(19)日本国特許庁(JP)　(12)公開特許公報(A)　(11)特許出願公開番号
特開2009-27720
(P2009-27720A)
(43)公開日　平成21年2月5日(2009.2.5)

(51)Int.Cl.		F I		テーマコード（参考）
H04L 1/00	(2006.01)	H04L 1/00	B	5K014

審査請求 有　請求項の数 21　ＯＬ　外国語出願　（全 61 頁）

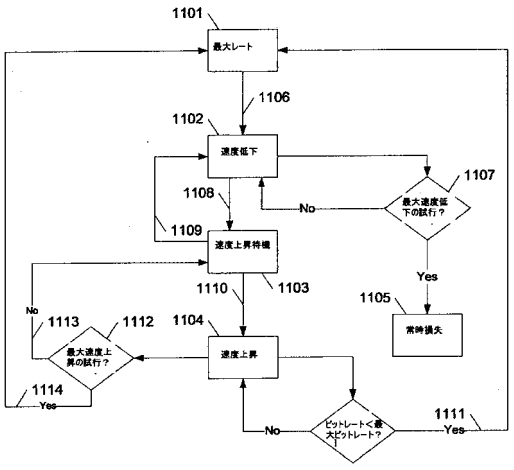
(21)出願番号　特願2008-189211 (P2008-189211)
(22)出願日　平成20年7月22日 (2008.7.22)
(31)優先権主張番号　60/951,399
(32)優先日　平成19年7月23日 (2007.7.23)
(33)優先権主張国　米国 (US)

(71)出願人　500080720
ポリコム・インコーポレイテッド
アメリカ合衆国　カリフォルニア州　９４５８８　プレザントン　ウィロウ・ロード　４７５０
(74)代理人　100070150
弁理士　伊東　忠彦
(74)代理人　100091214
弁理士　大貫　進介
(74)代理人　100107766
弁理士　伊東　忠重
(72)発明者　ステファン　ボッコ
アメリカ合衆国　マサチューセッツ州　０１８６７，レディング，ウォバーン・ストリート　１５０

最終頁に続く

(54)【発明の名称】輻輳回避と共に損失パケット回復を行うシステム及び方法

(57)【要約】
【課題】　損失パケット回復と輻輳回避とを統合することを目的とする。
【解決手段】　パケット交換ネットワークでメディア会議を行うときに、損失パケットを克服して輻輳を回避する装置及び技術が記載される。損失パケットの問題を回避するために、受信機が冗長情報から何らかの損失パケットを再構成することを可能にする冗長情報がメディアストリームに挿入される。輻輳回避技術は、メディアストリームのビットレートを調整し、輻輳によるパケット損失のないサポート可能な最高のビットレートを見つけることを含む。ビットレートを高いレートに増加させるときに、更なるビットは、損失パケットの回復に使用される冗長情報から生じ、これにより、ネットワーク輻輳により生じた何らかの損失パケットはビットストリームに悪影響を及ぼさない。
【選択図】　図１１



【特許請求の範囲】
【請求項１】
損失のあるネットワークを通じて送信されたメディアストリームの破損を回避する方法であって、
損失パケット回復アルゴリズムを前記メディアストリームに適用し、前記損失パケット回復アルゴリズムは、前記メディアストリームを含む送信データストリームに冗長情報を挿入し、１つ以上の損失パケットの再構成を可能にし、
輻輳回避アルゴリズムを前記送信データストリームに適用し、前記輻輳回避アルゴリズムは、前記送信データストリームに挿入された冗長情報の量を増加させることにより、前記送信データストリームのデータレートを一時的に増加させ、前記ネットワークが高いデータレートをサポートすることができるか否かを決定することを含む
ことを有する方法。
【請求項２】
前記冗長情報は、１つ以上の回復パケットを有する、請求項１に記載の方法。
【請求項３】
回復パケット数は、RTCP受信レポートから決定されたチャネル損失率に依存する、請求項２に記載の方法。
【請求項４】
ネットワーク輻輳を回避するために必要に応じて総計ビットレートを低減することを更に有する、請求項１に記載の方法。
【請求項５】
前記損失パケット回復アルゴリズムは、前方消失訂正符号化を含む、請求項１に記載の方法。
【請求項６】
前記損失パケット回復アルゴリズムは、リードソロモン符号化を含む、請求項１に記載の方法。
【請求項７】
前記メディアストリームは、ビデオデータを有する、請求項１に記載の方法。
【請求項８】
ネットワークでメディアデータを送信するメディア符号化装置であって、
メディアエンコーダと、
前記メディアエンコーダからの出力を受信し、リアルタイム送信プロトコルに従って前記メディアエンコーダの出力を複数の発信元パケットとしてパッケージ化するように接続されたRTP送信機と、
前記ネットワークからチャネル情報を受信するように接続されたRTCPモジュールと、
前記RTCPモジュールから前記チャネル情報を受信するように通信可能に結合され、前記チャネル情報から使用されるLPR保護パラメータを決定するように適合され、前記LPR保護パラメータをLPRパケット化器に提供するように通信可能に結合されたLPRモード判定モジュールと、
RTP送信機からの出力を受信し、前記LPRモード判定モジュールから受信したパラメータを使用して前記RTP送信機の出力を複数のLPRデータパケットとして再パケット化するように接続されたLPRパケット化器と、
前記LPRパケット化器及び前記メディアエンコーダに結合され、前記LPRパケット化器及び前記メディアエンコーダから受信した情報に基づいてLPR回復パケットを生成するように適合され、前記LPRデータパケットと前記LPR回復パケットとを受信機に送信するように適合されたLPR回復パケット生成器と
を有するメディア符号化装置。
【請求項９】
前記メディアエンコーダはビデオエンコーダである、請求項８に記載のメディア符号化装置。

【請求項１０】
前記LPRモード判定モジュールは、
保護期間を決定し、
保護期間毎の合計パケット数を決定し、
保護期間毎の回復パケット数を決定する
ことによりLPR保護パラメータを決定する、請求項８に記載のメディア符号化装置。
【請求項１１】
各LPRデータパケットは、唯一の発信元パケットからの情報を含む、請求項８に記載のメディア符号化装置。
【請求項１２】
前記回復パケットのペイロードは、リードソロモン符号化を使用して符号化される、請求項８に記載のメディア符号化装置。
【請求項１３】
パケット型ネットワークで行われるメディア会議のLPR保護パラメータを決定する方法であって、
保護期間を決定し、
保護期間毎の合計パケット数を決定し、
保護期間毎の回復パケット数を決定する
ことを有する方法。
【請求項１４】
前記保護期間は、前記メディア会議のビットレートに従って決定される、請求項１３に記載の方法。
【請求項１５】
メディアレートの関数としてLPR保護パラメータを調整することを更に有する、請求項３に記載の方法。
【請求項１６】
前記メディアレートの関数としてLPR保護パラメータを調整することは、グループ毎のパケット数を動的に適合させることを有する、請求項１５に記載の方法。
【請求項１７】
前記メディアレートの関数としてLPR保護パラメータを調整することは、パケットサイズを動的に適合させることを有する、請求項１５に記載の方法。
【請求項１８】
前記メディアレートの関数としてLPR保護パラメータを調整することは、空のデータパケットを送信することを有する、請求項１５に記載の方法。
【請求項１９】
ネットワークで送信機からメディアデータを受信するメディア復号化装置であって、
前記ネットワークから、LPRデータパケットとLPR回復パケットとを含むLPRパケットを受信するように結合されたLPR回復モジュールと、
前記LPRデータパケットを受信し、何らかの損失データパケットを再生成するように適合されたLPR再生成器と、
前記LPR再生成器に通信可能に結合され、前記LPR再生成器から損失パケットのレポートを受信し、損失パケット情報を前記送信機に送信するRTCPモジュールと、
前記LPR再生成器に結合され、前記LPRデータパケットと前記再生成されたデータパケットとを受信し、含まれるメディア情報をデコードするように結合されたメディアデコーダと
を有するメディア復号化装置。
【請求項２０】
前記メディアデコーダはビデオデコーダである、請求項１９に記載のメディア復号化装置。
【請求項２１】

前記LPR回復モジュール及び前記LPR再生成器は、LPRアルゴリズムが使用中でないときにパススルーとして動作する、請求項１９に記載のメディア復号化装置。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、輻輳回避と共に損失パケット回復を行うシステム及び方法に関する。

【背景技術】

【０００２】

ますます、テレビ会議システムは、回線交換ネットワーク（PSTN及びISDN等）ではなく、パケット交換ネットワーク（インターネット等）を使用し始めてきている。テレビ会議にパケット交換ネットワークを使用することでの１つの問題は、全てのパケット交換ネットワークが或る程度のパケット損失を受けるという点にある。控えめなパケット損失であっても、ビデオ品質はかなり損なわれる。

【０００３】

パケット損失の問題に対する既存の対策は、様々な誤り隠蔽技術と画像リフレッシュ機構とを使用することを含む。有用ではあるが、これらの技術はしばしば不十分である。これらは、頻繁に特定のビデオコーデックと統合されており、これらの技術がコーデック技術の進展毎に改革されることを必要とする。

【０００４】

典型的には、パケット交換ネットワークは、少なくとも２つの基本的な種類のパケット損失を有する。ネットワークのレイヤ1及び2が回復されない情報を損失したときに、物理的損失が生じ得る。例えば無線リンク（例えば、802.11a、802.11b、802.11g及び802.11nのようなWiFiネットワーク）が使用されるときに、この種類の損失は一般的である。しかし、有線及び光ファイバリンクでも物理レイヤの損失が生じ得る。第２の種類のパケット損失は、ネットワーク輻輳から生じ得る。

【０００５】

この第２の種類のパケット損失を克服するために、或るテレビ会議装置は、様々な輻輳回避技術を使用している。１つのこのような技術は、2002年11月26日に出願された“System and Method for Dynamic Bandwidth Allocation for Videoconferencing in Lossy Packet Switched Systems”という題の米国特許出願10/305,485号に記載されている。この特許出願の全ての内容を援用する。このような技術は、“動的帯域割り当て（Dynamic Bandwidth Allocation）”又は“DBA”と呼ばれることがある。これらの種類のDBAアルゴリズムは、帯域を増加使用とすると常にパケット損失を取り込むという本質的なリスクがある。このことは、損なわれた媒体という結果になり得る。他の種類の輻輳回避技術は、VCONのPacketAsssistと、損失パケット回復又は輻輳制御に関する他の標準とを含む。例えば、RFC2733は、XOR（パリティ）パケットを使用する損失パケット回復方法を提供する。RFC3448は、RFC4340と同様に、ユニキャスト輻輳制御方法を提供する。

【０００６】

更に、前方消失訂正（forward erasure correction）及び前方誤り訂正（forward error correction）のような損失データ回復技術が、ネットワークのレイヤ1及び2で使用され得る。これらの技術もまた、RAIDディスク及びCDROM/DVDのような記憶媒体で使用されている。近年、3GPPは前方誤り訂正を標準化した。前方誤り訂正はまた、H.320を介して送信されるビデオストリームにも提供され得る。

【特許文献１】米国特許出願10/305,485号

【非特許文献１】RFC2733

【非特許文献２】RFC3448

【発明の開示】

【発明が解決しようとする課題】

【０００７】

これまで、（例えば前方消失訂正を使用した）損失パケット回復と輻輳回避とを統合す

る実現は行われていない。このような統合を含むアルゴリズムを記載する。
【課題を解決するための手段】
【０００８】
一態様では、本発明は、損失のあるネットワークを通じて送信されたメディアストリームの破損を回避する方法に関する。この方法は、損失パケット回復アルゴリズムをメディアストリームに適用することを含み得る。損失パケット回復アルゴリズムは、メディアストリームを含む送信データストリームに冗長情報を挿入することにより動作し得る。冗長情報は、前方消失訂正及び／又はリードソロモン符号化を使用して生成され得る。この方法は、輻輳回避アルゴリズムを送信データストリームに適用することを更に含み得る。輻輳回避アルゴリズムは、送信データストリームのデータレートを一時的に増加させ、ネットワークが高いデータレートをサポートすることができるか否かを決定することを更に含み得る。データレートは、送信データストリームに挿入される冗長情報の量を増加させることにより、一時的に増加し得る。
【０００９】
本発明はまた、損失のあるネットワークで受信したメディアデータを回復する方法に関し得る。メディアストリームは、データストリームと冗長情報とを有するデータストリームの一部として送信されている。この回復方法は、メディアストリームと冗長情報とを有する複数のパケットを受信し、受信パケットからメディアストリームの損失部分を再構成することを含み得る。
【００１０】
本発明はまた、前述のデータ回復及び輻輳回避を実行するように適合されたテレビ会議装置に関し得る。
【発明の効果】
【００１１】
本発明の実施例によれば、パケット損失を低減することが可能になる。
【発明を実施するための最良の形態】
【００１２】
前方消失訂正と輻輳回避との使用を組み合わせたネットワーク送信で、損失パケットの影響を克服する技術が開示される。これらの技術は、“損失パケット回復（Lost Packet Recovery）”又は“LPR”と呼ばれることがある。
【００１３】
LPRは、損失RTPメディアパケットを回復するスケーラブルな方法である。ここに記載する例はビデオコーデックを対象としているが、メディアに特有ではない。ここに記載するLPR技術はまた、他の種類のRTPメディアストリームを保護するために使用され得る。LPRは、損失メディアパケットの前方消失訂正を提供するために、リードソロモン符号化を使用し、損失メディアパケットの回復を可能にする。
【００１４】
LPRが使用される場合、RTPペイロードは、ほぼ等しいサイズのデータパケットを生成するように再パケット化される。再パケット化されたストリームは、異なるRTPダイナミックペイロード形式（これは割り当てられたメディアペイロード番号と異なる）を使用し、完全にRTPに準拠する。ペイロード特有のヘッダは、元のRTPパケットを再構成するのに十分な情報を含む。LPRは、SSRC及びCSRC情報が回復セット（以下に説明する）の途中で変化しないことを仮定する。タイムスタンプ、シーケンス番号、元のペイロード形式及びマーカ・ビット（marker bit）が完全に回復され、回復されたRTPストリームが復号化可能になることを確保する。
【００１５】
次に、回復パケットは、このRTPストリームに追加される。回復パケットもRTPに準拠する。各回復パケットはまた、保護されたデータパケットを記述するペイロードヘッダを含む。各パケットのRTPペイロードは回復情報を含む。追加された回復パケットの数は、RTCP受信レポートから推定されたチャネル損失率に依存する。約50%のビットレートのオーバ

ーヘッドを有するが、約15%までのチャネル損失率に対応可能である。

【００１６】

LPR保護は、全体メディアのビットレートの一部と考えられない点に留意すべきである。従って、実際に圧縮されたペイロードのみが数えられる。しかし、ネットワーク輻輳を回避するために、総計ビットレートが必要に応じて低減され得る。ネットワーク輻輳は、パケット損失以外に又はパケット損失に加えて望ましくない影響を有し得る点に留意すべきである。例えば、ネットワーク輻輳は、媒体の待ち時間の増加と、接続（シグナリング接続を含む）の劣化を生じ得る。従って、LPRプロトコルは、輻輳回避のための内蔵メッセージを有し得る。１つの適切な輻輳回避アルゴリズムについて以下に詳細に説明するが、様々な輻輳回避アルゴリズムが使用されてもよい。

【００１７】

メディアストリームに適用されるLPRの例について、図１～３を参照して説明する。ビットレート、パケット損失率、パケットサイズ、データパケット対回復パケットの比等を含むこの例の詳細は、全てが単なる例であり、実装の詳細、チャネル状態等に応じて変化し得る。

【００１８】

2%のパケット損失率を有する300kbpsのビデオチャネルについて検討する。３つのパケット101、102及び103が送信される。(6+2)のLPRモードが使用される。これは、２つの回復パケット104、105が６つのデータパケット106～111の各グループに挿入され、それぞれ８パケットの回復セットを生成することを意味する。回復パケット104、105は、最大データパケットとほぼ同じサイズであるが、何らかの更なるオーバーヘッドを含む。以下に詳細に説明するように、回復パケットは、６つのデータパケット106～111のいずれか２つの再生成を可能にする。

【００１９】

300kbpsのチャネル制限内に留まるために、メディアレートは、約220kbpsまで低減可能であり、回復パケットに80kbpsを残す。それぞれの再パケット化されたデータパケットの目標サイズは500バイトになり得る。従って、各回復セットは、約32000ビットの情報を伝達し、これは、300kbpsのチャネルレートで107msの期間を表す。107msの期間は、保護期間と呼ばれる。

【００２０】

各パケット（すなわち、再パケット化されたデータパケット106～111及び回復パケット104、105）は、通常の40バイトのIP+UDP+RTPヘッダのオーバーヘッドを含む。簡単にするために、このオーバーヘッドは示されていない。しかし、LPR自体に関連するオーバーヘッドが示されている。例えば、“3+420”は、420バイトの元のデータに加えて3バイトのオーバーヘッドがパケットに存在することを示す。回復パケットは100%のオーバーヘッドになる。オーバーヘッドの詳細は、以下に詳細に説明する。

【００２１】

図２は、回復処理を示している。データパケット3及び5（108、111）が損失したことを仮定する。LPRデコーダは、受信した６つのパケット（すなわち、データパケット1、2、4及び6並びに回復パケット1及び2）から再生成パケット201及び202を生成する。損失パケットの再生成は、６つのパケットが受信されるまで生じることができない点に留意すべきである。再生成の後に、元のRTPパケット203～205がデータパケットから復元される。損失パケットの再生成は、以下に詳細に説明する。

【００２２】

図３は、２つより多くのパケットが損失したときのパケット回復の失敗を示している。この例では、データパケット3及び5と同様に、回復パケット1が損失している。グループのうち５つのみのパケットが受信されたため、パケット再生成は不可能である。LPRデコーダは部分的なパケットを再生成しない。元のパケットが完全に回復できない場合、部分的なパケット情報は破棄される。従って、データパケット1及び2（301、302）の双方が損失したものとして示されている。

【００２３】
図３に示すように、元のデータパケットの全てが回復できない場合、受信側デコーダが全体画像の高速更新を要求することが一般的である。代替として、2006年3月28日に発行された“Dynamic Intra-coded Macroblock Refresh Interval for Video Error Concealment”という題の米国特許7,020,203号に記載のようなウォークアラウンド・リフレッシュ（walk-around refresh）を要求する。この米国特許の全ての内容を援用する。いずれの場合でも、画像を訂正するために必要な時間は、損失したパケットの数の関数ではなく、回復セットの所定数のパケット（この例では２つ）より多くを損失することによりもたらされた失敗の間の平均時間になる。
【００２４】
300kbps及び2%のパケット損失のチャネルで、６つのデータパケットと２つの回復データパケットとの各保護セットが107msのチャネルデータをカバーする所定の例では、失敗の間の平均時間は、約258秒であると計算され得る。高速更新が約2秒を要求することを仮定すると、所与のLPRの例により保護されるチャネルで送信されるビデオは、99%より多くの時間で誤りのないことになる。この保護は、80kbpsの更なるオーバーヘッドをもたらし、これは約30%のチャネルレートになる。
【００２５】
前述のLPRの例の更なる態様は、図４を参照してより良く理解され得る。図４は、LPRエンコーダの実装を示している。この実装の変形も可能である。ビデオエンコーダ401は、ほとんど通常の方法で動作する様々なビデオエンコーダのいずれかでもよい。例えば、ビデオエンコーダは、H.261、H.263、H.264、MPEG-2、又はMPEG-4ビデオ圧縮標準に従って動作し得る。代替として、ビデオエンコーダは、様々な独自仕様のビデオコーディング技術のいずれかに従って動作し得る。
【００２６】
ビデオエンコーダ401の出力は、任意選択の暗号化モジュール402に供給され得る。暗号化モジュール402もまた、様々な既知の種類のいずれかでもよい。暗号化されたビデオデータ（又は暗号化が使用されない場合には暗号化されていないビデオデータ）は、RTP送信機403に供給され得る。RTP送信機403も通常のものでもよい。RTP送信機403により生成されたRTPパケットは、以下に詳細に説明するように、LPRパケット化器406によりLPRアルゴリズムを使用して再パケット化され得る。LPR回復パケット生成器407は、以下に詳細に説明するように、LPRパケット化器406及びビデオエンコーダ401から受信した情報に基づいて回復パケットを生成し得る。LPR回復パケット生成器は、LPRデータパケットとLPR回復パケットとを受信機（図示せず）に送信し得る。
【００２７】
RTCPモジュール404は、パケット損失及び他のチャネル統計を受信し、これらをLPRモード判定モジュール405に供給し得る。LPRモード判定モジュール405はLPRパケット化器406を制御する。LPRモード判定モジュール405は、LPR保護を行うか行わないかを判定し、使用されるLPR保護パラメータを決定し得る。LPR保護が行われない場合、LPRパケット化器406及びLPR回復パケット生成器407は、RTP送信機403により生成されたRTPパケットを、変更せずに通過し得る。
【００２８】
LPRモード判定モジュール405は以下のように動作し得る。まず、（前述の）保護期間が決定され得る。長い保護期間は効率的なパケット再生成（例えば少ない回復kbps）を提供するが、長い待ち時間を生成する。最も典型的なテレビ会議のビットレートでは、100msが適切な保護期間になり得る。128kbpsより小さいビットレートでは、長い保護期間が必要になってもよい。例えば、150msが64kbpsのレートに適する可能性がある。1Mbpsより大きいビットレートでは、短い保護期間が有利なことがある。例えば、50msが2Mbps以上のレートに適する可能性がある。他のパラメータも保護期間を決定する際に検討され得る。
【００２９】
次に、保護期間毎の全パケット数が決定され得る。一般的に、保護期間毎に小さいパケ

ット数は、保護の効率を減少させる傾向になり得る。逆に、保護期間毎に大きいパケット数は、保護パケットをエンコード及びデコードするために必要な計算量を増加させる傾向になり得る。従って、ほとんどのチャネルについて、保護期間毎に約13以上のパケットが適切であると考えられる。保護期間毎のパケット数についての判定は、所望のデータレートに対する結果のパケットサイズも考慮すべきである。出力パケットサイズ（IPオーバーヘッドを含む）は、分割せずにIPSECトンネルを横断するために、1260バイト未満であるべきである。

【００３０】

保護期間毎のパケット数が定められると、保護期間毎の回復パケット数が選択され得る。回復パケット数は、（通常の確率的分析技術を使用して）保護の中断の間の平均時間が所定の規定値を満たす又は超えるように選択され得る。更に、全ての送信されたビデオチャネルの総計（LPR保護パケット及びRTPメッセージを含む）が、指定の輻輳上限を超えないように選択され得る。

【００３１】

以下のテーブルは、様々な損失条件で様々なビットレートに使用され得る例示的な保護モデルを示している。テーブル１は、パケット損失が3%を超えるときにデータレートを低減する輻輳回避アルゴリズム（以下に説明する）と共に使用可能であり、4%のパケット損失条件に適し得るLPRパラメータを示している。テーブル２は、輻輳回避アルゴリズムが使用中でないときに使用され得る2%のパケット損失条件のLPRパラメータを示している。

［表１］

ビデオレート (kbs)	グループ毎のデータパケット	パケットペイロードサイズ（バイト）	グループ毎の回復パケット	保護期間 (ms)	MTBF (sec)
64	13	87	4	141	334
128	12	133	4	100	323
256	12	266	4	100	323
384	12	400	4	100	324
512	12	533	4	100	324
768	12	800	4	100	324
1024	16	800	5	100	757
2048	16	800	5	50	379
3072	24	800	6	50	338
4096	32	800	7	50	378
5120	32	1000	7	50	378
6144	39	1000	8	51	567

テーブル1　4%のパケット損失での常時保護

［表２］

ビデオレート (kbs)	グループ毎のデータパケット	パケットペイロードサイズ（バイト）	グループ毎の回復パケット	保護期間 (ms)	MTBF (sec)
64	13	87	3	141	589
128	12	133	3	100	545
256	12	266	3	100	545
384	12	400	3	100	546
512	12	533	3	100	546
768	12	800	3	100	546
1024	16	800	4	100	2591
2048	16	800	4	50	1296

3072	24	800	5	50	2444
4096	32	800	5	50	573
5120	32	1000	5	50	573
6144	39	1000	6	51	1704

テーブル2 2%のパケット損失での常時保護

LPRパケット化器406は以下のように動作し得る。LPRパケット化器406は、パケット再生成符号化の効率を改善するために、RTP送信機403により生成された元のメディアパケットを細分する。LPRデータパケットは、唯一の発信元パケットからの情報を含む。メディアチャネルが暗号化されている場合、暗号化ストリームでLPR再パケット化が実行され得る。一般的に、LPRストリーム自体は再暗号化されない。すなわち、暗号化パケットと新しく生成されたLPR回復パケットとに追加されたLPRカプセル化フィールド（ヘッダ）は、平文で送信される。

【００３２】

元の発信元パケットが細分される場合（例えば、図１～３のパケット1　101）、この発信元パケットの最初のLPRデータパケットは初期データパケット（例えば、図１～３のデータ1　106）と呼ばれる。（存在する場合には）次のデータパケットは、後続データパケット（例えば、図１～３のデータ2　107及びデータ3　108）と呼ばれる。後続データパケットの数は、初期データパケットで伝達され、従って、初期データパケットが送信されると変更できない。後続データパケット及び初期データパケットは、異なるLPR保護グループに存在してもよい。

【００３３】

シーケンス番号とペイロード形式とPフィールドとを除いて、元のRTPヘッダの全てのフィールドは、各LPRデータパケットで変更されずに伝達され得る。これは、タイムスタンプとマーカ・ビットとSSRC/CSRC情報とを含む。ペイロード番号は、ネゴシエーションされたLPRペイロード形式と交換され得る。シーケンス番号は、通常通りにLPRメディアストリームに連続的に割り当てられ得る。LPRパケットのPフィールドは0に設定され得る。LPRパケット化器406は、元のRTPシーケンス番号をLPR回復パケット生成器407に渡し得る。

【００３４】

各初期データパケットは、LPRに特有の7バイトのペイロードヘッダを含み得る。この例が図５に示されている。各後続データパケットは、3バイトのペイロードヘッダを含み得る。この例が図６に示されている。これらのペイロードヘッダは、RTPヘッダの直後に続き、ビデオ（又は他のメディア）ペイロードに先行し得る。ヘッダサイズ及び個々のフィールドのサイズ並びにこれらの構成は単なる例であり、他の構成も可能であることがわかる。

【００３５】

回復パケットのフィールド501、601は、回復セットで後に送信される回復パケット数を示すために使用され得る。回復セットの全てのデータパケットは、同じ数の回復パケットを伝達するべきである。伝達される回復パケット数は0でもよく、これは回復パケットが送信されないことを示す。このフィールドは6ビットのフィールドでもよく、64の回復パケットまでを許容する。形式のフィールド502、602は、それぞれ初期データパケット又は後続データパケットを示すために、00又は01に設定される2ビットのフィールドセットでもよい。これは、最初のペイロードバイトを検査することにより、LPRのヘッダ形式が決定されることを可能にする。

【００３６】

データインデックスのフィールド503、603は8ビットのフィールドでもよい。回復セットの最初のデータパケットは1のデータインデックスを割り当てられ得る。データインデ

ックスは、そのセットの次のデータパケットでインクリメントされ得る。後続パケットのフィールド504は、初期パケットに関連する後続パケット数を指定するために使用される8ビットのフィールドでもよい。データパケットのフィールド604は、回復セットに含まれるデータパケット数を伝達するために使用される8ビットのフィールドでもよい。全てのデータパケットは、同じ数のデータパケットを伝達するべきである。

【００３７】

元のシーケンスのフィールド505は、元のRTPパケットシーケンス番号を示す16ビットのフィールドでもよい。これは、元のRTPパケットを再生成するために使用される。元のPビットのフィールド506は、元のRTPパケットにより伝達されるパディングビット（padding bit）を伝達するために使用される1ビットのフィールドでもよい。元のペイロード形式のフィールド507は、元のRTPパケットにより伝達されるペイロード形式を伝達するために使用される7ビットのフィールドでもよい。

【００３８】

LPR回復パケット生成器407は、以下のように動作し得る。LPR回復パケット生成器407は、元のメディアのRTPシーケンス番号を、LPRデータパケットの最後のLPRシーケンス番号と交換し得る。LPR回復パケット生成器407はまた、各回復セットの終わりにLPR回復パケットを挿入し得る。

【００３９】

各回復パケットはまた、図７に示すような9バイトのヘッダを含み得る。回復パケットヘッダは、RTPヘッダの直後に続き、ビデオ（又は他のメディア）ペイロードに先行し得る。前述のヘッダと同様に、フィールドの形式及びサイズは例示的であり、他の構成も使用されてもよい。

【００４０】

回復インデックスのフィールド701は、回復セットの回復パケットのシーケンスを示すために使用される6ビットのフィールドでもよい。最初の回復パケットは、1の回復インデックスを割り当てられ得る。回復インデックスは、回復セットの次の回復パケットでインクリメントされ得る。形式のフィールド702は、パケットが回復パケットであることを示すために、10に設定される2ビットのフィールドでもよい。回復パケットのフィールドは、回復セットのデータパケット数を含む8ビットのフィールドでもよい。全ての回復パケットは、対応するデータパケットにより伝達された同じ数の回復パケットを伝達するべきである。データパケットのフィールド704は、回復セットで送信されるデータパケット数を示す8ビットのフィールドでもよい。回復パケットは、対応するデータパケットで示された同じ数のデータパケットを示すべきである。

【００４１】

保護タイムスタンプのフィールド705は、回復セットの各データパケットのリードソロモン符号化されたRTPタイムスタンプを保持する32ビットのフィールドでもよい。これは、回復ペイロード自体で使用されるものと同じリードソロモン符号化を使用して結合され得る。このフィールドにより、損失データパケットのRTPタイムスタンプが再生成されることが可能になる。4バイトはネットワークの指示（in network order）である。

【００４２】

保護マーカ、フラグ及びサイズのフィールド706（図８に更に示す）は、リードソロモン符号化されたRTPマーカ・ビット802、フラグ値801及びデータパケットサイズ803を保有し得る。フラグ値は、LPRデータパケットが初期パケットである場合に1になってもよく、LPRデータパケットが後続パケットである場合に0に設定されてもよい。サイズは、LPRヘッダと同様にIP、UDP及びRTPヘッダを含み得る。データパケットサイズはネットワークの指示でもよい。

【００４３】

保護フィールドは、回復ペイロード自体で使用されたものと同じリードソロモン符号化を使用して結合される。このフィールドにより、回復可能な何らかの損失データパケットについて、LPRヘッダ形式、マーカ及びデータパケットサイズが再生成されることが可能

になる。

【００４４】

回復パケットのペイロードは、リードソロモン256（RS256）符号化を使用して符号化され得る。初期データパケットの元のシーケンス505、元のPビット506、元のペイロード形式507及び後続パケット504のフィールドは、ペイロードバイトであるかのように保護され得る。これにより、初期データパケットが損失した場合に、この情報が回復可能になる。更に、回復パケットのペイロードヘッダは、３つのファントムデータ（Phantom Data）のフィールド（データパケット自体の長さ、マーカ・ビット及びタイムスタンプ）の符号化を含み得る。この場合も同様に、これにより、細分されたRTPパケットの全てのデータパケットが損失した場合に、この情報が回復可能になる。

【００４５】

各データパケットの各ペイロードバイトは、以下のRS256の生成関数に従って、各回復パケットの対応するバイトに寄与し得る。

【００４６】

【数１】

$$R_j = \sum_{i=1}^{d} B_i^{j-1} \otimes D_i$$

ただし、D_iは第iのデータパケット（1≦i≦d）であり、R_jは第jの回復パケット（1≦j≦r）であり、B_iはテーブル３に示す値を有する第iの基礎係数（1≦i≦d）であり、

【００４７】

【数２】

$$B_i^{j-1}, \otimes, \sum$$

は以下に示すGガロア域（2^8）算術演算を使用して計算される。所与の基礎係数のテーブルは、回復セットの最大データパケット数を128に制限するが、他のデータパケット数が選択可能であり、それに従って適切なテーブルが計算され得る点に留意すべきである。

［表３］

```
/*
** RS256_BaseTable[i] holds the base coefficient for data packet i-1
**
** These entries are selected to ensure that the dispersal matrix is invertible.
** Each entry b is chosen so b**x is not 1 unless x is 0 or 255.
*/
#define RS256_BASETABLE_LEN 128

STATIC CONST UINT8 RS256_BaseTable[RS256_BASETABLE_LEN]= {
 0x02, 0x04, 0x06, 0x09, 0x0D, 0x0E, 0x10, 0x12,
 0x13, 0x14, 0x16, 0x18, 0x19, 0x1B, 0x1D, 0x1E,
 0x1F, 0x22, 0x23, 0x28, 0x2A, 0x2B, 0x30, 0x31,
 0x32, 0x34, 0x39, 0x3C, 0x3F, 0x41, 0x42, 0x43,
 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x51, 0x52,
 0x53, 0x54, 0x58, 0x5A, 0x5B, 0x5C, 0x5D, 0x5F,
 0x62, 0x63, 0x68, 0x69, 0x6D, 0x6F, 0x70, 0x71,
 0x76, 0x77, 0x79, 0x7A, 0x7B, 0x7E, 0x80, 0x81,
 0x83, 0x85, 0x87, 0x88, 0x89, 0x8C, 0x8D, 0x8E,
 0x90, 0x94, 0x95, 0x97, 0x9A, 0x9B, 0x9D, 0x9E,
 0x9F, 0xA2, 0xA3, 0xA4, 0xA5, 0xAA, 0xAB, 0xAF,
 0xB0, 0xB1, 0xB2, 0xB7, 0xBB, 0xBC, 0xBD, 0xC0,
 0xC2, 0xC6, 0xC7, 0xC8, 0xC9, 0xCA, 0xCB, 0xCC,
 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD8, 0xDA, 0xDE,
 0xE0, 0xE1, 0xE3, 0xE5, 0xE8, 0xEA, 0xEC, 0xEE,
 0xF0, 0xF3, 0xF6, 0xF7, 0xF8, 0xF9, 0xFA, 0xFE
};
```

テーブル3 RS256基礎係数

ガロア域（2^8）算術演算は、２つの補助テーブル（ガロア・ログ関数（glog）テーブル及びガロア指数関数（gexp）テーブル）を使用して設定され得る。これらのテーブルは以下の通りである。

［表４］

```
unsigned short glog[256] = {
 0x200, 0x00, 0x01, 0x19, 0x02, 0x32, 0x1A, 0xC6,    /* 0 to 7 */
 0x03, 0xDF, 0x33, 0xEE, 0x1B, 0x68, 0xC7, 0x4B,     /* 8 to 15 */
 0x04, 0x64, 0xE0, 0x0E, 0x34, 0x8D, 0xEF, 0x81,     /* 16 to 23 */
 0x1C, 0xC1, 0x69, 0xF8, 0xC8, 0x08, 0x4C, 0x71,     /* 24 to 31 */
 0x05, 0x8A, 0x65, 0x2F, 0xE1, 0x24, 0x0F, 0x21,     /* 32 to 39 */
 0x35, 0x93, 0x8E, 0xDA, 0xF0, 0x12, 0x82, 0x45,     /* 40 to 47 */
 0x1D, 0xB5, 0xC2, 0x7D, 0x6A, 0x27, 0xF9, 0xB9,     /* 48 to 55 */
 0xC9, 0x9A, 0x09, 0x78, 0x4D, 0xE4, 0x72, 0xA6,     /* 56 to 63 */
 0x06, 0xBF, 0x8B, 0x62, 0x66, 0xDD, 0x30, 0xFD,     /* 64 to 71 */
 0xE2, 0x98, 0x25, 0xB3, 0x10, 0x91, 0x22, 0x88,     /* 72 to 79 */
 0x36, 0xD0, 0x94, 0xCE, 0x8F, 0x96, 0xDB, 0xBD,     /* 80 to 87 */
 0xF1, 0xD2, 0x13, 0x5C, 0x83, 0x38, 0x46, 0x40,     /* 88 to 95 */
 0x1E, 0x42, 0xB6, 0xA3, 0xC3, 0x48, 0x7E, 0x6E,     /* 96 to 103 */
 0x6B, 0x3A, 0x28, 0x54, 0xFA, 0x85, 0xBA, 0x3D,     /* 104 to 111 */
 0xCA, 0x5E, 0x9B, 0x9F, 0x0A, 0x15, 0x79, 0x2B,     /* 112 to 119 */
 0x4E, 0xD4, 0xE5, 0xAC, 0x73, 0xF3, 0xA7, 0x57,     /* 120 to 127 */
 0x07, 0x70, 0xC0, 0xF7, 0x8C, 0x80, 0x63, 0x0D,     /* 128 to 135 */
 0x67, 0x4A, 0xDE, 0xED, 0x31, 0xC5, 0xFE, 0x18,     /* 136 to 143 */
 0xE3, 0xA5, 0x99, 0x77, 0x26, 0xB8, 0xB4, 0x7C,     /* 144 to 151 */
 0x11, 0x44, 0x92, 0xD9, 0x23, 0x20, 0x89, 0x2E,     /* 152 to 159 */
 0x37, 0x3F, 0xD1, 0x5B, 0x95, 0xBC, 0xCF, 0xCD,     /* 160 to 167 */
 0x90, 0x87, 0x97, 0xB2, 0xDC, 0xFC, 0xBE, 0x61,     /* 168 to 175 */
 0xF2, 0x56, 0xD3, 0xAB, 0x14, 0x2A, 0x5D, 0x9E,     /* 176 to 183 */
 0x84, 0x3C, 0x39, 0x53, 0x47, 0x6D, 0x41, 0xA2,     /* 184 to 191 */
 0x1F, 0x2D, 0x43, 0xD8, 0xB7, 0x7B, 0xA4, 0x76,     /* 192 to 199 */
 0xC4, 0x17, 0x49, 0xEC, 0x7F, 0x0C, 0x6F, 0xF6,     /* 200 to 207 */
 0x6C, 0xA1, 0x3B, 0x52, 0x29, 0x9D, 0x55, 0xAA,     /* 208 to 215 */
 0xFB, 0x60, 0x86, 0xB1, 0xBB, 0xCC, 0x3E, 0x5A,     /* 216 to 223 */
 0xCB, 0x59, 0x5F, 0xB0, 0x9C, 0xA9, 0xA0, 0x51,     /* 224 to 231 */
 0x0B, 0xF5, 0x16, 0xEB, 0x7A, 0x75, 0x2C, 0xD7,     /* 232 to 239 */
 0x4F, 0xAE, 0xD5, 0xE9, 0xE6, 0xE7, 0xAD, 0xE8,     /* 240 to 247 */
 0x74, 0xD6, 0xF4, 0xEA, 0xA8, 0x50, 0x58, 0xAF      /* 248 to 255 */
};
```

テーブル4 ガロア･ログ補助関数

［表５］

```
unsigned char gexp[1065] = {
 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,     /* 0 to 7 */
 0x1D, 0x3A, 0x74, 0xE8, 0xCD, 0x87, 0x13, 0x26,     /* 8 to 15 */
 0x4C, 0x98, 0x2D, 0x5A, 0xB4, 0x75, 0xEA, 0xC9,     /* 16 to 23 */
 0x8F, 0x03, 0x06, 0x0C, 0x18, 0x30, 0x60, 0xC0,     /* 24 to 31 */
 0x9D, 0x27, 0x4E, 0x9C, 0x25, 0x4A, 0x94, 0x35,     /* 32 to 39 */
 0x6A, 0xD4, 0xB5, 0x77, 0xEE, 0xC1, 0x9F, 0x23,     /* 40 to 47 */
 0x46, 0x8C, 0x05, 0x0A, 0x14, 0x28, 0x50, 0xA0,     /* 48 to 55 */
 0x5D, 0xBA, 0x69, 0xD2, 0xB9, 0x6F, 0xDE, 0xA1,     /* 56 to 63 */
 0x5F, 0xBE, 0x61, 0xC2, 0x99, 0x2F, 0x5E, 0xBC,     /* 64 to 71 */
 0x65, 0xCA, 0x89, 0x0F, 0x1E, 0x3C, 0x78, 0xF0,     /* 72 to 79 */
 0xFD, 0xE7, 0xD3, 0xBB, 0x6B, 0xD6, 0xB1, 0x7F,     /* 80 to 87 */
 0xFE, 0xE1, 0xDF, 0xA3, 0x5B, 0xB6, 0x71, 0xE2,     /* 88 to 95 */
 0xD9, 0xAF, 0x43, 0x86, 0x11, 0x22, 0x44, 0x88,     /* 96 to 103 */
 0x0D, 0x1A, 0x34, 0x68, 0xD0, 0xBD, 0x67, 0xCE,     /* 104 to 111 */
 0x81, 0x1F, 0x3E, 0x7C, 0xF8, 0xED, 0xC7, 0x93,     /* 112 to 119 */
 0x3B, 0x76, 0xEC, 0xC5, 0x97, 0x33, 0x66, 0xCC,     /* 120 to 127 */
 0x85, 0x17, 0x2E, 0x5C, 0xB8, 0x6D, 0xDA, 0xA9,     /* 128 to 135 */
 0x4F, 0x9E, 0x21, 0x42, 0x84, 0x15, 0x2A, 0x54,     /* 136 to 143 */
 0xA8, 0x4D, 0x9A, 0x29, 0x52, 0xA4, 0x55, 0xAA,     /* 144 to 151 */
 0x49, 0x92, 0x39, 0x72, 0xE4, 0xD5, 0xB7, 0x73,     /* 152 to 159 */
 0xE6, 0xD1, 0xBF, 0x63, 0xC6, 0x91, 0x3F, 0x7E,     /* 160 to 167 */
 0xFC, 0xE5, 0xD7, 0xB3, 0x7B, 0xF6, 0xF1, 0xFF,     /* 168 to 175 */
 0xE3, 0xDB, 0xAB, 0x4B, 0x96, 0x31, 0x62, 0xC4,     /* 176 to 183 */
 0x95, 0x37, 0x6E, 0xDC, 0xA5, 0x57, 0xAE, 0x41,     /* 184 to 191 */
 0x82, 0x19, 0x32, 0x64, 0xC8, 0x8D, 0x07, 0x0E,     /* 192 to 199 */
 0x1C, 0x38, 0x70, 0xE0, 0xDD, 0xA7, 0x53, 0xA6,     /* 200 to 207 */
 0x51, 0xA2, 0x59, 0xB2, 0x79, 0xF2, 0xF9, 0xEF,     /* 208 to 215 */
```

```
0xC3, 0x9B, 0x2B, 0x56, 0xAC, 0x45, 0x8A, 0x09,     /* 216 to 223 */
0x12, 0x24, 0x48, 0x90, 0x3D, 0x7A, 0xF4, 0xF5,     /* 224 to 231 */
0xF7, 0xF3, 0xFB, 0xEB, 0xCB, 0x8B, 0x0B, 0x16,     /* 232 to 239 */
0x2C, 0x58, 0xB0, 0x7D, 0xFA, 0xE9, 0xCF, 0x83,     /* 240 to 247 */
0x1B, 0x36, 0x6C, 0xD8, 0xAD, 0x47, 0x8E, 0x01,     /* 248 to 255 */
0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1D,     /* 256 to 263 */
0x3A, 0x74, 0xE8, 0xCD, 0x87, 0x13, 0x26, 0x4C,     /* 264 to 271 */
0x98, 0x2D, 0x5A, 0xB4, 0x75, 0xEA, 0xC9, 0x8F,     /* 272 to 279 */
0x03, 0x06, 0x0C, 0x18, 0x30, 0x60, 0xC0, 0x9D,     /* 280 to 287 */
0x27, 0x4E, 0x9C, 0x25, 0x4A, 0x94, 0x35, 0x6A,     /* 288 to 295 */
0xD4, 0xB5, 0x77, 0xEE, 0xC1, 0x9F, 0x23, 0x46,     /* 296 to 303 */
0x8C, 0x05, 0x0A, 0x14, 0x28, 0x50, 0xA0, 0x5D,     /* 304 to 311 */
0xBA, 0x69, 0xD2, 0xB9, 0x6F, 0xDE, 0xA1, 0x5F,     /* 312 to 319 */
0xBE, 0x61, 0xC2, 0x99, 0x2F, 0x5E, 0xBC, 0x65,     /* 320 to 327 */
0xCA, 0x89, 0x0F, 0x1E, 0x3C, 0x78, 0xF0, 0xFD,     /* 328 to 335 */
0xE7, 0xD3, 0xBB, 0x6B, 0xD6, 0xB1, 0x7F, 0xFE,     /* 336 to 343 */
0xE1, 0xDF, 0xA3, 0x5B, 0xB6, 0x71, 0xE2, 0xD9,     /* 344 to 351 */
0xAF, 0x43, 0x86, 0x11, 0x22, 0x44, 0x88, 0x0D,     /* 352 to 359 */
0x1A, 0x34, 0x68, 0xD0, 0xBD, 0x67, 0xCE, 0x81,     /* 360 to 367 */
0x1F, 0x3E, 0x7C, 0xF8, 0xED, 0xC7, 0x93, 0x3B,     /* 368 to 375 */
0x76, 0xEC, 0xC5, 0x97, 0x33, 0x66, 0xCC, 0x85,     /* 376 to 383 */
0x17, 0x2E, 0x5C, 0xB8, 0x6D, 0xDA, 0xA9, 0x4F,     /* 384 to 391 */
0x9E, 0x21, 0x42, 0x84, 0x15, 0x2A, 0x54, 0xA8,     /* 392 to 399 */
0x4D, 0x9A, 0x29, 0x52, 0xA4, 0x55, 0xAA, 0x49,     /* 400 to 407 */
0x92, 0x39, 0x72, 0xE4, 0xD5, 0xB7, 0x73, 0xE6,     /* 408 to 415 */
0xD1, 0xBF, 0x63, 0xC6, 0x91, 0x3F, 0x7E, 0xFC,     /* 416 to 423 */
0xE5, 0xD7, 0xB3, 0x7B, 0xF6, 0xF1, 0xFF, 0xE3,     /* 424 to 431 */
0xDB, 0xAB, 0x4B, 0x96, 0x31, 0x62, 0xC4, 0x95,     /* 432 to 439 */
0x37, 0x6E, 0xDC, 0xA5, 0x57, 0xAE, 0x41, 0x82,     /* 440 to 447 */
0x19, 0x32, 0x64, 0xC8, 0x8D, 0x07, 0x0E, 0x1C,     /* 448 to 455 */
0x38, 0x70, 0xE0, 0xDD, 0xA7, 0x53, 0xA6, 0x51,     /* 456 to 463 */
0xA2, 0x59, 0xB2, 0x79, 0xF2, 0xF9, 0xEF, 0xC3,     /* 464 to 471 */
0x9B, 0x2B, 0x56, 0xAC, 0x45, 0x8A, 0x09, 0x12,     /* 472 to 479 */
0x24, 0x48, 0x90, 0x3D, 0x7A, 0xF4, 0xF5, 0xF7,     /* 480 to 487 */
0xF3, 0xFB, 0xEB, 0xCB, 0x8B, 0x0B, 0x16, 0x2C,     /* 488 to 495 */
0x58, 0xB0, 0x7D, 0xFA, 0xE9, 0xCF, 0x83, 0x1B,     /* 496 to 503 */
0x36, 0x6C, 0xD8, 0xAD, 0x47, 0x8E, 0x00, 0x00,     /* 504 to 511 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
```

```
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00      /* zeros for fast
multiply */
 };
```

テーブル5 ガロア指数補助関数

これらのテーブルは、以下に説明する様々な算術演算を計算するために使用され得る。

【００４８】

ガロア域（2^8）の加算及び減算は同じ演算であり、以下のように実行される。

【００４９】

【数３】

$$a \oplus b = a \wedge b$$
$$a - b = a \wedge b$$

ただし、^は排他的論理和又はXOR演算である。通常の演算と同様に、次のようになる。

【００５０】

【数４】

$$a \oplus 0 = 0 \oplus a = a$$

ガロア域（2^8）の乗算は以下のように実行される。

【００５１】

【数５】

$$a \otimes b = \mathrm{gexp}[\mathrm{glog}[a] + \mathrm{glog}[b]]$$

ただし、+は通常の加算演算である。通常の演算と同様に、次のようになる。

【００５２】

【数６】

$$a \otimes 0 = 0 \otimes a = 0$$

また、次のようになる。

【００５３】

【数７】

$$a \otimes 1 = 1 \otimes a = a$$

ガロア域（2^8）の除算（a÷b）は以下のように実行される。

a÷b=gexp[glog[a]-glog[b]+255]

ただし、+は通常の加算演算であり、-は通常の減算演算である。通常の演算と同様に、a÷1=aである。また、通常通り、bは0ではない。

【００５４】

ガロア域（2^8）のべき乗関数（a^b）は以下のように実行される。

a^b=gexp[(glog[a]*b)%0xFF]

ただし、+は通常の加算演算であり、-は通常の減算演算であり、*は通常の乗算演算であり、%は通常のmod関数である。通常の演算と同様に、0でないaについてa^0=1である。RS256参照コードでは、0は決してべき乗にならない。

【００５５】

前述のように、（パケットの）LPRグループサイズは、所望の保護期間（ミリ秒）を実現するように設定され得る。ビデオコーデックが全チャネルビットレートを使用しない場合、LPRグループは長い待ち時間を生じる。例えば、通常では1.5Mbpsで動作するビデオチャネルは、軽い動作期間に（in periods of light motion）750kbpsに低下し得る。これは、所望の２倍の長さの保護期間を生じる。このような状況は、選択されたLPRモードが所定のMTBFを満たし続けることを生じ得る。この例では、2の係数だけ所定のMTBFを超える。

【００５６】

或る場合には、この拡張保護期間が許容され得る。しかし、パケットが損失した場合に、これは長い待ち時間を生じ得る。従って、或る場合には、実際のビデオレートが変化するときに、送信機がそのLPRモードを調整することが望ましいことがある。３つのこのような調整について以下に説明するが、他の調整も可能である。

【００５７】

行うことができる１つの調整は、保護グループサイズに対するものである。或る実施例では、この設定は、各保護グループの境界で変更され得る。従って、送信機は、LPR出力パケットレートを監視し、グループ毎のパケット数を動的に適合させ得る。この調整が使用される場合、データレートが下がると、LPR保護オーバーヘッドは上がる傾向になる。しかし、このような場合のデータレートは一般的に通常の制限より小さいため、変更されたLPRモードは、一般的に依然として所定の輻輳上限の下に留まる。

【００５８】

行うことができる他の調整は、送信機が初期データパケットの送信時にその出力パケットサイズを変更することである。典型的には後続パケット数が初期データパケットで伝達されているため、一般的に、送信機は、後続パケットを送信するときに出力パケットサイズを変更するべきでない。従って、送信機は、LPR出力パケットレートと出力データレートとを監視し、保護期間毎のパケット数を維持するようにパケットサイズを動的に適合させ得る。

【００５９】

行うことができる他の調整は、送信機が空のデータパケットを送信して部分的な回復セットを完了することである。空のデータパケットは、前述の初期データパケットヘッダを含み得る。パケットに通常存在する他の情報（シーケンス番号、ペイロード形式、後続パケット数等）は、全て0に設定され得る。空のデータパケットは保護グループの一部であるため、前述の回復パケットの符号化はこれらのパケットを含み得る。これらのパケットはまた、デコード処理により回復可能である。また、空のデータパケットは、出力メディアパケットストリームの空のパケットを回避するために、LPRデコーダ（以下に説明する）により破棄され得る。

【００６０】

詰め込まれたパケット（fill packet）は、所定のデータレートを維持するために使用され得る。これらの詰め込まれたパケットは、受信機により破棄され得る。例示的な詰め込まれたパケットが図９に示されている。詰め込まれたパケットは、1の値を有する6ビットの拡張形式のフィールド901と、11の値を有する2ビットの形式のフィールド902とを含み得る。詰め込まれたパケットは、メディアストリームのLPRペイロード形式を使用したRTPパケットでもよい。タイムスタンプの値は、前に送信されたRTPパケットの値と同じでもよい。シーケンス番号は、前に送信されたパケットの値からインクリメントされてもよい。ペイロードは、前述の詰め込まれたヘッダで終了し得る。フィールドのペイロード長は、最大の指定パケットサイズ内で如何なる長さでもよい。

【００６１】

LPRデコーダの例が図１０に示されている。前述のLPRエンコーダと同様に、この実施例の変形も可能である。LPRデコーダの動作は、基本的に前述のLPRエンコーダの逆になり得る。具体的には、LPRパケットは、LPR回復モジュール1001により回復され得る。これらの回復されたLPRパケットは、RTP並び替えバッファ1002に渡され、RTP並び替えバッファは、通常のシステムと同様に、パケットを受信順に並び替え得る。LPR再生成器1003は、（LPR回復ブロック1001から）回復されたLPRパケットから受信した情報と、RTPCモジュール1006から受信したRTCP情報とに基づいて、何らかの損失パケットを再生成し得る。これらの再生成されたブロックは、（暗号化が発信元で使用されている場合）復号化ブロック1004により復号化され得る。最後に、復号化されたブロックがビデオデコーダ1005（又は必要に応じて他の種類のメディアエンコーダ）により処理され得る。

【００６２】

LPR回復モジュール1001及びLPR再生成器1003のみは、通常のシステムと比べて非標準の要素でもよく、これらの２つのモジュールはLPRが使用されないときにパススルー（pass-through）として動作し得る。

【００６３】

LPR回復モジュール1001は、全てのLPR RTPパケットを処理し得る。所与の例では、受信して再生成された全てのパケット（データ及び回復）は、直接にRTP並び替えバッファに渡され得る。各データパケットのオーバーヘッドデータにより、これらのパケットが何らかの順序でLPR回復モジュール1001により処理されることが可能になる。LPR回復モジュール1001はまた、LPR RTP受信レポート情報をRTCPモジュール1006に提供し得る。この情報は、何が受信されたかに基づいてもよい（例えば、受信されていない何らかのパケットが受信レポートで損失として特定され得る）。

【００６４】

回復可能な回復セットに損失データパケットが存在する場合、十分な回復パケットが受信されるとすぐに、LPR回復モジュールはこれらのパケットを回復し得る。前述のように、kのデータパケットが損失している場合、データを回復するためにkの回復パケットが受信されなければならない。この回復処理は、まず、マーカ・ビットと、データパケット長と、初期パケットフラグと、元のタイムスタンプとを回復することを含み得る。データパケット長は、元のデータパケットのペイロードを回復するために使用され得る。LPR RTPヘッダの“固定”の構成要素（例えば、SSRC、CSRC、LPRペイロード形式等）は、１つの回復パケットから採用され得る。LPR RTPヘッダの様々な部分（例えば、マーカ・ビット

、タイムスタンプ及びシーケンス番号）は、回復ヘッダから回復された値に設定され得る。

【００６５】

LPRデータパケットiのシーケンス番号は以下の式に従って計算され得る。

$S_i=S_r-(d+r-i)$

ただし、iは回復データパケットのインデックスであり、dは回復セットのデータパケット数であり、rは受信した回復パケットのインデックスであり、S_iはパケットiのシーケンス番号である。

【００６６】

データパケットを受信する間に、前述のRS256生成関数を使用して、部分的な回復パケットが回復モジュール1001により構築され得る。そのセットの対応する回復パケットが受信されると、これらのパケットは、部分的な回復パケットと排他的論理和（XOR）され得る。結果の残りのパケットは、各回復パケットへの何らかの損失データパケットの寄与を保持する。例示的なシステムでは、それぞれの残りのパケットは、損失データパケットのそれぞれの線形結合である。従って、kの連立方程式（kの残りのパケットのそれぞれに１つ）が存在し、それぞれkの未知数（損失したkのデータパケットのそれぞれに１つ）が存在する。

【００６７】

当業者にわかるように、これらの式はk×kの行列として表され得る。k×kの行列は、例えばガウス消去法により数学的に解決され得る。このような演算は周知であるため、詳細はここでは説明しない。残りのデータで結果の逆行列を乗算することにより、損失データパケットを回復することができる。

【００６８】

前述の回復処理を示すために、図２の回復シナリオが実行され得る。明瞭にするために、実際のガロア域（2^8）の演算ではなく、通常の算術演算が使用される。回復パケットのデータは、以下の式（前述のRS256生成関数及びテーブルから導かれる）を使用して生成され得る。

$R_1=D_1+D_2+D_3+D_4+D_5+D_6$

$R_2=2D_1+4D_2+6D_3+9D_4+13D_5+14D_6$

部分的な回復パケットは、受信したパケットでデコーダにより計算され得る。前述の例で記載したように、パケット3及び4（図３の108、110）は受信されない。その結果、以下のようになる。

$P_1=D_1+D_2+D_5+D_6$

$P_2=2D_1+4D_2+13D_5+14D_6$

残りは、受信した回復パケットから部分的な回復パケットを減算することにより計算され得る。

$r_1=R_1-P_1=D_3+D_4$

$r_2=R_2-P_2=6D_3+9D_4$

従って、残りの生成行列は以下のようになる。

【００６９】

【数８】

$$\begin{matrix}1 & 1\\ 6 & 9\end{matrix}$$

ガウス消去法を介してこの行列を逆にすると、以下の結果になる。

【００７０】

【数９】

$$\begin{matrix} 3 & -1/3 \\ -2 & 1/3 \end{matrix}$$

従って、損失データパケット（すなわち、データパケット3及び4）の情報は、以下の式を使用して回復され得る。

$D_3=3r_1-r_2/3$

$D_4=-2r_1+r_2/3$

LPRが使用中である場合、RTP並び替えバッファ1002はLPRデコーダと統合され得る。或る状況では、パケットが実際に到達する前に順序の狂ったパケットがLPRにより再生成され得るため、これは最小遅延の実装を可能にする。更に、或る実施例では、何らかのデータパケットが損失した場合にのみリードソロモンデコーダを動作して、回復セットの最初のdのパケットを取得することが望ましいことがある。この設計は最小の遅延を生じないことがあるが、計算効率が良くなり得る。

【００７１】

LPR再生成モジュール1003は、前述のように、データパケットから元のRTPストリームを再生成し得る。RTPパケットの何らかのデータパケットが損失している場合、そのRTPパケットは出力ストリームから省略される。LPRストリームは並び替えられているため、各パケットからの初期パケットが最初に受信されるべきである。後続パケットが最初に直面する場合、初期パケットが損失しており、後続パケットが破棄され得る。

【００７２】

出力RTPパケットは、初期データパケットのRTPヘッダ情報を使用する。ペイロード形式、シーケンス番号及びタイムスタンプは、初期データヘッダの値と交換される。ペイロードは、LPRヘッダが取り除かれて、初期データパケットと何らかの後続データパケットとから集められる。何らかの後続パケットが損失している場合、出力RTPパケットが抑制され得る。後続パケット数は初期データパケットのヘッダで伝達され得るため、既知でもよい点に留意すべきである。

【００７３】

前述のような損失パケット回復技術は、更に動作を拡張するために、輻輳回避アルゴリズムと結合され得る。例示的な輻輳回避アルゴリズムは前述の通りであり、動的帯域割り当て（DBA：Dynamic Bandwidth Allocation）と呼ばれる。DBAは、受信機のRTPスタック（図１０の1006）により報告されたパケット損失率に基づいてビットレートを下げることにより、輻輳を回避しようとし得る。受信機のRTPスタックは、一定の期間中（例えば毎200ms）の受信機のチャネルのパケット損失統計を生成する。これは、低いMTBF要件を有する最小のLPR構成を可能にしつつ、動的に変化するパイプへの高速な適合を可能にする（次に、これは低いオーバーヘッドを可能にする）。

【００７４】

図１１に示すように、DBAの制御ループは、最大レート状態1101と、速度低下状態1102と、速度上昇待機状態1103と、速度上昇状態1104と、常時損失状態1105とを有する状態機械として実現され得る。様々な状態遷移は以下のようになることがわかる。システムが最大レート状態1101（最大データレートが使用されていることを意味する）であることを仮定する。所定の閾値より大きいパケット損失は、速度低下状態1102への状態遷移1106を生じ得る。

【００７５】

速度低下状態において、システムは速度を減少させ、所定の閾値より大きいパケット損失を受け続けるか否かを検出するために待機する。閾値より大きいパケット損失が存在しない場合、システムは、速度上昇待機状態への遷移1108に従い得る。閾値より大きいパケット損失が存在し続け、最大数の速度低下の試行が使用されていない場合（ブロック1107

）、システムは速度低下状態1102を続け、従って、ビットレートを減少させ続け得る。最大数の速度低下の試行が使用された場合、システムは常時損失状態1105に遷移する。

【００７６】

速度上昇待機状態において、所定の閾値より大きい更なるパケット損失は、システムを速度低下状態への遷移1109に従わせ得る。代替として、所定の期間の後に閾値より大きいパケット損失が存在しない場合、システムは、速度上昇状態1104への遷移1110に従い得る。

【００７７】

速度上昇状態1104において、システムは、ビットレートを増加させ、所定の閾値より大きい更なるパケット損失を待機し得る。パケット損失が存在しない場合、システムは、ビットレートが最大ビットレートより小さいか否かを決定し得る。小さい場合、システムは最大レート状態1101への遷移1111に従い得る。小さくない場合、システムは速度上昇状態1104を続け得る。速度上昇状態1104の間に所定の閾値より大きいパケット損失を受ける場合、システムは、所定の最大数の速度上昇の試行が行われたか否かを決定し得る（ブロック1112）。最大数の速度上昇の試行が行われていない場合、システムは、速度上昇待機状態への遷移1113に従い得る。最大数の速度上昇の試みが使用されている場合、システムは最大レート状態1101への遷移1114に従い得る。

【００７８】

前述の状態のそれぞれにおいて、異なるパケット損失閾値が使用されてもよい。更に、状態遷移の一部又は全てが、これらに関連する遅延を有してもよい。速度上昇は、ビットレートの15%ステップの増加で行われてもよい。パイプが完全に利用されており、パケット損失が報告されていない場合、次の速度上昇が生じる前にXタイマ周期（200ms*X）に等しくなり得る遅延がもたらされ得る。初期設定でXの値は2に設定され、400msの遅延を生じてもよい。連続的なパケット損失のレポートではなく、エコーレポートではない何らかのパケット損失は、状態遷移を生じ得る。

【００７９】

図１２は、動的に変化する帯域の可用性とDBA適合とを示している。図１２はまた、３つの一般的な損失のシナリオ（バースト損失、抑制されたパイプ及び常時損失）と共に、状態遷移、エコー遅延を示している。LPR保護の使用は、速度低下状態、速度上昇待機状態及び速度上昇状態の間の図１２の斜線の領域で示されている。速度上昇の試行は、輻輳上限をテストするために、LPR保護のオーバーヘッドを使用し得る。すなわち、ネットワーク輻輳状態が高いビットレートを許容するか否かを調べるために“冗長”データのみが使用されてもよい。輻輳上限のテストが失敗すると、“冗長”データのみが使用されているため、LPRはもたらされたパケット損失により生じた何らかの損失を保護する。このことにより、ビデオの中断のリスクを実質的に低減して、輻輳上限の調査が可能になる。

【００８０】

測定された輻輳上限は、完全にデータフローに適用可能である。例えば、複数のビデオストリームが送信されている場合、輻輳上限は総計ビットレートに適用し得る。前方消失保護は各ビデオストリームに独立して適用可能である。しかし、同じレベルの保護が全てのストリームに適用され得る。システムはまた、共通の前方消失訂正パケットでビデオストリームを併せて保護するように適合されてもよく、各ビデオストリームに異なるレベルの保護を適用するように適合されてもよい。

【００８１】

前方消失訂正はまた、ビデオストリームの一部のみに適用され得る。例えば、階層化ビデオ符号化が使用される場合、前方消失保護は、基本レイヤに適用されて、拡張レイヤに適用されなくてもよい。これはまた、様々なレイヤを異なる程度に保護するように適合されてもよい（いわゆる“不平等な保護（unequal protection）”）。これらの構成では、輻輳回避は全体データフローに当てはまる。

【００８２】

輻輳上限測定の例示的なLPRの調査（probe）がテーブル６に示されている。これらの調

査は、所定値（例えば10%）だけデータレートを上昇させる可能性をテストするために使用され、短い期間（例えば800ms）の間に動作し得る。

［表６］

ビデオレート (kbs)	グループ毎のデータパケット	パケットペイロードサイズ(バイト)	グループ毎の回復パケット	保護期間 (ms)	MTBF (ms)
64	13	87	3	141	2067
128	12	133	3	100	1795
256	12	266	3	100	1795
384	12	400	3	100	1800
512	12	533	3	100	1799
768	12	800	3	100	1800
1024	16	800	3	100	870
2048	16	800	4	50	1158
3072	24	800	6	50	1936
4096	32	800	7	50	1365
5120	32	1000	7	50	1365
6144	39	1000	8	51	1235

テーブル6 10%の速度上昇レートの調査

ここに記載された技術は、受信機が輻輳上限と消失保護レベルとを送信機にフィードバックという点で受信機主導でもよい。代替として、これらは、送信機主導のフレームワーク、又は制御ループが送信機と受信機との間で分散されるハイブリッド型フレームワークで使用され得る。何らかの適合で、マルチキャストストリームでの使用にも適する。記載された技術はまた、パケット再送信、周期的画像リフレッシュ及びビデオ誤り隠蔽のような他の技術と結合され得る。

【００８３】

前述のように、輻輳回避と前方消失訂正との組み合わせは、何れかの機構がそれ自体で実現できない固有の利点を提供する。１つのこのような利点は、ビットストリームでの前方消失保護パケットの量を実験的に増加させ、配信されたデータフローでの影響を観測することにより、輻輳上限が周期的に測定され得るという点にある。これらの保護パケットは、パケット損失からメディアを保護するのと同時に（レベルが輻輳上限を超える場合）、全体データレートを所望の“調査”レベルに増加させる。第２の利点は、輻輳上限を動的に測定するこの方法は、輻輳回避アルゴリズムで通常に使用されている受動的な方法により提供されるものより、許容帯域の非常に高速な決定を生じ得るという点にある。第３の利点は、全体フロー（メディア及び保護）が動的に測定された輻輳上限の下に留まるように抑制されたときに、保護されたメデイアフローが少ない遅延及び小さい保護オーバーヘッドで配信され得るという点にある。

【００８４】

ここに記載された技術の更なる利点は、ビデオデータレートが増加したときに、システムがパケット損失に対して保護するのに効率的になり得るという点にある。従って、高解像度ビデオの使用等でビットレートが増加すると、ここに記載された技術は、有用且つ効率的になる。

【００８５】

前方消失訂正と輻輳回避との統合は、複数の固有の利点を提供する。例えば、輻輳回避は、前方消失訂正に対して複数の固有の利点をもたらす。データリンクが混雑すると、そのリンクを通じた媒体の送信遅延が増加する。前方消失訂正のみを使用することは、許容可能なパケット損失率を生じ得るが、待ち行列遅延を低減しない。輻輳回避を制御ループに統合することにより、非常に小さい待ち時間で所望のサービス品質（QoS）を実現できる。更に、前方消失保護の効率は、大きい待ち時間のアプリケーションより小さい待ち時

間のアプリケーションで小さくなる。データ損失の原因が簡単な輻輳である場合、ビットレートを低減することは、データフローに更なる前方消失訂正パケットを増加することより効率的になり得る。

【００８６】

逆に、前方消失訂正はまた、輻輳回避のみのものにかなりの価値を追加し得る。典型的には、輻輳回避技術は、データフローのレートを積極的に低減する。例えば、TCPの輻輳機構は、乗算係数だけデータレートを低減するが、加算係数を適用することにより非常に低く上がる。この理由は、データフローレートを増加することは、常に輻輳を生じるリスクがあり、従ってパケット損失を生成するからである。前方消失訂正が輻輳回避と統合されると、このリスクはかなり低減される。従って、前述のように、前方消失保護パケットの密度を単に増加させることにより、データレートが増加し得る。

【００８７】

更に、輻輳回避技術はまた、リンク速度を推定するように設計される。他のデータフローからの相互輻輳（cross-congestion）は、頻繁にパケット損失を生じ得る。相互輻輳が観測される時間までに、通常ではパケット損失は既に生じている。前方消失保護を含めることにより、このような相互輻輳によるパケット損失が回復可能になる。

【００８８】

ここに開示された技術は、マルチメディアデータ（ビデオデータ、オーディオデータ、静止画データ及び他の種類のデータ等）の送信用の何らかのシステムを含み、様々なシステムで使用され得る。このような技術は、典型的な汎用コンピュータシステム（デスクトップコンピュータ、ノートブックコンピュータ、ハンドヘルドコンピュータ、サーバ等）で使用され得る。代替として、これらの技術は、様々な装置形式のシステム（会議部屋のテレビ会議システム、デスクトップ型テレビ会議システム等）で使用され得る。ここに記載された技術はまた、インフラストラクチャ形式のテレビ会議装置（マルチポイント制御ユニット（MCU：multi-point control unit）、ブリッジ、メディアサーバ等）で使用され得る。

【００８９】

ここに記載された方法は、１つ以上のコンピュータプログラムにコード化され、コンピュータ可読媒体（コンパクトディスク、テープ、揮発性又は不揮発性メモリ等）に格納され得る。従って、プログラム記憶装置に格納された命令は、ここに記載された技術をプログラム可能制御装置（例えば、コンピュータ又は会議ユニット）に実行させるために使用され得る。開示された通信システムは、近端（near end）及び遠端（far end）の間の双方向通信を提供するものとして記載されているが、この開示の教示は一方向送信を提供するシステムにも適用可能であることがわかる。

【００９０】

好ましい実施例及び他の実施例の前述の説明は、出願人により考案された本発明の概念の範囲又は適用性を制限又は限定することを意図しない。ここに含まれる本発明の概念を開示する代わりに、出願人は特許請求の範囲により提供される全ての特許権を望む。従って、特許請求の範囲は、請求項又はその均等の範囲内にある全ての内容に対する全ての変更及び変形を含むことを意図する。

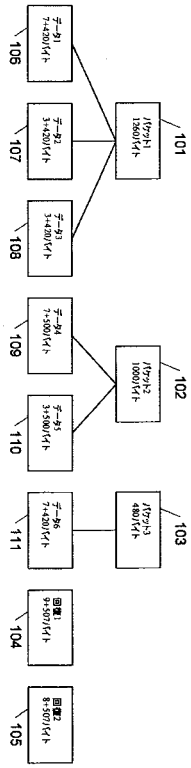
【図面の簡単な説明】

【００９１】

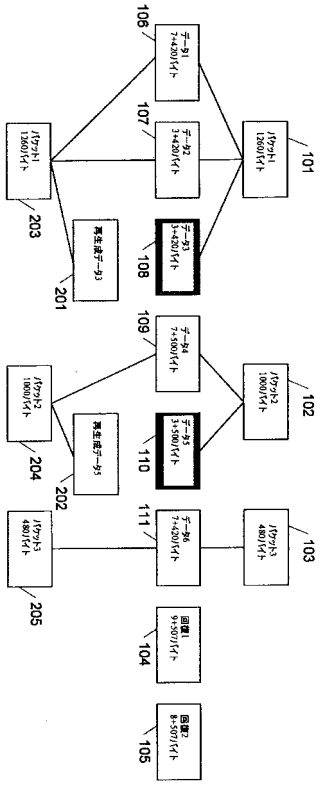
【図１】パケット損失を防ぐために使用され得る損失パケット回復技術
【図２】特定のパケットが損失してデータが再構成される損失パケット回復技術
【図３】特定のパケットが損失してデータが再構成される損失パケット回復技術
【図４】損失パケット回復エンコーダ
【図５】図１～４の損失パケット回復技術と共に使用され得るパケットヘッダ
【図６】図１～４の損失パケット回復技術と共に使用され得るパケットヘッダ
【図７】図１～４の損失パケット回復技術と共に使用され得るパケットヘッダ
【図８】図１～４の損失パケット回復技術と共に使用され得るパケットヘッダ

【図９】図１～４の損失パケット回復技術と共に使用され得るパケットヘッダ
【図１０】損失パケット回復デコーダ
【図１１】図１～１０の損失パケット回復技術、エンコーダ及びデコーダと共に使用され得る輻輳回避アルゴリズムの状態機械
【図１２】図１～１０による技術、エンコーダ、デコーダ及びアルゴリズムを使用して損失パケット回復と輻輳回避とを使用するエンコーダのデータレート

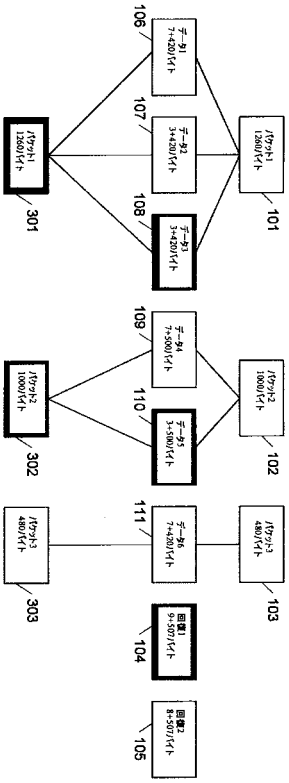
【図１】



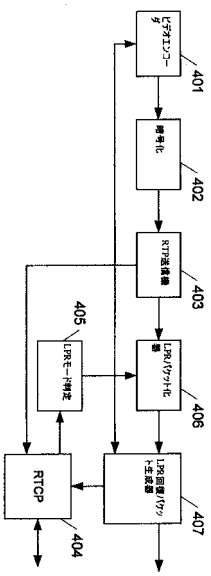
【図２】



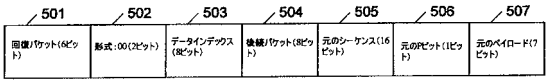
【図３】



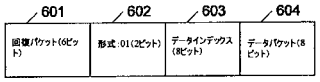
【図４】



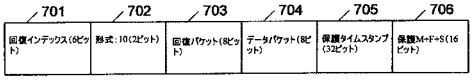
【図５】



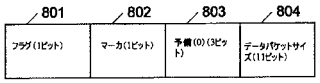
【図６】



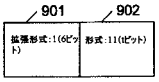
【図７】



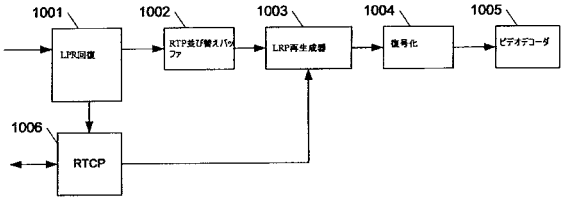
【図８】



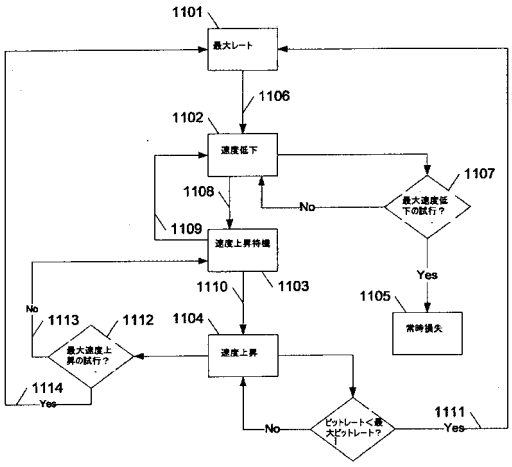
【図９】



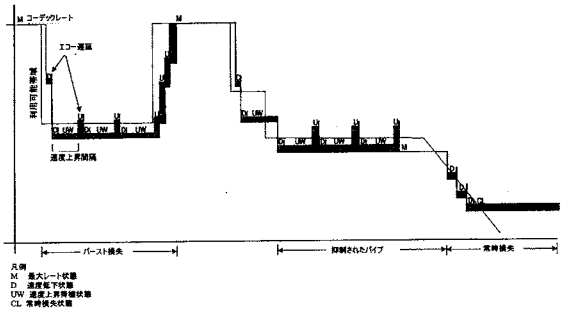
【図１０】



【図１１】



【図１２】


M コーデックレート
エコー遅延
利用可能帯域
速度上昇間隔
バースト損失
抑制されたパイプ
常時損失
凡例
M 最大レート状態
D 速度低下状態
UW 速度上昇待機状態
CL 常時損失状態

フロントページの続き

(72)発明者　ジョン　ポール　スピアーマン
アメリカ合衆国　テキサス州　７８７４８，オースティン，アミュニション・ドライヴ　３００７
(72)発明者　リチャード　イー　フロット，ジュニア
アメリカ合衆国　テキサス州　７８７２６，オースティン，アメサイト・トレイル　１１１２
(72)発明者　デイヴィッド　ヘイン
アメリカ合衆国　テキサス州　７８７３５，オースティン，フリース・フラワー・コヴ　３２００

Ｆターム(参考) 5K014 BA08　EA01

【外国語明細書】

1. Title of Invention

SYSTEM AND METHOD FOR LOST PACKET RECOVERY WITH CONGESTION AVOIDANCE

2. Detailed Description of Invention

BACKGROUND

[0001] Increasingly, videoconferencing systems have begun to use packet-switched networks, such as the Internet, rather than circuit-switched networks, such as PSTN and ISDN. One problem with the use of packet-switched networks for videoconferencing is that all packet-switched networks will experience some degree of packet loss. Video quality is signficantly impaired with even a modest packet loss.

[0002] Existing solutions to the problem of packet loss include the use of various error concealment techniques and picture refresh mechanisms. Though helpful, these techniques are often inadequate. They also are frequently integrated with specific video codecs, requiring the techniques to be re-invented for each advance in codec technology.

[0003] Packet-switched networks typically have at least two fundamental types of packet loss. Physical loss can occur when layers 1 and 2 of networks lose information that is not recovered. This type of loss is more common, for example, when radio links (*e.g.*, WiFi networks such as 802.11a, 802.11b, 802.11g, and 802.11n) are employed. However, physical layer loss can occur on wired and fiber links as well. A second type of packet loss can result from network congestion.

[0004] To overcome this second type of packet loss, some videoconferencing devices employ various techniques of congestion avoidance. One such technique is described in U.S. Patent Application 10/305,485, entitled "System and Method for Dynamic Bandwidth Allocation for Videoconferencing in Lossy Packet Switched Systems," filed November 26, 2002, which is incorporated by reference in its entirety. Such technqiues may be referred to as "Dynamic Bandwidth Allocation" or "DBA." These types of DBA algorithms

inherently risk introducing packet loss whenever they attempt to increase the bandwidth, which can result in impaired media. Other types of congestion avoidance techniques include VCON's PacketAssist and other standards relating to either lost packet recovery or congestion control. For instance, RFC 2733 provides a method for lost packet recovery which employs XOR (parity) packets. RFC 3448 provides a method a unicast congestion control, as does RFC 4340.

[0005] Additionally, lost data recovery techniques such as forward erasure correction and forward error correction can be employed on layers 1 and 2 of networks. These techniques have also been used with stored media such as RAID disks and CDROMs/DVDs. 3GPP has recently standardized forward erasure correction. Forward error correction can also provided for video streams transmitted via H.320.

[0006] Heretofore, there has been no implementation integrating lost packet recovery (*e.g.*, using forward erasure correction) with congestion avoidance. Described herein is an algorithm that includes such integration.

SUMMARY

[0007] In one aspect, the present invention relates to a method of preventing disruption of a media stream transmitted across a lossy network. The method can include applying a lost packet recovery algorithm to the media stream. The lost packet recovery mechanism can operate by inserting redundant information into a transmitted data stream that includes the media stream. The redundant information can be generated using forward erasure correction and/or Reed-Solomon encoding. The method can further include applying a congestion avoidance algorithm to the transmitted data stream. The congestion avoidance algorithm can include temporarily increasing a data rate of the transmitted data stream to determine whether the network can support a higher data rate. The data rate can be temporarily increased by increasing the amount of redundant information inserted into the transmitted data stream.

[0008] The present invention can also relate to a method of recovering media data received over a lossy network, wherein the media stream was transmitted as part of a data stream comprising the media stream and redundant

information. The recovery method can include receiving a plurality of packets containing the media stream and the redundant information and reconstructing lost portions of the media stream from the received packets.

[0009] The present invention can also relate to a videoconferencing device adapted to perform data recovery and congestion avoidance as described above.

DETAILED DESCRIPTION

[0010] Disclosed herein are techniques for overcoming the effect of lost packets in network transmission that combine the use of forward erasure correction and congestion avoidance. These techniques may be referred to as "Lost Packet Recovery" or "LPR."

[0011] LPR is a scalable method for recovery of lost RTP media packets. It is not media specific, although the examples described herein are intended for video codecs. LPR techniques described herein can also be used to protect other types of RTP media streams. LPR uses Reed-Solomon encoding to provide forward erasure correction of lost media packets, which allows for recovery of the lost media packets.

[0012] When LPR is employed the RTP payload is re-packetized to create data packets of approximately equal size. The re-packetized stream uses a distinct RTP dynamic payload type (which differs from the assigned media payload number), and is fully RTP compliant. A payload-specific header includes enough information to reconstruct the original RTP packets. LPR assumes that the SSRC and CSRC information does not change in the middle of a recovery set (as defined below). Timestamps, sequence numbers, the original payload type, and the marker bit are fully recovered, which ensures that the recovered RTP stream can be decrypted.

[0013] Recovery packets are then added to this RTP stream. The recovery packets are also RTP compliant. Each recovery packet also includes a payload header that describes the protected data packets. Each packet's RTP payload includes the recovery information. The number of recovery packets added depends on the channel loss rate as estimated from RTCP receiver reports.

Channel loss rates of up to about 15% can be accommodated, albeit with a bit rate overhead of about 50%.

[0014] It should be noted that the LPR protection is not considered to be part of the overall media bit rate. Only the actual compressed payload is counted as such. However, the aggregated bit rate can be reduced as necessary to avoid network congestion. It should be understood that network congestion can have undesirable affects other than or in addition to packet loss. For example, network congestion can cause increased media latency and dropped connections (including signaling connections). Therefore, the LPR protocol can have built-in messages for congestion avoidance. One suitable congestion avoidance algorithm is described in greater detail below, although a variety of congestion avoidance algorithms may be used.

[0015] An example of LPR applied to a media stream will now be described with reference to Figs. 1–3. The details of the example, including bit rates, packet loss rates, packet sizes, ratio of data packets to recovery packets, etc. are all mere examples and can vary depending on implementation details, channel conditions, etc.

[0016] Consider a 300 kbps video channel that has a packet loss rate of 2%. Three packets 101, 102, and 103 are to be transmitted. An LPR mode of (6+2) can be used, meaning that two recovery packets 104, 105 will be inserted into each group of six data packets 106–111, creating recovery sets of eight packets each. The recovery packets 104, 105 are about equal in size to the largest data packet, but include some additional overhead. The recovery packets allow regeneration of any two of the six data packets 106–111 as described in greater detail below.

[0017] To stay within the 300 kbps channel limit, the media rate can be reduced to approximately 220 kbps, leaving 80 kbps for the recovery packets. The target size for each re-packetized data packet can be 500 bytes. Each recovery set therefore carries about 32000 bits of information, representing a time period of 107 ms at the channel rate of 300 kbps. The 107 ms period is called the protection period.

[0018] Each of the packets (*i.e.*, the re-packetized data packets 106–111 and the recovery packets 104, 105) contains the normal 40 bytes of IP+UDP+RTP header overhead. For simplicity, this overhead is not shown. However, the overhead associated with LPR itself is shown. For example, "3+420" indicates that three overhead bytes plus 420 bytes of the original data are in the packet. The recovery packets are 100% overhead. The details of the overhead are described in greater detail below.

[0019] Figure 2 illustrates the recovery process. Assume that data packets 3 and 5 (108, 110) are lost. The LPR decoder creates regenerated packets 201 and 202 from the six packets that were received, *i.e.*, data packets 1, 2, 4, and 6 and recovery packets 1 and 2. Note that regeneration of the lost packets cannot occur until the sixth packet is received. After regeneration, the original RTP packets 203–205 are restored from the data packets. Regeneration of the lost packets is described in greater detail below.

[0020] Figure 3 illustrates a failed packet recovery where more than two packets are lost. In this example Recovery Packet 1 is lost as well as Data Packets 3 and 5. Since only 5 packets of the group were received, packet regeneration is not possible. The LPR decoder does not regenerate partial packets. If the original packet can not be recovered in its entirety, then the partial packet information is discarded. Therefore, data packets 1 and 2, 301, 302 are both shown as lost.

[0021] In the event that all of the original data packets cannot be recovered, as illustrated in Fig. 3, it is typical for the receiving decoder to request a fast update of the entire image. Alternatively, a walk-around refresh such as that described in U.S. Patent 7,020,203, entitled "Dynamic Intra-coded Macroblock Refresh Interval for Video Error Concealment," issued March 28, 2006, which is hereby incorporated by reference in its entirety. In either case, the time required to repair the image is not a function of the number of packets lost, but rather the mean time between failures caused by losing more than the predetermined number of packets in the recovery set (two, in this example).

[0022] For the given example of a channel at 300 kbps and 2% packet loss, with each protection set of six data packets and two recovery data packets covering 107 ms of channel data, the mean time between failures can be computed

to be approximately 258 seconds. Assuming that a fast update requires approximately two seconds, the video transmitted on a channel protected by the LPR example given will be error-free more than 99% of the time. This protection comes at an additional overhead of 80 kbps, which is approximately 30% of the channel rate.

[0023] Additional aspects of the LPR example given above may be better understood with reference to Fig. 4, which illustrates an implementation of an LPR encoder. Variations of this implementation are also possible. Video encoder 401 can be any of a variety of video encoders operating in a more or less conventional matter. For example, the video encoder can operate according to the H.261, H.263, H.264, MPEG-2, or MPEG-4 video compression standards. Alternatively, the video encoder can operate according to any of a variety of proprietary video coding techniques.

[0024] The output of video encoder 401 can be supplied to optional encryption module 402, which can also be of any of a variety of known types. The encrypted video data (or unencrypted video data, if encryption is not used) can be supplied to RTP sender 403, which can also be conventional. The RTP packets generated by RTP sender 403 can then be re-packetized using the LPR algorithm by LPR packetizer 406 as described in greater detail below. LPR recovery packet generator 407 can then generate the recovery packets based on information received from LPR packetizer 406 and video encoder 401, as described in greater detail below. The LPR recovery packet generator can then transmit the LPR data packets and the LPR recovery packets to the receiver (not shown).

[0025] RTCP module 404 can receive packet loss and other channel statistics and supply these to LPR mode decision module 405, which controls LPR packetizer 406. LPR mode decision module 405 can decide whether to engage or disengage LPR protection, as well as determine the LPR protection parameters to be used. If LPR protection is disengaged, LPR packetizer 406 and LPR recovery packet generator 407 can pass through the RTP packets generated by RTP sender 403 without alteration.

[0026] LPR mode decision module 405 can operate as follows. First the protection period (as described above) can be determined. Longer protection periods offer more efficient packet regeneration (*e.g.*, fewer recovery kbps), but create more latency. For most typical videoconferencing bit rates, 100 ms can be a suitable protection period. At bit rates below 128 Kbps longer protection periods might be necessary. For example, 150 ms can be suitable for a 64 Kbps rate. At bit rates above 1 Mbps shorter protection periods might be advantageous. For example, 50 ms can be suitable for rates of 2 Mbps or more. Other parameters can also be considered in determining the protection period.

[0027] Next, the number of total packets per protection period can be determined. In general, small numbers of packets per protection period may tend to decrease the efficiency of the protection. Conversely, larger numbers of packets per protection period may tend to increase the amount of computation required to encode and decode the protection packets. Therefore, for most channels, about thirteen or more packets per protection period is believed to be suitable. The decision on the number of packets per protection period should also take into account the resulting packet size for the desired data rate. The output packet size (including IP overhead) should be less than 1260 bytes to traverse an IPSEC tunnel without fragmentation.

[0028] Once the number of packets per protection period is established, the number of recovery packets per protection period can be chosen. The number of recovery packets can be chosen (using conventional probabilistic analysis techniques) so that the mean time between protection breakdowns meets or exceeds a predetermined specified value. Additionally, the aggregation of all transmitted video channels (including the LPR protection packets and RTP messages) can be selected so as not to exceed a specified congestion ceiling.

[0029] The tables below give example protection modes that can be used for various bitrates under various loss conditions. Table 1 gives LPR parameters that may be suitable for 4% packet loss conditions that can be used in conjunction with congestion avoidance algorithms (as described below) that reduce the data rate when packet loss exceeds 3%. Table 2 gives LPR parameters for 2% packet

loss conditions that may be used when congestion avoidance algorithms are not in use.

Video Rate (k\bs)	Data Packets Per Group	Packet Payload Size (bytes)	Recovery Packets Per Group	Protection Period (ms)	MTBF (sec)
64	13	87	4	141	334
128	12	133	4	100	323
256	12	266	4	100	323
384	12	400	4	100	324
512	12	533	4	100	324
768	12	800	4	100	324
1024	16	800	5	100	757
2048	16	800	5	50	379
3072	24	800	6	50	338
4096	32	800	7	50	378
5120	32	1000	7	50	378
6144	39	1000	8	51	567

Table 1 Constant Protection at 4% Packet Loss

Video Rate (k\bs)	Data Packets Per Group	Packet Payload Size (bytes)	Recovery Packets Per Group	Protection Period (ms)	MTBF (sec)
64	13	87	3	141	589
128	12	133	3	100	545
256	12	266	3	100	545
384	12	400	3	100	546
512	12	533	3	100	546
768	12	800	3	100	546
1024	16	800	4	100	2591
2048	16	800	4	50	1296
3072	24	800	5	50	2444
4096	32	800	5	50	573
5120	32	1000	5	50	573
6144	39	1000	6	51	1704

Table 2 Constant Protection at 2% Packet Loss

[0030] LPR packetizer 406 can operate as follows. LPR packetizer 406 sub-divides the original media packets generated by RTP sender 403 so as to improve the efficiency of packet regeneration coding. An LPR data packet contains information from only one source packet. If the media channel is encrypted, LPR re-packetization can be performed on the encrypted stream. In general, the LPR stream itself is not re-encrypted, *i.e.*, the LPR encapsulation

fields (headers) added to the encrypted packets and newly generated LPR recovery packets are sent in the clear.

[0031] If an original source packet is sub-divided (*e.g.*, Packet 1 101 in Figs. 1–3), the first LPR data packet for this source packet is called the *initial* data packet (*e.g.*, Data 1 106 in Figs. 1–3). Subsequent data packets (if any) are called *follow-on* data packets (*e.g.*, Data 2 107 and Data 3 108 in Figs. 1–3). The number of follow-on data packets is signaled in the initial data packet and therefore cannot be changed once the initial data packet is sent. The follow-on data packets and the initial data packet may be in different LPR protection groups.

[0032] All fields in the original RTP header except for the sequence number, the payload type, and the P field can be carried unchanged in each LPR data packet. This includes the time stamp, the marker bit, and the SSRC/CSRC information. The payload number can be replaced with the negotiated LPR payload type. Sequence numbers can be assigned sequentially to the LPR media stream in the usual fashion. The P field of the LPR packet can be set to 0. LPR packetizer 406 can pass through the original RTP sequence numbers to LPR recovery packet generator 407.

[0033] Each initial data packet can include a seven-byte payload header that is specific to LPR, an example of which is illustrated in Fig. 5. Each follow-on data packet can include a three-byte payload header, an example of which is illustrated in Fig. 6. These payload headers can immediately follow the RTP header and precede the video (or other media) payload. The header sizes and individual field sizes and their arrangement are exemplary only, and it will be understood that other arrangements are possible.

[0034] The recovery packet field 501, 601 can be used to indicate the number of recovery packets that will subsequently be sent in the recovery set. All data packets in a recovery set should signal the same number of recovery packets. The number of recovery packets signaled can be 0, indicating that no recovery packets will be sent. This field can be a six-bit field, which would allow for up to sixty-four recovery packets. The type field 502, 602 can be a two-bit field set to either 00 or 01 to indicate an initial data packet or a follow-on data packet,

respectively. This allows the header type for LPR to be determined by examining the first payload byte.

[0035] Data index field 503, 603 can be an 8-bit field. The first data packet in a recovery set can be assigned a data index of 1. The index can be incremented for subsequent data packets in the set. The follow-on packets field 504 can be an 8-bit field used to specify the number of follow-on packets associated with the initial packet. The data packet field 604 can be an eight-bit field used to signal the number of data packets included in the recovery set. All data packets should signal the same number of data packets.

[0036] Original sequence field 505 can be a sixteen-bit field that indicates the original RTP packet sequence number, which will be used to recreate the original RTP packets. Original p-bit field 506 can be a one-bit field used to carry the padding bit carried by the original RTP packet. Original payload type field 507 can be a seven-bit field used to carry the payload type as carried by the original RTP packet.

[0037] LPR recovery packet generator 407 can operate as follows. LPR recovery packet generator 407 can replace the RTP sequence numbers of the original media with the final LPR sequence numbers in the LPR data packets. LPR recovery packet generator 407 can also inserts the LPR recovery packets at the end of each recovery set.

[0038] Each recovery packet can also include a nine-byte header such as that illustrated in Fig. 7. The recovery packet header can immediately follows the RTP header and can precede the video (or other media) payload. As with the header described above, the field types and sizes are exemplary and other arrangements may also be used.

[0039] Recovery index field 701 can be a six-bit field used to indicate the sequence of the recovery packet in the recovery set. The first recovery packet can be assigned a recovery index of 1, which can be incremented for subsequent recovery packets in the recovery set. Type field 702 can be a two-bit field set to 10 to indicate the packet is a recovery packet. The recovery packets field can be an eight-bit field that includes the number of data packets in the recovery set. All recovery packets should signal the same number of recovery packets signalled by

the corresponding data packets. Data packets field 704 can be an eight-bit field that indicates the number of data packets sent in the recovery set. The recovery packets should indicate the same number of data packets indicated in the corresponding data packets.

[0040] Protected timestamps field 705 can be a 32 bit field that holds Reed-Solomon encoded RTP timestamps for each data packet in the recovery set. This can combined using the same Reed-Solomon encoding as used in the recovery payload itself. This field can allow the RTP timestamp of a missing data packet to be regenerated. The 4 bytes are in network order.

[0041] Protected marker, flag, and size field 706 (futher illustrated in Fig. 8) can hold the Reed-Solomon encoded RTP marker bit 802, a flag value 801, and data packet size 803. The flag value can be 1 if the LPR data packet is an initial packet, and can be set to 0 if the LPR data packet is a follow-on packet. The size can include the IP, UDP, and RTP header as well as the LPR header. The data packet size can be in network order.

[0042] The protected field is combined using the same Reed-Solomon encoding as is used in the recovery payload itself. The field allows the LPR header type, marker and data packet size to be regenerated for any missing data packet that can be recovered.

[0043] The payload of the recovery packet can be encoded using Reed-Solomon 256 (RS256) encoding. The Original Sequence 505, Original P bit 506, Original Payload Type 507 and Follow-on Packets 504 fields in the initial data packet can be protected as if they were payload bytes. This can allow this information to be recovered if the initial data packet is lost. Additionally, the recovery packet payload header can include encodings of three Phantom Data fields: the length of the data packet itself, the marker bit, and the timestamp. Again, this can allow this information to be recovered if all data packets for a sub-divided RTP packet are lost.

[0044] Each payload byte in each data packet can contribute to the corresponding byte in each recovery packet, according to the generating function for RS256:

$$R_j = \sum_{i=1}^{d} B_i^{j-1} \otimes D_i$$

where: *Di* is the *i*th data packet ($1 \le i \le d$); *Rj* is the *j*th recovery packet ($1 \le j \le r$); *Bi* is the *i*th base coefficient ($1 \le i \le d$) having values as specified in table 3; and $B_i^{j-1}, \otimes, \sum$ are computed using Galois Field (2^8) arithmetic operations as described below. It should be noted that the given base coefficient table restricts the maximum number of data packets in a recovery set to 128, although other numbers of data packets can be selected, and a suitable table can be computed accordingly.

```
/*
** RS256_BaseTable[i] holds the base coefficient for data packet i-1
**
** These entries are selected to ensure that the dispersal matrix is invertible.
** Each entry b is chosen so b**x is not 1 unless x is 0 or 255.
*/
#define RS256_BASETABLE_LEN 128

STATIC CONST UINT8 RS256_BaseTable[RS256_BASETABLE_LEN]= {
 0x02, 0x04, 0x06, 0x09, 0x0D, 0x0E, 0x10, 0x12,
 0x13, 0x14, 0x16, 0x18, 0x19, 0x1B, 0x1D, 0x1E,
 0x1F, 0x22, 0x23, 0x28, 0x2A, 0x2B, 0x30, 0x31,
 0x32, 0x34, 0x39, 0x3C, 0x3F, 0x41, 0x42, 0x43,
 0x47, 0x48, 0x49, 0x4A, 0x4B, 0x4C, 0x51, 0x52,
 0x53, 0x54, 0x58, 0x5A, 0x5B, 0x5C, 0x5D, 0x5F,
 0x62, 0x63, 0x68, 0x69, 0x6D, 0x6F, 0x70, 0x71,
 0x76, 0x77, 0x79, 0x7A, 0x7B, 0x7E, 0x80, 0x81,
 0x83, 0x85, 0x87, 0x88, 0x89, 0x8C, 0x8D, 0x8E,
 0x90, 0x94, 0x95, 0x97, 0x9A, 0x9B, 0x9D, 0x9E,
 0x9F, 0xA2, 0xA3, 0xA4, 0xA5, 0xAA, 0xAB, 0xAF,
 0xB0, 0xB1, 0xB2, 0xB7, 0xBB, 0xBC, 0xBD, 0xC0,
 0xC2, 0xC6, 0xC7, 0xC8, 0xC9, 0xCA, 0xCB, 0xCC,
 0xD1, 0xD2, 0xD3, 0xD4, 0xD5, 0xD8, 0xDA, 0xDE,
 0xE0, 0xE1, 0xE3, 0xE5, 0xE8, 0xEA, 0xEC, 0xEE,
 0xF0, 0xF3, 0xF6, 0xF7, 0xF8, 0xF9, 0xFA, 0xFE
};
```

Table 3. RS256 Base Coefficients

[0045] Galois field (2^8) arithmetic operations can be set up to use two helper tables: a Galois Log function (glog) table, and a Galois Exponentiation function (gexp) table. These tables are:

```
unsigned short glog[256] = {
 0x200, 0x00, 0x01, 0x19, 0x02, 0x32, 0x1A, 0xC6,    /* 0 to 7 */
 0x03, 0xDF, 0x33, 0xEE, 0x1B, 0x68, 0xC7, 0x4B,     /* 8 to 15 */
 0x04, 0x64, 0xE0, 0x0E, 0x34, 0x8D, 0xEF, 0x81,     /* 16 to 23 */
 0x1C, 0xC1, 0x69, 0xF8, 0xC8, 0x08, 0x4C, 0x71,     /* 24 to 31 */
 0x05, 0x8A, 0x65, 0x2F, 0xE1, 0x24, 0x0F, 0x21,     /* 32 to 39 */
 0x35, 0x93, 0x8E, 0xDA, 0xF0, 0x12, 0x82, 0x45,     /* 40 to 47 */
 0x1D, 0xB5, 0xC2, 0x7D, 0x6A, 0x27, 0xF9, 0xB9,     /* 48 to 55 */
 0xC9, 0x9A, 0x09, 0x78, 0x4D, 0xE4, 0x72, 0xA6,     /* 56 to 63 */
 0x06, 0xBF, 0x8B, 0x62, 0x66, 0xDD, 0x30, 0xFD,     /* 64 to 71 */
 0xE2, 0x98, 0x25, 0xB3, 0x10, 0x91, 0x22, 0x88,     /* 72 to 79 */
 0x36, 0xD0, 0x94, 0xCE, 0x8F, 0x96, 0xDB, 0xBD,     /* 80 to 87 */
 0xF1, 0xD2, 0x13, 0x5C, 0x83, 0x38, 0x46, 0x40,     /* 88 to 95 */
```

```
 0x1E, 0x42, 0xB6, 0xA3, 0xC3, 0x48, 0x7E, 0x6E,     /* 96 to 103 */
 0x6B, 0x3A, 0x28, 0x54, 0xFA, 0x85, 0xBA, 0x3D,     /* 104 to 111 */
 0xCA, 0x5E, 0x9B, 0x9F, 0x0A, 0x15, 0x79, 0x2B,     /* 112 to 119 */
 0x4E, 0xD4, 0xE5, 0xAC, 0x73, 0xF3, 0xA7, 0x57,     /* 120 to 127 */
 0x07, 0x70, 0xC0, 0xF7, 0x8C, 0x80, 0x63, 0x0D,     /* 128 to 135 */
 0x67, 0x4A, 0xDE, 0xED, 0x31, 0xC5, 0xFE, 0x18,     /* 136 to 143 */
 0xE3, 0xA5, 0x99, 0x77, 0x26, 0xB8, 0xB4, 0x7C,     /* 144 to 151 */
 0x11, 0x44, 0x92, 0xD9, 0x23, 0x20, 0x89, 0x2E,     /* 152 to 159 */
 0x37, 0x3F, 0xD1, 0x5B, 0x95, 0xBC, 0xCF, 0xCD,     /* 160 to 167 */
 0x90, 0x87, 0x97, 0xB2, 0xDC, 0xFC, 0xBE, 0x61,     /* 168 to 175 */
 0xF2, 0x56, 0xD3, 0xAB, 0x14, 0x2A, 0x5D, 0x9E,     /* 176 to 183 */
 0x84, 0x3C, 0x39, 0x53, 0x47, 0x6D, 0x41, 0xA2,     /* 184 to 191 */
 0x1F, 0x2D, 0x43, 0xD8, 0xB7, 0x7B, 0xA4, 0x76,     /* 192 to 199 */
 0xC4, 0x17, 0x49, 0xEC, 0x7F, 0x0C, 0x6F, 0xF6,     /* 200 to 207 */
 0x6C, 0xA1, 0x3B, 0x52, 0x29, 0x9D, 0x55, 0xAA,     /* 208 to 215 */
 0xFB, 0x60, 0x86, 0xB1, 0xBB, 0xCC, 0x3E, 0x5A,     /* 216 to 223 */
 0xCB, 0x59, 0x5F, 0xB0, 0x9C, 0xA9, 0xA0, 0x51,     /* 224 to 231 */
 0x0B, 0xF5, 0x16, 0xEB, 0x7A, 0x75, 0x2C, 0xD7,     /* 232 to 239 */
 0x4F, 0xAE, 0xD5, 0xE9, 0xE6, 0xE7, 0xAD, 0xE8,     /* 240 to 247 */
 0x74, 0xD6, 0xF4, 0xEA, 0xA8, 0x50, 0x58, 0xAF      /* 248 to 255 */
};
```

Table 4. Galois Log Helper Function

```
unsigned char gexp[1065] = {
 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,     /* 0 to 7 */
 0x1D, 0x3A, 0x74, 0xE8, 0xCD, 0x87, 0x13, 0x26,     /* 8 to 15 */
 0x4C, 0x98, 0x2D, 0x5A, 0xB4, 0x75, 0xEA, 0xC9,     /* 16 to 23 */
 0x8F, 0x03, 0x06, 0x0C, 0x18, 0x30, 0x60, 0xC0,     /* 24 to 31 */
 0x9D, 0x27, 0x4E, 0x9C, 0x25, 0x4A, 0x94, 0x35,     /* 32 to 39 */
 0x6A, 0xD4, 0xB5, 0x77, 0xEE, 0xC1, 0x9F, 0x23,     /* 40 to 47 */
 0x46, 0x8C, 0x05, 0x0A, 0x14, 0x28, 0x50, 0xA0,     /* 48 to 55 */
 0x5D, 0xBA, 0x69, 0xD2, 0xB9, 0x6F, 0xDE, 0xA1,     /* 56 to 63 */
 0x5F, 0xBE, 0x61, 0xC2, 0x99, 0x2F, 0x5E, 0xBC,     /* 64 to 71 */
 0x65, 0xCA, 0x89, 0x0F, 0x1E, 0x3C, 0x78, 0xF0,     /* 72 to 79 */
 0xFD, 0xE7, 0xD3, 0xBB, 0x6B, 0xD6, 0xB1, 0x7F,     /* 80 to 87 */
 0xFE, 0xE1, 0xDF, 0xA3, 0x5B, 0xB6, 0x71, 0xE2,     /* 88 to 95 */
 0xD9, 0xAF, 0x43, 0x86, 0x11, 0x22, 0x44, 0x88,     /* 96 to 103 */
 0x0D, 0x1A, 0x34, 0x68, 0xD0, 0xBD, 0x67, 0xCE,     /* 104 to 111 */
 0x81, 0x1F, 0x3E, 0x7C, 0xF8, 0xED, 0xC7, 0x93,     /* 112 to 119 */
 0x3B, 0x76, 0xEC, 0xC5, 0x97, 0x33, 0x66, 0xCC,     /* 120 to 127 */
 0x85, 0x17, 0x2E, 0x5C, 0xB8, 0x6D, 0xDA, 0xA9,     /* 128 to 135 */
 0x4F, 0x9E, 0x21, 0x42, 0x84, 0x15, 0x2A, 0x54,     /* 136 to 143 */
 0xA8, 0x4D, 0x9A, 0x29, 0x52, 0xA4, 0x55, 0xAA,     /* 144 to 151 */
 0x49, 0x92, 0x39, 0x72, 0xE4, 0xD5, 0xB7, 0x73,     /* 152 to 159 */
 0xE6, 0xD1, 0xBF, 0x63, 0xC6, 0x91, 0x3F, 0x7E,     /* 160 to 167 */
 0xFC, 0xE5, 0xD7, 0xB3, 0x7B, 0xF6, 0xF1, 0xFF,     /* 168 to 175 */
 0xE3, 0xDB, 0xAB, 0x4B, 0x96, 0x31, 0x62, 0xC4,     /* 176 to 183 */
 0x95, 0x37, 0x6E, 0xDC, 0xA5, 0x57, 0xAE, 0x41,     /* 184 to 191 */
 0x82, 0x19, 0x32, 0x64, 0xC8, 0x8D, 0x07, 0x0E,     /* 192 to 199 */
 0x1C, 0x38, 0x70, 0xE0, 0xDD, 0xA7, 0x53, 0xA6,     /* 200 to 207 */
 0x51, 0xA2, 0x59, 0xB2, 0x79, 0xF2, 0xF9, 0xEF,     /* 208 to 215 */
 0xC3, 0x9B, 0x2B, 0x56, 0xAC, 0x45, 0x8A, 0x09,     /* 216 to 223 */
 0x12, 0x24, 0x48, 0x90, 0x3D, 0x7A, 0xF4, 0xF5,     /* 224 to 231 */
 0xF7, 0xF3, 0xFB, 0xEB, 0xCB, 0x8B, 0x0B, 0x16,     /* 232 to 239 */
 0x2C, 0x58, 0xB0, 0x7D, 0xFA, 0xE9, 0xCF, 0x83,     /* 240 to 247 */
 0x1B, 0x36, 0x6C, 0xD8, 0xAD, 0x47, 0x8E, 0x01,     /* 248 to 255 */
 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1D,     /* 256 to 263 */
 0x3A, 0x74, 0xE8, 0xCD, 0x87, 0x13, 0x26, 0x4C,     /* 264 to 271 */
 0x98, 0x2D, 0x5A, 0xB4, 0x75, 0xEA, 0xC9, 0x8F,     /* 272 to 279 */
 0x03, 0x06, 0x0C, 0x18, 0x30, 0x60, 0xC0, 0x9D,     /* 280 to 287 */
 0x27, 0x4E, 0x9C, 0x25, 0x4A, 0x94, 0x35, 0x6A,     /* 288 to 295 */
 0xD4, 0xB5, 0x77, 0xEE, 0xC1, 0x9F, 0x23, 0x46,     /* 296 to 303 */
 0x8C, 0x05, 0x0A, 0x14, 0x28, 0x50, 0xA0, 0x5D,     /* 304 to 311 */
 0xBA, 0x69, 0xD2, 0xB9, 0x6F, 0xDE, 0xA1, 0x5F,     /* 312 to 319 */
 0xBE, 0x61, 0xC2, 0x99, 0x2F, 0x5E, 0xBC, 0x65,     /* 320 to 327 */
 0xCA, 0x89, 0x0F, 0x1E, 0x3C, 0x78, 0xF0, 0xFD,     /* 328 to 335 */
 0xE7, 0xD3, 0xBB, 0x6B, 0xD6, 0xB1, 0x7F, 0xFE,     /* 336 to 343 */
 0xE1, 0xDF, 0xA3, 0x5B, 0xB6, 0x71, 0xE2, 0xD9,     /* 344 to 351 */
 0xAF, 0x43, 0x86, 0x11, 0x22, 0x44, 0x88, 0x0D,     /* 352 to 359 */
 0x1A, 0x34, 0x68, 0xD0, 0xBD, 0x67, 0xCE, 0x81,     /* 360 to 367 */
 0x1F, 0x3E, 0x7C, 0xF8, 0xED, 0xC7, 0x93, 0x3B,     /* 368 to 375 */
```

```
0x76, 0xEC, 0xC5, 0x97, 0x33, 0x66, 0xCC, 0x85,     /* 376 to 383 */
0x17, 0x2E, 0x5C, 0xB8, 0x6D, 0xDA, 0xA9, 0x4F,     /* 384 to 391 */
0x9E, 0x21, 0x42, 0x84, 0x15, 0x2A, 0x54, 0xA8,     /* 392 to 399 */
0x4D, 0x9A, 0x29, 0x52, 0xA4, 0x55, 0xAA, 0x49,     /* 400 to 407 */
0x92, 0x39, 0x72, 0xE4, 0xD5, 0xB7, 0x73, 0xE6,     /* 408 to 415 */
0xD1, 0xBF, 0x63, 0xC6, 0x91, 0x3F, 0x7E, 0xFC,     /* 416 to 423 */
0xE5, 0xD7, 0xB3, 0x7B, 0xF6, 0xF1, 0xFF, 0xE3,     /* 424 to 431 */
0xDB, 0xAB, 0x4B, 0x96, 0x31, 0x62, 0xC4, 0x95,     /* 432 to 439 */
0x37, 0x6E, 0xDC, 0xA5, 0x57, 0xAE, 0x41, 0x82,     /* 440 to 447 */
0x19, 0x32, 0x64, 0xC8, 0x8D, 0x07, 0x0E, 0x1C,     /* 448 to 455 */
0x38, 0x70, 0xE0, 0xDD, 0xA7, 0x53, 0xA6, 0x51,     /* 456 to 463 */
0xA2, 0x59, 0xB2, 0x79, 0xF2, 0xF9, 0xEF, 0xC3,     /* 464 to 471 */
0x9B, 0x2B, 0x56, 0xAC, 0x45, 0x8A, 0x09, 0x12,     /* 472 to 479 */
0x24, 0x48, 0x90, 0x3D, 0x7A, 0xF4, 0xF5, 0xF7,     /* 480 to 487 */
0xF3, 0xFB, 0xEB, 0xCB, 0x8B, 0x0B, 0x16, 0x2C,     /* 488 to 495 */
0x58, 0xB0, 0x7D, 0xFA, 0xE9, 0xCF, 0x83, 0x1B,     /* 496 to 503 */
0x36, 0x6C, 0xD8, 0xAD, 0x47, 0x8E, 0x00, 0x00,     /* 504 to 511 */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
```

```
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,     /* zeros for fast multiply */
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00      /* zeros for fast
multiply */
 };
```

Table 5. Galois Exponentiation Helper Function

These tables can then be used to compute various arithmetic opeations as described below.

[0046] Addition and subtraction in Galois Field (2^8) are the same operation, and are performed as follows:

$$a \oplus b = a \wedge b$$
$$a - b = a \wedge b$$

where ^ is the logical *exclusive or (XOR)* operator. As in ordinary arithmetic, $a \oplus 0 = 0 \oplus a = a$.

[0047] Multiplication ($a \otimes b$) in Galois Field (2^8) is performed as follows:

$$a \otimes b = \text{gexp}[\text{glog}[a] + \text{glog}[b]]$$

where + is the ordinary *addition* operator. As in ordinary arithmetic, $a \otimes 0 = 0 \otimes a = 0$. Also, $a \otimes 1 = 1 \otimes a = a$.

[0048] Division ($a \div b$) in Galois Field (2^8) is performed as follows:

$$a \div b = \text{gexp}[\text{glog}[a] - \text{glog}[b] + 255]$$

where + is the ordinary *addition* operator and - is the ordinary *subtraction* operator. As in ordinary arithmetic, $a \div 1 = a$. As is also usual, b must be non-zero.

[0049] The power function (a^b) in Galois Field (2^8) is performed as follows:

$$a^b = \text{gexp}[(\text{glog[a]} * \text{b}) \,\%\, \text{0xFF}]$$

where + is the ordinary *addition* operator, - is the ordinary *subtraction* operator, * is the ordinary *multiplication* operator, and % is the ordinary *mod* function. As in

ordinary arithmetic, $a^0 = 1$ for non-zero a. In the RS256 reference code, 0 is never raised to a power.

[0050] As described above, the LPR group size (in packets) can be set to achieve a desired protection period (in milliseconds),. If the video codec does not use the entire channel bit rate, then the LPR group will result in a longer latency period. For example, a video channel that normally runs at 1.5 Mbps may drop to 750 Kbps in periods of light motion. This could result in a protection period twice as long as desired. Such a situation can result in the selected LPR mode continuing to meet the predetermined MTBF, which will, in this example, be exceeded by a factor of 2.

[0051] In some cases, this extended protection period can be allowed. However, this can result in a longer latency when packets are lost. Therefore, in some cases, it may be desirable for the transmitter to adjust its LPR mode as the actual video rate varies. Three such adjustments are described below, although other adjustments are also possible.

[0052] One adjustment that can be made is to the protection group size. In some embodiments, this setting can be changed at every protection group boundary. Therefore, a transmitter can monitor the LPR output packet rate and dynamically adapt the number of packets per group. When this adjustment is employed, LPR protection overhead will tend to rise as the data rate falls. However, because the data rate in such cases will typically be below the normal limit the altered LPR mode will still generally remain below a predetermined congestion ceiling.

[0053] Another adjustment that can be made is for the transmitter to change its output packet size upon transmission of an initial data packet. In general the transmitter should not change the output packet size when transmitting follow-on packets, as the number of follow-on packets will typically be signaled in the initial data packet. Therefore, a transmitter can monitor the LPR output packet rate and the output data rate and dynamically adapt the packet size to maintain the number of packets per protection period.

[0054] Another adjustment that can be made is for the transmitter to send empty data packets to complete a partial recovery set. An empty data packet can

include an initial data packet header as described above. Other information that would normally be in the packet, such as sequence number payload type, number of follow-on packets, etc., can all be set to zero. Because the empty data packets are part of the protection group, the recovery packet encoding described above can include these packets, which can also be recovered by the decoding process. Also, empty data packets can be discarded by the LPR decoder (discussed below) to prevent empty packets in the output media packet stream.

[0055] Fill packets can also be used to maintain a predetermined data rate. These fill packets can be discarded by the receiver. An example fill packet is illustrated in Fig. 9. The fill packet can include an extended type field 901 of six bits having a value of 1 and a type field 902 of two bits having a value of 11. The fill packet can be an RTP packet using the LPR payload type for the media stream. The timestamp value can be the same as the value in the previously sent RTP packet. The sequence number can be incremented from the value in the previously sent packet. The payload can end with the Fill header shown above. The payload length of the field can be any length within a maximum specified packet size.

[0056] An example of an LPR decoder is illustrated in Fig. 10. As with the LPR encoder discussed above, variations on this embodiment are possible. Operation of the LPR decoder can be basically an inverse of the LPR encoder operation described above. Specifically, LPR packets can be recovered by LPR recovery module 1001. These recovered LPR packets can be passed to the RTP reordering buffer 1002, which can, as in conventional systems, reorder the packets into their received order. LPR regenerator 1003 can then recreate any lost packets, relying on information received from the recovered LPR packets (from LPR recovery block 1001) and RTCP information received from RTCP module 1006. These regenerated blocks can then be decrypted by decryption block 1004 (if encryption was used at the source), and finally the decrypted blocks can be processed by video decoder 1005 (or other type of media decoder, as appropriate).

[0057] LPR recovery module 1001 and LPR renenerator 1003 can be the only non-standard elements as compared to conventional systems, and these two modules can operate as pass-throughs when LPR is not being used.

[0058] LPR recovery module 1001 can process all LPR RTP packets. In the given example, all received and regenerated packets (data and recovery) can be directly passed through to the RTP reordering buffer. The overhead data in each data packet can allow these packets to be processed by LPR recovery module 1001 in any order. LPR recovery module 1001 can also provide LPR RTP receiver report information to RTCP module 1006. This information can be based on what is received (*e.g.*, any packets that are not recovered can be identified as lost in the receiver report).

[0059] If there are missing data packets in the recovery set that can be recovered, the LPR recovery module can recover these packets as soon as sufficient recovery packets have been received. As discussed above, if k data packets are missing, k recovery packets must be received to recover the data. This recovery process can involve first recovering the marker bit, the data packet length, the initial packet flag, and the original timestamp. The data packet length can be used to recover the original data packet payload. "Fixed" components of the LPR RTP header (*e.g.*, SSRC, CSRC, LPR payload type, etc.) can be taken from one of the recovery packets. Variable portions of the LPR RTP header (*e.g.*, marker bit, timestamp, and sequence number) can be set to the values recovered from the recovery header.

[0060] The sequence number for LPR data packet i can be computed according to the following equation:

$$S_i = S_r - (d + r - i)$$

where: i is the index recovered data packet, d is the number of data packets in the recovery set, r is a received recovery packet index, and S_i is the sequence number for packet i.

[0061] While receiving data packets, partial recovery packets can be built up by recovery module 1001, using the RS256 generation function described above. As corresponding recovery packets for the set are received, these packets can be exclusive-or'ed (*i.e.*, XOR'd) with the partial recovery packet. The resulting residual packets will hold the contribution of any lost data packets to each recovery packet. In the exemplary system, each residual packet will be a

linear combination of each of the missing data packets. Thus, there will be k simultaneous equations (one for each of the k residual packets), each with k unknowns (one for each of the missing k data packets).

[0062] As would be understood by one skilled in the art, these equations can be represented as a k x k matrix, which can then be solved numerically, *e.g.*, by Gaussian elimination. Because such operations are well-known, the details are not reproduced herein. Multiplying the resulting inverted matrix by the residual data can then recover the missing data packets.

[0063] To illustrate the above recovery process, the recovery scenario in Figure 2 can be performed. For clarity ordinary arithmetic operations are used rather than the actual Galois Field (2^8) operations. The data in the recovery packets can generated using the following formula (derived from the RS256 generation function and tables described above:

$$R_1 = D_1 + D_2 + D_3 + D_4 + D_5 + D_6$$
$$R_2 = 2D_1 + 4D_2 + 6D_3 + 9D_4 + 13D_5 + 14D_6$$

[0064] The partial recovery packets can be computed by the decoder on the packets that are received. As noted in the example above, packets 3 and 4 (108, 110, Fig. 3) are not received. This results in:

$$P_1 = D_1 + D_2 + D_5 + D_6$$
$$P_2 = 2D_1 + 4D_2 + 13D_5 + 14D_6$$

Residuals can be computed by subtracting the partial recovery packets from the received recovery packets:

$$r_1 = R_1 - P_1 = D_3 + D_4$$
$$r_2 = R_2 - P_2 = 6D_3 + 9D_4$$

The generating matrix for the residuals is therefore:

$$\begin{matrix} 1 & 1 \\ 6 & 9 \end{matrix}$$

Inverting this matrix via Gaussian elimination results in the following:

$$\begin{matrix} 3 & -1/3 \\ -2 & 1/3 \end{matrix}$$

Therefore, the information in the missing data packets (*i.e.*, data packets 3 and 4) can be recovered using the formulas:

$$D_3 = 3r_1 - r_2/3$$
$$D_4 = -2r_1 + r_2/3$$

[0065] When LPR is in use, RTP reordering buffer 1002 can be integrated with the LPR decoder. This can permit a minimal delay implementation, because in some situations out-of-order packets can be regenerated by LPR before they actually arrive. Additionally, in some embodiments it may be desirable to acquire the first *d* packets of the recovery set, running the Reed-Solomon decoder only if some data packets were lost. This design may not produce minimal delay, but can be more computationally efficient.

[0066] LPR regenerator module 1003 can recreate the original RTP stream from the data packets as was described above. If some data packets for an RTP packet are missing, then that RTP packet is omitted from the output stream. Because the LPR stream has been re-ordered, the initial packet from each packet set should be received first. If a follow-on packet is encountered first, then the initial packet has been lost, and the follow-on packet can be discarded.

[0067] The output RTP packet uses the RTP header information of the initial data packet. The payload type, sequence number, and timestamp are replaced with the values in the initial data header. The payload is aggregated from the initial packet and any follow-on data packets with the LPR headers removed. If any follow-on packets are missing the output RTP packet can be suppressed. It should be noted that the number of follow-on packets can be known because it can be signaled in the initial data packet header.

[0068] Lost packet recovery techniques, like those described above, can be combined with congestion avoidance algorithms to further enhance operation. An example congestion avoidance algorithm is referenced above and is referred to as Dynamic Bandwidth Allocation ("DBA"). DBA can attempt to avoid congestion by downspeeding the bitrate based on the percent packet loss reported by the receiver's RTP stack (1006, Fig. 10). The receiver RTP stack generates packet loss statistics for the receiver channel during regular time periods (*e.g.*, every 200 ms). This can allow fast adaptation to a dynamically changing pipe while allowing for minimal LPR configurations with low MTBF requirements (which, in turn can allow for lower overhead).

[0069] As illustrated in Fig. 12, the control loop for DBA can be implemented as a state machine with the following states: max rate state 1101, downspeeding state 1102, upspeed wait state 1103, upspeeding state 1104, and constant loss state 1105. The various state transitions can be understood as follows. Suppose the system is in the max rate state 1101, meaning that the maximum data rate is being used. A packet loss above a predetermined threshold can cause state transition 1106 to downspeeding state 1102.

[0070] While in the downspeeding state, the system can decrease the speed and wait to see whether packet loss above the predetermined threshold continues to be experienced. If there is no more packet loss above the threshold, the system can follow transition 1108 to the upspeed wait state. If there continues to be packet loss above the threshold, and the maximum number of downspeed attempts have not been used (block 1107), the system can continue in the downspeeding state 1102 and can thus continue decreasing the bitrate. If the maximum number of downspeeding attempts has been used, the system can transition to constant loss state 1105.

[0071] While in the upspeed wait state, additional packet loss above the predetermined threshold can cause the system to follow transition 1109 to the downspeeding state. Alternatively, if, after a predetermined period of time, there is no packet loss above the threshold, the system can follow transition 1110 to the upspeeding state 1104.

[0072] In the upspeeding state 1104, the system can increase the bitrate and wait for additional packet loss above the predetermined threshold. If there is no packet loss, the system can determine whether the bitrate is less than the maximum bitrate. If so, the system can follow state transition 1111 to the maximum rate state 1101. If not, the system can continue in the upspeed state 1104. If packet loss above the predetermined threshold is experienced while in upspeed state 1104, the system can determine whether a predetermined maximum number of upspeed attempts have been made (block 1112). If not, the system can follow transition 1113 to the upspeed wait state. If the maximum number of upspeed attempts has been used, the system can follow transition 1114 to the max rate state 1101.

[0073] In each of the states described above, different packet loss thresholds may be employed. Additionally, some or all state transitions may have a delay associated with them. Upspeeds can be done in 15% step increases of the bit rate. If the pipe is fully utilized and no packet loss is reported, a delay can be incurred which can be equal to X timer cycles (200 ms * X) before the next upspeed will occur. By default the value of X can be set to 2, resulting in a 400 ms delay. Any packet loss that is not a sequential packet loss report and not an echo report can cause a state transition.

[0074] Figure 12 shows dynamically changing bandwidth availability and the DBA adaptation. Figure 12 also shows the state transitions, echo delays, along with three generalized loss scenarios: Burst Loss, Constrained Pipe, and Constant Loss. LPR protection use is designated in Figure 12 by the shaded area during the downspeeding state, the upspeed wait state, and the upspeeding state. The upspeed attempts can use the LPR protection overhead to test for the congestion ceiling, *i.e.*, only "redundant" data may be used to probe whether network congestion conditions would permit a higher bitrate. If this test of the congestion ceiling fails, LPR protects any loss caused by the packet loss incurred because only "redundant" data has been used. This allows probing of the congestion ceiling with substantially reduced risk of video disruption.

[0075] The measured congestion ceiling can be applied to the data flow in its entirety. For example, when multiple video streams are being sent, the congestion ceiling can apply to the aggregated bit rate. Forward erasure protection can be applied independently to each video stream. However, the same level of protection can be applied to all streams. The system can also be adapted to jointly protect the video streams with common forward erasure protection packets or to apply different levels of protection to each video stream.

[0076] Forward erasure correction can also be applied to just a part of the video stream. For instance, in cases where layered video encoding is employed, the forward erasure protection could be applied to the base layer but not the enhancement layers. It could also be adapted to protect the various layers to different degrees (so called "unequal protection"). In these configurations, the congestion avoidance applies to the entire data flow.

[0077] Example LPR probes for congestion ceiling management are given in Table 6. These probes can be used to test the possibility of raising the data rate by a predetermined value (*e.g.*, 10%) and can be activated for short periods of time (*e.g.*, 800 ms).

Video Rate (k\bs)	Data Packets Per Group	Packet Payload Size (bytes)	Recovery Packets Per Group	Protection Period (ms)	MTBF (ms)
64	13	87	3	141	2067
128	12	133	3	100	1795
256	12	266	3	100	1795
384	12	400	3	100	1800
512	12	533	3	100	1799
768	12	800	3	100	1800
1024	16	800	3	100	870
2048	16	800	4	50	1158
3072	24	800	6	50	1936
4096	32	800	7	50	1365
5120	32	1000	7	50	1365
6144	39	1000	8	51	1235

Table 6 Probe for a 10% Rate Up-speed

[0078] The techniques described herein can be receiver-driven, in that the receiver feeds back the congestion ceiling and erasure protection level to the sender. Alternatively, they can also be used in a transmitter-driven framework, or even in a hybrid framework where the control loop is distributed between sender and receiver. With some adaptation, it is also suitable for use with multicast streams. The techniques described also can be combined with other techniques such as packet retransmission, periodic picture refresh, and video error concealment.

[0079] As described above, the combination of forward erasure correction with congestion avoidance offers unique benefits that neither mechanism can achieve on its own. One such benefit is that congestion ceiling can be periodically measured by experimentally increasing the amount of forward erasure protection packets in the bitstream and observing the effect on the delivered data flow. These protection packets increase the overall data rate to a desired "probe" level while simultaneously protecting the media from packet loss (in case the level exceeds the congestion ceiling). A second benefit is that this method of dynamically measuring the congestion ceiling can result in a much faster

determination of the allowable bandwidth than that provided by the passive methods normally employed in congestion avoidance algorithms. A third benefit is that the protected media flow can be delivered with less delay and lower protection overhead when the entire flow (media and protection) is constrained to stay below the dynamically measured congestion ceiling.

[0080] An additional benefit of the techniques described herein arises in that as the video data rate increases, the system can become more and more efficient at protecting against packet loss. Therefore, as bitrates increase, such as with the use of high definition video, the techniques described herein become both more useful and more efficient.

[0081] The integration of congestion avoidance with forward erasure correction offers several unique advantages. For example, congestion avoidance brings several unique advantages to forward erasure correction. As a data link becomes congested, the transmission delay for media along that link increases. Using forward erasure correction alone may result in an acceptable packet loss rate, but will not reduce queueing delays. Integrating congestion avoidance into the control loop results can achieve a desired quality of service (QoS) with much lower latency. Additionally, the efficiency of forward erasure protection is less for low-latency applications than it is for high-latency applications. If the source of the data loss is simple congestion, reducing the bit rate can be more efficient than adding additional forward erasure protection packets to the data flow.

[0082] Conversely, forward erasure correction can also add considerable value to congestion avoidance alone. Congestion avoidance techniques typically reduce the rate of the data flow more aggressively then they raise it. For example, the congestion mechanism for TCP reduces the data rate by a multiplicative factor, but raises it much more slowly by applying an additive factor. The reason for this is that increasing the data flow rate always risks creating congestion, and therefore creating packet loss. When forward erasure protection is integrated with the congestion avoidance, this risk is greatly reduced. Thus, as noted above, the data rate can be increased by simply increasing the density of forward erasure protection packets.

[0083] Additionally, congestion avoidance techniques also are designed to estimate the link speed. Cross-congestion from other data flows can frequently result in packet loss. By the time the cross-congestion is observed, packet loss has usually already occurred. Including forward erasure protection allows packets lost due to such cross-congestion to be recovered.

[0084] The techniques disclosed herein can be employed in a wide variety of systems, including any systems for the transmission of multimedia data, such as video data, audio data, still image data, and other types of data. Such techniques can be employed in typical general purpose computer systems, such as desktop computers, notebook computers, handheld computers, servers, and the like. Alternatively, these techniques can be employed in various appliance-type systems such as conference room videoconferencing systems, desktop videoconferencing systems, and the like. The techniques described can also be employed in infrastructure type videoconferencing devices such as multi-point control units (MCUs), bridges, media servers, etc.

[0085] The methods described herein can be coded into one or more computer programs and stored on a computer-readable media, such as a compact disk, a tape, stored in a volatile or non-volatile memory, etc. Accordingly, instructions stored on a program storage device can be used to cause a programmable control device (*e.g.*, computer or conferencing unit) to perform the techniques disclosed herein. Although the disclosed communcation system has been described as providing bi-directional communication between a near end and a far end, it will be appreciated that the teachings of the present disclosure are applicable to systems that provide one-way transmission.

[0086] The foregoing description of preferred and other embodiments is not intended to limit or restrict the scope or applicability of the inventive concepts conceived of by the Applicant. In exchange for disclosing the inventive concepts contained herein, the Applicant desires all patent rights afforded by the appended claims. Therefore, it is intended that the appended claims include all modifications and alterations to the full extent that they come within the scope of the following claims or the equivalents thereof.

3. Brief Description of the Drawings

[0087] Figure 1 illustrates a lost packet recovery technique that may be used to protect against packet loss.

[0088] Figure 2 illustrates a lost packet recovery technique in which certain packets are lost and the data is reconstructed.

[0089] Figure 3 illustrates a lost packet recovery technique in which certain packets are lost and the data is reconstructed.

[0090] Figure 4 illustrates a lost packet recovery encoder.

[0091] Figures 5–9 illustrate packet headers that can be used in conjunction with the lost packet recovery techniques and encoders of Figs. 1–4.

[0092] Figure 10 illustrates a lost packet recovery decoder.

[0093] Figure 11 illustrates a state machine for a congestion avoidance algorithm that can be used in conjunction with the lost packet recovery techniques, encoders, and decoders of Figs. 1–10.

[0094] Figure 12 illustrates a data rate for an encoder employing lost packet recovery and congestion avoidance using techniques, encoders, decoders and algorithms according to Figs. 1–10.

1. Claims

WHAT IS CLAIMED IS:

1. A method of preventing disruption of a media stream transmitted across a lossy network, the method comprising:
   applying a lost packet recovery algorithm to the media stream, wherein the lost packet recovery algorithm inserts redundant information into a transmitted data stream including the media stream to permit reconstruction of one or more lost packets; and
   applying a congestion avoidance algorithm to the transmitted data stream, wherein the congestion avoidance algorithm includes temporarily increasing a data rate of the transmitted data stream by increasing the amount of redundant information inserted into the transmitted data stream to determine whether the network can support a higher data rate.

2. The method of claim 1 wherein the redundant information comprises one or more recovery packets.

3. The method of claim 2 wherein the number of recovery packets depends on the channel loss rate determined from RTCP receiver reports.

4. The method of claim 1 further comprising:
   reducing the aggregate bit rate as necessary to avoid network congestion.

5. The method of claim 1 wherein the lost packet recovery algorithm includes forward erasure correction coding.

6. The method of claim 1 wherein the lost packet recovery algorithm includes Reed-Solomon encoding.

7. The method of claim 1 wherein the media stream comprises video data.

8. A media encoding apparatus for transmitting media data over a network, the media encoding apparatus comprising:

a media encoder;

an RTP sender connected to receive output from the media encoder and package the output of the media encoder according to a real-time transmission protocol as a plurality of source packets;

an RTCP module connected to receive channel information from the network;

an LPR mode decision module, wherein the LPR mode decision module is:

communicatively coupled to receive the channel information from the RTCP module,

adapted to determine LPR protection parameters to be used from the channel information, and

communicatively coupled to provide the LPR protection parameters to the LPR packetizer;

an LPR packetizer connected to receive output from the RTP sender and repacketize the output of the RTP sender as a plurality of LPR data packets using parameters received from the LPR mode decision module; and

an LPR recovery packet generator, wherein the LPR recovery packet generator is:

coupled to the LPR packetizer and the media encoder,

adapted to generate LPR recovery packets based on information received from the LPR packetizer and the video encoder, and

adapted to transmit the LPR data packets and the LPR recovery packets to a receiver.

9. The media encoding apparatus of claim 8 wherein the media encoder is a video encoder.

10. The media encoding apparatus of claim 8 wherein the LPR mode decision module determines LPR protection parameters by:
determining a protection period;
determining a total number of packets per protection period;
determining a number of recovery packets per protection period.

11. The media encoding apparatus of claim 8 wherein each LPR data packet contains information from only one source packet.

12. The media encoding apparatus of claim 8 wherein a payload of the recovery packet is encoded using Reed-Solomon encoding.

13. A method of determining LPR protection parameters in a media conference conducted over a packet-based network, the method comprising:
determining a protection period;
determining a total number of packets per protection period; and
determining a number of recovery packets per protection period.

14. The method of claim 13 wherein the protection period is determined according to a bitrate of the media conference.

15. The method of claim 3 further comprising adjusting LPR protection parameters as a function of the media rate.

16. The method of claim 15 wherein adjusting LPR protection parameters as a function of the media rate comprises dynamically adapting the number of packets per group.

17. The method of claim 15 wherein adjusting LPR protection parameters as a function of the media rate comprises dynamically adapting the packet size.

18. The method of claim 15 wherein adjusting LPR protection parameters as a function of the media rate comprises sending empty data packets.

19. A media decoding apparatus for receiving media data from a transmitter over a network, the media decoding apparatus comprising:

an LPR recovery module coupled to receive LPR packets, including LPR data packets and LPR recovery packets, from the network;

an LPR regenerator adapted to receive the LPR data packets and regenerate any missing data packets ;

an RTCP module communicatively coupled to the LPR regenerator to receive a report of lost packets from the LPR regenerator and transmit lost packet information to the transmitter; and

a media decoder coupled to the LPR regenerator and adapted to receive the LPR data packets and the regenerated data packets and decode media information contained therein.

20. The media decoding apparatus of claim 19 wherein the media decoder is a video decoder.

21. The media decoding apparatus of claim 19 wherein the LPR recovery module and the LPR regenerator operate as pass throughs when LPR algorithms are not in use.

1. Abstract

Devices and techniques for overcoming lost packets and avoiding congestion when conducting media conferences over packet switched networks are described herein. To avoid the problem of lost packets, redundant information is inserted into the media stream that permits the receiver to reconstruct any lost packets from the redundant information. Congestion avoidance techniques include adjusting the bitrate of the media stream to find the highest bitrate that can be supported without packet loss due to congestion. When increasing the bitrate to a higher rate, the additional bits can come from the redundant information used for lost packet recovery so that any lost packets caused by network congestion will not adversely affect the bitstream.

【図１】

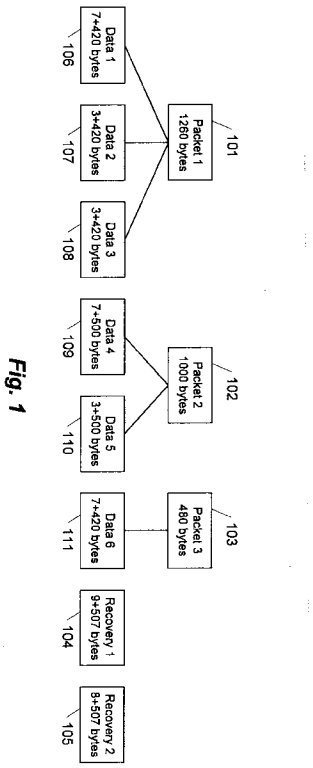
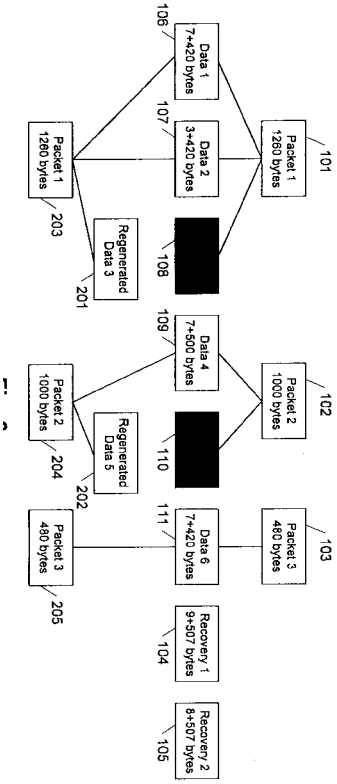


Fig. 1

【図２】



【図３】

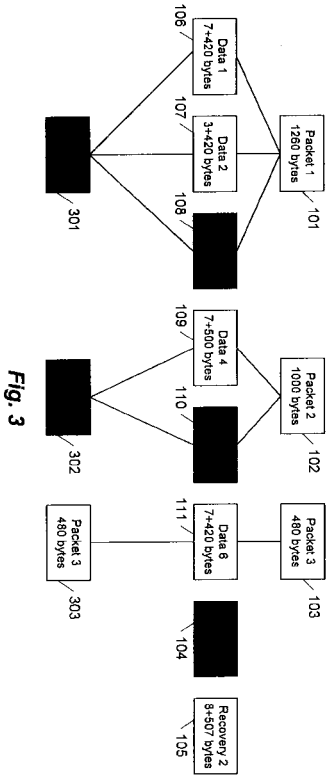


Fig. 3

【図４】

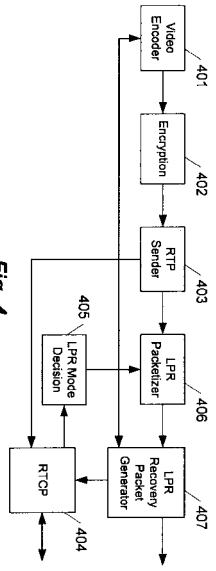


Fig. 4

【図５】

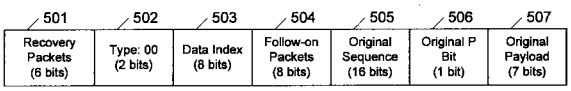


Fig. 5

【図６】

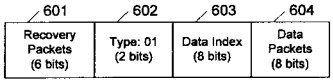


Fig. 6

【図７】

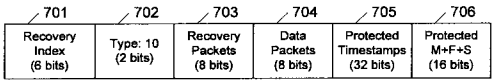


Fig. 7

【図８】

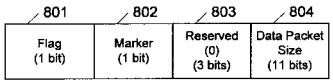


Fig. 8

【図９】

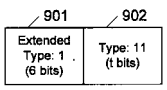


Fig. 9

【図１０】

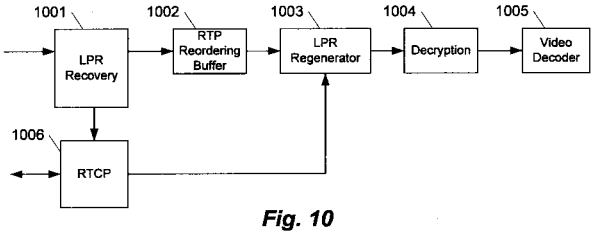


Fig. 10

【図１１】

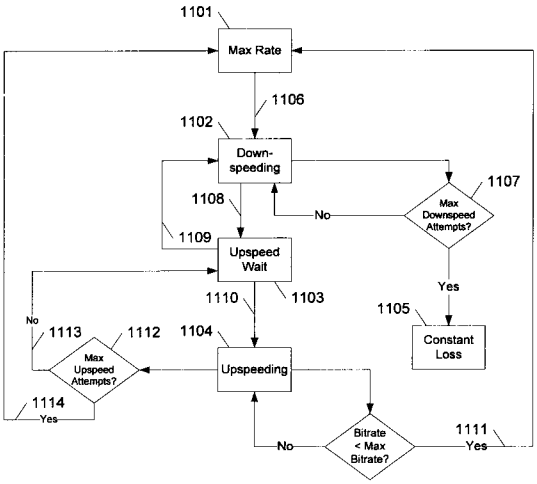


Fig. 11

【図１２】

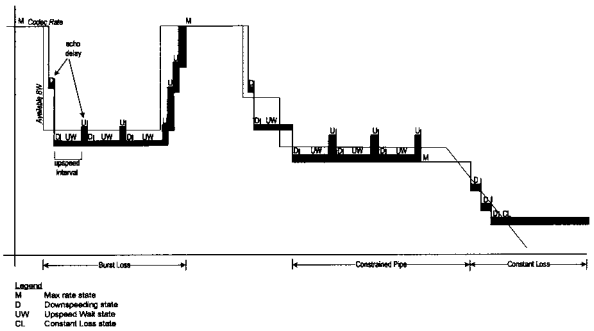


Fig. 12