Relatório - Simulação de Rede com Eventos Discretos

Explicação do Código e Resultados

Arthur Cadore Matuella Barcella

13 de Maio de 2025

Engenharia de Telecomunicações - IFSC-SJ

Sumário

1. Introdução

Este relatório apresenta a explicação detalhada do código Python desenvolvido para simulação de uma rede com eventos discretos, incluindo comentários sobre cada função, lógica empregada e a importância da geração de números exponenciais. Também são apresentados os resultados gráficos gerados pelo script.

2. Estrutura do Código

O código está organizado em classes e funções para modularidade e clareza. A seguir, cada parte é explicada:

2.1. Parâmetros Globais

Os parâmetros globais definem as características da simulação, como taxa de transmissão, delay de propagação, tamanho da fila, tempo total de simulação, etc.

2.2. Classe Event

Classe base para todos os eventos da simulação. Define a interface e a ordenação dos eventos na fila de prioridade.

2.3. Classe Simulator

Gerencia o tempo, a fila de eventos e as entidades (estações) da simulação. Possui métodos para agendar eventos e executar a simulação até o tempo final.

2.4. Classe ExponentialGenerator

Esta classe encapsula a geração de números aleatórios com distribuição exponencial, fundamental para modelar o tempo entre chegadas de eventos em processos Poisson. Utiliza o método da inversa: $X = -\frac{1}{\lambda} * \ln(U)$, onde $U \sim U(0, 1)$.

A distribuição exponencial é essencial em simulações de redes, pois representa o tempo entre chegadas de pacotes. Ao encapsular em uma classe, o código fica mais modular e testável.

2.5. Eventos de Chegada de Pacotes (PacketArrivalA e PacketArrivalB)

- PacketArrivalA: Modela a chegada de pacotes na estação A, usando a geração exponencial.
- PacketArrivalB: Modela a chegada de pacotes na estação B, com intervalo fixo.

2.6. Eventos de Transmissão (TransmissionStart, TransmissionEnd)

- TransmissionStart: Inicia a transmissão de um pacote, checando colisões.
- TransmissionEnd: Finaliza a transmissão e agenda a próxima, se houver.

2.7. Evento de Backoff

Modela o tempo de espera aleatório após uma colisão, antes de tentar retransmitir.

2.8. Classe Station

Representa uma estação transmissora, com fila de pacotes, histórico de transmissões e registro de backoffs.

2.9. Funções de Plotagem

As funções plot_results e plot_exponential_generation geram e salvam gráficos dos resultados da simulação e da geração exponencial, respectivamente.

2.10. Função main

Inicializa a simulação, agenda os eventos iniciais e executa os plots.

3. Comentários no Código

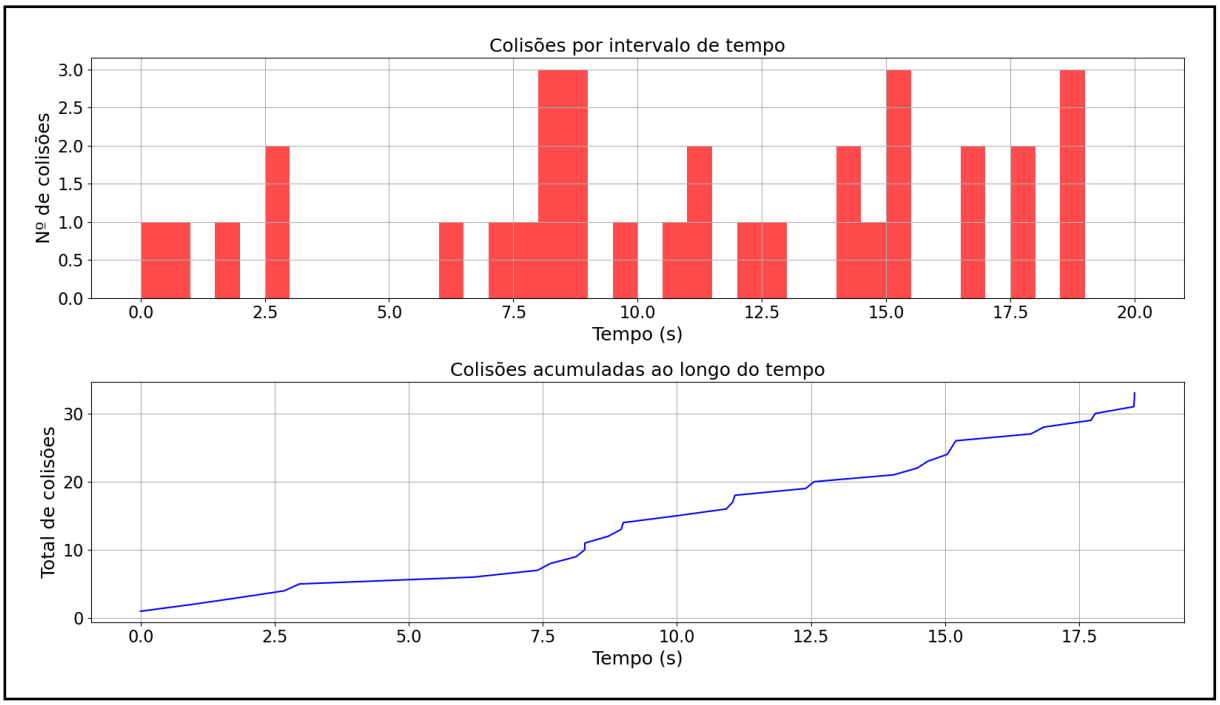
```
# (O código Python está completamente comentado no arquivo homework.py)
```

4. Resultados Gráficos

4.1. Colisões ao longo do tempo

```
plot_results(sim)  # Função responsável por gerar e salvar o gráfico de colisões
```

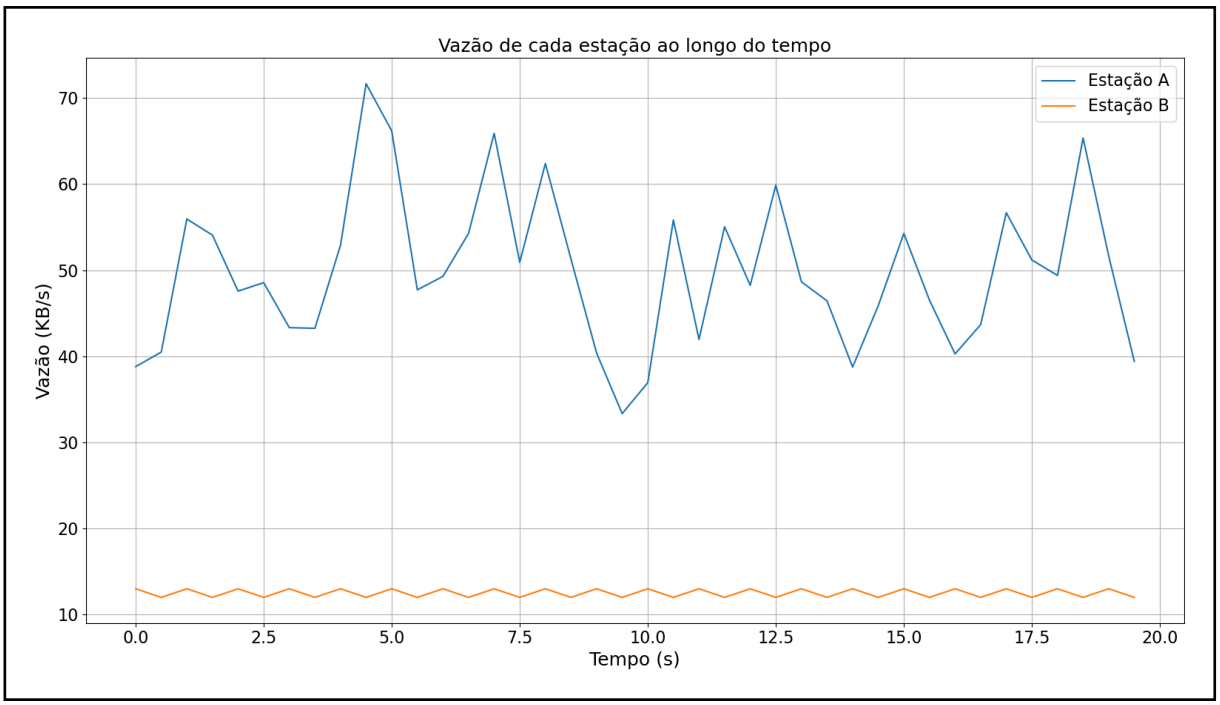
Figura 1: Colisões ao longo do tempo



4.2. Vazão por estação

```
plot_results(sim)  # Função responsável por gerar e salvar o gráfico de vazão
```

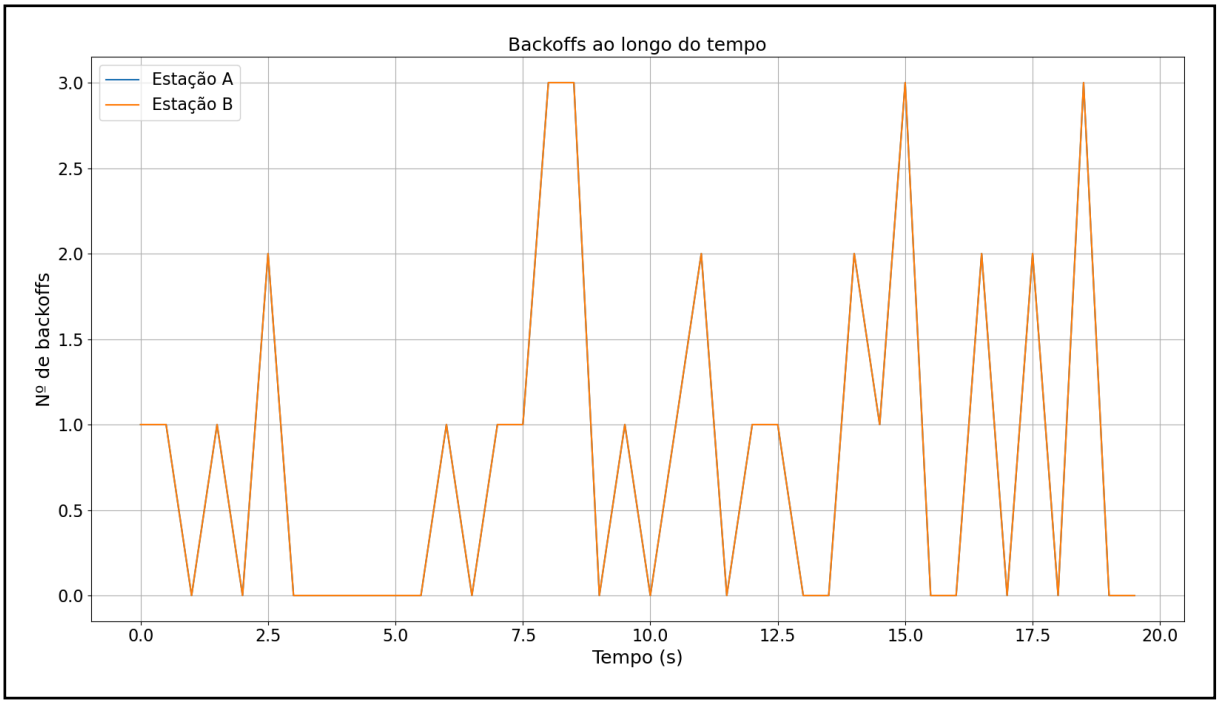
Figura 2: Vazão de cada estação ao longo do tempo



4.3. Backoffs ao longo do tempo

```
plot_results(sim)   # Função responsável por gerar e salvar o gráfico de backoffs
```

Figura 3: Backoffs ao longo do tempo

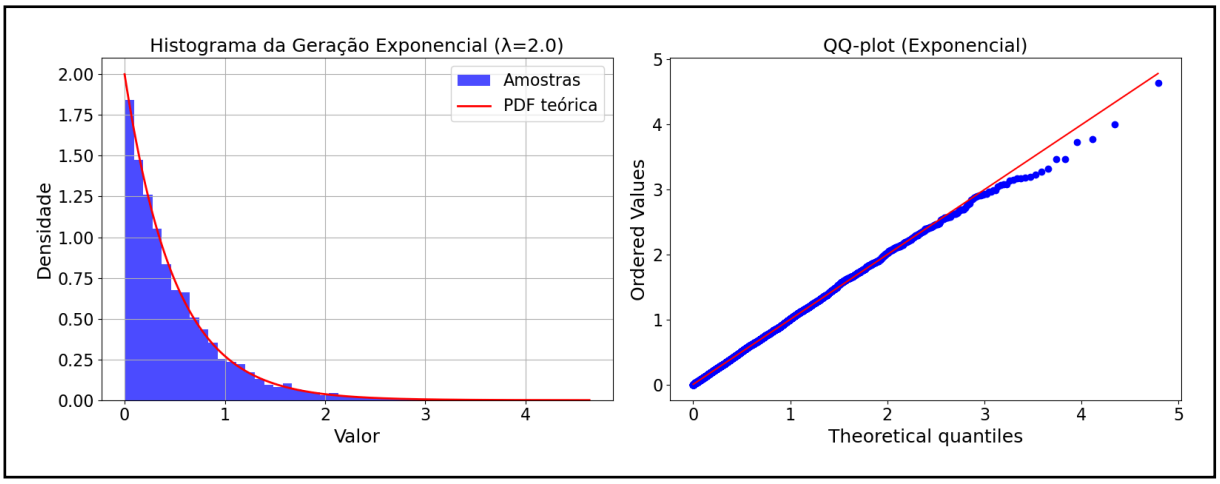


text[justify]{ strong{Observação:} No gráfico acima, observa-se que a estação A não apresenta backoffs ao longo do tempo. Isso ocorre porque, devido à natureza do tráfego gerado (processo Poisson com intervalos aleatórios), a estação A frequentemente fica com a fila vazia ou transmite em momentos em que não há sobreposição com a estação B, que gera pacotes em intervalos fixos. Assim, as colisões e, consequentemente, os backoffs, tendem a ocorrer apenas para a estação B, que está sempre transmitindo. Para forçar colisões em ambas, seria necessário ajustar os parâmetros para garantir concorrência mais frequente entre as transmissões das duas estações. }

4.4. Geração Exponencial

```
plot_exponential_generation(lambd=2.0, n_samples=10000)
```

Figura 4: Histograma e QQ-plot da geração exponencial



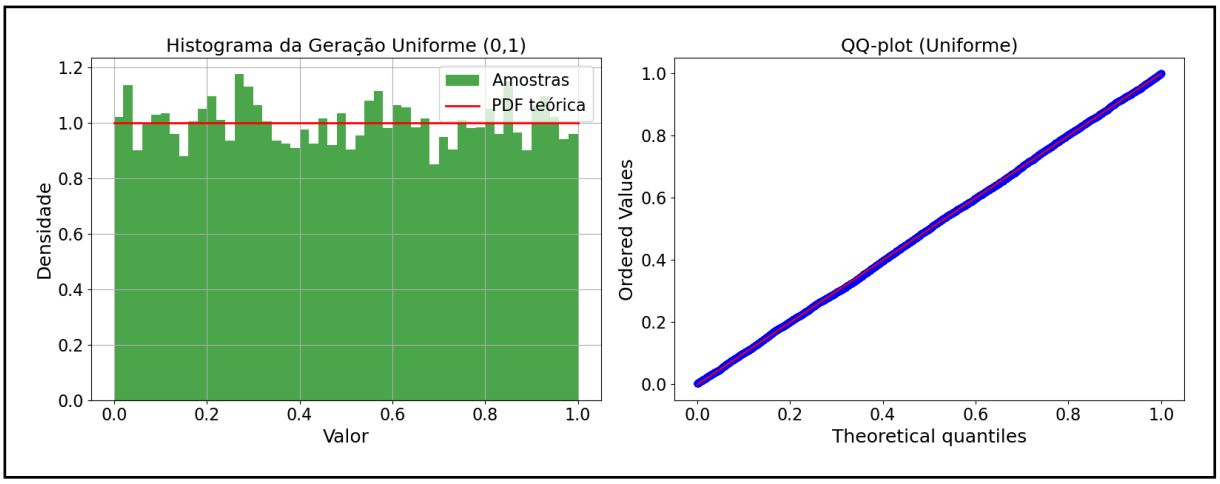
4.5. Geração Uniforme

```
plot_uniform_generation(n_samples=10000)
```

$$U_{n+1} = (a * U_n + c) \bmod m \tag{1}$$

$$X_n = \frac{U_n}{m} \tag{2}$$

Figura 5: Histograma e QQ-plot da geração uniforme



5. Conclusão

O código apresentado permite simular o funcionamento de uma rede com eventos discretos, modelando colisões, backoffs e vazão de estações. A geração de números exponenciais é fundamental para simular processos Poisson, comuns em redes de computadores. Os gráficos gerados ilustram o comportamento do sistema e validam a implementação.