Simulação de Pacotes e Conexões VoIP em Redes de Computadores com Distribuição Exponencial

Aluno: José Tobias
Professor: Flavio Barbieri Gonzaga
Disciplina: Análise de Desempenho

7 de junho de 2019

1 Introdução

Neste trabalho devemos criar um simulador para uma rede de computadores onde pacotes chegam em uma conexão contínua com distribuição exponencial, assim conexões de pacotes igualmente espaçados são abertas também com intervalos de tempo baseados na distribuição exponencial. Por conta da semelhança, tratarei as conexões como sendo do tipo *VoIP*, que geram pacotes do tipo CBR (*Constant BitRate*), para facilitar no entendimento e abstração do sistema.

Os pacotes experimentais que trafegam por essa rede em conexão contínua (mais informações: `https://wand.net.nz/pubs/19/html/node33.html`) têm tamanhos de 40Bytes, 550Bytes ou 1500Bytes, sendo as chances de serem gerados de 40%, 50% e 10% respectivamente. Os pacotes das conexões VoIP, chamados CBR, são gerados pelas conexões durante a duração da mesma, sendo assim um pacote é instantaneamente gerado na chegada de uma conexão e durante a duração da conexão com um intervalo fixo, sendo outro pacote gerado no tempo exato de fechamento da conexão.

Devemos ao fim, demonstrar matematicamente o quão correto o sistema se porta dada sua implementação. Essa demonstração será feita baseando-se no princípios da utilização de conexão e Lei de Little. A utilização possui em sua fórmula parâmetros de entrada do sistema que podem ser ajustados para se obter confirmação, enquanto a Lei de Little nos fornece métricas que são relacionadas com estruturas utilizadas pelo algoritmo na simulação, como a fila.

Assim sendo, o trabalho busca mostrar a importância de validações matemáticas para sistemas de escopo simulatório. O relatório é dividido em três etapas: explicação técnica e teórica do trabalho, validações e experimentos. O código fonte completo pode ser acessado em: `https://github.com/tobiasjc/router_cbr_voip`

2 Distribuição Exponencial

Como descrito, a entrega de pacotes em uma conexão contínua e a criação de novas conexões VoIP seguem uma distribuição exponencial. Esse tipo de distribuição aparece muito no mundo real quando olhamos para séries de eventos igualmente propensos a acontecer mas que não seguem um padrão de intervalo de tempo. A FDP (Função de Distribuição de Probabilidade) cumulativa da distribuição exponencial é dada pela Equação 1 abaixo:

$$F(x) = P(X \leq x) = 1 - e^{\lambda x} \tag{1}$$

Onde X é uma variável aleatória a ser associada a uma probabilidade, logo $P(X = x) = e^{\lambda x}$. O parâmetro λ, também chamado de taxa (de chegada de um evento, no nosso contexto), controla o quão íngreme é a ascensão de $F(x)$, como mostrado abaixo na Figura 1:

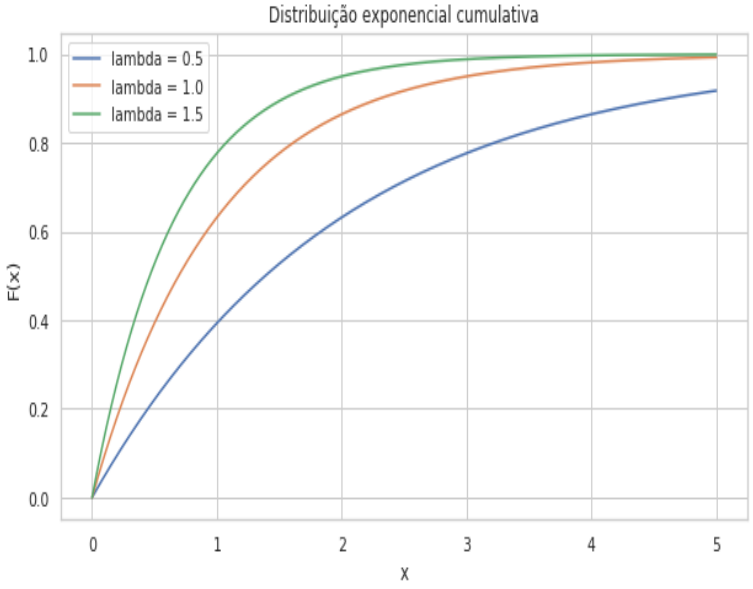

Figura 1: FDP acumulada, F(x), para diferentes valores de lambda, λ, com $x \in [0, 5]$.

A discussão é importante pois nos leva a formulação de dois parâmetros de entrada do sistema: o tempo médio entre pacotes em conexão contínua e o tempo médio entre chegada de conexões VoIP, tal qual uma função que calcule o tempo para a chegada do próximo evento. Essa função, $T(u)$, é definida na Equação 2 a seguir:

$$\begin{aligned} P(X \leq x) &= 1 - e^{(-\lambda x)};\ \text{fazendo-se } P(X \leq x) = u \\ u &= 1 - e^{(-\lambda x)} \\ u - 1 &= -e^{(-\lambda x)};\ \text{uma vez que } u \in [0, 1] \\ u &= -e^{(-\lambda x)} \end{aligned} \tag{2}$$

$$\begin{aligned}
\ln u &= -\ln e^{(-\lambda x)} \\
\ln u &= -\lambda x \\
x &= -\lambda^{-1} \ln u \\
T(u) &= -\lambda^{-1} \ln u; \{\forall u \in \mathbb{R} | 1 \geq u \geq 0\}
\end{aligned}$$

Com essa característica e formulação em mente, podemos focar agora nos requintes estruturais do sistema.

3 Abstração do Sistema

Para facilitar o entendimento e posterior implementação do sistema, imaginemos o esquema a seguir na Figura 2:

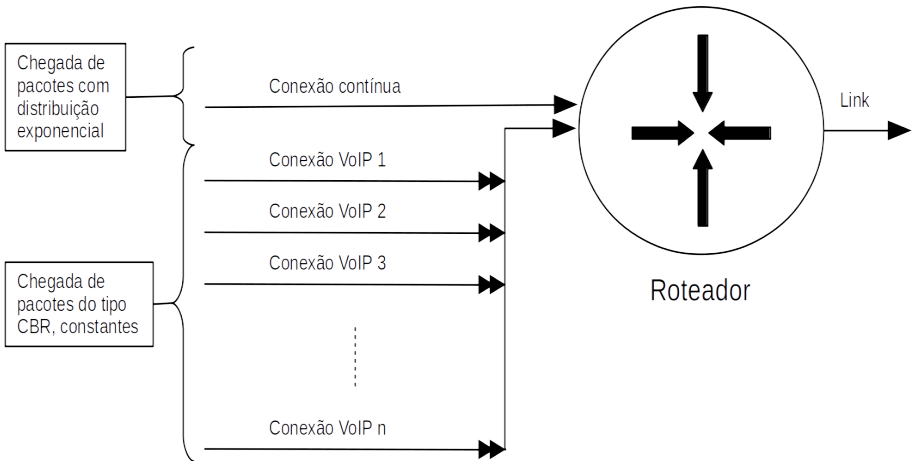

Figura 2: Esquema do sistema construído.

Temos então uma simulação como descrita, onde várias conexões do tipo VoIP de tempo limitado enviam pacotes CBR, e uma conexão contínua permanece enviando pacotes em uma distribuição exponencial. Pacotes a serem enviados pelo link devem entrar no roteador para serem devidamente manejados.

As várias figuras formadoras do esquema da Figura 2 possuem mecanismos para desempenhar a sua função de organizar a operar sobre a transmissão dos pacotes dentro do sistema, como pode ser visto na Figura 3 abaixo que explica o sistema numa visão das operações sobre pacotes:

Com essa visão podemos imaginar diversos mecanismos controladores de cada parte do sistema onde certa sincronização é necessária ou ações tomadas são inerentemente comparáveis. Essa visão é essencial para a construção do algoritmo de forma concisa e com reduzida chance de falhas.

Imaginando-se estruturas e otimizando as operações, estamos agora prontos para iniciar a implementação do algoritmo.

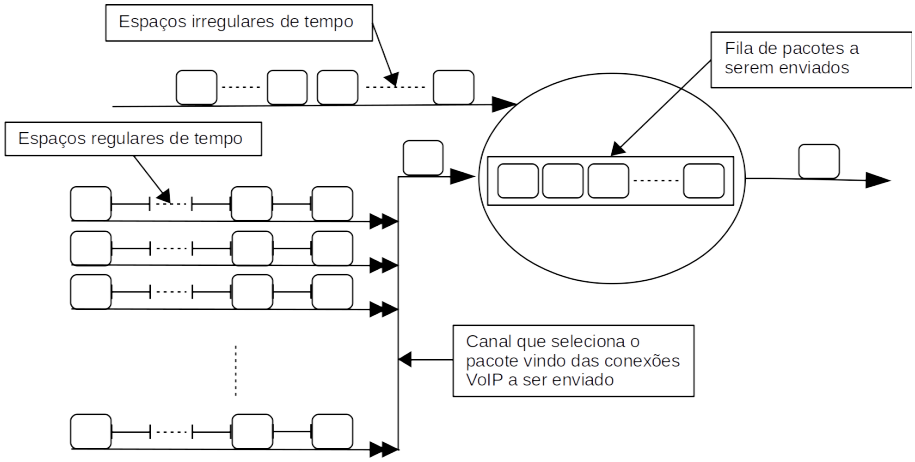

Figura 3: Sistema em uma visão de pacotes e mecanismos.

4 Algoritmo de Simulação

Dada nossa formulação, o código a ser escrito em C nos permite a modularização do código, tornando nossa implementação concisa e reutilizável em várias partes do código onde forem necessárias. A próximas subseções explicação as partes da implementação, discernindo partes que possam causar confusão. Certa ordem pode facilitar o entendimento e implementação do código.

As entradas do sistemas são todas feitas em *bytes* e *segundos*, exceto pelo tamanho do link que é feito em *Megabits*, então essas medidas devem ser normalizadas dentro do sistema. Para chegada de pacotes e conexões, devemos colocar o intervalo esperado entre eventos, sendo assim devemos converter intervalos e taxa dentro do algoritmo tendo em mente que $(taxa) = (intervalo)^{-1}$.

Comecemos pela parte crucial do sistema que é a geração dos eventos e pacotes, dando base do que será o sistema, passando para a implementação das estruturas como roteador, conexão VoIP e canal de seleção de conexão VoIP. A linguagem C nos impõe a difícil tarefa da modularização e não da objetificação do código, assim devemos juntar elementos dos módulos de maneira a se chegar na ideia de um objeto.

4.1 Geração de eventos

Os eventos do sistema serão apenas dois: chegada de pacote e chegada de conexões, seguindo a distribuição exponencial. Pela fórmula dada na equação 2, podemos criar a função $T(u)$ como em:

```
double gen_factor() {
    double u = random() % RAND_MAX;
    u = u / RAND_MAX;
    return (1.0 - u);
}
```

Geração de fator randômico u

```
double gen_arrival(double l) {
    return ((-1.0 / l) * log(
    gen_factor()));
}
```

Cálculo de $T(u)$

Onde a variável l é nosso fator da taxa λ, e a função gen_factor nos dará o fator u claramente com um valor em $[0, 1] \in R$ seguindo uma função randômica.

Para a geração de pacotes em conexão contínua, devemos fazer uma simples função que nos leve as proporções requisitadas pelo experimento como em:

```
double gen_packet_size() {
    double f = gen_factor();
    if (f <= 0.5) {
        return _byte_to_mb(550.00);
    } else if (f <= 0.9) {
        return _byte_to_mb(40.00);
    } else {
        return _byte_to_mb(1500.00);
    }
}
```

Geração de pacotes dadas proporções

```
double _byte_to_mb(double b) {
    return ((b * 8.00) / 1000000.00)
    ;
}
```

Ajuste de medidas dada padronização

Assim, está terminada a implementação para geração de eventos.

4.2 Roteamento de Pacotes

Para o roteamento de pacotes, precisamos abstrair uma estrutura capaz de receber estes pacotes de diversos tamanhos e os enviar pelo *link*, dada uma ordem de chagada. Porém, este link pode estar ocupado no momento da chegada de um novo pacote, e este novo pacote deve então aguardar até a liberação do *link* para seu envio. Esta função é passada à nossa abstração de um roteador.

O roteador então, é uma fila do tipo FIFO (*First In, First Out*) extremamente simples, que deve apenas controlar o envio de pacotes em ordem:

```
typedef struct queue_ {
    int size;
    struct packet_ *first;
    struct packet_ *last;
} queue;
```

Estrutura da fila para roteamento de pacotes

```
typedef struct packet_ {
    double size;
    struct packet_ *next;
} packet;
```

Estrutura de pacotes a serem roteados

E suas duas operações principais, de enfileiramento (*enqueue*) e desenfileiramento (*dequeue*), como a seguir:

```c
void enqueue(queue *q, double s) {
    packet *np = (packet *) malloc(
    sizeof(packet));
    np->size = s;
    np->next = NULL;
    if (q->size == 0) {
        q->first = np;
        q->last = q->first;
    } else {
        q->last->next = np;
        q->last = np;
    }
    q->size++;
}
```

Inserção de novos pacotes na cauda

```c
void dequeue(queue *q) {
    if (q->size == 0) {
        abort();
    }
    packet *tmp = q->first;
    q->first = q->first->next;
    free(tmp);
    q->size--;
}
```

Retirada de pacotes na cabeça

4.3 Conexões VoIP

Teremos então, dado a discussão anterior, uma estrutura formadora de conexão VoIP e uma como espécie de canal seletor para envio de pacotes, assim sendo as mesmas:

```c
typedef struct voip_connection_ {
    double end;
    double next_cbr;
    struct voip_connection_ *next;
} voip_connection;
```

Conexão VoIP

```c
typedef struct voip_tunnel_ {
    int size;
    struct voip_connection_ *head;
    struct voip_connection_ *tail;
} voip_tunnel;
```

Canal de conexões VoIP

Uma análise da implementação será feita nas seções 4.4, 4.5 e 4.6.

4.4 Conexões com envio rotativo

Tenhamos n conexões com duração d, que enviam pacotes p enumerados i em um tempo t de intervalos iguais a λ. Tendo-se uma primeira conexão K e $(n-1)$ conexões L, após o envio da conexão K, fica evidente que seu próximo pacote, $p_k(i+1)$ ocorrerá num tempo $(t_K + \lambda) > t_L$. O diagrama abaixo demonstra o ocorrido:

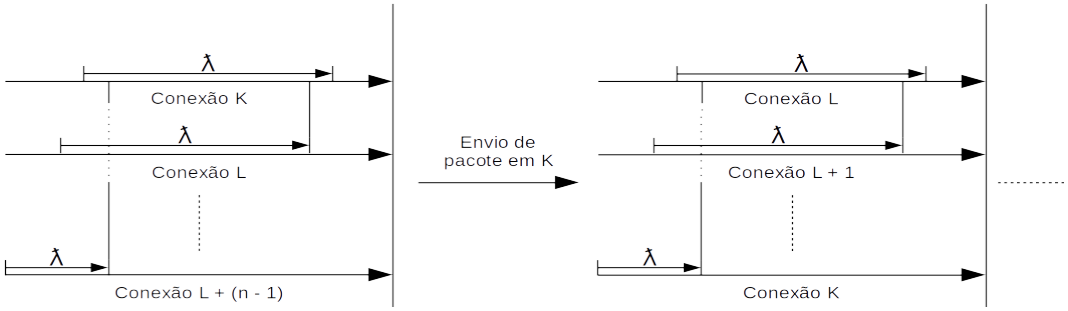

Figura 4: Circulação natural de conexões pelo canal de conexões VoIP

Temo então que este canal funciona de forma parecida com uma fila FIFO já implementada anteriormente, porém este não maneja pacotes e sim conexões e suas operações. A implementação da inserção de novas conexões e inserção de conexões em final de canal podem ser vistas abaixo:

```
voip_connection *
    create_voip_connection(double
    cbr_interval, double time,
    double duration) {
     voip_connection *tmp = (
    voip_connection *) malloc(sizeof
    (voip_connection));

     tmp->end = time + gen_arrival(
    duration);
     tmp->next_cbr = time +
    cbr_interval;
     tmp->next = NULL;

     return tmp;
}
```

Inserção de nova conexão VoIP em canal

```
void insert_tail_connection(
    voip_tunnel *vt, voip_connection
     *vc) {
     if (vt->size > 0) {
         if (vt->tail != NULL) {
             vt->tail->next = vc;
             vt->tail = vc;
             vc->next = NULL;
         } else {
             vt->tail = vc;
             vc->next = NULL;
         }
     } else {
         vc->next = NULL;
         vt->head = vc;
     }
     vt->size += 1;
}
```

Inserção de conexão VoIP no fim do canal

4.5 Finalizando conexões (DESCONTINUADO[1])

Uma conexão VoIP deve enviar um pacote no momento de seu fechamento. Assim, pode ocorrer deste momento não ser perfeito caso $\frac{d}{\lambda} = q + r$, onde $q \in \mathbb{Z}$ e $r \neq 0$.

Tenhamos então, mais uma vez, n conexões com duração d, que enviam pacotes p enumerados i em um tempo t de intervalos iguais a λ. Após o envio do pacote atual, a conexão K alocará seu próximo envio, porém $(t_k + \lambda) > d$. Façamos com que $t_k = d$, e isto fará com que a relação de rotatividade anterior de séries temporais seja desfeita. Logo, deveremos achar a posição de inserção que atenda $L_a < (t_K + \lambda) < L_b$, onde então $a < b$. O diagrama abaixo demonstra o explicado:

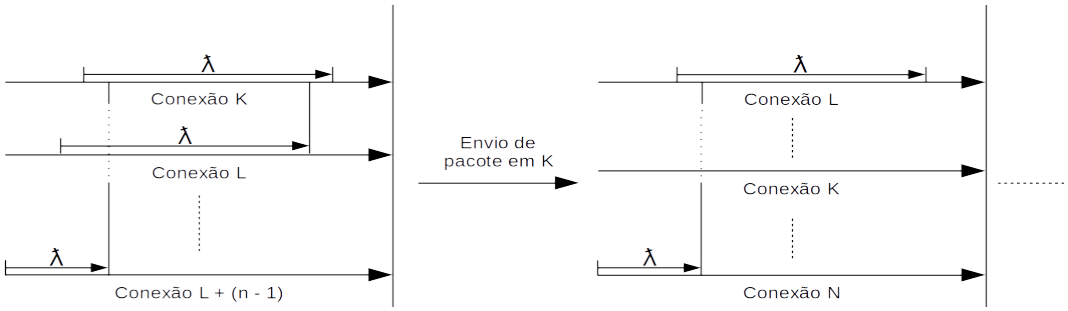

Figura 5: Circulação de conexões com fechamento da conexão K entre outras conexões

[1]Lógica não executada. Manterei a descrição para futuras possíveis implementações TCP.

Sendo assim, o techo de código a seguir mostra a implementação desta inserção:

```
void insert_ending_connection(voip_tunnel *vt, voip_connection *vc) {
    if (vt->size > 0) {
        if (vc->next_cbr < vt->head->next_cbr) {
            vc->next = vt->head;
            vt->head = vc;
        } else {
            voip_connection *pre = vt->head;
            voip_connection *fwd = vt->head->next;
            while ((fwd != NULL) && (fwd->next_cbr <= vc->next_cbr)) {
                pre = fwd;
                fwd = pre->next;
            }
            if (fwd == NULL) {
                pre->next = vc;
                vt->tail = vc;
                vc->next = NULL;
            } else {
                pre->next = vc;
                vc->next = fwd;
            }
        }
    } else {
        vt->head = vc;
        vc->next = NULL;
    }
    vt->size += 1;
}
```

Inserção de conexão VoIP a ser finalizada em posição ordenada

4.6 Atendimento pacotes

Dadas as implementações das seções 4.4 e 4.5 anteriores, implementar o atendimento do pacote se torna trivial, pois no canal sempre será apontado o pacote certo no momento exato de ser enviado, facilitando assim todo o mecanismo da simulação. A implementação para atendimento é então:

```
voip_connection *attend_connection(voip_tunnel *vt) {
    voip_connection *tmp = vt->head;
    vt->size -= 1;
    if (vt->size == 0) {
        vt->head = NULL;
    } else {
        vt->head = tmp->next;
    }
    return tmp;
}
```

Atendimento de pacote em canal de conexões VoIP

Temos assim a implementação de todas as operações necessárias para a simulação, devemos agora elucidar as validações para o sistema.

5 Relações e Validações

A implementação de um sistema de simulação deve conter mecanismos de validação para que erros possam ser constatados e ajustados. Assim, duas validações serão implementadas para esse sistema: a utilização do *link* e a Lei de Little (tempo médio de pacotes no sistema), nas seções 5.1 e 5.2 respectivamente a seguir.

5.1 Utilização

A utilização U, de um *link* L, é dada pelo tempo t_s, levado para se transferir uma dada quantidade de dados, dividido pelo tempo total de ativação do link t_t, entregue então pela fórmula:

$$U = \frac{t_s}{t_t}; 0 \geq (U \in \mathbb{R}) \geq 1 \tag{3}$$

Assim, podemos dividir essa utilização para diferentes fatias de dados do nosso sistema como mostrado nas seções 5.1.1 e 5.1.2.

5.1.1 Conexão contínua

Para a utilização U_c, apenas para os pacotes vindos da conexão contínua, temos que a quantidade de dados enviados por esta parte do sistema depende da taxa de envio dos pacotes p_{rc} e do tamanho dos mesmos p_{sc}, tendo assim que:

$$\begin{aligned} p_{sc} &= (0.4 * 40) + (0.5 * 550) + (0.1 * 1500) \\ t_{sc} &= \frac{p_{sc} p_{rc}}{L} \\ U_c &= \frac{t_{sc}}{t_t} \end{aligned} \tag{4}$$

5.1.2 Conexões VoIP

Para utilização das conexões VoIP U_v, temos que os dados que trafegam nas mesmas são dependentes do tamanho do pacote CBR p_{sv}, a taxa de envio dos mesmos p_{rv}, igualmente à conexões contínuas. Porém agora, temos que estes dados trafegam dentro de conexões com tempo limitado que seguem distribuição exponencial para chegada - logo teremos o tempo de duração da conexão c_{dv}, bem como a taxa de chegada de conexões c_{rv}. A fórmula pode ser montada como abaixo:

$$\begin{aligned} t_{sv} &= \frac{p_{sv} p_{rv} c_{dv} c_{rv}}{L} \\ U_v &= \frac{t_{vc}}{t_t} \end{aligned} \tag{5}$$

5.1.3 Variações para entrada do sistema

Observe que as fórmulas de utilização dadas em 5.1.1 e 5.1.2 utilizam como parâmetro de entrada a taxa de chegada de pacotes e conexões. Deve ser claro que taxa e tempo de intervalo entre acontecimento são medidas inversamente proporcionais.

Assim, para a conexão contínua Fórmula 4, se quiséssemos obter a utilização com parâmetro de intervalo p_{ic}:

$$\begin{aligned} p_{ic} &= \frac{1}{p_{rc}} \\ t_{sc} &= \frac{p_{sc}}{Lp_{ic}} \end{aligned} \tag{6}$$

O mesmo fica evidente para a utilização em conexões VoIP, tendo-se intervalo entre pacotes CBR p_{iv} e c_{iv} em:

$$\begin{aligned} p_{iv} &= \frac{1}{p_{rv}} \\ c_{iv} &= \frac{1}{c_{rv}} \\ t_{sv} &= \frac{p_{sv}c_{dv}}{Lp_{iv}c_{iv}} \end{aligned} \tag{7}$$

5.1.4 A fórmula total

Para fins didáticos e avaliativos, temos a fórmula total da utilização do sistema, expressa por U_t, que á nada mais que a soma da utilização pela conexão contínua e conexões VoIP, sendo assim:

Parâmetros em taxa U_{rt}

$$\begin{aligned} t_{st} &= t_{sc} + t_{sv} \\ t_{st} &= \frac{(p_{sc}p_{rc}) + (p_{sv}p_{rv}c_{dv}c_{rv})}{L} \implies \\ U_{rt} &= \frac{t_{st}}{t_t} \end{aligned} \tag{8}$$

Parâmetros em intervalo U_{it}

$$\begin{aligned} t_{st} &= t_{sc} + t_{sv} \\ t_{st} &= \frac{(p_{iv}c_{iv}p_{sc}) + (p_{ic}p_{sv}c_{dv})}{Lp_{ic}p_{iv}c_{iv}} \implies \\ U_{it} &= \frac{t_{st}}{t_t} \end{aligned} \tag{9}$$

5.1.5 A utilização medida no sistema

A implementação da utilização é muito simples, necessitando apenas de uma variável que soma o tempo de qualquer iteração do sistema com o *link* - ou seja, no nosso caso, envio de pacotes pelo *link*. Sendo assim, para uma variável t_s utilizada para validação do sistema ,processando um pacote de tamanho p_s e link de capacidade L, a cada momento necessário deve ser executado:

$$t_s = \sum \frac{p_s}{L} \tag{10}$$

E ao final, essa variável, t_s, deve ser dividida pelo tempo de execução. Não há estruturas para se executar tal verificação, somente uma variável.

5.2 Lei de Little

A Lei de Little é utilizada em sistemas que possuem espera para atendimento de algum tipo de evento de qualquer natureza, utilizando uma simples relação de produto entre o tempo de permanência médio W, para atendimento, e a taxa de chegada de eventos λ, nos dando um valor L sendo o número médio de eventos dentro do sistema durante sua operação. Logo:

$$L = \lambda W \tag{11}$$

No nosso sistema, o evento a ser acompanhado é a chegada e saída de pacotes. Esse acompanhamento pode ser feito imaginando-se histogramas nos indicando o número de pacotes no sistema num determinado momento de execução. Imaginemos então como exemplo, um histograma como abaixo, onde cada pacote tem um tempo de processamento de 1 unidade de tempo:

5.2.1 Obtendo-se L

Pelo histograma da Figura 6, podemos constatar a variável L da Lei de Little, que é a área do histograma formado divida pelo tempo de observação dos eventos. Tendo-se t unidades de tempo total, onde um dado atendimento de pacote demora um tempo t_i com uma quantidade q_i de pacotes em espera, a área de cada retângulo é dada por $A_t = t_i q_i$. Temos então a Equação 12 para L abaixo:

$$L = \frac{1}{t} \sum_{i=0}^{t} A_i \tag{12}$$

Que no nosso exemplo do histograma onde $t_i = 1, \forall i \in [0, t]$, a Figura 6 nos dá:

$$L = \frac{1}{6}(1 * 1 + 1 * 2 + 1 * 3 + 1 * 2 + 1 * 3 + 1 * 2) = \frac{13}{6}$$

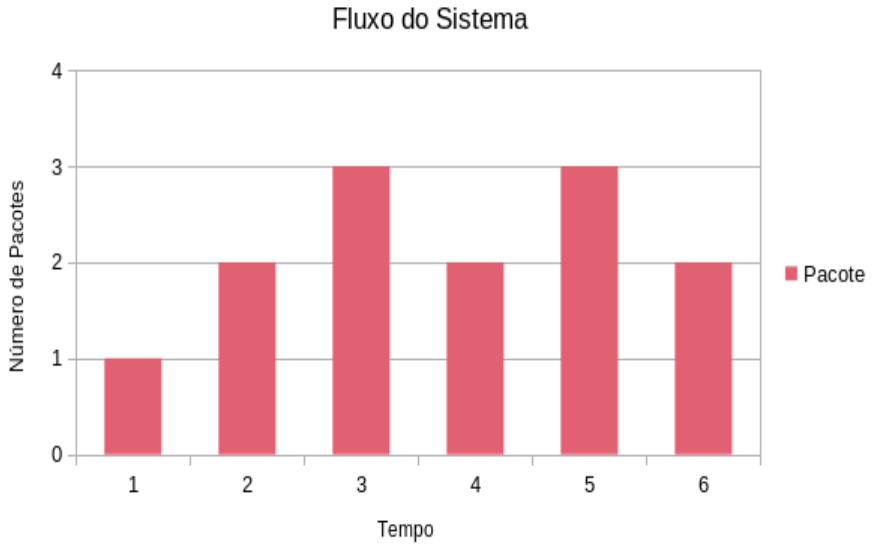

Figura 6: Histograma de uso do sistema com número de pacotes no sistema em função do tempo

5.2.2 Obtendo-se λ

A descrição dada para λ é auto explicativa quando se olha para o histograma da Figura 6. Definindo-se um sistema para tempo de simulação igual a t, onde eventos de chegada de pacotes $p_{\lambda t}$ acontecem, temos que a média de chegada de pacotes no sistema é explicada pelo histograma da Figura 5.2.2 e dada pela Equação 13:

$$\lambda = \frac{1}{t}\sum_{i=0}^{t} p_{\lambda i} \qquad (13)$$

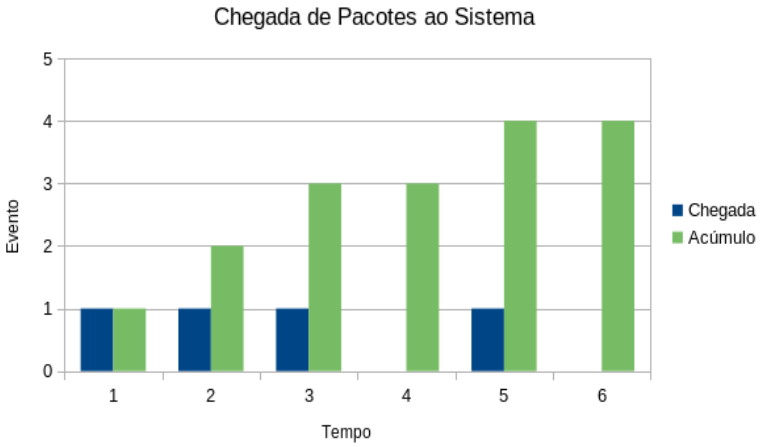

Figura 7: Histograma para chegada de pacotes no sistema, com valores de chegada e valores acumulados

Que no nosso exemplo do histograma da Figura 6 nos dá:

$$\lambda = \frac{1}{6}(1+1+1+0+1+0) = \frac{4}{6} = \frac{2}{3}$$

5.2.3 Obtendo-se W

Dada a definição de W, temos que para um sistema com tempo de simulação t, onde se define l como o tempo de permanência do pacote no sistema, tal qual $l = t_b$ - t_a, sendo t_b o tempo de saída e t_a o tempo da chegada, e sabendo-se que q pacotes passaram pelo sistema, podemos definir a variável então como:

$$W = \frac{1}{q} \sum_{i=0}^{t} p_{wl} \qquad (14)$$

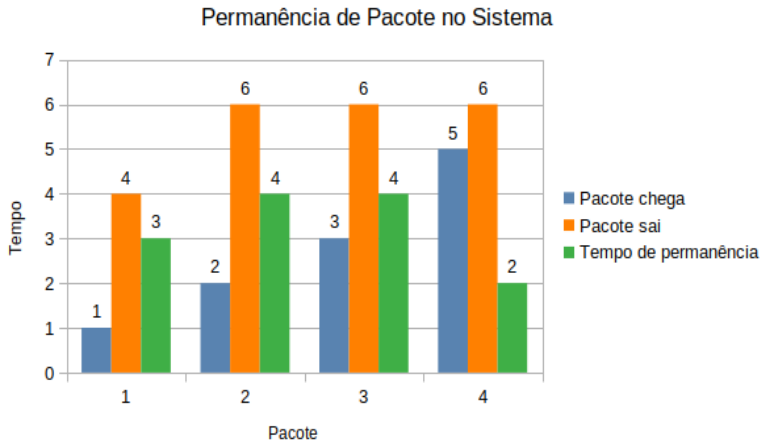

Figura 8: Histograma para chegada de pacotes no sistema, com valores de chegada e valores acumulados

Nos retornando então um valor de:

$$W = \frac{1}{4}(3 + 4 + 4 + 2) = \frac{13}{4} \qquad (15)$$

5.2.4 Validação

Para fins didáticos e de avaliação, temos que os valores do nosso sistema estarão corretos, quando tivermos a validação da fórmula. Assim sendo, com os valores do exemplo temos:

$$\begin{aligned} L &= \frac{13}{6} \\ &= \lambda W \\ &= \frac{2}{3}\frac{13}{4} = \frac{26}{12} = \frac{13}{6} \implies \\ L &= \lambda W \end{aligned} \qquad (16)$$

6 A Lei de Little executada no sistema

Precisamos de uma estrutura que some iterativamente a área de gráfico criada pelos retângulos. Assim, a cada ação que execute ações de chegada ou saída de pacotes do sistema, bem como seus atendimentos, resultarão em mudanças nas estruturas a serem processadas.

Podemos utilizar uma estrutura acumuladora e função para soma e subtração, como as descritas abaixo, para lidar com a abstração de histograma:

```
typedef struct accumulator_ {
    double past_time;
    double area;
    double packets_count;
} accumulator;
```

Estrutura acumuladora de operações sobre histograma

```
void cumulative_sum(accumulator *t,
    double time, int up) {
    t->area += t->packets_count * (
    time - t->past_time);
    t->past_time = time;

    if (up == 1) {
        t->packets_count++;
    } else if (up == 0) {
        t->packets_count--;
    }
}
```

Função para operações de sibtração ou adição de valores em em estrutura de histograma

É de se observar que para obtenção do valor de L, estas estruturas e operações nos darão o valor logo ao final da operação, uma vez que durante a execução vamos somar e subtrair pacotes valores indicando pacotes na estrutura, necessitando-se apenas dividir os valores pelo tempo total de simulação ao final.

Para o valor de λ algo muito parecido acontece, nós necessitaremos apenas somar valores na estrutura quando um novo pacote chegar ao sistema, e ao final, dividir pelo tempo total de simulação ao final.

Para o valor de W, operações diferentes são necessárias. Para se conseguir o tempo de permanência de um pacote no sistema, devemos (tendo em vista o histograma da Figura 5.2.3), fazer uma subtração de valores chegada e saída de pacotes, e dividir esse valor pelo número de pacotes que passaram pelo sistema. Isso pode ser feito manejando-se apenas o valor da "area"em ambas as estruturas de contagem.

7 Testes de utilização

Como nos foi pedido, testes de utilização devem ser executados. Os resultados e seus parâmetros de entrada para o sistema são apresentados na tabela abaixo:

U_t	U_v	U_c	t(s)	p_{sv}(B)	p_{iv}(s)	$c_{dv}(s)$	c_{iv}(s)	p_{ic}(s)	L(Mb/s)	res(s)
0.4	0.1	0.3	10452	380	0.02	10	0.116923	0.000090	130	0.400924
0.6	0.5	0.1	35000	1200	0.01	50	0.592593	0.000218	162	0.599887
0.8	0.52	0.28	22400	1777	0.1	100	1.518803	0.000700	18	0.801814
0.9	0.13	0.77	1220	9563	2	30	0.044137	0.000023	200	0.902162
0.95	0.45	0.5	104570	299	5	200	0.007087	0.000235	30	0.954750
0.99	0.12	0.87	167000	1239	1	150	0.065904	0.000022	188	0.990392

Tabela 1: Tabela com resultados de simulações e seus parâmetros usados. Um arquivo para gerar resultados da tabela pode ser encontrado juntamente do código do projeto.

As variáveis utilizadas são as mesma utilizadas durante todo o material, sendo resumidas em:

U_t Utilização total do sistema pela conexão contínua e diversas conexões VoIP.

U_v Utilização do sistema apenas pela conexão VoIP.

U_c Utilização do sistema apenas pela conexão contínua.

$t(s)$ Tempo de simulação em segundos.

$p_{sv}(B)$ Tamanho de pacotes CBR para conexão VoIP em Bytes.

$p_{iv}(s)$ Intervalo entre pacotes CBR para conexão VoIP em segundos.

$c_{dv}(s)$ Duração média de conexão VoIP em segundos.

$c_{iv}(s)$ Intervalo médio entre conexões VoIP em segundos.

$p_{ic}(s)$ Intervalo médio entre pacotes em conexão contínua em segundos.

$L(Mb/s)$ Tamanho do *link* em Megabits por segundo.

res Resultado conseguido à saída do sistema.

8 Conclusão

Este trabalho trás uma visão ainda não abordada no curso sobre sistemas e simulação. Conceitos aqui utilizados foram aprendidos em outras disciplinas mas nunca aplicados, como foi requisitado. A importância da esquematização do sistema antes da tentativa de implementar fica evidente quando utilizamos conceitos matemáticos para simular eventos. Espero mais trabalhos como este.