Blocking Performance of Video-Streaming Services

Desempenho e Dimensionamento de Redes
Universidade de Aveiro

Jorge Catarino, Ricardo Azevedo

Version 1.0

Blocking Performance of Video-Streaming Services

DETI
Desempenho e Dimensionamento de Redes
Universidade de Aveiro

(85028) jorge.catarino@ua.pt

(84730) azevedoricardo98@ua.pt

04 de Maio, 2021

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Tarefa 2

Neste capítulo iremos expor os resultados das várias alíneas da Tarefa 2, tal como o raciocínio para os alcançar. Será também apresentado o código desenvolvido e, quando considerado necessário, terá uma reflexão sobre os resultados obtidos.

1.1 Simulador

1.1.1 Método

Para resolver os exercícios propostos na Tarefa 2, foi necessário em primeiro lugar, proceder ao desenvolvimento de um simulador de eventos discretos para um serviço de streaming que utilize um server farm localizado num único datacenter. Este serviço disponibiliza vídeos em dois formatos, HD e 4K. Tem também implementado uma estratégia de *load balancing*, que permite a reserva de recursos para filmes 4K. O objetivo do simulador é então estimar a probabilidade de bloqueio relativo aos dois diferentes tipos de pedido.

A função de simulador tem os seguintes parâmetros de entrada:

- *lambda* - A taxa de pedidos de filmes por hora;
- *p* - Percentagem do número de pedidos que deverão ser 4K;
- *n* - Número de servidores do *server farm*;
- *S* - Capacidade da interface de cada server em mbps;
- *W* - Quantidade de recursos reservados para filmes 4K, em mbps;
- *R* - Número máximo de pedidos para a simulação;

- *fname* - Nome do ficheiro que contém a duração dos eventos, em minutos.

É importante também considerar que o formato HD terá uma taxa de transferência de 5 Mbps e o formato 4K terá uma de 25 Mbps. O capacidade total da conecção da *server farm* é dada por $C = n \times S$.

O primeiro passo no desenvolvimento de simulador foi definir os eventos que seriam considerados. Foi inicialmente sugerido que se considerasse três eventos: ARRIVAL, DEPARTURE_HD, DEPARTURE_4K.

No entanto, durante o desenvolvimento reparou-se que havia uma simplificação que permitiria só considerar o primeiro evento, ARRIVAL. Esta simplificação consiste no facto de ser possível identificar o tipo de pedido feito, tal como o servidor que fica encarregue deste com apenas um número inteiro no intervalo de $[1, 2n]$, onde os elementos menores que *n* correspondem a pedidos HD e os maiores que *n* a pedidos 4K. Sacrificando alguma da legibilidade do código, obtém-se desta forma ganhos a nível de performance e memória, dado que deixa de ser necessário ter uma terceira coluna na *Event List*.

Então a lista de eventos final fica:

- ARRIVAL - Instante de tempo de um pedido.

As variáveis de estado e os contadores estatísticos considerados foram os mesmos especificados no Apêndice C do Guião.

As variáveis de estado são:

- STATE(i) - Taxa de transferência total dos filmes a ser transmitidos pelo servidor *i*;
- STATE_HD - Taxa de transferência total dos filmes HD a ser transmitidos.

Já os contadores estocásticos considerados são:

- NARRIVALS - Número total de pedidos de filmes até ao instante de tempo atual;
- REQUESTS_HD - Número de pedidos de filmes HD até ao instante de tempo atual;
- REQUESTS_4K - Número de pedidos de filmes 4K até ao instante de tempo atual;
- BLOCKED_HD - Número de pedidos de filmes HD bloqueados até ao instante de tempo atual;
- BLOCKED_4K - Número de pedidos de filmes 4K bloqueados até ao instante de tempo atual.

Tendo estes parâmetros iniciais definidos, passa-se a explicar a lógica da solução.

O simulador está em funcionamento até NARRIVALS atingir o valor de R definido como parâmetro de entrada.

Quando o evento atual corresponde a um ARRIVAL, o simulador seleciona aleatoriamente o tipo de pedido a que este corresponde, tendo em conta a probabilidade *p* definida. No caso de ser um pedido 4K, o contador REQUESTS_4K é incrementado. Se o servidor com menor carga tiver no mínimo 25 Mbps de recursos disponíveis, a variável STATE, onde são armazenados as ocupações dos vários servidores, é atualizada, tal como a EventList. Nesta última é adicionada uma nova entrada com $i + n$, onde *i* corresponde ao índice do servidor menos ocupado (ver simplificação explicada acima) e com o instante de tempo de partida deste pedido. Na eventualidade de o servidor com menos carga não ser capaz de receber o pedido, este é então bloqueado, sendo então incrementado o contador BLOCKED_4K.

No caso dos pedidos HD, a lógica é relativamente semelhante, com a exceção que não só o servidor com menos carga deve ser capaz de lidar com o pedido, como temos de garantir que não estamos a utilizar recursos W que foram reservados para transmissões 4K.

Já quando o evento corresponde a uma partida, o simulador distingue o tipo de transmissão baseado no método já explicado acima. Se $event > n$, é porque a transmissão foi 4K, caso contrário é porque foi uma transmissão HD. O índice do servidor a ser atualizado obtém-se diretamente no caso da transmissão ser HD, e subtraindo n no caso da transmissão ser 4K.

Por fim, para calcular a probabilidade de bloqueio para um pedido de tipo X, podemos usar a fórmula:

$$b\_X = (\frac{BLOCKED\_X}{REQUESTS\_X}) \times 100;$$

O código pode ser consultado na subssecção 1.1.2.

1.1.2 Código

```
1  function [b_4k b_hd] = simulator2(lambda, p, n, S, W,
       R, fname)
2      % Parameters
3      % lambda - movies request rate (req/hour)
4      % p - percentage of request for 4k movies (%)
5      % n - number of servers
6      % S - interface capacity of each server (mbps)
7      % W - resourse reservation for 4K movies (mbps)
8      % R - number of movie requests to stop simulation
```

```
9       % fname - file name with the duration (in minutes)
            of the items
10
11      invlambda=60/lambda;      %average time between
            requests (in minutes)
12      invmiu= load(fname);      %duration (in minutes) of
            each movie
13      Nmovies= length(invmiu); %number of movies
14      C = n * S;                %Internet Connection
            Capability
15      M_4k = 25;                %throughoutput of a 4k
            movie
16      M_hd = 5;                 %throughoutput of a hd
            movie
17
18      %Events definition:
19      ARRIVAL = 0;              %movie request
20      %State variables initialization:
21      STATE = zeros(1, n);
22      STATE_HD = 0;
23      %Statistical counters initialization:
24      NARRIVALS = 0;
25      REQUESTS_HD = 0;
26      REQUESTS_4K = 0;
27      BLOCKED_HD = 0;
28      BLOCKED_4K = 0;
29      %Simulation Clock and initial List of Events:
30      Clock= 0;
31      EventList= [ARRIVAL exprnd(invlambda)];
32
33      while NARRIVALS < R
34          event= EventList(1,1);
35          Clock= EventList(1,2);
36          EventList(1,:)= [];
37
38          if event == ARRIVAL
39              EventList= [EventList; ARRIVAL Clock+
                    exprnd(invlambda)];
40              NARRIVALS= NARRIVALS+1;
41              [least_loaded_c, least_loaded_i] = min(
                    STATE);
42
43              % Find which request was made
44              % Choose the right server according
45              % to the load rules.
46              if rand <= p
47                  REQUESTS_4K = REQUESTS_4K + 1;
48                  if S - least_loaded_c >= 25
49                      STATE(least_loaded_i) =
                            least_loaded_c + M_4k;
```

```
50                      EventList= [EventList;
                            least_loaded_i+n Clock+invmiu(
                            randi(Nmovies))];
51                  else
52                      BLOCKED_4K = BLOCKED_4K + 1;
53                  end
54              else
55                  REQUESTS_HD = REQUESTS_HD + 1;
56                  if (S - least_loaded_c >= 5) && (
                        STATE_HD + M_hd <= C - W)
57                      STATE(least_loaded_i) =
                            least_loaded_c + M_hd;
58                      STATE_HD = STATE_HD + M_hd;
59                      EventList= [EventList;
                            least_loaded_i Clock+invmiu(
                            randi(Nmovies))];
60                  else
61                      BLOCKED_HD = BLOCKED_HD + 1;
62                  end
63              end
64          else
65              if event > n
66                  STATE(event-n) = STATE(event-n)-M_4k;
67              else
68                  STATE(event) = STATE(event)-M_hd;
69                  STATE_HD = STATE_HD-M_hd;
70              end
71          end
72          EventList= sortrows(EventList,2);
73      end
74      b_hd= 100*BLOCKED_HD/REQUESTS_HD;     % HD blocking
            probability in %
75      b_4k = 100*BLOCKED_4K/REQUESTS_4K;    % 4K blocking
            probability in %
76  end
```

1.2 Código Auxiliar

1.2.1 Função *runSimulator2*

Com o objectivo de reduzir a repetição de código no desenvolvimento desta tarefa, foi criada uma função que permite correr o simulador N vezes, para cada valor de lambda passado por argumento, calculando a médias e os erros associados a cada resultado.

O código desta função encontra-se reproduzido abaixo.

```
1  function [medias_b_4k, terms_b_4k, medias_b_hd,
       terms_b_hd] = runSimulator2(N_times, alfa,
       lambda_values, p, n, S, W, R, fname)
2      sz = length(lambda_values);
3      medias_b_4k = zeros(1, sz);
4      medias_b_hd = zeros(1, sz);
5      terms_b_4k = zeros(1, sz);
6      terms_b_hd = zeros(1, sz);
7
8      for z = 1:sz
9          results_b_4k = zeros(1, N_times);
10         results_b_hd = zeros(1, N_times);
11
12         for it = 1:N_times
13             [results_b_4k(it), results_b_hd(it)] =
                   simulator2(lambda_values(z), p, n, S, W
                   , R, fname);
14         end
15
16         medias_b_4k(z) = mean(results_b_4k);
17         medias_b_hd(z) = mean(results_b_hd);
18
19         terms_b_4k(z) = norminv(1-alfa/2)*sqrt(var(
               results_b_4k)/N_times);
20         terms_b_hd(z) = norminv(1-alfa/2)*sqrt(var(
               results_b_hd)/N_times);
21     end
22 end
```

1.2.2 Valores Padrão

As primeiras alíneas usam um conjunto de valores que foram definidos "à cabeça" do script. De forma a facilitar a eventual compreensão do código que será reproduzido nas próximas secções, este cabeçalho é então aqui reproduzido.

```
%Task2
% General Values
lambda_values = [100, 120, 140, 160, 180, 200];
fname = "movies.txt";

% Configuration 1
c1_n = 10;
c1_S = 100;

% Configuration 2
c2_n = 4;
c2_S = 250;

% Configuration 3
c3_n = 1;
c3_S = 1000;
```

1.3 Alínea a)

1.3.1 Método e Resultados

Utilizando então as funções definidas anteriormente com a configuração 1 e tendo $W = 0$, $p = 0.2$ e $R = 10000$, obtemos então o resultado da Figura 1.1

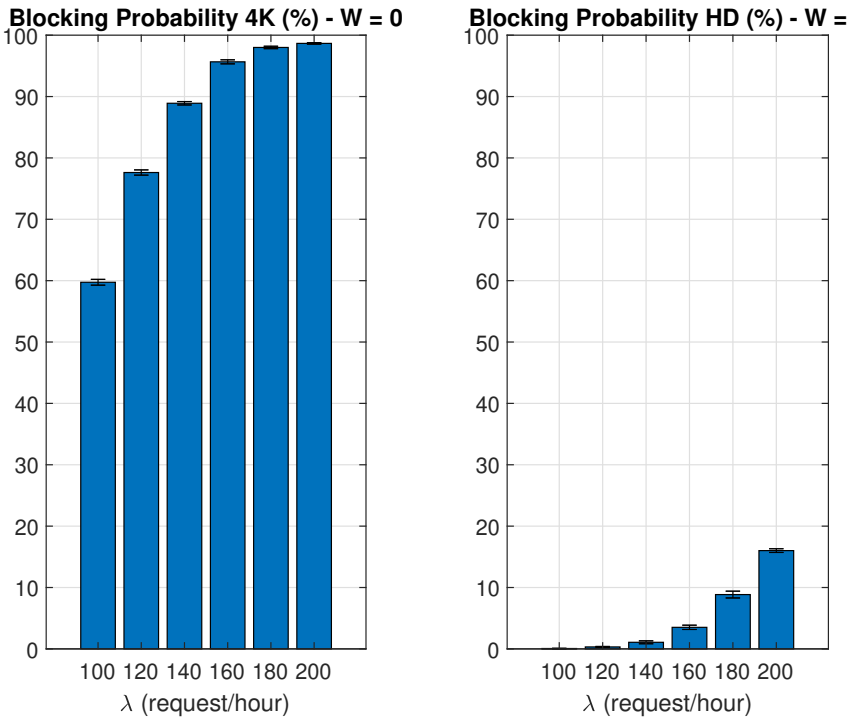

Figura 1.1: Resultado da alínea a)

1.3.2 Código

```
%%
%2a

N_times = 10;
alfa = 0.1;
W = 0;
p = 0.2;
R = 10000;

[medias_b_4k_c1, terms_b_4k_c1, medias_b_hd_c1,
    terms_b_hd_c1] = ...
    runSimulator2(N_times, alfa, lambda_values, p,
        c1_n, c1_S, W, R, fname);

% 4k Blocking Probability
figure(1)
tiledlayout(1,2)

nexttile;
bar(lambda_values,medias_b_4k_c1)
title("Blocking Probability 4K (%) - W = 0")
xlabel('\lambda (request/hour)')
ylim([0 100])
grid on

hold on
er = errorbar(lambda_values, medias_b_4k_c1,
    terms_b_4k_c1);
er.Color = [0 0 0];
er.LineStyle = 'none';
hold off

% HD Blocking Probability
nexttile;
bar(lambda_values,medias_b_hd_c1)
title("Blocking Probability HD (%) - W = 0")
xlabel('\lambda (request/hour)')
ylim([0 100])
grid on

hold on
er = errorbar(lambda_values, medias_b_hd_c1,
    terms_b_hd_c1);
er.Color = [0 0 0];
er.LineStyle = 'none';
hold off
```

1.3.3 Discussão

Analisando os resultados obtidos, é possível observar que à medida que taxa de chegada λ aumenta, a probabilidade bloqueio em ambos os formatos, HD ou 4K, também aumenta. Com o aumento da λ, o tempo médio entre pedidos diminui consideravelmente o que faz com que os recursos fiquem ocupados mais rapidamente, dado que o servidor acaba por estar a servir mais filmes em simultâneo. Os pedidos do tipo 4K são afetados desproporcionalmente neste exemplo dado que ocorrem com menos frequência, necessitam de mais recursos, e como $W = 0$, não existem recursos disponíveis apenas para pedidos deste tipo.

Podemos também afirmar que o critério de paragem para este exercício está bem definido, dado que os intervalos de confiança obtidos são relativamente reduzidos, o que significa que existe pouca incerteza nos resultados.

1.4 Alínea b)

1.4.1 Método e Resultados

Utilizando então as funções definidas anteriormente com as configurações 1,2 e 3 e tendo $W = 0$, $p = 0.2$ e $R = 10000$, obtemos então o resultado da Figura 1.2

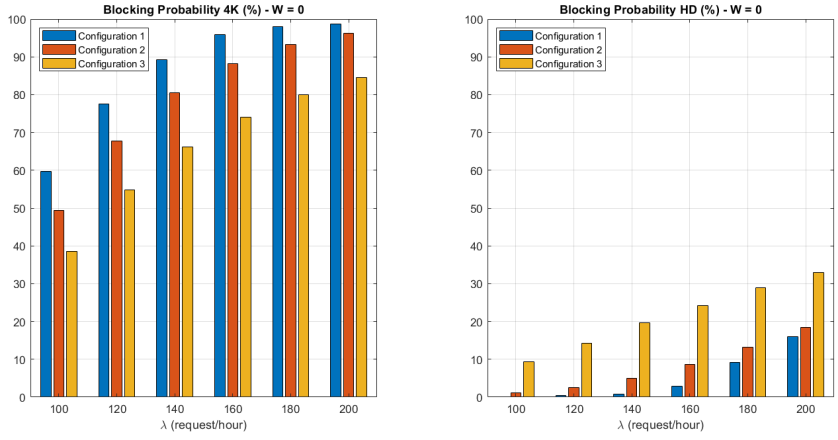

Figura 1.2: Resultado da alínea b)

1.4.2 Código

```
%%
%2b

N_times = 10;
alfa = 0.1;
W = 0;
p = 0.2;
R = 10000;

[medias_b_4k_c2, terms_b_4k_c2, medias_b_hd_c2,
    terms_b_hd_c2] = ...
    runSimulator2(N_times, alfa, lambda_values, p,
        c2_n, c2_S, W, R, fname);

[medias_b_4k_c3, terms_b_4k_c3, medias_b_hd_c3,
    terms_b_hd_c3] = ...
    runSimulator2(N_times, alfa, lambda_values, p,
        c3_n, c3_S, W, R, fname);

figure(2);

tiledlayout(1,2)

nexttile;
bar(lambda_values, [medias_b_4k_c1(:) medias_b_4k_c2
    (:) medias_b_4k_c3(:)])
legend("Configuration 1", "Configuration 2", "
    Configuration 3", "Location" ,"northwest")
title("Blocking Probability 4K (%) - W = 0")
xlabel('\lambda (request/hour)')
ylim([0 100])
grid on

nexttile;
bar(lambda_values, [medias_b_hd_c1(:) medias_b_hd_c2
    (:) medias_b_hd_c3(:)])
legend("Configuration 1", "Configuration 2", "
    Configuration 3", "Location" ,"northwest")
title("Blocking Probability HD (%) - W = 0")
xlabel('\lambda (request/hour)')
ylim([0 100])
grid on
```

1.4.3 Discussão

É possível observar que as configurações 2 e 3 tem probabilidade de bloqueio para o tipo 4K menores que a configuração 1, enquanto a probabilidade de bloqueio para o tipo HD é crescente.

A diferença entre as três configurações é entre o número de servidores disponíveis e a capacidade de interface de cada um destes, sendo que o total de capacidade da *server farm* é $C = 1000$ em todas as configurações.

A partir da análise podemos concluir que ter servidores com menos capacidade individual acaba por afectar adversamente o tipo 4K. Como é sempre escolhido o servidor menos ocupado no momento, é possível que vários servidores tenham só capacidade suficiente para um filme HD (5 Mbps). Já no caso de termos menos servidores, mas com mais capacidade, a chance destes terem apenas espaço para um filme HD é mais reduzida, o que resulta em menor probabilidade de bloqueio para filme 4K. No entanto, como os filmes 4K utilizam mais recursos, temos como consequência o aumento da probabilidade de bloqueio para filmes HD nestas configurações com servidores com maior interface.

1.5 Alínea c)

1.5.1 Método e Resultados

Utilizando então as funções definidas anteriormente com as configurações 1,2 e 3 e tendo $W = 400$, $p = 0.2$ e $R = 10000$, obtemos então o resultado da Figura 1.3

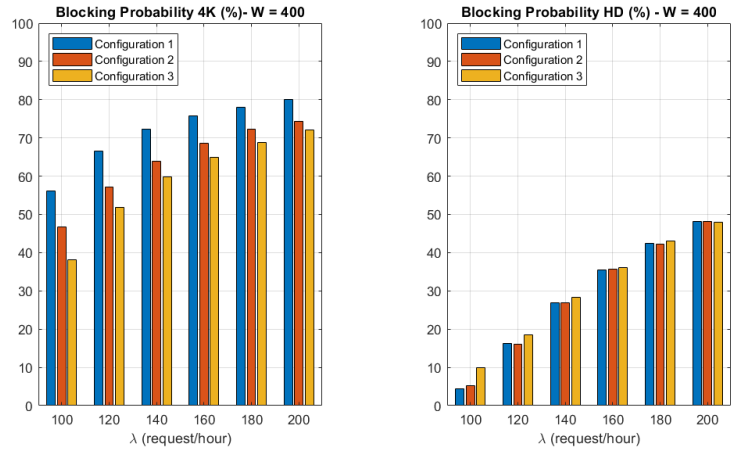

Figura 1.3: Resultado da alínea c)

1.5.2 Código

```
%%
%2c

N_times = 10;
alfa = 0.1;
W = 400;
p = 0.2;
R = 10000;

[medias_b_4k_c1, terms_b_4k_c1, medias_b_hd_c1,
    terms_b_hd_c1] = ...
    runSimulator2(N_times, alfa, lambda_values, p,
        c1_n, c1_S, W, R, fname);

[medias_b_4k_c2, terms_b_4k_c2, medias_b_hd_c2,
    terms_b_hd_c2] = ...
    runSimulator2(N_times, alfa, lambda_values, p,
        c2_n, c2_S, W, R, fname);

[medias_b_4k_c3, terms_b_4k_c3, medias_b_hd_c3,
    terms_b_hd_c3] = ...
    runSimulator2(N_times, alfa, lambda_values, p,
        c3_n, c3_S, W, R, fname);

figure(3);

tiledlayout(1,2)

nexttile;
bar(lambda_values, [medias_b_4k_c1(:) medias_b_4k_c2
    (:) medias_b_4k_c3(:)])
legend("Configuration 1", "Configuration 2", "
    Configuration 3", "Location" ,"northwest")
title("Blocking Probability 4K (%)- W = 400")
xlabel('\lambda (request/hour)')
ylim([0 100])
grid on

nexttile;
bar(lambda_values, [medias_b_hd_c1(:) medias_b_hd_c2
    (:) medias_b_hd_c3(:)])
legend("Configuration 1", "Configuration 2", "
    Configuration 3", "Location" ,"northwest")
title("Blocking Probability HD (%) - W = 400")
xlabel('\lambda (request/hour)')
ylim([0 100])
grid on
```

1.5.3 Discussão

Se compararmos os resultados desta alínea com os resultados obtidos anteriormente, é possível observar que os a probabilidade de bloqueio para filmes no formato 4K diminuiu consideravelmente em todas as configurações, especialmente para valores de λ superiores, enquanto a probabilidade de bloqueio para os filmes no formato HD aumentou, principalmente para as configurações 1 e 2. Isto deve-se a estratégia de *load balancing* implementada. No entanto, as probabilidades de bloqueio para 4K continuam a ser superiores as dos filmes HD para todas as configurações, o que significa que este pode não ser o valor de W ideal para estas configurações, apesar de os resultados serem mais balançados.

Com isto, é possível perceber que haverá sempre um compromisso a ser considerado. A partir da configuração do W podemos diminuir a probabilidade de bloqueio para filmes 4K, no entanto ao fazer isto iremos sempre aumentar a probabilidade de bloqueio para filmes HD.

1.6 Alínea d)

1.6.1 Método e Resultados

Utilizando então as funções definidas anteriormente com as configurações 1,2 e 3 e tendo $W = 600$, $p = 0.2$ e $R = 10000$, obtemos então o resultado da Figura 1.4

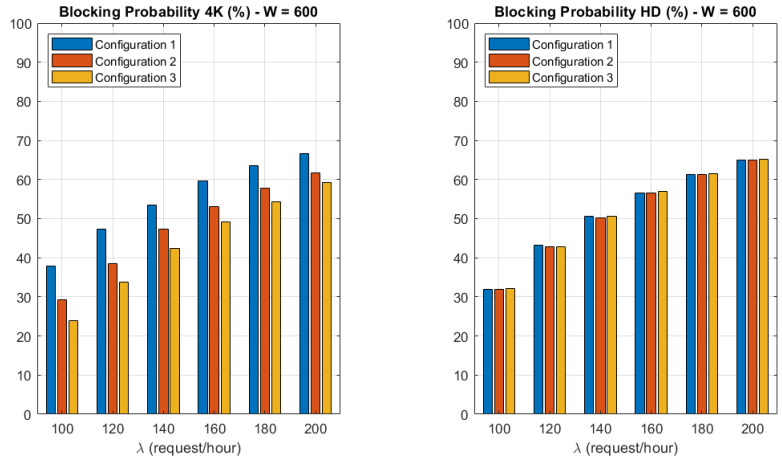

Figura 1.4: Resultado da alínea d)

1.6.2 Código

```
%%
%2d

N_times = 10;
alfa = 0.1;
W = 600;
p = 0.2;
R = 10000;

[medias_b_4k_c1, terms_b_4k_c1, medias_b_hd_c1,
    terms_b_hd_c1] = ...
    runSimulator2(N_times, alfa, lambda_values, p,
        c1_n, c1_S, W, R, fname);

[medias_b_4k_c2, terms_b_4k_c2, medias_b_hd_c2,
    terms_b_hd_c2] = ...
    runSimulator2(N_times, alfa, lambda_values, p,
        c2_n, c2_S, W, R, fname);

[medias_b_4k_c3, terms_b_4k_c3, medias_b_hd_c3,
    terms_b_hd_c3] = ...
    runSimulator2(N_times, alfa, lambda_values, p,
        c3_n, c3_S, W, R, fname);

figure(4);

tiledlayout(1,2)

nexttile;
bar(lambda_values, [medias_b_4k_c1(:) medias_b_4k_c2
    (:) medias_b_4k_c3(:)])
legend("Configuration 1", "Configuration 2", "
    Configuration 3", "Location" ,"northwest")
title("Blocking Probability 4K (%) - W = 600")
xlabel('\lambda (request/hour)')
ylim([0 100])
grid on

nexttile;
bar(lambda_values, [medias_b_hd_c1(:) medias_b_hd_c2
    (:) medias_b_hd_c3(:)])
legend("Configuration 1", "Configuration 2", "
    Configuration 3", "Location" ,"northwest")
title("Blocking Probability HD (%) - W = 600")
xlabel('\lambda (request/hour)')
ylim([0 100])
grid on
```

1.6.3 Discussão

Ao comparar os resultados desta alínea aos obtidos anteriormente, é possível observar que as probabilidades de bloqueio para os dois tipos de filme estão mais equilibradas, o que pode significar que o W utilizado pode ser o ideal para esta configuração, se for pretendido ter igual performance para dois tipos.

O formato 4K continua a ter pior performance para a configuração 1, enquanto o formato HD tem uma performance semelhante independentemente da configuração, variando só com o número de pedidos por hora.

1.7 Alínea e)

1.7.1 Método e Resultados

A configuração base utilizada neste exercício foi $R = 100000$, $S = 10000$ e com $lambda = \frac{R}{24}$, para obtermos uma média de um pedido por dia.

Para determinar o número mínimo de servidores necessários e o valor de reserva W para cumprir com os requisitos necessários do problema fomos experimentalmente alterando os valores, primeiro de n e depois de W, anotando as diferenças que as alterações provocavam nos resultados.

N	W	BK 4k	BK 4K FAILED	BK HD	BK HD FAILED	
4	20000	4.31e+01 +- 1.31e-01	4.35e+01 +- 1.93e-01	1.23e+01 +- 1.23e-01	5.53e+01 +- 9.06e-02	Not Passable
6	20000	5.67e-01 +- 1.02e-01	2.37e+01 +- 2.30e-01	2.24e-03 +- 2.50e-03	3.28e-01 +- 2.67e-02	Not Passable
6	35000	5.68e-01 +- 1.10e-01	3.49e+00 +- 3.01e-01	6.58e-04 +- 8.69e-04	3.34e+01 +- 1.35e-01	Not Passable
7	34950	0	4.62e-01 +- 9.06e-02	0	2.63e-04 +- 2.88e-04	Passable
7	35200	0	7.19e-01 +- 1.41e-01	0	2.63e-04 +- 2.89e-04	Passable
7	34850	0	6.39e-01 +- 1.18e-01	0	1.19e-03 +- 1.09e-03	Passable
7	36950	0	4.70e-01 +- 8.25e-02	0	5.52e-01 +- 8.13e-02	Passable

Tabela 1.1: Resultados das simulações da alínea e)

Com base neste método obtemos então que o valor mínimo de servidores necessários para cumprir com os requisitos impostos é:

$$n = 7;$$

Com o número mínimo de servidores definido, foi necessário estabelecer um valor de W que garantisse que a pior probabilidade de bloqueio fosse menor que 0.1%, quando todos os servidores estiverem em funcionamento, e menor que 1% quando um dos servidores falha. Foi procurado estabelecer um equilíbrio entre as taxas de bloqueio dos dois tipos de formato, de forma a que ambos tenham uma performance semelhante em qualquer uma das situações.

Com isto em mente, foi então obtido:

$$W = 36950;$$

Um exemplo de dos resultados que se podem obter quando se corre o simulador com estes parâmetros é então:

$$Blocking\_Probability\_4K(\%) = 0;$$

$$Blocking\_Probability\_4K\_Failed\_Server(\%) = 4.70e-01 \pm 8.25e-02;$$

$$Blocking\_Probability\_HD(\%) = 0$$

$$Blocking\_Probability\_HD\_Failed\_Server(\%) = 5.52e-01 \pm 8.13e-02$$

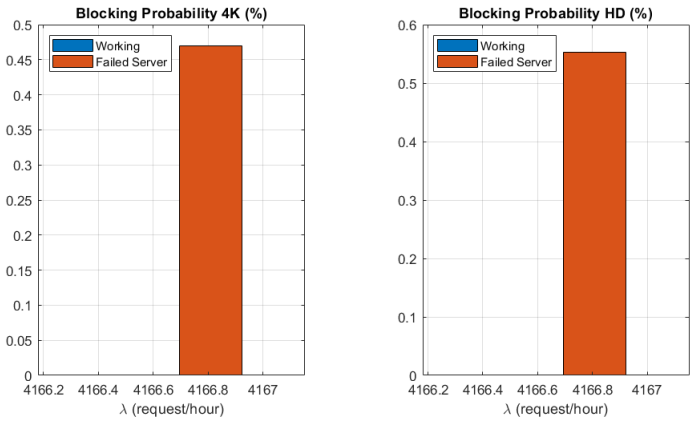

Figura 1.5: Resultado da alínea e)

Como se pode observar, a performance cumpre com os requisitos impostos.

1.7.2 Código

```
%%
%2e

% Configuration E
e_n = 7;
e_S = 10000;

N_times = 10;
alfa = 0.1;
W = 36950;
p = 0.24;
R = 100000;
lambda = R/24;

[medias_b_4k_e, terms_b_4k_e, medias_b_hd_e,
    terms_b_hd_e] = ...
     runSimulator2(N_times, alfa, lambda, p, e_n, e_S,
         W, R, fname);

[medias_b_4k_e_fail, terms_b_4k_e_fail,
    medias_b_hd_e_fail, terms_b_hd_e_fail] = ...
     runSimulator2(N_times, alfa, lambda, p, e_n-1, e_S
         , W, R, fname);

fprintf('Blocking probability 4K (%%) = %.2e +- %.2e\n
    ', medias_b_4k_e, terms_b_4k_e);
fprintf('Blocking probability 4K Failed Server (%%) =
    %.2e +- %.2e\n', medias_b_4k_e_fail,
    terms_b_4k_e_fail);
fprintf('Blocking probability HD (%%) = %.2e +- %.2e\n
    ', medias_b_hd_e, terms_b_hd_e);
fprintf('Blocking probability HD Failed Server (%%) =
    %.2e +- %.2e\n', medias_b_hd_e_fail,
    terms_b_hd_e_fail);
```

Capítulo 2

Tarefa 3

Neste capítulo iremos expor os resultados das várias alíneas da Tarefa 3, tal como o raciocínio para os alcançar. Será também apresentado o código desenvolvido e, quando considerado necessário, terá uma reflexão sobre os resultados obtidos.

2.1 Alínea a)

2.1.1 Método e Resultados

Neste exercício devemos encontrar um conjunto de AS, onde devem ser conectadas as *server farms* do serviço. Deve haver a garantia que existe pelo menos um caminho com menos de um AS intermédio entre cada um dos AS e uma *server farm*.

Este problema, por ser um problema de otimização, com o objetivo de minimizar uma dada quantia definida por uma função objetivo, pode ser modelado usando o modelo de programação inteira (ILP).

Para tal, foi então construído um programa que escreve num ficheiro o problema definido no formato LP.

Genéricamente, o problema pode ser definido a partir de um objetivo e duas restrições:

(1) Minimize $\sum_{i=n1,n2,...}(c_i x_i)$

Subject To

(2) $\sum_{i \in I(j)} x_i \geq 1, j = n_1, n_2, ...$

(3) $x_i \in \{0,1\}, i = n_1, n_2, ...$

Onde (1) é a função objetivo, que corresponde a minimização dos custos totais da conexão à Internet por partes dos AS com as *server farms*. Já a restrição (2) garante que cada AS j tem pelo menos uma *server farm* conectada a um AS $i \in I(j)$. A restrição (3) define todas as variáveis como binárias.

O programa, exposto na sub-secção 2.1.2, começa então por escrever no ficheiro a função objetivo (1) (linhas 75-82). Cada AS existente é representado por uma variável x_i e multiplicado pelo seu custo c_i correspondente, dependendo se é um AS Tier-2 ou Tier-3. Nesta implementação é utilizado uma variável auxiliar c_aux, que se mantém a 12 quando $i < 16$, sendo este valor correspondente ao custo dos AS Tier-2, e é trocado para 8 quando $i \geq 16$, que corresponde ao custo dos AS Tier-3.

Das linhas 84 a 97 encontra-se a implementação da restrição (2). Com o auxílio do grafo G, que denota as conexões entre AS e a função *shortestpath* pode, então ser calculado os caminhos mais curtos entre 2 nós. Nesta implementação é retirado ao caminho mais curto o primeiro e o último elemento. Se depois desta remoção, a lista de nós estiver vazia ou tiver um elemento, significa que o caminho cumpre os nossos requisitos e é então o AS_i adicionado a uma lista, *l_j_aux*.

Esta lista corresponde portanto a lista de nós i para os quais um nó j possui um caminho com 1 ou menos nós intermédios. Após estes caminhos serem todos calculados, os conteúdos da lista *l_j_aux* são então escritos no ficheiro LP, no formato necessário.

Por fim, nas linhas 99 a 102 encontra-se a implementação da restrição (3), que define os AS 6 a *n_nodes*, que neste caso corresponde a 40, como variáveis binárias, escrevendo então estes valores no ficheiro.

O ficheiro gerado por este programa é então:

```
1 Minimize
2  + 12.000000 x6 + 12.000000 x7 + 12.000000 x8 +
       12.000000 x9 + 12.000000 x10 + 12.000000 x11 +
       12.000000 x12 + 12.000000 x13 + 12.000000 x14 +
       12.000000 x15 + 8.000000 x16 + 8.000000 x17 +
       8.000000 x18 + 8.000000 x19 + 8.000000 x20 +
       8.000000 x21 + 8.000000 x22 + 8.000000 x23 +
       8.000000 x24 + 8.000000 x25 + 8.000000 x26 +
       8.000000 x27 + 8.000000 x28 + 8.000000 x29 +
       8.000000 x30 + 8.000000 x31 + 8.000000 x32 +
       8.000000 x33 + 8.000000 x34 + 8.000000 x35 +
       8.000000 x36 + 8.000000 x37 + 8.000000 x38 +
       8.000000 x39 + 8.000000 x40
3 Subject To
4  + x6 + x7 + x14 + x15 + x16 + x17 + x18 + x19 + x20
       >= 1
5  + x6 + x7 + x8 + x16 + x17 + x18 + x19 + x20 + x21 >=
```

```
      1
 + x7 + x8 + x9 + x10 + x20 + x21 + x22 + x23 + x24 +
    x25 >= 1
 + x8 + x9 + x10 + x11 + x21 + x22 + x23 + x24 + x25 +
    x26 + x27 >= 1
 + x8 + x9 + x10 + x11 + x12 + x13 + x22 + x23 + x24 +
    x25 + x26 + x27 + x28 + x29 + x30 >= 1
 + x9 + x10 + x11 + x12 + x13 + x26 + x27 + x28 + x29
    + x30 >= 1
 + x10 + x11 + x12 + x13 + x14 + x30 + x31 + x32 >= 1
 + x10 + x11 + x12 + x13 + x14 + x33 + x34 + x35 + x36
    + x37 + x38 >= 1
 + x6 + x12 + x13 + x14 + x15 + x33 + x34 + x35 + x36
    + x37 + x38 >= 1
 + x6 + x14 + x15 + x16 + x39 + x40 >= 1
 + x6 + x7 + x15 + x16 + x17 + x18 + x19 + x39 + x40
    >= 1
 + x6 + x7 + x16 + x17 + x18 + x19 >= 1
 + x6 + x7 + x16 + x17 + x18 + x19 >= 1
 + x6 + x7 + x16 + x17 + x18 + x19 + x20 >= 1
 + x6 + x7 + x8 + x19 + x20 + x21 >= 1
 + x7 + x8 + x9 + x20 + x21 + x22 >= 1
 + x8 + x9 + x10 + x21 + x22 + x23 + x24 + x25 >= 1
 + x8 + x9 + x10 + x22 + x23 + x24 + x25 >= 1
 + x8 + x9 + x10 + x22 + x23 + x24 + x25 >= 1
 + x8 + x9 + x10 + x22 + x23 + x24 + x25 >= 1
 + x9 + x10 + x11 + x26 + x27 >= 1
 + x9 + x10 + x11 + x26 + x27 + x28 + x29 + x30 >= 1
 + x10 + x11 + x27 + x28 + x29 + x30 >= 1
 + x10 + x11 + x27 + x28 + x29 + x30 >= 1
 + x10 + x11 + x12 + x27 + x28 + x29 + x30 + x31 + x32
    >= 1
 + x12 + x30 + x31 + x32 >= 1
 + x12 + x30 + x31 + x32 >= 1
 + x13 + x14 + x33 + x34 + x35 >= 1
 + x13 + x14 + x33 + x34 + x35 >= 1
 + x13 + x14 + x33 + x34 + x35 >= 1
 + x13 + x14 + x36 + x37 + x38 >= 1
 + x13 + x14 + x36 + x37 + x38 >= 1
 + x13 + x14 + x36 + x37 + x38 >= 1
 + x15 + x16 + x39 + x40 >= 1
 + x15 + x16 + x39 + x40 >= 1
Binary
 x6
 x7
 x8
 x9
 x10
 x11
 x12
```

```
 x13
 x14
 x15
 x16
 x17
 x18
 x19
 x20
 x21
 x22
 x23
 x24
 x25
 x26
 x27
 x28
 x29
 x30
 x31
 x32
 x33
 x34
 x35
 x36
 x37
 x38
 x39
 x40
End
```

Após ser executado num simulador Gurobi, disponível online, obtém-se então os seguintes resultados

```
Optimal solution found (tolerance 1.00e-04)
Best objective 4.800000000000e+01, best bound
   4.800000000000e+01, gap 0.0000%


********** Begin .sol file *************

# Objective value = 48
x6 0
x7 0
x8 0
x9 1
x10 0
x11 0
x12 0
x13 1
x14 0
```

```
x15 0
x16 1
x17 0
x18 0
x19 0
x20 0
x21 1
x22 0
x23 0
x24 0
x25 0
x26 0
x27 0
x28 0
x29 0
x30 1
x31 0
x32 0
x33 0
x34 0
x35 0
x36 0
x37 0
x38 0
x39 0
x40 0

********** End .sol file ***************
```

A partir da análise dos resultados podemos concluir que a solução terá no mínimo o **custo de 48**. Para ligar todos os AS de forma a atingir este custo, serão então necessários **5 *server farms***, **ligadas aos ASs 9, 13, 16, 21 e 30**.

2.1.2 Código

```
%Task3
% General Values
G= [ 1  2
     1  3
     1  4
     1  5
     1  6
     1 14
     1 15
     2  3
     2  4
     2  5
     2  7
     2  8
     3  4
     3  5
     3  8
     3  9
     3 10
     4  5
     4 10
     4 11
     4 12
     4 13
     5 12
     5 13
     5 14
     6  7
     6 16
     6 17
     6 18
     6 19
     7 19
     7 20
     8  9
     8 21
     8 22
     9 10
     9 22
     9 23
     9 24
     9 25
     10 11
     10 26
     10 27
     11 27
     11 28
```

```
     11 29
     11 30
     12 30
     12 31
     12 32
     13 14
     13 33
     13 34
     13 35
     14 36
     14 37
     14 38
     15 16
     15 39
     15 40
     20 21];

G_graph = graph(G(:, 1), G(:,2));

%%
% 3a

c = [12, 8];
c_aux = c(1);
n_nodes = numnodes(G_graph);
fid= fopen('res3a.lp','wt');

% Minimize (1)
fprintf(fid, 'Minimize\n');
for i=6:n_nodes
    if i >= 16
        c_aux = c(2);
    end
    fprintf(fid, ' + %f x%d', c_aux, i);
end
fprintf(fid, '\nSubject To \n');
% Constraint 1 (2)
for j=6:n_nodes
    I_j_aux = [];
    for i=6:n_nodes
        sp_aux = shortestpath(G_graph, j, i);
        sp_aux = sp_aux(2:end-1);
        if length(sp_aux) <= 1
            I_j_aux = [I_j_aux; i];
        end
    end
    for z=1:length(I_j_aux)
        fprintf(fid, ' + x%d', I_j_aux(z));
    end
    fprintf(fid, ' >= 1\n');
```

```
98  end
99  % Binary (3)
100 fprintf(fid, 'Binary\n');
101 for i=6:n_nodes
102     fprintf(fid, ' x%d\n', i);
103 end
104 fprintf(fid, 'End\n');
105 fclose(fid);
```

2.2 Alínea b)

2.2.1 Método e Resultados

Para resolver este exercício, em primeiro lugar foi necessário calcular o valor do critério de paragem, R. Sabendo que a rede que serve os subscritores deste serviço possui 10 AS Tier-2 e 25 AS Tier-1, e tendo que os Tier-2 conseguem servir 5000 clientes e que os Tier-3 conseguem servir 2500, temos então:

$$R = 5000 \times 10 + 2500 \times 25 = 112500$$

Tendo então este valor, foi então necessário determinar o número de servidores e o valor de reserva, n e W respetivamente. Para tal foi seguido um método semelhante ao exposto na Secção 1.7, onde fomos alterando os valores, primeiro de n e depois de W até alcançar os resultados pretendidos.

N	W	R	BK 4k	BK HD
19	10000	112500	8.00e+01 +- 5.86-02	6.13e+01 +- 1.09e-01
38	10000	112500	7.04e+01 +- 1.24e-01	1.74e+00 +- 5.61e-02
76	10000	112500	1.00e+00 +- 1.68e-01	0
76	25000	112500	8.54e-01 +- 1.43e-01	0
76	50000	112500	9.16e-01 +- 1.04e-01	0
76	52000	112500	6.66e-01 +- 1.30e-01	2.56e-01 +- 5.76e-02
76	52250	112500	7.40e-01 +- 7.90e-02	7.01e-01 +- 1.49e-01
76	52225	112500	7.98e-01 +- 1.67e-01	6.51e-01 +- 1.20e-01
76	52225	112500	6.69e-01 +- 1.54e-01	6.55e-01 +- 5.67e-02

Tabela 2.1: Resultados das simulações alínea b)

Com base neste método, obtemos então o valor mínimo de servidores necessários para cumprir com os requisitos:

$$n = 76;$$

Como na Secção 1.7.1, foi procurado um W que equilibre as taxas de bloqueio entre os dois formatos e em que estas não excedam o 1% .

O valor final atingido foi então:

$$W = 52225;$$

Na tabela 2.1 podem ser vistos dois exemplos de execuções distintas com estes parâmetros ($n = 76$;$W = 52225$), entre outros.

Os intervalos de confiança são elevados, o que significa que existe uma variância alta das taxas de bloqueio entre simulações. No entanto, estes intervalos ainda se mantém abaixo dos valores de performance ideais.

Em conclusão, serão precisos 76 servidores, divididos por 5 ASs, para garantir qualidade de serviço para os dois formatos.

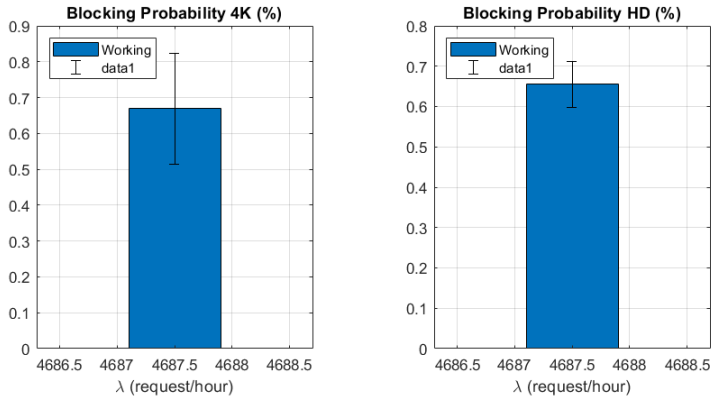

Figura 2.1: Resultado da alínea b)

2.2.2 Código

```
%%
%3b

% Configuration B
b_n = 76;
b_S = 1000;

N_times = 10;
```

```
alfa = 0.1;
W = 52000;
p = 0.30;
R = 112500;
lambda = R/24;
fname = "movies.txt";

[medias_b_4k_b, terms_b_4k_b, medias_b_hd_b,
    terms_b_hd_b] = ...
     runSimulator2(N_times, alfa, lambda, p, b_n, b_S,
         W, R, fname);

fprintf('Blocking probability 4K (%%) - (n = %d, W = %
    d) = %.2e +- %.2e\n', b_n, W, medias_b_4k_b,
    terms_b_4k_b);
fprintf('Blocking probability HD (%%) - (n = %d, W = %
    d) = %.2e +- %.2e\n', b_n, W, medias_b_hd_b,
    terms_b_hd_b);

figure(1);
tiledlayout(1,2)

nexttile;
bar(lambda, medias_b_4k_b(:))
legend("Working", "Location" ,"northwest")
title("Blocking Probability 4K (%)")
xlabel('\lambda (request/hour)')
grid on

hold on

er = errorbar(lambda, medias_b_4k_b, terms_b_4k_b);
er.Color = [0 0 0];
er.LineStyle = 'none';

hold off

nexttile;
bar(lambda, medias_b_hd_b(:))
legend("Working", "Location" ,"northwest")
title("Blocking Probability HD (%)")
xlabel('\lambda (request/hour)')
grid on

hold on

er = errorbar(lambda, medias_b_hd_b, terms_b_hd_b);
er.Color = [0 0 0];
er.LineStyle = 'none';

```

```
53 hold off
```

2.3 Alínea c)

2.3.1 Resultados e Discussão

A partir dos resultados das alíneas a) e b) sabemos que teremos 76 servidores para distribuir por 5 ASs. Para efectuar esta distribuição, inicialmente foi analisado, para cada *server farm*, o conjunto de ASs cujo caminho para a *server farm* não contém mais que 1 AS intermédio.

Foram então detectadas interseções entre conjuntos, o que significa que é possível optimizá-los. Com isto em mente foi decidido realizar uma nova distribuição dos AS com o objetivo de balançar as cargas de cada *server farm*, de forma a garantir a melhor distribuição de subscritores por *server farm*.

Nesta nova organização, é perdida alguma da redundância da rede, no entanto o critério usado foi baseado no número de subscritores por cada *server farm*. A distribuição original pode ser visualizada na figura 2.2 e a distribuição optimizada na figura 2.3.

Para distribuir os servidores por estes novos conjuntos, foi definido que cada servidor iria servir:

$$\frac{R}{n_{servers}} = \frac{112500}{76} \approx 1480 \text{ clientes}$$

Portanto, sabendo quantos clientes cada AS Tier-2 ou Tier-3 é capaz de servir, 5000 no caso de ser Tier-2 e 2500 no caso de ser Tier-3, é trivial calcular a capacidade máxima de cada área da rede. Dividindo esta capacidade total pelo número de clientes que cada servidor irá suportar, 1480, obtém-se então o número de servers que irão ser precisos para essa *server farm*.

Por exemplo:

$$NServers = \frac{Subscritores}{NClientes} = \frac{22500}{1480} \approx 15$$

Os resultados podem então ser consultados na tabela 2.2.

Server Farm	Conjunto de AS	Subscritores	nº Servers
9	{23,24,25,9,12,26,27}	22500	15
13	{33,34,35,13,36,37,38,14}	25000	17
16	{39,40,15,16,6,17,18,19}	25000	17
21	{21,20,2,8,22}	17500	12
30	{30,12,32,31,11,28,29}	22500	15

Tabela 2.2: Organização dos Conjuntos e Distribuição dos servers

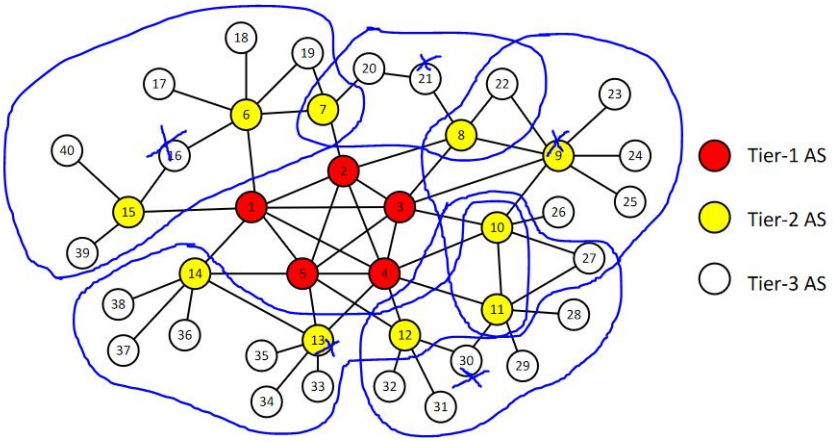

Figura 2.2: Conjuntos padrão

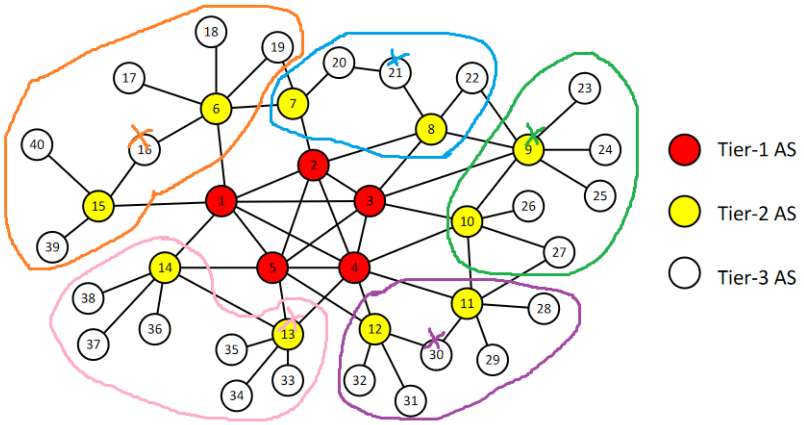

Figura 2.3: Conjuntos Otimizados