MODELAÇÃO E DESEMPENHO DE REDES E SERVIÇOS

MINI-PROJETO 1

Eduardo Alves: nºmec 104179
Mariana Silva: nºmec 98392

eduardoalves@ua.pt
marianabarbara@ua.pt

Link Repositório GitHub

Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática
2025

Contents

List of Figures

1 Notas Gerais

- Todos os membros do grupo contribuíram de forma equilibrada e ativa para o desenvolvimento deste projeto, participando nas fases de planeamento, implementação e testes. O trabalho foi realizado de forma colaborativa, garantindo uma divisão justa das tarefas e a integração das ideias de todos os elementos.
- O repositório completo do projeto encontra-se disponível no seguinte link:
  https://github.com/marianabarbara/MDRS_Projects
- Foram realizados testes funcionais e de integração para garantir o correto funcionamento do sistema e a coerência entre os diferentes módulos.
- É importante salientar que, embora o repositório do projeto tenha permanecido privado durante o seu desenvolvimento, foi disponibilizado publicamente perto do periodo de entrega para efeitos de avaliação.

2 Introdução

O objetivo deste projeto é analisar e avaliar o desempenho de ligações ponto-a-ponto que suportam serviços de pacotes, recorrendo à modelação e simulação de diferentes cenários em *MATLAB*. Pretende-se compreender o impacto de diversos parâmetros de rede, tais como capacidade do canal, tamanho médio dos pacotes e taxa de chegada no desempenho global do sistema, nomeadamente no que diz respeito à perda média de pacotes, ao atraso médio e à eficiência da ligação.

O projeto será dividido em duas tarefas principais, cada uma com objetivos e componentes específicos. Em todas as simulações, serão realizadas 50 execuções independentes com um critério de paragem de $P = 100\,000$, sendo posteriormente calculados os valores estimados e os intervalos de confiança a 90%, apresentados sob a forma de gráficos de barras com barras de erro.

- **Task 1** Nesta fase será avaliado o desempenho de um sistema de transmissão ponto-a-ponto através do desenvolvimento e adaptação de simuladores orientados a eventos (`Simulator1`, `Simulator1A` e `Simulator1B`). O principal objetivo é analisar o comportamento do sistema sob diferentes condições de tráfego e políticas de enfileiramento, bem como comparar os resultados obtidos por simulação com valores teóricos derivados de modelos analíticos de filas (*queueing models*).
  - `1a`: Estimar, por simulação, a perda média de pacotes e o atraso médio para diferentes taxas de chegada λ, considerando pacotes com tamanho médio de $f = 1\,000\,000$ Bytes.
  - `1b`: Repetir as experiências anteriores considerando $f = 10\,000$ Bytes e analisar as diferenças observadas em relação à alínea anterior.
  - `1c`: Desenvolver uma nova versão do simulador, denominada `Simulator1A`, capaz de calcular separadamente os parâmetros de desempenho para pacotes com tamanhos específicos (64, 110 e 1518 Bytes).
  - `1d`: Utilizar o `Simulator1A` para estimar a perda média e o atraso médio, tanto para o conjunto total de pacotes como para cada um dos três tamanhos especiais.

- `1e`: Determinar, através de um modelo analítico M/G/1, os valores teóricos correspondentes e compará-los com os resultados obtidos por simulação.
- `1f`: Repetir as experiências de `1d` para $f = 10\,000$ Bytes e justificar as diferenças obtidas.
- `1g`: Desenvolver uma nova versão do simulador, `Simulator1B`, que introduz um sistema de prioridades baseado no tamanho dos pacotes.
- `1h`: Utilizar o `Simulator1B` para estimar o desempenho do sistema com as prioridades definidas, considerando $f = 1\,000\,000$ Bytes.
- `1i`: Repetir as experiências de `1h` para $f = 10\,000$ Bytes e analisar as diferenças observadas.
- `1j`: Comparar e justificar as diferenças entre os resultados obtidos com os simuladores `Simulator1A` e `Simulator1B`.

- **Task 2** Nesta etapa será avaliado o desempenho de um sistema que suporta múltiplos serviços de tráfego (dados e VoIP), utilizando e modificando o simulador `Simulator3` para incluir erros de transmissão na ligação. O objetivo é compreender o impacto da taxa de erro de bits e do número de fluxos VoIP no desempenho do sistema e na eficiência do canal.

  - `2a`: Desenvolver uma nova versão do simulador, `Simulator3A`, que considera a introdução de uma taxa de erro de bits (b) como parâmetro adicional.
  - `2b`: Estimar, por simulação, a perda média de pacotes para os serviços de dados e VoIP, variando o número de fluxos VoIP (n).
  - `2c`: Estimar o atraso médio de pacotes para cada serviço, nas mesmas condições anteriores.
  - `2d`: Calcular o débito total (*throughput*) da ligação em cada cenário e analisar o comportamento do sistema.
  - `2e`: Desenvolver um modelo analítico em *MATLAB* para determinar o valor teórico do *throughput* total e compará-lo com os resultados obtidos por simulação.

3 Task 1

3.1 Exercício 1a

3.1.1 Contexto

Neste exercício foi analisado o desempenho de um sistema de transmissão considerando um canal com capacidade $C = 10$ Mbps, uma fila com tamanho $f = 10^6$ Bytes e diferentes taxas de chegada de pacotes (λ) correspondentes a 1100, 1300, 1500, 1700 e 1900 pacotes por segundo (pps). O objetivo consiste em estimar, através de simulação, a perda média de pacotes (*Packet Loss*) e o atraso médio de pacotes (*Average Packet Delay*) para cada um dos valores de λ definidos.

Estes parâmetros são fundamentais para avaliar a qualidade de serviço (QoS) em redes de comunicação, uma vez que permitem perceber o impacto da carga oferecida no desempenho do sistema. Para cargas baixas espera-se que o sistema funcione de forma eficiente, com atrasos reduzidos e perdas nulas. À medida que a taxa de chegada se aproxima da capacidade do canal, o atraso tende a aumentar significativamente, podendo eventualmente ocorrer perdas de pacotes.

3.1.2 Implementação

A simulação foi desenvolvida considerando $P = 10^5$ pacotes transmitidos e $N = 50$ execuções independentes, de forma a obter intervalos de confiança de 90% ($\alpha = 0.1$). Os principais parâmetros definidos no código foram:

- $P = 10^5$ – número de pacotes a simular;
- $N = 50$ – número de execuções independentes;
- $\alpha = 0.1$ – nível de significância para o cálculo do intervalo de confiança;
- $C = 10$ Mbps – capacidade do canal;
- $f = 10^6$ Bytes – tamanho de ficheiro transmitido;
- $\lambda = [1100, 1300, 1500, 1700, 1900]$ pps – taxas de chegada dos pacotes.

Durante a execução, para cada valor de λ, foram calculadas as métricas de *Packet Loss* e *Average Packet Delay*, fazendo uso do Simulador1 disponibilizado pelo professor no âmbito da unidade curricular, este, tem como tarefa simular a chegada e saída de pacotes de um canal com fila de tamanho limitado.

O simulador mantém uma lista de eventos composta por chegadas (ARRIVAL) e partidas (DEPARTURE), atualizando o estado do canal e da fila de forma dinâmica. Quando um pacote chega e o canal está ocupado, ele é colocado na fila se houver espaço disponível; caso contrário, é descartado, contribuindo para o *Packet Loss*. Cada pacote transmitido tem o seu atraso calculado como a diferença entre o tempo de chegada e o tempo de saída, permitindo determinar tanto o **atraso médio** (*Average Packet Delay*) como o **atraso máximo** (*Maximum Packet Delay*).

Os resultados foram apresentados em **diagramas de barras** com barras de erro indicando os intervalos de confiança, conforme ilustrado pelo seguinte trecho de código:

```
fprintf('%-30s = %.2e +- %.2e\n', 'PacketLoss (%)', mediaPL, termPL);
fprintf('%-30s = %.2e +- %.2e\n', 'Average Packet Delay (ms)', mediaAPD, termAPD);
```

Este procedimento permite analisar a relação entre a taxa de chegada de pacotes e o desempenho do canal em termos de perda e atraso.

3.1.3 Resultados

As Figuras 1 e 2 apresentam, respetivamente, a evolução da perda de pacotes e do atraso médio em função da taxa de chegada de pacotes. E a figura 3 demonstra os resultados obtidos no formato da linha de comandos.

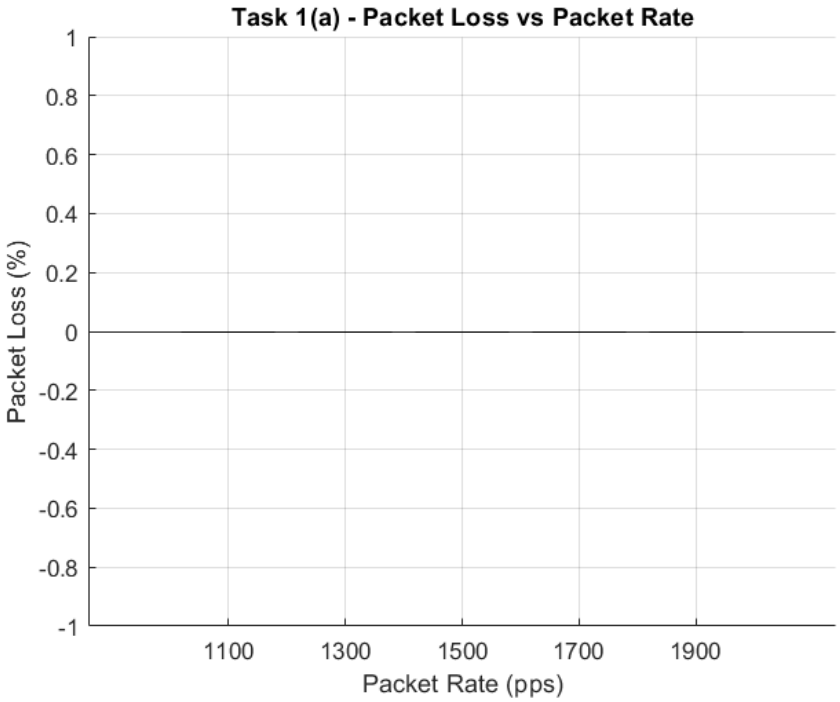

Figure 1: 1a) Perda média de pacotes em função da taxa de chegada (λ).

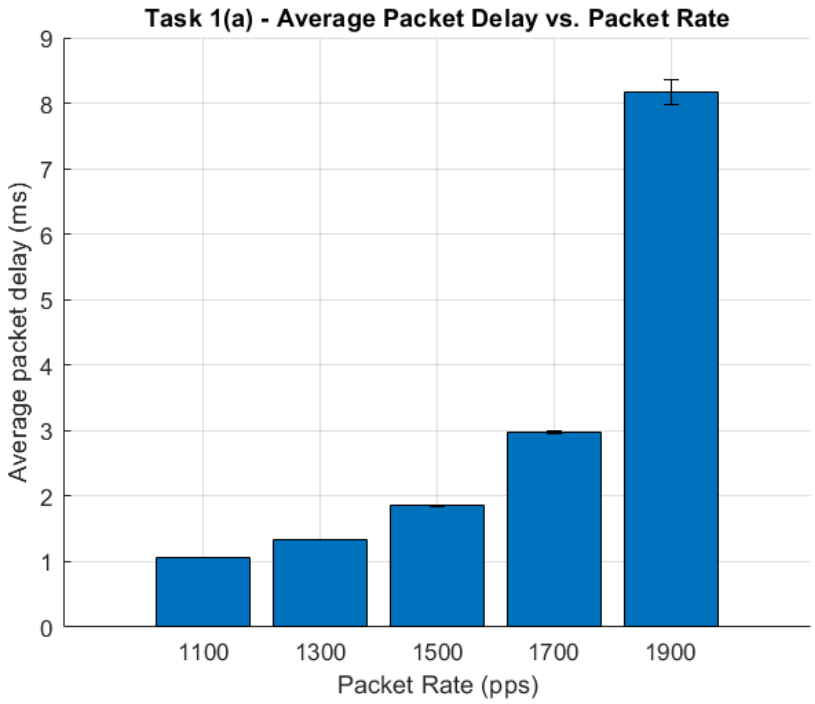

Figure 2: 1a) Atraso médio de pacotes em função da taxa de chegada (λ).

```
Command Window
>> task1
Valor de lambda: 1100
Packet Loss (%) = 0.00e+00 +- 0.00e+00
Average Packet Delay (ms) = 1.05e+00 +- 1.90e-03
Valor de lambda: 1300
Packet Loss (%) = 0.00e+00 +- 0.00e+00
Average Packet Delay (ms) = 1.34e+00 +- 4.33e-03
Valor de lambda: 1500
Packet Loss (%) = 0.00e+00 +- 0.00e+00
Average Packet Delay (ms) = 1.86e+00 +- 9.15e-03
Valor de lambda: 1700
Packet Loss (%) = 0.00e+00 +- 0.00e+00
Average Packet Delay (ms) = 3.00e+00 +- 2.73e-02
Valor de lambda: 1900
Packet Loss (%) = 0.00e+00 +- 0.00e+00
Average Packet Delay (ms) = 8.06e+00 +- 1.71e-01

Simulation ended!
>>
```

Figure 3: 1a) Resultados em formato de linha de comandos.

A análise dos resultados mostra que:

- Para todas as taxas de chegada de pacotes avaliadas, o sistema não apresenta perda de pacotes (0% de Packet Loss).
- É notável um acréscimo lento de atraso médio de envio de pacotes entre a taxa de chegada entre os valores de $\lambda = 1100$ (1.05ms) para $\lambda = 1300$ (1.34ms), o que demonstra um aumento curto de 0,29ms. De $\lambda = 1300$ (1,34ms) para $\lambda = 1500$ (1.86ms) o que demonstra um aumento curto de 0,52ms. E de $\lambda = 1500$ (1,86ms) para $\lambda = 1700$(3ms), o que já demonstra um aumento curto embora mais relevante de 1.14ms.
- A partir de $\lambda = 1900$ pps, observa-se um aumento significativo no atraso médio dos pacotes, atingindo os 8 ms. O que demonstra um aumento relativamente ao $\lambda = 1700$ de 5.06ms.

3.1.4 Conclusões

Com base nos resultados obtidos, conclui-se que:

- O tamanho da fila $f = 1 \times 10^6$ bytes é relativamente grande ($\approx 1\,\text{MB}$), logo, mesmo na taxa mais alta (1900 pacotes/segundo), a capacidade do ligação $C = 10\,\text{Mbps}$ é mais do que suficiente para atender aos pacotes sem ocorrer overflow. Portanto, nenhum pacote é descartado, e o simulador reporta $PacketLoss = 0$.
- Os resultados mostram o crescimento não linear do atraso à medida que a utilização cresce, o que está de acordo com a teoria das filas.
- O aumento abrupto do atraso médio a partir de 1900 pps demonstra a aproximação do limite de capacidade do canal, evidenciando a natureza não-linear da degradação de desempenho à medida que a carga se aproxima da capacidade máxima.
- Apesar da métrica do *Packet Loss* se manter nula em todos os casos simulados, o que indica que o sistema ainda não atingiu o ponto de congestão total, atraso já começa a demonstrar sinais de sobrecarga.

Desta forma, a simulação permitiu validar o impacto da carga oferecida no atraso médio e na perda de pacotes, comprovando que a saturação do canal tem um efeito exponencial no aumento do atraso, mesmo antes de ocorrer perda efetiva de pacotes.

3.2 Exercício 1b

3.2.1 Contexto

Neste exercício foram repetidas as simulações do Exercício 1a, alterando apenas o tamanho máximo da fila para $f = 10.000$ Bytes, mantendo a capacidade do canal $C = 10$ Mbps e as mesmas taxas de chegada $\lambda = [1100, 1300, 1500, 1700, 1900]$ pps. O objetivo é avaliar o impacto da diminuição do tamanho da fila no desempenho do sistema, analisando a perda média de pacotes (*Packet Loss*) e o atraso médio (*Average Packet Delay*).

3.2.2 Implementação

O código utilizado é idêntico ao do Exercício 1a, variando apenas o valor de f. Foram novamente realizadas $N = 50$ execuções para cada valor de λ, simulando $P = 10^5$ pacotes por execução, e calculando intervalos de confiança de 90% para as métricas de desempenho.

As métricas obtidas foram:

- **Perda média de pacotes (%)**: proporção de pacotes descartados devido à fila cheia.
- **Atraso médio (ms)**: tempo médio entre a chegada e o envio de um pacote.

Ambas relativas à taxa de envio de pacotes (λ).

3.2.3 Resultados

Os resultados experimentais obtidos foram os seguintes:

As Figuras 4 e 5 ilustram os resultados obtidos em formato de gráfico e apresentam a evolução da perda e do atraso médio em função da taxa de chegafa λ e a figura 6 mostra os resultados obtidos na consola.

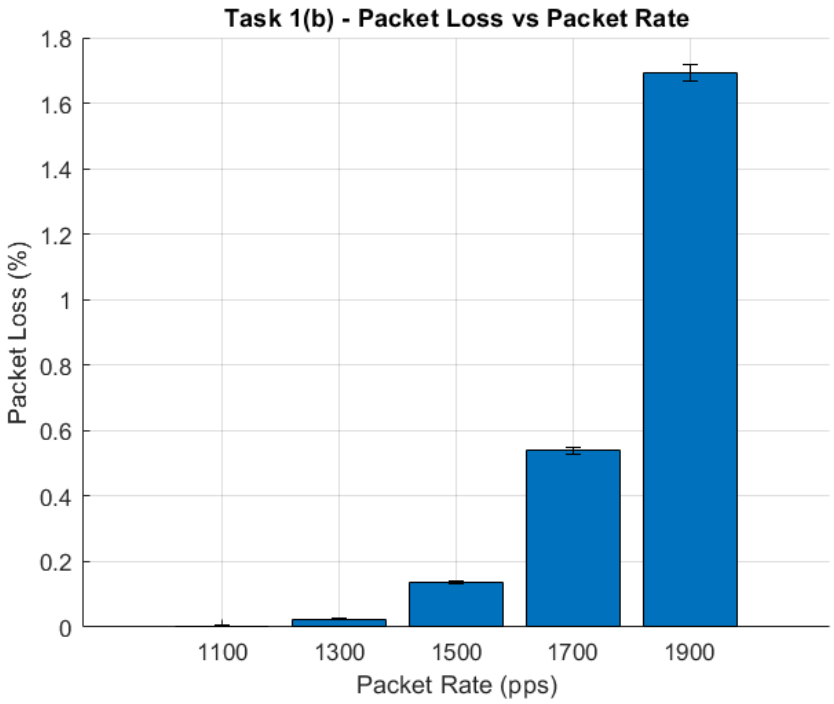

Figure 4: 1b) Perda de pacotes em função da taxa de chegada (λ) para $f = 10.000$ Bytes.

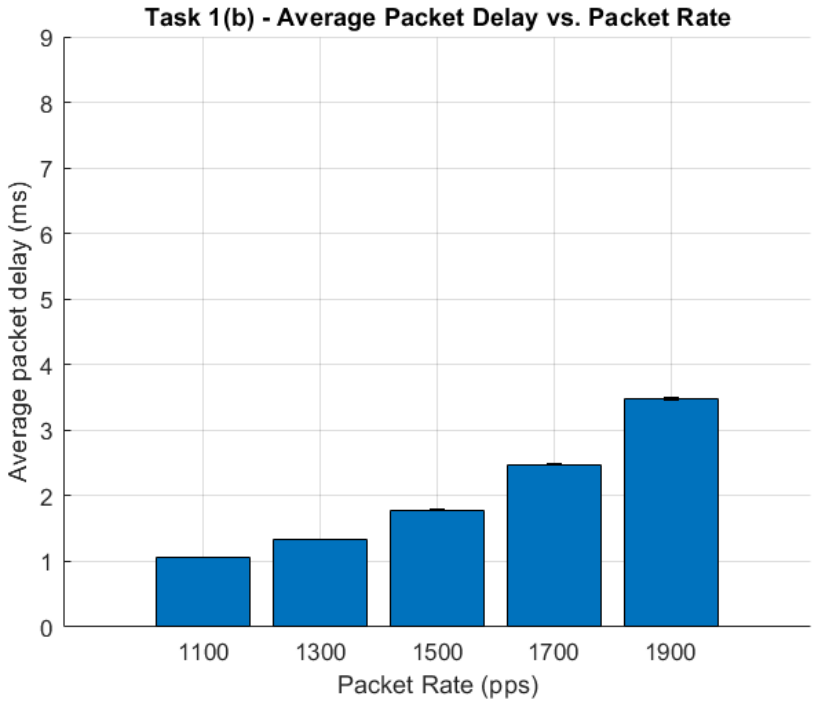

Figure 5: 1b) Atraso médio de pacotes em função da taxa de chegada (λ) para $f = 10.000$ Bytes.

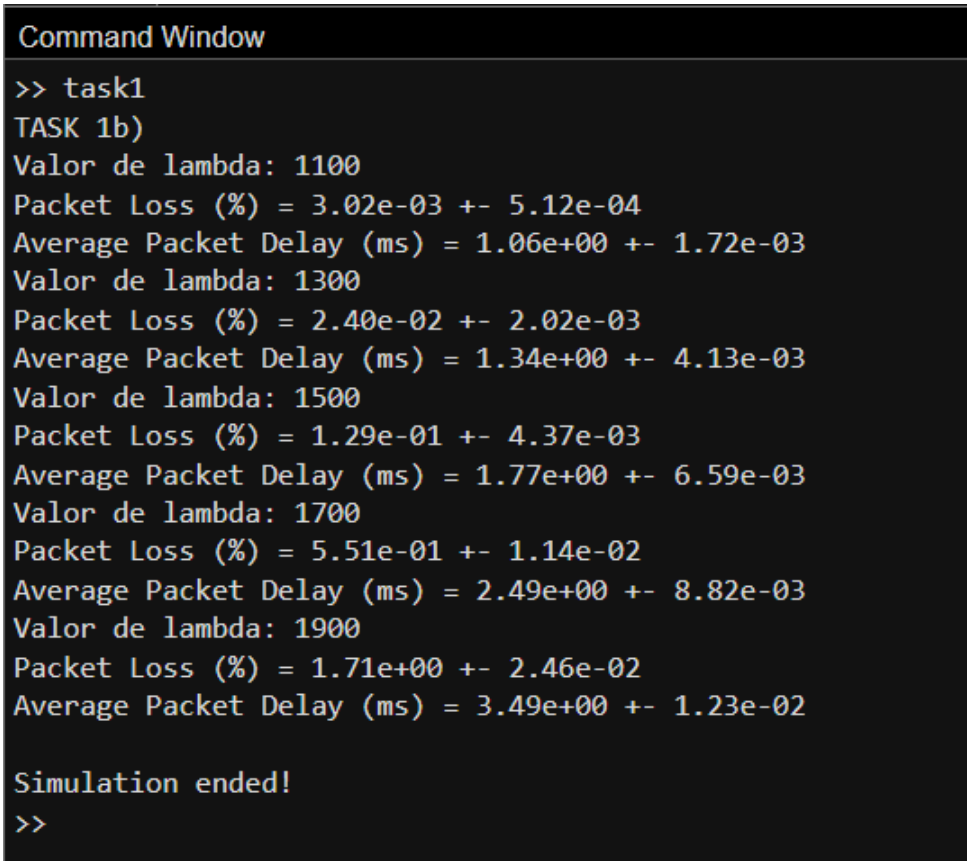

Figure 6: 1b) Resultados na linha de comandos.

3.2.4 Análise dos resultados

Observa-se que, com a fila muito menor ($f = 10.000$ Bytes), o sistema sofre uma degradação significativa de desempenho à medida que λ aumenta:

- Para $\lambda = 1100$ pps, a perda de pacotes é praticamente nula e o atraso médio é relativamente baixo (cerca de 1 ms), indicando um sistema estável.
- A partir de $\lambda = 1300$ pps, a **perda de pacotes começa a demonstrar um pequeno aumento**. Ao atingir $\lambda = 1500$ a perda de pacotes sobe consideravelmente e para $\lambda = 1900$ sobe já muito acentuadamente.
- Para $\lambda = 1900$ pps, a perda atinge cerca de 1.71%, e o atraso médio sobe para aproximadamente 3.49 ms, mostrando um estado de congestionamento acentuado.

Este comportamento confirma que um **tamanho de fila reduzido limita fortemente a capacidade do sistema de absorver picos de tráfego**, levando à rejeição de pacotes e ao aumento dos tempos de espera.

3.2.5 Comparação com o Exercício 1a

Em comparação com o Exercício 1a (onde $f = 1.000.000$ Bytes), verifica-se uma diferença clara:

- No 1a, o sistema manteve perdas nulas e atrasos baixos até $\lambda = 1900$ pps.

- No 1b, as perdas e atrasos aumentam de forma acentuada para os mesmos valores de λ.

Esta diferença é consequência direta da redução da capacidade de armazenamento. Com um valor de f menor, a fila satura mais rapidamente, o que leva a um crescimento rápido das perdas e do atraso médio. Além disso, o aumento do atraso antes da perda é típico de sistemas de filas com capacidades mais reduzidas, pois os pacotes começam a acumular até ao ponto de rejeição.

3.2.6 Conclusões

Conclui-se que a redução do tamanho da fila tem um impacto negativo na eficiência do sistema:

- O atraso médio aumenta de forma notável mesmo para taxas moderadas de chegada.
- A perda de pacotes deixou de ser nula e torna-se notável desde inicio.

Estes resultados demonstram a importância do dimensionamento adequado em sistemas de comunicação, de forma a equilibrar a utilização da rede e o tempo de resposta.

3.3 Exercício 1c (Simulator1A)

3.3.1 Contexto

Nesta secção foi desenvolvido um novo simulador, denominado **Simulator1A**, com base no código e na estrutura do *Simulator1*. O objetivo desta versão é estimar seis parâmetros de desempenho adicionais, correspondentes à **perda média de pacotes (PL)** e ao **atraso médio dos pacotes transmitidos (APD)** para três tamanhos específicos de pacotes: **64, 110 e 1518 Bytes**. Desta forma, além dos parâmetros gerais de desempenho, o simulador passa também a calcular individualmente o comportamento da rede para pacotes de tamanhos distintos, permitindo uma análise mais detalhada do impacto do tamanho do pacote no desempenho do sistema.

3.3.2 Implementação

O código do *Simulator1A* baseia-se na estrutura do *Simulator1*, mas com a introdução de variáveis e cálculos adicionais para monitorizar separadamente as estatísticas associadas a pacotes de 64, 110 e 1518 Bytes. O código deste simulador na sua totalidade está presente aqui! No entanto, as principais alterações introduzidas foram as seguintes:

1. **Criação de novos contadores de estatísticas por tamanho de pacote.**

   Foram adicionadas variáveis para contabilizar o número total de pacotes gerados, transmitidos, perdidos e o somatório dos atrasos, separadamente para cada tamanho de pacote:

   ```
   DELAYS = 0;
   DELAYS_64 = 0;
   DELAYS_110 = 0;
   DELAYS_1518 = 0;

   TOTALPACKETS = 0;
   TOTALPACKETS_64 = 0;
   TOTALPACKETS_110 = 0;
   TOTALPACKETS_1518 = 0;

   LOSTPACKETS = 0;
   LOSTPACKETS_64 = 0;
   LOSTPACKETS_110 = 0;
   LOSTPACKETS_1518 = 0;

   TRANSPACKETS = 0;
   TRANSPACKETS_64 = 0;
   TRANSPACKETS_110 = 0;
   TRANSPACKETS_1518 = 0;
   ```

   Estas variáveis permitem distinguir os resultados por categoria de tamanho de pacote, sem interferir na lógica geral do simulador.

2. **Contabilização dos pacotes totais por tipo no evento *ARRIVAL*.**

   Logo que um pacote chega, o seu tamanho é verificado e incrementa-se o contador correspondente:

```
TOTALPACKETS = TOTALPACKETS + 1;
if PacketSize == 64
    TOTALPACKETS_64 = TOTALPACKETS_64 + 1;
elseif PacketSize == 110
    TOTALPACKETS_110 = TOTALPACKETS_110 + 1;
elseif PacketSize == 1518
    TOTALPACKETS_1518 = TOTALPACKETS_1518 + 1;
end
```

3. **Contabilização de perdas por tipo de pacote.**

   Quando a fila está cheia e um pacote é descartado, o simulador identifica o seu tamanho e incrementa o contador de perdas específico:

```
LOSTPACKETS = LOSTPACKETS + 1;
if PacketSize == 64
    LOSTPACKETS_64 = LOSTPACKETS_64 + 1;
elseif PacketSize == 110
    LOSTPACKETS_110 = LOSTPACKETS_110 + 1;
elseif PacketSize == 1518
    LOSTPACKETS_1518 = LOSTPACKETS_1518 + 1;
end
```

4. **Registo dos atrasos e pacotes transmitidos por tipo no evento *DEPARTURE*.**

   Após a transmissão de cada pacote, o simulador regista o seu atraso e atualiza as contagens de pacotes transmitidos por tipo:

```
if PacketSize == 64
    TRANSPACKETS_64 = TRANSPACKETS_64 + 1;
    DELAYS_64 = DELAYS_64 + (Clock - ArrInstant);
elseif PacketSize == 110
    TRANSPACKETS_110 = TRANSPACKETS_110 + 1;
    DELAYS_110 = DELAYS_110 + (Clock - ArrInstant);
elseif PacketSize == 1518
    TRANSPACKETS_1518 = TRANSPACKETS_1518 + 1;
    DELAYS_1518 = DELAYS_1518 + (Clock - ArrInstant);
end
```

5. **Cálculo dos novos parâmetros de desempenho.**

   No final da simulação, foram adicionadas as expressões para calcular a perda e atraso médio de cada tipo de pacote:

```
PL_64   = 100 * LOSTPACKETS_64 / TOTALPACKETS_64;
APD_64  = 1000 * DELAYS_64 / TRANSPACKETS_64;

PL_110  = 100 * LOSTPACKETS_110 / TOTALPACKETS_110;
APD_110 = 1000 * DELAYS_110 / TRANSPACKETS_110;

PL_1518 = 100 * LOSTPACKETS_1518 / TOTALPACKETS_1518;
APD_1518= 1000 * DELAYS_1518 / TRANSPACKETS_1518;
```

Estas métricas complementam as globais já calculadas pelo simulador original.

3.4 Exercício 1d

3.4.1 Contexto

Neste exercício, pretende-se utilizar a função `Simulator1A` desenvolvida no exercício anterior para estimar, por simulação, os parâmetros de desempenho **perda média de pacotes** (*Packet Loss*) e **atraso médio dos pacotes** (*Average Packet Delay*). A simulação deve considerar os três tamanhos de pacotes adicionados no exercício anterior (64, 110 e 1518 Bytes) e também o conjunto total de pacotes.

Os parâmetros de simulação são:

$$C = 10\,\text{Mbps}, \quad f = 1{,}000{,}000\,\text{Bytes}, \quad \lambda = 1900\,\text{pps}$$

O critério de paragem é de $P = 10^5$ pacotes transmitidos e foram realizadas $N = 50$ execuções independentes para calcular intervalos de confiança a 90%.

3.4.2 Implementação

Como já concretizado previamente, o simulador **Simulator1A** foi desenvolvido de forma a permitir a recolha separada de estatísticas para três tamanhos específicos de pacotes: 64, 110 e 1518 Bytes.

Neste exercício, o código principal tem como objetivo utilizar o `Simulator1A` para comparar o desempenho do sistema entre os diferentes tamanhos de pacotes, mantendo as restantes condições fixas.

Em cada iteração da simulação, o `Simulator1A` devolve oito métricas principais:

- PL e APD: perda e atraso médios considerando todos os pacotes;
- $PL_{64}, PL_{110}, PL_{1518}$: perdas médias específicas por tamanho de pacote;
- $APD_{64}, APD_{110}, APD_{1518}$: atrasos médios específicos por tamanho de pacote;

Após todas as execuções, o código calcula a média e o intervalo de confiança a 90% para cada métrica.

Os resultados são então organizados em dois vetores:

- `PL_means` e `PL_terms` — contêm as médias e os intervalos de confiança da perda de pacotes;
- `APD_means` e `APD_terms` — contêm as médias e intervalos correspondentes ao atraso médio.

Por fim, os resultados são apresentados graficamente, como podermos analisar de seguida.

3.4.3 Resultados

A Figura 7 apresenta as perdas médias de pacotes e a Figura 8 mostra os atrasos médios. A figura 9 mostra os resultados obtidos na linha de comandos.

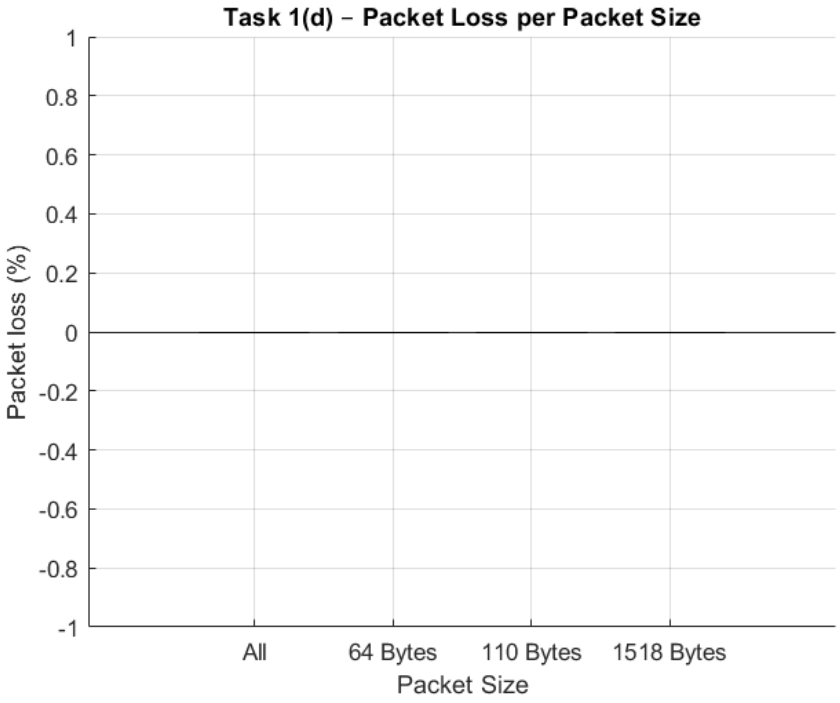

Figure 7: 1d) Perda média de pacotes para cada tipo de pacote.

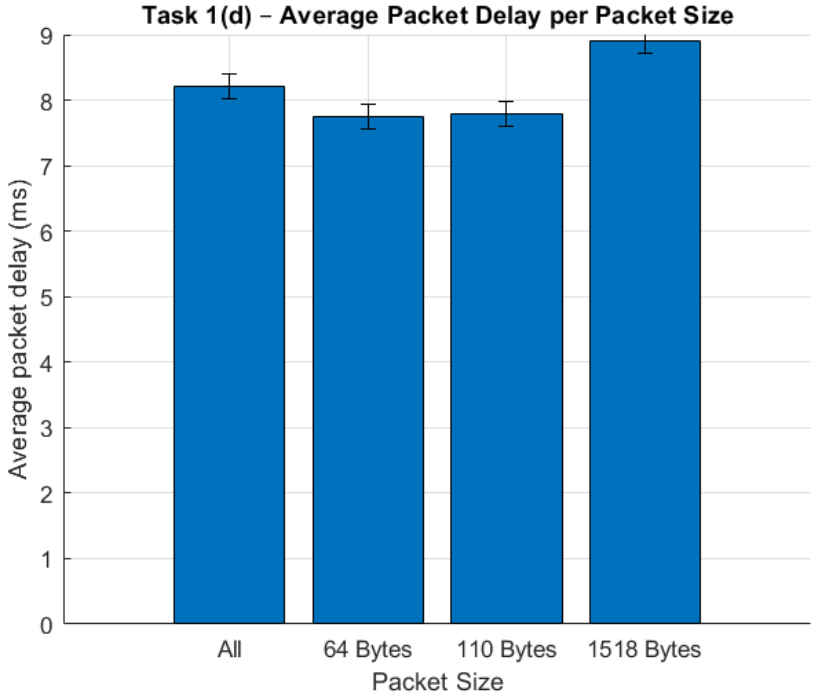

Figure 8: 1d) Atraso médio dos pacotes para cada tipo de pacote.

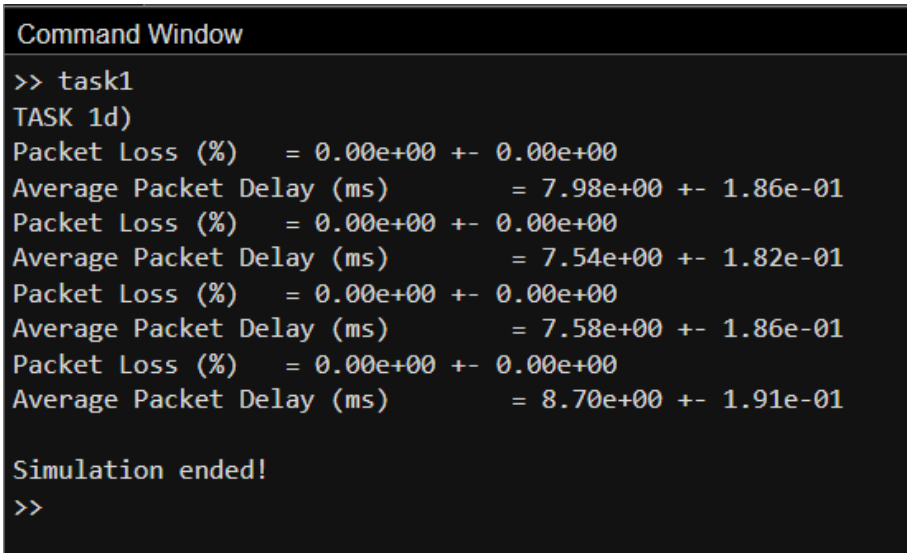

Figure 9: 1d) Resultados apresentados na linha de comandos.

Análise e conclusões dos Resultados

Verifica-se que a **perda média de pacotes é nula** para todos os tamanhos, o que se justifica pelo grande tamanho de $f = 1\,\text{MB}$ e pela capacidade da ligação ($C = 10\,\text{Mbps}$), que permitem processar todas as chegadas sem saturação.

Relativamente ao atraso médio, este mantém-se em torno de 8 ms, sendo ligeiramente superior para pacotes maiores (1518 Bytes), devido ao maior tempo de transmissão. Este comportamento é consistente com a teoria de filas M/G/1, onde o atraso total é composto pelo tempo de transmissão e o tempo médio de espera na fila, sendo o primeiro dominante neste regime. Assim, conclui-se que para os parâmetros definidos, o sistema opera sem perdas e com atrasos baixos e estáveis, refletindo um regime de funcionamento eficiente e não congestionado.

3.5 Exercício 1e

3.5.1 Contexto

Neste exercício pretende-se determinar, de forma **teórica**, os valores do atraso médio de pacotes e da perda média de pacotes para o sistema considerado no exercício **1.d**, modelando-o como uma fila **M/G/1**. Este modelo representa um sistema com chegadas de pacotes segundo um processo de Poisson (**M**), tempos de serviço genéricos (**G**) e um único servidor (**1**), assumindo uma fila de espera **infinita**. Consequentemente, neste modelo **não existe perda de pacotes**, pois todos os pacotes são eventualmente servidos. O objetivo é comparar os resultados teóricos com os obtidos por simulação no exercício 1.d e analisar a adequação do modelo M/G/1 à situação estudada.

3.5.2 Implementação

O sistema é modelado como uma fila M/G/1, onde o atraso médio teórico (W) é calculado pela fórmula:

$$W = \frac{\lambda E[S^2]}{2(1 - \lambda E[S])}$$

onde:

- λ é a taxa média de chegada (em pacotes/s);
- $E[S]$ é o tempo médio de serviço;
- $E[S^2]$ é o segundo momento do tempo de serviço.

A distribuição dos tamanhos de pacotes é definida da seguinte maneira:

$$P(64) = 0.19, \quad P(110) = 0.23, \quad P(1518) = 0.17,$$

e uma probabilidade uniforme é atribuída aos restantes tamanhos entre 65 e 1517 bytes.

Definição dos parâmetros iniciais

```
P = 1e5;
N = 50;
C = 10;
f = 1e6;
lambda = 1900;
alfa = 0.1;
```

Nesta secção definem-se os parâmetros principais, que são os mesmos do exercício 1d).

Definição do vetor de tamanhos e probabilidades

```
x = 64:1518;
prob_elements = (1 - 0.19 - 0.23 - 0.17) / ((109-65+1) + (1517-111+1));
```

O vetor `x` contém todos os tamanhos de pacotes possíveis (entre 64 e 1518 Bytes). Os tamanhos 64, 110 e 1518 têm probabilidades fixas de 0.19, 0.23 e 0.17, respetivamente. O restante (0.41) é distribuído uniformemente pelos intervalos de 65–109 e 111–1517 Bytes.

Cálculo do tamanho médio e tempo médio de serviço

```
avg_packet_size = 0.19*64 + 0.23*110 + 0.17*1518 + ...
                  sum(65:109)*prob_elements + sum(111:1517)*prob_elements;
avg_time = (avg_packet_size * 8) / (C * 10^6);
```

- `avg_packet_size` representa o valor esperado do tamanho de um pacote $E[\text{PacketSize}]$;
- `avg_time` converte este valor para tempo de serviço médio $E[S] = \frac{8E[\text{PacketSize}]}{C \cdot 10^6}$, em segundos.

Cálculo de $E[S]$ e $E[S^2]$

```
S = x .*8 / (C * 10^6);
S2 = S.^2;

for i = 1:length(x)
    if i == 1
       p = 0.19;
    elseif i == 110-64+1
       p = 0.23;
    elseif i == 1518-64+1
        p = 0.17;
    else
       p = prob_elements;
    end
    S(i) = S(i) * p;
    S2(i) = S2(i) * p;
end

ES = sum(S);
ES2 = sum(S2);
```

Cada elemento do vetor `S` contém o tempo de serviço correspondente a cada tamanho de pacote. O ciclo `for` pondera cada valor de `S` e `S2` pela sua probabilidade, permitindo obter:

$$E[S] = \sum_i p_i S_i \quad \text{e} \quad E[S^2] = \sum_i p_i S_i^2$$

Aplicação da fórmula de Pollaczek–Khinchine

```
W = (lambda * ES2) / (2*(1 - lambda*ES)) + ES;
Wq = (lambda * ES2) / (2*(1 - lambda*ES));
```

- `Wq` representa o tempo médio de espera na fila $W_q = \frac{\lambda E[S^2]}{2(1-\lambda E[S])}$;
- `W` é o tempo médio total no sistema $W = W_q + E[S]$.

Antes da aplicação da fórmula, deve-se garantir que o sistema é estável.

Atrasos médios por tamanho de pacote

```
S64 = (64 * 8) / (C * 1e6);
S110 = (110 * 8) / (C * 1e6);
S1518 = (1518 * 8) / (C * 1e6);

W64 = Wq + S64;
W110 = Wq + S110;
W1518 = Wq + S1518;
```

O atraso total para cada tipo de pacote é a soma do tempo médio na fila com o respetivo tempo de serviço. Como o modelo M/G/1 assume que todos os pacotes esperam, em média, o mesmo tempo na fila, apenas o tempo de serviço diferencia cada classe.

Cálculo do throughput teórico

```
TT = lambda * avg_packet_size * 8 / 10^6;
```

O throughput é obtido como:

$$TT = \frac{\lambda \times E[\text{PacketSize}] \times 8}{10^6} \quad [\text{Mb/s}]$$

Apresentação dos resultados

```
fprintf('PL_all (%%)      = 0.0000\n');
fprintf('PL_64B (%%)      = 0.0000\n');
fprintf('PL_110B (%%)     = 0.0000\n');
fprintf('PL_1518B (%%)    = 0.0000\n');
fprintf('Delay all (ms)   = %.4f\n', W*1000);
fprintf('Delay 64B (ms)   = %.4f\n', W64*1000);
fprintf('Delay 110B (ms)  = %.4f\n', W110*1000);
fprintf('Delay 1518B (ms) = %.4f\n', W1518*1000);
fprintf('Throughput (Mb/s)= %.4f\n', TT);
```

Como o modelo M/G/1 assume uma fila infinita, não ocorre perda de pacotes, resultando em `PL = 0%`. Os valores de atraso e throughput são convertidos para milissegundos e megabits por segundo, respetivamente, para comparação direta com os resultados de simulação.

3.5.3 Resultados e Discussão

A execução do script deste exercício produziu os seguintes resultados teóricos:

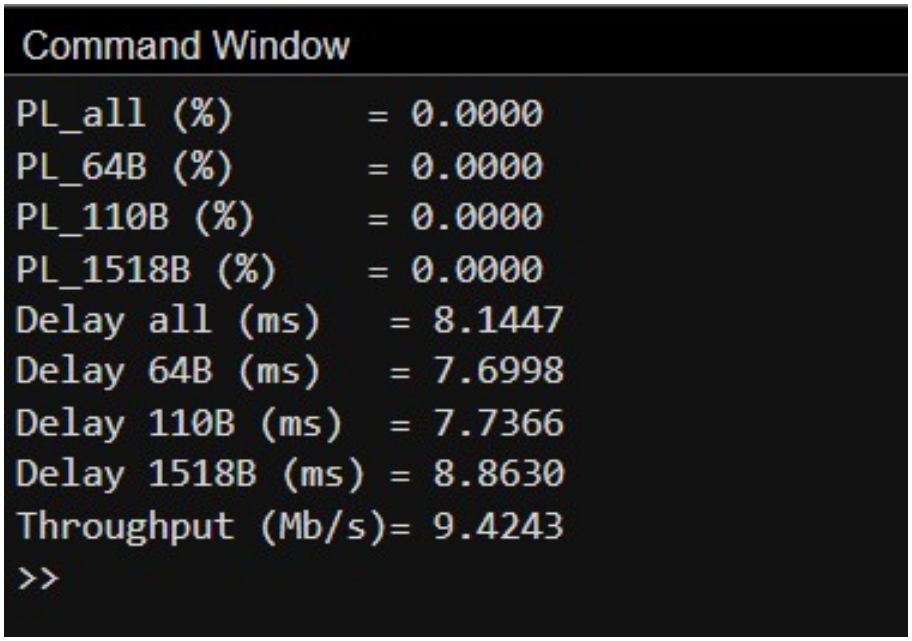

Figure 10: 1e) Resultados apresentados na linha de comandos dos resultados teóricos.

De seguida podemos analisar lado a lado a comparação entre os resultados teóricos obtidos neste exercício (1e) e os resultados obtidos no exercício anterior (1d):

Métrica	**Teórico (M/G/1)**	**Simulação (1.d)**
Perda total (%)	0.0000	0.0000
Atraso médio (ms)	8.14	7.98 ± 0.19
Atraso 64B (ms)	7.70	7.69 ± 0.18
Atraso 110B (ms)	7.74	7.72 ± 0.19
Atraso 1518B (ms)	8.86	8.82 ± 0.19

Table 1: Comparação entre resultados teóricos (M/G/1) e simulados (1.d).

Os resultados demonstram uma **forte concordância** entre o modelo teórico do exercício 1e) e a simulação do exercício 1d). O modelo M/G/1 demonstra a **ausência total de perdas**, visto que a fila é infinita. O atraso médio teórico e o obtido por simulação são praticamente idênticos, confirmando a precisão da aproximação teórica mesmo em condições de carga próximas do limite de saturação.

As pequenas diferenças entre os resultados devem-se a fatores como:

- A fila simulada no exercício 1d) possui capacidade finita ($f = 1000000$ Bytes enquanto os valores teóricos não);
- O modelo M/G/1 assume chegadas e serviços contínuos, enquanto o simulador trabalha com eventos discretos.

3.6 exercício 1f

3.6.1 Contexto

Neste exercício foram repetidas as simulações do Exercício 1d, alterando apenas o tamanho máximo da fila para $f = 10.000$ Bytes, mantendo a capacidade do canal $C = 10$ Mbps e a mesma taxa de chegada $\lambda = 1900$ pps. O objetivo é avaliar o impacto da diminuição do tamanho da fila no desempenho do sistema, analisando a perda média de pacotes (*Packet Loss*) e o atraso médio (*Average Packet Delay*) para todos os pacotes e para os três tamanhos especiais definidos previamente(64, 110 e 1518 Bytes).

3.6.2 Implementação

A implementação segue a mesma estrutura do exercício 1.d, recorrendo novamente ao simulador `Simulator1A`. Os resultados médios e respetivos intervalos de confiança são organizados em vetores (`PL_means`, `PL_terms`, `APD_means`, `APD_terms`) e representados tal como nos exercícios anteriores graficamente em dois diagramas de barras, um para a perda de pacotes e outro para o atraso médio.

A única diferença relativamente à implementação do exercício 1.d é o valor do parâmetro f, que nesta experiência é significativamente menor. Passando de 1000000 Bytes para 10000 Bytes. Esta alteração simula um sistema com um tamanho de fila reduzido, mais propenso à perda de pacotes em situações de sobrecarga.

3.6.3 Resultados

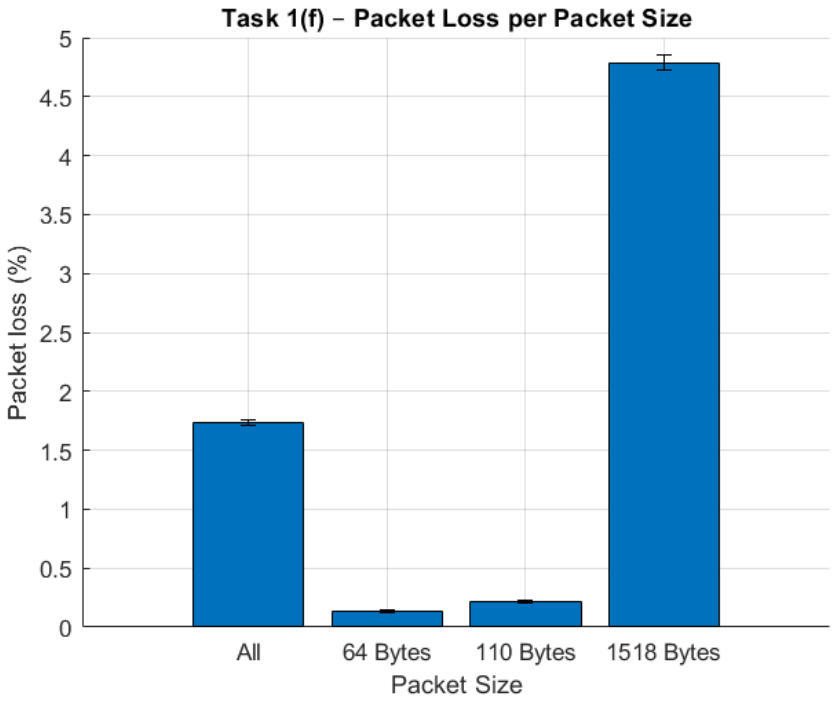

Figure 11: 1f) Perda média de pacotes para cada tipo de pacote.

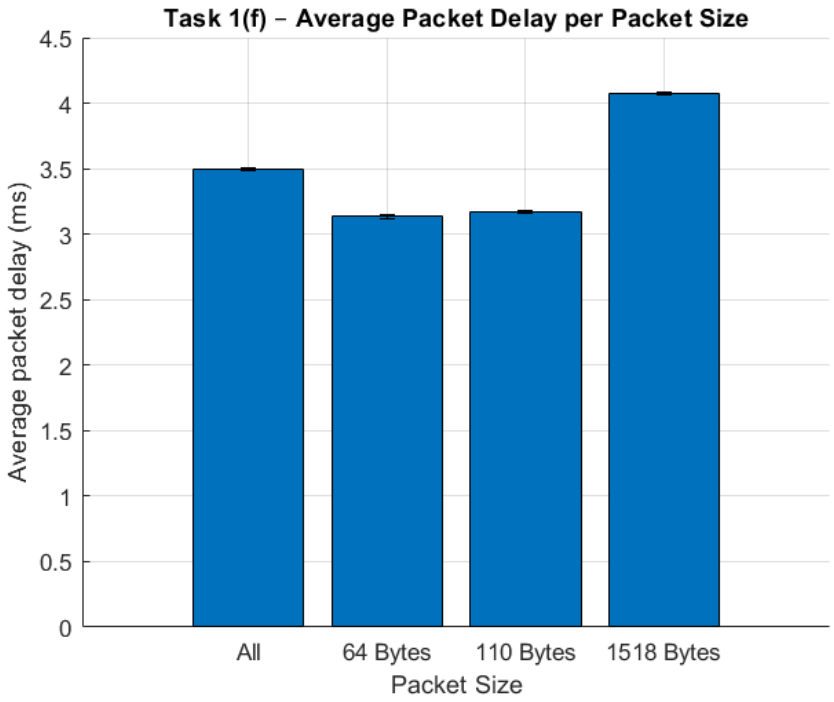

Figure 12: 1f) Atraso médio dos pacotes para cada tipo de pacote.

```
Command Window
TASK 1f)
Packet Loss (%)    = 1.69e+00 +- 2.36e-02
Average Packet Delay (ms)        = 3.48e+00 +- 1.27e-02
Packet Loss (%)    = 1.25e-01 +- 6.74e-03
Average Packet Delay (ms)        = 3.12e+00 +- 1.43e-02
Packet Loss (%)    = 2.06e-01 +- 1.10e-02
Average Packet Delay (ms)        = 3.14e+00 +- 1.29e-02
Packet Loss (%)    = 4.68e+00 +- 6.15e-02
Average Packet Delay (ms)        = 4.06e+00 +- 1.20e-02

Simulation ended!
>>
```

Figure 13: 1f) Resultados apresentados na linha de comandos.

Comparando com os resultados do exercício 1.d (f = 1,000,000 Bytes), verifica-se um aumento significativo da perda de pacotes, tendo em conta que no exercício 1d) não ocorriam perdas de pacotes e neste exercício (1f) notam-se para todos os tamanhos de pacotes enviados diferentes perdas de pacotes. Começando em perdes de 1,69% e atingindo até 4,68% de perdas de pacote para pacotes de tamanhos maiores (1518 Bytes).

Relativamente aos atrasos médios de chegada dos pacotes, nota-se relativamente ao exercício 1d) uma redução significativa dos atrasos, os resultados comparados demonstram uma redução de atrasos para metade ou menos do tempo. No exercício d) onde o tamanho da fila é de 1000000 Bytes, os atrasos são os seguintes: 7.98ms, 7.54ms, 7.58ms e 8.70ms (não contabilizando a margem de erro), enquanto no caso do exercício 1f) os atrasos são

os seguintes: 3.48ms, 3.12ms, 3.14ms e 4.06ms. Esta redução é devido ao tamanho da fila ser reduzido e a mesma ficar preenchida mais rapidamente, os pacotes são frequentemente descartados, reduzindo os atrasos.

Isto ocorre porque o tamanho da fila é mais pequeno ($f = 10{,}000\,\text{Bytes}$), logo atinge a sua capacidade máxima com maior frequência, levando à rejeição de pacotes quando o sistema se encontra congestionado.

Conclusão: A redução do tamanho da fila tem um impacto negativo na eficiência do sistema. A perda de pacotes aumenta drasticamente, enquanto que o atraso médio diminui. Estes resultados reforçam a importância de um tamanho adequado do buffer: uma fila demasiado grande introduz atrasos excessivos, enquanto uma fila demasiado pequena provoca perdas significativas.

3.7 Exercício 1g (Simulator1B)

3.7.1 Contexto

Neste exercício foi desenvolvida uma nova versão do simulador chamada **Simulator1B**, baseada na implementação do **Simulator1A**, com algumas adições. A principal diferença entre as duas versões está na política de gestão da fila de espera (*queuing discipline*).

Enquanto o **Simulator1A** utilizava uma disciplina FIFO (*First In, First Out*), o **Simulator1B** introduz um mecanismo de **prioridades**, de forma a diferenciar o tratamento dos pacotes conforme o seu tamanho. Esta alteração visa estudar o impacto de um esquema de priorização no desempenho da rede, especialmente em condições de elevada carga de tráfego e limitação de capacidade da fila.

A atribuição de prioridades foi definida da seguinte forma:

- **Prioridade 1:** pacotes com tamanho entre [1501, 1518] Bytes;
- **Prioridade 2:** pacotes com tamanho entre [1001, 1500] Bytes;
- **Prioridade 3:** pacotes com tamanho entre [1, 1000] Bytes.

Desta forma, pacotes maiores têm prioridade na transmissão sempre que existirem múltiplos pacotes na fila, simulando um cenário onde pacotes de maior dimensão representam tráfego de maior importância.

3.7.2 Implementação

A estrutura geral do simulador foi mantida, incluindo o mecanismo de eventos discretos e os contadores estatísticos, mas foram introduzidas modificações relevantes no módulo de gestão da fila. O código total do simulador Simulator1B está presente aqui!

A principal diferença em relação ao **Simulator1A** está na adição de três filas distintas para além da fila original (`QUEUE`), representadas pelas variáveis `QUEUE1`, `QUEUE2` e `QUEUE3`:

```
QUEUE1 = [];   % PRIORITY 1
QUEUE2 = [];   % PRIORITY 2
QUEUE3 = [];   % PRIORITY 3
```

Durante o evento de `ARRIVAL`, o simulador:

1. Determina a prioridade do pacote com a função auxiliar `priorityOf(PacketSize)`;
2. Caso exista espaço disponível na fila (`QUEUEOCCUPATION + PacketSize <= f`), o pacote é colocado na fila correspondente;
3. Caso contrário, o pacote é descartado e contabilizado como perda.

O código responsável por esta lógica é o seguinte:

```
if QUEUEOCCUPATION + PacketSize <= f
    priority = priorityOf(PacketSize);
    if priority == 1
        QUEUE1 = [QUEUE1; PacketSize, Clock];
    elseif priority == 2
```

```
        QUEUE2 = [QUEUE2; PacketSize, Clock];
    elseif priority == 3
        QUEUE3 = [QUEUE3; PacketSize, Clock];
    end
    QUEUEOCCUPATION = QUEUEOCCUPATION + PacketSize;
else
    LOSTPACKETS = LOSTPACKETS + 1;
    if PacketSize == 64
        LOSTPACKETS_64 = LOSTPACKETS_64 + 1;
    elseif PacketSize == 110
        LOSTPACKETS_110 = LOSTPACKETS_110 + 1;
    elseif PacketSize == 1518
        LOSTPACKETS_1518 = LOSTPACKETS_1518 + 1;
    end
end
```

Quando ocorre o evento de `DEPARTURE`, ou seja, quando um pacote termina a transmissão, o simulador verifica as filas em ordem de prioridade, começando pela `QUEUE1`, depois `QUEUE2`, e por fim `QUEUE3`. Assim, pacotes grandes têm sempre preferência na ocupação do canal:

```
if QUEUEOCCUPATION > 0
    if ~isempty(QUEUE1)
        QSize = QUEUE1(1,1);
        QInstant = QUEUE1(1,2);
        QUEUE1(1, :) = [];
    elseif ~isempty(QUEUE2)
        QSize = QUEUE2(1,1);
        QInstant = QUEUE2(1,2);
        QUEUE2(1, :) = [];
    elseif ~isempty(QUEUE3)
        QSize = QUEUE3(1,1);
        QInstant = QUEUE3(1,2);
        QUEUE3(1, :) = [];
    end
    QUEUEOCCUPATION= QUEUEOCCUPATION - QSize;
    Event_List = [Event_List; DEPARTURE, Clock + 8*QSize/(C*1e6), QSize, QInstant];
else
    STATE= 0;
    continue;
end
```

Após cada transmissão concluída, o simulador atualiza as estatísticas correspondentes ao pacote processado, distinguindo os resultados por tamanho de pacote. Assim, são calculadas as contagens de pacotes transmitidos e os atrasos acumulados por categoria:

```
if PacketSize == 64
    TRANSPACKETS_64 = TRANSPACKETS_64 + 1;
    DELAYS_64 = DELAYS_64 + (Clock - ArrInstant);
```

```
elseif PacketSize == 110
    TRANSPACKETS_110 = TRANSPACKETS_110 + 1;
    DELAYS_110 = DELAYS_110 + (Clock - ArrInstant);
elseif PacketSize == 1518
    TRANSPACKETS_1518 = TRANSPACKETS_1518 + 1;
    DELAYS_1518 = DELAYS_1518 + (Clock - ArrInstant);
end
```

Por fim, as métricas de desempenho são determinadas de forma separada para o total de pacotes e para cada tipo de tamanho. Estas métricas incluem a **Perda de Pacotes** (`PL`) e o **Atraso Médio** (`APD`).

```
PL = 100 * LOSTPACKETS / TOTALPACKETS;       % Percentagem de perda
APD = 1000 * DELAYS / TRANSPACKETS;          % Atraso médio (ms)

PL_64 = 100 * LOSTPACKETS_64 / TOTALPACKETS_64;
APD_64 = 1000 * DELAYS_64 / TRANSPACKETS_64;

PL_110 = 100 * LOSTPACKETS_110 / TOTALPACKETS_110;
APD_110 = 1000 * DELAYS_110 / TRANSPACKETS_110;

PL_1518 = 100 * LOSTPACKETS_1518 / TOTALPACKETS_1518;
APD_1518 = 1000 * DELAYS_1518 / TRANSPACKETS_1518;
```

Todas estas métricas são consistentes com as utilizadas no **Simulator1A**, exceto pelas adições explicadas previamente relativas ao sistema de prioridades de filas com pacotes maiores. Estas modificações (do sistema de prioridade) permitem observar o comportamento da rede sob um regime de prioridade, facilitando a análise do impacto que esta política tem na latência média e na perda de pacotes, em comparação com o modelo FIFO original.

3.8 Exercício 1h

3.8.1 Contexto

Neste exercício, pretende-se utilizar a função `Simulator1B` para estimar, por simulação, os parâmetros de desempenho **perda média de pacotes** (*Packet Loss*) e **atraso médio dos pacotes** (*Average Packet Delay*) em um sistema com diferentes classes de prioridade associadas ao tamanho dos pacotes.

A simulação deve considerar três tamanhos de pacotes distintos (64, 110 e 1518 Bytes), bem como o conjunto total de pacotes. O objetivo é analisar o impacto da política de prioridade no desempenho do sistema, avaliando como o tamanho do pacote afeta as métricas de atraso e perda.

Os parâmetros de simulação são:

$$C = 10\,\text{Mbps}, \quad f = 1{,}000{,}000\,\text{Bytes}, \quad \lambda = 1900\,\text{pps}$$

O critério de paragem é de $P = 10^5$ pacotes transmitidos, e foram realizadas $N = 50$ execuções independentes para calcular intervalos de confiança a 90%.

3.8.2 Implementação

Como já concretizado previamente, o simulador `Simulator1B` foi desenvolvido de forma a permitir a recolha separada de estatísticas para três tamanhos específicos de pacotes, 64, 110 e 1518 Bytes, considerando agora uma política de **prioridades** entre classes.

Neste exercício, o código principal tem como objetivo executar a função `Simulator1B` múltiplas vezes, de modo a calcular as médias e os intervalos de confiança a 90% para cada métrica de desempenho (*Packet Loss* e *Average Packet Delay*), tanto no conjunto total de pacotes como em cada classe de tamanho.

Inicialmente, são definidos os parâmetros de simulação:

$$C = 10\,\text{Mbps}, \quad f = 1{,}000{,}000\,\text{Bytes}, \quad \lambda = 1900\,\text{pps}, \quad P = 10^5, \quad N = 50$$

onde P representa o critério de paragem (número de pacotes transmitidos) e N o número de execuções independentes realizadas para o cálculo estatístico.

Durante cada iteração, a função `Simulator1B` devolve nove métricas principais:

- PL e APD: perda e atraso médios considerando todos os pacotes;
- $PL_{64}, PL_{110}, PL_{1518}$: perdas médias específicas por tamanho de pacote;
- $APD_{64}, APD_{110}, APD_{1518}$: atrasos médios específicos por tamanho de pacote;
- MPD e TT: atraso máximo e *throughput* total.

Após todas as execuções, o código calcula a média e o intervalo de confiança a 90% para cada métrica, semelhante a exercícios anteriores.

Os resultados são então organizados em quatro vetores principais:

- `PL_means` e `PL_terms` — médias e intervalos de confiança da perda de pacotes;
- `APD_means` e `APD_terms` — médias e intervalos correspondentes ao atraso médio.

Desta forma, o código permite observar claramente o impacto da política de prioridade no desempenho do sistema, evidenciando as diferenças de atraso entre pacotes de diferentes classes.

Por fim, os resultados são apresentados graficamente através de dois gráficos de barras que vamos analisar de seguida.

3.8.3 Resultados

A Figura 14 apresenta as perdas médias de pacotes, enquanto a Figura 15 mostra os atrasos médios. Os resultados obtidos na linha de comandos encontram-se representados na Figura 16.

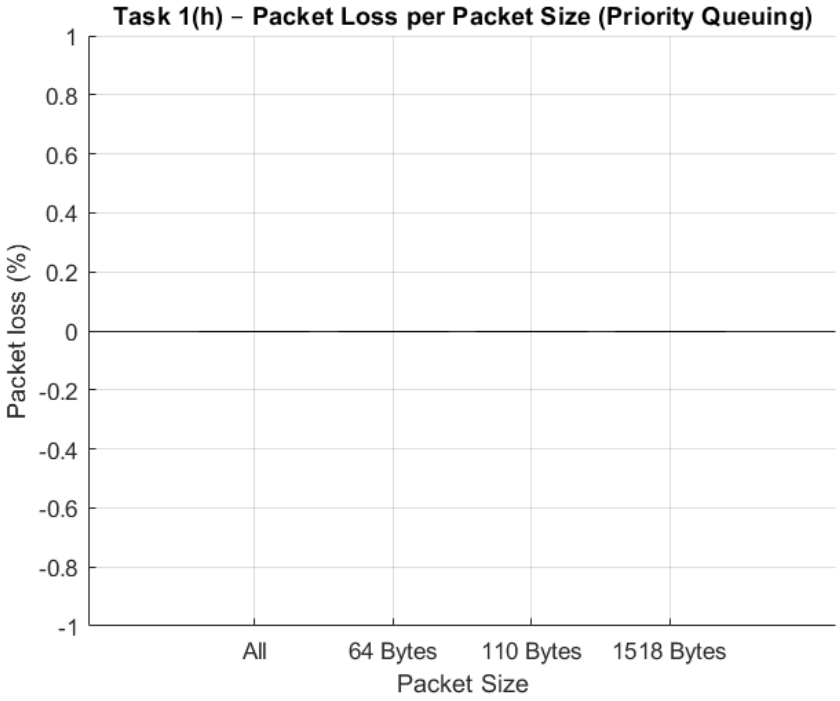

Figure 14: 1h) Perda média de pacotes para cada tipo de pacote.

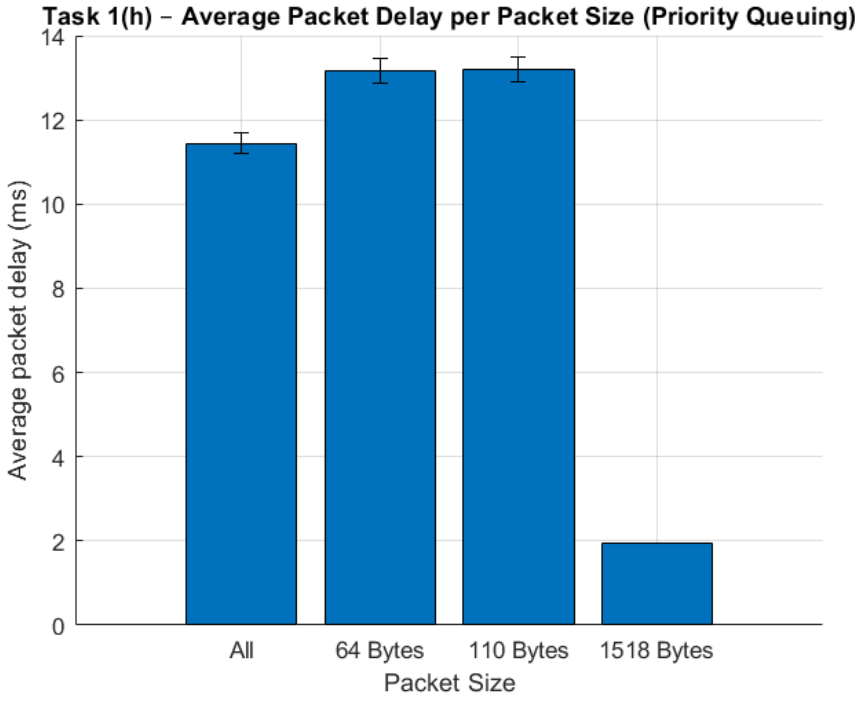

Figure 15: 1h) Atraso médio dos pacotes para cada tipo de pacote.

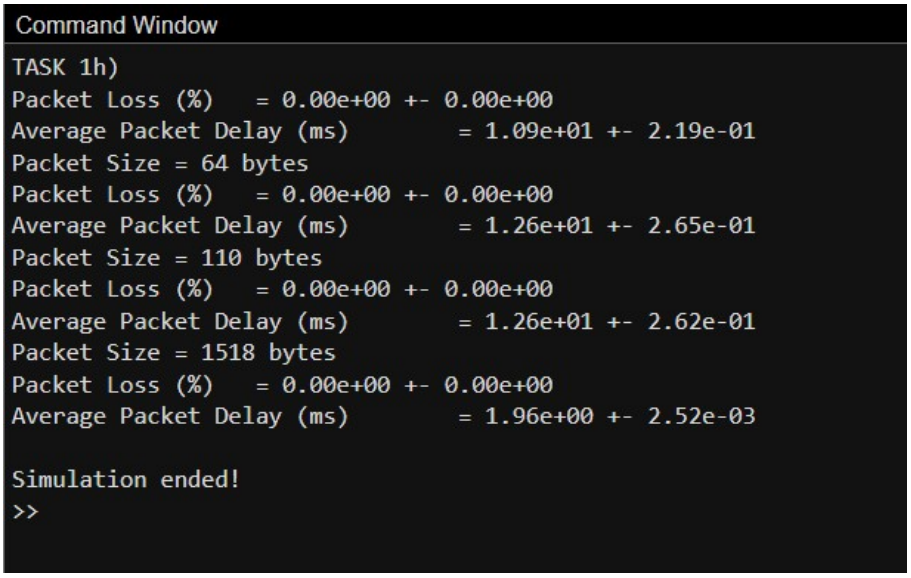

Figure 16: 1h) Resultados apresentados na linha de comandos.

Análise e conclusões dos Resultados

Os resultados indicam que a **perda média de pacotes é nula** para todas as classes, o que se justifica pela elevada capacidade da ligação ($C = 10\,\text{Mbps}$) e pelo tamanho máximo de fila ($f = 1\,\text{MB}$), que evitam a ocorrência de congestionamento.

No que diz respeito ao atraso médio, observa-se uma diferença significativa entre as classes:

- Pacotes de 1518 Bytes (classe de maior prioridade) apresentam um **atraso médio muito reduzido**, em torno de 1.96 ms;
- Pacotes de 64 e 110 Bytes (prioridades inferiores) registam atrasos médios de aproximadamente 12.6 ms.

Este comportamento confirma o impacto direto da política de prioridade: pacotes com maior prioridade têm acesso preferencial ao canal, resultando em atrasos menores. Por outro lado, pacotes de menor prioridade tendem a acumular-se na fila, sofrendo maiores tempos de espera.

Conclui-se, portanto, que o sistema implementado opera **sem perdas** e com **atrasos proporcionais à prioridade atribuída**.

3.9 Exercício 1i

3.9.1 Contexto

Neste exercício foram repetidas as simulações do Exercício 1h, alterando o tamanho máximo da fila para $f = 10{,}000$ Bytes, mantendo a capacidade do canal $C = 10$ Mbps e a mesma taxa de chegada $\lambda = 1900$ pps. O objetivo é avaliar o impacto da redução do tamanho da fila no desempenho do sistema quando se utiliza o mecanismo de **prioridade entre classes de pacotes**. Tal como anteriormente, são analisadas as métricas de *Packet Loss* e *Average Packet Delay* para todos os pacotes e para cada um dos três tamanhos específicos (64, 110 e 1518 Bytes).

3.9.2 Implementação

A implementação segue a mesma estrutura do exercício 1h, recorrendo novamente ao simulador `Simulator1B`, responsável por tratar o sistema com filas de prioridade. O código executa $N = 50$ simulações independentes, cada uma transmitindo $P = 10^5$ pacotes, e calcula as médias e intervalos de confiança a 90% para todas as métricas de desempenho.

A única diferença relativamente à implementação do exercício 1h é o valor do parâmetro f, que passa de 1,000,000 Bytes para 10,000 Bytes. Esta alteração introduz um sistema com uma fila muito mais pequena, levando a um maior número de perdas de pacotes em situações de congestionamento, especialmente para pacotes de menor prioridade.

Os resultados são organizados nos vetores `PL_means`, `PL_terms`, `APD_means` e `APD_terms`, e são posteriormente apresentados graficamente em dois diagramas de barras: um para a perda média de pacotes e outro para o atraso médio dos pacotes.

3.9.3 Resultados

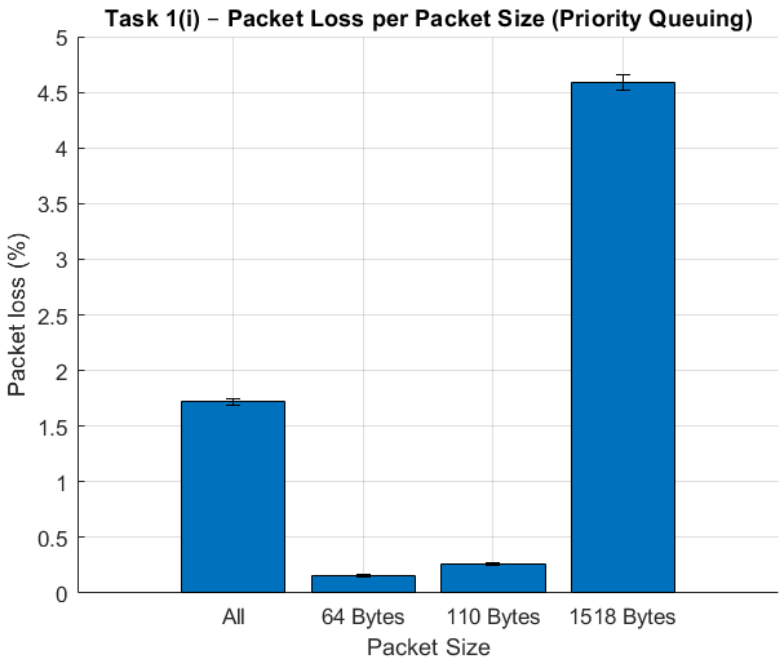

Figure 17: 1i) Perda média de pacotes para cada tipo de pacote (com prioridade).

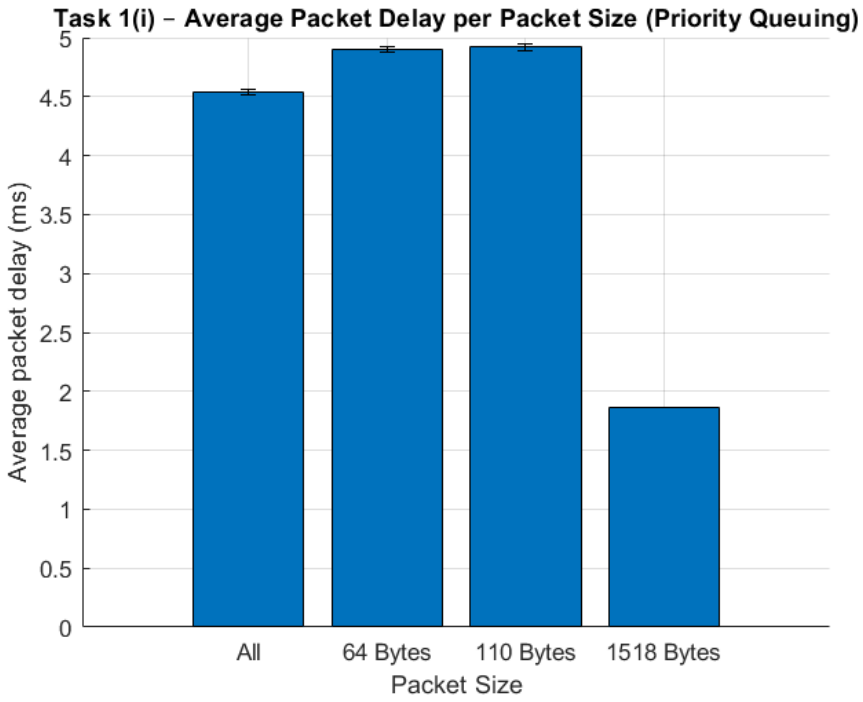

Figure 18: 1i) Atraso médio dos pacotes para cada tipo de pacote (com prioridade).

```
Command Window
TASK 1i)
Packet Loss (%)    = 1.72e+00 +- 2.33e-02
Average Packet Delay (ms)         = 4.52e+00 +- 1.58e-02
Packet Size = 64 bytes
Packet Loss (%)    = 1.60e-01 +- 8.02e-03
Average Packet Delay (ms)         = 4.88e+00 +- 2.06e-02
Packet Size = 110 bytes
Packet Loss (%)    = 2.60e-01 +- 9.72e-03
Average Packet Delay (ms)         = 4.90e+00 +- 2.13e-02
Packet Size = 1518 bytes
Packet Loss (%)    = 4.61e+00 +- 6.08e-02
Average Packet Delay (ms)         = 1.87e+00 +- 1.54e-03

Simulation ended!
>>
```

Figure 19: 1i) Resultados apresentados na linha de comandos.

Como esperado, ao comparar com os resultados do exercício 1h (f = 1,000,000 Bytes), observa-se um aumento substancial da perda média de pacotes (Packet Loss), passando de não haver perdas no exercício 1h para a ocorrência de perdas neste exercício (1i) em que o tamanho da fila corresponde a $f = 10000$ Bytes. Enquanto no 1h o sistema conseguia armazenar todos os pacotes devido ao grande tamanho da fila, neste caso a fila reduzida atinge a saturação rapidamente, resultando em perdas significativas, especialmente para pacotes grandes (1518 Bytes), que ocupam mais espaço e têm maior prioridade, visto que a fila ao tentar armazenar primeiro grandes quantidades de pacotes de 1518 Bytes, ao entrar em saturação, simplesmente terá que os descartar.

Em contrapartida, o atraso médio dos pacotes diminui, uma vez que os pacotes são descartados com maior frequência, evitando o acumular de grandes filas. Passando por exemplo de atrasos médios de pacotes de tamanhos de 64 Bytes e de 110 Bytes de 12,6ms para cerca de 4,9ms e de atrasos médios de pacotes de 1518 Bytes de 1,96 ms para 1.87ms (Reduzindo apenas ligeiramente devido à sua prioridade). No entanto, esta redução de atraso vem à custa de uma degradação da fiabilidade da transmissão.

Conclui-se, portanto, que a diminuição do tamanho da fila para (f = 10,000 Bytes) tem um efeito direto e negativo na taxa de perda de pacotes, ao mesmo tempo que reduz marginalmente o atraso médio do sistema à custa da sua fiabilidade.

3.10 Exercício 1j

3.10.1 Contexto

Durante o desenvolvimento deste primeiro mini-projeto, testámos experimentalmente como o tamanho da fila dos simuladores 1A e 1B pode ter efeitos diretos e drásticos nos resultados obtidos para cada tipo de exercício. A diferença chave nestas resultados foi a alteração do tamanho da fila de 1000000 de Bytes para 10000 Bytes em ambos as comparações nos exercícios 1d e 1f, usando o simulador Simulator1A, e nos exercícios 1h e 1i, usando o simulador Simulator1B. É importante voltar a mencionar que no caso do Simulator1A, este usa uma metodologia de FIFO, ou seja, o primeiro pacote a entrar será o primeiro a ser processado, enquanto no Simulator1B, usamos uma metodologia de prioridades, onde os pacotes maior tamanho têm prioridade sobre os de menor tamanho.

3.10.2 Resposta

Antes de justificar diretamente quais são as principais diferenças entre os resultados obtdos usando o Simulator1A e o Simulator1B, vamos voltar a apresentar lado a lado os resultados relevantes de cada um:

Para o Simulator1A, comparamos novamente os resultados entre 1d (f = 1000000 Bytes) e 1f (f = 10000 Bytes):

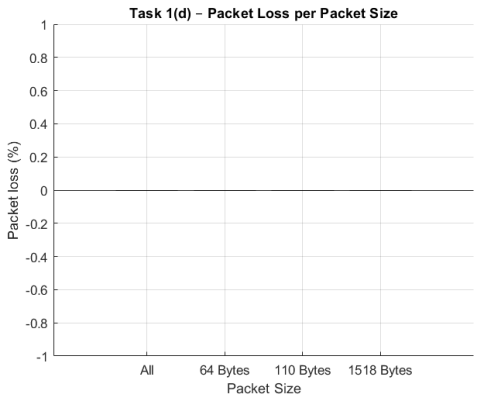

Figure 20: 1d - Packet Loss

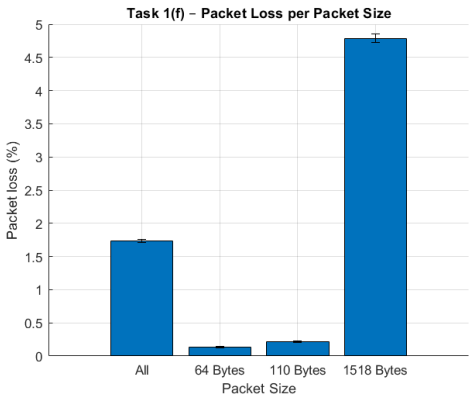

Figure 21: 1f - Packet Loss

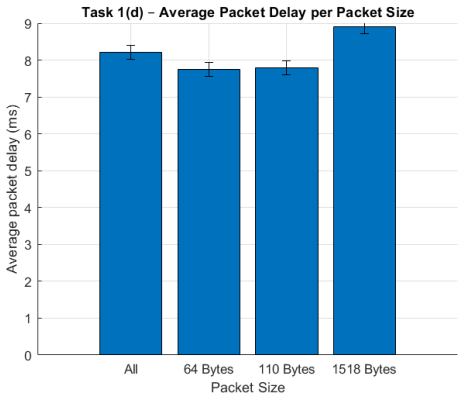

Figure 22: 1d - Packet Delay

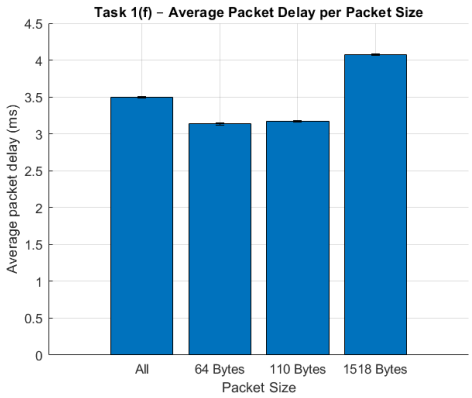

Figure 23: 1f - Packet Delay

Para o Simulator1B, comparamos novamente os resultados entre 1h (f = 1000000 Bytes) e 1i (f = 10000 Bytes):

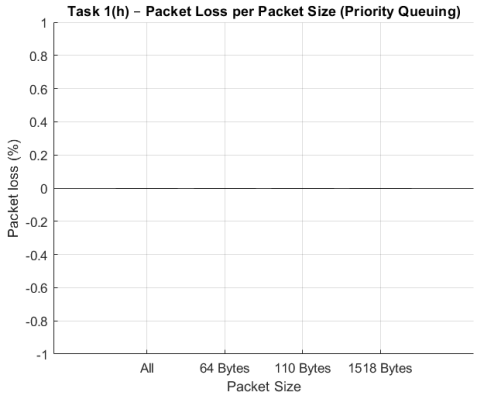

Figure 24: 1h - Packet Loss

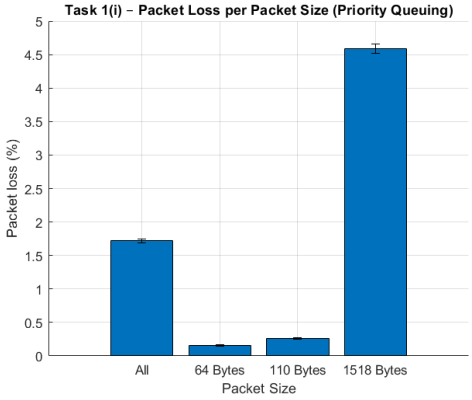

Figure 25: 1i - Packet Loss

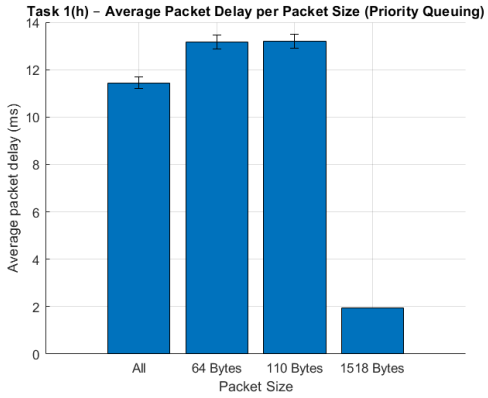

Figure 26: 1d - Packet Delay

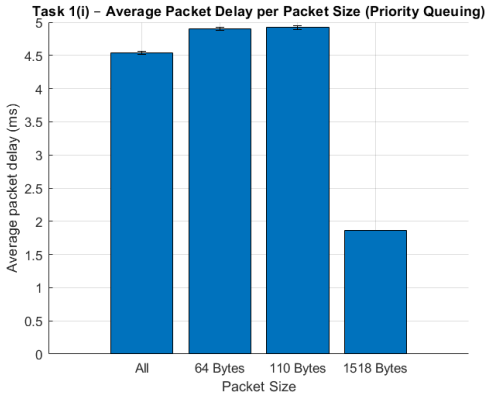

Figure 27: 1f - Packet Delay

Após analisar estes resultados, podemos justificar que apesar de ambos terem semelhanças, como quando a fila tem um tamanho de 1000000 Bytes, ambas as simulações demonstram que ocorrem perdas de pacotes nulas, ou seja, não existem perdas de pacotes durante o processo de transmissão, outra semelhança é que ao reduzirmos o tamanho

da fila para 10000 Bytes, quer o Simulator1A que usa FIFO como o Simulator1B que usa prioridades, demonstram perdas de pacotes semelhantes, em ambos os casos, os pacotes que se perdem com maior frequência são os de tamanho maior (1518 Bytes), enquanto os de tamanho mais reduzido mais fácilmente são transmitidos sem perdas.

Relativamente às diferenças, podemos concluir que a chave está no atraso médio dos pacotes. Onde é de notar que no Simulator1A, ao usar a metodologia de FIFO, o atraso médio dos pacotes é relativamente constante para todos os pacotes, havendo ligeiramente mais atraso para pacotes maiores (1518 Bytes) tanto para tamanho de fila de 1000000 Bytes em que o atraso se situa entre os 7.5ms e os 8.7ms, como para os tamanho de fila de 10000 Bytes onde os atrasos médios de pacotes estão entre 3,12ms e 4.06ms. Havendo em ambos os casos uma variação minima entre os mesmos. Ou seja, o intervalo de diferenças entre atrasos de pacotes de tamanhos diferentes é cerca de 1ms.

Já no Simulator1B, onde usamos a metodologia de prioridade, o atraso médio dos pacotes é bastante mais pequeno para pacotes maiores (1518 Bytes) que são os que têm maior prioridade, onde podemos analisar que apesar da diferença de tamanhos da fila nos exercícios 1h e 1i em ambos os casos os pacotes de maior tamanho e de maior prioridade têm atrasos muito menores que os de prioridades mais baixas. Como podemos analisar nos gráficos, quando o tamanho da fila é de 1000000 Bytes, o atraso médio dos pacotes de 1518 Bytes é de 1,96 ms compardos com os 12,6ms dos pacotes de tamanhos de 64 Bytes e de 110 Bytes.

No caso do exercício 1i, quando a fila tem o tamanho de 10000 Bytes, mesmo com a saturação eminente da fila, ou seja, quando já ocorrem perdas de pacotes, é de notar que os atrasos médios dos pacotes de tamanho maior continuam muito menores relativamente aos de tamanhos mais pequenos e de menor prioridade. Como podemos analisar, os pacotes de 1518 Bytes têm um atraso médio de 1.87ms comprados com os pacotes de tamanho menor de 4,9ms.

Isto prova que o uso eficiente de prioridades, quando combinado com o uso de um tamanho adequado para a fila, permite reduzir atrasos de forma consistente em pacotes de maior importância. Enquanto o uso de metodologis de FIFO mantêm atrasos contantes perante todos os diferentes tamanhos de pacotes. Mesmo em situações criticas onde já ocorrem perdas de pacotes e o tamanho da fila é inadequado esta diferença de atrasos entre ambos os simuladores é notável. Sendo essa a principal diferença.

4 Parte 2

4.1 Exercício 2a (Simulator3A)

4.1.1 Contexto

Neste exercício foi desenvolvida uma nova versão do simulador designada por **Simulator3A**, baseada diretamente na implementação original do **Simulator3**. O código na sua totalidade do Simulator3A pode ser consultado aqui!

O objetivo desta nova versão é introduzir a simulação de um **Bit Error Rate (BER)**, representado pelo parâmetro b, que modela a probabilidade de um bit ser transmitido incorretamente através da ligação.

Com esta modificação, o simulador torna-se mais realista, permitindo analisar o impacto de erros de transmissão nas métricas de desempenho da rede. Os erros de bits podem causar perdas adicionais de pacotes, mesmo quando a fila não está cheia, o que reflete situações comuns em comunicações sem fio ou ligações degradadas.

4.1.2 Implementação

A estrutura base do simulador, incluindo o sistema de eventos discretos (`ARRIVAL` e `DEPARTURE`) e as estatísticas de desempenho (`PL`, `APD`, `MPD`, `TT`), foi mantida. A principal alteração introduzida no **Simulator3A** foi a inclusão de um novo parâmetro de entrada, o `bit error rate (b)` e a modificação da forma como os pacotes são processados durante o evento de `DEPARTURE`.

De seguida explicamos e justificamos estas principais diferenças entre o **Simulator3** e o **Simulator3A**:

Novo parâmetro de entrada: `b` Foi adicionada uma nova variável de entrada ao cabeçalho da função, responsável por representar a *Bit Error Rate* (BER), ou taxa de erro binário do canal. Este parâmetro permite simular condições mais realistas de transmissão, em que os bits podem ser corrompidos devido a ruído ou interferências.

```
function [PLd ,APDd ,MPDd ,PLv ,APDv ,MPDv, TT] = Simulator3A(lambda,C,f,P,n,b)
% ...
%  b      - bit error
```

Alteração na secção de `DEPARTURE` A modificação mais significativa foi feita na secção correspondente ao evento de saída de pacotes (`DEPARTURE`). Enquanto na versão original (**Simulator3**) todos os pacotes eram considerados transmitidos com sucesso assim que saíam do sistema, na nova versão foi introduzida uma verificação de erros a nível de bits, que determina se o pacote chega corretamente ao destino.

Para isso, utiliza-se a probabilidade de um pacote ser recebido sem erros:

$$P(\text{sem erro}) = (1-b)^{8\times\text{PacketSize}}$$

Assim, quanto maior o tamanho do pacote ou a taxa de erro binário (`b`), maior será a probabilidade de o pacote ser corrompido e, portanto, descartado.

O novo trecho de código responsável por essa verificação é o seguinte:

```
if(rand() < (1-b)^(PacketSize *8))      % Pacote chega sem erros
    % Contabilização normal do pacote transmitido
else                                    % Pacote chega com erros
    if (PacketType == DATA)
        LOSTPACKETSd = LOSTPACKETSd + 1;
    else
        LOSTPACKETSv = LOSTPACKETSv + 1;
    end
end
```

Dessa forma, os pacotes que chegam com erros não são contabilizados como transmitidos, mas sim como **pacotes perdidos**, incrementando as variáveis `LOSTPACKETSd` ou `LOSTPACKETSv`, conforme o tipo de tráfego.

No entanto, consideramos importante mencionar que as restantes secções da função foram mantidas idênticas à versão original, assegurando a compatibilidade com os parâmetros de desempenho previamente calculados:

- `PL` — Percentagem de perda de pacotes;
- `APD` — Atraso médio dos pacotes;
- `MPD` — Atraso máximo observado;
- `TT` — Débito médio do canal.

Com isso, o simulador **Simulator3A** mantém a estrutura e os objetivos do **Simulator3**, mas passa a modelar também os efeitos da taxa de erro binário sobre o desempenho global do sistema, permitindo estudar cenários com condições de transmissão imperfeitas e observar o impacto direto da BER na perda de pacotes e na eficiência do canal.

4.2 Exercício 2b

4.2.1 Contexto

Neste exercício, pretende-se estudar o impacto do aumento do número de fluxos VoIP (n) no desempenho global do sistema, em particular na **perda média de pacotes** dos tráfegos de DATA e de VoIP.

A simulação foi realizada com base no simulador desenvolvido no exercício anterior, o que inclui a modelação da taxa de erro binário de modo a representar um fluxo sujeito a erros de transmissão.

O objetivo é analisar como o crescimento do número de fluxos VoIP afeta a fiabilidade do sistema, mantendo-se constantes os restantes parâmetros de rede.

4.2.2 Implementação

O código realiza N = 50 execuções independentes para cada valor de n = [10, 20, 30, 40], transmitindo P = 10^5 pacotes em cada simulação. Os parâmetros utilizados foram os seguintes:

$$C = 10\,\text{Mbps}, \quad f = 1{,}000{,}000\,\text{Bytes}, \quad \lambda = 1500\,\text{pps}, \quad P = 10^5, \quad N = 50, \quad b = 10^-5$$

Para cada simulação, o simulador retorna os parâmetros de desempenho médias e o respetivo intervalo de confiança a 90% para

- PLd;
- PLv;

e os resultados foram representados graficamente em gráficos de barras.

4.2.3 Resultados

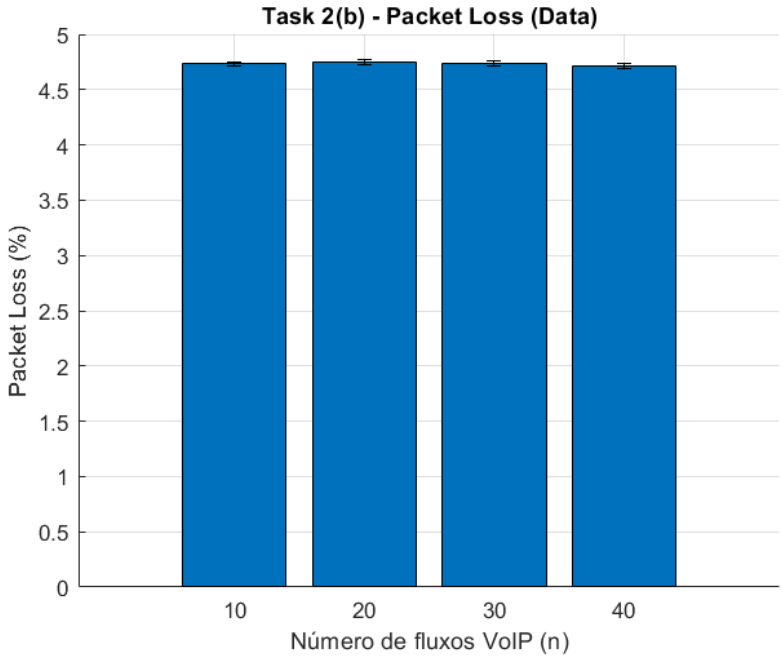

Figure 28: 2b – Perda média de pacotes de dados em função do número de fluxos VoIP (n).

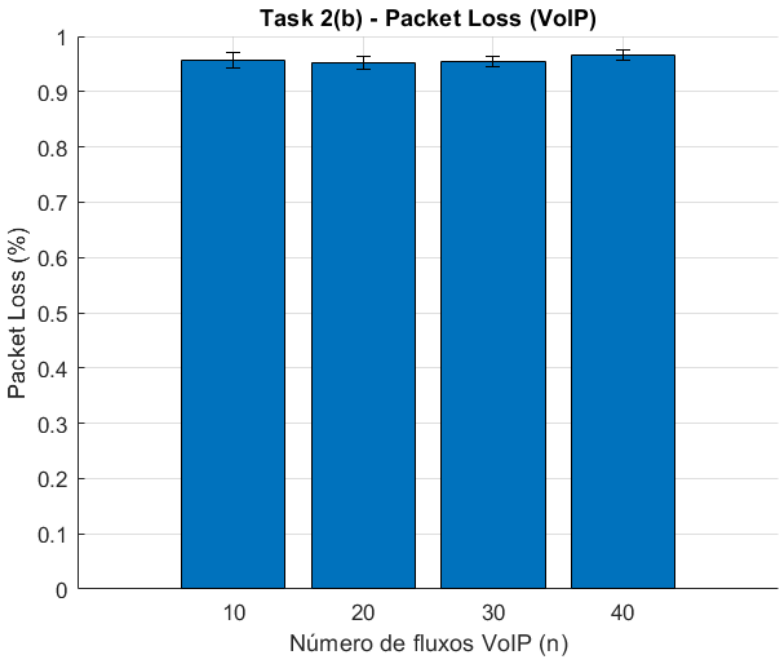

Figure 29: 2b – Perda média de pacotes VoIP em função do número de fluxos (n).

As figuras 28 e 29 apresentam as perdas médias de pacotes para tráfego de dados e VoIP, respetivamente.

Ao observar os gráficos podemos verificar que a perda média de pacotes se mantém praticamente constante para todos os valores de n.

Para o tráfego de dados, a perda situase em torno dos 4,7%, enquanto que para o tráfego de pacotes VoIP, a perda ronda os 0,95%.

Esta estabilidade indica que o fluxo e a capacidade da fila são suficientes para acomodar o aumento do número de fluxos VoIP sem provocar congestionamento na rede. A principal fonte de perdas acontece devido aos erros de transmissão introduzidos pela taxa de erro e não pela saturação da fila.

Também se pode verificar que o tráfego VoIP sofre perdas inferiores às do tráfego de dados. Isto deve-se ao facto dos pacotes VoIP serem mais pequenos, reduzindo a probabilidade de erro por bit e, consequentemente, a probabilidade de um pacote ser corrompido.

Com isto, podemos concluir que o aumento do número de fluxos VoIP não afeta significativamente a perda de pacotes, indicando que o sistema está dimensionado de forma adequada, que as perdas observadas são devidas, essencialmente, aos erros de transmissão e que os pacotes menores (neste caso, VoIP), têm menor probabilidade de perda.

4.3 Exercício 2c

4.3.1 Contexto

Neste exercício pretende-se analisar o impacto do número de fluxos VoIP no atraso médio dos pacotes, tanto para o tráfego de dados como para VoIP. Esta análise serve para complementar o estudo da alínea anterior, permitindo, assim, compreender de que forma o aumento da carga de pacotes VoIP influencia o tempo médio de transmissao e espera dos pacotes no sistema,

As simulações utilizam o mesmo cenário e parâmetros do exercício anterior e recorrendo ao Simulator3A.

4.3.2 Implementação

O código realiza N = 50 execuções independentes para cada valor de n = [10, 20, 30, 40], transmitindo P = 10^5 pacotes em cada simulação. Os parâmetros utilizados foram os seguintes:

$$C = 10\,\text{Mbps}, \quad f = 1{,}000{,}000\,\text{Bytes}, \quad \lambda = 1500\,\text{pps}, \quad P = 10^5, \quad N = 50, \quad b = 10^{-}5$$

Para cada simulação, o simulador retorna os parâmetros de desempenho médias e o respetivo intervalo de confiança a 90% para

- APDd;
- APDv;

e os resultados foram representados graficamente em gráficos de barras.

4.3.3 Resultados

As figuras 30 e 31 mostram, respetivamente o atraso médio dos pacotes de dados e dos pacotes VoIP em função do número de fluxos n.

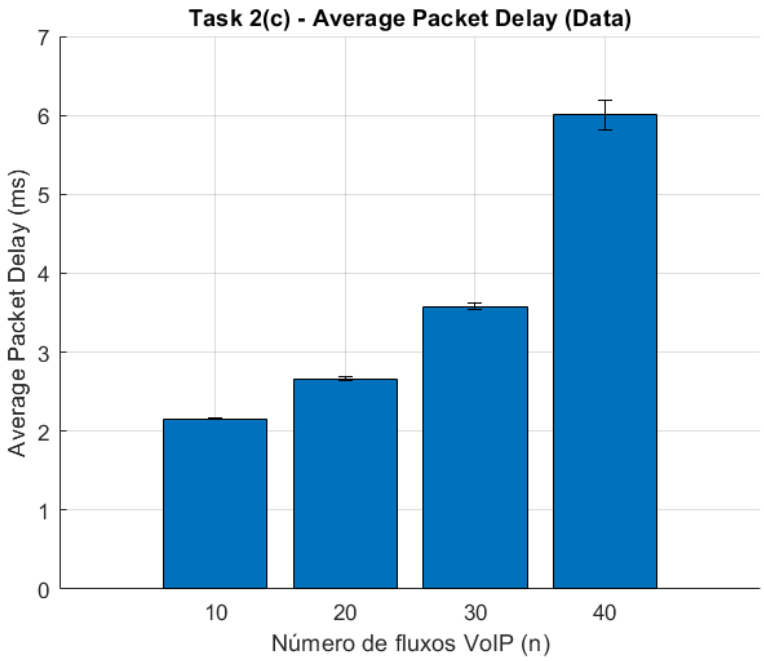

Figure 30: 2c – Atraso médio dos pacotes de dados em função do número de fluxos VoIP (n).

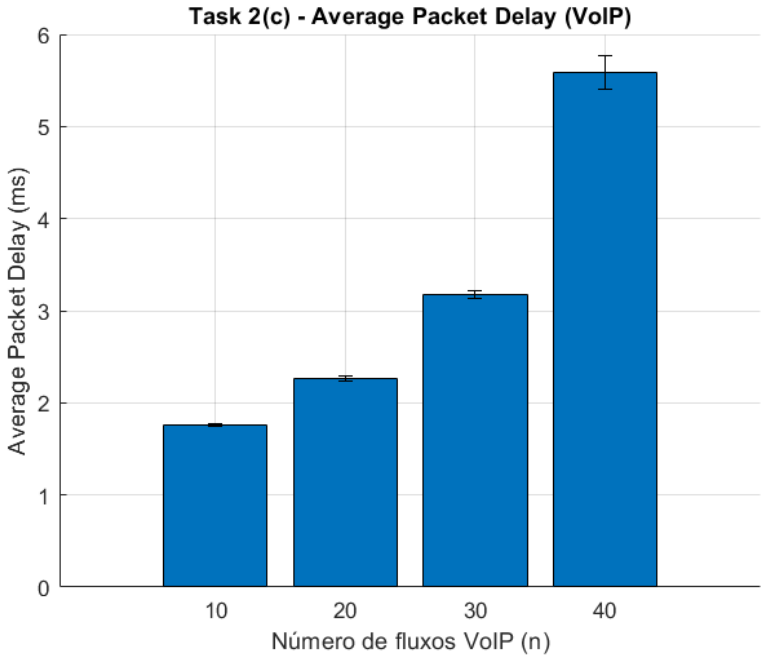

Figure 31: 2c – Atraso médio dos pacotes VoIP em função do número de fluxos (n).

Após a análise dos gráficos, é possível observar que o atraso médio aumenta progressivamente com o número de fluxos VoIP para ambos os casos de tráfego.

- Para o tráfego de dados (DATA), o atraso médio cresce de cerca de 2,1 ms (para n = 10) até aos 6,0 ms (n = 40).

- Para o tráfego VoIP, o atraso médio segue uma tendência semelhante, aumentando de 1,7 ms para, aproximadamente, 5,6 ms.

Este comportamento é o comportamento esperado, uma vez que à medida que o número de fluxos VoIP aumenta, o canal é partilhado por mais transmissões concorrentes, o que aumenta a ocupação da ligação e o tempo de espera médio da fila.

Pode-se concluir que o atraso médio aumenta com o número de fluxos VoIP por causa do aumento da carga do total do sistema e que o tráfego VoIP tende a experenciar atrasos ligeiramente menores devido ao menor tamanho dos pacotes.

4.4 Exercício 2d

4.4.1 Contexto

O objetivo deste exercício é avaliar a variação do throughput total do sistema em função do número de fluxos VoIP, considerando o mesmo cenário de simulação das alíneas anteriores.

Este parâmetro representa o débito efetivo do canal, medido em Mbps, e permite compreender como a capacidade total da ligação é utilizada à medida que a carga de tráfego VoIP aumenta.

4.4.2 Implementação

A implementação deste exercício seguiu a mesma lógica que os exercícios anteriores:

$$C = 10\,\text{Mbps}, \quad f = 1{,}000{,}000\,\text{Bytes}, \quad \lambda = 1500\,\text{pps}, \quad P = 10^5, \quad N = 50, \quad b = 10^{-}5$$

O throughput total foi calculado como

$$TT = \frac{Total\ de\ bits\ transmitidos\ com\ sucesso}{Tempo\ total\ de\ simulação} \quad [\text{Mb/s}]$$

4.4.3 Resultados

A figura 32 apresenta a evolução do throughput total em função do número de fluxos VoIP.

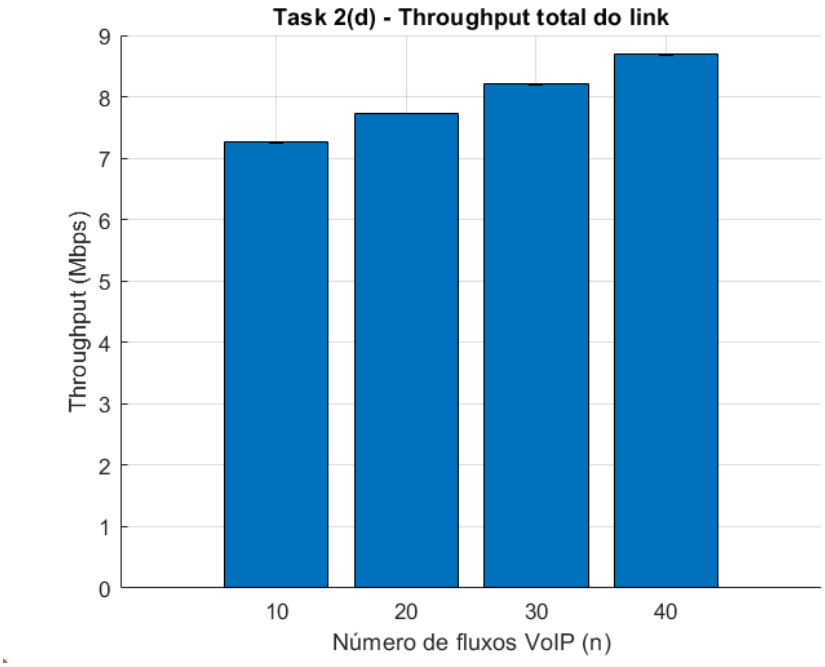

Figure 32: 2(d) – Throughput total da ligação em função do número de fluxos VoIP (n).

Os resultados mostram que o throughput total aumenta progressivamente com o número de fluxos VoIP, passando de cerca de 7,2 Mbps (n = 10) até, aproximadamente, 8,7 Mbps (n = 40).

Este comportamento confirma que o aumento do número de fluxos VoIP contribui para uma maior ocupação do canal.

Como o canal possui uma capacidade de 10 Mbos, o sistema aproxima-se gradualmente do seu limite máximo à medida que novos fluxos são introduzidos, mantendo-se contudo abaixo da saturação, o que significa que a rede consegue aguentar com até 40 fluxos VoIP de forma confortável, mas que para fluxos maiores seria necessário aumentar a capacidade da ligação de forma a evitar o congestionamento da rede.

Podemos concluir que o throuput total da ligação aumenta com o número de fluxos VoIP, refletindo um maior aproveitamento da capacidade do canal. Podemos concluir também que o sistema se mantém enficiente e estável até n = 40 fluxos sem atingir a sua saturação e que a BER introduz uma perda marginal, mas o seu impacto é pequeno face ao aumento de tráfego.

4.5 Exercício 2e

4.5.1 Contexto

Nesta alínea pretende-se obter, por análise teórica, o throughput total da ligação para os diferentes valores de n (número de fluxos VoIP), e compará-lo com os valores obtidos por simulação no exercício 2d).

Enquanto que no exercício 2d) o throughput foi medido diretamente pelo Simulator3A, nesta alínea calcula-se com base no modelo de tráfego das duas classes existentes:

- tráfego de dados modelado com um processo de chegadas Poisson com taxa $\lambda = 1500pps$ e distribuição de tamanhos de pacotes genérica;
- tráfego VoIP, composto por n fluxos independentes, cada um gerando pacotes pequenos e frequentes, com intervalos de chegada aproximadamente uniformes.

4.5.2 Implementação

O cálculo teórico do throughput é feito em duas partes:

1. contribuição do tráfego de dados,
2. contribuição do tráfego VoIP

O throughput total é a soma das duas contribuições.

1. Tráfego de dados Para o tráfego de dados, assume-se:
   - taxa de chegada média $\lambda = 1500$ pacotes/segundo;
   - distribuição de tamanhos de pacote igual à usada no simulador (a mesma dos exercícios anteriores):
     - 64 Bytes com probabilidade 0.19;
     - 110 Bytes com probabilidade 0.23;
     - 1518 Bytes com probablilidade 0.17;
     - os restantes valores são distribuídos uniformemente com a probabildade restante.

```
prob_elements = (1 - (0.19 + 0.23 + 0.17)) / ((109 - 65 + 1) +
(1517 - 111 + 1));

avg_packet_size_data = 0.19*64 + 0.23*110 + 0.17*1518 +
sum((65:109)*(prob_elements)) + sum((111:1517)*(prob_elements));

lambda = 1500;

data_TT = lambda * avg_packet_size_data * 8 / 10^6;
```

2. Tráfego VoIP Cada fluxo VoIP gera tráfego com as seguintes características:

- tamanho de cafa pacote VoIP entre 110 e 130 Bytes:

$$avg\_packet\_size\_voip = \frac{110 + 130}{2} Bytes$$

- intervalo entre pacotes de cada fluxo VoIP uniforme entre 16 ms e 24 ms:

$$time\_voip \approx \frac{16 + 24}{2} = 20ms$$

Logo, cada fluxo VoIP individual envia em média:

$$rate\_single\_voip = \frac{1}{0.02} \approx 50\ pacotes/s$$

```
avg_packet_size_voip = ( 110 + 130 ) / 2;
time_voip = (16 + 24) / 2 / 10^3;
rate_single_voip = 1 / time_voip;
```

Se existirem n fluxos VoIP ativos, a taxa total de pacotes VoIp é:

```
total_voip_rate = n_voip * rate_single_voip;
voip_TT = total_voip_rate * avg_packet_size_voip * 8 / 10^6;
```

Isto devolve o throughput VoIP para o valor de n.

Finalmente, o débito total é a soma de ambos:

```
final_TT = data_TT + voip_TT;
fprintf('n= %d; Throughput = %.2f Mbps\n', n_voip, final_TT);
```

Na 33 é possível observar os valores teóricos do throughput obtidos:

```
n= 10; Throughput = 7.92 Mbps
n= 20; Throughput = 8.40 Mbps
n= 30; Throughput = 8.88 Mbps
n= 40; Throughput = 9.36 Mbps
```

Figure 33: 2e – Throughput total da ligação em função do número de fluxos VoIP (n) - Valores Teorícos.

4.5.3 Resultados

Na tabela 2 é possível observar e comparar os valores obtidos por simulação (exercício 2d) e de teoricamente (exercício 2e)

Com base na análise da Tabela 2 podemos verificar que

- ambos os métodos mostram que ao aumentar o número de fluxos VoIP, o throughput total do sistema também aumenta.

Table 2: Tabela de comparação entre os valores de Throughput (Mbps) obtidos através de simulação e os valores teóricos

Numero de Fluxos VoIP	**10**	**20**	**30**	**40**
Simulação	7.25	7.74	8.20	8.68
Teórico	7.92	8.40	8.88	9.36

- ambos mostram que o sistema se aproxima gradualmente da capacidade máxima do canal (10 Mbps), mas sem a atingir.

Dito isto, podemos concluir que:

- O throughput total do sistema cresce aproximadamente de forma linear com o número de fluxos VoIP, ou seja:

  Mais fluxos VoIP ⇒ mais pacotes por segundo ⇒ mais bits a serem transmitidos ⇒ maior débito agregado.

- O modelo teórico obtido no exercício 2e) fornece uma boa aproximação ao desempenho observado na simulação.

- A diferença entre os valores teóricos e os simulados traduzem o impacto prático do BER.

Para além disso, o throughput calculado de forma teórica assume que todos os pacotes gerados são transmitidos com sucesso, sem considerar nenhuma perda de pacotes. Em situações reais, como mostram os resultados da simulação, a perda de pacotes pode ocorrer, particularmente, em condições de carga mais elevadas - como quando fluxos adicionais de VoIP são introduzidos. Esta razão faz com que os valores teóricos sejam superiores aos obtidos na simulação, uma vez que nos cálculos teóricos não consideram pacotes que podem ser descartados durante a transmissão.

A Anexos

A.1 Simulator1A

```
function [PL , PL_64, PL_110, PL_1518, APD , APD_64, APD_110,
    APD_1518, MPD , TT] = Simulator1A(lambda,C,f,P)
% INPUT PARAMETERS:
%  lambda - packet rate (packets/sec)
%  C      - link bandwidth (Mbps)
%  f      - queue size (Bytes)
%  P      - number of packets (stopping criterium)
% OUTPUT PARAMETERS:
%  PL   - packet loss (%)
%  APD  - average packet delay (milliseconds)
%  MPD  - maximum packet delay (milliseconds)
%  TT   - transmitted throughput (Mbps)

%Events:
ARRIVAL= 0;        % Arrival of a packet
DEPARTURE= 1;      % Departure of a packet

%State variables:
STATE = 0;           % 0 - connection is free; 1 - connection
   is occupied
QUEUEOCCUPATION= 0; % Occupation of the queue (in Bytes)
QUEUE= [];           % Size and arriving time instant of each
   packet in the queue

%Statistical Counters:
TOTALPACKETS= 0;      % No. of packets arrived to the system
TOTALPACKETS_64 = 0;
TOTALPACKETS_110 = 0;
TOTALPACKETS_1518 = 0;

LOSTPACKETS= 0;       % No. of packets dropped due to buffer
   overflow
LOSTPACKETS_64 = 0;
LOSTPACKETS_110 = 0;
LOSTPACKETS_1518 = 0;

TRANSPACKETS= 0;      % No. of transmitted packets
TRANSPACKETS_64 = 0;
TRANSPACKETS_110 = 0;
TRANSPACKETS_1518 = 0;

TRANSBYTES= 0;        % Sum of the Bytes of transmitted
   packets
```

```
DELAYS= 0;              % Sum of the delays of transmitted
   packets
DELAYS_64 = 0;
DELAYS_110 = 0;
DELAYS_1518 = 0;

MAXDELAY= 0;            % Maximum delay among all transmitted
   packets

% Initializing the simulation clock:
Clock= 0;

% Initializing the List of Events with the first ARRIVAL:
tmp= Clock + exprnd(1/lambda);
Event_List = [ARRIVAL, tmp, GenerateDataPacketSize(), tmp];

%Similation loop:
while TRANSPACKETS<P                        % Stopping criterium
    Event_List= sortrows(Event_List,2);  % Order EventList by
       time
    Event= Event_List(1,1);                    % Get first event
    Clock= Event_List(1,2);                    %    and all
    PacketSize= Event_List(1,3);               %    associated
    ArrInstant= Event_List(1,4);               %    parameters.
    Event_List(1,:)= [];                    % Eliminate first
       event
    switch Event
        case ARRIVAL          % If first event is an ARRIVAL
            TOTALPACKETS= TOTALPACKETS+1;
            if PacketSize == 64
                TOTALPACKETS_64 = TOTALPACKETS_64 + 1;
            elseif PacketSize == 110
                TOTALPACKETS_110 = TOTALPACKETS_110 + 1;
            elseif PacketSize == 1518
                TOTALPACKETS_1518 = TOTALPACKETS_1518 + 1;
            end

            tmp= Clock + exprnd(1/lambda);
            Event_List = [Event_List; ARRIVAL, tmp,
               GenerateDataPacketSize(), tmp];
            if STATE==0
                STATE= 1;
                Event_List = [Event_List; DEPARTURE, Clock +
                   8*PacketSize/(C*1e6), PacketSize, Clock];
            else
                if QUEUEOCCUPATION + PacketSize <= f
                    QUEUE= [QUEUE;PacketSize , Clock];
```

```
                    QUEUEOCCUPATION= QUEUEOCCUPATION +
                       PacketSize;
                else
                    LOSTPACKETS= LOSTPACKETS + 1;
                    if PacketSize == 64
                        LOSTPACKETS_64 = LOSTPACKETS_64 + 1;
                    elseif PacketSize == 110
                        LOSTPACKETS_110 = LOSTPACKETS_110 +
                           1;
                    elseif PacketSize == 1518
                        LOSTPACKETS_1518 = LOSTPACKETS_1518 +
                            1;
                    end
                end
            end
        case DEPARTURE          % If first event is a
           DEPARTURE
            TRANSBYTES= TRANSBYTES + PacketSize;
            DELAYS= DELAYS + (Clock - ArrInstant);
            if Clock - ArrInstant > MAXDELAY
                MAXDELAY= Clock - ArrInstant;
            end
            TRANSPACKETS= TRANSPACKETS + 1;
            if PacketSize == 64
                TRANSPACKETS_64 = TRANSPACKETS_64 + 1;
                DELAYS_64 = DELAYS_64 + (Clock - ArrInstant);
            elseif PacketSize == 110
                TRANSPACKETS_110 = TRANSPACKETS_110 + 1;
                DELAYS_110 = DELAYS_110 + (Clock - ArrInstant
                   );
            elseif PacketSize == 1518
                TRANSPACKETS_1518 = TRANSPACKETS_1518 + 1;
                DELAYS_1518 = DELAYS_1518 + (Clock -
                   ArrInstant);
            end

            if QUEUEOCCUPATION > 0
                QSize= QUEUE(1,1);
                QInstant= QUEUE(1,2);
                Event_List = [Event_List; DEPARTURE, Clock +
                   8*QSize/(C*1e6), QSize, QInstant];
                QUEUEOCCUPATION= QUEUEOCCUPATION - QSize;
                QUEUE(1,:)= [];
            else
                STATE= 0;
            end
    end
end
```

```
%Performance parameters determination:
PL= 100*LOSTPACKETS/TOTALPACKETS;  % in percentage
APD= 1000*DELAYS/TRANSPACKETS;     % in milliseconds

PL_64= 100*LOSTPACKETS_64/TOTALPACKETS_64;  % in percentage
APD_64= 1000*DELAYS_64/TRANSPACKETS_64;     % in milliseconds

PL_110= 100*LOSTPACKETS_110/TOTALPACKETS_110;  % in
   percentage
APD_110= 1000*DELAYS_110/TRANSPACKETS_110;     % in
   milliseconds

PL_1518= 100*LOSTPACKETS_1518/TOTALPACKETS_1518;  % in
   percentage
APD_1518= 1000*DELAYS_1518/TRANSPACKETS_1518;     % in
   milliseconds

MPD= 1000*MAXDELAY;                % in milliseconds
TT= 1e-6*TRANSBYTES*8/Clock;       % in Mbps

end

function out= GenerateDataPacketSize()
    aux= rand();
    aux2= [65:109 111:1517];
    if aux <= 0.19
        out= 64;
    elseif aux <= 0.19 + 0.23
        out= 110;
    elseif aux <= 0.19 + 0.23 + 0.17
        out= 1518;
    else
        out = aux2(randi(length(aux2)));
    end
end
```

A.2 Simulator1B

```
    function [PL , PL_64, PL_110, PL_1518, APD , APD_64,
       APD_110, APD_1518, MPD , TT] = Simulator1B(lambda,C,f,
       P)
% INPUT PARAMETERS:
%  lambda - packet rate (packets/sec)
%  C      - link bandwidth (Mbps)
%  f      - queue size (Bytes)
%  P      - number of packets (stopping criterium)
```

```
% OUTPUT PARAMETERS:
%  PL   - packet loss (%)
%  APD  - average packet delay (milliseconds)
%  MPD  - maximum packet delay (milliseconds)
%  TT   - transmitted throughput (Mbps)

%Events:
ARRIVAL= 0;        % Arrival of a packet
DEPARTURE= 1;      % Departure of a packet

%State variables:
STATE = 0;           % 0 - connection is free; 1 - connection
   is occupied
QUEUEOCCUPATION= 0; % Occupation of the queue (in Bytes)
QUEUE= [];           % Size and arriving time instant of each
   packet in the queue
QUEUE1 = [];         % PRIORITY1
QUEUE2 = [];         % PRIORITY2
QUEUE3 = [];         % PRIORITY3

%Statistical Counters:
TOTALPACKETS= 0;      % No. of packets arrived to the system
TOTALPACKETS_64 = 0;
TOTALPACKETS_110 = 0;
TOTALPACKETS_1518 = 0;

LOSTPACKETS= 0;       % No. of packets dropped due to buffer
   overflow
LOSTPACKETS_64 = 0;
LOSTPACKETS_110 = 0;
LOSTPACKETS_1518 = 0;

TRANSPACKETS= 0;      % No. of transmitted packets
TRANSPACKETS_64 = 0;
TRANSPACKETS_110 = 0;
TRANSPACKETS_1518 = 0;

TRANSBYTES= 0;        % Sum of the Bytes of transmitted
   packets

DELAYS= 0;            % Sum of the delays of transmitted
   packets
DELAYS_64 = 0;
DELAYS_110 = 0;
DELAYS_1518 = 0;

MAXDELAY= 0;          % Maximum delay among all transmitted
   packets
```

```
% Initializing the simulation clock:
Clock= 0;

% Initializing the List of Events with the first ARRIVAL:
tmp= Clock + exprnd(1/lambda);
Event_List = [ARRIVAL, tmp, GenerateDataPacketSize(), tmp];

%Similation loop:
while TRANSPACKETS<P                          % Stopping criterium
    Event_List= sortrows(Event_List,2);  % Order EventList by
        time
    Event= Event_List(1,1);                    % Get first event
    Clock= Event_List(1,2);                    %    and all
    PacketSize= Event_List(1,3);               %    associated
    ArrInstant= Event_List(1,4);               %    parameters.
    Event_List(1,:)= [];                      % Eliminate first
       event
    switch Event
        case ARRIVAL          % If first event is an ARRIVAL
            TOTALPACKETS= TOTALPACKETS+1;
            if PacketSize == 64
                TOTALPACKETS_64 = TOTALPACKETS_64 + 1;
            elseif PacketSize == 110
                TOTALPACKETS_110 = TOTALPACKETS_110 + 1;
            elseif PacketSize == 1518
                TOTALPACKETS_1518 = TOTALPACKETS_1518 + 1;
            end

            tmp= Clock + exprnd(1/lambda);
            Event_List = [Event_List; ARRIVAL, tmp,
               GenerateDataPacketSize(), tmp];
            if STATE==0
                STATE= 1;
                Event_List = [Event_List; DEPARTURE, Clock +
                   8*PacketSize/(C*1e6), PacketSize, Clock];
            else
                if QUEUEOCCUPATION + PacketSize <= f
                    priority = priorityOf(PacketSize);
                    if priority == 1
                        QUEUE1 = [QUEUE1; PacketSize, Clock];
                    elseif priority == 2
                        QUEUE2 = [QUEUE2; PacketSize, Clock];
                    elseif priority == 3
                        QUEUE3 = [QUEUE3; PacketSize, Clock];
                    end
                    QUEUEOCCUPATION= QUEUEOCCUPATION +
                       PacketSize;
```

```
                else
                    LOSTPACKETS= LOSTPACKETS + 1;
                    if PacketSize == 64
                        LOSTPACKETS_64 = LOSTPACKETS_64 + 1;
                    elseif PacketSize == 110
                        LOSTPACKETS_110 = LOSTPACKETS_110 +
                           1;
                    elseif PacketSize == 1518
                        LOSTPACKETS_1518 = LOSTPACKETS_1518 +
                            1;
                    end
                end
            end
        case DEPARTURE          % If first event is a
           DEPARTURE
            TRANSBYTES= TRANSBYTES + PacketSize;
            DELAYS= DELAYS + (Clock - ArrInstant);
            if Clock - ArrInstant > MAXDELAY
                MAXDELAY= Clock - ArrInstant;
            end
            TRANSPACKETS= TRANSPACKETS + 1;
            if PacketSize == 64
                TRANSPACKETS_64 = TRANSPACKETS_64 + 1;
                DELAYS_64 = DELAYS_64 + (Clock - ArrInstant);
            elseif PacketSize == 110
                TRANSPACKETS_110 = TRANSPACKETS_110 + 1;
                DELAYS_110 = DELAYS_110 + (Clock - ArrInstant
                   );
            elseif PacketSize == 1518
                TRANSPACKETS_1518 = TRANSPACKETS_1518 + 1;
                DELAYS_1518 = DELAYS_1518 + (Clock -
                   ArrInstant);
            end

            if QUEUEOCCUPATION > 0
                if ~isempty(QUEUE1)
                    QSize = QUEUE1(1,1);
                    QInstant = QUEUE1(1,2);
                    QUEUE1(1, :) = [];
                elseif ~isempty(QUEUE2)
                    QSize = QUEUE2(1,1);
                    QInstant = QUEUE2(1,2);
                    QUEUE2(1, :) = [];
                elseif ~isempty(QUEUE3)
                    QSize = QUEUE3(1,1);
                    QInstant = QUEUE3(1,2);
                    QUEUE3(1, :) = [];
                end
```

```
                QUEUEOCCUPATION= QUEUEOCCUPATION - QSize;
                Event_List = [Event_List; DEPARTURE, Clock +
                   8*QSize/(C*1e6), QSize, QInstant];
            else
                STATE= 0;
                continue;
            end
    end
end

%Performance parameters determination:
PL= 100*LOSTPACKETS/TOTALPACKETS;  % in percentage
APD= 1000*DELAYS/TRANSPACKETS;     % in milliseconds

PL_64= 100*LOSTPACKETS_64/TOTALPACKETS_64;  % in percentage
APD_64= 1000*DELAYS_64/TRANSPACKETS_64;     % in milliseconds

PL_110= 100*LOSTPACKETS_110/TOTALPACKETS_110;  % in
   percentage
APD_110= 1000*DELAYS_110/TRANSPACKETS_110;     % in
   milliseconds

PL_1518= 100*LOSTPACKETS_1518/TOTALPACKETS_1518;  % in
   percentage
APD_1518= 1000*DELAYS_1518/TRANSPACKETS_1518;     % in
   milliseconds

MPD= 1000*MAXDELAY;                % in milliseconds
TT= 1e-6*TRANSBYTES*8/Clock;       % in Mbps

end

function out= GenerateDataPacketSize()
    aux= rand();
    aux2= [65:109 111:1517];
    if aux <= 0.19
        out= 64;
    elseif aux <= 0.19 + 0.23
        out= 110;
    elseif aux <= 0.19 + 0.23 + 0.17
        out= 1518;
    else
        out = aux2(randi(length(aux2)));
    end
end

function priority = priorityOf(size)
    if size >= 1501
```

```
        priority = 1;
    elseif size <= 1500
        priority = 2;
    else
        priority = 3;
    end
end
```

A.3 Simulator3A

```
    function [PLd , APDd , MPDd , PLv , APDv , MPDv, TT] =
       Simulator3A(lambda,C,f,P, n, b)
% INPUT PARAMETERS:
%  lambda - packet rate (packets/sec)
%  C      - link bandwidth (Mbps)
%  f      - queue size (Bytes)
%  P      - number of packets (stopping criterium)
%  b      - bit error
% OUTPUT PARAMETERS:
%  PL   - packet loss (%)
%  APD  - average packet delay (milliseconds)
%  MPD  - maximum packet delay (milliseconds)
%  TT   - transmitted throughput (Mbps)

%Events:
ARRIVAL= 0;        % Arrival of a packet
DEPARTURE= 1;      % Departure of a packet

% Packet type:
DATA = 0;
VoIP = 1;

%State variables:
STATE = 0;           % 0 - connection is free; 1 - connection
   is occupied
QUEUEOCCUPATION= 0; % Occupation of the queue (in Bytes)
QUEUE= [];           % Size and arriving time instant of each
   packet in the queue

%Statistical Counters:
TOTALPACKETSd= 0;      % No. of packets arrived to the system
LOSTPACKETSd= 0;       % No. of packets dropped due to buffer
   overflow
TRANSPACKETSd= 0;      % No. of transmitted packets
TRANSBYTESd= 0;        % Sum of the Bytes of transmitted
   packets
```

```
DELAYSd= 0;             % Sum of the delays of transmitted
   packets
MAXDELAYd= 0;           % Maximum delay among all transmitted
   packets

TOTALPACKETSv= 0;       % No. of packets arrived to the system
LOSTPACKETSv= 0;        % No. of packets dropped due to buffer
   overflow
TRANSPACKETSv= 0;       % No. of transmitted packets
TRANSBYTESv= 0;         % Sum of the Bytes of transmitted
   packets
DELAYSv= 0;             % Sum of the delays of transmitted
   packets
MAXDELAYv= 0;           % Maximum delay among all transmitted
   packets

% Initializing the simulation clock:
Clock= 0;

% Initializing the List of Events with the first ARRIVAL of
   DATA packets:
tmp= Clock + exprnd(1/lambda);
Event_List = [ARRIVAL, tmp, GenerateDataPacketSize(), tmp,
   DATA];

% Initializing the List of Events with the first ARRIVAL OF
   VoIP Packets
for i = 1:n
    tmp = unifrnd(0, 0.02);
    Event_List = [Event_List; ARRIVAL, tmp, randi([110, 130])
       , tmp, VoIP];
end

%Similation loop:
while (TRANSPACKETSd + TRANSPACKETSv) <P
   % Stopping criterium
    Event_List= sortrows(Event_List,2);  % Order EventList by
       time
    Event= Event_List(1,1);                 % Get first event
    Clock= Event_List(1,2);                 %    and all
    PacketSize= Event_List(1,3);            %    associated
    ArrInstant= Event_List(1,4);            %    parameters.
    PacketType = Event_List(1,5);
    Event_List(1,:)= [];                  % Eliminate first
       event
    switch Event
        case ARRIVAL          % If first event is an ARRIVAL
            if (PacketType == DATA)
```

```
                TOTALPACKETSd= TOTALPACKETSd+1;
                tmp= Clock + exprnd(1/lambda);
                Event_List = [Event_List; ARRIVAL, tmp,
                   GenerateDataPacketSize(), tmp, DATA];
                if STATE==0
                    STATE= 1;
                    Event_List = [Event_List; DEPARTURE,
                       Clock + 8*PacketSize/(C*1e6),
                       PacketSize, Clock, DATA];
                else
                    if QUEUEOCCUPATION + PacketSize <= f
                        QUEUE= [QUEUE;PacketSize , Clock,
                           DATA];
                        QUEUEOCCUPATION= QUEUEOCCUPATION +
                           PacketSize;
                    else
                        LOSTPACKETSd= LOSTPACKETSd + 1;
                    end
                end
            else
                TOTALPACKETSv= TOTALPACKETSv+1;
                tmp= Clock + unifrnd(0.016, 0.024);
                Event_List = [Event_List; ARRIVAL, tmp, randi
                   ([110, 130]), tmp, VoIP];
                if STATE==0
                    STATE= 1;
                    Event_List = [Event_List; DEPARTURE,
                       Clock + 8*PacketSize/(C*1e6),
                       PacketSize, Clock, VoIP];
                else
                    if QUEUEOCCUPATION + PacketSize <= f
                        QUEUE= [QUEUE;PacketSize , Clock,
                           VoIP];
                        QUEUEOCCUPATION= QUEUEOCCUPATION +
                           PacketSize;
                    else
                        LOSTPACKETSv= LOSTPACKETSv + 1;
                    end
                end

            end

        case DEPARTURE          % If first event is a
           DEPARTURE
            if(rand() < (1-b)^(PacketSize *8))          %
               chegar sem erros
                if (PacketType == DATA)
                    TRANSBYTESd= TRANSBYTESd + PacketSize;
```

```
                    DELAYSd= DELAYSd + (Clock - ArrInstant);
                    if Clock - ArrInstant > MAXDELAYd
                        MAXDELAYd= Clock - ArrInstant;
                    end
                    TRANSPACKETSd= TRANSPACKETSd + 1;
                else
                   TRANSBYTESv= TRANSBYTESv + PacketSize;
                   DELAYSv= DELAYSv + (Clock - ArrInstant);
                   if Clock - ArrInstant > MAXDELAYv
                       MAXDELAYv= Clock - ArrInstant;
                   end
                   TRANSPACKETSv= TRANSPACKETSv + 1;
                end
            else                                          % chegar
               com erros
                if (PacketType == DATA)
                    LOSTPACKETSd = LOSTPACKETSd + 1;
                else
                    LOSTPACKETSv = LOSTPACKETSv + 1;
                end
            end

            if QUEUEOCCUPATION > 0
                QSize= QUEUE(1,1);
                QInstant= QUEUE(1,2);
                Qso = QUEUE(1,3);
                Event_List = [Event_List; DEPARTURE, Clock +
                   8*QSize/(C*1e6), QSize, QInstant, Qso];
                QUEUEOCCUPATION= QUEUEOCCUPATION - QSize;
                QUEUE(1,:)= [];
            else
                STATE= 0;
            end
    end
end

%Performance parameters determination:
PLd= 100*LOSTPACKETSd/TOTALPACKETSd;  % in percentage
APDd= 1000*DELAYSd/TRANSPACKETSd;     % in milliseconds
MPDd= 1000*MAXDELAYd;                % in milliseconds

PLv= 100*LOSTPACKETSv/TOTALPACKETSv;  % in percentage
APDv= 1000*DELAYSv/TRANSPACKETSv;     % in milliseconds
MPDv= 1000*MAXDELAYv;                % in milliseconds

TT= 1e-6*(TRANSBYTESd + TRANSBYTESv)*8/Clock;       % in Mbps

end
```

```
function out= GenerateDataPacketSize()
    aux= rand();
    aux2= [65:109 111:1517];
    if aux <= 0.19
        out= 64;
    elseif aux <= 0.19 + 0.23
        out= 110;
    elseif aux <= 0.19 + 0.23 + 0.17
        out= 1518;
    else
        out = aux2(randi(length(aux2)));
    end
end
```

A.4 Task 1

```
    %% Task 1 a)
fprintf('TASK 1a)');
P = 1e5;       % stopping criteria
N = 50;          % number of runs
alfa = 0.1;      % 90% confidence intervals
C = 10;          % C = 10 Mbps
f = 1e6;
lambda = [1100 1300 1500 1700 1900];

PL = zeros(1, N);
APD = zeros(1, N);
MPD = zeros(1, N);
TT = zeros(1, N);

PL_values = zeros(1, length(lambda));
PL_term = zeros(1, length(lambda));
APD_values = zeros(1, length(lambda));
APD_term = zeros(1, length(lambda));

for i = 1:length(lambda)
    for it = 1:N
        [PL(it), APD(it), MPD(it), TT(it)] = Simulator1(
           lambda(i),C,f,P);
    end

    fprintf('Valor de lambda: %d\n', lambda(i));
    mediaPL = mean(PL);
    termPL = norminv(1-alfa/2) * sqrt(var(PL)/N);
    PL_values(i) = mediaPL;
    PL_term(i) = termPL;
```

```
    fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL,
       termPL);

    mediaAPD = mean(APD);
    termAPD = norminv(1-alfa/2) * sqrt(var(APD)/N);
    APD_values(i) = mediaAPD;
    APD_term(i) = termAPD;
    fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
       mediaAPD, termAPD);
end

fprintf('\nSimulation ended!\n');



figure(1);
hold on;
grid on;
bar(lambda, PL_values);
er = errorbar(lambda, PL_values, PL_term);
er.Color = [0 0 0];
er.LineStyle = 'none';
xlabel('Packet Rate (pps)');
xticks(lambda);
ylabel('Packet Loss (%)');
title ('Task 1(a) - Packet Loss vs Packet Rate');
hold off

figure(2);
hold on;
grid on;
bar(lambda, APD_values);
ylim([0 9]);
er = errorbar(lambda, APD_values, APD_term);
er.Color = [0 0 0];
er.LineStyle = 'none';
xlabel('Packet Rate (pps)');
xticks(lambda);
ylabel('Average packet delay (ms)');
title('Task 1(a) - Average Packet Delay vs. Packet Rate');
hold off

%% Task 1b)
fprintf('TASK 1b)\n');
P = 1e5;          % stopping criteria
N = 50;           % number of runs
alfa = 0.1;       % 90% confidence intervals
C = 10;           % C = 10 Mbps
```

```
f = 1e4;
lambda = [1100 1300 1500 1700 1900];

PL = zeros(1, N);
APD = zeros(1, N);
MPD = zeros(1, N);
TT = zeros(1, N);

PL_values = zeros(1, length(lambda));
PL_term = zeros(1, length(lambda));
APD_values = zeros(1, length(lambda));
APD_term = zeros(1, length(lambda));

for i = 1:length(lambda)
    for it = 1:N
        [PL(it), APD(it), MPD(it), TT(it)] = Simulator1(
           lambda(i),C,f,P);
    end

    fprintf('Valor de lambda: %d\n', lambda(i));
    mediaPL = mean(PL);
    termPL = norminv(1-alfa/2) * sqrt(var(PL)/N);
    PL_values(i) = mediaPL;
    PL_term(i) = termPL;
    fprintf('Packet Loss (%%) = %.2e +- %.2e\n', mediaPL,
       termPL);

    mediaAPD = mean(APD);
    termAPD = norminv(1-alfa/2) * sqrt(var(APD)/N);
    APD_values(i) = mediaAPD;
    APD_term(i) = termAPD;
    fprintf('Average Packet Delay (ms) = %.2e +- %.2e\n',
       mediaAPD, termAPD);
end

fprintf('\nSimulation ended!\n');

figure(1);
hold on;
grid on;
bar(lambda, PL_values);
er = errorbar(lambda, PL_values, PL_term);
er.Color = [0 0 0];
er.LineStyle = 'none';
xlabel('Packet Rate (pps)');
xticks(lambda);
ylabel('Packet Loss (%)');
title ('Task 1(b) - Packet Loss vs Packet Rate');
```

```
hold off

figure(2);
hold on;
grid on;
bar(lambda, APD_values);
ylim([0 9]);
er = errorbar(lambda, APD_values, APD_term);
er.Color = [0 0 0];
er.LineStyle = 'none';
xlabel('Packet Rate (pps)');
xticks(lambda);
ylabel('Average packet delay (ms)');
title('Task 1(b) - Average Packet Delay vs. Packet Rate');
hold off

%% TASK 1d
fprintf('TASK 1d)\n');
P = 1e5;          % stopping criteria
N = 50;           % number of runs
alfa = 0.1;       % 90% confidence intervals
C = 10;           % C = 10 Mbps
f = 1e6;
lambda = 1900;

PL = zeros(1, N);
PL_64 = zeros(1, N);
PL_110 = zeros(1, N);
PL_1518 = zeros(1, N);
APD = zeros(1, N);
APD_64 = zeros(1, N);
APD_110 = zeros(1, N);
APD_1518 = zeros(1, N);
MPD = zeros(1, N);
TT = zeros(1, N);

for it = 1:N
    [PL(it) , PL_64(it), PL_110(it), PL_1518(it), APD(it) ,
       APD_64(it), APD_110(it), APD_1518(it), MPD(it) , TT(it
       )] = Simulator1A(lambda,C,f,P);
end

mediaPL = mean(PL);
termPL = norminv(1-alfa/2) * sqrt(var(PL)/N);
fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL,
   termPL);

mediaAPD = mean(APD);
```

```
termAPD = norminv(1-alfa/2) * sqrt(var(APD)/N);
fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
   mediaAPD, termAPD);

fprintf('Packet Size = 64 bytes\n');
mediaPL_64 = mean(PL_64);
termPL_64 = norminv(1-alfa/2) * sqrt(var(PL_64)/N);
fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL_64,
   termPL_64);

mediaAPD_64 = mean(APD_64);
termAPD_64 = norminv(1-alfa/2) * sqrt(var(APD_64)/N);
fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
   mediaAPD_64, termAPD_64);

fprintf('Packet Size = 110 bytes\n');
mediaPL_110 = mean(PL_110);
termPL_110 = norminv(1-alfa/2) * sqrt(var(PL_110)/N);
fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL_110,
   termPL_110);

mediaAPD_110 = mean(APD_110);
termAPD_110 = norminv(1-alfa/2) * sqrt(var(APD_110)/N);
fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
   mediaAPD_110, termAPD_110);

fprintf('Packet Size = 1518 bytes\n');
mediaPL_1518 = mean(PL_1518);
termPL_1518 = norminv(1-alfa/2) * sqrt(var(PL_1518)/N);
fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL_1518,
    termPL_1518);

mediaAPD_1518 = mean(APD_1518);
termAPD_1518 = norminv(1-alfa/2) * sqrt(var(APD_1518)/N);
fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
   mediaAPD_1518, termAPD_1518);


fprintf('\nSimulation ended!\n');


PL_means  = [mediaPL,  mediaPL_64,  mediaPL_110,
   mediaPL_1518];
PL_terms  = [termPL,   termPL_64,   termPL_110,   termPL_1518
   ];
APD_means = [mediaAPD, mediaAPD_64, mediaAPD_110,
   mediaAPD_1518];
```

```
APD_terms = [termAPD, termAPD_64, termAPD_110,
   termAPD_1518];

figure(1);
hold on;
grid on;
bar(PL_means);
er = errorbar(1:4, PL_means, PL_terms);
er.Color = [0 0 0];
er.LineStyle = 'none';
xticks([1 2 3 4])
xticklabels({'All','64 Bytes','110 Bytes','1518 Bytes'})
xlabel('Packet Size')
ylabel('Packet loss (%)');
title('Task 1(d)     Packet Loss per Packet Size')

hold off;

figure(2);
hold on;
grid on;
bar(APD_means);
ylim([0 9]);
er = errorbar(1:4, APD_means, APD_terms);
er.Color = [0 0 0];
er.LineStyle = 'none';
ylabel('Average packet delay (ms)');
xticks([1 2 3 4])
xticklabels({'All','64 Bytes','110 Bytes','1518 Bytes'})
xlabel('Packet Size')
title('Task 1(d)     Average Packet Delay per Packet Size');
hold off;

%% Task 1e - Te rica - n o sei como fazer ainda, mas deve
   ser adaptar este c digo!
P = 1e5;
N = 50;
C = 10;
f = 1e6;
lambda = 1900;
alfa = 0.1;

x = 64:1518;
prob_elements = (1 - 0.19 - 0.23 - 0.17) / ((109-65+1) +
   (1517-111+1));

avg_packet_size = 0.19*64 + 0.23*110 + 0.17*1518 + sum
   (65:109)* prob_elements + sum(111:1517)*prob_elements;
```

```
avg_time = (avg_packet_size * 8) / (C * 10^6);

S = x .*8 / (C * 10^6);
S2 = S.^2;

for i = 1:length(x)
    if i == 1
       p = 0.19;
    elseif i == 110-64+1
       p = 0.23;
    elseif i == 1518-64+1
        p = 0.17;
    else
       p = prob_elements;
    end
    S(i) = S(i) * p;
    S2(i) = S2(i) * p;
end

ES = sum(S);
ES2 = sum(S2);

W = (lambda * ES2) / (2*(1 - lambda*ES)) + ES;
Wq = (lambda * ES2) / (2*(1 - lambda*ES));

S64 = (64 * 8) / (C * 1e6);
S110 = (110 * 8) / (C * 1e6);
S1518 = (1518 * 8) / (C * 1e6);

W64 = Wq + S64;
W110 = Wq + S110;
W1518 = Wq + S1518;

TT = lambda * avg_packet_size * 8 / 10^6;

% Lista de espera infinita - n o tem packet loss.
fprintf('PL_all (%%)      = 0.0000\n');
fprintf('PL_64B (%%)      = 0.0000\n');
fprintf('PL_110B (%%)     = 0.0000\n');
fprintf('PL_1518B (%%)    = 0.0000\n');
fprintf('Delay all (ms)   = %.4f\n', W*1000);
fprintf('Delay 64B (ms)   = %.4f\n', W64*1000);
fprintf('Delay 110B (ms)  = %.4f\n', W110*1000);
fprintf('Delay 1518B (ms) = %.4f\n', W1518*1000);
fprintf('Throughput (Mb/s)= %.4f\n', TT);


%% Task 1f
```

```
fprintf('TASK 1f)\n');
P = 1e5;          % stopping criteria
N = 50;           % number of runs
alfa = 0.1;       % 90% confidence intervals
C = 10;           % C = 10 Mbps
f = 1e4;
lambda = 1900;

PL = zeros(1, N);
PL_64 = zeros(1, N);
PL_110 = zeros(1, N);
PL_1518 = zeros(1, N);
APD = zeros(1, N);
APD_64 = zeros(1, N);
APD_110 = zeros(1, N);
APD_1518 = zeros(1, N);
MPD = zeros(1, N);
TT = zeros(1, N);

for it = 1:N
    [PL(it) , PL_64(it), PL_110(it), PL_1518(it), APD(it) ,
       APD_64(it), APD_110(it), APD_1518(it), MPD(it) , TT(it
       )] = Simulator1A(lambda,C,f,P);
end


mediaPL = mean(PL);
termPL = norminv(1-alfa/2) * sqrt(var(PL)/N);
fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL,
   termPL);

mediaAPD = mean(APD);
termAPD = norminv(1-alfa/2) * sqrt(var(APD)/N);
fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
   mediaAPD, termAPD);

fprintf('Packet Size = 64 bytes\n');
mediaPL_64 = mean(PL_64);
termPL_64 = norminv(1-alfa/2) * sqrt(var(PL_64)/N);
fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL_64,
   termPL_64);

mediaAPD_64 = mean(APD_64);
termAPD_64 = norminv(1-alfa/2) * sqrt(var(APD_64)/N);
fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
   mediaAPD_64, termAPD_64);

fprintf('Packet Size = 110 bytes\n');
```

```
mediaPL_110 = mean(PL_110);
termPL_110 = norminv(1-alfa/2) * sqrt(var(PL_110)/N);
fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL_110,
   termPL_110);

mediaAPD_110 = mean(APD_110);
termAPD_110 = norminv(1-alfa/2) * sqrt(var(APD_110)/N);
fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
   mediaAPD_110, termAPD_110);

fprintf('Packet Size = 1518 bytes\n');
mediaPL_1518 = mean(PL_1518);
termPL_1518 = norminv(1-alfa/2) * sqrt(var(PL_1518)/N);
fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL_1518,
    termPL_1518);

mediaAPD_1518 = mean(APD_1518);
termAPD_1518 = norminv(1-alfa/2) * sqrt(var(APD_1518)/N);
fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
   mediaAPD_1518, termAPD_1518);


fprintf('\nSimulation ended!\n');


PL_means  = [mediaPL,  mediaPL_64,  mediaPL_110,
   mediaPL_1518];
PL_terms  = [termPL,   termPL_64,   termPL_110,   termPL_1518
   ];
APD_means = [mediaAPD, mediaAPD_64, mediaAPD_110,
   mediaAPD_1518];
APD_terms = [termAPD,  termAPD_64,  termAPD_110,
   termAPD_1518];



figure(1);
hold on;
grid on;
bar(PL_means);
er = errorbar(1:4, PL_means, PL_terms);
er.Color = [0 0 0];
er.LineStyle = 'none';
ylabel('Packet loss (%)');
xticks([1 2 3 4])
xticklabels({'All','64 Bytes','110 Bytes','1518 Bytes'})
xlabel('Packet Size')
title('Task 1(f)     Packet Loss per Packet Size');
```

```
hold off;

figure(2);
hold on;
grid on;
bar(APD_means);
er = errorbar(1:4, APD_means, APD_terms);
er.Color = [0 0 0];
er.LineStyle = 'none';
ylabel('Average packet delay (ms)');
xticks([1 2 3 4])
xticklabels({'All','64 Bytes','110 Bytes','1518 Bytes'})
xlabel('Packet Size')
title('Task 1(f)     Average Packet Delay per Packet Size');
hold off;


%% Task 1h
fprintf('TASK 1h)\n');
P = 1e5;     %stopping criteria
N = 50;          % number of runs
alfa = 0.1;      % 90% confidence intervals
C = 10;          % C = 10 Mbps
f = 1e6;
lambda = 1900;

PL = zeros(1, N);
PL_64 = zeros(1, N);
PL_110 = zeros(1, N);
PL_1518 = zeros(1, N);
APD = zeros(1, N);
APD_64 = zeros(1, N);
APD_110 = zeros(1, N);
APD_1518 = zeros(1, N);
MPD = zeros(1, N);
TT = zeros(1, N);

for it = 1:N
    [PL(it) , PL_64(it), PL_110(it), PL_1518(it), APD(it) ,
       APD_64(it), APD_110(it), APD_1518(it), MPD(it) , TT(it
       )] = Simulator1B(lambda,C,f,P);
end

mediaPL = mean(PL);
termPL = norminv(1-alfa/2) * sqrt(var(PL)/N);
fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL,
   termPL);
```

```
mediaAPD = mean(APD);
termAPD = norminv(1-alfa/2) * sqrt(var(APD)/N);
fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
   mediaAPD, termAPD);

fprintf('Packet Size = 64 bytes\n');
mediaPL_64 = mean(PL_64);
termPL_64 = norminv(1-alfa/2) * sqrt(var(PL_64)/N);
fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL_64,
   termPL_64);

mediaAPD_64 = mean(APD_64);
termAPD_64 = norminv(1-alfa/2) * sqrt(var(APD_64)/N);
fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
   mediaAPD_64, termAPD_64);

fprintf('Packet Size = 110 bytes\n');
mediaPL_110 = mean(PL_110);
termPL_110 = norminv(1-alfa/2) * sqrt(var(PL_110)/N);
fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL_110,
   termPL_110);

mediaAPD_110 = mean(APD_110);
termAPD_110 = norminv(1-alfa/2) * sqrt(var(APD_110)/N);
fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
   mediaAPD_110, termAPD_110);

fprintf('Packet Size = 1518 bytes\n');
mediaPL_1518 = mean(PL_1518);
termPL_1518 = norminv(1-alfa/2) * sqrt(var(PL_1518)/N);
fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL_1518,
    termPL_1518);

mediaAPD_1518 = mean(APD_1518);
termAPD_1518 = norminv(1-alfa/2) * sqrt(var(APD_1518)/N);
fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
   mediaAPD_1518, termAPD_1518);


fprintf('\nSimulation ended!\n');


PL_means  = [mediaPL,  mediaPL_64,  mediaPL_110,
   mediaPL_1518];
PL_terms  = [termPL,   termPL_64,   termPL_110,   termPL_1518
   ];
APD_means = [mediaAPD, mediaAPD_64, mediaAPD_110,
   mediaAPD_1518];
```

```
APD_terms = [termAPD, termAPD_64, termAPD_110,
   termAPD_1518];

figure(1);
hold on
grid on;
bar(PL_means);
er = errorbar(1:4, PL_means, PL_terms);
er.Color = [0 0 0];
er.LineStyle = 'none';
ylabel('Packet loss (%)');
title('Task 1(h)     Packet Loss per Packet Size (Priority
   Queuing)')
xlabel('Packet Size');
xticks(1:4);
xticklabels({'All','64 Bytes','110 Bytes','1518 Bytes'});
hold off;

figure(2);
hold on;
grid on;
bar(APD_means);
er = errorbar(1:4, APD_means, APD_terms);
er.Color = [0 0 0];
er.LineStyle = 'none';
ylabel('Average packet delay (ms)');
title('Task 1(h)     Average Packet Delay per Packet Size (
   Priority Queuing)')
xlabel('Packet Size');
xticks(1:4);
xticklabels({'All','64 Bytes','110 Bytes','1518 Bytes'});
hold off;

%% Task 1i
fprintf('TASK 1i)\n');
P = 1e5;     %stopping criteria
N = 50;          % number of runs
alfa = 0.1;      % 90% confidence intervals
C = 10;          % C = 10 Mbps
f = 1e4;
lambda = 1900;

PL = zeros(1, N);
PL_64 = zeros(1, N);
PL_110 = zeros(1, N);
PL_1518 = zeros(1, N);
APD = zeros(1, N);
APD_64 = zeros(1, N);
```

```
APD_110 = zeros(1, N);
APD_1518 = zeros(1, N);
MPD = zeros(1, N);
TT = zeros(1, N);

for it = 1:N
    [PL(it) , PL_64(it), PL_110(it), PL_1518(it), APD(it) ,
       APD_64(it), APD_110(it), APD_1518(it), MPD(it) , TT(it
       )] = Simulator1B(lambda,C,f,P);
end

mediaPL = mean(PL);
termPL = norminv(1-alfa/2) * sqrt(var(PL)/N);
fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL,
   termPL);

mediaAPD = mean(APD);
termAPD = norminv(1-alfa/2) * sqrt(var(APD)/N);
fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
   mediaAPD, termAPD);

fprintf('Packet Size = 64 bytes\n');
mediaPL_64 = mean(PL_64);
termPL_64 = norminv(1-alfa/2) * sqrt(var(PL_64)/N);
fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL_64,
   termPL_64);

mediaAPD_64 = mean(APD_64);
termAPD_64 = norminv(1-alfa/2) * sqrt(var(APD_64)/N);
fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
   mediaAPD_64, termAPD_64);

fprintf('Packet Size = 110 bytes\n');
mediaPL_110 = mean(PL_110);
termPL_110 = norminv(1-alfa/2) * sqrt(var(PL_110)/N);
fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL_110,
   termPL_110);

mediaAPD_110 = mean(APD_110);
termAPD_110 = norminv(1-alfa/2) * sqrt(var(APD_110)/N);
fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
   mediaAPD_110, termAPD_110);

fprintf('Packet Size = 1518 bytes\n');
mediaPL_1518 = mean(PL_1518);
termPL_1518 = norminv(1-alfa/2) * sqrt(var(PL_1518)/N);
fprintf('Packet Loss (%%)\t  = %.2e +- %.2e\n', mediaPL_1518,
    termPL_1518);
```

```
mediaAPD_1518 = mean(APD_1518);
termAPD_1518 = norminv(1-alfa/2) * sqrt(var(APD_1518)/N);
fprintf('Average Packet Delay (ms)\t = %.2e +- %.2e\n',
   mediaAPD_1518, termAPD_1518);


fprintf('\nSimulation ended!\n');


PL_means  = [mediaPL,  mediaPL_64,  mediaPL_110,
   mediaPL_1518];
PL_terms  = [termPL,   termPL_64,   termPL_110,   termPL_1518
   ];
APD_means = [mediaAPD, mediaAPD_64, mediaAPD_110,
   mediaAPD_1518];
APD_terms = [termAPD,  termAPD_64,  termAPD_110,
   termAPD_1518];

% Figure 1     Average Packet Loss (%)
figure(1);
hold on;
grid on;
bar(PL_means);
er = errorbar(1:4, PL_means, PL_terms);
er.Color = [0 0 0];
er.LineStyle = 'none';
ylabel('Packet loss (%)');
title('Task 1(i)     Packet Loss per Packet Size (Priority
   Queuing)')
xlabel('Packet Size');
xticks(1:4);
xticklabels({'All','64 Bytes','110 Bytes','1518 Bytes'});
hold off;

figure(2);
hold on;
grid on;
bar(APD_means);
er = errorbar(1:4, APD_means, APD_terms);
er.Color = [0 0 0];
er.LineStyle = 'none';
ylabel('Average packet delay (ms)');
title('Task 1(i)     Average Packet Delay per Packet Size (
   Priority Queuing)')
xlabel('Packet Size');
xticks(1:4);
xticklabels({'All','64 Bytes','110 Bytes','1518 Bytes'});
```

```
hold off;
```

A.5 Task 2

```
    %% Task 2b, c, d)

fprintf( Task2b, c, d) - Valores\n );
P = 1e5;       % stopping criteria
N = 50;          % number of runs
alfa = 0.1;      % 90% confidence intervals
C = 10;          % C = 10 Mbps
f = 1e6;
lambda = 1500;
b = 10^-5;
n = [10 20 30 40];

PLd = zeros(1,N);
APDd = zeros(1,N);
MPDd = zeros(1,N);

PLv = zeros(1,N);
APDv = zeros(1,N);
MPDv = zeros(1,N);

TT = zeros(1,N);

PLd_values = zeros(1, length(n));
PLd_term = zeros(1, length(n));
PLv_values = zeros(1, length(n));
PLv_term = zeros(1, length(n));

APDd_values = zeros(1, length(n));
APDd_term = zeros(1, length(n));
APDv_values = zeros(1, length(n));
APDv_term = zeros(1, length(n));

TT_values = zeros(1, length(n));
TT_term = zeros(1, length(n));

for i = 1:length(n)
    for it = 1:N
       [PLd(it) , APDd(it) , MPDd(it) , PLv(it) , APDv(it) ,
          MPDv(it), TT(it)] = Simulator3A(lambda,C,f,P, n(i),
           b);
    end

    fprintf('Valor de n: %d\n', n(i));
```

```
    mediaPLd = mean(PLd);
    termPLd = norminv(1-alfa/2) * sqrt(var(PLd)/N);
    fprintf('PacketLoss of data(%%)\t  = %.2e +- %.2e\n',
       mediaPLd, termPLd);

    mediaPLv = mean(PLv);
    termPLv = norminv(1-alfa/2) * sqrt(var(PLv)/N);
    fprintf('PacketLoss of VoIP (%%)\t  = %.2e +- %.2e\n',
       mediaPLv, termPLv);

    mediaAPDd = mean(APDd);
    termAPDd = norminv(1-alfa/2) * sqrt(var(APDd)/N);
    fprintf('Ag. Packet Delay of data (ms)\t = %.2e +- %.2e\n
       ', mediaAPDd, termAPDd);

    mediaAPDv = mean(APDv);
    termAPDv = norminv(1-alfa/2) * sqrt(var(APDv)/N);
    fprintf('Ag. Packet Delay of VoIP (ms)\t = %.2e +- %.2e\n
       ', mediaAPDv, termAPDv);

    mediaMPDd = mean(MPDd);
    termMPDd = norminv(1-alfa/2) * sqrt(var(MPDd)/N);
    fprintf('Max. Packet Delay of data (ms)\t = %.2e +- %.2e\
       n', mediaMPDd, termMPDd);

    mediaMPDv = mean(MPDv);
    termMPDv = norminv(1-alfa/2) * sqrt(var(MPDv)/N);
    fprintf('Max. Packet Delay of VoIP (ms)\t = %.2e +- %.2e\
       n', mediaMPDv, termMPDv);

    mediaTT = mean(TT);
    termTT = norminv(1-alfa/2) * sqrt(var(TT)/N);
    fprintf('Througtput (Mbps)\t = %.2e +- %.2e\n', mediaTT,
       termTT);

    PLd_values(i) = mediaPLd;
    PLd_term(i) = termPLd;
    PLv_values(i) = mediaPLv;
    PLv_term(i) = termPLv;

    APDd_values(i) = mediaAPDd;
    APDd_term(i) = termAPDd;
    APDv_values(i) = mediaAPDv;
    APDv_term(i) = termAPDv;

    TT_values(i) = mediaTT;
    TT_term(i) = termTT;
end
```

```
%% Task 2b) - Gr ficos
figure(1);
hold on; grid on;
bar(n, PLd_values);
er=errorbar(n, PLd_values, PLd_term);
er.Color = [0 0 0];
er.LineStyle = 'none';
xlabel('N mero de fluxos VoIP (n)');
ylabel('Packet Loss (%)');
xticks(n);
title('Task 2(b) - Packet Loss (Data)');
hold off;

figure(2);
hold on; grid on;
bar(n, PLv_values);
er= errorbar(n, PLv_values, PLv_term);
er.Color = [0 0 0];
er.LineStyle = 'none';
xlabel('N mero de fluxos VoIP (n)');
xticks(n);
ylabel('Packet Loss (%)');
title('Task 2(b) - Packet Loss (VoIP)');
hold off;

%% Task 2c) - Gr ficos
figure(1);
hold on; grid on;
bar(n, APDd_values);
er=errorbar(n, APDd_values, APDd_term);
er.Color = [0 0 0];
er.LineStyle = 'none';
xlabel('N mero de fluxos VoIP (n)');
ylabel('Average Packet Delay (ms)');
xticks(n);
title('Task 2(c) - Average Packet Delay (Data)');
hold off;

figure(2);
hold on; grid on;
bar(n, APDv_values);
er= errorbar(n, APDv_values, APDv_term);
er.Color = [0 0 0];
er.LineStyle = 'none';
xlabel('N mero de fluxos VoIP (n)');
ylabel('Average Packet Delay (ms)');
xticks(n);
```

```
title('Task 2(c) - Average Packet Delay (VoIP)');
hold off;

%% Task 2d) - Gr fico
figure(1);
hold on; grid on;
bar(n, TT_values);
er=errorbar(n, TT_values, TT_term);
er.Color = [0 0 0];
er.LineStyle = 'none';
xlabel('N mero de fluxos VoIP (n)');
ylabel('Throughput (Mbps)');
xticks(n);
title('Task 2(d) - Throughput total do link');
hold off;

%% Task 2e)
prob_elements = (1 - (0.19 + 0.23 + 0.17)) / ((109 - 65 + 1)
   + (1517 - 111 + 1));

avg_packet_size_data = 0.19*64 + 0.23*110 + 0.17*1518 + sum
   ((65:109)*(prob_elements)) + sum((111:1517)*(prob_elements
   ));
lambda = 1500;
data_TT = lambda * avg_packet_size_data * 8 / 10^6;

% VoIP parameters
avg_packet_size_voip = ( 110 + 130 ) / 2;
time_voip = (16 + 24) / 2 / 10^3;
rate_single_voip = 1 / time_voip;

voip_flows = [10, 20, 30, 40];

for i = 1:length(voip_flows)
    n_voip = voip_flows(i);
    total_voip_rate = n_voip * rate_single_voip;
    voip_TT = total_voip_rate * avg_packet_size_voip * 8 /
       10^6;
    final_TT = data_TT + voip_TT;

    fprintf('n= %d; Throughput = %.2f Mbps\n', n_voip,
       final_TT);
end
```