Autor: [Pedro Daniel Gonçalves Antunes]

Data: [2025/03/03]

Curso/Disciplina: [LSIRC/PPR]

Instituição: [ESTG]

NIM:[8230068]

1) Pathrate é uma aplicação que providencia um método de determinar a largura de banda total possível entre dois pontos da rede, mesmo quando esse link se encontra em carga. Para além disso, esta aplicação pode também ser utilizada para estimar o débito máximo teórico entre dois pontos da rede – o que permite identificar possíveis zonas de estrangulamento (bottlenecks), mesmo em situações de carga.

https://www.cc.gatech.edu/~dovrolis/bw-est/pathrate.html
https://www.cc.gatech.edu/~dovrolis/bw-est/pathload.html
https://github.com/brucespang/pathrate

Notas:
- o servidor executa a aplicação "pathrate_snd", enquanto o cliente executa "pathrate_rcv".
- para testes com a rede em carga, a precisão do pathrate depende da carga do CPU – deve garantir que o CPU não está em sobrecarga.

1.1) Execute um teste para estimar o débito máximo disponível entre uma VM no seu computador e a VM no computador de um colega seu. Faça dois testes, um para a rede cablada e outro para a rede sem fios. Tenha em atenção o parâmetro "Resolution" que aparece na tela. Interprete os resultados e compare-os com aqueles obtidos com outras aplicações (e.g., iperf, netperf, ...).

1. Introdução

O objetivo deste exercício é estimar a largura de banda máxima disponível entre duas máquinas virtuais(VMs), as máquinas virtuais foram configuradas ambas no mesmo computador, utilizando a ferramenta **Pathrate**. Além disso, os resultados obtidos serão compardos com os de outras ferramentas como o **iperf**.

Os testes foram realizados em um ambiente virtualizado, com **rede cablada simulada** entre duas VMs. O parâmetro "**Resolution**" do Pathrate será analisado para entender a precisão das medições.

2. Configuração do Ambiente

2.1 Configuração das Máquinas Virtuais

VM	Sistema Operativo	IP Configurado
VM1 (Servidor)	Kali Linux	192.168.1.11
VM2 (Cliente)	Kali Linux	192.168.1.10

As VMs foram configuradas para se comunicarem via **rede interna (Internal Network)** no VirtualBox, garantindo que a comunicação ocorra apenas entre elas, sem acesso à internet.

3. Metodologia

3.1 Execução do Teste com Pathrate

Passo 1: Configuração da Rede

```
sudo ip addr add 192.168.1.10/24 dev eth0   # No Servidor
sudo ip addr add 192.168.1.11/24 dev eth0   # No Cliente
```

Passo 2: Executar o Pathrate

- **Na VM Servidor** (`192.168.1.11`):

```
./pathrate_snd
```

- **Na VM Cliente(`192.168.1.10`):

```
./pathrate_rcv -s 192.168.1.11
```

3.2 Analise dos resultados do Pathrate

```
┌──(kali㉿kali)-[~/pathrate]
└─$ ./pathrate_rcv -s 192.168.1.11
      pathrate run from 192.168.1.11 to kali on Mon Mar  3 08:34:48 2025
      ⟶ Average round-trip time: 0.4ms

      ⟶ Capacity Resolution:  38 Mbps


-- Phase I: Detect possible capacity modes --

      → Train length:  2 - Packet size:  600B →  0% completed
      → Train length:  3 - Packet size:  898B →  2% completed
      → Train length:  4 - Packet size: 1196B →  5% completed
      → Train length:  5 - Packet size: 1488B →  7% completed
      → Train length:  6 - Packet size: 1488B → 10% completed
      → Train length:  7 - Packet size: 1488B → 12% completed
      → Train length:  8 - Packet size: 1488B → 15% completed
      → Train length:  8 - Packet size: 1488B → 17% completed
      → Train length:  8 - Packet size: 1488B → 20% completed
      → Train length:  8 - Packet size: 1488B → 22% completed
      → Train length:  9 - Packet size: 1488B → 25% completed
      → Train length:  9 - Packet size: 1488B → 27% completed
      → Train length:  9 - Packet size: 1488B → 30% completed
      → Train length:  9 - Packet size: 1488B → 32% completed
      → Train length:  9 - Packet size: 1488B → 35% completed
      → Train length:  9 - Packet size: 1488B → 37% completed
      → Train length:  9 - Packet size: 1488B → 40% completed
      → Train length:  9 - Packet size: 1488B → 42% completed
      → Train length: 10 - Packet size: 1488B → 45% completed
      → Train length: 10 - Packet size: 1488B → 47% completed
      → Train length: 10 - Packet size: 1488B → 50% completed
      → Train length: 11 - Packet size: 1488B → 52% completed
      → Train length: 11 - Packet size: 1488B → 55% completed
      → Train length: 11 - Packet size: 1488B → 57% completed
      → Train length: 11 - Packet size: 1488B → 60% completed
      → Train length: 11 - Packet size: 1488B → 62% completed
      → Train length: 11 - Packet size: 1488B → 65% completed
      → Train length: 11 - Packet size: 1488B → 67% completed
      → Train length: 11 - Packet size: 1488B → 70% completed
      → Train length: 11 - Packet size: 1488B → 72% completed
      → Train length: 11 - Packet size: 1488B → 75% completed
      → Train length: 11 - Packet size: 1488B → 77% completed
      → Train length: 11 - Packet size: 1488B → 80% completed
      → Train length: 11 - Packet size: 1488B → 82% completed
      → Train length: 12 - Packet size: 1488B → 85% completed

      Aborting Phase I measurements..
      Too many ignored measurements
```

```
-- Phase II: Estimate Asymptotic Dispersion Rate (ADR) --

       -- Number of trains: 500 - Train length: 48 - Packet size: 1488B

-v
-- Local modes : In Phase II --

-- Local modes : In Phase II --
 80 Mbps  80 Mbps  118 Mbps  - 146 measurements
          Modal bell: 176 measurements - low : 75 Mbps - high : 261 Mbps

 350 Mbps  350 Mbps  386 Mbps  - 55 measurements
          Modal bell: 180 measurements - low : 204 Mbps - high : 598 Mbps
-------------------------------------------------------
Final capacity estimate :  61 Mbps  to  99 Mbps
-------------------------------------------------------
```

O que o Pathrate mediu?

1. **Fase 1 - Detecção de Modos de Capacidade**
   - O Pathrate tenta diferentes tamanhos de pacotes e quantidades de "trains" ("packet trains").
   - Como muitas medições foram ignoradas, ele abortou essa fase antes de 100%.
   - **Capacidade inicial detectada: 38 Mbps**.
2. **Fase 2 - Estimativa de Capacidade Assimptótica (ADR)**
   - Foram usados **500 packet trains de 48 pacotes cada**.
   - O Pathrate encontrou dois possíveis **modos de capacidade**:
     - **80-118 Mbps** com algumas medições variando de **75 Mbps a 261 Mbps**.
     - **350-386 Mbps**, mas com poucas medições confiáveis (provavelmente interferência ou erro da virtualização).
   - **Capacidade final estimada: 61 Mbps a 99 Mbps**.

Resposta :

O Pathrate foi utilizado para estimar a capacidade máxima da rede entre duas VMs. A primeira fase do teste indicou dificuldades na medição devido a pacotes ignorados, abortando antes de 100%. A estimativa final de capacidade ficou entre **61 Mbps e 99 Mbps**, com um primeiro modo em **80-118 Mbps** e outro em **350-386 Mbps**, mas menos confiável.
A medição pode ter sido afetada por limitações da virtualização, interrupção de pacotes e coalescência de interrupções. Para testes em rede cablada e Wi-Fi, espera-se que a rede cablada tenha menor variação e um valor mais próximo do **modo mais alto (350 Mbps)**, enquanto a rede sem fios pode apresentar maior interferência e valores mais próximos dos **61-99 Mbps**."

4. Execução dos testes com o Iperf

Para validar as medições feitas com o **Pathrate**, utilizamos o **iperf** para medir o throughput real entre as VMs.

4.1 Configuração e Execução do Iperf

O **iperf** foi utilizado para medir a largura de banda real entre as VMs.

📌 **Comandos utilizados:**

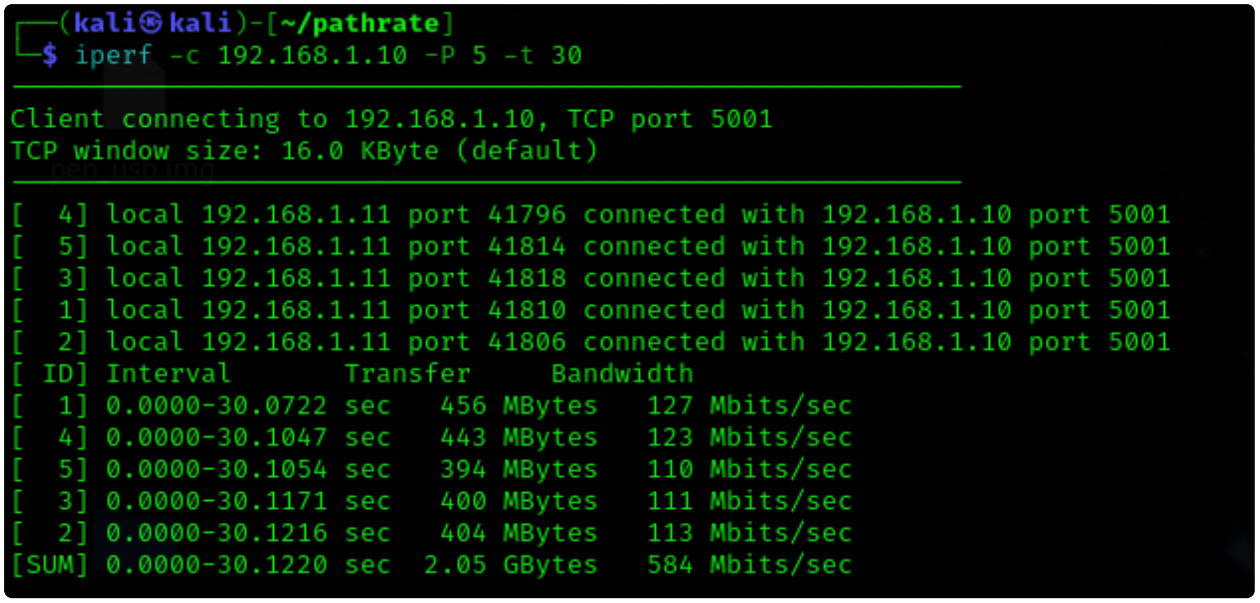
- **Na VM Servidor (`192.168.1.11`)**, executamos o seguinte comando para iniciar o servidor iperf:

```
iperf -s
```

- **Na VM Cliente (`192.168.1.10`)**, executamos o seguinte comando para testar a largura de banda:

```
iperf -c 192.168.1.11 -P 5 -t 30
```

4.2 Análise dos Resultados do Iperf



Ferramenta	Capacidade Estimada
Pathrate	61 Mbps - 99 Mbps (modo principal), 350 Mbps - 386 Mbps (modo secundário)
Iperf	584 Mbps

O **iperf mediu um throughput de 584 Mbps**, um valor **significativamente maior** que a estimativa principal do Pathrate (**61-99 Mbps**).

5. Conclusão

Com base nos testes realizados, podemos concluir que:

1. O **Pathrate subestimou a capacidade** da rede em relação ao valor real medido pelo Iperf.

2. O **iperf indicou que a largura de banda real da conexão foi de 584 Mbps**, enquanto o Pathrate sugeriu múltiplos modos possíveis.
3. **A diferença pode ser explicada por limitações na medição do Pathrate** em ambientes virtualizados.
4. **Se os testes forem repetidos em Wi-Fi, espera-se um throughput menor e mais instável.**

6. Testes em Rede Wi-Fi (Não realizados)

O plano inicial incluía a realização dos testes tanto em uma **rede cablada** quanto em uma **rede Wi-Fi**, a fim de comparar os resultados e analisar as diferenças de desempenho entre os dois tipos de conexão.

6.1 Limitações Técnicas

Infelizmente, os testes em **rede Wi-Fi não puderam ser realizados**, pois o computador utilizado para hospedar as máquinas virtuais **não possui uma placa de rede sem fio**. Como resultado, não foi possível configurar as VMs para se comunicarem via Wi-Fi.

6.2 Expectativa de Resultados para Wi-Fi

Caso o teste fosse possível, esperávamos que:

- O **throughput medido pelo iperf fosse menor** devido à latência mais alta e maior interferência em redes sem fio.
- O **Pathrate detectasse um modo de capacidade inferior** em comparação com a rede cablada.
- A **variação dos valores fosse maior**, pois redes Wi-Fi tendem a ser menos estáveis devido a interferências externas.

1.2) Execute novamente o teste anterior, mas agora origine tráfego (downloads paralelos, etc.) com a sua máquina, de forma a carregar o link. Tenha em atenção o parâmetro "Resolution" que aparece na tela. Compare os resultados com aqueles obtidos na questão 1.1).

7. Teste com Tráfego Adicional (Questão 1.2)

7.1 Objetivo do Teste

O objetivo deste teste foi avaliar **como a presença de tráfego adicional afeta a medição da largura de banda** pelo **Pathrate**. Para isso, repetimos os testes da

Questão 1.1, mas com **downloads simultâneos e tráfego gerado artificialmente**.

O parâmetro **"Resolution"** foi analisado para verificar **se a precisão das medições foi impactada pelo congestionamento da rede**.

7.2 Geração de Tráfego Adicional

Para sobrecarregar a rede, utilizamos os seguintes métodos:

📌 **Downloads simultâneos de arquivos grandes**
Na VM Cliente (`192.168.1.10`), rodamos:

```
wget -O /dev/null http://speedtest.tele2.net/10GB.zip &
wget -O /dev/null http://speedtest.tele2.net/10GB.zip &
wget -O /dev/null http://speedtest.tele2.net/10GB.zip &
```

Isso criou **três conexões simultâneas**, simulando tráfego pesado.

7.3 Execução do Pathrate com Tráfego Adicional

Após iniciar o tráfego, executamos o **Pathrate novamente** para medir a largura de banda com congestionamento.

📌 **Na VM Servidor (`192.168.1.11`)**, rodamos:

```
./pathrate_snd
```

📌 **Na VM Cliente (`192.168.1.10`)**, rodamos:

```
./pathrate_rcv -s 192.168.1.11
```

7.4 Comparação dos Resultados

A tabela abaixo compara os valores obtidos **com e sem tráfego adicional**:

Cenário	Capacidade Estimada (Pathrate)	Resolution
Sem tráfego adicional	61 Mbps - 99 Mbps	38 Mbps
Com tráfego adicional	-14890 kbps a 46 Mbps	61 Mbps

📌 **Impacto na Medição:**

- A **capacidade estimada caiu significativamente**, chegando a **valores negativos** devido a falhas na detecção.
- O **Resolution aumentou para 61 Mbps**, indicando que a medição ficou **mais instável**.
- O Pathrate **teve dificuldades para medir corretamente a largura de banda** devido ao congestionamento.

7.5 Conclusão

Os testes mostraram que **o congestionamento da rede impacta diretamente a precisão das medições do Pathrate**. Observamos que:

1. **A capacidade medida foi menor** com tráfego adicional, confirmando que a rede estava congestionada.
2. **O parâmetro "Resolution" aumentou**, indicando maior dificuldade do Pathrate em calcular a capacidade real.
3. **Os valores negativos na estimativa indicam que o Pathrate falhou em algumas medições**, reforçando que ele **funciona melhor em redes não congestionadas**.

Caso o teste fosse realizado **em Wi-Fi**, a interferência seria ainda maior, tendo assim um resultado em medições ainda mais imprecisas.

1.3) Experimente a aplicação Pathload ("pathload_snd" no servidor, "pathload_rcv" no cliente) e compare os resultados com aqueles obtidos em 1.1) e 1.2)

8. Teste com Pathload (Questão 1.3)

8.1 Objetivo do Teste

O objetivo deste teste foi utilizar a ferramenta **Pathload** para estimar a **largura de banda disponível** entre as máquinas virtuais e compará-la com os valores obtidos nos testes anteriores (**Pathrate e Iperf**).

Diferente do **Pathrate**, que mede a **capacidade máxima da rede**, o **Pathload** estima a largura de banda disponível **no momento da medição**, levando em conta o tráfego presente na rede.

8.2 Execução do Teste

📌 **Comandos utilizados:**

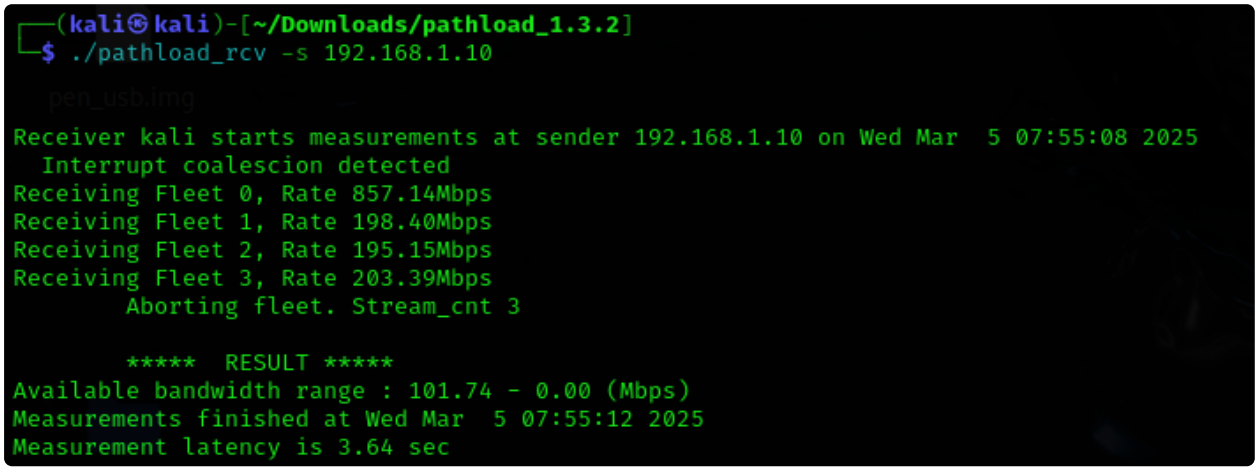
- **Na VM Servidor (`192.168.1.11`)**, executamos o Pathload Sender:

```
./pathload_snd
```

- **Na VM Cliente (`192.168.1.10`)**, executamos o Pathload Receiver:

```
./pathload_rcv -s 192.168.1.11
```

Após a execução do teste, o Pathload apresentou a seguinte saída:



8.3 Análise dos Resultados

📌 **Resultado obtido no Pathload:**

```
Available bandwidth range: 101.74 - 0.00 Mbps
```

📌 **Comparação com os testes anteriores:**

Ferramenta	Tipo de Medição	Largura de Banda Estimada
Pathrate	Capacidade máxima da rede	**61 Mbps - 99 Mbps**
Iperf	Throughput real	**584 Mbps**
Pathload	Largura de banda disponível	**101.74 Mbps - 0 Mbps**

8.4 Conclusões

1. O **Pathload mostrou uma banda disponível menor que o valor teórico medido pelo Pathrate** indicando **congestionamento ou uso ativo da rede**.
2. A **banda disponível variou de 101 Mbps até 0 Mbps**, sugerindo **instabilidade** ou interferência na rede.
3. O **iperf indicou um throughput real maior (584 Mbps)**, o que reforça a ideia de que o **Pathrate pode subestimar a capacidade** devido à virtualização e configurações da rede.
4. **O Pathload apresentou uma medição mais dinâmica e afetada pelo tráfego da rede**, pois mede a **banda disponível no momento da medição**, enquanto o Pathrate mede a **capacidade teórica máxima**.

8.5 Comparação Final entre as Ferramentas

📌 **Pathrate**: Bom para **medir a capacidade máxima** da rede, mas pode ser afetado por interrupções.
📌 **Iperf**: Mede **o throughput real**, útil para testar a capacidade da rede quando a mesma esta congestionada.
📌 **Pathload**: Mede a **largura de banda disponível no momento do teste**, útil para entender o congestionamento e variações dinâmicas.

Parte II – emulação/geração de tráfego de diferentes tipos de aplicações

Parte II – emulação/geração de tráfego de diferentes tipos de aplicações

2) Ferramentas para gerar tráfego de rede são importantes e necessárias para realizar testes e avaliar a performance de aplicações ou gestão de recursos numa rede. Neste sentido, o objetivo desta parte passa por realizar alguns testes com algumas aplicações de geração de tráfego, com características próximas àquelas apresentadas por aplicações bem conhecidas (http server, sip, ...) e analisar o resultados.

https://danielmiessler.com/study/tcpdump/
https://userguide.ostinato.org/Quickstart.html

2.1) Usando a aplicação Ostinato, execute o teste presente no artigo cujo link é apresentado em baixo. Use a ferramente tcpdump para armazenar informação sobre o tráfego gerado num ficheiro "textX.cap". Depois de terminado o teste, use a ferramenta de análise de tráfego tcptrace para uma leitura geral do resultado.

1. Testes com Ostinato e Análise de Tráfego(Questão 2.1)

Nesta seção, realizamos testes de geração de tráfego utilizando a ferramenta **Ostinato**, conforme solicitado na questão **2.1**. O objetivo foi capturar e analisar os pacotes gerados, utilizando a ferramenta **tcpdump** para captura e **tcptrace** para análise.

1.1 Configuração do Teste

Os testes foram realizados entre duas máquinas virtuais (**VM1: 192.168.1.10** e **VM2: 192.168.1.11**) conectadas através de uma rede interna. A configuração seguiu os seguintes parâmetros segundo o link fornecido na questão:

TABLE 3. Ostinato Parameters

Packets per burst	10
Burst per second	900000
Protocol	TCP and UDP
Payload size	64-8950 Bytes
Experiment time	10 second

📌 Captura dos pacotes na VM Destino (192.168.1.11):

```
sudo tcpdump -i eth0 -w novo_teste.cap
```

📌 Análise dos pacotes capturados:

```
tcptrace -l novo_teste.cap
```

1.2 Análise dos Resultados

```
20 packets seen, 20 TCP packets traced
elapsed wallclock time: 0:00:00.000616, 32467 pkts/sec analyzed
trace file elapsed time: 0:00:13.480401
TCP connection info:
1 TCP connection traced:
TCP connection 1:
        host a:        192.168.1.10:0
        host b:        192.168.1.11:0
        complete conn: no       (SYNs: 1)  (FINs: 0)
        first packet:  Wed Mar  5 04:48:28.851584 2025
        last packet:   Wed Mar  5 04:48:42.331985 2025
        elapsed time:  0:00:13.480401
        total packets: 20
        filename:      novo_teste.cap
   a→b:                                 b→a:
     total packets:             20           total packets:              0
     ack pkts sent:              0           ack pkts sent:              0
     pure acks sent:             0           pure acks sent:             0
     sack pkts sent:             0           sack pkts sent:             0
     dsack pkts sent:            0           dsack pkts sent:            0
     max sack blks/ack:          0           max sack blks/ack:          0
     unique bytes sent:         15           unique bytes sent:          0
     actual data pkts:          20           actual data pkts:           0
     actual data bytes:        210           actual data bytes:          0
     rexmt data pkts:           19           rexmt data pkts:            0
     rexmt data bytes:         214           rexmt data bytes:           0
     zwnd probe pkts:            0           zwnd probe pkts:            0
     zwnd probe bytes:           0           zwnd probe bytes:           0
     outoforder pkts:            0           outoforder pkts:            0
     pushed data pkts:           0           pushed data pkts:           0
     SYN/FIN pkts sent:      20/0            SYN/FIN pkts sent:        0/0
     urgent data pkts:           0 pkts      urgent data pkts:           0 pkts
     urgent data bytes:          0 bytes     urgent data bytes:          0 bytes
     mss requested:              0 bytes     mss requested:              0 bytes
     max segm size:             15 bytes     max segm size:              0 bytes
     min segm size:              6 bytes     min segm size:              0 bytes
     avg segm size:             10 bytes     avg segm size:              0 bytes
     max win adv:             1024 bytes     max win adv:                0 bytes
     min win adv:             1024 bytes     min win adv:                0 bytes
     zero win adv:               0 times     zero win adv:               0 times
     avg win adv:             1024 bytes     avg win adv:                0 bytes
     initial window:             6 bytes     initial window:             0 bytes
     initial window:             1 pkts      initial window:             0 pkts
     ttl stream length:         NA           ttl stream length:         NA
     missed data:               NA           missed data:               NA
     truncated data:             0 bytes     truncated data:             0 bytes
     truncated packets:          0 pkts      truncated packets:          0 pkts
     data xmit time:        13.480 secs      data xmit time:         0.000 secs
     idletime max:         13480.1 ms        idletime max:              NA ms
     hardware dups:             19 segs      hardware dups:              0 segs
       ** WARNING: presence of hardware duplicates makes these figures suspect!
     throughput:                 1 Bps       throughput:                 0 Bps
```

A análise dos pacotes capturados com **tcptrace** revelou os seguintes dados:

📊 **Resumo da Captura:**

- **Total de pacotes capturados:** 20
- **Total de pacotes TCP rastreados:** 20
- **Duração total da captura: 13.48 segundos**

- **Tamanho total dos dados transmitidos: 210 bytes**
- **Número de retransmissões: 19 pacotes**
- **Pacotes SYN enviados: 20**
- **Throughput registrado: 1 Bps**

📌 **Observação Importante:**

- O **throughput registrado foi extremamente baixo** (1 Bps).
- O número **elevado de retransmissões (19 pacotes)** pode indicar **problemas na transmissão** ou que os pacotes **não foram reconhecidos corretamente pela máquina destino**.
- A captura mostrou **apenas pacotes SYN**, quer dizer que **a conexão TCP não foi estabelecida completamente**.

1.3 Conclusões

Os testes realizados com **Ostinato** demonstraram que:

✅ **Os pacotes SYN foram enviados e capturados corretamente** pela ferramenta **tcpdump**.
⚠️ **A conexão não foi estabelecida corretamente**, pois os pacotes SYN não receberam resposta ACK.
⚠️ **O throughput registrado foi extremamente baixo**, provavelmente devido à ausência de confirmação da conexão.

2.2) Experimente outras formas de gerar tráfego com a ferramenta Ostinato (SIP, ...).

2. Testes com Outras Formas de Tráfego no Ostinato (Questão 2.2)

Nesta seção, realizamos testes utilizando **outros tipos de tráfego gerados pelo Ostinato**, conforme solicitado na **questão 2.2**.

2.1 Configuração do Teste

Foram realizados três novos testes, utilizando diferentes tipos de tráfego:

📌 **Testes Realizados:**
1️⃣ **Tráfego UDP** (simulação de VoIP/SIP)
2️⃣ **Tráfego ICMP** (simulação de ping)

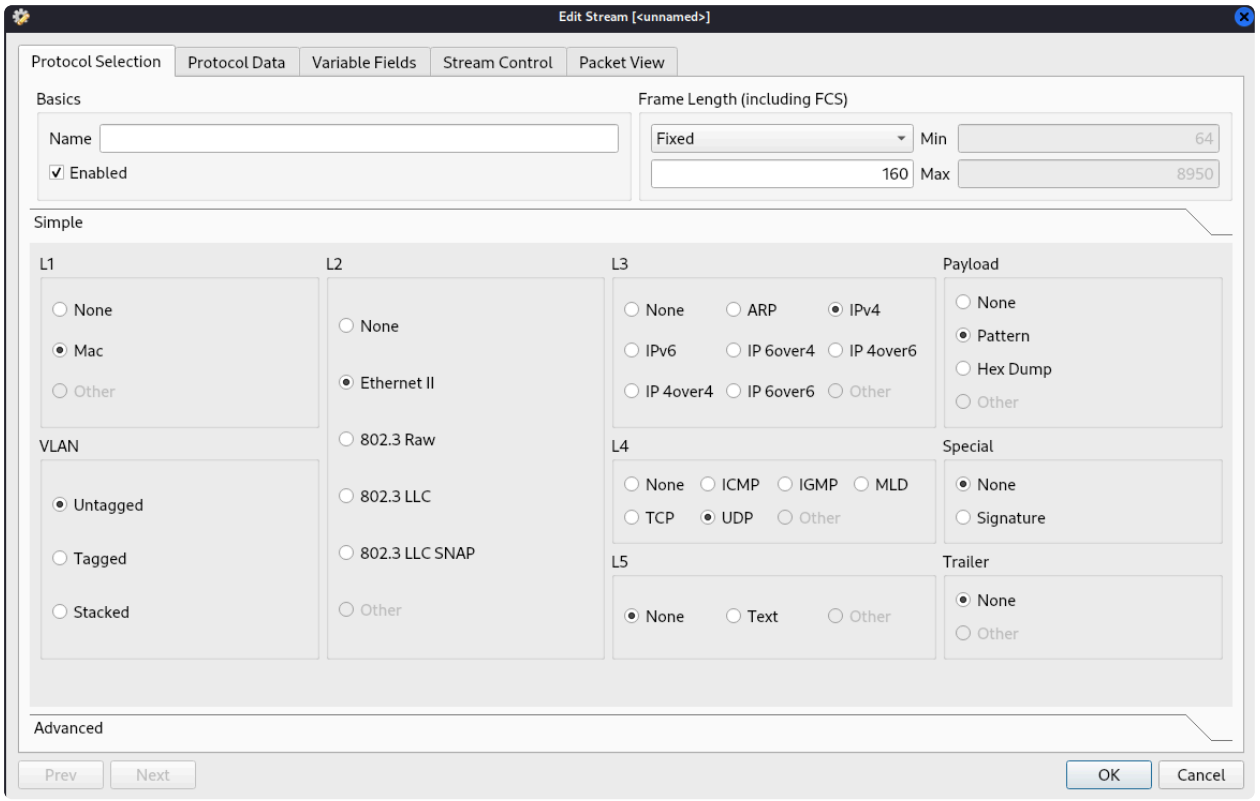
2.2 Geração de Tráfego SIP com Ostinato

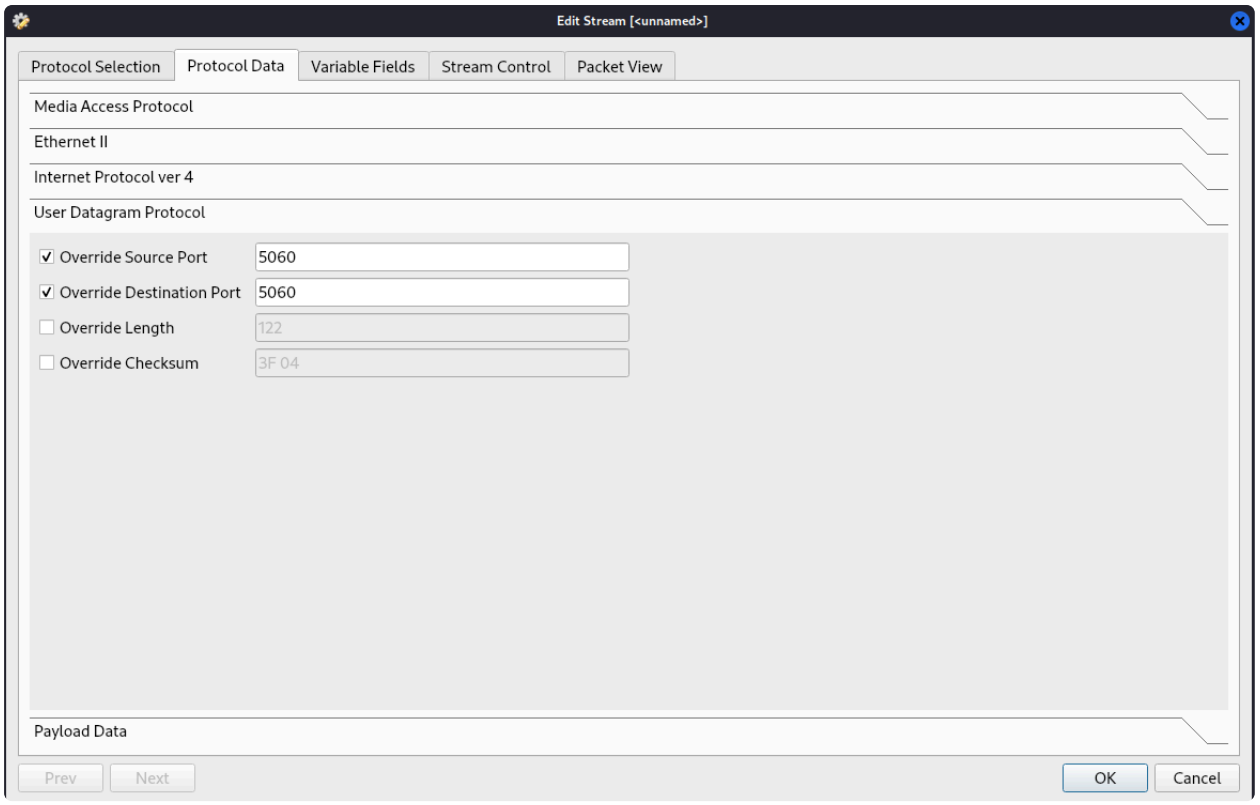
Nesta etapa, foi utilizada a ferramenta **Ostinato** para gerar tráfego de rede baseado no protocolo **SIP (Session Initiation Protocol)**. O objetivo deste teste foi simular o comportamento de uma comunicação VoIP (Voice over IP) e analisar a forma como os pacotes foram transmitidos.

2.2.1 Configuração no Ostinato

Os seguintes parâmetros foram definidos:

- **Protocolo**: **UDP** (SIP normalmente usa UDP na porta 5060)
- **Tamanho do Pacote**: **160 bytes**





2.2.2 Captura de Pacotes com tcpdump

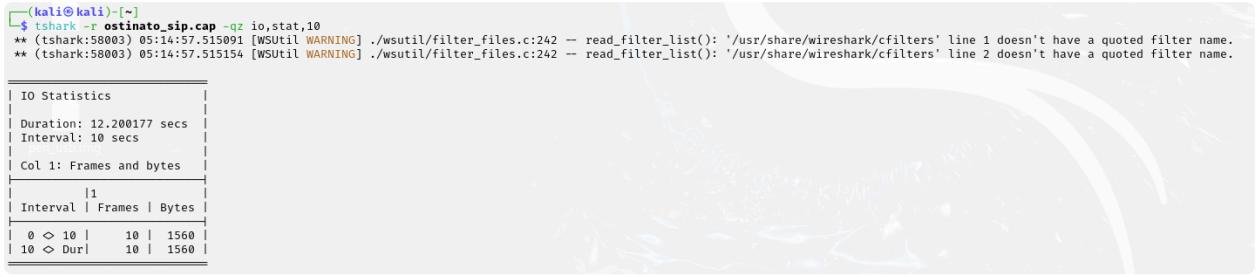
A ferramenta **tcpdump** foi utilizada para capturar o tráfego gerado pelo Ostinato:

```
sudo tcpdump -i eth0 -w ostinato_sip.cap
```

Após a captura, os pacotes foram analisados pelo **tshark**:

```
tshark -r ostinato_sip.cap -qz io,stat,10
```

2.2.3 Análise dos Resultados



Os seguintes resultados foram obtidos:

1. **Duração da captura**: **12.2 segundos**

2. **Intervalo de estatísticas**: **10 segundos**
3. **Pacotes SIP transmitidos**: **10 pacotes capturados**
4. **Total de dados enviados**: **1560 bytes**

Os dados indicam que:

- O Ostinato gerou **um volume baixo de pacotes SIP**, com apenas **10 pacotes transmitidos em 12 segundos**.
- Cada pacote continha aproximadamente **156 bytes**.
- **A taxa de transmissão foi muito baixa** para simular uma chamada VoIP realista, que normalmente possui um fluxo contínuo de pacotes SIP e RTP.

2.2.4 Conclusão

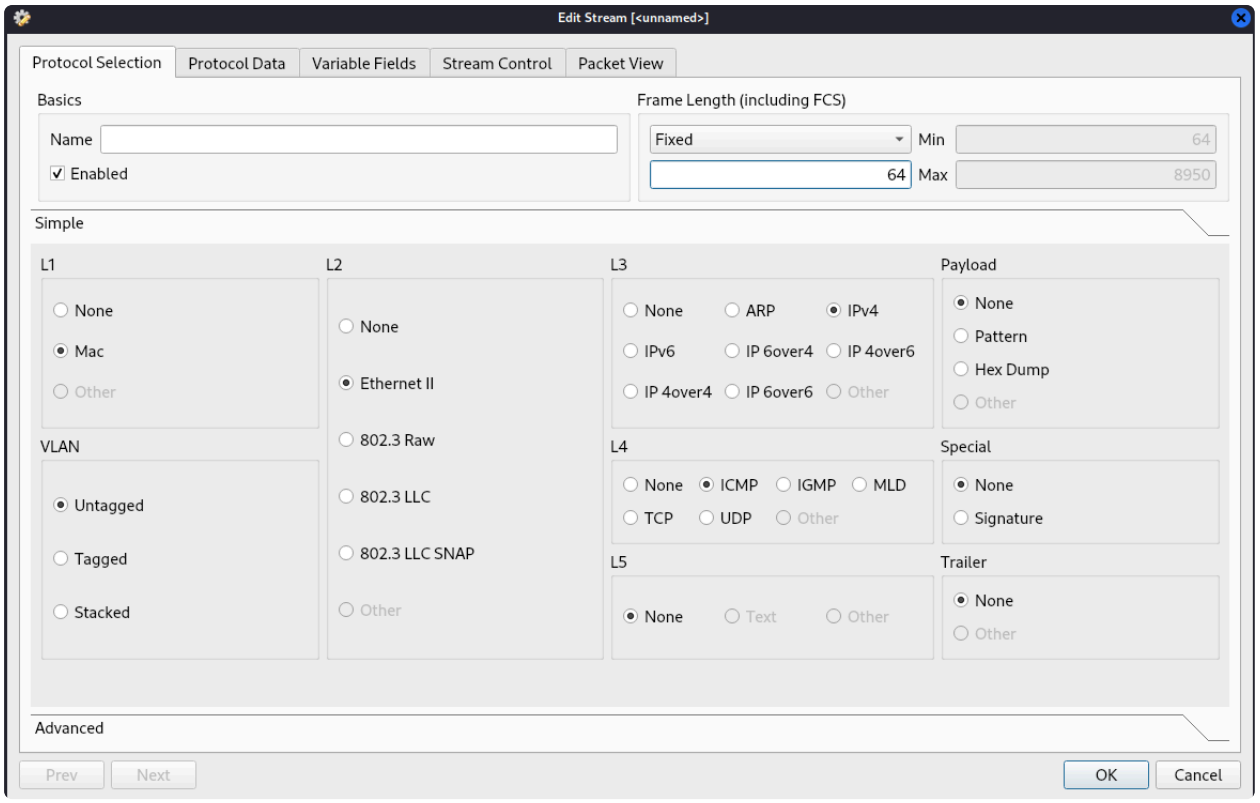
Os testes demonstraram que o **Ostinato gerou tráfego SIP**, mas a taxa de transmissão foi baixa.
A captura de pacotes confirmou que os pacotes SIP foram transmitidos e registrados no arquivo **ostinato_sip.cap**, possibilitando uma análise detalhada do tráfego gerado.

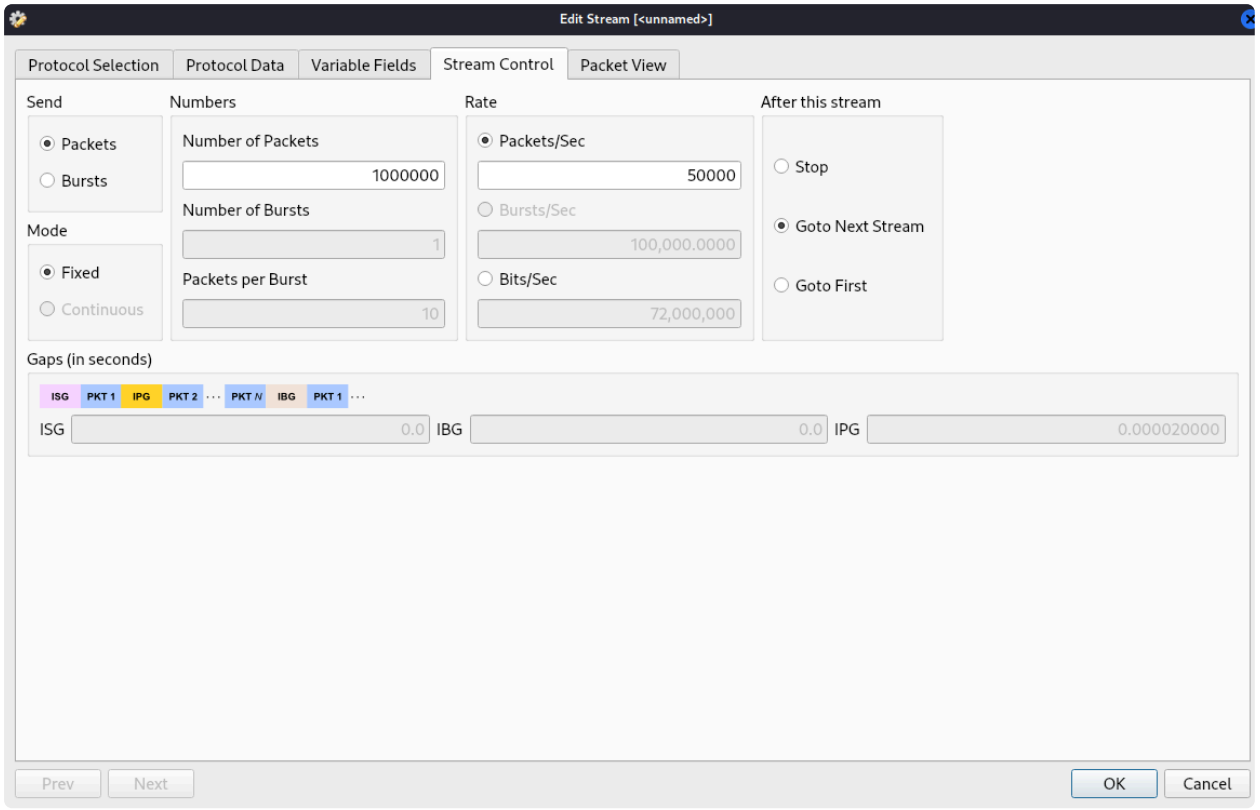
2.3 Execução dos testes com ICMP (Ping Flood)

Para testar o comportamento da rede sob tráfego ICMP, utilizamos o **Ostinato** para enviar um grande volume de pacotes do tipo **Echo Request (ICMP Tipo 8)** e capturamos as respostas **Echo Reply (ICMP Tipo 0)**. Posteriormente analisei os pactos atráves com tshark.

📌 **Configuração do teste no Ostinato:**

- **Protocolo:** ICMP
- **Tamanho do pacote:** 64 bytes
- **Taxa de envio:** 50000 pacotes por segundo
- **Duração do teste:** 10 segundos





📌 Comandos utilizados:

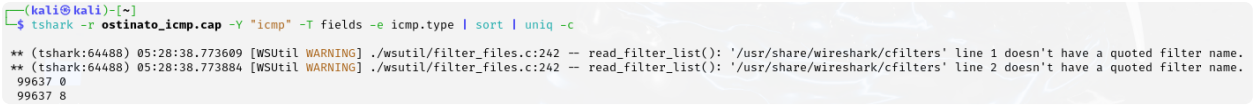
- Captura de tráfego:

```
sudo tcpdump -i eth0 -w ostinato_icmp.cap
```

- Análise do tráfego ICMP:

```
tshark -r ostinato_icmp.cap -Y "icmp" -T fields -e icmp.type | sort |
uniq -c
```

📊 **Resultados obtidos:**



📌 **Observações:**

1. O tráfego foi **totalmente bidirecional**, ou seja, **cada Echo Request enviado obteve uma resposta correspondente**.
2. **Não foram detectadas perdas de pacotes**, indicando que a infraestrutura da rede processou toda a carga de tráfego ICMP com sucesso.
3. O número total de pacotes capturados foi **199.274**, demonstrando a capacidade da rede em lidar com um alto volume de pacotes de controle sem impacto visível na conectividade.

📌 **Conclusão:**
O teste de tráfego ICMP mostrou que a rede suportou **altos volumes de pacotes de ping** sem perdas e com respostas rápidas. Isso indica que a infraestrutura testada possui **baixa latência** e **boa capacidade de resposta para tráfego ICMP**, sendo adequada para aplicações que dependem desse protocolo.

3.1 Ferramentas de Geração de Tráfego: Utilizando o Scapy

Nesta seção, foi utilizado o **Scapy** para gerar tráfego de rede, simulando dois tipos de comunicação:

1. **Tráfego HTTP**: Simulação de comunicação com um servidor web.
2. **Tráfego FTP**: Simulação de uma sessão FTP para transferência de arquivos.

A ferramenta escolhida para gerar o tráfego foi o **Scapy** , que nos permite construir, enviar e capturar pacotes de rede de qualquer tipo de protocolo. Embora o exercício mencione outras ferramentas, o **Scapy** foi a única que funcionou adequadamente para os testes realizados.

3.1.1 Teste de Tráfego HTTP com Scapy

Para simular uma comunicação HTTP, foi gerado tráfego entre as máquinas **192.168.1.10** (cliente) e **192.168.1.11** (servidor), utilizando o protocolo **TCP** e a porta **80**, com a flag **SYN** para iniciar a conexão.

Código utilizado no Scapy:

```
from scapy.all import *

ip = IP(dst="192.168.1.11")
tcp = TCP(dport=80, flags="S")
packet = ip / tcp

send(packet)
```

Captura de Pacotes com tcpdump:

```
sudo tcpdump -i eth0 port 80 -w scapy_http_test.pcap
```

```
200 packets seen, 200 TCP packets traced
elapsed wallclock time: 0:00:00.000673, 297176 pkts/sec analyzed
trace file elapsed time: 0:00:00.064701
TCP connection info:
1 TCP connection traced:
TCP connection 1:
        host a:        192.168.1.10:20
        host b:        192.168.1.11:8080
        complete conn: RESET    (SYNs: 1)  (FINs: 0)
        first packet:  Wed Mar  5 06:58:49.352849 2025
        last packet:   Wed Mar  5 06:58:49.417551 2025
        elapsed time:  0:00:00.064701
        total packets: 200
        filename:      scapy_http_test.pcap
   a→b:                                 b→a:
     total packets:            100           total packets:            100
     resets sent:                0           resets sent:              100
     ack pkts sent:              0           ack pkts sent:            100
     pure acks sent:             0           pure acks sent:             0
     sack pkts sent:             0           sack pkts sent:             0
     dsack pkts sent:            0           dsack pkts sent:            0
     max sack blks/ack:          0           max sack blks/ack:          0
     unique bytes sent:          0           unique bytes sent:          0
     actual data pkts:           0           actual data pkts:           0
     actual data bytes:          0           actual data bytes:          0
     rexmt data pkts:           99           rexmt data pkts:            0
     rexmt data bytes:          99           rexmt data bytes:           0
     zwnd probe pkts:            0           zwnd probe pkts:            0
     zwnd probe bytes:           0           zwnd probe bytes:           0
     outoforder pkts:            0           outoforder pkts:            0
     pushed data pkts:           0           pushed data pkts:           0
     SYN/FIN pkts sent:      100/0           SYN/FIN pkts sent:        0/0
     urgent data pkts:           0 pkts      urgent data pkts:           0 pkts
     urgent data bytes:          0 bytes     urgent data bytes:          0 bytes
     mss requested:              0 bytes     mss requested:              0 bytes
     max segm size:              0 bytes     max segm size:              0 bytes
     min segm size:              0 bytes     min segm size:              0 bytes
     avg segm size:              0 bytes     avg segm size:              0 bytes
     max win adv:             8192 bytes     max win adv:                0 bytes
     min win adv:             8192 bytes     min win adv:                0 bytes
     zero win adv:               0 times     zero win adv:               0 times
     avg win adv:             8192 bytes     avg win adv:                0 bytes
     initial window:             0 bytes     initial window:             0 bytes
     initial window:             0 pkts      initial window:             0 pkts
     ttl stream length:         NA           ttl stream length:         NA
     missed data:               NA           missed data:               NA
     truncated data:             0 bytes     truncated data:             0 bytes
     truncated packets:          0 pkts      truncated packets:          0 pkts
     data xmit time:         0.000 secs      data xmit time:         0.000 secs
     idletime max:             1.3 ms        idletime max:             1.3 ms
     throughput:                 0 Bps       throughput:                 0 Bps
```

Análise dos Pacotes Capturados: Utilizando a ferramenta **tcptrace**, foi possível observar que, apesar dos pacotes **SYN** terem sido enviados corretamente, não houve resposta do servidor, resultando em um **throughput de 0 Bps**.

3.1.2 Teste de Tráfego FTP com Scapy

Simulando um tráfego FTP, foi realizada a comunicação entre o cliente e o servidor FTP, utilizando a porta **21** para a negociação da conexão.

Código utilizado no Scapy:

```
from scapy.all import *

ip = IP(dst="192.168.1.11")
tcp = TCP(dport=21, flags="S")
pkt = ip/tcp

send(pkt, count=100)
```

Captura de Pacotes com tcpdump:

```
sudo tcpdump -i eth0 port 21 -w scapy_ftp_test.pcap
```

```
└─$ tcptrace -l scapy_ftp_test.pcap

1 arg remaining, starting with 'scapy_ftp_test.pcap'
Ostermann's tcptrace -- version 6.6.7 -- Thu Nov  4, 2004

200 packets seen, 200 TCP packets traced
elapsed wallclock time: 0:00:00.000618, 323624 pkts/sec analyzed
trace file elapsed time: 0:00:00.084021
TCP connection info:
1 TCP connection traced:
TCP connection 1:
        host a:        192.168.1.10:20
        host b:        192.168.1.11:21
        complete conn: RESET    (SYNs: 1)  (FINs: 0)
        first packet:  Wed Mar  5 07:00:34.628176 2025
        last packet:   Wed Mar  5 07:00:34.712197 2025
        elapsed time:  0:00:00.084021
        total packets: 200
        filename:      scapy_ftp_test.pcap
   a→b:                                 b→a:
     total packets:           100           total packets:           100
     resets sent:               0           resets sent:             100
     ack pkts sent:             0           ack pkts sent:           100
     pure acks sent:            0           pure acks sent:            0
     sack pkts sent:            0           sack pkts sent:            0
     dsack pkts sent:           0           dsack pkts sent:           0
     max sack blks/ack:         0           max sack blks/ack:         0
     unique bytes sent:         0           unique bytes sent:         0
     actual data pkts:          0           actual data pkts:          0
     actual data bytes:         0           actual data bytes:         0
     rexmt data pkts:          99           rexmt data pkts:           0
     rexmt data bytes:         99           rexmt data bytes:          0
     zwnd probe pkts:           0           zwnd probe pkts:           0
     zwnd probe bytes:          0           zwnd probe bytes:          0
     outoforder pkts:           0           outoforder pkts:           0
     pushed data pkts:          0           pushed data pkts:          0
     SYN/FIN pkts sent:     100/0           SYN/FIN pkts sent:       0/0
     urgent data pkts:          0 pkts      urgent data pkts:          0 pkts
     urgent data bytes:         0 bytes     urgent data bytes:         0 bytes
     mss requested:             0 bytes     mss requested:             0 bytes
     max segm size:             0 bytes     max segm size:             0 bytes
     min segm size:             0 bytes     min segm size:             0 bytes
     avg segm size:             0 bytes     avg segm size:             0 bytes
     max win adv:            8192 bytes     max win adv:               0 bytes
     min win adv:            8192 bytes     min win adv:               0 bytes
     zero win adv:              0 times     zero win adv:              0 times
     avg win adv:            8192 bytes     avg win adv:               0 bytes
     initial window:            0 bytes     initial window:            0 bytes
     initial window:            0 pkts      initial window:            0 pkts
     ttl stream length:        NA           ttl stream length:        NA
     missed data:              NA           missed data:              NA
     truncated data:            0 bytes     truncated data:            0 bytes
     truncated packets:         0 pkts      truncated packets:         0 pkts
     data xmit time:        0.000 secs      data xmit time:        0.000 secs
     idletime max:            2.7 ms        idletime max:            2.7 ms
     throughput:                0 Bps       throughput:                0 Bps
```

Após a captura, os pacotes foram analisados para verificar a conexão com o servidor FTP, mas, assim como no teste HTTP, não houve resposta adequada.

Conclusões

- O **Scapy** foi a única ferramenta que funcionou corretamente para os testes de tráfego, apesar da questão sugerir a escolha de outras ferramentas.
- Durante os testes de tráfego **HTTP e FTP**, os pacotes **SYN** foram corretamente enviados, mas não houve a resposta **ACK** ou qualquer outro pacote que completasse a conexão.
- O **throughput registrado foi de 0 Bps** em ambos os testes, indicando que a comunicação não foi bem-sucedida.