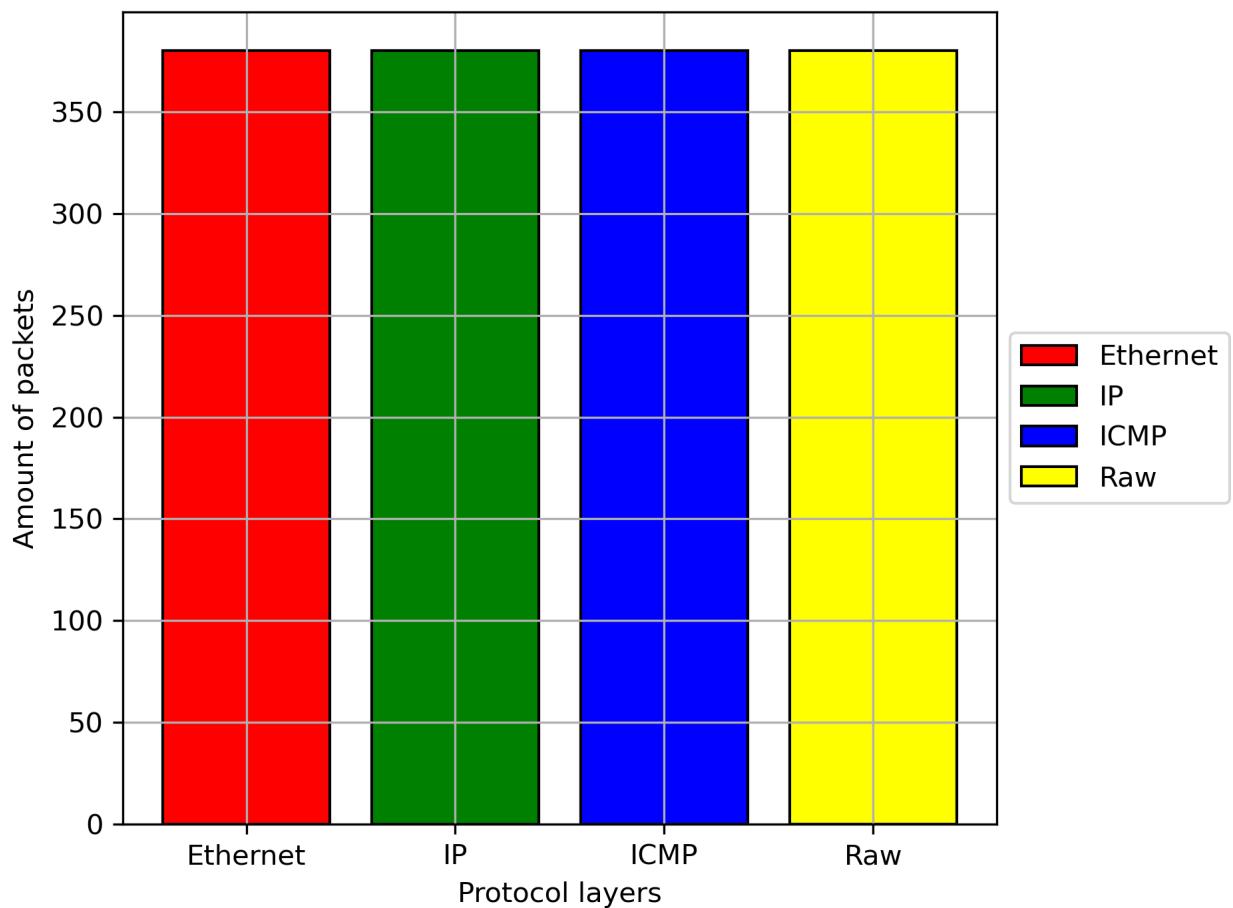


Figura 8: Gráfico de quantidade de pacotes por camada de protocolo, para a captura da interface de rede do host h2

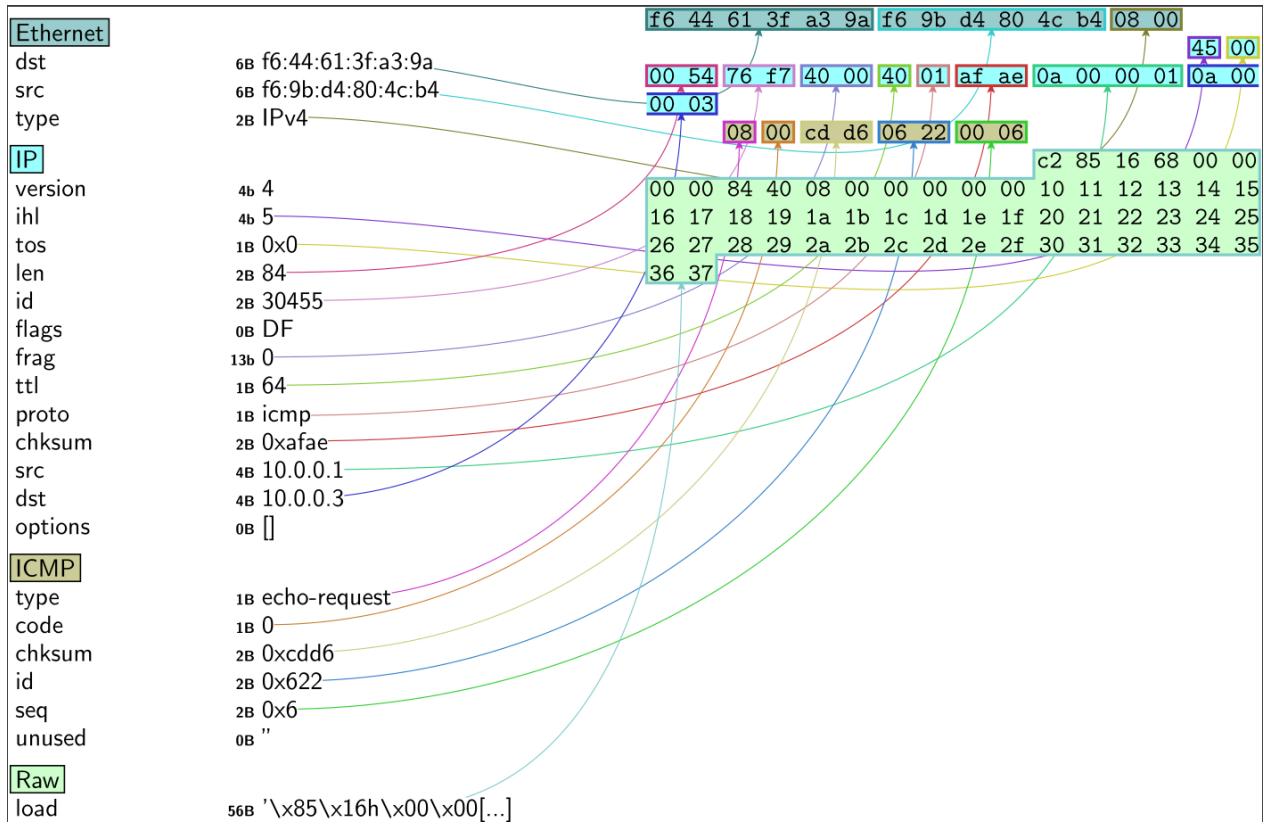


Figura 9: Diagrama de camadas de protocolo do pacote enviado pelo host h1

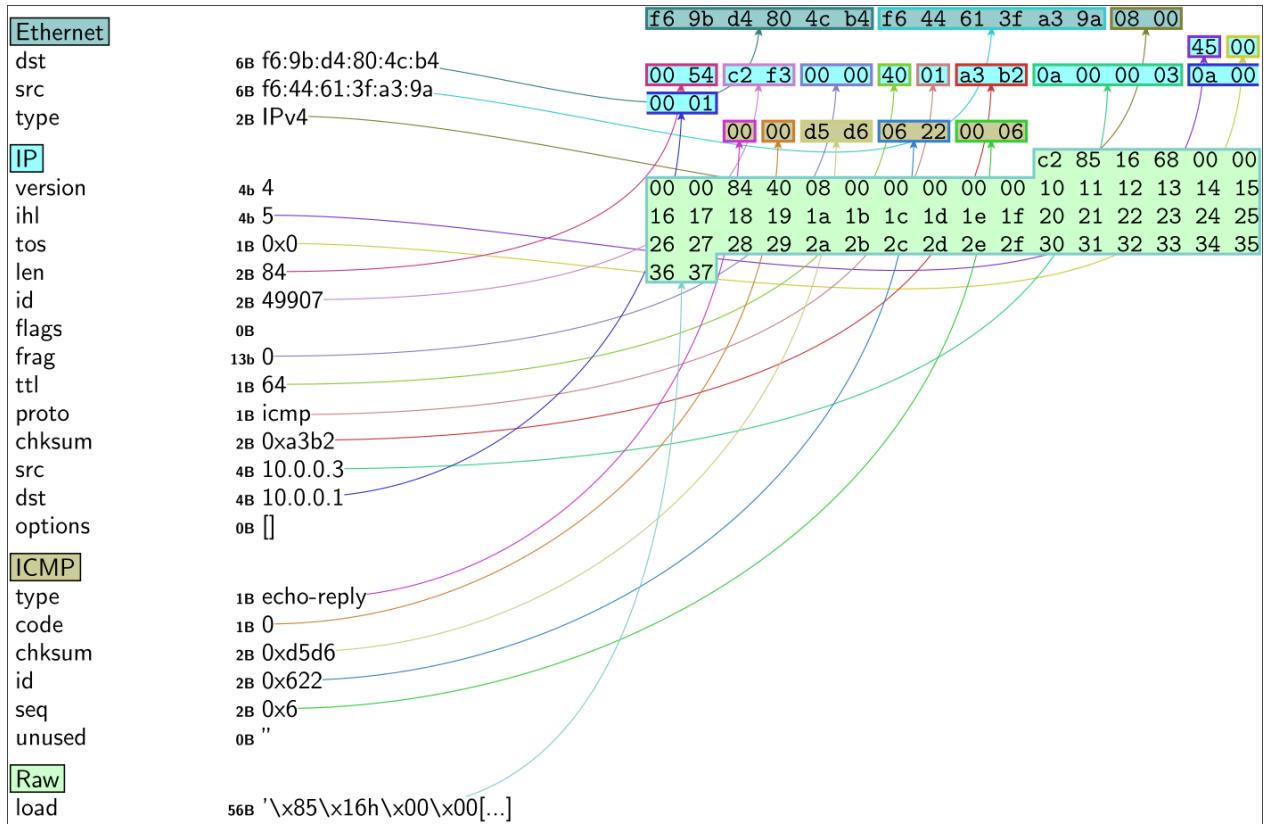


Figura 10: Diagrama de camadas de protocolo do pacote enviado pelo host h3

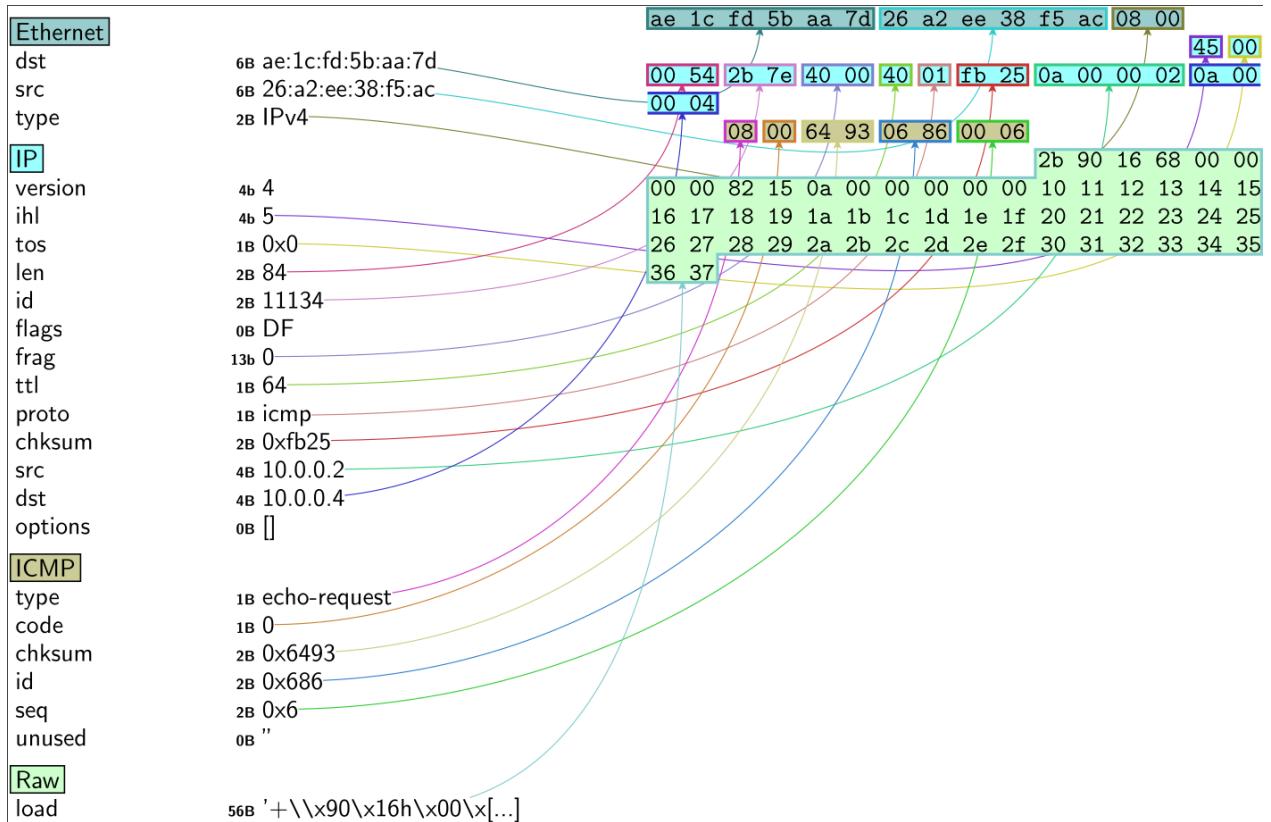


Figura 11: Diagrama de camadas de protocolo do pacote enviado pelo host h2

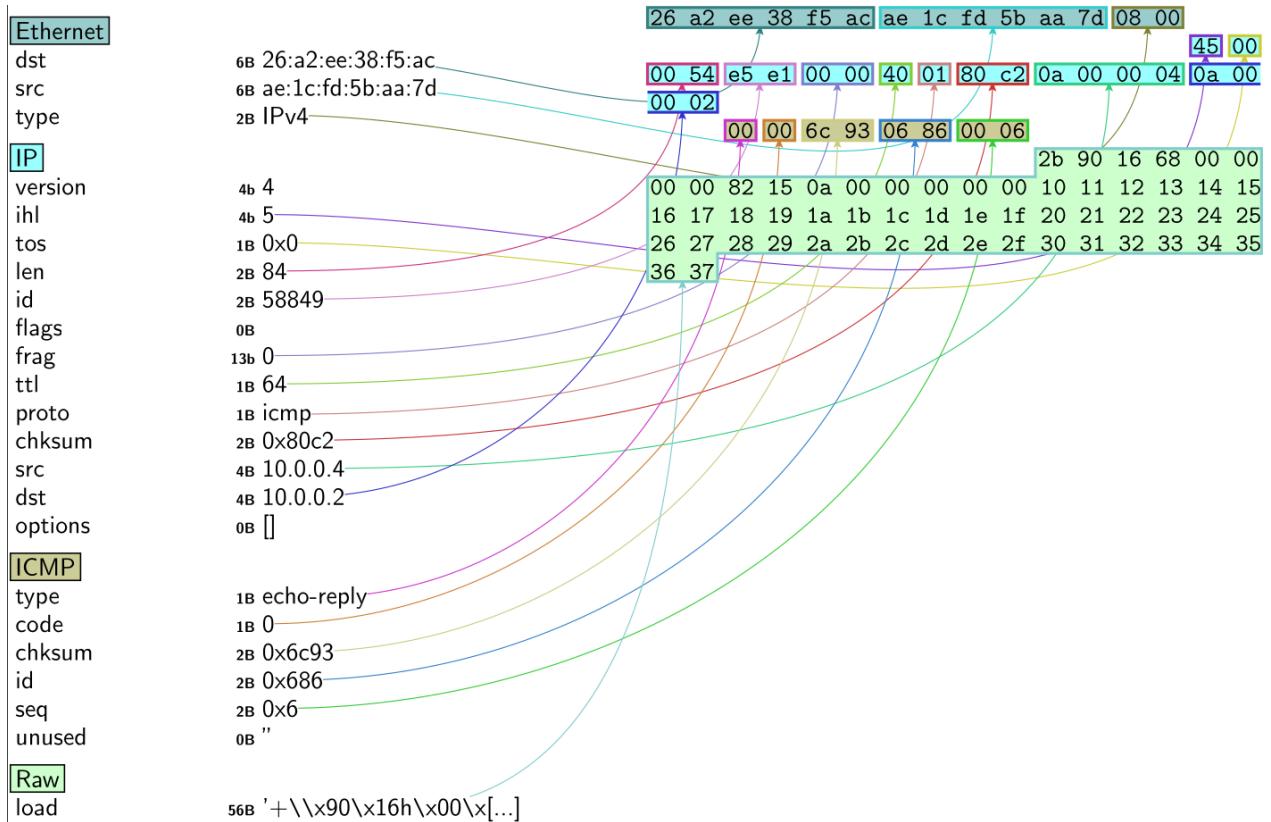


Figura 12: Diagrama de camadas de protocolo do pacote enviado pelo host h4

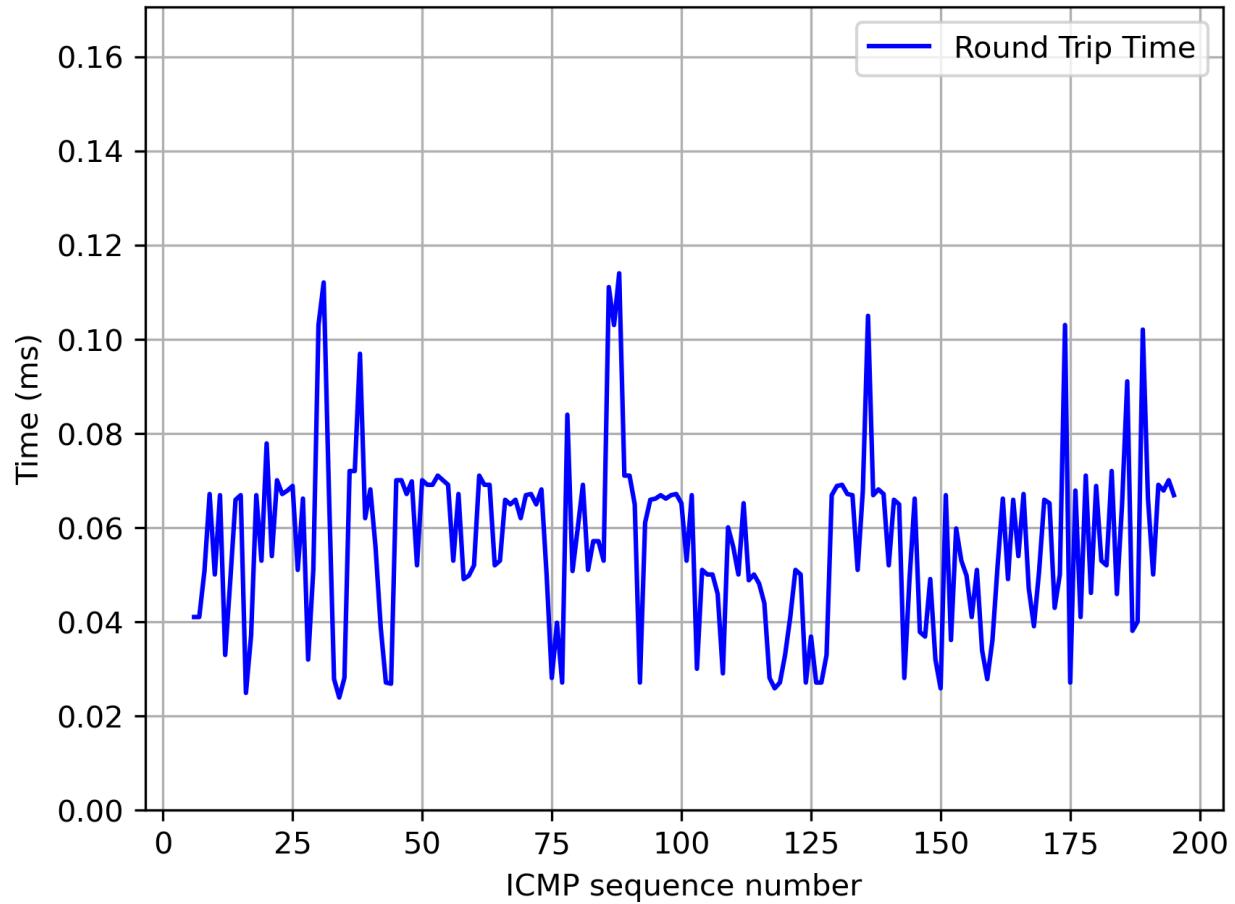


Figura 13: Gráfico de RTT por número de sequência de pacote ICMP, para a captura com interface de rede do host h1

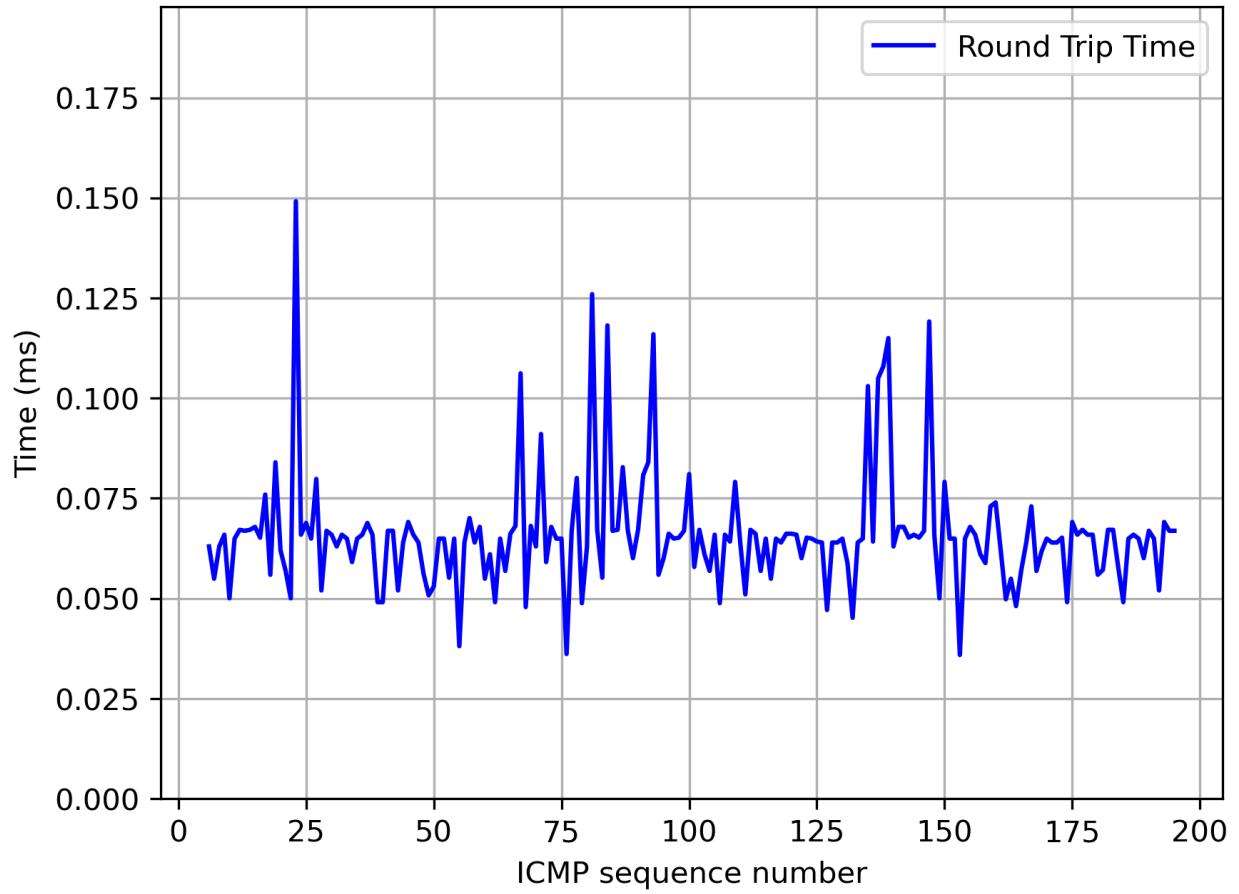


Figura 14: Gráfico de RTT por número de sequência de pacote ICMP, para a captura com interface de rede do host h2

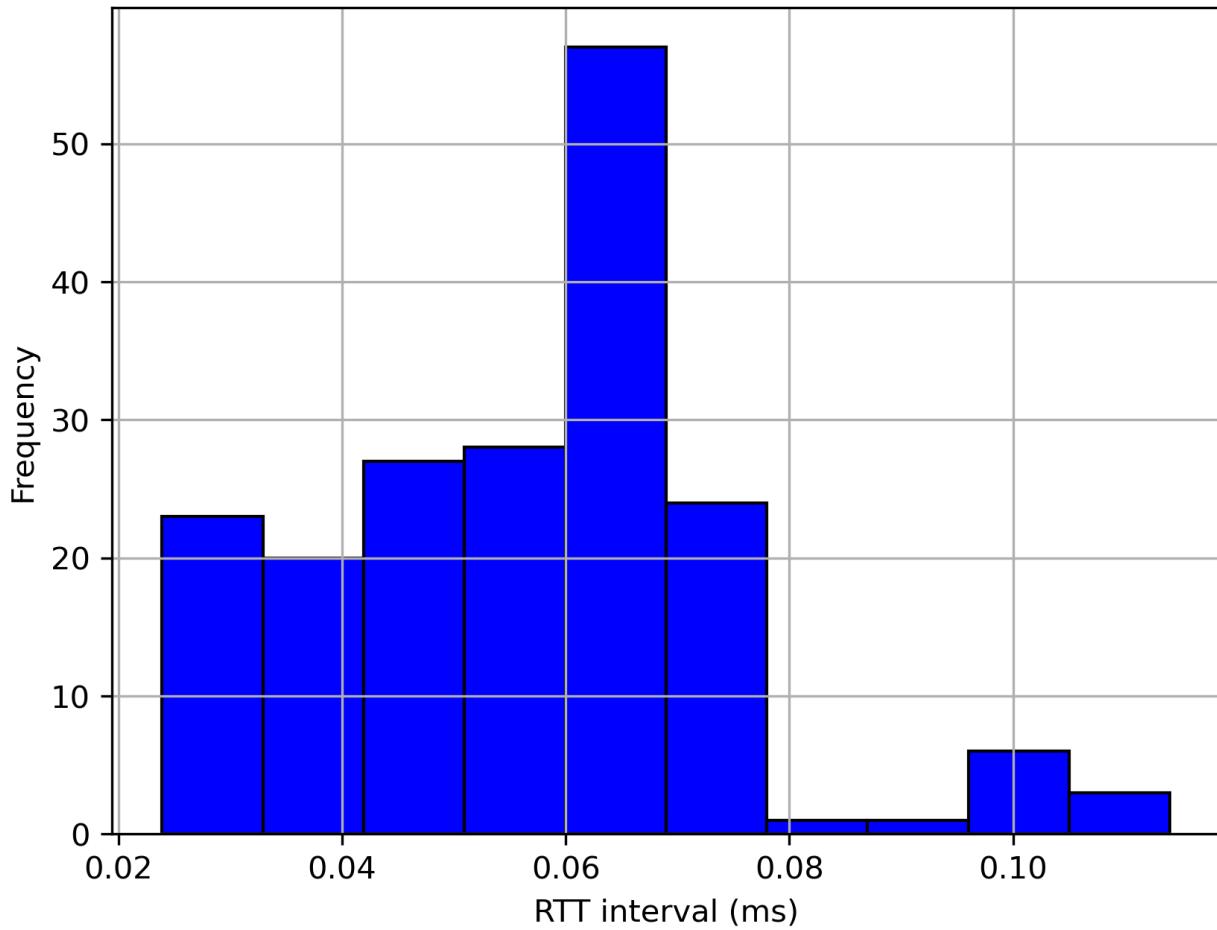


Figura 15: Histograma de RTT em troca ICMP, para a captura com interface de rede do host h1

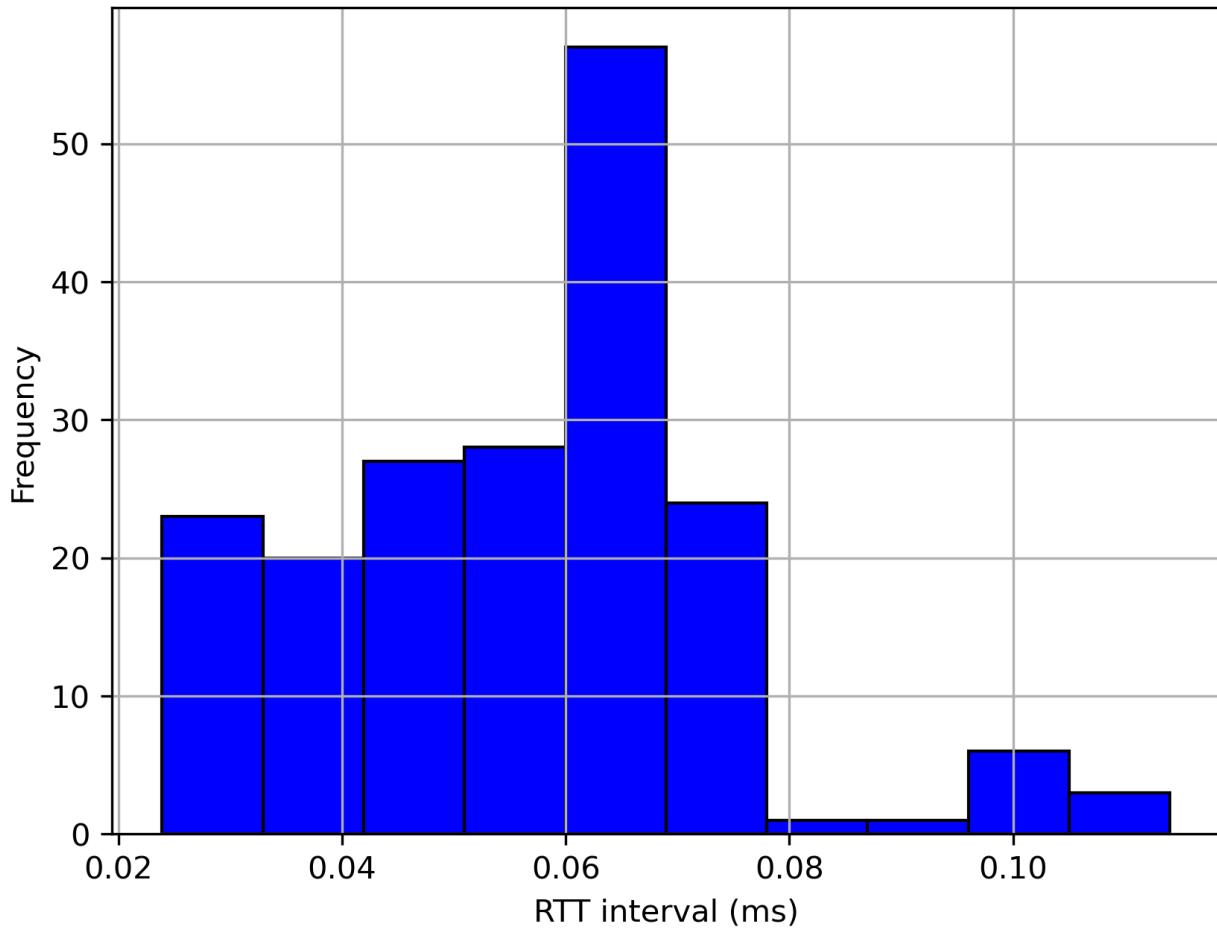


Figura 16: Histograma de RTT em troca ICMP, para a captura com interface de rede do host h2

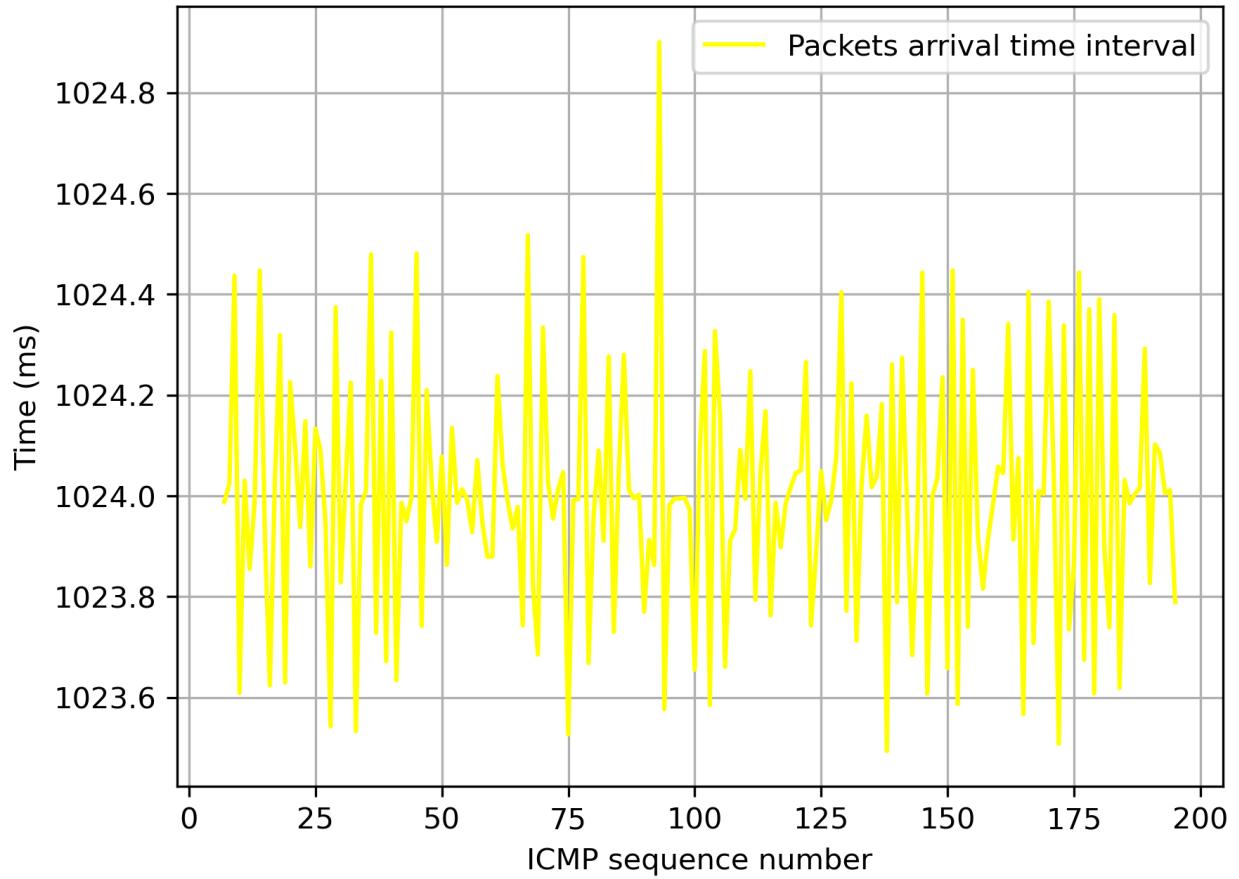


Figura 17: Gráfico de tempo de intervalos de chegada por número de sequência de pacote ICMP, para a captura com interface de rede do host h1

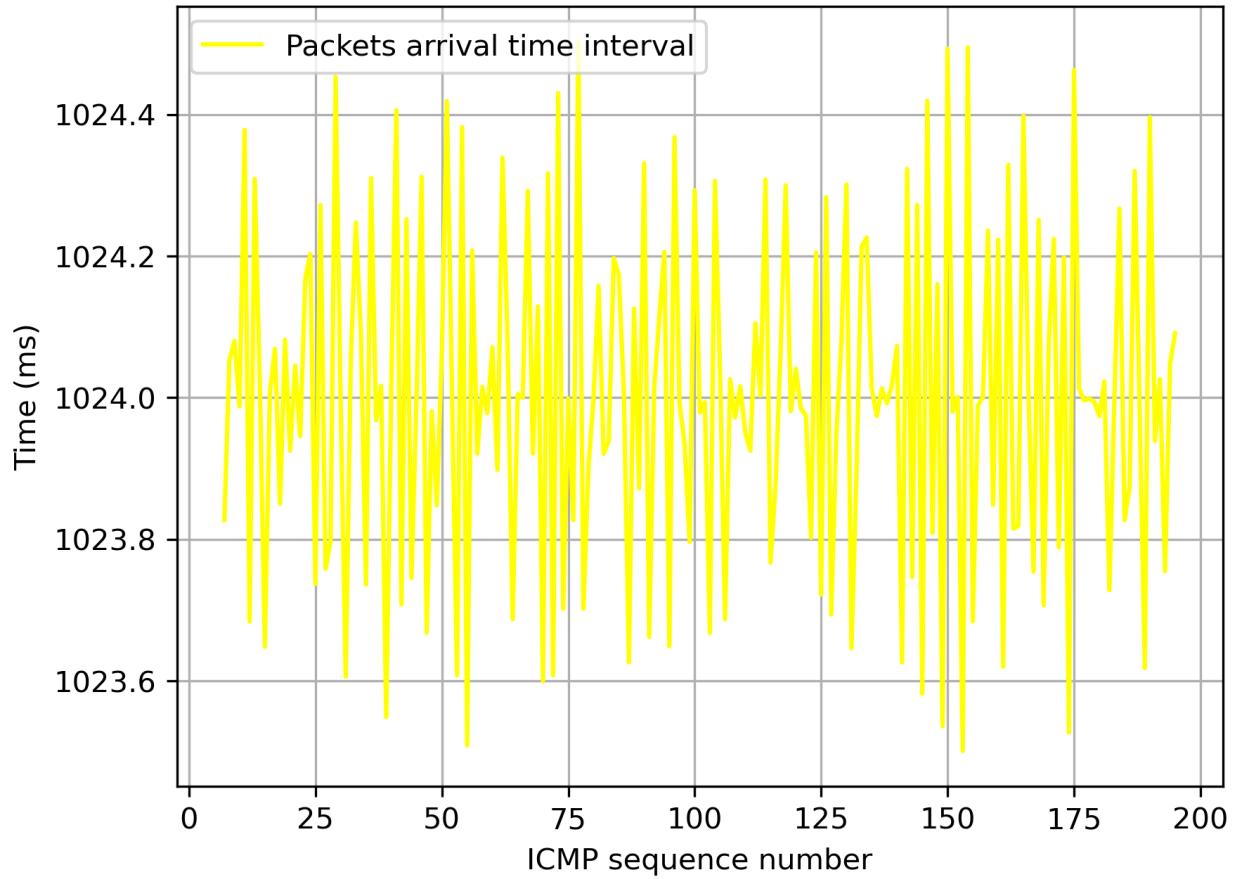


Figura 18: Gráfico de tempo de intervalos de chegada por número de sequência de pacote ICMP, para a captura com interface de rede do host h2

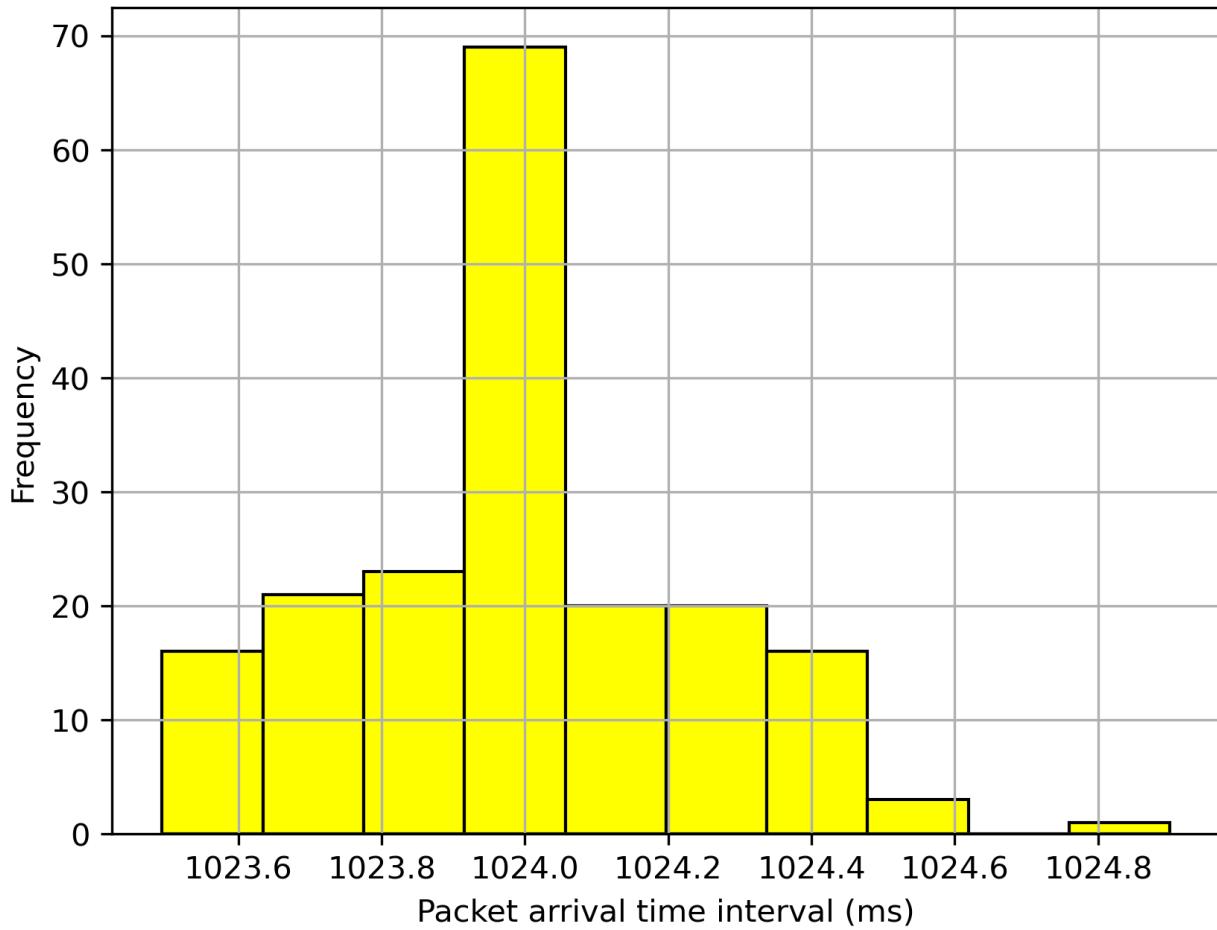


Figura 19: Histograma de intervalos entre chegada de pacotes ICMP request, para a captura com interface de rede do host h1

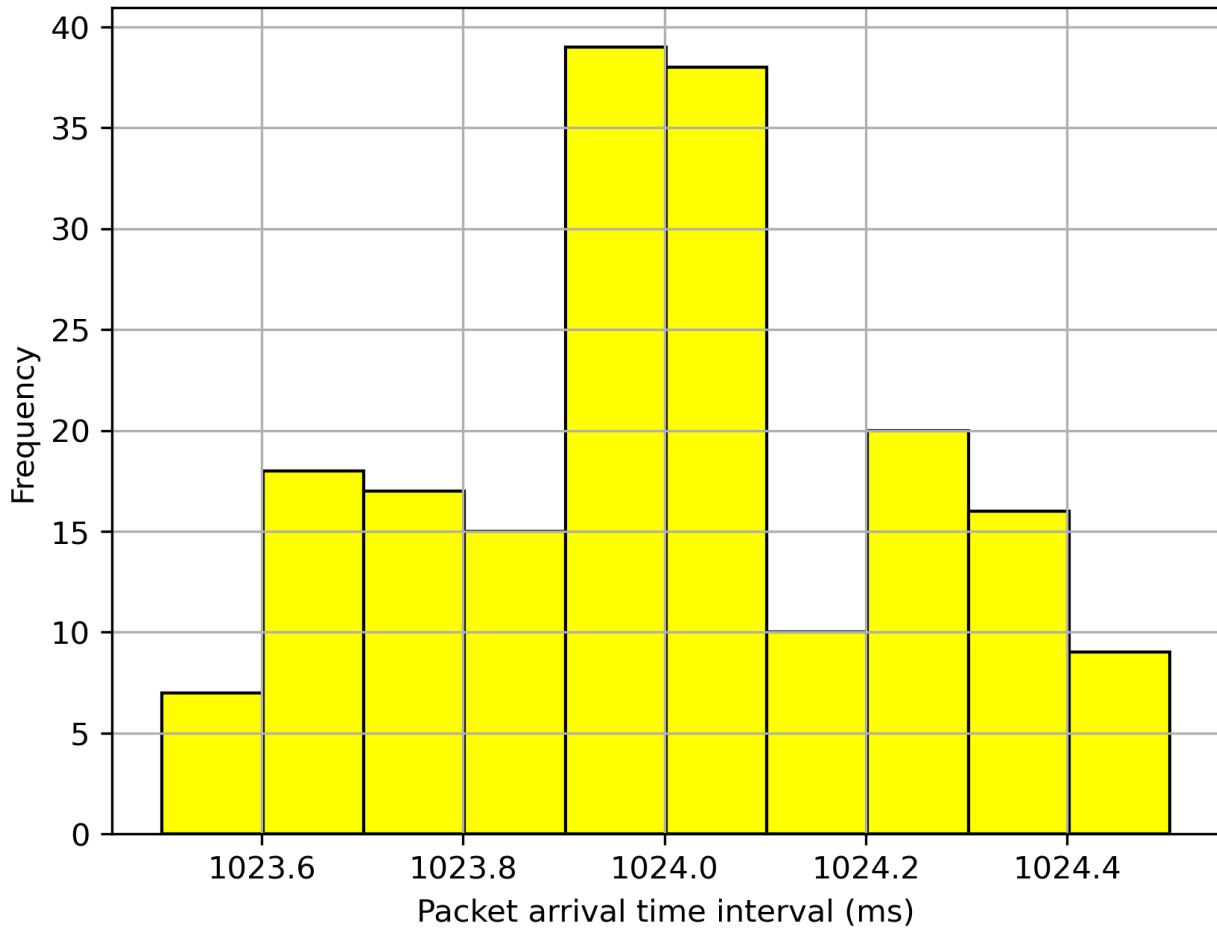


Figura 20: Histograma de intervalos entre chegada de pacotes ICMP request, para a captura com interface de rede do host h2

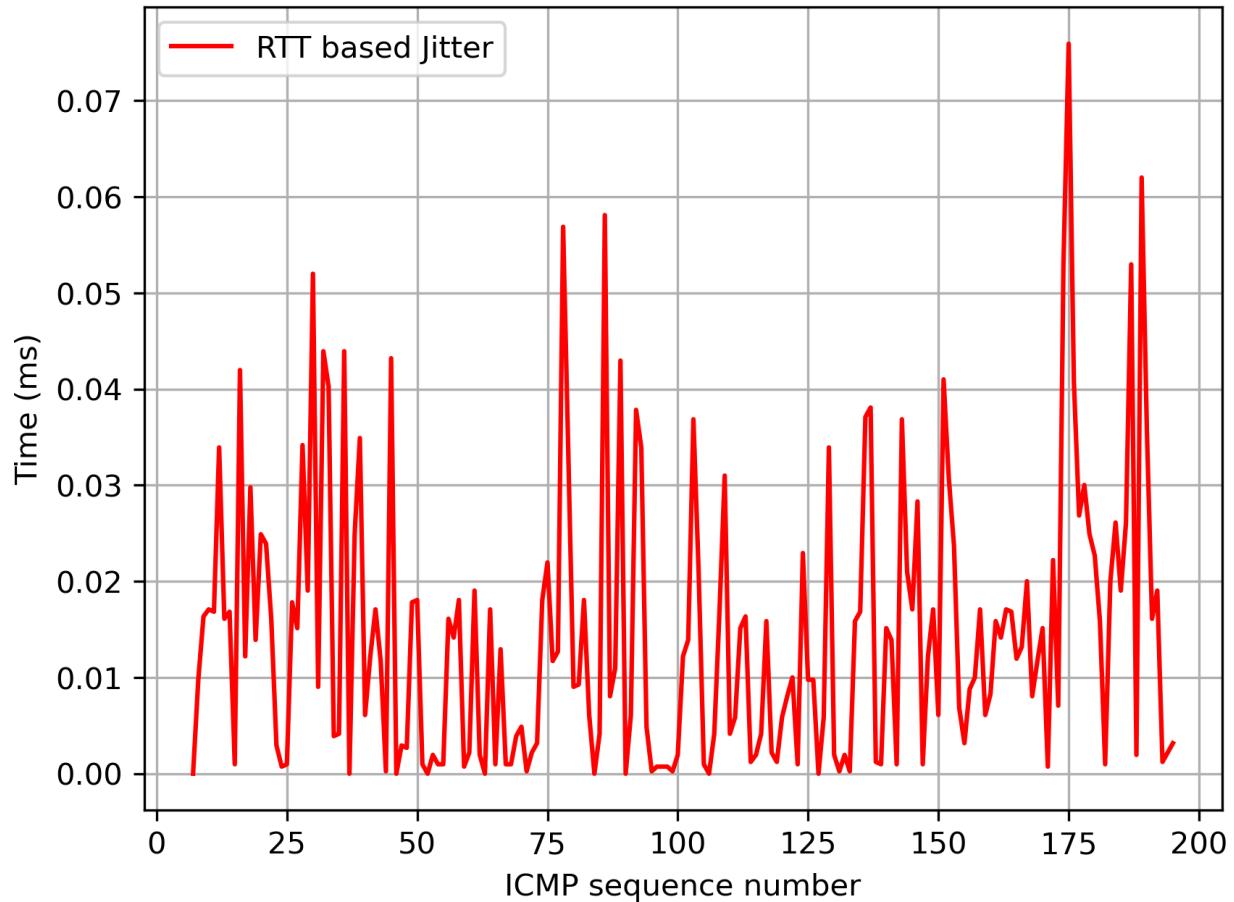


Figura 21: Gráfico de jitter baseado em RTT, para a captura com interface de rede do host h1

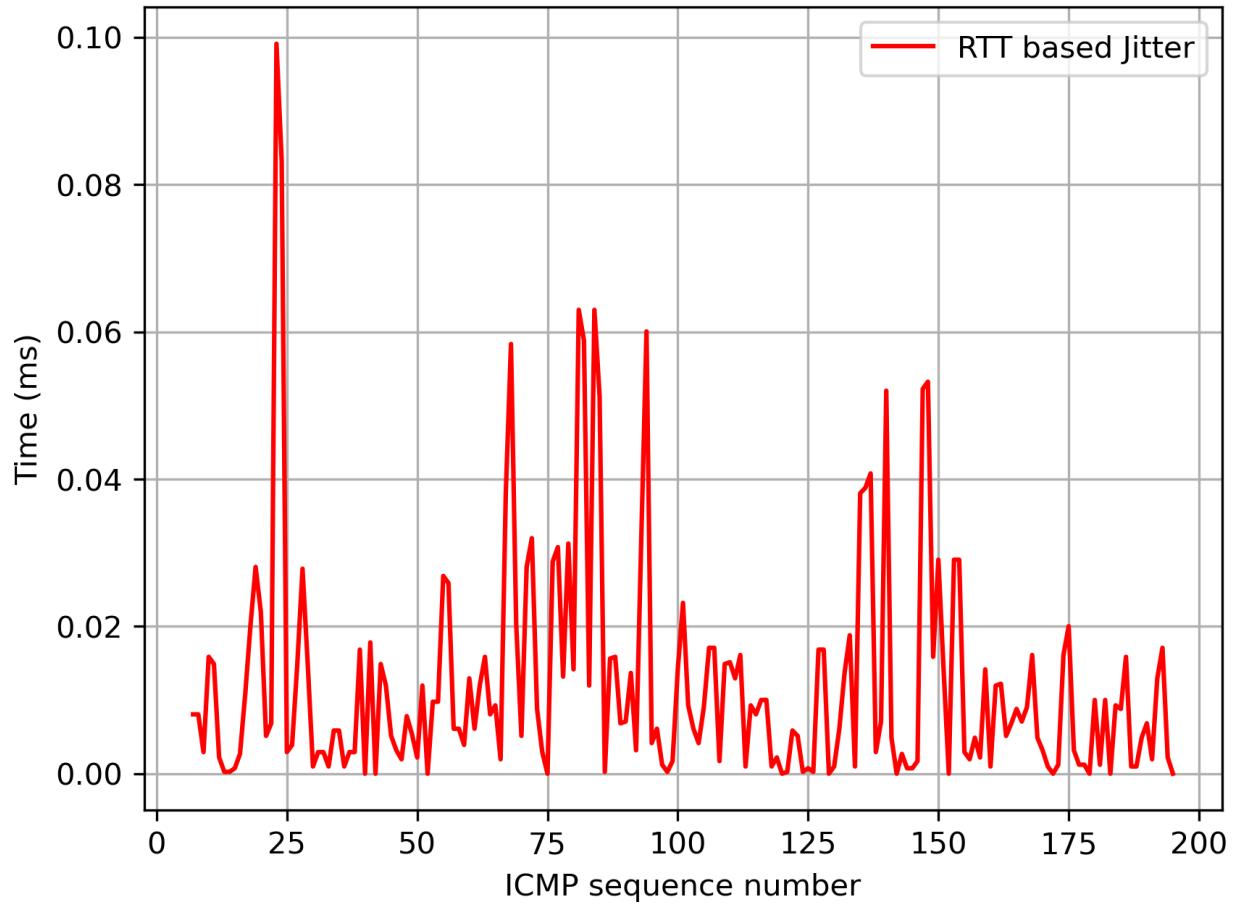


Figura 22: Gráfico de jitter baseado em RTT, para a captura com interface de rede do host h2

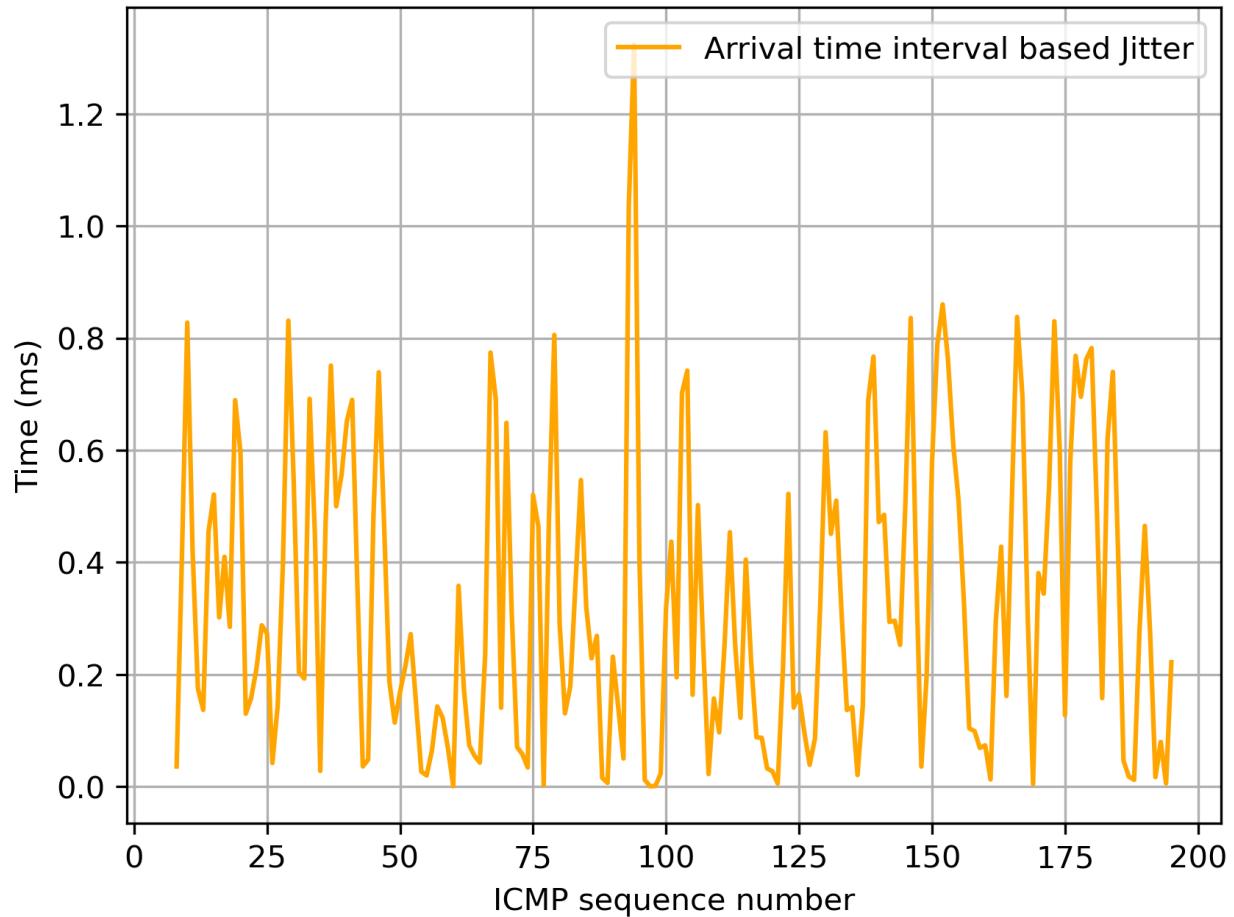


Figura 23: Gráfico de jitter baseado em intervalo de chegada entre pacotes, para a captura com interface de rede do host h1

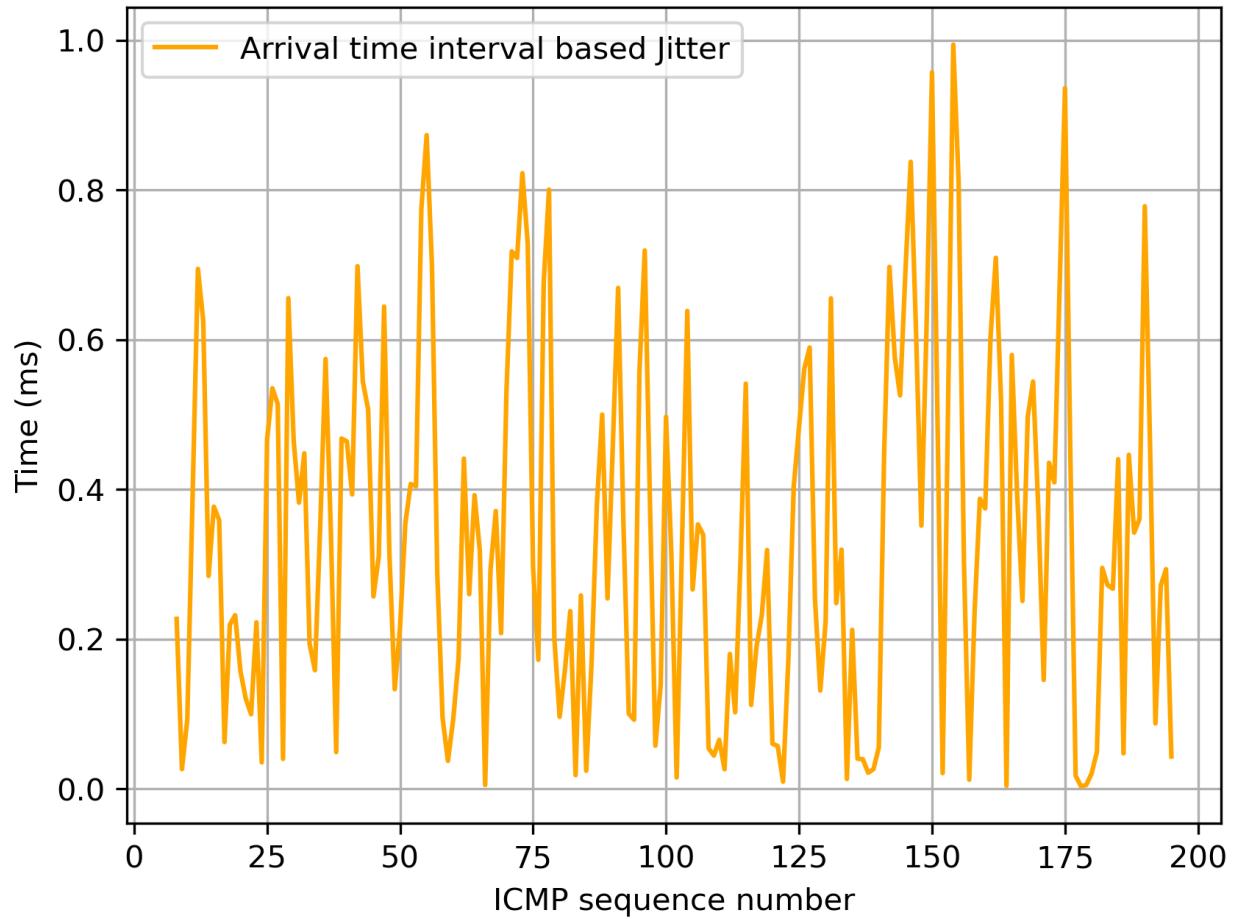


Figura 24: Gráfico de jitter baseado em intervalo de chegada entre pacotes, para a captura com interface de rede do host h2

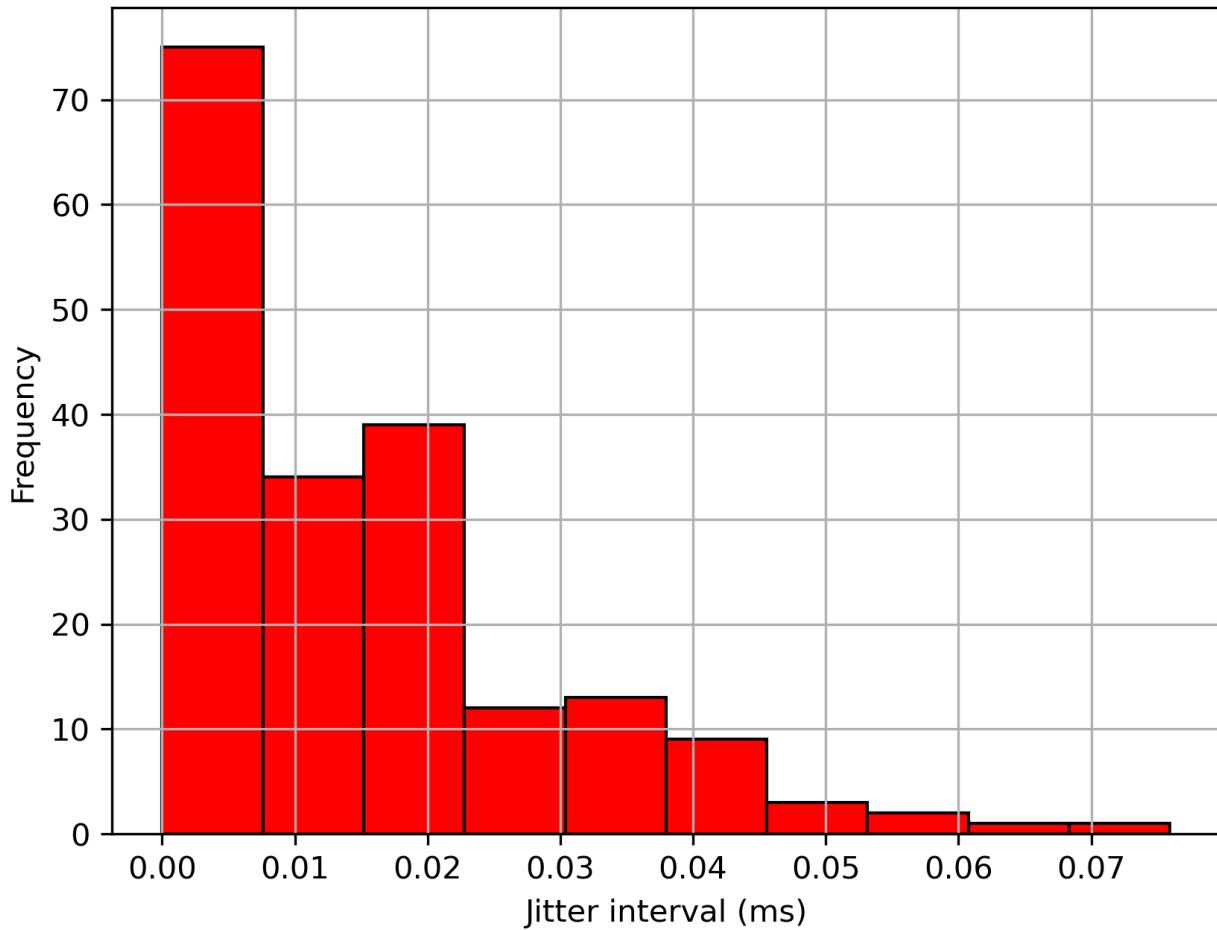


Figura 25: Histograma de jitter baseado em RTT, para a captura com interface de rede do host h1

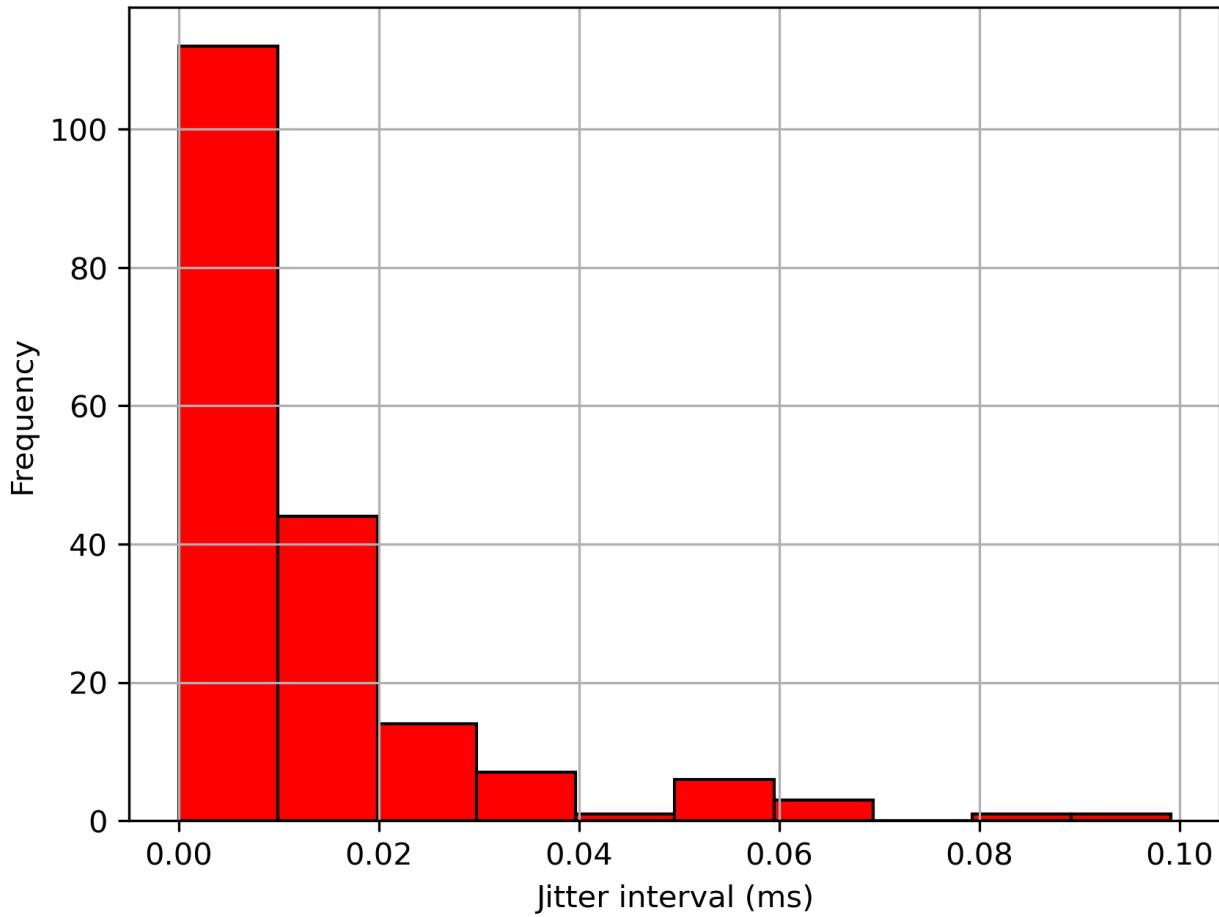


Figura 26: Histograma de jitter baseado em RTT, para a captura com interface de rede do host h2

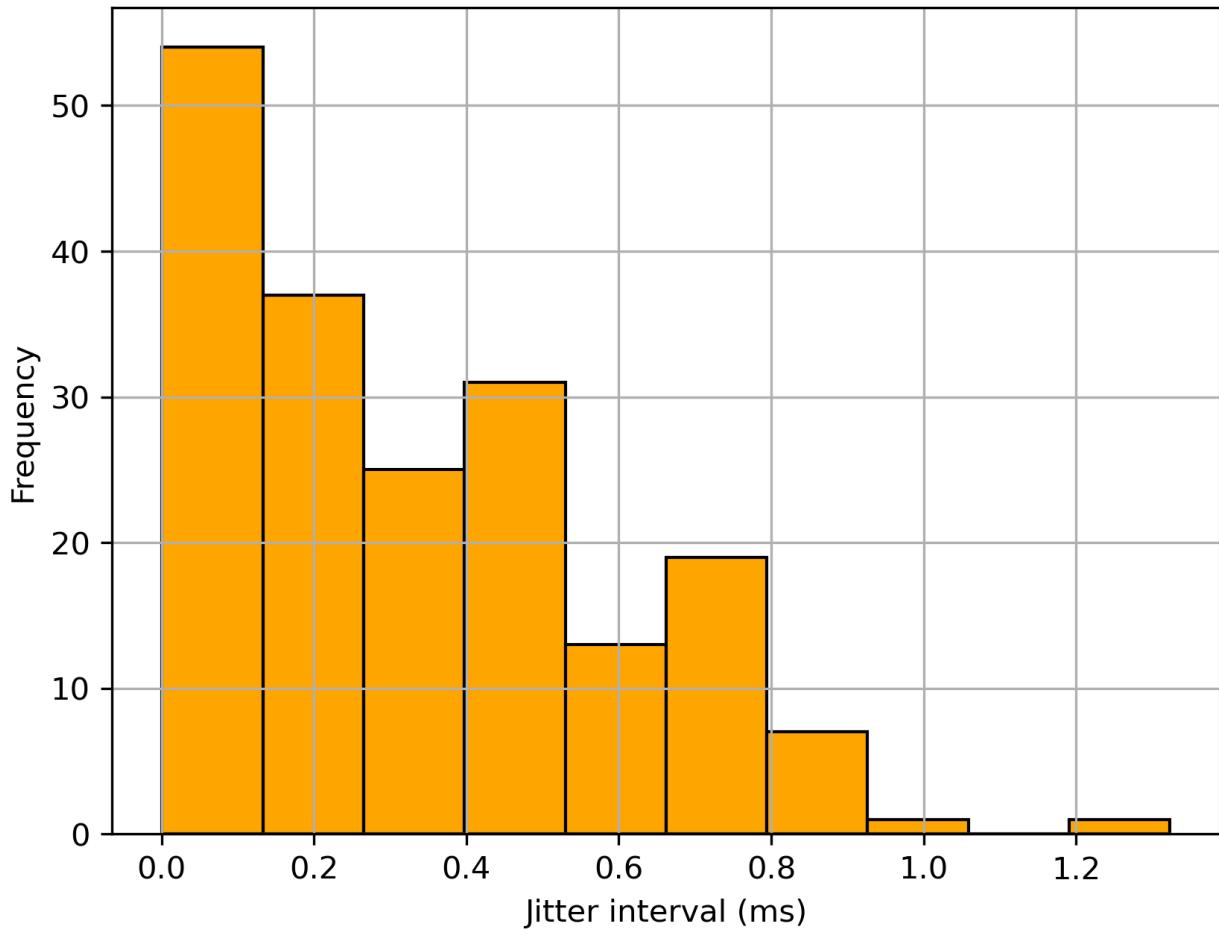


Figura 27: Histograma de jitter baseado em intervalo de chegada entre pacotes, para a captura com interface de rede do host h1

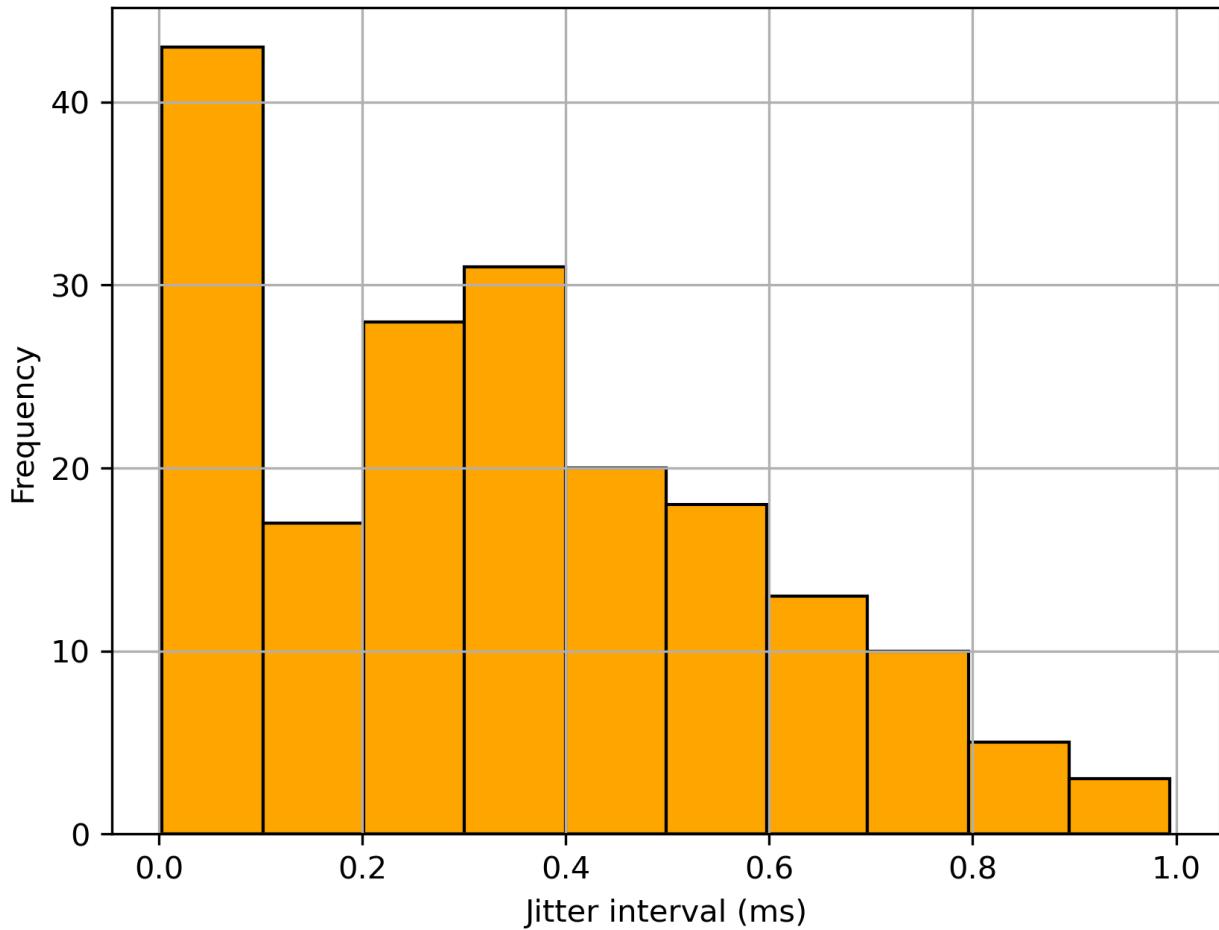


Figura 28: Histograma de jitter baseado em intervalo de chegada entre pacotes, para a captura com interface de rede do host h2

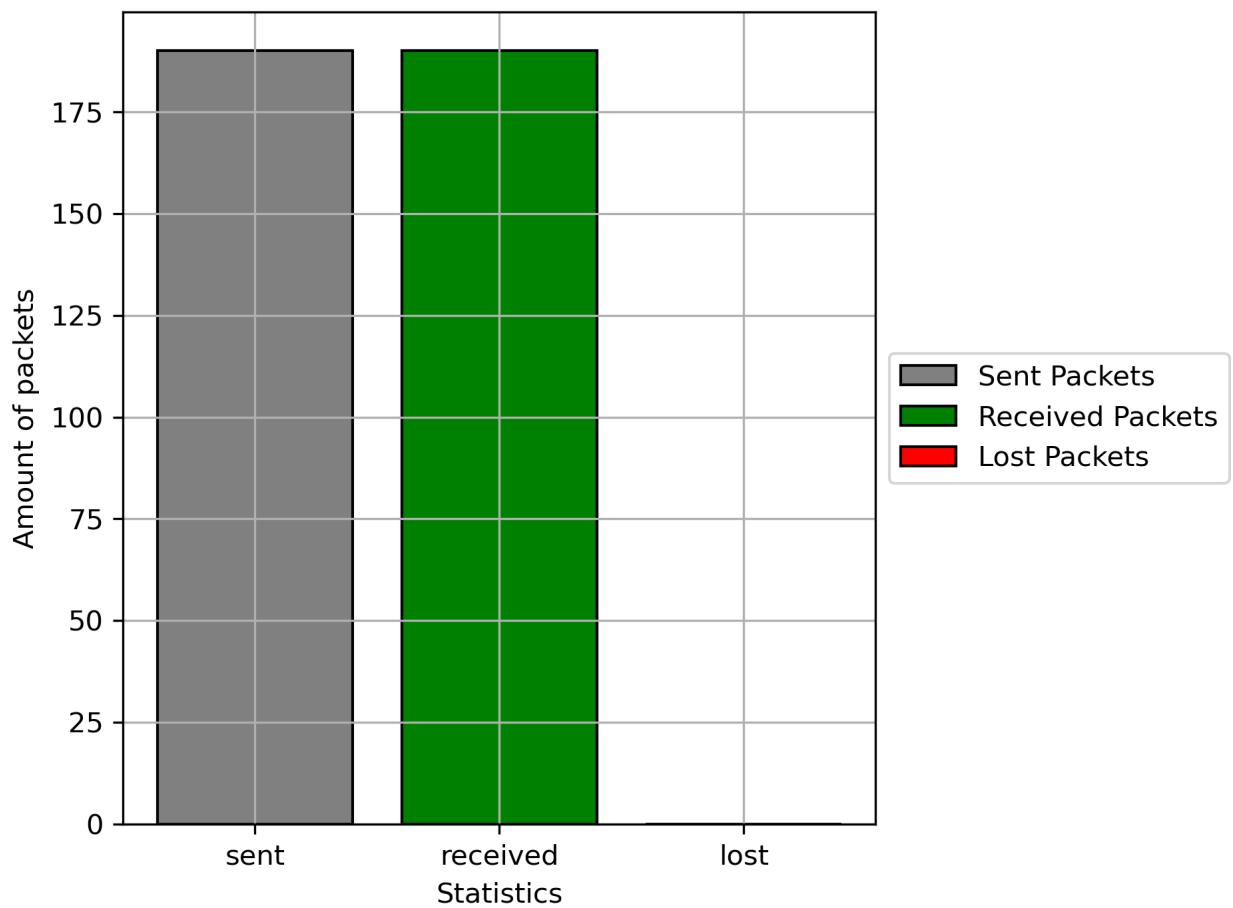


Figura 29: Gráfico de estatísticas de perda de pacotes, para a captura com interface de rede do host h1

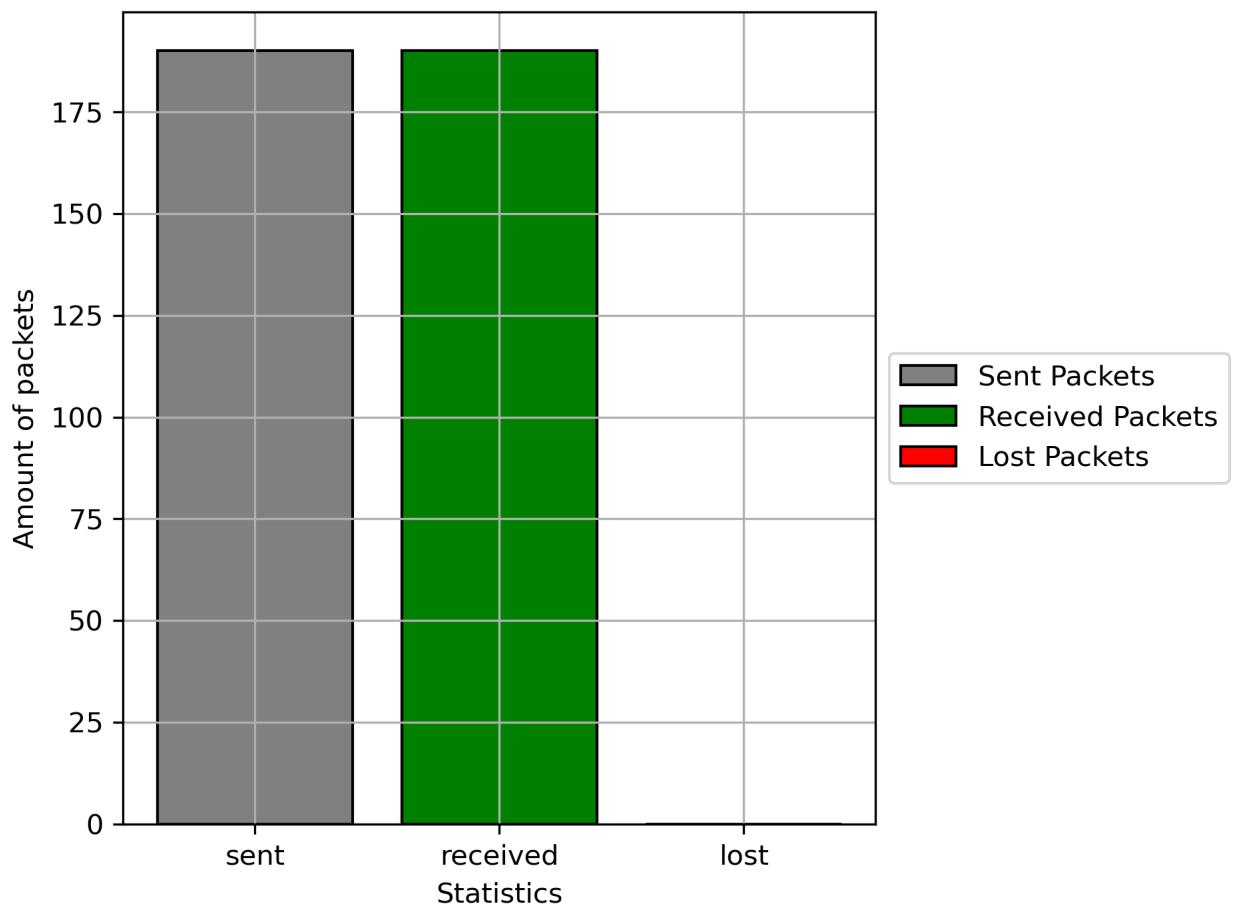


Figura 30: Gráfico de estatísticas de perda de pacotes, para a captura com interface de rede do host h2

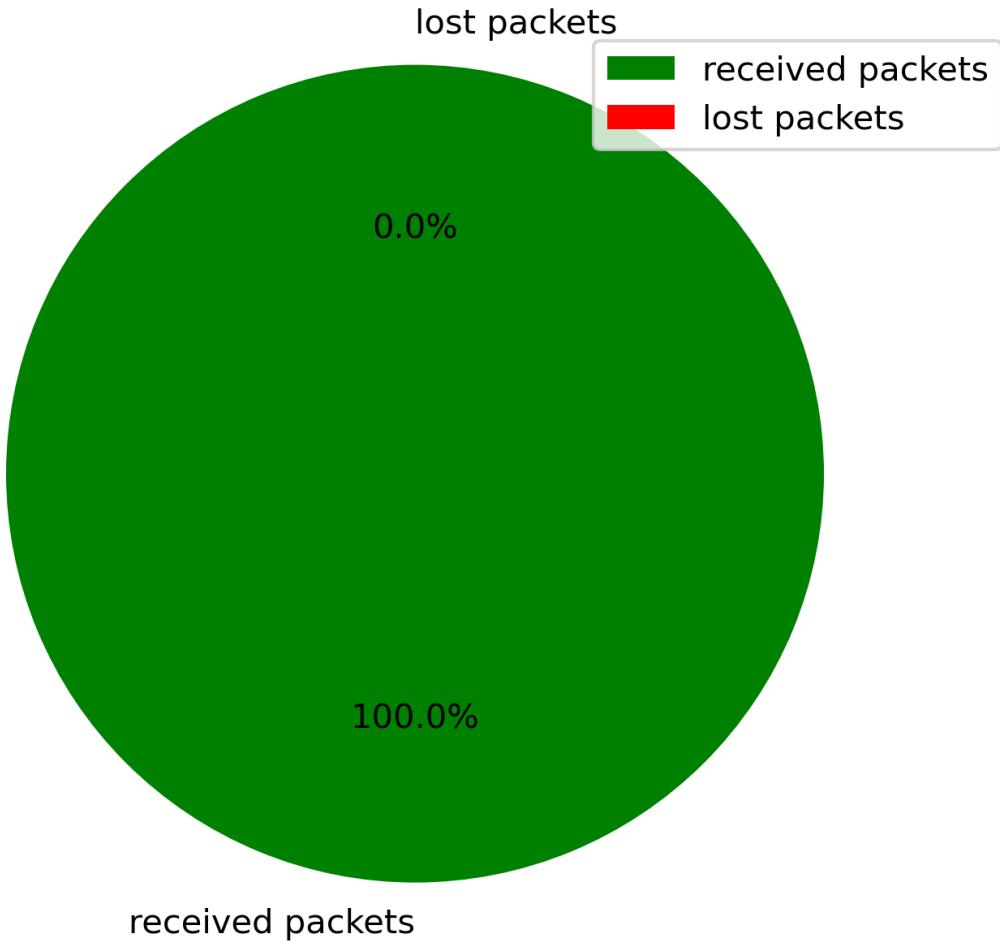


Figura 31: Gráfico de taxa de perda de pacotes, para a captura com interface de rede do host h1

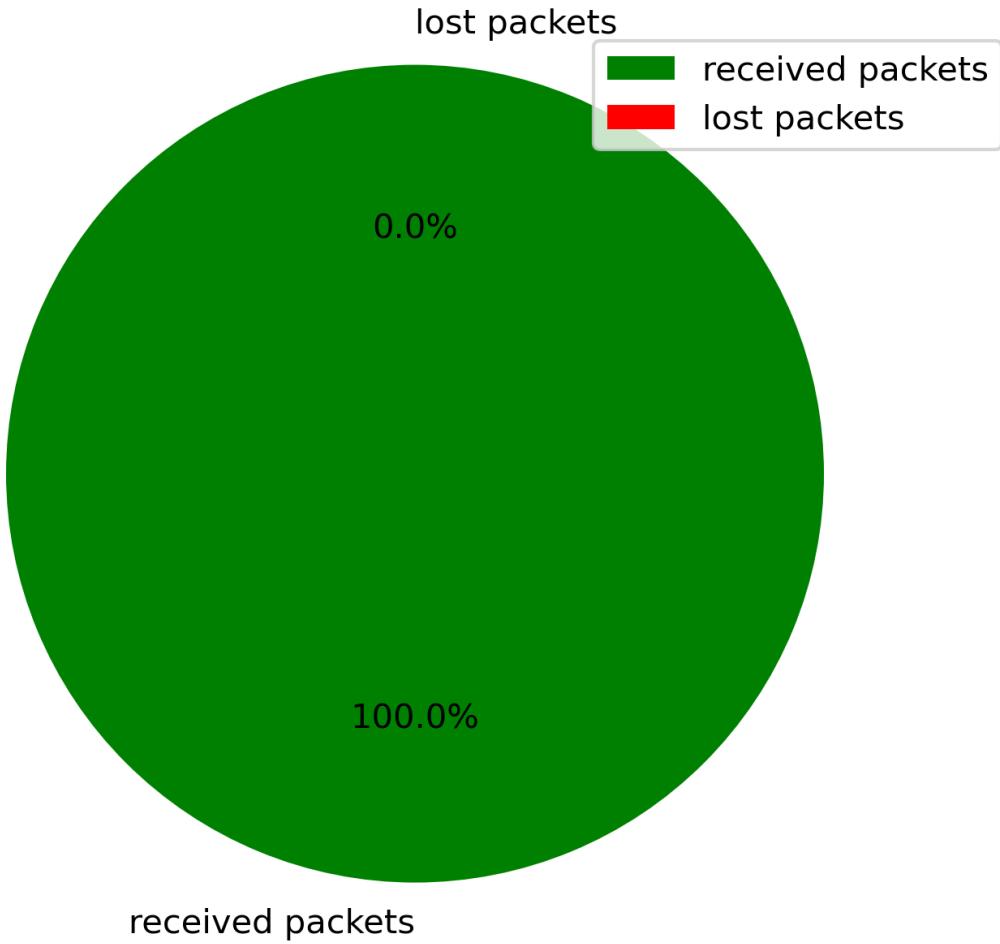


Figura 32: Gráfico de taxa de perda de pacotes, para a captura com interface de rede do host h2

## B Implementação de classes

```
1 # analisador de pacotes em capturas .pcap
2 class PacketAnalyzer(ABC):
3
4     @abstractmethod
5     def __init__(self, id=None, packetsMargin=None, path=None):
6         self.id = id
7         self.packetsMargin = packetsMargin
8
9     try:
10         self.packets = rdpcap(path)
11     except Exception as e:
12         print(f"Capture path is wrong or not specified: {e}")
13         sys.exit(1)
14
15     # retorna pacotes, pode excluir os n primeiros e n últimos para evitar
16     # viés de borda
17     def getPackets(self):
18
19         if self.packetsMargin != None:
20             return self.packets[self.packetsMargin:-self.packetsMargin]
21         else:
22             return self.packets
23
24     # retorna pacote específico
25     def getPacket(self, pkt):
26         return self.getPackets()[pkt] if len(self.getPackets()) > 0 else 0
27
28     # retorna tempo de captura de pacote em ms
29     def getTime(self, pkt):
30         return float(pkt.time*1000)
31
32     # retorna id
33     def getId(self):
34         return self.id
35
36     # retorna número total de pacotes
37     def getTotalPackets(self):
38         return len(self.getPackets())
39
40     # retorna total de bytes capturados
41     def getTotalBytes(self):
42         return sum(len(pkt) for pkt in self.getPackets()) if
43             self.getTotalPackets() > 0 else 0
44
45     # retorna tempo total de captura em ms
46     def getTotalTime(self):
47         return self.getTimeDiff(self.getPackets()[0], self.getPackets()[-1])
48             if self.getTotalPackets() > 0 else 0
49
50     # retorna pacotes capturados por segundo
51     def getCaptureRate(self):
```

```

49     return self.getTotalPackets()/self.getTotalTime() if self.getTotalTime
50     ↪ > 0 else 0
51
52 # recebe dois pacotes e retorna a diferença de tempo de captura entre eles
53 def getTimeDiff(self, pkt1, pkt2):
54     return self.getTime(pkt2) - self.getTime(pkt1)
55
56 # retorna throughput medido em Mbps
57 def getThroughput(self):
58
59     totalBits = self.getTotalBytes() * 8
60
61     return (totalBits/self.getTotalTime())/1000 if self.getTotalTime() > 0
62     ↪ else 0
63
64 # retorna lista de camadas e quantidade total encontrada por camada
65 def getLayers(self):
66
67     layers = Counter()
68
69     for pkt in self.getPackets():
70         while pkt:
71             layers[pkt.name] += 1 # incrementa número de camadas
72             pkt = pkt.payload # próxima camada, payload da camada atual
73
74     nLayers = list(layers.values())
75     layers = list(layers.keys())
76
77     return {"layers": layers,
78             "nLayers": nLayers
79             }
80
81 # retorna estatísticas de jitter baseado na variação de dados: lista de
82 # jitters, média, desvio padrão, máximo, mínimo, erro padrão e
83 # coeficiente de variação
84 def getJitterStats(self, data):
85
86     if self.getTotalPackets() < 3:
87         print("There is no way to measure jitter with less than three
88             ↪ packets")
89         return None
90
91     jitters = np.abs(np.diff(data)) if len(data) > 0 else []
92     mean = np.mean(jitters) if len(jitters) > 0 else 0
93     std = np.std(jitters) if len(jitters) > 0 else 0
94     max = np.max(jitters) if len(jitters) > 0 else 0
95     min = np.min(jitters) if len(jitters) > 0 else 0
96     error = std/np.sqrt(len(jitters)) if len(jitters) > 0 else 0
97     cv = (std/mean)*100 if mean > 0 else 0
98
99     return {"jitters": jitters,
100            "mean": mean,
101            "std": std,
102            "max": max,
103            "min": min,
104            "error": error}

```

```

98         "min": min,
99         "error": error,
100        "cv": cv
101    }
102
103    # salva visualização gráfica de pacote em pdf
104    def getPdfDump(self, filename, pkt):
105        self.getPacket(pkt).pdfdump(filename, layer_shift=1)
106
107    def getRttStats(self):
108        pass
109
110    def getIntervalStats(self):
111        pass
112
113    def getLossStats(self):
114        pass
115
116    # retorna quantidade correta de casas decimais para representação (value ±
117    # → error)
118    @staticmethod
119    def getDecimalPlaces(error):
120
121        if error == 0:
122            return 0
123
124        places = np.floor(np.log10(np.abs(error)))
125
126        return max(0, int(-places))
127
128    # imprime métricas gerais
129    def printGeneralMetrics(self, id, src, dst, totalPackets, totalBytes,
130    # → layers, throughput):
131
132        print(f"Capture {id}")
133        print(f"Source IPv4: {src}")
134        print(f"Destination IPv4: {dst}")
135        print(f"Total packets: {totalPackets}")
136        print(f"Total bytes: {totalBytes} bytes")
137        print(f"Layers: {layers}")
138        print(f"Throughput: {throughput:.4f} Mbps\n")
139
140    # imprime métricas de RTT
141    def printRttMetrics(self, layer, mean, std, max, min, error, cv):
142
143        places = self.getDecimalPlaces(error)
144        print(f"Mean {layer} RTT: {mean:.{places}f} ms")
145        print(f"{layer} RTT standard deviation: {std:.{places}f} ms")
146        print(f"Maximum {layer} RTT: {max:.{places}f} ms")
147        print(f"Minimum {layer} RTT: {min:.{places}f} ms")
148        print(f"Standard error: {error:.{places}f} ms")
149        print(f"Percentage of standard deviation from the mean: {cv:.2f}%\n")
150
151    # imprime métricas de intervalo de chegada entre pacotes

```

```

150 def printIntervalMetrics(self, layer, mean, std, max, min, error, cv):
151
152     places = self.getDecimalPlaces(error)
153     print(f"Mean {layer} packets arrival time interval: {mean:.{places}f}
154         ↵ ms")
155     print(f"Standard deviation of {layer} packets arrival time interval:
156         ↵ {std:.{places}f} ms")
157     print(f"Maximum {layer} packet arrival time interval: {max:.{places}f}
158         ↵ ms")
159     print(f"Minimum {layer} request packet arrival time Interval:
160         ↵ {min:.{places}f} ms")
161     print(f"Standard error: {error:.{places}f} ms")
162     print(f"Percentage of standard deviation from the mean: {cv:.2f}%\n")
163
164 # imprime métricas de jitter baseado em rtt
165 def printRttJitterMetrics(self, layer, mean, std, max, min, error, cv):
166
167     places = self.getDecimalPlaces(error)
168     print(f"{layer} RTT based jitter mean: {mean:.{places}f} ms")
169     print(f"{layer} RTT based jitter standard deviation: {std:.{places}f}
170         ↵ ms")
171     print(f"{layer} RTT based maximum jitter: {max:.{places}f} ms")
172     print(f"{layer} RTT based minimum jitter: {min:.{places}f} ms")
173     print(f"Standard error: {error:.{places}f} ms")
174     print(f"Percentage of standard deviation from the mean: {cv:.2f}%\n")
175
176 # imprime métricas de jitter baseado em intervalo de chegada
177 def printIntervalJitterMetrics(self, layer, mean, std, max, min, error,
178     ↵ cv):
179
180     places = self.getDecimalPlaces(error)
181     print(f"{layer} arrival time interval based jitter mean:
182         ↵ {mean:.{places}f} ms")
183     print(f"{layer} arrival time interval based jitter standard deviation:
184         ↵ {std:.{places}f} ms")
185     print(f"{layer} arrival time interval based maximum jitter:
186         ↵ {max:.{places}f} ms")
187     print(f"{layer} arrival time interval based minimum jitter:
188         ↵ {min:.{places}f} ms")
189     print(f"Standard error: {error:.{places}f} ms")
190     print(f"Percentage of standard deviation from the mean: {cv:.2f}%\n")
191
192 # imprime métricas de perda de pacotes
193 def printLossMetrics(self, layer, sent, received, lost, lossRate):
194
195     print(f"{layer} sent packets: {sent}")
196     print(f"{layer} received packets: {received}")
197     print(f"{layer} lost packets: {lost}")
198     print(f"{layer} loss rate: {lossRate}%\n")
199
200     ↵ print("-----")
201
202 # plota gráfico de barra para o total de camadas

```

```

192     def plotLayersGraph(self, path, id, layers, nLayers, title, xlabel,
193                         ylabel):
194
195         layersGraph = GraphPlotter(title=title, xlabel=xlabel, ylabel=yLabel,
196                                   legendPosition="right")
197         layersGraph.plotBarGraph(layers, nLayers, plotLabel=layers)
198         layersGraph.saveGraph(path+id+"-layers.png")
199
200
201     # plota gráfico de rtt
202     def plotRttGraph(self, path, id, xAxis, rtts, title, xlabel, ylabel):
203
204         rttGraph = GraphPlotter(title=title, xlabel=xLabel, ylabel=yLabel)
205         rttGraph.plotLineGraph(xAxis, rtts, color="blue", plotLabel="Round
206                               Trip Time", marker=None, autoScaleY=True)
207         rttGraph.saveGraph(path+id+"-rtt.png")
208
209     # plota gráfico de intervalos de chegada entre pacotes
210     def plotIntervalGraph(self, path, id, xAxis, intervals, title, xlabel,
211                           ylabel):
212
213         intervalGraph = GraphPlotter(title=title, xlabel=xLabel,
214                                       ylabel=yLabel)
215         intervalGraph.plotLineGraph(xAxis, intervals, color="yellow",
216                                      plotLabel="Packets arrival time interval", marker=None)
217         intervalGraph.saveGraph(path+id+"-interval.png")
218
219     # plota gráfico de jitter baseado em rtt
220     def plotRttJitterGraph(self, path, id, xAxis, jitters, title, xlabel,
221                           ylabel):
222
223         rttJitterGraph = GraphPlotter(title=title, xlabel=xLabel,
224                                       ylabel=yLabel)
225         rttJitterGraph.plotLineGraph(xAxis, jitters, color="red",
226                                      plotLabel="RTT based Jitter", marker=None)
227         rttJitterGraph.saveGraph(path+id+"-rtt-jitter.png")
228
229     # plota gráfico de jitter baseado em intervalo de chegada
230     def plotIntervalJitterGraph(self, path, id, xAxis, jitters, title, xlabel,
231                                ylabel):
232
233         intervalJitterGraph = GraphPlotter(title=title, xlabel=xLabel,
234                                           ylabel=yLabel)
235         intervalJitterGraph.plotLineGraph(xAxis, jitters, color="orange",
236                                         plotLabel="Arrival time interval based Jitter", marker=None)
237         intervalJitterGraph.saveGraph(path+id+"-interval-jitter.png")
238
239     # plota histograma de rtt
240     def plotRttHistogram(self, path, id, rtts, title, xlabel, ylabel):
241
242         rttHistogram = GraphPlotter(title=title, xlabel=xLabel, ylabel=yLabel,
243                                     legendFlag=False)
244         rttHistogram.plotHistogram(rtts, color="blue")
245         rttHistogram.saveGraph(path+id+"-rtt-histogram.png")

```

```

233 # plota histograma de intervalos de chegada
234 def plotIntervalHistogram(self, path, id, intervals, title, xLabel,
235   ↪ yLabel):
236
237   intervalHistogram = GraphPlotter(title=title, xLabel=xLabel,
238     ↪ yLabel=yLabel, legendFlag=False)
239   intervalHistogram.plotHistogram(intervals, color="yellow")
240   intervalHistogram.saveGraph(path+id+"-interval-histogram.png")
241
242 # plota histograma de jitter baseado em rtt
243 def plotRttJitterHistogram(self, path, id, jitters, title, xLabel,
244   ↪ yLabel):
245
246   jitterHistogram = GraphPlotter(title=title, xLabel=xLabel,
247     ↪ yLabel=yLabel, legendFlag=False)
248   jitterHistogram.plotHistogram(jitters, color="red")
249   jitterHistogram.saveGraph(path+id+"-rtt-jitter-histogram.png")
250
251 # plota histograma de jitter baseado em intervalo de chegada
252 def plotIntervalJitterHistogram(self, path, id, jitters, title, xLabel,
253   ↪ yLabel):
254
255   jitterHistogram = GraphPlotter(title=title, xLabel=xLabel,
256     ↪ yLabel=yLabel, legendFlag=False)
257   jitterHistogram.plotHistogram(jitters, color="orange")
258   jitterHistogram.saveGraph(path+id+"-interval-jitter-histogram.png")
259
260 # plota gráfico de perda de pacotes
261 def plotLossGraph(self, path, id, lossStats, title, xLabel, yLabel):
262
263   lossGraph = GraphPlotter(title=title, xLabel=xLabel, yLabel=yLabel,
264     ↪ legendPosition="right")
265   lossGraph.plotBarGraph(["sent", "received", "lost"], lossStats,
266     ↪ ["gray", "green", "red"], ["Sent Packets", "Received Packets",
267     ↪ "Lost Packets"])
268   lossGraph.saveGraph(path+id+"-loss.png")
269
270 # plota gráfico de porcentagem de perda de pacotes
271 def plotLossRateGraph(self, path, id, lossRate):
272
273   lossRateGraph = GraphPlotter()
274   lossRateGraph.plotPizzaGraph(["received packets", "lost packets"],
275     ↪ [100-lossRate, lossRate], ["green", "red"])
276   lossRateGraph.saveGraph(path+id+"-loss-rate.png")

```

Implementação da classe PacketAnalyzer

```

1  # analisador de camada ICMP
2  class IcmpAnalyzer(PacketAnalyzer):
3
4      def __init__(self, id=None, packetsMargin=None, path=None):
5          super().__init__(id, packetsMargin, path)
6
7      # retorna número de sequência de pacote ICMP
8      def getIcmpSeq(self, pkt):
9
10         if ICMP in pkt:
11             return pkt[ICMP].seq
12
13         else:
14             print("The packet doesn't have an ICMP layer")
15             return 0
16
17     # retorna tipo de ICMP: 0 = echo request , 8 = echo reply
18     def getIcmpType(self, pkt):
19
20         if ICMP in pkt:
21             return pkt[ICMP].type
22
23         else:
24             print("The packet doesn't have an ICMP layer")
25             return 0
26
27     # retorna lista de sequência de pacotes ICMP em ordem crescente (sem
28     # → duplicatas)
29     def getIcmpSeqsList(self):
30
31         seqsList = set() # conjunto sem duplicatas
32         for pkt in self.getPackets():
33             if ICMP in pkt:
34                 seqsList.add(self.getIcmpSeq(pkt))
35
36         return sorted(list(seqsList)) if seqsList else []
37
38     # retorna estatísticas de rtt ICMP: lista de rtt, desvio padrão, média,
39     # → máximo, mínimo, erro padrão e coeficiente de variação
40     # override
41     def getRttStats(self):
42
43         rtts = []
44         requests = {}
45
46         for pkt in self.getPackets():
47             if ICMP in pkt:
48                 seq = self.getIcmpSeq(pkt)
49
50                 if self.getIcmpType(pkt) == 8: # echo request
51                     requests[seq] = pkt
52
53                 elif self.getIcmpType(pkt) == 0 and seq in requests: # echo
54                     → reply

```

```

52             rtts.append(self.getTimeDiff(requests[seq], pkt))
53
54         rtts = np.array(rtts) if rtts else []
55         mean = np.mean(rtts) if len(rtts) > 0 else 0
56         std = np.std(rtts) if len(rtts) > 0 else 0
57         max = np.max(rtts) if len(rtts) > 0 else 0
58         min = np.min(rtts) if len(rtts) > 0 else 0
59         error = std/np.sqrt(len(rtts)) if len(rtts) > 0 else 0
60         cv = (std/mean)*100 if mean > 0 else 0
61
62     return {"rtts": rtts,
63             "mean": mean,
64             "std": std,
65             "max": max,
66             "min": min,
67             "error": error,
68             "cv": cv
69             }
70
71     # retorna estatísticas de intervalo de chegada entre requisições ICMP:
72     # → lista de intervalos, média, desvio padrão, máximo, mínimo, erro padrão
73     # → e coeficiente de variação
74     # override
75     def getIntervalStats(self):
76
76         if self.getTotalPackets() < 2:
77             print("There is no way to measure interval with less than two
78                  packets")
79             return None
80
81         requestTimes = []
82
83         for pkt in self.getPackets():
84             if ICMP in pkt:
85                 if self.getIcmpType(pkt) == 8:
86                     requestTimes.append(self.getTime(pkt))
87
88         intervals = np.diff(requestTimes) if requestTimes else [] # diferença
89         # entre tempos consecutivos
90         mean = np.mean(intervals) if len(intervals) > 0 else 0
91         std = np.std(intervals) if len(intervals) > 0 else 0
92         max = np.max(intervals) if len(intervals) > 0 else 0
93         min = np.min(intervals) if len(intervals) > 0 else 0
94         error = std/np.sqrt(len(intervals)) if len(intervals) > 0 else 0
95         cv = (std/mean)*100 if mean > 0 else 0
96
97     return {"intervals": intervals,
98             "mean": mean,
99             "std": std,
100            "max": max,
101            "min": min,
102            "error": error,
103            "cv": cv
104            }

```

```

102
103     # retorna estatísticas de perda de pacotes: enviados, recebidos, perdidos,
104     # taxa de perdas
105     # override
106     def getLossStats(self):
107
107         sent = 0
108         seqReceived = set()
109
110         for pkt in self.getPackets():
111             if ICMP in pkt:
112                 if self.getIcmpType(pkt) == 8:
113                     sent += 1
114
115                 elif self.getIcmpType(pkt) == 0:
116                     seqReceived.add(self.getIcmpSeq(pkt))
117
118         received = len(seqReceived)
119         lost = sent - received
120         lossRate = (lost * 100)/sent if sent > 0 else 0
121         lossStats = [sent, received, lost]
122
123         return {"sent": sent,
124                 "received": received,
125                 "lost": lost,
126                 "lossRate": lossRate,
127                 "lossStats": lossStats
128                 }
129
130     # imprime métricas ICMP
131     # override
132     def printGeneralMetrics(self):
133
134         id = self.getId()
135         src = IpAnalyzer.getSrcIp(self.getPacket(0))
136         dst = IpAnalyzer.getDstIp(self.getPacket(0))
137         totalPackets = self.getTotalPackets()
138         totalBytes = self.getTotalBytes()
139         layers = self.getLayers()["layers"]
140         throughput = self.getThroughput()
141
142         return super().printGeneralMetrics(id, src, dst, totalPackets,
143                                         totalBytes, layers, throughput)
143
144     # override
145     def printRttMetrics(self):
146
147         layer = "ICMP"
148         mean = self.getRttStats()["mean"]
149         std = self.getRttStats()["std"]
150         max = self.getRttStats()["max"]
151         min = self.getRttStats()["min"]
152         error = self.getRttStats()["error"]
153         cv = self.getRttStats()["cv"]

```

```

154
155     return super().printRttMetrics(layer, mean, std, max, min, error, cv)
156
157     # override
158     def printIntervalMetrics(self):
159
160         layer = "ICMP"
161         mean = self.getIntervalStats()["mean"]
162         std = self.getIntervalStats()["std"]
163         max = self.getIntervalStats()["max"]
164         min = self.getIntervalStats()["min"]
165         error = self.getIntervalStats()["error"]
166         cv = self.getIntervalStats()["cv"]
167
168         return super().printIntervalMetrics(layer, mean, std, max, min, error,
169             ↪   cv)
170
171     # override
172     def printRttJitterMetrics(self):
173
174         layer = "ICMP"
175         rtts = self.getRttStats()["rtts"]
176         mean = self.getJitterStats(rtts)["mean"]
177         std = self.getJitterStats(rtts)["std"]
178         max = self.getJitterStats(rtts)["max"]
179         min = self.getJitterStats(rtts)["min"]
180         error = self.getJitterStats(rtts)["error"]
181         cv = self.getJitterStats(rtts)["cv"]
182
183         return super().printRttJitterMetrics(layer, mean, std, max, min,
184             ↪   error, cv)
185
186     # override
187     def printIntervalJitterMetrics(self):
188
189         layer = "ICMP"
190         intervals = self.getIntervalStats()["intervals"]
191         mean = self.getJitterStats(intervals)["mean"]
192         std = self.getJitterStats(intervals)["std"]
193         max = self.getJitterStats(intervals)["max"]
194         min = self.getJitterStats(intervals)["min"]
195         error = self.getJitterStats(intervals)["error"]
196         cv = self.getJitterStats(intervals)["cv"]
197
198         return super().printIntervalJitterMetrics(layer, mean, std, max, min,
199             ↪   error, cv)
200
201     # override
202     def printLossMetrics(self):
203
204         layer = "ICMP"
205         sent = self.getLossStats()["sent"]
206         received = self.getLossStats()["received"]
207         lost = self.getLossStats()["lost"]

```

```

205     lossRate = self.getLossStats()["lossRate"]
206
207     return super().printLossMetrics(layer, sent, received, lost, lossRate)
208
209     # plotagem de gráficos ICMP
210     # override
211     def plotLayersGraph(self, path):
212
213         id = self.getId()
214         layers = self.getLayers()["layers"]
215         nLayers = self.getLayers()["nLayers"]
216         title = None
217         xLabel = "Protocol layers"
218         yLabel = "Amount of packets"
219
220         return super().plotLayersGraph(path, id, layers, nLayers, title,
221             xLabel, yLabel)
222
223     # override
224     def plotRttGraph(self, path):
225
226         id = self.getId()
227         xAxis = self.getIcmpSeqsList()
228         rtts = self.getRttStats()["rtts"]
229         title = None
230         xLabel = "ICMP sequence number"
231         yLabel = "Time (ms)"
232
233         return super().plotRttGraph(path, id, xAxis, rtts, title, xLabel,
234             yLabel)
235
236     # override
237     def plotIntervalGraph(self, path):
238
239         id = self.getId()
240         xAxis = self.getIcmpSeqsList()
241         intervals = self.getIntervalStats()["intervals"]
242         title = None
243         xLabel = "ICMP sequence number"
244         yLabel = "Time (ms)"
245
246         return super().plotIntervalGraph(path, id, xAxis[1:], intervals,
247             title, xLabel, yLabel)
248
249     # override
250     def plotRttJitterGraph(self, path):
251
252         id = self.getId()
253         rtts = self.getRttStats()["rtts"]
254         xAxis = self.getIcmpSeqsList()
255         jitters = self.getJitterStats(rtts)["jitters"]
256         title = None
257         xLabel = "ICMP sequence number"
258         yLabel = "Time (ms)"

```

```

256
257     return super().plotRttJitterGraph(path, id, xAxis[1:], jitters, title,
258                                         xlabel, ylabel)
259
260     # override
261     def plotIntervalJitterGraph(self, path):
262
262         id = self.getId()
263         intervals = self.getIntervalStats()["intervals"]
264         xAxis = self.getIcmpSeqsList()
265         jitters = self.getJitterStats(intervals)["jitters"]
266         title = None
267         xlabel = "ICMP sequence number"
268         ylabel = "Time (ms)"
269
270         return super().plotIntervalJitterGraph(path, id, xAxis[2:], jitters,
271                                         title, xlabel, ylabel)
271
272     # override
273     def plotRttHistogram(self, path):
274
275         id = self.getId()
276         rtts = self.getRttStats()["rtts"]
277         title = None
278         xlabel = "RTT interval (ms)"
279         ylabel = "Frequency"
280
281         return super().plotRttHistogram(path, id, rtts, title, xlabel, ylabel)
282
283     # override
284     def plotIntervalHistogram(self, path):
285
286         id = self.getId()
287         intervals = self.getIntervalStats()["intervals"]
288         title = None
289         xlabel = "Packet arrival time interval (ms)"
290         ylabel = "Frequency"
291
292         return super().plotIntervalHistogram(path, id, intervals, title,
293                                         xlabel, ylabel)
293
294     # override
295     def plotRttJitterHistogram(self, path):
296
297         id = self.getId()
298         rtts = self.getRttStats()["rtts"]
299         jitters = self.getJitterStats(rtts)["jitters"]
300         title = None
301         xlabel = "Jitter interval (ms)"
302         ylabel = "Frequency"
303
304         return super().plotRttJitterHistogram(path, id, jitters, title,
305                                         xlabel, ylabel)
305

```

```

306     # override
307     def plotIntervalJitterHistogram(self, path):
308
309         id = self.getId()
310         intervals = self.getIntervalStats()["intervals"]
311         jitters = self.getJitterStats(intervals)["jitters"]
312         title = None
313         xLabel = "Jitter interval (ms)"
314         yLabel = "Frequency"
315
316         return super().plotIntervalJitterHistogram(path, id, jitters, title,
317             xLabel, yLabel)
317
318     #override
319     def plotLossGraph(self, path):
320
321         id = self.getId()
322         lossStats = self.getLossStats()["lossStats"]
323         title = None
324         xLabel = "Statistics"
325         yLabel = "Amount of packets"
326
327         return super().plotLossGraph(path, id, lossStats, title, xLabel,
328             yLabel)
328
329     #override
330     def plotLossRateGraph(self, path):
331
332         id = self.getId()
333         lossRate = self.getLossStats()["lossRate"]
334
335         return super().plotLossRateGraph(path, id, lossRate)

```

Implementação da classe IcmpAnalyzer

```

1  # analisador de camada IP
2  class IpAnalyzer(PacketAnalyzer):
3
4      def __init__(self, id=None, path=None):
5          super().__init__(id, path)
6
7      # retorna IPv4 de origem
8      @staticmethod
9      def getSrcIp(pkt):
10
11         if IP in pkt:
12             return pkt[IP].src
13
14     else:
15         print("The packet doesn't have an IP layer")
16         return 0
17
18     # retorna IPv4 de destino
19     @staticmethod
20     def getDstIp(pkt):
21
22         if IP in pkt:
23             return pkt[IP].dst
24
25     else:
26         print("The packet doesn't have an IP layer")
27         return 0

```

Implementação da classe IpAnalyzer

## Referências

- [1] MININET. mininet/mininet. Disponível em:<<https://github.com/mininet/mininet>>.
- [2] MININET TEAM. Download/Get Started with Mininet - Mininet. Disponível em:<<http://mininet.org/download>>.
- [3] BONDI, P. Introduction — Scapy 2.4.3. documentation. Disponível em:<<https://scapy.readthedocs.io/en/latest/introduction.html>>.