信州大学
大学院工学系研究科

修士論文

パケット間隔で情報を送るシステムの符号構成について

指導教員　西新 幹彦 准教授

専攻　電気電子工学専攻
学籍番号　08TA235D
氏名　西村 啓希

2010 年 3 月 18 日

目次

1 はじめに

1.1 研究の背景と目的

人間のコミュニケーション手段として最も広く使われている「言語」が情報理論の基本となる概念である. 例えば, 周りの騒音のために文の最初の方を聞き逃しても, 聞いていた人は言いたかったメッセージを理解出来る場合がある. そのような言語の堅牢性を構築するのが通信路符号化であり, 情報理論の基礎となる概念の一つである. 今日, 情報と呼ばれている様々なものの本質を定量的に捉え, そのための理論的な骨組みを数学的に体系化し, 現代のそして未来の情報機器に欠かすことの出来ない工学的手法の基礎が与えられている. 情報伝達のモデルを考える際, 多くは伝えたいメッセージそのものを送信する. メッセージを小さくパケットに分割して送るとすると, パケットを一度にまとめて, 圧縮して送信したり, 冗長を付加して送信したり様々な方法でそのものを受信側に伝える.

Anantharam and Verdú[1] は隣り合うパケットの時間間隔によっても情報を送ることが出来ると指摘している. 話し言葉の休止, 電話, 電子メールの時間間隔の休止等が情報を伝えるということである. タイミングとは送信されたパケットと隣り合うパケットの時間間隔のことであり, 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ことが出来る. この単純な見解から促される当面の問題は, 時間間隔による情報を組み入れて興味深い通信路モデルが構築できるかどうかである.

通常, 情報はパケットの内容として運ばれるのだが, 本研究では時間間隔のみによる情報伝達に着目したいためパケットの大きさ, 内容は考えない. そのような送信モデルを図 1 のような通信システムに導入する. 送信対象のメッセージを符号器によってパケットに変換し時間間

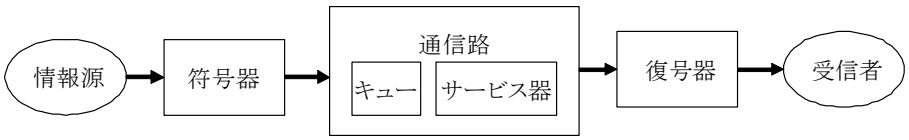

図 1 通信システムの構成

隔を作成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.

文献 [1] よりこの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, それを達成できるような符号を構成することを本研究において大きな目標とする. この通信路はサービスレートのみで特徴づけられ, 一般性を損なうことなく $\mu = 1$ の場合の符号を構成すれば十分である. 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.

まず符号構成のために本研究の通信システムの情報伝送能力を調べる. その部分問題として 2 種類の符号語となる時間間隔を送信する符号を定める. その時の伝送速度やビット誤り率 (BER) の特性をシミュレーションを行い評価する.

次に, 通信路符号化定理に基づいて実際に構成する. この定理で用いられる補題の証明で実際に符号の構成がされている. 実際に構成行い詳しい手順と性能結果を検証する.

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 種類の符号語で構成される符号の構成を試み, 伝送性能を評価する. 4 章では通信路符号化定理に基づいた符号構成を行う. 構成方法を紹介して実際に符号の導出を試みる. 5 章でまとめる.

2　通信システム

2 章では図 1 の通信システムの働きを確認する.

2.1　符号器と符号語

符号化は伝えたいシンボルを通信路に入力する形の符号語に変換する役割がある. その動作を実現する装置が符号器である.

本研究の場合は, ある送りたいメッセージを符号化することで対応する時間間隔の組に変換する動作を行う. その時間間隔が一般的な符号語を意味する. 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. 符号語シンボルの組合わせと使用頻度によって通信路に入力する際の入力レート λ が決まる.

符号化と逆の動作を復号化といい, それを実現する装置を復号器という. 復号器は 3 章のシミュレーション時にのみの使用となるのでそこで紹介する.

2.2　キューとサービス器

符号器によって送信されたパケットは通信路を介して復号器へと送られる. パケット通信を行なう際, ネットワーク内での様々な処理時間の変化や転送経路の変動等が発生する. 本研究のモデルでは, 図 1 のサービス器がこれらの処理を表していて, 処理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. μ によってパケットの一秒当たりの処理されるパケット数が決まるのでサービスレート μ という.

サービス器の手前にキューが一つ存在する. サービス器は一度に一つのパケットしか受けられない. サービス中に次のパケットが到着した場合, 到着順を乱すことなく待たせるところがである. 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. しかし, 到着レート λ の大きさがサービスレート μ より大きくなると, サービス器でパケット 1 つ当たり処理する時間よりパケットが到着する時間が下回り, 時間が経つにつれにキュー内のパケット数が増加していき, 十分大きいキューであったとしても溢れてしまう. したがって,λ と μ の間に条件

$$\lambda < \mu \tag{1}$$

が成り立っている場合だけを考える.

2.3 通信路の特徴

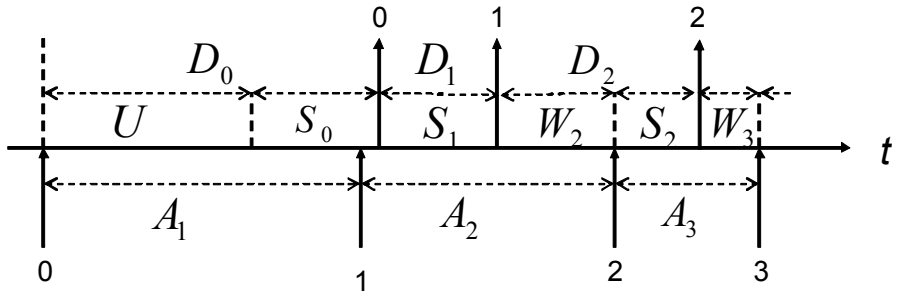

図 2　パケットの到着, 出発の様子

図 2 はパケットの通信路への到着, そして通信路を介してからの出発の簡単な様子を時間軸上で表したものである. 下から時間軸への矢印はパケットが通信路に到着する時刻であり, 次に到着するパケットにより形成される到着時刻の間隔 A_n が送りたい時間間隔となる. また時間軸から上への矢印はパケットが通信路を出発する時刻を表してあり, そして隣り合う出発時刻の間隔 D_n を出発間隔という.

最初に空もしくは平衡状態である通信路にパケット 0 を入力する. パケット 0 を入力した時, システム内にサービス処理が終わっていない n 個のパケットが存在するとする. それらのパケットをダミーパケットと呼ぶ. D_0 はパケット 0 のシステム滞在時間であり, 全てのダミーパケットの処理時間 U とパケット 0 自身のサービス時間 S_0 の和,

$$D_0 = S_0 + U$$

となる. 文献 [1, p.10] より D_0 は期待値が $\frac{1}{\mu-\lambda}$ の指数分布であると記してある. しかし確認すると, 期待値 $\frac{1}{(\mu-\lambda)\rho}$ の指数分布であると異なる結果が得られたためそれを指摘する. 導出過程は付録 A に示す. 4 章の理論的に符号構成を行う時にパケット 0 の滞在時間を考慮する. しかし 3 章のシミュレーションでは通信路が空の状態から開始する.

情報を運んでいる時間間隔は $A_1, A_2, \cdots$ と $D_1, D_2, \cdots$ である. 例えば図 2 の場合, パケット 0 以降の 3 つのパケットによって 2 つの時間間隔を送ったことになる. 通信路にはサービス器があり, パケットが到着した時にサービスを受けているパケットがいなければ到着と同時にサービスを開始する. しかし, 既にパケットが存在する時, キューで順番待ちをしてサービス順が回ってくるのを待つ. サービス器に入ったパケットは指数分布に従って S_n 秒間のサービス処理を受ける. また, サービス処理がパケットの到着の早さを上回った時, 通信路内に何もない

状況が起こることがある．図 2 では, 2 つ目, 3 つ目のパケットの到着を待つ時間が存在している. その空である時間をアイドリング時間といい,W_n で表している．アイドリング時間は

$$W_i = \max\Big\{0, \sum_{j=1}^{i} A_j - \sum_{j=0}^{i-1} D_j\Big\} \tag{2}$$

と表される．このように, 通信路ではサービス時間や待ち時間, アイドリング時間によって送信した到着間隔が影響を受け, 出発間隔が変化する．図 2 からも対応する時間間隔が変化していることが読み取れる．それらの時間間隔の変化の要因となるものを雑音と呼ぶ. 雑音を N_n で表すとすると N_n は出発間隔 D_n と到着間隔 A_n の差分,

$$N_n = D_n - A_n$$

となる．このように符号語となる時間間隔を変動させる要因となる雑音が存在するため, 誤りが発生する通信路であることが分かる．図 2 からパケット 1 が入力した時パケット 0 の出力を待っている. その待ちが次のパケットのサービス開始時間に影響を与えている．このように雑音は各時間間隔で独立しておらず, 一度雑音が入ると多少なりと以降に影響を与える．よって記憶のある扱い辛い通信路である.

2.4 通信路容量

通信路容量とは, 通信路の雑音の分布を考慮に入れて最も情報が多く伝わるように符号語の分布を定めた時の情報伝送速度である. 情報伝送速度とは一定時間当たりに受信者に転送することの出来る情報の量である．文献 [1] より, 通信路への到着レート λ が可変でサービスレートが $\mu = 1$ と固定された G/M/1 キューシステムの時の通信路容量が λ の関数として

$$C(\lambda) = \lambda \log \frac{\mu}{\lambda} \quad [\text{bit/sec}] \tag{3}$$

のように求められている. ここで対数の底は 2 とし, 情報量は bit で換算する．「G/M/1」はケンドールの記号と呼ばれ, この場合, 到着過程は一般的, サービス器は指数分布に従う, サービス器が一つ存在するシステムを意味している．負荷に対する通信路容量 (3) を図 3 に示す. λ/μ を負荷といい, 条件 (1) を満たす．範囲は $0 \leq \frac{\lambda}{\mu} \leq 1$ となる．図から通信路容量が小さくなるのが二点見られる．λ が小さく負荷が 0 に近い時通信路容量が小さいのは, サービス処理の早さに対して到着が遅いためキューの中は空に近い状態で誤りは少ないが単位時間伝達できる情報量自体が少ないからである. 負荷が 1 付近の時は, 単位時間当たりのパケット到着数と処理数が近くキューの中にはパケットがたくさん存在している. ある到着間隔で送られてきてもキューで待つことによって間隔による情報がそこで失われる. そして誤る確率も増え正しい情報をあまり伝えられなくなる. よって正しく送ることのできる伝送レートは小さくなる.

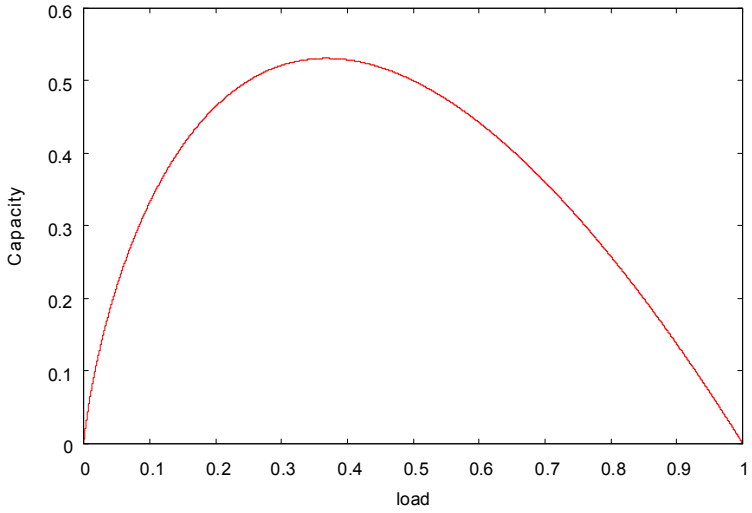

図 3　負荷 λ/μ に対する「G/M/1」システムの通信路容量 (λ 可変)

本研究で課題となるのはいかにして通信路容量に近づけられるような符号を構成するかである．次の章で符号構成のために本研究の通信システムの情報伝送能力を調べる．そしてこの通信路のもつ特徴を捉える．

3　符号構成へのアプローチ

3 章では以上で紹介した通信システムの情報伝送能力を調べる．時間間隔を通信路に通した時の様子を評価したいため符号器で変換する工程を省き, 符号語となる時間間隔を定める．

導入問題として 2 種の符号語となる到着時間間隔を用いることから始める. 各符号語の出現頻度によって単位時間当たり通信路に送られる情報量, つまり伝送速度が変化する．まず伝送速度の評価をコスト付き符号化と照らし合わせながら行う．

3.1　コスト付き符号化

ここで雑音の影響は無視し, 符号器が通信路に送り出す情報の伝送速度に着目する．時間間隔によって情報を送るということから, その時間間隔はあるメッセージの送信に必要なコストとみなす事ができる．この時, 問題は情報源のコスト付き符号化に帰着される．本研究で用いるパケット間隔は 1 つのシンボルの送信に必要なコストに相当する．これは雑音のない通信路符号化問題として定式化されたものであることに注意する．時間 t で送信可能な系列の総数を $N(t)$ によって表し, 単位時間当たり送信できる情報量の上限であるコスト容量 α_c は次のように定義される．

$$\alpha_c = \lim_{t\to\infty} \frac{\log N(t)}{t} \tag{4}$$

コストを c とした時コスト容量 α_c は,

$$\sum_{c\in\mathcal{C}} 2^{-\alpha_c c} = 1 \tag{5}$$

の唯一の実数解として与えられる [3]．これより, 誤りなしでのコスト当り送信できるビット数の上限が求められる．通信路がこのコスト容量を有するとき, 記号当たりの平均コストを最小化する符号を構成する必要がある．

コスト付き符号化においては, コストの大きい符号語はあまり出てこず, コストの小さい符号語が何度も出現すると考えられる．その時, コスト付き符号化によって得られる符号語列における各記号の分布 (符号語の使用頻度) はどうなるか．漸近的に平均コストを最小にする符号において, その符号語中の記号の分布が次のようにコストによって定まる分布 Q を有していることが知られている [4].

$$Q(a) = 2^{-\alpha_c c} \tag{6}$$

特に, すべてのコストが等しければ分布 Q は一様分布となり, これは情報理論でよく知られた事実と一致する．

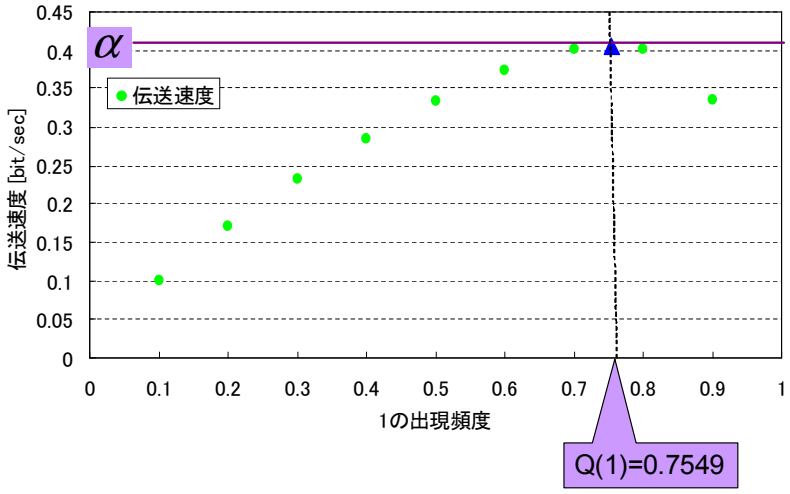

図 4　伝送速度とコスト容量

3.2　符号語が {1,5} の時

はじめに, 2 種の時間間隔 {1,5} を送信する符号器を用いた場合を考える.

3.2.1　コスト付き符号化の適応と伝送速度

3.1 節にしたがってコストが $\mathcal{C} = \{1, 5\}$ の時のコスト容量とその時の符号語の分布を算出する. 式 (5) より, $2^{-\alpha_c \cdot 1} + 2^{-\alpha_c \cdot 5} = 1$ となるので数値計算でこれを解くと,

$$\alpha_c \simeq 0.4056 \tag{7}$$

が得られる. 式 (6) より符号語の分布は,

$$\begin{aligned} Q(1) &= 2^{-\alpha_c \cdot 1} \\ &\simeq 0.7549 \\ Q(5) &= 2^{-\alpha_c \cdot 5} \\ &\simeq 0.2451 \end{aligned}$$

と求まる. 図 4 に時間間隔 1 の出現頻度を 0.1 刻みで変化させた時の伝送速度とコスト容量 (7) を示す.

伝送速度は $Q(1) = 0.1$ の時が最も小さくなり $Q(1) = 0.7549$ の時まで単調増加となり, そこから少し減少した. {1,5} の内時間の短い方を多用すると多くの情報が送られるため伝送速

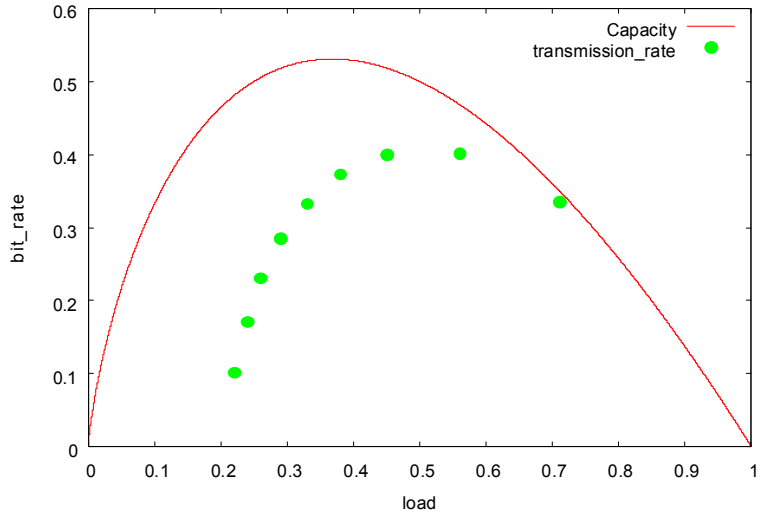

図 5　通信路容量と伝送速度

度が右上がりとなることはわかる．$Q(1) = 0$, $Q(5) = 0$ の時は用いるコストがどちらか一方のみとなってしまうので送られる情報量は 0 となり伝送速度も 0 となる．つまりエントロピーが 0 となるからである．コスト付き符号化を適用し, 与えたコスト {1,5} の元で伝送速度を最大にすることができた．

次に図 5 に通信路容量 (3) と一緒に示す．横軸は負荷 λ/μ で統一する．頻度と負荷の関係は次である．間隔 {1,5} の出現頻度を $p(1)$, $p(5)$ とすると到着レート λ は $\lambda = \frac{1}{1 \cdot p(1) + 5 \cdot p(5)}$ となり, サービスレートは $\mu = 1$ なので $\frac{\lambda}{\mu} = \lambda$ となる．左から順に $p(1) = 0.1, 0.2, \cdots, 0.9$ の時である．間隔の短い 1 の使用頻度を大きくすると負荷も大きくなる．これは通信路へ入力される伝送速度を示している．図 5 の負荷が 0.7 付近でビットレートが通信路容量に近づけられているように見えるがそうではない．通信路容量は通信路内の誤りを考慮した上のビットレートであるのに対しこれは入力側のビットレートなのである．前にも述べたようにこの通信路は誤りが発生するため, 受信者に誤りなしで送信されるビットレートの結果はまた大きく変わる．

3.2.2　**復号器**

本研究における復号器は 2 種類の符号語を推定する働きがある．符号語は時間間隔であるので通信路を通った後, 雑音により変化して任意の正数になる可能性がある．そこで最も単純な復号の仕方として, 基準となる閾値を定める方法がある．最尤法という観察された出来事が理論的に最も起こりやすいような値を観察されたデータから推測する方法論からも導ける．本研究では通信路容量に対してどれほど達成できるのか観察をしたいので, その都度最良の性能結

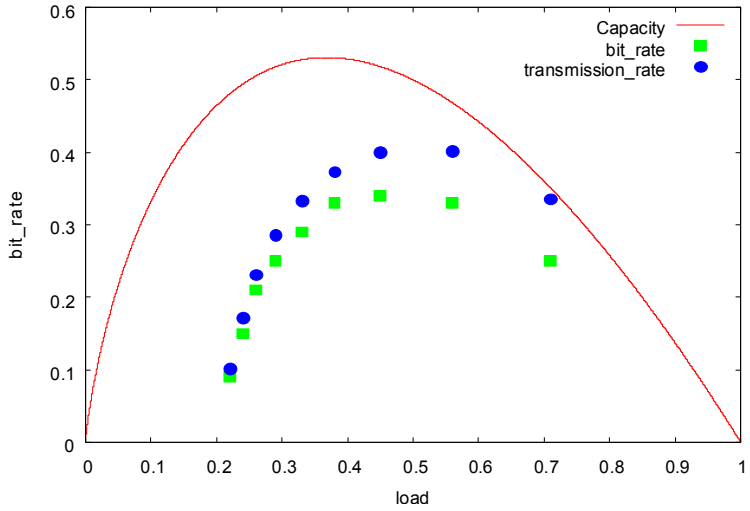

図 6　通信路容量と比較

果を必要としたい．したがって今回用いる復号器は, 実際に通信路を通った間隔を観測し誤り確率が最小となる閾値を採用した復号器を扱うことにする．

3.2.3　シミュレーション結果

実際に通信路に通した時の正しく復号されるビットレートを示す. 図 5 にシミュレーション結果を追加したものを図 6 に示す．通信路容量と比較したいのはこの結果である．

やはり誤りが発生し, 符号器から送った情報量に対して受信者に伝えられたビットレートは低下してしまう．前記のような実際の観測時の誤りが最小となるような復号器を用いているため, 符号語を {1,5} の時の達成可能領域はこれ以上よくはできない．

次は正しくどれだけ送れたかではなくどれだけの誤りが生じたのかについて評価する．図 7 に伝送速度と対応する BER を示す．参考としてコスト容量も載せる．横軸が伝送速度で縦軸が BER である. 下側のプロットから順に間隔 1 の使用頻度が 0.1,0.2,$\cdots$ の時である．1 の使用頻度が大きくなるにつれて BER が大きくなっていることに気づく. コスト付き符号化を適応し, 平均コストを最小にし伝送速度を最大にできたとしても BER がおおきくなっている. 2.3 節より, 短い間隔を通信路に入力するとキューで待つパケットが多くなり, 情報となる時間間隔が失われる確率が高くなるため BER が大きくなったと考えられる．よく知られた事実のように本研究においても伝送速度と BER はトレードオフの関係となることがわかった．

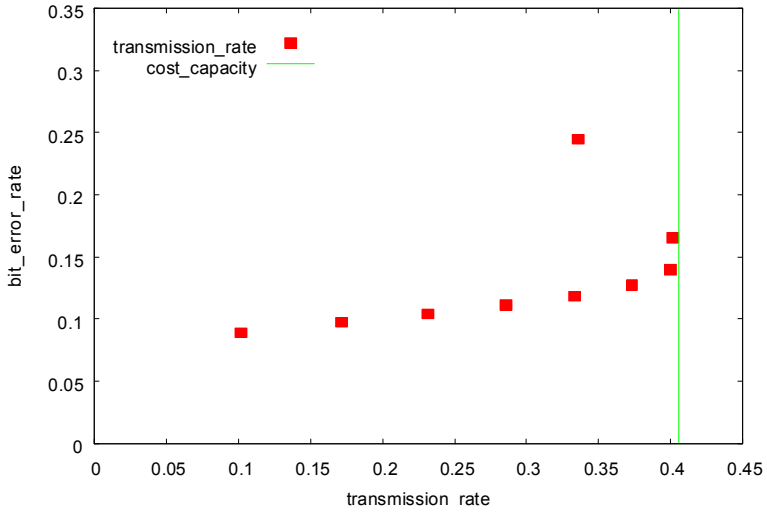

図 7　伝送速度と BER

3.3　2 種の符号語の拡張

3.2 節では符号語を {1,5} と限定させた符号を用いた. その結果, 2 元対称通信路のようには扱えないことが分かった. ここでは 2 種類のままではあるが値を限定せずに間隔の大きさを変える. 正数全体を取り得るため限界はあるが, できる範囲でシミュレーションを行い特徴を詳しく掴んでいく.

3.3.1　シミュレーション結果

2 種の符号語を $\{a, b\}$ とおき図が描けるよう様々な組でシミュレーションを行う. 組み合わせの条件は $a < b$ である. また各間隔の使用頻度はそれぞれの組において 0.1 毎に変化させ BER が最小となる時を選ぶ.

図 8 に様々な組の BER を示す.

間隔 {a,b} 両方が長くお互い離れている時に BER は最も小さくなり, 反対に短くてお互い近い間隔の時は BER は大きくなることが分かった. 3.2 節の結果と同様である. 時間間隔を扱っているため長い間隔の方が余裕があり雑音の影響はあまりないと考えられる.

図 9 にあらゆる組の伝送速度と BER を示す. 1 組に 1 プロット打つことができる.

これからもトレードオフの関係が読み取れる. 符号語が 2 種類の場合で符号を構成する際の達成可能領域が明らかになった. BER を下げたまま伝送速度を上げ, 達成可能領域がより右へ拡大すると良い符号だと言える.

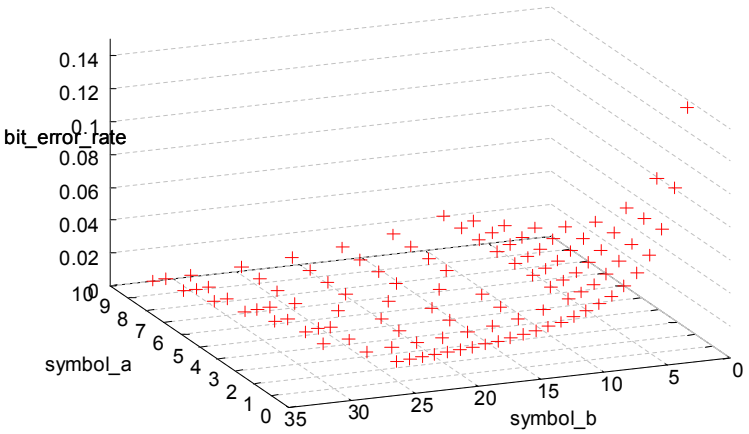

図 8　組み合わせ $\{a, b\}$ の BER

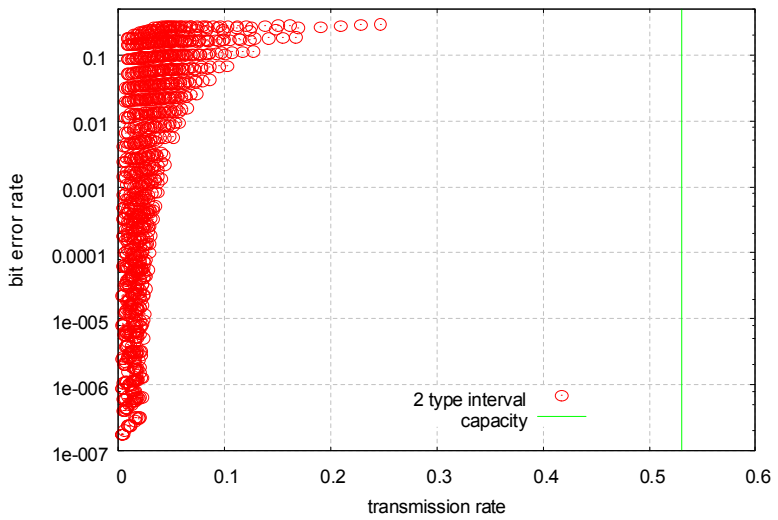

図 9　伝送速度-BER 特性と通信路容量

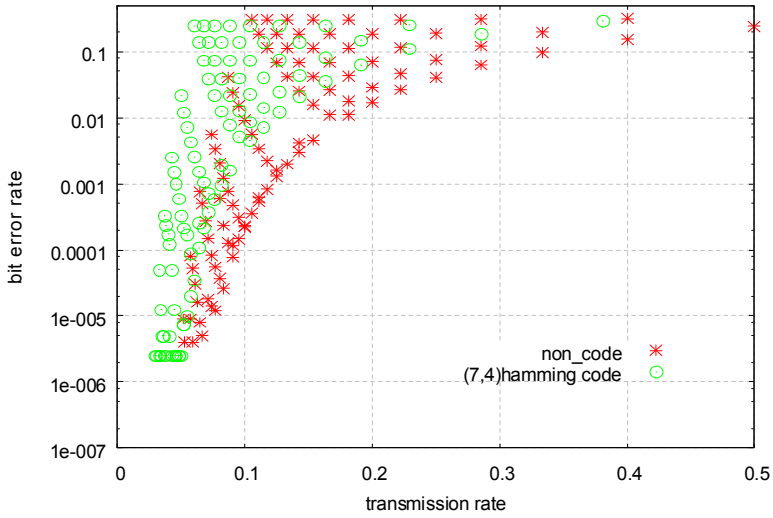

図 10　{a,b} 使用頻度等確率時

3.4　誤り訂正符号の導入

次は BER を抑えて信頼性を向上させることに着目する．BER を下げる改善策の一つとして, 単一誤りの訂正ができる基本的な (7,4)Hamming code を適用する．

3.4.1　シミュレーション結果

様々な 2 種類の符号語の組 $\{a, b\}$ のシミュレーション結果に (7,4)Hamming code を適応した結果を以下に示す．図 10 に $\{a, b\}$ を等確率で使用した時の伝送速度, BER 特性を示す．

符号化時に冗長を付加しているため伝送速度は大幅に低下した．BER はわずかに小さくすることができた．これは使用頻度を等確率とした時だが,3.2.3 項の図 7 の結果より,2 つの符号語の内短い方の使用頻度が小さいほど BER が小さくなることがわかっている．よって間隔 a の使用頻度を抑えた組の結果は誤りの数が減り, Hamming 符号によって単一誤りを訂正する機会が多くなるため BER はさらに抑えられることが予想される．逆に間隔 a の頻度が多い符号に適応させると誤りの数が多くなる. すると単一誤り訂正ができない 2 つ以上の誤りが発生し, 誤訂正を行うことで誤り訂正を行うより悪くなる．その結果を図 11 に示す．

これは a の頻度を大小全て網羅した結果である. 考察した要因より BER が改善された符号もあるだろうが, 全体からは BER は悪くなったように見える．(7,4)Hamming code の訂正能力を越える誤りが生じるような通信路であることがわかる．

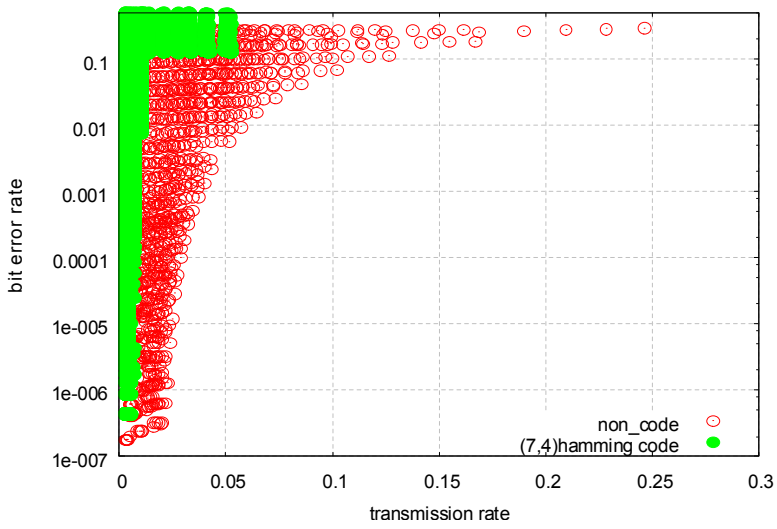

図 11　{a,b} 使用頻度偏りあり

他にも様々な誤り訂正符号は存在するのだが, 闇雲に適用しても意味がない．本研究ではやはり符号器においてどのような時間間隔を用いるかに興味がある．次章で符号化定理における理論的な符号構成を試みる．

4 符号構成

通信路符号化定理に基づいた符号構成を行う.

4.1 通信路符号化定理

文献 [1] では通信路容量を求める際に次の通信路符号化定理が用いられている.

伝送すべきメッセージの集合を $\mathcal{M}_n = \{1, 2, \cdots, M_n\}$ とし, それらを長さ n の通信路入力に変換する写像 $\varphi_n : \mathcal{M}_n \to \mathcal{X}^n$ を符号器と呼ぶ. ここで, $\mathcal{X}^n$ は長さ n の通信路入力の集合, $u_i = \varphi_n(i)$ はメッセージ i の符号語であり, $\mathcal{C}_n = \{u_1, u_2, \cdots, u_{M_n}\}$ は符号となる.

定理 1([2, 定理 3.2]) $\boldsymbol{X} = \{X^n\}_{n=1}^{\infty}$ を任意の通信路入力としそれに対する通信路 $\boldsymbol{W} = \{W^n\}_{n=1}^{\infty}$ からの出力を $\boldsymbol{Y} = \{Y^n\}_{n=1}^{\infty}$ とする. 通信路$\boldsymbol{W}$の通信路容量 $C(\boldsymbol{W})$ は

$$C(\boldsymbol{W}) = \sup_{\boldsymbol{X}} \text{p-}\liminf_{n\to\infty} \frac{1}{n} i(X^n; Y^n)$$

で与えられる. ただし,

$$\text{p-}\liminf_{n\to\infty} Z_n \equiv \sup\{\beta | \lim_{n\to\infty} \Pr\{Z_n < \beta\} = 0\}$$

$$i(\boldsymbol{x}; \boldsymbol{y}) \equiv \log \frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})}$$

と定義する. ここで, この章では対数の底を e とする.

この定理の証明では符号の存在を保証する補題が用いられる. その補題は次である.

補題 1([2, 補題 3.8]) M_n を任意に与えられた正整数とすると, すべての $n = 1, 2, \cdots$ に対して誤り確率 ϵ_n が

$$\epsilon_n \le \Pr\Big\{\frac{1}{n} i(X^n; Y^n) \le \frac{1}{n} \log M_n + \gamma\Big\} + e^{-n\gamma} \tag{8}$$

を満たす符号が存在する. $\gamma > 0$ は任意の定数である.

右辺を λ_n とおくと

$$\epsilon_n \le \lambda_n$$

となる. $\alpha = M_n e^{n\gamma}$ とすると

$$\begin{aligned} \lambda_n &= \Pr\Big\{\frac{1}{n} \log \frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})} \le \frac{1}{n} \log \alpha\Big\} + \frac{M_n}{\alpha} \\ &= \Pr\Big\{\frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})} \le \alpha\Big\} + \frac{M_n}{\alpha} \end{aligned} \tag{9}$$

と書き換えられる.

ここで, 通信路入力シンボル $\boldsymbol{x}$ が本研究の到着間隔 a, 出力シンボル $\boldsymbol{y}$ が出発間隔 d に対応することになる.

4.2 符号構成法

実は補題 1 の証明では実際に符号を構成しているのである. 全体の証明は文献 [2] で記してあるが, ここで符号構成法を取り上げて紹介する.

$\boldsymbol{x}\in\mathcal{X}^n$ に対して,

$$B(\boldsymbol{x}) = \left\{\boldsymbol{y}\in\mathcal{Y}^n | i(\boldsymbol{x};\boldsymbol{y}) \geq \log M_n + \gamma\right\} \tag{10}$$

とおく. 変形すると

$$B(\boldsymbol{x}) = \left\{\boldsymbol{y}\in\mathcal{Y}^n | \frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})} \geq \alpha\right\} \tag{11}$$

そして, 通信路に対する符号 C_n を次のように構成していく. まず, 条件

$$W^n(B(u_1)|u_1) \geq 1-\lambda_n \tag{12}$$

を満たす符号語 $u_1\in\mathcal{X}^n$ を任意に選ぶ. 次に,

$$W^n(B(u_2)-B(u_1)|u_2) \geq 1-\lambda_n \tag{13}$$

なる $u_2\in\mathcal{X}^n$ を任意に選ぶ. このような操作を繰り返し行うことで符号 C_n を構成できる. この手順で漸近的に最良な符号が構成できることは理論的に分かっているのだが, 以降で実際に構成を行う.

4.3 通信路パラメータ

通信路 $\mathbf{W}$ として本研究の通信路を代入することにより, 漸近的に最適な符号を構成することができる. 本研究の通信路の容量は λ の関数として式 (3) となる. 底は e なので単位の変更に注意する.

$$C(\lambda) = \lambda \log \frac{\mu}{\lambda} \quad [\mathrm{nat/sec}]$$

のようになることに注意する.

通信路容量はこの最大値なので

$$\begin{aligned} C &= \sup C(\lambda) \\ &= e^{-1}\mu \quad [\mathrm{nat/sec}] \end{aligned} \tag{14}$$

その時の到着レートは

$$\lambda = e^{-1}\mu \quad [\text{symbol/sec}] \tag{15}$$

である.

これを最適レートとみなし, 式 (15) のレートで入力させると仮定する. また,「M/M/1 キューシステムの出力過程は入力過程と同じレートのポアソン過程」を意味する Burke の定理を適応させると出力過程までも決定することができるので, 入力は式 (15) のレートのポアソンに従うと仮定する. すると, 出力過程も同様の過程となる.

通信路容量の次元を変換する.

$$\begin{aligned} R_0 &= \frac{C}{\lambda} \\ &= \frac{e^{-1}\mu}{\lambda} \\ &= 1 \quad [\text{nat/symbol}] \end{aligned} \tag{16}$$

3 章の符号設定と同様に行いたいので, この章でも符号語の長さ n が 1, メッセージの数 M_n を 2 とする. 符号化レートは $R = \frac{1}{n}\log M_n$ なので, $R = \log M_1 = \log 2 \approx 0.6931$ また γ は

$$\gamma = \frac{R_0 - R}{3}$$

と定義する. よって $\gamma = \frac{1-0.6931}{3} = 0.1023$ となる. また, $\alpha = M_n e^{n\gamma}$ なので $\alpha = 2e^{0.1023} \approx 2.2154$ となる.

$n = 1$ の時のパラメータは以上である. 本研究における通信路はサービスレートのみで特徴づけられる. 一般性を損なうことなく $\mu = 1$ の場合の符号を構成すれば十分である.

4.4 1 符号語シンボルによる符号化

$n = 1$ の時の結果を以下に示す. 導出過程は付録 B にて示す.

誤り確率の上限 λ_n, 式 (9) は

$$\begin{aligned} \lambda_n &= \Pr\Big\{ \frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})} \leq \alpha \Big\} + \frac{M_n}{\alpha} \\ &\approx 1.8077 \end{aligned} \tag{17}$$

となり, 正しく復号される確率の下限 $1 - \lambda_n$ は

$$1 - \lambda_n \approx -0.8077 \tag{18}$$

である.

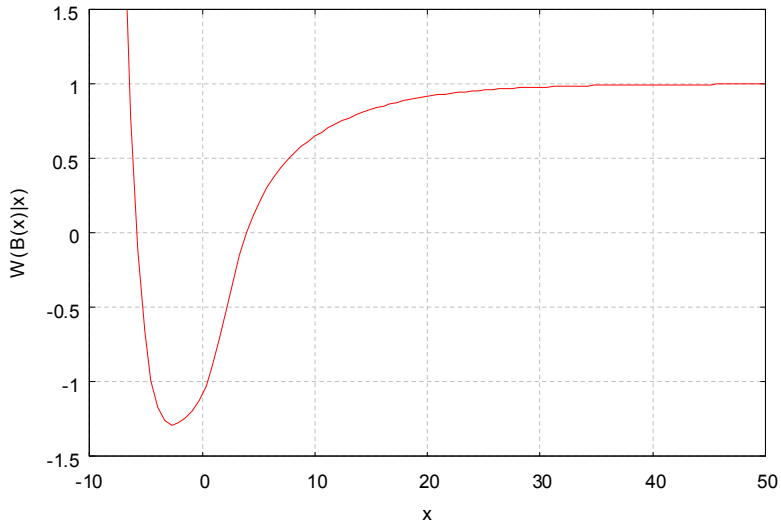

図 12　W(B(x)|x)

次に確率分布 (11) の $W(B(x)|x)$ を求め, 符号構成に必要な条件式 (12),(13) を用いて符号の構成を行う. $W(B(x)|x)$ を図 12 に示す.

条件 (12) と結果 (18) より

$$\begin{aligned} W^1(B(u_1)|u_1) &\geq 1-\lambda_n \\ &= -0.8077 \end{aligned} \tag{19}$$

正しく復号される確率の下限が負であるため, 式 (12) を満たす符号語 u_1 は任意の値となる.

従来研究に合わせて一方を $u_1 = 5.0$ とする. u_1 の復号領域 $B(u_1)$ は式 (33),(38) の $v_1(u_1)$, $v_1'(u_1)$ より領域が求まる.

$$\begin{aligned} B(u_1) &= [v_1'(u_1), v_1(u_1)] \\ &= [4.7027, 5.2966] \end{aligned}$$

となる.

次に符号語 u_2 を条件 (13) に従って選ぶ. $M_n = 2$ なので, u_2 の復号領域 $B(u_2) = [v_1'(u_2), v_1(u_2)]$ の左端 $v_1'(u_2)$ は $B(u_1)$ の右端 $v_1(u_1)$ と一致させることで復号領域に隙間がなくなる. これを閾値とすることが最も効率がよい. すると式 (38) より

$$u_2 \approx 6.5335$$

が得られる. 復号領域は

$$\begin{aligned} B(u_2) &= [v_1'(u_2), v_1(u_2)] \\ &= [5.2966, 7.7105] \end{aligned}$$

である. しかし, メッセージ数は $M_n = 2$ であるため特に上限を定める必要はない. よって閾値が 5.2966 となるような復号器が構成された. 以上が $n = 1$ の時の符号構成である.

4.5 考察

ここで符号構成手順の紹介と実際に導出を行った. すると導出の際の問題点や難解さが浮かび上がった. 付録 B で導出している通信路の条件付確率 (23) の分解, 場合分けが n を増す度に指数的に増加する. また式 (34) を解くのが困難となってしまう. 本研究では明確な根拠なしに荒い近似を採用したがさらに深い根拠づめをする必要がある. これも n を増す度に増加する.

$n = 1$ の時, 正しく復号される確率の下限 (19) が負であるため性能の保証が全くできない. そのためシミュレーションを行う意義がない. 下限が正の値となることで初めてそれだけの保証が持てる符号が構成されたこととなる. そこで本当にそれだけの性能の保証がされているのか確認し, 符号構成方法も間違いがないということを再認識できる. 符号化定理によると n を極限に飛ばした時誤り確率を 0 にすることができ, 通信路容量を求めることができる. n を大きくしていくとメッセージ数 M_n も増え, 符号化レートもより大きくなることが予想される. 次は順番通り $n = 2$ の時についての符号構成も行いたい. 次数を大きくすることで性能はよくなっていくと予想したが次の $n = 2$ で正しく復号される確率が正となり保証できるものとなるのかは分からない. 途中までであるが付録 C でそれを確認する.

5 まとめ

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ことができることが文献 [1] で指摘された. 同時に図 1 のような通信システムで時間間隔を情報として送受信した場合の通信路容量が求められている.

通信路容量を達成する符号の構成が目的であるが, 本研究ではまず通信路内で起こる現象を把握した. サービス器によるサービス時間, キューによる待ち時間によって送りたい時間間隔が変動する. そしてその雑音は各時間間隔で独立しておらず記憶のある扱いにくい通信システムであることが分かった.

通信路容量を達成するための最初の取り組みとして 2 種類の時間間隔を用いる符号について性能を評価した. まず平均コストを最小にできるコスト付き符号化を適応させた. コスト付符号化を適応させることで伝送速度を最大にできるのだが, 雑音が生じる通信路であるため BER が発生する. 伝送速度と BER はトレードオフ関係にあることが分かった. また 2 種の時間間隔を変化させシミュレーションを行うと短い間隔よりも長い間隔の方がキューでの待ち時間が少なくなり BER は小さくなった. その後参考程度に誤り訂正符号の導入を試みた.

最後に通信路符号化定理に基づいた符号構成を行った. 前のシミュレーション時と同様, 符号語長が 1 の時の符号構成を実際に行った. 実際に使用可能な符号はできなかったが, 構成手順の詳細を確認することができ, 計算が困難となるポイントを発見した. これは今後符号語長を拡大していく上で必ずある障害であるため, その克服が不可欠となる.

謝辞

本研究を行うにあたり, 数多くの助言, 指導をしていただいた指導教員の西新幹彦准教授, 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, January 1996.

[2] 韓 太舜, “情報理論における情報スペクトル的方法,” 培風館, 1998 年

[3] 植松 友彦, “情報源のコスト付き符号化とその応用,” 電子情報通信学会論文誌 A, vol. J87-A, No.5, pp.588-596, 2004 年 5 月

[4] T.S.Han and O.Uchida, “Source Code with Cost as a Nonuniform Random Number Generator,” IEEE Trans. Inf. Theory, vol. 46, no.2, pp.712-717, 2000.

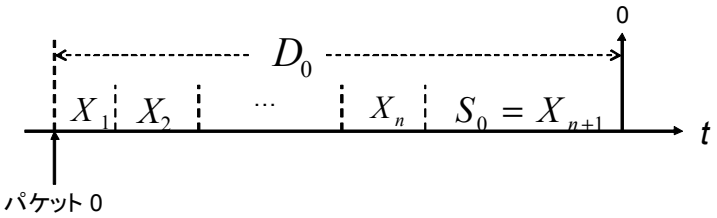

図 13　パケット 0 の入出力

付録 A　パケット 0 の出力分布の導出

パケット 0 の出力分布を求め直す. 図 13 にパケット 0 の入出力の様子を示す.

ダミーパケットにより n 個の間隔があるとする. それとパケット 0 が滞在する. ダミーパケットはサービス器の指数分布より滞在時間が決まる. サービス時間を確率変数 X_n とする. またパケット 0 のサービス時間も同様に指数分布で決定するので,S_0 を X_{n+1} とおく.

D_0 の分布は

$$\Pr\{D_0 < \alpha\} = \sum_{n=0}^{\infty} \Pr\Big\{\sum_{i=1}^{n+1} X_i < \alpha\Big\} \Pr\{N = n\} \tag{20}$$

となる.

$\Pr\{N = n\}$ はシステムにパケットが n 個滞在している確率であり, システムの利用率が $\rho = \frac{\lambda}{\mu}$ なので, 幾何分布

$$\Pr\{N = n\} = (1 - \rho)\rho^n \quad (n = 0, 1, 2, \cdots) \tag{21}$$

に従う. 次に $\Pr\Big\{\sum_{i=1}^{n+1} X_i < \alpha\Big\}$ を

$$\Pr\{X_1 + X_2 + \cdots + X_{n+1} < \alpha\} = F_{n+1}(\alpha)$$

のようにおく. $\Pr\{N(t) = n\}$ は

$$\begin{aligned}
\Pr\{N(t) = n\} &= \Pr\{N(t) \leq t\} - \Pr\{N(t) \leq n - 1\} \\
&= \Pr\{S_{n+1} > t\} - \Pr\{S_n > t\} \\
&= (1 - F_{n+1}(t)) - (1 - F_n(t)) \\
&= F_n(t) - F_{n+1}(t)
\end{aligned}$$

と変形できる. これより漸化式

$$F_n(t) = F_{n-1}(t) - \Pr\{N(t) = n-1\} \quad (n \geq 1)$$

が得られる. n に値を代入する.

n=0 の時, $F_n(t)$ は

$$F_0(t) = \begin{cases} \Pr\{0 < t\} = 1 & t > 0 \\ 0 & t = 0 \end{cases}$$

となる. $\Pr\{N(t) = n\}$ は

$$\begin{aligned} \Pr\{N(t) = 0\} &= \Pr\{X_1 > t\} \\ &= 1 - \Pr\{X_1 < t\} \\ &= 1 - (1 - e^{-\lambda t}) \\ &= e^{-\lambda t} \end{aligned}$$

となる. また、システム内のパケット数は

$$\Pr\{N(t) = n\} = \frac{(\lambda t)^n e^{-\lambda t}}{n!}$$

のポアソン分布に従う.

したがって n=1 の時の $F_n(t)$ は

$$\begin{aligned} F_1(t) &= F_0(t) - \Pr\{N(t) = 0\} \\ &= 1 - e^{-\lambda t} \end{aligned}$$

であり, n=2 の時の $F_n(t)$ は

$$\begin{aligned} F_2(t) &= F_1(t) - \Pr\{N(t) = 1\} \\ &= (1 - e^{-\lambda t}) - \lambda t e^{-\lambda t} \\ &= 1 - (1 + \lambda t) e^{-\lambda t} \end{aligned}$$

であり, n=3 の時の $F_n(t)$ は

$$\begin{aligned} F_3(t) &= F_2(t) - \Pr\{N(t) = 2\} \\ &= (1 - (1 + \lambda t) e^{-\lambda t}) - \frac{(\lambda t)^2 e^{-\lambda t}}{2!} \\ &= 1 - \left\{ 1 + \lambda t + \frac{(\lambda t)^2}{2!} \right\} e^{-\lambda t} \end{aligned}$$

となり, 帰納法より

$$\begin{aligned} F_{n+1}(t) &= 1 - \Big\{1 + \lambda t + \frac{(\lambda t)^2}{2!} + \cdots + \frac{(\lambda t)^n}{n!}\Big\} e^{-\lambda t} \\ &= 1 - \sum_{i=0}^{n} \frac{(\lambda t)^i e^{-\lambda t}}{i!} \end{aligned}$$

が示される. したがって式 (20) は

$$\begin{aligned} \Pr\{D_0 < \alpha\} &= \sum_{n=0}^{\infty} \Pr\Big\{\sum_{i=1}^{n+1} X_i < \alpha\Big\} \Pr\{N = n\} \\ &= \sum_{n=0}^{\infty} \Big\{1 - \sum_{i=0}^{n} \frac{(\lambda t)^i e^{-\lambda \alpha}}{i!}\Big\} (1-\rho)\rho^n \\ &= \sum_{n=0}^{\infty} \Big\{(1-\rho)\rho^n - (1-\rho)\rho^n \sum_{i=0}^{n} \frac{(\lambda t)^i e^{-\lambda \alpha}}{i!}\Big\} \\ &= 1 - (1-\rho) \sum_{n=0}^{\infty} \rho^n \sum_{i=0}^{n} \frac{(\lambda \alpha)^i e^{-\lambda \alpha}}{i!} \\ &= 1 - (1-\rho) e^{-\lambda \alpha} \sum_{i=0}^{\infty} \sum_{n=i}^{\infty} \rho^n \frac{(\lambda \alpha)^i}{i!} \\ &= 1 - e^{-\lambda \alpha} \sum_{i=0}^{\infty} \frac{(\rho \lambda \alpha)^i}{i!} \\ &= 1 - e^{-\lambda \alpha} e^{\rho \lambda \alpha} \\ &= 1 - e^{-(\mu - \lambda)\rho \alpha} \end{aligned}$$

となる. これより, D_0 は期待値 $\frac{1}{(\mu-\lambda)\rho}$ の指数分布となる.

付録 B　$n = 1$ の時の符号導出過程

通信路の条件付確率 $W^n(\boldsymbol{y}|\boldsymbol{x})$ を求める. 文献 [1] の式 (2.24) より

$$\begin{aligned} W^n(\boldsymbol{y}|\boldsymbol{x}) &= P_{Y^n|X^n}(y_0, \cdots, y_n|x_1, \cdots, x_n) \\ &= e_{(\mu-\lambda)\rho}(y_0)\prod_{i=1}^{n} e_\mu(y_i - w_i) \end{aligned}$$

と表され, また

$$= \int_{y_0=0}^{\infty} P_{y_0,Y^n|X^n}(y_0, \boldsymbol{y}|\boldsymbol{x})y_0$$

のようにも表現できる

$n = 1$ を代入すると

$$W^1(\boldsymbol{y}|\boldsymbol{x}) = \int_{y_0=0}^{\infty} e_{(\mu-\lambda)\rho}(y_0)e_\mu(y_1 - w_1)dy_0 \tag{22}$$

となる. ここで, アイドリング時間 w_i は一般に

$$w_i = \max\Big\{0, \sum_{j=1}^{i} x_j - \sum_{j=0}^{i-1} y_j\Big\}$$

と表され, 特に

$$w_1 = \max\Big\{0, x_1 - y_0\Big\}$$

なので代入した指数部分は

$$e_\mu(y_1 - w_1) = \begin{cases} e_\mu(y_1) & y_0 \geq x_1 \\ e_\mu(y_1 - (x_1 - w_1)) & y_0 < x_1 \end{cases}$$

となる. したがって

$$W^1(\boldsymbol{y}|\boldsymbol{x}) = \int_{y_0=0}^{x_1} e_{(\mu-\lambda)\rho}(y_0)e_\mu(y_1 - (x_1 - y_0))dy_0 + \int_{y_0=x_1}^{\infty} e_{(\mu-\lambda)\rho}(y_0)e_\mu(y_1)dy_0 \tag{23}$$

と分解される. 指数部分の肩は負にはならないため, 次のように場合分けされる. 以降から添字 1 は省略する.

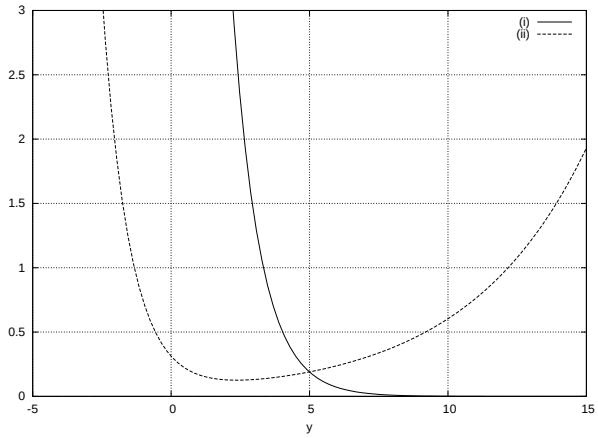

図 14　W(y|x) (x=5)

(i) $x < y$ の時

$$\begin{aligned} W^1(\boldsymbol{y}|\boldsymbol{x}) &= \int_{y_0=0}^{x} e_{(\mu-\lambda)\rho}(y_0)e_{\mu}(y-(x-y_0))dy_0 + \int_{y_0=x}^{\infty} e_{(\mu-\lambda)\rho}(y_0)e_{\mu}(y)dy_0 \\ &= \int_{y_0=0}^{x} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu(y-x+y_0)}dy_0 + \int_{y_0=x}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y}dy_0 \\ &= (\mu-\lambda)\rho\mu e^{-\mu(y-x)}\Big[\frac{e^{-(\mu\rho-\lambda\rho+\mu)y_0}}{-(\mu-\lambda)\rho-\mu}\Big]_{y_0=0}^{x} + (\mu-\lambda)\rho\mu e^{-\mu y}\Big[\frac{e^{-(\mu\rho-\lambda\rho)y_0}}{(\mu-\lambda)\rho}\Big]_{y_0=x}^{\infty} \\ &= (\mu-\lambda)\rho\mu e^{-\mu y}\Big\{\frac{e^{-(\mu-\lambda)\rho x}-e^{\mu x}}{-(\mu-\lambda)\rho-\mu} + \frac{e^{-(\mu-\lambda)\rho x}}{(\mu-\lambda)\rho}\Big\} \end{aligned} \tag{24}$$

(ii) $x \geq y$ の時

$$\begin{aligned} W^1(\boldsymbol{y}|\boldsymbol{x}) &= \int_{y_0=x-y}^{x} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu(y-x+y_0)}dy_0 + \int_{y_0=x}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y}dy_0 \\ &= (\mu-\lambda)\rho\mu e^{-\mu y}\Big\{\frac{e^{-(\mu-\lambda)\rho x}-e^{-(\mu-\lambda)\rho x}e^{((\mu-\lambda)\rho+\mu)y}}{-(\mu-\lambda)\rho-\mu} + \frac{e^{-(\mu-\lambda)\rho x}}{(\mu-\lambda)\rho}\Big\} \end{aligned} \tag{25}$$

となる. $x = 5$ に固定した時を図 14 に示す. (i) の場合の式 (24) と (ii) の式 (25) との交点は x となる.

次に $\frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})}$ を求める.

Burke の定理より, 通信路の出力過程はレート λ のポアソン過程となるのでパケットの到着間隔は指数分布

$$P_Y(y) = \lambda e^{-\lambda y}$$

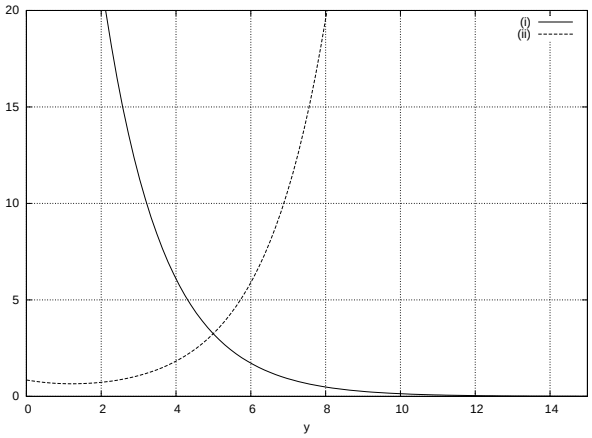

図 15　$\frac{W(y|x)}{P(y)}$ (x=5)

である. 次に $\frac{W^1(\boldsymbol{y}|\boldsymbol{x})}{P_Y(y)}$ は

(i) $x < y$ の時

$$\frac{W^1(\boldsymbol{y}|\boldsymbol{x})}{P_Y(y)} = (\mu-\lambda)e^{-(\mu-\lambda)y}\Big\{\frac{e^{-(\mu-\lambda)\rho x} - e^{\mu x}}{-(\mu-\lambda)\rho-\mu} + \frac{e^{-(\mu-\lambda)\rho x}}{(\mu-\lambda)\rho}\Big\} \tag{26}$$

(ii) $x \geq y$ の時

$$\frac{W^1(\boldsymbol{y}|\boldsymbol{x})}{P_Y(y)} = (\mu-\lambda)e^{-(\mu-\lambda)y}\Big\{\frac{e^{-(\mu-\lambda)\rho x} - e^{-(\mu-\lambda)\rho x}e^{((\mu-\lambda)\rho+\mu)y}}{-(\mu-\lambda)\rho-\mu} + \frac{e^{-(\mu-\lambda)\rho x}}{(\mu-\lambda)\rho}\Big\} \tag{27}$$

となる. これを図 15 に示す.

次に式 (9) の第一項を求める. $\Pr\Big\{\frac{W^1(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^1}(\boldsymbol{y})} \leq \alpha\Big\}$ は

$$\begin{aligned}
\Pr\Big\{\frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})} \leq \alpha\Big\} &= \int_x \int_{y|\frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})} \leq \alpha} P_{XY}(x,y)dydx \\
&= \int_x \int_{y|\frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})} \leq \alpha} W(y|x)P_X(x)dydx \\
&= \int_x P_X(x) \int_{y|\frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})} \leq \alpha} W(y|x)dydx
\end{aligned} \tag{28}$$

のように変形される. 式 (28) の $\int_{y|\frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})} \leq \alpha} W(y|x)dy$ をまず求める.

図 15 の (i) である式 (26) と (ii) である式 (27) の交点, つまり式 (26) に $y = x$ を代入した関数を $f(x)$ とおく.

$$f(x) = (\mu - \lambda)e^{-(\mu-\lambda)x}\left\{\frac{e^{-(\mu-\lambda)\rho x} - e^{\mu x}}{-(\mu-\lambda)\rho - \mu} + \frac{e^{-(\mu-\lambda)\rho x}}{(\mu-\lambda)\rho}\right\} \tag{29}$$

$f(x)$ と α で次のように場合分けされる.

$$\int_{y|\frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})}\leq\alpha} W(y|x)dy = \begin{cases} \int_{y|\frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})}\leq\alpha} W(y|x)dy & f(x) > \alpha \\ 1 & f(x)\leq\alpha \end{cases} \tag{30}$$

ここで, 式 (30) の y の積分範囲を決定する.

(i) $x < y$ の時

$$\frac{W^1(v_1|x)}{P_{Y^1}(y)} = \alpha \tag{31}$$

(ii) $x \geq y$ の時

$$\frac{W^1(v_1'|x)}{P_{Y^1}(y)} = \alpha \tag{32}$$

から x の関数 v_1,v_1' が得られる. これより条件を満たす積分範囲は $[0, v_1'], [v_1, \infty]$ となる. 式 (26),(31) より

$$\begin{aligned}
(\mu-\lambda)e^{-(\mu-\lambda)y}\left\{\frac{e^{-(\mu-\lambda)\rho x} - e^{\mu x}}{-(\mu-\lambda)\rho-\mu} + \frac{e^{-(\mu-\lambda)\rho x}}{(\mu-\lambda)\rho}\right\} &= \alpha \\
e^{-(\mu-\lambda)v_1} &= \frac{\alpha}{(\mu-\lambda)\left\{\frac{e^{-(\mu-\lambda)\rho x}-e^{\mu x}}{-(\mu-\lambda)\rho-\mu} + \frac{e^{-(\mu-\lambda)\rho x}}{(\mu-\lambda)\rho}\right\}} \\
-(\mu-\lambda)v_1 &= \log\frac{\alpha}{(\mu-\lambda)\left\{\frac{e^{-(\mu-\lambda)\rho x}-e^{\mu x}}{-(\mu-\lambda)\rho-\mu} + \frac{e^{-(\mu-\lambda)\rho x}}{(\mu-\lambda)\rho}\right\}} \\
v_1 &= \frac{-1}{\mu-\lambda}\log\frac{\alpha}{(\mu-\lambda)\left\{\frac{e^{-(\mu-\lambda)\rho x}-e^{\mu x}}{-(\mu-\lambda)\rho-\mu} + \frac{e^{-(\mu-\lambda)\rho x}}{(\mu-\lambda)\rho}\right\}}
\end{aligned} \tag{33}$$

v_1 が得られた. 次に式 (27),(32) より

$$\begin{aligned}
(\mu-\lambda)e^{-(\mu-\lambda)y}\left\{\frac{e^{-(\mu-\lambda)\rho x} - e^{-(\mu-\lambda)\rho x}e^{((\mu-\lambda)\rho+\mu)y}}{-(\mu-\lambda)\rho-\mu} + \frac{e^{-(\mu-\lambda)\rho x}}{(\mu-\lambda)\rho}\right\} &= \alpha \\
\frac{(\mu-\lambda)e^{-(\mu-\lambda)\rho x}}{-(\mu-\lambda)\rho-\mu}\left\{e^{-(\mu-\lambda)v_1'} - e^{((\mu-\lambda)\rho+\mu)v_1'} + \frac{-(\mu-\lambda)\rho-\mu}{(\mu-\lambda)\rho}e^{-(\mu-\lambda)v_1'}\right\} &= \alpha
\end{aligned} \tag{34}$$

となるがこの方程式は解析的に求められない. そこで, 式 (34) の左辺を

$$\frac{(\mu-\lambda)e^{-(\mu-\lambda)\rho x}}{-(\mu-\lambda)\rho-\mu}e^{-(\mu-\lambda)v_1'}-\frac{(\mu-\lambda)e^{-(\mu-\lambda)\rho x}}{-(\mu-\lambda)\rho-\mu}e^{((\mu-\lambda)\rho+\mu)v_1'}+\frac{e^{-(\mu-\lambda)\rho x}}{\rho}e^{-(\mu-\lambda)v_1'}=\alpha \tag{35}$$

のように 3 項に分ける. 式 (35) 左辺各項と式 (34) の左辺を比較すると, $y>0$ では第 2 項が強く影響していることが分かった. 式 (32) を書き直すと

$$\frac{(\mu-\lambda)e^{-(\mu-\lambda)\rho x}}{-(\mu-\lambda)\rho-\mu}e^{((\mu-\lambda)\rho+\mu)v_1''}=\alpha \tag{36}$$

となり

$$v_1''=\frac{1}{(\mu-\lambda)\rho+\mu}\log\frac{\alpha((\mu-\lambda)\rho+\mu)}{(\mu-\lambda)e^{-(\mu-\lambda)\rho x}} \tag{37}$$

が得られる. v_1'' を v_1' の近似とする.

$$v_1'\approx v_1''=\frac{1}{(\mu-\lambda)\rho+\mu}\log\frac{\alpha((\mu-\lambda)\rho+\mu)}{(\mu-\lambda)e^{-(\mu-\lambda)\rho x}} \tag{38}$$

以後 v_1' と扱う. よって y の積分範囲は (ii) の時 $[0,v_1']$, (i) の時 $[v_1,\infty]$ である.

これより式 (30) は

$$\int_{y|\frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})}\le\alpha}W(y|x)dy=\begin{cases}\int_0^{v_1'}W(y|x)dy+\int_{v_1}^{\infty}W(y|x)dy & f(x)>\alpha\\ 1 & f(x)\le\alpha\end{cases} \tag{39}$$

となる. ここで, 第一項の $W(y|x)$ は式 (25), 第二項は式 (24) である.

したがって式 (28) が

$$\Pr\Big\{\frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})}\le\alpha\Big\}=\int_x P_X(x)\Big\{\int_0^{v_1'}W(y|x)dy+\int_{v_1}^{\infty}W(y|x)dy\Big\}dx+\int_x P_X(x)1dx \tag{40}$$

のように書き直すことができる. x の積分範囲は $f(\hat{x})=\alpha$ から得られる $\hat{x}$ で分けられる.$f(x)$ は式 (29) である.

$$f(x)=(\mu-\lambda)e^{-(\mu-\lambda)x}\Big\{\frac{e^{-(\mu-\lambda)\rho x}-e^{\mu x}}{-(\mu-\lambda)\rho-\mu}+\frac{e^{-(\mu-\lambda)\rho x}}{(\mu-\lambda)\rho}\Big\}=\alpha \tag{41}$$

$$\tag{42}$$

$x>0$ より

$$\hat{x}\approx 3.9471 \tag{43}$$

と求まった. 式 (40) に戻すと

$$\Pr\Big\{\frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})}\leq\alpha\Big\}=\int_{\hat{x}}^{\infty}P_X(x)\Big\{\int_0^{v_1'}W(y|x)dy+\int_{v_1}^{\infty}W(y|x)dy\Big\}dx+\int_0^{\hat{x}}P_X(x)1dx$$
$$=\int_{\hat{x}}^{\infty}P_X(x)\int_0^{v_1'}W(y|x)dydx+\int_{\hat{x}}^{\infty}P_X(x)\int_{v_1}^{\infty}W(y|x)dydx$$
$$+\int_0^{\hat{x}}P_X(x)1dx \quad (44)$$

となる.

そして, 本研究の通信路のパラメータを代入し, 数値計算ソフト Maxima と WolframAlpha を用いて算出する.

誤り確率の上限 λ_n, 式 (9) は

$$\lambda_n\approx 1.8077 \quad (45)$$

となり, 正しく復号される確率の下限 $1-\lambda_n$ は

$$1-\lambda_n\approx -0.8077 \quad (46)$$

である.

次に条件式 (12),(13) を用いて符号の構成を行う. まず確率分布 $W(B(u_1)|u_1)$ を求める. 式 (11) より, $W(B(u_1)|u_1)$ は

$$W(B(u_1)|u_1)=\int_{v_1'}^{v_1}W(y|x)dy$$
$$=\int_{v_1'}^{x}W(y|x)dy+\int_{x}^{v_1}W(y|x)dy \quad (47)$$

のように書き換えられる. ここで, 第一項の $W(y|x)$ は式 (25), 第二項は式 (24) である. 代入すると

$$\int_{v_1'}^{x}W(y|x)dy=\int_{v_1'}^{x}(\mu-\lambda)\rho\mu e^{-\mu y}\Big\{\frac{e^{-(\mu-\lambda)\rho x}-e^{-(\mu-\lambda)\rho x}e^{((\mu-\lambda)\rho+\mu)y}}{-(\mu-\lambda)\rho-\mu}+\frac{e^{-(\mu-\lambda)\rho x}}{(\mu-\lambda)\rho}\Big\}dy$$
$$=\frac{\mu e^{-(\mu-\lambda)\rho x}}{(\mu-\lambda)\rho+\mu}\Big\{e^{(\mu-\lambda)\rho x}-e^{-\mu x}+e^{-\mu v_1'}-e^{\mu v_1'}\Big\} \quad (48)$$
$$\int_{x}^{v_1}W(y|x)dy=\int_{x}^{v_1}(\mu-\lambda)\rho\mu e^{-\mu y}\Big\{\frac{e^{-(\mu-\lambda)\rho x}-e^{\mu x}}{-(\mu-\lambda)\rho-\mu}+\frac{e^{-(\mu-\lambda)\rho x}}{(\mu-\lambda)\rho}\Big\}dy$$
$$=(\mu-\lambda)\rho\Big\{\frac{e^{-(\mu-\lambda)\rho x}-e^{\mu x}}{-(\mu-\lambda)\rho-\mu}+\frac{e^{-(\mu-\lambda)\rho x}}{(\mu-\lambda)\rho}\Big\}(e^{-\mu x}-e^{-\mu v_1}) \quad (49)$$

となる. 式 (47) を図 16 に示す. 条件 (12) と (46) より

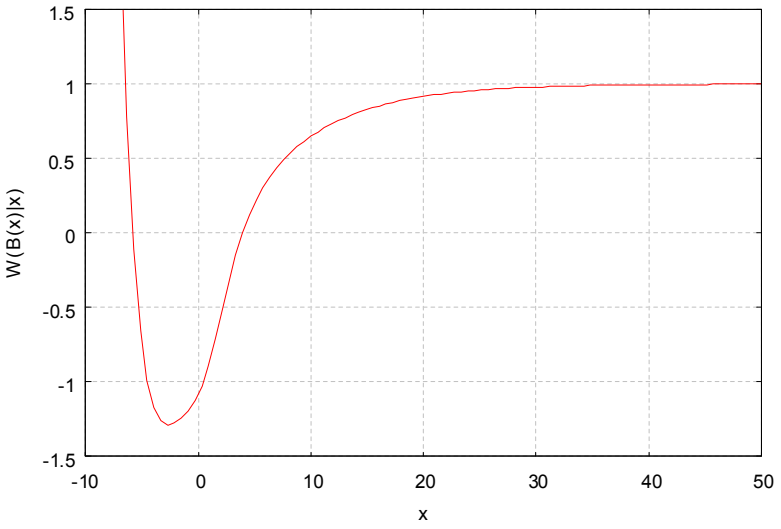

図 16　W(B(x)|x)

$$\begin{aligned} W^1(B(u_1)|u_1) &\geq 1-\lambda_n \\ &= -0.8077 \end{aligned}$$

正しく復号される確率の下限が負であるため, 式 (12) を満たす符号語 u_1 は任意の値となる.

従来研究に合わせて一方を $u_1 = 5.0$ とする. u_1 の復号領域 $B(u_1)$ は式 (33),(38) の $v_1'(u_1)$,$v_1(u_1)$ より領域が求まる.

$$\begin{aligned} B(u_1) &= [v_1'(u_1), v_1(u_1)] \\ &= [4.7027, 5.2966] \end{aligned}$$

となる.

次に符号語 u_2 を条件 (13) に従って選ぶ. $M_n = 2$ なので, u_2 の復号領域 $B(u_2) = [v_1'(u_2), v_1(u_2)]$ の $v_1'(u_2)$ は

$$v_1'(u_2) = v_1(u_1)$$

と, これを閾値とすることが最も効率がよい. すると式 (38) より

$$u_2 \approx 6.5335$$

が得られる. 復号領域は

$$\begin{aligned} B(u_1) &= [v_1'(u_2), v_1(u_2)] \\ &= [5.2966, 7.7105] \end{aligned}$$

であるが, $M_n = 2$ であるため特に上限を定める必要はない.

付録 C　$n = 2$ の時の符号導出過程

ここで $n = 2$ の時の符号構成を試みる. しかし, 通信路の条件付確率が 16 通りの場合分けがされることや, 逆関数の出現により計算が不可能になることから符号導出まで至ることが出来なかった. 途中までとなるが, 導出過程を示す.

4.3 章と同様に, $n = 2$ とした場合のパラメータを決定する. $n = 1$ の時はメッセージの数 M_n をあらかじめ 2 と設定したが, 本来は $M_n = e^{nR}$ と定義されている. メッセージの数は正整数となるので, 小数点以下は切り捨てとなることに注意する. 符号化レートは $R = \frac{1}{n}\log M_n \approx 0.9729$ となり, γ は $\gamma = \frac{R_0 - R}{3} \approx 0.009$, そして $\alpha = M_n e^{n\gamma} \approx 7.1271$ となる.

$n = 2$ の時のパラメータは以上である.

誤り確率の上限 (9) を算出したい. まず第一項を求める. $\Pr\left\{\frac{W^2(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^2}(\boldsymbol{y})} \leq \alpha\right\}$ は

$$\begin{aligned}
\Pr\left\{\frac{W^n(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^n}(\boldsymbol{y})} \leq \alpha\right\} &= \int_{x^2}\int_{y^2|\frac{W^2(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^2}(\boldsymbol{y})}\leq\alpha} P_{X^2Y^2}(x^2, y^2)dy^2dx^2 \\
&= \int_{x^2}\int_{y^2|\frac{W^2(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^2}(\boldsymbol{y})}\leq\alpha} W^2(y^2|x^2)P_{X^2}(x^2)dy^2dx^2 \\
&= \int_{x^2} P_{X^2}(x^2)\int_{y^2|\frac{W^2(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^2}(\boldsymbol{y})}\leq\alpha} W^2(y^2|x^2)dy^2dx^2 \qquad (50)
\end{aligned}$$

のように変形される.

式 (50) の $\int_{y^2|\frac{W^2(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^2}(\boldsymbol{y})}\leq\alpha} W^2(y^2|x^2)dy^2$ 部分を取り出し,

$$\int_{y^2|\frac{W^2(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^2}(\boldsymbol{y})}\leq\alpha} W^2(y^2|x^2)dy^2 = \int_{y_1|\frac{W^2(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^2}(\boldsymbol{y})}\leq\alpha}\int_{y_2|\frac{W^2(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^2}(\boldsymbol{y})}\leq\alpha} W^2(y^2|x^2)dy_2dy_1 \qquad (51)$$

と変形する. そして $\int_{y_2|\frac{W^2(\boldsymbol{y}|\boldsymbol{x})}{P_{Y^2}(\boldsymbol{y})}\leq\alpha} W^2(y^2|x^2)dy_2$ 部分から導出する. 式 (22) と同様に $n = 2$ を代入した W^n は

$$W^2(\boldsymbol{y}|\boldsymbol{x}) = \int_{y_0=0}^{\infty} e_{(\mu-\lambda)\rho}(y_0)e_\mu(y_1 - w_1)e_\mu(y_2 - w_2)dy_0 \qquad (52)$$

となる. この時指数部分が次の 2 つ

$$e_\mu(y_1 - w_1) = \begin{cases} e_\mu(y_1) & y_0 \geq x_1 \\ e_\mu(y_1 - (x_1 - w_1)) & y_0 < x_1 \end{cases}$$

$$e_\mu(y_1 - w_1) = \begin{cases} e_\mu(y_2) & y_0 \geq x_1 + x_2 - y_1 \\ e_\mu(y_2 - ((x_1 + x_2) - (y_0 + y_1))) & y_0 < x_1 + x_2 - y_1 \end{cases}$$

が存在する．y_0 は

$$y_0 > \max\{x_1 - y_1, x_1 + x_2 - y_1 - y_2\}$$

となる．

まず 2 つに場合分けされる．(I)$x_1 \leq x_1 + x_2 - y_1$, (Ⅱ)$x_1 > x_1 + x_2 - y_1$.

(I)(Ⅱ) をそれぞれ, (A)$x_1 - y_1 \leq x_1 + x_2 - y_1 - y_2$, (B)$x_1 - y_1 > x_1 + x_2 - y_1 - y_2$ に分ける．

次に,(I) の (A) を (a)$x_1 + x_2 - y_1 - y_2 < 0$, (b)$0 \leq x_1 + x_2 - y_1 - y_2 < x_1$, (c)$x_1 \leq x_1 + x_2 - y_1 - y_2 < x_1 + x_2 - y_1$, (d)$x_1 + x_2 - y_1 \leq x_1 + x_2 - y_1 - y_2$ の 4 つに．(I) の (B) を (e)$x_1 - y_1 < 0$, (f)$0 \leq x_1 - y_1 < x_1$, (g)$x_1 \leq x_1 - y_1 < x_1 + x_2 - y_1$, (h)$x_1 + x_2 - y_1 \leq x_1 - y_1$ の 4 つに分けられる．

同様に (Ⅱ) の (A),(B) を 4 つずつに分ける．よって 16 通りに場合分けされる．

(I)$x_1 \leq x_1 + x_2 - y_1$ (A)$x_1 - y_1 \leq x_1 + x_2 - y_1 - y_2$ (a)$x_1 + x_2 - y_1 - y_2 < 0$ の時

$$\begin{aligned} W^2(\boldsymbol{y^2}|\boldsymbol{x^2}) = &\int_{y_0=0}^{x_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu(y_1-x_1+y_0)}\mu e^{-\mu(y_2-x_1-x_2+y_1+y_0)}dy_0 \\ &+\int_{x_1}^{x_1+x_2-y_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu(y_2-x_1-x_2+y_1+y_0)}dy_0 \\ &+\int_{x_1+x_2-y_1}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu y_2}dy_0 \end{aligned}$$

(I)$x_1 \leq x_1 + x_2 - y_1$ (A)$x_1 - y_1 \leq x_1 + x_2 - y_1 - y_2$ (b)$0 \leq x_1 + x_2 - y_1 - y_2 < x_1$ の時

$$\begin{aligned} W^2(\boldsymbol{y^2}|\boldsymbol{x^2}) = &\int_{y_0=x_1+x_2-y_1-y_2}^{x_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu(y_1-x_1+y_0)}\mu e^{-\mu(y_2-x_1-x_2+y_1+y_0)}dy_0 \\ &+\int_{x_1}^{x_1+x_2-y_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu(y_2-x_1-x_2+y_1+y_0)}dy_0 \\ &+\int_{x_1+x_2-y_1}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu y_2}dy_0 \end{aligned}$$

(I)$x_1 \leq x_1 + x_2 - y_1$ (A)$x_1 - y_1 \leq x_1 + x_2 - y_1 - y_2$ (c)$x_1 \leq x_1 + x_2 - y_1 - y_2 < x_1 + x_2 - y_1$ の時

$$\begin{aligned} W^2(\boldsymbol{y^2}|\boldsymbol{x^2}) = &\int_{x_1+x_2-y_1-y_2}^{x_1+x_2-y_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu(y_2-x_1-x_2+y_1+y_0)}dy_0 \\ &+\int_{x_1+x_2-y_1}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu y_2}dy_0 \end{aligned}$$

(I)$x_1 \leq x_1 + x_2 - y_1$ (A)$x_1 - y_1 \leq x_1 + x_2 - y_1 - y_2$ (d)$x_1 + x_2 - y_1 \leq x_1 + x_2 - y_1 - y_2$ の時

$$W^2(\boldsymbol{y^2}|\boldsymbol{x^2}) = \int_{x_1+x_2-y_1-y_2}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu y_2} dy_0$$

ただし $y_2 \geq 0$ なので, 条件 (d) より $y_2 = 0$ である.

(I)$x_1 \leq x_1 + x_2 - y_1$ (B)$x_1 - y_1 > x_1 + x_2 - y_1 - y_2$ (e)$x_1 - y_1 < 0$ の時

$$\begin{aligned} W^2(\boldsymbol{y^2}|\boldsymbol{x^2}) = &\int_{y_0=0}^{x_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu(y_1-x_1+y_0)}\mu e^{-\mu(y_2-x_1-x_2+y_1+y_0)} dy_0 \\ &+ \int_{x_1}^{x_1+x_2-y_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu(y_2-x_1-x_2+y_1+y_0)} dy_0 \\ &+ \int_{x_1+x_2-y_1}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu y_2} dy_0 \end{aligned}$$

(I)$x_1 \leq x_1 + x_2 - y_1$ (B)$x_1 - y_1 > x_1 + x_2 - y_1 - y_2$ (f)$0 \leq x_1 - y_1 < x_1$ の時

$$\begin{aligned} W^2(\boldsymbol{y^2}|\boldsymbol{x^2}) = &\int_{y_0=x_1-y_1}^{x_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu(y_1-x_1+y_0)}\mu e^{-\mu(y_2-x_1-x_2+y_1+y_0)} dy_0 \\ &+ \int_{x_1}^{x_1+x_2-y_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu(y_2-x_1-x_2+y_1+y_0)} dy_0 \\ &+ \int_{x_1+x_2-y_1}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu y_2} dy_0 \end{aligned}$$

(I)$x_1 \leq x_1 + x_2 - y_1$ (B)$x_1 - y_1 > x_1 + x_2 - y_1 - y_2$ (g)$x_1 \leq x_1 - y_1 < x_1 + x_2 - y_1$ の時

$$\begin{aligned} W^2(\boldsymbol{y^2}|\boldsymbol{x^2}) = &\int_{x_1-y_1}^{x_1+x_2-y_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu(y_2-x_1-x_2+y_1+y_0)} dy_0 \\ &+ \int_{x_1+x_2-y_1}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu y_2} dy_0 \end{aligned}$$

(I)$x_1 \leq x_1 + x_2 - y_1$ (B)$x_1 - y_1 > x_1 + x_2 - y_1 - y_2$ (h)$x_1 + x_2 - y_1 \leq x_1 - y_1$ の時

$$W^2(\boldsymbol{y^2}|\boldsymbol{x^2}) = \int_{x_1-y_1}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu y_2} dy_0$$

ただし $x_2 \geq 0$ なので, 条件 (h) より $x_2 = 0$ である.

(Ⅱ)$x_1 > x_1 + x_2 - y_1$ (A)$x_1 - y_1 \leq x_1 + x_2 - y_1 - y_2$ (i)$x_1 + x_2 - y_1 - y_2 < 0$ の時

$$W^2(\boldsymbol{y^2}|\boldsymbol{x^2}) = \int_{y_0=0}^{x_1+x_2-y_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu(y_1-x_1+y_0)}\mu e^{-\mu(y_2-x_1-x_2+y_1+y_0)}dy_0 + \int_{x_1+x_2-y_1}^{x_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu(y_1-x_1+y_0)}\mu e^{-\mu y_2}dy_0 + \int_{x_1}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu y_2}dy_0$$

(Ⅱ)$x_1 > x_1 + x_2 - y_1$ (A)$x_1 - y_1 \leq x_1 + x_2 - y_1 - y_2$ (j)$0 \leq x_1 + x_2 - y_1 - y_2 < x_1 + x_2 - y_1$ の時

$$W^2(\boldsymbol{y^2}|\boldsymbol{x^2}) = \int_{y_0=x_1+x_2-y_1-y_2}^{x_1+x_2-y_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu(y_1-x_1+y_0)}\mu e^{-\mu(y_2-x_1-x_2+y_1+y_0)}dy_0 + \int_{x_1+x_2-y_1}^{x_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu(y_1-x_1+y_0)}\mu e^{-\mu y_2}dy_0 + \int_{x_1}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu y_2}dy_0$$

(Ⅱ)$x_1 > x_1 + x_2 - y_1$ (A)$x_1 - y_1 \leq x_1 + x_2 - y_1 - y_2$ (k)$x_1 + x_2 - y_1 \leq x_1 + x_2 - y_1 - y_2 < x_1$ の時

$$W^2(\boldsymbol{y^2}|\boldsymbol{x^2}) = \int_{x_1+x_2-y_1-y_2}^{x_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu(y_1-x_1+y_0)}\mu e^{-\mu y_2}dy_0 + \int_{x_1}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu y_2}dy_0$$

ただし $y_2 \geq 0$ なので, 条件 (k) より $y_2 = 0$ である.

(Ⅱ)$x_1 > x_1 + x_2 - y_1$ (A)$x_1 - y_1 \leq x_1 + x_2 - y_1 - y_2$ (l)$x_1 \leq x_1 + x_2 - y_1 - y_2$ の時

$$W^2(\boldsymbol{y^2}|\boldsymbol{x^2}) = \int_{x_1+x_2-y_1-y_2}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu y_2}dy_0$$

(Ⅱ)$x_1 > x_1 + x_2 - y_1$ (B)$x_1 - y_1 > x_1 + x_2 - y_1 - y_2$ (m)$x_1 - y_1 < 0$ の時

$$W^2(\boldsymbol{y^2}|\boldsymbol{x^2}) = \int_{y_0=0}^{x_1+x_2-y_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu(y_1-x_1+y_0)}\mu e^{-\mu(y_2-x_1-x_2+y_1+y_0)}dy_0 + \int_{x_1+x_2-y_1}^{x_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu(y_1-x_1+y_0)}\mu e^{-\mu y_2}dy_0 + \int_{x_1}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu y_2}dy_0$$

(Ⅱ)$x_1 > x_1 + x_2 - y_1$ (B)$x_1 - y_1 > x_1 + x_2 - y_1 - y_2$ (n)$0 \leq x_1 - y_1 < x_1 + x_2 - y_1$ の時

$$W^2(\boldsymbol{y^2}|\boldsymbol{x^2}) = \int_{y_0=x_1-y_1}^{x_1+x_2-y_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu(y_1-x_1+y_0)}\mu e^{-\mu(y_2-x_1-x_2+y_1+y_0)}dy_0 + \int_{x_1+x_2-y_1}^{x_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu(y_1-x_1+y_0)}\mu e^{-\mu y_2}dy_0 + \int_{x_1}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu y_2}dy_0$$

(Ⅱ)$x_1 > x_1 + x_2 - y_1$ (B)$x_1 - y_1 > x_1 + x_2 - y_1 - y_2$ (o)$x_1 + x_2 - y_1 \leq x_1 - y_1 < x_1$ の時

$$W^2(\boldsymbol{y^2}|\boldsymbol{x^2}) = \int_{x_1+x_2-y_1-y_2}^{x_1} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu(y_1-x_1+y_0)}\mu e^{-\mu y_2}dy_0 + \int_{x_1}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu y_2}dy_0$$

ただし $x_2 \geq 0$ なので, 条件 (o) より $x_2 = 0$ である.

(Ⅱ)$x_1 > x_1 + x_2 - y_1$ (B)$x_1 - y_1 > x_1 + x_2 - y_1 - y_2$ (p)$x_1 \leq x_1 - y_1$ の時

$$W^2(\boldsymbol{y^2}|\boldsymbol{x^2}) = \int_{x_1+x_2-y_1-y_2}^{\infty} (\mu-\lambda)\rho e^{-(\mu-\lambda)\rho y_0}\mu e^{-\mu y_1}\mu e^{-\mu y_2}dy_0$$

ただし $y_1 \geq 0$ なので, 条件 (p) より $y_1 = 0$ である.

諸計算は省く.

次に y_2 の積分範囲を導出したい. $n = 1$ の時の式 (31),(32) 同様

(I)$x_1 \leq x_1 + x_2 - y_1$ (A)$x_1 - y_1 \leq x_1 + x_2 - y_1 - y_2$ (a)$x_1 + x_2 - y_1 - y_2 < 0$ の時

$$\frac{W^2(\boldsymbol{y^2}|\boldsymbol{x^2})}{P(\boldsymbol{y^2})} = \alpha$$

を y_2 について解く. この時, $P(y^2) = \lambda e^{-\lambda y_1} \cdot \lambda e^{-\lambda y_2}$ である. 場合分け (b) から (p) までも同様に算出したい.

しかし, 逆関数の存在などが原因で解析的に求められない方程式がある. このため $n = 2$ 時の符号導出はここで行き詰まる. $n = 1$ 時のような荒い近似を行うという手段はあるが精度に関する議論が必要である. サービス器のサービス時間が指数分布であることが逆関数の出現を引き起こしたと考えられる. 今後は問題設定を見直すことも考慮する必要がある.

付録 D　ソースコード

D.1　$\{a, b\}$ と a の使用頻度 $p(a)$ を与えた時, 閾値を動かして見つける最小 BER と伝送速度を出力するプログラム

```
/***** 間隔 a, b, a の頻度 prob_a を与えた時、最小の BER を探すプログラム *****/

/*** 閾値をシンボル a,b の間からだけでなく 0 から 10 まで広く探すプログラム ***/

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

/******* 乱数を発生 *******/
#define MRND 1000000000L
static int jrand;
static long ia[56];

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();
}
```

```
/** 指数乱数 **/
double rand_exp(void)
{
    double rate = 1.0;                          /*** サービスレートμを変える ***/
    return -1.0 / rate * log(1.0 - rnd());
}

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

/** リングバッファ **/

#define BUFFER_CAPACITY 65536         /* 16384=2^14 ：2のべき乗だとmod計算のコンパイラが早い */

char buffer_list[BUFFER_CAPACITY];
int buffer_count = 0;
int buffer_in = 0;
int buffer_out = 0;

void put_buffer(char symbol)   /* リングバッファにsymbol入れる */
{
//printf("[%d]\n", __LINE__);
    if (buffer_count >= BUFFER_CAPACITY){
        puts("送信バッファあふれ");
        exit(0);
    }
    else{
        buffer_list[buffer_in] = symbol;
        buffer_in = (buffer_in + 1) % BUFFER_CAPACITY;  /* 余り%でリングさせる */
        buffer_count++;
    }
    return;
}

char get_buffer()     /* リングバッファからsymbol取り出す */
{
    int symbol;
//printf("[%d]\n", __LINE__);
    if(buffer_count == 0){
        puts("送信バッファ空");
        exit(0);
    }
    else{
        symbol = buffer_list[buffer_out];
        buffer_out = (buffer_out + 1) % BUFFER_CAPACITY;
        buffer_count--;
    }
    return(symbol);
}

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

#define MIN_OUT_INT 0               /** 出発間隔の最小 **/
#define MAX_OUT_INT 50.0            /** 出発間隔の最大 **/
#define PRECISION 0.01              /** 精度 **/
#define NUM_OF_SYMBOLS 1000000      /** シンボルの数100万 **/

/* コマンドラインから引数を渡す */
double symbol_a ;               /** argv[1]  シンボル(到着間隔)a **/
double symbol_b ;               /** argv[2]  シンボル(到着間隔)b **/    /*  a<b  */
double prob_a ;             /** argv[3]  シンボルaの出現頻度 **/

long arrived;                       /** 通信路に到着した合計 **/
long arrived_a, arrived_b;

long freq_a[(int)(MAX_OUT_INT / PRECISION)];        /* 配列の数 */
long freq_b[(int)(MAX_OUT_INT / PRECISION)];        /* 配列の数 */


int queue;
```

```
double t , arrival_time , departure_time;
double prev_departure;
double transmission_rate;


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

void arrival(void)
{
    arrived++;
    if (departure_time < 0.0){
        departure_time = t + rand_exp();
    } else {
        queue++;
    }
    if (arrived <= NUM_OF_SYMBOLS +1){                              /* 0 秒のパケットもカウントするので 100
万 +1 と比較 */
        char symbol = (rnd() < prob_a)? 'a': 'b';                   /* [0,1] 乱数が a の頻度より小さい (大き
い) 時, シンボル a(b) をエンコーダ */
//printf("[%d,%c]\n", __LINE__, symbol);
        arrival_time = t + ((symbol == 'a')? symbol_a: symbol_b);
        put_buffer(symbol);                                         /* symbol をリングバッファに入れる */
        if (symbol == 'a'){
            arrived_a++;                    /* a,b が何個送信されたかカウント */
        } else {
            arrived_b++;
        }
    }else {
        arrival_time = -1.0;
    }
    return;
}

void departure(void)
{
//printf("[%d]\n", __LINE__);
    if (prev_departure >= 0.0){
        char symbol = get_buffer();                                 /* リングバッファから取り出す */
        double interval = t - prev_departure;
        int index = (int)(interval / PRECISION);
        if (symbol == 'a'){
            freq_a[index]++;
        } else {
            freq_b[index]++;
        }
    }
    prev_departure = t;
    if(queue > 0){
        queue--;
        departure_time = t + rand_exp();
    }
    else{
        departure_time = -1.0;
    }
    return;
}

void
transmission(void)
{
    t = 0.0;
    queue = 0;
    arrival_time = 0.0;
    departure_time = -1.0;
    prev_departure = -1.0;
    arrived = 0;
    arrived_a = 0;
    arrived_b = 0;
```

```
    while(arrival_time >= 0.0 || departure_time >= 0.0){
        if(arrival_time < 0.0 ||
          (departure_time >= 0.0 && arrival_time > departure_time)){
            t = departure_time;
            departure();
        } else {
            t = arrival_time;
            arrival();
        }
    }
//printf("arrived_a = %d arrived_b = %d\n", arrived_a, arrived_b);
    return;
}

void
optimize(void)
{
    double log2 = log(2.0);
    double min_bit_error_rate = 1.0;
    double thre , arg_thre , bit_error_rate;
    int i , j;
    int arg_correct_a, arg_correct_b;

    for (i = 0; i <= MAX_OUT_INT / PRECISION; i++){      /** 0.01 刻みでチェックする閾値の回数 **/ /* 閾値と
なる点 */
        int correct_a = 0;
        int correct_b = 0;
        for(j = 0; j < i; j++){             /* 0 から閾値までの symbol_a → a へと正しく伝わった間隔のカウント */
            correct_a += freq_a[j];
        }
        for(j = i; j < MAX_OUT_INT / PRECISION ; j++){           /* 閾値から 30 までの symbol_b → b へと正し
く伝わった間隔のカウント */
            correct_b  += freq_b[j];
        }
//      correct_information = (correct_a * (-log(prob_a)) + correct_b * (-log(1.0 - prob_a))) / log2
/ (symbol_a * (NUM_OF_SYMBOLS * prob_a) + symbol_b * (NUM_OF_SYMBOLS * (1.0 - prob_a)));   /* 正しく伝わる
情報量の割合（ビット/秒）*/
        bit_error_rate = 1.0 - (correct_a * (-log(prob_a)) + correct_b * (-log(1.0 - prob_a))) / log2
/ (arrived_a * (-log(prob_a)) / log2 + arrived_b * (-log(1.0 - prob_a)) / log2);    /* bit error rate */

        thre = i * PRECISION;                       /* 閾値計算 */
//printf("%.2f %f\n" , thre , bit_error_rate);      /* 閾値 thre の時の正しく伝わる情報量を全部出力（グラフで形
を見るため）*/
        if(bit_error_rate < min_bit_error_rate){
            min_bit_error_rate = bit_error_rate;
            arg_thre = thre;
            arg_correct_a = correct_a;
            arg_correct_b = correct_b;
        }

    }
transmission_rate = (double)(arrived_a * (-log(prob_a)) / log(2.0) + arrived_b * (-log(1.0 - prob_a))
/ log(2.0)) / (symbol_a * (NUM_OF_SYMBOLS * prob_a) + symbol_b * (NUM_OF_SYMBOLS * (1.0 - prob_a)));
/*  伝送速度（ビット/秒）*/

    printf("#閾値 = %.2f 最小 bit_error_rate = %e 伝送速度 = %f\n", arg_thre , min_bit_error_rate
, transmission_rate);
//printf("#a の correct 数 = %d b の correct 数 = %d\n", arg_correct_a, arg_correct_b);
    return;
}

int
main(int argc, char **argv)
{
/* コマンドラインから引数を渡す */
    if (argc != 4) goto ERROR;
    if (argv[1] == NULL || argv[2] == NULL || argv[3] == NULL) goto ERROR;
    if ((symbol_a = atof(argv[1])) == 0.0) goto ERROR;
    if ((symbol_b = atof(argv[2])) == 0.0) goto ERROR;
```

```
    if ((prob_a = atof(argv[3])) == 0.0) goto ERROR;

    /* init */
    init_rnd(5);

    transmission();
    optimize();
//printf("buffer_count = %d\n", buffer_count);

/*  double small = MIN_OUT_INT;
    double large = MIN_OUT_INT + PRECISION;
    int h;

    for(h = 0; small <= MAX_OUT_INT - PRECISION; h++){              /* 出発間隔の分布を作成 */
        small = large;
        large += PRECISION;
    } */

    return(0);
ERROR:
    puts(" 引数を確認してください \n abp_info <symbol_a> <symbol_b> <prob_a>");
    exit(1);

}
```

D.2 $\{a, b\}$ を与えた時, 使用頻度と閾値を動かして見つける最小 BER と各伝送速度を出力するプログラム

```
/*** 間隔 a, b を与えた時、伝送速度と bit error rate の最小を探すプログラム ***/
/*** BER が最小となる prob_a も探す ***/
/*** 閾値をシンボル a,b の間からだけでなく 0 から 10 まで広く探すプログラム ***/

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

/******* 乱数を発生 *******/
#define MRND 1000000000L
static int jrand;
static long ia[56];

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();
}

/** 指数乱数 **/
double rand_exp(void)
{
    double rate = 1.0;                          /*** サービスレートμを変える ***/
    return -1.0 / rate * log(1.0 - rnd());
}

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

/** リングバッファ **/

#define BUFFER_CAPACITY 131072          /* 2^17 : 2 のべき乗だと mod 計算のコンパイラが早い */

char buffer_list[BUFFER_CAPACITY];
int buffer_count = 0;
int buffer_in = 0;
int buffer_out = 0;

void put_buffer(char symbol)   /* リングバッファに symbol 入れる */
{
//printf("[%d]\n", __LINE__);
    if (buffer_count >= BUFFER_CAPACITY){
        puts("送信バッファあふれ");
        exit(0);
    }
    else{
        buffer_list[buffer_in] = symbol;
        buffer_in = (buffer_in + 1) % BUFFER_CAPACITY;  /* 余り % でリングさせる */
        buffer_count++;
    }
//printf("バッファへ入力後の数 = %d\n", buffer_count);
    return;
}

char get_buffer()     /* リングバッファから symbol 取り出す */
{
    int symbol;
//printf("[%d]\n", __LINE__);
    if(buffer_count == 0){
        puts("送信バッファ空");
        exit(0);
    }
    else{
        symbol = buffer_list[buffer_out];
        buffer_out = (buffer_out + 1) % BUFFER_CAPACITY;
```

```
        buffer_count--;
    }
//printf(" バッファから出力後の数 = %d\n", buffer_count);
    return(symbol);
}

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

#define PRECISION2 0.01             /** prob_a(a の頻度) の精度 0~1 未満まで **/

#define MIN_OUT_INT 0               /** 出発間隔の最小 **/
#define MAX_OUT_INT 100.0           /** 出発間隔の最大 **/
#define PRECISION 0.01              /** 精度 **/
#define NUM_OF_SYMBOLS 700000       /** シンボルの数 100 万 **/

/* コマンドラインから引数を渡す */
double symbol_a ;               /** argv[1]  シンボル (到着間隔)a **/
double symbol_b ;               /** argv[2]  シンボル (到着間隔)b **/    /*  a<b  */

double prob_a ;                     /** シンボル a の出現頻度 **/
long arrived;                       /** 通信路に到着した合計 **/
long arrived_a, arrived_b;

long freq_a[(int)(MAX_OUT_INT / PRECISION)];        /* 配列の数 */
long freq_b[(int)(MAX_OUT_INT / PRECISION)];        /* 配列の数 */


int queue;
double t , arrival_time , departure_time;
double prev_departure;

double ab_min_bit_error_rate = 1.0;
double arg_prob_a;
double transmission_rate , arg_transmission_rate;


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

void arrival(void)
{
    arrived++;
    if (departure_time < 0.0){
        departure_time = t + rand_exp();
    } else {
        queue++;
    }
    if (arrived <= NUM_OF_SYMBOLS){                     /* 0 秒のパケットもカウントする */
        char symbol = (rnd() < prob_a)? 'a': 'b';       /* [0,1] 乱数が a の頻度より小さい (大きい) 時, シンボル
a(b) をエンコーダ */
//printf("[%d,%c]\n", __LINE__, symbol);
        arrival_time = t + ((symbol == 'a')? symbol_a: symbol_b);
        put_buffer(symbol);                             /* symbol をリングバッファに入れる */
        if (symbol == 'a'){
            arrived_a++;                                /* a,b が何個送信されたかカウント */
        } else {
            arrived_b++;
        }
    }else {
        arrival_time = -1.0;
    }
//printf("  キューへ入力後の数 = %d\n", queue);
    return;
}

void departure(void)
{
//printf("[%d]\n", __LINE__);
    if (prev_departure >= 0.0){
        char symbol = get_buffer();                    /* リングバッファから取り出す */
```

```
        double interval = t - prev_departure;
        int index = (int)(interval / PRECISION);
        if (symbol == 'a'){
            freq_a[index]++;
        } else {
            freq_b[index]++;
        }
    }
    prev_departure = t;
    if(queue > 0){
        queue--;
        departure_time = t + rand_exp();
    }
    else{
        departure_time = -1.0;
    }
//printf("   キューから出力後の数 = %d\n", queue);
    return;
}

void
transmission(void)
{
    int h;

    t = 0.0;
    queue = 0;
    arrival_time = 0.0;
    departure_time = -1.0;
    prev_departure = -1.0;
    arrived = 0;
    arrived_a = 0;
    arrived_b = 0;

    for(h = 0; h <= (int)(MAX_OUT_INT / PRECISION); h++){       /* 配列の中身を空にする */
        freq_a[h] = 0;
        freq_b[h] = 0;
    }
    while(arrival_time >= 0.0 || departure_time >= 0.0){
        if(arrival_time < 0.0 ||
          (departure_time >= 0.0 && arrival_time > departure_time)){
            t = departure_time;
            departure();
        } else {
            t = arrival_time;
            arrival();
        }
    }
//printf("arrived_a = %d arrived_b = %d\n", arrived_a, arrived_b);
    return;
}

void
optimize(void)
{
    double log2 = log(2.0);
    double min_bit_error_rate = 1.0;
    double thre , arg_thre , bit_error_rate;
    int i , j;
    int arg_correct_a, arg_correct_b;

    for (i = 0; i < (int)(MAX_OUT_INT / PRECISION); i++){      /** 0.01 刻みでチェックする閾値の回
数 **/ /* 閾値となる点 */
        int correct_a = 0;
        int correct_b = 0;
        for(j = 0; j < i; j++){             /* 0 から閾値までの symbol_a → a へと正しく伝わった間隔のカウント */
            correct_a += freq_a[j];
        }
        for(j = i; j < MAX_OUT_INT / PRECISION ; j++){           /* 閾値から MAX_OUT_INT までの symbol_b → b
```

```
へと正しく伝わった間隔のカウント */
            correct_b  += freq_b[j];
        }

        bit_error_rate = ((arrived_a - correct_a) * (-log(prob_a)) + (arrived_b - correct_b)
* (-log(1.0 - prob_a))) / log2 / (arrived_a * (-log(prob_a))
/ log2 + arrived_b * (-log(1.0 - prob_a)) / log2);            /* bit error rate */

        thre = i * PRECISION;                           /* 閾値計算 */
        //printf("%.2f %f\n" , thre , correct_information);         /* 閾値 thre の時の正しく伝わる情報量を全
部出力（グラフで形を見るため） */
        if(bit_error_rate < min_bit_error_rate){
            min_bit_error_rate = bit_error_rate;
            arg_thre = thre;
            arg_correct_a = correct_a;
            arg_correct_b = correct_b;

            transmission_rate = (arrived_a * (-log(prob_a)) / log2 + arrived_b * (-log(1.0 - prob_a))
/ log2) / (symbol_a * (NUM_OF_SYMBOLS * prob_a) + symbol_b * (NUM_OF_SYMBOLS * (1.0 - prob_a)));
/*  伝送速度（ビット/秒） */
        }
    }
    if(min_bit_error_rate < ab_min_bit_error_rate){             /* 各 prob_a の情報量を比較して組 (a,b) 最適の
prob_a(arg_prob_a) を探す */
        ab_min_bit_error_rate = min_bit_error_rate;
        arg_prob_a = prob_a;
        arg_transmission_rate = transmission_rate;
    }

    return;
}

int
main(int argc, char **argv)
{
    int k;

/* コマンドラインから引数を渡す */
    if (argc != 3) goto ERROR;
    if (argv[1] == NULL || argv[2] == NULL) goto ERROR;
    if ((symbol_a = atof(argv[1])) == 0.0) goto ERROR;
    if ((symbol_b = atof(argv[2])) == 0.0) goto ERROR;

    /* init */
    init_rnd(5);

//prob_a をループ
    for(k = 1; k <= 10; k++){
        prob_a = k * PRECISION2;        /* a の頻度計算 */

        transmission();
        optimize();
    }


//バッチファイルから入力してデータのみまとめて保存する時↓
    printf("%.2f %f %e\n", arg_prob_a , arg_transmission_rate , ab_min_bit_error_rate);

    return(0);
ERROR:
    puts(" 引数を確認してください \n bit_error_rate <symbol_a> <symbol_b>");
    exit(1);

}
```

D.3 $\{a, b\}$ を与えた時, (7,4)Hamming code を適用させた最小 BER と各伝送速度を出力するプログラム

```
/***    (7,4) ハミング符号    ***/

/*** 間隔 a, b を与えた時、伝送速度と bit error rate の最小を探すプログラム ***/
/*** BER が最小となる prob_a も探す ***/
/*** 閾値をシンボル a,b の間からだけでなく 0 から 10 まで広く探すプログラム ***/

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

/******* 乱数を発生 *******/
#define MRND 1000000000L
static int jrand;
static long ia[56];

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();
}
```

```
/** 指数乱数 **/
double rand_exp(void)
{
    double rate = 1.0;                          /*** サービスレートμを変える ***/
    return -1.0 / rate * log(1.0 - rnd());
}

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


#define PRECISION2 0.01             /** prob_a(a の頻度) の精度 0~1 未満まで **/

#define MIN_OUT_INT 0               /** 出発間隔の最小 **/
#define MAX_OUT_INT 30.0            /** 出発間隔の最大 **/
#define PRECISION 0.01              /** 精度 **/
#define NUM_OF_SYMBOLS 700000       /** シンボルの数 100 万 **/

/* コマンドラインから引数を渡す */
double symbol_a ;               /** argv[1]  シンボル (到着間隔)a **/
double symbol_b ;               /** argv[2]  シンボル (到着間隔)b **/    /*  a<b  */

double prob_a ;                     /** シンボル a の出現頻度 **/
long arrived;                       /** 通信路に到着した合計 **/
long arrived_a, arrived_b;
long arrived_0, arrived_1;
long bit_error_a, bit_error_b;
double thre, arg_thre, ab_arg_thre;

int queue;
double t , arrival_time , departure_time;
double prev_departure;

double bit_error_rate;
double min_bit_error_rate;
double ab_min_bit_error_rate = 1.0;
double arg_prob_a;
double transmission_rate , arg_transmission_rate;


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

/** リングバッファ **/

#define BUFFER_CAPACITY 131072         /* 2^17 : 2 のべき乗だと mod 計算のコンパイラが早い */

char buffer_list[BUFFER_CAPACITY];
int buffer_count = 0;
int buffer_in = 0;
int buffer_out = 0;

void put_buffer(char symbol)   /* リングバッファに symbol 入れる */
{
//printf("[%d]\n", __LINE__);
    if (buffer_count >= BUFFER_CAPACITY){
        puts("送信バッファあふれ");
        exit(0);
    }
    else{
        buffer_list[buffer_in] = symbol;
        buffer_in = (buffer_in + 1) % BUFFER_CAPACITY;  /* 余り % でリングさせる */
        buffer_count++;
    }
//printf("バッファへ入力後の数 = %d\n", buffer_count);
    return;
}

char get_buffer()     /* リングバッファから symbol 取り出す */
{
    int symbol;
```

```
//printf("[%d]\n", __LINE__);
    if(buffer_count == 0){
        puts("送信バッファ空");
        exit(0);
    }
    else{
        symbol = buffer_list[buffer_out];
        buffer_out = (buffer_out + 1) % BUFFER_CAPACITY;
        buffer_count--;
    }
//printf(" バッファから出力後の数 = %d\n", buffer_count);
    return(symbol);
}

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


int
encode(void)
{
    static int x[7];                /* static:最初読まれた時だけ初期化 */
    static int through = 7;

    if (through == 7){              /* 通過が 7 回目毎に情報ビット・符号語が生成される */
        int l;
        for(l = 0; l <= 3; l++){    /* 4bit ずつ */
            x[l] = (rnd() < prob_a)? 0: 1;
//情報点の送った数のカウント
            if(x[l] == 0){
                arrived_0++;
            }else{
                arrived_1++;
            }

        }
        x[4] = x[0]^x[2]^x[3];      /* 検査点（パリティ）を XOR で算出 */
        x[5] = x[0]^x[1]^x[3];
        x[6] = x[1]^x[2]^x[3];
        through = 0;
    }

    return(x[through++]);           /* 通過を一つ増やして一つずつ返す */
}


void arrival(void)
{
    arrived++;
    if (departure_time < 0.0){
        departure_time = t + rand_exp();
    } else {
        queue++;
    }
    if (arrived <= NUM_OF_SYMBOLS){         /* 終了？ */

//ここを通過する度に encode() に飛び、順序通り符号語を読み込む
        char symbol = (encode() == 0)? 'a': 'b';                /* 0,1 で表された符号語が 0,(1) の時, シンボル
a,(b) に変換 */

        arrival_time = t + ((symbol == 'a')? symbol_a: symbol_b);
        put_buffer(symbol);                 /* symbol をリングバッファに入れる */
        if (symbol == 'a'){
            arrived_a++;                    /* a,b が何個送信されたかカウント */
        } else {
            arrived_b++;
        }

    }else {
        arrival_time = -1.0;                /* 送信終了合図 */
```

```
    }
//printf("  キューへ入力後の数 = %d\n", queue);
    return;
}

void
decode(char decode_interval)
{
    static int y[7];
    static int z[3];
    static int bit[7];
    static int through2 = 0;
    int h;
    char symbol;

/* 7 つ揃うまで */
    y[through2] = (decode_interval == 'a')? 0: 1;       /* シンボル a,(b) を 0,1 に戻して配列へ */

    symbol = get_buffer();          /* リングバッファから取り出す */
    bit[through2] = (symbol == 'a')? 0: 1;              /* 符号語（誤りチェックのため）*/

    if (through2 == 6){         /* 7 つ揃ったら検出・訂正 */
/* シンドローム計算 */
        z[0] = y[0]^y[2]^y[3]^y[4];
        z[1] = y[0]^y[1]^y[3]^y[5];
        z[2] = y[1]^y[2]^y[3]^y[6];

/* エラー検出・訂正 */
        if(z[0] == 1 && z[1] == 1 && z[2] == 0){
            y[0] = 1 - y[0];    /* !y[] でも反転 */
        }
        if(z[0] == 0 && z[1] == 1 && z[2] == 1){
            y[1] = 1 - y[1];
        }
        if(z[0] == 1 && z[1] == 0 && z[2] == 1){
            y[2] = 1 - y[2];
        }
        if(z[0] == 1 && z[1] == 1 && z[2] == 1){
            y[3] = 1 - y[3];
        }
        if(z[0] == 1 && z[1] == 0 && z[2] == 0){
            y[4] = 1 - y[4];
        }
        if(z[0] == 0 && z[1] == 1 && z[2] == 0){
            y[5] = 1 - y[5];
        }
        if(z[0] == 0 && z[1] == 0 && z[2] == 1){
            y[6] = 1 - y[6];
        }

//シンボル単位での誤りチェック↓
        for(h = 0; h < 3; h++){     /* 情報ビット部分だけの誤りチェック */
            if(y[h] != bit[h]){
                if(bit[h] == 0){        /* 情報量違うので別個で誤りチェック */
                    bit_error_a++;
                }else{
                    bit_error_b++;
                }
            }
        }

//ブロック単位での誤りチェック↓     /* ブロック内の 1 つ（1bit）でも誤れば 1 ブロック（4bit）誤る */
//      if(y[0] != bit[0] || y[1] != bit[1] || y[2] != bit[2] || y[3] != bit[3]){
//          bit_error += 4;         /* 誤りが 4 ずつ増える */
//      }

        through2 = 0;   /* 通過数をリセット */
    }
    else{
```

```
        through2++;
    }
    return;
}


void departure(void)
{
//printf("[%d]\n", __LINE__);
    if (prev_departure >= 0.0){

        double interval = t - prev_departure;       /* 出発間隔 */

        char decode_interval = (interval <= thre)? 'a': 'b';        /* 1 間隔毎で閾値でシンボル a,(b) に復
号 */
//ここを通過する度に decode() に飛び、順序通り 7 個ずつ復号語を読み、誤り検出・訂正
        decode(decode_interval);
    }
    prev_departure = t;
    if(queue > 0){
        queue--;
        departure_time = t + rand_exp();
    }
    else{
        departure_time = -1.0;
    }
//printf("   キューから出力後の数 = %d\n", queue);
    return;
}

void
transmission(void)
{

    t = 0.0;
    queue = 0;
    arrival_time = 0.0;
    departure_time = -1.0;
    prev_departure = -1.0;
    arrived = 0;
    arrived_a = 0;
    arrived_b = 0;
    arrived_0 = 0;
    arrived_1 = 0;


    while(arrival_time >= 0.0 || departure_time >= 0.0){
        if(arrival_time < 0.0 ||
          (departure_time >= 0.0 && arrival_time > departure_time)){
            t = departure_time;
            departure();
        } else {
            t = arrival_time;
            arrival();
        }
    }
//printf("arrived_a = %d arrived_b = %d\n", arrived_a, arrived_b);
    return;
}

void
optimize(void)
{
    bit_error_rate = (bit_error_a * (-log(prob_a)) + bit_error_b* (-log(1.0 - prob_a))) / log(2.0)
/ (arrived_0 * (-log(prob_a)) / log(2.0) + arrived_1 * (-log(1.0 - prob_a)) / log(2.0));

    if(bit_error_rate < min_bit_error_rate){
        min_bit_error_rate = bit_error_rate;
        arg_thre = thre;
```

```
    }
    transmission_rate = (arrived_0 * (-log(prob_a)) + arrived_1 * (-log(1.0 - prob_a))) / log(2.0)
/ (symbol_a * (4 * NUM_OF_SYMBOLS / 7 * prob_a) + symbol_b * (4 * NUM_OF_SYMBOLS / 7 * (1.0 - prob_a)));
         /*  伝送速度（ビット/秒） */

//  printf("a の頻度:%.2f 閾値:%.2f 伝送速度:%f bit_error_rate:%e\n", prob_a , thre
, transmission_rate , bit_error_rate);

//全てをプロットするため
    printf("%.2f %.2f %f %e\n", prob_a , thre , transmission_rate , bit_error_rate);


//printf("#a の correct 数 = %d b の correct 数 = %d\n", arg_correct_a, arg_correct_b);
    return;
}

int
main(int argc, char **argv)
{
    int k, i;

/* コマンドラインから引数を渡す */
    if (argc != 3) goto ERROR;
    if (argv[1] == NULL || argv[2] == NULL) goto ERROR;
    if ((symbol_a = atof(argv[1])) == 0.0) goto ERROR;
    if ((symbol_b = atof(argv[2])) == 0.0) goto ERROR;

    /* init */
    init_rnd(5);


//prob_a をループ
    for(k = 1; k <= 1; k++){            /* prob_a(a の頻度) の精度 0〜0.99 未満まで 0.01 毎（なので本当は 99 回）
  かなり時間かかるので一番 BER 小さくなるところ (prob_a = 0.01) を */
        prob_a = k * PRECISION2;        /* a の頻度計算 */

        min_bit_error_rate = 1.0;
//さらにどの閾値で BER が最小となるか探す
        for (i = 1; i < (int)(MAX_OUT_INT / PRECISION); i++){      /** 0.01 刻みでチェックする閾値の回
数 **/ /* 閾値となる点 */
            thre = i * PRECISION;       /* 実際の閾値 */
            bit_error_a = 0;            /* リセット？ */
            bit_error_b = 0;

            transmission();
            optimize();

        }

        if(min_bit_error_rate < ab_min_bit_error_rate){             /* 各 prob_a の情報量を比較して組 (a,b)
最適の prob_a(arg_prob_a) を探す */
            ab_min_bit_error_rate = min_bit_error_rate;
            arg_prob_a = prob_a;
            ab_arg_thre = arg_thre;
            arg_transmission_rate = transmission_rate;
        }

    }

    return(0);
ERROR:
    puts(" 引数を確認してください \n bit_error_rate <symbol_a> <symbol_b>");
    exit(1);

}
```