

Modelação e Desempenho de Redes e Serviços

2021/2022

Mini-Projeto 1

Performance evaluation of point-to-point links supporting packet Services

Bruno Aguiar
mec: 80177

Paulo Sousa
mec: 80000

Task 1

- a) Consider the case of $C = 10$ Mbps and $f = 1.000.000$ Bytes. Run Simulator1 50 times with a stopping criterion of $P = 10000$ each run and compute the estimated values and the 90% confidence intervals of the average delay performance parameter when $\lambda = 400, 800, 1200, 1600$ and 2000 pps. Present the average packet delay results in bar charts with the confidence intervals in error bars¹. Justify the results and take conclusions concerning the impact of the packet rate in the obtained average packet delay.

Analisando o simulador, podemos verificar que este retorna a percentagem de pacotes perdidos, o *average packet delay (ms)*, o *maximum packet delay (ms)* e o *transmited throughput (Mbps)*. De modo a calcular estes valores, são precisos 5 entradas: *lambda*, que representa o *packet rate (p.s⁻¹)*, o *C*, que representa a largura de banda em Mbps, o *f*, que representa o tamanho da fila de espera em Bytes, e o *P*, que representa o número de pacotes transmitidos necessários para que o simulador termine a simulação (*stopping criterium*).

O seu método de funcionamento segue então a seguinte lógica: se o evento for um *Arrival*, o número total de pacotes é incrementado, e se a ligação estiver livre, esta é posta no estado *busy*, caso contrário é verificado se a fila de espera está cheia ou não. Se estiver cheia, o pacote é descartado, senão, o pacote é posto na fila. Se o evento for um *Departure*, então o número de bytes e de pacotes transmitidos é incrementado, os atrasos (*delays*) são calculados e, caso maior que 0, a ocupação da fila de espera é decrementada e o valor lógico do estado do simulador é posto de novo a 0, o que significa que a ligação está livre (*free*).

A geração de pacotes é feita pela função `GeneratePacketSize()`, que cria um pacote de tamanho variável, entre 64 a 1518 bytes, respeitando uma distribuição genérica com as seguintes probabilidades: 19% para pacotes de tamanho igual a 64 bytes, 23% para pacotes com tamanho de 110 bytes, 17% para pacotes com tamanho de 1518 bytes e probabilidades iguais para os restantes tamanhos (de 65 a 109 e de 110 a 1517 bytes). O algoritmo irá seguidamente gerar um número aleatório entre 0..1 usando o algoritmo `Linear Congruential Generator` através do método `rand()`, sendo esse número posteriormente comparado com a distribuição cumulativa das probabilidades acima atribuídas.

O código do simulador pode ser vizualizado através da secção `Code attachments`, em Listing 12.

Testando o simulador de acordo com diferentes valores para o *packet rate*, e analisando os resultados da Figura 1, podemos concluir que, quanto maior é a taxa de pacotes recebidos, não variando a largura de manda e/ou o tamanho da fila de espera, maior será o número de pacotes na fila de espera, o que impõe um maior *Average Packet Delay*. O código de teste para a alínea a) pode ser visualizado em Listing 1: Código a).

Listing 1: Código a)

```

1 % TASK 1
2 % A
3
4 lambda = [400, 800, 1200, 1600, 2000];
5 C = 10;
6 P = 10000;
7
8 N = 50;
9 f = 1000000;
10
11 per=zeros(N,4); % vetor com N valores de simula o
12
13 lambda_len = length(lambda);
14 lambda_res = zeros(N,4,lambda_len);
15
16 for i= 1:lambda_len
17     for j = 1:N
18         [per(j,1),per(j,2),per(j,3),per(j,4)] = Simulator1(lambda(i), C, f, P);
19     end
20     lambda_res(:, :, i) = per(:, :);
21 end
22
23
24 %%
25
26 alfa = 0.1; % intervalo de confian a 90%
27
28 media = mean(lambda_res);
29
30 term = norminv(1-alfa/2)*sqrt(var(lambda_res)/N);
31
32 av_pktd_term = zeros(lambda_len,2);
33
34 for it = 1:lambda_len
35     av_pktd_term(it,:) = [media(:, 2, it), term(:, 2,it)];
36 end
37
38 %%
39 dados = zeros(1, lambda_len);
40 err = zeros(1, lambda_len);
41
42 for it2 = 1:lambda_len
43     dados(1, it2) = av_pktd_term(it2, 1);
44     err(1, it2) = av_pktd_term(it2, 2);
45 end
46
47 x = 1:lambda_len;
48 figure('Name', 'Average Packet Delay')
49 bar(x, dados)
50 xlabel('Packet rate (p/s)')
51 set(gca,'xticklabel',lambda)
52 ylabel('Avg. Packet Delay (ms)')
53
54 hold on
55 er = errorbar(x, dados, err);
56 er.Color = [0 0 0];
57 er.LineStyle = 'none';
58 hold off

```

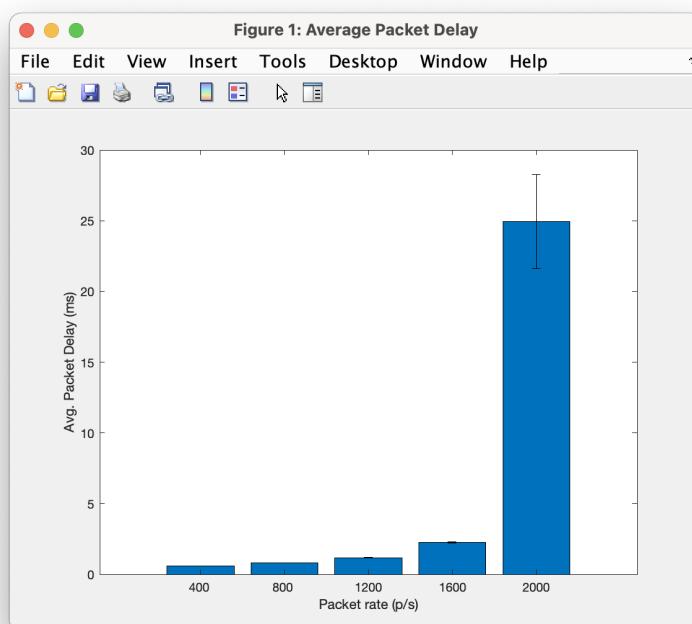


Figura 1: Average Packet Delay (ms)

b) Consider the case of $\lambda = 1800$ pps and $C = 10$ Mbps. Run Simulator1 50 times with a stopping criterion of $P = 10000$ each run and compute the estimated values and the 90confidence intervals of the average delay and packet loss performance parameters when $f = 100.000, 20.000, 10.000$ and 2.000 Bytes. Present the average packet delay results in one figure and the average packet loss results in another figure (in both cases, in bar charts with the confidence intervals in error bars). Justify the results and take conclusions concerning the impact of the queue size in the obtained average packet delay and average packet loss.

Seguindo a mesma lógica de funcionamento do simulador no exercício anterior e sendo que agora o valor que difere é o tamanho da fila de espera, podemos observar que o packet loss aumenta e o packet delay diminui com a diminuição do tamanho da fila. Podemos afirmar que quanto menor for o tamanho da fila, menor será o packet delay. Sendo uma regra de first in first out, se a fila de espera é mais pequena é expectável que um pacote demore menos tempo a ser atendido. Quanto menor for o tamanho da queue, maior será o número de pacotes a serem descartados (*packet loss*), já que existe uma maior probabilidade de a fila de espera se encontrar cheia e, por isso, ter que descartar pacotes que não consigam lugar na fila. Quanto maior for a fila de espera, menor é a probabilidade de perda de um pacote. A verificação destas conclusões podem ser feitas através da Figura 2. O código de teste da alínea b) pode ser consultado no Listing 2.

Listing 2: Código b)

```

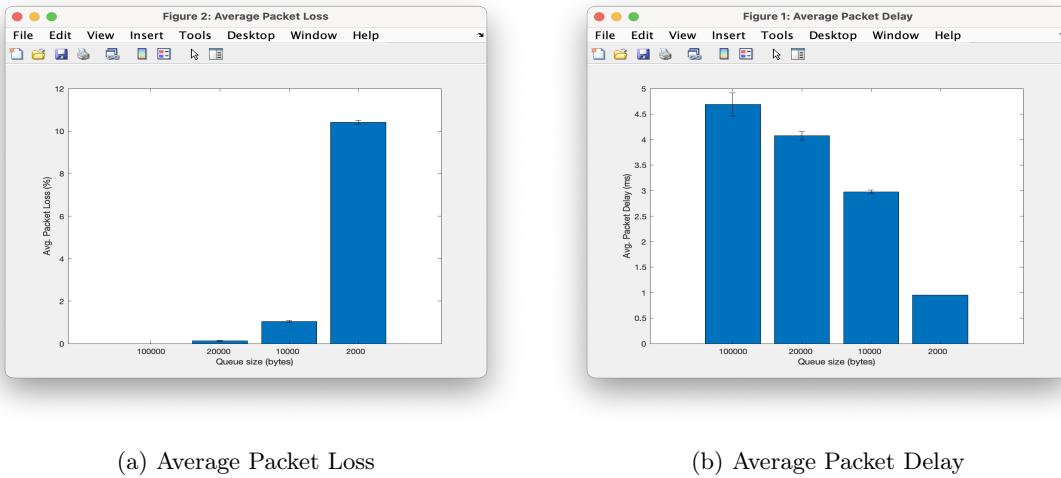
1 % TASK 1
2 % B
3
4 lambda = 1800;
5 C = 10;
6 P = 10000;
7
8 N = 50;
9 f = [100000, 20000, 10000, 2000];
10
11 per=zeros(N,4); % vetor com N valores de simula o
12
13 qsize = length(f);
14 qsize_res = zeros(N,4,qsize);
15
16 for i= 1:qsize
17     for j = 1:N
18         [per(j,1),per(j,2),per(j,3),per(j,4)] = Simulator1(lambda, C, f(i), P);
19     end
20     qsize_res(:, :, i) = per(:, :);
21 end
22
23 %%
24
25 alfa = 0.1; % intervalo de confian a 90%
26
27 media = mean(qsize_res);
28
29 term = norminv(1-alfa/2)*sqrt(var(qsize_res)/N);

```

```

30
31 av_pktd_term = zeros(qsize,2);
32 av_pktd_term = zeros(qsize,2);
33
34 for it = 1:qsize
35     av_pktd_term(it,:) = [media(:, 2, it), term(:, 2,it)];
36     av_pktd_term(it,:) = [media(:, 1, it), term(:, 1, it)];
37 end
38
39 ddados = zeros(1, qsize);
40 derr = zeros(1, qsize);
41 ldados = zeros(1, qsize);
42 lerr = zeros(1, qsize);
43
44 for it2 = 1:qsize
45     ddados(1, it2) = av_pktd_term(it2, 1);
46     derr(1, it2) = av_pktd_term(it2, 2);
47     ldados(1, it2) = av_pktd_term(it2, 1);
48     lerr(1, it2) = av_pktd_term(it2, 2);
49 end
50
51 x = 1:qsize;
52
53 figure('Name', 'Average Packet Delay')
54 bd = bar(x, ddados);
55 xlabel('Queue size (bytes)')
56 set(gca, 'xticklabel',f)
57 ylabel('Avg. Packet Delay (ms)')
58
59 hold on
60 er = errorbar(x, ddados, derr);
61 er.Color = [0 0 0];
62 er.LineStyle = 'none';
63 hold off
64
65 figure('Name', 'Average Packet Loss')
66 bl = bar(x, ldados);
67 xlabel('Queue size (bytes)')
68 set(gca, 'xticklabel',f)
69 ylabel('Avg. Packet Loss (%)')
70
71 hold on
72 er = errorbar(x, ldados, lerr);
73 er.Color = [0 0 0];
74 er.LineStyle = 'none';
75 hold off

```



(a) Average Packet Loss

(b) Average Packet Delay

Figura 2: Average Packet Delay e Average Packet Loss baseado em filas de espera com diferentes tamanhos

c) Consider the case of $\lambda = 1800$ pps and $f = 1.000.000$ Bytes. Run Simulator1 50 times with a stopping criterion of $P = 10000$ at each run and compute the estimated values and the 90% confidence intervals of the average delay performance parameter when $C = 10, 20, 30$ and 40 Mbps. Present the average packet delay results in bar charts with the confidence intervals in error bars. Justify the results and take conclusions concerning the impact of the link capacity in the obtained average packet delay.

Testando o simulador de acordo com diferentes valores para a largura de banda, e analisando os resultados da Figura 3, podemos concluir que quanto maior a largura de banda for, menor será o atraso médio por pacote. Ora isto é esperável de acontecer, uma vez que a largura de banda dita a velocidade da ligação e sendo esta um factor que pesa no atraso total dos pacotes (podemos facilmente verificar no código do simulador que o atraso total é dado por: $DELAYS = DELAYS + (Clock - ArrivalInstant)$, em que a velocidade da largura de banda (dado pela variável C) pesa no cálculo do Clock: $Clock + 8 * QUEUE(1, 1)/(C * 10^6)$), onde para valores altos de C o valor de Clock irá diminuir, diminuindo também a diferença entre o Clock e o ArrivalInstant, o que diminui, por consequente, o valor do Delays.), é natural que quando maior a velocidade da ligação (largura de banda), menor é o atraso. O código de teste para a alínea c) pode ser visualizado em Listing 3: Código c).

Listing 3: Código c)

```

1 % TASK 1
2 % C
3
4 lambda = 1800;
5 C = [10, 20, 30, 40];
6 P = 10000;
7
8 N = 50;
9 f = 1000000;
10
11 per=zeros(N,4); % vetor com N valores de simula o
12
13 bwsize = length(C);
14 bw_res = zeros(N,4,bwsize);
15
16 for i= 1:bwsize
17     for j = 1:N
18         [per(j,1),per(j,2),per(j,3),per(j,4)] = Simulator1(lambda, C(i), f, P);
19     end
20     bw_res(:, :, i) = per(:, :);
21 end
22
23 %%
24 alfa = 0.1; % intervalo de confiança 90%
25
26 media = mean(bw_res);
27
28 term = norminv(1-alfa/2)*sqrt(var(bw_res)/N);
29
30 av_pktd_term = zeros(bwsize,2);
31

```

```

32 for it = 1:bwsize
33     av_pktd_term(it,:) = [media(:, 2, it), term(:, 2,it)];
34 end
35 av_pktd_term %
36
37 %%
38 dados = zeros(1, bwsize);
39 err = zeros(1, bwsize);
40
41 for it2 = 1:bwsize
42     dados(1, it2) = av_pktd_term(it2, 1);
43     err(1, it2) = av_pktd_term(it2, 2);
44 end
45
46 x = 1:bwsize;
47 figure("Name", "Average Packet Delay")
48 bd = bar(x, dados);
49 xlabel('Link bandwidth (Mbps)')
50 set(gca,'xticklabel',C)
51 ylabel('Avg. Packet Delay (ms)')
52
53 hold on
54 er = errorbar(x, dados, err);
55 er.Color = [0 0 0];
56 er.LineStyle = 'none';
57
58 hold off

```

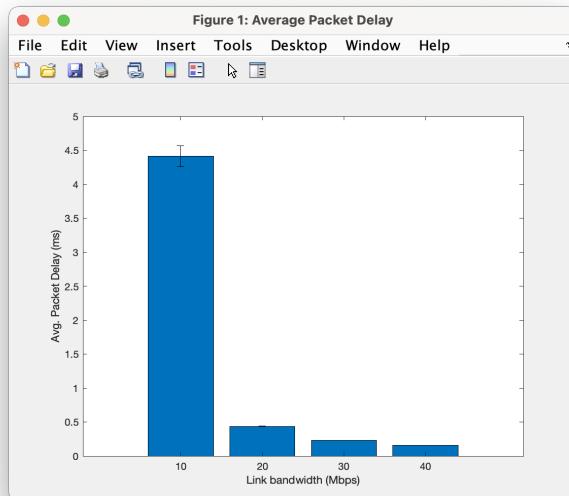


Figura 3: Average Packet Delay (ms)

d) Consider that the system is modelled by a M/G/1 queueing model. Determine the theoretical values of the average packet delay using the M/G/1 model for all cases of 1.c. Compare the theoretical values with the simulation results of experiments 1.c and take conclusions.

O código dado pelo Listing 4 (secção "%% task d"), permite calcular o modelo teórico do atraso médio num sistema M/G/1, que se caracteriza pela chegada de clientes ser um processo de Poisson com taxa λ , um tempo de atendimento S com distribuição genérica e independente das chegadas dos clientes, com um servidor e uma fila de espera com tamanho infinito.

Este atraso teórico é calculado somando o atraso da fila de espera com o tempo médio de ser atendido (variável `avgPktDelay`). O tempo médio para ser atendido é dado pela variável `avgPacketTransmission`, que não é nada mais nada menos que o *throughput* em *pps* (pacotes por segundo), que é dado pela fórmula $(avgPacketSize * 8) ./ C$. Já o atraso da fila de espera é calculado pela fórmula $\frac{\lambda E[S^2]}{2 - 2\lambda E[S]} + E[S]$, como se pode verificar através da variável `avgQueuingDelay`.

No gráfico da Figura 4 mostramos os resultados calculados no exercício anterior a azul e o valor teórico de Packet Delay calculado com o modelo M/G/1 para cada valor de largura de banda a vermelho. Depois de algumas simulações, podemos observar que quando largura de banda é menor, a oscilação da diferença entre o valor do atraso médio por pacote teórico e o da simulação (sendo que em algumas simulações o valor teórico é maior e noutras o oposto) é maior que para maiores larguras de banda, sendo que estas últimas possuem sempre valores teóricos e simulados muito próximos. Ou seja, podemos concluir que, para larguras de banda de baixa velocidade, além de o atraso ser elevado, também se torna-se algo instável e difícil de prever, tornando assim o modelo teórico numa boa aproximação ao modelo real para larguras de banda elevadas e uma má aproximação ao modelo real para larguras de banda baixas.

Listing 4: Código d)

```

1 % TASK 1
2 % D
3
4 %% task 1c - avgPktDelay para construir o grafico
5
6 %% task 1d
7 C = [10*10^6, 20*10^6, 30*10^6, 40*10^6];
8
9 prob = zeros(1,1518) ;
10 prob(size) = (1 - 0.19 - 0.23 - 0.17) / ((length(size) - 3)) ;
11 prob(64) = 0.19 ;
12 prob(110) = 0.23 ;
13 prob(1518) = 0.17 ;
14
15 avgPacketSize = sum(prob.*size);
16 avgPacketTransmission = (avgPacketSize * 8) ./ C ;
17 ES = avgPacketTransmission ;
18 ES2 = zeros(1, length(C)) ;
19
20 for i = 1:length(C)
21     aux = ((size .* 8) / C(i)).^2 ;
22     ES2(1, i) = sum(prob.*aux) ;

```

```

23 end
24
25 avgQueuingDelay = (lambda.*ES2)./(2-(2*lambda.*ES));
26 avgPktDelay = (avgQueuingDelay + avgPacketTransmission) * 1000 % s -> ms
27
28 x = 1:bwsizer;
29 figure("Name", "Average Packet Delay")
30 bd = bar(x, [dados; avgPktDelay]);
31 xlabel('Link bandwidth (Mbps)')
32 set(gca,'xticklabel',C./10^6)
33 ylabel('Avg. Packet Delay (ms)')
34 legend('Valor do Simulador','Valor Teórico')

```

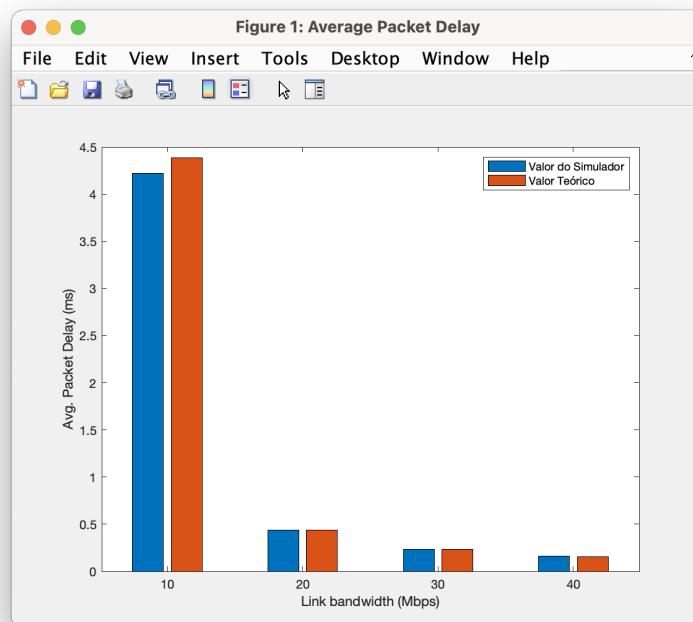


Figura 4: Average Packet Delay (ms)

e) Develop a new version of Simulator1 to estimate 3 additional performance parameters: the average packet delay of the packets of size 64, 110 and 1518 Bytes, respectively. Consider the case of $\lambda = 1800$ pps and $f = 1.000.000$. Run the new version of Simulator1 50 times with a stopping criterion of $P = 10000$ at each run and compute the estimated values and the 90% confidence intervals of the 3 new average delay performance parameters when $C = 10, 20, 50$ and 100 Mbps. Present the average packet delay results in bar charts with the confidence intervals in error bars. Justify these results and the differences between them and the results of 1.c. Take conclusions concerning the impact of the link capacity in the obtained average packet delay of packets with different sizes.

A nova versão do Simulator1, `Simulator1_new` está em anexo na secção `Code attachments`, no Listing 13. De modo a implementar os 3 parâmetros de performance adicionais, decidiu-se que estes, como têm um comportamento bastante semelhante ao APD, seriam implementados dentro do `case DEPARTURE`. No entanto as seus variáveis complementares só atualizam caso o tamanho do pacote for o correspondido, como o seguinte exemplo de código pretende ilustrar:

```

1 if (PacketSize == 64)
2     TRANSMITTEDPACKETS_64 = TRANSMITTEDPACKETS_64 + 1; % variavel complementar: TRANSMITTEDPACKETS_64
3     DELAYS_64 = DELAYS_64 + (Clock - ArrivalInstant); % variavel complementar: DELAYS_64
4 end

```

Calculadas as variáveis complementares para cada um dos 3 parâmetros adicionais, e decorrida a simulação, esses parâmetros depois serão atualizados, como é abaixo ilustrado:

```
1 APD_64 = 1000*DELAYS_64/TRANSMITTEDPACKETS_64;
```

Dito isto, e analisando os resultados mostrados na Figura 5, podemos concluir que para pacotes com menor tamanho, o atraso médio por pacote é menor do que para pacotes com um tamanho maior. Ora, isto é expectável, devido ao facto de que, para a mesma largura de banda, os pacotes com maior tamanho possuem mais bits a serem enviados, demorando assim mais tempo a serem enviados todos os bits dos pacotes com maior tamanho. O código de teste para o exercício e) está exemplificado em Listing 5.

Listing 5: Código e)

```

1 % TASK 1
2 % E.
3
4 lambda = 1800;
5 C = [10, 20, 50, 100];
6 P = 10000;
7
8 N = 50;
9 f = 1000000;
10
11 per=zeros(N,7); % vetor com N valores de simula o
12
13 size = length(C);

```

```

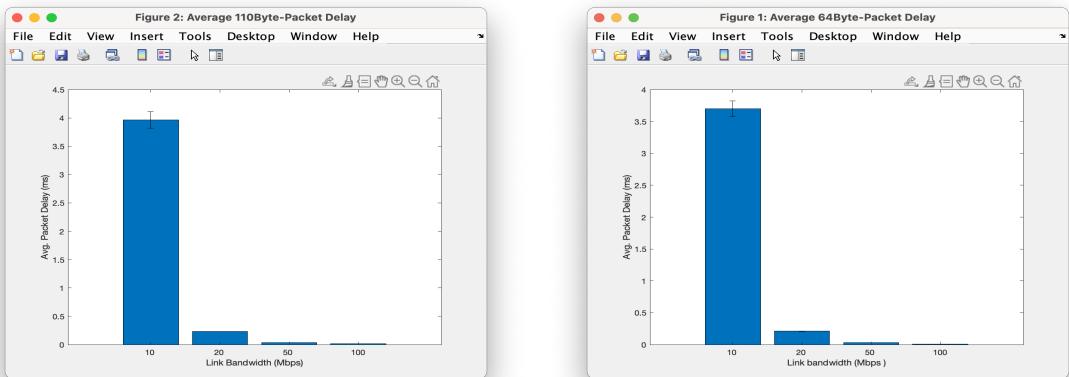
14 sim_output = zeros(N,7,size);
15
16 for i= 1:size
17     for j = 1:N
18         [per(j,1),per(j,2),per(j,3),per(j,4), per(j, 5), per(j, 6), per(j, 7)] =...
19             Simulator1_new(lambda, C(i), f, P);
20     end
21     sim_output(:, :, i) = per(:, :);
22 end
23
24 %%
25
26 alfa = 0.1; % intervalo de confian a 90%
27
28 media = mean(sim_output);
29
30 term = norminv(1-alfa/2)*sqrt(var(sim_output)/N);
31
32 av_pktd64_term = zeros(size,2);
33 av_pktd110_term = zeros(size, 2);
34 av_pktd1518_term = zeros(size, 2);
35
36 for it = 1:size
37     av_pktd64_term(it,:) = [media(:, 3, it), term(:, 3,it)];
38     av_pktd110_term(it,:) = [media(:, 4, it), term(:, 4,it)];
39     av_pktd1518_term(it,:) = [media(:, 5, it), term(:, 5,it)];
40 end
41
42 %%
43 dados64 = zeros(1, size);
44 derr64 = zeros(1, size);
45 dados110 = zeros(1, size);
46 derr110 = zeros(1, size);
47 dados1518 = zeros(1, size);
48 derr1518 = zeros(1, size);
49
50 for it2 = 1:size
51     dados64(1, it2) = av_pktd64_term(it2, 1);
52     derr64(1, it2) = av_pktd64_term(it2, 2);
53
54     dados110(1, it2) = av_pktd110_term(it2, 1);
55     derr110(1, it2) = av_pktd110_term(it2, 2);
56
57     dados1518(1, it2) = av_pktd1518_term(it2, 1);
58     derr1518(1, it2) = av_pktd1518_term(it2, 2);
59
60 end
61
62 x = 1:size;
63 figure("Name", "Average 64Byte-Packet Delay")
64 bd64 = bar(x, dados64);
65 xlabel('Link bandwidth (Mbps )')
66 set(gca,'xticklabel',C)
67 ylabel('Avg. Packet Delay (ms)')
68
69
70 hold on
71 er = errorbar(x, dados64, derr64);
72 er.Color = [0 0 0];
73 er.LineStyle = 'none';

```

```

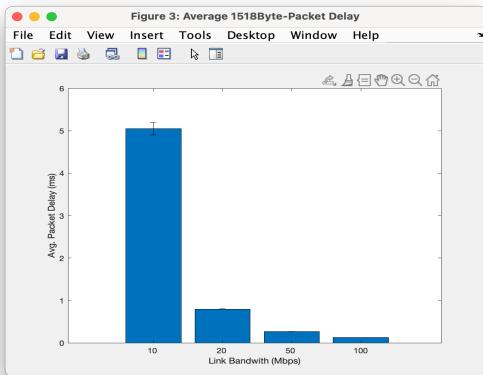
74 hold off
75
76 x = 1:size;
77 figure('Name', 'Average 110Byte-Packet Delay')
78 bar(x, dados110)
79 xlabel('Link Bandwidth (Mbps)')
80 set(gca,'xticklabel',C)
81 ylabel('Avg. Packet Delay (ms)')
82
83 hold on
84 er = errorbar(x, dados110, derr110);
85 er.Color = [0 0 0];
86 er.LineStyle = 'none';
87 hold off
88
89 x = 1:size;
90 figure('Name', 'Average 1518Byte-Packet Delay')
91 bar(x, dados1518)
92 xlabel('Link Bandwith (Mbps)')
93 set(gca,'xticklabel',C)
94 ylabel('Avg. Packet Delay (ms)')
95
96 hold on
97 er = errorbar(x, dados1518, derr1518);
98 er.Color = [0 0 0];
99 er.LineStyle = 'none';
100 hold off

```



(a) Average 110 Byte Packet Delay

(b) Average 64 byte Packet Delay



(c) Average 1518 Byte Packet Delay

Figura 5: Average Packet Delay baseado em pacotes com diferentes tamanhos

Task 2

a) Consider the case of $\lambda = 1500$ pps, $C = 10$ Mbps and $f = 1.000.000$ Bytes. Run Simulator3 50 times with a stopping criterion of $P = 10000$ each run and compute the estimated values and the 90% confidence intervals of the average delay performance parameter of data packets and VoIP packets when $n = 10, 20, 30$ and 40 VoIP flows. Present the average data packet delay results in one figure and the average VoIP packet delay results in another figure (in both cases, in bar charts with the confidence intervals in error bars). Justify the results and take conclusions concerning the impact of the number of VoIP flows in the obtained average packet delay of each service when both services (data and VoIP) are statistically multiplexed in a single FIFO queue.

O código para o simulador 3 está listado na secção `Code attachments`, em [Listing 14](#) e o código de teste da alínea a) está listado em [Listing 6](#).

Correndo o código de teste e analisando os resultados mostrados, tendo agora flows adicionais para VoIP, utilizando apenas uma queue que faz uma multiplexagem estatística dos pacotes de dados e de VoIP, podemos afirmar que o atraso médio para os dados é ligeiramente menor que o atraso médio para os pacotes VoIP, para todo o número de fluxos testados. Podemos também verificiar que quantos mais fluxos, maior é o atraso médio imposto em ambos os tipos de pacotes.

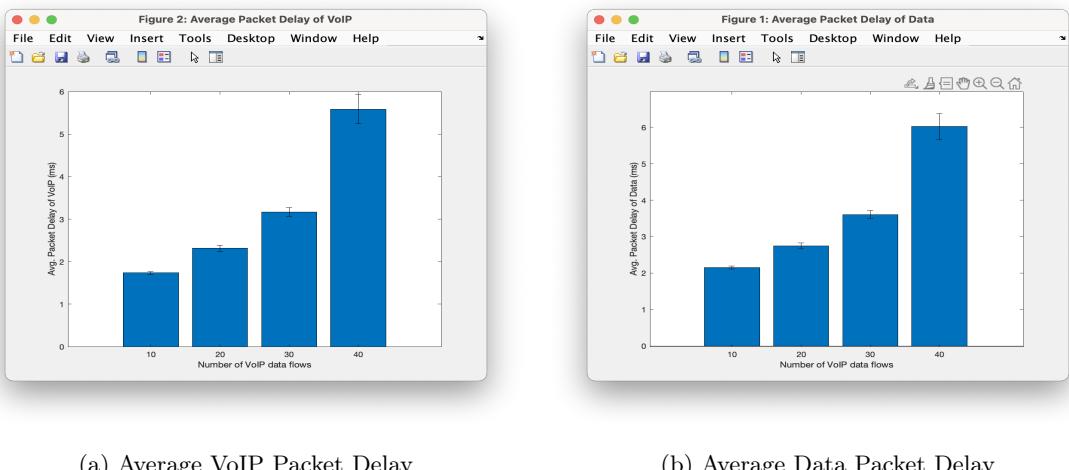
Listing 6: Código a)

```
1 lambda = 1500;
2 C = 10;
3 f = 1000000;
4 N = 50;
5 P = 10000;
6 b = 10^-6;
7 n = [10, 20, 30, 40];
8
9 per=zeros(N,7); % vetor com N valores de simula o
10
11 nsize = length(n);
12 res = zeros(N,7,nsize);
13
14 for i = 1:nsize
15     for it = 1:N
16         [per(it,1),per(it,2),per(it,3),per(it,4),per(it,5),per(it,6),per(it,7)] =...
17             Simulator3(lambda, C, f, P, n(i));
18         %PLd , PLv , APDd , APDv , MPDd , MPDv , TT
19     end
20     res(:,:,i) = per(:,:,i);
21 end
22
23 %%
24 alfa = 0.1; % intervalo de confian a 90%
25 media = mean(res);
```

```

26 term = norminv(1-alfa/2)*sqrt(var(res)/N);
27
28 av_pkt_delay_data = zeros(nsize,2);
29 av_pkt_delay_voip = zeros(nsize,2);
30
31 for it = 1:nsize
32     av_pkt_delay_data(it,:) = [media(:, 3, it), term(:, 3,it)];
33     av_pkt_delay_voip(it,:) = [media(:, 4, it), term(:, 4,it)];
34 end
35
36 %%
37 ddados = zeros(1, nsize);
38 derr = zeros(1, nsize);
39 vdados = zeros(1, nsize);
40 verr = zeros(1, nsize);
41
42 for it2 = 1:nsize
43     ddados(1, it2) = av_pkt_delay_data(it2, 1);
44     derr(1, it2) = av_pkt_delay_data(it2, 2);
45     vdados(1, it2) = av_pkt_delay_voip(it2, 1);
46     verr(1, it2) = av_pkt_delay_voip(it2, 2);
47 end
48
49 %%
50 x = 1:nsize;
51
52 figure('Name', 'Average Packet Delay of Data')
53 bd = bar(x, ddados);
54 xlabel('Number of VoIP data flows')
55 set(gca, 'xticklabel',n)
56 ylabel('Avg. Packet Delay of Data (ms)')
57
58 hold on
59 er = errorbar(x, ddados, derr);
60 er.Color = [0 0 0];
61 er.LineStyle = 'none';
62 hold off
63
64 figure('Name', 'Average Packet Delay of VoIP')
65 bl = bar(x, vdados);
66 xlabel('Number of VoIP data flows')
67 set(gca, 'xticklabel',n)
68 ylabel('Avg. Packet Delay of VoIP (ms)')
69
70 hold on
71 er = errorbar(x, vdados, verr);
72 er.Color = [0 0 0];
73 er.LineStyle = 'none';
74 hold off

```



(a) Average VoIP Packet Delay

(b) Average Data Packet Delay

Figura 6: Average Packet Delay de dados e VoIP baseado no número de VoIP flows

b) Repeat experiment 2.a but now with Simulator4. Justify these results and the differences between them and the results of 2.a. Take conclusions concerning the impact of the number of VoIP flows in the obtained average packet delay of each service when VoIP service is supported with a priority which is higher than the data service.

Usando agora o Simulador 4 (ver secção [Code attachments](#)) com uma política de prioridades, tendo o VoIP a prioridade mais elevada, e correndo o código de teste listado em Listing 7, podemos verificar através da Figura 7 que o atraso médio dos pacotes VoIP é substancialmente mais pequeno que o atraso médio verificado em pacotes de dados. Isto deve-se ao facto de que sempre que um pacote é processado no **Departure**, a fila de espera é ordenada pela ordem de prioridades, então, sempre que ouver um pacote de VoIP ele é sempre escolhido primeiro em relação a pacotes de dados que possam existir na mesma fila, e portanto, o seu atraso médio irá ser substancialmente reduzido.

Listing 7: Código b)

```

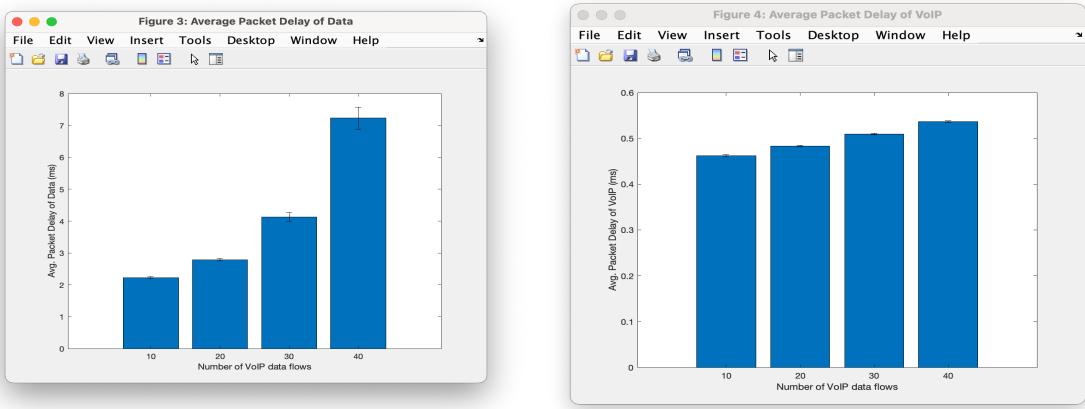
1 % TASK 2
2 % B
3
4 lambda = 1500;
5 C = 10;
6 f = 1000000;
7 N = 50;
8 P = 10000;
9 n = [10, 20, 30, 40];
10
11 per=zeros(N,7); % vetor com N valores de simula o
12
13 nsize = length(n);
14 res = zeros(N,7,nsize);
15
16 for i = 1:nsize
17     for it = 1:N
18         [per(it,1),per(it,2),per(it,3),per(it,4),per(it,5),per(it,6),per(it,7)] =...
19             Simulator4(lambda, C, f, P, n(i));
20         %PLd , PLv , APDd , APDv , MPDd , MPDv , TT
21     end
22     res(:,:,i) = per(:,:,i);
23 end
24
25 %%
26 alfa = 0.1; % intervalo de confian a 90%
27 media = mean(res);
28 term = norminv(1-alfa/2)*sqrt(var(res)/N);
29
30 av_pkt_delay_data = zeros(nsize,2);
31 av_pkt_delay_voip = zeros(nsize,2);
32
33 for it = 1:nsize
34     av_pkt_delay_data(it,:) = [media(:, 3, it), term(:, 3,it)];
35     av_pkt_delay_voip(it,:) = [media(:, 4, it), term(:, 4,it)];
36 end
37
38 %%
39 ddados = zeros(1, nsize);

```

```

40 derr = zeros(1, nsize);
41 vdados = zeros(1, nsize);
42 verr = zeros(1, nsize);
43
44 for it2 = 1:nsize
45     ddados(1, it2) = av_pkt_delay_data(it2, 1);
46     derr(1, it2) = av_pkt_delay_data(it2, 2);
47     vdados(1, it2) = av_pkt_delay_voip(it2, 1);
48     verr(1, it2) = av_pkt_delay_voip(it2, 2);
49 end
50
51 %%
52 x = 1:nsize;
53
54 figure('Name', 'Average Packet Delay of Data')
55 bd = bar(x, ddados);
56 xlabel('Number of VoIP data flows')
57 set(gca, 'xticklabel',n)
58 ylabel('Avg. Packet Delay of Data (ms)')
59
60 hold on
61 er = errorbar(x, ddados, derr);
62 er.Color = [0 0 0];
63 er.LineStyle = 'none';
64 hold off
65
66 figure('Name', 'Average Packet Delay of VoIP')
67 bl = bar(x, vdados);
68 xlabel('Number of VoIP data flows')
69 set(gca, 'xticklabel',n)
70 ylabel('Avg. Packet Delay of VoIP (ms)')
71
72 hold on
73 er = errorbar(x, vdados, verr);
74 er.Color = [0 0 0];
75 er.LineStyle = 'none';
76 hold off

```



(a) Average VoIP Packet Delay

(b) Average Data Packet Delay

Figura 7: Average Packet Delay de dados e VoIP baseado no número de VoIP flows

c) Consider that the system is modelled by a M/G/1 queueing model with priorities. Determine the theoretical values of the average data packet delay and average VoIP packet delay using the M/G/1 model for all cases of 2.b. Compare the theoretical values with the simulation results of experiments 2.b and take conclusions.

O código para o cálculo dos valores teóricos encontra-se listado em Listing 8. As linhas 2-4 permitem gerar pacotes VoIP com tamanho uniformemente distribuído entre 110 Bytes a 130 Bytes, possuindo todos os tamanhos a mesma probabilidade. Já as linhas 6-11 permitem a geração de uma distribuição genérica para os pacotes de dados.

O packet rate, λ , que é dado corresponde ao fluxo de dados, como agora temos os fluxos de VoIP adicionais, precisou-se de calcular o packet rate para o tipo de pacotes VoIP. Sabe-se, através do ex.7 guião prático, que o tempo entre chegadas de pacotes do tipo VoIP em todos os fluxos é uniformemente distribuído num intervalo entre 16 a 24 milissegundos. Calculou-se assim o inverso da sua média de modo a descobrir o packet rate para cada fluxo e multiplicou-se por cada fluxo de modo a descobrir o packet rate total do tipo de pacotes VoIP, já que apenas existe uma fila de espera que faz uma multiplexagem estatística, podendo estes cálculos serem vistos nas linhas 23-24.

Calculou-se também os tamanhos médios do VoIP e dados nas linhas 19-20, somando cada tamanho possível para cada um dos tipos de pacotes e, usando as probabilidades para cada um deles, fazendo-se assim uma média ponderada para cada tipo de pacote (VoIP ou dados). Isto é importante para podermos calcular o $E[S]$ (média do tempo de transmissão dos pacotes) e o $E[S^2]$ (tempo de transmissão ao quadrado dos pacotes) nas linhas 26-30, que são calculados dividindo o tamanho médio do tipo de pacote em bits pela largura de banda (que nos dá a velocidade de transmissão em Mbps).

Por fim, sendo o average packet delay dado pela soma do atraso médio na fila de espera, com o atraso médio de ser atendido, $E[S]$, calculou-se assim o atraso médio da fila de espera para o fluxo VoIP usando a expressão $W_v = \frac{\lambda_v E[S_v^2] + \lambda_d E[S_d^2]}{2(1 - \lambda_v E[S_v])}$, sendo v = VoIP e d = dados, já que possui a prioridade mais elevada dos dois tipos de pacotes, e adicionou-se o atraso médio de ser atendido para o fluxo VoIP, $E[S_v]$, de modo a obter-se o average packet delay do VoIP, multiplicando por 1000 para ficar em milissegundos. Para o atraso médio da fila de espera para o fluxo de dados, usou-se a expressão $W_d = \frac{\lambda_v E[S_v^2] + \lambda_d E[S_d^2]}{2(1 - \lambda_v E[S_v])(1 - \lambda_v E[S_v] - \lambda_d E[S_d])}$, já que este possui menor probabilidade, adicionou-se $E[S_d]$ de modo a calcular o average packet delay dos dados e multiplicou-se por 1000 de modo a converter para milissegundos. Estes cálculos podem ser vistos nas linhas 32-38.

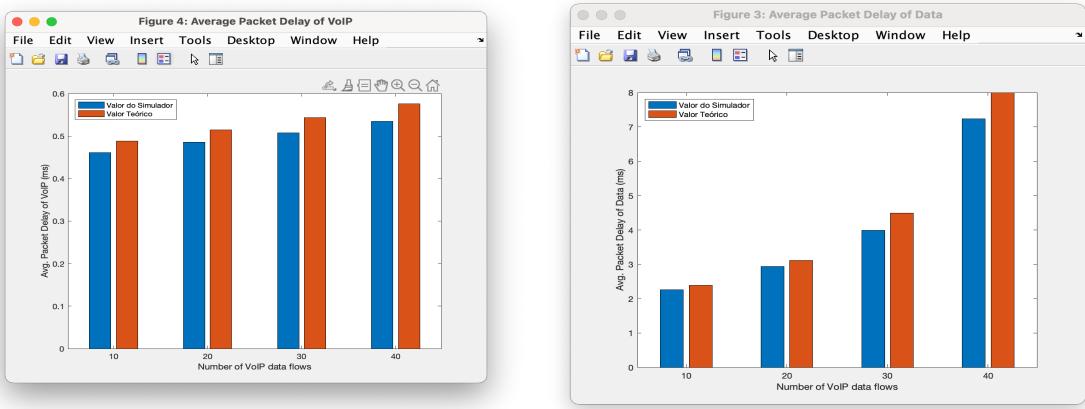
Analisando os resultados e comparando os valores práticos com os valores teóricos, dados na Figura 8, podemos verificar que tanto os valores práticos para os valores teóricos de delay para o tipo de pacotes VoIP possuem substancialmente menos atrasos médios que os pacotes do tipo dados. Podemos também concluir que para um número elevado de fluxos e de atrasos médios, a discrepância entre os valores teóricos e práticos tende a aumentar, no entanto, não consideramos que ainda seja uma diferença significativa, pelo que podemos dizer que o modelo teórico é, de facto, uma boa aproximação do modelo real.

Listing 8: Código c)

```

1 % TASK 2
2 % C
3
4 %% task 2b - d    valores de delay e loss para construir os graficos da mesma forma que no ex 1.d
5
6 %% task 2c
7 C = 10e6;
8 size_VoIP = 110:130;
9 prob_VoIP = zeros(1, 130);
10 prob_VoIP(size_VoIP) = 1/length(size_VoIP) ;
11
12 size_data = 64:1518 ;
13 prob_data = zeros(1,1518) ;
14 prob_data(size_data) = (1 - 0.19 - 0.23 - 0.17) / ((length(size_data) - 3)) ;
15 prob_data(64) = 0.19 ;
16 prob_data(110) = 0.23 ;
17 prob_data(1518) = 0.17 ;
18
19 avgPacketSizeVoIP = sum(prob_VoIP.*size_VoIP); %
20 avgPacketSizeData = sum(prob_data.*size_data); %
21
22
23 media_pcktarrivals = ((16/1000)+(24/1000))/2;
24 lambda_VoIP = (1/media_pcktarrivals).*n;
25
26 ES_data = (avgPacketSizeData*8)/C ;%
27 ES_VoIP = (avgPacketSizeVoIP*8)/C ;%
28
29 ES2_data = (ES_data^2*2);
30 ES2_VoIP = (ES_VoIP^2*2);
31
32 W_VoIP = lambda*ES2_data + (lambda_VoIP.*ES2_VoIP);
33 W_VoIP = W_VoIP ./ (2 *(1-lambda_VoIP.*ES_VoIP)); %
34 W_VoIP = (W_VoIP + ES_VoIP)*1000
35
36 W_data = lambda*ES2_data + lambda_VoIP.*ES2_VoIP;
37 W_data = W_data ./ (2*(1-lambda_VoIP.*ES_VoIP).*(1-lambda_VoIP.*ES_VoIP-lambda*ES_data)) ; %
;
38 W_data = (W_data + ES_data).*1000

```



(a) Average VoIP Packet Delay

(b) Average Data Packet Delay

Figura 8: Comparação teórica e prática sobre o Average Packet Delay de dados e VoIP

d) Consider the case of $\lambda=1500\text{pps}$, $C=10\text{Mbps}$ and $f=10.000$ Bytes. Run Simulator3 50 times with a stopping criterion of $P = 10000$ each run and compute the estimated values and the 90% confidence intervals of the average delay and packet loss performance parameters of data packets and VoIP packets when $n = 10, 20, 30$ and 40 VoIP flows. Present the results of each of the 4 performance parameters (average data packet delay, average VoIP packet delay, data packet loss and VoIP packet loss) in different figures (in all cases, in bar charts with the confidence intervals in error bars). Justify the results and take conclusions concerning the impact of the number of VoIP flows in the obtained average packet delay and packet loss of each service when both services (data and VoIP) are statistically multiplexed in a single FIFO queue of small size.

Usando novamente o Simulador 3 e efetuando a simulação para diferente número de VoIP flows podemos constatar que quanto mais flows existirem maior será o atraso médio, tanto para pacotes VoIP como para pacotes de dados. Em ambos os casos os valores de atraso estão na mesma gama. Por outro lado, no caso da análise de perda de pacotes podemos observar que quanto maior o número de flows, mais acentuada é perda de pacotes, sejam eles VoIP ou não. No entanto, embora sejam percentagens residuais em ambos os casos podemos constatar que a perda de pacotes VoIP é percentualmente dez vezes menor do que de pacotes de dados, sendo possível que o tamanho médio dos pacotes VoIP seja fulcral para que a sua taxa de perda seja muito mais baixa que a dos pacotes de dados, já que estes possuem uma média de 120Bytes em comparação com o tamanho médio dos pacotes de VoIP que possuem em média 620 Bytes, se analisar-mos as variáveis `avgPacketSizeVoIP` e `avgPacketSizeData`, presentes no código de teste do exercício c).

Listing 9: Código d)

```

1 % TASK 2
2 % D
3
4 lambda = 1500;
5 C = 10;
6 f = 10000;
7 N = 50;
8 P = 10000;
9 b = 10^-6;
10 n = [10, 20, 30, 40];
11
12 per=zeros(N,7); % vetor com N valores de simula o
13
14 nsize = length(n);
15 res = zeros(N,7,nsize);
16
17
18 for i = 1:nsize
19   for it = 1:N
20     [per(it,1),per(it,2),per(it,3),per(it,4),per(it,5),per(it,6),per(it,7)] =...
21       Simulator3(lambda, C, f, P, n(i));
22       %PLd , PLv , APDd , APDv , MPDd , MPDv , TT
23   end

```

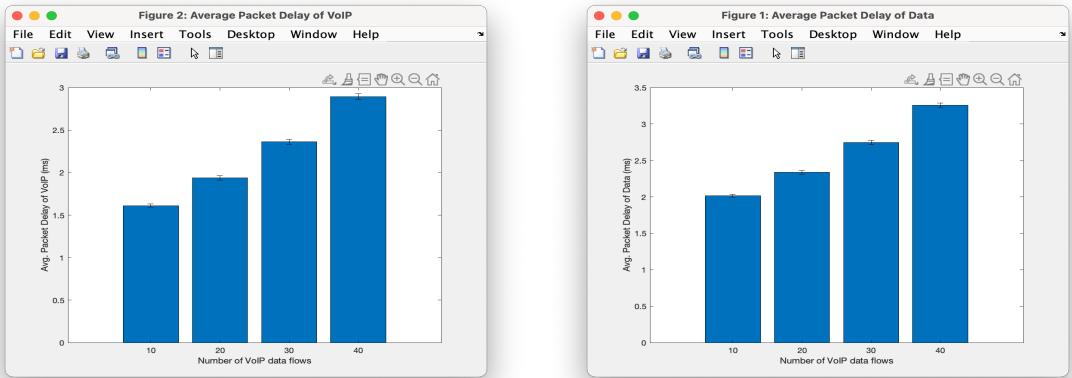
```

24     res(:, :, i) = per(:, :);
25 end
26 %%
27 alfa = 0.1; % intervalo de confian a 90%
28 media = mean(res);
29 term = norminv(1-alfa/2)*sqrt(var(res)/N);
30
31 av_pkt_delay_data = zeros(nsize,2);
32 av_pkt_delay_voip = zeros(nsize,2);
33 av_pkt_loss_data = zeros(nsize,2);
34 av_pkt_loss_voip = zeros(nsize,2);
35
36 for it = 1:nsize
37     av_pkt_delay_data(it,:) = [media(:, 3, it), term(:, 3,it)];
38     av_pkt_delay_voip(it,:) = [media(:, 4, it), term(:, 4,it)];
39     av_pkt_loss_data(it,:) = [media(:, 1, it), term(:, 1,it)];
40     av_pkt_loss_voip(it,:) = [media(:, 2, it), term(:, 2,it)];
41 end
42
43 %%
44 data_delay = zeros(1, nsize);
45 data_delay_err = zeros(1, nsize);
46 voip_delay = zeros(1, nsize);
47 voip_delay_err = zeros(1, nsize);
48 data_loss = zeros(1, nsize);
49 data_loss_err = zeros(1, nsize);
50 voip_loss = zeros(1, nsize);
51 voip_loss_err = zeros(1, nsize);
52
53 for it2 = 1:nsize
54     data_delay(1, it2) = av_pkt_delay_data(it2, 1);
55     data_delay_err(1, it2) = av_pkt_delay_data(it2, 2);
56     voip_delay(1, it2) = av_pkt_delay_voip(it2, 1);
57     voip_delay_err(1, it2) = av_pkt_delay_voip(it2, 2);
58
59     data_loss(1, it2) = av_pkt_loss_data(it2, 1);
60     data_loss_err(1, it2) = av_pkt_loss_data(it2, 2);
61     voip_loss(1, it2) = av_pkt_loss_voip(it2, 1);
62     voip_loss_err(1, it2) = av_pkt_loss_voip(it2, 2);
63 end
64
65 %%
66 x = 1:nsize;
67
68 figure('Name', 'Average Packet Delay of Data')
69 bdd = bar(x, data_delay);
70 xlabel('Number of VoIP data flows')
71 set(gca, 'xticklabel',n)
72 ylabel('Avg. Packet Delay of Data (ms)')
73
74 hold on
75 er = errorbar(x, data_delay, data_delay_err);
76 er.Color = [0 0 0];
77 er.LineStyle = 'none';
78 hold off
79
80 figure('Name', 'Average Packet Delay of VoIP')
81 bvd = bar(x, voip_delay);
82 xlabel('Number of VoIP data flows')
83
```

```

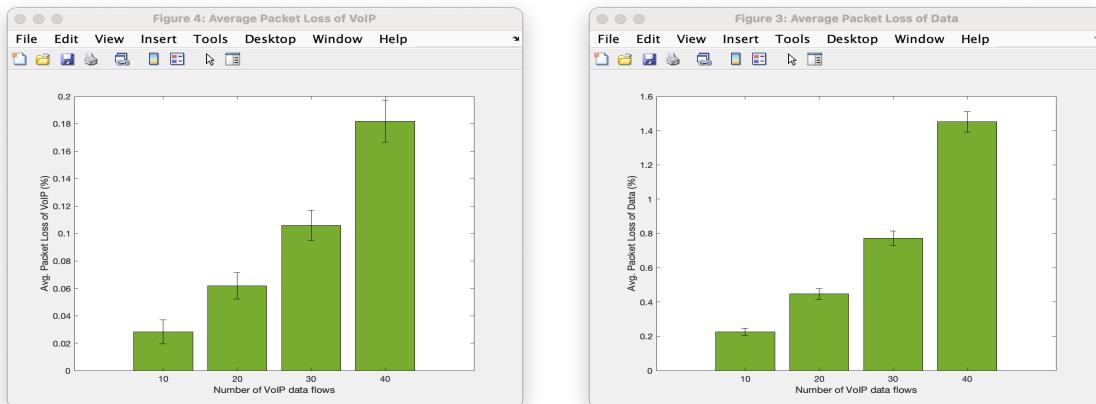
84 set(gca, 'xticklabel',n)
85 ylabel('Avg. Packet Delay of VoIP (ms)')
86
87 hold on
88 er = errorbar(x, voip_delay, voip_delay_err);
89 er.Color = [0 0 0];
90 er.LineStyle = 'none';
91 hold off
92
93 figure('Name', 'Average Packet Loss of Data')
94 bdl = bar(x, data_loss);
95 bdl.FaceColor = '#77AC30';
96 xlabel('Number of VoIP data flows')
97 set(gca, 'xticklabel',n)
98 ylabel('Avg. Packet Loss of Data (%)')
99
100 hold on
101 er = errorbar(x, data_loss, data_loss_err);
102 er.Color = [0 0 0];
103 er.LineStyle = 'none';
104 hold off
105
106 figure('Name', 'Average Packet Loss of VoIP')
107 bvl = bar(x, voip_loss);
108 bvl.FaceColor = '#77AC30';
109 xlabel('Number of VoIP data flows')
110 set(gca, 'xticklabel',n)
111 ylabel('Avg. Packet Loss of VoIP (%)')
112
113 hold on
114 er = errorbar(x, voip_loss, voip_loss_err);
115 er.Color = [0 0 0];
116 er.LineStyle = 'none';
117 hold off

```



(a) Average VoIP Packet Delay

(b) Average Data Packet Delay



(c) Average VoIP Packet Loss

(d) Average Data Packet Loss

Figura 9: Average Packet Delay/Loss de dados e VoIP baseado no número de VoIP flows

e) Repeat experiment 2.d but now with Simulator4. Justify these results and the differences between them and the results of 2.d. Take conclusions concerning the impact of the number of VoIP flows in the obtained average packet delay and packet loss of each service when VoIP service is supported with a priority which is higher than the data service and the queue is of small size.

O código de teste para o exercício e) é bastante semelhante ao do exercício d), sendo que a única alteração efetuada está listada em Listing 10.

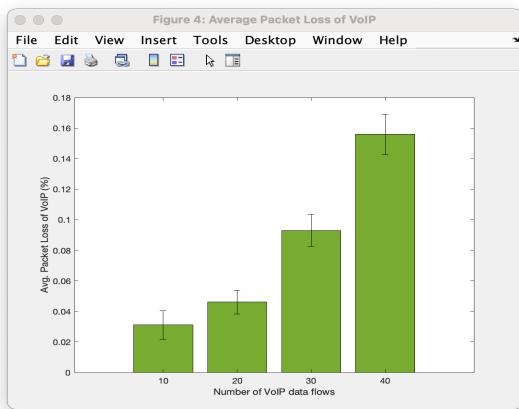
Listing 10: Código e)

```

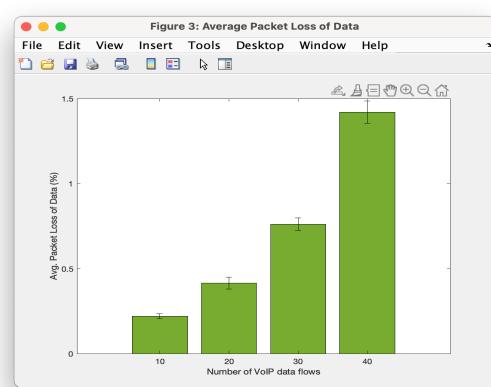
1 for i = 1:nsize
2   for it = 1:N
3     [per(it,1),per(it,2),per(it,3),per(it,4),per(it,5),per(it,6),per(it,7)] =...
4       Simulator4(lambda, C, f, P, n(i));
5       %PLd , PLv , APDd , APDv , MPDd , MPDv , TT
6   end
7   res(:, :, i) = per(:, :);
8 end

```

Podemos observar na Figura 10 que a diferença mais significativa é ao nível do delay para pacotes VoIP. Na implementação com o Simulador 3 há uma diferença maior no atraso médio quando varia o número de flows. Aqui essa diferença é substancialmente mais reduzida. Também podemos constatar que o atraso médio global para pacotes VoIP é mais reduzido do que na implementação anterior, o que se deve ao facto de o Simulador 4 dar mais prioridade a pacotes VoIP do que a pacotes de dados, como é explicado na alínea b). A perda de pacotes no caso do VoIP também é substancialmente mais reduzida que para os pacotes dos dados devido ao facto dos pacotes VoIP possuirem, em média, menor tamanho que os pacotes de dados, como já foi explicado em d).



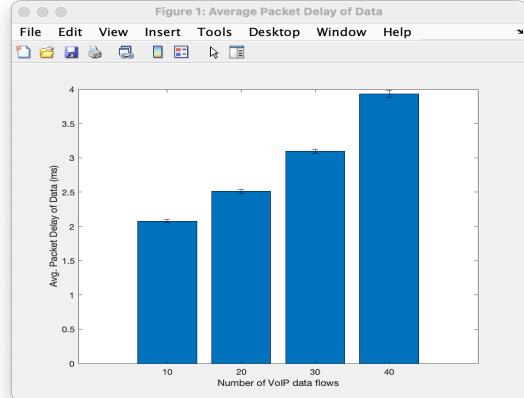
(a) Average VoIP Packet Loss



(b) Average Data Packet Loss



(c) Average VoIP Packet Delay



(d) Average Data Packet Delay

Figura 10: Average Packet Delay/Loss de dados e VoIP baseado no número de VoIP flows, com base em prioridades

f) Develop a new version of Simulator4 to consider that VoIP packets are always accepted in the queue (if there is enough space) but data packets are accepted in the queue only if the total queue occupation does not become higher than 90% (a simplified version of WRED – Weighted Random Early Discard). Repeat experiment 2.e but now with the new version of Simulator4. Justify these results and the differences between them and the results of 2.e. Take conclusions concerning the impact of the number of VoIP flows in the obtained average packet delay and packet loss of each service when (i) VoIP service is supported with a priority which is higher than the data service and (ii) the packet acceptance in the queue is differentiated.

Aqui queremos garantir que pacotes de dados são aceites na queue apenas se a total ocupação desta for inferior a 90 %. Para isso, como pode ser observado em Listing 11, no Simulador 4 adicionamos essa condição para que se aceite pacotes de dados. Por outro lado, os pacotes VoIP são aceites sem qualquer condicionante. Para testar este simulador usámos novamente o código do exercício anterior.

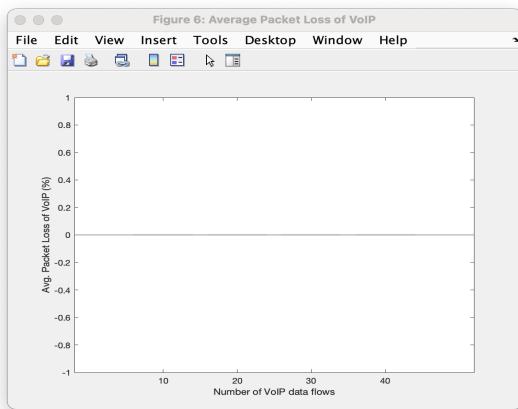
Listing 11: Código e)

```

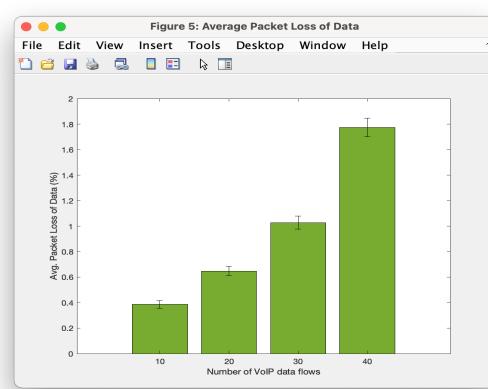
1  if QUEUEOCCUPATION + PacketSize <= f*0.9
2      QUEUE = [QUEUE;PacketSize , Clock, data];
3      QUEUEOCCUPATION= QUEUEOCCUPATION + PacketSize;
4  else
5      LOSTPACKETS = LOSTPACKETS + 1;
6 end

```

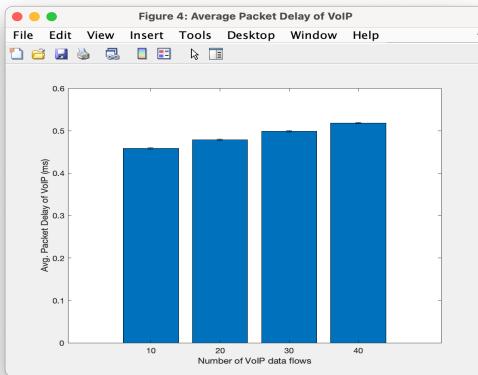
Podemos observar que os tempos de delay são semelhantes aos do exercício anterior no caso de pacotes de dados. Já no caso dos pacotes VoIP, como seria expectável, verifica-se uma diminuição no tempo de delay. Os pacotes VoIP ficam menos tempo à espera pois têm prioridade sobre os pacotes de dados. Em ambos os casos o delay aumenta com o aumento de fluxos, de forma mais residual para pacotes VoIP e de forma mais acentuada para pacotes de dados. Analisando a perda de pacotes podemos constatar um aumento ligeiro da perda de pacotes de dados em relação ao exercício anterior. No caso dos pacotes VoIP não se verificam quaisquer perdas de pacotes. Isto deve-se ao facto de, não só estes pacotes terem prioridade sobre os restantes mas também por terem menor tamanho. Tudo isto faz diminuir drasticamente o risco de perda.



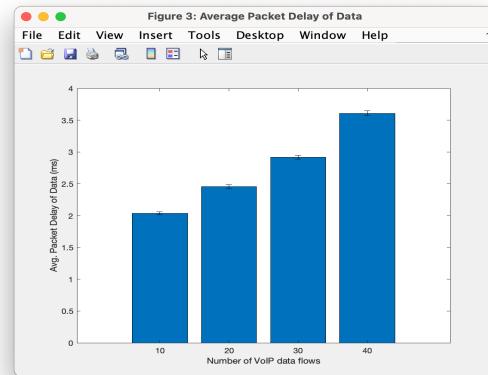
(a) Average VoIP Packet Loss



(b) Average Data Packet Loss



(c) Average VoIP Packet Delay



(d) Average Data Packet Delay

Figura 11: Average Packet Delay/Loss de dados e VoIP baseado no número de VoIP flows, com base em prioridades

Code attachments

Listing 12: Simulador 1

```

1 function [PL , APD , MPD , TT] = Simulator1(lambda,C,f,P)
2 % INPUT PARAMETERS:
3 % lambda - packet rate (packets/sec)
4 % C      - link bandwidth (Mbps)
5 % f      - queue size (Bytes)
6 % P      - number of packets (stopping criterium)
7 % OUTPUT PARAMETERS:
8 % PL    - packet loss (%)
9 % APD   - average packet delay (milliseconds)
10 % MPD  - maximum packet delay (milliseconds)
11 % TT   - transmitted throughput (Mbps)
12
13 %Events:
14 ARRIVAL= 0;           % Arrival of a packet
15 DEPARTURE= 1;         % Departure of a packet
16
17 %State variables:
18 STATE = 0;            % 0 - connection free; 1 - connection bysy
19 QUEUEOCCUPATION= 0;   % Occupation of the queue (in Bytes)
20 QUEUE= [];             % Size and arriving time instant of each packet in the queue
21
22 %Statistical Counters:
23 TOTALPACKETS= 0;       % No. of packets arrived to the system
24 LOSTPACKETS= 0;         % No. of packets dropped due to buffer overflow
25 TRANSMITTEDPACKETS= 0; % No. of transmitted packets
26 TRANSMITTEDBYTES= 0;   % Sum of the Bytes of transmitted packets
27 DELAYS= 0;              % Sum of the delays of transmitted packets
28 MAXDELAY= 0;            % Maximum delay among all transmitted packets
29
30 % Initializing the simulation clock:
31 Clock= 0;
32
33 % Initializing the List of Events with the first ARRIVAL:
34 tmp= Clock + exprnd(1/lambda);
35 EventList = [ARRIVAL, tmp, GeneratePacketSize(), tmp];
36
37 %Simulation loop:
38 while TRANSMITTEDPACKETS<P           % Stopping criterium
39     EventList= sortrows(EventList,2);    % Order EventList by time
40     Event= EventList(1,1);              % Get first event and
41     Clock= EventList(1,2);             % and
42     PacketSize= EventList(1,3);        % associated
43     ArrivalInstant= EventList(1,4);    % parameters.
44     EventList(1,:)= [];                % Eliminate first event
45     switch Event
46         case ARRIVAL                  % If first event is an ARRIVAL
47             TOTALPACKETS= TOTALPACKETS+1;
48             tmp= Clock + exprnd(1/lambda);
49             EventList = [EventList; ARRIVAL, tmp, GeneratePacketSize(), tmp];
50             if STATE==0
51                 STATE= 1;
52                 EventList = [EventList; DEPARTURE, Clock + 8*PacketSize/(C*10^6),...
53                               PacketSize, Clock];

```

```

54     else
55         if QUEUEOCCUPATION + PacketSize <= f
56             QUEUE= [QUEUE;PacketSize , Clock];
57             QUEUEOCCUPATION= QUEUEOCCUPATION + PacketSize;
58         else
59             LOSTPACKETS= LOSTPACKETS + 1;
60         end
61     end
62     case DEPARTURE % If first event is a DEPARTURE
63         TRANSMITTEDBYTES= TRANSMITTEDBYTES + PacketSize;
64         DELAYS= DELAYS + (Clock - ArrivalInstant);
65         if Clock - ArrivalInstant > MAXDELAY
66             MAXDELAY= Clock - ArrivalInstant;
67         end
68         TRANSMITTEDPACKETS= TRANSMITTEDPACKETS + 1;
69         if QUEUEOCCUPATION > 0
70             EventList = [EventList; DEPARTURE, Clock + 8*QUEUE(1,1)/(C*10^6),...
71                         QUEUE(1,1), QUEUE(1,2)];
72             QUEUEOCCUPATION= QUEUEOCCUPATION - QUEUE(1,1);
73             QUEUE(1,:)= [];
74         else
75             STATE= 0;
76         end
77     end
78 end
79
80 %Performance parameters determination:
81 PL= 100*LOSTPACKETS/TOTALPACKETS; % in %
82 APD= 1000*DELAYS/TRANSMITTEDPACKETS; % in milliseconds
83 MPD= 1000*MAXDELAY; % in milliseconds
84 TT= 10^(-6)*TRANSMITTEDBYTES*8/Clock; % in Mbps
85
86 end
87
88 function out= GeneratePacketSize()
89     aux= rand();
90     aux2= [65:109 111:1517];
91     if aux <= 0.19
92         out= 64;
93     elseif aux <= 0.19 + 0.23
94         out= 110;
95     elseif aux <= 0.19 + 0.23 + 0.17
96         out= 1518;
97     else
98         out = aux2(randi(length(aux2)));
99     end
100 end

```

Listing 13: Simulador_new

```

1 function [PL , APD, APD_64, APD_110, APD_1518 , MPD , TT] = Simulator1_new(lambda,C,f,P)
2 % INPUT PARAMETERS:
3 % lambda - packet rate (packets/sec)
4 % C - link bandwidth (Mbps)
5 % f - queue size (Bytes)
6 % P - number of packets (stopping criterium)
7 % OUTPUT PARAMETERS:
8 % PL - packet loss (%)
9 % APD - average packet delay (milliseconds)
10 % APD_64 - average packet delay for 64 byte packets (milliseconds)
11 % APD_110 - average packet delay for 110 byte packets (milliseconds)
12 % APD_1518 - average packet delay for 1518 byte packets (milliseconds)
13 % MPD - maximum packet delay (milliseconds)
14 % TT - transmitted throughput (Mbps)
15
16 %Events:
17 ARRIVAL= 0; % Arrival of a packet
18 DEPARTURE= 1; % Departure of a packet
19
20 %State variables:
21 STATE = 0; % 0 - connection free; 1 - connection bysy
22 QUEUEOCCUPATION= 0; % Occupation of the queue (in Bytes)
23 QUEUE= []; % Size and arriving time instant of each packet in the queue
24
25 %Statistical Counters:
26 TOTALPACKETS= 0; % No. of packets arrived to the system
27 LOSTPACKETS= 0; % No. of packets dropped due to buffer overflow
28 TRANSMITTEDPACKETS= 0; % No. of transmitted packets
29 TRANSMITTEDPACKETS_64 = 0;
30 TRANSMITTEDPACKETS_110 = 0;
31 TRANSMITTEDPACKETS_1518 = 0;
32 TRANSMITTEDBYTES= 0; % Sum of the Bytes of transmitted packets
33 DELAYS= 0; % Sum of the delays of transmitted packets
34 DELAYS_64 = 0;
35 DELAYS_110 = 0;
36 DELAYS_1518 = 0;
37 MAXDELAY= 0; % Maximum delay among all transmitted packets
38
39 % Initializing the simulation clock:
40 Clock= 0;
41
42 % Initializing the List of Events with the first ARRIVAL:
43 tmp= Clock + exprnd(1/lambda);
44 EventList = [ARRIVAL, tmp, GeneratePacketSize(), tmp];
45
46 %Simulation loop:
47 while TRANSMITTEDPACKETS<P % Stopping criterium
48     EventList= sortrows(EventList,2); % Order EventList by time
49     Event= EventList(1,1); % Get first event and
50     Clock= EventList(1,2); % and
51     PacketSize= EventList(1,3); % associated
52     ArrivalInstant= EventList(1,4); % parameters.
53     EventList(1,:)= []; % Eliminate first event
54     switch Event
55         case ARRIVAL % If first event is an ARRIVAL
56             TOTALPACKETS= TOTALPACKETS+1;
57             tmp= Clock + exprnd(1/lambda);
58             EventList = [EventList; ARRIVAL, tmp, GeneratePacketSize(), tmp];
59             if STATE==0

```

```

60         STATE= 1;
61         EventList = [EventList; DEPARTURE, Clock + 8*PacketSize/(C*10^6),...
62                         PacketSize, Clock];
63     else
64         if QUEUEOCCUPATION + PacketSize <= f
65             QUEUE= [QUEUE;PacketSize , Clock];
66             QUEUEOCCUPATION= QUEUEOCCUPATION + PacketSize;
67         else
68             LOSTPACKETS= LOSTPACKETS + 1;
69         end
70     end
71     case DEPARTURE                                % If first event is a DEPARTURE
72         TRANSMITTEDBYTES= TRANSMITTEDBYTES + PacketSize;
73         DELAYS= DELAYS + (Clock - ArrivalInstant);
74         if Clock - ArrivalInstant > MAXDELAY
75             MAXDELAY= Clock - ArrivalInstant;
76         end
77         TRANSMITTEDPACKETS= TRANSMITTEDPACKETS + 1;
78         if (PacketSize == 64)
79             TRANSMITTEDPACKETS_64 = TRANSMITTEDPACKETS_64 + 1;
80             DELAYS_64 = DELAYS_64 + (Clock - ArrivalInstant);
81         end
82         if (PacketSize == 110)
83             TRANSMITTEDPACKETS_110 = TRANSMITTEDPACKETS_110 + 1;
84             DELAYS_110 = DELAYS_110 + (Clock - ArrivalInstant);
85         end
86         if (PacketSize == 1518)
87             TRANSMITTEDPACKETS_1518 = TRANSMITTEDPACKETS_1518 + 1;
88             DELAYS_1518 = DELAYS_1518 + (Clock - ArrivalInstant);
89         end
90         if QUEUEOCCUPATION > 0
91             EventList = [EventList; DEPARTURE, Clock + 8*QUEUE(1,1)/(C*10^6),...
92                             QUEUE(1,1), QUEUE(1,2)];
93             QUEUEOCCUPATION= QUEUEOCCUPATION - QUEUE(1,1);
94             QUEUE(1,:)= [];
95         else
96             STATE= 0;
97         end
98     end
99 end
100
101 %Performance parameters determination:
102 PL= 100*LOSTPACKETS/TOTALPACKETS;           % in %
103 APD= 1000*DELAYS/TRANSMITTEDPACKETS;        % in milliseconds
104 APD_64 = 1000*DELAYS_64/TRANSMITTEDPACKETS_64;
105 APD_110 = 1000*DELAYS_110/TRANSMITTEDPACKETS_110;
106 APD_1518 = 1000*DELAYS_1518/TRANSMITTEDPACKETS_1518;
107 MPD= 1000*MAXDELAY;                          % in milliseconds
108 TT= 10^(-6)*TRANSMITTEDBYTES*8/Clock;       % in Mbps
109
110 end
111
112 function out= GeneratePacketSize()
113     aux= rand();
114     aux2= [65:109 111:1517];
115     if aux <= 0.19
116         out= 64;
117     elseif aux <= 0.19 + 0.23
118         out= 110;
119     elseif aux <= 0.19 + 0.23 + 0.17

```

```
120         out= 1518;
121     else
122         out = aux2(randi(length(aux2)));
123     end
124 end
```

Listing 14: Simulador 3

```

1      function [PLd , PLv , APDd , APDv , MPDd , MPDv , TT] = Simulator3(lambda,C,f,P,n)
2 % INPUT PARAMETERS:
3 % lambda - packet rate (packets/sec)
4 % C      - link bandwidth (Mbps)
5 % f      - queue size (Bytes)
6 % P      - number of packets (stopping criterium)
7 % n      - number of VoIP data flows
8 % OUTPUT PARAMETERS:
9 % PLd   - data packet loss (%)
10 % PLv   - VoIP packet loss (%)
11 % APDd  - average data packet delay (milliseconds)
12 % APDv  - average VoIP packet delay (milliseconds)
13 % MPDd  - maximum data packet delay (milliseconds)
14 % MPDv  - maximum VoIP packet delay (milliseconds)
15 % TT    - transmitted throughput (Mbps)
16
17
18 %Types of package
19 data = 0;
20 voip = 1;
21
22 %Events:
23 ARRIVAL= 0;           % Arrival of a packet
24 DEPARTURE= 1;         % Departure of a packet
25
26 %State variables:
27 STATE = 0;            % 0 - connection free; 1 - connection bysy
28 QUEUEOCCUPATION= 0;   % Occupation of the queue (in Bytes)
29 QUEUE= [];             % Size and arriving time instant of each packet in the queue
30
31 %Statistical Counters:
32 TOTALPACKETS= 0;       % No. of packets arrived to the system
33 LOSTPACKETS= 0;         % No. of packets dropped due to buffer overflow
34 TRANSMITTEDPACKETS= 0; % No. of transmitted packets
35 TRANSMITTEDBYTES= 0;    % Sum of the Bytes of transmitted packets
36 DELAYS= 0;              % Sum of the delays of transmitted packets
37 MAXDELAY= 0;            % Maximum delay among all transmitted packets
38
39 TOTALVOIPPACKETS= 0;    % No. of VoIP packets arrived to the system
40 LOSTVOIPPACKETS= 0;     % No. of VoIP packets dropped due to buffer overflow
41 TRANSMITTEDVOIPACKETS= 0; % No. of transmitted VoIP packets
42 TRANSMITTEDVOIPBYTES= 0; % Sum of the Bytes of transmitted VoIP packets
43 VOIPDELAYS= 0;           % Sum of the delays of transmitted VoIP packets
44 VOIPMAXDELAY= 0;         % Maximum delay among all transmitted VoIP packets
45
46 % Initializing the simulation clock:
47 Clock= 0;
48
49 % Initializing the List of Events with the first ARRIVAL:
50 tmp= Clock + exprnd(1/lambda);
51 EventList = [ARRIVAL, tmp, GeneratePacketSize(), tmp, data];
52
53 for x = 1:n
54     tmpvi = Clock + rand() * 0.02;
55     EventList = [EventList; ARRIVAL, tmpvi, randi(20)+110, tmpvi, voip];
56 end
57
58 %for n vezes para acrescentar a lista de eventos para meter pacotes voips aleatorios entre 0 e 20 ms
59

```

```

60 %Simulation loop:
61 while TRANSMITTEDPACKETS<P
62     EventList= sortrows(EventList,2); % Stopping criterium
63     Event= EventList(1,1); % Order EventList by time
64     Clock= EventList(1,2); % Get first event and
65     PacketSize= EventList(1,3); % and
66     ArrivalInstant= EventList(1,4); % associated
67     Type = EventList(1,5); % parameters.
68     EventList(1,:)= [];
69     switch Event % Eliminate first event
70         case ARRIVAL % If first event is an ARRIVAL
71             if Type==data
72                 TOTALPACKETS= TOTALPACKETS+1;
73                 tmp= Clock + exprnd(1/lambda);
74                 EventList = [EventList; ARRIVAL, tmp, GeneratePacketSize(), tmp, data];
75                 if STATE==0
76                     STATE= 1;
77                     EventList = [EventList; DEPARTURE, Clock + 8*PacketSize/(C*10^6),...
78                                 PacketSize, Clock, data];
79                 else
80                     if QUEUEOCCUPATION + PacketSize <= f
81                         QUEUE = [QUEUE;PacketSize , Clock, data];
82                         QUEUEOCCUPATION= QUEUEOCCUPATION + PacketSize;
83                     else
84                         LOSTPACKETS = LOSTPACKETS + 1;
85                     end
86                 end
87             else
88                 TOTALVOIPPACKETS = TOTALVOIPPACKETS+1;
89                 tmpv = Clock + rand() * 0.008 + 0.016;
90                 EventList = [EventList; ARRIVAL, tmpv, randi(20)+110, tmpv, voip];
91                 if STATE==0
92                     STATE= 1;
93                     EventList = [EventList; DEPARTURE, Clock + 8*PacketSize/(C*10^6),...
94                                 PacketSize, Clock, voip];
95                 else
96                     if QUEUEOCCUPATION + PacketSize <= f
97                         QUEUE = [QUEUE;PacketSize , Clock, voip];
98                         QUEUEOCCUPATION= QUEUEOCCUPATION + PacketSize;
99                     else
100                         LOSTVOIPACKETS = LOSTVOIPACKETS + 1;
101                     end
102                 end
103             end
104         case DEPARTURE % If first event is a DEPARTURE
105             if Type==data
106                 TRANSMITTEDBYTES= TRANSMITTEDBYTES + PacketSize;
107                 DELAYS= DELAYS + (Clock - ArrivalInstant);
108                 if Clock - ArrivalInstant > MAXDELAY
109                     MAXDELAY= Clock - ArrivalInstant;
110                 end
111                 TRANSMITTEDPACKETS= TRANSMITTEDPACKETS + 1;
112             else
113                 TRANSMITTEDVOIPBYTES= TRANSMITTEDVOIPBYTES + PacketSize;
114                 VOIPDELAYS= VOIPDELAYS + (Clock - ArrivalInstant);
115                 if Clock - ArrivalInstant > VOIPMAXDELAY
116                     VOIPMAXDELAY= Clock - ArrivalInstant;
117                 end
118                 TRANSMITTEDVOIPACKETS= TRANSMITTEDVOIPACKETS + 1;
119             end

```

```

120
121     if QUEUEOCCUPATION > 0
122         EventList = [EventList; DEPARTURE, Clock + 8*QUEUE(1,1)/(C*10^6),...
123             QUEUE(1,1), QUEUE(1,2), QUEUE(1,3)];
124         QUEUEOCCUPATION= QUEUEOCCUPATION - QUEUE(1,1);
125         QUEUE(1,:)= [];
126     else
127         STATE= 0;
128     end
129 end
130
131 %Performance parameters determination:
132 PLd= 100*LOSTPACKETS/TOTALPACKETS;          % in %
133 APDd= 1000*DELAYS/TRANSMITTEDPACKETS;        % in milliseconds
134 MPDd= 1000*MAXDELAY;                         % in milliseconds
135
136 PLv= 100*LOSTVOIPPACKETS/TOTALVOIPPACKETS;    % in %
137 APDv= 1000*VOIPDELAYS/TRANSMITTEDVOIPACKETS;  % in milliseconds
138 MPDv= 1000*VOIPMAXDELAY;                      % in milliseconds
139
140 TT= 10^(-6)*TRANSMITTEDBYTES*8/Clock;      % in Mbps
141
142 end
143
144 function out= GeneratePacketSize()
145     aux= rand();
146     aux2= [65:109 111:1517];
147     if aux <= 0.19
148         out= 64;
149     elseif aux <= 0.19 + 0.23
150         out= 110;
151     elseif aux <= 0.19 + 0.23 + 0.17
152         out= 1518;
153     else
154         out = aux2(randi(length(aux2)));
155     end
156 end
157

```

Listing 15: Simulador 4

```

1
2 function [PLd , PLv , APDd , APDv , MPDd , MPDv , TT] = Simulator4(lambda,C,f,P,n)
3 % INPUT PARAMETERS:
4 % lambda - packet rate (packets/sec)
5 % C      - link bandwidth (Mbps)
6 % f      - queue size (Bytes)
7 % P      - number of packets (stopping criterium)
8 % n      - number of VoIP data flows
9 % OUTPUT PARAMETERS:
10 % PLd   - data packet loss (%)
11 % PLv   - VoIP packet loss (%)
12 % APDd  - average data packet delay (milliseconds)
13 % APDv  - average VoIP packet delay (milliseconds)
14 % MPDd  - maximum data packet delay (milliseconds)
15 % MPDv  - maximum VoIP packet delay (milliseconds)
16 % TT    - transmitted throughput (Mbps)
17
18 %Types of package
19 data = 0;
20 voip = 1;
21
22 %Events:
23 ARRIVAL= 0;           % Arrival of a packet
24 DEPARTURE= 1;         % Departure of a packet
25
26 %State variables:
27 STATE = 0;            % 0 - connection free; 1 - connection bysy
28 QUEUEOCCUPATION= 0;   % Occupation of the queue (in Bytes)
29 QUEUE= [];             % Size and arriving time instant of each packet in the queue
30
31 %Statistical Counters:
32 TOTALPACKETS= 0;       % No. of packets arrived to the system
33 LOSTPACKETS= 0;         % No. of packets dropped due to buffer overflow
34 TRANSMITTEDPACKETS= 0; % No. of transmitted packets
35 TRANSMITTEDBYTES= 0;   % Sum of the Bytes of transmitted packets
36 DELAYS= 0;              % Sum of the delays of transmitted packets
37 MAXDELAY= 0;            % Maximum delay among all transmitted packets
38
39 TOTALVOIPPACKETS= 0;   % No. of VoIP packets arrived to the system
40 LOSTVOIPPACKETS= 0;     % No. of VoIP packets dropped due to buffer overflow
41 TRANSMITTEDVOIPACKETS= 0; % No. of transmitted VoIP packets
42 TRANSMITTEDVOIPBYTES= 0; % Sum of the Bytes of transmitted VoIP packets
43 VOIPDELAYS= 0;          % Sum of the delays of transmitted VoIP packets
44 VOIPMAXDELAY= 0;        % Maximum delay among all transmitted VoIP packets
45
46 % Initializing the simulation clock:
47 Clock= 0;
48
49 % Initializing the List of Events with the first ARRIVAL:
50 tmp= Clock + exprnd(1/lambda);
51 EventList = [ARRIVAL, tmp, GeneratePacketSize(), tmp, data];
52
53 for x = 1:n
54     tmpvi = Clock + rand() * 0.02;
55     EventList = [EventList; ARRIVAL, tmpvi, randi(20)+110, tmpvi, voip];
56 end
57
58 %for n vezes para acrescentar a lista de eventos para meter pacotes voips aleatorios entre 0 e 20 ms
59

```

```

60 %Simulation loop:
61 while TRANSMITTEDPACKETS<P
62     EventList= sortrows(EventList,2); % Stopping criterium
63     Event= EventList(1,1); % Order EventList by time
64     Clock= EventList(1,2); % Get first event and
65     PacketSize= EventList(1,3); % and
66     ArrivalInstant= EventList(1,4); % associated
67     Type = EventList(1,5); % parameters.
68     EventList(1,:)= [];
69     switch Event % Eliminate first event
70         case ARRIVAL % If first event is an ARRIVAL
71             if Type==data
72                 TOTALPACKETS= TOTALPACKETS+1;
73                 tmp= Clock + exprnd(1/lambda);
74                 EventList = [EventList; ARRIVAL, tmp, GeneratePacketSize(), tmp, data];
75                 if STATE==0
76                     STATE= 1;
77                     EventList = [EventList; DEPARTURE, Clock + 8*PacketSize/(C*10^6),...
78                                 PacketSize, Clock, data];
79                 else
80                     if QUEUEOCCUPATION + PacketSize <= f
81                         QUEUE = [QUEUE;PacketSize , Clock, data];
82                         QUEUEOCCUPATION= QUEUEOCCUPATION + PacketSize;
83                     else
84                         LOSTPACKETS = LOSTPACKETS + 1;
85                     end
86                 end
87             else
88                 TOTALVOIPPACKETS = TOTALVOIPPACKETS+1;
89                 tmpv = Clock + rand() * 0.008 + 0.016;
90                 EventList = [EventList; ARRIVAL, tmpv, randi(20)+110, tmpv, voip];
91                 if STATE==0
92                     STATE= 1;
93                     EventList = [EventList; DEPARTURE, Clock + 8*PacketSize/(C*10^6),...
94                                 PacketSize, Clock, voip];
95                 else
96                     if QUEUEOCCUPATION + PacketSize <= f
97                         QUEUE = [QUEUE; PacketSize , Clock, voip];
98                         QUEUEOCCUPATION= QUEUEOCCUPATION + PacketSize;
99                     else
100                         LOSTVOIPPACKETS = LOSTVOIPPACKETS + 1;
101                     end
102                 end
103             end
104         case DEPARTURE % If first event is a DEPARTURE
105             if Type==data
106                 TRANSMITTEDBYTES= TRANSMITTEDBYTES + PacketSize;
107                 DELAYS= DELAYS + (Clock - ArrivalInstant);
108                 if Clock - ArrivalInstant > MAXDELAY
109                     MAXDELAY= Clock - ArrivalInstant;
110                 end
111                 TRANSMITTEDPACKETS= TRANSMITTEDPACKETS + 1;
112             else
113                 TRANSMITTEDVOIPBYTES= TRANSMITTEDVOIPBYTES + PacketSize;
114                 VOIPDELAYS= VOIPDELAYS + (Clock - ArrivalInstant);
115                 if Clock - ArrivalInstant > VOIPMAXDELAY
116                     VOIPMAXDELAY= Clock - ArrivalInstant;
117                 end
118                 TRANSMITTEDVOIPPACKETS= TRANSMITTEDVOIPPACKETS + 1;
119             end

```

```

120
121     if QUEUEOCCUPATION > 0
122         QUEUE = sortrows(QUEUE, 3, "descend");
123         EventList = [EventList; DEPARTURE, Clock + 8*QUEUE(1,1)/(C*10^6),...
124             QUEUE(1,1), QUEUE(1,2), QUEUE(1,3)];
125         QUEUEOCCUPATION= QUEUEOCCUPATION - QUEUE(1,1);
126         QUEUE(1,:)= [];
127     else
128         STATE= 0;
129     end
130 end
131
132
133 %Performance parameters determination:
134 PLd= 100*LOSTPACKETS/TOTALPACKETS;           % in %
135 APDd= 1000*DELAYS/TRANSMITTEDPACKETS;        % in milliseconds
136 MPDd= 1000*MAXDELAY;                         % in milliseconds
137
138 PLv= 100*LOSTVOIPPACKETS/TOTALVOIPPACKETS;   % in %
139 APDv= 1000*VOIPDELAYS/TRANSMITTEDVOIPACKETS; % in milliseconds
140 MPDv= 1000*VOIPMAXDELAY;                      % in milliseconds
141
142 TT= 10^(-6)*TRANSMITTEDBYTES*8/Clock;      % in Mbps
143
144 end
145
146 function out= GeneratePacketSize()
147     aux= rand();
148     aux2= [65:109 111:1517];
149     if aux <= 0.19
150         out= 64;
151     elseif aux <= 0.19 + 0.23
152         out= 110;
153     elseif aux <= 0.19 + 0.23 + 0.17
154         out= 1518;
155     else
156         out = aux2(randi(length(aux2)));
157     end
158 end

```