信州大学工学部

学士論文

パケット間隔で情報を送るシステムのための復号法に関する検討

指導教員　西新 幹彦 准教授

学科　電気電子工学科
学籍番号　09T2002D
氏名　ABDULLAH FAHIM

2014 年 2 月 16 日

目次

1　はじめに

1.1　研究背景と目的

現代の通信システムの中にはいろいろな情報伝達方法が存在する．情報を的確に伝達するためには，伝達内容の特徴に即していろいろな方法の工夫とコンピュータや情報通信ネットワークなどの適切な活用が必要である．その情報伝達方法の一つとしてパケット通信がある．パケット通信とはコンピュータ通信において、データを小さなまとまりに分割して一つ一つ送受信する通信方式である．分割されたデータはパケットと呼ばれる．情報伝達のモデルを考える際，伝えたいメッセージそのものをそのまま送信できる場合は極めてまれである．多くの場合はメッセージを符号化し，その符号語を分割してパケットに収める．符号語を一度にまとめて圧縮してパケットに入れて送信したり様々な方法で伝えたいメッセージを受信者に送る．

Anantharam and Verdú[1] では情報はパケットの内容だけではなく，隣り合うパケットとパケットの時間間隔によっても送ることができると指摘されている．パケット間隔を用いて通信するということは，間隔にメッセージの意味を持たせることにより受信側が受け取った間隔から送信側の伝えたいメッセージを読み取るということである．しかし，パケットの間隔を用いて情報を送っても通信路でその間隔が崩れてしまい，誤りが発生する．誤りが発生しても送信された情報に戻すために復号器が必要である．誤りがほぼ発生しないような符号化レートの上限は求められているが [1]，その上限を達成可能な符号の設計方法は知られていない．

本研究では，パケット間隔を用いて通信する場合その通信路のモデルについて考え，復号器の性能を検討する．

1.2　本論文の構成

本論文は次のような構成で成る．2 章では本研究で扱う通信システムの説明や問題設定などについて述べる．3 章では 2 種類の復号器の構成について述べる．4 章では復号器の評価を行い，考察を述べる．5 章ではまとめを行う．6 章では今後の課題について述べる．

2　通信システム

2.1　通信路モデル

パケット通信システムでは，パケットは到着したサーバーによって次の配送先を決定され，最終的に目的地に届けられる．ユーザがパケットを送信すると，出発してから目的地までにかかる時間は，他のユーザがどの程度パケットを送信しているか，すなわちネットワークのトラ

フィックに依存する．今，一対一の送受信者がいると考えて，それに着目するとネットワークの全体は一つの待ち行列システムとしてみなすことができる．本研究では，パケットの間隔によってのみ情報を送るシステムを考え，パケットの大きさを考えない．すなわち，パケットに情報を載せないことを仮定する．これをモデル化し，単一サーバーによって構成されたFIFO(First In First Out) 待ち行列システムを通信路とみなして通信路符号化することを考える．

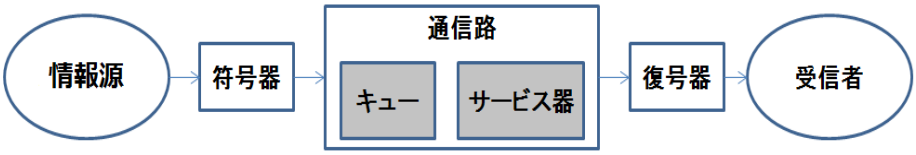

図 1　通信路の構成

このような送信モデルを図 1 に示す．送信対象のメッセージを符号器によってパケットの時間間隔に変換し通信路へ送信する．通信路は単一のキューとサービス器より構成される．

サービス器で処理するためにかかる時間をサービス時間という．そのサービス時間はパラメータ μ の指数分布に従うとする．このサービス器のサービスレートは μ である．

図 1 のサービス器の手前にある装置はキューという．サービス器は一度に一つのパケットしか処理できない．一つのパケットがサービスを受けている途中で次のパケットが到着した場合，到着順番を乱すことなくキューで待たせる．キューは待つパケットを全て格納できるように十分な大きさがあると仮定する．

パケット間にある間隔をあけて送ってもキューで待ったり，サービス器で指数的に処理されたりすることでパケットがサービス器から出る間隔は一般に送信した間隔とは異なる．これが誤りの原因である．

文献 [1] よりこの単一サーバー通信システムの通信路容量は既に求められている． しかしどのような符号を用いることで通信路容量に近づけるのかは未だ知られていない．そこで，これを達成できるような符号を構成することを目指し，本研究ではよい符号の構成法について考察する．

2.2　符号

メッセージを通信する時はそのままでは送信できない．メッセージを符号語に変換後，通信路に入力しなければならない．メッセージを符号語に変換する装置は符号器である．

通信路から出てくる受信語はまた復元しなければならない．受信語を復元する装置は復号器である．

本研究の場合は，送りたいメッセージをパケットの時間間隔に変換することが符号化である．符号器からパケットを送ることで隣り合うパケットの時間間隔を定められ，それを符号語として送信する．時間間隔のみに着目したいので，パケット自身には情報はなく大きさもないとする．符号語シンボルが時間間隔ということより取り得る値は正数全体となる．

本研究では，従来研究 [1] に倣い，パケットの間隔を用いた符号を次のように定義する．

定義 1　自然数 n を考える．非負実数ベクトル $(a_1, \cdots, a_n)$ を符号語という．符号器が符号語 $(a_1, \cdots, a_n)$ を送信するとき，時刻 $\sum_{i=1}^{k} a_i$ に k 番目 $(k \geq 1)$ のパケットが待ち行列に到着する．符号器は M 個の符号語を持っているとし，それらは等確率で選ばれて送信される．復号器は n 個のパケットを受け取り，それらの受信時刻から符号語を復号する．平均誤り確率を ε とおく．最後のパケットが復号器に届く時刻の平均を T とおく．このような符号器・復号器を (n, M, T, ε) 符号という．この符号の符号化レートを $(\log M)/T$ と定める．符号化レートとは，単位時間当たりに符号器から送信する情報量を表している．これは，単位時間当たりに符号器が受け付ける情報量と言い換えてもよい．

2.3　達成可能性と通信路容量

符号化レートは大きいほうが良いが，復号器で正しく復号されなければ意味がない．したがって，与えられた符号化レートに対して，ほぼ確実に正しく復号できるような復号器が存在するかどうかが重要である．この概念を以下のように定義する．以下に示す 2 つの定義は Anantharam and Verdú[1] にもとづいて [2] で定義されたものである．

定義 2　符号化レート R が達成可能とは，

$$\liminf_{n\to\infty} \frac{\log M_n}{T_n} \geq R \tag{1}$$

$$\lim_{n\to\infty} \frac{n}{T_n} > 0 \tag{2}$$

$$\lim_{n\to\infty} \varepsilon_n = 0 \tag{3}$$

となるような符号の列 $\{(n, M_n, T_n, \varepsilon_n)\}_{n=1}^{\infty}$ が存在することである．通信路容量を

$$C \triangleq \sup\{R| \text{ 符号化レート } R \text{ は達成可能．}\}$$

と定める．

定義 3 符号化レート R が**出力レート** λ で ε-**達成可能**とは，

$$\liminf_{n\to\infty} \lambda \frac{\log M_n}{n} \geq R \tag{4}$$

$$\liminf_{n\to\infty} \frac{n}{T_n} \geq \lambda \tag{5}$$

$$\limsup_{n\to\infty} \varepsilon_n \leq \varepsilon \tag{6}$$

となるような符号の列 $\{(n, M_n, T_n, \varepsilon_n)\}_{n=1}^{\infty}$ が存在することである．$0 < \varepsilon < 1$ となる任意の ε において R が出力レート λ で ε-達成可能であれば，R は出力レート λ で達成可能であるという．出力レート λ で達成可能な R の上限を，出力レート λ における通信路容量と呼び，$C(\lambda)$ と表す．すなわち

$$C(\lambda) \triangleq \sup\{R| \text{ 符号化レート } R \text{ は出力レート}\lambda\text{で達成可能．}\}$$

と定める．

定義 2 と定義 3 は以下の関係である．

定理 1([2]) サービスレート μ のサーバを持つ通信路に対して C と $C(\lambda)$ は

$$C = \sup_{0<\lambda<\mu} C(\lambda) \tag{7}$$

を満たす．

定理 2([1]) 出力レート λ における通信路容量 $C(\lambda)$ は

$$C(\lambda) = \lambda \log \frac{\mu}{\lambda} \quad (0 < \lambda < \mu) \tag{8}$$

と表される．

定理 2 を定理 1 に代入して最大値を求めると，通信路容量は

$$C = \frac{\mu}{e} \tag{9}$$

となる．そのときの出力レートは

$$\lambda = e^{-1}\mu \tag{10}$$

である．

出力レート λ における通信路容量 $C(\lambda)$ のグラフを図 2 に示す．ただしサービスレートは $\mu = 1$ と固定した．横軸と縦軸はそれぞれ出力レート λ とそのときの通信路容量 $C(\lambda)$ である．

定理 2 の証明では，出力レート λ における通信路容量 $C(\lambda)$ を達成するような符号器は，到着レート λ のポアソン到着を符号語としてランダムに選ぶことによって構成されている．したがって，通信路容量 C を達成する符号器の符号語は，到着レート $\lambda = \mu/e$ のポアソン到着にしたがってランダムに選べばよい．

ちなみに，定理 2 の証明では，待ち行列への到着過程がレート λ のポアソン到着であれば，出力過程もレート λ のポアソン到着になるという Burke の定理 [3] が一つの重要な役割を持っている．

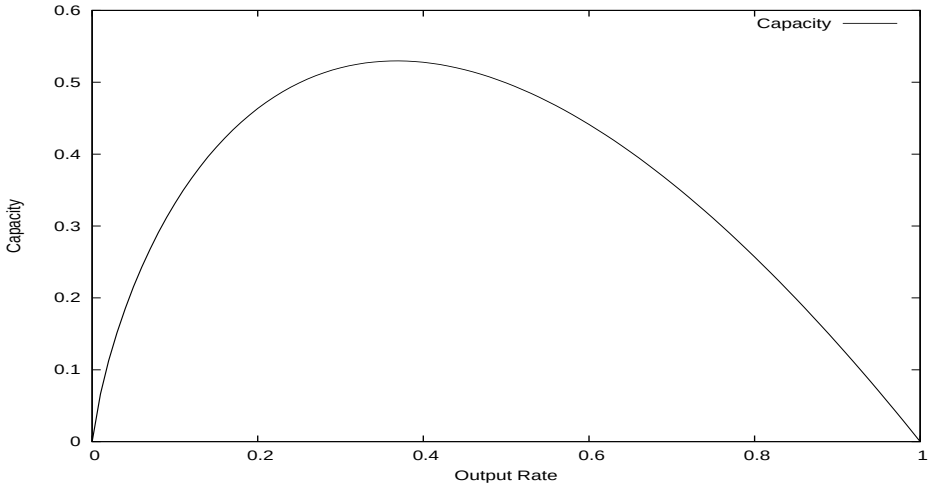

図 2　通信路容量

3　符号器と復号器の設計

本研究ではよい符号の列を作ることが目的である．すなわち，符号器と復号器を設計しなければならない．本研究では一つの符号器とそれに対して 2 種類の復号器を作り，それぞれの性能を計測し，比較する．

3.1　符号器の設計

文献 [1] では，通信路容量を達成する符号器は到着レート $\lambda = \mu/e$ のポアソン到着にしたがってランダムに選んで作ることができた．これにならい，本研究でも符号器は同様に構成するものとする．

符号器の符号化レートを R とおく．定義 1 で符号器の符号化レートは $\frac{\log M_n}{T}$ と定めたので

$$R = \frac{\log M_n}{T}$$

つまり

$$M_n = e^{RT} \tag{11}$$

の関係がある．一方，符号器で符号語をレート λ のポアソン到着で作っているので Burke の

定理 [3] より通信路の出力過程もレート λ のポアソン過程となる．よって

$$T = \frac{n}{\lambda}$$

となる．これを式 (11) に代入すると

$$M_n = e^{\frac{nR}{\lambda}} \tag{12}$$

になる．符号語数は自然数なので実際には小数点以下を切り捨てる．よって $e^{\frac{nR}{\lambda}}$ の値が小さいときにはある程度の誤差が発生するが，値が大きくなるにつれて誤差は小さくなる．

以上の方法で符号器のために M_n 個の符号語を作る．考え方として，最初に時刻 t は 0 であるとする．パケットの間隔は符号語として $(a_1, a_2, a_3, \cdots, a_n)$ のように表す．a_n とは $(n-1)$ と n 番目のパケットの到着間隔である．符号器で作った M_n 個の符号語を v とすると

$$v = (v_1, v_2, v_3 \cdots, v_{M_n}) \tag{13}$$

さらに，各 $v_1, v_2, \cdots$，v_M の中身は $(a_1, a_2, a_3, \cdots, a_n)$ である．

符号器で作った M_n 個の符号語から一つの符号語を等確率で選び，通信路に送る．

3.2 復号器の設計

復号器はサービス器から送信された n 個のパケットの出発を観測する．復号器はパケットの間隔を受信語として $(b_1, b_2, b_3, \cdots, b_n)$ のように受け取る．復号器が受け取った受信語を w とすると

$$w = (b_1, b_2, b_3, \cdots, b_n) \tag{14}$$

と表される．そして，復号器は受信語を符号器で作られた M_n 個の符号語と比較し，符号語に復号する．本研究で考える復号法は，基本的に受信語と符号語の間の距離を定義し，最も距離の小さい符号語を復号語とする．

本研究では受信語 w から符号語 v を復号する方法として以下の 2 つ の復号法を考える．

3.2.1 復号法 1

復号法 1 では一般的な AWGN 通信路 [4] で誤り訂正するために使われる差の二乗和法を用いてメッセージの復号を行う．受信した n 個のパケットと送信された n 個のパケットの距離を考える．これは式 (15) のように表すことができる．

$$d_n^{(1)}[(b_1, b_2, \cdots, b_n), (a_1, a_2 \cdots, a_n)] = \sum_{i=1}^{n} (b_i - a_i)^2 \tag{15}$$

3.2.2　復号法 2

パケット間隔を用いて通信すると符号語はキューとサーバを通過するので受信語の n 番目のパケットの受信時刻は必ず送信された符号語の n 番目のパケットの送信時刻より後になる．よって，パケットの受信時刻が符号語の送信時刻と同じか早くなるのは起こりえない．この単純な見解から起こりえない符号語を復号されないように取り除けばよいと考えた．

考え方を説明するために，最初にパケットが一つの場合を考える．このとき，符号語や受信語は一つのパケットの到着時刻として表される．受信機が $w = (b)$ を受信語として受信したとする．復号器が符号器で作られた M_n 個の符号語のうちどの符号語が送られてきたかを推測する．起こりえない符号語が取り除かれるように，その符号語との距離を ∞ とおく．これをまとめると式 (16) のようになり，b の方が a より後の場合のみ b と a の差を距離とする．

$$d_1^{(2)}(b,a) = \begin{cases} b-a & (b > a \text{ のとき}) \\ \infty & (b \leq a \text{ のとき}) \end{cases} \tag{16}$$

次にパケットが 2 つ の場合を考える．このとき，受信語は $w = (b_1, b_2)$ であるとする．パケットが一つの場合と同様に，$(b_1 + b_2)$ の時刻が符号語である $(a_1 + a_2)$ の時刻より後の場合しかありえない．つまり，$(b_1 + b_2) \leq (a_1 + a_2)$ は起こりえないので式 (16) と同様に復号されないようにする．以上をまとめると式 (17) のように表すことができる．

$$d_2^{(2)}[(b_1,b_2),(a_1,a_2)] = \begin{cases} d_1(b_1,a_1) + (b_1+b_2) - (a_1+a_2) & (b_1+b_2 > a_1+a_2 \text{のとき}) \\ \infty & (\text{Otherwise}) \end{cases} \tag{17}$$

そして，パケットが n 個の場合を考える．このとき，符号語と受信語はそれぞれ $v = (a_1, a_2, a_3, \cdots, a_n)$ と $w = (b_1, b_2, b_3, \cdots, b_n)$ であるとする．式 (16) と (17) を考慮し，$(b_1 + \cdots + b_n > a_1 + \cdots + a_n)$ の場合のみ考えると式 (18) のように表すことができる．

$$d_n^{(2)}[(b_1 \cdots, b_n),(a_1, \cdots, a_n)] = \begin{cases} d_{n-1}[(b_1, \cdots b_{n-1}),(a_1, \cdots, a_{n-1})] & \\ +(b_1 + \cdots + b_n) & (b_1 + \cdots + b_n \\ -(a_1 + \cdots + a_n) & > a_1 + \cdots + a_n \text{のとき}) \\ \infty & (\text{Otherwise}) \end{cases} \tag{18}$$

4　復号器の評価

本章では，3 章で設計した 2 種類の復号器の復号誤り確率とその収束値を考察することによって，復号器の性能を比較する．

4.1 評価方法

本研究で扱うサービス器のサービスレートは，$\mu = 1$ と仮定して一般性を損なわない．なぜなら, 一般にサービスレート μ の場合は $\mu = 1$ のときに得られたパケットの送信間隔を $\frac{1}{\mu}$ 倍にしたものを符号語とすれば同じ誤り確率が得られるからである．

式 (7)，(8)，(9) から通信路の最大値を求めると

$$C = \sup_{0<\lambda<1} C(\lambda) = 0.529[\text{bit/sec}]$$

になる．そのとき出力レート λ は

$$\lambda = e^{-1} = 0.367[\text{symbol/sec}]$$

である．

これに従い，符号語とその数を決める．λ の値を 0.367 に固定し，パケット数 n を増加させ、符号化レート R の値を変更しながら復号誤り確率を計測する実験を行った．

4.2 結果と考察

M_n 個の符号語から一つの符号語を等確率で選んで送信するという試行を 10 万回行い，3 章で用意した 2 種類の復号器を用いて復号を行って，それぞれの平均復号誤り確率 ε_n を測定した．符号化レート R の値は 0.367 から減少させていった．そして，横軸にパケット数，縦軸に対数スケールで ε_n をプロットした．その結果を以下の 5 つ のグラフに示した．

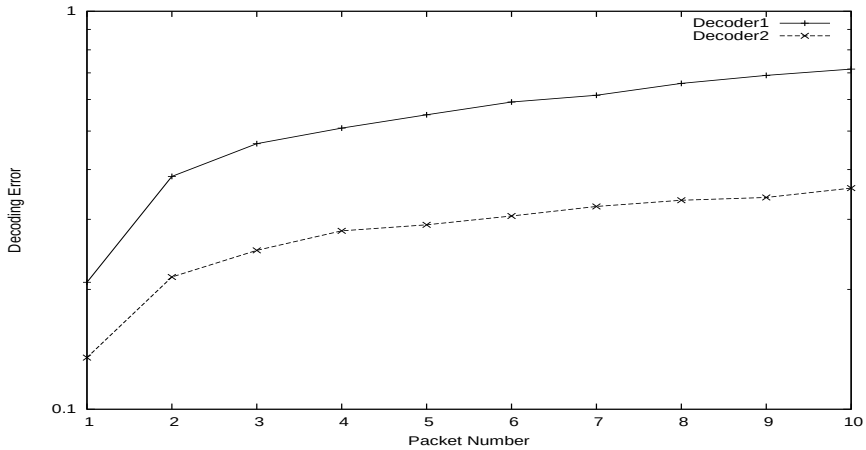

図 3　復号誤り確率 (R=0.367 のとき)

図 3 から R の値が λ と同じときパケットの増加とともに誤りも増加し，極限で 0 にいかないと考えられる．よって，符号化レート R が λ と同じ場合，本研究で用いた復号器でよい符号の列が作れないと考えられる．

次に，符号化レート R の値をいくつになれば復号誤り確率が 0 にいくのかを調べた．そのため R の値を λ の値からはじめ $\frac{\lambda}{11} = 0.0333$ まで少しずつ減少しながら実験を行った．その中で復号器 1 と復号器 2 の変化が比較しやすい $\frac{\lambda}{1.5} = 0.244$，$\frac{\lambda}{2} = 0.184$，$\frac{\lambda}{2.5} = 0.146$ と $\frac{\lambda}{7} = 0.0524$ 場合の結果を以下のグラフに示した．

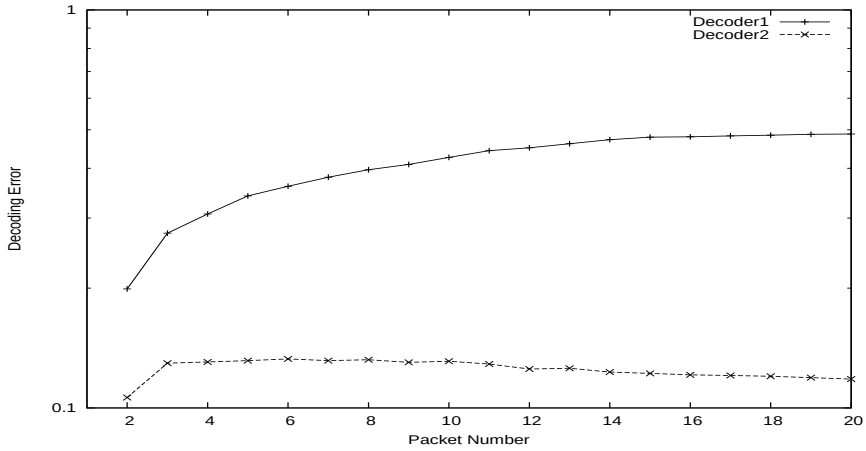

図 4　復号誤り確率 (R=0.244 のとき)

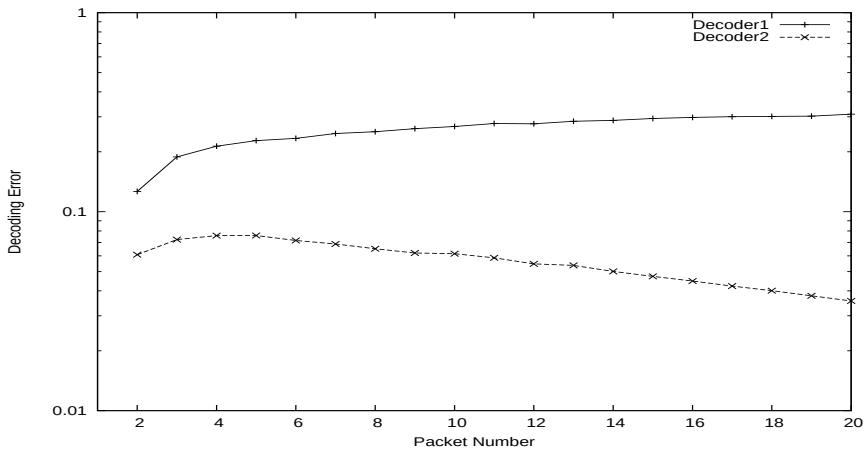

図 5　復号誤り確率 (R=0.184 のとき)

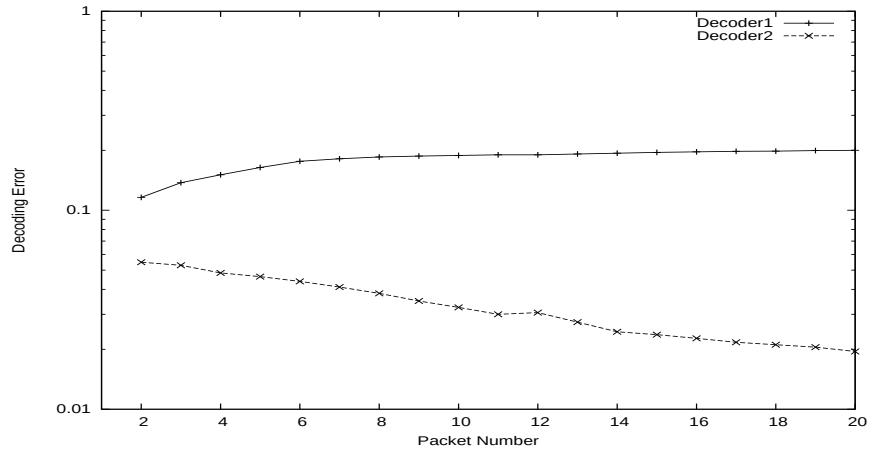

図 6　復号誤り確率 (R=0.146 のとき)

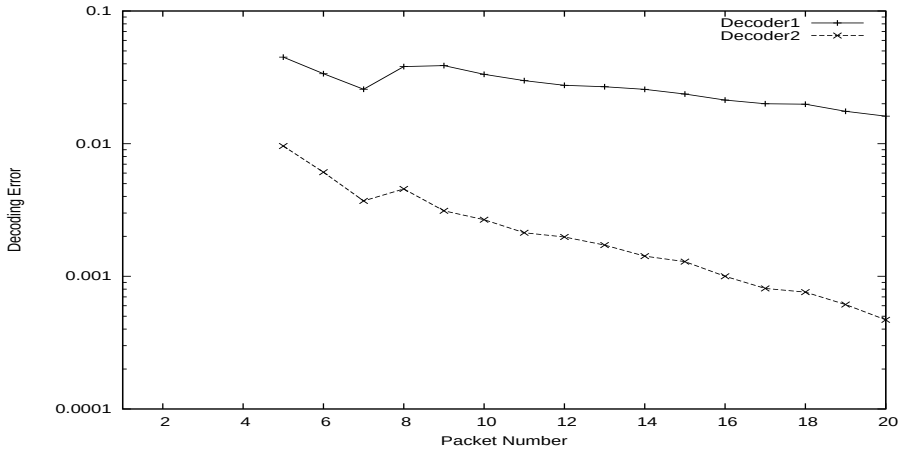

図 7　復号誤り確率 (R=0.0524 のとき)

上の結果から R の値を 0.367 から 0.0524 まで変化させたときの復号誤り確率 (Decoding Error) はどの場合も復号器 2(Decoder2) の方が復号器 1(Decoder1) よりも小さいことがわかる．

図 4，5 を比較すると復号器 2 の場合，符号化レート R の値が λ のおよそ半分の値 (0.184) からパケットの増加とともに 0 に近づいていることが読み取れる．そのとき復号器 1 の誤り確率が緩やかであるが，まだ上昇している傾向がみられる．符号化レート R の値がそれ以上になると復号器 1 の誤り確率は 0 にいかないと考えられる．

図 6 では符号化レート $R = 0.146$ のとき復号器 1 の誤り確率がまだ 0 に近づかないことがわかる．R をそれより少しずつ小さくしておよそ 0.0524 で復号器 1 の誤り確率が 0 に近づくと考えられる．

よい符号の意味では復号器 2 は復号器 1 よりおよそ $(0.184/0.0524) = 3.5$ 倍の符号化レート R が得られると考えられる．

5 まとめ

現代の通信手段の一つとして通信路に送られるパケットの時間間隔に着目し, その間隔よって情報を送る通信路の符号化について考えた．

$R < C(\lambda)$ ならば，十分大きなすべての n でよい符号の列が存在する．本研究ではよい符号の列を作るために 2 種類の復号器を考えた．その復号器のどれも符号化レート R を小さくしていけばいくほど誤りが小さくなって行く．しかし，符号化レート R を小さくして行くことで情報量も小さくなる．誤りはできる限り小さくし，情報量はできる限り大きくしたい．実験結果をみると，およそ $R = 0.184$ のとき復号器 2 の誤り確率が 0 に近づいていた．そのとき復号器 1 の誤り確率がまだ上昇していた．それは復号器 1 と 2 の決定的な違いである．復号器 1 の場合誤り確率がおよそ $R = 0.0524$ のとき 0 に近づいていた．すなわち，復号器 2 は復号器 1 よりおよそ 3.5 倍の性能を持っていることが分かった．また，よい符号を作るために復号器の設計が非常に重要であることがわかった．

復号器 1 が一般的な AWGN 通信路 [3] の誤り訂正で使われているが，本研究で扱った単一待ち行列システムは特殊な通信路であるため，復号器 2 の性能がよかった．そのため，単一待ち行列システムの場合復号器 2 を提案する．

6 今後の課題

本研究では出力レート λ の値を 0.367 に固定し，符号化レート R を変化させて実験を行ったが，今後 λ と R 両方とも変化させるとどうなるかが興味深い．

復号器をどのように設計すれば符号化レートを通信路容量に近づけられるのかが今後の課題として残される．

謝辞

本研究を行うにあたり，数多くの助言，指導をしてくださった指導教員西新幹彦准教授，また西新研究室の皆様に感謝の意を表する．

参考文献

[1] Venkat Anantharam, Sergio Verdú, “Bits Through Queues,” IEEE Transactions on Information Theory, vol.42, no.1, pp.4-18, January 1996.

[2] 西新幹彦，藤山直貴，「パケット間隔で情報を送る通信路の悲観的な符号化について」，第35 回情報理論とその応用シンポジウム (SITA2012)，pp.590–595, December 2012.

[3] Robert G. Gallager, “Discrete Stochastic Processes,” Kluwer Academic Publishers, 1996.

[4] 和田山正，誤り訂正技術の基礎，森北出版，2010.

付録 A　ソースコード

A.1　復号器で誤り確率を計測するためのプログラム

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


//int PACKET_NUMBER=1;
//#define recall double exp(PACKET_NUMBER)
#define INFINITY    -1

#define SEND_DATA 100000

#define MRND 1000000000L
static int jrand;
static long ia[56];

int minimum_distance_patern(double *a, int n);

double distance2(const double *array_a, const double *array_b, int n);
double distance1(const double *array_a, const double *array_b, int n);

double distance_3(double b1, double b2, double b3, double a1, double a2, double a3);
double distance_2(double b1, double b2, double a1, double a2);
double distance_1(double b1, double a1);


int main_a(const int PACKET_NUMBER, const int recall,
double **packet_interval , double *recieved_packet_interval,
double *recieved_packet_time, double *distance, FILE *graph);


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();
    jrand=55;
}

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

double rnd(void)
{
    return (1.0/MRND)*irnd();
}

//一様乱数→指数乱数を生成
double rand_exp(double rate){
    return (-(1/rate)*log(1-(rnd())));
}

int main(void){
    int PACKET_NUMBER;
    int recall;

    unsigned long a = ((unsigned long)time(NULL))%1000000000;
    init_rnd(a);

    int max_packet_number;

//  printf("max packet number\t");
    scanf("%d",&max_packet_number);

    int max_recall = (int)exp((double)max_packet_number / 1.0);
    //printf("max_packet_number\t%d\n", max_packet_number);
    //printf("max_recall\t\t%d\n\n", max_recall);

    //scanf("%d",&SEND_DATA);


    double **packet_interval = (double**)calloc(max_recall, sizeof(double*));
    if(packet_interval == NULL){
        printf("error:%d\n", __LINE__);
        exit(1);
    }
    for(int i=0; i<max_recall; i++){
        packet_interval[i] = (double*)calloc(max_packet_number, sizeof(double));
        if(packet_interval[i] == NULL){
            printf("error:%d\n", __LINE__);
        exit(1);
        }
    }


    double *recieved_packet_interval = (double*)calloc(max_packet_number, sizeof(double));
    if(recieved_packet_interval == NULL){
        printf("error:%d\n", __LINE__);
        exit(1);
    }

    double *recieved_packet_time = (double*)calloc(max_packet_number, sizeof(double));
    if(recieved_packet_time == NULL){
        printf("error:%d\n", __LINE__);
        exit(1);
    }

    double *distance = (double*)calloc(max_recall, sizeof(double));
    if(distance == NULL){
        printf("error:%d\n", __LINE__);
```

```
        exit(1);
    }

//  double packet_interval[recall][PACKET_NUMBER];
//  double recieved_packet_interval[PACKET_NUMBER];
//  double recieved_packet_time[PACKET_NUMBER];

    FILE *graph = fopen("graph.csv", "w");
    if(graph == NULL){
        printf("error:%d\n", __LINE__);
        exit(1);
    }

    for (int PACKET_NUMBER = max_packet_number; PACKET_NUMBER <= max_packet_number; PACKET_NUMBER++){
        recall = (int)exp((double)PACKET_NUMBER / 1.0);
        //printf("PACKET_NUMBER:%d\trecall:%d\n", PACKET_NUMBER, recall);
        printf("PACKET_NUMBER:%d\t", PACKET_NUMBER);

        main_a(PACKET_NUMBER, recall, packet_interval, recieved_packet_interval, recieved_packet_time, distance);
        printf("\n");
    }

    return 0;
}

int main_a(const int PACKET_NUMBER, const int recall, double **packet_interval ,
double *recieved_packet_interval, double *recieved_packet_time, double *distance){
    int i;
    int arrival_packet;
    int recieved_packet;
    int queue_in;
    double current_time;
    double next_packet;
    double service_end;

    const double server_rate = 1;

    int selected_pattern;
    int correct_distance1 = 0;
    int correct_distance2 = 0;

//  printf("line %d\n", __LINE__);

    for(int send_data=0; send_data<SEND_DATA; send_data++){
    //  printf("line %d\n", __LINE__);
        for(i=0;i<recall;i++){
            //printf("packet_interval[%d]\n", i);
            for(int j=0; j<PACKET_NUMBER; j++){
                packet_interval[i][j] = fabs(rand_exp(0.36));
                //printf("%f\n",packet_interval[i][j]);
            }
            //printf("\n");
        }
    //  printf("line %d\n", __LINE__);
        selected_pattern = irnd()%recall;
        //printf("slected_pattern : %d\n", selected_pattern);



        //printf("-------------------------\n");
        arrival_packet  = 0;
        recieved_packet = 0;
        queue_in        = 0;
        current_time    = 0;
        next_packet     = packet_interval[selected_pattern][arrival_packet];
        service_end     = -1.0;

        while(1){
LOOP:
```

```
            if(service_end < 0.0){
Arrival_Block:
                current_time = next_packet;
                arrival_packet++;
                queue_in++;
                if(arrival_packet < PACKET_NUMBER){
                    next_packet = current_time + packet_interval[selected_pattern][arrival_packet];
                }
                if(queue_in == 1){
                    service_end = current_time + rand_exp(server_rate);
                }
                goto LOOP;
            }else{
                if(arrival_packet < PACKET_NUMBER){
                    if(next_packet < service_end){
                        goto Arrival_Block;
                    }
                }
End_Block:
                current_time = service_end;
                queue_in--;
                //printf("%f\n",current_time);
                //recieved_packet_time の保存
                recieved_packet_time[recieved_packet] = current_time;
                recieved_packet++;
                if(queue_in > 0){
                    service_end = current_time + rand_exp(server_rate);

                    goto LOOP;
                }else{
                    service_end = -1;
                    if(arrival_packet < PACKET_NUMBER){
                        goto LOOP;
                    }else{
                        break;
                    }
                }
            }
        }

        //printf("\n");
        //recieved_packet_time から recieved_packet_interval への変換
        for(int i=0; i<PACKET_NUMBER; i++){
            if(i == 0)
                recieved_packet_interval[i] = recieved_packet_time[0];
            else{
                recieved_packet_interval[i] = recieved_packet_time[i] - recieved_packet_time[i-1];
            }
            //printf("%f\n", recieved_packet_interval[i]);
        }


    //  double distance[recall] = {0};
        //printf("\n");
        for(int i=0; i<recall; i++){
            distance[i] = distance1(packet_interval[i], recieved_packet_interval, PACKET_NUMBER);
            //printf("pattern%d:%f\n",i, distance[i]);
        }
        //printf("%d\n", minimum_distance_patern(distance, recall));
        if( minimum_distance_patern(distance, recall) == selected_pattern)
            correct_distance1++;
        //printf("\n");
        for(int i=0; i<recall; i++){
            distance[i] = distance2(packet_interval[i], recieved_packet_interval, PACKET_NUMBER);
            //printf("pattern%d:%f\n",i, distance[i]);
        }
        //printf("%d\n", minimum_distance_patern(distance, recall));
        if( minimum_distance_patern(distance, recall) == selected_pattern)
            correct_distance2++;
```

```
        //printf("\n");
    }
    printf("%f\t%f",1.0-(double)correct_distance1/(double)SEND_DATA,1.0-(double)correct_distance2/(double)SEND_DATA);
    //printf("%f\n",1.0-(double)correct_distance2/(double)SEND_DATA);


    return 0;
}

int minimum_distance_patern(double *a, int n){
    double minimum_distance;
    int place = -1;


    for(int i=0; i<n; i++){
        if(a[i] == INFINITY)
            continue;
        else{
            minimum_distance = a[i];
            place = i;
            break;
        }
    }
    if(place == -1)
        return -1;

    for(int i=place+1; i<n; i++){
        if(a[i] == INFINITY)
            continue;
        else if(a[i] < minimum_distance){
            minimum_distance = a[i];
            place = i;
        }
    }
    return place;
}

double distance2(const double *array_a, const double *array_b, int n){
    double Sum = 0;


    for(int i=0; i<n; i++){
        Sum += pow(array_b[i]-array_a[i], 2);
    }
    return Sum;
}

double distance_3(double b1, double b2, double b3, double a1, double a2, double a3){
    if(b1+b2+b3 > a1+a2+a3){
        if(distance_2(b1, b2, a1, a2) == INFINITY)
            return INFINITY;
        else
            return distance_2(b1,b2,a1,a2) + (b1+b2+b3)-(a1+a2+a3);
    }
    return INFINITY;
}

double distance_2(double b1, double b2, double a1, double a2){
    if(b1+b2 > a1+a2){
        if(distance_1(b1, a1) == INFINITY)
            return INFINITY;
        else
            return distance_1(b1, a1) + (b1+b2)-(a1+a2);
    }
    return INFINITY;
}

double distance_1(double b1, double a1){
    if(b1 > a1)
```

```
        return b1-a1;
    return INFINITY;
}


double distance1(const double *array_a, const double *array_b, int n){

    if(n == 1){
        if(array_b[0] > array_a[0])
            return (array_b[0]-array_a[0]);
        else
            return INFINITY;
    }


    double sum_a=0, sum_b=0;

    for(int i=0; i<n; i++){
        sum_a += array_a[i];
        sum_b += array_b[i];
    }
    if(sum_b > sum_a){
        if(distance1(array_a, array_b, n-1) < 0)
            return INFINITY;
        return sum_b - sum_a + distance1(array_a, array_b, n-1);
    }

    return INFINITY;
}
```