UNIVERSIDADE FEDERAL DE ALFENAS

Caio Eduardo Marcondes
Guilherme Augusto Gouveia

RELATÓRIO

Alfenas/MG

2023

RELATÓRIO

TRABALHO PRÁTICO 2

Relatório apresentando a análise de dados gerados por um simulador e suas conclusões, como parte do trabalho de Análise de Desempenho pela Universidade Federal de Alfenas.

Professor:
Prof. Dr. Flávio Barbieri Gonzaga

Alfenas/MG

2023

RESUMO

Esta atividade é composta por duas etapas, que, no geral, se preocupa em criar uma implementação em C que seja capaz de simular o processamento de pacotes embutido dentro de um comutador de dados. Na primeira etapa, nosso objetivo será descobrir qual a largura de link necessária para que um comutador responsável por processar pacotes de três tamanhos diferentes: 1500 bytes, 40 bytes e 550 bytes, onde, a probabilidade de que cada pacote chegue é de 10%, 40% e 50%, respectivamente, resulta em uma taxa de ocupação definida previamente. Feito isso, a segunda parte se enquadra como um aprimoramento da primeira, onde o elemento “chamada” é um novo evento para nos preocuparmos em nosso cenário, a existência deste tipo de evento traz consigo uma natureza de ocorrência simultânea, algo sem precedentes em nosso simulador.

Palavras-chave: Análise de desempenho, teoria de filas, simulação de rede, simulador.

ABSTRACT

This activity consists of two steps, which, in general, is concerned with creating an implementation in C that is capable of simulating the processing of packets embedded within a data switch. In the first step, our goal will be to find out the link width needed for a switch responsible for processing packets of three different sizes: 1500 bytes, 40 bytes and 550 bytes, where the probability that each packet arrives is 10%, 40% and 50%, respectively, result in the previously defined occupancy rate. Once this is done, the second part is configured as an improvement of the first, where the element “call” is a new event to worry about in our scenario, the existence of this type of event brings with it a nature of simultaneous occurrence, something unprecedented during our discipline.

Keywords: Performance analysis, queuing theory, network simulation, simulator.

LISTA DE FIGURAS

PARTE 1

PARTE 2

SUMÁRIO

1 INTRODUÇÃO

Este relatório consiste em demonstrar e detalhar como e quais foram nossos resultados diante do desafio proposto pela atividade, a grande questão aqui, é modificar o código de modo que seja possível realizar uma simulação da comutação de pacotes, onde o tempo de serviço empregado para tratar cada pacote passa a ser o atraso de transmissão e descobrir como definir a largura de link necessária para atender os pacotes com um dado nível de ocupação, a ponto de ser possível chegar a uma conclusão inteligível do que está acontecendo. Alguns obstáculos que tornam essa simulação mais próxima ao mundo real serão adicionados na segunda parte deste trabalho, porém o objetivo permanece o mesmo. O código teve sua base desenvolvida em paralelo com as aulas de Análise de Desempenho utilizando a linguagem de programação C.

2 DESENVOLVIMENTO

2.1 PARTE 1

2.1.1 EXPLICAÇÃO SOBRE OS PONTOS CRUCIAIS DO CÓDIGO

Nossa primeira preocupação foi criar uma forma de gerar pacotes para o simulador seguindo a descrição do trabalho, para isso, o que fazemos é gerar um número aleatório entre 0 e 9, e definir os intervalos (por meio de condicionais) que produzem a probabilidade correta de que o número sorteado se refira à um dos três pacotes existentes no sistema. Para não ficarmos apenas em uma explicação abstrata, veja o código abaixo:

```
68  int gera_pacote() {
69      int tamPacotes[3] = {1500, 40, 550};
70      int i = rand() % 10;
71      if (i < 1) {
72          i = 0;
73      } else if (i < 5) {
74          i = 1;
75      } else {
76          i = 2;
77      }
78      return tamPacotes[i];
79  }
```

Figura 1 - Função “gera_pacote”, imagem retirada do GitHub: simulacao_web.c.

A variável *largura_link* representa algo crucial em nosso trabalho, é nela que está contido o valor necessário para a largura de link que gera a taxa de ocupação que desejamos, a expressão contida na linha 131 representa toda a parte matemática dessa etapa do trabalho:

```
129     printf("Informe o percentual de ocupação desejado (entre 0 e 1): ");
130     scanf("%lF", &porc_ocupacao);
131     largura_link = (1 / intervalo_medio_chegada) * (0.1 * 1500 + 0.4 * 40 + 0.5 * 550) / porc_ocupacao;
132     printf("Largura do link: %lF", largura_link);
133     printf("\n%.2lF%%,0", porc_ocupacao * 100);
```

Figura 2 - Fórmula como é calculada a largura do link, imagem retirada do GitHub: simulacao_web.c.

A expressão matemática para esse trecho de código é dada por:

$$largura\_link = \left(\frac{1}{intervalo\_medio\_chegada}\right) \cdot \frac{(0.1 * 1500 + 0.4 * 40 + 0.5 * 550)}{porc\_ocupacao}$$

Devido ao grau de relevância dessa expressão, logo a seguir uma seção deste trabalho será dedicada exclusivamente à sua explanação.

Com tudo isso definido, a preocupação agora se torna em como utilizar o que foi criado até aqui. É fácil perceber que esses elementos se remetem a como gerar o tempo de serviço para processar um pacote, e, com uma simplória analogia, podemos perceber que o tempo de serviço representa o que é retratado na literatura de redes de computadores como atraso de transmissão (a_t), já que os outros atrasos existentes nesse tipo ambiente, como o atraso de processamento e propagação são desconsiderados aqui devido ao seu baixo impacto no atraso final. Como a função *gera_pacote* retorna nada mais do que o tamanho de um pacote (em bytes) seguindo alguns critérios pseudo estocásticos, apenas precisamos implementar a seguinte expressão para gerar o tempo de serviço:

$$a_t = \frac{gera\_pacote()}{largura\_link}$$

Tornando essa discussão mais palpável com auxílio de uma imagem do código, veja abaixo:

```
171             {
172                 servico = tempo_decorrido + gera_pacote() / largura_link;
173                 soma_tempo_servico += servico - tempo_decorrido;
174             }
```

Figura 3 - Tempo de serviço, imagem retirada do GitHub: simulacao_web.c.

Apenas em intuito comparativo, o tempo de serviço era de natureza exponencial, como podemos ver no trecho de código a seguir:

```
157             {
158                 servico = tempo_decorrido + (-1.0 / (1.0 / tempo_medio_servico)) * log(aleatorio());
159                 soma_tempo_servico += servico - tempo_decorrido;
160             }
```

Figura 4 - Cálculo do Tempo de serviço retirado do Trabalho 1.

2.1.2 FÓRMULA MATEMÁTICA UTILIZADA

No trabalho anterior, a fórmula descoberta para calcular a taxa de ocupação era:

$$t_o = \frac{ts}{\gamma} \quad \text{(eq. 1)}$$

Onde:

t_o: taxa de ocupação

γ: intervalo médio de chegada de pacotes

ts: intervalo médio de serviço

A grande diferença para este trabalho, é que o tempo de serviço passa a ser o atraso de transmissão que um pacote de tamanho L gera, o atraso de transmissão pode ser calculado da seguinte forma:

$$a_t = \frac{L}{R}$$

E como o atraso de transmissão é novo tempo de serviço, podemos dizer que:

$$ts = a_t$$

O nosso objetivo aqui é determinar o valor da largura de link (R) previamente de modo a gerar a taxa de ocupação (to) que desejamos. Primeiramente, substituímos a nova equivalência de ts encontrada na equação 1, e então teremos o seguinte:

$$t_o = \frac{1}{\gamma} \cdot \frac{L}{R}$$

Agora basta colocar a largura de link (R) em evidência, e então teremos uma expressão para calculá-la a partir de uma taxa de ocupação informada previamente.

$$R = \frac{1}{\gamma} \cdot \frac{L}{t_o}$$

Como nosso simulador lida com três tamanhos de pacotes diferentes com chances específicas de ocorrência, o valor de L passa a ser o tamanho médio que esses pacotes possuem, que nada mais é do que uma média ponderada onde os pesos são as chances de que cada pacote ocorra. Os 3 tipos de pacotes possuem 1500 bytes, 40 bytes e 550 bytes, onde a probabilidade de que cada pacote ocorra é de 10%, 40% e 50%, respectivamente. Logo:

$$L = 1500 \cdot 0,1 + 40 \cdot 0,4 + 550 \cdot 0,5$$

2.1.3 INFORMAÇÕES GERADAS PELA COLETA DE DADOS

- E [N]:

Para os valores de E[N] podemos observar que quanto maior é a taxa de ocupação desejada, maior é o número de clientes em média que estarão na fila. Dedicamos um gráfico exclusivo para a ocupação de 99% com o intuito de facilitar a visualização.

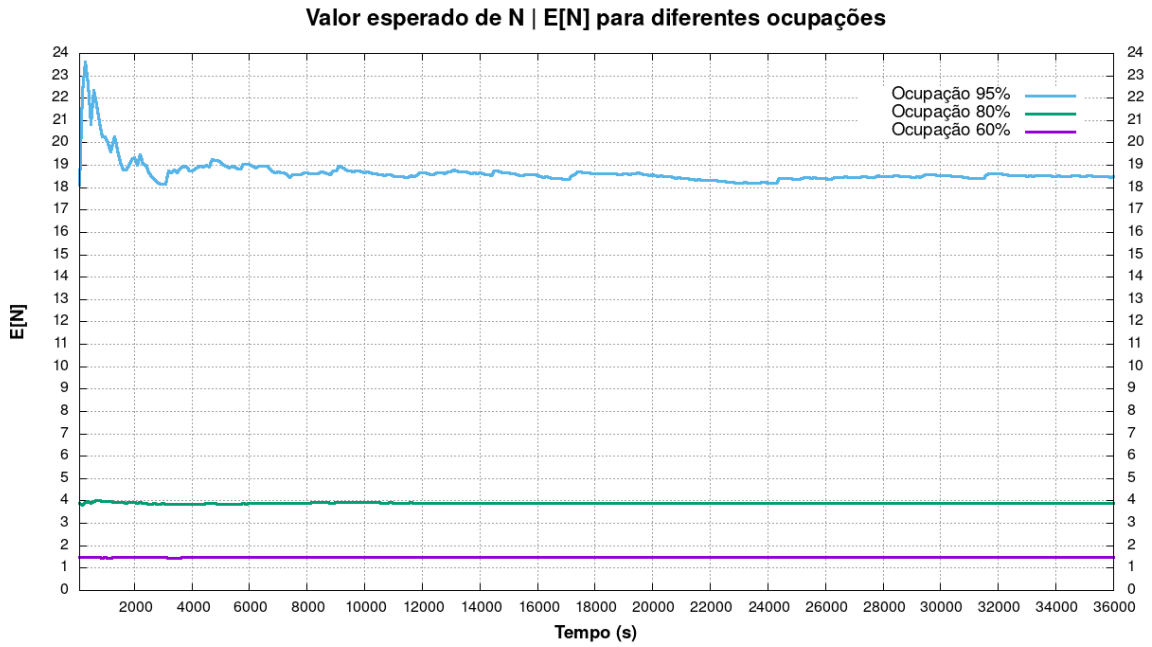

Figura 5 - Gráfico de Medidas de Little (E[N]).

- E [N] para a ocupação de 99%:

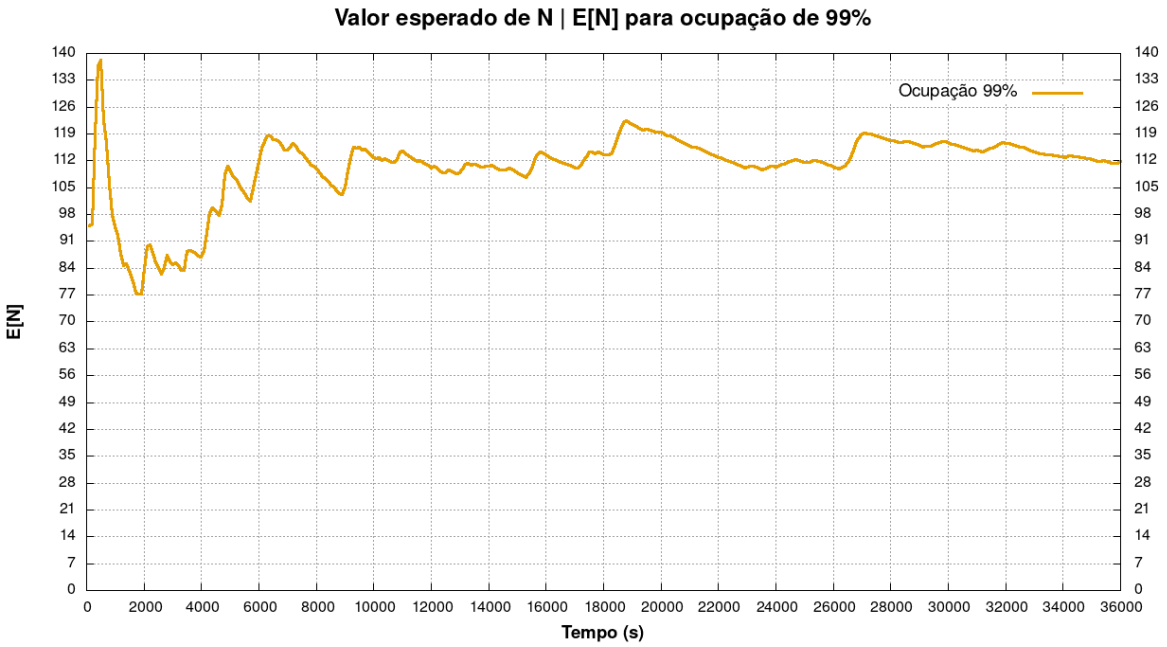

Figura 6 - Gráfico de Medidas de Little (E[N]) com ocupação de 99%.

- E [W]:

Para os valores de E[W] podemos observar que quanto maior é a taxa de ocupação desejada, maior é o tempo de espera médio em que os clientes ficam dentro do sistema, o que é naturalmente compreensível. Além disso, podemos ver também que o comportamento dos valores de E[W] são idênticos ao de E[N] no que se refere a quando os valores crescem ou decrescem, o que nos dá uma ideia do porquê da lei de Little ser válida.

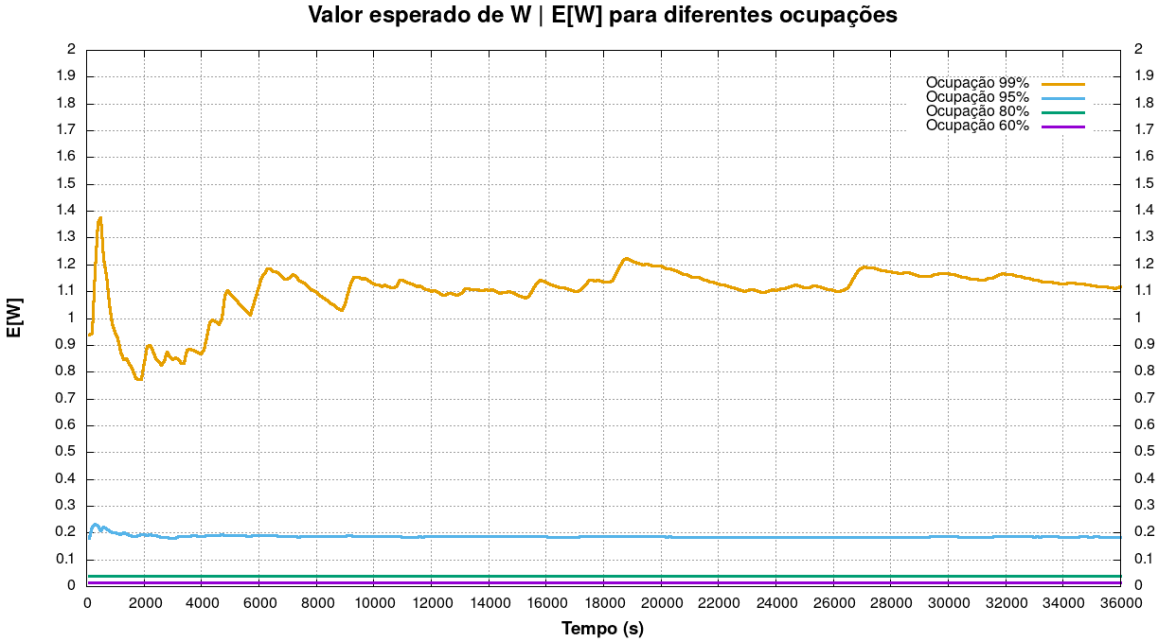

Figura 7 - Gráfico de Medidas de Little (E[W]).

- Erro de Little:

Para calcularmos o erro de nosso sistema de simulação utilizamos a Lei de Little:

$$E[N] = \lambda \cdot E[W]$$

E definimos o *Erro* como sendo:

$$Erro = E[N] - \lambda E[W]$$

Quanto mais perto de 0, mais o nosso simulador caminha para atender à Lei de Little, válida para qualquer tipo de fila, como nosso sistema de simulação trabalha com aleatoriedade o algoritmo nunca é preciso, mas caminha para isso. Logo, a medida que o tempo tende ao infinito ($t \rightarrow \infty$) o erro tende a zero ($Erro \rightarrow 0$).

Interpretando o gráfico abaixo, podemos observar que o erro é bem pequeno, e conforme avançamos no tempo existe uma divergência nos valores, mas a tendência é que com o decorrer do tempo exista uma convergência dos valores e o *Erro* tende a 0.

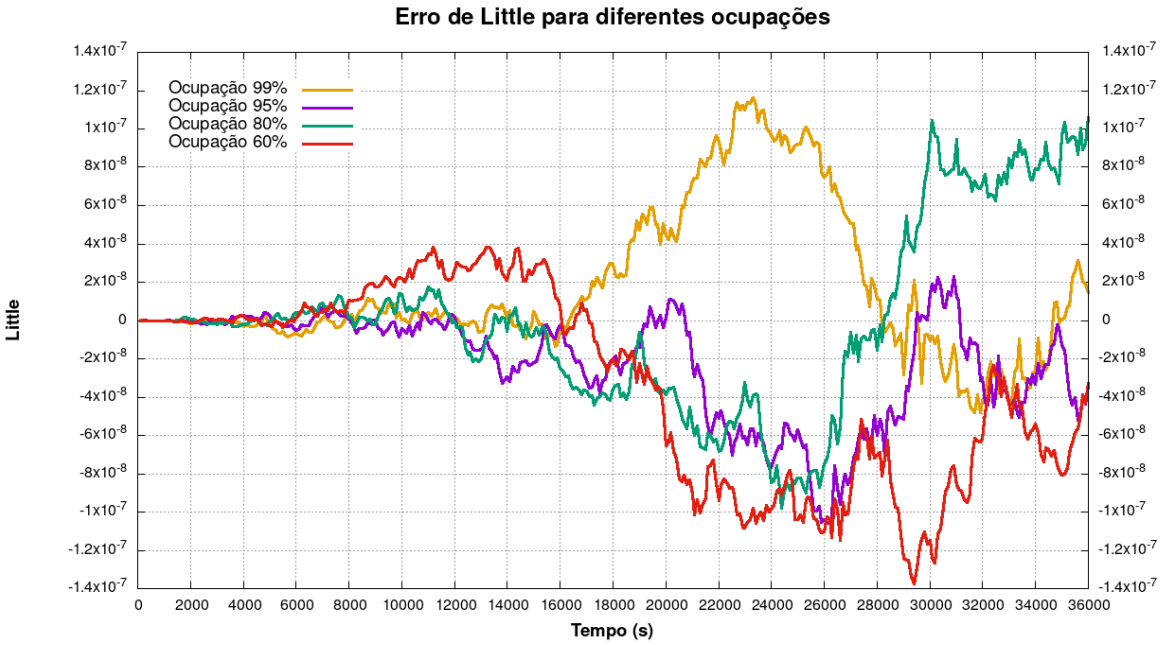

Figura 8 - Gráfico do Erro de Little.

- Ocupação:

A ocupação era o critério principal da atividade, pois deveríamos gerar todos os dados referentes à fila de pacotes com o objetivo de, ao final, gerar taxas de ocupação que foram estabelecidas de maneira premeditada. Como já muito bem detalhado na seção (2.1.2), calculamos qual a largura de link necessária para gerar o atraso de transmissão (tempo de serviço) que no fim gera a taxa de ocupação requerida.

Veja o gráfico da taxa de ocupação logo abaixo:

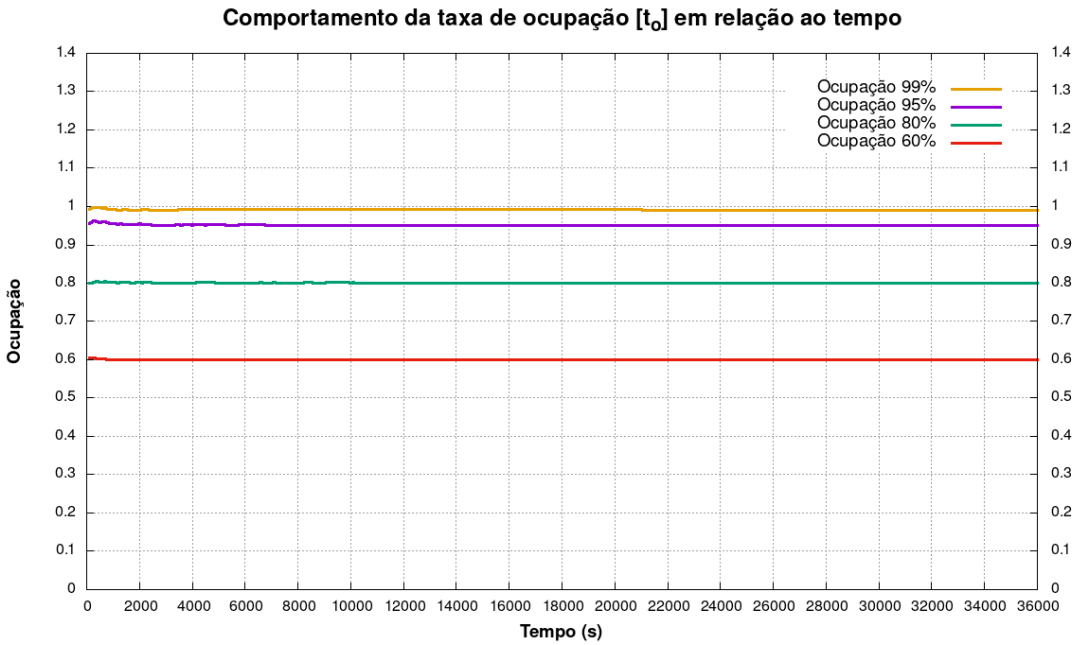

Figura 9 - Gráfico de Ocupação.

2.2.4 VALORES FINAIS

Tabela 1 - Valores finais obtidos na execução de cada caso de ocupação

Valores finais Tempo de simulação: 36.000 segundos — Intervalo médio de chegada de pacotes: 0.01					
Ocupação (Largura de link)	**E[N]**	**E[W]**	**Lambda**	**Erro de Little**	**Aproximação para ocupação**
60% (73.500 B/s)	1.469292	0.014699	99.959412	3.3195×10^{-8}	0.599914
80% (55.125 B/s)	3.884953	0.038854	99.989640	1.0567×10^{-7}	0.799747
95% (46.421,05 B/s)	18.476852	0.184777	99.995357	3.4044×10^{-8}	0.949930
99% (44.545,45 B/s)	111.930595	1.118871	100.038860	1.4613×10^{-8}	0.990703

Informações obtidas do código.

2.2 PARTE 2

2.2.1 EXPLICAÇÃO SOBRE OS PONTOS CRUCIAIS DO CÓDIGO

Antes de tudo, é importante ressaltar que algumas refatorações foram realizadas do código original do simulador para o que foi utilizado no simulador da parte 2. Veja abaixo:

- Inclusão da estrutura switch\case para tratar cada evento
- Definição do tipo Event para representar cada evento existente
- Criação da função *create_event* que encapsula a ação de criar um evento deixando o código mais legível.
- Definição da função *exponential*, devido ao seu desabalado uso ao longo do código

Mas como foi dito no início, são apenas refatorações, o que não altera a funcionalidade do código, apenas sua inteligibilidade. Sem mais delongas, vamos ao que interessa.

2.2.2 ESTRUTURA DE ARMAZENAMENTO DE EVENTOS

Como nosso simulador poderá lidar com mais eventos em simultâneo do que estávamos acostumados, o uso de uma estrutura que facilite o trabalho com variados eventos se torna fundamental e determinante para a performance de nosso simulador. Dado sua característica, a necessidade de sempre extrair um evento de tempo mínimo de nosso sistema é um fator determinante em nossa escolha, por conta disso, a estrutura escolhida foi a Árvore Binária Min-Heap, essa estrutura tem a característica de possibilitar a extração de valores mínimos de um conjunto de dados em tempo de complexidade $O(log(n))$, algoritmos mais simples envolvem percorrer uma lista buscando o mínimo com uso de um loop, porém isso gera complexidade de $O(n)$.

2.2.3 NOVOS EVENTOS

A primeira grande diferença entre a parte 2 e 1, está na adição de um novo tipo de evento que pode ocorrer em natureza simultânea, diferentes dos eventos de chegada que ocorrem no sistema respeitando um esquema de filas, as chamadas (novo evento) podem coexistir, a implicação disso é que teremos mais chegadas ocorrendo na simulação à medida que mais chamadas começam coexistir. Veja abaixo:

```
case NOVA_CHAMADA:

    no_chamadas++;
    chamada = create_event(heapEventos, NOVA_CHAMADA, tempo_decorrido + exponential(intervalo_medio_chamada));
    create_event(heapEventos, FIM_CHAMADA, chamada.time + exponential(duracao_chamada));
    break;

case FIM_CHAMADA:
    no_chamadas--;
    break;
}
```

Figura 1 - Código demonstrando os “novos eventos”.

2.2.4 EVENTO NOVA_CHAMADA

Essa ação apenas trata quando uma nova chamada se inicia no simulador, o que fazemos aqui é apenas incrementar a variável *no_chamadas* (algo muito importante) e criar o evento da próxima chamada no sistema. A variável *no_chamadas* é importante pois irá refletir na taxa de chegada de pacotes de ligação no sistema no instante t em nossa simulação de tempo discreto.

2.2.5 EVENTO FIM_CHAMADA

Um evento *fim_chamada* está intimamente ligado ao evento *nova_chamada*, possuindo a característica de atomicidade, ou seja, sempre que um evento *nova_chamada* é criado, um evento *fim_chamada* também deve ser criado logo em seguida. A única ação a ser tomada quando este evento ocorrer é decrementar a variável *no_chamadas*. Perceba que uma ótima oportunidade de refatoração para essa abordagem que utilizamos é transformar os eventos *nova_chamada* e *fim_chamada* em um único evento, porém isso implicaria numa mudança na *struct* que define o tipo Event.

2.2.6 GERAÇÃO DE PACOTES

Agora, existem duas funções responsáveis pela geração de pacotes, uma para a web e outra para as ligações:

```
int gera_pacote_ligacao()
{
    no_pacotes_ligacao++;
    return 160;
}

int gera_pacote_web()
{
    int tamPacotes[3] = {1500, 40, 550};
    int i = rand() % 100;
    if (i < 10)
    {
        i = 0;
    }
    else if (i < 50)
    {
        i = 1;
    }
    else
    {
        i = 2;
    }
    no_pacotes_web++;

    return tamPacotes[i];
}
```

Figura 2 - Funções *gera_pacote_ligacao* e *gera_pacote_web.*

A função *gera_pacote_web* continua com mesmo comportamento definido anteriormente, já a *gera_pacote_ligacao* retorna o tamanho dos pacotes gerados por uma ligação, neste caso seu tamanho é constantemente 160 bytes.

2.2.7 SEPARAÇÃO DOS EVENTOS DE CHEGADA E SERVIÇO PARA WEB E TEMPO REAL

Uma necessidade que surgiu para conseguirmos atender aos critérios descritos pela proposta do trabalho, foi a separação dos eventos de chegada e serviço para pacotes da web e tempo real, para assim podermos realizar o cálculo das medidas de little separadamente. Porém, algumas observações são muito pertinentes aqui, essa tomada de decisão implicou em alguns cuidados na geração do tempo de serviço para os pacotes, em uma implementação incorreta a separação entre a chegada e a saída dos pacotes poderia gerar o comportamento inadequado dos pacotes web e ligação serem tratados como coisas distintas e totalmente independentes um do outro, o que não é verídico. A resolução está na hora de adicionar o tempo de serviço ao tempo decorrido, antes disso, por exemplo, se

estivermos gerando o tempo de serviço de um pacote web, temos que comparar o último tempo de serviço gerado para um pacote de ligação (para esteja ocorrendo ainda) com o tempo decorrido, e considerar apenas o máximo entre essas duas contagens de tempo.

2.2.8 EXPRESSÃO UTILIZADA PARA CÁLCULO DA LARGURA DE LINK

Uma explicação mais detalhada sobre a expressão obtida será feita mais adiante, por agora, basta sabermos que ela é uma extensão da expressão utilizada pelo simulador web. Veja o código a seguir:

```
194     // calculando chances que cada pacote tem de ser gerado
195     double no_pacotes_web_por_segundo = 1.0 / intervalo_medio_chegada_web;
196     double no_pacotes_ligacao_por_segundo = (duracao_chamada / intervalo_medio_chamada) / intervalo_medio_chegada_ligacao;
197     double no_pacotes_por_segundo = no_pacotes_web_por_segundo + no_pacotes_ligacao_por_segundo;
198     double chance_web = no_pacotes_web_por_segundo / no_pacotes_por_segundo;
199
200     scanf("%lF", &porc_ocupacao);
201     double intervalo_medio_chegada = 1 / (1 / intervalo_medio_chegada_web + (duracao_chamada / intervalo_medio_chamada) / intervalo_medio_chegada_ligacao);
202     largura_link = (1.00 / intervalo_medio_chegada) * ((0.1 * 1500.00 + 0.4 * 40.00 + 0.5 * 550.00) * chance_web + 160.00 * (1 - chance_web)) / porc_ocupacao;
203     printf("Largura do link para %.2lF%% de ocupação: %lF\n", porc_ocupacao * 100, largura_link);
```

Figura 3 - Cálculo da largura de link.

A expressão matemática para esse trecho de código é dada por:

$$largura\_link = \left(\frac{1}{intervalo\_medio\_chegada}\right) \cdot \frac{(0.1 * 1500 + 0.4 * 40 + 0.5 * 550) \cdot chance\_web + 1280 \cdot (chance\_web - 1)}{porc\_ocupacao}$$

2.2.9 FÓRMULA MATEMÁTICA UTILIZADA

A fórmula obtida inicialmente e que foi utilizada no simulador web, ainda é aplicável a este cenário, o grande ponto é que algumas variáveis deverão ser manipuladas de maneira a agregar as chamadas que ocorrem no sistema. A fórmula original para largura de link é a seguinte:

$$R = \frac{1}{\gamma} \cdot \frac{L}{t_o}$$

Como agora existem dois elementos que fazem injeção de pacotes no nosso simulador, o intervalo médio de tempo em que novos pacotes chegam (γ) e o

tamanho médio dos pacotes (L) devem ser reescritos em função das novas variáveis acrescidas. Vamos cuidar disso logo a seguir.

2.2.10 DEFININDO NOVO INTERVALO MÉDIO DE CHEGADA (γ)

Seja:

γ_w: intervalo médio de chegada de pacotes web

γ_l: intervalo médio de chegada de pacotes de ligação

n_w: número de pacotes web por segundo

n_l: número de pacotes de ligação por segundo

A questão aqui é que para este problema nós temos os valores de γ_w e γ_l, nosso objetivo é descobrir o valor de γ. Sabemos que o intervalo médio da chegada de pacotes é inversamente proporcional ao número de pacotes que chegam no simulador. Isso é descrito pela seguinte expressão:

$$\gamma = \frac{1}{n} \text{ (eq. 3)}$$

Além disso, sabemos que n (número de pacotes gerados a cada segundo) é nada mais do que a soma de n_w e n_l. Uma observação importante é que número de pacotes aumenta linearmente conforme novas chamadas se iniciam em simultâneo, supondo que existem N chamadas coexistindo em média no simulador, podemos dizer que a ligação produzirá N vezes mais pacotes. Logo a expressão para calcular o número de n pacotes que ocorrem por segundo no simulador é:

$$n = n_w + N \cdot n_l$$

Reutilizando a equação 3, podemos dizer também que:

$$n_w = \frac{1}{\gamma_w} \text{ e } n_l = \frac{1}{\gamma_l}$$

Fazendo as substituições óbvias, conseguimos então determinar γ a partir de γ_w e γ_l por meio da seguinte expressão:

$$\gamma = \frac{1}{\frac{1}{\gamma_w} + \frac{1}{\gamma_l} \cdot N}$$

Pronto! Agora já temos o novo intervalo médio geral para nossa fórmula de largura de link.

2.2.11 DEFININDO NOVO TAMANHO MÉDIO DE PACOTES (L)

Para calcularmos o tamanho médio dos pacotes devemos primeiro fazer isso separadamente para da tipo de pacote que temos no sistema, já sabemos que o tamanho médio dos pacotes web (L_w) é o seguinte:

$$L_W = 1500 \cdot 0,1 + 40 \cdot 0,4 + 550 \cdot 0,5$$

Agora o nosso desafio é descobrir o tamanho médio dos pacotes de chamadas (L_l). Sabemos que o tráfego de chamadas tem a característica CBR (Constant Bit Rate), logo o tamanho de pacotes é constante, além disso temos também que a cada segundo 64 Kb de dados são produzidos, e que os pacotes chegam num intervalo (γ_l) de 20ms cada. Sendo assim, basta converter os Kb em bytes uma simples regra de três para descobrirmos o tamanho de cada pacote:

$$L_l = \frac{64.000}{8} \cdot 0,02 = 160\ bytes$$

Seguindo a ideia que utilizamos para calcular o tamanho médio dos pacotes web, precisamos definir a probabilidade de que cada tipo de pacote ocorra para usarmos como peso em nossa média ponderada, e assim chegarmos em um valor para L.

Para isso, basta pegarmos o número de pacotes n_w e n_l (de ambos os tipos) que o simulador gera por segundo, e medir a porcentagem disso com relação ao número total de pacotes:

$$c_w = \frac{n_w}{n} \text{ e } c_l = \frac{n_l}{n} \cdot N$$

A explicação de como chegar nos valores de n_w, n_l e n já foi dada acima, logo não retomou essa discussão aqui. Com tudo o que foi definido, podemos chegar em um novo valor de L:

$$L = L_l \cdot c_l + L_w \cdot c_w$$

2.2.12 EXPRESSÃO FINAL PARA LARGURA DE LINK

Com os novos valores de L e γ obtidos, podemos então chegar na expressão final para cálculo prévio da largura de link em nosso simulador:

$$R = \frac{1}{\frac{1}{\frac{1}{\gamma_w} + \frac{1}{\gamma_l} \cdot N}} \cdot \frac{L_l \cdot c_l + L_w \cdot c_w}{t_o}$$

Fazendo as simplificações possíveis, obtemos então:

$$R = \left(\frac{1}{\gamma_w} + \frac{N}{\gamma_l}\right) \cdot \frac{L_l \cdot c_l + L_w \cdot c_w}{t_o}$$

2.2.13 INFORMAÇÕES GERADAS PELA COLETA DE DADOS

- E [N]:

Para os valores de E[N] para o cenário em que há chegada de pacotes tanto de ligação quanto web, podemos observar que quanto maior é a taxa de ocupação desejada, maior é o número de pacotes em média que estarão na fila. Dedicamos um gráfico exclusivo para a ocupação de 99% com o intuito de facilitar a visualização.

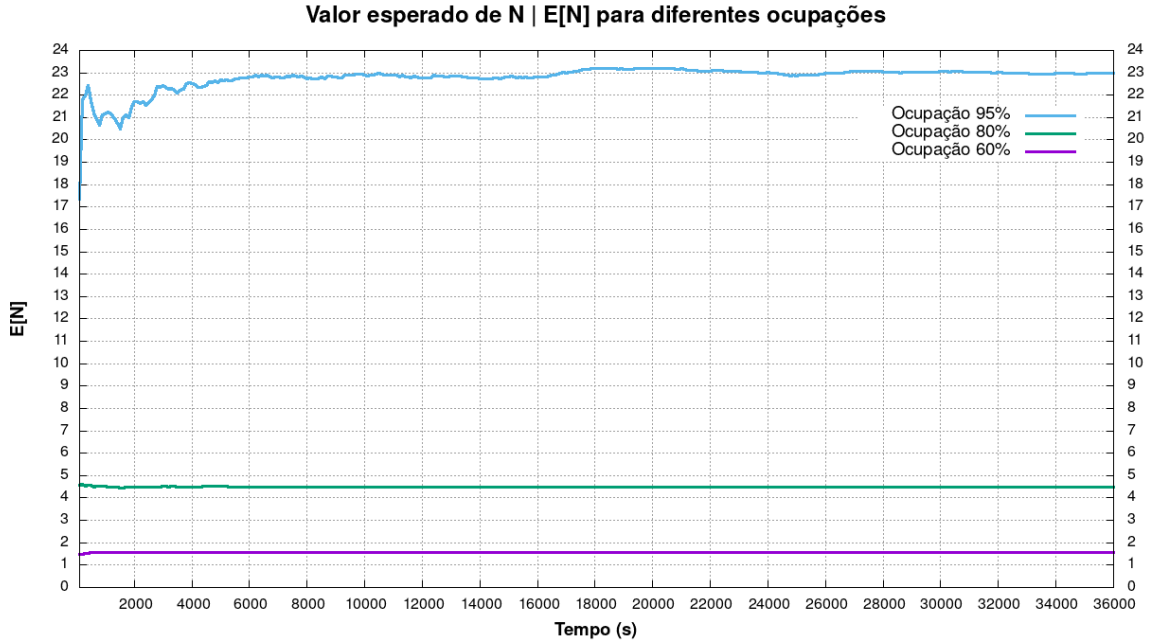

Figura 4 - Gráfico de Medidas de Little (E[N]).

- E [N] para a ocupação de 99%:

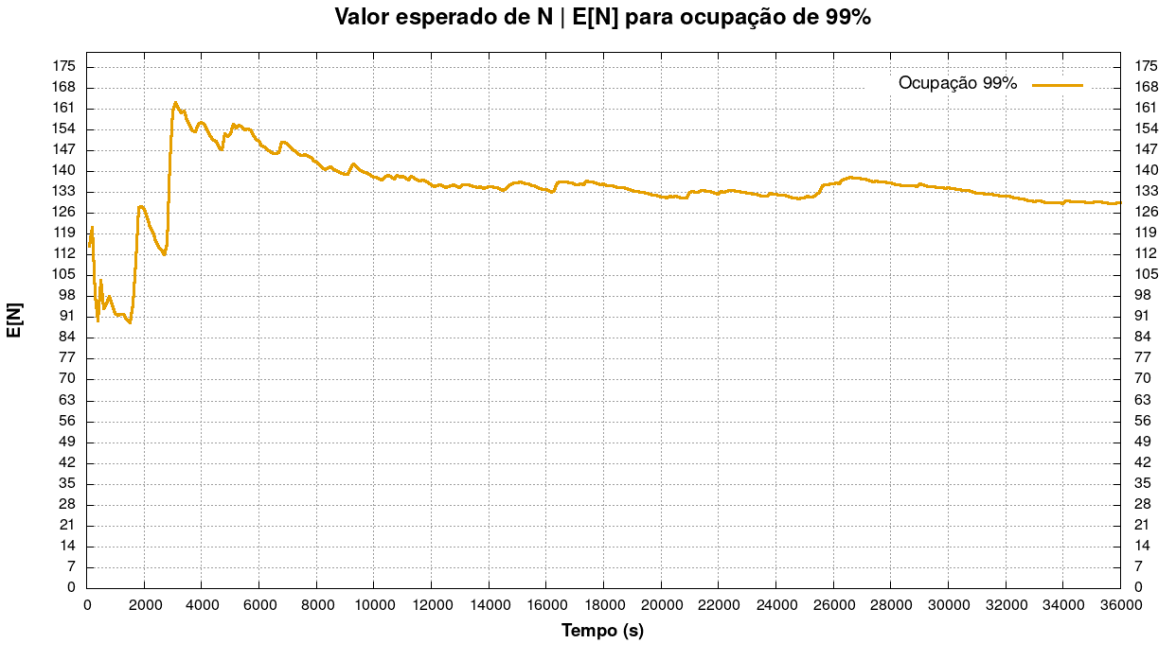

Figura 5 - Gráfico de Medidas de Little (E[N]) com ocupação de 99%.

- E [N] - Tempo Real:

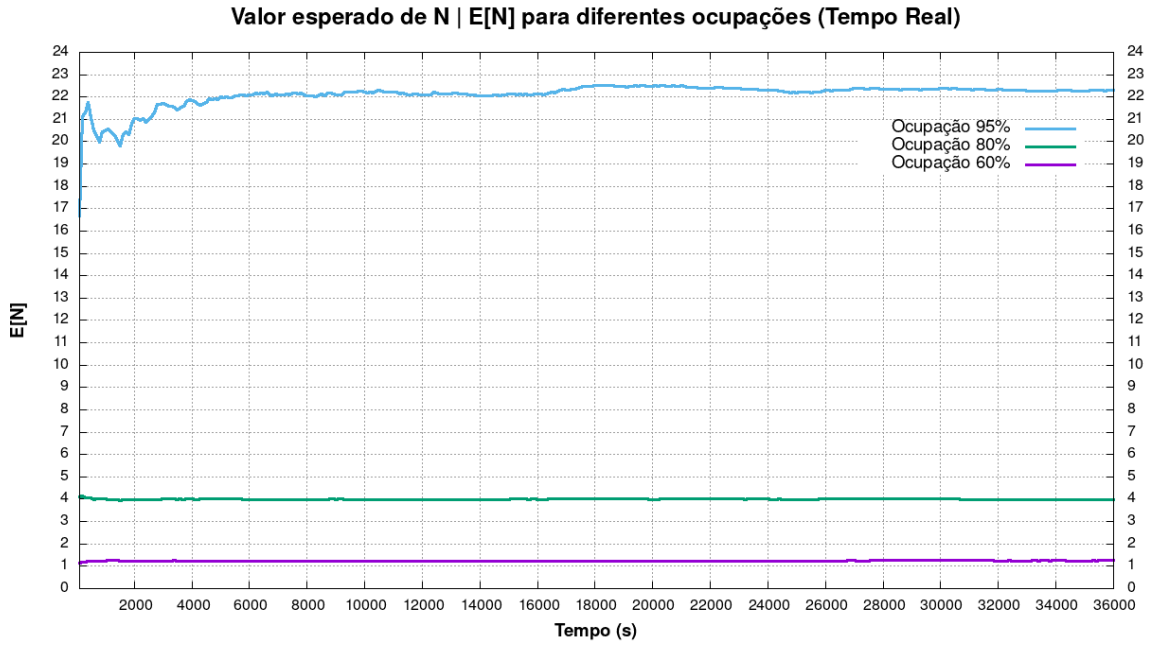

Figura 6 - Gráfico de Medidas de Little (E[N]) (tempo real).

- E [N] para ocupação de 99% - Tempo Real:

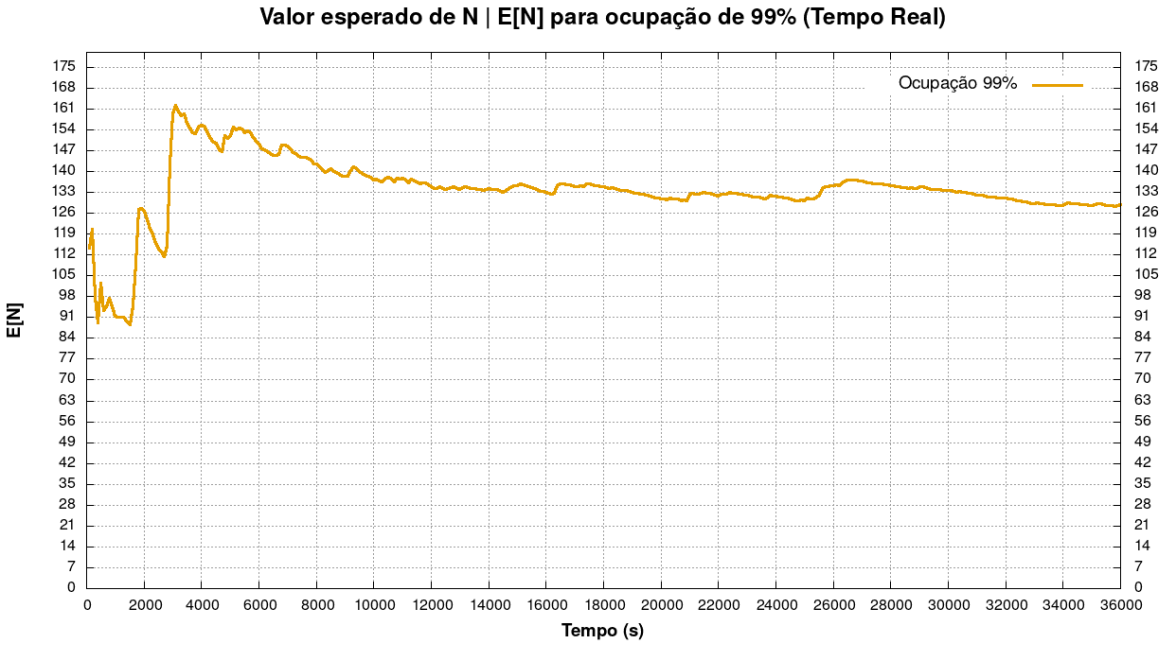

Figura 7 - Gráfico de Medidas de Little (E[N]) com ocupação de 99% (tempo real).

- E [W]:

No cálculo dos valores de E[W] para os pacotes de ligação e web, podemos observar que quanto maior é a taxa de ocupação desejada, maior é o tempo de espera médio em que os clientes ficam dentro do sistema, o que é naturalmente compreensível.

Além disso, podemos ver também que o comportamento dos valores de E[W] são idênticos ao de E[N] no que se refere a quando os valores crescem ou decrescem, o que nos dá uma ideia do porquê da lei de Little ser válida.

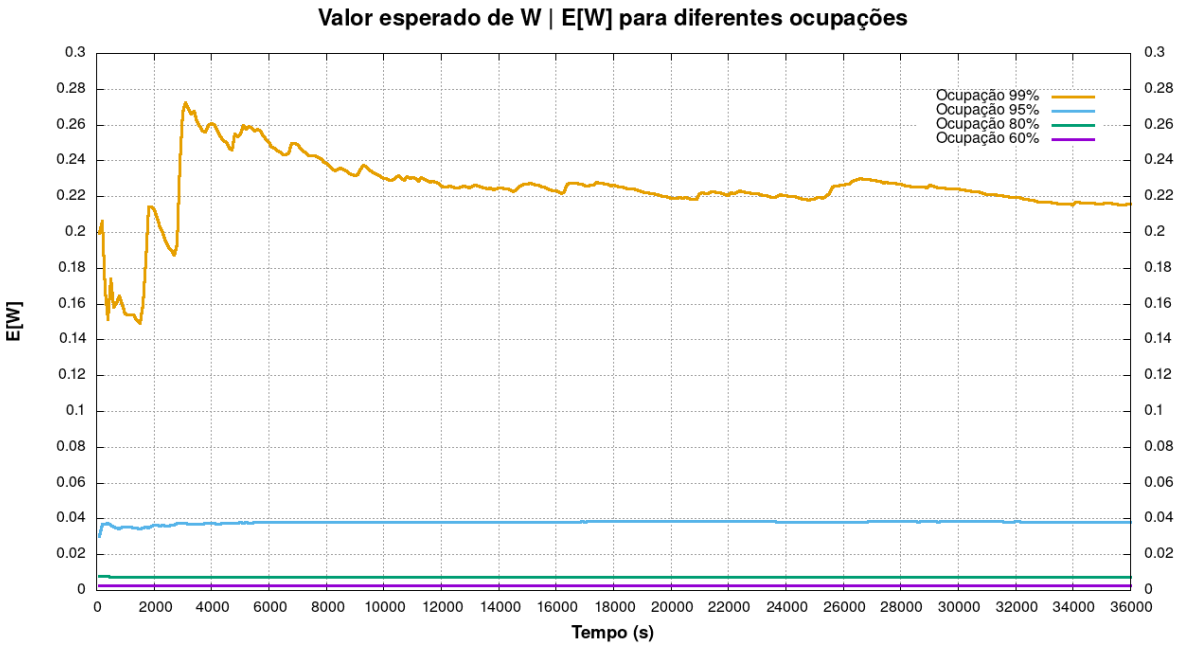

Figura 8 - Gráfico de Medidas de Little (E[W]).

- E [W] - Tempo Real:

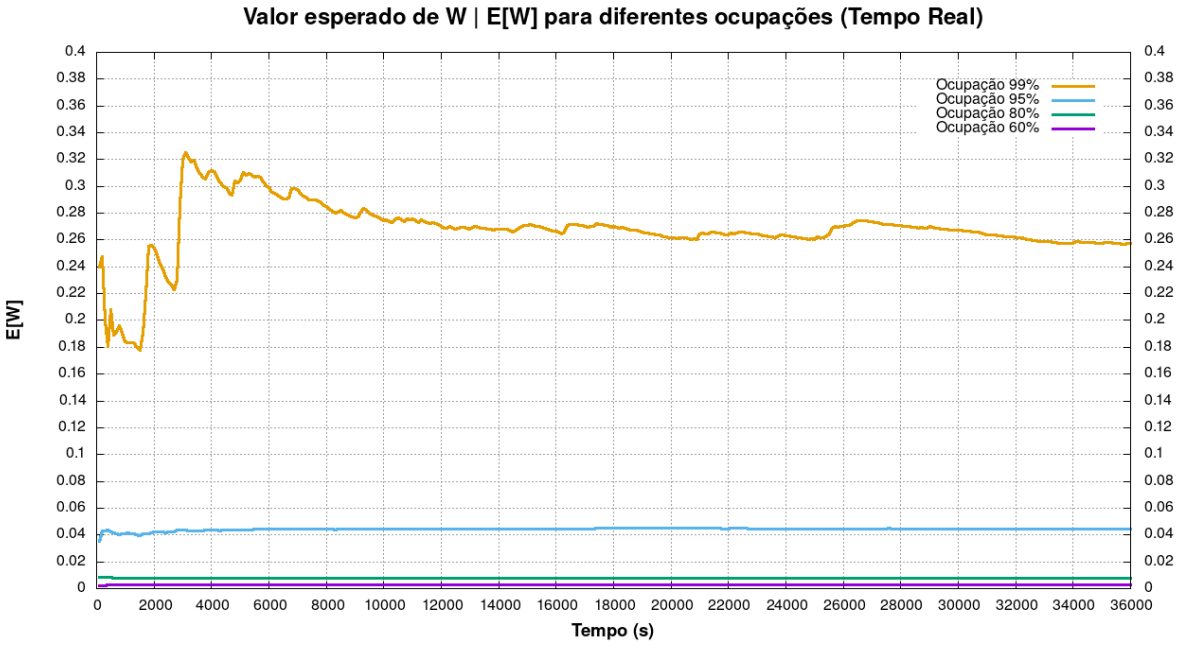

Figura 9 - Gráfico de Medidas de Little (E[W]) (tempo real).

- Erro de Little:

A descrição para o erro é a mesma já dada na parte 1, o comportamento é similar e o erro varia entre $1.6x10^{-6}$ e $-4x10^{-7}$.

A seguir temos os gráficos para todos os pacotes, um zoom no trecho final do gráfico geral a fim de obter melhor visualização e também um que considera apenas pacotes de tempo real:

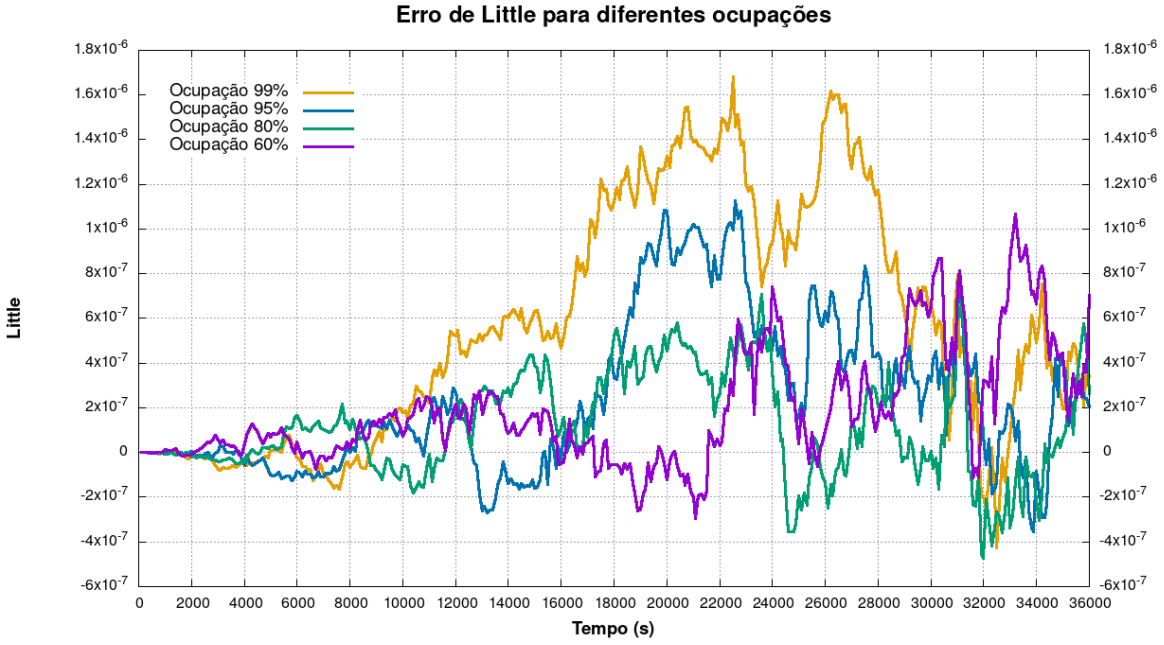

Figura 10 - Gráfico do Erro de Little.

- Erro de Little (Zoom):

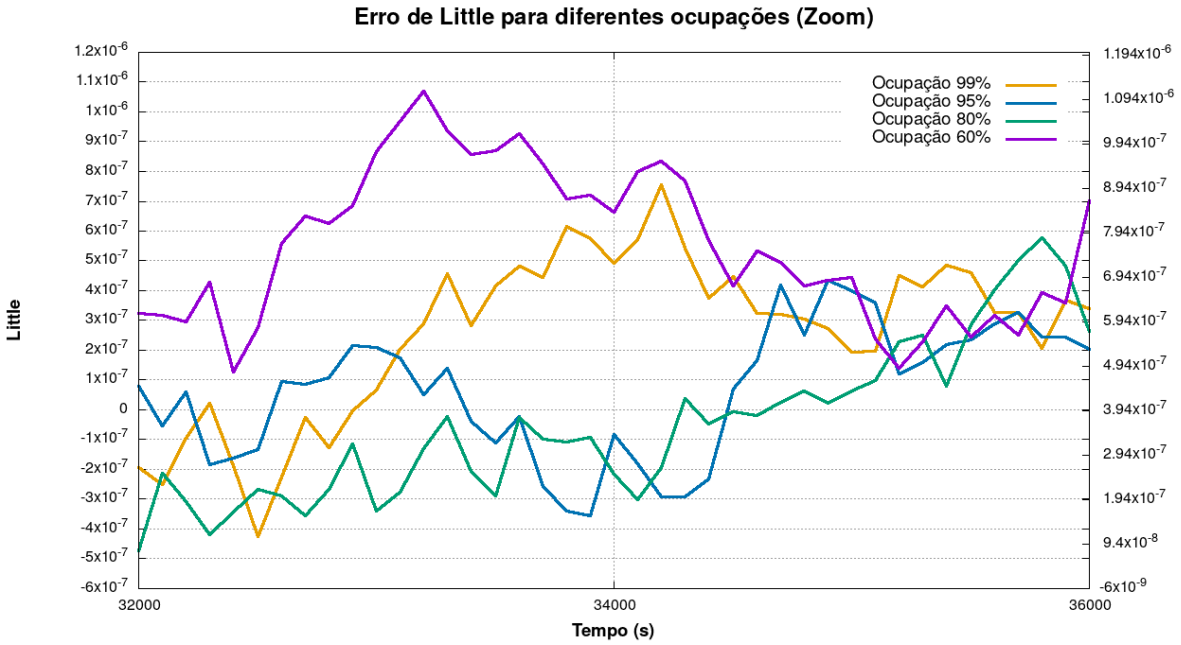

Figura 11 - Gráfico do Erro de Little (zoom).

- Erro de Little - Tempo Real

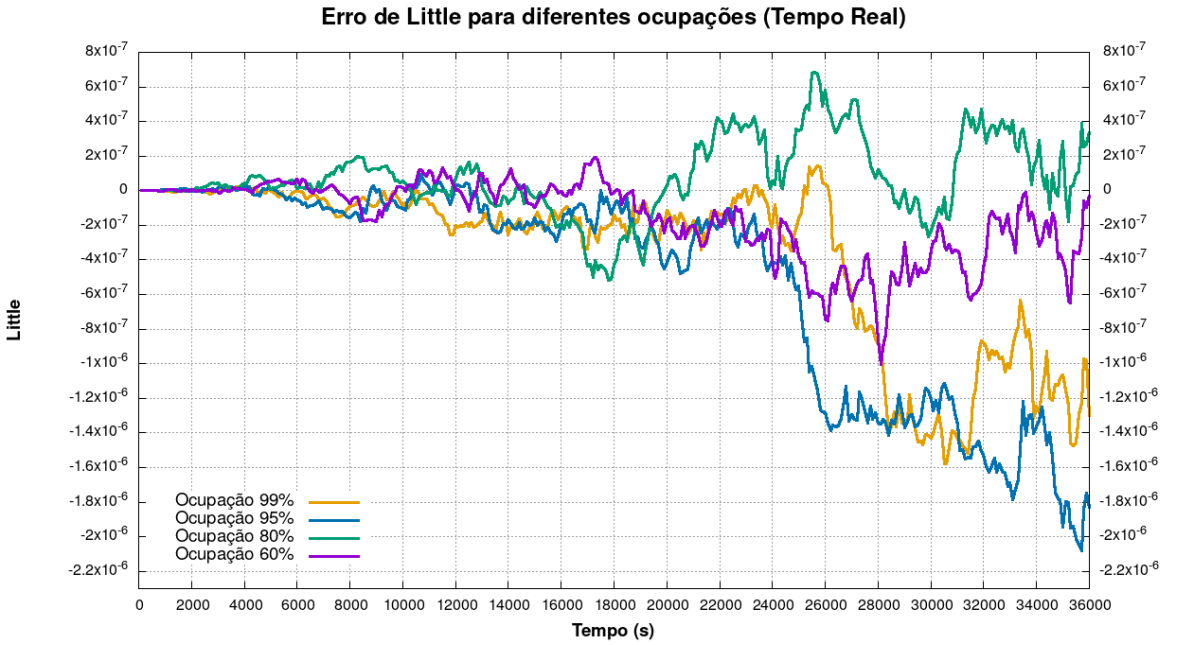

Figura 12 - Gráfico do Erro de Little (tempo real).

- Ocupação:

Abaixo, o comportamento da ocupação conforme avançamos no tempo, percebemos que sua tendência é a estabilização com relação a taxa de ocupação previamente definida.

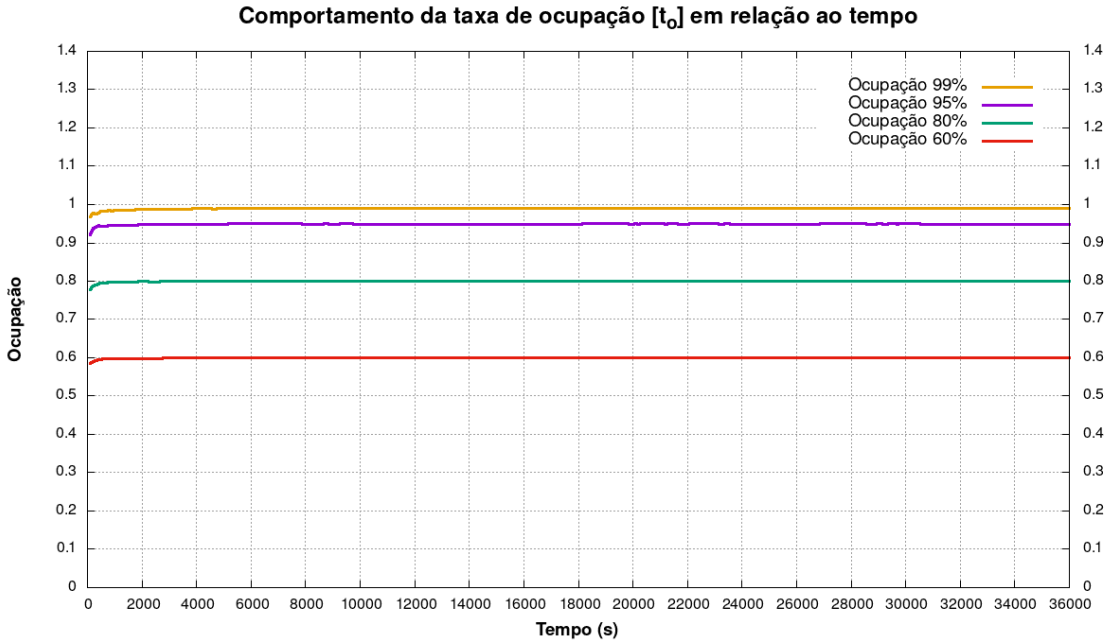

Figura 13 - Gráfico de Ocupação.

2.2.14 VALORES FINAIS

2.2.14.1 VALORES GERAIS

Tabela 1 - Valores finais obtidos na execução de cada caso de ocupação

Valores finais Tempo de simulação: 36.000 segundos Intervalo médio de chegada de pacotes (web) : 0.01 segundos Intervalo médio de chegada de pacotes (tempo real): 0.02 segundos Duração chamada: 100 segundos Intervalo médio entre chamadas: 10 segundos					
Ocupação (Largura de link)	**E[N]**	**E[W]**	**Lambda**	**Erro de Little**	**Aproximação para ocupação**
60% (206.833,33 B/s)	1.564582	0.002608	599.885824	7.0168×10^{-7}	0.599917
80% (155.125 B/s)	4.488581	0.007482	599.898408	2.6430×10^{-7}	0.799723
95% (130.631,57 B/s)	22.987321	0.038321	599.864126	2.0463×10^{-7}	0.949359
99% (125.353,53 B/s)	129.495850	0.215874	599.867040	3.3954×10^{-7}	0.989617

Informações obtidas do código

2.2.14.2 VALORES PARA PACOTES DE TEMPO REAL

Tabela 1 - Valores finais obtidos na execução de cada caso de ocupação

Valores finais Tempo de simulação: 36.000 segundos Intervalo médio de chegada de pacotes (web) : 0.01 Intervalo médio de chegada de pacotes (tempo real): 0.02 Duração chamada: 100 segundos Intervalo médio entre chamadas: 10 segundos				
Ocupação	**E[N]**	**E[W]**	**Lambda**	**Erro de Little**
60%	1.249048	0.002499	499.898747	3.3158×10^{-8}
80%	3.988919	0.007979	499.953243	3.3582×10^{-7}
95%	22.302162	0.044609	499.948887	1.8304×10^{-6}
99%	128.749170	0.257551	499.897153	1.3020×10^{-8}

Informações obtidas do código

3 CONCLUSÃO

A primeira parte foi fácil quanto às modificações de código e a teoria matemática envolvida, não houve muita dificuldade e sua concretização foi quase que em uma hora. Porém, as mudanças necessárias eram cirúrgicas, logo era necessário pensar bem antes de mexer para evitar erros que desencadeassem uma série de mudanças desnecessárias e novos erros em busca de corrigir o original.

A segunda parte era definitivamente mais complexa com relação a qualquer critério se comparada com a primeira parte, houveram muitos problemas ao longo de sua implementação, como por exemplo, a necessidade de realizar a contagem de pacotes na fila separadamente para cada tipo de pacote, pois quando usávamos apenas uma contagem de fila o código apresentava um comportamento inadequado com relação ao tempo de serviço gerado devido a sua dependência ao fato de existir ou não pacotes em fila na hora de definir o tempo de serviço de acordo com a ocorrência de certos eventos. Outro problema surgiu quando precisamos calcular as medidas de Little separadamente, o que acarretou na necessidade de criar uma nova solução de código, como citado na seção 2.2.1.4 deste relatório. Apesar dos problemas, é indiscutível que a refatoração que fizemos nos ajudou muito com relação a facilidade de entendimento e redução de código.

Outra questão interessante que podemos contar sobre o desenvolvimento de nosso trabalho é que inicialmente havia criado uma solução para a segunda etapa que consistia em um único evento de chegada e serviço e uma única função de geração de pacotes que calculava a probabilidade de um pacote ser web ou de ligação e os gerava, porém essa abordagem nos impossibilitava de calcular medidas como a de Little separadamente, isso culminou na necessidade de elaborar uma nova forma de resolver no tocante ao código, a solução obviamente foi a separação dos eventos de chegada e serviço para web e ligação tomando alguns cuidados para que eles possam trabalhar em conjunto e respeitando a geração do tempo de serviço um do outro, para que não ocorressem em simultâneo o que geraria irregularidades quanto as validações realizadas em nosso simulador.

Além disso, esse trabalho nos motivou a aprender uma nova ferramenta que, devido ao quanto gostamos, utilizaremos com certeza em nosso trabalho da disciplina de Heurística, Iniciação Científica e TCC, a ferramenta em questão é o gnuplot, muito útil e versátil no que se diz respeito a geração de gráficos.