信州大学
大学院工学系研究科

修士論文

ポアソン到着シンボルに対する伝送システムとその遅延特性

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　08TA252D
氏名　森田 圭一

2010 年 1 月 29 日

目次

1 序論

1.1 研究の背景

インターネットに代表されるようなコンピュータネットワークの end-to-end の遅延特性を知ることはアプリケーションの QoS(Quality of Service) を向上させるためには非常に重要である. 本研究では, ポアソン到着するシンボルがルータに代表されるような伝送システムに到着し, サービスを受けてから通信路を通り復号器で復号されるまでの end-to-end 遅延の特性を定量的に数式でモデル化する.

ポアソン到着するシンボルというのは主にネットワーク理論で扱われ, シンボルが到着してからシステムを出るまでの遅延や待ち時間, サービス時間を直接的に計算することができる. しかし, シンボルが受けるノイズや干渉の問題というのは無視される. 一方で, 情報理論ではこれらの問題に対処できるが end-to-end 遅延の分析を行うことはできない. そこで, これら二つの理論を組み合わせることで, コンピュータネットワークの end-to-end 遅延特性を正確にモデル化する.

本研究では特に, ポアソン到着したシンボルが符号器とバッファ, 送信機で構成される伝送システムを介し通信路を経て復号器で復号されるまでのシンボルの遅延特性を解析する. さらに,2006 年に Stephane Musy and Emre Telatar が文献 [1] で示したランダムな間隔でシンボルを発生する情報源に対する最適な送信方法をさらに検証し導き出した, 送信機からのフィードバックの有効性に関しても論ずる.

1.2 研究の目的

本論文ではポアソン到着するシンボルに対し, 符号器がシャノン符号, 算術符号を行うような伝送システムを考え, そこで処理されるシンボルの遅延特性を解析する. また, 送信機から符号器へ行われるフィードバックの有効性についても考察する.

1.3 本論文の構成

本論文は次のような構成をとる. 第 1 章では本研究で用いたシステムのフレームワークを紹介し,Musy and Telatar[1] が示した最適な送信方法を説明する. 第 2 章では符号器を含まない伝送モデルの遅延特性及び送信シンボル数の特性について解析する. 第 3 章では符号器がシャノン符号化を行うような伝送モデルの遅延特性を理論的に示す. 第 4 章では符号器が算術符号化を行う伝送モデルの遅延特性を示し, またフィードバックの有効性も考察する. 第 5 章では全体のまとめとして, フィードバックを行ったもとでシャノン符号と算術符号の遅延を比較す

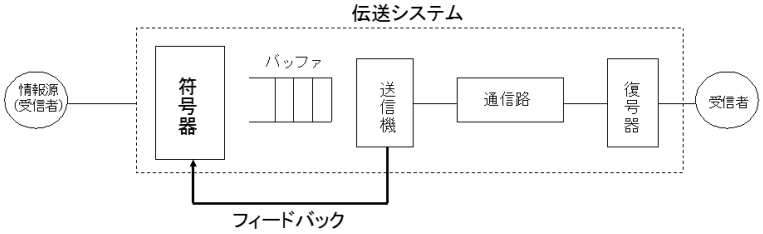

図 1　システムモデル

る. 第 6 章で結論を述べる.

2　システムのフレームワーク

この章では本研究で用いたシステムモデルのフレームワークを紹介する. コンピュータネットワークをモデル化するにあたって, 送信者と受信者は 1 人ずつで伝送システムも 1 つであると仮定する. 送信者からの情報がランダムな間隔で伝送システムに到着することを表現するために, 本研究ではこの到着モデルをポアソン到着で表した. 伝送システムは符号器, バッファ, 送信機で構成されるものと仮定し, 伝送システムを出た情報は通信路を通って受信者へと送られる. 伝送システム内のバッファに溜められたシンボルを送信機へ送る瞬間を送信フェーズと呼び, 送信フェーズのタイミングは Musy and Telatar が示した最適な送信方法を用いる [1]. また, 通信路内ではノイズは発生しないものと仮定した. 上記のようなシステムモデルを図 1 に示す.

2.1　Musy 氏と Telatar 氏の最適な送信方法

Musy and Telatar[1] が示したランダムに到着するシンボルに対するバッファや送信機の最適な送信方法を紹介する. 本研究における最適な送信方法とは, シンボルが伝送システムに入ってから出るまでの遅延が最も小さくなる方法のことを指す. まず, n 個のシンボルを送信機が送信するのにかかる時間 $T(n)$ を

$$T(n) = D + nk \qquad [\text{sec}] \tag{1}$$

とする.

この式において $k, D > 0$ である. $1/k$[packet/sec] は通信路の通信路容量である. D は固定遅延とよばれ, 送信機がアイドル状態からシンボルを送信するまでのウォーミングアップ時間のことである. 通信路容量が $1/k$ であることから, バッファ溢れを起こさないために到着レート λ は $\lambda < \frac{1}{k}$ を満たす範囲を考える. ここで, 到着レートは指数分布のレートをさす. 一般的

に送信時間 $T(n)$ は

$$T(n+m) \leq T(n) + T(m) \tag{2}$$

の関係が成り立つ. つまり, なるべく多くのシンボルをまとめて送信するほうが効率はいいということである. 不規則に到着するシンボルに対して Musy and Telatar は, 送信機が現在行っている送信が終わり次第すぐにバッファ内にためられているシンボルを送信機に送ることが最適な送信方法であるということを示した.

また,Musy and Telatar はこの方法におけるシンボルの平均遅延の下界が

$$\overline{D} \geq \frac{3}{2} \frac{D}{(1-\lambda k)} \tag{3}$$

で表されることを示した [1].

2.2 フィードバック

Musy and Telatar[1] の最適な送信方法において, 送信機はバッファに対して送信機が送信可能時にバッファ内のシンボルを要求するという情報を送っていると考えられる. この情報によって送信フェーズが送信機の状態に依存することがわかる. 本研究ではこの送信機からバッファへの情報というのをフィードバックと呼ぶことにする.

3 符号器を含まない伝送モデル

本章では, 符号器を含まずバッファと送信機, 通信路と復号器で構成される伝送モデルについて考える. つまり伝送システム内に到着したシンボルはまずバッファに溜められ, 最適な送信方法に従って送信機へ送られるというモデルである.

3.1 実験原理

3.1.1 ポアソン分布

一般的に $\lambda > 0$ として, 非負整数上の分布

$$\Pr(X = x) = e^{-\lambda} \frac{\lambda^x}{x!} \tag{4}$$

をパラメータ λ のポアソン分布という [2].

3.1.2 ポアソン到着

不規則に到着するシンボルのモデルとして, 本研究ではポアソン到着に従って到着するシンボルを考える. ここで, ポアソン到着とはポアソン過程に従う到着のことである. 各々が自然数

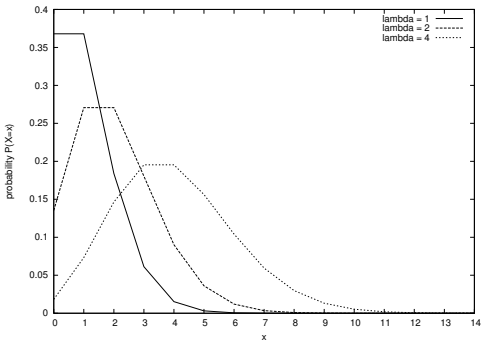

図 2　ポアソン分布

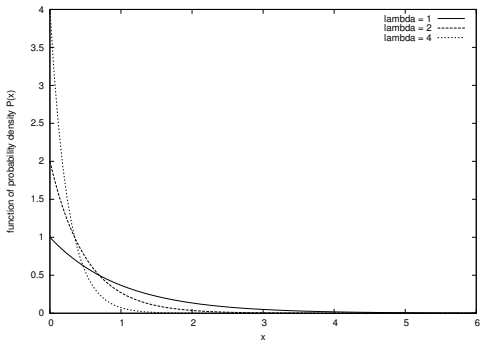

図 3　指数分布

をとる確率変数 $\{A(t)|t \geq 0\}$ は, 以下の条件を満たすとき, パラメータ λ のポアソン過程と呼ばれる.

1. $A(t)$ は $[0,t]$ の時間に到着したシンボルの合計を示すもので, $s < t$ のとき,$A(t) - A(s)$ は時間 $(s,t]$ で到着したシンボル数を示す. また,$A(0) = 0$ である.
2. 与えられた区間で起こる到着数は, それと重複しない他の区間で起こる到着数に対して独立である.
3. 任意の時刻 t から, 長さ τ の区間における到着数の確率は

$$\Pr\{A(t+\tau) - A(t) = n\} = e^{\lambda\tau}\frac{(\lambda\tau)^n}{n!}, \quad n = 0, 1, 2... \tag{5}$$

   で表され, これはパラメータ $\lambda\tau$ のポアソン分布である.

図 2 はパラメータ P を変化させたときのポアソン分布をグラフに表した図である.

3.1.3　指数分布

一般的に $\lambda > 0$ として, 非負実数上の密度分布

$$\Pr(X = x) = \lambda e^{-\lambda x} \tag{6}$$

をパラメータ λ の指数分布という [2].

また, ポアソン到着するシンボルの到着間隔は指数分布に従うといえる.

図 3 はパラメータを変化させたときの指数分布を表した図である.

3.1.4　リトルの定理

リトルの定理とは, 定常状態におけるシステム内のシンボルの個数とシステム内にいる時間との関係を示したものである. ここでは, その二つの値の意味をより明確にし, そこからリトル

の定理を導く.

まず時間が $t \in [0, \infty)$ におけるシステムを考え, 時間に関するさまざまな量を定義する.

$N(t)$: 時刻 t においてシステム内にいるシンボルの個数 [packet]

$\alpha(t)$: $[0, t]$ でシステムに到着したシンボルの個数　[packet]

$\beta(t)$: $[0, t]$ でシステムから送信されたシンボルの個数　[packet]

T_i : i 番目に到着した客がシステム内にいる時間　[sec]

まず, 時間 $[0, t]$ の平均到着率は

$$\lambda_t = \frac{\alpha(t)}{t} \tag{7}$$

と表される. λ_t を定常状態で表すと

$$\lambda = \lim_{t \to \infty} \lambda_t \tag{8}$$

となる. また, 時刻 t までのシンボルがシステム内にいる時間の平均は

$$T_t = \frac{\sum_{i=0}^{\alpha(t)} T_i}{\alpha(t)} \tag{9}$$

で表される. T_t を定常状態で表すと

$$T = \lim_{t \to \infty} T_t \tag{10}$$

となる.

次に, 時刻 t においてシステム内のシンボルが $N(t) = 0$ とするとき, 各シンボルがシステム内に滞在する時間の総数は, 微小時間を τ として

$$\int_0^t N(\tau) d\tau = \sum_{i=1}^{\alpha(t)} T_i \tag{11}$$

で表される. (11) 式から平均滞在時間は

$$\frac{1}{t} \int_0^t N(\tau) d\tau = \frac{1}{t} \sum_{i=1}^{\alpha(t)} T_i = \frac{\alpha(t)}{t} \frac{\sum_{i=1}^{\alpha(t)} T_i}{\alpha(t)} \tag{12}$$

と表される. ここで, 左辺を時刻 t までのシステム内の平均シンボル数 N_t とすると

$$N_t = \lambda_t T_t \tag{13}$$

が得られる.

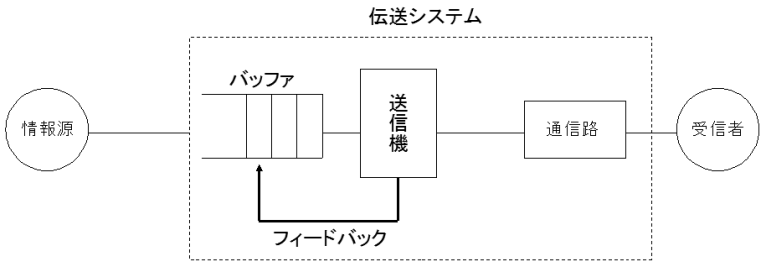

図 4　システムモデル (符号器なし)

次に,$N(t) > 0$ ならば

$$\sum_{i=1}^{\beta(t)} T_i \leq \int_0^t N(t)dt \leq \sum_{i=1}^{\alpha(t)} T_i \tag{14}$$

が成り立ち, 上式を変形して

$$\begin{aligned} \frac{\beta(t)}{t} \sum_{i=1}^{\beta(t)} \frac{T_i}{\beta(t)} \leq N_t \leq \frac{\alpha(t)}{t} \sum_{i=1}^{\alpha(t)} \frac{T_i}{\alpha(t)} \\ \frac{\beta(t)}{t} \sum_{i=1}^{\beta(t)} \frac{T_i}{\beta(t)} \leq N_t \leq \lambda_t T_t \end{aligned} \tag{15}$$

を得る.

ここで定常状態のとき,$\lim_{t\to\infty} \frac{\beta(t)}{t} = \lambda$ と $\lim_{t\to\infty} \frac{\sum_{i=1}^{\beta(t)} T_i}{\beta(t)} = T$ と仮定すると, はさみうちの原理より

$$\lim_{t\to\infty} N_t = N = \lambda T \tag{16}$$

を得る.

(16) 式を「リトルの定理」と呼ぶ.

3.2　システムモデル

符号器を含まない伝送システムのシステムモデルを図 4 に示す. レート λ でポアソン到着したシンボルはまずバッファに溜められる. Musy and Telatar[1] が提案した最適な送信方法に従い, 送信機が現在行っている送信が全て完了したら, フィードバックを行いバッファ内に溜まったシンボルをまとめて送信機へ送る [1]. 送信機の送信時間はバッファ内のシンボルの個数を n 個とすると,$T(n) = D + nk$ とする.

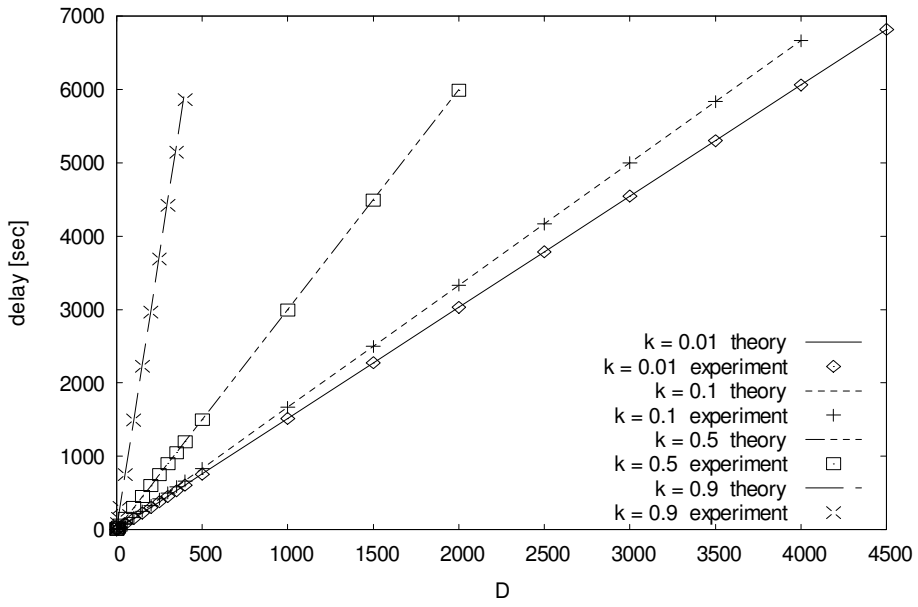

図 5　符号器なしの遅延特性 ($\lambda = 1$)

3.3　遅延特性

Musy and Telatar は図 4 のようなモデルにおけるシンボルの遅延の期待値の下界を式 (3) で示した [1]. そこで実際にシミュレーションを行いシンボルの遅延の期待値とその下界を比較する.

理論式とシミュレーション結果を比較した結果を図 5 に示す.

図 5 は到着レートを $\lambda = 1$ に設定し横軸を D としパラメータを k としたときの平均遅延を表している. シミュレーション結果から平均遅延の値は, 理論的に示された平均遅延の下界を示す直線上にほぼ乗った. よって理論的に式 (3) は平均遅延の下界であるが, シミュレーション結果から, 平均遅延そのものを表すのではないかと推測できる.

3.4　送信シンボル数の期待値 (符号器なし)

送信シンボル数を求めることによってバッファの大きさを設計することができる. そこで理論的に送信シンボル数の期待値を求める.

まず, 送信機の i 番目の送信について考える. バッファ内にシンボルが n_i 個あったとすると, i 番目の送信時間中にバッファ内に溜まるシンボル数の期待値 $\overline{N}_{i+1}$ は $\overline{N}_{i+1} = \lambda T(n_i)$ で表

される. ここで,$n_{i+1} > n_i$ となる必要十分条件は

$$\overline{N}_{i+1} = \lambda(D + n_i k) > n_i$$

ゆえに

$$n_i < \frac{\lambda D}{1 - \lambda k} \tag{17}$$

で与えられる. このとき $i+1$ 番目に送信するシンボル数の期待値 $\overline{N}_{i+1}$ を変形すると

$$\begin{aligned} \overline{N}_{i+1} &= \lambda T(n_i) \\ &< \lambda\left(D + \frac{\lambda D}{1 - \lambda k} k\right) \\ &= \frac{\lambda D}{1 - \lambda k} \end{aligned} \tag{18}$$

となる. 従って

$$n_i < \overline{N}_{i+1} < \frac{\lambda D}{1 - \lambda k} \tag{19}$$

の関係が成り立つ. つまり i 番目の送信シンボル数の個数が式 (17) を満たすならば,$i+1$ 番目の送信シンボル数の期待値は式 (19) を満たす.

次に,$\overline{N}_i$ と $\overline{N}_{i+1}$ の関係を考える.$\overline{N}_{i+1}$ は

$$\begin{aligned} \overline{N}_{i+1} &= E[n_{i+1}] \\ &= \sum_n E[n_{i+1} | n_i = n] \Pr\{n_i = n\} \\ &= \sum_n \lambda T(n) \Pr\{n_i = n\} \\ &= \sum_n \lambda(D + nk) \Pr\{n_i = n\} \\ &= \lambda(D + k\overline{N}_i) \\ &= \lambda T(\overline{N}_i) \end{aligned} \tag{20}$$

の関係が得られる. この式から現在行っている送信シンボル数の期待値がわかっていれば次のシンボル数の期待値も判明するということがわかる.

$\overline{N}_{i+1} > \overline{N}_i$ となる必要十分条件は式 (17) と同様に

$$\overline{N}_i < \frac{\lambda D}{1 - \lambda k} \tag{21}$$

で与えられ, このとき

$$\overline{N}_{i+1} < \frac{\lambda D}{1 - \lambda k} \tag{22}$$

となる. 従って, 常に $n_1 = 1$ が成り立つという事実から, 定常状態におけるバッファ内のシンボル数は

$$\lim_{i\to\infty} \overline{N}_i = \frac{\lambda D}{1-\lambda k} \tag{23}$$

となり送信シンボル数の期待値が求まった.

3.5 送信シンボル数の分布

次に送信シンボル数の分布を示す式を求める.

まず最初にシステムに到着したシンボルはバッファを通り越して直接送信機へ送られるので, 送信シンボル数 N の確率分布 P_N は

$$P_{N_1}(1) = 1 \tag{24}$$

となる.2 番目の送信フェーズ時の送信シンボル数はパラメータ $\lambda T(1)$ のポアソン分布に従うことから

$$\Pr\{N_2 = n | N_1 = 1\} = P_{N_2}(n) = \frac{(\lambda T(1))^n}{n!} e^{-\lambda T(1)} \tag{25}$$

となる.

これを一般化し i 番目の送信フェーズ時の送信シンボル数の分布は

$$\Pr\{N_i = n\} = \sum_{n_{i-1}} \Pr\{N_i = n | N_{i-1} = n_{i-1}\} P_{N_{i-1}}(n_{i-1}) \tag{26}$$

となる. 定常状態 $(i \to \infty)$ のとき　$i \simeq i+1$ より

$$\begin{aligned} P_N(n) &= \sum_{\tilde{n}} \Pr\{n|\tilde{n}\} P_N(\tilde{n}) \\ &= \sum_{\tilde{n}} \frac{(\lambda T(\tilde{n})^n)}{n!} e^{-\lambda T(\tilde{n})} P_N(\tilde{n}) \end{aligned} \tag{27}$$

と表される. この式を P_N に関して解くことができれば送信シンボル数の分布を求めることができるが, 現段階では解くことができていない.

図 6 はシミュレーションを行い送信シンボル数の分布を表したものである. 図 6 内の実線は先ほど求めた期待値の理論式と, シミュレーションによって得られたシンボル数から実際に標準偏差を数値計算によって求め, それらを用いて正規分布を描いたものである. 点線はシミュレーション結果である. 正規分布と送信シンボル数の分布がほぼ一致したことから, 送信シンボル数の分布 $P_N(n)$ は正規分布を描くと予想される.

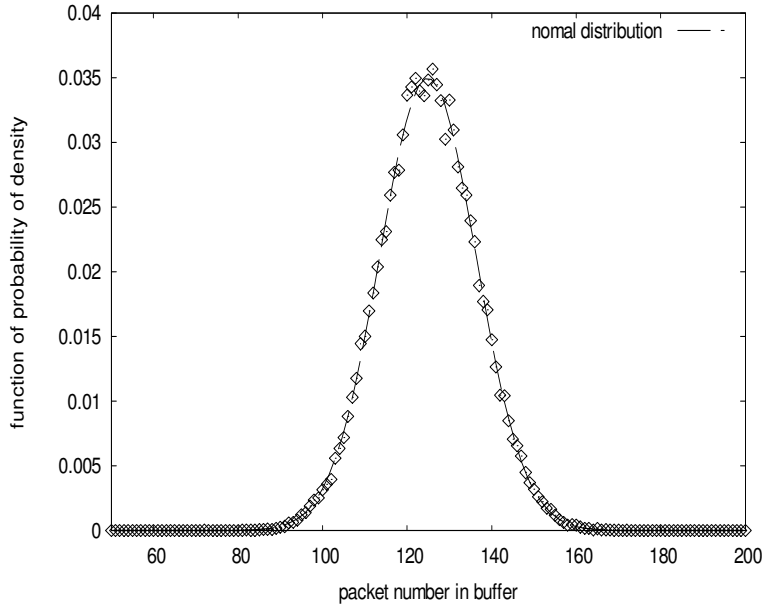

図 6　送信シンボル数の分布 (k=0.2 , D=100)

4　シャノン符号を含む伝送モデル

本章では, 伝送システムに符号器を加えたモデルを扱う. 符号器はブロック符号 (シャノン符号) を行うモデルを考え, そのときの遅延特性や送信シンボル数の特性を示す. 本論文でいうシャノン符号とは通常シャノン-ファノ符号として知られているものである. 符号器にシャノン符号を用いる理由は, 符号語長を表すのが容易であり, しかも符号化レートがエントロピーに漸近的に近づくためである.

4.1　実験原理

4.1.1　シャノン符号

$\mathcal{X} = \{0, 1, ..., J-1\}$ と考え, 必要であればソートを行い

$$Q(0) \geq Q(1) \geq ... \geq Q(J-1) > 0$$

と仮定する. そして累積確率を,$a_0 = 0, i = 1, ..., J-1$ に対して

$$a_i = \sum_{k=0}^{i-1} Q(k)$$

対象となる数	出力
1	1
2	01
3	001
4	0001
5	00001
6	000001

図 7　単進符号の例

と定め, 符号語長を $l_i = \lceil \log_K(1/Q(i)) \rceil$ とする. そして,a_i を K 進数表示してその小数点以下第 1 桁から第 l_i 桁までを符号語とする. ここで, 明らかに $l_0 \leq l_1 \leq ... \leq l_{J-1}$ である. 一般に a を小数点以下 l 桁までで打ち切った数を $\lfloor a \rfloor_l$ と表すことにすると, 各符号語は半開区間 $I_i \equiv [\lfloor a_i \rfloor_{l_i}, \lfloor a_i \rfloor_{l_i} + K^{-l_i})$ に対応し, その中に a_i が含まれる. 一方,$a_{i+1} - a_i = Q(i) \geq K^{-l_i}$ であることを考慮すると,$\lfloor a_{i+1} \rfloor_{l_i} \geq \lfloor a_i \rfloor_{l_i} + K^{-l_i}$. ゆえに,$l_{i+1} \geq l_i$ により $\lfloor a_{i+1} \rfloor_{l_{i+1}} \geq \lfloor a_i \rfloor_{l_i} + K^{-l_i}$ となり, よって区間 I_i と区間 I_{i+1} は重なり合わず, 任意の $x \in I_i, y \in I_{i+1}$ に対して $x < y$ である. すなわち, すべての半開区間 $I_i (i = 0, 1, ..., J-1)$ は互いに排反である. したがって, シャノン-ファノ符号は語頭符号となっている.[2]

バイナリー情報源からのシンボルは伝送システムに到着し, まずバッファに溜められる. 溜められたシンボルは送信フェーズごとにまとめて符号器へ送られるのだが, 符号器への入力シンボルは送信フェーズ時にバッファに溜められているシンボル系列となる. 従って符号器への入力長は毎回異なる. 本研究ではシンボルの遅延の大きさにのみ着目するので, 入力シンボルに対して符号器がどのように符号語を構成するとかということは考えず, 符号語の長さのみに着目する. よって符号器は符号語の長さ $l_i = \lceil \log_K(1/Q(i)) \rceil$ のみを出力する. これによって, 符号化されたシンボルが通信路を通って復号器で復号されるまでの時間を観測することができるので, 全体の遅延を求めることができるのである. 本章におけるシャノン符号では符号化確率 $Q(i)$ として, 送信フェーズ時にバッファに溜められたシンボル系列の発生確率を用いる.

4.1.2　単進符号

伝送システムにおいて復号化を考慮する場合, 符号語の長さの情報を復号器へ届けなくてはならない. そこで, 符号語の長さの符号化すなわち整数の符号化を行う. 研究では単進符号を行ったのでその方法を説明する.

単進符号は対象となる整数 X の数 -1 個の数の 0 を出力し, その後に 1 を出力する. 長さ n のシンボルを符号化すれば, 符号語長は n となる. 図 7 に単進符号の例を示す.

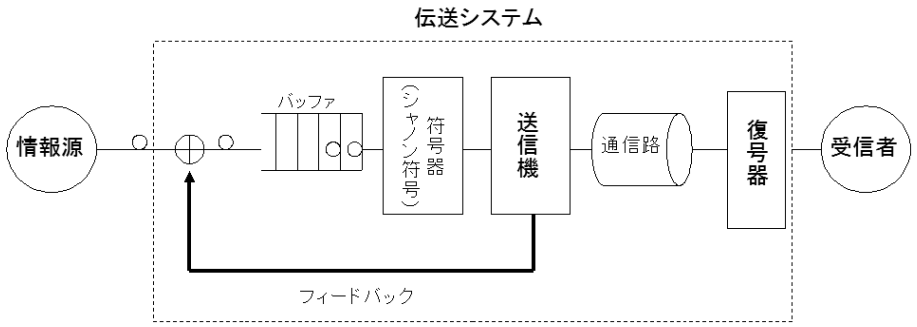

図 8　伝送モデル (シャノン符号)

4.2　システムモデル

伝送システムに符号器を加えた場合のシステムモデルを図 8 に示す．バイナリー情報源 $(1:P, 0:1-P)$ から発生したシンボルはポアソン過程で伝送システムに到着しバッファに溜められる．そして, 符号器でシャノン符号化され送信機へ送られる. 送信機へ送られるタイミングは前章と同じく,Musy and Telatar が提案した最適な送信方法に従う [1]. また通信路内のノイズは発生しないものとし, 復号器内での復号化は瞬時に行われるものとする.

4.3　遅延の期待値 (復号化なし)

この節では符号化のみを考慮し復号化は無視する. つまり, 復号化を行うための送信シンボル数の個数に関する符号化は行わないモデルを考える.

復号化を行わない場合の送信機の送信時間は,X^n を情報源シンボル系列とすると,$T(X) = D + k \log \frac{1}{\Pr(X^n)}$ で表される.

まず送信機の送信時間の期待値を求める.i 番目の送信フェーズにおける送信シンボルの系列を X_i とし, 送信シンボル数の個数を $N_i = n$ とすると送信時間の期待値は

$$\begin{aligned} E[T(X_i)] &= \sum_n E[T(X_i)||X_i| = n] \Pr\{N_i = n\} \\ &= \sum_n E[T(X_i)] \Pr\{N_i = n\} \\ &= \sum_n E[D + k \log \frac{1}{P_{X^n}(X^n)}] \Pr\{N_i = n\} \\ &= \sum_n (D + knH) \Pr\{N_i = n\} \\ &= D + k\overline{N_i}H \end{aligned} \tag{28}$$

となる. ここで,H は 1 シンボルあたりのエントロピーを表す.$i+1$ 番目の送信シンボル数の期待値は

$$\begin{aligned}
\overline{N_{i+1}} &= E[N_{i+1}] \\
&= \sum_n \sum_{x^n} E[N_{i+1}|N_i = n, X^n = x^n] \Pr\{N_i = n, X^n = x^n\} \\
&= \sum_n \sum_{x^n} (\lambda D + \lambda k \log \frac{1}{P_{X^n}(x^n)}) \Pr\{N_i = n, X^n = x^n\} \\
&= \lambda D + \lambda k \sum_n \sum_{x^n} \log \frac{1}{P_{X^n}(x^n)} \Pr\{N_i = n, X^n = x^n\} \\
&= \lambda D + \lambda k \sum_n \Pr\{N_i = n\} \sum_{x^n} \log \frac{1}{P_{X^n}(x^n)} \Pr\{X^n = x^n|N_i = n\} \\
&= \lambda D + \lambda k \sum_n \Pr\{N_i = n\} \sum_{x^n} \log \frac{1}{P_{X^n}(x^n)} P_{X^n}(x^n) \\
&= \lambda D + \lambda k \sum_n \Pr\{N_i = n\} nH \\
&= \lambda D + \lambda k \overline{N_i} H \\
&= \lambda (D + k \overline{N_i} H) \qquad (29) \\
&= \lambda E[T(X_i)] \qquad (30)
\end{aligned}$$

二つの式から

$$\begin{aligned}
E[T(X_{i+1})] &= D + k\overline{N_{i+1}}H \\
&= D + \lambda k H E[T(X_i)] \qquad (31)
\end{aligned}$$

定常状態のとき $i \approx i+1$ より $E[T(X)] = \overline{D_t}$ とおくと

$$\begin{aligned}
\overline{D_t} &= D + \lambda k H \overline{D_t} \\
\overline{D_t} &= \frac{D}{1 - \lambda k H} \qquad (32)
\end{aligned}$$

となり送信時間の期待値を得た.

次に全体の遅延の期待値を求める. シンボルがバッファ内で過ごす時間の期待値は [1] から $\frac{1}{2}\overline{D_t}$ で表されることから, シンボルの遅延の期待値 $\overline{D}$ は

$$\overline{D} = \frac{3}{2}\overline{D_t} = \frac{3}{2}\frac{D}{1 - \lambda k H} \qquad (33)$$

を得た. 式 (3) と式 (33) から明らかに符号化を行ったほうが遅延が小さくなるといえる.

実際にシミュレーションを行い導いた遅延が正しいか確認した結果を図 9, 10 に示す. パラメータは $D = 10, \lambda = 0.1$ に設定し, 情報源の確率を変化させて式 (33) で表される期待値と実験値の平均遅延を比較している.

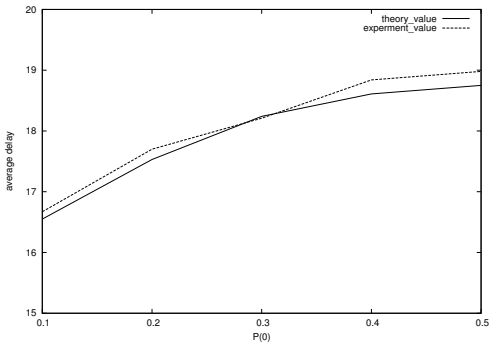

図 9　シャノン符号の遅延特性 ($k = 0.2$)

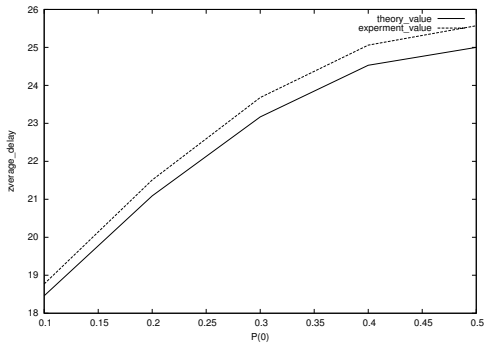

図 10　シャノン符号の遅延特性 ($k = 0.4$)

図から理論値と実験値はほぼ等しい値となり, 導出した遅延の期待値の理論式は正しいといえる.

4.4　伝送システム内のシンボル数 (復号化なし)

伝送システム内にあるシンボル数の期待値を求める. まず, 送信フェーズ時の送信シンボル数の期待値は, リトルの定理 $\overline{N_t} = \lambda \overline{D_t}$ より

$$\overline{N_t} = \frac{\lambda D}{1 - \lambda k H} \tag{34}$$

となり, バッファ内シンボル数は $[1]$ から $\frac{1}{2}\overline{N_t}$ で表されることから, 符号化を行った場合の伝送システム内のシンボル数の期待値は

$$\overline{N} = \frac{3}{2} \frac{\lambda D}{1 - \lambda k H} \tag{35}$$

を得た.

4.5　復号化を考慮する

この節では復号化を行うために, 送信シンボルの個数の符号化も行った場合のモデルを考える. 個数の符号化を行った場合の送信時間は $T(X) = D + k \log \frac{1}{P_{X^n}(X^n)} + k\gamma(n)$ で表される. ここで,$\gamma(n)$ は n 個のシンボルを符号化したときの符号語長とする. このときの送信時間の期

待値は

$$\begin{aligned}
E[T(X_i)] &= \sum_n E[T(X_i)||X_i| = n] \Pr\{N_i = n\} \\
&= \sum_n E[T(X_i)] \Pr\{N_i = n\} \\
&= \sum_n E[D + k \log \frac{1}{P_{X^n}(X^n)} + k\gamma(n)] \Pr\{N_i = n\} \\
&= \sum_n (D + knH + k\gamma(n)) \Pr\{N_i = n\} \\
&= D + k\overline{N_i}H + k \sum_n \gamma(n) \Pr(n) \qquad (36)
\end{aligned}$$

で表され, 送信シンボル数の期待値は式 (36) より

$$\overline{N_i} = \lambda(D + k\overline{N_i}H + k \sum_n \gamma(n) \Pr(n)) \qquad (37)$$

で表される. ここで, 個数の符号化を単進符号で行うとすると $\gamma(n) = n$ が成り立つ. よって式 (36) と式 (37) から送信時間の期待値 $\overline{D_t}$ は

$$\overline{D_t} = \lim_{i \to \infty} E[T(X_i)] = \frac{D}{1 - \lambda kH - \lambda k} \qquad (38)$$

を得た. また, 送信シンボル数の期待値は

$$\overline{N_t} = \frac{\lambda D}{1 - \lambda kH - \lambda k} \qquad (39)$$

を得た.

5 算術符号を含む伝送モデル

本章では符号器に逐次符号 (算術符号) を用いた伝送システムの遅延特性を示す. また, フィードバックの有効性についても論じる.

5.1 実験原理

5.1.1 算術符号

算術符号は記号列を実数の $[0, 1)$ の区間を用いて符号化を行う方法である. 以下の例では, 記号は $\{0, 1\}$ の 2 種類があり, 出現確率はそれぞれ $0.2, 0.8$ とする. 算術符号は, 区間を記号の出現確率に比例した小区間に分割していくことで符号化を行う. ここで, 記号列 01111 を符号化する. この過程を図 11 に示す. x 以上 y 未満の区間を $[x, y)$ と表すこととする. 区間の初期

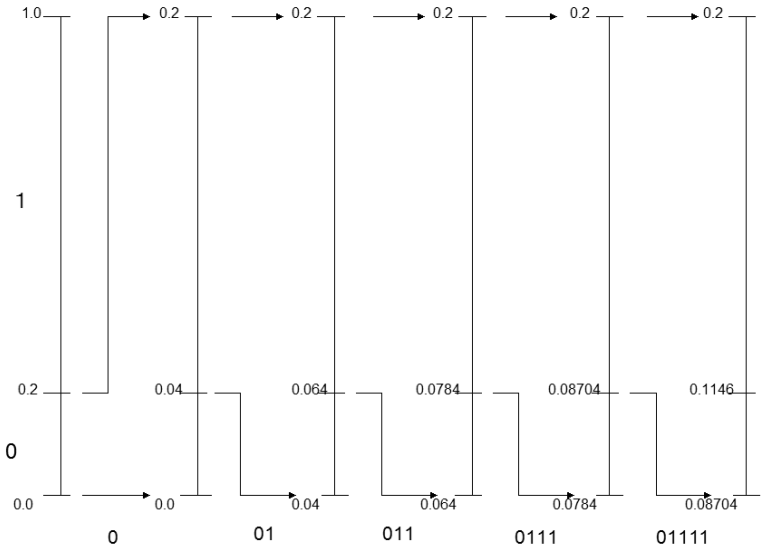

図 11　算術符号の符号化の過程

値は $[0, 1)$ である．記号を読み込んだら区間 $[0, 1)$ を分割する．記号が 0 ならば区間の 0 から 0.2 までの部分,1 ならば 0.2 から 1.0 までの部分に分割する．

最初の記号は 0 なので区間は $[0, 0.2)$ である．次の記号は 1 なので, 区間 $[0, 0.2)$ の 1 の部分 $[0.04, 0.20)$ が新しい区間である．このように, 記号を読み込むたびに区間を分割していくと, 記号列 01111 を表す区間は $[0.08704, 0.1146)$ となる．そして, 実際の算術符号の符号語は, この区間に含まれる一つの実数を指定する．

ここで符号語を 2 進数で表して, 区間内で小数点以下のビット数の少ない値を選ぶことにする．例えば 0.1171875 を 2 進数で表すと次のようになる．

$$0.1171875 = \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128} = (0.0001111)_2$$

0001111 の 7 ビットを符号語として出力すれば, 記号列 01111 の 5 文字を 7 ビットに圧縮することができるわけである．この例では 1 文字が 1.4 ビットに圧縮されたが, 記号の出現確率によっては 1 文字が 1 ビット未満に圧縮できる場合もある．

5.1.2　**算術符号の復号**

次に復号について説明する．ここでは説明の都合上, 符号語は下限値の 0.08704 とする．0.08704 は,$[0, 0.2)$ の間にあるので, 最初の記号は 0 であることがわかる．次に 0 を表す区間 $[0, 0.2)$ を $[0, 1.0)$ になるように拡大すると, 符号語は次のように変換できる．

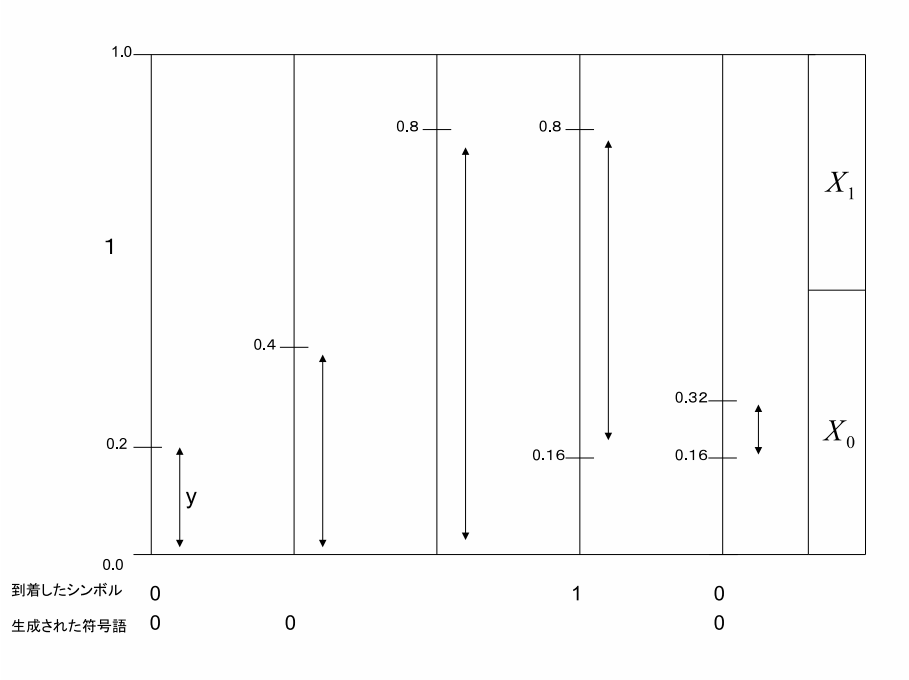

図 12　研究で用いた算術符号

$$\text{新しい符号語} = (\text{符号語} - \text{記号の下限値})/\text{記号の区間幅} \tag{40}$$
$$= (0.08704 - 0)/0.2 \tag{41}$$
$$= 0.4352 \tag{42}$$

新しい符号語 0.4352 は $[0.2, 0.8)$ の間にあるので, 次の記号は 1 であることがわかる. このような操作を繰り返すことにより記号列 01111 を復号することができる.

5.1.3　研究で用いた算術符号

前節は一般的な算術符号の説明である. しかしこの方法では生成される符号語が小数点以下の値をとるため, プログラムで扱う時に有効桁数の問題などの困難が生じる. そこで本研究では符号語が実数値でなく $0, 1$ で生成されるような算術符号を用いた. この節ではその算術符号の原理を紹介する.

図 12 は研究で用いた算術符号の原理図である. まず例として, 情報源確率を $P(0) = 0.2, P(1) = 0.8$ に設定し符号化確率も同様に $p(0) = 0.2, p(1) = 0.8$ とする. また符号語 0 が生成される区間を $X_0 = [0.0, 0.5)$, 符号語 1 が生成される区間を $X_1 = [0.5, 1.0)$ とす

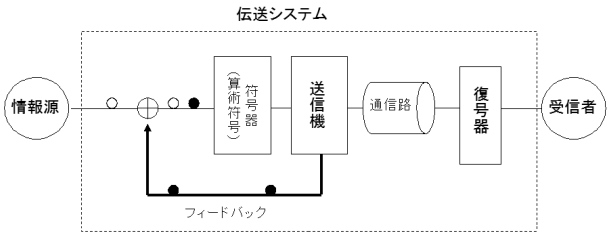

図 13　フィードバックモデル

る．まず符号器に 0 のシンボルが到着したとする.0 の符号化確率からシンボル 0 の区間 y は $0 \leq y \leq 0.2$ である. この区間 y は X_0 の区間に含まれることから, 符号語 0 が生成される. 次に区間 y を 2 倍する. すると先の区間は $0 \leq y \leq 0.4$ になり, この区間も X_0 に含まれることから符号語 0 が生成される. 同様に区間 y を 2 倍すると $0 \leq y \leq 0.8$ となる. この区間 y は X_0, X_1 両方の区間に含まれるので符号語は生成されない．次に 1 が到着すると 1 の符号化確率の関係から,$0.16 \leq y \leq 0.8$ となり符号語は生成されない. 次に 0 が到着すると y の区間は $0.16 \leq y \leq 0.32$ となり符号語 0 が生成される. このようにして符号化を行っていく.

復号化は逆の手順で区間が X_0 に含まれるのか X_1 に含まれるのかを決定して復号を行う.

5.2　フィードバックモデル

フィードバックとは前章でも扱っていた,Musy and Telatar が示した最適な送信方法に基づくものだが, 符号器が算術符号化を行う場合は少し様相が変わる. 送信機が送信可能時にフィードバックによって符号語を符号器から要請するのは同じだが, 算術符号の場合はフィードバックによってかえって符号語が長くなるという特徴を持つ. そこでフィードバックが有効であるかどうかが問題となる.

5.2.1　システムモデル

フィードバックを用いた伝送システムのモデルを図 13 に示す．まず, 情報源から発生したシンボルは伝送システムにポアソン過程で到着する. 到着したシンボルは符号器内で算術符号化され, 生成された符号語は送信機へ送られる. そして伝送レート $\frac{1}{k}$[symbol / sec] の通信路へと送信され, 復号器で復号される．フィードバックを行わない場合, 送信機が送信可能時に送信バッファ内に符号シンボルがなければ送信機は遊休状態となり遅延を大きくする要因になると考えられる．そこで, それを防ぐため送信可能時に送信機から, シンボルを要求する情報を符号器へ送るというフィードバックを行い, 強制的に符号語を生成させることで送信機を常に稼動状態にする．この送信機から符号器へ送られるシンボルをフィードバックシンボルと呼ぶ.

この伝送モデルは前章まで用いてきた最適な送信方法と原理的には同じである. しかし, フィードバックを行うことによって送信機の遊休時間が小さくなる一方で生成される符号語長は長くなる. そこで, 結果的にシンボルの遅延は小さくなるのかをシュミレーションにより明らかにする.

5.2.2 フィードバックシンボル

フィードバックシンボルを含む算術符号の説明を行う.

まず情報源シンボル $\{0,1\}$ の発生確率を $P(0),\ P(1)$ とする. このとき $P(0),\ P(1)$ は $P(0)+P(1)=1$ を満たす.

次にフィードバックシンボルを含む符号化確率を $p(0), p(1), p(\epsilon)$ とする. $p(\epsilon)$ を設定した下で $p(0)=P(0)\,(1-p\,(\epsilon)), p(1)=P\,(1)\,(1-p\,(\epsilon))$ と置く. これらの値は $p(0)+p(1)+p(\epsilon)=1$ を満たす.

このように符号化確率を設定することで, シンボルの情報が符号器の内部に滞っている時でも, 送信機からのフィードバックシンボル ϵ を受けとることで, 符号器の内部状態が決定され符号語を生成することができる. $p(\epsilon)$ を小さく設定すると, フィードバック時に生成される符号語は非常に長くなる. 一方で $p(\epsilon)$ を大きく設定すれば普段（フィードバックを行わない）生成される符号語が長くなる.

5.3 フィードバックモデルの遅延特性

この章では図 13 のモデルにおけるシンボルの遅延を各パラメータを変化ながら解析を行う. また, フィードバックを行わないモデルにおけるシンボルの遅延と比較することで, フィードバックの有効性を示す.

5.3.1 フィードバックの有効性

まず情報源シンボルの発生確率を $P(0)=0.3$, 伝送システムへの到着レートを $\lambda=0.1, 0.5$ に設定する. このとき, フィードバックを行った場合と行わなかった場合とで遅延特性を比較したものを図 14, 15 に示す. グラフは横軸をフィードバックシンボルの符号化確率 $p(\epsilon)$, 縦軸は平均遅延を表す.

グラフから, フィードバックシンボルの符号化確率が $0<p(\epsilon)\leq 0.1$ の範囲ではフィードバックを行う方がシンボルの遅延が小さくなることが判明した. また, 到着レートを大きくすることで遅延の大きさの差が小さくなった. ただ, 到着レートを 1 に近づけてもフィードバックを行う方が遅延は小さくなった. よって符号器が算術符号の場合でもフィードバックは有効であることが判明した.

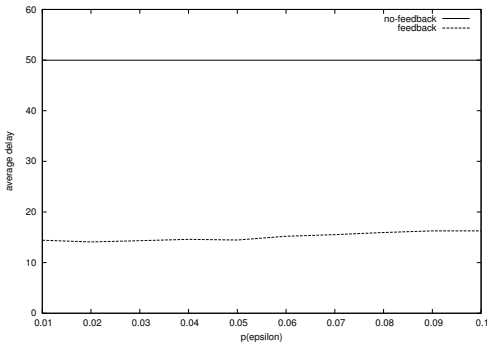

図 14　遅延特性の比較 $(P(0) = 0.3, \lambda = 0.1)$

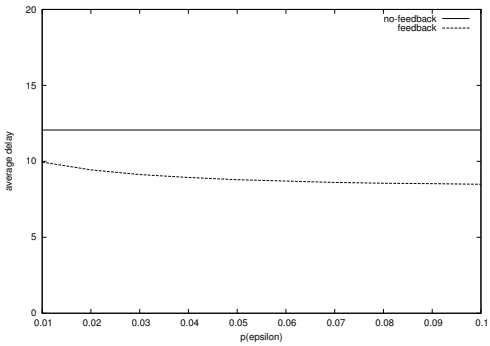

図 15　遅延特性の比較 $(P(0) = 0.3, \lambda = 0.5)$

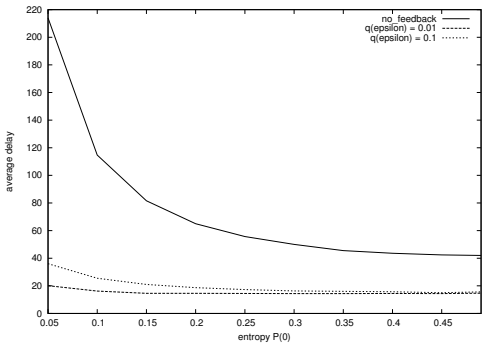

図 16　エントロピー　- 遅延特性 $(\lambda = 0.1)$

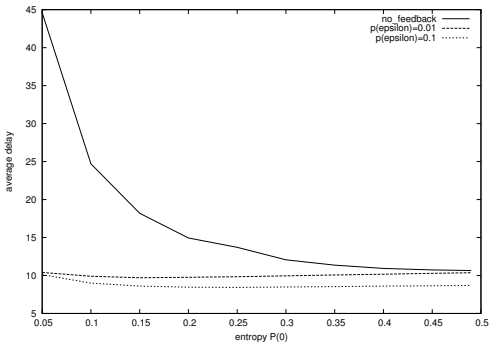

図 17　エントロピー　- 遅延特性 $(\lambda = 0.5)$

5.3.2　エントロピー-平均遅延特性

次に, 伝送システムへの到着レートを $\lambda = 0.1, 0.5$ に設定して, 情報源 $P(0)$ の出現確率に対する平均遅延の大きさに関してシミュレーションを行った結果を図 16, 17 に表す.

グラフからフィードバックを行う場合, 情報源確率を変化させても遅延の大きさはあまり変化しないことが判明した. 情報源確率が小さい範囲では, フィードバックなしに比べてフィードバックを行う方が遅延がはるかに小さくなった. 従ってフィードバックの効果として, 情報源確率が小さいときに遅延の大きさが大きく改善されることが判明した.

5.3.3　$p(\epsilon)$-利用率特性

この節では通信路の利用率を検証する. ある情報源からの情報を送る時, 通信路の利用率をなるべく小さくなるように設定することが望まれる. なぜなら利用率が小さいほうがその通信

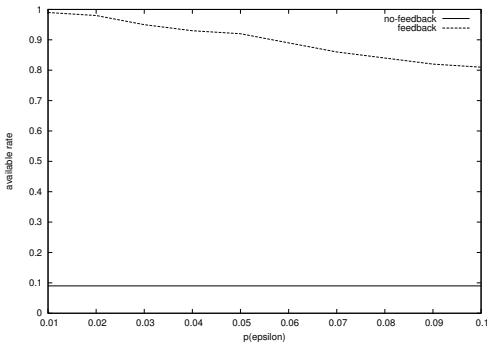

図 18　$p(\epsilon)$-利用率特性 $(P(0) = 0.3, \lambda = 0.1)$

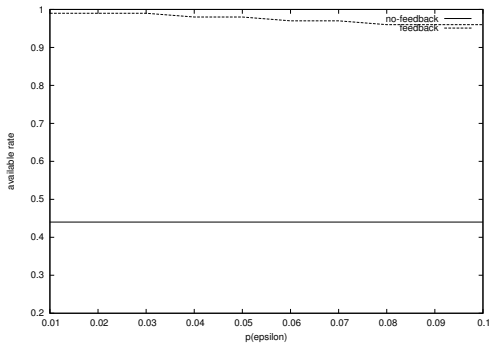

図 19　$p(\epsilon)$-利用率特性 $(P(0) = 0.3, \lambda = 0.5)$

路に空きが生じ, 他の情報源からも情報を送ることが可能となることで通信路を有効に使えるからである. 従って, フィードバックを行うことによって利用率がどのように変化するかを検証する. グラフは情報源シンボルの発生確率を $P(0) = 0.3$, 到着レートを $\lambda = 0.1, 0.5$ としたときの利用率を示す. 横軸はフィードバックシンボルの符号化確率 $p(\epsilon)$, 縦軸は利用率を示す.

図 18, 19 ともにフィードバックを行う場合の利用率が 1 近くなり, フィードバックを行わない場合に比べて大きくなるという結果になった. これは, フィードバックを行うことで送信機を常に稼動状態にしているので利用率が高くなったと考えられる. 一般的に利用率は低い方が他の用途にも使用できるという点で優れていると考えられるが, 本研究では遅延の大きさに着目するので, この結果に関しては深く論じない.

5.3.4　$p(\epsilon)$-平均遅延特性

シンボルの符号化確率 $p(\epsilon)$ を変化させたときの平均遅延の振る舞いを図 20 に示す.

グラフから, 遅延の大きさは $p(\epsilon)$ を変化させることにより単調増加や単調減少せずに凹型のグラフになることが判明した. よって各パラメータごとに遅延の値が最小となる $p(\epsilon)$ が存在することが判明した.

次章では最適な遅延をとるような $p(\epsilon)$ を定量的に式で表す方法を考察する.

5.4　最適な $p(\epsilon)$ の設定

前節から, 送信機から符号器へフィードバックシンボルを送るというフィードバックを行う方が遅延が小さくなることが判明した. この章では, 遅延が最小となるようなフィードバックシンボルの符号化確率を定量的に示す方法を紹介する.

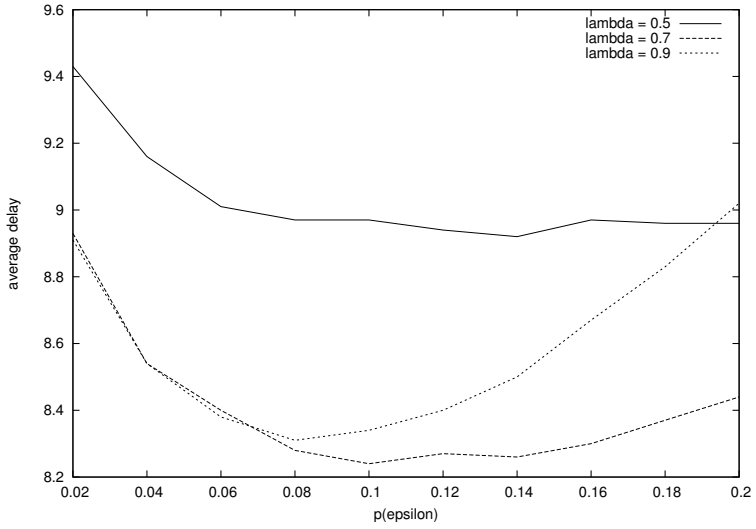

図 20　$p(\epsilon)$ - 遅延特性 $(P(0) = 0.1)$

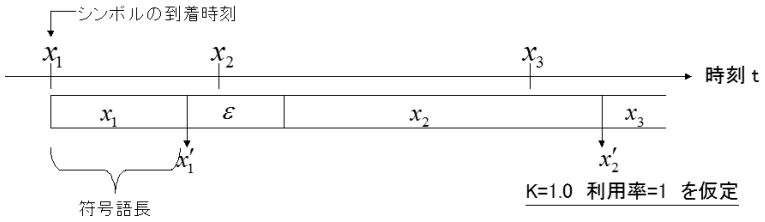

図 21　シンボルの到着時刻と復号時刻

5.4.1　**原理**

送信機の送信レートを $k = 1.0$, そしてシミュレーションからフィードバックによって通信路の利用率がほぼ 1 に近づくことから利用率 $= 1.0$ と仮定した場合, シンボルの到着時刻と復号時刻の関係は図 21 のように表せる.

まず,1 シンボルあたりの平均符号語長を定量的に示す.

$p(0), p(1), p(\epsilon)$: 符号化確率

$q(\epsilon)$: フィードバックが付加される確率

$p(0) = P(0)(1 - p(\epsilon))$: シンボル 0 の符号化確率

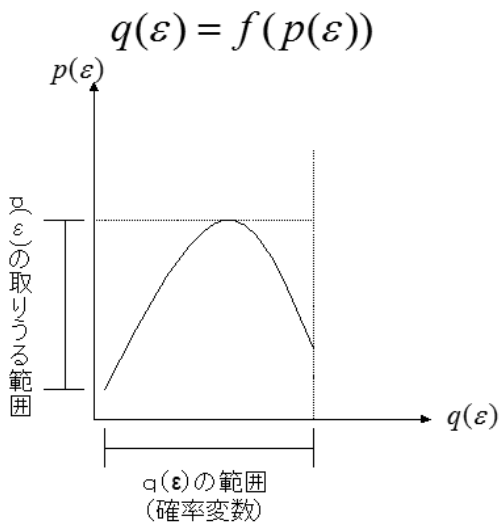

図 22　$p(\epsilon) - q(\epsilon)$ 特性

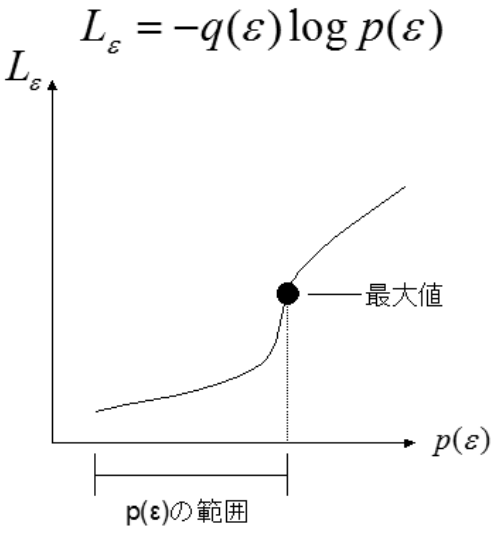

図 23　$p(\epsilon) - L_\epsilon$ 特性

$p(1) = P(1)(1 - p(\epsilon))$: シンボルの 1 符号化確率

を用いて,1 シンボルあたりの平均符号語長は

$$\begin{aligned} &= -p(0)(1 - q(\epsilon)) \log p(0) - p(0)q(\epsilon) \log p(p(0)p(\epsilon) \\ &\quad -p(1)(1 - q(\epsilon)) \log p(1) - p(1)q(\epsilon) \log p(p(1)p(\epsilon) \end{aligned} \tag{43}$$

と表される.

$k = 1.0$ より利用率 $= 1.0$ とするなら, 送信機は符号を 1 秒あたり高々 1 個送ることになる. よってリトルの定理から λ[symbol/sec] をかけると,

$$\lambda(1 \text{ シンボルあたりの平均符号語長}) \leq 1 \tag{44}$$

の関係が成立する. この式を変形して $q(\epsilon) = f(p(\epsilon))$ の形で表すと

$$\log P(0) = A, \log P(1) = B, \log p(\epsilon) = X, \log(1 - p(\epsilon)) = Y$$

として

$$q(\epsilon) = \frac{\frac{1}{\lambda} + (P(0)A + P(1)B)(1 - p(\epsilon)) + (1 - p(\epsilon))Y}{(p(\epsilon) - 1)X} \tag{45}$$

と表される.$q(\epsilon)$ は確率変数であるので取りうる値の範囲は $0 < q(\epsilon) \leq 1$ である. 従って,$p(\epsilon)$ の取りうる値は $0 < q(\epsilon) \leq 1$ の範囲の中で $p(\epsilon)$ が取る値となる. 図 22 にこの様子を示した.

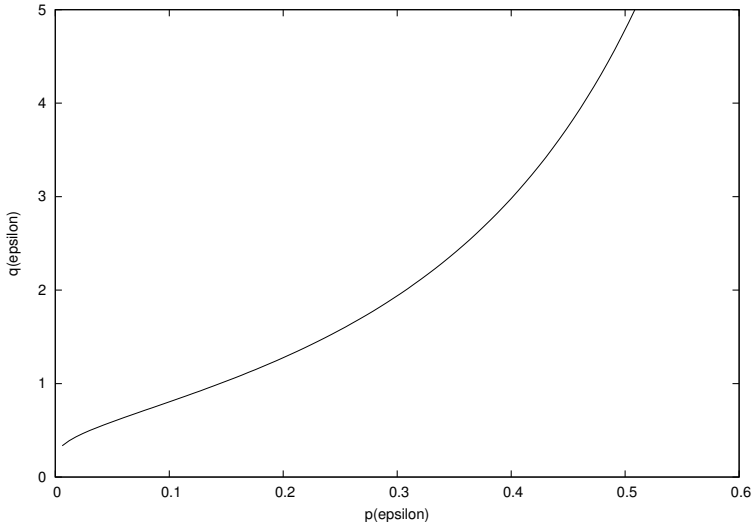

図 24　$p(\epsilon) - q(\epsilon)$ 特性

通常のシンボル 0, 1 の後ろに付加される符号語の期待値を L_ϵ とすると,

$$L_\epsilon = -q(\epsilon) \log p(\epsilon) \tag{46}$$

が成り立つ. 利用率が 1.0 であるという仮定から送信機がシンボルを高々 1 個しか送れないので, この L_ϵ が大きくなるにつれて通常のシンボルの符号語長が小さくなると考えられる（図 22 参照）. よって, シンボルの遅延が最も小さくなるのはこの L_ϵ が最も大きくなるときであると考えられる. (図 23 参照)

よって式 (45), (46) を用いて L_ϵ が最大となるような $p(\epsilon)$ を求める.

5.4.2　シミュレーション

はじめに, 前節で述べた L_ϵ が最大のときに遅延が最小となるかどうかをシミュレーションによって確認する.

まず,$p(\epsilon)$ の範囲を設定する. 式 (45) から $q(\epsilon) = f(p(\epsilon))$ は確率より, $0 < q(\epsilon) \leq 1$ の中に収まるはずである. その中で $p(\epsilon)$ がとりうる値を確認する. パラメータは $\lambda = 0.3, P(0) = 0.3$ に設定してシミュレーションを行った結果を図 24 に示す.

図から $q(\epsilon) = f(p(\epsilon))$ の値は確率であるにもかかわらずは 1 以上の値をとった. よって上記のような考え方は間違っているのではと考えられる.

補足として,$p(\epsilon)$ の範囲を $0 < p(\epsilon) \leq 1$ のときの L_ϵ のグラフを図 25 に載せる.

ちなみに,L_ϵ が最大のときの $p(\epsilon)$ の値をパラメータにしてシミュレーションを行った時の遅

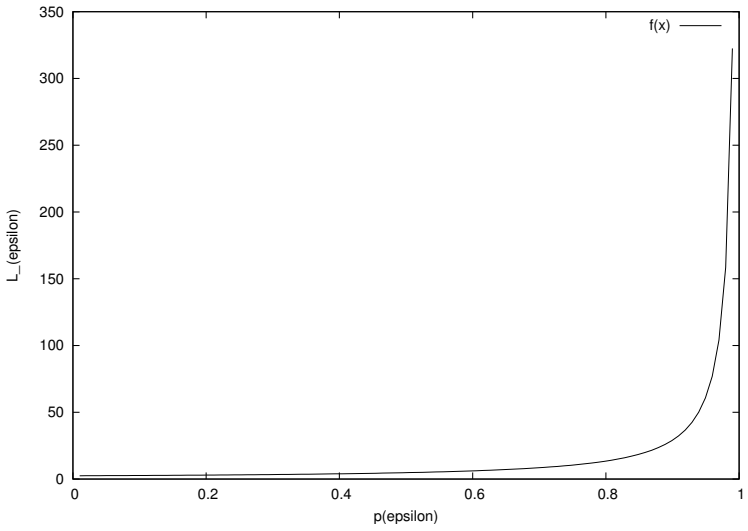

図 25　$p(\epsilon) - L_\epsilon$ 特性

延の大きさは最小値をとらなかった.

5.5　まとめ

この章のまとめを行う. 符号器に算術符号を用いたときの伝送システムは, 送信機から符号器へのフィードバックを行う方が遅延が小さくなることが判明した. これは, ブロック符号の時と同じ結果である. フィードバックの効果として, 情報源のエントロピーが小さいほうが遅延を大きく改善できることが判明した. また, 各パラメータごとに遅延の値が最小となる $p(\epsilon)$ が存在することが判明した.

最適な $p(\epsilon)$ を求める方法として, 利用率を 1.0 と仮定して平均符号語長を導出し, それを用いて L_ϵ が最大のときに遅延が最小になると考え,$p(\epsilon)$ を求めようと試みた. しかし, 導出した $q(\epsilon) = f(p(\epsilon))$ の式を用いてシミュレーションを行うと,$q(\epsilon)$ が確率変数にも関わらず 1 を超えたので, この考え方は間違いであるということがわかった.

今後の課題としては, 別のアプローチから最適な $p(\epsilon)$ を求めることである.

6　ブロック符号と算術符号の伝送モデルの比較

この章では全体のまとめとして, 符号器をブロック符号・算術符号の各々でフィードバックを行った場合の遅延を比較する. ブロック符号の符号器はシャノン符号化と ω 符号化を行う.

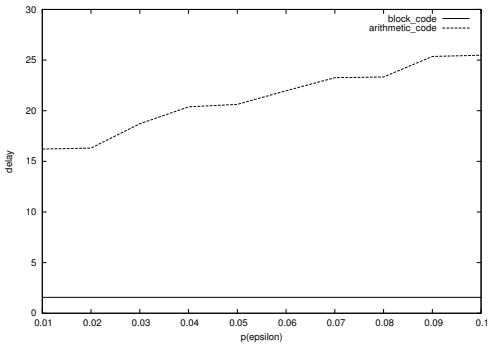

図 26　$p(\epsilon)-$ 遅延特性 $(P(0) = 0.1, \lambda = 0.1)$

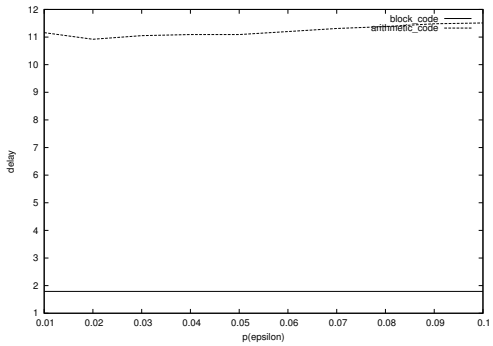

図 27　$p(\epsilon)-$ 遅延特性 $(P(0) = 0.1, \lambda = 0.3)$

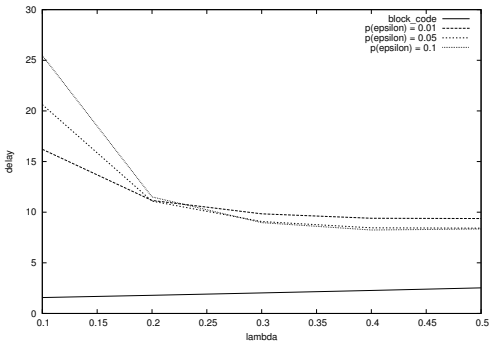

図 28　$\lambda-$ 遅延特性 $(P(0) = 0.1)$

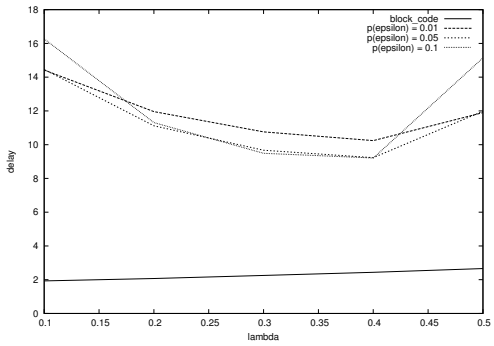

図 29　$\lambda-$ 遅延特性 $(P(0) = 0.3)$

まずは各々の情報源シンボルの発生確率を $P(0) = 0.1$, 到着レートを $\lambda = 0.1, 0.3$ に設定したときの遅延を比較した結果を図 26, 27 に示す. 横軸はフィードバックシンボルの符号化確率 $p(\epsilon)$ を表す. 図から, ブロック符号を行う方が遅延が小さくなることが判明した.

次に情報源シンボルの発生確率を $P(0) = 0.1, 0.3$ に設定し, 到着レートを動かしたときの遅延を比較した結果を図 28, 29 に示す. 結果から,$p(\epsilon)$ をどのように設定しても, 算術符号よりブロック符号の方が遅延が小さくなることが判明した.

ブロック符号と算術符号とで伝送システムが異なるので一概に比較することはできないが, その点を無視すれば, ブロック符号より算術符号の方が遅延は小さくなるという結果になった. これは厳密に伝送モデルを検証した上での結果ではないので, 遅延の観点からはブロック符号の方が算術符号より優れていると結論付けるのは早い.

7 結論

Musy and Telatar が提案した最適な送信方法に基づいて, フィードバックを用いた伝送システムを考案し遅延特性を解析した. 伝送システム内がバッファと送信機のみで構成されるモデルにおいて, シミュレーションを行い遅延特性を求めた結果,Musy and Telatar が提案した遅延の期待値の理論式とシミュレーション結果が一致した. 次に送信フェーズ時の送信シンボル数の期待値を理論的に求めた.

伝送システム内の符号器がシャノン符号化を行うモデルでは, 遅延の期待値を理論的に求めた. また, 送信シンボル数の期待値も求めた.

次に伝送システム内の符号器が算術符号化を行うモデルを考え, フィードバックの有無で遅延を比較することでフィードバックの有効性を考察した. 結果フィードバックを行うことによって遅延が小さくなり, 遅延の観点からはフィードバックが有効であることが判明した. またフィードバックの効果として, 情報源のエントロピーが小さい範囲では遅延が大きく改善されることが判明した. 遅延は ϵ の符号化確率によって変化することから, 今後の課題として遅延が最小となるような $p(\epsilon)$ を定量的に求めることが今後の課題として挙げられる.

謝辞

本研究を行うにあたって, 数多くの助言をいただいた西新幹彦先生に感謝の意を表する.

参考文献

[1] Stephane Musy, Emre Telatar,“On the transmission of bursty sources,” ISIT 2006, pp.2899-2903, 2006.

[2] 韓太舜, 小林欣吾 (1999)「情報と符号化の数理」pp.119. 培風館

[3] Dimitri Bertsekas, Robert Galager, *Data Netwarks,* Prentice Hall, 1987.

付録 A　ソースコード

A.1　フィードバックを行う伝送システムのソースコード (算術符号)

```
/*符号器が算術符号でフィードバックあり*/
/*平均遅延を求める*/
/*情報源は{1 , 2}で構成　フィードバックシンボル ε は{0}*/
/*コマンドラインで入力*/


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



/*関数定義*/
#define TOTAL_SYMBOL 1048576        /*入力長  (2^20bit)*/

#define CODE_VALUE_BITS 16
#define TOP_VALUE (((long)1 << CODE_VALUE_BITS) - 1)    /*符号化の精度*/

#define FIRST_QTR (TOP_VALUE / 4 + 1)
#define HALF (2 * FIRST_QTR)
#define THIRD_QTR (3 * FIRST_QTR)

#define NO_OF_CHARS (3)         /*シンボルの種類 ( ε ,1,2)*/

#define EPSILON 0



#define MAX_FREQUENCY (16383)



#define P 0.1       /*到着レート λ*/
#define K 1     /*伝送レート・通信路容量 [sec/シンボル]*/


#define ARRIVING_LIST_MAX 1000
#define SENDING_BUFFER_LIST_MAX (1000)



/******関数プロトタイプ宣言******/
void add_arriving_list(double) ;
void encode_symbol(int , int[]) ;
void bit_plus_follow(int) ;
void decode_symbol(int, int[]) ;
double get_arriving_list() ;
void add_sending_buffer_list(int) ;
void sending_finish() ;
int get_sending_buffer_list() ;
void epsilon_event() ;



/*******指数分布乱数を発生*******/
#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();
}

double rand_(void)
{
    return (-log(1-(rnd())))/P;  /*パラメータ P の指数乱数 (λ=P)*/
}

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

/*グローバル変数*/
double present_time ;       /*現時時刻*/
double arrival_time ;       /*到着時刻*/
double sending_time ;       /*送信終了時刻*/



int next_ch ;


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

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


int bit ;
int cum_freq[NO_OF_CHARS + 1] ;          /*累積出現確率*/

double long cum_delay ;     /*平均遅延を求める*/

int freq[NO_OF_CHARS] ;
int symbol_0 ;       /*情報源シンボル{0,1, ε}*/
int symbol_1 ;
int symbol_e ;       /*フィードバック信号*/

int nofeedback_code_number ;     /*フィードバックしても符号が生成されなかった回数*/
int feedback_number ;        /*フィードバックした回数*/
double total_sending_time ; /*通信路の利用率の計算用*/

int encoder_zero ;  /*エンコーダーに入る前のエントロピー計算*/
int encoder_one ;

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

void start_model()
{
    /*累積出現確率の計算*/
    cum_freq[0] = 0 ;
    cum_freq[1] = symbol_e ;
    cum_freq[2] = symbol_e + symbol_0 ;
    cum_freq[3] = symbol_e + symbol_0 + symbol_1 ;

    if(cum_freq[NO_OF_CHARS] > MAX_FREQUENCY)
        abort() ;

    return ;
}


/*****************************************************************/
int low ;
int high ;
long bits_to_follow ;
unsigned long input_length = 0 ;
unsigned long output_length = 0 ;

void start_encoding()
{
    low = 0 ;
    high = TOP_VALUE ;
    bits_to_follow = 0 ;

    return ;
}
/*****************************************************************/
int d_low ;
int d_high ;
int v_low ;
int v_high ;

void start_decoding()
{
    d_low = 0 ;
    d_high = TOP_VALUE ;
    v_low = 0 ;
    v_high = TOP_VALUE ;
```

```
    return ;
}

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

/*到着イベント*/
void arrival_symbol()
{
    input_length++ ;

    if (input_length >= TOTAL_SYMBOL){          /*情報源から全てのシンボルが到着した*/
        arrival_time = -1.0 ;
    }else{
        arrival_time = present_time + rand_() ;     /*次の到着時刻を設定*/
    }

    add_arriving_list(present_time) ;       /*到着リスト処理*/

    encode_symbol(next_ch , cum_freq) ;     /*エンコード処理*/

    return ;
}



/*到着リスト処理*/
void add_arriving_list(double time)
{
    if (arriving_count >= ARRIVING_LIST_MAX){
        printf("warning1!!") ;
        exit(0) ;
    }
    arriving_list[arriving_in] = time ;
    arriving_in = (arriving_in + 1) % ARRIVING_LIST_MAX ;
    arriving_count++ ;

    return ;
}


/*算術符号*/
void encode_symbol(int symbol , int cum_freq[])
{
    long range;

    range = (long)(high - low) + 1 ;
    high = low + (range * cum_freq[symbol+1]) / cum_freq[NO_OF_CHARS] - 1 ;
    low = low + (range * cum_freq[symbol]) / cum_freq[NO_OF_CHARS] ;

    if(high <= low){
        printf("Warning5!!") ;
        exit(0) ;
    }

    for(;;){
        if (high < HALF){
            bit_plus_follow(0) ;

        } else if (low >= HALF){
            bit_plus_follow(1) ;

            low -= HALF ;
            high -= HALF ;
        } else if (low >= FIRST_QTR && high < THIRD_QTR){
            bits_to_follow += 1 ;
            low -= FIRST_QTR ;
            high -= FIRST_QTR ;
        } else {
        break ;
```

```
        }
        low = 2 * low ;
        high = 2 * high + 1 ;
    }

    return ;
}


void bit_plus_follow(int bit)       /*符号語を sending_buffer に入れる*/
{
    add_sending_buffer_list(bit) ;

    while(bits_to_follow > 0){
        add_sending_buffer_list(1 - bit) ;
        bits_to_follow-- ;
    }

    if(sending_time < 0.0){
        sending_time = present_time + K ;       /*送信終了時刻を設定*/
        total_sending_time += K ;           /*通信路の利用率の計算*/
    }

    return ;
}


/*送信バッファに符号語を追加*/
void add_sending_buffer_list(int bit)
{
    if (sending_buffer_count >= SENDING_BUFFER_LIST_MAX){
        printf("warning3!!") ;
        exit(0) ;
        }
        sending_buffer_list[sending_buffer_in] = bit;
        sending_buffer_in = (sending_buffer_in + 1) % SENDING_BUFFER_LIST_MAX ;
        sending_buffer_count++ ;

        return ;
}


/*****************************************************************/
/*送信イベント*/
void sending_finish()
{
    int bit ;

    bit = get_sending_buffer_list() ;       /*送信バッファ内の符号語を 1 つ読み出す*/

    decode_symbol(bit , cum_freq) ;     /*デコード処理*/

    if(sending_buffer_count == 0){
        if(input_length >= TOTAL_SYMBOL){
            sending_time = -1.0 ;
        }else{
            epsilon_event() ;       /*フィードバック信号により符号化中のシンボルを送信機へ*/
            if(sending_buffer_count > 0){
                sending_time = present_time + K ;
                total_sending_time += K ;           /*通信路の利用率の計算*/
            }else{
                sending_time = -1.0 ;        /*フィードバック信号でも符号語が生成されなかった*/
                nofeedback_code_number++ ;   /*フィードバックしても符号が生成されなかった回数*/
            }
        }
    }else{
        sending_time = present_time + K ;       /*次の符号語の送信時刻を設定*/
        total_sending_time += K ;           /*通信路の利用率の計算*/
    }
```

```
    return ;
}


/*バッファ内の符号語を 1 つ読み出す*/
int get_sending_buffer_list()
{
    int bit ;

    if(sending_buffer_count == 0){
        printf("warning4!!") ;
        exit(0) ;
    }

    bit = sending_buffer_list[sending_buffer_out] ;
    sending_buffer_out = (sending_buffer_out + 1) % SENDING_BUFFER_LIST_MAX ;
    sending_buffer_count-- ;

    return(bit) ;
}


/*デコード処理*/
void decode_symbol(int bit , int cum_freq[])
{
    if( bit == 0 ){
        v_high = (v_high + v_low + 1) / 2 - 1 ;
    } else {
        v_low = (v_high + v_low + 1) / 2 ;
    }
    for(;;){
        int d_b_1 ;
        int d_b_2 ;
        double delay ;
        d_b_1 = ((d_high - d_low + 1) * cum_freq[1]) / cum_freq[3] + d_low ;
        d_b_2 = ((d_high - d_low + 1) * cum_freq[2]) / cum_freq[3] + d_low ;

        if (v_high+1 <= d_b_1){
            d_high = d_b_1 - 1 ;
        }else if ((v_low >= d_b_1) && (v_high+1 <= d_b_2)){
            if(output_length < input_length){
                delay = present_time - get_arriving_list() ;
                cum_delay += delay ;        /*平均遅延を求める*/

                output_length++ ;

            }
            d_high = d_b_2 - 1 ;
            d_low = d_b_1 ;
        } else if (v_low >= d_b_2){
            if(output_length < input_length){
                delay = present_time - get_arriving_list() ;
                cum_delay += delay ;        /*平均遅延を求める*/

                output_length++ ;

            }
            d_low = d_b_2 ;
        } else {
            break ;
        }
        for (;;){
            if (d_high+1 <= HALF){
                d_low = d_low * 2 ;
                d_high = (d_high + 1) * 2 - 1 ;
                v_low = v_low * 2 ;
                v_high = (v_high + 1) * 2 - 1 ;
            } else if (d_low >= HALF){
```

```
                d_low = (d_low - HALF) * 2 ;
                d_high = (d_high - HALF + 1) * 2 - 1 ;
                v_low = (v_low - HALF) * 2 ;
                v_high = (v_high - HALF + 1) * 2 - 1 ;
            } else if(d_low >= FIRST_QTR && d_high + 1 <= THIRD_QTR){
                d_low = (d_low - FIRST_QTR) * 2 ;
                d_high = (d_high - FIRST_QTR + 1) * 2 - 1 ;
                v_low = (v_low - FIRST_QTR) * 2 ;
                v_high = (v_high - FIRST_QTR + 1) * 2 - 1 ;
            } else {
        break ;
                }
            }
        }
        return;
}


/*到着時刻を読み出す*/
double get_arriving_list()
{
    double time ;

    if(arriving_count == 0){
        printf("warning2!!") ;
        exit(0) ;
    }
    time = arriving_list[arriving_out] ;
    arriving_out = (arriving_out + 1) % ARRIVING_LIST_MAX ;
    arriving_count-- ;

    return(time) ;
}

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

/*フィードバック信号*/
void epsilon_event()
{
    encode_symbol(EPSILON , cum_freq) ;
    feedback_number++ ;
    return ;
}


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

int main(int argc , char **argv)
{
    double average_delay = 0 ;
    double encoder_average_delay = 0 ;

    double probability_zero ;
    double probability_one ;
    double probability_epsilon ;
    double encoder_entropy ;
    double encoder_arrival_rate ;


    double kakuritu ;


    if(argc != 4) goto ERROR ;
    if(argv[1] == NULL || argv[2] == NULL || argv[3] == NULL) goto ERROR ;
    if((symbol_e = atof(argv[1])) == 0) goto ERROR ;     /*情報源シンボルの頻度をコマンドラインで入力*/
    if((symbol_0 = atof(argv[2])) == 0) goto ERROR ;     /*入力の順番は{ε,0,1}*/
    if((symbol_1 = atof(argv[3])) == 0) goto ERROR ;
```

```
    present_time = 0.0 ;        /*現在時刻*/
    arrival_time = rand_() ;    /*到着時刻*/
    sending_time = -1.0 ;       /*送信終了時刻*/


    init_rnd(3) ;       /*seed 決定*/


    start_model() ;
    start_encoding() ;
    start_decoding() ;


    kakuritu = (double) symbol_0 / (symbol_0 + symbol_1) ;      /*情報源シンボル{0}の発生確率*/


    while((arrival_time >= 0.0) || (sending_time >= 0.0)){      /*終了判定*/
        if(arrival_time < 0 || (sending_time >= 0 && arrival_time > sending_time)){    /*イベント判定*/
            present_time = sending_time ;
            sending_finish() ;      /*送信イベント*/
        }else{
            if(rnd() <= kakuritu){      /*{0,1}を出現確率に基づいて判定*/
                next_ch = 1 ;
                encoder_zero ++ ;   /*エンコーダーに入る前のエントロピー計算*/
            }else{
                next_ch = 2 ;
                encoder_one ++ ;

            }
            present_time = arrival_time ;
            arrival_symbol() ;      /*到着イベント*/
        }
    }

    average_delay = cum_delay / TOTAL_SYMBOL ;


    /*エンコーダに入るときの到着レート*/
    encoder_arrival_rate = (double) (feedback_number + TOTAL_SYMBOL) / present_time) ;




    printf("平均遅延 : %.2f\n" , average_delay) ;
    printf("フィードバックしなかった回数 : %d\n" , nofeedback_code_number) ;
    printf("フィードバックした回数 : %d\n" , feedback_number) ;
    printf("通信路の利用率 : %.2f\n" , total_sending_time / present_time) ;

    printf("\n") ;
    printf("エンコーダの到着レート : %.2f\n" , encoder_arrival_rate) ;


    return(0) ;

    ERROR :
    puts("引数を確認してください") ;

    exit(1) ;
}
```