Universidade de Aveiro

Modelação e Desempenho de Redes e Seviços

Mini-Project nr.1

Guilherme Duarte (107766), Guilherme Andrade (107696)

Departamento de Eletrónica, Telecomunicações e Informática

October 21, 2024

Conteúdo

1.1 Exercise 1.a) 4
1.1.1 Results 4
1.1.2 Conclusion 5

1.2 Exercise 1.b) 6
1.2.1 Results 6
1.2.2 Conclusion 7

1.3 Exercise 1.c) 8
1.3.1 Code 8
1.3.2 Results and Conclusions 11

2.1 Approach 12

2.1.1 Justifying the changes introduced 17

2.2 Exercise 2.b) 18
2.2.1 Results 18
2.2.2 Conclusions 19

2.3 Exercise 2.c) 20
2.3.1 Results 20
2.3.2 Conclusions 21

2.4 Exercise 2.d) 22
2.4.1 Results 22
2.4.2 Conclusions 23

2.5 Exercise 2.e) 24
2.5.1 Results 24
2.5.2 Conclusions 25

2.6 Exercise 2.f) 26
2.6.1 Code 26
2.6.2 Results 29
2.6.3 Conclusions 30

3.1 Exercise 3.a) 31
3.1.1 Results 31
3.1.2 Conclusions 33

3.2 Exercise 3.b) 34
3.2.1 Results 34
3.2.2 Conclusions 36

3.3 Exercise 3.c) 38
3.3.1 Code 38

3.4 Exercise 3.d) 42
3.4.1 Results 42
3.4.2 Conclusions 44

3.5 Exercise 3.e) 45
3.5.1 Results 45
3.5.2 Conclusions 47

List of Figures

Figure 1:The average packet loss results and the average packet delay results, b = 1e-6 .4
Figure 2: The average packet loss results and the average packet delay results, b = 1e-4 6
Figure 3: The average packet loss results for VoIP and Data, b = 1e-5 18
Figure 4: The average Data packet delay and the average VoIP packet delay results, b = 1e-5.....20
Figure 5: The maximum Data packet delay and the maximum VoIP packet delay results, b = 1e-5.....22
Figura 6: Simulated Transmitted Throughput results, b = 1e-5.....24
Figure 7: Theoretical Transmitted Throughput results with BER, b = 1e-5.....29
Figure 8: The average packet loss results for VoIP and Data 31
Figure 9: The average packet delay results for Data and VoIP 32
Figure 10: The average packet loss results for Data and VoIP 34
Figure 11: The average packet delay results for Data and VoIP 35
Figure 12: The average packet loss results for Data and VoIP 42
Figure 13: The average packet delay results for Data and VoIP 43
Figure 14: The average packet loss results for Data and VoIP 45
Figura 15: The average packet delay results for Data and VoIP 46

Chapter 1

Task 1

1.1 Exercise 1.a)

1.1.1 Results

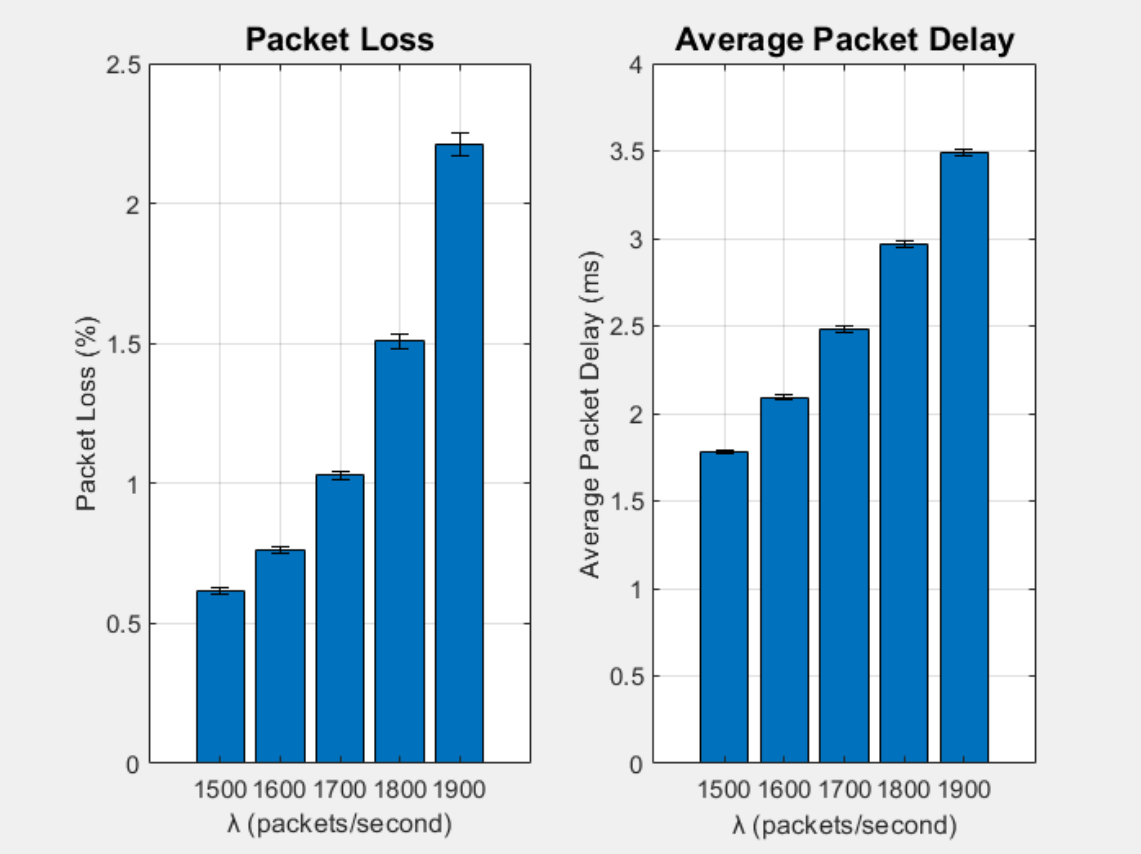

Figure 1:The average packet loss results and the average packet delay results, b = 1e-6

1.1.2 Conclusion

Packet Loss:

The increase in packet loss as the arrival rate (λ) increases is an expected behavior in queuing systems. Initially, when the arrival rate is low (e.g., λ = 1500 pps), the system can process packets efficiently, leading to minimal packet loss (around 0.5%).

As the arrival rate increases, especially after λ = 1700 pps, packet loss starts to rise more sharply. This happens because the queue reaches its capacity, and packets are dropped due to buffer overflow. At λ = 1900 pps, the system becomes highly overloaded, resulting in a packet loss of approximately 2.5%. This indicates that the system is operating beyond its efficient capacity.

Average Packet Delay:

Average packet delay also increases with higher arrival rates. This is because, as more packets arrive, they spend more time in the queue waiting to be processed.

The delay is relatively low at lower arrival rates (around 1.5 ms for λ = 1500 pps). However, as λ reaches 1900 pps, the average delay exceeds 3.5 ms, indicating that packets are waiting longer in the queue due to system overload. This suggests that, as congestion builds up, packets take more time to be transmitted.

Conclusions:

The results clearly show a trade-off between increasing the packet arrival rate and system performance. While the system can handle higher rates, this comes at the cost of increased packet delay and packet loss. For systems where minimizing delay and packet loss is critical, it is important to maintain the arrival rate within a more controlled range, around 1500-1600 pps.

These results are valuable for designing network capacity and communication systems. They demonstrate that as network traffic increases, performance can degrade quickly if the system capacity is not adjusted to match demand. This study helps determine the optimal operating limit to avoid congestion and ensure acceptable quality of service.

1.2 Exercise 1.b)

1.2.1 Results

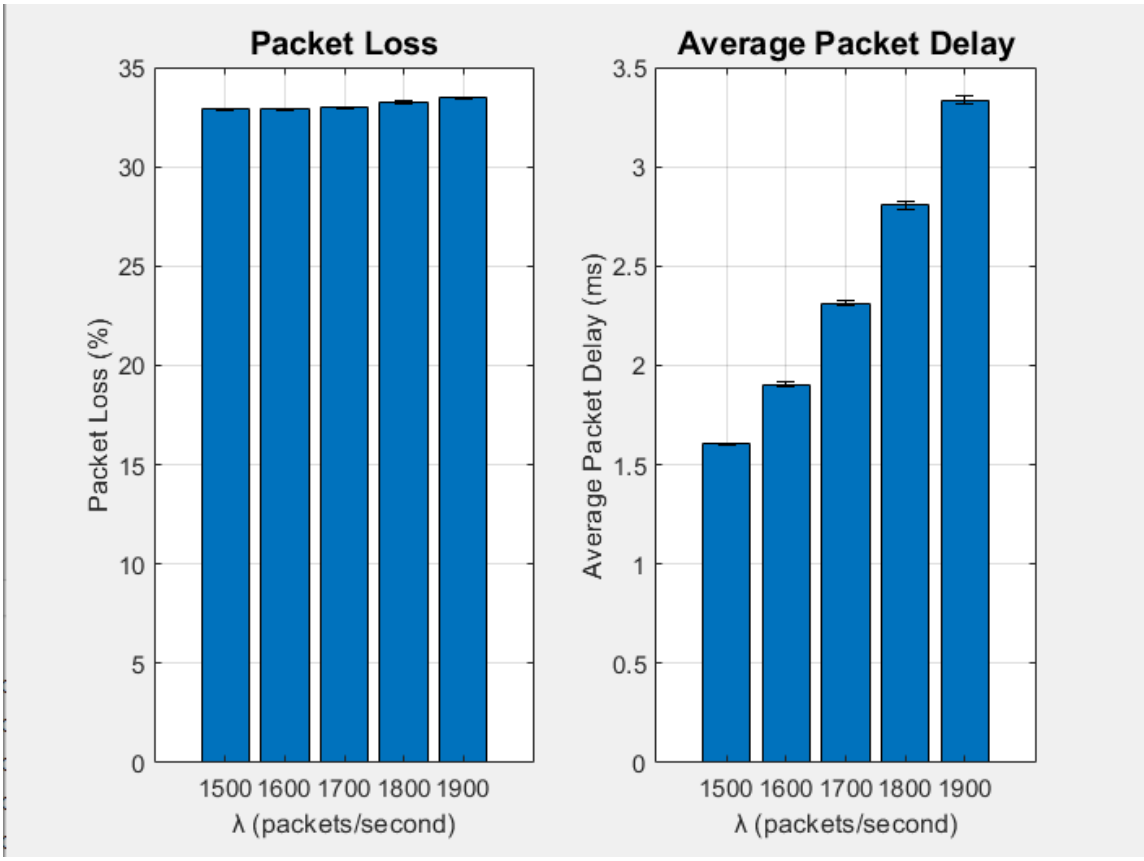

Figure 2: The average packet loss results and the average packet delay results, b = 1e-4

1.2.2 Conclusion

Packet Loss:

The results clearly show a consistent packet loss rate of approximately 30% across all arrival rates (λ = 1500 to 1900 pps). This high level of packet loss can be attributed primarily to the high bit error rate (BER) of 10^-4.

As the packet transmission faces a high probability of bit errors, many packets are dropped due to corruption in transmission, regardless of the queue occupancy or congestion. This explains why the packet loss does not vary significantly as the arrival rate (λ) increases. The dominant factor here is the **BER** rather than network congestion.

Average Packet Delay:

The **average packet delay** remains very similar to the results obtained in **Experiment 1.a**, where **BER = 10^-6**. The reason for this behavior is that packets experiencing bit errors are discarded early in the process, meaning they do not contribute to congestion within the queue or the transmission process.

This indicates that the Average Packet Delay (APD) is primarily influenced by queue congestion and arrival rate (λ), not by bit errors. The discarded packets due to bit errors do not have an impact on the queue, which keeps the APD stable across different scenarios of BER.

Conclusions:

Packet Loss (PL) significantly increases with the rise in **bit error rate (BER)**. This is expected because a higher probability of bit errors during transmission means more packets are being discarded due to corruption. As a result, the packet loss rates are much higher for all packet arrival rates (λ).

Average Packet Delay (APD), on the other hand, remains consistent regardless of the bit error rate. This is because the bit errors cause packets to be dropped, preventing them from entering the queue and contributing to congestion. As a result, only the successfully transmitted packets contribute to the delay measurement, keeping the APD like the results from **Experiment 1.a**.

1.3 Exercise 1.c)

1.3.1 Code

```
% Define the probabilities for predefined packet sizes (64, 110, 1518)
packet_probs = [0.19, 0.23, 0.17];
packet_sizes = [64, 110, 1518];

% Equal probability for Data Packets with sizes between 65-109 and 111-1517 bytes
prob_left = (1 - sum(packet_probs)) / ((109 - 65 + 1) + (1517 - 111 + 1));

% Define the possible sizes for the remaining packets (65-109, 111-1517)
remaining_sizes = [65:109, 111:1517];
remaining_prob = prob_left * ones(1, length(remaining_sizes)); % Equal probability for these packet sizes

% Combine all packet sizes and their corresponding probabilities
all_sizes = [packet_sizes, remaining_sizes];
all_probs = [packet_probs, remaining_prob];

% Define BER values to test
ber_values = [1e-6, 1e-4];

% Initialize the packet loss array
ploss_ber = zeros(1, length(ber_values));

% Loop through each BER value to compute the theoretical packet loss
for i = 1:length(ber_values)
    ber = ber_values(i);

    % Compute packet loss for each size
    packet_loss_prob = 1 - (1 - ber).^(all_sizes * 8); % Probability of at least one bit error

    % Compute weighted sum of packet loss probabilities
    ploss_ber(i) = sum(packet_loss_prob .* all_probs);
end

% Convert the results to percentages
ploss_ber = ploss_ber * 100;

% Display the results
fprintf('Considering bit error rate as the sole cause of packet loss, the theoretical packet loss (%%)\n');
for i = 1:length(ber_values)
    fprintf('For BER = %.1e: %.4f %%\n', ber_values(i), ploss_ber(i));
end
```

Packet Size Probabilities Setup:

```
packet_probs = [0.19, 0.23, 0.17];
packet_sizes = [64, 110, 1518];
```

These are the predefined probabilities for specific packet sizes (64, 110, and 1518 bytes) as given in the problem. The values indicate the likelihood of each of these packet sizes appearing in the data stream.

Remaining Packet Sizes Setup:

```
prob_left = (1 - sum(packet_probs)) / ((109 - 65 + 1) + (1517 - 111 + 1));
remaining_sizes = [65:109, 111:1517];
remaining_prob = prob_left * ones(1, length(remaining_sizes));
```

prob_left calculates the equal probability assigned to packet sizes that are **not** 64, 110, or 1518 bytes. This ensures that the total probability sums to 1.
remaining_sizes lists all other possible packet sizes (65-109 and 111-1517 bytes).
remaining_prob is the probability assigned to these other packet sizes.

Combining All Packet Sizes and Their Probabilities:

```
all_sizes = [packet_sizes, remaining_sizes];
all_probs = [packet_probs, remaining_prob];
```

all_sizes combines both the predefined and remaining packet sizes into a single array.
all_probs combines the corresponding probabilities of the predefined and remaining packet sizes.

Bit Error Rate (BER) Setup:

```
ber_values = [1e-6, 1e-4];
ploss_ber = zeros(1, length(ber_values));
```

ber_values defines the bit error rates you want to test (e.g., 10^{-6} and 10^{-4}).
ploss_ber initializes an array to store the calculated packet loss probabilities for each BER.

Computing Packet Loss for Each BER:

```
for i = 1:length(ber_values)
    ber = ber_values(i);

    % Compute packet loss for each size
    packet_loss_prob = 1 - (1 - ber).^(all_sizes * 8); % Probability of at least one bit error

    % Compute weighted sum of packet loss probabilities
    ploss_ber(i) = sum(packet_loss_prob .* all_probs);
end
```

Looping over BER values:

For each BER, the code computes the probability of packet loss for all packet sizes.

Packet Loss Calculation:

The probability of a packet being lost due to bit errors is calculated with the formula: P(loss)=1−(1−BER) ^ size×8

This formula gives the probability that **at least one bit** in the packet is corrupted. The packet size is multiplied by 8 to convert the size from bytes to bits.

Weighted Sum:

The loss probability for each packet size is multiplied by the probability of that packet size occurring (all_probs), and then summed to get the total packet loss probability for the given BER.

Convert to Percentage and Display:

```
ploss_ber = ploss_ber * 100;
fprintf('Considering bit error rate as the sole cause of packet loss, the theoretical packet loss (%%)\n');
for i = 1:length(ber_values)
   fprintf('For BER = %.1e: %.4f %%\n', ber_values(i), ploss_ber(i));
end
```

The final loss probabilities are converted to percentages multiplying by 100.
The results are displayed using fprintf for each tested BER value.

1.3.2 Results and Conclusions

The program's ouput was the following: "Considering that the bit error rate is the only factor that can cause Packet to be lost, the theoretical packet loss (%) for a bit error rate of 10^{-6} is: 0.4937 % and for a bit error rate of 10^{-4} is: 32.8278 %".

In experiment "a" (with a lower BER, such as b=10^{-6}):

Main cause of packet loss: With a lower bit error rate, the primary factor causing packet loss is the queue size. When the queue fills up due to the high packet arrival rate (λ), packets are dropped, leading to a gradual increase in packet loss as the queue saturates.

Difference between simulated and theoretical values: In the case of a very low BER, packet loss is more related to queue congestion than to bit errors. This explains the discrepancy between the simulated and theoretical values-the simulation accounts for queue congestion, while the theoretical calculation only considers BER.

In experiment "b" (with a lower BER, such as b=10^{-4}):

Main cause of packet loss: Here, the bit error rate is high enough that **most packets are lost due to bit errors** before being affected by queue congestion. Packet loss is dominated by bit errors rather than by queue size or arrival rates.

Similarity between simulated and theoretical values: With b=10^{-4}, the behavior is dominated by the BER, and since the theoretical calculation considers only the BER, the simulated and theoretical results tend to match closely. The queue may have space for more packets, but many packets are already being dropped due to bit errors, leading to a high packet loss rate similar to the theoretical value.

Conclusions:

In case "a": The **queue size** and the packet arrival rate (λ) play a significant role in packet loss. Packet loss is mainly driven by queue congestion, which is why the simulated value differs from the theoretical one (which only accounts for BER).

In case "b": Packet loss is primarily caused by **the high BER**, not by queue congestion. Since BER is the dominant factor, the simulated and theoretical values align closely.

Chapter 2

Task 2

2.1 Approach

In Sim3A we need to add the bit error rate function that is implemented on Sim2 but apply it on Sim3. Therefore we only change the code that processes the Departure packets as the code below shows.

```
function [PLD, PLV, APDD, APDV, MPDD, MPDV, TT] = Sim3A(lambda, C, f, P, n, b)
% INPUT PARAMETERS:
% lambda - packet rate (packets/sec)
% C      - link bandwidth (Mbps)
% f      - queue size (Bytes)
% P      - number of packets (stopping criterion)
% n      - number of VoIP packet flows
% b      - bit error rate (BER)

% OUTPUT PARAMETERS:
% PLD  - packet loss of data packets (%)
% PLV  - packet loss of VoIP packets (%)
% APDD - average packet delay for data packets (milliseconds)
% APDV - average packet delay for VoIP packets (milliseconds)
% MPDD - maximum packet delay for data packets (milliseconds)
% MPDV - maximum packet delay for VoIP packets (milliseconds)
% TT   - transmitted throughput (Mbps)

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

% Packet Types:
Dados = 0;         % Data packets
Voip = 1;          % VoIP packets

% State variables:
STATE = 0;         % 0 - connection is free, 1 - connection is occupied
QUEUEOCCUPATION = 0; % Occupation of the queue (in Bytes)
QUEUE = [];        % Queue to store packet size and arrival time

% Statistical Counters:
TOTALPACKETSD = 0;   % Total data packets arrived
TOTALPACKETSV = 0;   % Total VoIP packets arrived
LOSTPACKETSD = 0;    % Data packets lost due to buffer overflow
LOSTPACKETSV = 0;    % VoIP packets lost due to buffer overflow
TRANSPACKETSD = 0;   % Data packets transmitted
TRANSPACKETSV = 0;   % VoIP packets transmitted
TRANSBYTESD = 0;     % Total Bytes of transmitted data packets
TRANSBYTESV = 0;     % Total Bytes of transmitted VoIP packets
DELAYSD = 0;         % Total delay of transmitted data packets
DELAYSV = 0;         % Total delay of transmitted VoIP packets
MAXDELAYD = 0;       % Maximum delay for data packets
MAXDELAYV = 0;       % Maximum delay for VoIP packets

% Initialize simulation clock:
Clock = 0;

% Initialize event list with the first ARRIVAL event:
tmp = Clock + exprnd(1/lambda);
EventList = [ARRIVAL, tmp, GeneratePacketSize(), tmp, Dados];

% Add VoIP packet arrival events:
for i = 1:n
    tmp = unifrnd(0, 0.02); % Uniform distribution between 0 ms and 20 ms for VoIP
    EventList = [EventList; ARRIVAL, tmp, randi([110, 130]), tmp, Voip];
end
```

```
while (TRANSPACKETSD + TRANSPACKETSV) < P
    EventList = sortrows(EventList, 2);  % Order EventList by time
    Event = EventList(1, 1);              % Get first event
    Clock = EventList(1, 2);              % Time of the event
    PacketSize = EventList(1, 3);         % Packet size
    ArrInstant = EventList(1, 4);         % Arrival time
    PacketType = EventList(1, 5);         % Packet type (Dados/Voip)
    EventList(1, :) = [];                 % Remove the processed event

    switch Event
        case ARRIVAL
            if PacketType == Dados
                TOTALPACKETSD = TOTALPACKETSD + 1;
                tmp = Clock + exprnd(1/lambda);
                EventList = [EventList; ARRIVAL, tmp, GeneratePacketSize(), tmp, Dados];
                if STATE == 0
                    STATE = 1;
                    EventList = [EventList; DEPARTURE, Clock + 8*PacketSize/(C*10^6), PacketSize, Clock, Dados];
                else
                    if QUEUEOCCUPATION + PacketSize <= f
                        QUEUE = [QUEUE; PacketSize, Clock, Dados];
                        QUEUEOCCUPATION = QUEUEOCCUPATION + PacketSize;
                    else
                        LOSTPACKETSD = LOSTPACKETSD + 1;
                    end
                end
            else
                TOTALPACKETSV = TOTALPACKETSV + 1;
                tmp = Clock + unifrnd(0.016, 0.024);  % VoIP inter-arrival time
                EventList = [EventList; ARRIVAL, tmp, randi([110, 130]), tmp, Voip];
                if STATE == 0
                    STATE = 1;
                    EventList = [EventList; DEPARTURE, Clock + 8*PacketSize/(C*10^6), PacketSize, Clock, Voip];
                else
                    if QUEUEOCCUPATION + PacketSize <= f
                        QUEUE = [QUEUE; PacketSize, Clock, Voip];
                        QUEUEOCCUPATION = QUEUEOCCUPATION + PacketSize;
                    else
                        LOSTPACKETSV = LOSTPACKETSV + 1;
                    end
                end
            end

        case DEPARTURE
            if rand() < (1 - b)^(PacketSize*8) % Simulate packet loss due to bit errors
                if PacketType == Dados
                    TRANSBYTESD = TRANSBYTESD + PacketSize;
                    DELAYSD = DELAYSD + (Clock - ArrInstant);
                    if (Clock - ArrInstant) > MAXDELAYD
                        MAXDELAYD = Clock - ArrInstant;
                    end
                    TRANSPACKETSD = TRANSPACKETSD + 1;
                else
                    TRANSBYTESV = TRANSBYTESV + PacketSize;
                    DELAYSV = DELAYSV + (Clock - ArrInstant);
                    if (Clock - ArrInstant) > MAXDELAYV
                        MAXDELAYV = Clock - ArrInstant;
```

```
                if PacketType == Dados
                    TRANSBYTESD = TRANSBYTESD + PacketSize;
                    DELAYSD = DELAYSD + (Clock - ArrInstant);
                    if (Clock - ArrInstant) > MAXDELAYD
                        MAXDELAYD = Clock - ArrInstant;
                    end
                    TRANSPACKETSD = TRANSPACKETSD + 1;
                else
                    TRANSBYTESV = TRANSBYTESV + PacketSize;
                    DELAYSV = DELAYSV + (Clock - ArrInstant);
                    if (Clock - ArrInstant) > MAXDELAYV
                        MAXDELAYV = Clock - ArrInstant;
                    end
                    TRANSPACKETSV = TRANSPACKETSV + 1;
                end
            else
                if PacketType == Dados
                    LOSTPACKETSD = LOSTPACKETSD + 1;
                else
                    LOSTPACKETSV = LOSTPACKETSV + 1;
                end
            end

            if QUEUEOCCUPATION > 0
                EventList = [EventList; DEPARTURE, Clock + 8*QUEUE(1, 1)/(C*10^6), QUEUE(1, 1), QUEUE(1, 2), QUEUE(1, 3)];
                QUEUEOCCUPATION = QUEUEOCCUPATION - QUEUE(1, 1);
                QUEUE(1, :) = [];
            else
                STATE = 0;
            end
    end
end

% Performance parameters:
PLD = 100 * LOSTPACKETSD / TOTALPACKETSD; % Packet loss for data packets (%)
PLV = 100 * LOSTPACKETSV / TOTALPACKETSV; % Packet loss for VoIP packets (%)
APDD = 1000 * DELAYSD / TRANSPACKETSD;    % Average delay for data packets (ms)
APDV = 1000 * DELAYSV / TRANSPACKETSV;    % Average delay for VoIP packets (ms)
MPDD = 1000 * MAXDELAYD;                  % Maximum delay for data packets (ms)
MPDV = 1000 * MAXDELAYV;                  % Maximum delay for VoIP packets (ms)
TT = 1e-6 * (TRANSBYTESD + TRANSBYTESV) * 8 / Clock; % Transmitted throughput (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
```

Bit Error Rate (BER) Handling:

if rand() < (1 - b)^(PacketSize * 8) % Simulate packet loss due to bit errors

This line introduces packet loss due to bit errors. The probability of a successful packet transmission is modeled as (1−BER)PacketSize in bits(1 - BER)^PacketSize in bits, where b is the bit error rate (BER) and PacketSize * 8 converts the packet size from bytes to bits.

Packet Arrival for Both Data and VoIP:

```
for i = 1:n
   tmp = unifrnd(0, 0.02);  % Uniform distribution between 0 ms and 20 ms for
VoIP
   EventList = [EventList; ARRIVAL, tmp, randi([110, 130]), tmp, Voip];
End
```

This loop schedules the arrival of n VoIP packet flows, where each flow has a uniformly distributed inter-arrival time between 0 ms and 20 ms. The packet size for VoIP is randomly selected between 110 and 130 bytes.

Simulating Packet Loss due to Buffer Overflow:

```
if QUEUEOCCUPATION + PacketSize <= f
   QUEUE = [QUEUE; PacketSize, Clock, Dados];
   QUEUEOCCUPATION = QUEUEOCCUPATION + PacketSize;
else
   LOSTPACKETSD = LOSTPACKETSD + 1;
End
```

This code checks whether there is enough space in the queue to accommodate the incoming data packet. If the queue has enough capacity (QUEUEOCCUPATION + PacketSize <= f), the packet is added to the queue; otherwise, it is dropped, and the LOSTPACKETSD counter is incremented.

Event Handling: Arrival and Departure:

```
EventList = sortrows(EventList, 2);  % Order EventList by time
Event = EventList(1, 1);            % Get first event
Clock = EventList(1, 2);            % Time of the event
PacketSize = EventList(1, 3);       % Packet size
ArrInstant = EventList(1, 4);       % Arrival time
PacketType = EventList(1, 5);       % Packet type (Dados/Voip)
EventList(1, :) = [];               % Remove the processed event
```

This block handles the main event-driven structure of the simulation. The event list is sorted by the time of each event. Then the next event (either an **ARRIVAL** or **DEPARTURE**) is processed by updating the clock and fetching the associated packet information.

Scheduling Departures:

```
EventList = [EventList; DEPARTURE, Clock + 8*PacketSize/(C*10^6),
PacketSize, Clock, Dados];
```

This line schedules the departure of a packet after it has been processed. The time until departure is calculated based on the packet size and the link capacity (C), converting the size from bytes to bits and adjusting for Mbps.

Packet Size Generation:

```
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
```

This function generates a random packet size based on probabilities. There is a 19% chance of generating a 64-byte packet, a 23% chance for a 110-byte packet, and a 17% chance for a 1518-byte packet. If none of these apply, a random packet size is chosen from the range [65-109] or [111-1517].

2.1.1 Justifying the changes introduced

```
if rand() < (1 - b)^(PacketSize*8) % Simulate packet loss due to bit errors
```

Modification in the DEPARTURE Case:

is designed to simulate the probability of a packet successfully transmitting without errors over a communication link.

2.2 Exercise 2.b)

2.2.1 Results

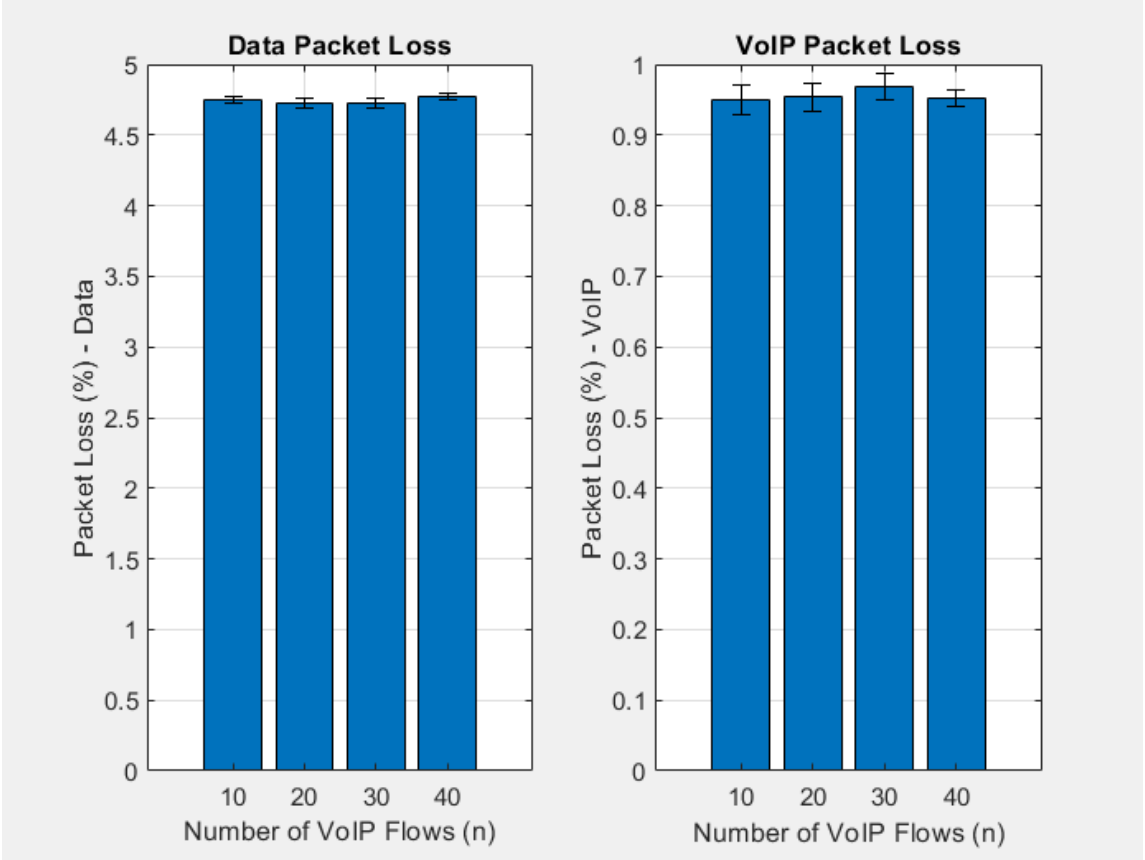

Figure 3: The average packet loss results for VoIP and Data, b = 1e-5

2.2.2 Conclusions

Packet Loss for Data: The packet loss for data traffic remains around **4.7%** across different numbers of VoIP flows. Since the queue is large enough, the packet loss can be attributed mainly to **BER-induced errors**. The relatively stable packet loss percentage suggests that **bit errors** are the dominant cause of data packet loss, with minimal influence from queue congestion.

The **packet loss percentage for data** is **higher than for VoIP** due to the **larger size of data packets**. Data packets typically range up to **1518 bytes**, while VoIP packets are much smaller, generally around **110-130 bytes**. Since larger packets have more bits, the probability that at least one bit in a data packet will be corrupted by a bit error is higher. This leads to a greater chance that a data packet will be discarded due to transmission errors, explaining why data packet loss is higher than VoIP packet loss.

Packet Loss for VoIP: The packet loss for VoIP traffic is slightly lower, ranging around **0.96%**. VoIP packets are smaller in size compared to data packets, making them less likely to encounter bit errors that cause the entire packet to be discarded. This explains why the VoIP packet loss is lower than data packet loss, even though both are impacted by the same BER. The smaller packet size means fewer bits are transmitted, reducing the chance of encountering a bit error within any given packet.

Conclusion: The main conclusion is that bit errors (BER) are the primary cause of packet loss in this scenario, given the large queue size. The queue size is more than sufficient to accommodate both data and VoIP traffic without causing overflow, so any packet loss observed is primarily due to errors introduced during transmission. As a result, packet loss remains relatively constant across different numbers of VoIP flows, as the number of flows does not influence the error rate caused by BER.

2.3 Exercise 2.c)

2.3.1 Results

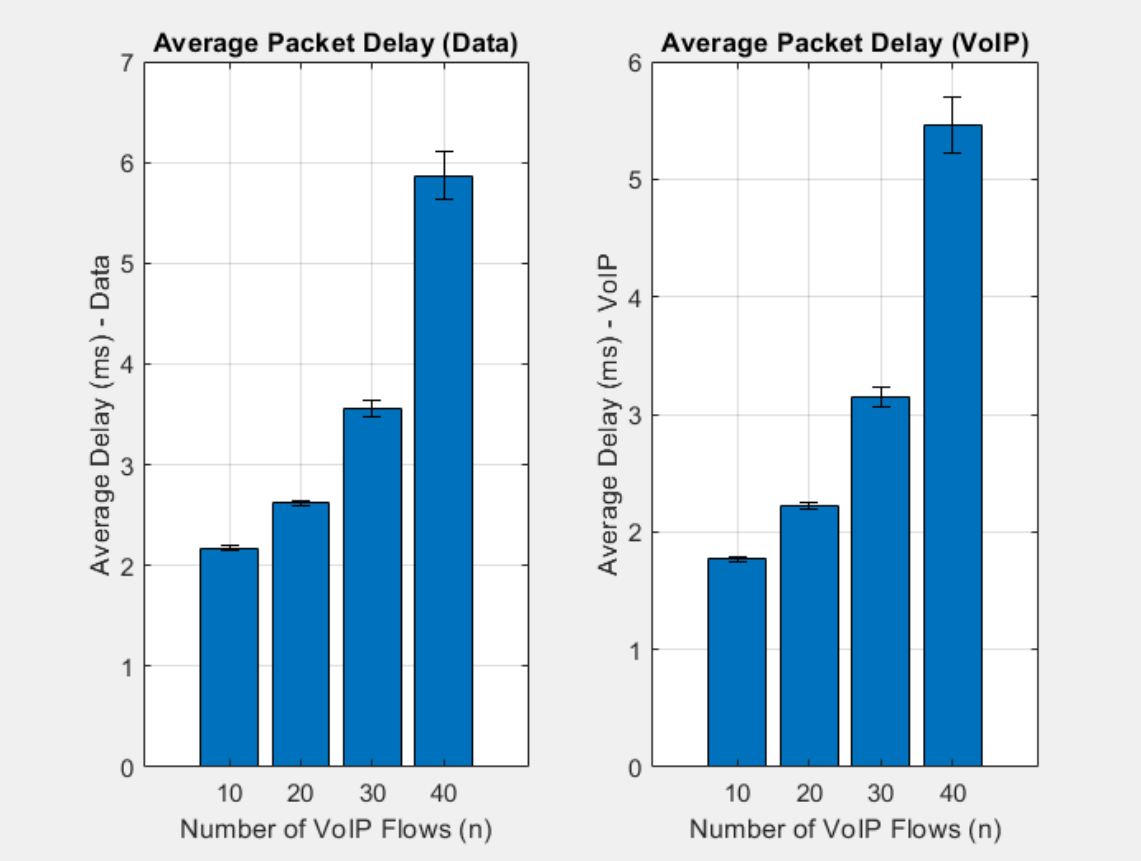

Figure 4: The average Data packet delay and the average VoIP packet delay results, b = 1e-5

2.3.2 Conclusions

Data Packet Delay:

As the number of VoIP flows increases, the average delay for data packets also increases. This is expected because more VoIP flows increase the overall load on the system, leading to longer wait times for data packets in the queue.

The steep increase in delay from 30 to 40 VoIP flows suggests that the network may be approaching congestion, where the queue becomes increasingly occupied, leading to more significant delays.

VoIP Packet Delay:

Like data packets, the average delay for VoIP packets also increases as the number of flows increases. This is again expected due to higher contention for bandwidth and queue space.

The delay for VoIP packets remains lower than data packets, which makes sense if VoIP packets are smaller, but as the network becomes more saturated (especially at 40 flows), the delays converge.

2.4 Exercise 2.d)

2.4.1 Results

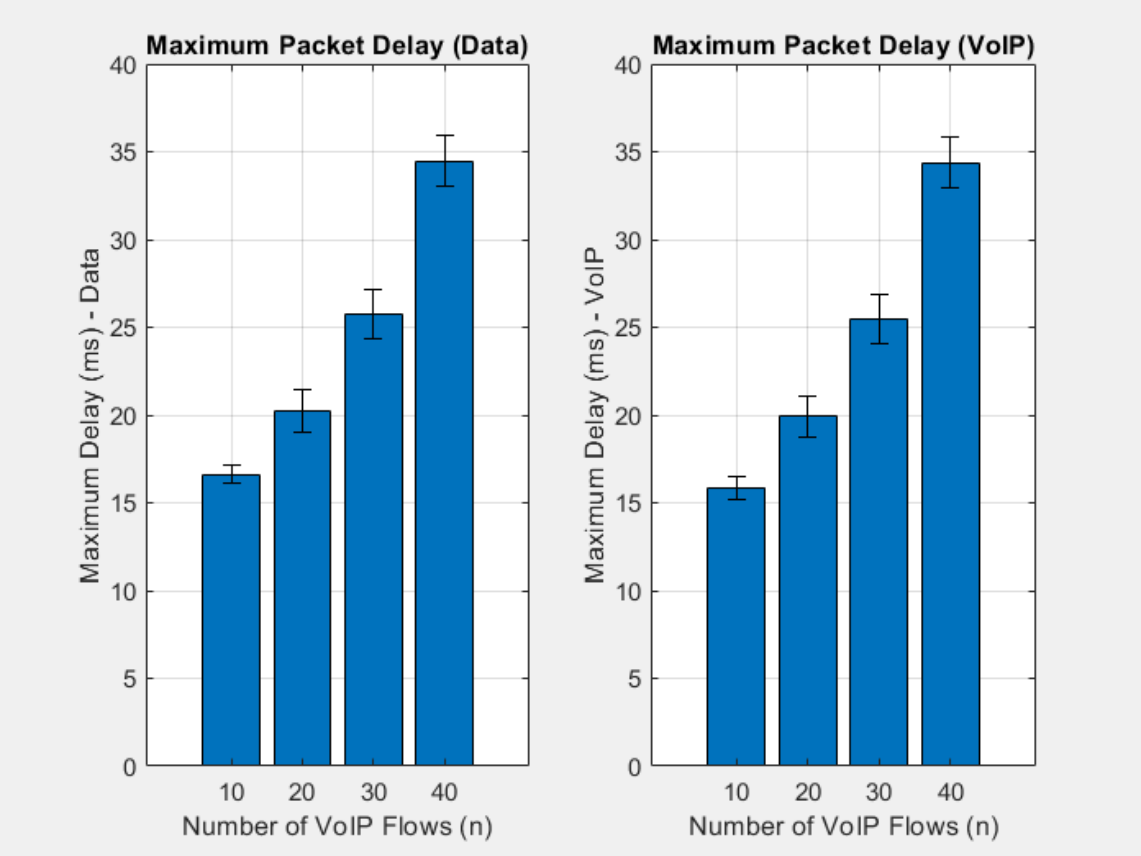

Figure 5: The maximum Data packet delay and the maximum VoIP packet delay results, b = 1e-5

2.4.2 Conclusions

Congestion as the Primary Cause of Delay:

The increasing delays for both data and VoIP traffic indicate that **congestion is the primary factor** affecting packet delay as the number of VoIP flows increases. As more flows are introduced, the link capacity becomes more contested, leading to longer queue times and, consequently, higher delays for both types of traffic.

Data Traffic Faces Higher Delays:

Due to the larger packet size for data traffic, the delays for data packets are generally higher than for VoIP packets. This reflects the additional time required to transmit larger packets over a congested network, even when no prioritization is in place.

2.5 Exercise 2.e)

2.5.1 Results

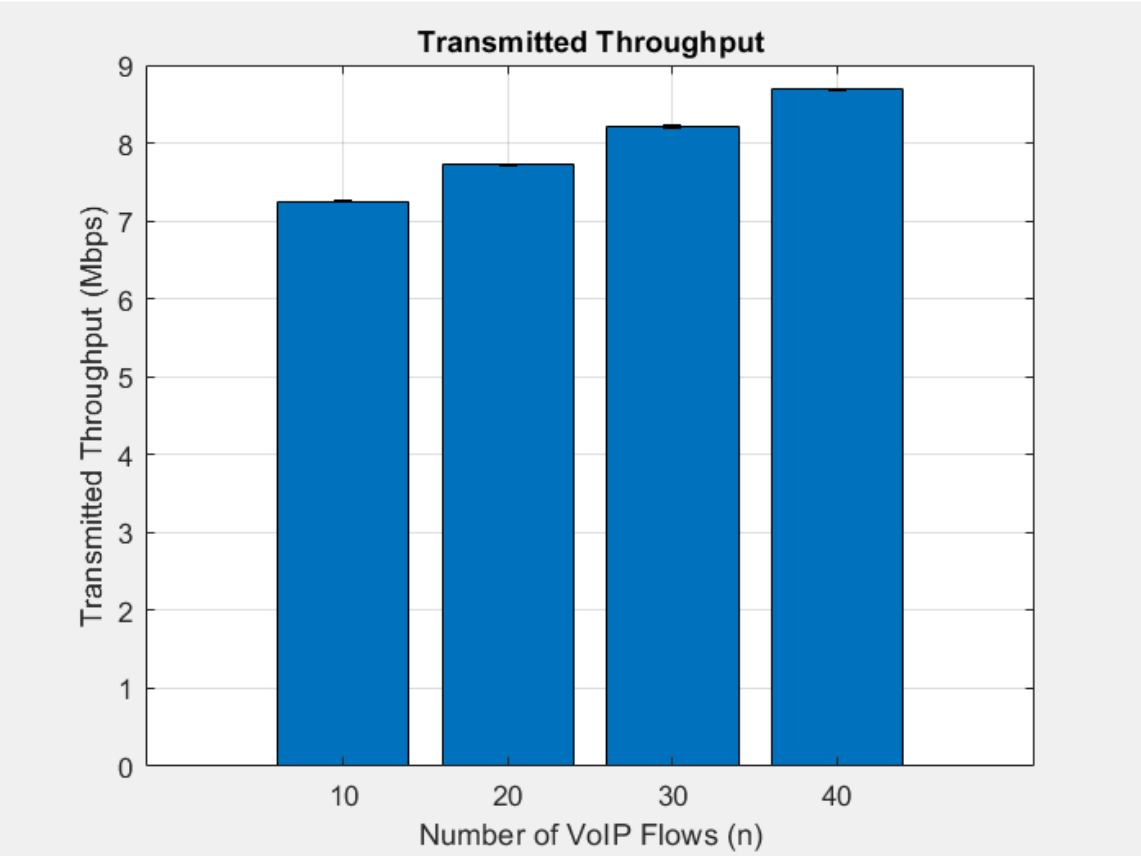

Figura 6: Simulated Transmitted Throughput results, b = 1e-5

2.5.2 Conclusions

Increasing Number of VoIP Flows (n):

The transmitted throughput shown in the graph increases as the number of VoIP flows (n) increases from 10 to 40. This is expected because adding more VoIP flows increases the total amount of data being transmitted over the network, contributing to a higher throughput.

Saturation of Bandwidth:

The throughput values reach close to the maximum link bandwidth capacity of 10 Mbps as the number of VoIP flows increases. For 40 VoIP flows, the throughput is nearing the 9 Mbps mark, indicating that the link is approaching saturation. With more VoIP flows, the available bandwidth becomes more fully utilized.

Packet Loss Impact:

Even though the throughput increases with more VoIP flows, there is a limit due to the queue size and bit error rate (BER). The BER (1e-5) introduces errors in transmitted packets, causing some to be discarded, especially at higher rates. However, despite the BER, the system manages to transmit a large portion of the traffic, which is why the throughput values are still high, especially at 30 and 40 VoIP flows.

Conclusion:

The results suggest that the network is capable of efficiently managing both data and VoIP traffic, but as VoIP flows increase further, the network may eventually reach a limit where throughput growth tapers off or packet loss becomes more significant.

2.6 Exercise 2.f)

2.6.1 Code

```
% General Parameters
lambda_data = 1500;             % Packet arrival rate for data (pps)
C = 10;                         % Link bandwidth (Mbps)
n_values = [10, 20, 30, 40];    % Different numbers of VoIP flows
pps_voip = 50;                  % Packet arrival rate for VoIP per flow (pps)
ber = 1e-5;                     % Bit Error Rate (BER)

% Average size of VoIP packets (110 to 130 bytes)
mean_voip_size = mean(110:130);  % Equal probability for all VoIP packet sizes

% Possible data packet sizes (65:109, 111:1517 bytes)
prob_left = (1 - (0.19 + 0.23 + 0.17)) / ((109 - 65 + 1) + (1517 - 111 + 1));

% Weighted average size of data packets
mean_data_size = 0.19 * 64 + 0.23 * 110 + 0.17 * 1518 + ...
    sum((65:109) * prob_left) + sum((111:1517) * prob_left);

% Probability of successful transmission (without errors) for a packet of size S bits
% P(success) = (1 - BER)^(packet_size_in_bits)
P_success_data = (1 - ber)^(mean_data_size * 8);      % for data packets
P_success_voip = (1 - ber)^(mean_voip_size * 8);      % for VoIP packets

% Initialize vectors to store results
throughput_data_mean = zeros(1, length(n_values));
throughput_voip_mean = zeros(1, length(n_values));
throughput_total_mean = zeros(1, length(n_values));
```

```
% Loop to calculate theoretical throughput for each value of n
for i = 1:length(n_values)
    n = n_values(i);  % Number of VoIP flows

    % Effective Throughput for data packets
    throughput_data = lambda_data * mean_data_size * 8 / 10^6 * P_success_data;  % Convert to Mbps

    % Effective Throughput for VoIP packets
    throughput_voip = n * pps_voip * mean_voip_size * 8 / 10^6 * P_success_voip;  % Convert to Mbps

    % Total throughput calculation
    total_throughput = throughput_data + throughput_voip;

    % Store results for the bar chart
    throughput_data_mean(i) = throughput_data;
    throughput_voip_mean(i) = throughput_voip;
    throughput_total_mean(i) = total_throughput;

    % Display the results
    fprintf('\nTheoretical Results with BER for n = %d VoIP flows:\n', n);
    fprintf('  Theoretical Throughput (Data) = %.2f Mbps\n', throughput_data);
    fprintf('  Theoretical Throughput (VoIP) = %.2f Mbps\n', throughput_voip);
    fprintf('  Total Theoretical Throughput = %.2f Mbps\n', total_throughput);
end

% Generate bar chart for total theoretical throughput
figure;
bar(n_values, throughput_total_mean);
hold on;

% Customize the chart
xlabel('Number of VoIP Flows (n)');
ylabel('Total Theoretical Throughput (Mbps)');
title('Total Theoretical Throughput with BER (Data + VoIP)');
grid on;
hold off;

% Display the chart
fprintf('Bar chart displayed for total theoretical throughput considering BER.\n');
```

General Parameters:

lambda_data = 1500: This is the packet arrival rate for data, defined as 1500 packets per second (pps).

C = 10: The link bandwidth is set to 10 Mbps.

n_values = [10, 20, 30, 40]: These are the different numbers of VoIP flows being considered in the simulation.

pps_voip = 50: Each VoIP flow generates 50 packets per second (pps).

ber = 1e-5: This is the **Bit Error Rate (BER)**, representing the likelihood of being a bit corrupt during transmission. A BER of 1e-5 means there is a 0.001% chance of a bit error.

Packet Size Calculations:

mean_voip_size = mean(110:130): The average size of VoIP packets is computed as the mean of sizes ranging from 110 to 130 bytes.

mean_data_size: The average size of data packets is computed using a weighted formula based on probabilities for 64-byte, 110-byte, and 1518-byte packets. The remaining sizes between 65-109 and 111-1517 bytes are handled with equal probability.

Probability of Successful Transmission:

P_success_data = (1 - ber)^(mean_data_size * 8): This formula calculates the probability that a data packet is transmitted without errors, based on the **BER** and the average size of the data packet in bits.

P_success_voip = (1 - ber)^(mean_voip_size * 8): Similarly, this calculates the probability of a successful transmission for a VoIP packet.

Loop for Different VoIP Flow Values:

Data Throughput Calculation:

throughput_data = lambda_data * mean_data_size * 8 / 10^6 * P_success_data;

This formula computes the throughput for data packets in Mbps, considering the packet arrival rate (lambda_data), average packet size (mean_data_size), and the probability of successful transmission (P_success_data). The multiplication by 8 converts bytes to bits, and the division by 10^6 converts bits per second to megabits per second (Mbps).

VoIP Throughput Calculation:

throughput_voip = n * pps_voip * mean_voip_size * 8 / 10^6 * P_success_voip;

This calculates the throughput for VoIP packets. It multiplies the number of VoIP flows (n), VoIP packet rate per flow (pps_voip), and average VoIP packet size (mean_voip_size), along with the successful transmission probability (P_success_voip).

Total Throughput:

total_throughput = throughput_data + throughput_voip;

The total throughput is the sum of both the data and VoIP throughputs for each n.

VoIP packets are typically sent every **20 ms** (milliseconds).
This is because many VoIP systems use a packetization interval of **20 ms** to balance latency and bandwidth efficiency.

1 second / 0.02 second = 50 pps => pps_voip = 50;

The **theoretical throughput** for a network with both **data and VoIP traffic**. The total throughput is the sum of the throughputs for both types of traffic. The code uses a **weighted average packet size** for data and an average packet size for VoIP, assuming 50 pps for each VoIP flow.

2.6.2 Results

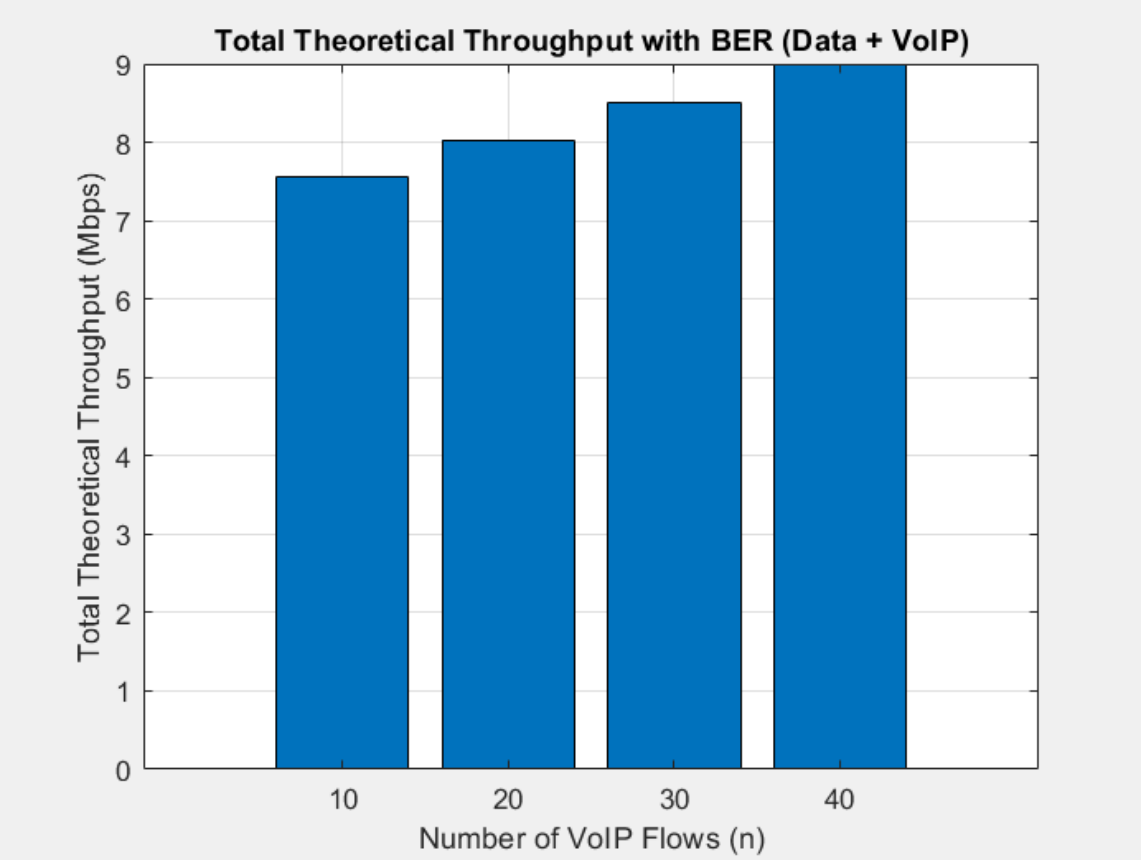

Figure 7: Theoretical Transmitted Throughput results with BER, b = 1e-5

2.6.3 Conclusions

Conclusion:

The main reason for the slight difference between theoretical and simulated throughput is the **queue management**, including congestion, packet loss, and delays introduced by the queue. These factors directly impact system performance in the simulation but are typically ignored in theoretical calculations, resulting in a small discrepancy between the values.

The difference between theoretical and simulated values ranges from **0.3 to 0.6 Mbps**, depending on the number of VoIP flows.

Chapter 3

Task 3

3.1 Exercise 3.a)

3.1.1 Results

Code of the experiment in exercise 3.a), using Simulator3 (developed in the lab classes), that estimates the performance parameters when both services (VoIP and Data) are statistically multiplexed in a single FIFO queue.

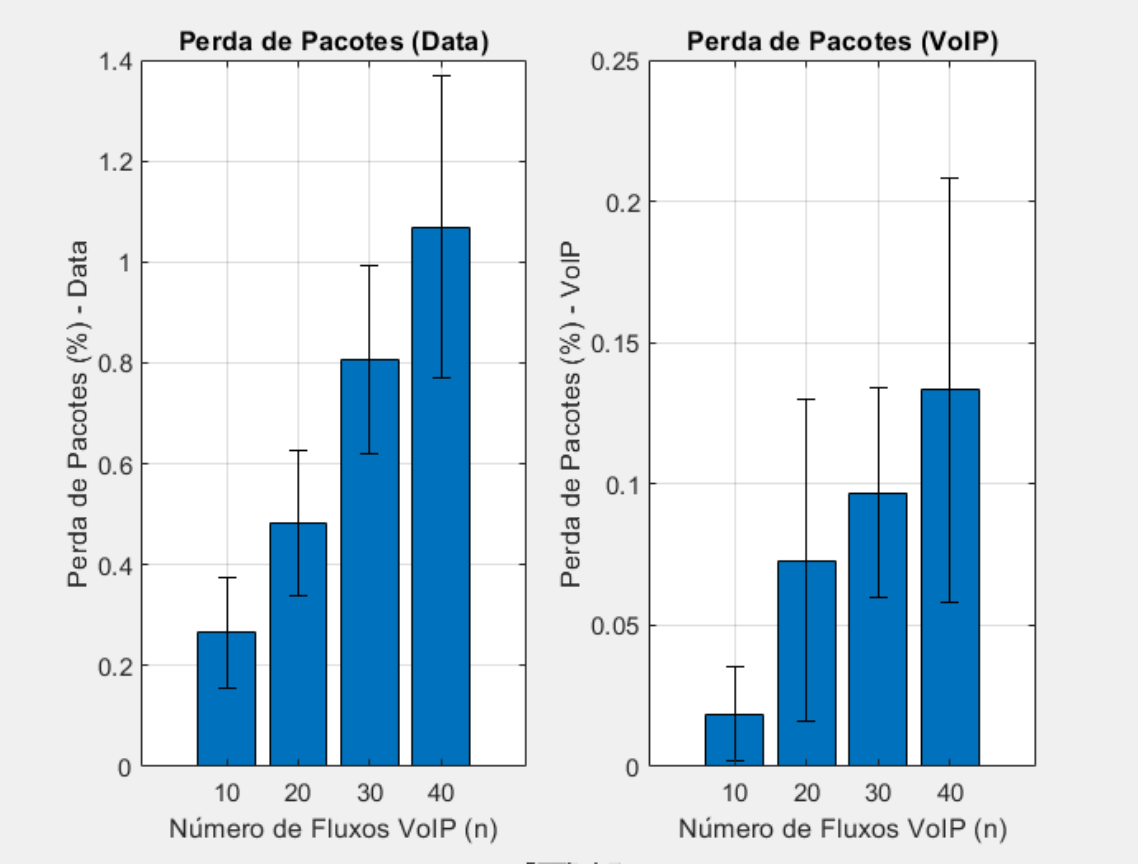

Figure 8: The average packet loss results for VoIP and Data

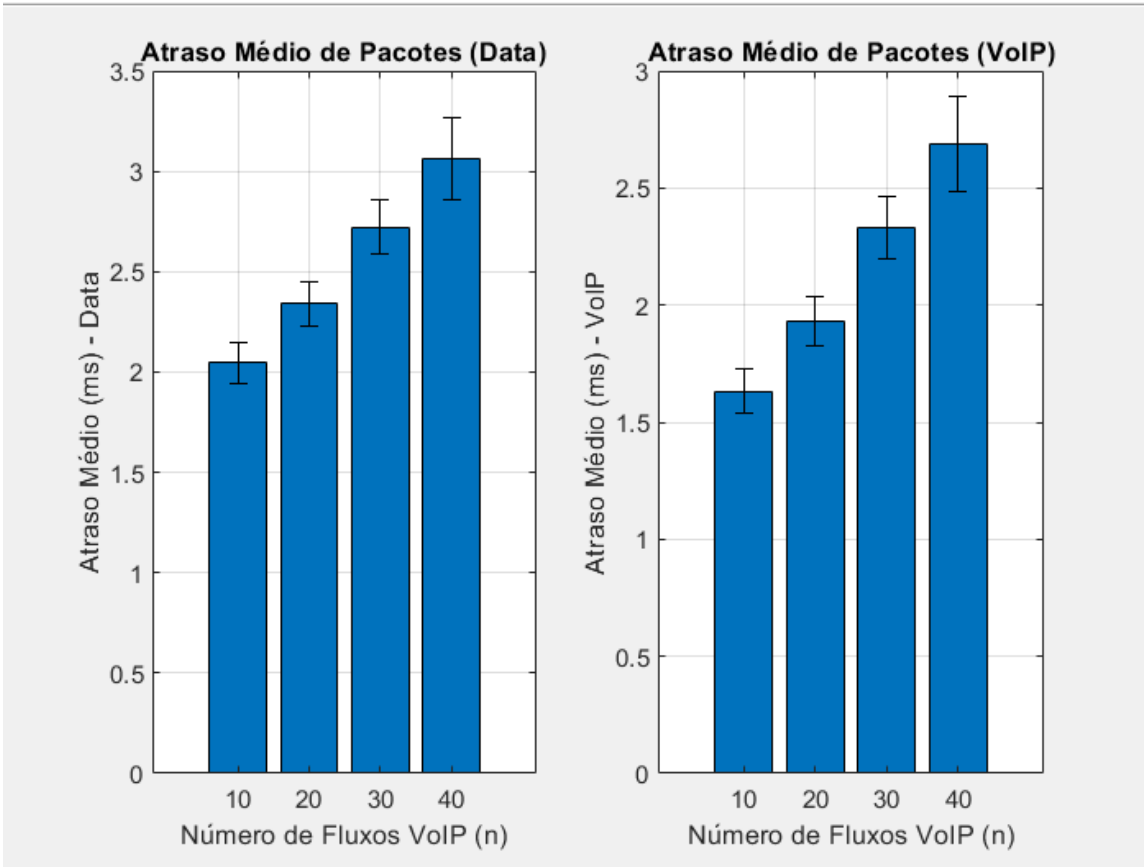

Figure 9: The average packet delay results for Data and VoIP

3.1.2 Conclusions

Data Packet Loss (PLD): In a FIFO (First In, First Out) queue system, data packets and VoIP packets compete for the same queue space. As the number of VoIP flows (n) increases, more packets enter the queue, increasing competition for bandwidth and buffer space. Data packets are generally larger and due to this size, they occupy more space in the buffer. As a result, the buffer fills up faster, and this leads to more frequent **packet drops** as the buffer reaches its capacity. As the VoIP load increases, data packets face higher losses due to the increased congestion.

VoIP Packet Loss (PLV): VoIP packet loss also increases as n grows, but the effect is less pronounced compared to data packets. VoIP packets are typically smaller (between 110–130 bytes), allowing more of them to fit into the buffer before it overflows. However, as n increases substantially, the number of VoIP packets competing for buffer space becomes significant, leading to an increase in packet loss for VoIP as well.

Data Packet Delay (APDD): As the number of VoIP flows increases, the average delay for data packets increases. This is because data packets, which arrive at a fixed rate of 1500 pps, must wait longer in the queue due to the increased presence of VoIP packets. Data packets are also larger, meaning that once a data packet starts being transmitted, it occupies the transmission channel for longer, increasing the average delay as n increases.

VoIP Packet Delay (APDV): The delay for VoIP packets also increases as the number of VoIP flows increases. However, the delay is generally less severe compared to data packets because VoIP packets are smaller. They can be transmitted more quickly, and despite increased competition for bandwidth with data packets, their smaller size helps to mitigate the impact on delay. Nonetheless, as n increases significantly, the delay for VoIP packets will also rise as more packets compete for the same transmission channel.

FIFO Queue Impact: Since the system uses a FIFO queue, no prioritization is given to either type of packet. Both data and VoIP packets are treated equally in terms of queuing and transmission. This means that as the number of VoIP flows increases, both types of traffic suffer from increased competition, but data packets tend to be more severely impacted due to their larger size and longer transmission times.

Packet Size Differences: Data packets have variable sizes, ranging from 65 to 1518 bytes, with some very large packets (1518 bytes) that can significantly increase transmission time and buffer occupation. VoIP packets, on the other hand, are smaller and more uniform in size (110-130 bytes), which allows them to be processed faster and with less delay.

3.2 Exercise 3.b)

3.2.1 Results

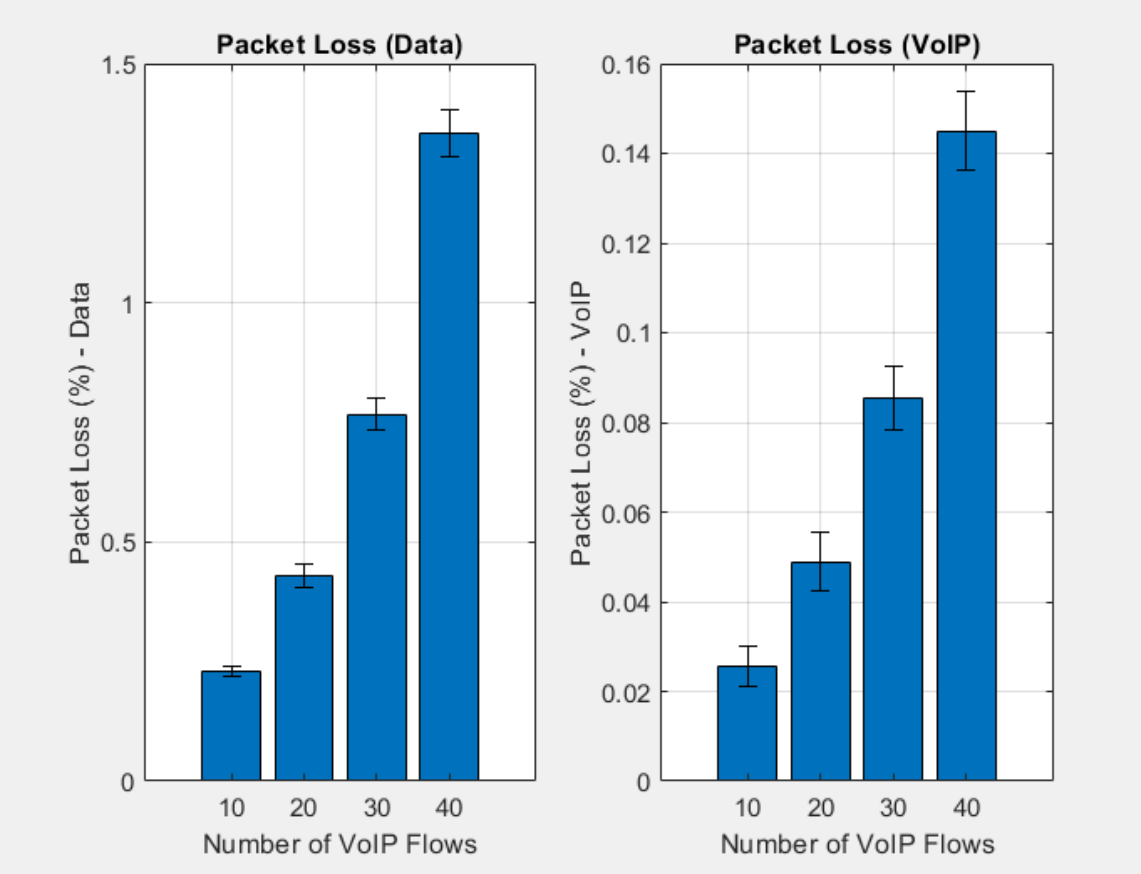

Figure 10: The average packet loss results for Data and VoIP

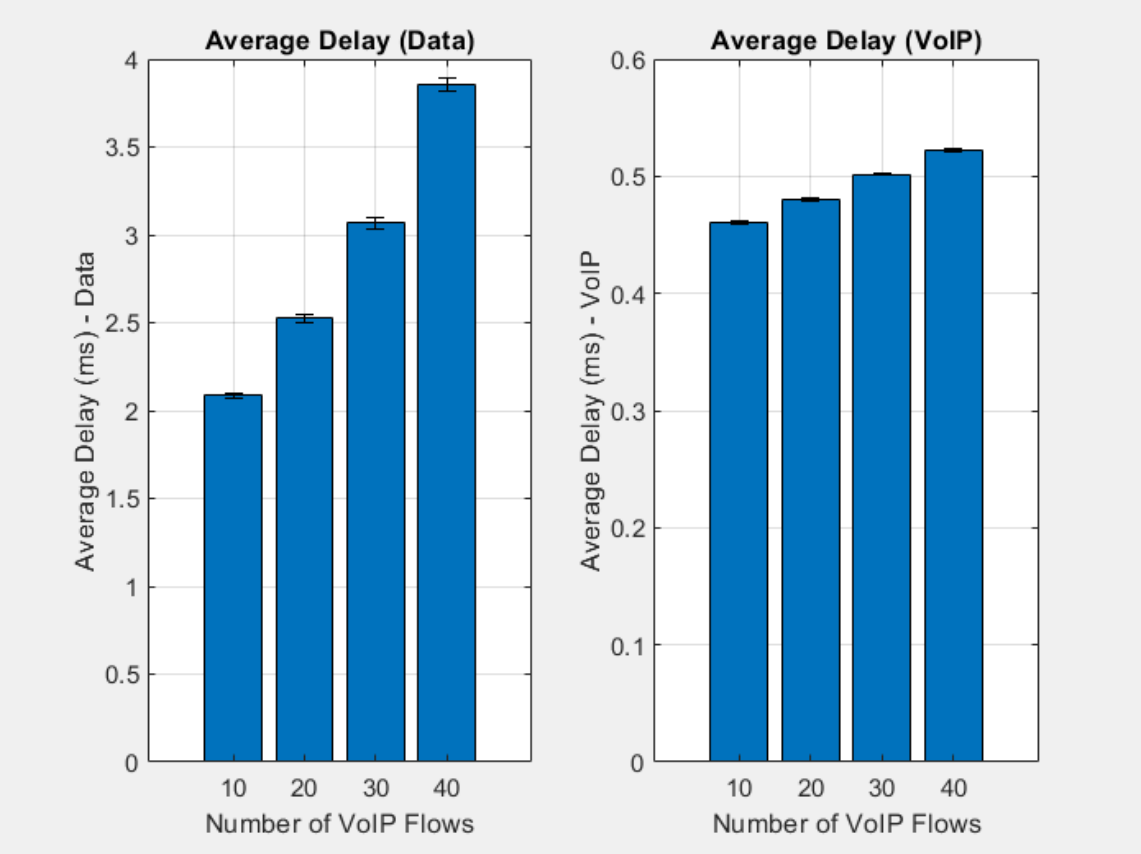

Figure 11: The average packet delay results for Data and VoIP

3.2.2 Conclusions

Data Packet Delay (APDD): With VoIP given priority in **Sim4**, data packets often have to wait longer in the queue. As the number of VoIP flows increases, more VoIP packets occupy the transmission link, which forces data packets to be delayed. This effect becomes especially pronounced at higher flow numbers, where the delay for data packets becomes substantial due to the preemption by VoIP traffic.

VoIP Packet Delay (APDV): The prioritization mechanism in **Sim4** ensures that VoIP packets are transmitted with minimal delay. VoIP traffic experiences consistently low delays because it is processed immediately in the queue, without waiting behind data traffic. The slight increase in delay as the number of VoIP flows increases is due to the competition among VoIP flows themselves, but the overall delay remains low due to the prioritization.

In Sim3 (FIFO Queue, No Prioritization): In Sim3, both data and VoIP traffic share the same priority. As a result, the packet loss and delay for both services increases somewhat similarly as the number of VoIP flows increases.

Data and VoIP traffic are both subject to queue congestion equally, leading to more balanced performance degradation between the two services. VoIP experiences higher packet loss and delay in Sim3 compared to Sim4 because it does not have priority.

In Sim4 (VoIP Priority): Data traffic suffers more in terms of both packet loss and delay, especially as the number of VoIP flows increases. This is because the prioritization mechanism diverts queue resources to VoIP traffic at the expense of data traffic.

VoIP performance improves dramatically compared to **Sim3**. Packet loss for VoIP is significantly lower in **Sim4**, and the delay is also much more consistent and lower. This reflects the advantage of prioritizing VoIP traffic in time-sensitive applications like real-time communication.

Conclusions:

Trade-off Between Services: The use of a priority-based queue in **Sim4** highlights the trade-off between different types of services. While VoIP benefits significantly in terms of packet loss and delay, the performance of data traffic is compromised. This reflects a fundamental trade-off in network design: prioritizing certain traffic can improve quality of service for those flows, but at the cost of degrading performance for other traffic types.

VoIP Performance: In scenarios where **real-time applications** like VoIP are critical, using prioritization as in **Sim4** can ensure a high quality of service. This is particularly useful in networks where minimizing delay and packet loss for VoIP is essential (e.g., video conferencing, voice over IP).

Data Traffic Degradation: The degradation in data performance in **Sim4** indicates that network administrators must carefully consider the implications of prioritization schemes. For applications where data integrity or transmission is sensitive (e.g., large file transfers), such prioritization might not be suitable without adequate bandwidth or QoS mechanisms in place to mitigate the negative effects on data traffic.

3.3 Exercise 3.c)

3.3.1 Code

The code of exercise 3.c) has the intent of doing the same experiments that were made in exercise 3.b), using a different version of Simulator4, therefore, the code shown below is the Simulator4A's most relevant differences.

```
function [PLD, PLV, APDD, APDV, MPDD, MPDV, TT] = Sim4A(lambda, C, f, P, n, p)
    % INPUT PARAMETERS:
    % lambda - packet rate (packets/sec)
    % C      - link bandwidth (Mbps)
    % f      - queue size (Bytes)
    % P      - stopping criterion (number of packets)
    % n      - number of VoIP packet flows
    % p      - percentage of queue for data packet occupation

    % Events and State Variables
    ARRIVAL = 0;
    DEPARTURE = 1;

    Dados = 0;  % Data packet type
    Voip = 1;   % VoIP packet type

    STATE = 0;
    QUEUEOCCUPATION = 0;
    QUEUE = [];

    % Statistical Counters
    TOTALPACKETSD = 0;
    TOTALPACKETSV = 0;
    LOSTPACKETSD = 0;
    LOSTPACKETSV = 0;
    TRANSPACKETSD = 0;
    TRANSPACKETSV = 0;
    TRANSBYTESD = 0;
    TRANSBYTESV = 0;
    DELAYSD = 0;
    DELAYSV = 0;
    MAXDELAYD = 0;
    MAXDELAYV = 0;

    % Initialize simulation clock
    Clock = 0;
    tmp = Clock + exprnd(1/lambda);
    EventList = [ARRIVAL, tmp, GeneratePacketSize(), tmp, Dados];

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

```
while (TRANSPACKETSD + TRANSPACKETSV) < P
    EventList = sortrows(EventList, 2);  % Sort events by time
    Event = EventList(1, 1);
    Clock = EventList(1, 2);
    PacketSize = EventList(1, 3);
    ArrInstant = EventList(1, 4);
    PacketType = EventList(1, 5);
    EventList(1, :) = [];  % Remove the event after processing

    switch Event
        case ARRIVAL
            if PacketType == Dados
                TOTALPACKETSD = TOTALPACKETSD + 1;
                tmp = Clock + exprnd(1/lambda);
                EventList = [EventList; ARRIVAL, tmp, GeneratePacketSize(), tmp, Dados];

                % Check if the queue is occupied by more than p% of its capacity
                if STATE == 0
                    STATE = 1;
                    EventList = [EventList; DEPARTURE, Clock + 8 * PacketSize / (C * 1e6), PacketSize, Clock, Dados];
                else
                    % Data packets are only accepted if the queue occupation is less than p% of the total capacity
                    if QUEUEOCCUPATION + PacketSize <= (p / 100) * f
                        QUEUE = [QUEUE; PacketSize, Clock, Dados];
                        QUEUEOCCUPATION = QUEUEOCCUPATION + PacketSize;
                        % Sort the queue to prioritize VoIP packets
                        QUEUE = sortrows(QUEUE, 3, 'descend');  % Sort by PacketType, prioritize VoIP
                    else
                        LOSTPACKETSD = LOSTPACKETSD + 1;
                    end
                end
            else
                TOTALPACKETSV = TOTALPACKETSV + 1;
                tmp = Clock + unifrnd(0.016, 0.024);
                EventList = [EventList; ARRIVAL, tmp, randi([110, 130]), tmp, Voip];

                % VoIP packets are accepted if there is enough space in the queue
                if STATE == 0
                    STATE = 1;
                    EventList = [EventList; DEPARTURE, Clock + 8 * PacketSize / (C * 1e6), PacketSize, Clock, Voip];
                else
                    if QUEUEOCCUPATION + PacketSize <= f
                        QUEUE = [QUEUE; PacketSize, Clock, Voip];
                        QUEUEOCCUPATION = QUEUEOCCUPATION + PacketSize;
                        % Sort the queue to prioritize VoIP packets
                        QUEUE = sortrows(QUEUE, 3, 'descend');  % Sort by PacketType, prioritize VoIP
                    else
                        LOSTPACKETSV = LOSTPACKETSV + 1;
```

```
            case DEPARTURE
                if PacketType == Dados
                    TRANSBYTESD = TRANSBYTESD + PacketSize;
                    DELAYSD = DELAYSD + (Clock - ArrInstant);
                    if Clock - ArrInstant > MAXDELAYD
                        MAXDELAYD = Clock - ArrInstant;
                    end
                    TRANSPACKETSD = TRANSPACKETSD + 1;
                else
                    TRANSBYTESV = TRANSBYTESV + PacketSize;
                    DELAYSV = DELAYSV + (Clock - ArrInstant);
                    if Clock - ArrInstant > MAXDELAYV
                        MAXDELAYV = Clock - ArrInstant;
                    end
                    TRANSPACKETSV = TRANSPACKETSV + 1;
                end
                if QUEUEOCCUPATION > 0
                    % Process the next packet, which is now sorted by priority
                    EventList = [EventList; DEPARTURE, Clock + 8 * QUEUE(1, 1) / (C * 1e6), QUEUE(1, 1), QUEUE(1, 2), QUEUE(1, 3)];
                    QUEUEOCCUPATION = QUEUEOCCUPATION - QUEUE(1, 1);
                    QUEUE(1, :) = [];
                else
                    STATE = 0;
                end
        end
    end

    % Performance Parameters
    PLD = 100 * LOSTPACKETSD / TOTALPACKETSD;
    PLV = 100 * LOSTPACKETSV / TOTALPACKETSV;
    APDD = 1000 * DELAYSD / TRANSPACKETSD;
    APDV = 1000 * DELAYSV / TRANSPACKETSV;
    MPDD = 1000 * MAXDELAYD;
    MPDV = 1000 * MAXDELAYV;
    TT = 1e-6 * (TRANSBYTESD + TRANSBYTESV) * 8 / Clock;

end

function out = GeneratePacketSize()
    aux = rand();
    aux2 = [65:109 111:1517];  % Range of packet sizes
    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
```

Queue Acceptance for Data Packets:

```
if QUEUEOCCUPATION + PacketSize <= (p / 100) * f
```

This condition checks if the queue occupation for data packets exceeds a percentage `p` of the total queue size `f`. The parameter `p` is passed as an input to the function, which means the user can control what percentage of the total queue can be occupied by data packets.

Sorting the Queue:

```
QUEUE = sortrows(QUEUE, 3, 'descend');  % Sort by PacketType, prioritize VoIP
```

This line reorders the queue by packet type after each data packet is added. It ensures that VoIP packets always receive priority over data packets when packets are processed for departure.

VoIP Packet Acceptance:

```
if QUEUEOCCUPATION + PacketSize <= f
```

This condition checks if there is enough space in the queue for a VoIP packet. If the queue is not full, the VoIP packet is added.

Handling Data Packet Loss:

```
LOSTPACKETSD = LOSTPACKETSD + 1;
```

If a data packet cannot be added to the queue because the queue occupation has already exceeded the allowed p%, this line increases the counter for lost data packets.

Queue Processing:

```
if QUEUEOCCUPATION > 0
   EventList = [EventList; DEPARTURE, Clock + 8 * QUEUE(1, 1) / (C * 1e6),
QUEUE(1, 1), QUEUE(1, 2), QUEUE(1, 3)];
   QUEUEOCCUPATION = QUEUEOCCUPATION - QUEUE(1, 1);
   QUEUE(1, :) = [];
else
   STATE = 0;
End
```

This block checks if there are any packets remaining in the queue when a packet departs. If so, the next packet (now prioritized with VoIP packets at the front) is scheduled for departure.

This logic ensures that data packets are only allowed in the queue if they don't exceed a pre-defined percentage (p%) of the queue's capacity. This way, some portion of the queue remains reserved for VoIP packets, which may have higher priority.

The main purpose of these changes is to **prioritize VoIP traffic** over data traffic, ensuring better performance for VoIP services, which are delay sensitive.

By limiting the amount of data packets in the queue, we prevent the queue from being filled up by data traffic, leaving enough room for VoIP packets to be processed quickly.

This is especially relevant in scenarios with mixed traffic types (e.g., file transfers and voice calls) where maintaining a high quality of service (QoS) for VoIP is critical.

3.4 Exercise 3.d)

3.4.1 Results

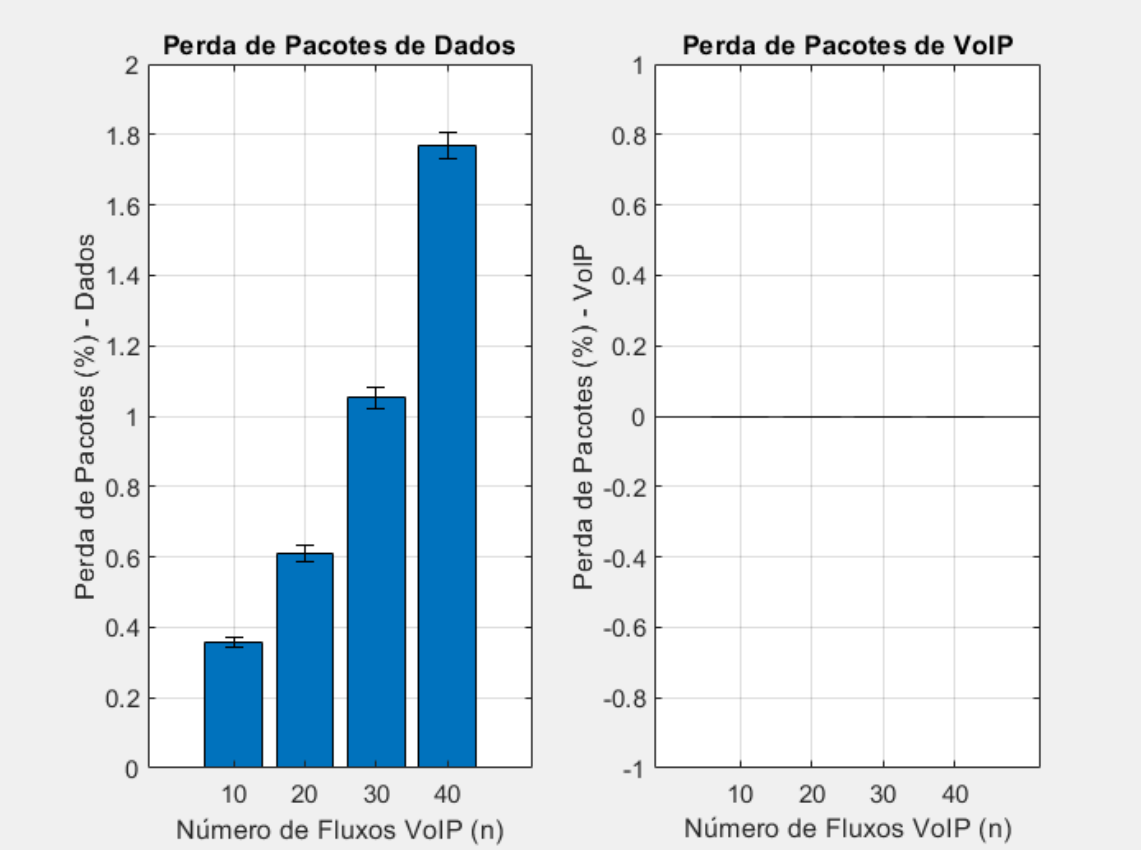

Figure 12: The average packet loss results for Data and VoIP

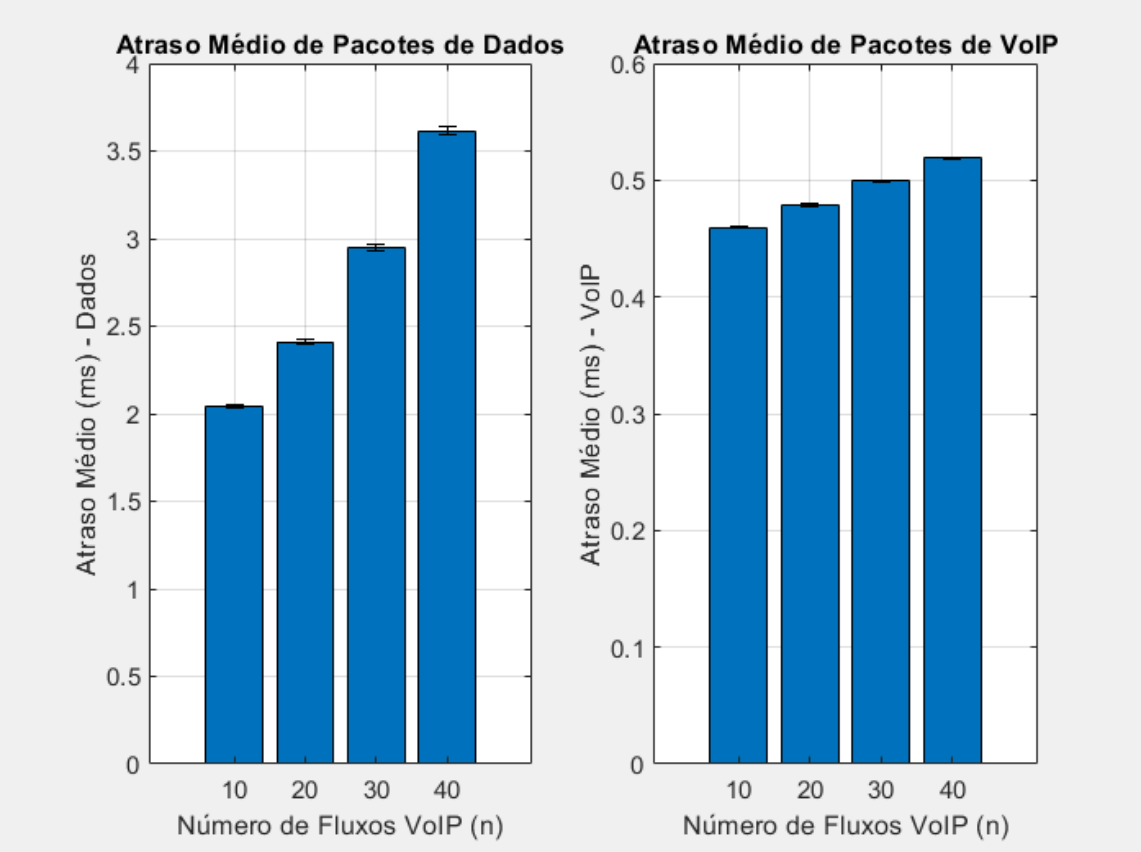

Figure 13: The average packet delay results for Data and VoIP

3.4.2 Conclusions

The introduction of the packet prioritization mechanism in Sim4A results in improved performance for VoIP packets in terms of both packet loss and delay. VoIP packet loss is virtually eliminated, and delays remain minimal and consistent, even as the number of VoIP flows increases.

However, this prioritization comes at the cost of data service quality. Data packets experience both higher packet losses and increased delays, particularly as the number of VoIP flows increases. This demonstrates the trade-off inherent in prioritizing one service over another in a shared queue.

Overall, Sim4A ensures superior performance for VoIP but sacrifices data packet performance. The system achieves its objective of enhancing VoIP performance at the cost of lower-quality data service.

3.5 Exercise 3.e)

3.5.1 Results

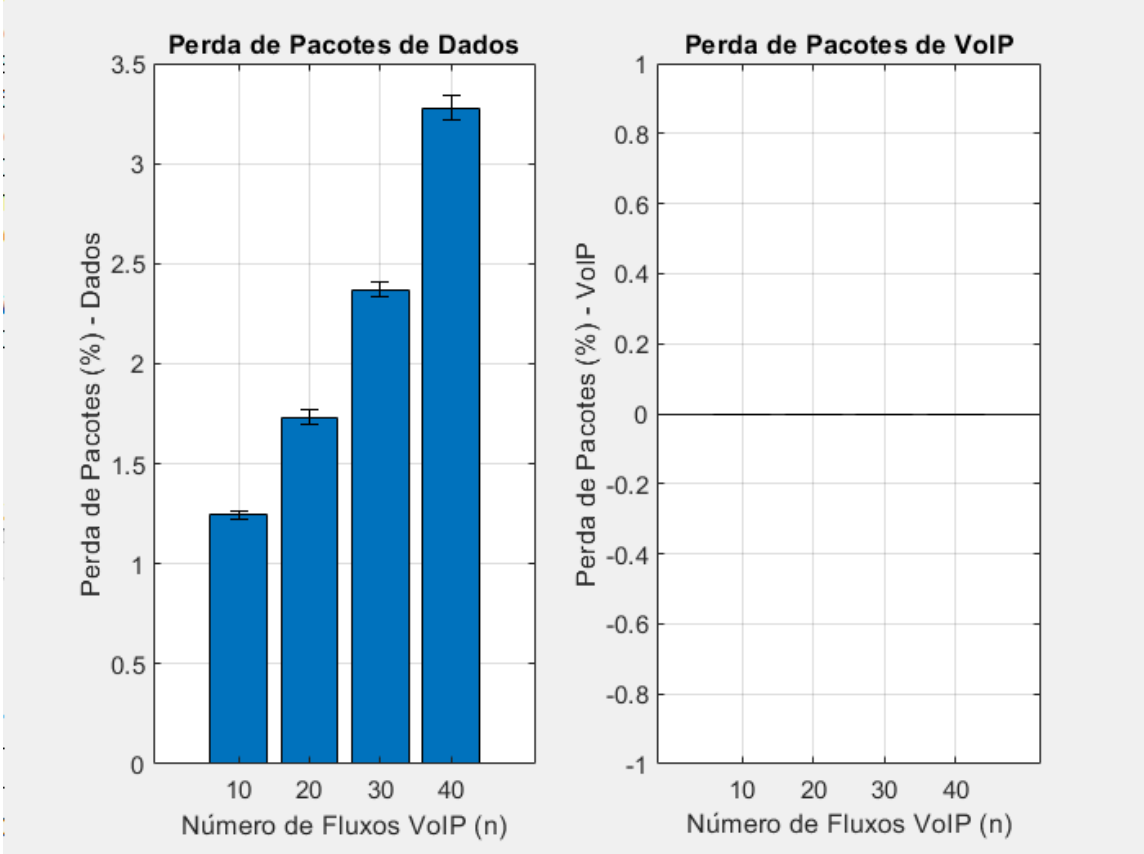

Figure 14: The average packet loss results for Data and VoIP

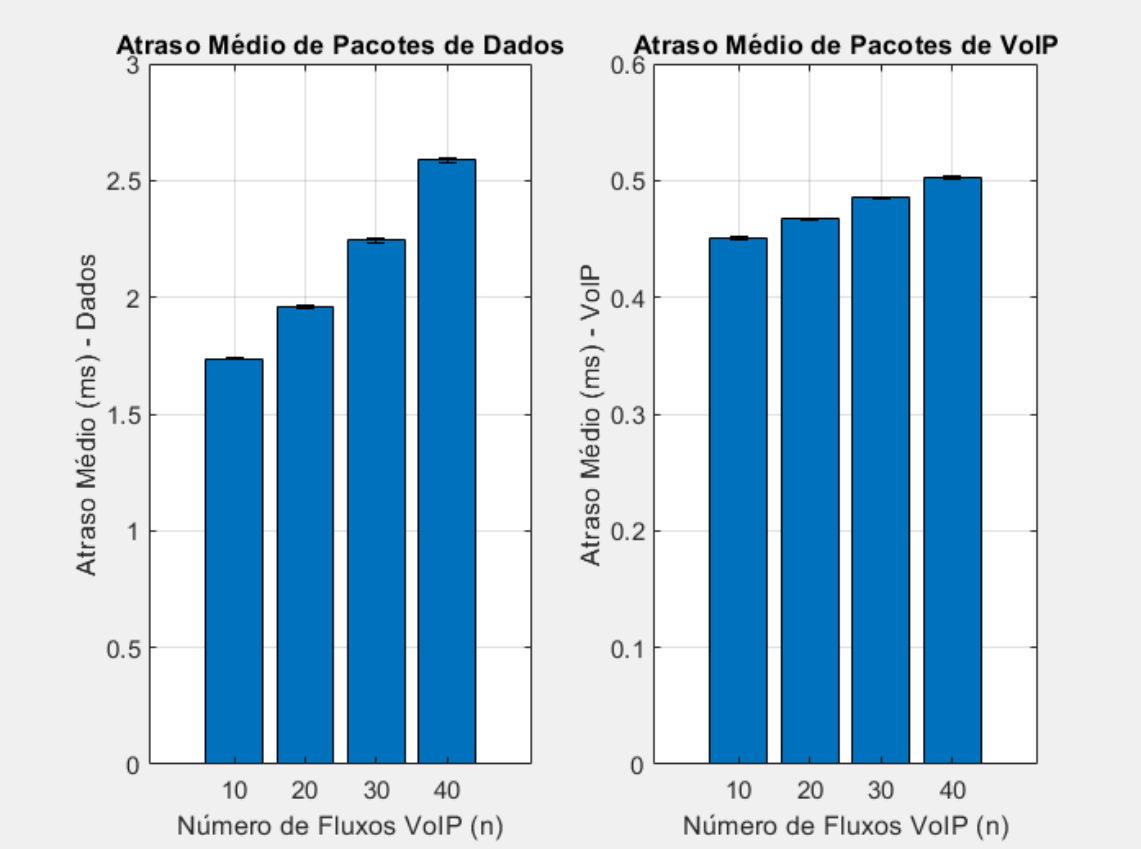

Figura 15: The average packet delay results for Data and VoIP

3.5.2 Conclusions

Packet Loss (Data and VoIP):

Data Packets: The packet loss for data packets has increased significantly compared to both experiment 3.b and 3.d, as shown in the left plot. This is due to the stricter limitation on queue occupation. When the queue exceeds 60% of its capacity, additional data packets are discarded, which leads to a higher packet loss rate as the number of VoIP flows increases.

VoIP Packets: VoIP packet loss remains effectively zero across all scenarios (just as in experiment 3.b and 3.d). Since VoIP packets are prioritized and allowed to enter the queue even when data packets are restricted, they experience no significant packet loss.

Average Delay (Data and VoIP):

Data Packets: The average delay for data packets is somewhat lower than in experiment 3.b, but higher than in 3.d. By limiting the queue occupation to 60%, fewer data packets are stored in the queue, reducing congestion and lowering the overall delay. However, this comes at the cost of increased packet loss, which explains why the delay is not reduced further.

VoIP Packets: The average delay for VoIP packets remains stable, just as in experiment 3.b and 3.d. Since VoIP packets are given priority and are not subject to the queue restriction applied to data packets, they do not experience significant changes in performance.

Comparison to Previous Experiments:

Compared to Experiment 3.b: In experiment 3.b, there was no queue limitation, which led to less data packet loss but higher average delay as the queue could become fully occupied. Here, the stricter queue limitation reduces delay at the expense of a dramatic increase in data packet loss.

Compared to Experiment 3.d: In experiment 3.d, where the queue occupation for data was limited to 90%, there was some balance between delay and packet loss. In this case (with a 60% limit), packet loss increases even further while delay continues to decrease. This shows that stricter queue restrictions significantly affect the balance between these two metrics.

Conclusions:

Trade-off Between Delay and Packet Loss: Reducing the queue threshold for data packets helps decrease average delay at the cost of increased packet loss. The stricter the queue limitation, the more pronounced this trade-off becomes.

VoIP Service Resilience: VoIP packets are consistently prioritized, and their performance is unaffected by the changes in queue restrictions, showing the robustness of the prioritization mechanism for VoIP flows in these experiments.

Optimal Queue Management: For scenarios where delay is critical (such as real-time data applications), applying stricter queue limits can be beneficial, though it comes at the cost of higher packet loss. For applications where packet loss is less acceptable, a more relaxed queue policy (like in experiment 3.d) would be preferable.

Chapter 4

Information and Conclusion

In conclusion, the experiments conducted using Sim3, Sim4, and Sim4A provided valuable insights into the dynamics of network traffic under different prioritization and queue management strategies.

VoIP traffic, being highly sensitive to delay, benefited greatly from prioritization, while data traffic, more tolerant of delays, suffered from increased packet loss under stricter queue restrictions.

The results underline the importance of selecting appropriate traffic management techniques depending on the specific application needs and network goals, especially in networks that must handle mixed traffic types (real-time vs. non-real-time).

These findings contribute to understanding how to optimize network performance to ensure quality of service (QoS) across varying traffic demands and service priorities.

The contributions between each member of the group were equal.