平成 14 年度　研究開発成果報告書
「高速高品質コンテンツ配信を実現する自律適応型メタコンテンツ・ネットワーク技術に関する研究開発」

目　　次

（添付資料）

平成１４年度　研究開発成果報告書
「高速高品質コンテンツ配信を実現する自律適応型 メタコンテンツ・ネットワーク技術に関する研究開発」

目　　次

１　研究開発課題の背景

近年、インターネットによる映像配信の気運が高まっているが、乱れのない映像配信を実現するためには、誤り訂正技術が重要な役割を占めている。これまでの誤り訂正技術は、有限体における多項式演算を用いた手法が主流であったが、復号化処理が符号長の自乗に比例して増大するため、専用のハードウェアを用いて処理する必要があった。高速高品質のコンテンツ配信を実現するためには、ソフトウェア的に処理可能な負荷の軽い誤り訂正技術によって、ネットワークの状況(誤り率)に応じて自律的に適応できるメカニズムを導入することが、重要であると考えられる。

２　研究開発分野の現状

符号理論の歴史は、1948 年のシャノンの論文、あるいは 1950 年のハミング符号の発明から始まる。1960 年には、複数のビット誤りを訂正できる BCH (Bose-Chaudhuri-Hocquenghem)符号やバイト誤りを訂正できる Reed-Solomon 符号が発明された。最近、米国 Digital Fountain 社では、新しい誤り訂正符号が開発された。この符号は、バイト誤りを訂正するのではなく、パケット(シンボル)そのものの欠損を補うことの出来る符号であり、これまでの訂正という概念ではなく確率的に元のパケット(シンボル)を復元している。また、この符号は処理時間が符号長に比例するという LDPC (Low Density Parity Check)という誤り訂正符号の流れをくみ、ソフトウェア的に復号可能であるという特徴を持っている。この符号を用いて、自律適応型ネットワークを構成する研究は、有意義であると考える。

３　研究開発の全体計画

3-1　研究開発課題の概要

本研究開発は、最新の超高速低遅延順方向誤り訂正技術を活かした新たな信頼性保証型通信手順によって、従来のフロー制御と廃棄パケットの再送要求やラベルスイッチングによって信頼性を確保するネットワークとはまったく異なる、新たなサービスレベル保証型ネットワークの実現を目指すものであり、基盤技術の研究開発と標準化機関への提案および検証試験により構成される。

3-2　研究開発目標

3-2-1　最終目標

（１）自律適応型のフロー毎アプリケーション毎サービスレベル毎の順方向誤り訂正機能選択制御機能を持つ多段ネットワークのモデル化とシミュレーション技術の確立

（２）自律適応型のフロー毎アプリケーション毎サービスレベル毎の順方向誤り訂正機能選択制御機能を持つ多段ネットワークを実現するためのあら

たなIP上の高信頼性コンテンツ配信プロトコルならびにパケット構造案の提唱

（３） 最新の超高速低遅延順方向誤り訂正技術の国際的調査研究と、MPEG-1/2/4等のストリーミング配信やダウンロード配信におけるネットワーク上での多段処理（エンコード・デコード・トランスコード）に適した同技術のクライテリアの確立

（４） デファクト標準化を念頭においた、順方向誤り訂正機能およびその選択制御機能のデバイス化と試作評価プラットフォームとしてのサーバ・クライアントおよびホームゲートウエイ等低速低価格中継装置の試作ならびに実証試験

（５） 実証試験結果に基づく、ＩＥＴＦ等への自律適応型のフロー毎アプリケーション毎サービスレベル毎の順方向誤り訂正機能選択制御機能を持つ多段ネットワークを実現するためのあらたな高信頼性コンテンツ配信プロトコルのドラフト提案

（６） 本研究の信頼性保証型通信手順が、インターネット世界に受け入れられるための、適切な抑制手段に関する研究と提唱。

（７） 中継装置に適した超高速低遅延の順方向誤り訂正アルゴリズムの調査研究とデバイス試作。

（８） キャリア・ＩＳＰ等異業種の参画を得たフィールド実験と有用性の検証

3-2-2　中間目標

（１） MPEG1/2/4等のストリーミング配信やダウンロード配信におけるネットワーク上での処理（エンコード・デコード・トランスコード）に適した順方向誤り訂正機能のクライテリアの明確化と、これに適した最新の超高速低遅延順方向誤り訂正技術ならびに実用化課題に関する調査研究をおこなう。

　最新の超高速低遅延順方向誤り訂正技術の調査研究と動向を把握し、本研究開発におけるネットワーク上の利用に適した１つまたは複数の順方向誤り訂正技術候補の選択を行なう。

ネットワーク上での実用化のための課題（ハードウェアを用いた高速化の必要性、チューニングパラメータ、トレードオフの関係にある特性等）について研究する。更に、独自の超高速低遅延順方向誤り訂正技術の研究開発の可能性について検討を行なう。

（２） 自律適応型のフロー毎アプリケーション毎サービスレベル毎の順方向誤り訂正機能選択制御機能を持つ多段ネットワークのモデル化を行なう。

　自律適応型のフロー毎アプリケーション毎サービスレベル毎の順方向誤り訂正機能選択制御機能を持つ多段ネットワークのモデル化を行なう。

　シミュレーション技術の調査研究を行ない、必要に応じてシミュレーションプログラムの試作を行なう。

（３）自律適応型のフロー毎アプリケーション毎サービスレベル毎の順方向誤り訂正機能選択制御機能を持つ多段ネットワークを実現するためのあらたなIPプロトコルならびにパケット構造案の素案を作成する。

　自律適応型のフロー毎アプリケーション毎サービスレベル毎の順方向誤り訂正機能選択制御機能を持つ多段ネットワークを実現するためのあらたなIPプロトコルならびにパケット構造のクライテリアに関する調査研究を行なう。

　上記クライテリアと既存のプロトコル提案との関連について調査研究を行なう。

（４）デファクト標準化を念頭においた、順方向誤り訂正機能およびその選択制御機能のデバイス化と評価プラットフォームとしてのサーバ・クライアントおよびホームゲートウエイ等低速低価格中継装置の試作を行なう。

3-3　研究開発の年度別計画

研究開発項目	１４年度	１５年度	１６年度	備考
1）　ネットワークのモデル化①とシミュレーション技術の開発②ならびに高信頼性コンテンツ配信プロトコルの研究③	①----→ ②------ ③------	 -------→ -------------	 ------→	①②③の一部を大阪大学に再委託
2）　最新の高速低遅延順方向誤り訂正技術の国際的調査研究④とネットワーク上の多段利用を想定したクライテリアの確立⑤	④----→ ⑤------	 --→		④⑤の一部を京都大学に再委託
3）　デバイス化⑥とホームゲートウエイ等評価プラットフォームの試作⑦、実証試験⑧およびＩＥＴＦ等へのドラフト提案⑨	⑥------ ⑦------	→ ---→ ⑧--------→ ⑨--	 -------------→	
4）　帯域抑制手段に関する研究⑩、中継装置用超高速低遅延順方向誤り訂正デバイスの試作・実験⑪		⑩----→ ⑪--	 ------------→	⑩の一部を大阪大学に再委託

４　研究開発の概要

4-1　研究開発実施計画

4-1-1　研究開発の計画内容

（A）超高速低遅延順方向誤り訂正技術に関するクライテリアの分析と国際的視点からの最新技術の調査研究

（１）ネットワーク上で多段利用可能な順方向誤り訂正技術に関するクライテリアの分析

　インターネット上で、MPEG1/2/4，WMT，Real，QT，MP3等のファーマットの映像・音声・音楽のストリーミング配信や大容量ファイルの高速ダウンロード配信を行なう場におけるサーバ・クライアント・中継装置を想定したOn_the_flyでの順方向誤り訂正エンコード・デコード・トランスコード処理に求められるクライテリアについて整理する。

（２）最新の超高速低遅延順方向誤り訂正技術に関する国際的調査研究

　超高速低遅延順方向誤り訂正技術に関する最新の国際的研究動向を把握し、本研究開発におけるネットワーク上の利用に適した１つまたは複数の順方向誤り訂正技術候補の選択と、実用化のための課題（ハードウェアを用いた高速化の必要性、チューニングパラメータ、トレードオフの関係にある特性等）について研究する。更に、独自の超高速低遅延順方向誤り訂正技術の研究開発の可能性について検討を行う。

（B）モデル化とシミュレーション技術の調査研究

　自律適応型のフロー毎アプリケーション毎サービスレベル毎の順方向誤り訂正機能選択制御機能を持つ多段ネットワークのモデル化を行なう

（１）学習的なフロー毎アプリケーション毎サービスレベル毎の順方向誤り訂正機能選択制御機能を持つ多段ネットワークのモデル化を行なう。

（２）シミュレーション技術の調査研究を行い、必要に応じてシミュレーションプログラムの試作を行う。

（C）プロトコルの調査研究

　自律適応型のフロー毎アプリケーション毎サービスレベル毎の順方向誤り訂正機能選択制御機能を持つ多段ネットワークを実現するためのあらたなIPプロトコルならびにパケット構造案の素案を作成する。

（１）自律適応型のフロー毎アプリケーション毎サービスレベル毎の順方向誤り訂正機能選択制御機能を持つ多段ネットワークを実現するためのあらたなIPプロトコルならびにパケット構造のクライテリアに関する調査研究を行なう。

（２）上記クライテリアと既存のプロトコル提案との関連について調査研究を行なう。

（D）デファクト標準化を念頭においた試作

　デファクト標準化を念頭においた、順方向誤り訂正機能およびその選択制御機能のデバイス化と評価プラットフォームとしてのサーバ・クライアントおよ

びホームゲートウエイ等低速低価格中継装置の試作を行なう。

（１） 順方向誤り訂正機能およびその選択制御機能のデバイス化(プロトタイプ試作)をおこなう。上記デバイスを用いた、ホームゲートウエイ等低速低価格中継装置の試作を行なう。

4-2　研究開発の実施内容

（Ａ） 誤り訂正符号として、Reed-Solomon 等の誤り訂正符号を取り上げ、これらの符号と米国 Digital Fountain 社 Dr. Luby らにより開発された LT Coding について評価検討を行っている。LT Coding は、Low-Density Parity Check の流れをくむ符号であり、復号のアルゴリズムが符号長の一乗に比例する、つまり O(n)であることを特徴としている。現在、広く用いられている Reed-Solomon 符号は、符号長の自乗に比例する O(n^2)であり、復号時間を短縮するために専用ハードウェアを使用している。汎用 CPU でソフトウェア的にデコード処理ができることが、LT Coding の大きな利点である。ソフトウェア的に処理できるため、広範囲のネットワーク誤り率に自律的に適応することが可能で、今回の研究テーマに対応できる符号であると考えられる。

Tarbo 符号は、シャノンの限界に迫る符号として最近注目を集めているが、ビット誤りがパケット欠損に波及しない領域(例えば、物理層)での利用が想定されており、アプリケーション層などでの誤り訂正に適さないと考えられる。さらに LT Coding の改良型である Raptor Coding が考案され、LT Coding において 10^{-8} 程度であった復元失敗率が 10^{-12} 以下に改良されている。

誤り訂正符号では、バースト状の誤りに対して、符号長を長くする等の方法を用いて対応することが可能である。ただ Reed Solomon 符号では、符号長を長くすると、デコードにかかる時間が O(n^2)であるため、実用的ではない。上記の点を数値計算で比較し、IP ネットワークにおけるパケット欠損の修復において、Digital Fountain 社 LT Coding の改良型である Raptor Coding が、有望であるという目処を得ている。

さらにネットワーク上での(実用化のための)トレードオフに関する研究として、Raptor Coding において、ネットワーク上のパケット欠損として二項分布を仮定し、帯域オーバヘッドを考慮した誤り訂正能力と遅延時間に関する検討を行った。

（Ｂ） 多段ネットワークのモデル化を行う第一歩として、一段のネットワークモデルを考え、このシミュレーションを行っている。ネットワーク上にサポートサーバと呼ばれる機器を設置し、このサーバによってネットワーク上で発生したパケット損失を補うというモデルである。この結果を、電子情報通信学会全国大会で発表した。

現在のシミュレーションモデルは、UDP プロトコルに限定したモデルであるが、将来的には TCP/IP と混在した環境でシミュレーションを行う可能性も、検討している。

今後、これらのモデルにプロトコルを組み込んだ形での改良を行い、最終形態である自律適応型のフロー毎アプリケーション毎サービスレベル毎に必要な誤り訂正方法の検討を行う。

（C） 誤り訂正機能を持つサーバ、持たないサーバ、誤り訂正機能を持つクライアント、持たないクライアントが共存するヘテロジーニアスなネットワーク環境において、自律適用的にサーバ・クライアント間で誤り訂正機能のネゴシエーションを実施するための基本的なプロトコルの必要要件に関して検討を進め、ＲＴＰ／ＲＴＳＰをベースに機能拡張を行なう方向で新たな通信制御プロトコルの素案を検討した。

（D） Digital Fountain 社から、LT Coding を改良した Raptor 符号化／復号化ソフトウェアモジュールを導入し、上記誤り訂正機能・選択制御機能とネットワークインタフェースを組み込んだデバイスを開発し、評価プラットフォームとして最新の MPEG2 および MPEG4 のエンコード・デコード機能を持つＳＴＢ（セットトップボックス）を試作した。

また、誤り訂正機能を持つサーバの試作として、市販ＶＯＤサーバへの上記誤り訂正符号化・復号化ソフトウェアモジュールの実装試作を行なった。さらに、誤り訂正機能を持たないサーバに対して、ネットワーク上で誤り訂正機能を選択的に付与する装置（ライブ FEC サーバ）の試作開発を進めるため、必要なハードウェア構成の検討・導入とソフトウェアモジュールの設計を進めた。

５　研究開発実施状況

5-1　超高速低遅延順方向誤り訂正技術に関するクライテリアの分析と国際的視点からの最新技術の調査研究

5-1-1　序論

インターネット上で、MPEG1/2/4、WMT、Real、QT、MP3等のファーマットの映像・音声・音楽のストリーミング配信や大容量ファイルの高速ダウンロード配信を行なう場におけるサーバ・クライアント・中継装置を想定したOn_the_flyでの順方向誤り訂正エンコード・デコード・トランスコード処理に求められるクライテリアについて整理する。

超高速低遅延順方向誤り訂正技術に関する最新の国際的研究動向を把握し、本研究開発におけるネットワーク上の利用に適した１つまたは複数の順方向誤り訂正技術候補の選択と、実用化のための課題（ハードウェアを用いた高速化の必要性、チューニングパラメータ、トレードオフの関係にある特性等）について研究する。更に、独自の超高速低遅延順方向誤り訂正技術の研究開発の可能性について検討を行なう。

5-1-2　研究内容

誤り訂正符号として、Reed-Solomon 等の誤り訂正符号を取り上げ、これらの符号と米国 Digital Fountain 社 Dr. Luby らにより開発された LT Coding について評価検討を行っている。LT Coding は、Low-Density Parity Check の流れをくむ符号であり、復号のアルゴリズムが符号長の一乗に比例する、つまり $O(n)$

であることを特徴としている。現在、広く用いられているReed-Solomon符号は、符号長の自乗に比例する$O(n^2)$であり、復号時間を短縮するために専用ハードウェアを使用している。汎用 CPU でソフトウェア的にデコード処理ができることが、LT Coding大きな利点である。ソフトウェア的に処理できるため、広範囲のネットワーク誤り率に自律的に適応することが可能で、今回の研究テーマに対応できる符号であると考えられる。

Tarbo符号は、シャノンの限界に迫る符号として最近注目を集めているが、ビット誤りがパケット欠損に波及しない領域(例えば、物理層)での利用が想定されており、アプリケーション層などでの誤り訂正に適さないと考えられる。さらにLT Codingの改良型であるRaptor符号が考案され、LT Codingにおいて10^{-8}程度であった復元失敗率が10^{-12}以下に改良されている。

誤り訂正符号では、バースト状の誤りに対して、符号長を長くする等の方法を用いて対応することが可能である。ただReed-Solomon符号では、符号長を長くすると、デコードにかかる時間が$O(n^2)$であるため、実用的ではない。上記の点を数値計算で比較し、IPネットワークにおけるパケット欠損の修復において、Digital Fountain社のRaptor符号が、有望であるという目処を得ている。

さらにネットワーク上での(実用化のための)トレードオフに関する研究として、Raptor符号において、ネットワーク上のパケット欠損として二項分布を仮定し、帯域オーバヘッドを考慮した誤り訂正能力と遅延時間に関する検討を行った。

なお本研究に関して、基礎的な部分は京都大学・高橋研究室に再委託を行った。再委託の研究成果報告書は、本報告書の最後に添付する。

5-1-3　誤り訂正符号の基本的な考え方と復号化能力

5-1-3-1　はじめに

符号理論の歴史は、1948年のシャノンの論文、あるいは1950年のハミング符号の発明から始まる。1960年には、複数のビット誤りを訂正できるBCH(Bose-Chaudhuri-Hocquenghem)符号やバイト誤りを訂正できるReed-Solomon符号が発明された。今回取り上げるDigital Fountain社の誤り訂正符号(LT Codeならびに、その改良型であるRaptor code)は、パケット(シンボル)そのものの欠損を補うことの出来る符号であり、これまでの訂正という概念ではなく確率的に元のパケット(シンボル)を復元できるという方法である。

5-1-3-2　Sequential送信方式とRandom送信方式

伝送中の欠損パケット(シンボル)を補う方法としては、以下の２つに分類される。

（１）ARQ (Automatic Repeat reQuest)

（２）FEC (Forward Error Correction)

今年度は、この ARQ (Automatic Repeat reQuest)を利用しない方法で、受信側(Forward 側)で発生したパケット(シンボル)の欠損を補うという Forward

Error Correction (FEC)方式を中心に研究した。この方式は、送信側で冗長なコードをつけることによって、伝送路でビットエラーやパケット(シンボル)欠損が発生しても、受信側で損失したパケット(シンボル)を回復できる方法である。この方法は、遅延の少ない高信頼性送信を可能にするが、本来の情報部分に冗長部分を付与する必要があるため、余分にデータを送信する必要がある。このため「受信時間が長くなる」「余分な伝送帯域が必要となる」などの問題が発生する。

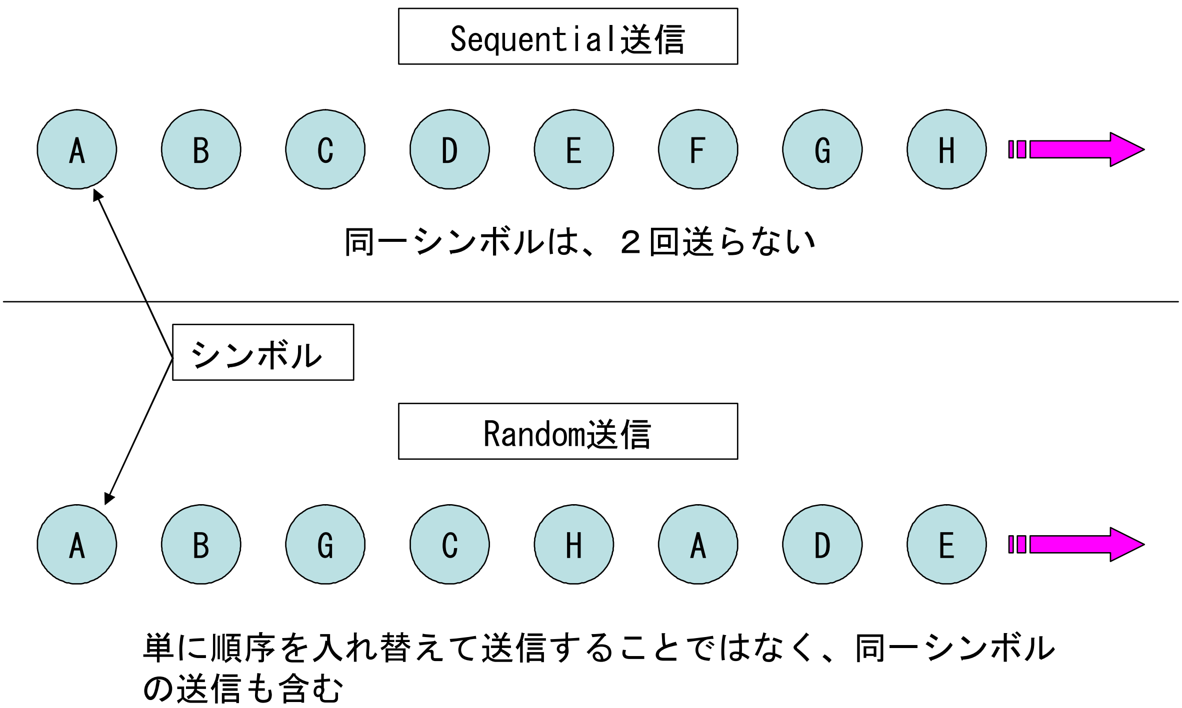


図 5-1-3-1　データの伝送方法

データ通信では、図 5-1-3-1(上)に示す様に、(当然のことながら)必要となるデータシンボルを本来の順番通りに相手側に送る(Sequential 送信方式)。本章で述べる「シンボル」とは、データを送り出す時の単位で、基本的には「パケット」と同様である。しかし今回の LT code 等の基本的な考え方は、図 5-1-3-1(下)に示す様に、ランダムに選ばれたシンボルを順不同に送信するという方式を取っている(Random 送信方式)。あたかもサイコロを振って、シンボルを選んで送信するようなものである。Sequential 送信では、同じシンボルは(１周期中に)１回しか送出されないが、Random 送信では、確率的に同じシンボルを２回以上送信することもあり得る。

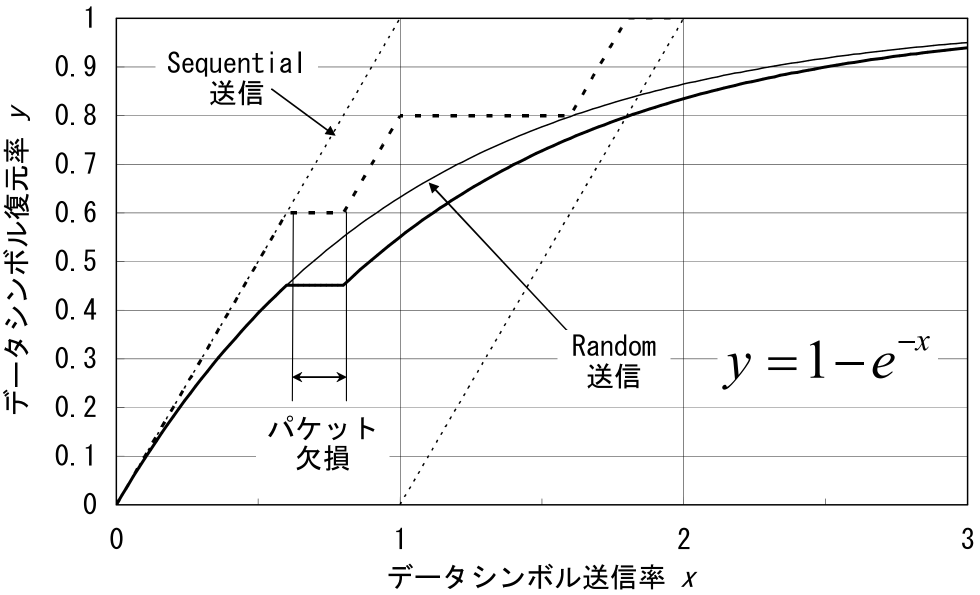


図 5-1-3-2　データシンボルの復元率

図5-1-3-2に示すようにSequential送信方式では、100%のデータシンボルを送信すれば、(伝送路にシンボルの欠損がない場合)受信側で100%のデータを復元できる。しかし、伝送路でシンボル(パケット)欠損が発生すると、その部分だけ、元のシンボルを復元(受信)できなくなる。TCP/IPを用いた伝送では、ARQ (Automatic Repeat reQuest)と呼ばれる自動反復要求(受信パケットに問題を検出した時に、再送を要求する通信方法)によって、欠損したシンボルが再送信される。この方法は、シンボル(パケット)に特別の工夫が不要で、パケットオーバヘッドを回避できるという長所がある。しかし欠損が発生した場合、修復までデータシンボルが１往復する時間だけ、遅延が発生するという欠点を持っている。

受信側が、どのシンボルが欠損したかを送信側に伝えることが出来ない場合、すべてのデータシンボルを再送する必要がある。図5-1-3-2のSequential送信方式では、シンボル欠損が発生すると、全データの再送を開始し、欠損したシンボルが受信側に届いた時点で、伝送完了となる。従って、最初のシンボルが欠損した場合は、再送するデータが少なくてすむが、最後のシンボルが欠損した場合は、２倍のデータを送信する必要がある。一方Random送信では、シンボルの送信数xとデータの復元数yの間に、

$$y = 1 - e^{-x}$$

という関係がある。シンボル欠損が発生した場合、送信シンボル自体がランダムに送信されているので、途中で発生するランダムなシンボル欠損に対しても、この曲線の形状は変化しない。つまり、どのシンボルが欠損しても、このグラフが(欠損したシンボル数に応じて)右にシフトするだけであることが分かる。伝送路で発生するランダムなシンボル欠損によって、送信数－復元数の基本的

な関係は変わらないというのが、Random 送信の大きな特徴である。

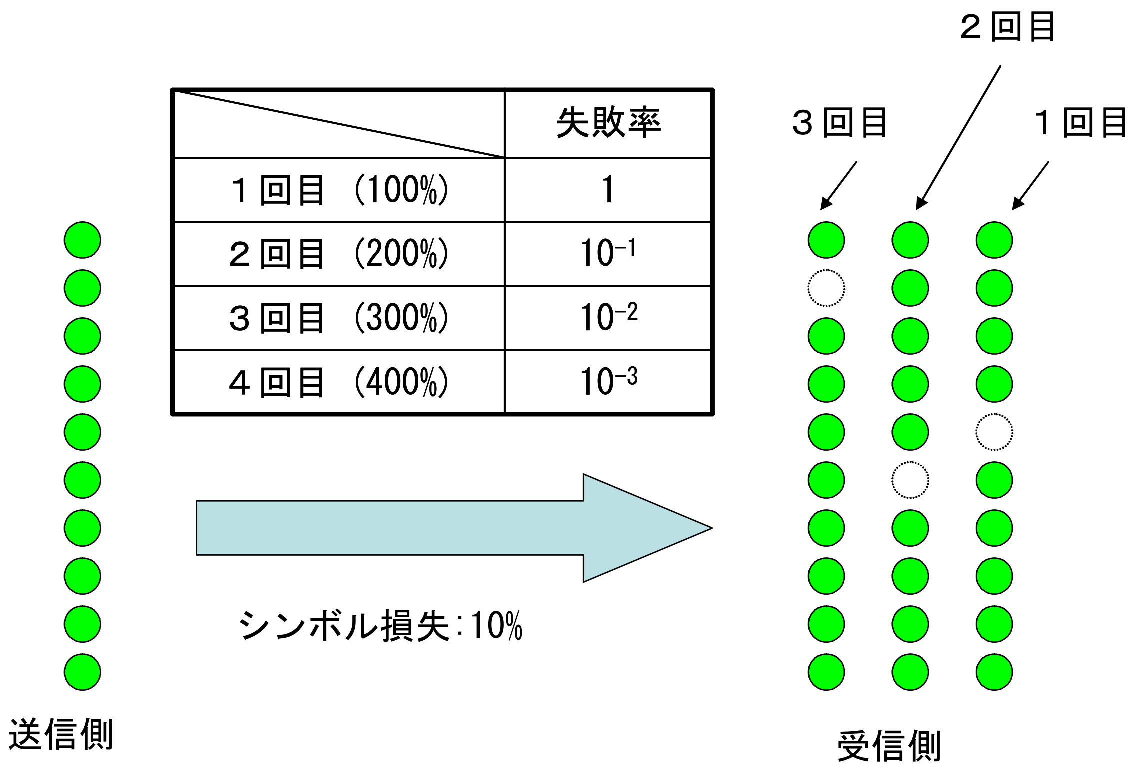


図5-1-3-3　もっとも単純なForward Error Correction

Sequential送信では、伝送途中で発生したシンボル欠損を補うために、データシンボルの送信が一巡した後、さらに同じデータシンボルを送信する。この方法による伝送失敗確率の計算例を、図5-1-3-3に示す。伝送路におけるシンボル欠損率が10%の場合、10個のシンボルを送信すると、平均的に9個のシンボルを受信することが出来る。平均的に見れば、必ず1個のシンボルが欠損するため、必ずデータ伝送に失敗(失敗率：100%)することになる。受信側から送信側へは、ARQの様な情報のフィードバックがないため、送信側は受信側でどのパケットが欠損したかを知ることは出来ない。このため、欠けたシンボルを復元するために、再度全シンボルを送信する。この際、前回の送信で欠損したシンボルが受信できれば伝送成功となるが、10%の確率で前回と同じシンボルが欠損するため、同じデータを一巡(送信)しても、受信側での成功率は100%とならない。つまり、200%のデータシンボルを送信しても、失敗率は10%(10^{-1})となる。

さらに図5-1-3-3に示すように、300%, 400%と送信シンボル数を増加させる毎に、失敗率は1/10に低下することが分かるが、送信シンボル数を増やしても失敗率は0%になることはない。ちなみにパケット(シンボル)伝送の失敗率が、このように0%にならないことは、FEC方式固有の現象ではない。ARQ方式においても、再送されるパケット(シンボル)自体にも欠損が発生する確率があるため、パケット(シンボル)欠損のある伝送路に対して失敗率は0%にならない。

5-1-3-3　Digital Fountain社誤り訂正符号の基本概念

今回我々が検討するDigital Fountain(DF)社の誤り訂正符号は、前節のような確率的な概念に基づいて作成されており、元のデータを確実に復元できるわけではなく、非常に低い確率ではあるが復元失敗率が存在する。つまり、何%かのシンボルを余分に送ることによって、受信側で(一定の失敗率以下で)欠損シンボルを復元する方法である。一方、Reed-Solomon等の誤り訂正符号は、訂正可能な誤り数以下であれば確実に誤りを訂正できる。

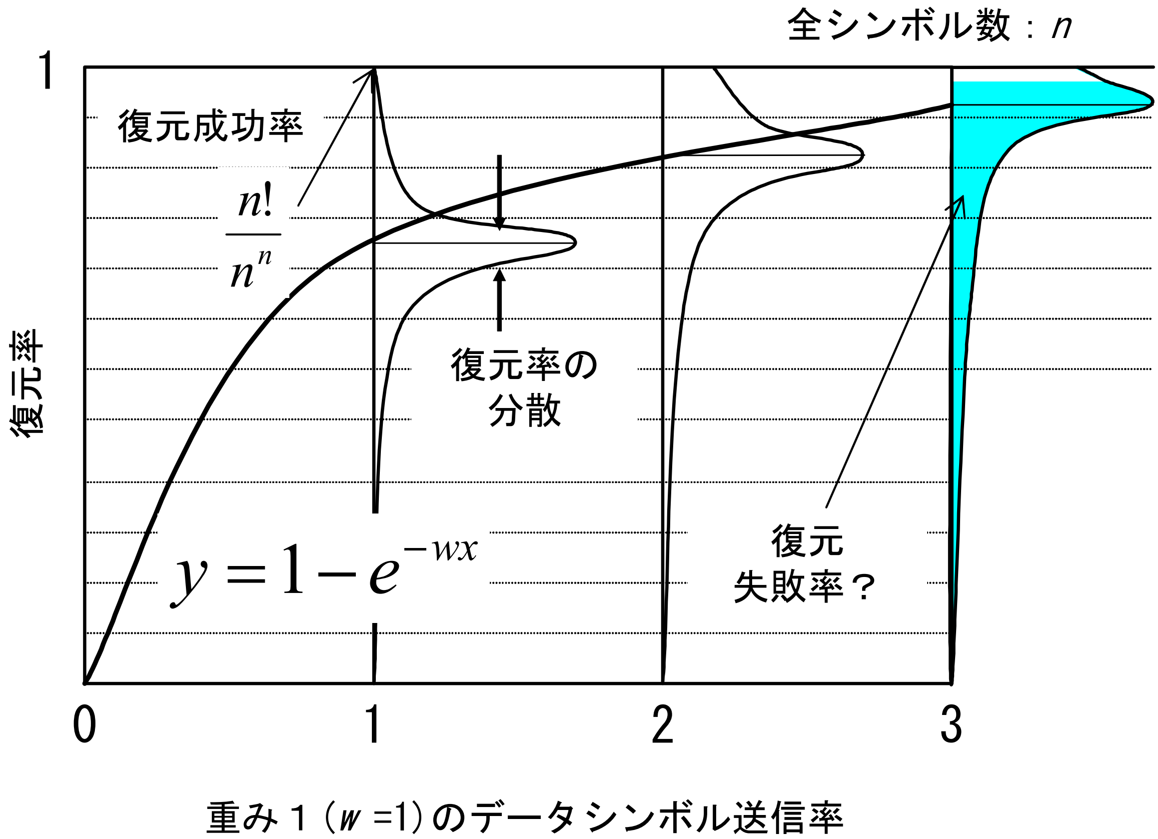


図5-1-3-4シンボル送信による復元率の分散

上記のような確率的な送信では、シンボル受信の成功率が100とはならないため、「無限の時間をかけてもすべてのデータを受信できない」という印象を受けることがある。シンボルが無限にある場合には、この考え方が成立するが、実際には有限個のシンボルを取り扱うため、復元率は図5-1-3-4に示すように、一定の分散をもつことになる。つまり100%のシンボルを受信した場合、1-1/eしかシンボルが復元できないような気がするが、実際には$n!/n^n$の確率で全てのシンボル送信に成功する。

DF社の誤り訂正符号では、基本的な概念は上記の通りであるが、実際には色々な工夫がなされている。元のシンボルをランダムに送信するだけでは、元のデータを復元するために送るデータシンボル数が、非常に多くなる。例えば、100%のデータシンボルを受信しても、平均的に63.2%の元データしか復元することは出来ない。

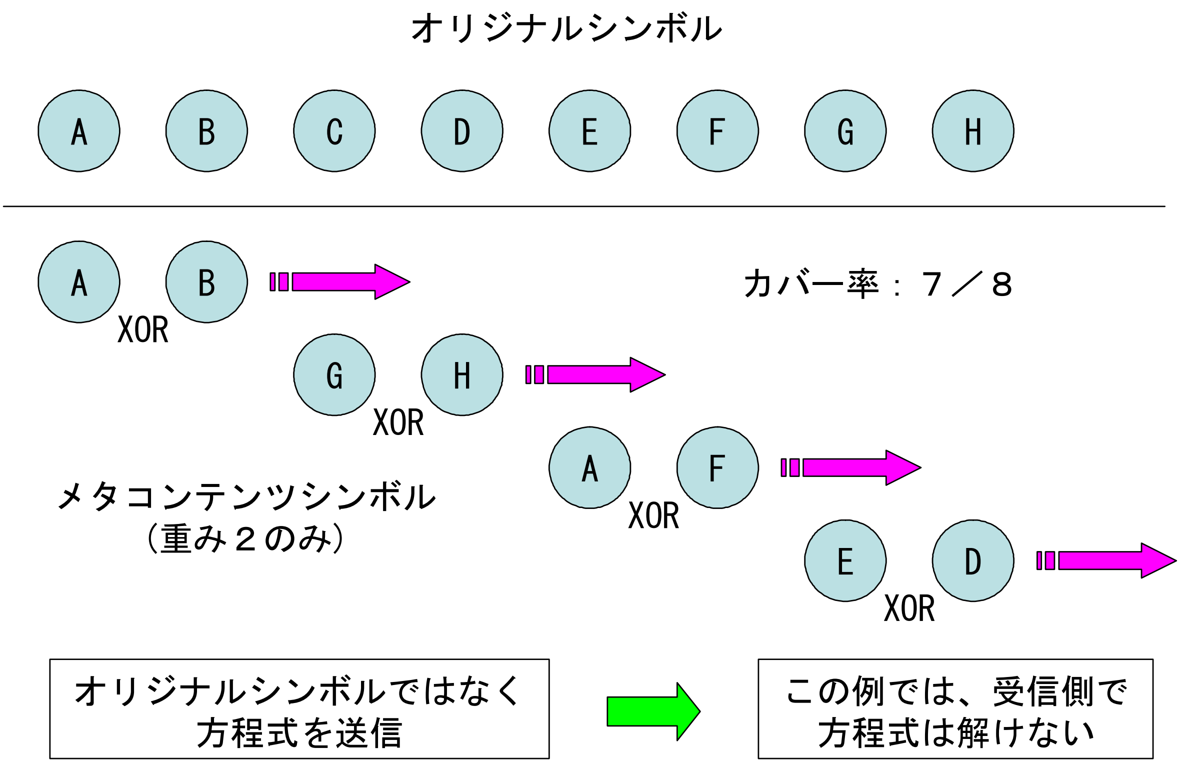
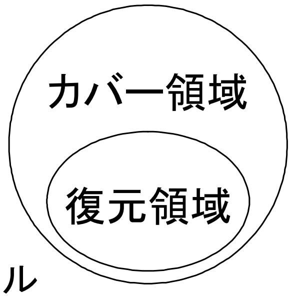


図5-1-3-5　メタコンテンツでの送信方法

そこで図5-1-3-5に示すように、2つのシンボルの排他的論理和(XOR)を取って、このシンボル(メタコンテンツシンボル)を相手側に送る。この場合、メタコンテンツの重みは2であり、何も演算しないで送信する場合は、重み1のメタコンテンツシンボルとなる。この様に、複数のコンテンツから生成されたメタコンテンツシンボルは、受信側で再度排他的論理和演算を行うことによって、元のデータシンボルを復元できる。つまり方程式を順次送信し、それを受信側で解くことに相当する。従って、「受信した方程式の数」と「復元できたデータシンボル数」の関係は、状況によって異なり、一意に決めることは出来ない。この関係を明確にするために、「復元率」「カバー率」という2つの概念を導入する。図5-1-3-5の例では、全シンボル数が8個であり、C以外の7個のシンボルがカバーされているので、カバー率は7/8となる。



復元領域⊆カバー領域

復元率
メタコンテンツシンボルによって復元されたシンボル

カバー率
メタコンテンツシンボルで参照されているシンボル

メタコンテンツの重み	
全て１	全て２
復元率＝カバー率	復元率≠カバー率
メタコンテンツシンボルは全て復元	復元率＝０

図5-1-3-6　復元率とカバー率の関係

この定義では、復元率はカバー率より小さいか等しく、集合論的に言えば、カバー領域は復元領域を包含している。図5-1-3-6に示されるように、メタコンテンツの重みが全て1の場合は、復元率＝カバー率となるが、これ以外の場合は、両者の値は異なる。ちなみに、全てのメタコンテンツシンボルの重みが2の場合は、元のシンボルを復元することは出来ない(詳細は、後述の表5-1-3-2参照)。

図5-1-3-7は、メタコンテンツの(全ての)重みを変化させた場合の、メタコンテンツシンボル送信率xと受信シンボルカバー率yの関係である。平均重みをwとした場合には、

$$y = 1 - e^{-wx}$$

という関係が成立し、重みを増加させることによって、カバー率は急速に立ち上がることが分かる。つまりメタコンテンツの重みを増加させると、カバー領域は100%に近づき、元のシンボル復元の必要条件を満たしていることが分かる。

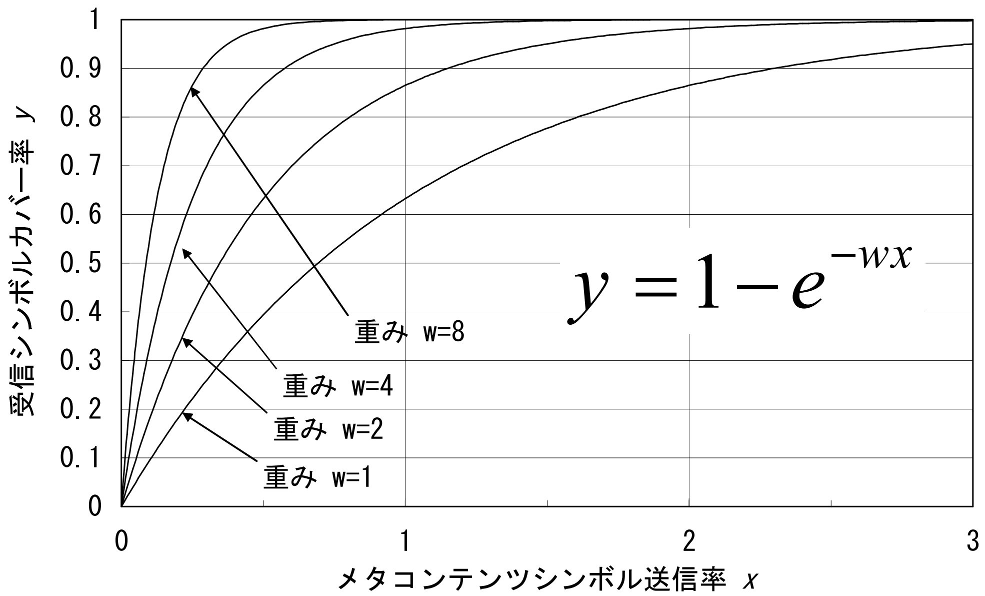


図5-1-3-7　メタコンテンツの重みを変化させた場合の復元率

図5-1-3-8は、集合論的に見た各領域の関係を示したものである。1から無限大までの重みを含んだメタコンテンツを送信した場合、受信側でのシンボル復元は、集合論的に4つの領域に分類される。

（１）重み1のメタコンテンツでのみカバーされる領域(復元可)

（２）重み1と重み2以上のメタコンテンツ両方でカバーされる領域(復元可)

（３）重み2以上のメタコンテンツのみでカバーされる領域(復元？)

（４）どのメタコンテンツからもカバーされない領域(復元不可)

(２)と(３)の領域をまたがって参照しているメタコンテンツシンボルがあれば、そのメタコンテンツによって、(３)の領域での元のシンボルを復元することが可能となる。この様な復元方式は、連鎖反応コード(Chain Reaction Code)と呼ばれている。

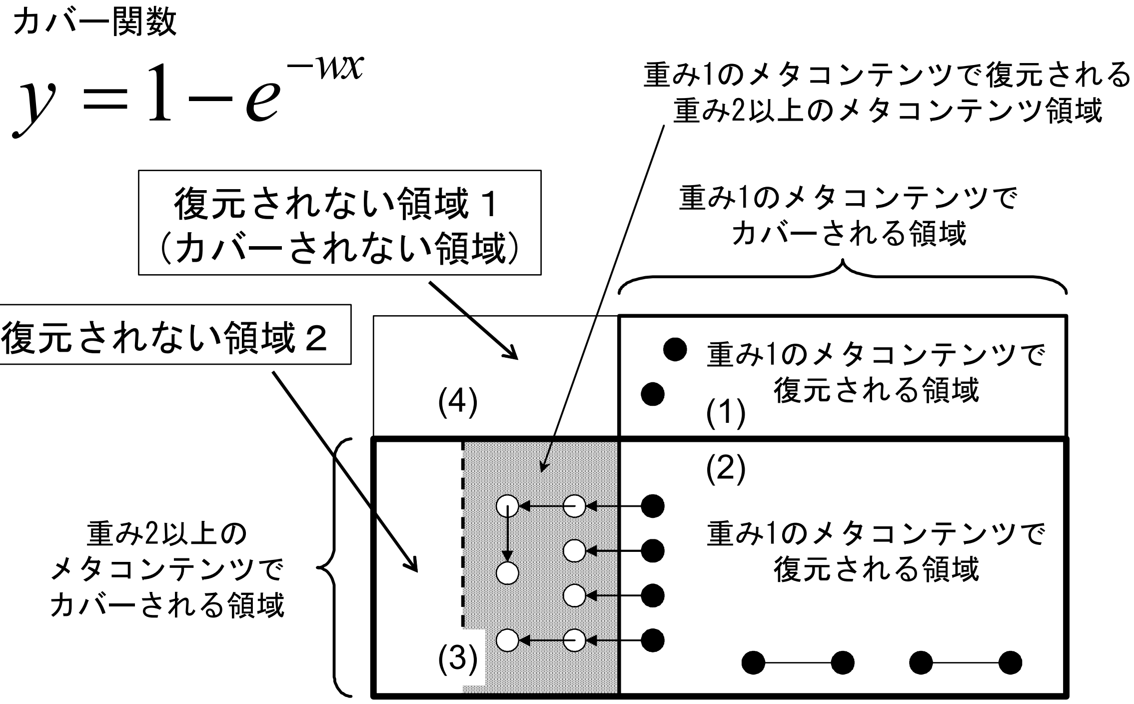


図5-1-3-8　集合論的に見た各領域の関係

また図5-1-3-8に示すように、元のシンボルが復元されない領域は、2つに分類される。

＜復元されない領域１＞　　　　　　　　　　　　　上記（４）に対応する領域

復元されない領域１の減少方法としては、「重み1のメタコンテンツを増加させる」方法と「重み2以上のメタコンテンツを増加させる」方法が考えられるが、図5-1-3-7に示すように関数値の増加が早い「重み2以上のメタコンテンツを増加させる」方法が有利であると考えられる。

＜復元されない領域２＞　　　　　　　上記（３）のうち連鎖反応で復元されない領域

復元されない領域2の減少方法としては、シンボルによる連鎖反応が促進するように、重みの分布（重み関数）を最適化する必要がある。つまり、重み1のメタコンテンツシンボルを受信した場合に、復元できる元のシンボル数が多いほど良いことが分かる。

5-1-3-4　Digital Fountain社誤り訂正符号の符号化／復号化

図5-1-3-9に、Digital Fountain(DF)社で開発されたLT codingのアルゴリズム(送信側)を示す。この例では、入力データである元のシンボル数は、a～hの8個で、一定数の列を発生するマトリックスによって選ばれたシンボル同士で排他的論理和を取り、これをメタコンテンツとして送信する。平均的な重みは、受信側での復元率が高まるように設定する必要がある。またこの演算の関係は、図5-1-3-9(右)に示すように、グラフ表示することも可能である。入力シンボルは白丸で表され、出力シンボルは黒丸で表される。例えば、a xor gという出力シンボルは、入力シンボルaとgとに繋がっており、この両者の排他的論理和で構成されていることを示している。また、この接続線の数は、「排他的論理和演算を行った回数＋受信メタコンテンツ数」を示しており、符号化／復号化に必要な処理時間と関係していることが分かる。

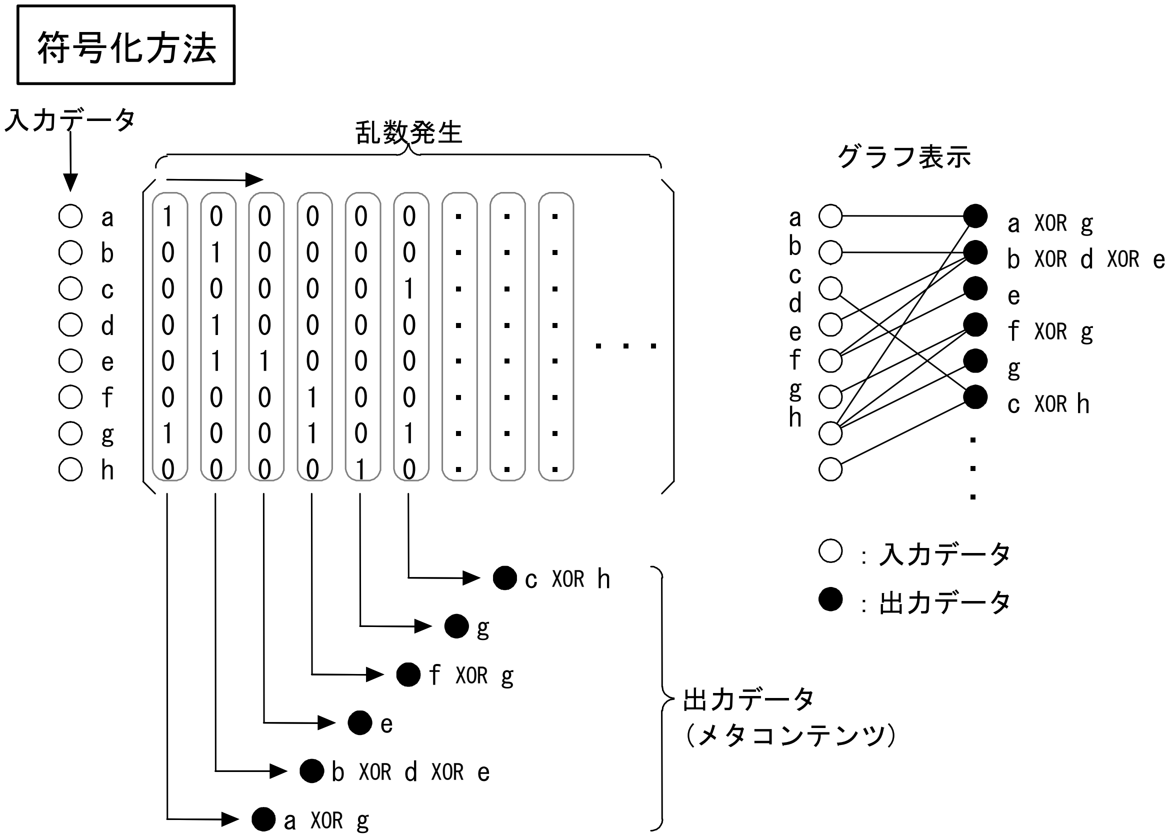


図5-1-3-9　LT codingの符号化アルゴリズム

図5-1-3-10は、受信側での復号アルゴリズムである。図5-1-3-9で生成されたメタコンテンツシンボルのうち、最初の2つを受信した状態では、どの入力シンボルも復元することは出来ない[ステップ１]。ここで、重み1のシンボルであるeを受信すると、元のシンボルeを復元することが出来ると同時に、eというシンボルを含んだ他のメタコンテンツシンボルの重みを1つ下げることが出来る。この例では、b xor d xor eというメタコンテンツを、b xor d xor e xor e=b xor dという演算によって、b xor dにすることが出来る[ステップ２]。また排他的論理和演算によって、接続線の数は減少する。さらに、gという重み１のメタコ

ンテンツを受信することによって、gを参照しているa xor g, f xor gという２つのメタコンテンツから、aならびにfを復元することが出来る[ステップ４，５]。この処理を続けることによって、最終的に全てのシンボルを復元することが可能となる。ただこれは確率的な処理であり、何個のメタコンテンツを受信する必要があるかは、一意には決まらない。メタコンテンツは方程式であるため、少なくとも元のシンボル数以上のメタコンテンツシンボルを受信する必要があるが、その中には同じメタコンテンツシンボルを2回以上重複して受信している場合があり、必要数は確率的に決定される。

復号化方法

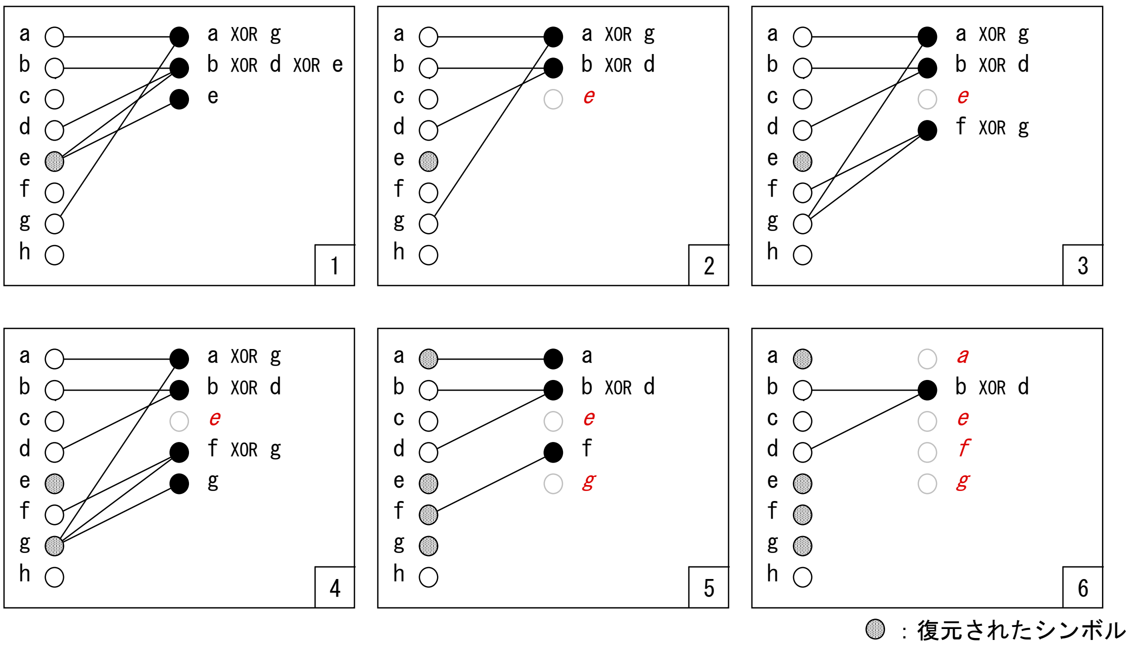


図5-1-3-10　LT codingの復号化アルゴリズム

表 5-1-3-1 に示すように、メタコンテンツの重複受信は「明示型」と「暗黙型」の 2 つに分類できる。明示型の場合は、明らかに同じメタコンテンツを受信する場合であり、重みが大きくなるほど受信確率は小さくなる。一方暗黙型は、A xor B と B xor C を受信している状態で、C xor A というメタコンテンツを受信しても、以前の 2 つのメタコンテンツから導出できるので、意味のないメタコンテンツとなる。受信したメタコンテンツが参照したシンボルを 1 で示し、参照されないシンボルを 0 で示すマトリックス(図 5-1-3-9)で表現すると、その階数(rank)が元のシンボル数に等しくなった時、元のシンボルを復元できることになる。

表 5-1-3-1　メタコンテンツの重複受信

MC シンボル 重複受信の型	内　容
明示型 (Explicit)	同一MCシンボルを、２回受信 A 受信→再度A 受信 A xor B xor C 受信→再度 A xor B xor C 受信
暗黙型 (Implicit)	同一MCシンボルではないが、受信側にとって 意味のないMCシンボルを受信 A xor B, B xor C 受信→A xor C 受信 A xor B xor C, B xor C 受信→A 受信

MC：メタコンテンツ

表 5-1-3-2 は、復元のためのアルゴリズムを示している。シンボルの重みが全て 1 の場合、復元操作を行うことなく元のシンボルを復元できる。すべてのメタコンテンツの重みが 2 以上の偶数の場合、メタコンテンツ同士の演算を繰り返しても、最低の重みは 2 以上となり、元のシンボルを復元することは出来ない。重みが 3 以上の奇数の場合、メタコンテンツによっては、元のシンボルを復元することが可能となる。重みが 1 以上のメタコンテンツシンボルが混じった状態で受信した場合は、3 つのアルゴリズムが考えられる。

(１)重み１のメタコンテンツ受信を待って、復元のトリガとする

重み 1 のメタコンテンツを受信することによって、連鎖反応が進み、元のシンボルを復元する。

(２)復元シンボルと受信メタコンテンツシンボルを比較し、重みを減少させる

内部で復元されたシンボルと受信したメタコンテンツを比較し、受信したメタコンテンツの重みを減少させ、重み１になった場合は、復元処理を開始する。

(３)メタコンテンツ同士を比較し、重みを減少させる

他のメタコンテンツコンテンツシンボルを包含しているようなメタコンテンツシンボルを検索し、相互に重みを減少させる。

表 5-1-3-2　メタコンテンツ復元のためのアルゴリズム

<table>
<tr><th>シンボルの重み</th><th colspan="2">復元方法</th></tr>
<tr><td>全て１</td><td colspan="2">復元不要（受信と同時に復元）</td></tr>
<tr><td>全て２</td><td colspan="2">復元不可能（重み１生成不可）
A xor B, B xor C→A xor C</td></tr>
<tr><td>全て奇数
（３以上）</td><td colspan="2">A xor B xor C, B xor C xor D, A xor C xor D
→A xor D, A xor C xor D→C</td></tr>
<tr><td>全て偶数
（４以上）</td><td colspan="2">復元不可能（重み１生成不可）
A xor B xor C xor D, B xor C xor D xor E
→A xor E</td></tr>
<tr><td rowspan="3">重み１-L
(1,2,3,4‥)</td><td>受信</td><td>重み１シンボルの受信を待つ</td></tr>
<tr><td rowspan="2">内部
生成</td><td>シンボルとMCシンボルを比較して重みを減少
B, C：既知, A xor B xor C 受信</td></tr>
<tr><td>MC シンボル同士を比較して重みを減少</td></tr>
</table>

MC：メタコンテンツ

表 5-1-3-3 は、シンボルが「A」「B」「C」の３つで、重みがすべて２のシンボルを送った場合の、メタコンテンツデータの組合せである。簡単のために、シンボル長は 1bit としてある。この図に示されるように、メタコンテンツのデータは縮退し、このメタコンテンツシンボル情報のみでは元のコンテンツを復元できないことが分かる(例：(A, B, C)=(0, 0, 0)と(1, 1, 1)を区別できない)。

表 5-1-3-3　メタコンテンツの内容(シンボル数：３，重み：すべて２)

A	B	C	A xor B	B xor C	C xor A
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	1	0
0	1	1	1	0	1
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	0	1	1
1	1	1	0	0	0

5-1-3-5　重み分布によるメタコンテンツ受信数

メタコンテンツの重みは、一定の値ではなく、重み 1 からある有限個数までを一定の比率で配分することが有効である。今回、特定の重み分布に対して、どの程度余計にメタコンテンツシンボルを受信すればよいかを、シミュレーションを行った。

表 5-1-3-4 は、重み関数の分布を示したものである。基本的な関数として、

$$\frac{1}{n-1}-\frac{1}{n}$$

を用いている。この関数は、2 から無限大までの総和は、 1 となる。

$$\sum_{i=2}^{\infty}\left(\frac{1}{i-1}-\frac{1}{i}\right)=1$$

今回のシミュレーションに用いた重み関数は、係数 L, r, q を変化させることによって、重み関数の分布を変更することが出来る。

表 5-1-3-4　重みの確率分布

重み	発生確率
1	$[1-r+\{r/(L-1)\}]*q$
2	$\{(1/1)-(1/2)\}*r$
3	$\{(1/2)-(1/3)\}*r$
4	$\{(1/3)-(1/4)\}*r$
・ ・	・ ・
L-3	$\{1/(L-4)-1/(L-3)\}*r$
L-2	$\{1/(L-3)-1/(L-2)\}*r$
L-1	$\{1/(L-2)-1/(L-1)\}*r$
L	$[1-r+\{r/(L-1)\}]*(1-q)$

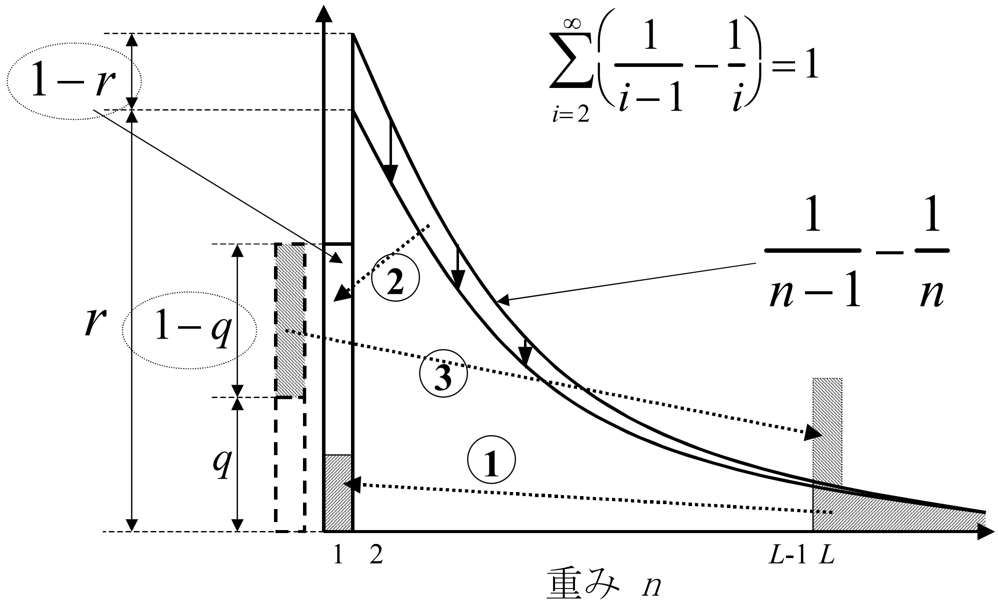


図 5-1-3-11　重み関数の意味

図 5-1-3-11 は、表 5-1-3-4 に示された重み関数の意味を示したものである。重み関数は、無限大までの重みを取るのではなく、一定の値(L-1)で打ち切り、打ち切られた部分を、重み 1 の発生確率に割り当てる。さらに、重み 1 の発生確率を変化させるために、重み関数の 2 から L-1 の部分に係数 r をかけて減少させ、余った部分を重み 1 に割り当てる。さらに、最大重み L の発生確率を変化させるために、重み 1 の発生確率を q: (1-q) の比率で最大重みに割り当てる。この関数を用いて、メタコンテンツを復元するのに必要なメタコンテンツシンボル数をシミュレーションにより算出した。

表 5-1-3-5　カバー率と重みの関係 (r=0.9, L=255)
(元のシンボルと同数のメタコンテンツシンボルを受信した場合)

q	重み 2 以上で カバーされない領域	平均重み	重み 1 の割合	重み 255 の割合
1.0	$4.081*10^{-3}$	5.60	10.4%	0%
0.9	$2.911*10^{-4}$	8.23	9.3%	1.0%
0.7	$1.482*10^{-6}$	13.50	7.2%	3.1%
0.5	$7.540*10^{-9}$	18.75	5.2%	5.2%
0.3	$3.837*10^{-11}$	24.01	3.1%	7.2%
0.1	$1.954*10^{-13}$	29.27	1.0%	9.3%
0.0	$1.388*10^{-14}$	31.90	0%	10.4%

シミュレーションを実施する前に、パラメータ q を変化させて、図 5-1-3-8 の「復元されない領域 1」がどの程度になるかを計算した。パラメータ q を変化させることによって、重み 1 と最大重みの割合ならびに平均重みを求め、カバーされない領域を算出したのが、表 5-1-3-5 である。平均重みを増加させると、カバーされない領域が低下することが分かる。図 5-1-3-12 は、平均重みとカバーされない確率との関係を示したものである。縦軸を対数表示すると、両者の関係は、直線で表されることが分かる。

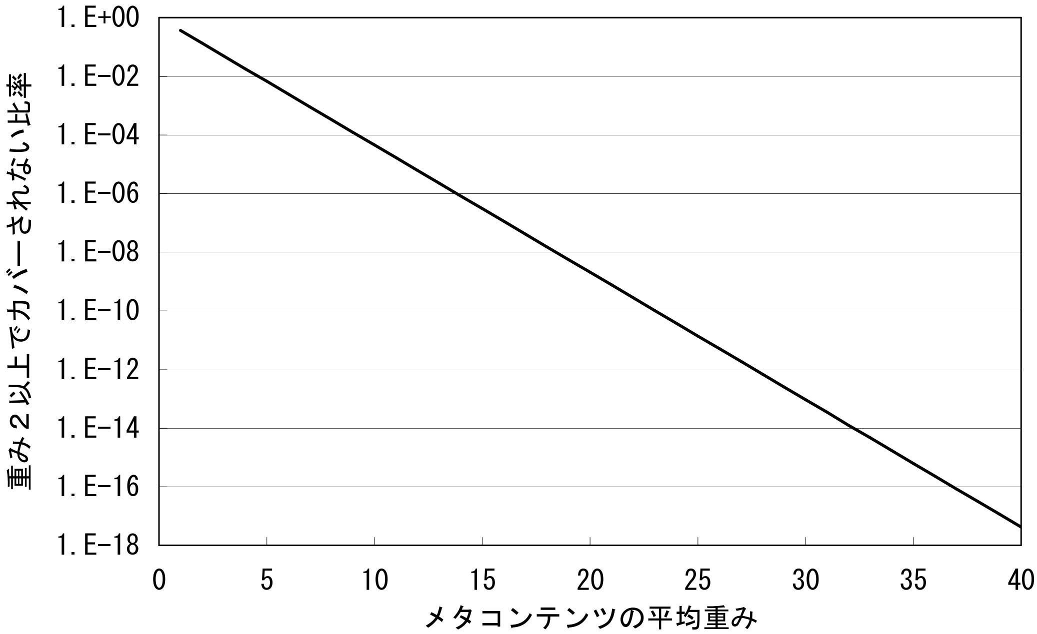


図 5-1-3-12　平均重みとカバーされない比率との関係

5-1-3-6　シミュレーションの結果

図 5-1-3-13 は、シミュレーションにおいて生成されたメタコンテンツシンボルと、復元されたメタコンテンツシンボルの関係を示したものである。元のシンボル数は 65,536 個で、最大重みは 255 である。シミュレーションの回数は、それぞれ１回であるが、パラメータの絞り込みを厳密に行うためには、数百回以上の試行が必要だと思われる。

図 5-1-3-13 でプロットが途切れているところで、元のシンボルが復元できたことが分かる。図 5-1-3-13 では、メタコンテンツシンボル数を元のシンボル数より 1.2～1.8 倍程度生成しないと､元のシンボルを復元できないことが分かる。この様に、元のシンボルを復元するために余分に必要なメタコンテンツシンボル数を、オーバヘッド・メタコンテンツシンボル数と呼ぶ。表 5-1-3-5 に示されるように､q の値を小さくすると､重み１の発生割合が小さくなる。図 5-1-3-13 では、q の値を小さくすると、65,536 個(100%)のシンボルを受信した場合の復元率が小さくなっていることが分かる。しかし、65,536 個(100%)のシンボルを受信した後、急激に復元率が高まり、元のシンボルを完全に復元するのに必要なオーバヘッド・メタコンテンツシンボル数が減少していることが分かる。つまり、重み 1 のメタコンテンツを減少させた方が、少ないオーバヘッドで、元

のシンボルを復元できることが分かる。ちなみに、r=0.89, q=0.1, L=255 の場合、重み 1 のメタコンテンツシンボル発生割合は、表 5-1-3-4 より 1.14%となる。

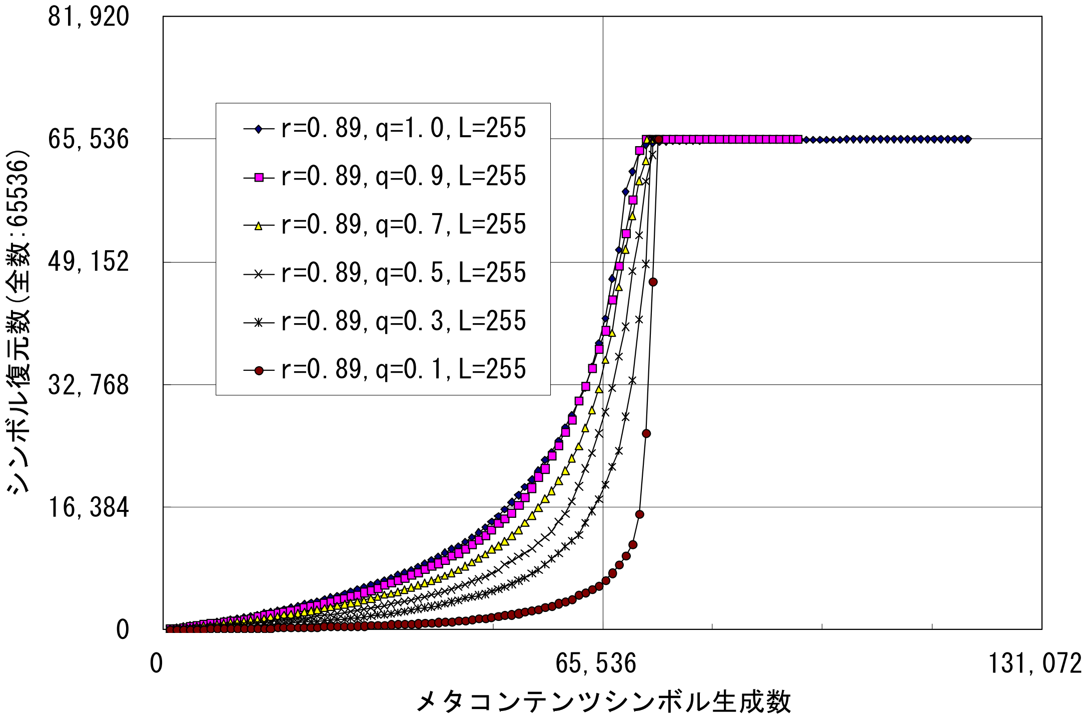


図 5-1-3-13　生成シンボル数と復元シンボル数（1）

次に q=0.1 とし r を変化させた時、復元に必要なオーバヘッド・メタコンテンツシンボル数をシミュレーションで求めたものが、図 5-1-3-14 である。この場合、r=0.92 の時に、オーバヘッド・メタコンテンツシンボル数が最小になった。ちなみに、r=0.92, q=0.1, L=255 の場合、重み 1 のメタコンテンツシンボル発生割合は、表 5-1-3-4 より 0.84%となる。

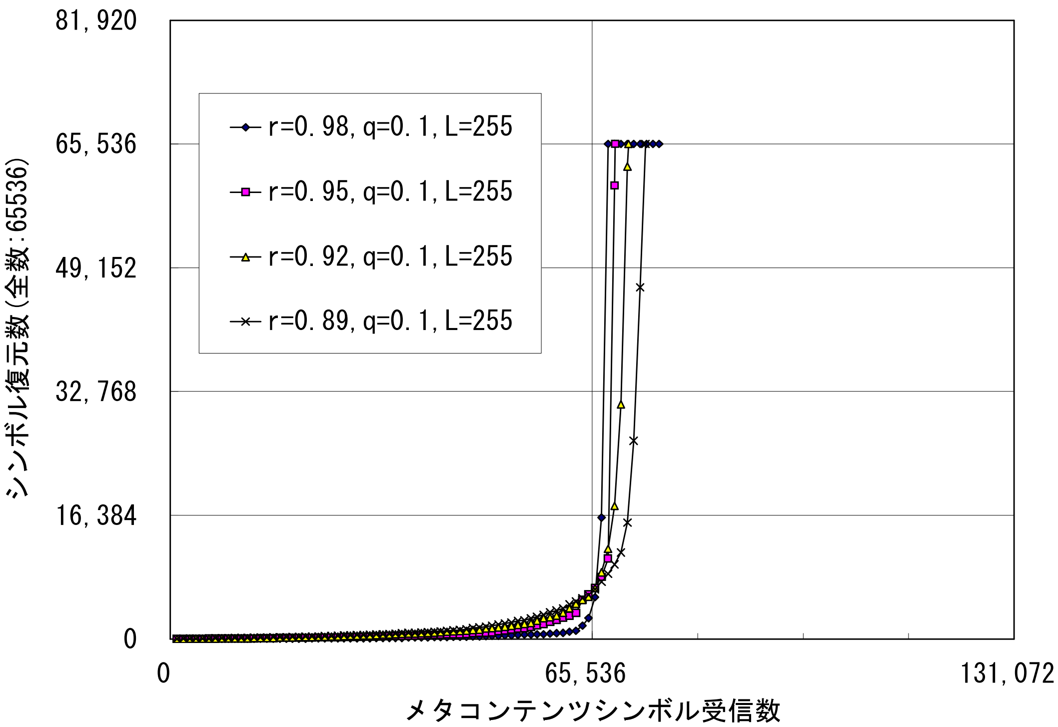


図 5-1-3-14　生成シンボル数と復元シンボル数（2）

図 5-1-3-15 は、これらのシミュレーション結果をまとめたものである。横軸は、qの値によって決まるシンボルの平均重みで、左側の縦軸は、元のシンボル数に対するオーバヘッド・メタコンテンツシンボル数の割合を示したものであり、右側の縦軸は、復号にかかった時間(Dec 処理時間)を示したものである。当初予想されたとおり、シンボルの平均重みを増大させていくと、オーバヘッドが減少していき、逆に Dec 処理時間は増加傾向にあることが分かる。ただ、Dec 処理時間曲線で、平均重みが最大の場合に、Dec 処理時間が下がっているが、この原因については現在のところ不明である。

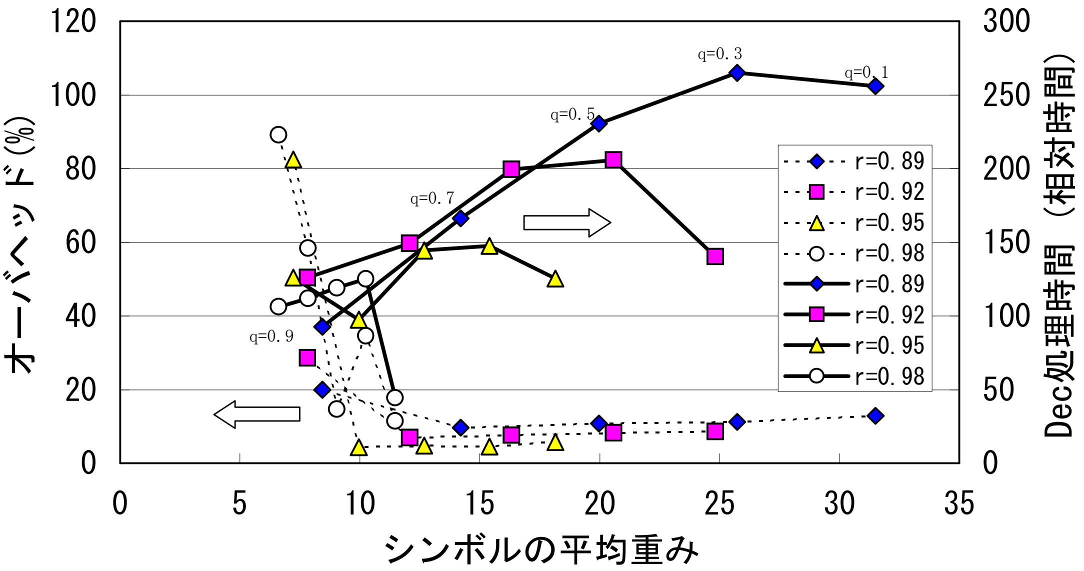


図 5-1-3-15　シンボルの平均重みと
オーバヘッドならびにデコーダ処理時間の関係

5-1-4　誤り訂正符号の性能比較

5-1-4-1　はじめに

今回我々が取り上げた Digital Fountain 社の誤り訂正符号(LT 符号と、その改良型である Raptor 符号)と、一般に広く用いられている Reed-Solomon 符号の性能について比較を行う。それぞれの符号は、符号ごとに異なる特徴を持ち、用途に対して性能が異なるため、直接的に比較することは難しい。今回は、送出されたパケットが欠損した場合の修復能力という点で、Reed-Solomon 符号と Raptor 符号を比較する。

5-1-4-2　Reed-Solomon 符号と Raptor 符号

表 5-1-4-1 は、Raptor 符号と Reed-Solomon 符号を比較した結果を、簡単にまとめたものである。Reed-Solomon 符号は、連続したビット誤り(つまりバイト誤り)を訂正するように設計された符号であり、消失したパケットを復元する機能はない。また符号化／復号化に際しては、有限体の中で多項式演算を行うことによって、誤り訂正に必要な冗長部分を計算するため、必要な処理時間が情報長の自乗に比例するという特徴を持っている。情報長が 200Byte 程度の場合、Raptor 符号と同一のハードウェアを用いて、Reed-Solomon 符号をソフトウェア的に処理した場合、１００倍以上の処理時間が必要になる。この様に大きな処理時間がかかると、符号化／復号化の処理が終了しないうちに次のデータが到着するという問題が発生する。従って、処理速度が情報長に比例する([O(n)]である)Raptor 符号が、処理能力において優れていることが分かる。

表 5-1-4-1 誤り訂正符号の比較

	Raptor符号	Reed-Solomon符号
符号化による効果	欠損パケットを復元	バイト誤りを訂正
符号化／復号化	ソフトウェア $0(n)$	ハードウェア $0(n^2)$
設定変更	容易	困難

今回のプロジェクトでは、ネットワークの誤り率に応じて自律的に誤り訂正能力を変更させることの出来る誤り訂正符号が必要となる。Reed-Solomon 符号では、複雑な多項式演算や復号アルゴリズムによって、専用の ASIC や FPGA (Field Programmable Gate Array)で処理されるために、Reed-Solomon 符号自体の誤り訂正能力を変更することが出来ない。ただ専用のハードウェア(ASIC or FPGA)の外部に設置されたソフトウェア的な処理によって、誤り訂正能力を変更させることは可能であり、以下のような方法が考えられる。

(1) Reed-Solomon 符号＋インターリーブ法

(2) Reed-Solomon 符号＋パンクチャド法＋インターリーブ法

インターリーブ法は、Reed-Solomon 符号を基にインターリーブを行ってブロック化し、誤り訂正能力を改善させる方法であり、バイト単位の誤り訂正能力を欠損パケットの復元能力に高めることが出来る。パンクチャド法は、Reed-Solomon 符号の特定のバイトを取り除き(パンクチャドし)、その符号をベースにインターリーブを行う方法である。この方法では、どのバイトを取り除くかによって、誤り訂正個数が大きく低下するので、注意が必要である。誤り訂正個数が低下した場合でも、符号長も低下するので、全体としては誤り訂正能力が向上する場合がある。

5-1-4-3　パケット欠損の発生原因

インターネットで発生するパケット欠損は、以下のように２つに分類される。

(1) 伝送路でのノイズによる(パケット)損傷によるパケット廃棄

(2) 中継器機(ルータ等)の輻輳によるパケット廃棄

ノイズによるビット誤り(パケットの損傷)は、0,1 レベルの分布が正規分布(ガウス分布)であると仮定し、しきい値のレベルを超えた場合に誤りが発生すると考えるとガウス分布の積分となり、補誤差関数になることが分かる(図 5-1-4-1 参照)。

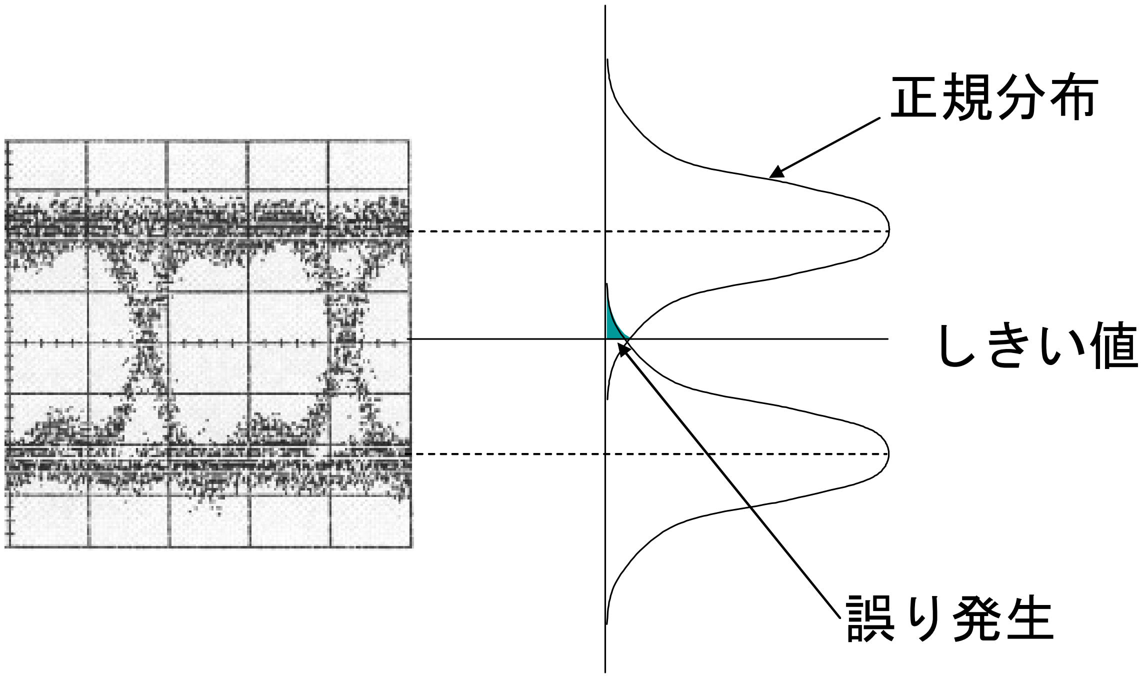


図 5-1-4-1　ノイズによる誤り発生のメカニズム

$$\mathrm{erfc}(x)=\frac{2}{\sqrt{\pi}}\int_x^{\infty}e^{-t^2}dt$$

一般的には、上記で示された補誤差関数で表される曲線が、誤り訂正を行うことによって、(同等の誤り率を実現する場合に必要な)SN 比が何 dB 低下したかを調べ、その値を符号化利得と定義し、符号の良し悪しを判断している。しかし、中継器機(ルータ等)の輻輳によるパケット廃棄を考えた場合、SN 比という概念は適切ではない。

今回我々は図 5-1-4-2 に示すように、符号化利得ではなく、誤り訂正の前後で誤り率がどの程度低下するかという「誤り訂正能力」を用いて、符号の良し悪しを検討する。厳密には、パケットをブロック化するので、横軸に平均的なパケット欠損率を取り、縦軸にブロック復元失敗率を取る。この場合、「誤り訂正能力」ではなく「ブロック復元能力」を比較することになる。

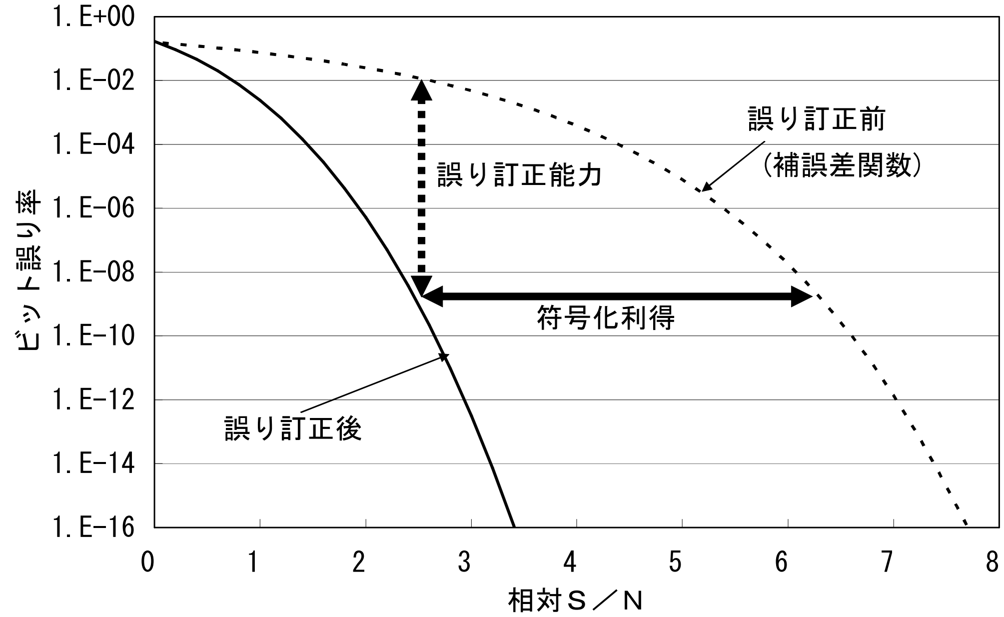


図 5-1-4-2　符号化利得と誤り訂正能力の関係

インターネットでの平均的なパケット欠損率を仮定し、複数のパケットをまとめてブロックとして取り扱った場合、その中に発生するパケット欠損は、一定の幅を持った二項分布 f(p,n,t)となることが知られている。

$$f(p,n,t)=\sum_{i=t+1}^{n} {}_nC_i \cdot p^i(1-p)^{n-i}$$

p：平均的なパケット欠損率
n：ブロック内のパケット数
t：ブロック内での復元可能なパケット数

（１）受信側で誤り訂正を行わなかった場合

受信側で誤り訂正を行わなかった場合、上記のブロック内でパケット欠損が発生した場合、そのブロックの受信に失敗したと考え、そのブロック全体を廃棄する。ブロックの中には、受信に成功したパケットも含まれているが、このように定義すると、「ブロック復元失敗率」と「パケット欠損率」とが等しくなり、計算が簡単になる。

（２）受信側で誤り訂正を行う場合

ブロック内のパケット欠損数が設定された値以下であれば、ブロック内のパケットを復元できる。設定値を超えるパケット欠損がブロック内で発生した場合には、ブロックの復元に失敗したと判断する。

5-1-4-4　ブロック復元能力

平均的なパケット欠損率を横軸に取り、ブロック化を行った場合のブロック復元失敗率を縦軸に取って比較することにより、誤り訂正(欠損復元)符号の能力を把握することが出来る。誤り訂正を行わない場合は、パケットの(平均)欠損率とブロックの復元失敗率の関係は、傾き1の直線になるような気がするが、実際には､ブロック内に含まれるパケット数 Ns (統計でいうサンプル数)により、復元失敗率が増加することが分かる。パケットの(平均)欠損率(統計学でいう母集団の欠損率)を f_p とすると、誤り訂正を行わない場合のブロック復元失敗率 f_B は

$$f_B = 1 - (1 - f_p)^{N_s}$$

となり、f_p が小さい時には、

$$f_B \cong N_s f_p (f_p << 1)$$

f_p が1に近づくと、

$$f_B \to 1 (f_p \to 1)$$

となることが分かる。受信側で誤り訂正を行うと、ブロック復元失敗率 f_B は低下し、この結果から符号の誤り訂正能力を求める。

Raptor 符号では、入力データをシンボルという単位に分割し、そのシンボル同士の排他的論理和(XOR)を取ることによって、出力シンボルを作成し、それをパケットとしてまとめて送出している。さらにこのパケットをブロック化することによって、パケット欠損に対応している。特別な場合は、1シンボルを1パケットで送出し、この場合シンボルとパケットは、同等のものと考えることが出来る。受信側で必要な出力シンボル数は、入力シンボル数以上必要となり、[余分に必要なシンボル数]／[入力シンボル数]をオーバヘッドと呼ぶ。今回の計算では、オーバヘッドが 5%の場合、10^{-13} の失敗率で元の入力シンボルを復元できると仮定して計算を進める。

図 5-1-4-3 に、Raptor 符号を用いた場合の伝送系を模式的に示している。まず設計パラメータとして、伝送路での平均的なパケット欠損率を設定する。さらにブロックを構成するパケット数を決めると、ブロック内で発生する欠損パケット数の分布が決まる。この分布は、二項分布であることが知られている。ブロックサイズを大きくすると、ブロック内での欠損パケット数の分布は、平均的なパケットの欠損率に集束する。つまり欠損パケット数の分布は、インパルス関数となる。ブロック内のパケット数を少なくすると、二項分布が広がり、伝送路に設定された(平均パケット欠損率)以上のパケット欠損率が発生する割合が増加し、欠損パケットを復元できない確率が増大する。

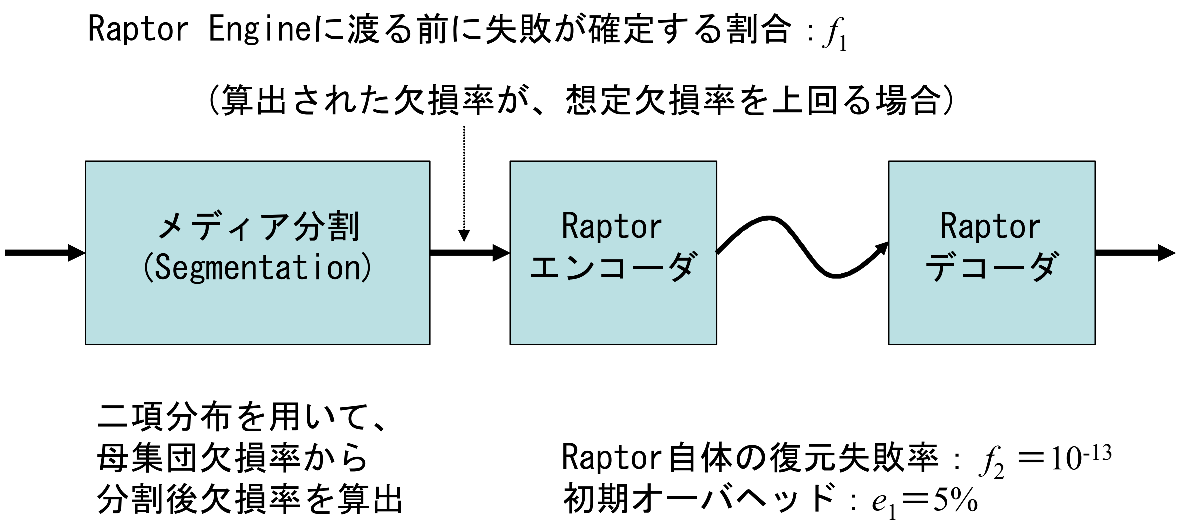


図 5-1-4-3　Raptor 符号を用いた伝送系

また、この伝送系全体での失敗率 f_Tは、「ブロック内での欠損パケット数が復元可能数を超える確率(失敗確率)を f_1」「Raptor 符号での失敗率を f_2」とすると、

$$f_T = 1-(1-f_1)(1-f_2) = f_1 + f_2 - f_1 f_2 \cong f_1 + f_2$$

で与えられる。f_1ならびに f_2が非常に小さい値の場合には、f_1と f_2の積は無視できる。

Raptor 符号では、出力シンボル数と入力シンボル数の関係によって、ブロック内で復元可能なパケット欠損数が決まる。つまり、伝送路で欠損が想定されるシンボル数に応じて、余分にシンボルを送出しておけば、受信側の復元に充分なシンボルを確保することが出来る。この関係は、出力パケット数と入力パケット数の関係と同等である。e_1を初期オーバヘッド(伝送路にパケット欠損がない場合に必要なオーバヘッド)，e_2を追加オーバヘッド(伝送路のパケット欠損に対応して決まるオーバヘッド)とすると、出力パケット数／入力パケット数＝$1+e_1+e_2$と表される。伝送路では想定されたパケット欠損が発生した場合、(出力パケット数ー欠損パケット数)／入力パケット数＝$1+e_1$という関係になり、受信側では初期オーバヘッド e_1に相当する分だけ余計にパケットを受信すれば、ブロック内のすべてのパケットを復元することが(Raptor 符号自体の失敗率は存在しているが)可能となる。

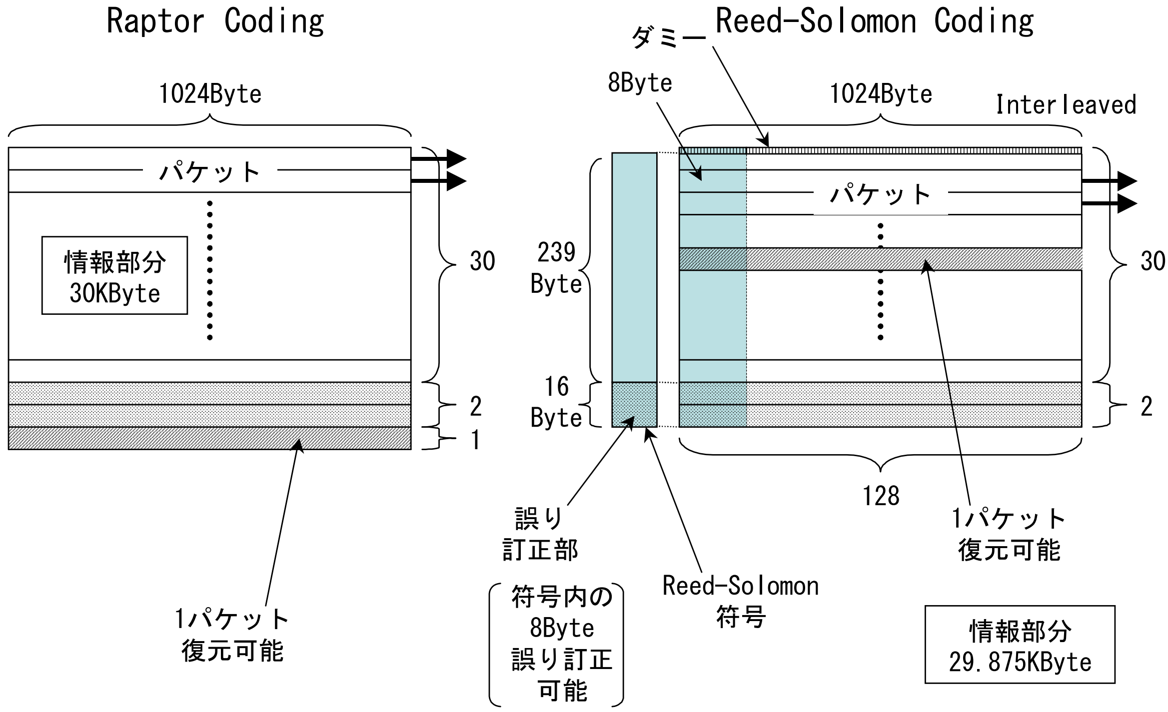


図 5-1-4-4　ブロック化の方法（1）

上記の考えに基づいて、送出ブロックのサイズを決める(図 5-1-4-4 参照)。映像のライブ配信を想定し、一定量のデータパケットを一定時間内に送出すると仮定する。Reed-Solomon 符号では、情報長 239Byte, 符号長 255Byte の符号語を想定し、これを 128 個集めて、インターリーブのためのブロックとする。元になる Reed-Solomon 符号は、8Byte の誤り訂正能力(復元能力)があり、この 8Byte を最小単位とするように、1024Byte のパケットとして送出する。この場合、1 パケットが欠損しても、各 Reed-Solomon 符号での 8Byte 欠損になるため、ブロック内の欠損パケットを復元できる。この場合、情報部分は 239Byte であり 8Byte で分割できないため、ブロック内にはダミーデータが必要となり、ブロック内の情報長は 29.875KByte となる。

Raptor 符号でも、情報長を Reed-Solomon とほぼ同じ値にするため、1024byte のパケットサイズで情報長 30KByte を送ると仮定した。この場合、情報長に相当するパケット数は 30 となり、5%の初期オーバヘッドとして 30 パケット×0.05＝1.5 パケットとなり、符号化時に 2 パケットを追加する必要がある。この状態では、欠損したパケットを復元できないので、想定されるパケット欠損に対応するパケットを追加する。図 5-1-4-4 の Reed-Solomon 符号は、1 パケット欠損しても、そのパケットを復元できる。Reed-Solomon 符号と同様のパケット欠損復元能力を与えるために、追加オーバヘッドとして 1 パケットを追加する。この場合、情報パケット数 32 に対して 1 パケットを復元できる Reed-Solomon 符号の方が、僅かに性能がよいことが分かる(表 5-1-4-2 参照)。

表 5-1-4-2　欠損パケットの復元数

	符号内のパケット数	復元可能パケット数
Reed-Solomon 符号 ＋インターリーブ	32	1
Raptor 符号	33	1

Reed-Solomon 符号は、生成多項式による演算を高速で行う必要があるために、ASIC や FPGA 等のハードウェア処理を行っている。そのため、Reed-Solomon 符号自体の誤り訂正能力を変更することは難しい。一方 Raptor 符号は、ソフトウェア的に欠損パケットの復元能力を高めることが出来る。図 5-1-4-4 に示すように、想定されるパケット欠損に応じて余分なパケットをブロックに追加することによって、欠損したパケットを復元できるのである。つまり、元のパケット数より 5%余計に出力パケットを受信出来れば、元のシンボルを復元することが可能となる。これは、符号化／復号化アルゴリズムがシンボル数に比例するという特性(線形性)を持ち、ネットワークの状況に応じてソフトウェア的に随時切り替えることが可能となる特性を利用したものである。

一方、Reed-Solomon 符号でも、以下の方法と併用して、欠損パケット復元能力を高めることが出来る。

(1) インターリーブの分散パターン変更

(2) パンクチャド符号＋インターリーブ

上記の方法では、Reed-Solomon 符号化／復号化をする前に、ソフトウェア的にデータの組み替え等を行い、実質的にパケット復元能力を変更することが可能になる。この方法を用いた場合、Reed-Solomon 符号と Raptor 符号のパケット欠損復元能力を比較する。

5-1-4-4-1　インターリーブによるデータの分散

図 5-1-4-4 では、Reed-Solomon 符号を 8Byte 単位に分割しインターリーブをかけているが、これを 4Byte, 2Byte, 1Byte に変更したものが、それぞれ図 5-1-4-5～7 である。例えば図 5-1-4-7 のように 1Byte に分割すると、符号長は 255Byte であるため、255 個のパケットに分割される。これをそのまま 1024Byte のパケットとしてブロック化すると、ブロック全体のサイズが 255Kbyte(ダミーを含めると 256KByte)と、当初のブロックサイズの 8 倍になる。ブロックサイズが大きくなると、符号化／復号化のためのバッファサイズが増加するため、遅延が増大する。今回の評価はストリーミングを想定し、一定量のデータを一定時間に送出すると考えているため、パケットサイズを 1/8 にし、ブロックサイズを一定して計算を行う。ただ実際には、パケット長を変化させても、符号自体の復元能力は変化しない。

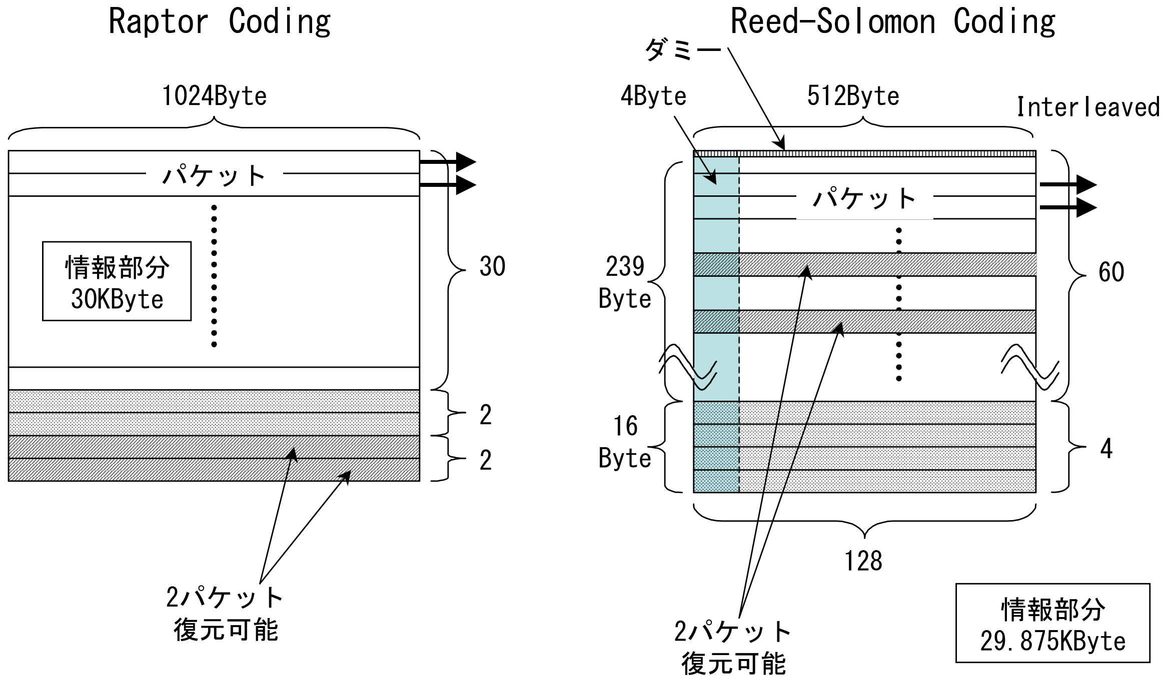


図 5-1-4-5　ブロック化の方法（２）

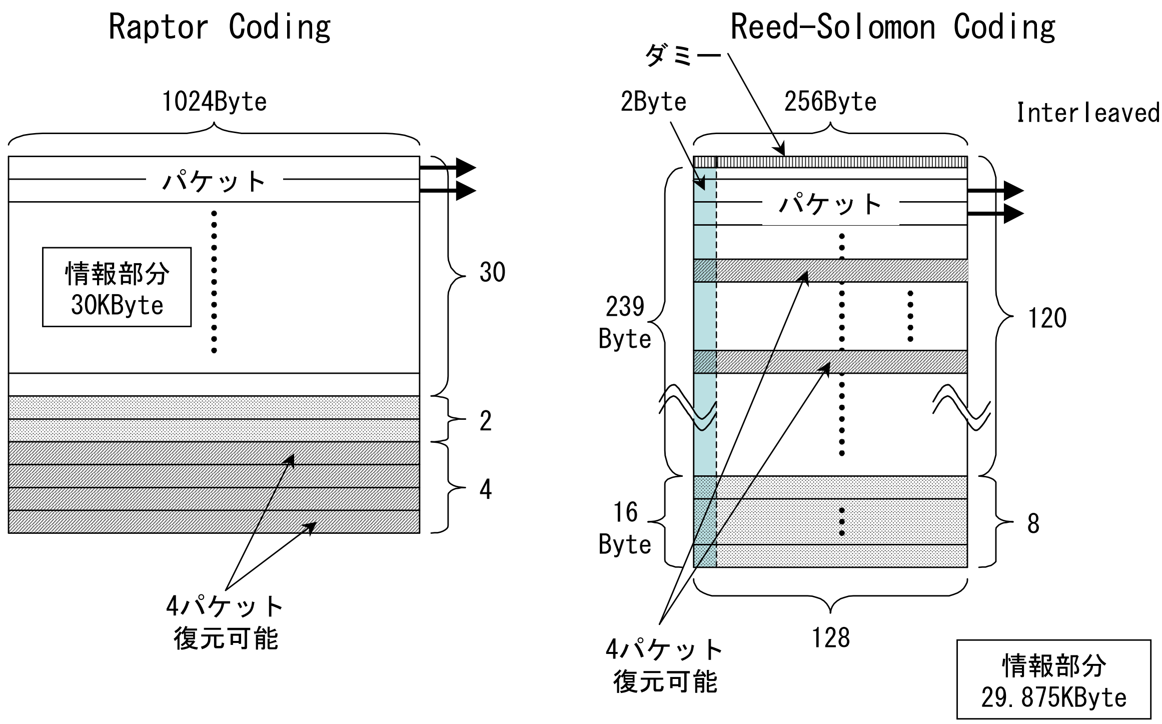


図 5-1-4-6　ブロック化の方法（３）

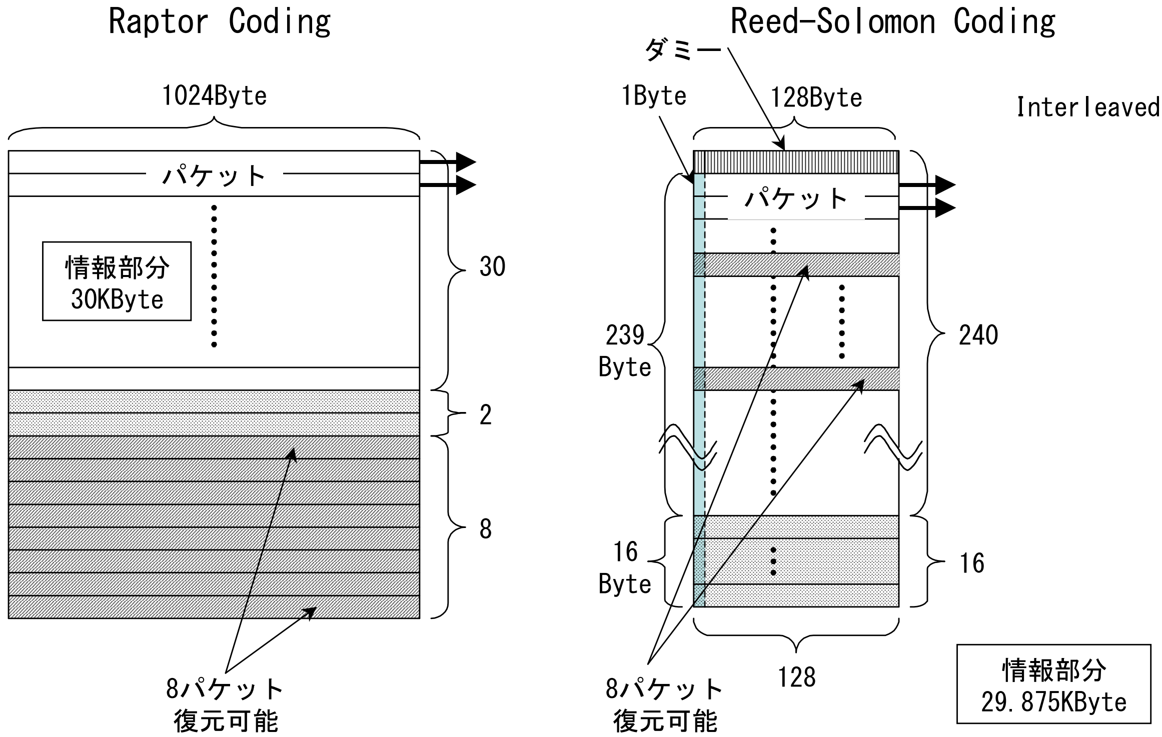


図 5-1-4-7　ブロック化の方法（４）

図 5-1-4-4～7 のようにインターリーブによってデータを分散させると、復元可能なパケット数は増加する。つまり Reed-Solomon 符号自体は、8Byte の訂正能力があるため、分割単位を 4Byte，2Byte，1Byte に変更する毎に復元可能個数は、2 個，4 個，8 個と増加する。ただ、分割単位を例えば 0.5Byte(4bit)にした場合、16 個の欠損パケットを復元することは出来ない。これは今回使用した Reed-Solomon 符号自体が、バイト単位で最大 8 個の誤りを訂正できるからである。従って、Reed-Solomon 符号とインターリーブ法を組合せた場合、パケット復元能力に一定の上限があることが分かる。

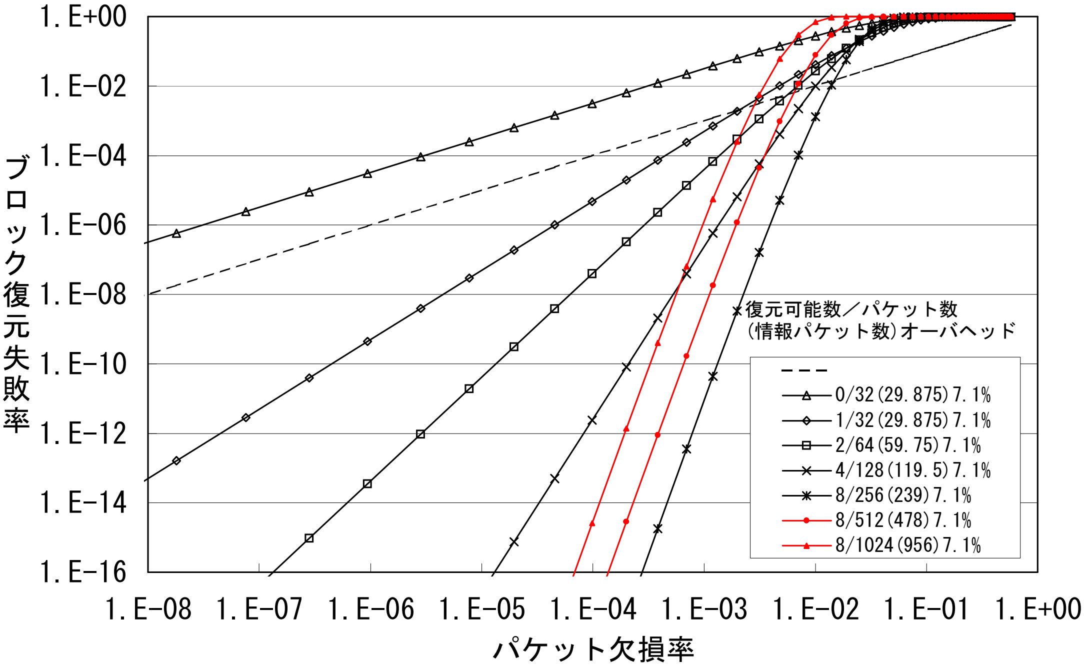


図 5-1-4-8　Reed-Solomon 符号のブロック復元能力

図 5-1-4-8 は、Reed-Solomon 符号のブロック復元能力を示したものである。横軸が、伝送路で発生する平均的なパケット欠損率であり、これを図 5-1-4-4～7 のようにブロック化し、そのブロック中に発生したパケット欠損を復元した場合の失敗率を、縦軸に示してある。凡例で、[$t/n(k)$ s%]と表示されているが、この意味は以下に示す通りである。

t：ブロック内で復元可能なパケットの個数

(Reed-Solomon では、$t=(n-k)/2$)

n：ブロック内のパケット数（符号長）

k：ブロック内の情報パケット数（情報長）

s：$(n-k)/n\times100$ (%)

図 5-1-4-8 では、「パケット欠損率」と「ブロック復元失敗率」が等しくなる場合の関係を、破線で示している。このグラフは横方向に 2 倍に拡大してあるため、本来傾き 1 になる直線が、傾き 1／2 になっている。この直線より必ず上にくる曲線[0/32(29.875)7.1%]は、ブロック化を行ってはいるが、欠損パケットの復元を行っていない場合の結果である。この場合「(平均的な)パケット欠損率：f_p」と「ブロック復元失敗率：f_B」が等しくなるような気がするが、3－4 で述べたように、この両者の関係は、下記の式で規定される。つまり N_s=1 の時にのみ、この両者は等しくなる。

$$f_B = 1-(1-f_p)^{N_s}$$

図 5-1-4-8 のグラフと、図 5-1-4-4～7 の関係をまとめると、表 5-1-4-3 の様になる。図 5-1-4-4～7 に示されるように、復元可能なパケット数を増加させる

と、図 5-1-4-8 における「ブロック復元失敗率」は低下し、ブロック復元能力は向上していることが分かる。一方、復元に必要なオーバヘッドは、7.1%と一定の値である。本来この誤り訂正符号は、復元能力 t/n (3.1%)の以下のパケット欠損に対して、元のパケットを復元できる能力を持っている。ブロック内でのパケット欠損率が、復元能力 t/n (3.1%)より大きくなった場合に、ブロック復元に失敗することが分かる。復元能力 t/n (3.1%)より大きくなる割合は、二項分布に示されるように、ブロックに含まれるパケット数が増加することによって低減する。このため、復元能力 t/n の分母(つまりブロックに含まれるパケット数：符号長)が大きくなるほど、ブロック内でパケット欠損率が 3.1%を越える確率が低下し、見かけ上ブロック復元失敗率が低下する。

表 5-1-4-3　凡例の表示と対応する図番号

凡例の表示 [$t/n(k)$ s%]	対応する Reed-Solomon のブロック化
1/32(29.875) 7.1%	図 5-1-4-4(右)
2/64(59.75) 7.1%	図 5-1-4-5(右)
4/128(119.5) 7.1%	図 5-1-4-6(右)
8/256(239) 7.1%	図 5-1-4-7(右)

インターリーブ法方法を変更し、最小単位が 4bit 以下になるように分割しても、最大 8 個のパケットしか復元できないため、図 5-1-4-8 の[8/512(478) 7.1%]ならびに[8/1024(956) 7.1%]に示されるように、欠損パケット復元能力は低下する。

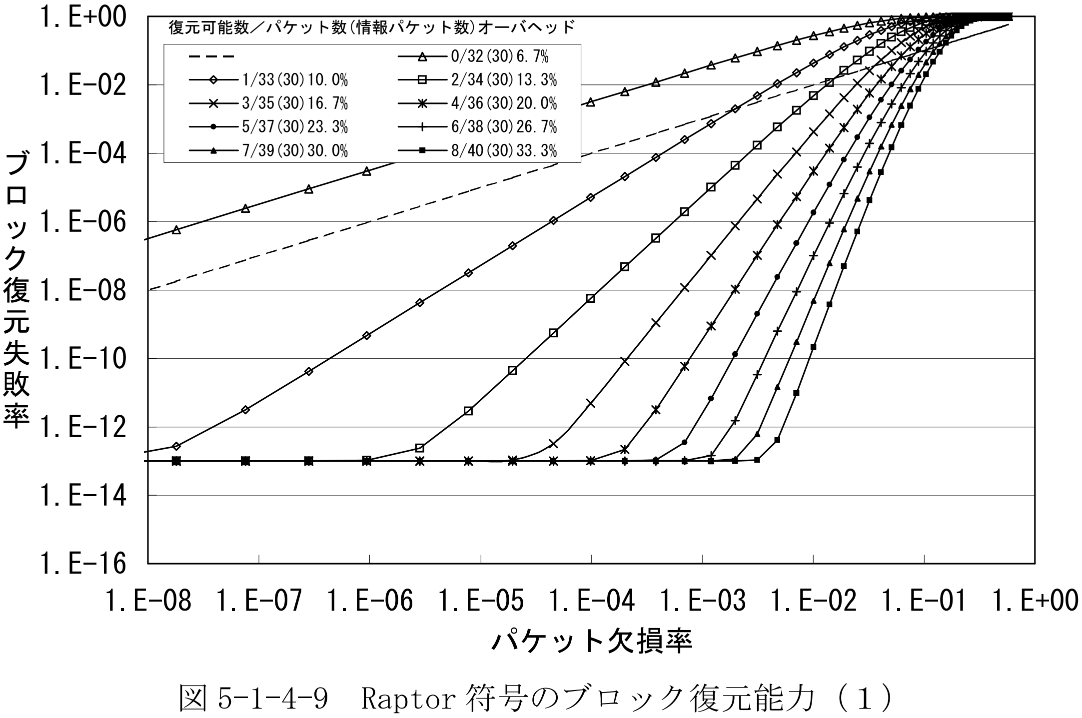


図 5-1-4-9　Raptor 符号のブロック復元能力（１）

一方Raptor符号では、欠損パケットに対応して任意のパケット数を追加できるので、図 5-1-4-4～7(左)に示す様に、誤り訂正能力を無制限に増加させることが可能となる。図 5-1-4-9, 10 は、Raptor符号のブロック復元能力を示したものである。横軸は、伝送路で発生する平均的なパケット欠損率であり、これを図 5-1-4-4～7 のようにブロック化し、そのブロック中に発生したパケット欠損を復元した場合の失敗率を、縦軸に示してある。凡例で、[$t/n(k)$ s%]と表示されているが、この意味は図 5-1-4-9 と同様である。Reed-Solomon との大きな違いは、Raptor符号自体の持つ失敗率であり、図 5-1-4-9, 10 では、10^{-13}のところに表れている。

表 5-1-4-4　凡例の表示と対応する図番号

凡例の表示 [$t/n(k)$ s%]	対応するブロック
1/33(30) 10.0%	図 5-1-4-4(左)
2/34(30) 13.3%	図 5-1-4-5(左)
4/36(119.5) 20.0%	図 5-1-4-6(左)
8/40(239) 33.3%	図 5-1-4-7(左)

表 5-1-4-4 は、凡例の表示と対応する図番号である。表 5-1-4-3 と比較すると、表 5-1-4-5 に示す様になる。つまり Reed-Solomon 符号では、オーバヘッド一定の状態(つまり、t/nが一定)で、インターリーブによってブロック復元失敗率を変化させているが、Raptor 符号では欠損パケット復元能力に応じて、オーバヘッドが増加し、直接、欠損パケット復元能力を高めている。

表 5-1-4-5　Reed-Solomon 符号と Raptor 符号の特徴

	Reed-Solomon 符号	Raptor 符号
復元可能個数 t と符号長 n の関係	t/n が一定	$n-t$ が一定
情報長 k	一定	一定
オーバヘッド s	一定	t に対応して増加

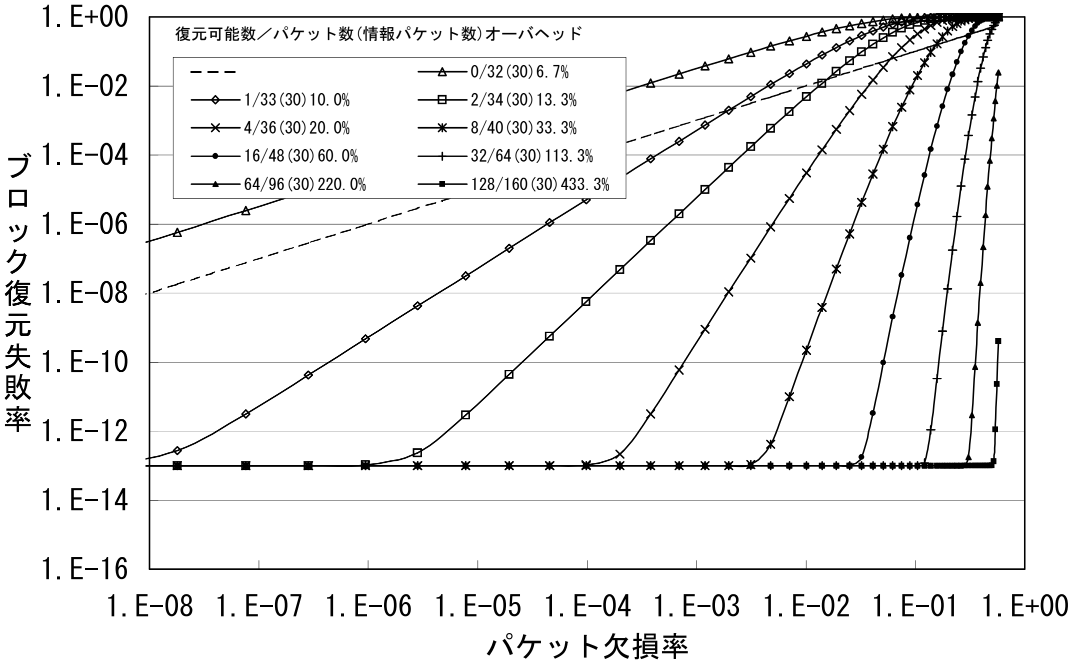


図 5-1-4-10　Raptor 符号のブロック復元能力（２）

図 5-1-4-10 に示すように、Raptor 符号においては、欠損が想定されるパケットを、無数に追加することによって、パケットの復元能力をいくらでも高めることが出来る。例えば、復元可能個数 t が 128 個の場合、パケット欠損率が数十%あっても 10^{-13} の復元失敗率で、パケット伝送が可能であることを示している。この結果から、パケット欠損復元能力を自由に変更することが可能な Raptor 符号は、性能的に遥かに優れていることが分かる。しかしオーバヘッドの大きい(符号化率の小さな)Raptor 符号と、オーバヘッドの小さな(符号化率の大きな)Reed-Solomon 符号を比較することは、不公平である。オーバヘッド(符号化率)を考慮し、この両者を比較するために、オーバヘッドを考慮した実効的な欠損復元能力について考察する。

通常の誤り訂正符号では、SN比を用いて比較する場合と、E_b/N_0という値を用いる場合がある。E_bは、ビット当たりの電力であり、N_0は雑音電力である(一般的な表記法として N_0を用いるが、ここでは $N_0=N$ である)。つまり、送出時のオーバヘッドを増加させるということは、送出するデータ量が増加することに相当し、ストリーミングの場合には、一定時間に一定量の情報部分を送出する必要があるため、オーバヘッドが増加した分だけ伝送速度を増加させる必要がある。

伝送速度の増加とビット当たりの電力との関係は、以下のように考えることが出来る。10Mbps データを高速の幹線系を経由して送信する場合、幹線系に誤り訂正機能を付加し、3 倍のデータ(つまりオーバヘッド 200%)で送出すると仮定する。この場合、幹線系で見ると、1bit 当たりの電力は同じであるが、このデータは幹線系の受信側で復号化され、3bit が 1bit に変換される。従って幹線系では、本来必要な 3 倍の電力が消費されていることが分かる。

この様に SN 比を捉えると、欠損パケットの復元は、2 つのステップで考える

ことが出来る。つまり、下記のような3つの場合である(図5-1-4-11参照)。

(1) 未符号化(ブロック化しない)の場合の、パケット欠損率(=ブロック復元失敗率)

(2) 復元のために符号化したが、欠損を復元しない場合のブロック復元失敗率

(3) 復元のために符号化し、復元機能を用いて復元した場合の、ブロック復元失敗率

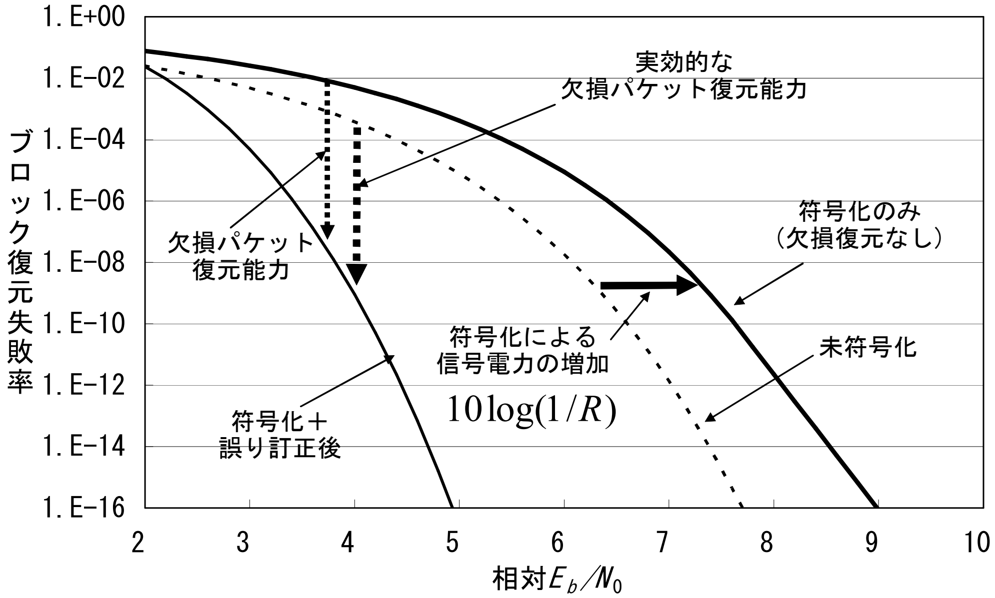


図5-1-4-11　オーバヘッドを考慮したブロック復元失敗率

(1) 未符号化(ブロック化しない)の場合の、パケット欠損率(=ブロック復元失敗率)

復元のための符号化を何も行わない場合のE_b/N_0とパケット欠損率のグラフを描く。この場合、$E_b/N_0=S/N$となる。パケットの欠損は、既に述べたように「伝送路でのノイズによる(パケット)損傷によるパケット廃棄」「中継器機(ルータ等)の輻輳によるパケット廃棄」の2つの原因で発生するが、この2つを等価的に扱い、すべて伝送路でのノイズによるものと便宜的に考える。この場合、等価的にブロックサイズ1と考えることが出来るので、パケット欠損率をブロック復元失敗率は、同じ値となる。

(2) 復元のために符号化したが、欠損を復元しない場合のブロック復元失敗率

符号化だけを行い、欠損復元を行わない場合のパケット欠損率を考える。この場合、符号化だけが行われるので、ビット当たりの電力であるE_bの値が1/Rとなる。ここでRは、符号化率であり、オーバヘッドとの関係は以下のように表される。

$$R = \frac{1}{1 + e_1 + e_2}$$

ビット当たりの電力は、$1/R$ となるため、パケット欠損率は変化せず、グラフが右に $\log(1/R)$ だけ移動することが分かる。

(3) 復元のために符号化し、復元機能を用いて復元した場合の、ブロック復元失敗率

(2)で描かれたグラフに対して、欠損パケットの復元を行うと、ブロック復元失敗率も低下する。従って、グラフが下方向にシフトする。

この時の欠損パケット復元能力は、「最初に E_b/N_0 比を基に描いたグラフ」と「最終的に得られたグラフ」との関係で示される。オーバヘッドを増加させることで欠損パケット復元能力を増大させても、オーバヘッド増加によるビット当たりの電力が増加すれば、その効果をうち消すことになる。従ってこの方法を用いると、異なるオーバヘッドを持つ２つの符号を比較することが可能となる。

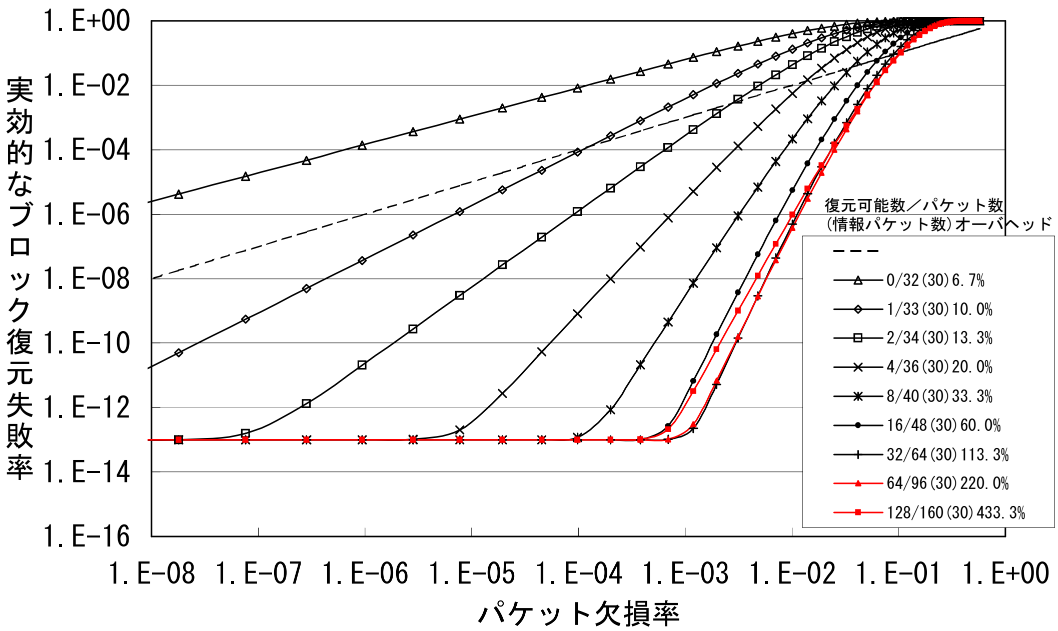


図 5-1-4-12　Raptor 符号の実効的なブロック復元能力

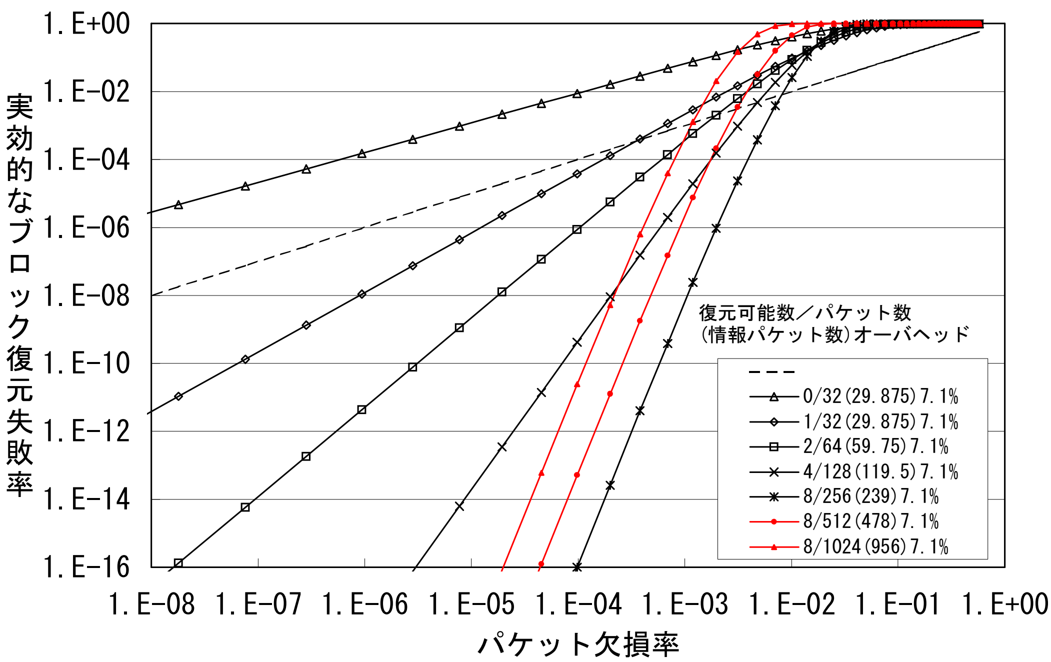


図 5-1-4-13　Reed-Solomon 符号の実効的なブロック復元能力

図 5-1-4-12 は、上記の考え方を基にして算出した「実効的なブロック復元能力」である。この結果を見ると、オーバヘッドを増大させるとある程度までは「実効的なパケット復元能力」が改善するが、それ以上改善しないことが分かる。つまり Raptor 符号には、実効的な欠損復元能力の限界があることが分かる。

同様の考え方で、オーバヘッド分だけ移動させた Reed-Solomon 符号の「実効パケット復元能力」を示したものが、図 5-1-4-13 である。図 5-1-4-12 と図 5-1-4-13 において、最も右の曲線が最も高いパケット欠損復元能力を有しているため、この両者を比較した。その結果パケット欠損復元能力は、3×10^{-4} 以上の高いパケット欠損率で、Raptor 符号の方が Reed-Solomon 符号より実効的なブロック復元失敗率が高いことが分かる。これ以下のパケット欠損が発生する領域では、Raptor 符号自体に復元失敗率があるため、復元失敗率の下限は 10^{-13} 以下にはならない。このため、ブロック復元失敗率の要求仕様が 10^{-13} を下回るような領域では、Raptor 符号を用いることは出来ない。

5-1-4-4-2　パンクチャド法+インターリーブ法

本来の誤り訂正符号から特定の部分を取り除いて、別の誤り訂正符号を構成することが出来る。ただこの方法では、訂正可能な個数(バイト数等)が変化するので、注意が必要である。今回の計算では、Reed-Solomon 符号にパンクチャド法によって符号長を変化させるが、訂正可能なバイト数は変化しないと仮定して計算を進める。

基本とするのは(255, 239, 8)Reed-Solomon 符号で、順に符号長, 情報長, 誤り訂正能力を示す。この符号を基に、(128, 112, 8), (64, 48, 8), (32, 16, 8)を構成できたと仮定する。この符号を基にブロック化を行い、ブロック復元失敗率を比較した(図 5-1-4-14)。その結果、欠損パケット復元能力自体は、パンクチャドによって、ある程度向上していくことが分かる。実際のパンクチャド符号では、8Byte という誤り訂正能力を維持することは難しいので、誤り訂正能力はさらに低下すると考えられ、単にインターリーブをかけた方が有効であると考えられる。

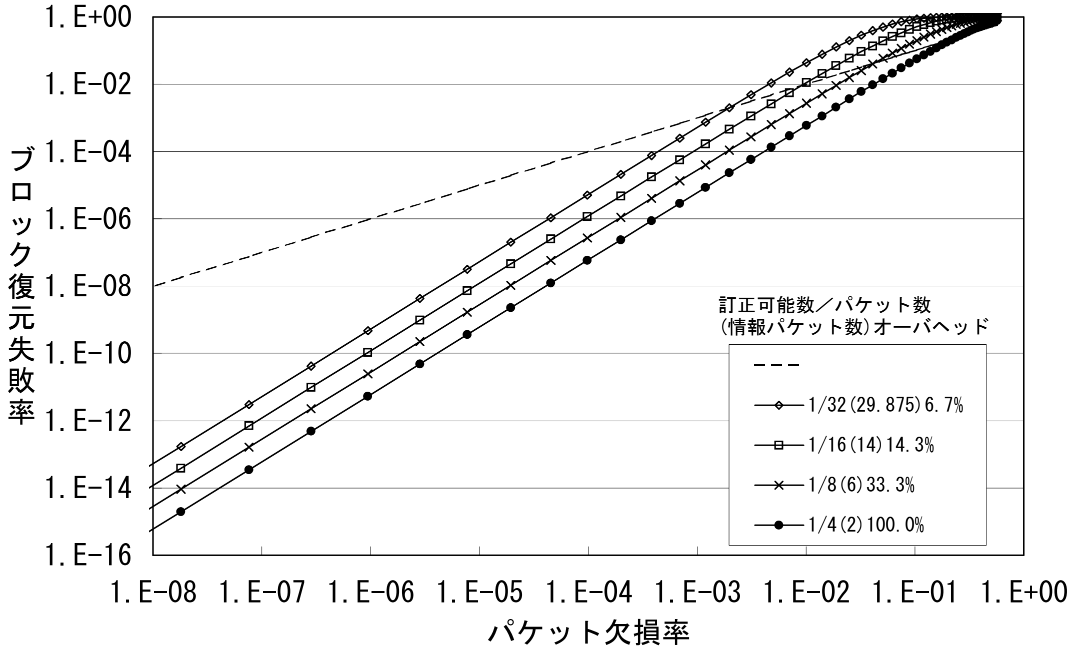


図 5-1-4-14　パンクチャドによるブロック復元能力

5-1-4-5　誤り訂正基本能力の比較

Reed-Solomon 符号と Raptor 符号の細かな違いは無視して、それぞれの符号を誤り訂正符号として捉え、その能力を比較してみる。つまり、Reed-Solomon 符号ではバイトではなく誤りの発生したパケットを訂正できると考え、Raptor 符号では、誤りの発生したパケットを修復すると考える。

一般的な誤り訂正符号の記述方法は、(n, k, t)である。誤り訂正能力を比較する場合、符号化率 k/n を横軸にとり、誤り訂正能力 t/n を縦軸にとる方法が用いられる。この場合の比較は、横軸である符号化率を一定にした場合の比較である。縦軸の t/n の値は、比の値として同じであっても、t/n=1/2 と t/n=100/200 という２つの誤り訂正能力は、その性能が大きく異なる。つまり、分母が大きいほど誤り訂正能力が高い。これは、先ほどのブロック化の例に示すように、ブロックサイズを大きくすることによってブロック内の誤り率が、伝送路の平均的な誤り率に近づくために、見かけ上の誤り訂正能力が上がるためである。また t/n=100/200 という誤り訂正能力を持つ符号は、100 個の連続的な誤りを訂正することが出来るが、t/n=1/2 という符号では、このような誤りを訂正する能力はない。この様な問題点はあるが、一般的な比較として、Reed-Solomon 符号と Raptor 符号を比較した。

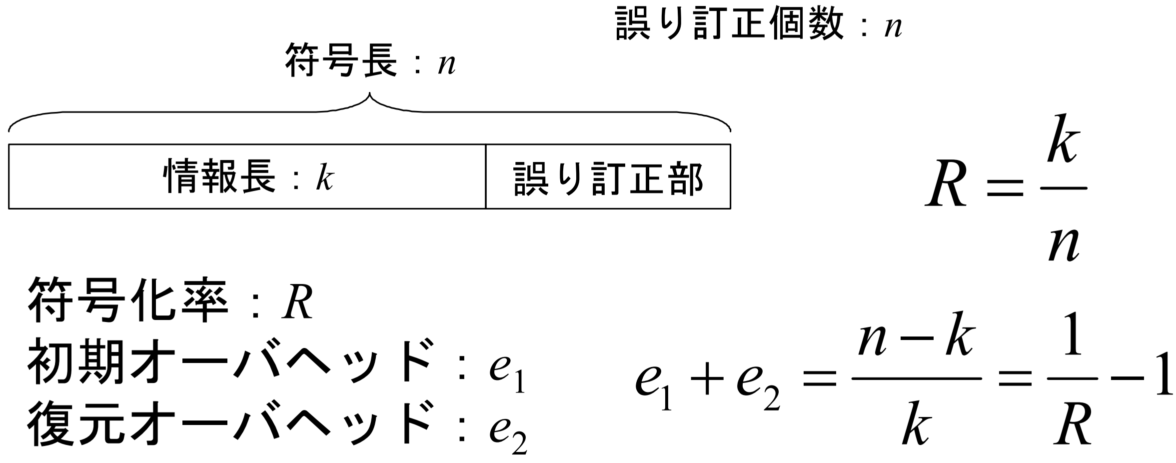


図 5-1-4-15　比較パラメータの定義

図 5-1-4-15 に、符号の性能比較に用いたパラメータをまとめる。Reed-Solomon 誤り訂正符号は、誤り訂正個数 t と情報長 k ならびに符号長 n との間に、下記のような「Reiger の限界式」が存在している。

$$t=\frac{n-k}{2}$$

Reed-Solomon 符号の誤り訂正能力 t/n は、以下のように表される。

$$\frac{t}{n}=\frac{(n-k)/2}{n}=\frac{1}{2}(1-\frac{k}{n})=\frac{1}{2}(1-R)$$

また Raptor 符号の場合は、以下のようになる。

$$\frac{t}{n}=\frac{n-(1+e_1)k}{n}=1-(1+e_1)\frac{k}{n}=1-(1+e_1)R$$

つまり Reed-Solomon 符号は、誤り訂正用の冗長部分の半分の誤りしか訂正できない。この結果は、符号化率を 0 に近づけても全体の半分しか、誤りを訂正できないことを意味する。一方 Raptor 符号は、符号化率を小さくすることで、符号全体の誤りを訂正することが出来る。ただ Raptor 符号には、初期オーバヘッドが存在し、この値によって符号化率が影響を受ける。

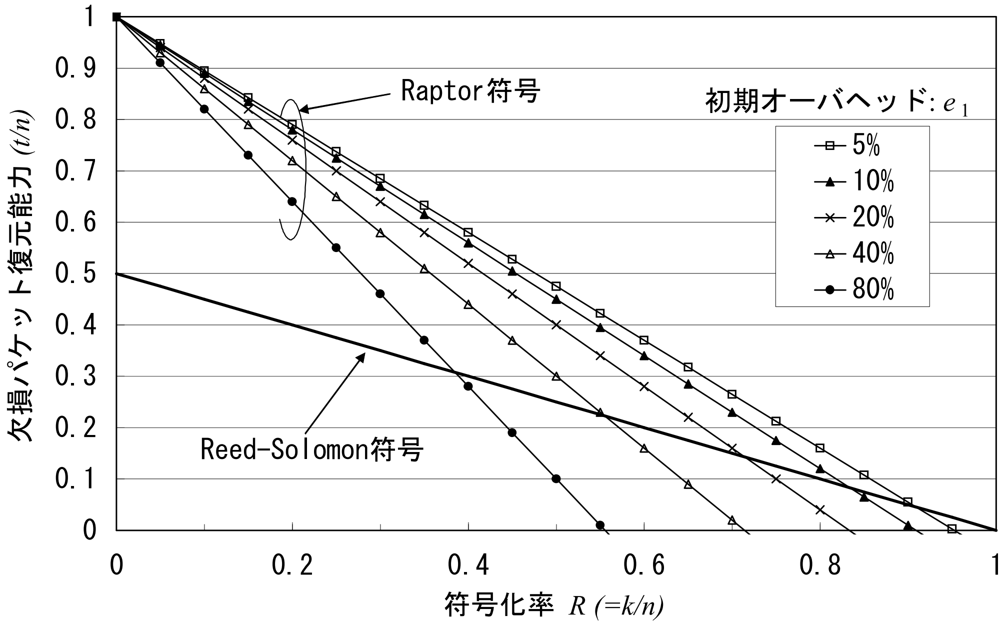


図 5-1-4-16　符号化率と誤り訂正能力の関係

図 5-1-4-16 は、Raptor 符号の初期オーバヘッドをパラメータとして、Reed-Solomon 符号と Raptor 符号の誤り訂正能力を比較したものである。Raptor 符号の初期オーバヘッドを 5%と仮定すると、符号化率が 90%以下では、Raptor 符号の方が、高い誤り訂正能力を有していることが分かる。逆に、初期オーバヘッドが大きくなると、Reed-Solomon 符号にとって有利な領域が拡大することが分かる。この結果から符号化率を 0 に近づけると、Raptor 符号の方が有利であることが分かる。

5-1-5　結論

Digital Fountain 社の誤り訂正符号について調査し、基本的な概念についてまとめを行った。この符号においては、重み関数の分布を決めることが重要である。基本的には、重みを増加させることによって、オーバヘッドを低減できるが、復号にかかる処理時間も同時に増加する。この両者のバランスを取る形で、重み関数を決める必要がある。

Reed-Solomon 誤り訂正符号と Raptor 符号の比較検討を行った。直接的に両者の誤り訂正能力を比較することは出来ないが、下記のような点で、Raptor 符号の方が有利であると考えられる。

(1) Raptor 符号の復号化アルゴリズムが、符号長に比例して増加する

(2) パケット欠損率が 3×10^{-3} 以上で、Raptor 符号の実効的な欠損パケット復元能力は高い

(3) 符号化率を同じ値にして比較すると、符号化率 90%以下で、Raptor 符号が

有利となる

今回の評価で、Digital Fountain社のRaptor符号(LT Codingの改良型誤り訂正符号)が、有望であるという結果を得た。

5-2　モデル化とシミュレーション技術の調査研究

5-2-1　序論

自律適応型のフロー毎アプリケーション毎サービスレベル毎の順方向誤り訂正機能選択制御機能を持つ多段ネットワークのモデル化を行なう

5-2-2　研究内容

多段ネットワークのモデル化を行う第一歩として、一段のネットワークモデルを考え、このシミュレーションを行っている。ネットワーク上にサポートサーバと呼ばれる機器を設置し、このサーバによってネットワーク上で発生したパケット損失を補うというモデルである。この結果を、電子情報通信学会全国大会で発表を行った。

現在のシミュレーションモデルは、UDP プロトコルに限定したモデルであるが、将来的には TCP/IP と混在した環境でシミュレーションを行う可能性も、検討している。

今後、これらのモデルにプロトコルを組み込んだ形での改良を行い、最終形態である自律適応型のフロー毎アプリケーション毎サービスレベル毎に必要な誤り訂正方法の検討を行う。なお本研究に関して、基礎的な部分は大阪大学・村上研究室に再委託を行った。再委託の研究成果報告書は、本報告書の最後に添付する。

5-2-3　誤り訂正符号の性能評価結果

5-2-3-1　冗長量に関する考察

ここでは、一般的なＦＥＣコードであるReed-Solomonと水平･垂直パリティ、および、メタコンテンツを用いた最新のMulti-stage符号であるRaptorの３つのアルゴリズムについての冗長量について考察する。なお、アルゴリズムにかけられる元のデータは、まずアルゴリズムの入力単位であるソースブロックに分割されていると仮定する。さらにこのソースブロックは、エンコードユニットという、Reed-Solomon、水平・垂直パリティ、Raptor のアルゴリズムにおけるエンコードの単位に分割されており、ソースブロック中のエンコードユニット数をｋとすると

k = ソースブロックサイズ/エンコードユニットサイズ

が成り立つ。

以下の考察は、エンコードされた単位が一般的なパケットサイズにして伝送されることを想定しており、各考察は以下の条件とする。

<u>共通パラメータ：</u>

エンコードユニットサイズ： 1024Byte固定

k（ソースブロックサイズ中のエンコードユニット数）:

16, 32, 64, 128, 256, 512, 1024

これより、必然的にソースブロックサイズは(16KB、32 KB、64 KB、128 KB、256 KB、512 KB、1024 KB)となる。

Maximum Loss Provability in a Block（ソースブックサイズ内の最大パケットロス率）:ネットワークの平均パケットロスから、パケットロスの分布モデルをもとに算出され、ｋが小さくなるほど、大きな値をとる。

<u>アルゴリズム固有のパラメータ</u>

Reed-Solomon：

p：パケット廃棄率

m：ｋ個のエンコードユニットに対して付加する冗長符号数

水平・垂直パリティ：

v：垂直エンコードユニット数

h：水平エンコードユニット数

ここで、水平・垂直パリティアルゴリズムではエンコードユニットは長方形の形状にならべて水平・垂直それぞれにパリティを付与するため、ｋ＝ｖ×ｈが常に成り立つ。このため、(k,v,h) = (16,4,4)、(32,4,8)、(64,8,8)、(128,8,16)、(256,16,16)、(512,16,32)、(1024,32,32,)の条件について考察する。

5-2-3-2　冗長符号データサイズに関する考察

Reed-Solomon

　冗長符号データサイズ（冗長符号数：m×エンコードユニットサイズ= m KB）を計算し、これらをプロットした結果を図 5-2-3-1 に示す。

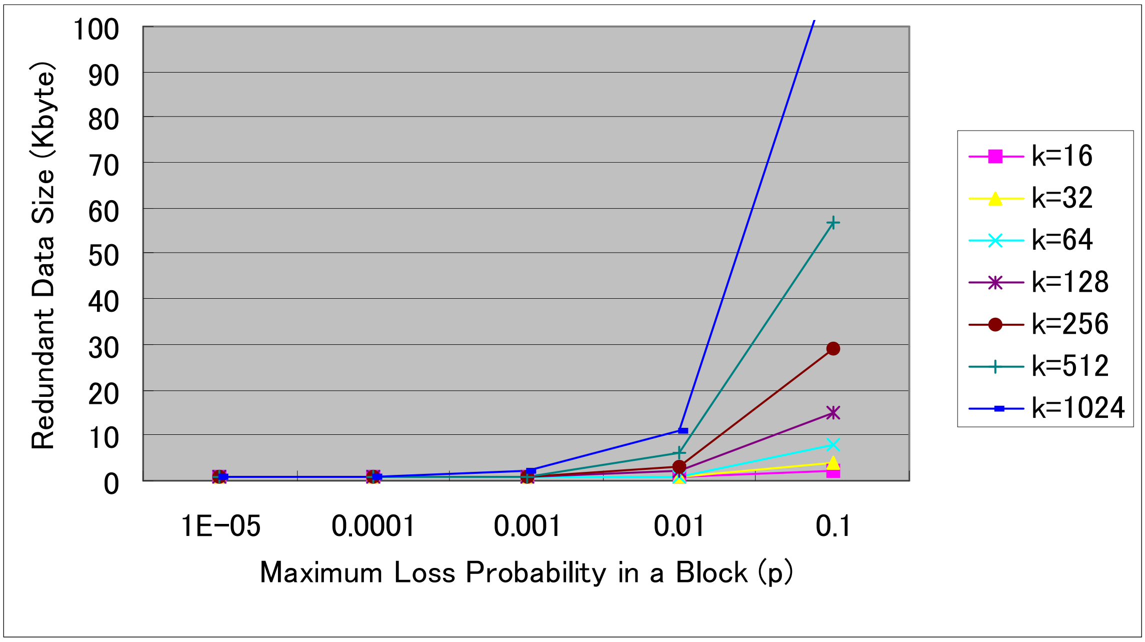


図 5-2-3-1　Reed-Solomon **アルゴリズム のロス率に対する冗長量**

　Reed-Solomon アルゴリズムの場合、sender が送信する k＋m個のパケットから receiver は任意の k 個を受信することでデータの復元が可能となる。したがって、送信するパケットが確率 p で廃棄された値（つまり 1-p が残る）が k 以上であればデータは復元できる。よって、パケット廃棄率 p から最低限必要な冗長符号数mは

$$(k+m)\times(1-p)=k$$

よって

$$m=p\times k/(1-p)$$

となる。図 5-3-2-1 では上式で得られた*m*の ceiling（mより大きい最小の整数）を選択している。

水平・垂直パリティ

水平・垂直パリティの場合は、パラメータ（v, h）により、あらかじめ作成される冗長データサイズが決められているため、パケット廃棄率により冗長データが変化することはない。

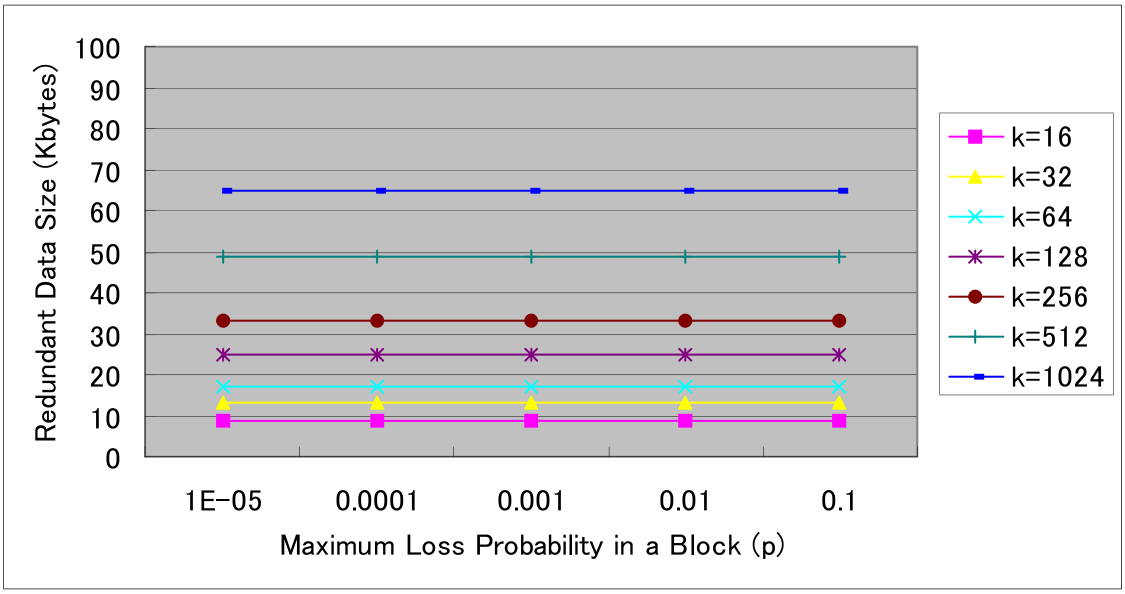


図 5-2-3-2　水平・垂直パリティアルゴリズムの冗長量

さらに、パケット廃棄率の値にかかわらずある行・列のパケット廃棄が 2 つ以上あるときに復元不可能な場合が存在するという性質を持つ。

図 5-2-3-2（(k, v, h)=(9, 3, 3)）の場合について説明する。図 5-2-3-3 の左では繰り返し、水平・垂直のパリティ計算をしていけば、デコード可能である。しかしながら図 5-2-3-3 の右では、２つ以上のパケット廃棄がある行・列を繋げると四角形ができており、互いにエラーを 1 追加にすることができず、情報を復元することは不可能である。これは（単一）パリティチェックは 2 重誤りを復元できないためである。

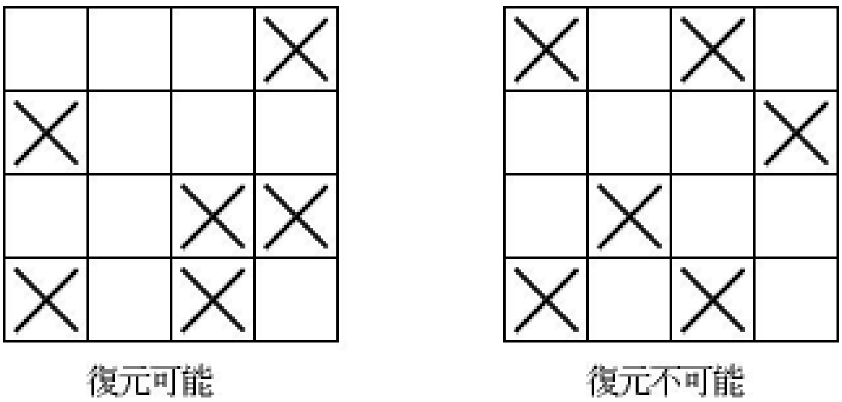


図 5-2-3-3　パリティエラー

Raptor

Raptor の冗長度は、デコードの失敗率に依存する冗長度とパケット廃棄率に依存する冗長度の積により計算される。前者は、たとえば Standard Mode の場合、パケット廃棄率 10^-11 のとき冗長度５％ 固定となる。後者は、ソースブロックサイズ内の最大パケットロス率により求まる。

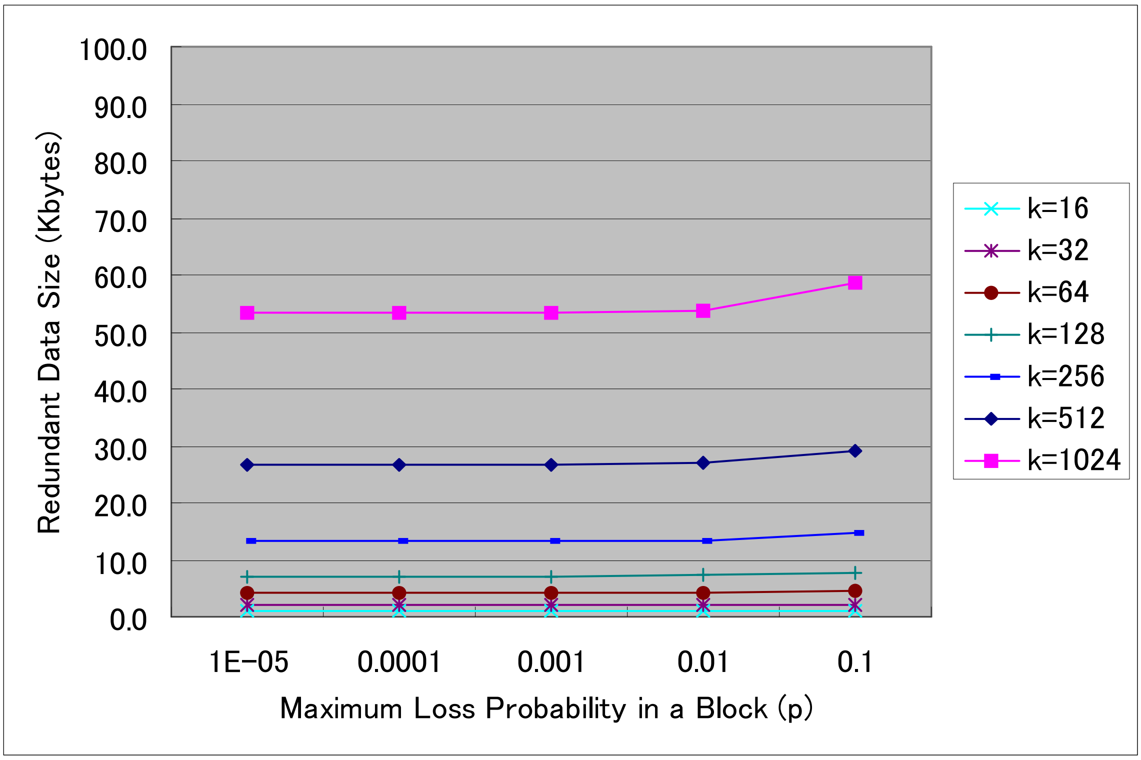


図 5-2-3-4　Raptor （Protocol-3 Standard Mode）アルゴリズムの冗長量

冗長量の比較

3 つのアルゴリズムの冗長量について比較する(図 5-2-3-5)。

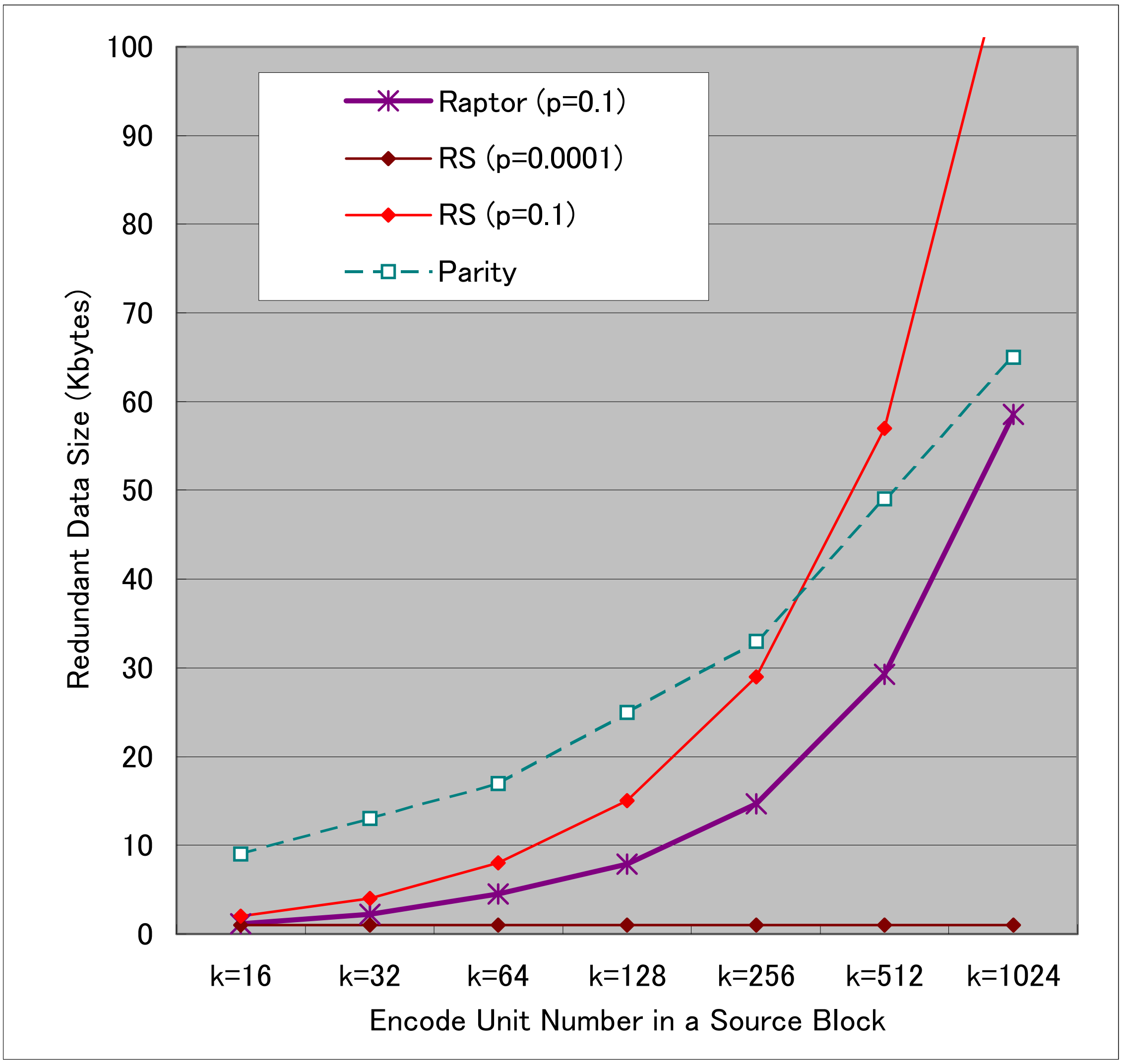


図 5-2-3-5　3 アルゴリズムの冗長量の比較

まず、Raptor、水平垂直パリティは、ブロック数の数によって決まった割合で付与するアルゴリズムであるため、エンコードユニットサイズが 1024byte と固定している状況では、ソースブロックサイズ中のエンコードユニット数 k が増加するに従い、線形に増加する。(ただし ceiling)。これに対して Reed-Solomon はパケットロス率に依存して冗長量が大きく変化するためアルゴリズムであるため、ロス率が低い状況では、k によらず低い値にとどまるが、ロス率が極端に高い領域では付与しなければならない冗長量が増加する。

この比較で注意すべきことは、それぞれのアルゴリズムの考え方の違いからデコードの失敗率が異なる、即ち、水平垂直パリティは大きなデコード失敗率を有する点である。

5-2-3-3　処理負荷の評価

5-2-3-3-1　概要および測定条件

各フォワードエラーコレクションアルゴリズムの相対的な処理負荷を比較するため、以下のような機器構成上で実験を行った。

表 5-2-3-1　機器構成

OS(*1)	Mandrake Linux 2.2.17-21mdk
CPU	Pentium III 933MHz
メモリ	768Mbyte

(*1) 実機検証で使用する MPEG-2 ストリーミング･エンジンを含むシステムでは、ストリーミングサーバは Windows-2000、デコーダは Linux ベースのシステム上で動作することが想定されているが、本章における評価では、エンコードとデコードの相対的な負荷を比較するため、同一の機器上で測定を行なっている。

測定条件：

(1) 送信するソースデータのサイズは 1MByte に固定する（本測定ではランダムのテキストファイルを使用)。

(2) これに対して、エンコードユニットサイズは 1024byte 固定とし、k（ソースブロックサイズ / エンコードユニットサイズ)の値については、k=16, 32, 64, 128, 256, 512, 1024 の値で測定を行なう　（すなわち、ソースブロックサイズを 16Kbyte, 32Kbyte, 64Kbyte, 128Kbyte, 256Kbyte, 512Kbyte, および 1024Kbyte の範囲で変更し、それぞれの条件におけるデータのエンコード・デコード時間を測定する。

(3) それぞれの条件のもと、エンコーダとデコーダを同じ Linux ホスト上で動作させ、エンコーダにより符号化、あるいは冗長コードを負荷されたデータをソケット通信でデコーダへ転送し、デコーダ側でデータを復号する。

(4) 本章の実験では、ネットワーク上で実際にパケットロスが起こることは想定しない。すなわち全てのパケットが正常にデコーダへ到着する場合における処理負荷を測定する。

(5) ホストの負荷状況により、処理時間に揺らぎが生じるため、各実験ごとに最低十回の測定を行い、その単純平均を結果として採用する。

(6) 各アルゴリズムに対応するエンコード/デコード関数で要し CPU 時間を処理時間とする一従って通常ファイルまたはネットワークへのアクセスは処理時間に含めないが、処理の過程におけるバッファ間のメモリコピー等の操作等も符号化・復号化処理の一部と見なす。

5-2-3-3-2　Raptor アルゴリズム

本節では、Raptor アルゴリズムによる符号化、復号化処理時間の測定結果を示し、その考察を行う。

表 5-2-3-2 に、評価システム上で 1Mbyte のソースデータを転送した際、各条件におけるエンコード・デコード処理時間の平均を示す。

表 5-2-3-2　Raptor アルゴリズムによる
1MByte のソースデータに対する処理時間(単位:ms)

	k=16	k=32	k=64	k=128	k=256	k=512	k=1024
エンコード	0.94	0.71	0.62	0.31	0.31	0.17	0.10
デコード	0.28	0.18	0.10	0.07	0.05	0.03	0.03

これらの結果から、図 5-2-3-6 に示すように、ソースデータ全体に対する処理時間は、ソースブロックサイズとエンコードユニットサイズの比である k パラメータが小さい場合、(本実験ではエンコードユニットサイズを 1024 バイトに固定しているため、ソースブロックサイズが小さいことに相当する)、処理の負荷が大きくなる傾向にある。

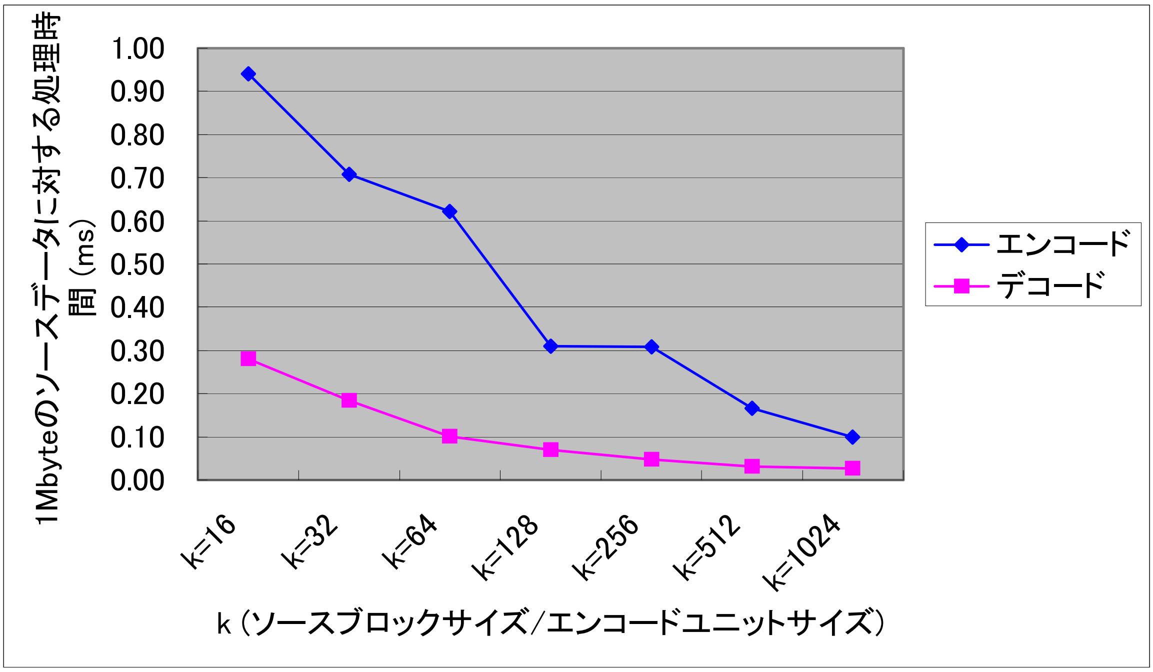


図 5-2-3-6　Raptor アルゴリズムによる
1 Mbyte のソースデータに対する処理時間　(単位:ms)

図 5-2-3-7 のように、Raptor アルゴリズムでは１ソースブロックあたりの平均処理時間は、そのソースブロックのサイズによる処理量の増加が後述する Reed-Solomon アルゴリズムと比べ、非常に緩やかであることが分かる。これは、Raptor によるエンコード・デコード処理の計算量が小さいためと考えられる。

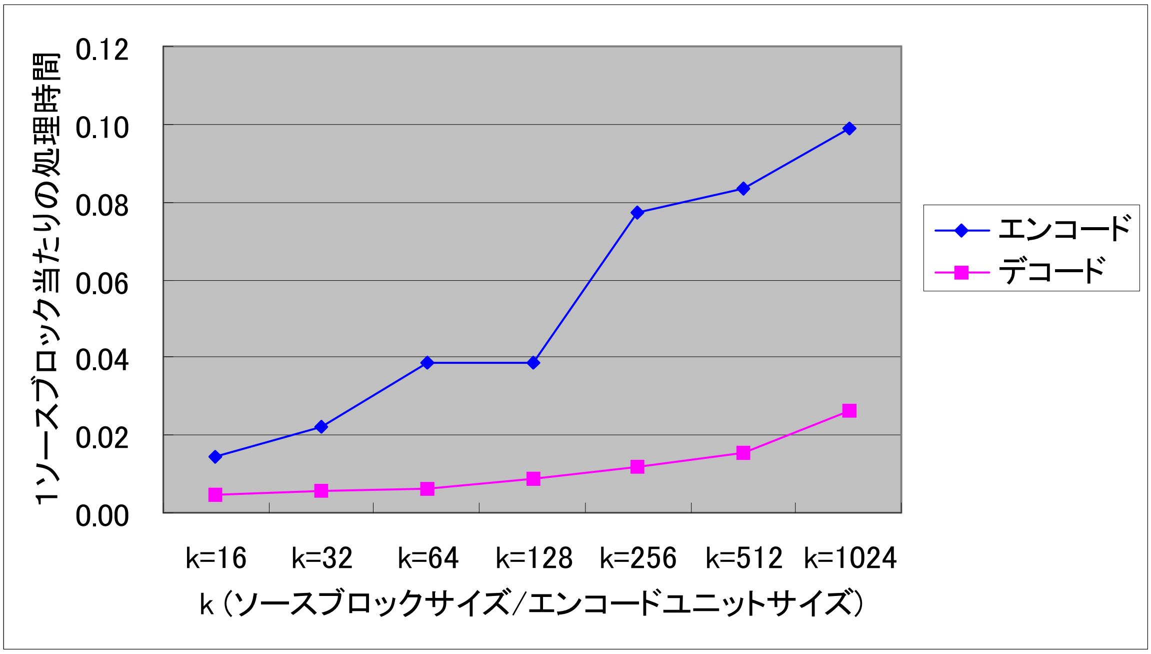


図 5-2-3-7　Raptor アルゴリズムによる１ソースブロックあたりの
平均処理時間（単位:ms）

5-2-3-3-3　Reed-Solomon アルゴリズム

本節では、Reed-Solomon アルゴリズムにおけるエンコード・デコード処理時間の測定結果を示し、これについて考察する。

Reed-Solomon アルゴリズムでソースブロックサイズ・エンコードユニットサイズのパラメータの他にネットワーク上の予想パケットロス率ｐがパラメータとなる。本実験では m=0.00001，0.0001，0.001，0.01，0.1 について測定を行った。

エンコード時の処理時間の測定結果を表 5-2-3-3 および図 5-2-3-8 に示す。図に示すように、エンコード処理は Raptor と異なり、ｋの値が大きくなるに従い処理時間が急速に増加する。また、ソースブロック内最大パケットロス率ｐが増加するにつれて、処理負荷が増加していく。

表 5-2-3-3　Reed-Solomon アルゴリズム(エンコード)による 1MByte の
ソースデータに対する処理時間（単位:ms）

エラーレート p	k=16	k=32	k=64	k=128	k=256	k=512	k=1024
0.00001	1.04	1.83	2.92	4.99	8.97	17.99	36.99
0.0001	1.05	1.85	2.95	5.15	9.21	19.02	39.25
0.001	1.06	1.84	2.99	5.34	9.61	19.40	57.88
0.01	1.08	1.85	3.02	8.18	24.02	77.85	269.30
0.1	1.56	4.30	13.50	46.46	175.84	665.75	2759.30

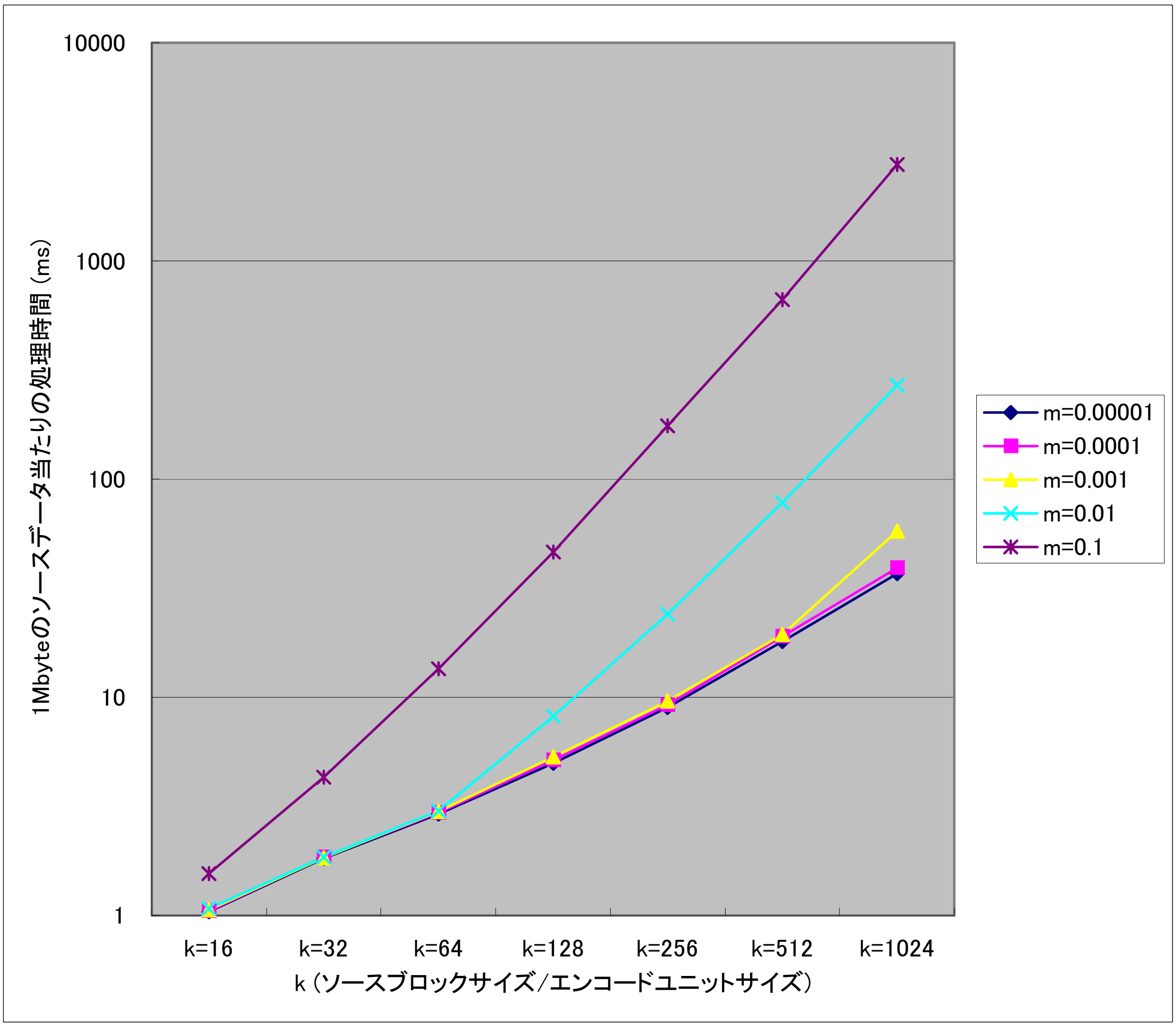


図 5-2-3-8　Reed-Solomon アルゴリズム(エンコード)による
1 Mbyte のソースデータに対する処理時間(単位:ms)

表 5-2-3-4 および図 5-2-3-9 に示すように、デコーダでもソースブロック内最大パケットロス率 p が増加するにつれて、処理負荷が増加していく。Raptor と異なり、k の値が大きくなるに従い処理時間が急速に増加する。

表 5-2-3-4　Reed-Solomon アルゴリズム(デコード)による
1MByte のソースデータに対する処理時間(単位:ms)

エラーレート	k=16	k=32	k=64	k=128	k=256	k=512	k=1024
p=0.00001	1.00	1.71	2.97	4.79	9.52	17.99	40.17
p=0.0001	1.00	1.78	2.99	4.99	9.59	18.20	39.18
p=0.001	1.00	1.80	3.02	4.83	9.94	18.27	66.30
P=0.01	1.02	1.73	2.92	7.49	23.20	68.12	291.60
P=0.1	1.55	4.02	14.24	45.70	200.20	650.06	2625.25

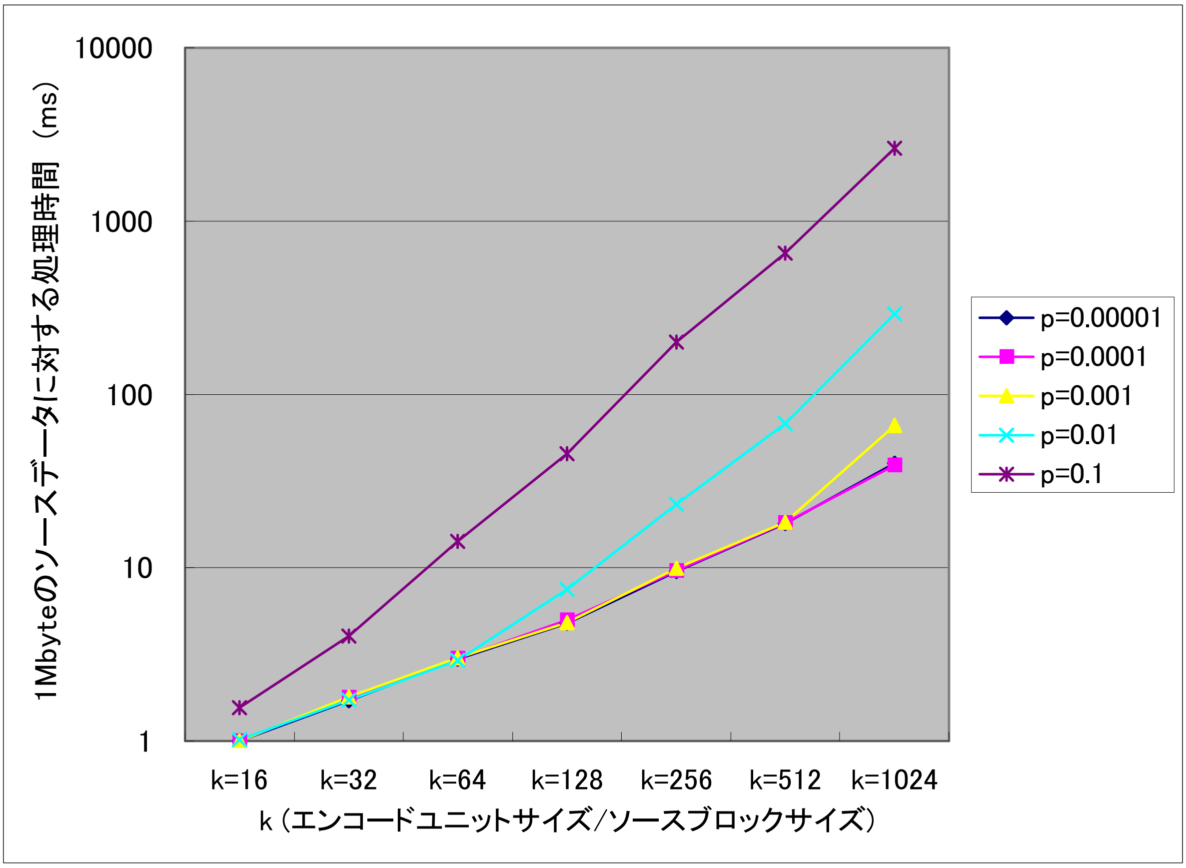


図 5-2-3-9　Reed-Solomon アルゴリズム(エンコード)による
1 Mbyte のソースデータに対する処理時間(単位:ms)

5-2-3-3-4　水平・垂直Parityアルゴリズム

本節では、水平・垂直パリティアルゴリズムによるエンコード・デコード処理時間の測定結果およびその考察を示す。パリティアルゴリズムではソースブロックサイズとエンコードユニットサイズの他に、水平方向のマトリックスのサイズｈおよび垂直方向のマトリックスサイズｖがパラメータとなる。

本実験では、ｖとｈはソースブロックが正方形に近い形となるよう(すなわちｈとｖが近い値をとるよう)、各ｋごとに固定値を選択している。また、特にデコーダの処理負荷はパケットの損失数によって変化するが、ここでは本章の他のアルゴリズムに対する評価と同様、全てのパケットがデコーダに到着するという条件で測定を行なっている。

表 5-2-3-5　水平・垂直パリティアルゴリズムによる
1Mbyte のソースデータに対する処理時間　（単位：ms）

	(k,v,h)=(16,4,4)	(k,v,h)=(32,4,8)	(k,v,h)=(64,8,8)	(k,v,h)=(128,8,16)	(k,v,h)=(256,16,16)	(k,v,h)=(512,16,32)	(k,v,h)=(1024,32,32)
エンコーダ	0.35	0.22	0.21	0.16	0.16	0.15	0.15
デコーダ	0.28	0.2	0.17	0.15	0.16	0.14	0.14

これらの結果で明らかなように、水平・垂直パリティアルゴリズムは Raptor と同様、エンコード・デコード処理の負荷が比較的軽いため、ｋが小さい、すなわちソースブロックサイズが小さい場合は、バッファの切り替え・初期化などのオーバヘッドにより、処理時間が増加する傾向にあるが、ｋによる処理時間の変化は Reed-Solomon に比べ小さい。ただし、水平・垂直パリティアルゴリズムは、本質的に全ての欠損パケットを救済できる訳ではない点に注意を要する。

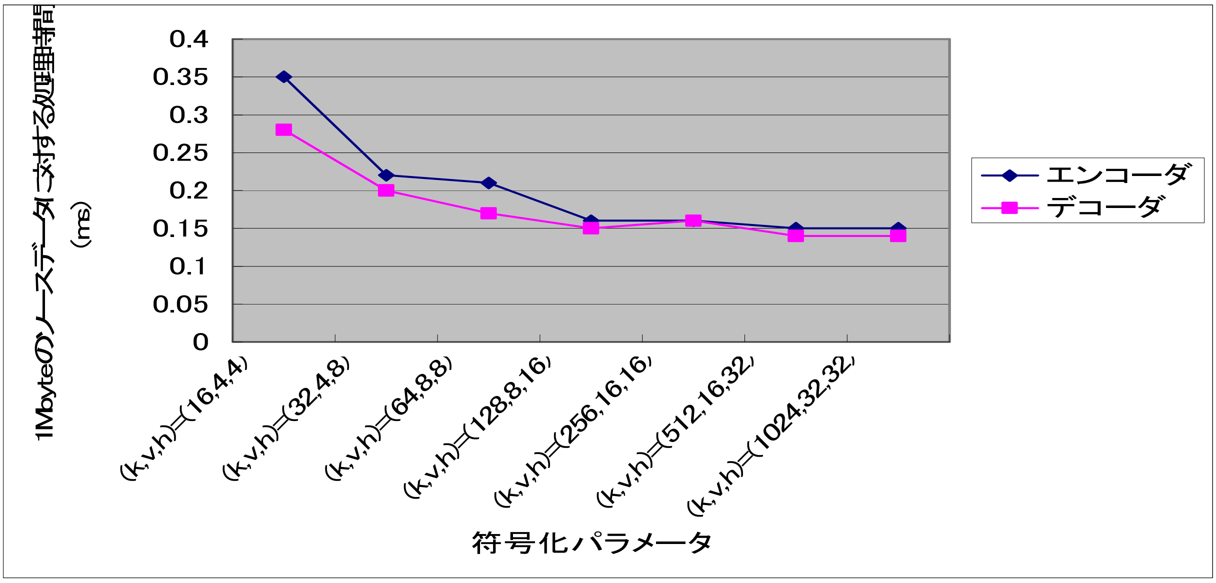


図 5-2-3-10　水平・垂直パリティアルゴリズムによる
1Mbyte のソースデータに対する処理時間（単位:ms）

5-2-3-3-5　各方式の比較

本節では、得られた測定データをもとに、Raptor, Reed-Solomon, Parity の三方式の比較を行う。負荷はパケットロス率により異なるが、ここでは p=0.0001 の場合について検討する。

エンコード時の各アルゴリズムの処理時間を表 5-2-3-6 に示す。

表5-2-3-6　各FECアルゴリズム（エンコード）による
1MByteのソースデータの平均処理時間の比較（単位:ms）

	k=16	k=32	k=64	k=128	k=256	k=512	k=1024
Raptor	0.94	0.71	0.62	0.31	0.31	0.17	0.10
RS	1.26	2.54	5.10	10.29	20.71	41.37	82.73
Parity	0.35	0.22	0.21	0.16	0.16	0.15	0.15

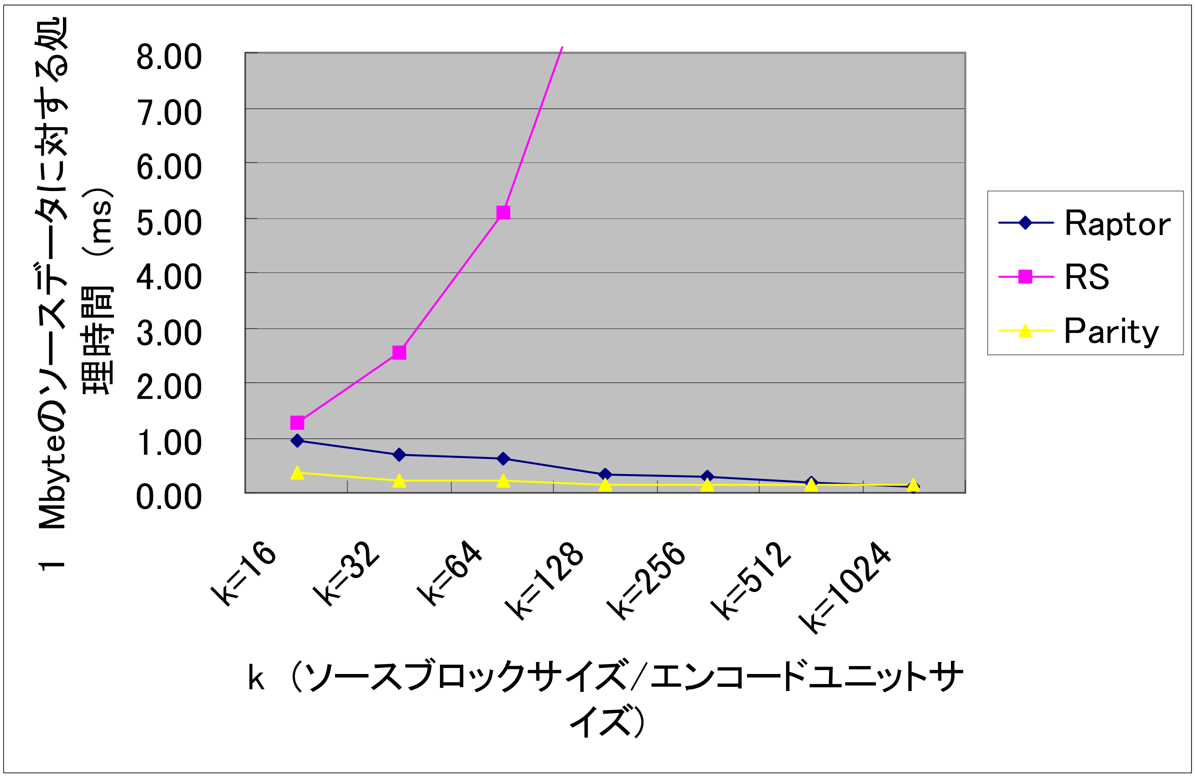


図5-2-3-11　各FECアルゴリズム(エンコード)による
1MByteのソースデータの平均処理時間の比較（単位:ms）

表 5-2-3-6 および図 5-2-3-11 の結果を比較すると、特に *k*=32～64 以上の値については Reed-Solomon のみ急速に処理負荷が増加することが分かる。一方、Parity および Reed-Solomon アルゴリズムは、ほとんど変化しないことが分かる。

同様に表 5-2-3-7 にデコード時の各アルゴリズムの平均処理時間を示す。デコードの場合についてもエンコードと同様に、*k* の値が大きくなるにつれて Reed-Solomon アルゴリズムの処理負荷の増加が顕著となる。

表 5-2-3-7　各 FEC アルゴリズム（デコード）による
1MByte のソースデータの平均処理時間の比較（単位: ms）

	k=16	k=32	k=64	k=128	k=256	k=512	k=1024
Raptor	0.28	0.18	0.10	0.07	0.05	0.03	0.03
RS	1.00	1.78	2.99	4.99	9.59	18.20	39.18
Parity	0.28	0.2	0.17	0.15	0.16	0.14	0.14

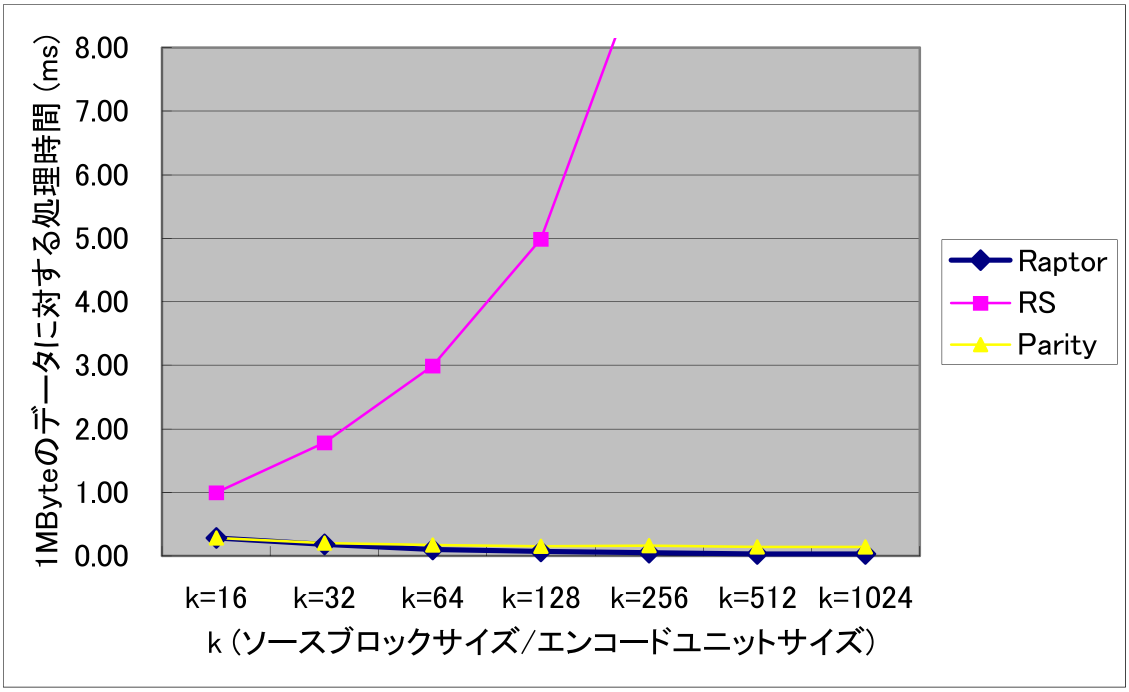


図 5-2-3-12　各 FEC アルゴリズム(デコード)による
1MByte のソースデータの平均処理時間の比較（単位:ms）

5-2-3-4　総合評価

以上の評価結果より、Raptor アルゴリズムは、Reed-Solomon より圧倒的に高速で、水平・垂直パリティーコード（バーストロスに弱い）並の高速処理性を有し、Reed-Solomon より僅かな冗長度で、広い範囲のパケットロス分布に対して、10^-11 オーダの失敗率で、パケット欠損の修復を行うことが出来る高性能なＦＥＣであると言える。

5-2-4　映像品質主観評価

5-2-4-1　概要

人間にとって、映像の劣化や乱れを感じさせないような圧縮方式で伝送を行うことは、伝送帯域を節約する点で重要な技術である。MPEG 圧縮技術では、人間が感じない高い周波数成分を取り除き、低い伝送速度でも映像の劣化を感じさせないことが可能になる。MPEG 等で圧縮された映像をインターネットで伝送する場合、映像乱れを発生させないためには、符号化された画像データを誤りなく、受信側で元のデータに復元する必要がある。映像データパケットは、下記のような場合に廃棄され、映像の乱れを発生させる。

（１）データパケットのビット誤りによるパケット廃棄

伝送路で、ノイズによるビット誤りが発生したパケットが、ネットワークの下位レイヤで廃棄される。

（２）伝送機器(ルータ)での輻輳等によるパケット廃棄(バッファ・オーバフロー)

処理能力を超えたパケットの受信(輻輳)やジッタ(パケット到着時刻の変動)によって、伝送機器(ルータ)内でバッファがオーバフローすることによってパケットが廃棄される。

（３）復号器内部でのジッタによるパケット廃棄(バッファ・オーバフロー)

ジッタによって復号器内部でもバッファオーバフローが発生し、パケットが廃棄される。一方、パケットの欠損等が発生した場合でも、同じ画像を繰り返す方法や誤り発生部分を周辺色で保管することによって、発生する映像の乱れを認識できないようにする事も可能である。

（４）受信側に映像データが届かない(バッファ・アンダーフロー)

ネットワークの遅延によって、映像パケットが届かず、バッファがアンダーフローを起こす。

さらに、上記のように伝送された MPEG データに問題が発生した場合でも、(MPEG 圧縮技術の延長として)人間が映像の劣化や乱れを認識できないような技術を開発することは、別の意味で重要である。

ここでは、発生した映像の乱れが、どのような原因で発生したのかを明らかにすると同時に、今回導入した誤り訂正技術によって、映像の劣化ならびに乱れを、どの程度緩和できるかを明らかにする。

5-2-4-2　プロトタイプ機での実験

5-2-4-2-1　実験系

図 5-2-4-1 に、Raptor の評価に用いた実験構成を示す。

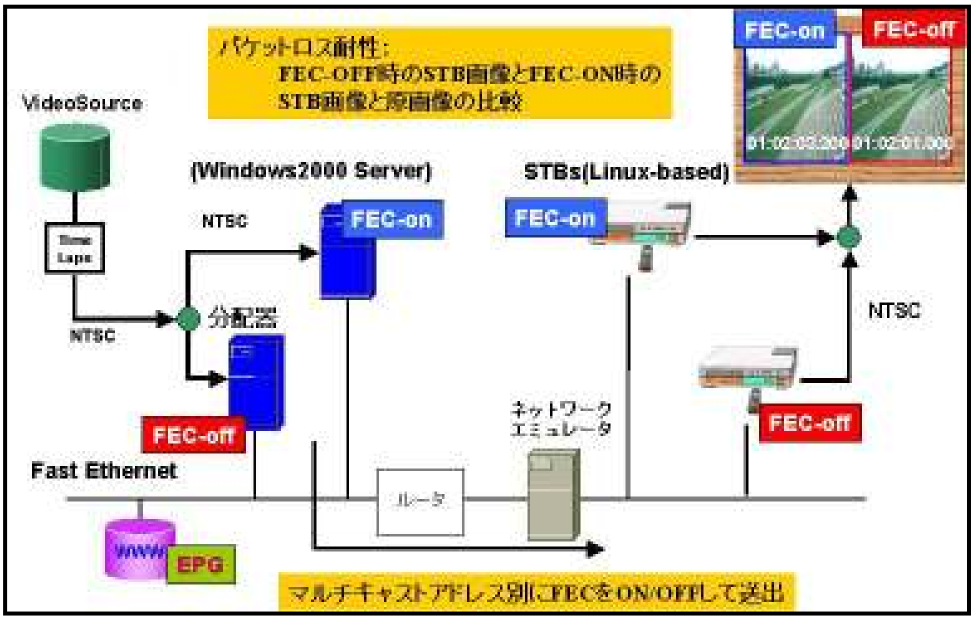


図 5-2-4-1　映像評価実験構成

構成品

（１）MPEG 配信サーバソフトウェア：Infovalue 社　QuickVideo　Multicast6.3

　ＭＰＥＧ２コンテンツをＶｏＤ配信、マルチキャスト配信するのに使用。試作品はこのサーバソフトウェアとのインタフェースを持ち、データ送信時にＦＥＣ（Raptor）をエンコードする関数を呼出し、ＦＥＣ（Raptor）処理したパケットを送出する。

（２）STB(Set Top Box)：Coventive 社　LibraII

ＷＷＷサーバ上の電子プログラムガイド（ＥＰＧ）を表示することで、ユーザがコンテンツ選択をリモコン操作で行うことができるようにした。試作品は同社のＳＴＢの受信ソフトウェアとのインタフェースを持ち、データ受信時にＦＥＣ（Raptor）をデコードする関数を呼出し、ＦＥＣ（Raptor）をデコードしたデータを上位層へ渡す。

（３）ネットワークエミュレータ：SHUNRA 社　STORM

ネットワークの各種特性（パケットロス、ジッタ、遅延等）をエミュレートして発生させる装置。

（４）ルータ：Unisphere 社製　ERX705

Internet 等のルータ網を模擬するための装置

5-2-4-2-2　映像乱れの確認

ネットワークエミュレータで、所定のパケット欠損を与え、映像を「主観的」に評価した。(表 5-2-4-1 参照)

（1）ネットワークエミュレータでの設定項目

・Periodic：何パケット毎にいくつ欠損させるかを設定する。

・ランダムロス：何%欠損させるかを設定。乱数で欠損パケットを決定

表 5-2-4-1　パケット欠損と主観評価結果

パケット欠損種類	テスト内容	主観評価結果	
		FEC あり	FEC なし
Periodic	(1)100 回に 1 回欠損	音声・映像ともに乱れなし	音とび・ブロックノイズ発生
	(2)30 回に 1 回欠損		(1)に加えてコマ送りのような状態
	(3)20 回に 1 回欠損		画像・音声ともに停止
ランダム	(1) 0.1%	音声・映像ともに乱れなし	音とび・ブロックノイズ発生
	(2) 0.5%		(1)に加えてコマ送りのような状態
	(3) 1.0%		画像・音声ともに停止

5-2-5　今後の課題

5-2-5-1　ネットワークエミュレータの特性

【パケット欠損】

パケット欠損に対しては、ネットワークエミュレータで設定した条件通りの欠損率であることを確認済であるが、パケットの到着時刻間隔を測定し、パケット欠損のバースト性についてもチェックする必要がある。

【ジッタ】

伝送パケットに対して、所望のジッタを設定できるか調べる必要がある。このために、

（１）等間隔でパケットを送出する

（２）MPEG のタイムスタンプ情報を利用する

という２つの方法が考えられる。スニーファ等の測定器を用いて、パケットの到着時刻を測定し、横軸に到着時刻、縦軸に出発時刻をプロットする。理想的なネットワークでは、グラフは傾き１の直線になるが、ジッタが発生すると、この直線からずれることになる。最小自乗法を用いて、ジッタのグラフに最も近い直線の式を導出し、その差を取ることによってジッタの標準偏差を測定する。ネットワークエミュレータで設定した値と、測定結果を比較し、設定通りのジッタを発生しているかをチェックする。

5-2-5-2　ジッタによる影響の調査

ジッタのない状態では、上記の誤り訂正手法で、完全に誤りを訂正できることが分かった。しかし実際のネットワークでは、ネットワーク上にジッタが発生するので、設定された誤り率以下でも、映像に乱れが発生する。この影響を確認するために、ジッタによる映像の乱れを「主観的」「客観的」に測定する必要がある。ネットワーク上に発生したジッタによって、本来の誤り訂正能力が低下するが、今後はこの程度を調べていく。

5-2-5-3　商用ネットワークへの適用

上記の実験はネットワークエミュレータを用いた実験であったが、今後は商用ネットワークでの「パケット欠損率」「ジッタ」についても調査する。これらの結果から、商用ネットワークに対する誤り訂正能力やバッファに対する設計手法を明らかにしていく。このような目的のために住友電工リサーチセンターを設置したので、各種回線サービスなどを導入、実験を進めていく。

＜住友電工 CDN リサーチセンター＞

5-2-6　結論

多段ネットワークのモデル化を行う第一歩として、一段のネットワークモデルを考え、このシミュレーションを行い、モデルの有効性を確認できた。Reed-Solomon符号とRaptor符号をシミュレーションによってその処理時間を比較し、Raptorが有利であることを確認した。また購入した設備を用いて、映像評価実験の環境を整備し、主観的な映像評価実験の基礎検討を開始した。

5-3　プロトコルの調査研究

5-3-1　序論

自律適応型のフロー毎アプリケーション毎サービスレベル毎の順方向誤り訂正機能選択制御機能を持つ多段ネットワークを実現するためのあらたなIPプロトコルならびにパケット構造案の素案を作成する。

5-3-2　研究内容

まず、誤り訂正機能を持つサーバ、持たないサーバ、誤り訂正機能を持つクライアント、持たないクライアントが共存するヘテロジーニアスなネットワーク環境において、自律適用的にサーバ・クライアント間で誤り訂正機能のネゴシエーションを実施するための基本的なプロトコルの必要要件に関して検討を進め、ＲＴＰ／ＲＴＳＰをベースに機能拡張を行なう方向で新たな通信制御プロトコルの素案を検討した。

5-3-3　国際標準化への準備

5-3-3-1　標準化機関選定ならびにIETFへのアプローチ

本研究開発成果を国際的に標準化するためにIETFを選定した。IETFでの標準化プロセスの調査のため、標準化の方法を説明したセッションに参加し、プロセスの確認を行った。また、本研究開発成果を標準化提案するWGの調査を実施した。

03年3月16日から21日まで米国カリフォルニア州サンフランシスコで開催された第56回IETFミーティングに参加、またDigital Fountain社を訪問、これから着手の予定の FEC 関連プロトコルの標準化のための状況調査を行ってきたので以下に報告する。

5-3-3-2　IETFでの標準化プロセス

IETFでのプロトコルの標準化についての一般論を説明したセッションが開催されたので参加した。

RFCの種類

(1) Standard Track

インターネット標準にするためには、この種類のドキュメントを発行しなければならない。Standard TrackのドキュメントにはProposed Standadrd、Draft Standard および Internet Standard の三つの標準化レベルがある。Standard TrackのRFCはWG等からArea Directorを介してIESGにサブミットする。必要な要件が満たされるとIESGからRFC Editorに渡され正式にRFCとして発行され、適当な標準化レベルが付与される。

(2) BCP

プロトコル自体を規定するものではなく、運用やポリシーのガイドラインを示すドキュメント。BCPとしてドキュメントを発行するためにはStandard Trackのドキュメントに準じた手続きが必要。

(3) Informational

Internet Communityへの一般的な情報を記述したドキュメント。IETF以外で開発され、今後IETFのプロトコルとして発行されないプロトコル仕様などがInformationalとなる。InformationalとExperimentalのRFCはWG、IESG、IABあるいは個人などから直接RFC Editorにサブミットすることにより発行可能。

(4) Experimental

実験／研究段階のプロトコル仕様を記述したもの。IRTFのRGやIETFのWGが発行するか、あるいは個人の寄書かもしれない。Experimentalとして発行されたプロトコルが後日、Standard Trackとなることもある。

(5) Historical

Standard TrackのRFCのうち、そこで規定するプロトコルが新しいバージョンあるいはプロトコルにより置換えられた時、そのRFCはHistoricalのステータスとなる。

標準レベル

(0) Internet Draft

Proposed Standardになる以前の状態のプロトコル仕様はInternet Draftとして公開される。Internet Draftはワーク用のドキュメントであり、正式に発行されたドキュメントとはみなされない。またInternet Draft は、原則として６ヶ月後には消去され将来的に残らない。

(1) Proposed Standard (PS)

プロトコル案をProposed Standardにするためには、仕様が基本的に安定しており、決定するべきことがすべて決まっていることが要求される。またインターネットのコミュニティによく知られており、強い興味が示されていることが必要である。

Proposed Standardのプロトコルは、実装あるいは運用実績は必須ではないが強く要求される。特に既存のインターネットに大きな影響を与える場合、IESGが実装や運用実績を要求する場合もある。

PSは６ヶ月以上２年以内にDSにするか、更新するか、廃棄する必要がある。

一旦、RFCとして発行されると誤字レベルであってもドキュメントを変更することはできない。何らかの変更があれば新しいRFC番号が与えられる。

(2) Draft Standard (DS)

プロトコル案をDraft Standardにするためには、そこで規定されるすべてのオプションを含めて、複数の独立に開発された実装があり、それが相互接続することが要求される。

* 我々が標準化を進める上でも、どこかのメーカ（あるいは大学）に我々の提案するプロトコルを(移植ではなく独自に)実装してもらうための根回しを事前にしておくことが必要。

Draft Standardは非常に安定したプロトコルと認識され、フィールドで見つかった問題に対処する変更以外の変更は原則として許されない。

DSは４ヶ月以上２年以内に標準にするか、更新するか、廃棄するか、Proposed Standardに戻す必要がある。

(3) Internet Standard (S)

十分な実装と運用実績のあるプロトコルがインターネット標準となる。

インターネット標準は、新しいバージョンが標準として発行されHistoricalとならない限り、インターネット標準としてのステータスを維持する。

標準化手続き

インターネット標準にするためには、Standard TrackにのせPS->DS->Sという段階を踏まなければならない。

プロトコル案を新たにStandard Trackにのせ、あるいは標準化段階を進めるには、

・ まず、そのプロトコル案をInternet Draftとして発行。基本はWGからだが、

個人が直接出すことも可能。

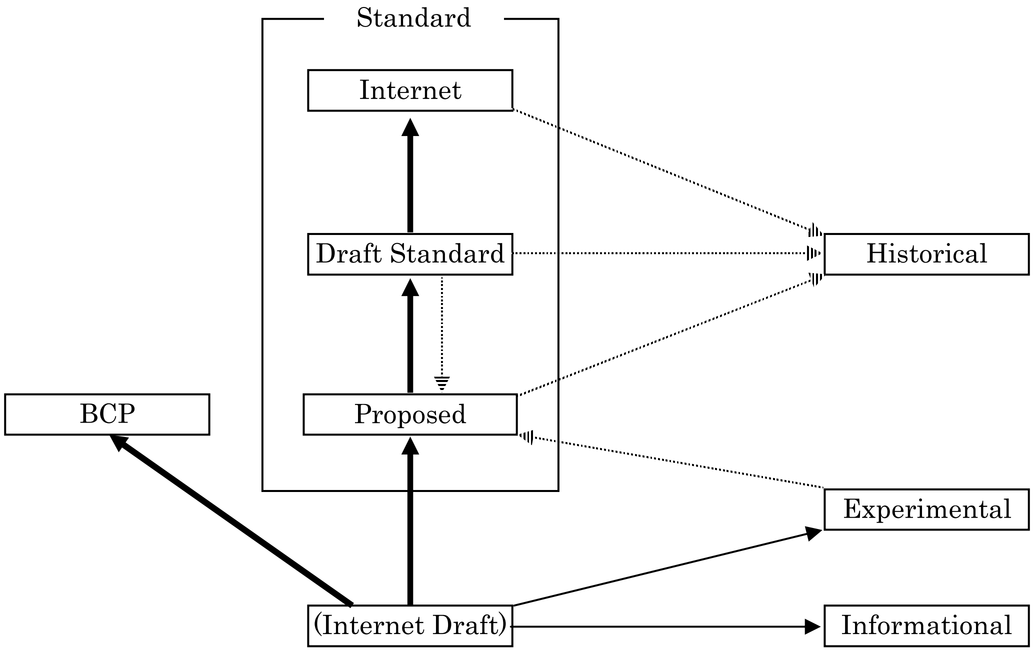
➔ 2週間以上のWG Last Call

・ そのプロトコルを開発しているWGからIESGに要求

➔ IESGでレビュー：IESGはそのプロトコル案を他のカテゴリとして発行することも可能

➔ 2週間以上のIESG Last Call

➔ 特に異論がでなければRFCが発行され、新しい標準化ステータスが付与される



特許について

Standard Track のドキュメント発行を目指す場合、著者は抵触する可能性のある特許について、適切な努力をして知る限り報告することが奨励されている(義務ではないが)。Standard Track のドキュメントが何らかの特許に抵触する可能性が知られている場合、その特許がリーゾナブルな条件で差別なく使えることを明示しなければならない。

5-3-3-3 標準化活動方針について

当社で現在検討しているFEC関連プロトコルの標準化について、IETFでの活動方法についてDigital Fountain社Michael Lubyよりアドバイスを受けた。

どのような形で標準化を行うにせよWGチェアやArea Directorに事前のロビー活動を行うことが重要である。技術的によい提案であっても、うまく根回しを行っておかないと標準化することは難しい。今後のSEIからの標準化提案に関連しそうなWGは、MMUSIC、AVT、RMTの三つ。すべてトランスポート・エリアに属しており、そのエリア・ディレクタは Scott Bradner, HarvardとAllison Mankin。この二人とは何らかの接触を行っていく必要がありそう。

次回のIETFミーティングは7月13日から18日までウイーンにて。そこで発表するためには６月中旬までにInternet Draftを発行する必要あり。

IETFでの標準化活動にはDigital Fountain社も協力してくれる。

RTSP への FEC ネゴ機能追加

RTSP の標準化は MMUSIC WG の担当。RTSP の標準化は一応完了しており、既存部分への変更をせまる提案を通すのは難しい。RTSP は拡張性に富んでおり、その枠内での拡張を提案することが望ましい。MMUSIC WG のチェアは Joerg Ott, ipDialog と Collin Perkins, ISI。

RTP への FEC サポート機能追加

RTP/RTCP の標準化は AVT WG の担当。RTP には既に FEC をサポートするための機能が一部入っているが（RFC2733 など）あまり役に立たないだろう。

RTP パケットをカプセル化する形で FEC ヘッダをつけるのであれば、RMT で標準化を進めている FEC BB を利用することができる。この場合、AVT WG での標準化活動は不要。

クライアントからサーバへ RTCP により受信ステータスを通知し、それにより FEC の冗長度を変更することを考えるのであれば、AVT WG での標準化活動が必要。

AVT WG のチェアは Steve Casner, Packet Design と Colin Perkins, ISI であるが、今回の WG Meeting では Steve Casner が司会、そして各プレゼンへの質問と大活躍していた。AVT WG に提案を行う場合、まず Steve Casner に相談する必要がありそうである。

FEC に関するパラメータの標準化

FEC に関するパラメータ（end-to-end でのパラメータの交換フォーマット）は RMT WG が標準化を行っている。チェアは Roger Kermode, Motorola と Lorenzo Vicisano, Cisco であるが、DF 社の Michael Luby もアクティブに活動しているため、標準化活動が必要であればかなりの部分 Michael Luby に協力を仰げると考える。

5-3-3-4　RMT WGミーティング

新しい WG チャーター案

・シンプル・リプリケーション・サービス・サブグループの結成 → これについては議論の末に否決

ユニキャストを使ってネットワークの途中でコピー、小規模なマルチキャストの置き換え。

・RMT WG で開発するプロトコルから TRACK を落と、NORM と ALC のみ標準化 → 合意

・新しいマイルストーン案 → 一部変更

- 03 年３月 Congestion control building block を experimental としてサブミット。

- 03 年８月 NACK building block and protocol を experimental としてサブミット。

- 03 年８月 Simple unicast replication building block を experimental とし

てサブミット、混雑制御を含むことも検討。

- 03 年１２月いつどのように experimental を proposed standard としてサブミットするかを検討、合わせて WG チャーターの変更も検討する。

NORM についての最新情報

・ NORM (NACK Oriented Reliable Multicast) のドラフトの最新版 draft-ietf-rmt-pi-norm-06/bb-norm-05 が発行された。

- TBD はなくなり、また、ガイドラインにそった記述になった

- 最新ドラフトの仕様は落ち着いている

・ NORM を実装した NRL のコードがあり利用可能

ALC についての最新情報 - by M. Luby

・ ALC PI、LCT BB、FEC BB、FEC INFO が Experimental RFC (RFC3450-52) として発行された。

・ WEBRC BB は WG Last call がかかっており、間もなく IESG にサブミットされる。

・ Compact FEC BB の Internet Draft が発行された。

- Compact FEC BB とは FEC BB の一種で RFC3452 への追加ドキュメント。近く WG Last Call をかけ Experimental RFC として発行したい。

→ WG として特に異論はなく、計画どおりに Experimental RFC が発行されそう。

- FEC Encoding ID を二つ新規に規定 - どちらも RFC3452 の Last Call 等で要求のあったもの

ID 0: No coding - 相互接続テストに有効。

ID 130: ID 128 とほぼ同じだがよりコンパクトな仕様とした。FEC Payload ID を 8 バイトから 4 バイトに。

またオブジェクト長情報を不要にした -> これにより長さの確定しないオブジェクトの送信が可能に。

FDALC - by T. Paila

新たに FDALC (File Delivery for Asynchronous Layered Coding) のドラフトが発行された。

ALC を使うアプリケーションの一つとしてファイル転送の方式を規定したもの。

- ファイル転送セッションの概念を定義

- SDP (および SDPng) を使ったファイル転送セッションの定義の仕方を定義

・ 問題点 - FDT (File Delivery Table)の内容についてはチェック、変更が必要

・ FDALC は ALC を広めるためにも重要。

・ 今後のスケジュール案

- 03 年 7 月　r01 を発行。クリティカルな問題を解決したバージョン。

- 03 年 9 月　WG Last Call

- 03 年 12 月　IESG に Experimental としてサブミット。

- 04 年末に Proposed Standard にすることを目標とする

・WG としては FDALC を活動項目の一つとして認める方向

A simple Unicast replication service – by Rick Boivie, IBM T. J. Watson R.C.

・ これまで IP マルチキャストは巨大なグループに効率良くパケットを配信することに注力してきたが、VoIP、ビデオ会議などでは無数の小規模グループに効率良くパケットを配信することが重要である。

・ IBM では Xcast と呼ぶシステムを開発

- 既存のネットワークに最小限の追加で非常に多くの少人数グループへのマルチキャストを実現

- ネットワーク上に中継サーバを置く。

- 送信元は中継サーバに IP パケットを送る（宛先 IP アドレスは中継サーバ）。

- パケットの IP ヘッダの次に Xcast ヘッダを置き、そこに最終宛先となるノードの IP アドレスおよび UDP ポート番号を記述。

- 中継サーバは Xcast ヘッダを見てパケットをコピーしながら転送を行う。

・ 議論：　アプリケーションレベルでマルチキャストを検討するグループもあるが、そことの関係が不明確。既に解決されている問題を解こうとしている可能性もある。Xcast は信頼性について言及していないこともあり、RMT WG で標準化作業を行うことの是非も議論するべき。

・ まずはオープン・ディスカッションからということになり、RMT WG チャーター案からも RMT 標準化はさしあたり削除されることになった。

5-3-3-5　AVT WGミーティング

AVT WG 関連のドキュメント状況

　AVT WG はかなり活発に活動しており、最近 RFC になった、あるいは近く RFC になる可能性のあるものだけで下記のように十数本のドキュメントがある。FEC に関するドキュメントも出ており要チェック。

・ 最近 RFC として発行されたもの

- RFC3497 Payload format for SMPTE 292M (Proposed Standard)

・ RFC として発行待ちのドキュメント

- RTP Profile MIME Registrations

- SDP Bandwidth Modifiers for RTCP Bandwidth

- Payload Format for EVRC/SMV

- Payload Format for ETSI ES 201 108 DSR

- RTP Specification and A/V Profile

・ IESG にサブミットされたドキュメント

- Enhanced Ip/UDP/RTP Header Compression (IESG Last Call)

- Tunneling Multiplexed Compressed RTP

- Secure RTP

- Payload Format for MPEG-4

・ AVT WG Last Call 中、あるいは間もなく Last Call のドキュメント

- RTCP Feedback

- RTP Retransmission

- RTP Payload Format for Generic FEC with Uneven Level Protection（RFC2733 および RFC3009 の改訂版）

- RTP Payload Format for Unequal Erasure Protection

- RTCP Extended Report

SRTP の標準化状況

SRTP（Secure RTP）は IESG のレビュー段階に入っているが、前回の IEST で変更を要求された他、さらに幾つかの変更要求が最近出ている。主な変更点は以下のとおり。

・ SRTP NULL 認証を受け入れ可能な時、そのリスクについての記述を明確化

・ リプレイ攻撃についての記述追加

・ 80 ビット SRTCP 認証を強制

・ 80 ビット SRTP 認証をデフォルト化

・ エラーを許容する仕様を削除

・ 自動キー生成、管理機能を強制

・ SRTP の実装者からのフィードバックを反映

RTP/RTCP 拡張レポートの標準化状況

RTCP の SR（Sender Report）および RR（Receiver Report）の拡張仕様を定義しようとするもの。VoIP などのアプリケーションを想定して従来から定義されている以外の情報を送信することを可能にしている。FEC 用と明記される情報はないが、ここで定義される情報あるいは、この拡張仕様に従った新しい情報により FEC ストリームの受信状況をサーバに伝えることができる可能性がある。

・ 現在の仕様は draft-avt-rtcp-report-extns-03.txt

・ 02 版から 03 版では大きな変更はなく、まだペンディング事項も残るものの仕様は安定してきた。

RTP ペイロード・フォーマットの標準化状況

RTP ペイロードのフォーマットは従来からペイロードのエンコーディングの種別に応じて各種定義されてきたが、今回のミーティングでも

・ JVT ビデオ -- H.264 ビデオコーデック

・ 非圧縮ビデオ -- 標準解像度のテレビ映像

・ ATRAC-X

・ iLBC Speech -- Global IP Sound が開発したフリーウェアの音声圧縮アルゴリズム

・ RGL コーデック -- G.711 ロスレス・コーデック

・ ETSI ES-202-050 Distributed Speech Recognition Encoding

について標準化に向けたプレゼンが行われた。

その他

　ミーティングの最後に RTP over DCCP (Datagram Congestion Control Protocol) についての議論もあった。DCCP は信頼性のないパケットフローの確立、維持、切断を行い、また、パケットフローの混雑制御を実現するためのプロトコルで現在 DCCP WG で標準化に向けた作業が進められている。FEC の冗長度をダイナミックに変更するメカニズムを考える場合、DCCP を使う、あるいは DCCP WG で議論されているコンセプトにそった動きをさせる必要があるかもしれない。

5-3-3-6　インターネットのセキュリティ

　これはFEC関連の標準化に関連して、参考までにインターネットのセキュリティについて述べる。

インターネットのセキュリティについての問題点

・ インターネットはもともと Good guys だけがつながっていると認識され、セキュリティに関する考慮はなされていなかった。

・ インターネットが巨大化し、Good guys だけがつながっているという状況は変わってしまった。

・ インターネット全体を管理する主体は存在せず、中間のサービスには期待できないので、エンドシステム間でセキュリティを保つ解が必要である。

インターネットで行われている攻撃

・ 受動攻撃 -盗聴- データストリームの改変は行わず、パスワードやクレジットカード番号を盗むといった行為

・ 能動攻撃 - 悪意を持ってネットワークにデータを挿入する攻撃

- tcpnice - TCP のコネクションをスローダウンさせる

- DoS 攻撃 - 最近注目を浴びている攻撃 - 一般に阻止が難しい

・ ネットワークベース攻撃 -プロトコルの性質を利用したもの- DNS を利用したなりすましなど

・ 非ネットワークベース攻撃 - プロトコルや OS の実装バグを利用したもの- バッファオーバフローを利用した death of ping 攻撃など

セキュリティ確保の難しさ

・ セキュリティ確保は後ろ向きの作業

- ネットワークのユーザはセキュリティが確保されている時にはそれに気付かず、セキュリティが確保されなかった時にはじめて気付く

- 一般的なソフト開発とはことなる感覚が要求される

・ 守りが強くなると攻撃も強まる。攻撃者は常にシステムの弱点をついてくる

セキュリティの最大の課題は認証である

・ ネットワーク・セキュリティの第一歩は、パケットがどこから来たかを知ることにある

-［問題点］127.0.0.1 を送信元 IP アドレスに持つパケットは自ノードからのパケットと思い無条件に信頼してしまう実装が多い

・ では、どのようにパケットを認証すればよいのか？

- チェックサムでは悪意のパケット改変は検出できないので、HMAC（Hashed Message Authentication Code）を使用する → AH の使用など

- HMAC でもリプレイアタックには対処できない → シーケンス番号を使うなどの方法あり

・ 通信データの秘匿は、データの暗号化により実現することができる

インターネットでの暗号化

・ 従来型暗号アルゴリズムは一つの鍵で「暗号化」と「復号」が可能である（対称型暗号アルゴリズム)。

・ 暗号アルゴリズムは独自のものを開発するべきではなく、既製のものを利用するべき。

・ 対称型暗号アルゴリズムの標準として DES がある。これは標準となってから 25 年以上たち、最近の計算能力の向上により十分安全とは言えなくなった。DES のアルゴリズムが破られた訳ではない。

・ トリプル DES は十分強力であるが処理に時間がかかる。

・ 対称型暗号アルゴリズムの新しい規格として AES（Advanced Encryption System）が採用された。

- 鍵長は 128 ビット。今後 30 年以上にわたって安全性が保てると想定されている

セキュリティ確保の上での注意点

・ 暗号化／認証アルゴリズムはパフォーマンス上重大な問題を引き起こす可能性があるので十分注意が必要。

・ 対称型暗号／認証アルゴリズムを使用する場合、エンドシステム間で秘密の値を共有する（鍵交換）メカニズムが必要。

・ リプレイ攻撃への対処が必要。

・ すべてのシステムがすべてのプロトコルスタックを持つとの仮定は許されない。したがって、セキュリティ確保のため IPsec を使えとだけ言うのは妥当ではない。

鍵交換

・ 通信の両端が鍵を共有するため、秘密の値を交換する手段が必要。

・ ネットワークは安全ではないので、鍵をそのまま送信することはできない。

・ 鍵交換アルゴリズムとしては、IKE（RFC2409）や Kerberos（RFC1510）がある。

・ Man in the middle attack（認証を行うホスト間に入って通信を傍受、改変する攻撃）に注意。

・ 鍵はその性質から二つに分類される

- セッション鍵：セッション毎に定義される鍵で、短い期間で無効になる。

- 一般に人あるいはシステムを識別するための長期間有効な鍵

公開鍵暗号アルゴリズム

・ 対称型暗号アルゴリズムと違い暗号化時（公開鍵：Public Key）と復号時（秘密鍵：Private Key）で別の鍵を使用する。

・ 公開鍵暗号アルゴリズムは、一般に対称型暗号アルゴリズムよりも処理時間が長くなるので、大量のデータを暗号化するのには通常使われない。

・ 1976 年にはじめて発表された新しい技術。公開鍵を知っていても、そこから秘密鍵を求めることは不可能なので、鍵の配布が簡単なのが特長。

・ 秘密鍵で暗号化したデータのみが公開鍵で正しく復号できるのでデジタル署名にも使うことができる。

トランスポート・セキュリティとオブジェクト・セキュリティ

・ データ・フローを保護する方式と、データ自体を保護する方式がある。

・ Secure WEB では end-to-end でデータ・フローが保護される。

・ Secure Mail ではデータ自体が保護される。これはメールの場合、いくつものメールサーバ間をバッジ的に転送されることによる。

IETF で開発されたセキュリティ保護のためのプロトコル

・ IPSEC – IP 層でセキュリティを確保

- 各 IP パケットを保護する仕掛けを提供

- 低レベルでの標準 – パケットのインテグリティを保証する AH およびパケットのペイロードの暗号化を行う ESP

- 高レベルでの標準 – ISAKMP/IKE – 鍵管理システム

- ファイアウォール間の VPN 確立、あるいは end-to-end で使用可能

・ TLS – トランスポート層（TCP）でセキュリティを確保

- データストリームの暗号化、認証および一貫性の保証機能を提供

- TCP を使うアプリケーションでは TLS を使うのが適当であることが多い

・ SMIME および PGP/MIME

- 安全なメッセージングのための仕掛け

- 基本的には E メール用だが他の用途にも使用可能 – バケツリレー型の通信に適する

- SMIME は公開鍵基盤を利用、PGP/MIME は対称暗号アルゴリズムを利用

・ UDP を使うアプリケーションについては、セキュリティを保証する一般的な方法は確立されていない。

・ PKIX – 鍵管理システムと公開鍵暗号システム

その他のグループおよびセキュリティ技術

・ セキュアシェル（SSH）

- TLS プロトコルに類似、遠隔コマンド実行に関連した機能もあり

・ 侵入検出 –侵入された時にどのように検出するか

- パターンベース・メカニズム、シグニチュアベース・メカニズムなどがある

- まだ技術は十分確立されていない

- IETF は侵入検出システムの各要素が相互にうまく動作することを確実にしようとしている

・ DNS のセキュリティ

- 従来の DNS は信頼性に問題があり、なりすましや DoS 攻撃に容易に利用することが可能

- セキュア DNS

5-3-4　結論

自律適用的に誤り訂正機能のネゴシエーションを実施するための基本的なプロトコルに関して検討を進め、新たな通信制御プロトコルの素案を検討中である。

5-4　デファクト標準化を念頭においた試作

5-4-1　序論

デファクト標準化を念頭においた、順方向誤り訂正機能およびその選択制御機能のデバイス化と評価プラットフォームとしてのサーバ・クライアントおよびホームゲートウエイ等低速低価格中継装置の試作を行なう。

5-4-2　研究内容

Digital Fountain 社から LT Coding を改良しさらに誤り訂正能力を高めた Raptor 符号化・復号化ソフトウェアモジュールを導入、これにＲＴＰ／ＲＴＳＰプロトコルをベースとした上記誤り訂正機能・選択制御機能とネットワークインタフェースを組み込んだデバイスをＭＩＰＳ型ＣＰＵ上で開発し、このＭＩＰＳ－ＣＰＵを組み込んだ評価プラットフォームとして最新の MPEG2 および MPEG4 のエンコード・デコード機能を持つＳＴＢ（セットトップボックス）を試作した。

また、誤り訂正機能を持つサーバの試作として、市販ＶＯＤサーバへの上記誤り訂正符号化・復号化ソフトウェア・モジュールの実装試作を行った。さらに、誤り訂正機能を持たないサーバに対して、ネットワーク上で誤り訂正機能を選択的に付与する装置の試作開発を進めるため、必要なハードウェア構成の検討・導入とソフトウェア・モジュールの設計を進めた。

5-4-3　ライブ FEC 搭載機器の設計と試作

5-4-3-1　ライブ FEC サーバプラットフォームの基本設計

5-4-3-1-1　ライブ FEC サーバの実現方法の検討

（案１）RTSP Snoop によるパケット迂回方式

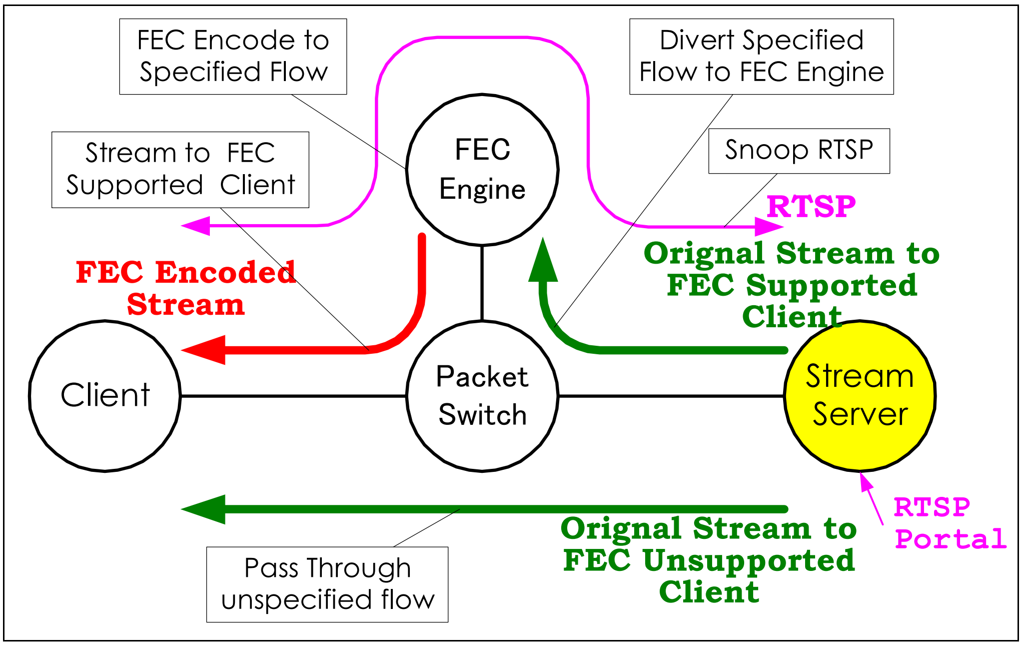


図 5-4-3-1　ライブ FEC システムモデル（RTSP Snoop 方式）

［1］処理概要

① クライアントはストリーミングサーバに RTSP で接続する。

② パケットスイッチは RTSP パケットを FEC Engine に迂回させる。

③ FEC Engine は、RTSP パケットからクライアントが FEC デコード機能をサポートしているか否かを判断し、必要なパラメータを抽出した後、パケットをパケットスイッチに戻す。

④ パケットスイッチは戻された RTSP パケットをストリーミングサーバに転送する（パケットの宛先アドレス・送信元アドレスは変更しない）。

⑤ パケットスイッチは FEC サポートクライアント宛ストリームのパケットを全て FEC Engine に転送する（RTSP から該当する RTP パケットのポート番号を取得する）。

⑥ FEC Engine は転送されたパケットに FEC エンコード処理を施し、パケットスイッチに戻す。

⑦ パケットスイッチは FEC エンコード処理を施されたパケットをクライアントに転送する（パケットの宛先アドレス・送信元アドレスは変更しない）。

⑧ FEC Engine は、パケットスイッチに FEC 非サポートクライアントに関連するパケットの迂回を停止させる。

［2］メリット

① ストリーミングサーバを改造する必要がなく、性能劣化もない。

② ストリーミングシステムのポータルサイトへの変更が不要

［3］課題

① RTSPパケットをFEC Engineへ迂回させ、かつ関連するRTPパケットもFEC Engineに迂回させることのできるパケットスイッチの調査

② FEC Engineのプラットフォーム検討と性能評価

（案２）パケット迂回方式

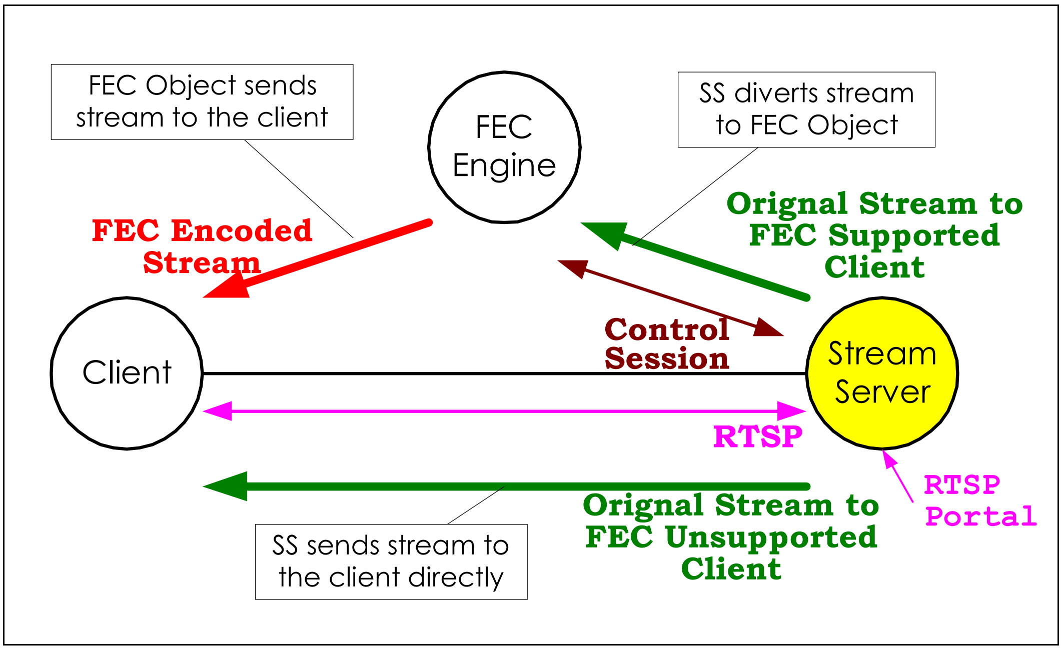


図 5-4-3-2　ライブ FEC システムモデル(パケット迂回方式)

[1] 処理概要

① クライアントはストリーミングサーバに RTSP で接続する。

② ストリームサーバは、クライアントが FEC デコード機能をサポートしている場合、FEC Engine にストリームを送信する。

③ FEC Engine は受信したパケットに FEC エンコード処理を施し、クライアントに転送する。

[2] メリット

① ストリーミングサーバの性能劣化が軽微

② FEC Engine には FEC エンコードが必要なストリームしか転送されない

③ ストリーミングシステムのポータルサイトへの変更が不要

[3] 課題

① 既存のストリーミングサーバを改造する必要がある。

② ストリーミングサーバと FEC Engine の間に制御用セッションが必要

③ ストリームのパケットの送信元アドレスがストリーミングサーバのものと異なる。

（案３）ストリーミングサーバ内蔵方式

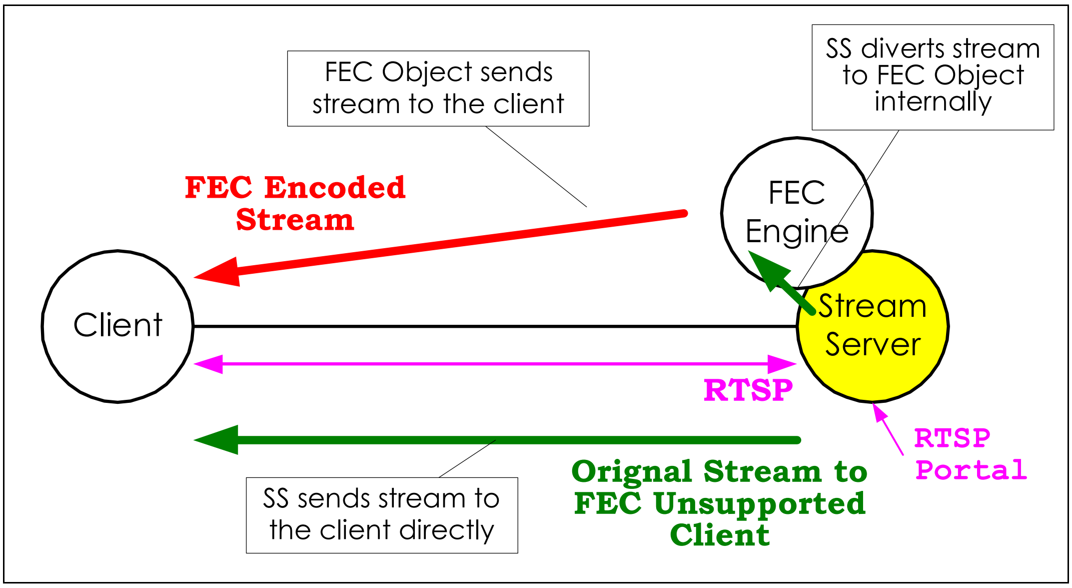


図 5-4-3-3　FEC システムモデル(サーバ内蔵方式)

［1］処理概要：

①　クライアントがストリーミングサーバに RTSP で接続する。

②　ストリーミングサーバはRTSP パケットからクライアントが FEC デコード機能をサポートしているか否かを判断する。

③　クライアントが FEC デコード機能をサポートしている場合、ストリーミングサーバはストリームを FEC Engine でエンコード処理したあと、クライアントに送信する。

④　クライアントが FEC デコード機能をサポートしていない場合、ストリーミングサーバは裏クライアントに直接ストリームを送信する。

［2］メリット：

①　新たに機器を増設する必要がない

②　ストリーミングシステムのポータルサイトへの変更が不要

［3］課題

①　ストリーミングサーバのパフォーマンス劣化が課題になる。既存のストリーミングサーバを改造する必要がある。

（案４）RTSP リダイレクト方式(その１)

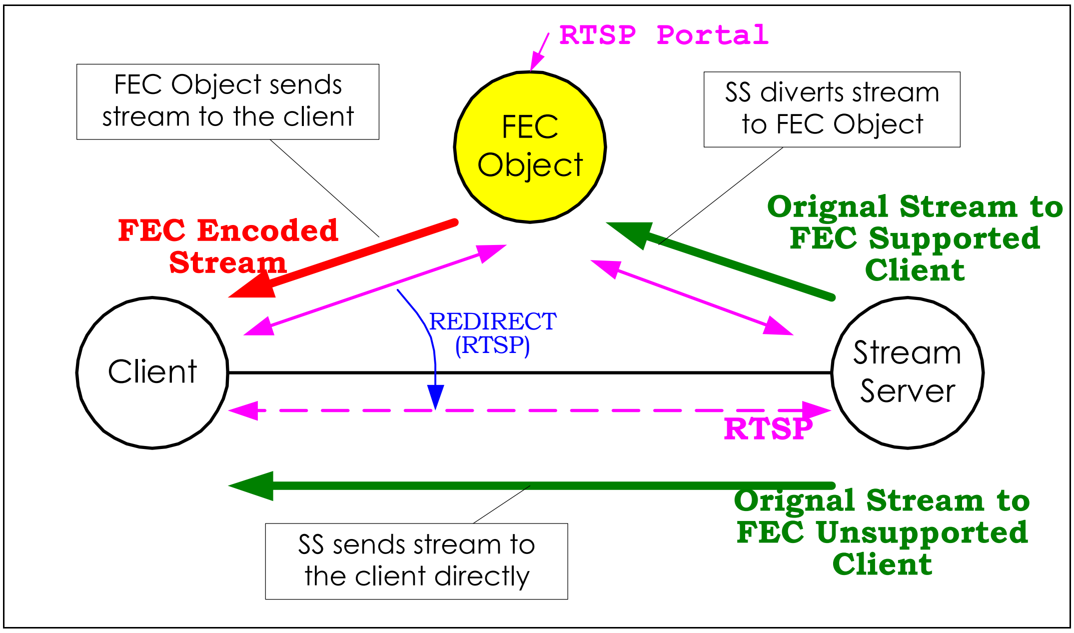


図 5-4-3-4　ライブ FEC サーバシステムモデル(RTSP リダイレクト方式)

[1] 処理概要

① クライアントは FEC Engine に RTSP で接続する。

② FEC Engine は、クライアントが FEC デコード機能をサポートしている場合、クライアントの代わりにストリーミングサーバに RTSP で接続する（RTSP プロキシとして動作する）。

③ ストリーミングサーバは FEC Engine にストリーム配信する。

④ FEC Engine は受信したパケットに FEC Encode 処理を施し、クライアントに配信する。

⑤ FEC Engine は、クライアントが FEC デコード機能をサポートしていない場合、クライアントへ RTSP リダイレクトコマンドでストリーミングサーバにリダイレクト要求する。

⑥ FEC デコード機能をサポートしていないクライアントは、RTSP リダイレクトコマンドに記述されているストリーミングサーバへ直接接続する。

[2] メリット

① ストリーミングサーバの性能劣化が軽微

② ストリーミングサーバへの改造が軽微

③ FEC Engine には FEC エンコードが必要なストリームしか転送されない

[3] 課題

① ストリームサーバから見たクライアントの IP アドレスが FEC Engine のものになっているため、クライアント認証の機能への改造が必要

② クライアントからみたストリームのパケットの送信元アドレスがストリー

ミングサーバのものと異なる。

③　クライアントは RTSP リダイレクトコマンドをサポートしている必要がある。(RFC2326 ではオプション)

④　ストリーミングシステムのポータルサイトの設定を変更する必要がある。

（案５）RTSP リダイレクト方式(その２)

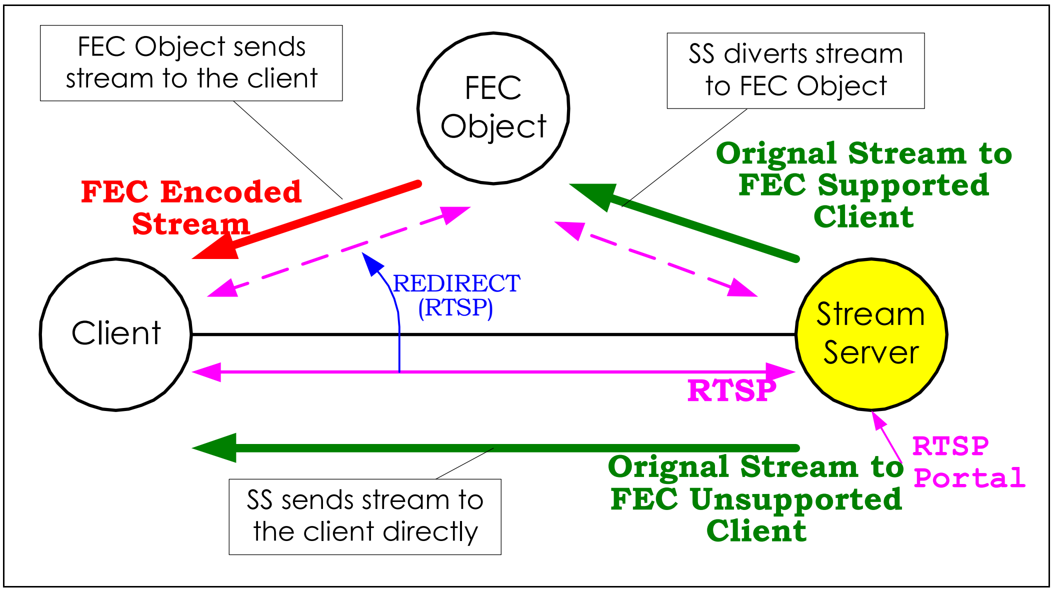


図 5-4-3-5　実証試験時のシステムモデル

［1］処理概要

① クライアントはストリーミングに RTSP で接続する。ストリーミングサーバは、クライアントが FEC デコード機能をサポートしている場合、クライアントへ RTSP リダイレクトコマンドで FEC Engine にリダイレクト要求する。

② FEC デコード機能をサポートしているクライアントは、RTSP リダイレクトコマンドに記述されている FEC Engine へ直接接続する。

③ FEC Engine は、クライアントが FEC デコード機能をサポートしている場合、クライアントの代わりにストリーミングサーバに RTSP で接続する（RTSP プロキシとして動作する）。

④ ストリーミングサーバは FEC Engine にストリーム配信する。

⑤ FEC Engine は受信したパケットに FEC Encode 処理を施し、クライアントに配信する。

⑥ ストリーミングサーバは FEC デコード機能をサポートしていないクライアントに直接ストリーム配信する。

［2］メリット

① ストリーミングサーバの性能劣化が軽微

② FEC Engine には FEC エンコードが必要なストリームしか転送されない

③ ストリーミングシステムのポータルサイトの設定を変更する必要がない。

［3］課題

① ストリームサーバから見たクライアントの IP アドレスが FEC Engine のものになっているため、クライアント認証の機能への改造が必要

② クライアントからみたストリームのパケットの送信元アドレスがストリー

ミングサーバのものと異なる。

③　ストリーミングサーバは RTSP リダイレクトコマンドをサポートしている必要がある(RFC2326 ではオプション)。

5-4-3-1-2　システム構成と動作概要

ライブ FEC サーバ実現方法の検討の結果、ライブ FEC サーバを用いたストリーミングシステムは、図 5-4-3-6 のようなシステム構成がよいと考える。

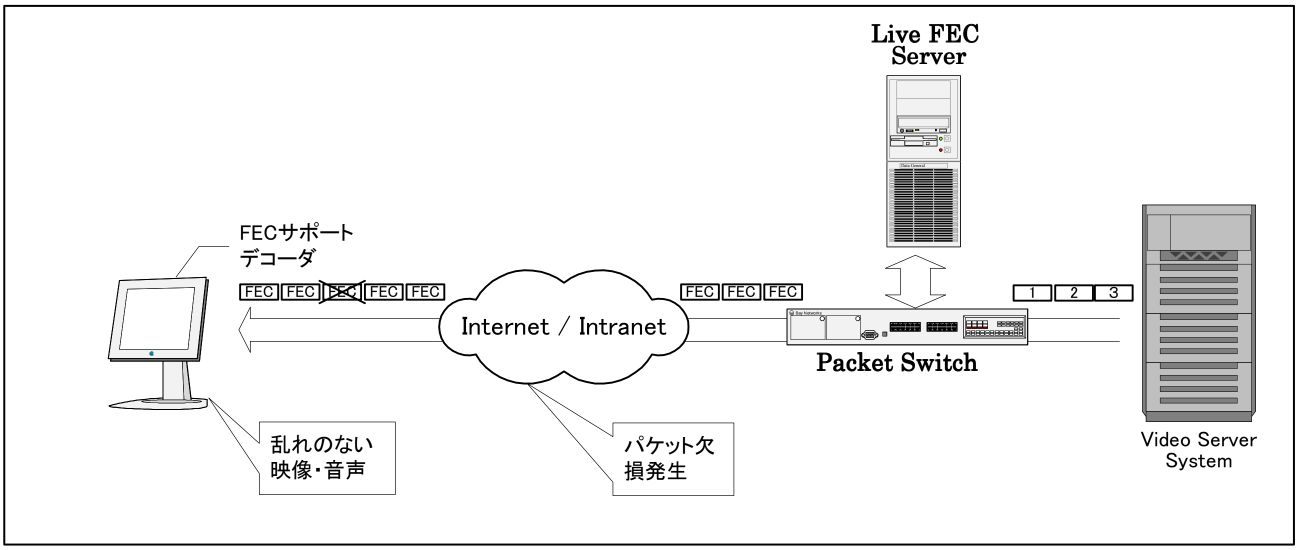


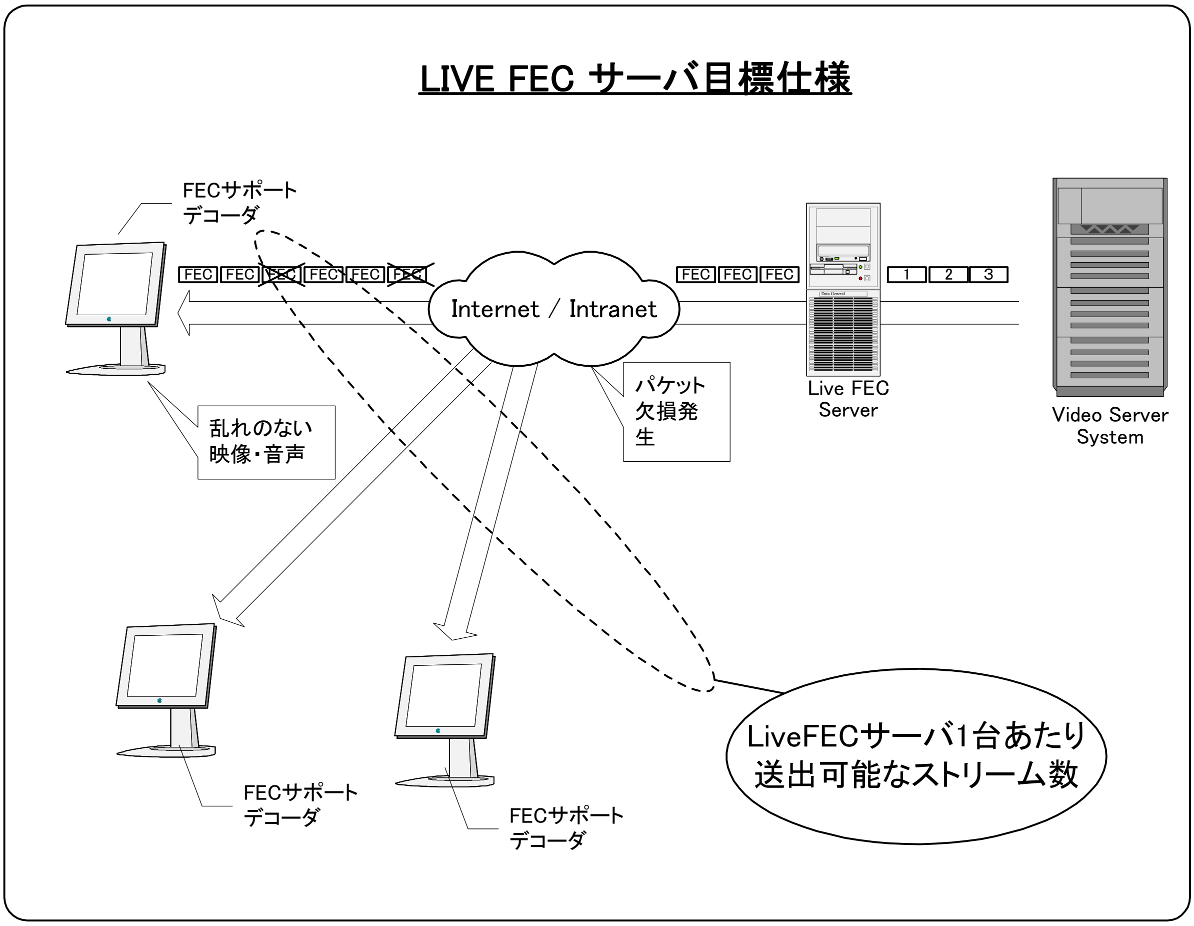
図 5-4-3-6　ライブ FEC サーバのシステム構成

5-4-3-1-3　ライブ FEC サーバ目標仕様の検討

ライブ FEC サーバでは、一台で可能な限り多くのビデオストリームを同時に FEC エンコード処理できることが望ましい。

本評価では、高性能化の著しい市販のアプライアンスサーバを用いて、一台で何本のストリームを処理できるか調査することで、ライブ FEC サーバの目標仕様を定める。

＜目標とするシステム構成＞



5-4-3-1-4　FEC 中継サーバ試作

ライブ FEC サーバはハードディスクを OS 起動以外に必要とせず、かつ屋内で使用されるものであるため、要求される使用環境条件は汎用のサーバマシンと同等でよい。また、PC ベースのサーバ(以下アプライアンスサーバ)の性能は目覚しい勢いで向上している一方で低価格化の傾向にある。

ライブ FEC サーバのプラットフォームとして市販のアプライアンスサーバを活用できれば、要求される性能に合わせて、プラットフォームを選べば良いため、専用のハードウェアを開発するコストを削減できる。

そこで、市販のアプライアンスサーバでライブ FEC サーバを構築した場合、どの程度の中継性能が実現できるかを調査するため、FEC 処理のみ行うソフトウェアを試作し、評価試験を行う。

（1）ライブ FEC サーバ評価用装置構成

今回評価に使用したアプライアンスサーバの諸元を表に示す。

①　サーバＡ（１ＣＰＵ）

	スペック	備考
ＣＰＵ	Intel(R) Pentium(R) 4 CPU	
ＣＰＵクロック	2.40GHz	
ＣＰＵ個数	1	
ＣＰＵキャッシュ	512 KB	
メモリ	512 MB	
ディスク	IDE 80 GB	基本的にはディスク装置は不要であるが、今回の評価実施時はログ出力用に使用している
ネットワーク	1000 Base-T	

②　サーバＢ（２ＣＰＵ）

	スペック	備考
ＣＰＵ	Intel(R) Xeon(TM) CPU	
ＣＰＵクロック	2.80GHz	
ＣＰＵ個数	２	実際のＣＰＵ数は２であるが、Xeon プロセッサを使用しているため、ＯＳからは４ＣＰＵと認識される（ハイパースレッディングテクノロジーによる）
ＣＰＵキャッシュ	512 KB	
メモリ	1 GB	
ディスク	SCSI 32 GB	基本的にはディスク装置は不要であるが、今回の評価実施時はログ出力用に使用している
Ｅｔｈｅｒｎｅｔ	1000 Base-T × ２I/F	２I/F を使用する試験も実施する。

③　ＭＰＥＧエンコーダ

ＭＰＥＧエンコーダに関しては、住友電気工業株式会社製のＶＳ－１００を複数台使用することにより、性能評価を行う。

④　模擬受信端末

模擬受信端末には、上述の①サーバＡおよび②サーバＢにて模擬受信用のソフトウェアを開発することにより、性能評価を行う。

したがって、上述の①サーバＡの性能評価を行う場合は、②サーバＢを模擬受信装置として用い、②サーバＢの性能評価を行う場合は、①サーバＡを模擬受信装置として用いる。

（２）ライブ FEC サーバソフトウェア

今回試作したライブ FEC サーバソフトウェアの１ストリームに対する処理概要を図に示し、以下に説明を加える。また、試作のライブ FEC サーバソフトウェアは、下記の図の処理が複数ストリームに実施できるように設計した。

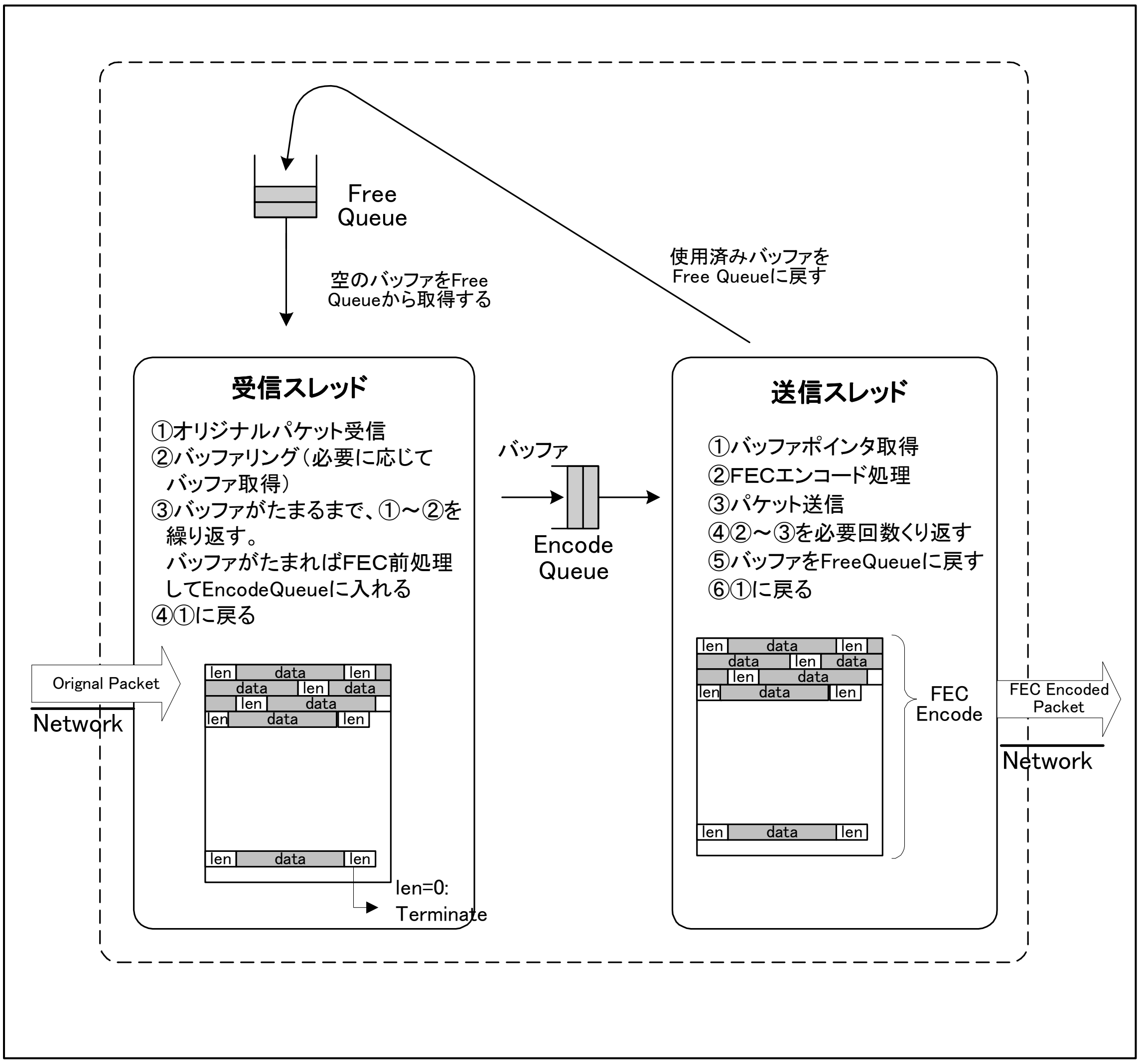


図 5-4-3-7　ライブ FEC サーバソフトウェア構成

本ソフトウェアは 1 つの入力ストリームにつき、受信と送信のそれぞれのスレッドを使用するマルチスレッド構成とした。次頁以降に、それぞれのスレッドの処理概要を示す。

（3）模擬受信ソフトウェア

今回試作した模擬受信ソフトウェアの処理概要を下記の図に示し、以下に説明を加える。また、模擬受信ソフトウェアでは、受信デコード処理が１ストリームにのみ実施できるように設計した。

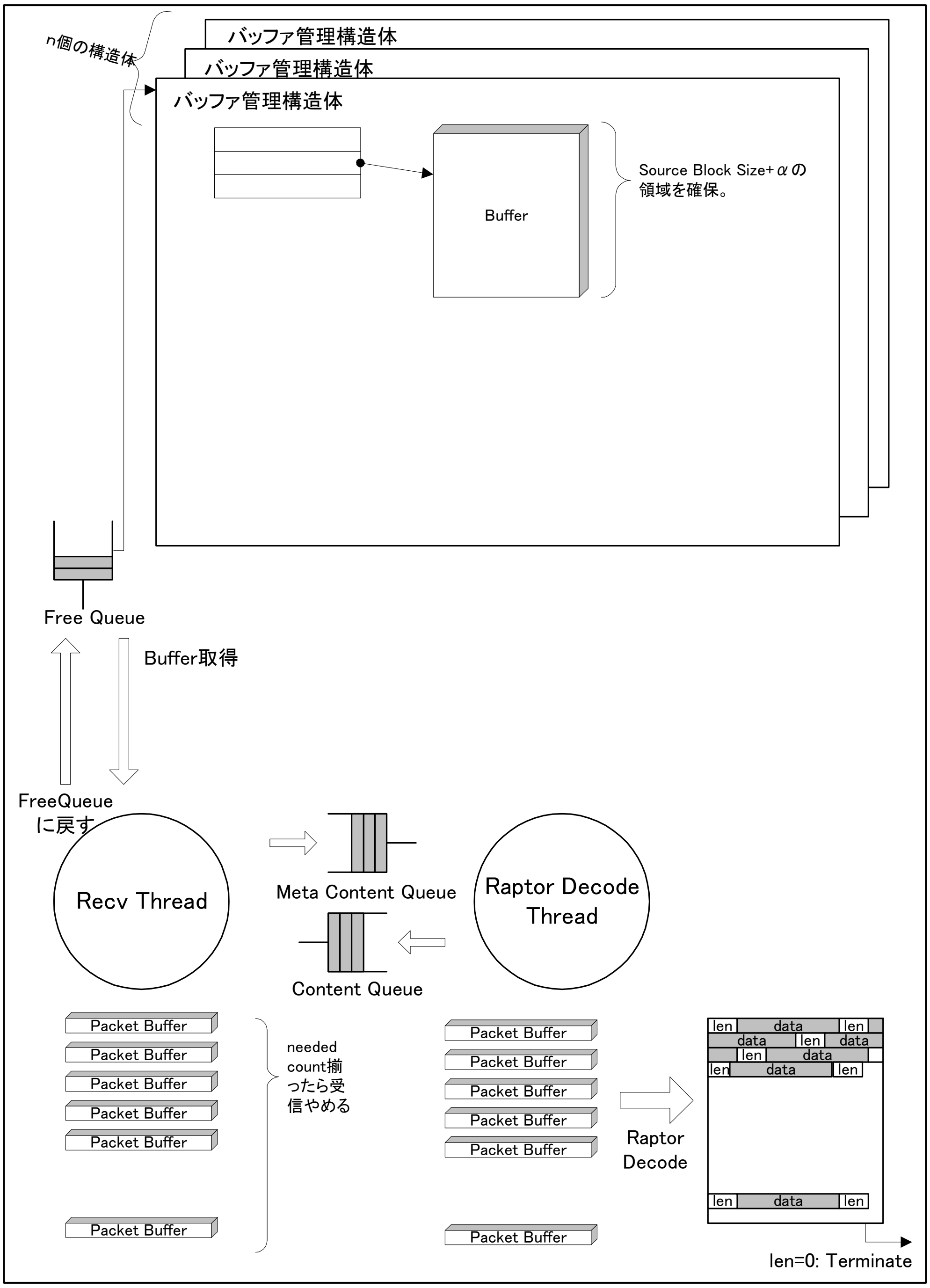


図 5-4-3-8　模擬受信ソフトウェア構成

5-4-3-1-5　ライブ FEC サーバ性能評価試験

　今回試作したライブ FEC サーバの性能評価試験は、以下の図の構成にて行った。また、各アプライアンスサーバに関しては、「ライブ FEC サーバ評価用装置構成」にて記述したスペックである。

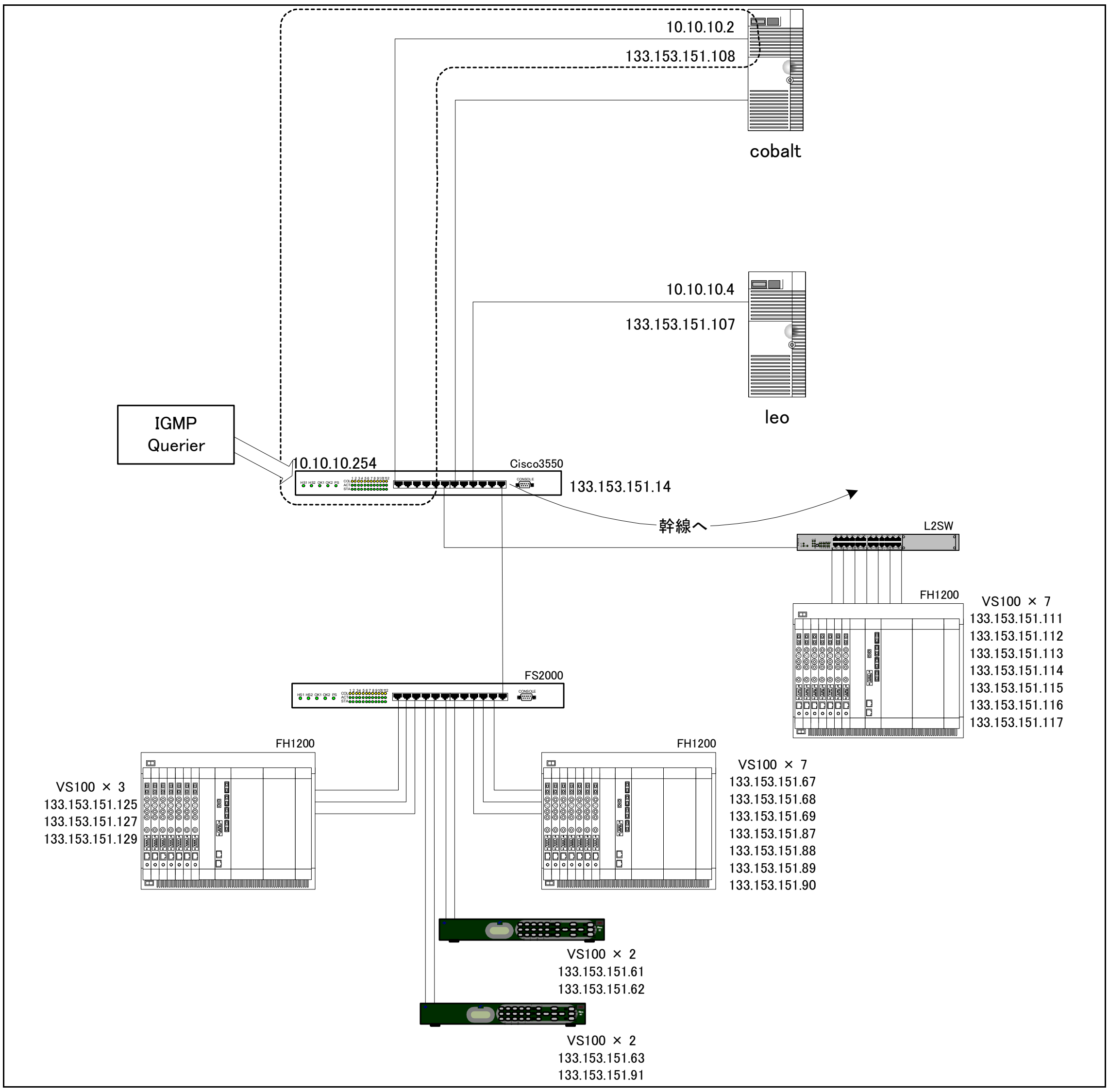


図 5-4-3-9　性能評価試験構成

マシン名	ライブ FEC サーバ評価用装置構成	IP アドレス
cobalt	サーバ B(2-CPU)	133.153.151.108
leo	サーバ A (1-CPU)	133.153.151.107

（1）パケット中継性能評価試験

本試験は、ライブ FEC サーバ性能評価試験を行うに先立ち、「ライブ FEC サーバ評価用装置構成」にて記述したアプライアンスサーバが通常の中継を行う場合のパケット送受信性能に関して評価を行う。この試験により、評価用アプライアンスサーバによる FEC エンコーディング時の性能劣化などを調査することが可能となる。

①サーバ B（2-CPU、単一ネットワークインタフェース）

以下の構成で、性能評価を行った。

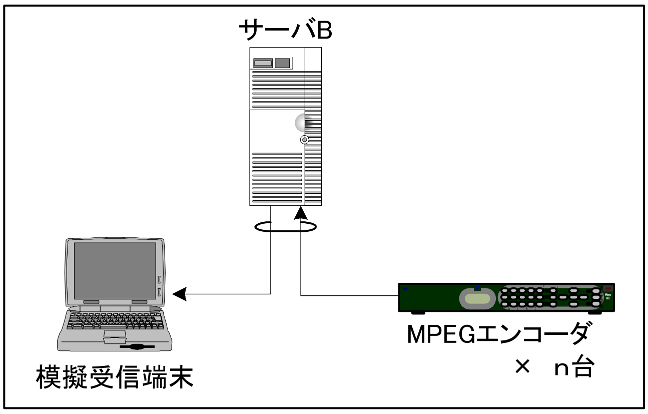
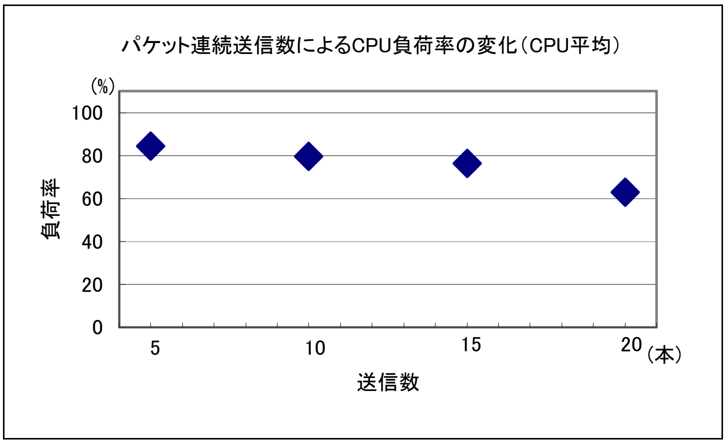


表 5-4-3-1　パケット中継性能評価試験結果

ストリーム数	パケット送信数	中継サーバ		模擬受信端末
		パケット欠損有無	CPU 負荷率(%)	パケット欠損有無
１７	５	２回／１５分	84.4	なし
	１０	２回／１５分	79.6	なし
	１５	１回／１５分	76.3	なし
	２０	８回／１５分	63.0	４回／１５分

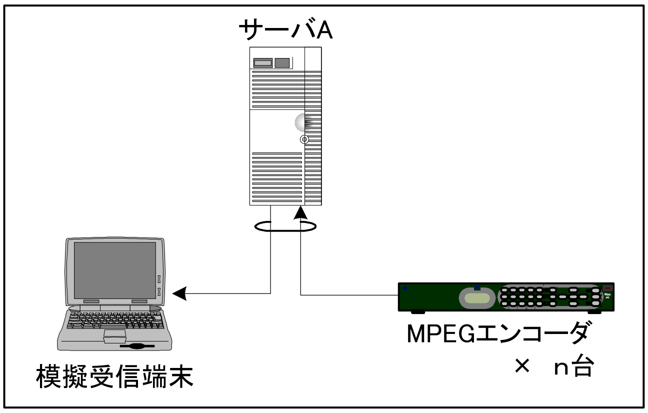


（２）ライブ FEC サーバ性能評価試験

本試験は、パケット受信性能評価試験と同様の構成にてライブ FEC サーバの中継性能を評価することにより、FEC エンコードを実施した際に生じるサーバの負荷状況を判断し、ライブ FEC サーバに必要なスペックのアプライアンスサーバを検討する。

①　サーバ A(1-CPU、単一ネットワークインタフェース)

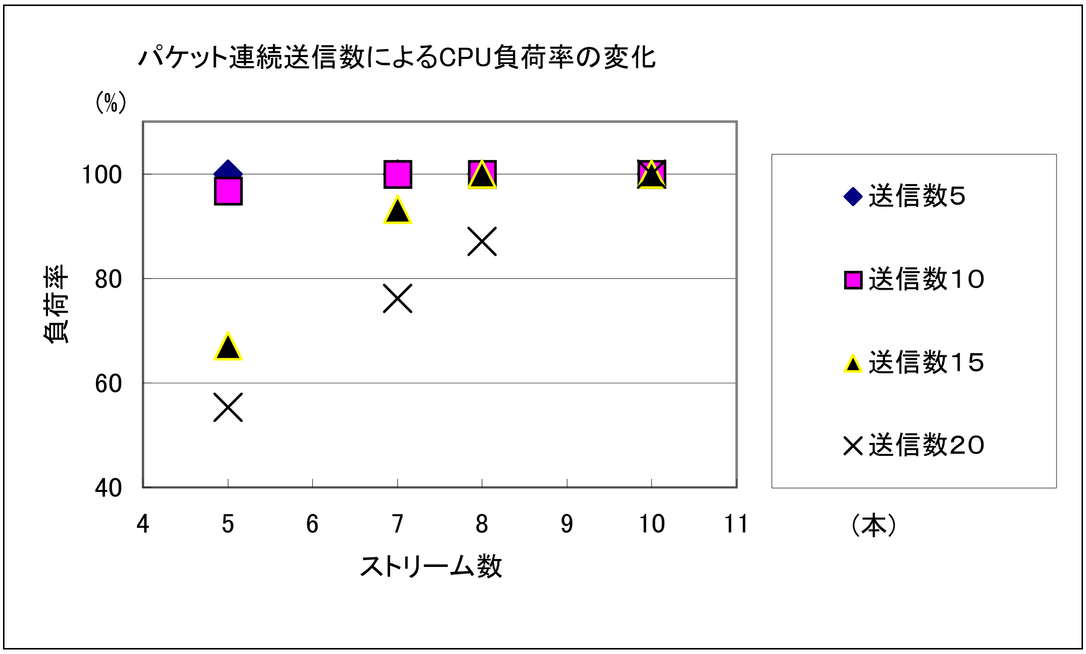
以下の構成で、性能評価を行う。



【評価結果】

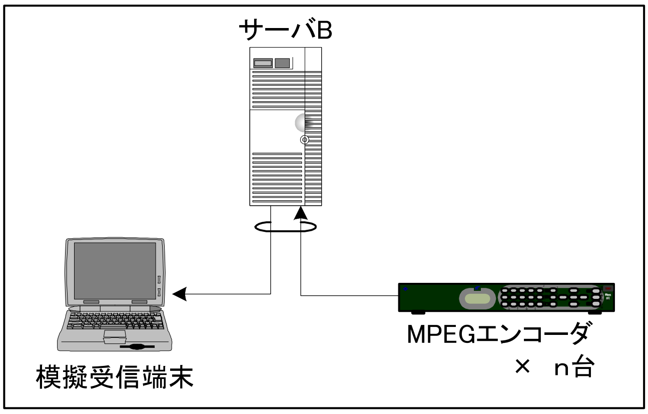
表 5-4-3-2　ライブ FEC サーバ性能評価試験結果
(Pentium 4-2.4GHz 1CPU の場合)

ストリーム数	パケット送信数	中継サーバ		模擬受信端末
		パケット欠損有無	CPU 負荷率	ブロック欠損有無
5	5	頻繁に欠損する	100.0	頻繁に欠損する
	１０	頻繁に欠損する	96.8	頻繁に欠損する
	１５	２回／３０分	67.1	頻繁に欠損する
	２０	１回／３０分	55.3	５回／３０分
7	5	頻繁に欠損する	100.0	頻繁に欠損する
	１０	頻繁に欠損する	100.0	頻繁に欠損する
	１５	なし	93.2	５回／３０分
	２０	なし	76.2	２回／３０分
8	5	頻繁に欠損する	100.0	頻繁に欠損する
	１０	頻繁に欠損する	100.0	頻繁に欠損する
	１５	頻繁に欠損する	100.0	頻繁に欠損する
	２０	１回／３０分	87.1	６回／３０分
１０	5	頻繁に欠損する	100.0	頻繁に欠損する
	１０	頻繁に欠損する	100.0	頻繁に欠損する
	１５	頻繁に欠損する	100.0	頻繁に欠損する
	２０	頻繁に欠損する	100.0	頻繁に欠損する



② サーバB（2CPU、単一ネットワークインタフェース）

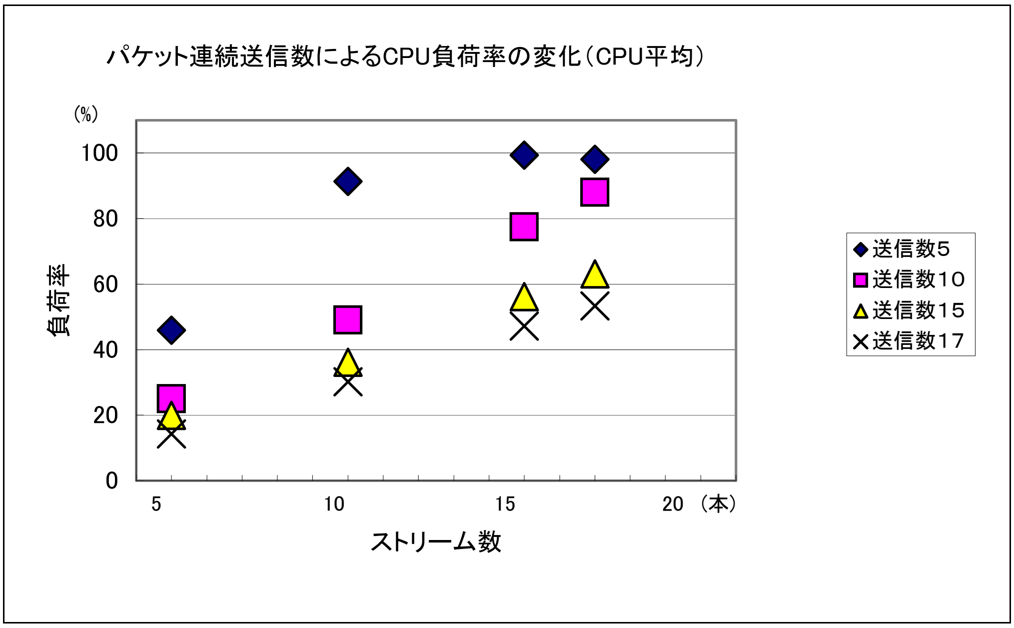
以下の構成で、性能評価を行う。



【評価結果】

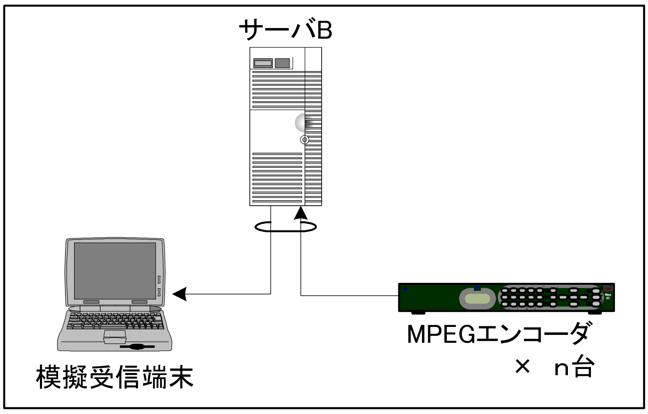
表 5-4-3-3　ライブ FEC サーバ性能評価試験結果
(Xeon 2.8GHz ２CPU の場合)

ストリーム数	パケット送信数	中継サーバ		模擬受信端末
		パケット欠損有無	CPU 負荷率(%)	ブロック欠損有無
５	５	２回／３０分	45.9	なし
	１０	７回／３０分	25.0	なし
	１５	３回／３０分	20.0	なし
	２０	４回／３０分	14.2	なし
１０	５	８回／３０分	91.4	なし
	１０	３回／３０分	49.0	なし
	１５	１２回／３０分	36.1	なし
	２０	５回／３０分	30.2	なし
１５	５	頻繁に欠損する	99.3	頻繁に欠損する
	１０	１回／３０分	77.5	なし
	１５	６回／３０分	56.2	なし
	２０	６回／３０分	47.2	なし
１７	５	頻繁に欠損する	99.5	頻繁に欠損する
	１０	頻繁に欠損する	94.5	なし
	１５	頻繁に欠損する	70.6	なし
	２０	９回／３０分	62.0	なし



③　サーバＢ（2CPU、2 ネットワークインタフェース）

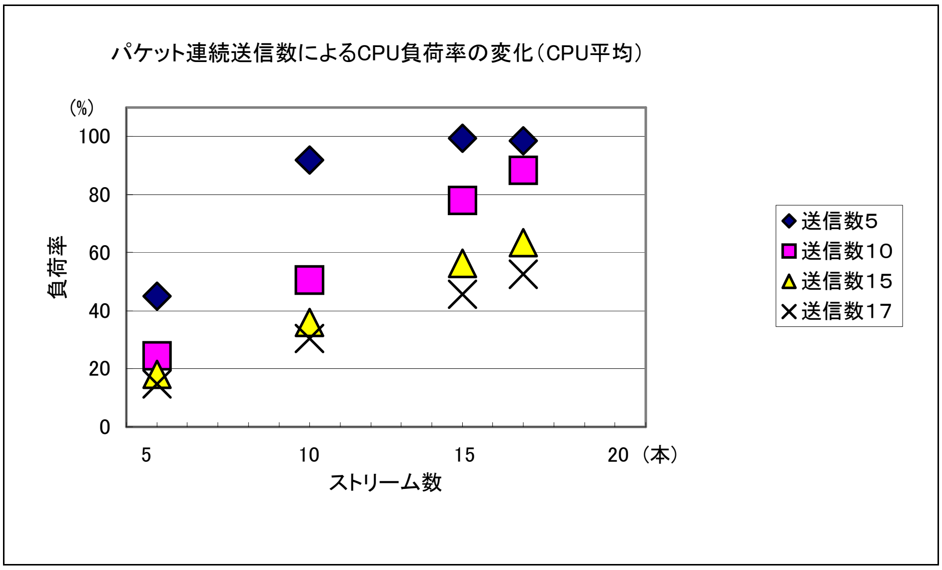
以下の構成で、性能評価を行う。



【評価結果】

表 5-4-3-4　ライブ FEC サーバ性能評価試験結果
（Xeon 2.8GHz ２CPU の場合）

ストリーム数	パケット送信数	中継サーバ		模擬受信端末
		パケット欠損有無	CPU 負荷率	パケット欠損有無
５	５	５回／３０分	45.1	１回／３０分
	１０	６回／３０分	91.9	１回／３０分
	１５	４回／３０分	99.4	なし
	２０	６回／３０分	98.7	なし
１０	５	５回／３０分	24.6	１回／３０分
	１０	１２回／３０分	50.5	１回／３０分
	１５	６回／３０分	78.0	１回／３０分
	２０	１２回／３０分	100.0	２回／３０分
１５	５	頻繁に欠損する	18.1	頻繁に欠損する
	１０	５回／３０分	35.9	なし
	１５	４回／３０分	56.4	なし
	２０	１１回／３０分	74.2	２回／３０分
１７	５	頻繁に欠損する	98.6	頻繁に欠損する
	１０	１０回／３０分	88.3	１回／３０分
	１５	なし	63.5	１回／３０分
	２０	なし	52.5	なし



5-4-3-2　評価結果の考察

今回の評価は、「パケット中継性能評価試験」および「ライブ FEC サーバ性能評価試験」を同一の構成でそれぞれ行い、FEC エンコード時のアプライアンスサーバでの処理負荷状況増加を確認する事にあった。

表 5-4-3-1～4 の実験結果では、処理するストリーム数が増えるにしたがって、発生するパケット欠損が増えている。ストリームソースとして使用した MPEG エンコーダから出力されるパケットにはパケット欠損が見られなかったことから、パケット欠損は、ライブ FEC サーバかネットワーク機器(Cisco3550)のいずれかで発生していると考えられる。今回の実験では実験機材の問題からいずれで落ちているかを厳密に突き止めることはできなかったため、今後調査する必要がある。

今回のライブ FEC サーバ性能評価試験では、ライブ FEC サーバ側のパケット欠損がストリーム本数にかかわらず 30 分間に 10 回以下で定期的な欠損でないもの、かつ模擬受信端末側でのパケット欠損がないものを、「中継可能」とした。

表 5-4-3-1 ならびに表 5-4-3-3 からわかるように、実際の FEC 中継時でも通常の中継と比べて CPU の処理負荷はほとんど変わらず、受信フレームに対して正常に FEC エンコードを行い、送信出来るという結果となった。従って、今回用いた FEC アルゴリズムは十分高速であると言える。

また、表 5-4-3-1～4 からわかるように、CPU 負荷率はパケットの連続送信数を減らすと高くなっている。これは連続して送信するパケットの数が少なければ、パケットインターギャップを挿入する頻度が増え、その結果 CPU に対するオーバヘッドが大きくなることを示している。

CPU 数と処理性能の関係については、表 5-4-3-2 と表 5-4-3-3 の結果が示すように、サーバ A では７本、サーバ B は１７本と、サーバ B の方が処理可能なストリーム数が約２倍多いことから、CPU 数が多いほど処理性能が向上することがわかった。

ネットワーク I/F の数と処理性能の関係については、表 5-4-3-3 と表 5-4-3-4 の比較からわかるように、２インタフェースの方が高速であった。これは、いずれも全二重通信ではあるが、単一のネットワークでは、ネットワークインタフェースやドライバにおけるリソース競合が発生するためと考えられる。

今回の評価により、ライブ FEC サーバは、サーバ B のスペック【Intel(R) Xeon(TM) CPU 2.80GHz×2、1000 Base-T×2】のアプライアンスサーバ１台で、１７本のビデオストリームを同時 FEC エンコード処理できることがわかった。

5-4-3-3　課題

今後検討すべき課題について、まとめる。

（1）パケットインターギャップ

今回の評価作業を行う際に、ライブ FEC 中継サーバでパケット間のギャップを設けずバースト的に転送を行った場合に模擬受信端末にてデータの読み落としが発生した。この事は、実際のデコーダに対してもバースト的に中継装置よりデータを転送した際に高負荷状態になる事を意味している。したがって、ライブ FEC 中継サーバにて、ストリームの転送レートにあわせたパケットインターギャップの挿入が不可欠であると考えている。

ただし、今回のライブ FEC サーバ試作にて使用した OS は、リアルタイム系の OS ではない為に msec オーダーでの制御が不可能であるため今回は擬似的に1 msec のパケットギャップを設けている。今後の改善点としては、リアルタイム処理が可能な OS を使用して、ストリームの転送レートにあわせたパケットインターギャップを挿入する必要があると考えている。

（2）マルチ CPU への対応

今回の評価試験では、マルチ CPU 対応の OS を用いて OS 自体のスケジューリング機能に基づき、各 CPU で処理を決定していた。しかし、OS 自体でのスケジューリング機能では、各 CPU の処理を決定するにあたり各スレッドの実行時間を測定後に負荷分散を行うために起動直後では特定の CPU に負荷が発生し正常に中継出来ない場合があることが判明した。そのため、各スレッドの実行する CPU を起動時に予め決定できる機構が必要と考えている。

（3）重複パケットへの対応

この問題は、模擬受信装置または実際の FEC デコード装置の問題であるが、同一のパケットがネットワークの状態により、デコード装置に複数回到達した場合に FEC デコードが完了しないと言う問題が発生する。この問題は、同一パケットを FEC エンジン自体が必要とするパケット数とカウントされてしまい処理されることにより発生している。今後は、同一パケットを FEC エンジンに引き渡す前にチェックを行い不要としてカウントしないなどの処理が必要である。

（4）パケットの到着順の入れ替わりに関しての問題

この問題は、模擬受信装置または実際の FEC デコード装置の問題であるが、ネットワークの状態により、デコード装置に到着するパケットが多数に入れ替わった場合、FEC デコード処理が正常に完了しないという問題が発生する。

模擬受信装置が FEC デコードエンジンに対して、必要数に到達したフレームを引き渡す前に受信したブロック番号によりデータを蓄積する処理を行っているが、そのブロック番号毎の蓄積するための管理領域があふれる事により発生している。この対策として、ブロック番号の管理テーブルを拡大するなどの対応が必要であると考えている。

5-4-3-4　まとめ

今回の設計・評価作業により、設計の基本方針自体には問題ない事が確認できた。ただし、前述の課題などを踏まえて今後はこれらの課題の解決に対して取り組む方針である。以下に、それぞれの課題に対しての現在考えている設計方針を記述する。

（１）パケットインターギャップの実現方法

リアルタイム系の OS により、ライブ FEC サーバを実装する事により、msec オーダー以下のウェイトを可能とし、各パケットを送信後にウェイト処理を挿入する事により実現する。ただし、ストリームのビットレートはセッション毎に固定出来るものとして、構成定義により初期設定を行い運用する方針である。

（２）マルチ CPU への対応

現在使用している OS が提供している I/F には、あるスレッドがどの CPU で実行するか指定する方法は、存在しない。ただし、OS のカーネルのみが実行可能な I/F が提供されているようであり、今後は、その I/F を使用するために OS が拡張用に提供しているモジュールと言う手法を用いて、カーネル部から今回作成しているライブ FEC サーバの送受信スレッドの動作 CPU を指定する方針で検討を進める。

5-4-4　結論

評価プラットフォームとしてのサーバ・クライアントおよびホームゲートウエイ等の中継装置の実験結果と、今後の課題についてまとめた。

5-5　総括

平成 14 年度の目標であった理論的な基礎検討は終了し、Digital Fountain 社の誤り訂正符号が有力であるとの結論を得た。平成 15 年度に向けた、シミュレーションモデルやプロトコルの検討も進み、ホームゲートウエイ等の中継装置も試作段階にある。

参考資料、参考文献

[1] Michael G. Luby, "Information Additive Code Generator and Decoder for Communication Systems," US Patent No. 2001/0019310.

[2] M. Amin Schkrollahi, Soren Lassen, and Michael Luby, "Multi-Stage Code Generator and Decoder for Communication Systems," US Patent No. 2003/0058958.

（添付資料）

１　研究発表、講演、文献等一覧

・学会発表(査読なし)

メタコンテンツ技術を用いたマルチキャスト配信制御に関する検討

（平成１５年３月　電子情報通信学会 全国大会にて発表）

・論文発表

論文発表なし

３　再委託研究報告書

最新の高速低遅延順方向誤り訂正技術の国際的調査研究とネットワーク上の多段利用を想定したクライテリアの確立
(京都大学大学院 情報学研究科 システム科学専攻 高橋研究室)

ネットワークのモデル化とシミュレーション技術の開発ならびに高信頼性コンテンツ配信プロトコルの研究
(大阪大学大学院 情報科学研究科 情報ネットワーク学専攻 村上研究室)

以上