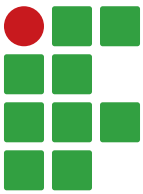


INSTITUTO FEDERAL
Santa Catarina

Câmpus
São José

Mini Projeto ADS

Avaliação de Desempenho do Protocolo TCP em condições de congestionamento: cubic x reno

Curso: Engenharia de Telecomunicações
Disciplina: ADS29009 - Avaliação de Desempenho de Sistemas
Professor: Eraldo Silveira e Silva

Aluno

João Pedro Menegali Salvan Bitencourt

20 de março de 2024

Sumário

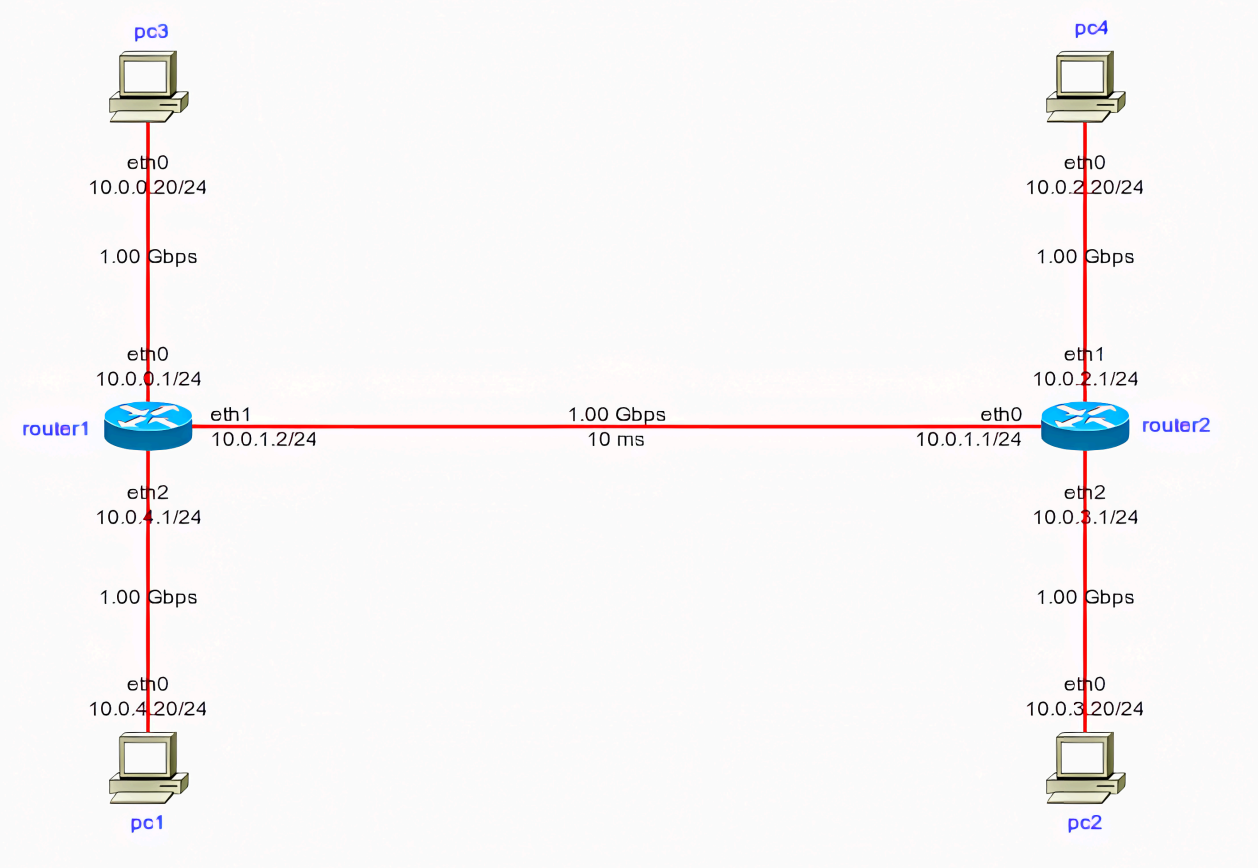
Introdução

O TCP é um protocolo de transporte que possui com principal característica a garantia de entrega através de mecanismos para controle de perda de pacotes. Através do uso da aplicação `iperf` e da suíte de simulação Imunes, foi realizado um experimento para medir a eficiência dos algorítmos para controle de congestionamento CUBIC e Reno

Cenário

O cenário foi montado na suíte de simulação Imunes e conta com dois roteadores ligados entre si. Além disso, cada um dos roteadores possui dois computadores conectados. Abaixo, está a ilustração do cenário montado:

Figura 1: Cenário montado na suíte de simulação Imunes.



Fonte: Autoria própria.

Parâmetros fixados

Como parâmetros fixados foram definidas as seguintes configurações para o cenário estabelecido:

- Todos os *links* são de 1 Gbps;
- Todos os *links* entre computadores e roteadores possuem BER 0;
- Utilizada a janela padão do TCP;
- Tráfego UDP de banda 1 Gbps entre PC3 e PC4.

Automatização de processos

Todo o experimento foi automatizado com o auxílio de *scripts* nas linguagens BASH e Python 3.

Captura das medições

Para a inicialização do cenário e captura dos dados provindos do Iperf, foram criados dois *shell scripts*. O motivo de haver mais um *script* para a mesma finalidade é a possibilidade de comparar diferentes métodos de testes com o Iperf. No primeiro *script* as ações são executadas na seguinte ordem:

1. Inicia a suíte de simulação Imunes com o cenário mostrado na Fig. 1;
2. Inicia, em segundo plano, o Iperf, em modo servidor, na máquina PC3, de modo a gerar tráfego UDP;
3. Inicia, em segundo plano, o Iperf, em modo cliente, na máquina PC4, de modo a receber tráfego UDP;
4. Inicia, em segundo plano, o Iperf, em modo servidor, na máquina PC1, de modo a gerar tráfego TCP. Os dados são gravados em um arquivo no formato CSV;
5. Executa um laço de oito repetições, cujo cada repetição conterá outro laço que faz o teste com cada um dos algorítmos de congestionamento. Dentro deste laço, há outro laço que fará o teste para a BER de 100 mil e 1 milhão. Adicionalmente, dentro deste laço, há um último laço que faz o teste variando o atraso em 10 ms e 100 ms. Os dados são gravados em arquivos individuais e também e um arquivo geral que possui a identificação de cada teste;
6. Encerra o cenário de simulação do Imues;
7. Chama um *script* em Python que gera os gráficos de barras com os dados coletados.
8. Caso haja mais um valor para a largura de banda do tráfego de fundo, todo o processo é repetido de com o próximo valor.

Já o segundo *script* interrompe todas as instâncias do Iperf em cada computador a cada troca de característica da conexão entre os roteadores. Dessa forma, a rotina de execução ficou a seguinte forma:

1. Inicia a suíte de simulação Imunes com o cenário mostrado na Fig. 1;
2. Ajusta a taxa de erro de bit para a conexão entre os roteadores;
3. Ajuste no atraso da conexão entre os roteadores;
4. Inicia o Iperf em modo servidor UDP na máquina PC3 em segundo plano;
5. Inicia o Iperf em modo cliente UDP na máquina PC4 em segundo plano;
6. Inicia o Iperf em modo servidor TCP na máquina PC1 em segundo plano;
7. Executa o Iperf em modo cliente TCP na máquina PC2;
8. Encerra o processo do Iperf em cada computador;
9. Repete os passos acima para cada uma das conbinações de configuração;
10. Repete todas as combinações oito vezes, totalizando 64 execuções;
11. Encerra o cenário de simulação do Imues;
12. Chama o *script* em Python que gera os gráficos de barras com os dados coletados;
13. Caso haja mais um valor para a largura de banda do tráfego de fundo, todo o processo é repetido de com o próximo valor.

Ambos os *scripts* recebem como primeiro argumento de linha de comando o nome do arquivo de cenário do Imunes (*arquivo.imn*). Ambos criam diretórios para armazenar os arquivo CVS gerados. Na rotina que reinicia todos os processos do Iperf, o nome do diretório criado tem o formato “iperf-csv-BW-para-e-retoma”, cujo “BW” refere-se à largura de banda do tráfego UDP de fundo. Já a outra rotina cria o diretório que armazena os arquivos CVS com o seguinte padrão de nomenclatura: “iperf-csv-BW”. Abaixo é demostrado o uso de cada rotina:

```
./simulacao cenario.imn
./simulacao-para-e-retoma cenario.imn
```

Os dois *scripts* completos estão disponíveis no final deste documento.

Arquivo CSV customizado

Para facilitar o tratamento dos dados, foram adicionados campos extras no arquivo CSV gerado pelo Iperf. Os campos adicionados foram:

- Letra identificadora;
- Número da repetição;
- Largura de banda do tráfego de fundo;
- Nome do algorítmo de congestionamento;
- Valor da BER;
- Valor do atraso.

Dessa forma, o arquivo CSV gerado pelo Iperf ficou com a seguinte estrutura:

```
letra identificadora,repetição,largura de banda,algorítmo de
congestionamento,BER,atraso,timestamp,ip de origem,porta de origem,ip de destino,porta de
destino,ID,tempo de execução,bytes transferidos,bits por segundo
```

Uma amostra é exemplificada abaixo:

```
A,1,1G,cubic,100000,10000,20240320234903,10.0.3.20,41460,10.0.4.20,5001,3,
0.0-12.4,8519680,5499338
```

Geração dos gráficos de barras

Para a geração dos gráficos de barras, foi criado um *script* em Python 3 que lê os arquivos CSV gerados pelo Iperf e gera os gráficos de barras. Para tal, foi utilizada a biblioteca `matplotlib`. A rotina recebe como primeiro argumento de linha de comando o nome do arquivo CSV contendo todas as medições e, opcionalmente, um segundo argumento de linha de comando contendo um sufixo para o nome do arquivo de saída, que é uma imagem no formato SVG.

Abaixo, é demostrado o uso do *script*:

```
1 ./computar.py iperf-csv-500M/dados-cliente.csv
```

ou com sufixo:

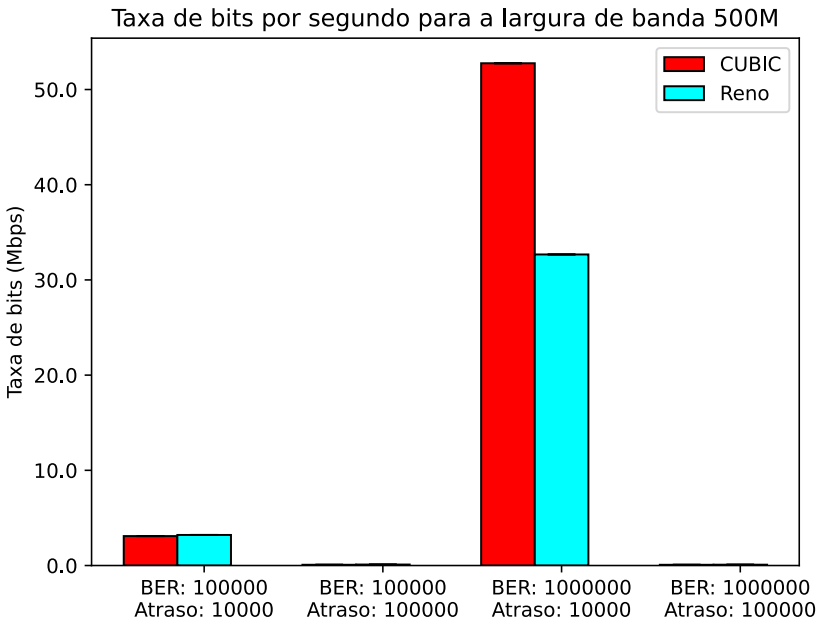
```
1 ./computar.py iperf-csv-500M/dados-cliente.csv -modo-para-e-retoma
```

A rotina calcula a média da taxa de bits por segundo e utiliza a letra identificadora para filtrar quais linhas participarão do cálculo, de forma que somente linhas com a mesma letra tenham os valores da coluna correspondente somados. O campo com o algorítmo é utilizado para separar quais valores serão usados em cada barra. Já os campos com a largura de banda do tráfego UDP, a BER e o atraso, são utilizados para escrever os textos na figura do gráfico.

Análises

Para obter consistência nos resultados, foram executados diversas rodadas de testes, totalizando 64 rodadas para cada figura gerada. Na Fig. 2, é mostrado o gráfico comparando a largura de banda entre os algorítmos de congestionamento CUBIC e Reno para um tráfego UDP de fundo com banda de 500 Mbps:

Figura 2: Comparação da largura de banda entre os algorítmos de congestionamento CUBIC e Reno para um tráfego UDP de fundo com banda de 500 Mbps.



Fonte: Autoria própria.

Na Fig. 2 é possível notar a maior largura de banda ficou para o cenário com BER de 1 milhão e atraso de 10 ms, cujo algorítmo CUBIC teve a maior largura de banda, 52,75782238 Mbps.

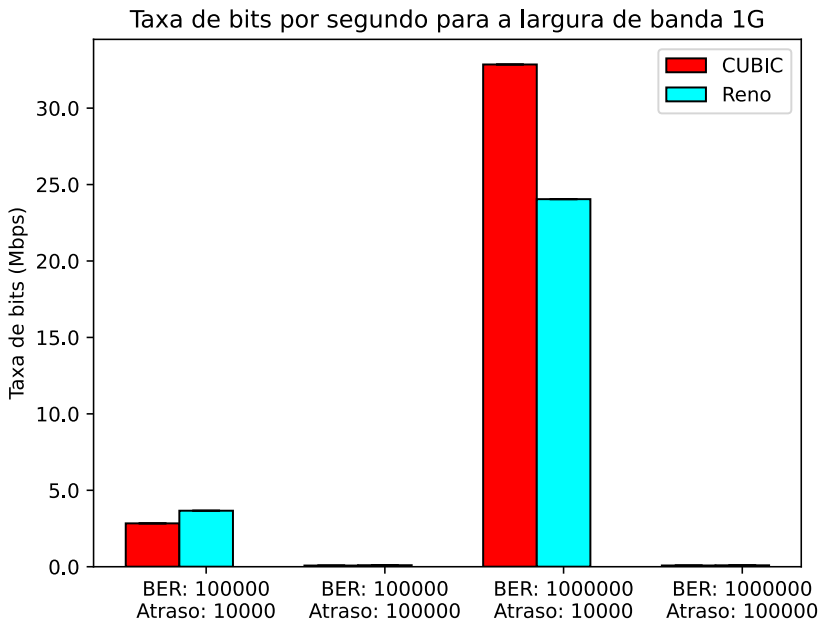
A segunda maior largura de banda foi no cenário com BER de 100 mil e atraso de 10 ms, cujo algorítmo Reno teve a maior largura de banda, 3,21028525 Mbps.

Já para os cenários com BER de 100 mil e atraso de 100 ms e BER de 1 milhão e atraso de 100 ms, a largura de banda para ambos os algorítmos ficou extremamente baixa. Abaixo, há um quadro comparativo com todas as medições:

Largura de banda UDP	Algorítmo	BER	Atraso	Média de bis por segundo
500 Mbps	cubic	100 mil	10 ms	3,09190688 Mbps
500 Mbps	reno	100 mil	10 ms	3,21028525 Mbps
500 Mbps	cubic	100 mil	100 ms	77,85288 kbps
500 Mbps	reno	100 mil	100 ms	102,88762 kbps
500 Mbps	cubic	1 milhão	10 ms	52,75782238 Mbps
500 Mbps	reno	1 milhão	10 ms	3,21028525 Mbps
500 Mbps	cubic	1 milhão	100 ms	75,96112 kbps
500 Mbps	reno	1 milhão	100 ms	86,65150 kbps

Na Fig. 3, é mostrado o gráfico comparando a largura de banda entre os algorítmos de congestionamento CUBIC e Reno quando o tráfego UDP de fundo possui banda de 1 Gbps:

Figura 3: Comparação da largura de banda entre os algorítmos de congestionamento CUBIC e Reno para um tráfego UDP de fundo com banda de 1 Gbps.



Fonte: Autoria própria.

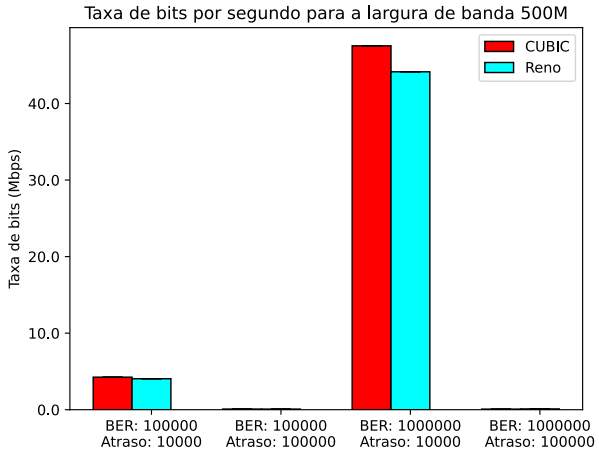
Assim como na medição anterior, o cenário com BER de 1 milhão e atraso de 10 ms teve a maior largura de banda, 32,85510138 Mbps, para o algorítmo CUBIC. Já para o algorítmo Reno, a maior largura de banda foi no cenário com BER de 100 mil e atraso de 10 ms, 3,66415925 Mbps.

Novamente, os piores cenários foram os com BER de 100 mil e atraso de 100 ms e BER de 1 milhão e atraso de 100 ms. Abaixo, há um quadro comparativo com todas as medições:

Largura de banda UDP	Algorítmo	BER	Atraso	Média de bis por segundo
1 Gbps	cubic	100 mil	10 ms	2,83648188 Mbps
1 Gbps	reno	100 mil	10 ms	3,66415925 Mbps
1 Gbps	cubic	100 mil	100 ms	77,61912 kbps
1 Gbps	reno	100 mil	100 ms	99,05075 kbps
1 Gbps	cubic	1 milhão	10 ms	32,85510138 Mbps
1 Gbps	reno	1 milhão	10 ms	24,04517475 Mbps
1 Gbps	cubic	1 milhão	100 ms	81,02350 kbps
1 Gbps	reno	1 milhão	100 ms	92,36812 kbps

As figuras à seguir mostram os gráficos gerados a partir dos dados coletados na rotina que parava todas as instâncias do Iperf a cada troca de característica da conexão entre os roteadores. Na Fig. 4, é mostrada a comparação das larguras de banda alcançadas por cada algorítmo de controle de congestionamento quando o tráfego de fundo UDP possui taxa de 500 Mbps:

Figura 4: Comparação da largura de banda entre os algorítmos de congestionamento CUBIC e Reno para um tráfego UDP de fundo com banda de 500 Mbps.

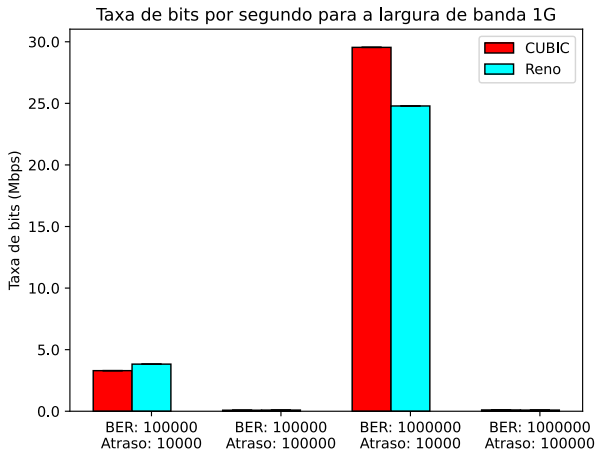


Fonte: Autoria própria.

Na Fig. 4, assim como nas anteriores, os melhores cenários são os que possuem BER de 1 milhão e atraso de 10 ms e BER de 100 mil e atraso de 10 ms. Dentro desses cenários, o algoritmo CUBIC foi que teve o maior valor de largura de banda. Abaixo, há um quadro comparativo com todas as medições:

Largura de banda UDP	Algorítmo	BER	Atraso	Média de bis por segundo
500 Mbps	cubic	100 mil	10 ms	4,251236 Mbps
500 Mbps	reno	100 mil	10 ms	4,04704238 Mbps
500 Mbps	cubic	100 mil	100 ms	82,71788 kbps
500 Mbps	reno	100 mil	100 ms	79,13375 kbps
500 Mbps	cubic	1 milhão	10 ms	47,53579338 Mbps
500 Mbps	reno	1 milhão	10 ms	44,14313625 Mbps
500 Mbps	cubic	1 milhão	100 ms	75,62675 kbps
500 Mbps	reno	1 milhão	100 ms	102,41262 kbps

Figura 4: Comparação da largura de banda entre os algorítmos de congestionamento CUBIC e Reno para um tráfego UDP de fundo com banda de 1 Gbps.



Fonte: Autoria própria.

Por fim, na Fig. 5, repetiram-se os cenários com melhor desempenho. Na rodada com BER em 100 mil e atraso em 10 ms, o algoritmo CUBIC teve o maior resultado: 3,296063 Mbps. O mesmo se repetiu na rodada com BER em 1 milhão e atraso em 10 ms, com o algoritmo CUBIC atingindo 29,54644012 Mbps de largura de banda. Abaixo, há um quadro comparativo com todas as medições:

Largura de banda UDP	Algorítmo	BER	Atraso	Média de bis por segundo
1 Gbps	cubic	100 mil	10 ms	3,296063 Mbps
1 Gbps	reno	100 mil	10 ms	3,82610138 Mbps
1 Gbps	cubic	100 mil	100 ms	83,40212 kbps
1 Gbps	reno	100 mil	100 ms	93,62288 kbps
1 Gbps	cubic	1 milhão	10 ms	29,54644012 Mbps
1 Gbps	reno	1 milhão	10 ms	24,78752425 Mbps
1 Gbps	cubic	1 milhão	100 ms	105,08588 kbps
1 Gbps	reno	1 milhão	100 ms	93,51562 kbps

Métricas

Para a análise dos resultados, foi realizada a média da taxa de bits por segundo de todas as rodadas para cenário e algorítmo.

Fatores e níveis

Como fatores, há os dois algorítmos CUBIC e Reno, as diferentes taxas de erro de bit e os atrasos. Os níveis desses fatores são a variação de cada um desses elementos entre si, que compõem as rodadas dos teste.

Conclusão

A análise dos dados mostrou que os cenários cujo atraso é de 10 ms teve um desempenho muito melhor se comparado aos cenários com o atraso de 100 ms. Além disso, os algoritmos obtiveram melhor desempenho quando a BER foi de 1 milhão, ou seja, quando a probabilidade de erro é de uma em um milhão, o resultado foi melhor.

Outro ponto interessante foi a mudança na forma como os testes foram realizados, que mostraram uma pequena diferença nos valores finais, porém com os resultados gerais sendo coerentes entre todos os gráficos.

Anexos

Os *scripts* utilizados para a realização do experimento estão disponíveis abaixo:

- *Shell script* `simulacao`:

```
#!/bin/bash

algoritmo=('cubic' 'reno')
BER=('100000' '1000000')
e2e_delay=('10000' '100000')
largura_de_banda=('500M' '1G')
repeticao=8

id_experimento='i1234'
arquivo_de_projeto=$1
prefixo_dir_csv="iperf-csv"

# Início da simulação completa
for BW in "${largura_de_banda[@]}"; do
    diretorio_csv="$prefixo_dir_csv-$BW"
    mkdir -p "$diretorio_csv"

    echo -e "\nIniciando a simulação para o experimento \"$id_experimento\" e o arquivo
de projeto \"$arquivo_de_projeto\""
    echo -e " - Largura de banda do tráfego UDP: \"$BW\""
    cmd_imunes_iniciar="imunes -b -e $id_experimento $arquivo_de_projeto"
    echo -e " - Executando o comando: \"$cmd_imunes_iniciar\"\n"
    bash -c "$cmd_imunes_iniciar"

    # Criando tráfego UDP em segundo plano com o IPERF entre PC3 e PC4
    echo -e "\nIniciando o servidor IPERF em modo UDP no \"PC3\""
    cmd_iperf_server_UDP="himage pc3@$id_experimento iperf -s -u"
    echo -e " - Executando o comando: \"$cmd_iperf_server_UDP\"\n"
    bash -c "$cmd_iperf_server_UDP" &

    echo -e "\nIniciando o cliente IPERF em modo UDP no \"PC4\""
    cmd_iperf_client_UDP="himage pc4@$id_experimento iperf -u -t 7200 -b $BW -c 10.0.0.20"
    echo -e " - Executando o comando: \"$cmd_iperf_client_UDP\"\n"
    bash -c "$cmd_iperf_client_UDP" &

    # Iniciando o IPERF no servidor
    echo -e "\nIniciando o IPERF no servidor \"PC1\""
    cmd_iperf_server="himage pc1@$id_experimento iperf -s -y C"
    echo -e " - Executando o comando: \"$cmd_iperf_server\"\n"
    bash -c "$cmd_iperf_server" >> $diretorio_csv/dados-servidor.csv &

    for ((i=1; i<=$repeticao; i++)); do
        letra="A"
        for proto in "${algoritmo[@]}"; do
            for ber in "${BER[@]}"; do
                for e2e in "${e2e_delay[@]}"; do
                    echo -e "\nRepetição \"$i\" para o algoritmo \"$proto\", BER \"$ber\"
e atraso fim-a-fim \"$e2e\""
                    cmd_muda_ber_router="vlink -BER $ber router1:router2@$id_experimento"
                    echo -e " - Executando o comando: \"$cmd_muda_ber_router\""
                    bash -c "$cmd_muda_ber_router"

                cmd_muda_delay_router="vlink -dly $e2e router1:router2@$id_experimento"
                    echo -e " - Executando o comando: \"$cmd_muda_delay_router\""
                    bash -c "$cmd_muda_delay_router"

                    # Iniciando o IPERF no cliente
                    echo -e "\nIniciando o IPERF no cliente \"PC2\""
                     cmd_iperf_client="himage pc2@$id_experimento iperf -y C -Z $proto -
c 10.0.4.20"
                    echo -e " - Executando o comando: \"$cmd_iperf_client\"\n"
```

```
                    while (true); do
                        resultado_iperf="$(bash -c "$cmd_iperf_client")"
                         if [[ ! "$resultado_iperf" =~ "connect failed: Connection timed
out" ]]; then
                            break
                        fi
                        echo "Repetindo teste..."
                        sleep 1
                    done

                       printf "$letra,$i,$BW,$proto,$ber,$e2e," >> $diretorio_csv/dados-
cliente.csv
                    echo "$resultado_iperf" >> $diretorio_csv/dados-cliente.csv
                  echo "$resultado_iperf" >> $diretorio_csv/dados-$proto-BER$ber-r$i.csv

                    letra_ascii=$(printf "%d" "'$letra'")
                    letra_ascii=$((letra_ascii+1))
                    letra=$(printf "\x$(printf %x $letra_ascii)")
                done
            done
        done
    done

    # Finalizaçao da simulação
    echo -e "\nParando o experimento $id_experimento"
    cmd_imunes_parar="imunes -b -e $id_experimento"
    echo -e " - Executando o comando: \"$cmd_imunes_parar\"\n"
    bash -c "$cmd_imunes_parar"

    # Chamando a rotina em python que gera os gráficos
    bash -c "./computar.py $diretorio_csv/dados-cliente.csv"
done
```

- *Shell script* `simulacao-para-e-retoma`:

```
#!/bin/bash

iniciar_trafego_udp(){
    echo -e "\nIniciando o servidor IPERF em modo UDP no \"PC3\""
    cmd_iperf_server_UDP="himage pc3@$id_experimento iperf -s -u"
    echo -e " - Executando o comando: \"$cmd_iperf_server_UDP\"\n"
    bash -c "$cmd_iperf_server_UDP" > /dev/null 2>&1 &

    echo -e "\nIniciando o cliente IPERF em modo UDP no \"PC4\""
   cmd_iperf_client_UDP="himage pc4@$id_experimento iperf -u -t 7200 -b $BW -c 10.0.0.20"
    echo -e " - Executando o comando: \"$cmd_iperf_client_UDP\"\n"
    bash -c "$cmd_iperf_client_UDP" > /dev/null 2>&1 &
}

parar_trafego_udp(){
    echo -e "\nParando o servidor IPERF em modo UDP no \"PC3\""
    cmd_iperf_server_UDP="himage pc3@$id_experimento pkill -f iperf -9"
    echo -e " - Executando o comando: \"$cmd_iperf_server_UDP\"\n"
    bash -c "$cmd_iperf_server_UDP" > /dev/null 2>&1

    echo -e "\nParando o cliente IPERF em modo UDP no \"PC4\""
    cmd_iperf_client_UDP="himage pc4@$id_experimento pkill -f iperf -9"
    echo -e " - Executando o comando: \"$cmd_iperf_client_UDP\"\n"
    bash -c "$cmd_iperf_client_UDP" > /dev/null 2>&1
}

iniciar_servidor_iperf_tcp(){
    echo -e "\nIniciando o IPERF no servidor \"PC1\""
    cmd_iperf_server="himage pc1@$id_experimento iperf -s -y C"
    echo -e " - Executando o comando: \"$cmd_iperf_server\"\n"
```

```
    bash -c "$cmd_iperf_server" >> $diretorio_csv/dados-servidor.csv &
}

iniciar_cliente_iperf_tcp(){
    echo -e "\nIniciando o IPERF no cliente \"PC2\""
    cmd_iperf_client="himage pc2@$id_experimento iperf -y C -Z $proto -c 10.0.4.20"
    echo -e " - Executando o comando: \"$cmd_iperf_client\"\n"

    while (true); do
        resultado_iperf="$(bash -c "$cmd_iperf_client")"
        if [[ ! "$resultado_iperf" =~ "connect failed: Connection timed out" ]]; then
            break
        fi
        echo "Repetindo teste..."
        sleep 1
    done
}

parar_servidor_iperf_tcp(){
    echo -e "\nParando o IPERF no servidor \"PC1\""
    cmd_iperf_server="himage pc1@$id_experimento pkill -f iperf -9"
    echo -e " - Executando o comando: \"$cmd_iperf_server\"\n"
    bash -c "$cmd_iperf_server" > /dev/null 2>&1
}

algoritmo=('cubic' 'reno')
BER=('100000' '1000000')
e2e_delay=('10000' '100000')
largura_de_banda=('500M' '1G')
repeticao=8

id_experimento='i1234'
arquivo_de_projeto=$1
prefixo_dir_csv="iperf-csv"

# Início da simulação completa
for BW in "${largura_de_banda[@]}"; do
    diretorio_csv="$prefixo_dir_csv-$BW-para-e-retoma"
    mkdir -p "$diretorio_csv"

     echo -e "\nIniciando a simulação para o experimento \"$id_experimento\" e o arquivo
de projeto \"$arquivo_de_projeto\""
    echo -e " - Largura de banda do tráfego UDP: \"$BW\""
    cmd_imunes_iniciar="imunes -b -e $id_experimento $arquivo_de_projeto"
    echo -e " - Executando o comando: \"$cmd_imunes_iniciar\"\n"
    bash -c "$cmd_imunes_iniciar"

    for ((i=1; i<=$repeticao; i++)); do
        letra="A"
        for proto in "${algoritmo[@]}"; do
            for ber in "${BER[@]}"; do
                for e2e in "${e2e_delay[@]}"; do
                   echo -e "\nRepetição \"$i\" para o algoritmo \"$proto\", BER \"$ber\"
e atraso fim-a-fim \"$e2e\""
                    cmd_muda_ber_router="vlink -BER $ber router1:router2@$id_experimento"
                    echo -e " - Executando o comando: \"$cmd_muda_ber_router\""
                    bash -c "$cmd_muda_ber_router"

                  cmd_muda_delay_router="vlink -dly $e2e router1:router2@$id_experimento"
                    echo -e " - Executando o comando: \"$cmd_muda_delay_router\""
                    bash -c "$cmd_muda_delay_router"

                    # Criando tráfego UDP em segundo plano com o IPERF entre PC3 e PC4
                    iniciar_trafego_udp

                    # Iniciando o IPERF no servidor
                    iniciar_servidor_iperf_tcp
```

```
                    sleep 5

                    # Iniciando o IPERF no cliente
                    iniciar_cliente_iperf_tcp

                    # Parando tráfego UDP e servidor IPERF TCP
                    parar_trafego_udp
                    parar_servidor_iperf_tcp

                       printf "$letra,$i,$BW,$proto,$ber,$e2e," >> $diretorio_csv/dados-
cliente.csv
                    echo "$resultado_iperf" >> $diretorio_csv/dados-cliente.csv
                  echo "$resultado_iperf" >> $diretorio_csv/dados-$proto-BER$ber-r$i.csv

                    letra_ascii=$(printf "%d" "'$letra'")
                    letra_ascii=$((letra_ascii+1))
                    letra=$(printf "\x$(printf %x $letra_ascii)")
                done
            done
        done
    done

    # Finalizaçao da simulação
    echo -e "\nParando o experimento $id_experimento"
    cmd_imunes_parar="imunes -b -e $id_experimento"
    echo -e " - Executando o comando: \"$cmd_imunes_parar\"\n"
    bash -c "$cmd_imunes_parar"

    # Chamando a rotina em python que gera os gráficos
    bash -c "./computar.py $diretorio_csv/dados-cliente.csv -modo-para-e-retoma"
done
```

- *Script* `computar.py`:

```python
#!/usr/bin/env python3

import numpy as np
import matplotlib.pyplot as plt
import sys
import csv
from collections import defaultdict
from scipy.stats import t

arquivo_csv = sys.argv[1]

if len(sys.argv) > 2:
    sufixo_arquivo = sys.argv[2]
else:
    sufixo_arquivo = ""

medicoes = []

with open(arquivo_csv, 'r') as linha:
    leitura = csv.reader(linha)
    for linha in leitura:
        medicoes.append(linha)

dados_por_letra = defaultdict(list)

for linha in medicoes:
    letra = linha[0]
    dados_por_letra[letra].append(linha)
```

```
media_de_medicoes = {}
intervalo_cubic = []
intervalo_reno = []

for letra, linhas in dados_por_letra.items():
    bytes_transferidos = [float(linha[13]) for linha in linhas]
    taxa_de_bits = [float(linha[14]) for linha in linhas]
    algoritmo = [linha[3] for linha in linhas][0]
    ber = [linha[4] for linha in linhas][0]
    atraso = [linha[5] for linha in linhas][0]

    # Média dos dados transferidos e taxa de bits
      media_dados_transferidos = sum(bytes_transferidos) / len(bytes_transferidos) if
bytes_transferidos else 0
    media_taxa_de_bits = sum(taxa_de_bits) / len(taxa_de_bits) if taxa_de_bits else 0
       media_de_medicoes[letra] = (algoritmo, ber, atraso, media_dados_transferidos,
media_taxa_de_bits)

    # Intervalo de confiança de 99% (t-student)
    # alfa = 1 - 0.995 = 0.005
    # n - 1 = 8 - 1 = 7
    # Valor na tabela = 3.499
    n = len(taxa_de_bits)
    t_student = t.ppf(0.995, n - 1)
    desvio_padrao = np.std(taxa_de_bits)
    erro_inferior = media_taxa_de_bits - (t_student * desvio_padrao / np.sqrt(n))
    erro_superior = media_taxa_de_bits + (t_student * desvio_padrao / np.sqrt(n))
    tamanho_intervalo = (erro_superior * 0.1) - (erro_inferior * 0.1)

      print(f"erro_inferior = {media_taxa_de_bits} - ({t_student} * {desvio_padrao} /
sqrt({n})) = {erro_inferior}")
      print(f"erro_superior = {media_taxa_de_bits} + ({t_student} * {desvio_padrao} /
sqrt({n})) = {erro_superior}")
    print()

    if algoritmo == "cubic":
        intervalo_cubic.append(tamanho_intervalo)
    elif algoritmo == "reno":
        intervalo_reno.append(tamanho_intervalo)

barra_cubic = []
barra_reno = []
descricao = []
largura_de_banda = [linha[2] for linha in linhas][0]

print("Sumarização dos dados medidos:")
for letra, parametro in media_de_medicoes.items():
    algoritmo = parametro[0]
    ber = parametro[1]
    atraso = parametro[2]
    bits_por_segundo = parametro[4]

    if algoritmo == "cubic":
        barra_cubic.append(bits_por_segundo)
    elif algoritmo == "reno":
        barra_reno.append(bits_por_segundo)

    descricao.append("BER: " + ber + "\n" + "Atraso: " + atraso)

    # print(f"Letra: {letra}")
    print(f"Largura de banda do tráfego UDP de fundo: {largura_de_banda}")
    print(f"Algoritmo: {algoritmo}")
    print(f"BER: {ber}")
    print(f"Atraso: {atraso}")
    print(f"Média de bits por segundo: {bits_por_segundo:.2f}")
```

```
    print()

descricao = descricao[:len(descricao)//2]
largura = 0.3

# Erros
if len(intervalo_cubic) == 0:
    err_cubic = [0, 0, 0, 0]
else:
    err_cubic = intervalo_cubic

if len(intervalo_reno) == 0:
    err_reno = [0, 0, 0, 0]
else:
    err_reno = intervalo_reno

# Posições das barras
pos_cubic = np.arange(len(barra_cubic))
pos_reno = [x + largura for x in pos_cubic]

# Plotando gráficos de barras
plt.bar(pos_cubic, barra_cubic, width = largura, color = 'red', edgecolor = 'black',
yerr=err_cubic, capsize=7, label='CUBIC')
plt.bar(pos_reno, barra_reno, width = largura, color = 'cyan', edgecolor = 'black',
yerr=err_reno, capsize=7, label='Reno')

plt.xticks([r + largura for r in range(len(barra_cubic))], descricao)
plt.ylabel('Taxa de bits (Mbps)')
plt.title(f'Taxa de bits por segundo para a largura de banda {largura_de_banda}')
plt.gca().yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f'{x/1e6:.1f}'))
plt.legend()
plt.savefig(f'simulado-banda-{largura_de_banda}{sufixo_arquivo}.svg', format='svg')
# plt.show()
```