信州大学工学部

学士論文

ハフマン符号を用いた伝送システムにおける ポアソン到着シンボルの遅延

指導教員　西新 幹彦 准教授

学科　電気電子工学科
学籍番号　05T2035F
氏名　小泉 太生樹

2009 年 2 月 28 日

目次

1　序章

1.1　背景

インターネットなどの情報通信システムにおいてそのデータ量は莫大である．不規則にデータが発生する場合もある．例として，降水量の測定や交通量の測定などのシステムが挙げられる．これらの莫大なデータをそのまま伝送することは不可能であることから，誤りがなくかつ効率よく伝送するにはどうすべきかが問題となる．本研究ではデータを圧縮し，かつ遅延を最小限にすることにより最適にデータを伝送することを考える．符号化方法は，いくつかあるがその中で，平均符号語長が最小になるハフマン符号 [1] があり，これにより遅延も最小限に留めることができるのではないかと予想される．まずはじめに本研究の概要について述べる．

1.2　研究の概要

本研究では，パケットの到着間隔が指数分布に従うというポアソン到着でシンボルを発生させる．ハフマン符号は，ポアソン到着のパケットの出現頻度をあらかじめ知ることができるエンコーダによって，符号語シンボルを生成し圧縮して，これらの符号語を送信バッファに送り，そこからデコーダに送る．デコーダは，符号語シンボルを受け取り，符号語が揃うと復元する．本研究において符号語シンボルは 0 と 1 とする．本研究で考えたハフマン符号のシステムのモデルを図 1 に示す．

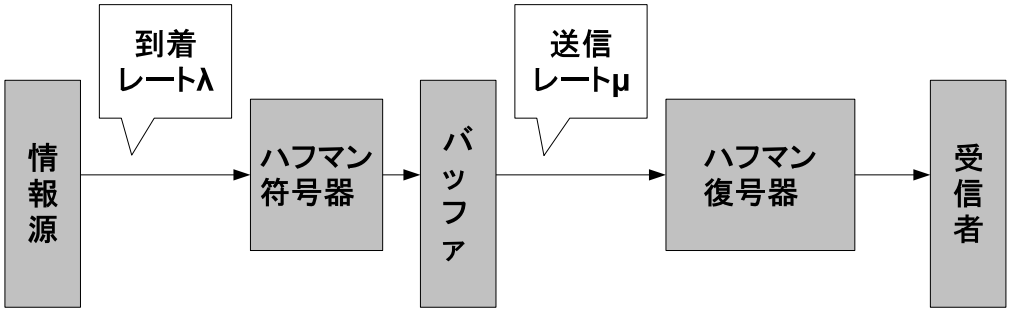

図 1　ハフマン符号のシステム

図 1 で用いた λ は入力シンボルの到着レートであり，μ は送信バッファレートである．λ と μ を両方とも大きくしてもシステム全体の処理速度を高めるだけなので，送信レート μ は 1(シンボル/秒) で固定する．なお λ の単位についても (シンボル/秒) とする．

従来研究 [4] に算術符号を用いた伝送システムがあり，今回の研究ではその算術符号をハフ

マン符号に置き換えたものである．算術符号は 1 シンボルずつ符号化していくシステムであったが，ハフマン符号ではシンボルをブロック化して符号化することが特徴である．ハフマン符号を用いると平均符号語長が最小になることから，平均遅延は小さく抑えられるのではないかと予想される．

本研究における遅延とは，一つのシンボルがエンコーダに入力されて符号語となり送信バッファに送られて，そこからデコーダに送られシンボルが復元されるまでに要する時間をいう．入力シンボルの到着レート λ がある値より大きいと，送信バッファ内に符号語が蓄積されてここで遅延が生じてしまう．そして送信バッファに到着する符号語が多すぎると, どんなに大きい送信バッファを用いても最終的には送信バッファに符号語が入りきらなくなってしまう．

本研究においてこの状態をパンク状態と定義する．以下にバッファがパンクしないための条件式を記す．

$$0 < \lambda \overline{L}/n < \mu \tag{1}$$

より，この実験では送信レート μ を 1 で固定しているので，この式を変形すると

$$0 < \lambda < n/\overline{L} \tag{2}$$

となる．λ を送信バッファがパンクしない範囲で動かす．なお，$\overline{L}$ は平均符号語長である．平均符号語長は以下のように表される．

$$\overline{L} = \sum_{i=1}^{n} p_i l_i$$

この研究では情報源のシンボルを簡単化のため 0，1 とする．シンボルの数は百万個とし，ブロック化してからハフマン符号化する．

2 遅延

伝送システムを用いて情報を送る場合，そこには必ず遅延が生じる．しかし遅延は少ないほうが情報を送るほうにとっても, 受け取るほうにとっても良いと言えるはずである．よって遅延がより小さいシステムは優れたシステムということになる．本研究における遅延とは，一つのシンボルがエンコーダに入力されて符号語となり送信バッファに送られて，そこからデコーダに送られシンボルが復元されるまでに要する時間をいう．この研究では以下で述べる平均遅延を如何に小さくするかということにまず着目する．そして次に，あるユーザーがいて，そのユーザーがある程度の遅延を許容する場合を考える．その遅延の許容値までの範囲内でどれだけシンボルの到着レート λ を大きくできるかということを考える．つまり平均遅延とシンボルの到着レートの関係を知り，平均遅延がある値のときにどこまで λ を大きくできるかということを目的としている．

この研究では，符号器の手前で最初に決めたブロックになるまでの待ち時間，バッファでの待ち時間，符号語を送信するための送信にかかる時間，計 3 つの遅延が生じる．またブロック一つ一つの遅延はポアソン到着の影響により確率過程となっている．

平均遅延とは，百万個のシンボル一つ一つの遅延の値の平均を取ったものである．

上記で述べたようにブロックごとの遅延は確率過程となっており，遅延を単なる値として見るために平均遅延を取る必要がある．

3 ポアソン到着

ポアソン到着とは，ランダムに到着するシンボルを表す確率過程のひとつである．ポアソン到着のシンボルの到着間隔は指数分布にしたがう．

ポアソン到着 [3][4] は，シンボルの到着が次の 3 つの条件を満たすと仮定した 1 つの到着の仕方のモデルである．到着間隔は指数分布に従う．ポアソン到着は以下の性質をもつ．

- 定常性
  $t > 0$, $x \geq 0$ なる任意の実数 t，任意の整数 x に対し，時間間隔 $(a,\ a+t)$ の間に x 人到着する確率は，全ての a について同一で，$v_x(t)$ と表すことができる．つまり，時間区間の幅が同じであれば，どこをとってもシステムの確率的状態は同じである．この性質を定常性という．
- 独立性
  上記に記した $v_x(t)$ は，時刻 a までにどれだけきたか，またいつ来たかには無関係である．もし前に到着したシンボルの時刻や個数によってこの確率 $v_x(t)$ が影響を受けるならば，前のシンボルはあとに残る影響，つまり残留効果をもつことになるが，そういうことがないというのが，この独立性である．
- 希少性
  $\psi(t) = \sum_{x=2}^{\infty} v_x(t)$ と置くとき，t が十分小さければ $\psi(t)$ は無視してよい．つまり，

  $$\psi(t) = o(t) \quad (t \to 0)$$

  が成り立つ．この式は，

  $$\lim_{t \to 0} \frac{\psi(t)}{t} = 0$$

  を意味する．$o(t)$ とは t が十分小さければ無視してよい程度であることを示している．つまり $\psi(t) = o(t)$ は，2 つ以上のシンボルが同時に連れだってこないことを意味する．

4　ハフマン符号

ハフマン符号はシンボルをブロック拡大して符号化できることが特徴である．
またハフマン符号は平均符号語長が最小になるという点で最適である．

4.1　ハフマン符号の符号化

ハフマン符号の構成法は次のとおりである．

1. 最も確率の小さい二つのシンボルを選び，それらを子供にもつ新しい節点を作る．そしてその節点に二つの確率の和をつける．そして節点と子供をつなぐそれぞれの枝に 0, 1 のラベルをつける．
2. 新しく作った節点をシンボルとみなす．
3. 1，2 番を新しく作った節点の確率が 1 になるまで繰り返す．
4. 根から出発してシンボルまで枝をたどることによって得られるラベルの列をそのシンボルの符号語とする．

図 2 にてハフマン符号化の例を示す．

4.2　ハフマン符号の復号

復号器は符号シンボルをひとつずつ受け取り，受け取った符号シンボルの列が，あるシンボルの符号語になったら，そのシンボルを復号する．

4.3　アルファベットの拡大

ハフマン符号はシンボルをブロック拡大して符号化できることが特徴であり，本研究では情報源アルファベット 0,1 を拡大している．以下でその手順を述べる．

アルファベットの n 次の拡大とは，n 個のシンボルをブロック化して 1 つのシンボルとみなすことである．本研究では 0，1 の二進数で行っているので，拡大されたアルファベットの総数は 2 の n 乗個となる．例えば，ブロック長を 3 にすると 000,001,010,011,100,101,110,111 の 8 通りにアルファベットを拡大できる．

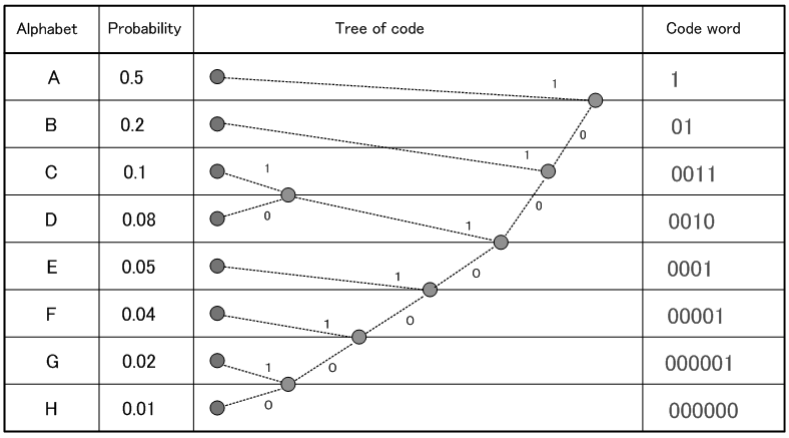

図 2　ハフマン符号化の説明

5　システムにおける平均遅延

5.1　平均遅延の計測

5.1.1　実験内容

ハフマン符号を用いた場合，どれほどの遅延が発生するのかを調べるために平均遅延を調べる実験を行う．上記で述べたようにブロックごとの遅延は確率過程となっており，遅延を単なる値として見るために平均遅延をとり，システムを評価する．

ハフマン符号はシンボルをブロック化して符号化するというのが特徴であり，この実験ではシンボルの発生確率 p を固定し，ブロック長 n を動かして平均遅延を計ることを目的とする．

シンボルの発生確率 p の範囲は 0.01〜0.5 とする．

シンボルの到着レート λ をバッファがパンクしない範囲で動かす．その λ の範囲内で λ の

上限値の 0.1 倍から 0.9 倍までの等間隔の 9 個の点において平均遅延をとる．またバッファがパンクする状態に近づく様子を観察したいので，0.95 倍の点においても平均遅延をとる．またこの実験では符号器とバッファ間に着目し，またグラフの横軸が同じ幅で各グラフの傾向を調べたいので到着レート λ を正規化する．バッファがあふれない条件は 2 ページの式 (2) より，到着レート λ を 0 ～ 1 まで正規化して表す．結果を図 3 と図 4 に示す．

5.1.2 結果と考察

$p = 0.01$ と $p = 0.5$ においてブロック長 n を 1〜5 まで動かしたときの結果を図 3 と 4 にして示す．ブロック長 n が 1 の場合はハフマン符号化しないことと同じなので，n=1 のときの p はどんな値でも変わりない．シンボルの発生確率 p が小さい，つまり情報源の偏りが大きいと平均遅延は小さくなり，偏りが小さい場合，平均遅延は大きくなる．

ハフマン符号は情報源の偏りが大きいと平均符号語長が短くなるので，バッファでの待ち時間，送信時間が小さくなり，このような結果が得られたと考えられる．

また $p = 0.01$ において到着レート λ が約 0.6 以下，$p = 0.5$ においてはブロック長 n が 1 の場合のほうが平均遅延が小さいことがグラフより読み取れる．これはハフマン符号で符号化を行うより，シンボルを符号化せずに素通りさせてしまうほうが平均遅延は小さいということである．

5.2 平均遅延の内訳

5.2.1 実験内容

本研究の伝送システムでは，3 箇所で遅延が発生する．符号器の手前でシンボルが最初に決めたブロック長になるまでの待ち時間，バッファでの待ち時間，符号語を送信するための送信にかかる時間，その 3 つの遅延の特徴がどのようなものかを調べるために，実験を行う．

まずはじめに，シンボルが決められたブロック長になるまでの待ち時間，そして符号語を送信するための送信にかかる時間の理論式を導く．そして上記で得られた平均遅延の値から 2 つの遅延の理論式を引き，バッファでの待ち時間を導く．理論式は以下に示す．

$$(\text{待ち時間の期待値}) = \frac{n-1}{2\lambda}$$
$$(\text{送信時間の期待値}) = \frac{\overline{L}}{\mu}$$

ここでは $n = 5$ における遅延の内訳を以下に示す．この実験も先の実験と同じように到着レート λ を正規化して表す．結果を図 5 と 6 に示す．

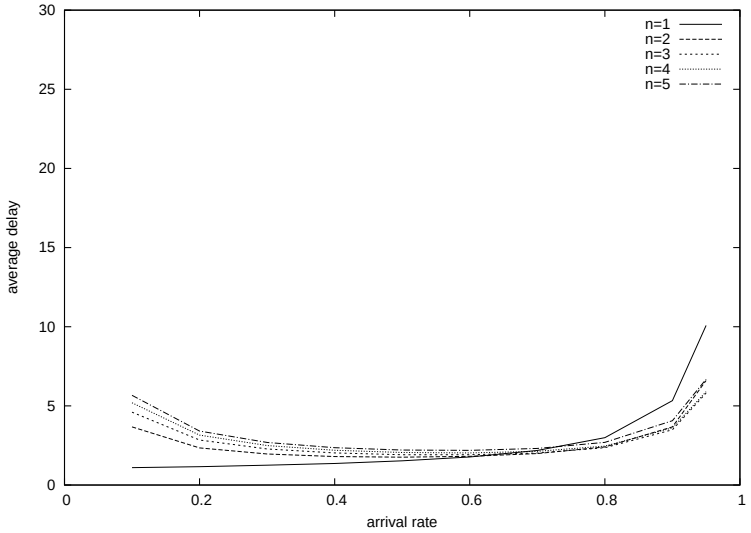

図 3　平均遅延 p=0.01 の場合

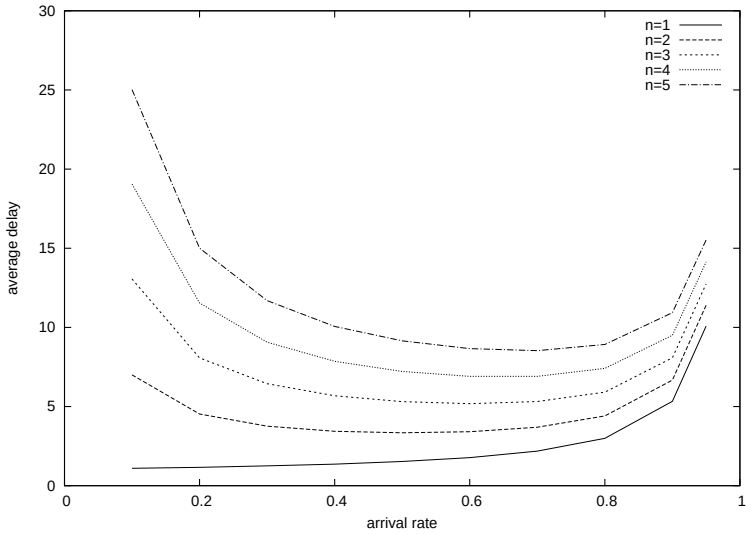

図 4　平均遅延 p=0.5 の場合

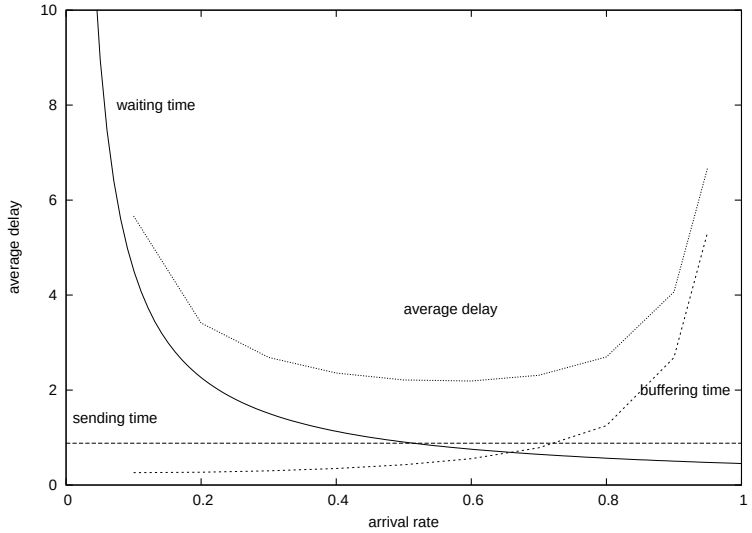

図 5　平均遅延の内訳 p=0.01，n=5 の場合

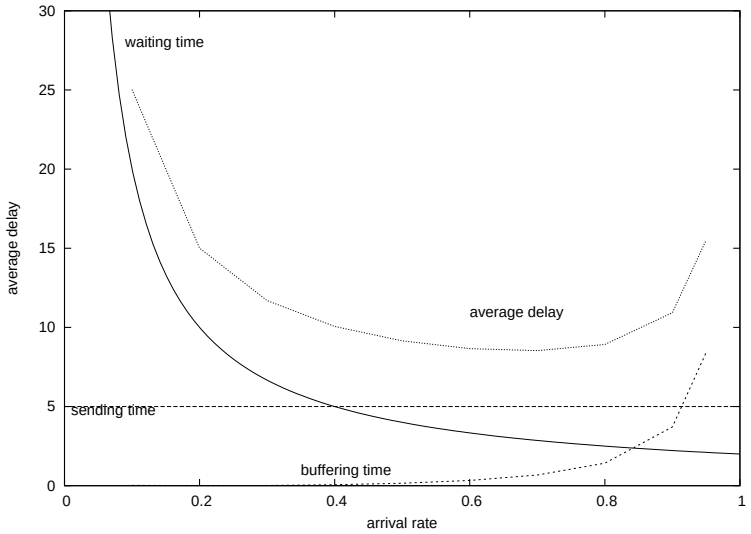

図 6　平均遅延の内訳 p=0.5，n=5 の場合

5.2.2 結果と考察

まずグラフ上の waiting time，すなわちシンボルが決められたブロック長になるまでの待ち時間について述べる．到着レート λ が小さいとシンボルの到着頻度が低いので，待ち時間は大きくなることがわかる．そして到着レート λ が大きいとシンボルの到着頻度は高くなるので，待ち時間はその分小さくなる．

次にグラフ上の sending time，すなわち送信時間について述べる．送信時間は送信レート μ が一定で，かつ平均符号語長も一定なので変化しない．またシンボルの発生確率 p が小さいと平均符号語長は短くなるので，送信時間は小さくなる．

最後にグラフ上の buffering time，すなわちバッファでの待ち時間について述べる．到着レート λ が小さいとシンボルの到着頻度が低いので，バッファでの待ち時間は小さくなることがわかる．そして到着レート λ が大きいとシンボルの到着頻度は高くなるので，バッファでの待ち時間はその分大きくなり，λ が正規化した値の上限 1 に近づくと発散することがわかる．

5.3 平均遅延－入力レート λ 導出の実験

5.3.1 実験内容

ハフマン符号化伝送システムを使用するときに平均遅延の許容範囲を決め，そのときにどこまで入力レート λ，アルファベットの拡大を大きくできるかということが本研究の最終目標である．

この実験では平均遅延の許容範囲を決め，そのときにどこまで到着レート λ を大きくできるかという事を行う．シンボルの発生確率 p を 0.5〜0.001 の範囲にして行う．またブロック長 n の最大値を 6 とする．

平均遅延の計測で得られた結果を用いて，平均遅延―到着レート λ のグラフを作成する．結果を図 6，7，8，9 に示す．

5.3.2 結果と考察

p=0.5，0.4 の場合ではいかなる平均遅延の値においてもブロック長 n が 1，すなわち符号化しないほうが良く，そのとき到着レート λ を最大にできる．符号化したほうがメリットが出てくるのは p が 0.3 以下のときである．しかし p=0.3 では平均遅延においてかなり妥協しなければならず，実用的とは言い難い．そして p が小さくなるにしたがって符号化のメリットが顕著に現れ，p=0.001 においてはかなりのメリットがあるということを確認できる．

またこの実験では簡単化のためにシンボルを 0，1 としているが，シンボルの数を増やした場合にも同じようなグラフの形になる，つまり情報源の偏りが大きいとブロック長 n を増やすに従って到着レート λ を大きくできると予想される．

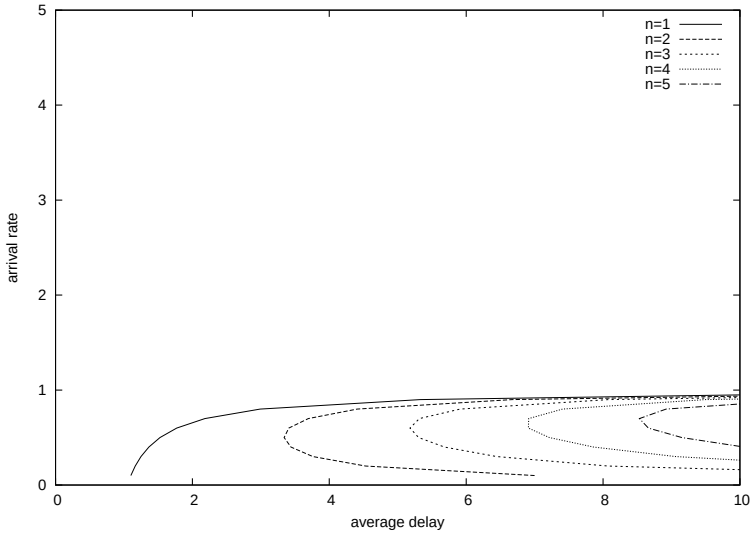

図 7　p=0.5 の場合

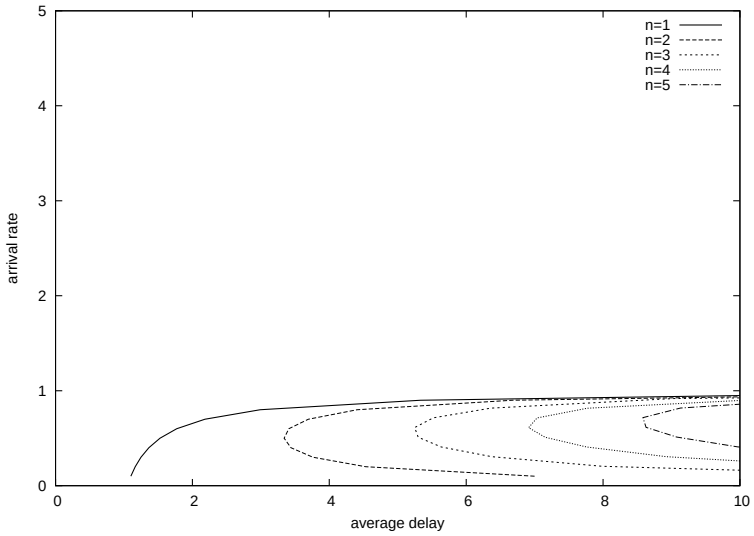

図 8　p=0.4 の場合

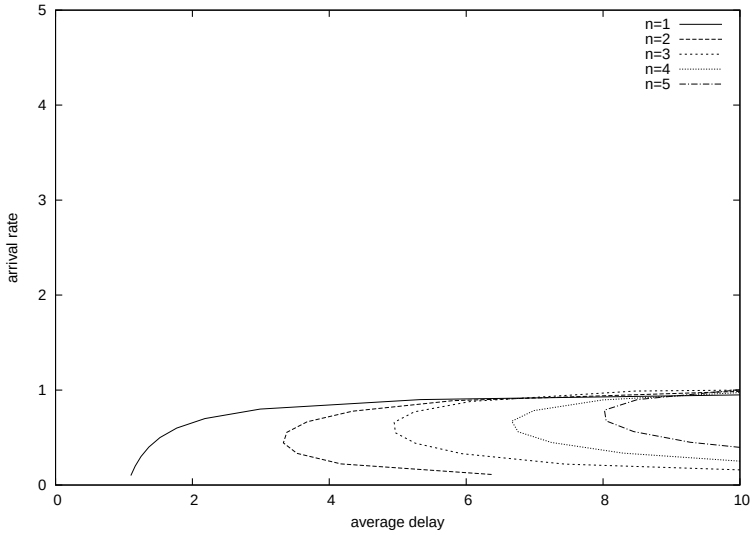

図 9　p=0.3 の場合

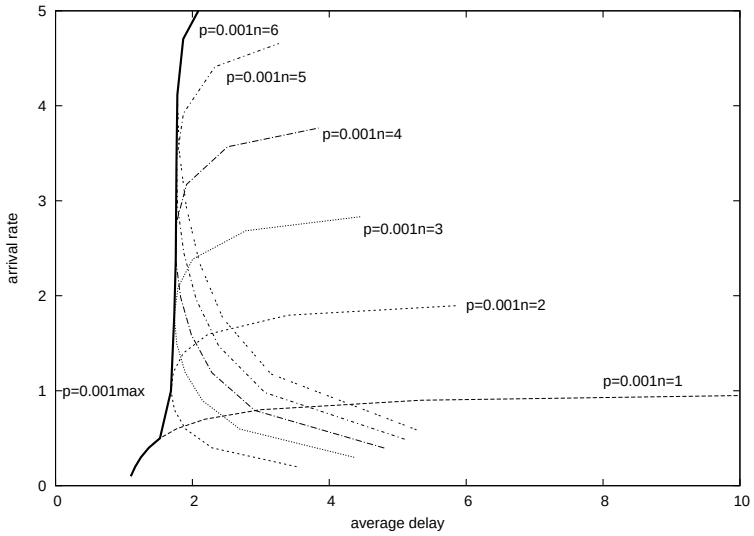

図 10　p=0.001 の場合

6　結論とまとめ

ハフマン符号を用いた伝送システムを構築し，シンボルを到着させたときに，どれだけ遅延が発生するかを調べる研究を行った．本研究では，従来研究の算術符号を用いた伝送システムの算術符号をハフマン符号に取り替えたシステムを用いている．

実験を行った結果，平均遅延は情報源に偏りがある方が小さいという結果が得られた．またシンボルの到着レート λ が小さいと理論式より，シンボルが決められたブロック長になるまでにかかる待ち時間が大きくなり，到着レートが大きいと，待ち時間は小さくなる．そしてバッファでの待ち時間は，到着レート λ が大きくなると，発散することが分かった．

また最後の三番目の実験では，情報源の偏りが大きい場合，ハフマン符号は平均遅延という点において非常に有効であるということが分かった．

本研究ではハフマン符号を用いて伝送システムを構築したが，今後の課題としては，ハフマン符号や算術符号などのさまざまな符号化システムを用いて，どの符号化システムを組み合わせた場合に平均遅延が最も小さくなるかを調べることなどが挙げられる．

謝辞

本研究を終えるにあたり，本研究において終始一貫して懇切丁寧なご指導を賜りました指導教員の西新幹彦先生に心より感謝と敬意の意を申し上げます．

参考文献

[1] Thomas M. Cover, Joy A. Thomas, Elements of Information Theory second edition, Wiley, 2006

[2] 森村英典，大前義次，応用待ち行列理論，日本技連，1975.

[3] 西田俊夫，待ち行列の理論と応用，朝倉書店，1971.

[4] 杉原辰徳「算術符号を用いた伝送システムにおけるポアソン到着シンボルの遅延」信州大学工学部学士論文，2008.

[5] 奥村晴彦，c 言語によるアルゴリズム辞典，技術評論社，1991.

付録 A　ソースコード

A.1　ハフマン符号を用いて実装した符号化のソースコード

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


/*****************指数分布乱数を発生*****************************************/



#define MRND  1000000000L
static int jrand;
static long ia[56]; /* ia[1.. 55] */

static void irn55(void)
{
    int i;
    long j;

    for (i=1; i<=24; i++){
        j=ia[i]-ia[i + 31];
        if (j<0) j += MRND;
        ia[i] = j;
    }
    for (i=25; i<=55; i++) {
        j=ia[i] - ia[i -24];
        if(j<0) j += MRND;
        ia[i]=j;
    }
}

void init_rnd(unsigned long seed)
{
    int i, ii;
    long k;

    ia[55] =seed;
    k=1;
    for (i=1; i<=54; i++) {
        ii = (21*i)%55;
        ia[ii]=k;
        k=seed-k;
        if(k<0) k += MRND;
            seed = ia[ii];
    }
    irn55();  irn55();  irn55(); /* warm up */
    jrand = 55;
}

long irnd(void)  /* 0 <= irnd() < MRND */
{
    if (++jrand > 55) {  irn55(); jrand = 1; }
    return ia[jrand];
}

double rnd(void)  /* 0 <= rnd() <1 */
{
    return (1. 0 / MRND) * irnd();
}

double exp_rnd(double lambda)
```

```
{
    return(-log(1 - rnd()) / lambda);
}

int main4(void)
{
    int i;
    init_rnd(3);
    for (i=1; i<=1000000; i++ ) { /*数を変える*/
        printf("%f\n", exp_rnd(1. 0));
    }
    return 0;
}


/************ 実験パラメータ ************************************************************/

double prob_of_one = 0. 5; /*確率を変える*/
double arriving_lambda = 0. 3;
double sending_mu = 1. 0;
int block_length = 2; /*ブロック長*/

/*************************文字の定義、初期条件*************************************************/


double present_time;
double arrival_time;
double sending_time;

#define ARRIVING_LIST_MAX 1000 /*数を変える？*/
#define SENDING_BUFFER_LIST_MAX 1000
int next_ch;

double arriving_list [ARRIVING_LIST_MAX];
int arriving_count = 0;
int arriving_in = 0;
int arriving_out = 0;

char sending_buffer_list[SENDING_BUFFER_LIST_MAX];
int sending_buffer_count = 0;
int sending_buffer_in = 0;
int sending_buffer_out = 0;

int bit ;



/************************バッファ*************************************************/


void add_arriving_list(double time)
{
    if(arriving_count >= ARRIVING_LIST_MAX){
        printf("error in %d\n", __LINE__);
        exit(0);
    }
    arriving_list[arriving_in] = time;
    arriving_in = (arriving_in + 1) % ARRIVING_LIST_MAX;
    arriving_count++;
    return;
}

double get_arriving_list()
{
    double time;

    if(arriving_count == 0){
        printf("error in %d\n", __LINE__);
        exit(0);
```

```
    }
    time = arriving_list[arriving_out];
    arriving_out = (arriving_out + 1) % ARRIVING_LIST_MAX;
    arriving_count--;
    return(time);
}

void add_sending_buffer_list(char bit)
{
    if(sending_buffer_count >= SENDING_BUFFER_LIST_MAX){
        printf("error in %d\n", __LINE__);
        exit(0);
    }
    sending_buffer_list[sending_buffer_in] = bit;
    sending_buffer_in = (sending_buffer_in + 1) % SENDING_BUFFER_LIST_MAX;
    sending_buffer_count++;
    return;
}

char get_sending_buffer_list()
{
    char bit;


    if(sending_buffer_count == 0){
        printf("error in %d\n", __LINE__);
        exit(0);
    }
    bit = sending_buffer_list[sending_buffer_out];
    sending_buffer_out = (sending_buffer_out + 1) % SENDING_BUFFER_LIST_MAX;
    sending_buffer_count--;
    return(bit);
}


/*****************ハフマン符号化*********************************************/


#include <stdio. h>
#include <stdlib. h>

typedef struct t_huff t_huff;
struct t_huff {
    char codesymbol;
    t_huff *parent;
    t_huff *child[2];
};

typedef struct t_leaf {
    double prob;
    t_huff *node;
} t_leaf;

void
setprob(t_leaf *a, int depth, double p, double sum)
{
    if (depth == 0){
        a->prob = sum;
    } else {
        depth--;
        setprob(a, depth, p, sum * (1 - p));
        setprob(a + (1 << depth), depth, p, sum * p);
    }
    return;
}

t_huff *Huffmantree;
```

```
int leaf_cmp(t_leaf *leaf_a, t_leaf *leaf_b)
{
    double diff = leaf_b->prob - leaf_a->prob;
    return((diff > 0)? 1: (diff < 0)? -1: 0);
}

void
maketree(int length, double p)
{
    t_leaf *leaves;
    int num = 1 << length;
    int i;
    t_huff *newnodep;
    int smallest;

    Huffmantree = (t_huff *)calloc(num * 2 - 1, sizeof(t_huff));
    if (Huffmantree == NULL){
        printf("%s in %d\n", "calloc error", __LINE__);
        exit(1);
    }

    for(i = 0; i < num; i++){
        Huffmantree[i]. child[0] = NULL;
        Huffmantree[i]. child[1] = NULL;
    }

    leaves = (t_leaf *)calloc(num, sizeof(t_leaf));
    if (leaves == NULL){
        printf("%s in %d\n", "calloc error", __LINE__);
        exit(1);
    }

    setprob(leaves, length, p, 1. 0);
    for (i = 0; i < num; i++){
        leaves[i]. node = Huffmantree + i;
    }
    newnodep = Huffmantree + num;

    /*
    for (i = 0; i < num; i++){
        printf("%d %f\n", i, leaves[i]. prob);
    }
    */

    qsort(leaves, num, sizeof(t_leaf), (int(*)(const void *, const void *))leaf_cmp);

    /*
    for (i = 0; i < num; i++){
        printf("%d %d %f\n", i, (int)(leaves[i]. node - Huffmantree), leaves[i]. prob);
    }
    */

    for (smallest = num - 1; smallest > 0; smallest--, newnodep++){
        leaves[smallest - 1]. node->codesymbol = '0';
        leaves[smallest    ]. node->codesymbol = '1';
        newnodep->child[0] = leaves[smallest - 1]. node;
        newnodep->child[1] = leaves[smallest    ]. node;
        leaves[smallest - 1]. node->parent = newnodep;
        leaves[smallest    ]. node->parent = newnodep;
        leaves[smallest - 1]. node = newnodep;
        leaves[smallest - 1]. prob += leaves[smallest]. prob;
        for (i = smallest - 1; i > 0 && leaves[i]. prob > leaves[i - 1]. prob; i--){
            t_leaf tmp;
            tmp = leaves[i];
            leaves[i] = leaves[i - 1];
            leaves[i - 1] = tmp;
        }
    }
    leaves[0]. node->parent = NULL;
```

```
    free(leaves);
    return;
}

void
encode_rec(t_huff *node)
{
    if (node->parent != NULL){
        encode_rec(node->parent);
        add_sending_buffer_list(node->codesymbol);
        if ( sending_time < 0. 0){
            sending_time = present_time + (1. 0 / sending_mu) ;
        }
    }
    return;
}

void
encode(long supersymbol)
{
    encode_rec(Huffmantree + supersymbol);
    return;
}

int main2()
{
    int i;

    maketree(block_length, 0. 5); /*変数にする    確率を変える*/
    for (i = 0; i < (1 << (block_length - 1)); i++){ /*変数にする, 2の○乗の書き方*/
        printf("%d: ", i);
        encode(i);
        printf("\n");
    }
    return(0);
}

t_huff *decode_current;

void
init_decoder()
{
    decode_current = Huffmantree + ((2 << block_length) - 2); /*変数にする, 2 n 乗-2*/
    return;
}

void
decode(char c)
{
    if (c == '0') {
        decode_current = decode_current->child[0];
    } else {
        decode_current = decode_current->child[1];
    }
    if (decode_current->child[0] == NULL){
        int i;
        for (i = 0; i < block_length; i++){
            double arrive = get_arriving_list();
            double delay = present_time - arrive;
            printf("%f\n", delay);

        }
        decode_current = Huffmantree + ((2 << block_length) - 2); /*変数にする, 2 n 乗-2*/
    }
    return;
}
```

```
int main3()
{
    int c;

    maketree(block_length, 0.5); /*変数にする    確率を変える*/
printf("DEBUG %d\n", __LINE__);
    while ((c = getchar()) != EOF){
        decode(c);
    }
    return(0);
}


/*****************************************************************************/



void
sending_finish()
{
    char bit;

    bit = get_sending_buffer_list();
    decode(bit);
    if(sending_buffer_count == 0){
        sending_time = -1.0;
    } else {
        sending_time = present_time + (1.0 / sending_mu);
    }
    return;
}


void
arrival_symbol()
{
    static long mask = 0;
    static long block;

    add_arriving_list(present_time);

    if (mask == 0){
        mask = 1 << (block_length - 1);
        block = 0;
    }
    if (next_ch == '1'){
        block |= mask;
    }
    mask >>= 1;
    if (mask == 0){
        encode(block);
    }
    if ((next_ch = getchar()) == EOF) {
        arrival_time = -1.0 ;
    } else {
        arrival_time = present_time + exp_rnd(arriving_lambda);
    }
    return;
}

/*****************************************************************************/

int
main(int argc, char *argv[])
{

    if (argc != 4){
        printf("Block length?\n");
        printf("Prob of one?\n");
```

```
        printf("Arriving lambda?\n");
        exit(1);
    }
    block_length = atoi(argv[1]);
    if (block_length <= 0){
        printf("Block length range error\n");
        exit(1);
    }
    prob_of_one = atof(argv[2]);
    if (prob_of_one <= 0){
        printf("Prob of one range error\n");
        exit(1);
    }
    arriving_lambda = atof(argv[3]);
    if (arriving_lambda <= 0){
        printf("Arriving lambda range error\n");
        exit(1);
    }

    maketree(block_length, prob_of_one);
    init_decoder();
    init_rnd(3);
    present_time = 0.0;
    arrival_time = exp_rnd(arriving_lambda);
    sending_time = -1.0;
    next_ch = getchar();
    while ((arrival_time >= 0.0) || (sending_time >= 0.0)) {
        if (arrival_time < 0.0 || (sending_time >= 0.0 && arrival_time > sending_time)){
            present_time = sending_time; /* 現在時刻を進ませる */
            sending_finish();
        }else{
            present_time = arrival_time; /* 現在時刻を進ませる */
            arrival_symbol();
        }
    }
    return (0);
}
```

A.2 平均遅延導出のソースコード

```
#include <stdio.h>
#include <stdlib.h>
#define N_SLOTS (1 << 16)

long count[N_SLOTS];
double lower, upper, range;

double
psum(int i, int size)
{
    if (size == 1){
        return((lower + (i + 0.5L) * range / N_SLOTS) * count[i]);
    }
    return(psum(i, size / 2) + psum(i + size / 2, size / 2));
}

int
main(int argc, char **argv)
{
    long number;
    double value;
    double sum, expect;
    int i;
```

```
    if (argc != 3) goto ERROR;
    if (argv[1] == NULL || argv[2] == NULL) goto ERROR;
    if (sscanf(argv[1], "%lf", &lower) != 1) goto ERROR;
    if (sscanf(argv[2], "%lf", &upper) != 1) goto ERROR;
    range = upper - lower;

    for (i = 0; i < N_SLOTS; i++){
        count[i] = 0;
    }
    number = 0;

    while (scanf("%lf\n", &value) == 1){
        number++;
        i = (int)(((value - lower) / range) * N_SLOTS);
        if (i >= 0 && i < N_SLOTS){
            count[i]++;
        } else {
            printf("range error (%d)\n", number);
            exit(1);
        }
    }

    sum = psum(0, N_SLOTS);
    expect = sum / number;
    printf("%g\n", expect);

    return(0);

ERROR:
    puts("distribution <lower> <upper>");
    exit(1);
}
```