
# MiniProjeto - Medição Ativa em Redes com o Iperf

Avaliação de Desempenho de Sistemas

**Arthur Cadore Matuella Barcella**
**Deivid Fortunato Frederico**

11 de Abril de 2025

Engenharia de Telecomunicações - IFSC-SJ

# Sumário

# 1. Introdução

Este relatório tem como objetivo apresentar o desenvolvimento e os resultados obtidos no mini projeto de medição ativa em redes utilizando o Iperf.

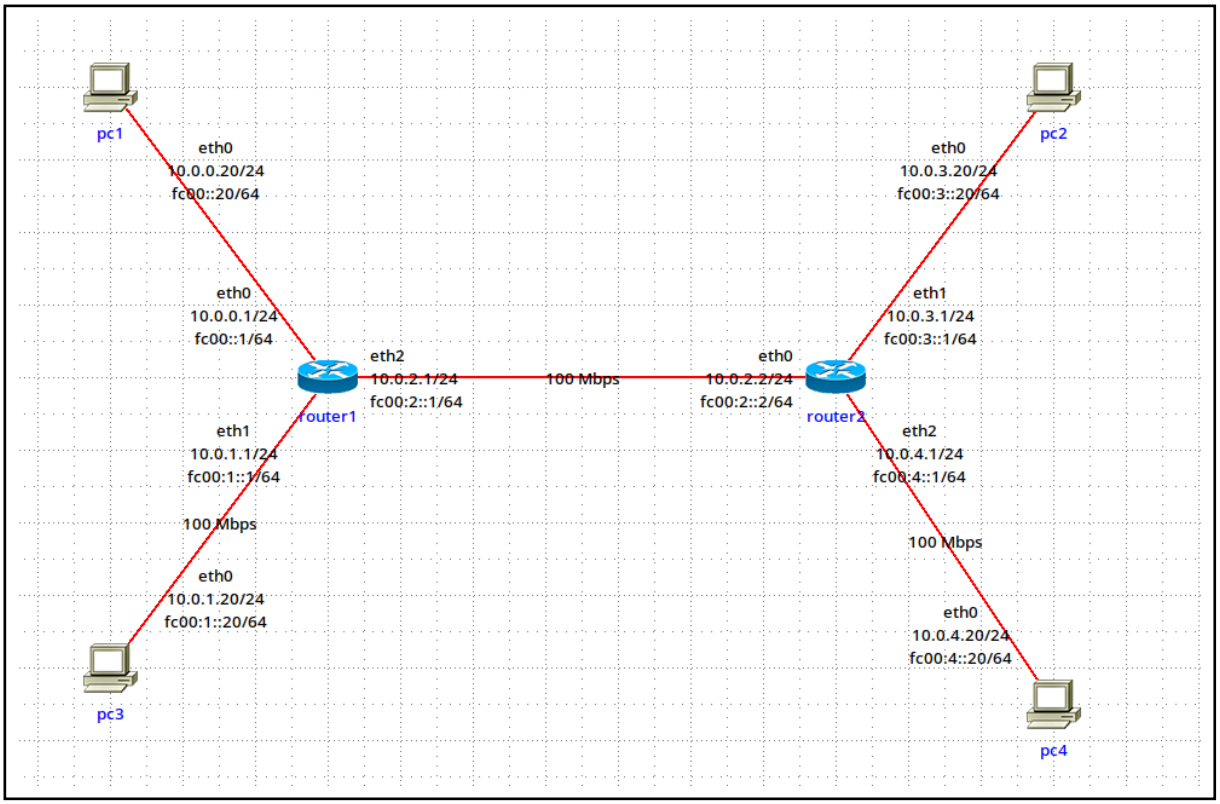
## 1.1. Especificação de cenário

- Uso da ferramenta iperf;
- Uso da ferramenta imunes;
- Automação de tarefas via scritpt shell e comandos do simulador imunes;
- Conceitos de intervalo de confiança e
- Projeto fatorial 2kr e boas práticas de projeto de experimentos.

## 1.2. Topologia:

A topologia utilizada no experimento foi a seguinte:

Figura 1: Elaborada pelo Autor




## 1.3. Objetivo

Avaliar, por meio de medição ativa com iperf, como a vazão de uma conexão TCP é afetada por dois fatores:

- O número de fluxos TCP paralelos (parâmetro -P no cliente);
- O retardo de rede, emulado com o comando vlink.

Além disso, verificar se há interação entre os fatores na determinação da vazão.

#### 1.3.1. Fatores e níveis

Os fatores e níveis utilizados no experimento foram:

| Fator | Nivel Baixo | Nivel Alto |
|---|---|---|
| A: Paralelismo TCP (-P) | 1 fluxo | 4 fluxo |
| B: Retardo de rede (RTT) | 10 ms (ida ⟹ RTT20ms) | 100 ms (ida ⟹ RTT200ms) |

#### 1.3.2. Metrica avaliada

- Vazão total (em Mbps), somando todos os fluxos, medida no cliente ao final da execução.

#### 1.3.3. Execuções:

Os ciclos de execução utilizados foram os seguintes:

| Execução | Fluxos (-P) | Delay (ms) | Repetições |
|---|---|---|---|
| 1 | 1 | 10 | 8 |
| 2 | 1 | 100 | 8 |
| 3 | 4 | 10 | 8 |
| 4 | 4 | 100 | 8 |

# 2. Desenvolvimento

Abaixo está a descrição do desenvolvimento do experimento, incluindo a configuração do ambiente, o script utilizado para automatizar a execução e o Makefile utilizado para facilitar a execução de múltiplas iterações.

A implementação do experimento pode ser visualizada no `link` https://github.com/deividffrederico/ads-av2.

## 2.1. Script Shell

O script shell desenvolvido para automatizar a execução do experimento foi o seguinte. O script é executado em três passsos:

- Primeiramente, ele inicia o simulador IMUNES em background, utilizando o cenário desejado.
- Em seguida, configura os links de acordo com os parâmetros desejados, utilizando o comando vlink.
- Por fim, inicia o servidor iperf nas máquinas desejadas e executa o cliente iperf, coletando os resultados.

```
#!/bin/bash

TOPOLOGY_FILE=scenario.imn
BANDWIDTH=100000000
# SCENARIO_ID=i2002
# DELAY=10000
# FLUXES=1

# Create a log directory if it doesn't exist
mkdir -p ./log

echo "============================================================"
echo "Starting IMUNES simulation with scenario ID: $SCENARIO_ID"

# Run IMUNES in background and redirect all output to log
sudo imunes -b -e $SCENARIO_ID $TOPOLOGY_FILE > /dev/null
sleep 2

echo "============================================================"
echo "Simulation started, applying commands..."

# Link configurations
sudo vlink -bw $BANDWIDTH -dly $DELAY router1:pc1@$SCENARIO_ID > /dev/
null
sudo vlink -bw $BANDWIDTH -dly $DELAY router1:pc3@$SCENARIO_ID > /dev/
null
sudo vlink -bw $BANDWIDTH -dly $DELAY router2:pc2@$SCENARIO_ID > /dev/
null
sudo vlink -bw $BANDWIDTH -dly $DELAY router2:pc4@$SCENARIO_ID > /dev/
null
sudo vlink -bw $BANDWIDTH -dly $DELAY router2:router1@$SCENARIO_ID > /
dev/null
sleep 2

# Check status (optional)
sudo vlink -s router1:pc1@$SCENARIO_ID
sudo vlink -s router1:pc3@$SCENARIO_ID
sudo vlink -s router2:pc2@$SCENARIO_ID
sudo vlink -s router2:pc4@$SCENARIO_ID
sudo vlink -s router2:router1@$SCENARIO_ID
sleep 2

# Start iperf servers
sudo himage pc2@$SCENARIO_ID iperf -s &> /dev/null &
sudo himage pc4@$SCENARIO_ID iperf -s &> /dev/null &

# Generate background UDP traffic (mute output)
sudo himage pc1@$SCENARIO_ID iperf -c 10.0.3.20 -u -t 100000 -b 10M &> /
dev/null &
sleep 2


# Run TCP test (this is the one you want to see)
echo "============================================================"
echo "Running TCP test between PC3 and PC4..."
sudo himage pc3@$SCENARIO_ID iperf -c 10.0.4.20 -n 100M -P $FLUXES -i 1

# Stop simulation
```

```
53 echo "============================================================"
54 echo "Stopping IMUNES simulation with scenario ID: $SCENARIO_ID"
55 sudo imunes -b -e $SCENARIO_ID > /dev/null
56 echo "============================================================"
57 echo "Simulation stopped"
```

Para executa-lo, 3 variáveis são necessárias junto com o script:

- SCENARIO_ID: ID do cenário a ser utilizado.
- DELAY: Retardo de rede a ser utilizado.
- FLUXES: Número de fluxos TCP a serem utilizados.

Essas variáveis podem ser configuradas previamente no OS, ou então descomentar as linhas 3, 4 e 5 do script e definir os valores desejados.

## 2.2. Script Makefile

O script make utilizado para automatizar a execução do experimento foi o seguinte. O objetivo da utilização de um Makefile é passar as variáveis definidas no script shell como parâmetro e então executar o script shell com os parâmetros desejados.

```
1  # Caminho do script shell
2  SCRIPT=./run.sh
3
4  # Caminho do diretório de log
5  LOGDIR=./log
6
7  # Targets
8  all: exec1 exec2 exec3 exec4
9
10 exec1:
11   @echo "ETAPA1: Executando com DELAY=10000 FLUXES=1" | tee -a $(LOGDIR)/output.log
12   @for i in $$(seq 1000 1008); do \
13     echo ">> Execução $$i da ETAPA1" | tee -a $(LOGDIR)/output.log; \
14     DELAY=10000 FLUXES=1 SCENARIO_ID=$$i $(SCRIPT) 2>&1 | tee -a $(LOGDIR)/output.log; \
15   done
16
17 exec2:
18   @echo "ETAPA2: Executando com DELAY=100000 FLUXES=1" | tee -a $(LOGDIR)/output.log
19   @for i in $$(seq 1000 1008); do \
20     echo ">> Execução $$i da ETAPA2" | tee -a $(LOGDIR)/output.log; \
21     DELAY=100000 FLUXES=1 SCENARIO_ID=$$i $(SCRIPT) 2>&1 | tee -a $(LOGDIR)/output.log; \
22   done
23
24 exec3:
25   @echo "ETAPA3: Executando com DELAY=10000 FLUXES=4" | tee -a $(LOGDIR)/output.log
26   @for i in $$(seq 1000 1008); do \
27     echo ">> Execução $$i da ETAPA3" | tee -a $(LOGDIR)/output.log; \
```

```
      DELAY=10000 FLUXES=4 SCENARIO_ID=$$i $(SCRIPT) 2>&1 | tee -a
$(LOGDIR)/output.log; \
  done

exec4:
  @echo "ETAPA4: Executando com DELAY=100000 FLUXES=4" | tee -a
$(LOGDIR)/output.log
  @for i in $$(seq 1000 1008); do \
    echo ">> Execução $$i da ETAPA4" | tee -a $(LOGDIR)/output.log; \
    DELAY=100000 FLUXES=4 SCENARIO_ID=$$i $(SCRIPT) 2>&1 | tee -a
$(LOGDIR)/output.log; \
  done

.PHONY: all exec1 exec2 exec3 exec4
```

## 2.3. Resultados obtidos

Parar extrair apenas os logs de interesse, o seguinte comando pode ser utilizado:

```
# comando de filtragem:
awk '/^Running TCP test between/,/^=+/' seu_arquivo_de_log.txt >
filtrado.txt
```

## 2.4. Análise dos resultados

Para fazer a análise dos resultados, coletaram-se do arquivo filtrado.txt somente os dados necessários para compilar as médias das vazões de cada rodada para cada uma das quatro etapas. Com os valores obtidos, foram gerados os arquivos etapa_agregado.csv.

### Importação de Bibliotecas

```
import csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
from collections import defaultdict
import os
import shutil

# Configuração para melhor visualização dos gráficos
plt.rcParams['figure.figsize'] = (12, 8)
plt.rcParams['font.size'] = 12
```

### Processamento dos Arquivos de Dados

Processamos os arquivos de texto diretamente para CSV, extraindo apenas os dados essenciais.

```python
def processar_arquivo_txt_fluxo_unico(arquivo_entrada, arquivo_saida):
    """Processa um arquivo de texto para CSV para experimentos com 1 fluxo."""
    # Criar uma cópia do arquivo original antes de modificá-lo
    backup_file = arquivo_entrada + '.bak'
    if not os.path.exists(backup_file):
        shutil.copy2(arquivo_entrada, backup_file)
    else:
        # Restaurar do backup para garantir dados originais
        shutil.copy2(backup_file, arquivo_entrada)

    # Ler o arquivo e filtrar linhas [SUM]
    with open(arquivo_entrada, 'r') as file:
        lines = file.readlines()

    filtered_lines = [line for line in lines if not line.startswith('[SUM]')]

    # Escrever as linhas filtradas de volta no arquivo
    with open(arquivo_entrada, 'w') as file:
        file.writelines(filtered_lines)

    # Processar o arquivo para CSV
    with open(arquivo_entrada, 'r') as infile, open(arquivo_saida, 'w', newline='') as outfile:
        csv_writer = csv.writer(outfile)
        csv_writer.writerow(['ID', 'Interval', 'Transfer', 'Bandwidth'])

        execution_value = "-"  # Valor padrão para execução

        for line in lines:
            # Identificar o início de uma nova seção
            if "Starting IMUNES simulation with scenario ID:" in line:
                scenario_id = line.split(":")[-1].strip()
                execution_value = scenario_id[-1] if scenario_id else "-"
                continue

            # Processar linhas que contêm dados
            if line.startswith('[') and 'sec' in line and 'Bytes' in line and 'bits/sec' in line:
                parts = line.split()
                id_col = execution_value
                interval = parts[2].split('-')[-1] if '-' in parts[2] else parts[2]  # Pegar apenas o segundo valor

                # Extrair e converter Transfer
                transfer_value = float(parts[4])
                transfer_unit = parts[5]
                if 'KBytes' in transfer_unit:
                    transfer_value /= 1024  # Converter para MBytes

                # Extrair e converter Bandwidth
                bandwidth_value = float(parts[6])
                bandwidth_unit = parts[7]
                if 'Kbits/sec' in bandwidth_unit:
                    bandwidth_value /= 1024  # Converter para Mbits/sec
```

```
                csv_writer.writerow([id_col, interval, transfer_value,
bandwidth_value])

    print(f"Arquivo {arquivo_entrada} processado e salvo como
{arquivo_saida}")
    return arquivo_saida
```

```
def
processar_arquivo_txt_multiplos_fluxos(arquivo_entrada, arquivo_saida):
    """Processa um arquivo de texto para CSV para experimentos com 4
fluxos, somando os valores de bandwidth."""
    # Criar uma cópia do arquivo original antes de modificá-lo
    backup_file = arquivo_entrada + '.bak'
    if not os.path.exists(backup_file):
        shutil.copy2(arquivo_entrada, backup_file)
    else:
        # Restaurar do backup para garantir dados originais
        shutil.copy2(backup_file, arquivo_entrada)

    # Ler o arquivo e filtrar linhas [SUM]
    with open(arquivo_entrada, 'r') as file:
        lines = file.readlines()

    filtered_lines = [line for line in lines if not
line.startswith('[SUM]')]

    # Escrever as linhas filtradas de volta no arquivo
    with open(arquivo_entrada, 'w') as file:
        file.writelines(filtered_lines)

    # Dicionário para armazenar os dados agrupados por execução e
intervalo
    dados_agrupados = defaultdict(lambda: {'fluxos': [],
'bandwidth_total': 0})

    # Primeira passagem: coletar todos os dados
    execution_value = "-"  # Valor padrão para execução

    for line in lines:
        # Identificar o início de uma nova seção
        if "Starting IMUNES simulation with scenario ID:" in line:
            scenario_id = line.split(":")[-1].strip()
            execution_value = scenario_id[-1] if scenario_id else "-"
            continue

        # Processar linhas que contêm dados
        if line.startswith('[') and 'sec' in line and 'Bytes' in line and
'bits/sec' in line:
            parts = line.split()
            flow_id = parts[1].strip(']')  # ID do fluxo
            interval = parts[2].split('-')[-1] if '-' in parts[2] else
parts[2]  # Pegar apenas o segundo valor

```

```
            # Extrair e converter Bandwidth
            bandwidth_value = float(parts[6])
            bandwidth_unit = parts[7]
            if 'Kbits/sec' in bandwidth_unit:
                bandwidth_value /= 1024  # Converter para Mbits/sec

            # Agrupar por execução e intervalo
            chave = (execution_value, interval)
            dados_agrupados[chave]['fluxos'].append((flow_id, bandwidth_value))

    # Segunda passagem: calcular a soma dos bandwidths para cada intervalo
    with open(arquivo_saida, 'w', newline='') as outfile:
        csv_writer = csv.writer(outfile)
        csv_writer.writerow(['ID', 'Interval', 'Bandwidth_Total'])

        for (exec_id, interval), dados in dados_agrupados.items():
            # Somar os bandwidths de todos os fluxos para este intervalo
            bandwidth_total = sum(bw for _, bw in dados['fluxos'])

            # Escrever a linha com a soma
            csv_writer.writerow([exec_id, interval, bandwidth_total])

    print(f"Arquivo {arquivo_entrada} processado com soma de fluxos e salvo como {arquivo_saida}")
    return arquivo_saida
```

```
def agregar_dados(arquivo_entrada, arquivo_saida):
    """Agrega os dados por ID (execução)."""
    # Carregar os dados
    df = pd.read_csv(arquivo_entrada)

    # Verificar se a coluna é 'Bandwidth' ou 'Bandwidth_Total'
    bandwidth_col = 'Bandwidth_Total' if 'Bandwidth_Total' in df.columns else 'Bandwidth'

    # Agregar por ID
    df_agregado = df.groupby('ID')[bandwidth_col].mean().reset_index()

    # Renomear a coluna para padronizar
    df_agregado.rename(columns={bandwidth_col: 'Bandwidth'}, inplace=True)

    # Salvar o resultado
    df_agregado.to_csv(arquivo_saida, index=False)

    print(f"Dados agregados salvos como {arquivo_saida}")
    return arquivo_saida
```

```
# Processar os quatro arquivos
etapa1_csv = processar_arquivo_txt_fluxo_unico('Etapa1.txt', 'etapa1.csv')
```

```
etapa2_csv = processar_arquivo_txt_fluxo_unico('Etapa2.txt', 'etapa2.csv')
etapa3_csv = processar_arquivo_txt_multiplos_fluxos('ETAPA3.txt', 'etapa3.csv')  # Usando a função para múltiplos fluxos
etapa4_csv = processar_arquivo_txt_multiplos_fluxos('ETAPA4.txt', 'etapa4.csv')  # Usando a função para múltiplos fluxos

# Agregar os dados
etapa1_agregado = agregar_dados(etapa1_csv, 'etapa1_agregado.csv')
etapa2_agregado = agregar_dados(etapa2_csv, 'etapa2_agregado.csv')
etapa3_agregado = agregar_dados(etapa3_csv, 'etapa3_agregado.csv')
etapa4_agregado = agregar_dados(etapa4_csv, 'etapa4_agregado.csv')
```

### Análise Estatística e Visualização

Calculamos as médias e intervalos de confiança para cada etapa e criamos um único gráfico comparativo.

```
def calcular_medias_e_ic(nome_arquivo, nome_etapa):
    """Calcula médias e intervalo de confiança para um arquivo CSV"""
    try:
        # Ler o arquivo CSV
        df = pd.read_csv(nome_arquivo)

        # Calcular média por rodada (ID)
        medias_por_rodada = df.groupby('ID')['Bandwidth'].mean()

        # Calcular média geral e intervalo de confiança (95%)
        media_geral = medias_por_rodada.mean()
        n = len(medias_por_rodada)
        if n > 1:  # Verificar se há mais de uma amostra para calcular o IC
            intervalo_confianca = stats.t.ppf(0.975, n-1) * (medias_por_rodada.std() / np.sqrt(n))
        else:
            intervalo_confianca = 0

        return {
            'nome_etapa': nome_etapa,
            'medias_por_rodada': medias_por_rodada,
            'media_geral': media_geral,
            'intervalo_confianca': intervalo_confianca
        }
    except Exception as e:
        print(f"Erro ao processar arquivo {nome_arquivo}: {str(e)}")
        return None
```

```
# Calcular estatísticas para cada etapa
resultados = [
    calcular_medias_e_ic(etapa1_agregado, 'Etapa 1 (10ms, 1 Fluxo)'),
    calcular_medias_e_ic(etapa2_agregado, 'Etapa 2 (100ms, 1 Fluxo)'),
```

```
    calcular_medias_e_ic(etapa3_agregado, 'Etapa 3 (10ms, 4 Fluxos)'),
    calcular_medias_e_ic(etapa4_agregado, 'Etapa 4 (100ms, 4 Fluxos)')
]

# Exibir resultados numéricos
print("\nResultados das médias de Bandwidth:\n")
print("-" * 70)

for res in resultados:
    if res is not None:
        print(f"\nEtapa: {res['nome_etapa']}")
        print("\nMédias por rodada (ID):")
        print(res['medias_por_rodada'].to_string())
        print(f"\nMédia geral: {res['media_geral']:.2f} Megabits/
segundo")
        print(f"Intervalo de confiança (95%): ±
{res['intervalo_confianca']:.2f}")
        print("-" * 70)
```

### Resultados Agregados

| Configuração | Parâmetros | Média (Mbit/s) | IC 95% |
|---|---|---|---|
| Etapa 1 | “10ms, 1 fluxo” | 85.14 | “±1.01” |
| Etapa 2 | “100ms, 1 fluxo” | 31.08 | “±0.02” |
| Etapa 3 | “10ms, 4 fluxos” | 83.28 | “±1.09” |
| Etapa 4 | “100ms, 4 fluxos” | 60.14 | “±4.02” |

Dados por Rodada:

Etapa 1 (10ms, 1 fluxo)

| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Média | 85.87 | 81.67 | 85.57 | 85.45 | 85.44 | 85.55 | 85.65 | 85.49 | 85.58 |

Etapa 2 (100ms, 1 fluxo)

| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Média | 31.08 | 31.04 | 31.11 | 31.09 | 31.11 | 31.05 | 31.06 | 31.03 | 31.10 |

Etapa 3 (10ms, 4 fluxos)

| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Média | 83.32 | 83.88 | 83.60 | 83.26 | 80.06 | 83.56 | 83.54 | 85.51 | 82.79 |

Etapa 4 (100ms, 4 fluxos)

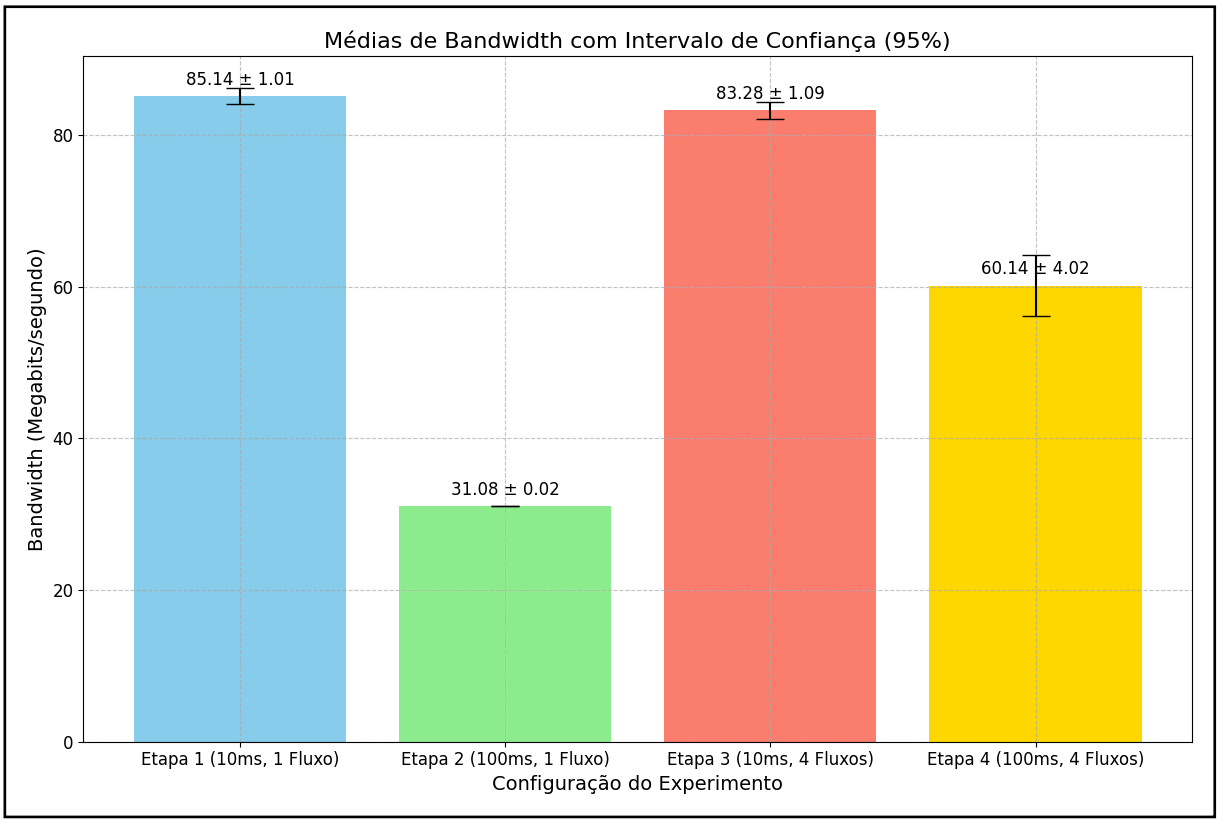
| ID | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Média | 60.20 | 55.25 | 60.78 | 53.07 | 65.74 | 63.06 | 66.91 | 53.11 | 63.13 |

Plotando o gráfico de comparativo

```python
def plotar_grafico_ic(resultados):
    """Cria gráfico com intervalos de confiança para cada etapa"""
    # Filtrar resultados válidos
    resultados_validos = [res for res in resultados if res is not None]

    if not resultados_validos:
        print("Não há resultados válidos para plotar.")
        return

    # Preparar dados para o gráfico
    nomes = [res['nome_etapa'] for res in resultados_validos]
    medias = [res['media_geral'] for res in resultados_validos]
    ics = [res['intervalo_confianca'] for res in resultados_validos]

    # Criar figura
    plt.figure(figsize=(12, 8))

    # Criar barras com intervalos de confiança
    barras = plt.bar(nomes, medias, yerr=ics, capsize=10,
color=['skyblue', 'lightgreen', 'salmon', 'gold'])

    # Configurações do gráfico
    plt.ylabel('Bandwidth (Megabits/segundo)', fontsize=14)
    plt.xlabel('Configuração do Experimento', fontsize=14)
    plt.title('Médias de Bandwidth com Intervalo de Confiança (95%)',
fontsize=16)
    plt.grid(True, linestyle='--', alpha=0.7)

    # Adicionar valores nas barras
    for bar, media, ic in zip(barras, medias, ics):
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2., height + 1,
                f'{media:.2f} ± {ic:.2f}',
                ha='center', va='bottom', fontsize=12)

    plt.tight_layout()
    plt.savefig("grafico_ic_bandwidth.png", dpi=300)
    plt.show()
```

Como resultado obtivemos o gráfico da média de Bandwidth com o intervalo de confiança de 95% para cada uma das etapas do experimento como mostrado na figura abaixo:

Figura 2: Elaborada pelo Autor




# 3. Conclusão

Neste relatório, apresentamos o desenvolvimento e os resultados obtidos no mini projeto de medição ativa em redes utilizando o Iperf. O experimento foi realizado em um cenário simulado com diferentes configurações de retardo e paralelismo TCP, permitindo avaliar o impacto desses fatores na vazão da conexão. Os resultados mostraram que o aumento do retardo de rede teve um impacto significativo na vazão, especialmente quando comparado entre 10ms e 100ms. Além disso, a utilização de múltiplos fluxos TCP também demonstrou um aumento na vazão, embora o efeito não tenha sido tão pronunciado quanto o aumento do retardo. O gráfico comparativo ilustrou claramente as diferenças nas médias de vazão entre os diferentes cenários, destacando a importância do retardo e do paralelismo na performance da rede.