Relatório - Simulação de Rede Filas

Avaliação de Desempenho de Sistemas

Arthur Cadore Matuella Barcella

04 de Julho de 2025

Engenharia de Telecomunicações - IFSC-SJ

Sumário

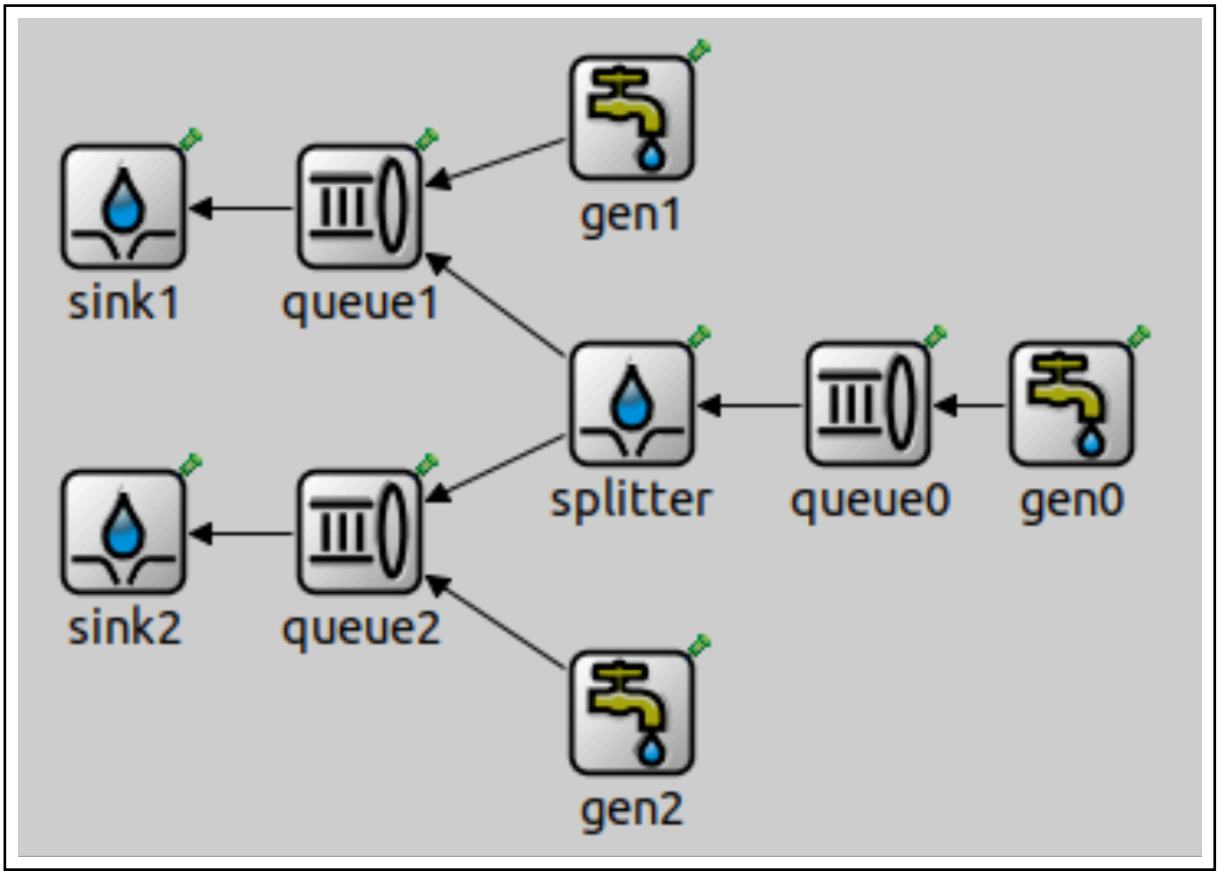
1. Introdução

2. Implementação do modelo

A implementação foi realizada em Python, utilizando os dados de simulação do OMNeT++. O script processou os arquivos de saída e gerou os gráficos solicitados. A seguir, cada item é apresentado com os respectivos resultados e comentários.

Crie um modelo para representar uma rede de filas MM1 conforme indicado abaixo. Uma fila (estação) e ou um splitter deverá ser acrescentado a rede abaixo.

Figura 1: Elaborada pelo Autor



Esquematico do trabalho

Os módulos gen0, gen1 e gen2 são geradores de mensagens com intervalos de envio seguindo distribuição exponencial, sendo o tempo médio parametrizável.

- Os módulos queue0, queue1 e queue 2 são filas com buffer infinito, um servidor e tempo de serviço configurável. Estes módulos devem armazenar os tamanhos da fila de forma vetorial
- O splitter deve ter 2 saídas com probabilidades configuráveis
- O sorvedouro deve permitir gerar uma estátistica do tempo médio de requisições das requisições desde a sua entrada no sistema

2.1. Implementação do Splitter

O Splitter é um módulo que recebe mensagens e as encaminha para uma de duas saídas baseado em uma probabilidade configurável. Ele utiliza uma distribuição uniforme para decidir qual saída usar, permitindo balanceamento de carga entre as filas subsequentes.

```
#include <omnetpp.h>
using namespace omnetpp;

class Splitter : public cSimpleModule {
    double prob;

  protected:
    virtual void initialize() override {
        prob = par("prob");
    }

    virtual void handleMessage(cMessage *msg) override {
        int outGate = (uniform(0, 1) < prob) ? 0 : 1;
        send(msg, "out", outGate);
    }
};

Define_Module(Splitter);
```

2.2. Implementação do Queue

A implementação da fila (Queue) segue o modelo M/M/1 com buffer infinito. O módulo gerencia uma fila FIFO, processa mensagens com tempo de serviço exponencial configurável, e registra estatísticas do tamanho da fila ao longo do tempo. Quando uma mensagem chega e o servidor está ocupado, ela é inserida na fila; caso contrário, o processamento inicia imediatamente.

```
#include <string.h>
#include <omnetpp.h>

using namespace omnetpp;

class Queue : public cSimpleModule {
private:
    // local variable
    cQueue buffer;
    cMessage *endServiceEvent;
    cMessage *currentClient;
    simtime_t service_time;
    simsignal_t queueSizeSignal;
    cOutVector queueLengthVector;
public:
    // constructor
    Queue(); // constructor
    virtual ~Queue(); // destructor
protected:
    virtual void initialize();
    virtual void finish();
```

```
22      virtual void handleMessage(cMessage *msg);
23  };
24
25  Define_Module(Queue);
26      Queue::Queue() {
27      endServiceEvent=NULL;
28  }
29
30  Queue::~Queue() {
31      cancelAndDelete(endServiceEvent);
32  }
33
34  void Queue::initialize() {
35      queueSizeSignal = registerSignal("queueLength");
36      endServiceEvent=new cMessage("endService");
37      emit(queueSizeSignal, 0);
38      queueLengthVector.setName("queueLength");
39      queueLengthVector.record(0);
40  }
41
42  void Queue::finish() {}
43
44  void Queue::handleMessage(cMessage *msg) {
45      cMessage *pkt;
46      if (msg==endServiceEvent) {
47          send(currentClient,"out");
48          if (!buffer.isEmpty()) {
49              currentClient=(cMessage*)buffer.pop();
50              emit(queueSizeSignal, buffer.getLength());
51              queueLengthVector.record(buffer.getLength());
52              service_time=par("serviceTime");
53              scheduleAt(simTime()+service_time,endServiceEvent);
54          } else {
55              emit(queueSizeSignal, 0);
56              queueLengthVector.record(0);
57          }
58      } else {
59          if (endServiceEvent->isScheduled()) {
60              buffer.insert(msg);
61              emit(queueSizeSignal, buffer.getLength());
62              queueLengthVector.record(buffer.getLength());
63          } else {
64              currentClient = msg;
65              service_time=par("serviceTime");
66              scheduleAt(simTime()+service_time,endServiceEvent);
67          }
68      }
69  }
```

2.3. Implementação do Sink

O Sink é o módulo final que recebe as mensagens processadas e calcula estatísticas de atraso. Ele registra o tempo total de permanência no sistema (desde a criação da mensagem até sua chegada), mantém estatísticas online (média, contagem) e gera vetores de saída para análise posterior. Este módulo é essencial para avaliar o desempenho end-to-end do sistema.

```cpp
#include <string.h>
#include <omnetpp.h>

using namespace omnetpp;

class Sink : public cSimpleModule {
private:
    // online stats
    cStdDev delayStats;
    cOutVector delayVector;
public:
    Sink(); // constructor
    virtual ~Sink(); // destructor
protected:
    virtual void initialize();
    virtual void finish();
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Sink);
Sink::Sink() {}

Sink::~Sink() {}

void Sink::initialize() {
    delayStats.setName("TotalDelay");
    delayVector.setName("Delay");
}

void Sink::finish() {
    recordScalar("Ave delay",delayStats.getMean());
    recordScalar("Number of packets",delayStats.getCount());
}

void Sink::handleMessage(cMessage *msg) {
    // compute queueing delay
    simtime_t delay=simTime() - msg->getCreationTime();
    // update stats
    delayStats.collect(delay);
    delayVector.record(delay);
    // delete msg
    delete(msg);
}
```

2.4. Implementação do Modelo

O modelo de rede define a topologia completa do sistema, conectando todos os módulos através de portas de entrada e saída. A rede possui três geradores, três filas, um splitter e dois sinks. O splitter distribui o tráfego do gerador 0 entre as filas 1 e 2, enquanto os geradores 1 e 2 alimentam diretamente suas respectivas filas. Esta configuração permite estudar o impacto do balanceamento de carga e da distribuição de tráfego no desempenho do sistema.

```
package src;
import src.Generator;
import src.Queue;
```

```
import src.Sink;
import src.Splitter;

network MM1
{
    submodules:
        gen0: Generator;
        gen1: Generator;
        gen2: Generator;

        queue0: Queue;
        queue1: Queue;
        queue2: Queue;

        splitter: Splitter {
            parameters:
                prob = default(0.6); // pode ser sobrescrito no .ini
        }

        sink1: Sink;
        sink2: Sink;

    connections allowunconnected:
        gen0.out --> queue0.in[0];
        queue0.out --> splitter.in[0];
        splitter.out[0] --> queue1.in[0];
        splitter.out[1] --> queue2.in[0];
        gen1.out --> queue1.in[1];
        gen2.out --> queue2.in[1];
        queue1.out --> sink1.in[0];
        queue2.out --> sink2.in[0];
}
```

3. Coleta de Dados

Configurar a geração de requisições (workload) para cobrir todas seguintes combinações de carga abaixo:

Módulo	carga 1 - 1/(req/s)	carga 2 - 1/(req/s)
gen0	0.7	0.8
gen1	0.9	1.3
gen2	0.7	1.5

Supor que os servidores trabalham na seguinte taxa:

Módulo	tempo médio de serviço 1/(req/s)
queue0	0.1s
queue1	0.3s
queue2	0.5s

Com base nas configurações de carga e serviço, foram gerados os seguintes gráficos:

- a) Gráfico de barras mostrando o tempo médio para cada saída. Qual a configuração que produziu o maior tempo médio de permanência no sistema em cada saída? Qual a possível explicação?
- b) Gráfico de linha mostrando o tempo de estadia no sistema para

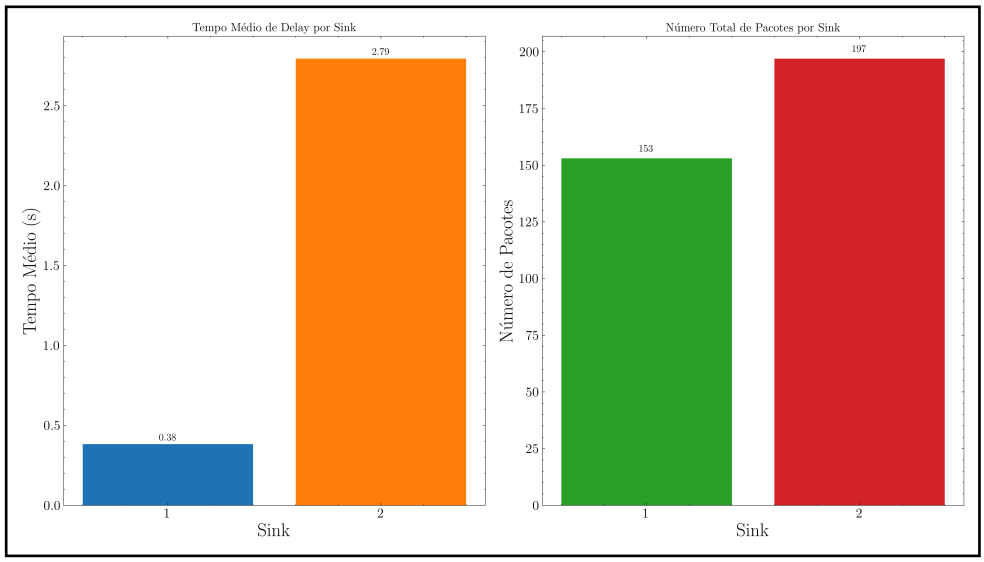
cada requisição, para o cenário de maior tempo médio observado no item (a);

- c) Histograma do tamanho da fila em cada subsistema (ver tutorial tic toc item 5.2). Apresente um histograma para cada fila para o melhor cenário em (a).

4. Geração de Gráficos

4.1. Tempo médio e número de pacotes por saída

Figura 2: Elaborada pelo Autor

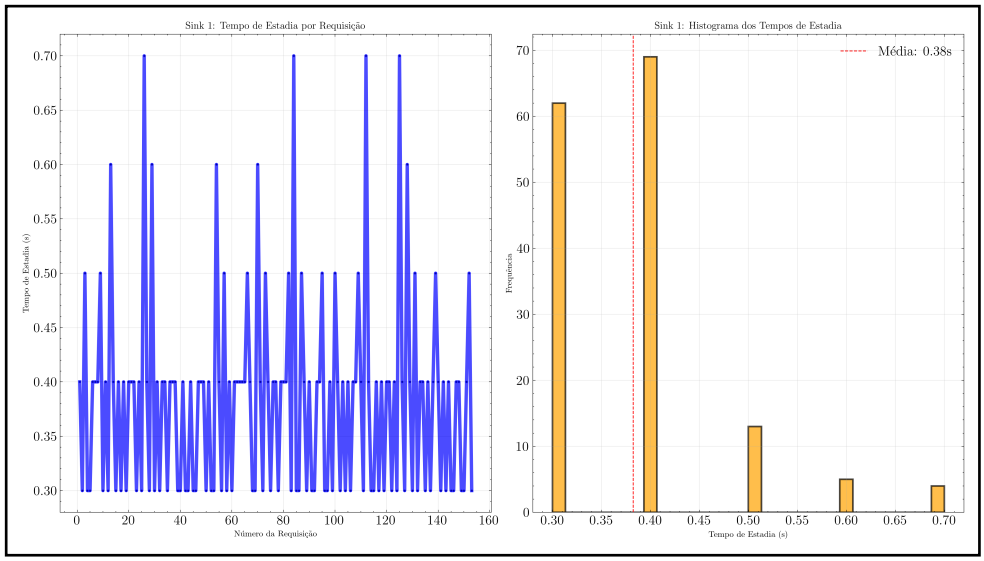


Tempo médio de permanência e número de pacotes por sink

O gráfico acima mostra, para cada configuração de carga, o tempo médio de permanência no sistema para cada saída (sink), bem como o número de pacotes processados. Observa-se que o sink 2 apresenta o maior tempo médio de permanência, indicando que este é o gargalo do sistema. Isso ocorre devido à configuração dos tempos de serviço e à distribuição de carga, que faz com que mais requisições se acumulem na fila 2.

4.2. Tempo de estadia no sistema para cada requisição

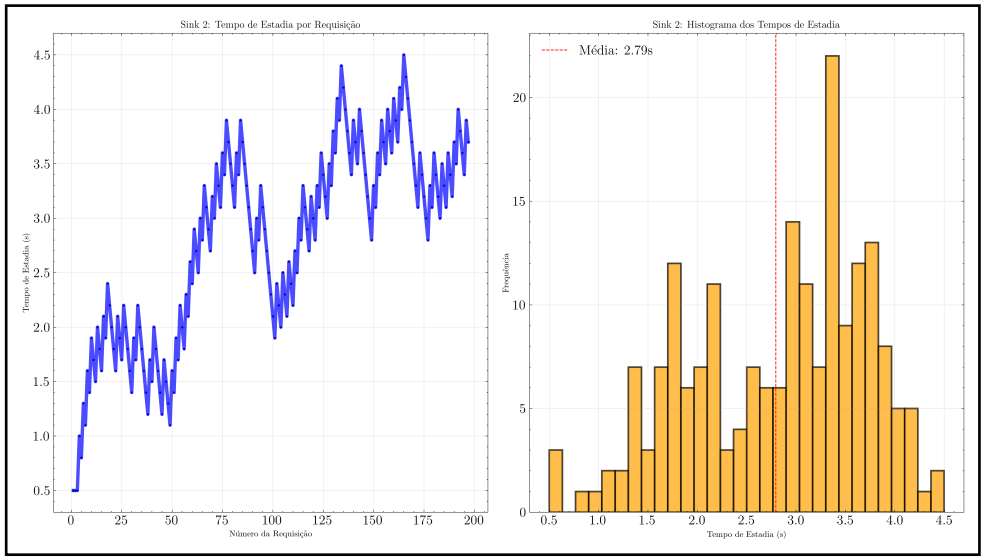
Figura 3: Elaborada pelo Autor



Tempo de estadia no sistema para cada requisição - Sink 1

O gráfico acima mostra a evolução do tempo de estadia no sistema para cada requisição que saiu pelo sink 1, no cenário de maior tempo médio observado. Nota-se uma variação relativamente pequena, indicando que o sistema estável para este caminho.

Figura 4: Elaborada pelo Autor

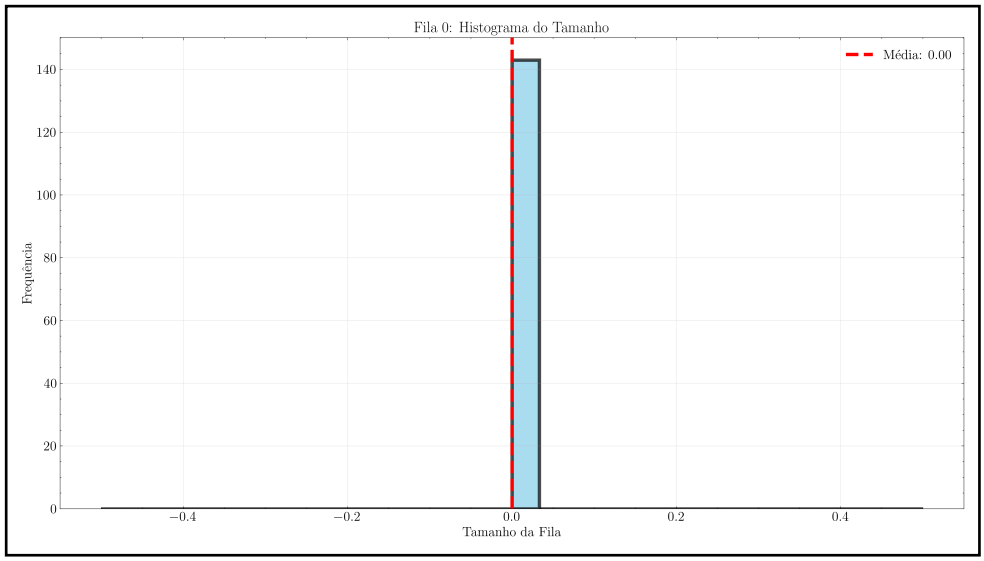


Tempo de estadia no sistema para cada requisição - Sink 2

Para o sink 2, observa-se uma maior dispersão dos tempos de estadia, com alguns picos elevados. Isso reforça a conclusão de que a fila 2 é o principal gargalo do sistema, acumulando requisições e aumentando o tempo de permanência.

4.3. Histogramas do tamanho das filas

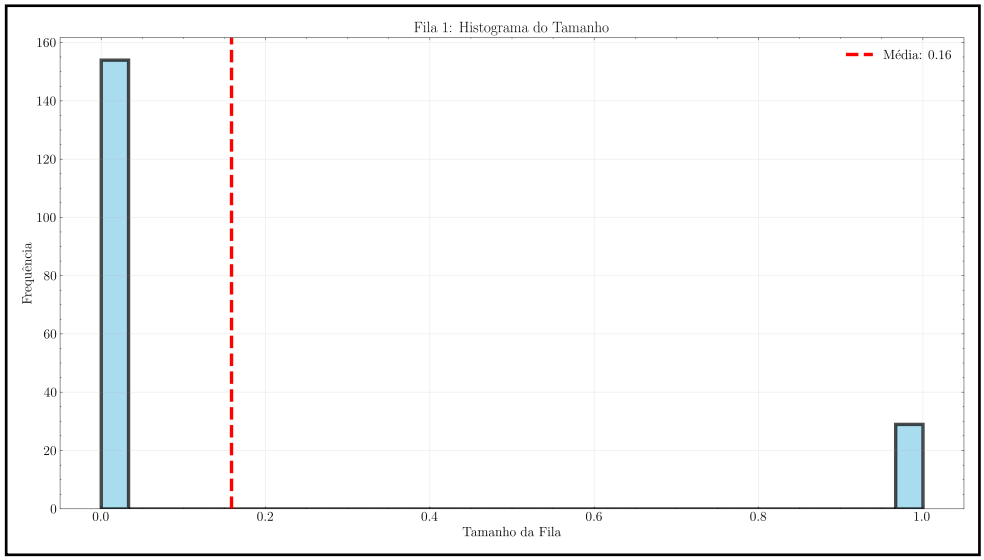
Figura 5: Elaborada pelo Autor



Histograma do tamanho da fila 0

O histograma da fila 0 mostra que a maior parte do tempo o tamanho da fila permanece baixo, indicando que este subsistema não é um gargalo.

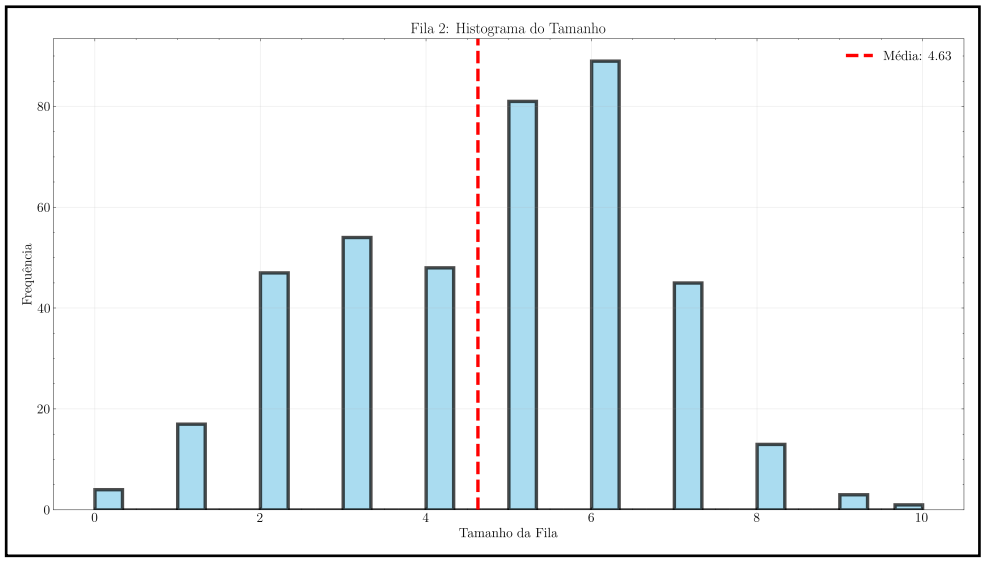
Figura 6: Elaborada pelo Autor



Histograma do tamanho da fila 1

A fila 1 apresenta uma distribuição semelhante, com poucos momentos de acúmulo significativo.

Figura 7: Elaborada pelo Autor

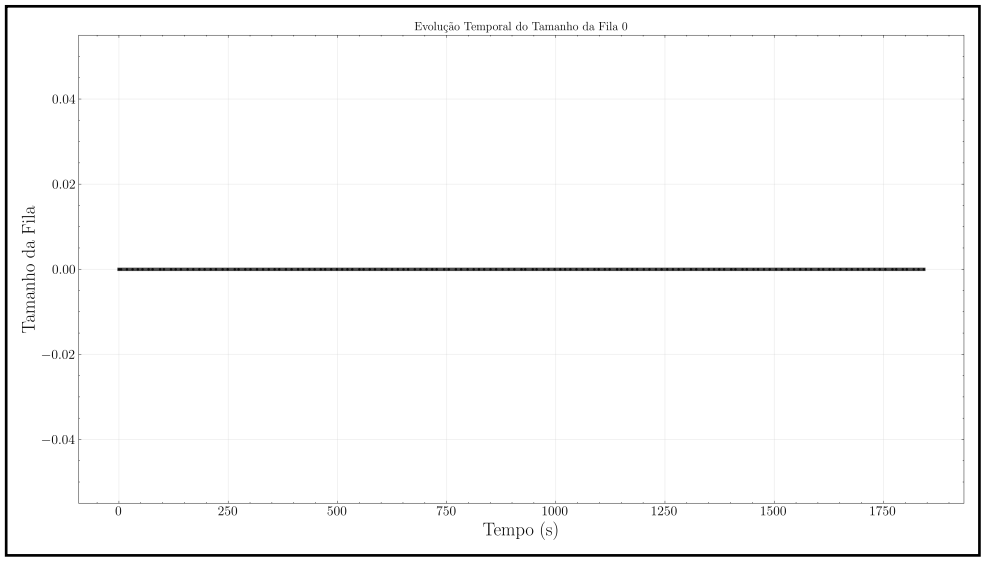


Histograma do tamanho da fila 2

O histograma da fila 2 mostra uma maior frequência de tamanhos elevados, evidenciando o acúmulo de requisições e confirmando que este é o principal ponto de congestionamento do sistema.

4.4. Evolução do tamanho das filas

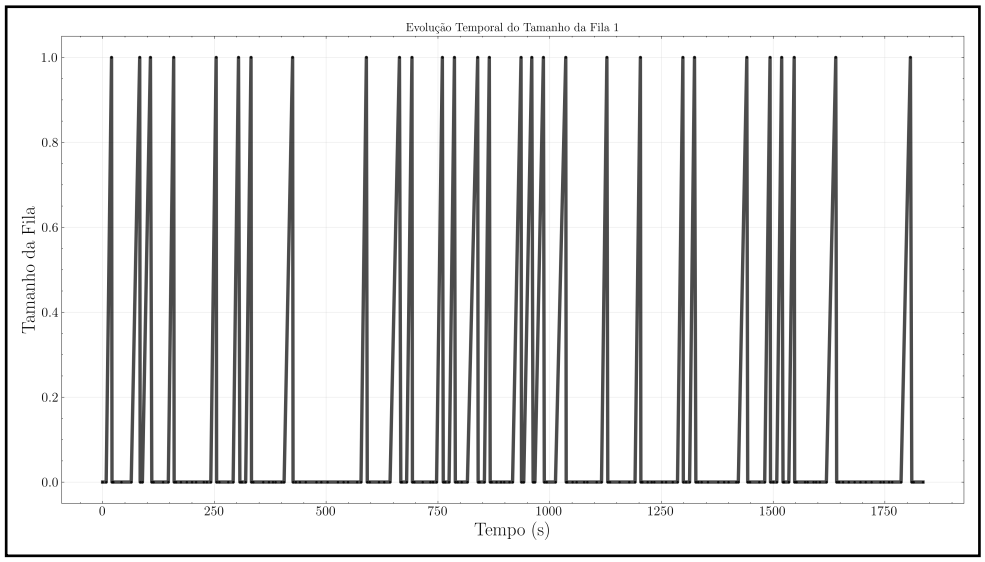
Figura 8: Elaborada pelo Autor



Evolução do tamanho da fila 0 ao longo do tempo

A fila 0 apresenta pequenas oscilações, sem acúmulo significativo.

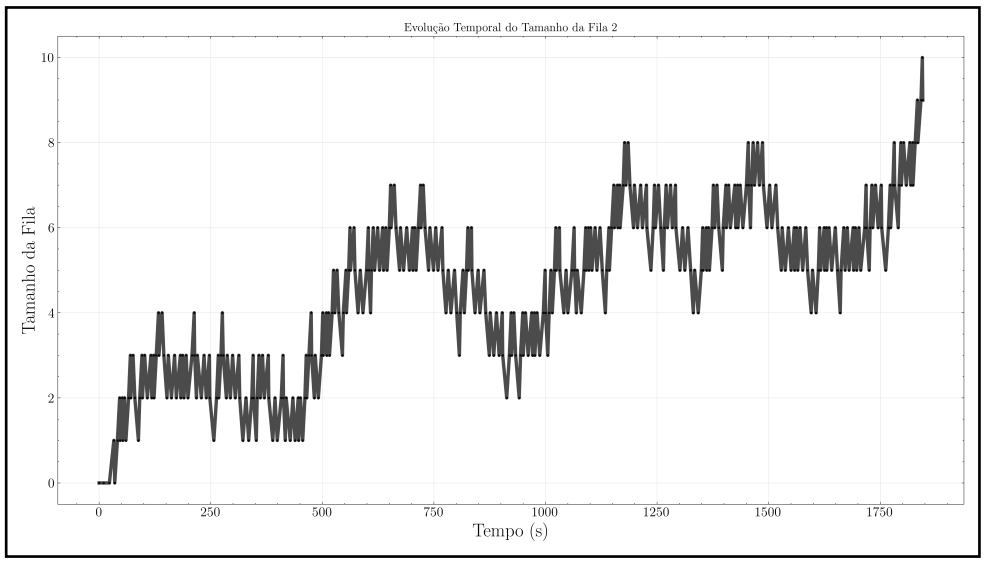
Figura 9: Elaborada pelo Autor



Evolução do tamanho da fila 1 ao longo do tempo

A fila 1 também se mantém estável, com poucas variações.

Figura 10: Elaborada pelo Autor



Evolução do tamanho da fila 2 ao longo do tempo

A fila 2 apresenta crescimento e oscilações mais acentuadas, reforçando a análise de que é o principal gargalo do sistema.

5. Conclusão

A análise dos resultados mostra que o desempenho do sistema é limitado principalmente pela fila 2, que acumula mais requisições e apresenta maiores tempos de permanência. Recomenda-se, para futuras otimizações, avaliar o aumento da capacidade de serviço deste subsistema ou a redistribuição das cargas para balancear melhor o fluxo de requisições.