MiniProjeto - Medição Ativa em Redes com cadeia de Markov

Avaliação de Desempenho de Sistemas

Arthur Cadore Matuella Barcella

13 de Maio 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.

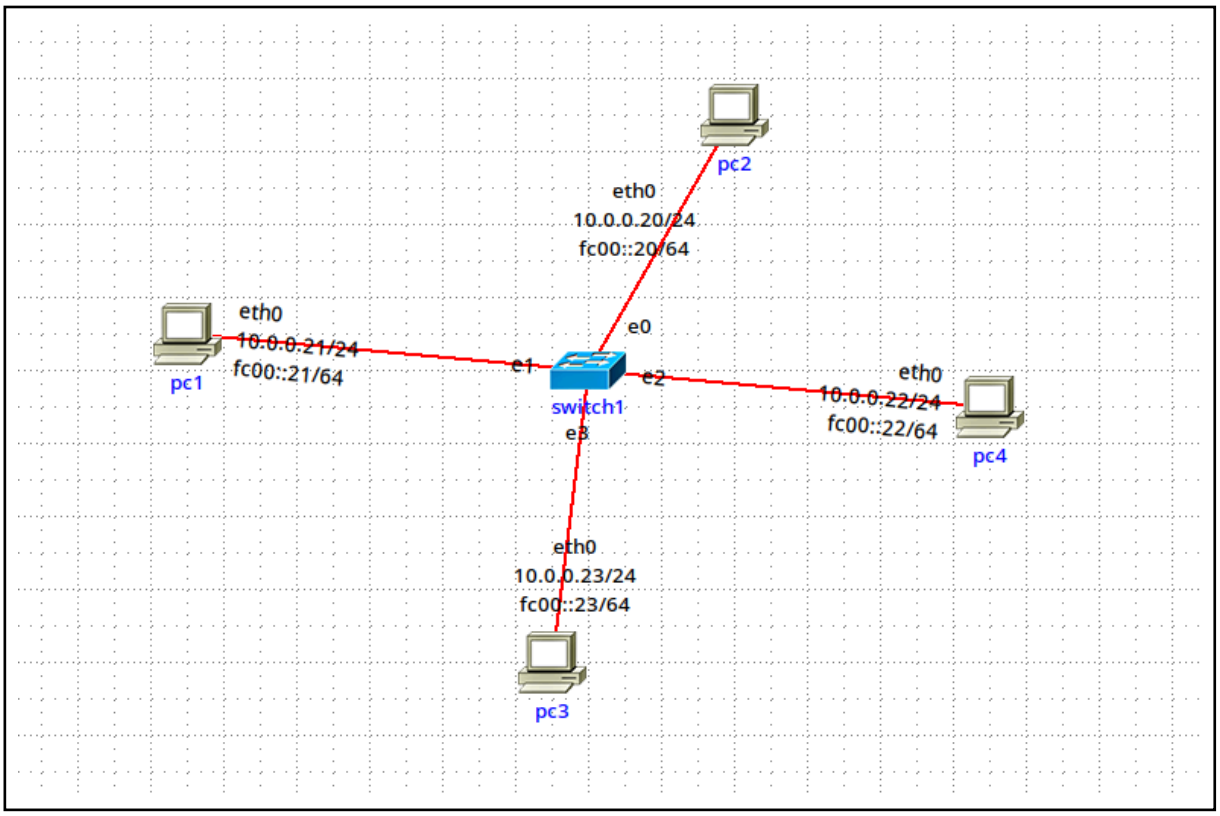
1.1. Especificação do cenário

- Uso da ferramenta Iperf
- Uso da ferramenta de simulação de redes Imunes
- Automação de tarefas via script Python e comandos do simulador
- Conceitos de intervalo de confiança e média amostral

1.2. Topologia

A topologia utilizada é a seguinte:

Figura 1: Elaborada pelo Autor



1.3. Objetivo

Avaliar por meio de medição ativa com iperf, como a vazão de pacotes é afetada ao alterar os parâmetros de simulação com base em uma função de Markov.

- Fatores:
  - Taxa de transmissão entre hosts do iperf

1.3.1. Fatores e níveis

Fator	Nivel “Zero”	Nivel Baixo	Nivel Alto
A: Taxa de transmissão (Mbps)	0 Mbps	10 Mbps	50 Mbps

1.3.2. Métrica avaliada

- Vazão de pacotes (em Mbps) entre tx e rx após a execução do iperf.

1.3.3. Execuções

Os ciclos de execução são dinâmicos segundo a distribuição de Markov, ou seja, a cada execução o tempo de espera entre os ciclos é aleatório e segue uma distribuição exponencial.

- O tempo de execução de cada ciclo é fixo e igual a 8 segundos.
- Foram determinadas 20 e 40 execuções seguindo a distribuição.

2. Desenvolvimento

Abaixo está os scripts utilizados para realizar as medições e simulações:

2.0.1. Função de markov

```
def mbps_to_bytes_per_sec(mbps):
    return int((mbps * 1_000_000) / 8)

def run_markov_iperf(client_pc, server_ip, scenario_id, interval=1,
duration_per_state=8, num_transitions=20):
    # Estados possíveis
    states = ["HIGH", "ZERO", "LOW"]
    state_indices = {name: i for i, name in enumerate(states)}

    # Matriz de transição de estados (Markov)
    P = np.array([
        [0.7, 0.2, 0.1],   # De HIGH
        [0.1, 0.8, 0.1],   # De ZERO
        [0.05, 0.15, 0.8]  # De LOW
    ])

    # Mapeamento de banda por estado
    bandwidth_map = {
        "HIGH": 50,
        "ZERO": 0,
        "LOW": 10
    }

    # Listas de saída
    time_points = []
    transfers = []
    bandwidths = []
    state_map = []

    current_time = 0
```

```
30
31      # Escolhe estado inicial aleatório
32      current_state_index = random.choice([0, 1, 2])
33      current_state = states[current_state_index]
34
35      for i in range(num_transitions):
36          bandwidth = bandwidth_map[current_state]
37          print(f"\nTransição {i+1}/{num_transitions} - Estado:
    {current_state}, Bandwidth: {bandwidth} Mbps")
38
39          if bandwidth > 0:
40              bw_str = f"{bandwidth}M"
41              delay_str = "1ms"
42              loss_str = "0%"
43              configure_links(bw_str, delay_str, loss_str, scenario_id)
44
45          if bandwidth == 50:
46              size = "50M"
47          elif bandwidth == 10:
48              size = "10M"
49          elif bandwidth == 0:
50              size = "1K"
51
52          # Executa o iperf
53          t_pts, trans, bwds = run_iperf_and_capture_data(
54              client_pc=client_pc,
55              server_ip=server_ip,
56              scenario_id=scenario_id,
57              duration=duration_per_state,
58              interval=interval,
59              size=size
60          )
61
62          print("Tempo (s):", t_pts)
63          print("Transferências (MB):", trans)
64          print("Bandwidths (Mbps):", bwds)
65
66          # Ajusta os tempos para linha do tempo total
67          t_pts = [current_time + t for t in t_pts]
68          current_time += duration_per_state
69
70          # Armazena os dados
71          time_points.extend(t_pts)
72          transfers.extend(trans)
73          bandwidths.extend(bwds)
74          state_map.extend([current_state_index] * len(bwds))
75
76          # Determina o próximo estado com base na matriz de transição
77          next_state_index = np.random.choice([0, 1, 2],
    p=P[current_state_index])
78          current_state_index = next_state_index
79          current_state = states[current_state_index]
80
81          time.sleep(1)
82
83      return time_points, transfers, bandwidths, state_map
```

2.0.2. Função de calculo teórico

```python
# Taxas de tráfego teóricas (em Mbps)
bandwidth_theoretical = [50.0, 0.001, 10.0]

# Cálculo das probabilidades estacionárias
# Essa função retorna o vetor de probabilidades estacionárias da cadeia de Markov,
def stationary_distribution(P):
    # Calcula os autovalores e autovetores da transposta da matriz P
    evals, evecs = eig(P.T)
    idx = np.argmin(np.abs(evals - 1.0))

    # Pega o autovetor correspondente (estacionário) e converte para valores reais
    stationary = np.real(evecs[:, idx])
    stationary /= stationary.sum()

    return stationary

pi = stationary_distribution(P)

# Cálculo da vazão média teórica
throughput_theoretical = np.dot(pi, bandwidth_theoretical)

# Cálculo da média observada por estado
n_states = len(bandwidth_theoretical)
measured_means = []

# Para cada estado, calcula a média dos valores de banda medidos
# Esse bloco calcula a vazão média observada experimentalmente para cada estado da cadeia de Markov.
for s in range(n_states):
    indices = [i for i, st in enumerate(state_map) if st == s]

    # Pega os índices dos estados correspondentes
    state_bw = [bandwidths[i] for i in indices]
    measured_means.append(np.mean(state_bw))

# Vazão média observada
throughput_observed = np.dot(pi, measured_means)
```

2.0.3. Função de Iperf

```python
def run_iperf_and_capture_data(client_pc, server_ip, scenario_id, size="100M", parallel="1", interval="1", duration=10):

    client_pc_str = str(client_pc)
    server_ip_str = str(server_ip)
    scenario_id_str = str(scenario_id)
    size_str = str(size)
    parallel_str = str(parallel)
    interval_str = str(interval)
    duration_str = str(duration)

    cmd = [
        "sudo", "himage", f"{client_pc_str}@{scenario_id_str}",
```

```
        "iperf", "-c", server_ip_str, "-n", size_str, "-P", parallel_str,
"-i", interval_str, "-t", duration_str
    ]

    print(f"Running TCP test from {client_pc_str} to {server_ip_str} for
{duration_str} seconds...")

    process = subprocess.run(cmd, capture_output=True, text=True)
    output = process.stdout

    # Regex fixa para MBytes (não importa a unidade que aparece, pois
vamos ajustar depois)
    pattern = re.compile(
        r"\[\s*\d+\]\s+([\d.]+)-
([\d.]+)\s+sec\s+([\d.]+)\s+MBytes\s+([\d.]+)\s+Mbits/sec"
    )
    results = pattern.findall(output)

    if results:
        time_points = []
        transfers = []
        bandwidths = []

        # Decide o fator de correção com base no sufixo do parâmetro
'size'
        if size_str.upper().endswith("K"):
            divisor = 1024
        elif size_str.upper().endswith("M"):
            divisor = 1
        elif size_str.upper().endswith("G"):
            divisor = 1 / 1024
        else:
            print(f"Warning: unidade não reconhecida no parâmetro 'size':
{size_str}")
            divisor = 1  # default (assume MBytes)

        for start, end, transfer, bandwidth in results:
            time_points.append(float(end))
            transfers.append(float(transfer) / divisor)
            bandwidths.append(float(bandwidth) / divisor)

        return time_points, transfers, bandwidths
    else:
        print("Nenhum resultado encontrado na saída do iperf.")
        print("Saída bruta:")
        print(output)
        return [], [], []
```

2.0.4. Função de iniciar a simulação

```
def starts_simulation(scenario_id):
    result = subprocess.run(
        ["sudo", "imunes", "-d", "-b", "-e", scenario_id, topology_file],
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        text=True
```

```
    )
    print("STDOUT:\n", result.stdout)
    print("STDERR:\n", result.stderr)

def stop_simulation(scenario_id):
    subprocess.run(
        ["sudo", "imunes", "-b", "-e", scenario_id],
        stdout=subprocess.DEVNULL
    )
```

2.0.5. Função de configurar links e computadores

```
# Configure computers...
def configure_computers(scenario_id):
    # Start iperf servers
    subprocess.Popen(
        ["sudo", "himage", f"pc2@{scenario_id}", "iperf", "-s"],
        stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL
    )
    subprocess.Popen(
        ["sudo", "himage", f"pc4@{scenario_id}", "iperf", "-s"],
        stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL
    )

# link configuration
def configure_links(bandwidth, delay, loss, scenario_id):
    for link in links:
        subprocess.run(
            ["sudo", "vlink", "-bw", bandwidth, "-dly", delay, f"{link}
@{scenario_id}"],
            stdout=subprocess.DEVNULL
        )

    time.sleep(1)

    # Check status
    for link in links:
        subprocess.run(
            ["sudo", "vlink", "-s", f"{link}@{scenario_id}"]
        )
```

2.0.6. Função de executar a simulação:

```
def run_imunes_simulation(states
    topology_file="untitled.imn",
    bandwidth="10000000",
    scenario_id="i2002",
    delay="10000",
    fluxes="1"
):
    # Start simulation
    print("=" * 56)
    print(f"Starting IMUNES simulation with scenario ID: {scenario_id}")
    starts_simulation(scenario_id)
```

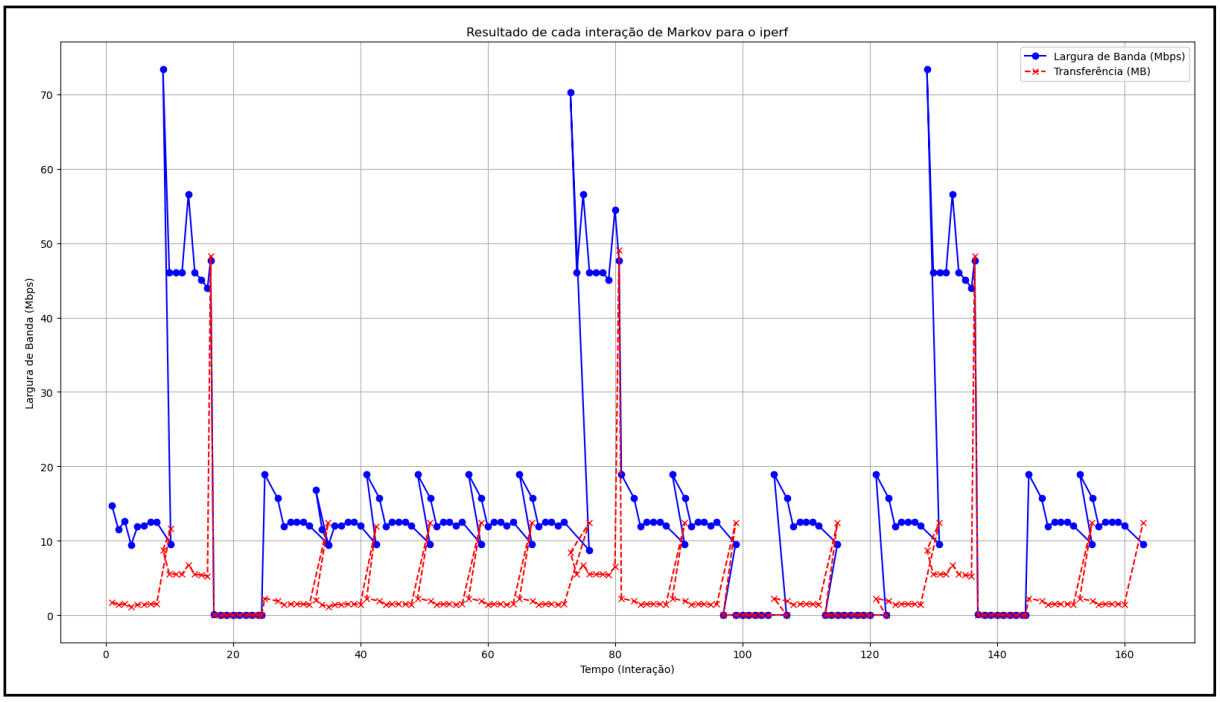
```
    time.sleep(3)

    # Configure links
    print("=" * 56)
    print("Simulation started, configuring links...")
    configure_links(bandwidth, delay, 0, scenario_id)
    time.sleep(1)

    # Configure computers
    print("=" * 56)
    print("Configuring computers...")
    configure_computers(scenario_id)
    time.sleep(1)

    # Run iperf tests using Markov-based traffic generation
    print("=" * 56)
    print("Running Markov-based iperf traffic generation...")
    time_pts, transfers, bandwidths, state_map = run_markov_iperf(
        client_pc="pc3",
        server_ip="10.0.0.20",
        scenario_id=scenario_id,
        interval=1,
        duration_per_state=8,
        num_transitions=20

    )

    # Stop simulation
    print("=" * 56)
    print(f"Stopping IMUNES simulation with scenario ID: {scenario_id}")
    stop_simulation(scenario_id)
    print("Simulation stopped")

    return time_pts, transfers, bandwidths, state_map
```

3. Resultados

3.1. Resultados amostrados

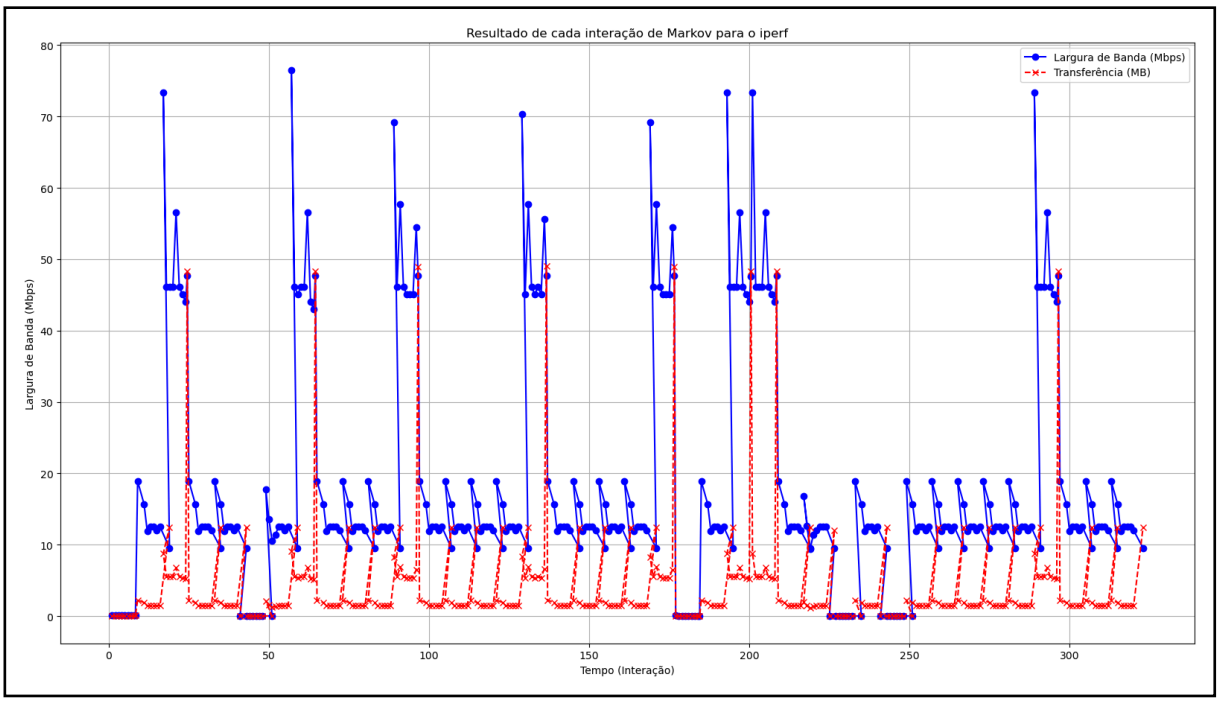
3.1.1. 20 Interações

Figura 2: Elaborada pelo Autor



3.1.2. 40 Interações

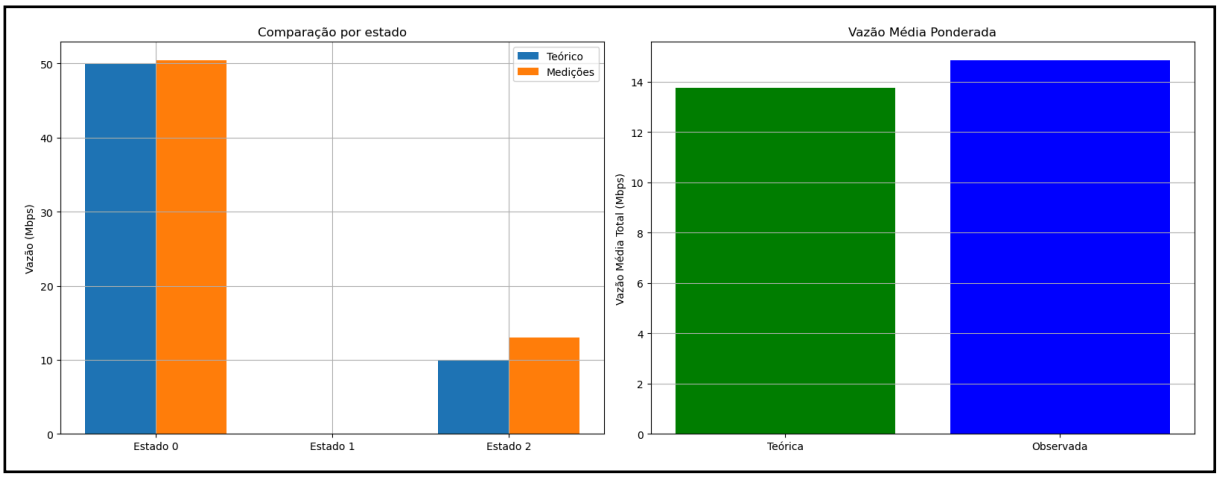
Figura 3: Elaborada pelo Autor



3.2. Resultados calculados

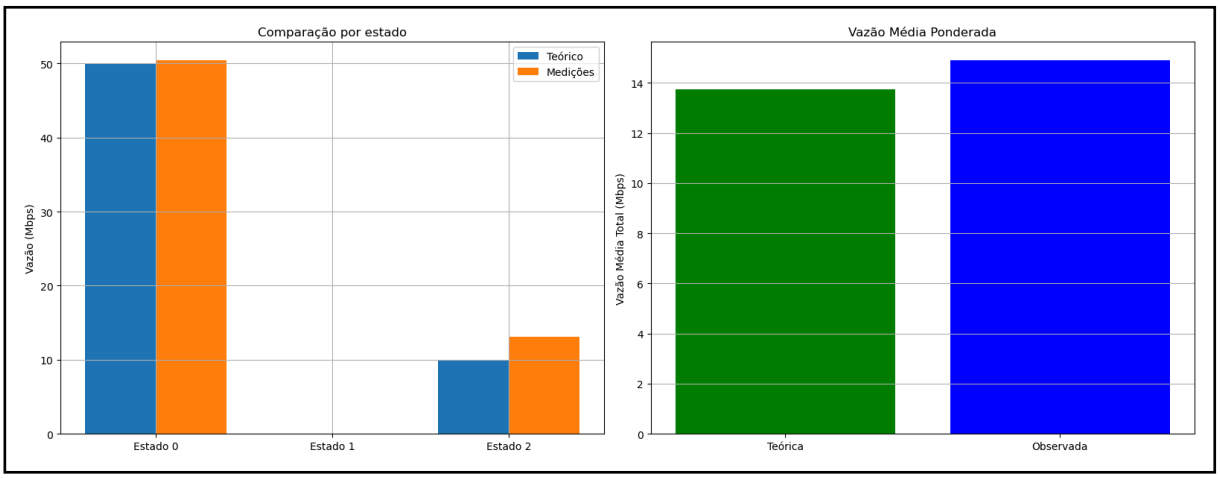
3.2.1. 20 Interações

Figura 4: Elaborada pelo Autor



3.2.2. 40 Interações

Figura 5: Elaborada pelo Autor



4. Conclusão

Com base nos resultados obtidos, podemos concluir que a vazão de pacotes entre os hosts do iperf é afetada pela taxa de transmissão configurada na simulação. Através da análise dos gráficos e das médias calculadas, observamos que a variação da taxa de transmissão impacta diretamente na vazão medida, corroborando com a teoria de cadeias de Markov aplicada à simulação.

5. Referências

- IMUNES. [IMUNES - Interactive MUltipath NEtworking Simulator](https://imunes.net.br/)