Universidade de Aveiro

Departamento de Eletrónica, Telecomunicações e Informática

universidade de aveiro

Modelação e Desempenho de Redes e Serviços

Projeto 1

Tiago Alves (104110)

Rafael Amorim (98197)

31 de outubro de 2023

Índice

Índice das Figuras

Índice da Tabela

Introdução

De acordo com o solicitado no mini projeto da unidade curricular de Modelação e Desempenho de Redes e Serviços realizou-se este relatório apresentando excertos de código importantes para a explicação do raciocínio e descrevendo de forma sintetizada todas as conclusões retiradas dos resultados de cada exercício.

A estrutura do relatório consiste em duas partes, uma para cada tarefa.

O código do projeto, tal como, toda a gestão de tarefas encontra-se disponível em:

https://github.com/Tiago-AlvesUA /MDRS/

Tarefa 1

Exercício 1a

Para este exercício utilizou-se o simulador1 de eventos discretos para verificar o desempenho de uma ligação ponto a ponto entre um router de uma empresa e o seu ISP.

function [PL, APD, MPD, TT] = Simulator1(lambda, C, f, P);

Este simulador tem como parâmetros de entrada:

lambda - Taxa de chegada de pacotes (pps),
C - Capacidade da ligação (Mbps),
f - Tamanho da fila de espera (Bytes),
P - Número de pacotes (Critério de paragem),

Através deste simulador conseguimos obter 4 tipos de resultados para análise:

PL - Percentagem de perda de pacotes (%),
APD - Atraso médio de um pacote (ms),
MPD - Atraso máximo de um pacote (ms),
TT - Throughput Transmitido (Mbps),

Código:

```
Num = 20;
P = 100000;
alfa = 0.1;
rate = 1800;
f = 1000000;
C = [10 20 30 40];

APD = zeros(4, Num);
mean_APD = zeros(4, 1);
term_APD = zeros(4, 1);

for i = 1:4
    for j = 1:Num
        [~, APD(i, j), ~, ~] = Simulator1(rate, C(i), f, P);
    end
    mean_APD(i,1) = mean(APD(i,:));
    term_APD(i,1) = norminv(1-alfa/2) * sqrt(var(APD(i,:)) / Num);
    fprintf('APD C%d Mbps (ms) = %.2e +-%.2e\n',C(i),mean_APD(i,1),term_APD(i,1));
end

figure(1);
bar(C, mean_APD);
hold on;

er = errorbar(C, mean_APD, term_APD);
er.Color = [1 0 0];
er.LineStyle = 'none';

hold off;

xlabel('C (Mbps)');
ylabel('Average Packet Delay (ms)');
title('Average Packet Delay with Error Bars');
```

Resultados:

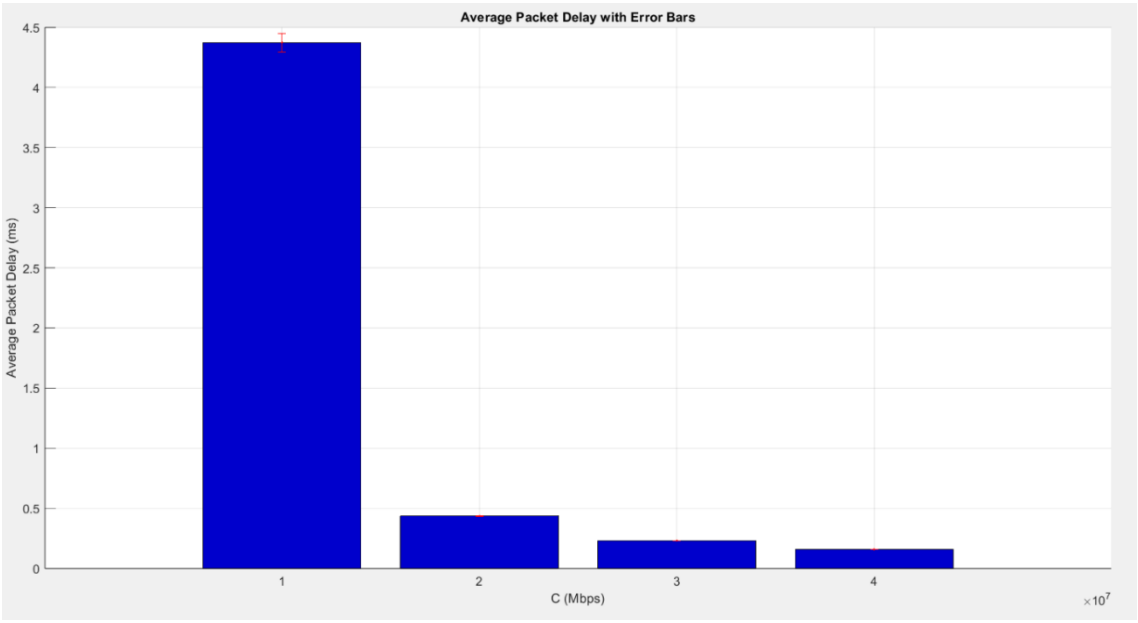

Figura 1: Resultado do exercício 1a

Explicação:

À medida que a capacidade da ligação aumenta, para 10, 20, 30 e 40 Mbps, conclui-se que o atraso médio dos pacotes no sistema vai diminuindo significativamente, porque estes são transmitidos mais rapidamente.

É também possível observar uma queda brusca do atraso médio entre os 10 e 20 Mbps sendo que depois esta descida passa a ser mais linear. A média do tamanho dos pacotes é aproximadamente 620 bytes, sendo possível confirmar pelos seguintes cálculos:

```
perc_left = (1 - (0.19 + 0.23 + 0.17)) / ((109 - 64) + (1517 - 110));
nMedio = (64)*0.19 + (110)*0.23 + (1518)*0.17 ...
    + sum((65:109) * perc_left) + sum((111:1518) * perc_left); %bytes
```

Multiplicando o tamanho do pacote por 8, para obter o resultado em bits, e pela taxa de chegada de pacotes (1800 pps) obtemos 8.93 Mbps. Em relação a uma capacidade de 10 Mbps, o valor obtido apresenta pouca diferença. No entanto, para as outras capacidades, a diferença já é mais significativa. A diferença para C10 é de 1.07, para C20 é de 11.07, para C30 é de 21.07 e para C40 é de 31.07 Mbps. A discrepância observada deve-se a 8.93 Mbps ser próximo do limite da capacidade de ligação.

Exercício 1b

O sistema apresentado é modelado por uma fila de espera M/G/1:

i) A chegada de clientes são processos de Poisson com taxa λ;

ii) O tempo de transmissão dos pacotes é genérico porque a distribuição dos tamanhos não segue uma distribuição comum;

iii) Este sistema tem 1 servidor, pois o canal de transmissão é usado para transmitir um pacote de cada vez;

iv) A fila de espera é de tamanho infinito.

Para a realização deste exercício foi utilizada a seguinte fórmula para cálculo dos valores teóricos:

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

Esta representa o atraso médio de cada cliente no sistema e soma o atraso médio na fila de espera com o tempo médio de atendimento.

Código:

O código foi implementado de forma a calcular os atrasos para as quatro capacidades diferentes, utilizando os diferentes valores da mesma para cálculo do tempo de atendimento, S.

```
C = [10 20 30 40];
C = C.*10^6;
perc_left = (1 - (0.19+0.23+0.17))/((109 - 64) + (1517 - 110));
x = 64:1518;
S = zeros(4,length(x)); % Tempo de Atendimento, S
S2 = zeros(4,length(x)); % S^2
APD = zeros(1,4);
for j = 1:4 % Loop valores de C
      for i = 1:length(x)
      if x(i) == 64
            s = (x(i) * 8) / C(j);
            S(j,i) = 0.19 * s;
            S2(j,i) = 0.19 * s^2;
      elseif x(i) == 110
            s = (x(i) * 8) / C(j);
            S(j,i) = 0.23 * s;
            S2(j,i) = 0.23 * s^2;
      elseif x(i) == 1518
            s = (x(i) * 8) / C(j);
            S(j,i) = 0.17 * s;
            S2(j,i) = 0.17 * s^2;
      else
            s = (x(i) * 8) / C(j);
            S(j,i) = perc_left * s;
            S2(j,i) = perc_left * s^2;
      end
end
ES = sum(S(j,:));
ES2 = sum(S2(j,:));
APD(j) = (rate * ES2)/(2*(1 - rate*ES)) + ES;
```

Resultados:

	C10	C20	C30	C40
Valores teóricos APD (ms)	4.39e+00	4.36e-01	2.31e-01	1.58e-01
Valores simulados APD (ms)	4.36e+00	4.37e-01	2.32e-01	1.58e-01

Tabela 1: Resultado do exercício 1b

Explicação:

Como podemos ver pelos valores teóricos obtidos, estes estão muito próximos dos valores obtidos pela simulação realizada na alínea anterior. Com isto concluímos que o simulador permite obter valores próximos em relação aos valores teóricos.

Exercício 1c

Para este exercício, continuamos a utilizar o simulador 1. Sabendo que a taxa de chegada de pacotes varia nos valores de 1000, 1300, 1600 e 1900 pacotes por segundo (pps), o nosso objetivo é analisar o impacto desta variação no atraso médio de pacotes e na taxa de transferência.

Nesta alínea utilizámos um script semelhante ao da alínea acima, mas mantemos a capacidade constante em 10 Mbps e variamos apenas as taxas de chegada de pacotes.

Código:

```
(...)

C = 10;
rates = [1000 1300 1600 1900];
TT = zeros(4, Num);
mean_Throughput = zeros(4, 1);
term_Throughput = zeros(4, 1);
APD = zeros(4, Num);
mean_APD = zeros(4, 1);
term_APD = zeros(4, 1);

for i = 1:4
    for j = 1:Num
        [~, APD(i, j), ~, TT(i, j)] = Simulator1(rates(i), C, f, P);
    end

    mean_APD(i,1) = mean(APD(i,:));
    term_APD(i,1) = norminv(1-alfa/2) * sqrt(var(APD(i,:)) / Num);

     (...)
    mean_Throughput(i,1) = mean(TT(i,:));
    term_Throughput(i,1) = norminv(1-alfa/2) * sqrt(var(TT(i,:)) / Num);

     (...)
```

Resultados:

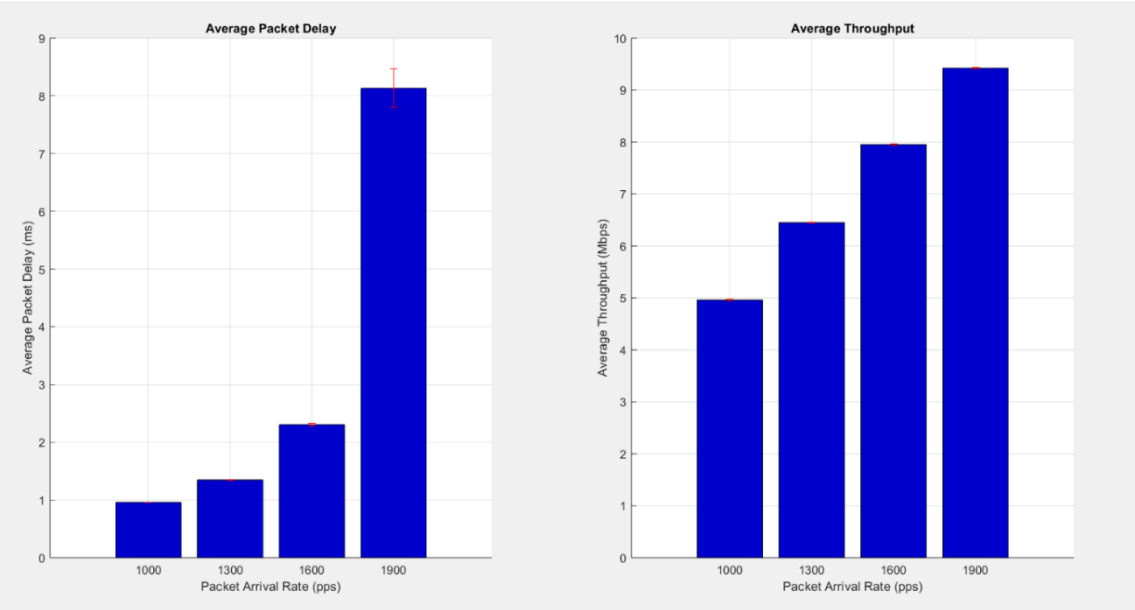

Figura 2: Resultado do exercício 1c

Explicação:

Analisando os gráficos, observamos que há um crescimento exponencial do atraso médio dos pacotes à medida que a taxa de chegada de pacotes aumenta, este fator deve-se à fila de espera acabar por ficar mais carregada porque a taxa de transferência está a aumentar e a aproximar da capacidade do sistema, isto acontece sequencialmente daí piorar cada vez mais o atraso.

A taxa de transferência vai aumentar de forma linear porque há mais pacotes a chegar ao sistema.

Exercício 1d

Neste exercício, utilizámos o simulador da alínea anterior, incorporando o Bit Error Rate (BER) como uma variável de entrada adicional, representada pela letra 'b' com um valor de 10^{-5}.

A fórmula binomial foi utilizada para descartar pacotes no simulador:

$$f(i) = \binom{n}{i} p^i (1-p)^{n-i}$$

Calculamos a probabilidade de um pacote ter zero erros com $f(i) = (1-p)^n$, se i = 0. Se o valor calculado aleatoriamente estiver dentro desta probabilidade então o pacote é transmitido, caso contrário o pacote é descartado.

Código:

```
(...)
b = 10^-5;
C = 10;
rates = [1000 1300 1600 1900];

TT_ber = zeros(4, Num);
mean_Throughput_ber = zeros(4, 1);
term_Throughput_ber = zeros(4, 1);
APD_ber = zeros(4, Num);
mean_APD_ber = zeros(4, 1);
term_APD_ber = zeros(4, 1);

for i = 1:4
      for j = 1:Num
         [~, APD_ber(i, j), ~, TT_ber(i, j)] = Simulator2(rates(i), C, f, P, b);
      end

      mean_APD_ber(i,1) = mean(APD_ber(i,:));
      term_APD_ber(i,1) = norminv(1-alfa/2) * sqrt(var(APD_ber(i,:)) / Num);

      (...)


      mean_Throughput_ber(i,1) = mean(TT_ber(i,:));
      term_Throughput_ber(i,1) = norminv(1-alfa/2) * sqrt(var(TT_ber(i,:)) / Num);

      (...)
```

Resultados:

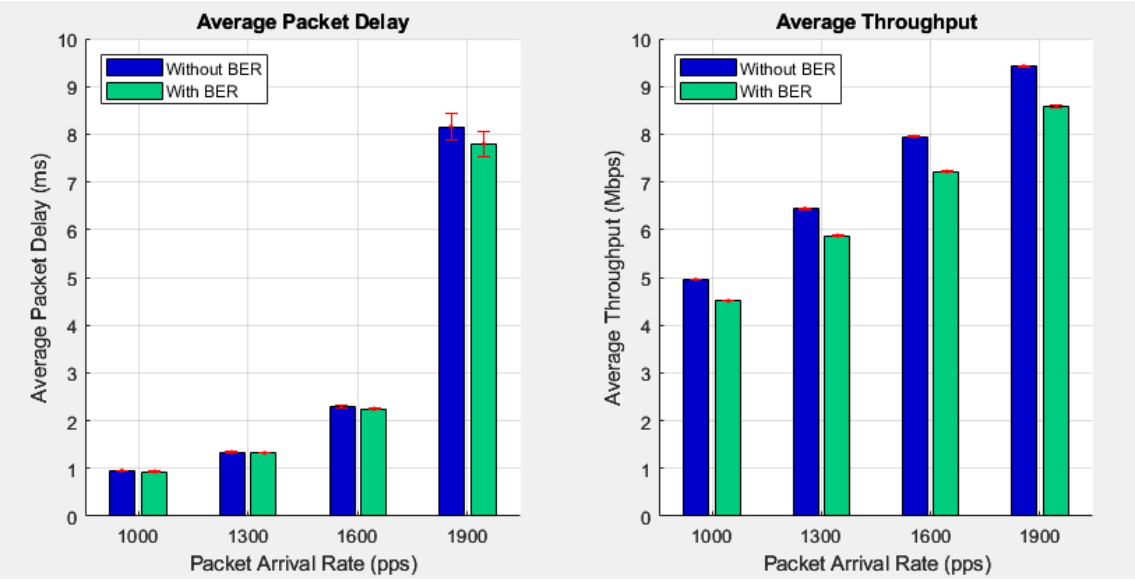

Figura 3: Resultado do exercício 1d

Conclusões:

Neste exercício, ao introduzir uma Taxa de Erro de Bit (BER) de 10^{-5}, podemos observar uma diminuição média na taxa de transferência, quando comparado com a transmissão sem BER. Além disso, notamos uma diferença significativa no atraso médio de pacotes, especialmente quando a taxa de chegada é de 1900 pps.

Quando consideramos a presença da Taxa de Erro de Bit (BER), observamos uma redução nos atrasos. Isso ocorre porque, com uma BER mais alta, os pacotes são descartados, resultando em menores atrasos e numa menor taxa de transferência.

Exercício 1e

Código:

Foram alteradas as percentagens de ocorrência dos vários tamanhos dos pacotes na função GeneratePacketSize() dos simuladores 1 e 2.

```
function out= GeneratePacketSize()
        aux= rand();
        aux2= [65:109 111:1517];
        if aux <= 0.25
                out= 64;
        elseif aux <= 0.25 + 0.17
                out= 110;
        elseif aux <= 0.25 + 0.17 + 0.11
                out= 1518;
        else
                out = aux2(randi(length(aux2)));
        end
end
```

No exercício e), as novas percentagens de probabilidade para cada tamanho dos pacotes indicam que os pacotes terão no geral tamanhos menores. Foram realizados os seguintes cálculos para confirmar esta ideia:

```
avg_sizeCeD = 0.19*64 + 0.23*110 + 0.17*1518 + 0.00028*sum([65:109 111:1517]);
avg_sizeE = 0.25*64 + 0.17*110 + 0.11*1518 + 0.00032*sum([65:109 111:1517]);
```

Os pacotes dos exercícios c) e d) têm em média, aproximadamente, 617 bytes, enquanto, que os pacotes do exercício e) têm apenas 569 bytes.

Resultados APD sem BER:

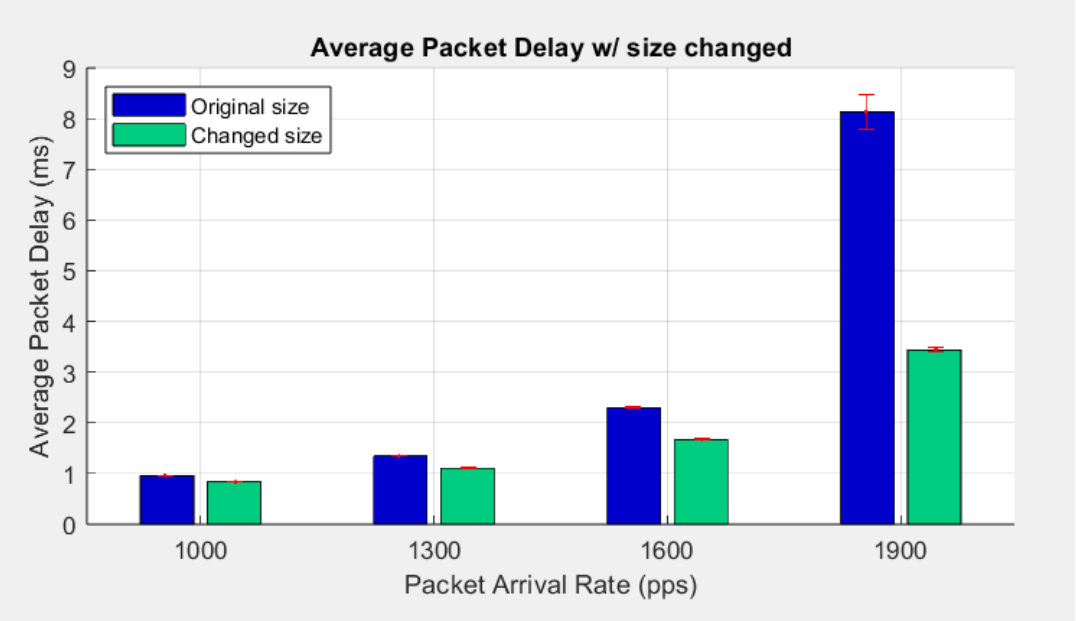

Figura 4: Resultado do exercício 1e APD sem BER

Conclusões:

Na figura acima temos presentes os valores de atraso médio dos pacotes no sistema para os pacotes com o tamanho inicial e para os pacotes já com as novas probabilidades de tamanhos (para fazer comparação do APD entre o exercício 'c' e 'e').

Como os pacotes são menores para o exercício e), o atraso médio será menor porque é realizada uma transmissão mais rápida dos mesmos e estes esperam menos tempo na fila.

Também é importante notar a ausência da subida abrupta do atraso para pacotes menores quando a taxa de chegada de pacotes por segundo é 1900. Como os pacotes são menores, a taxa de transferência é menor e esta não se encontra tão perto do limite de capacidade que são 10 Mbps. Isto é evidenciado pela Figura 5.

Resultados TT sem BER:

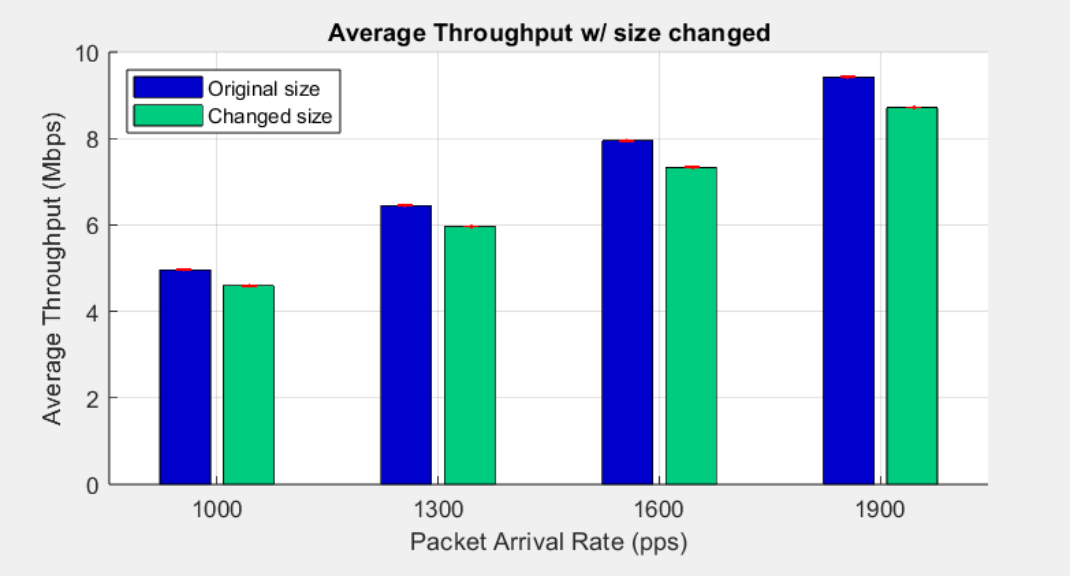

Figura 5: Resultado do exercício 1e TT sem BER

Conclusões:

A Taxa de transferência diminui um pouco para todas as taxas de chegada quando os tamanhos dos pacotes são mais pequenos, já que são transmitidos menos bytes durante a simulação.

Resultados APD com BER:

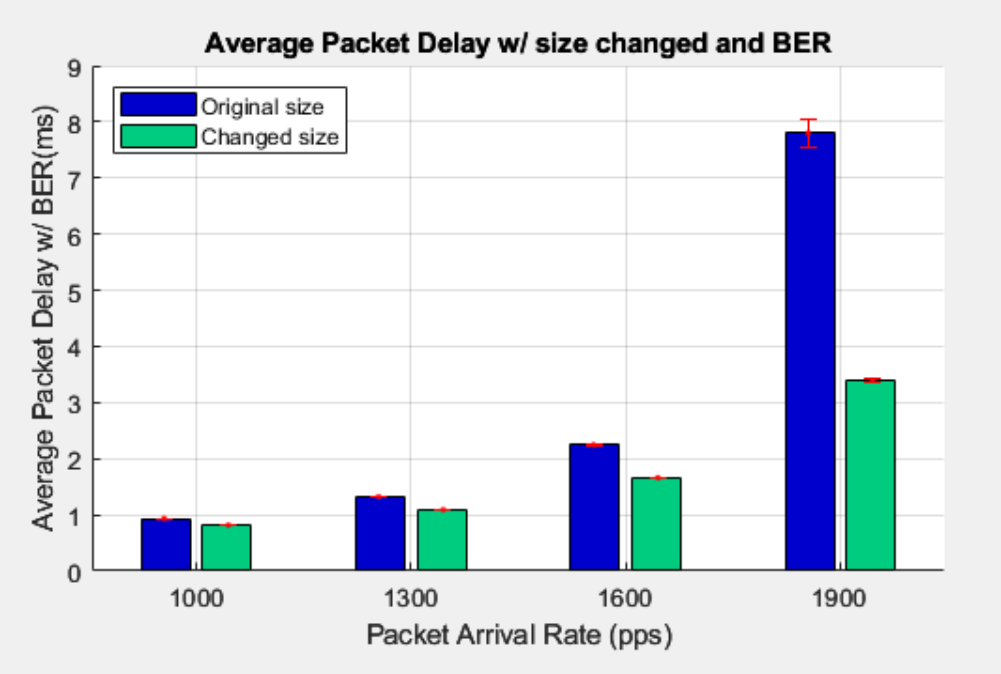

Figura 6: Resultado do exercício 1e APD com BER e pacotes mais pequenos

Conclusões:

A ideia principal é a mesma do sistema sem o BER: como os pacotes são menores para o exercício e), o atraso médio será menor porque é realizada uma transmissão mais rápida dos mesmos e estes esperam menos tempo na fila.

Introduzir o BER tem como consequência que pacotes maiores tenham maior probabilidade de serem descartados por terem erros.

Confirmamos sabendo que a probabilidade de não existirem erros no pacote pode ser dada pela fórmula binomial reduzida para $f(i) = (1-p)^n$, com i = 0. Assim para pacotes menores há maior probabilidade de não existir erros ('p' corresponde à Bit Error Rate e 'n' ao tamanho dos pacotes em bits). Consequentemente, haverá menos pacotes, ou pacotes mais pequenos.

Relativamente à Figura 4, quando incluímos o BER, os atrasos são geralmente um pouco menores, porque há menos pacotes a serem processados pelo sistema.

Resultados TT com BER:

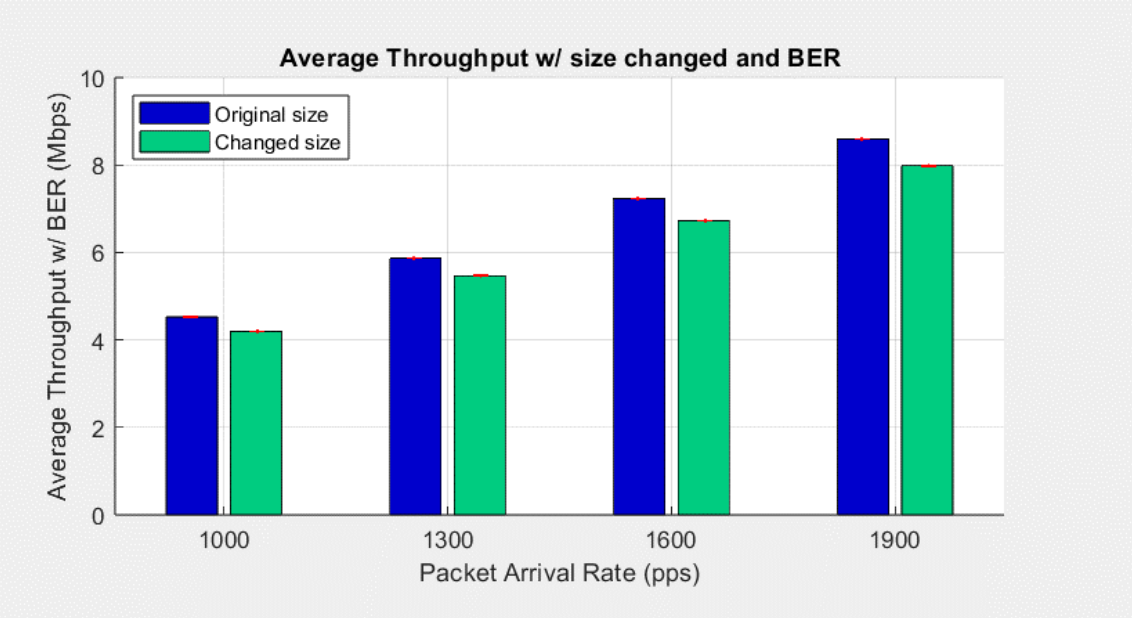

Figura 7: Resultado do exercício 1e TT com BER e pacotes mais pequenos

Conclusões:

Na última comparação podemos confirmar pela Figura 7 que a taxa de transferência com o BER incluído não varia muito dos tamanhos originais para os novos tamanhos dos pacotes.

A taxa de transferência desceu um pouco em relação aos valores quando o sistema não incluía BER. Isto deve-se a alguns pacotes serem descartados por terem erros, sobrando menos pacotes, ou pacotes mais pequenos, já que têm menor probabilidade de terem erros.

Concluímos que com menos pacotes e pacotes mais pequenos a taxa de transferência irá diminuir.

Tarefa 2

Exercício 2a

Para criar o Simulador 3 alterámos o Simulador 1 de forma ao sistema suportar, para além de um fluxo de pacotes de Data, n fluxos de pacotes de VoIP.

Por forma a adicionar o novo parâmetro de desempenho, atraso médio na fila de espera, é registado o valor do relógio em QUEUE(1,2) quando o pacote entra para a fila de espera. Quando este pacote sai da fila, e antes de lhe ser adicionado o tempo que demora na transmissão, é feita a incrementação dos atrasos dos pacotes na fila, subtraindo o relógio do momento pelo guardado no momento de chegada do pacote à fila.

No final a soma dos atrasos na fila de todos os pacotes é dividida pelo número de pacotes transmitidos e multiplicada por 1000 para ficar em milissegundos. Estes cálculos são realizados para pacotes de Data e VoIP separadamente.

É importante notar que depois de ser ultrapassado o número total de pacotes transmitidos na simulação o atraso do próximo pacote que sairia da fila já não é incluído para o cálculo do atraso médio dos pacotes na fila.

Código

```
function [PLd, PLv , APDd , APDv , AQDd, AQDv, MPDd, MPDv, TT] =
Simulator3(lambda,C,f,P,n)
% INPUT PARAMETERS:
%  lambda - packet rate (packets/sec)
%  C      - link bandwidth (Mbps)
%  f      - queue size (Bytes)
%  P      - number of packets (stopping criterium)
%  n      - number of VoIP packet flows
% OUTPUT PARAMETERS:
%  PLd   - packet loss (%) DATA
%  APDd  - average packet delay (milliseconds) DATA
%  AQDd  - average queue delay (milliseconds) DATA
%  MPDd  - maximum packet delay (milliseconds) DATA
%  PLv   - packet loss (%) VOIP
%  APDv  - average packet delay (milliseconds) VOIP
%  AQDv  - average queue delay (milliseconds) VOIP
%  MPDv  - maximum packet delay (milliseconds) VOIP
%  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 free; 1 - connection bysy
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
TOTALPACKETSV = 0;
LOSTPACKETSD= 0;        % No. of packets dropped due to buffer overflow
LOSTPACKETSV= 0;
TRANSMITTEDPACKETSD= 0; % No. of transmitted packets
TRANSMITTEDPACKETSV=0;
TRANSMITTEDBYTESD= 0;   % Sum of the Bytes of transmitted packets
TRANSMITTEDBYTESV=0;
DELAYSD= 0;             % Sum of the delays of transmitted packets
DELAYSV=0;
QUEUEDELAYSD = 0;
QUEUEDELAYSV = 0;
MAXDELAYD= 0;           % Maximum delay among all transmitted packets
MAXDELAYV=0;
% Initializing the simulation clock:
Clock= 0;
% Initializing the List of Events with the first ARRIVAL:
tmp= Clock + exprnd(1/lambda);
Event_List = [ARRIVAL, tmp, GeneratePacketSize(), tmp, DATA];

for i = 1:n
    tmp = unifrnd(0,0.02);
    Event_List = [Event_List; ARRIVAL, tmp, randi([110 130]), tmp, VOIP];
end
```

```
%Similation loop:
while (TRANSMITTEDPACKETSD+TRANSMITTEDPACKETSV) < P              % Stopping
criterium
    Event_List= sortrows(Event_List,2);  % Order EventList by time
    Event= Event_List(1,1);              % Get first event and
    Clock= Event_List(1,2);              %   and
    Packet_Size= Event_List(1,3);        %   associated
    Arrival_Instant= Event_List(1,4);    %   parameters.
    Packet_type = Event_List(1,5);
    Event_List(1,:)= [];                 % Eliminate first event

    if Event == ARRIVAL         % If first event is an ARRIVAL
        if Packet_type == DATA
            TOTALPACKETSD= TOTALPACKETSD+1;
            tmp= Clock + exprnd(1/lambda);
            Event_List = [Event_List; ARRIVAL, tmp, GeneratePacketSize(), tmp,
DATA];
            if STATE==0
                STATE= 1;
                Event_List = [Event_List; DEPARTURE, Clock + 8*Packet_Size/(C*10^6),
Packet_Size, Clock, DATA];
            else
                if QUEUEOCCUPATION + Packet_Size <= f
                    QUEUE= [QUEUE;Packet_Size , Clock, DATA];
                    QUEUEOCCUPATION= QUEUEOCCUPATION + Packet_Size;
                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*Packet_Size/(C*10^6),
Packet_Size, Clock, VOIP];
            else
                if QUEUEOCCUPATION + Packet_Size <= f
                    QUEUE= [QUEUE;Packet_Size , Clock, VOIP];
                    QUEUEOCCUPATION= QUEUEOCCUPATION + Packet_Size;
                else
                    LOSTPACKETSV= LOSTPACKETSV + 1;
                end
            end
        end
```

```
    else                        % If first event is a DEPARTURE
            if Packet_type == DATA
                TRANSMITTEDBYTESD= TRANSMITTEDBYTESD + Packet_Size;
                DELAYSD= DELAYSD + (Clock - Arrival_Instant);
                if Clock - Arrival_Instant > MAXDELAYD
                    MAXDELAYD= Clock - Arrival_Instant;
                end
                TRANSMITTEDPACKETSD= TRANSMITTEDPACKETSD + 1;
            else
                TRANSMITTEDBYTESV= TRANSMITTEDBYTESV + Packet_Size;
                DELAYSV= DELAYSV + (Clock - Arrival_Instant);
                if Clock - Arrival_Instant > MAXDELAYV
                    MAXDELAYV= Clock - Arrival_Instant;
                end
                TRANSMITTEDPACKETSV= TRANSMITTEDPACKETSV + 1;
            end
            if (TRANSMITTEDPACKETSD+TRANSMITTEDPACKETSV) < P
                if QUEUEOCCUPATION > 0
                    % quando o pacote é adicionado como departure à lista de
                    % eventos já lhe é adicionado o tempo que vai demorar na ligação
                    Event_List = [Event_List; DEPARTURE, Clock + 8*QUEUE(1,1)/(C*10^6),
    QUEUE(1,1), QUEUE(1,2), QUEUE(1,3)];
                    QUEUEOCCUPATION= QUEUEOCCUPATION - QUEUE(1,1);
                    if QUEUE(1,3) == DATA
                        QUEUEDELAYSD = QUEUEDELAYSD + (Clock - QUEUE(1,2));
                    elseif QUEUE(1,3) == VOIP
                        QUEUEDELAYSV = QUEUEDELAYSV + (Clock - QUEUE(1,2));
                    end
                    QUEUE(1,:)= [];% Para calcular o qD não precisamos do tempo do link
                else
                    STATE= 0;
                end
            end
        end
    end
    %Performance parameters determination:
    PLd= 100*LOSTPACKETSD/TOTALPACKETSD;      % in %
    PLv = 100*LOSTPACKETSV/TOTALPACKETSV;                           % in %
    APDd = 1000*DELAYSD/TRANSMITTEDPACKETSD;                        % in milliseconds
    APDv = 1000*DELAYSV/TRANSMITTEDPACKETSV;                        % in milliseconds
    MPDd = 1000*MAXDELAYD;                                          % in milliseconds
    MPDv = 1000*MAXDELAYV;                                          % in milliseconds
    AQDd = 1000 * QUEUEDELAYSD / TRANSMITTEDPACKETSD;
    AQDv = 1000 * QUEUEDELAYSV / TRANSMITTEDPACKETSV;
    TT = 10^(-6)*(TRANSMITTEDBYTESD+TRANSMITTEDBYTESV)*8/Clock;  % in Mbps
    end
    function out= GeneratePacketSize()
        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
```

Resultados:

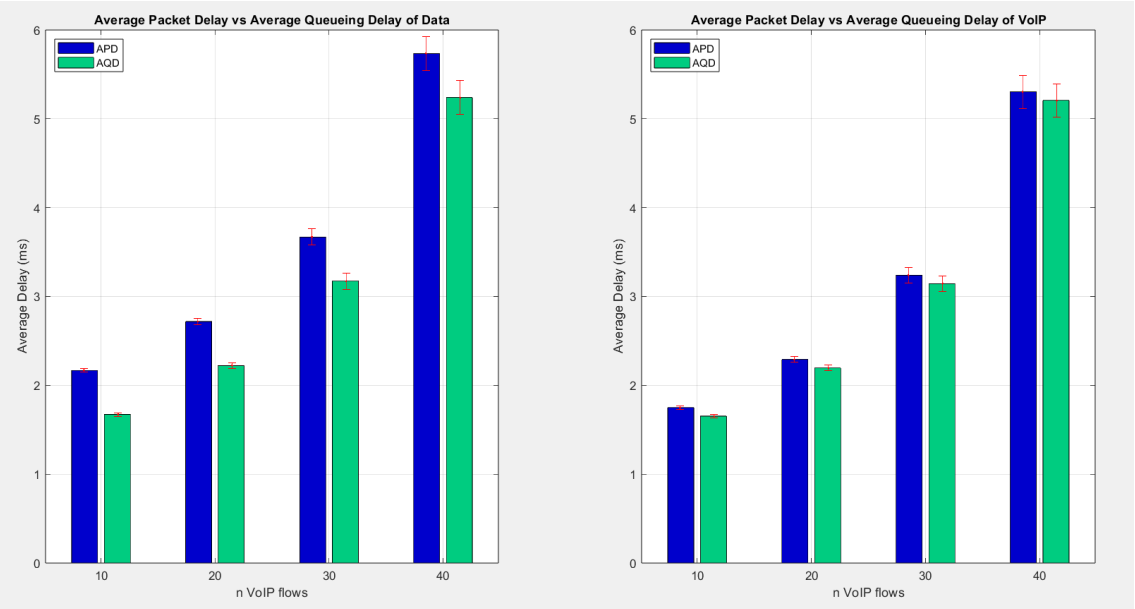

Figura 8: Resultado do exercício 2a APD vs AQD

Conclusões:

Com o aumento dos fluxos de pacotes VoIP há um aumento geral dos atrasos dos pacotes, tanto na fila como no sistema, como é possível confirmar pela

Figura 8. Isto deve-se ao facto de chegarem mais pacotes à fila de espera única, fazendo esta ficar mais congestionada e, consequentemente, os pacotes terem de esperar mais tempo para serem transferidos.

Os aumentos nos atrasos levam a intervalos de confiança maiores, uma vez que há um aumento da ampliação da variação e incerteza.

Se calcularmos as diferenças entre valores dos atrasos no sistema e atrasos na fila de espera verificamos que, para qualquer fluxo, este valor dá praticamente igual (Tanto para Data como para VoIP). Isto deve-se ao facto dos tempos que demoram as transferências permanecerem iguais, porque onde o atraso aumenta é na fila de espera.

Exercício 2b

Neste exercício foi pedido que fizéssemos o mesmo que na alínea anterior, no entanto, agora pretende-se acrescentar uma maior prioridade para o serviço VoIP.

Código:

Foi adicionado valor 1 ao VoIP e 0 à Data, com o propósito de que quando se fez a ordenação se pudesse dar mais prioridade aos pacotes VoIP.

```
%Packet type
DATA = 0;
VOIP = 1;

(...)

else                        % If first event is a DEPARTURE
        if Packet_type == DATA
            TRANSMITTEDBYTESD= TRANSMITTEDBYTESD + Packet_Size;
            DELAYSD= DELAYSD + (Clock - Arrival_Instant);
            if Clock - Arrival_Instant > MAXDELAYD
                MAXDELAYD= Clock - Arrival_Instant;
            end
            TRANSMITTEDPACKETSD= TRANSMITTEDPACKETSD + 1;
        else
            TRANSMITTEDBYTESV= TRANSMITTEDBYTESV + Packet_Size;
            DELAYSV= DELAYSV + (Clock - Arrival_Instant);
            if Clock - Arrival_Instant > MAXDELAYV
                MAXDELAYV= Clock - Arrival_Instant;
            end
            TRANSMITTEDPACKETSV= TRANSMITTEDPACKETSV + 1;
        end
        if (TRANSMITTEDPACKETSD+TRANSMITTEDPACKETSV) < P
            if QUEUEOCCUPATION > 0
                % VoIP packets are given higher
                % priority than data packets in the queue
                QUEUE = sortrows(QUEUE,3,"descend");
                Event_List = [Event_List; DEPARTURE, Clock + 8*QUEUE(1,1)/(C*10^6),
QUEUE(1,1), QUEUE(1,2), QUEUE(1,3)];
                QUEUEOCCUPATION= QUEUEOCCUPATION - QUEUE(1,1);

                if QUEUE(1,3) == DATA
                    QUEUEDELAYSD = QUEUEDELAYSD + (Clock - QUEUE(1,2));
                elseif QUEUE(1,3) == VOIP
                    QUEUEDELAYSV = QUEUEDELAYSV + (Clock - QUEUE(1,2));
                end

                QUEUE(1,:)= [];
            else
                STATE= 0;
            end
        end
    end
```

Resultado:

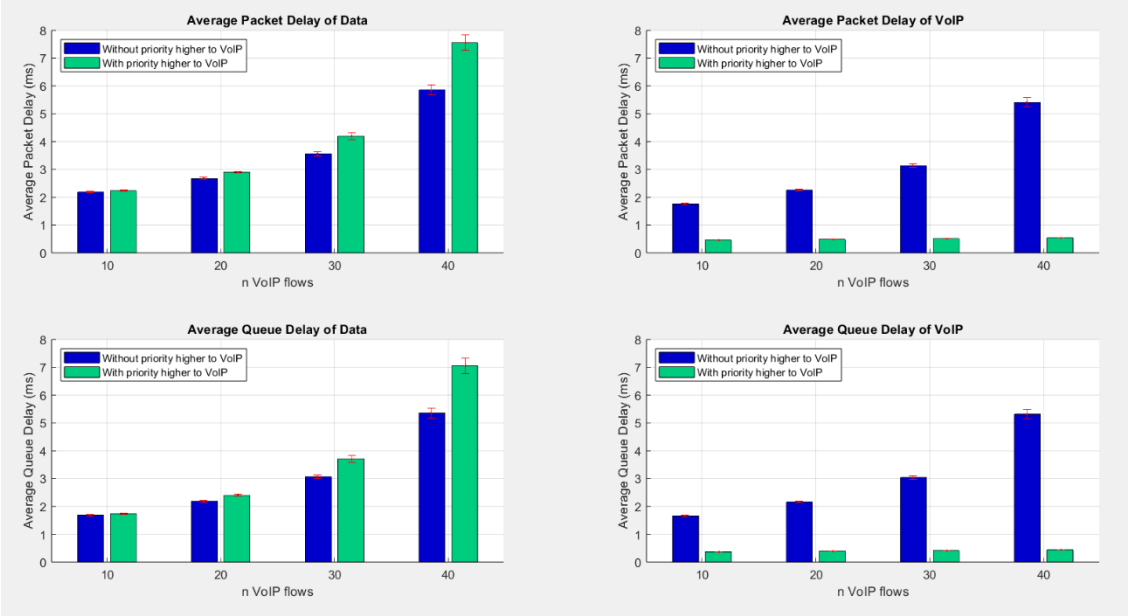

Figura 9: Resultado do exercício 2b

Explicação:

Tal como esperado, quanto mais fluxos de pacotes VoIP maior o atraso médio de ambos os tipos de pacote. Comparando os resultados com prioridade com os obtidos sem prioridade, onde apenas é valorizada a ordem de chegada, nos gráficos de VoIP é percetível uma diminuição muito significativa e uma obtenção de valores quase uniformes do atraso médio na fila dos pacotes de tipo VoIP. A razão é autoexplicativa, dado a maior prioridade dos pacotes VoIP, passando estes à frente dos pacotes de Data na fila.

Para os gráficos direcionados a pacotes Data, o atraso dos pacotes quando há maior prioridade para os pacotes VoIP, aumentou. Os pacotes de Data têm prioridade mais baixa e por isso demoram mais tempo a ser servidos. É importante notar que a transmissão de um pacote não é interrompida pela chegada de um pacote de maior prioridade (non-preemptive).

Exercício 2c

Uma possibilidade para diferenciar o tratamento dos pacotes de diferentes fluxos é atribuir prioridades aos fluxos.

Para este exercício utilizou-se o sistema M/G/1 com 2 prioridades em que 1 corresponde à prioridade mais alta (VoIP) e 0 à mais baixa (Data).

Para a realização deste exercício consideraram-se 2 taxas de chegada uma para cada tipo de pacotes e ainda a média (ou 1º momento) e 2º momento do tempo de transmissão dos pacotes: $E[S_k]$ e $E[S_k^2]$, sendo 'k' o agregado de fluxos de pacotes da prioridade k, 1 ≤ k ≤ n.

Para os cálculos do 1º e 2º momento dos pacotes tipo Data, utilizámos um raciocínio semelhante ao Exercício 1b, mas desta vez apenas para uma capacidade de 10 Mbps. Para obter tamanho e tempo de intervalo entre envio de pacotes médios somamos todas as possibilidades de valores para os mesmos e dividimos pela quantidade total dos valores possíveis.

Com o valor médio do tamanho dos pacotes chegamos aos 1º e 2º momentos através do inverso do μ e do μ^2, respetivamente. Já com o tempo médio de intervalo entre envio de pacotes obtemos a taxa de pacotes por segundo de cada fluxo VoIP. Este último valor vai depois ser multiplicado por n fluxos de VoIP.

Por fim, o cálculo que realizamos para descobrir o atraso médio por pacote na fila de espera correspondente aos pacotes da prioridade k é dado por:

$$W_{Qk} = \begin{cases} \dfrac{1}{2(1-\rho_1)} \times \sum_{i=1}^{n} (\lambda_i E[S_i^2]) \,, k = 1 \\ \dfrac{1}{2(1-\rho_1-\cdots-\rho_{k-1})(1-\rho_1-\cdots-\rho_k)} \times \sum_{i=1}^{n} (\lambda_i E[S_i^2]) \,, k > 1 \end{cases}$$

Excerto de código:

```
n = [10 20 30 40]; % Number of VoIP packet flows
C = 10;
C = C * 10^6; % Convert capacity to bps

perc_left = (1 - (0.19 + 0.23 + 0.17)) / ((109 - 64) + (1517 - 110));
x = 64:1518;
S = zeros(4,length(x));
S2 = zeros(4,length(x));

for i = 1:length(x)
    if x(i) == 64
        s = (x(i) * 8) / C;
        S(j,i) = 0.19 * s;
        S2(j,i) = 0.19 * s^2;
    elseif x(i) == 110
        s = (x(i) * 8) / C;
        S(j,i) = 0.23 * s;
        S2(j,i) = 0.23 * s^2;
    elseif x(i) == 1518
        s = (x(i) * 8) / C;
        S(j,i) = 0.17 * s;
        S2(j,i) = 0.17 * s^2;
    else
        s = (x(i) * 8) / C;
        S(j,i) = perc_left * s;
        S2(j,i) = perc_left * s^2;
    end
end
ES = sum(S(j,:));
ES2 = sum(S2(j,:));

% tamanho dos pacotes voip é randi([110 130])
% taxa de chegada do tipo voip é 1 a ([16 24])
size_voip = sum((110:130)/21); % 120 bytes
lambda_voip = 1/(sum((16:24)/9) * 10^-3); % 0.02 seg = 20 ms % lambda_voip = 50 pps
u_voip = 10e6/(8*size_voip); % 9843 pps
ES_v = 1/u_voip; % Service time, S
ES2_v = 1/(u_voip^2); % S^2
% A seguir calculamos o M/G/1 with priorities

APDv = zeros(4, 1); % Initialize an array for VoIP APD
APDd = zeros(4, 1); % Initialize an array for VoIP APD
fprintf('Valores teóricos\n');
% Percorremos os fluxos todos
for i = 1:4
    % pk = lambda_k * E[S_k]
    lambda_nflows_voip = lambda_voip * n(i);
    pA = (lambda_nflows_voip) * ES_v; % 0.1920
    pB = lambda * ES; % para o type data = 0.7440
    % Verifica-se a condição de validade pA+pB < 1 -> 0.9360
    wA = (((lambda_nflows_voip *ES2_v+lambda*ES2)/(2*(1-pA)))+ES_v)*1e3;% seg
    wB = (((lambda_nflows_voip *ES2_v+lambda*ES2)/(2*(1-pA))*(1-pA-pB))+ES_v)*1e3;
    APDv(i) = wA;
    APDd(i) = wB;
end
```

Resultado:

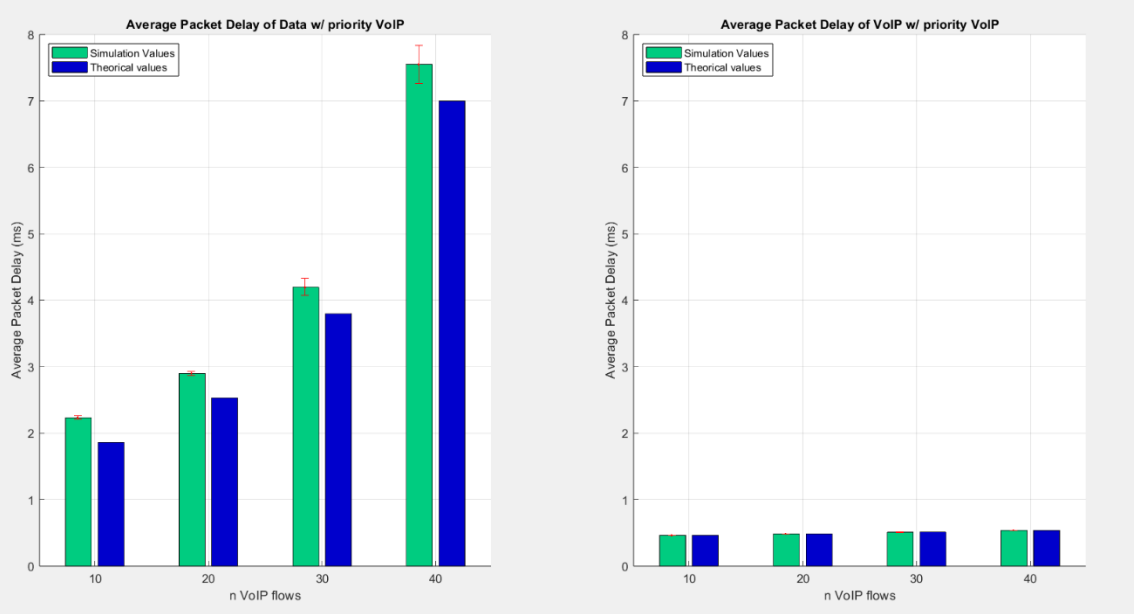

Figura 10: Resultado do exercício 2c

Explicação:

Ao analisar os resultados calculamos a condição de validade em que a soma de todos os ρ's de cada prioridade tem de ser menor que 1 e verificámos que o resultado foi 0.9360 logo dentro do esperado.

Os resultados teóricos aproximam-se dos simulados em relação aos pacotes do tipo VoIP. Contudo, nos pacotes do tipo Data observamos que os valores teóricos são menores do que os da experiência.

Contribuição dos autores

Tiago Alves – 50 %
Rafael Amorim – 50 %