



**INSTITUTO
FEDERAL**

Santa Catarina

Câmpus
São José

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	3
1.1. Especificação do cenário	3
1.2. Topologia	3
1.3. Objetivo	3
1.3.1. Fatores e níveis	4
1.3.2. Métrica avaliada	4
1.3.3. Execuções	4
2. Desenvolvimento	4
2.0.1. Função de markov	4
2.0.2. Função de calculo teórico	6
2.0.3. Função de Iperf	6
2.0.4. Função de iniciar a simulação	7
2.0.5. Função de configurar links e computadores	8
2.0.6. Função de executar a simulação:	8
3. Resultados	10
3.1. Resultados amostrados	10
3.1.1. 20 Interações	10
3.1.2. 40 Interações	10
3.2. Resultados calculados	11
3.2.1. 20 Interações	11
3.2.2. 40 Interações	11
4. Conclusão	11
5. Referências	11

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.

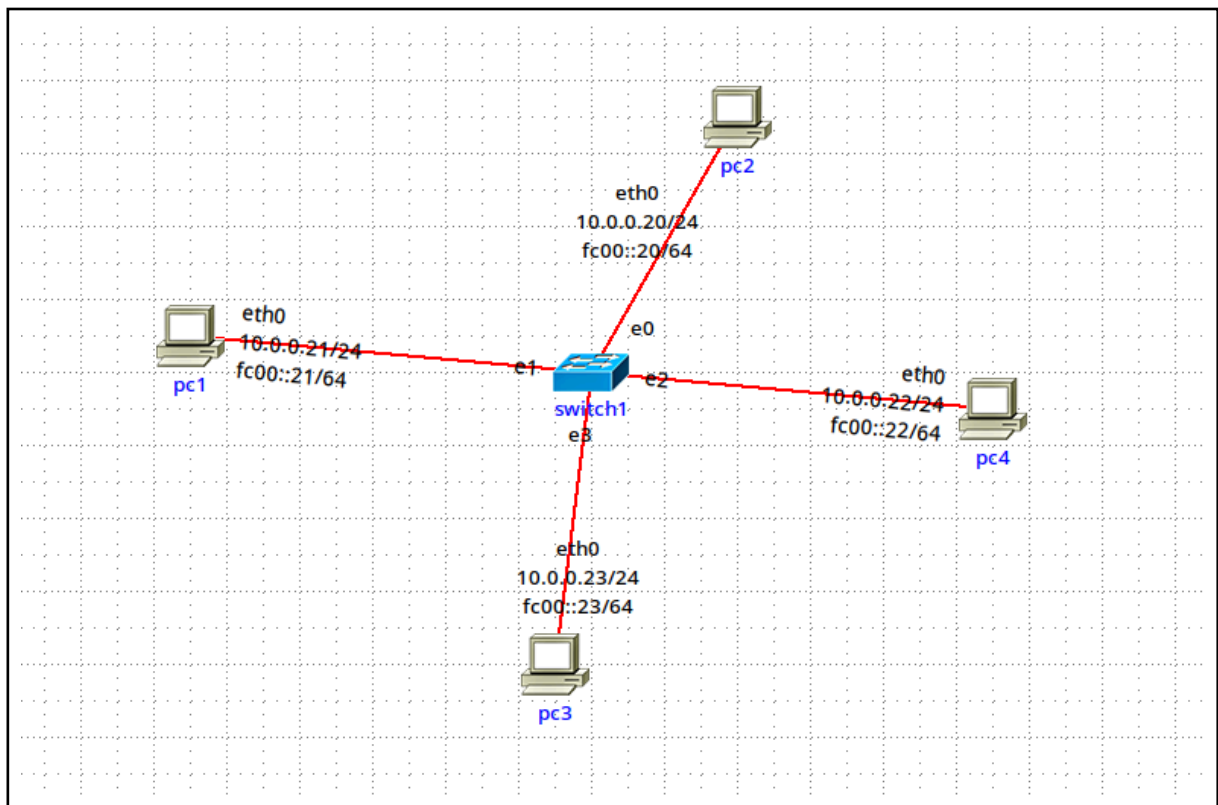
1.1. Especificação do cenário

- 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	Nível “Zero”	Nível Baixo	Nível 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

```
1 def mbps_to_bytes_per_sec(mbps):
2     return int((mbps * 1_000_000) / 8)
3
4 def run_markov_iperf(client_pc, server_ip, scenario_id, interval=1,
5 duration_per_state=8, num_transitions=20):
6     # Estados possíveis
7     states = ["HIGH", "ZERO", "LOW"]
8     state_indices = {name: i for i, name in enumerate(states)}
9
10    # Matriz de transição de estados (Markov)
11    P = np.array([
12        [0.7, 0.2, 0.1], # De HIGH
13        [0.1, 0.8, 0.1], # De ZERO
14        [0.05, 0.15, 0.8] # De LOW
15    ])
16
17    # Mapeamento de banda por estado
18    bandwidth_map = {
19        "HIGH": 50,
20        "ZERO": 0,
21        "LOW": 10
22    }
23
24    # Listas de saída
25    time_points = []
26    transfers = []
27    bandwidths = []
28    state_map = []
29
30    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:
38 {current_state}, Bandwidth: {bandwidth} Mbps")
39
40     if bandwidth > 0:
41         bw_str = f"{bandwidth}M"
42         delay_str = "1ms"
43         loss_str = "0%"
44         configure_links(bw_str, delay_str, loss_str, scenario_id)
45
46     if bandwidth == 50:
47         size = "50M"
48     elif bandwidth == 10:
49         size = "10M"
50     elif bandwidth == 0:
51         size = "1K"
52
53     # Executa o iperf
54     t_pts, trans, bwds = run_iperf_and_capture_data(
55         client_pc=client_pc,
56         server_ip=server_ip,
57         scenario_id=scenario_id,
58         duration=duration_per_state,
59         interval=interval,
60         size=size
61     )
62
63     print("Tempo (s):", t_pts)
64     print("Transferências (MB):", trans)
65     print("Bandwidths (Mbps):", bwds)
66
67     # Ajusta os tempos para linha do tempo total
68     t_pts = [current_time + t for t in t_pts]
69     current_time += duration_per_state
70
71     # Armazena os dados
72     time_points.extend(t_pts)
73     transfers.extend(trans)
74     bandwidths.extend(bwds)
75     state_map.extend([current_state_index] * len(bwds))
76
77     # Determina o próximo estado com base na matriz de transição
78     next_state_index = np.random.choice([0, 1, 2],
79 p=P[current_state_index])
80     current_state_index = next_state_index
81     current_state = states[current_state_index]
82
83     time.sleep(1)
84
85     return time_points, transfers, bandwidths, state_map

```

2.0.2. Função de calculo teórico

```
1 # Taxas de tráfego teóricas (em Mbps)
2 bandwidth_theoretical = [50.0, 0.001, 10.0]
3
4 # Cálculo das probabilidades estacionárias
5 # Essa função retorna o vetor de probabilidades estacionárias da cadeia
  de Markov,
6 def stationary_distribution(P):
7     # Calcula os autovalores e autovetores da transposta da matriz P
8     evals, evecs = eig(P.T)
9     idx = np.argmin(np.abs(evals - 1.0))
10
11     # Pega o autovetor correspondente (estacionário) e converte para
    valores reais
12     stationary = np.real(evecs[:, idx])
13     stationary /= stationary.sum()
14
15     return stationary
16
17 pi = stationary_distribution(P)
18
19 # Cálculo da vazão média teórica
20 throughput_theoretical = np.dot(pi, bandwidth_theoretical)
21
22 # Cálculo da média observada por estado
23 n_states = len(bandwidth_theoretical)
24 measured_means = []
25
26 # Para cada estado, calcula a média dos valores de banda medidos
27 # Esse bloco calcula a vazão média observada experimentalmente para cada
    estado da cadeia de Markov.
28 for s in range(n_states):
29     indices = [i for i, st in enumerate(state_map) if st == s]
30
31     # Pega os índices dos estados correspondentes
32     state_bw = [bandwidths[i] for i in indices]
33     measured_means.append(np.mean(state_bw))
34
35 # Vazão média observada
36 throughput_observed = np.dot(pi, measured_means)
```

2.0.3. Função de Iperf

```
1 def run_iperf_and_capture_data(client_pc, server_ip, scenario_id,
  size="100M", parallel="1", interval="1", duration=10):
2
3     client_pc_str = str(client_pc)
4     server_ip_str = str(server_ip)
5     scenario_id_str = str(scenario_id)
6     size_str = str(size)
7     parallel_str = str(parallel)
8     interval_str = str(interval)
9     duration_str = str(duration)
10
11     cmd = [
12         "sudo", "himage", f"{client_pc_str}@{scenario_id_str}",
```

```

13     "iperf", "-c", server_ip_str, "-n", size_str, "-P", parallel_str,
14     "-i", interval_str, "-t", duration_str
15 ]
16
17     print(f"Running TCP test from {client_pc_str} to {server_ip_str} for
18     {duration_str} seconds...")
19
20     process = subprocess.run(cmd, capture_output=True, text=True)
21     output = process.stdout
22
23     # Regex fixa para MBytes (não importa a unidade que aparece, pois
24     # vamos ajustar depois)
25     pattern = re.compile(
26         r"[\s*\d+\s]+\s+([\d.]+)-
27         ([\d.]+)\s+sec\s+([\d.]+)\s+MBytes\s+([\d.]+)\s+Mbps/sec"
28     )
29     results = pattern.findall(output)
30
31     if results:
32         time_points = []
33         transfers = []
34         bandwidths = []
35
36         # Decide o fator de correção com base no sufixo do parâmetro
37         'size'
38         if size_str.upper().endswith("K"):
39             divisor = 1024
40         elif size_str.upper().endswith("M"):
41             divisor = 1
42         elif size_str.upper().endswith("G"):
43             divisor = 1 / 1024
44         else:
45             print(f"Warning: unidade não reconhecida no parâmetro 'size':
46             {size_str}")
47             divisor = 1 # default (assume MBytes)
48
49         for start, end, transfer, bandwidth in results:
50             time_points.append(float(end))
51             transfers.append(float(transfer) / divisor)
52             bandwidths.append(float(bandwidth) / divisor)
53
54         return time_points, transfers, bandwidths
55     else:
56         print("Nenhum resultado encontrado na saída do iperf.")
57         print("Saída bruta:")
58         print(output)
59         return [], [], []

```

2.0.4. Função de iniciar a simulação

```

1 def starts_simulation(scenario_id):
2     result = subprocess.run(
3         ["sudo", "imunes", "-d", "-b", "-e", scenario_id, topology_file],
4         stdout=subprocess.PIPE,
5         stderr=subprocess.PIPE,
6         text=True

```

```

7     )
8     print("STDOUT:\n", result.stdout)
9     print("STDERR:\n", result.stderr)
10
11 def stop_simulation(scenario_id):
12     subprocess.run(
13         ["sudo", "imunes", "-b", "-e", scenario_id],
14         stdout=subprocess.DEVNULL
15     )

```

2.0.5. Função de configurar links e computadores

```

1  # Configure computers...
2  def configure_computers(scenario_id):
3      # Start iperf servers
4      subprocess.Popen(
5          ["sudo", "himage", f"pc2@{scenario_id}", "iperf", "-s"],
6          stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL
7      )
8      subprocess.Popen(
9          ["sudo", "himage", f"pc4@{scenario_id}", "iperf", "-s"],
10         stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL
11     )
12
13 # link configuration
14 def configure_links(bandwidth, delay, loss, scenario_id):
15     for link in links:
16         subprocess.run(
17             ["sudo", "vlink", "-bw", bandwidth, "-dly", delay, f"{link}
18             @{scenario_id}"],
19             stdout=subprocess.DEVNULL
20         )
21         time.sleep(1)
22
23     # Check status
24     for link in links:
25         subprocess.run(
26             ["sudo", "vlink", "-s", f"{link}@{scenario_id}"]
27         )

```

2.0.6. Função de executar a simulação:

```

1  def run_imunes_simulation(states
2      topology_file="untitled.imn",
3      bandwidth="10000000",
4      scenario_id="i2002",
5      delay="10000",
6      fluxes="1"
7  ):
8      # Start simulation
9      print("=" * 56)
10     print(f"Starting IMUNES simulation with scenario ID: {scenario_id}")
11     starts_simulation(scenario_id)

```



```

12     time.sleep(3)
13
14     # Configure links
15     print("=" * 56)
16     print("Simulation started, configuring links...")
17     configure_links(bandwidth, delay, 0, scenario_id)
18     time.sleep(1)
19
20     # Configure computers
21     print("=" * 56)
22     print("Configuring computers...")
23     configure_computers(scenario_id)
24     time.sleep(1)
25
26     # Run iperf tests using Markov-based traffic generation
27     print("=" * 56)
28     print("Running Markov-based iperf traffic generation...")
29     time_pts, transfers, bandwidths, state_map = run_markov_iperf(
30         client_pc="pc3",
31         server_ip="10.0.0.20",
32         scenario_id=scenario_id,
33         interval=1,
34         duration_per_state=8,
35         num_transitions=20
36     )
37
38
39     # Stop simulation
40     print("=" * 56)
41     print(f"Stopping IMUNES simulation with scenario ID: {scenario_id}")
42     stop_simulation(scenario_id)
43     print("Simulation stopped")
44
45     return time_pts, transfers, bandwidths, state_map

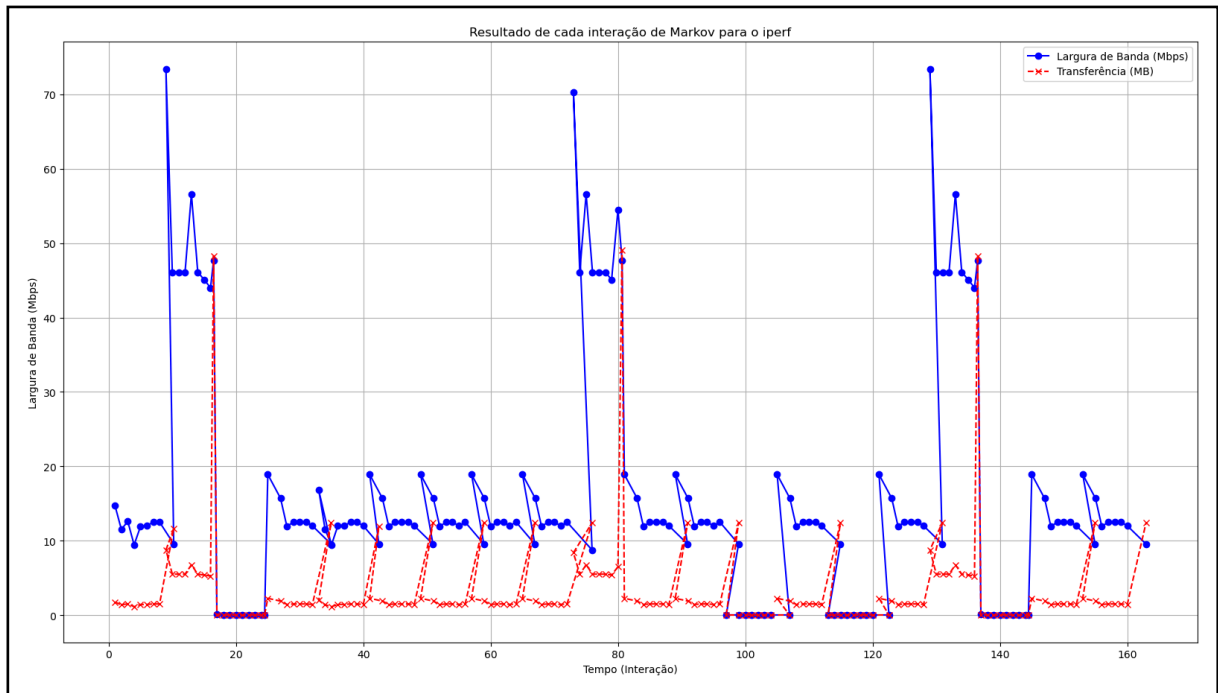
```

3. Resultados

3.1. Resultados amostrados

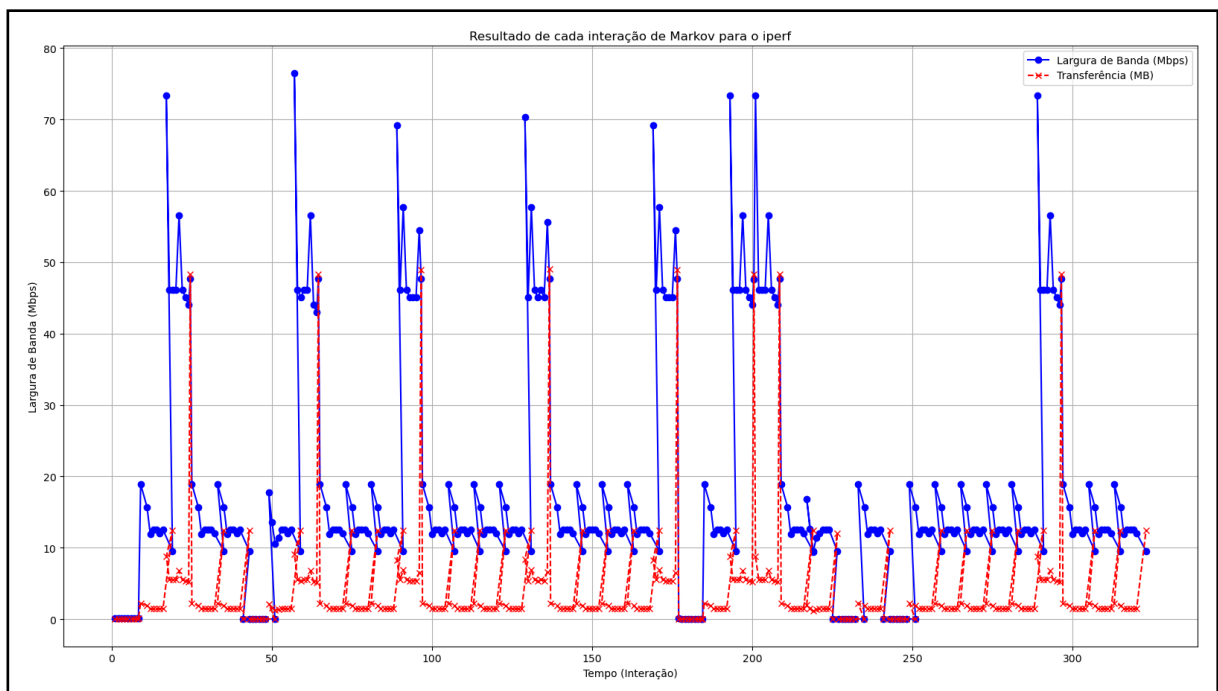
3.1.1. 20 Interações

Figura 2: Elaborada pelo Autor



3.1.2. 40 Interações

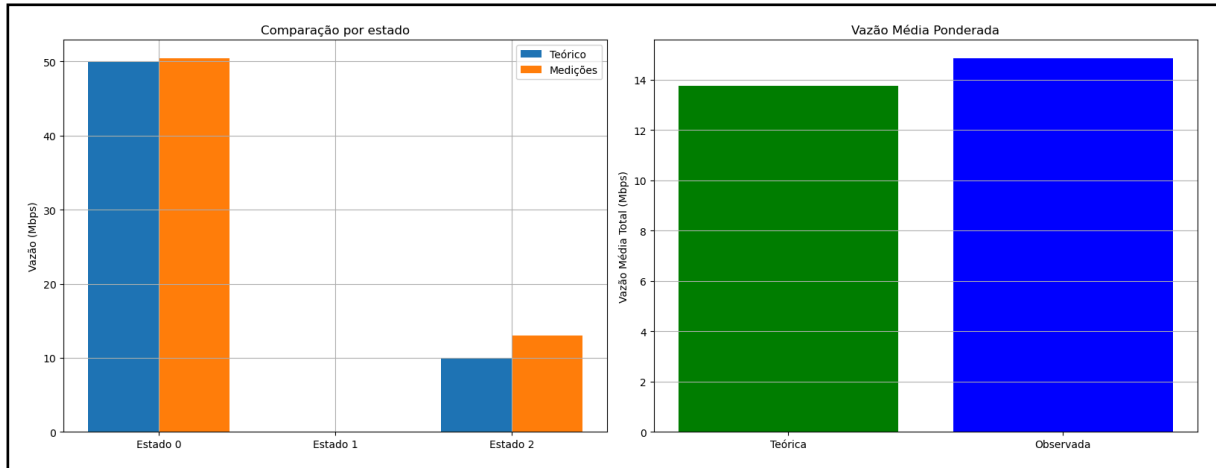
Figura 3: Elaborada pelo Autor



3.2. Resultados calculados

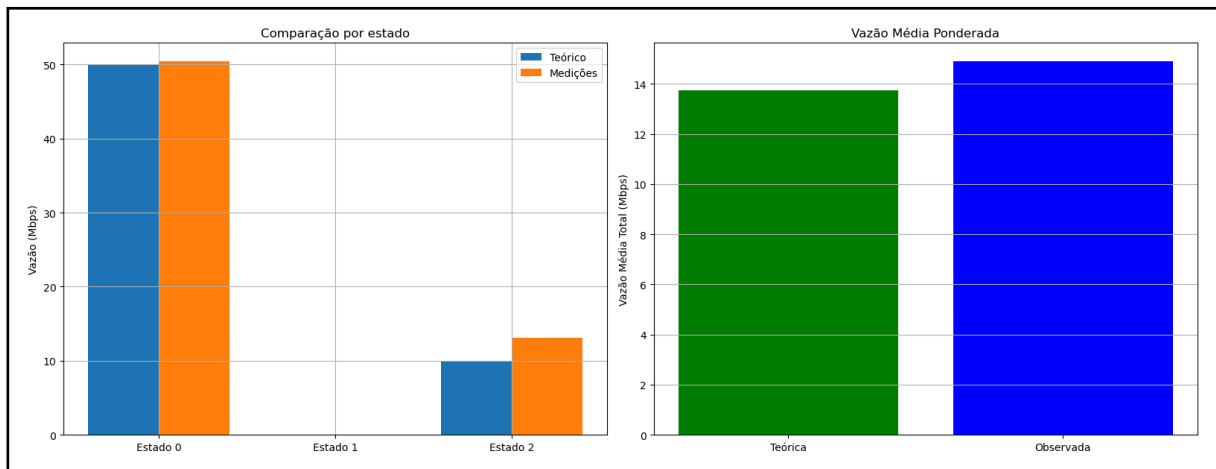
3.2.1. 20 Interações

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/>)