第X部

信頼性を有するマルチキャスト技術

第 10 部
信頼性を有するマルチキャスト技術

第 1 章　誤り訂正符号を用いたパケット回復の性能測定

1.1 はじめに

インターネット上でマルチメディアデータをストリーム転送（リアルタイム転送）する場合に、パケット損失が障害となる。データの損失はマルチメディアデータにおいてもかなり悪い影響を与える。マルチメディアの符号化の圧縮率が高く情報密度が高くなるほど、データ欠損の被害が大きくなり、たとえば MPEG の I フレームデータの欠損はフレーム間の参照の影響により後続の複数フレームに影響を与えることがあるし、ヘッダー部分の欠損はそのヘッダーの管理下だけではなく、ヘッダー内のデータ長情報が失われると次のフィールドの開始位置が分からないという同期問題が発生し、次の GOP で同期信号により同期が取られるまでの区間のデータが失われることにもなる。

ユニキャストにおいては、パケット損失の回復手段は大別して

- 再転送による回復
- 誤り訂正符号を用いた回復

の 2 つが考えられている。再転送による方法（ARQ: Automatic Repeat reQuest）は、ユニキャストでは TCP におけるデータ信頼性確保の手段として用いられ、インターネット上のほとんどのトラフィックはこれによって損失を回復されている。他方、誤り訂正符号による損失パケットの回復（FEC: Forward Error Correction、前方誤り訂正）は、送信側で送信パケットに加えて冗長データからなるパケットを送り、受信側では一部のパケットが失われても回復できると言う手法であり、近年幾つかのやり方が提案されている。

マルチメディアのストリーム転送を考慮した場合、一般に遅延が一時的に伸びると受信バッファが枯渇する問題がある。この遅延変動は、TCP のようなエンド・エンドでの再送制御によって損失回復した場合には、再送時にパケット往復時間分の遅延が付け加わるため、距離が遠いと大きくなる。仮に大陸間ケーブルや静止衛星などを経由する場合、パケット往復時間が 0.5〜1 秒程度になることが考えられるので、損失がない場合とある場合の遅延の変動がその程度になり、バッファリングにより吸収することが難しくなる。誤り訂正符号によるパケット回復は、この差が発生しないため好ましい。

また、高信頼マルチキャスト（RM、Reliable Multicast）のように、マルチキャストを対象として損失パケットを回復しなければならないとき、再転送による ARQ では、応答パケットが送信端に集中する応答爆発が発生する。これを避ける手法もさまざま提案されているが、誤り訂正符号を用いて回復する FEC に比較すると複雑になりがちである。これらの観点から、誤り訂正符号による損失パケットの回復 FEC は有用であると考えられる。

他方、FEC は送信側で追加する誤り訂正パケット（冗長パケット）の量は、送信側で予め決めてしまうのが普通である。たとえばネットワークの状況が変化してパケットの損失率が増加し、誤り訂正符号の回復限界を超えると回復できなくなる。FEC はその観点から損失率変動に対して弱いと言える。

更に、一般に用いられる誤り訂正符号は、一定数パケットからなるブロックを単位として、その中で発生するパケット損失をその中の冗長パケットで回復する。つまり、ブロック内で数えたときに損失率が冗長度を下回っていなければ回復できない。インターネット上ではパケット損失がバースト的に発生すると言われており、その場合には、パケット損失率が長時間平均で回復限界以下であっても、バースト的に発生する期間では回復限界を超えて回復不能となる場合が出てくる。

この研究では、インターネット上のパケット損失の発生パターンを測定し、我々が考えるブロック型の FEC を適用した場合の回復可能な率、不能な率を判定し、回復不能率が一定以下となるように冗長度を設定できる技術を開発することを目的とする。

1.2 FEC によるパケット損失の回復

FEC は、誤り訂正符号の理論を応用して、送信側で冗長符号を付加し、それによって受信側でパケット損失を回復する手段である。誤り訂正符号は、送信するデータから冗長ビットを計算し、冗長ビットを用いることで誤りを訂正する。パケット損失は、消失した部分がなんらかの誤りを生じて正しい値かわからなくなったものとみなすことができるので、誤り訂正符号を用いて復元することができる。

パケット損失は、パケット内のビット列がすべて消失した状態であり、ビット列として見ると非常に長い区間の連続ビット誤り（バースト誤り）と同じ状態なので、そのままでは損失を回復するために大きな計算量を必要とする。これを防ぐため、インターリービングと呼ばれる手法によって、誤りを分散させることが行われている。インターリービングとは、ビット列を誤り訂正符号で処理するブロックごとに分割するとき、複数のブロックのビットを 1 ビット（もしくは複数ビット）ずつ集め、図 1.1 に示すように、新たな処理ブロックを形成させることによって、誤りを分散させる手法である。こうすることで、バースト誤りのような連続ビット誤りが生じても、バースト誤りを生じていない他のブロックのビットによって誤りの訂正を行うことができる。

インターリービングを用いた場合、FEC によるパケットレベルでの損失回復の性能は、複数のパケットを単位とする冗長符号処理ブロック（複数のパケットの中身を分散させて新たに作ったブロック）の中

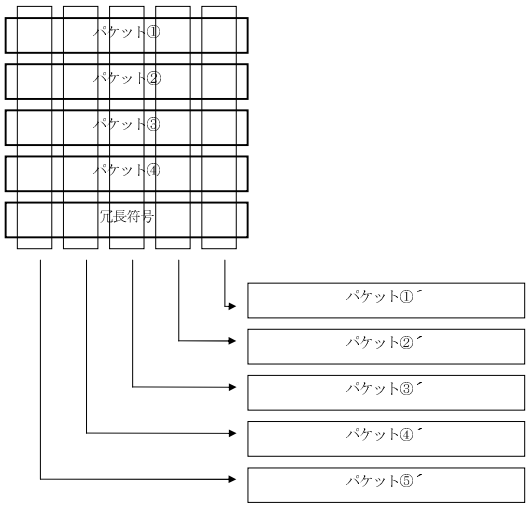


図 1.1.　インターリービングのメカニズム

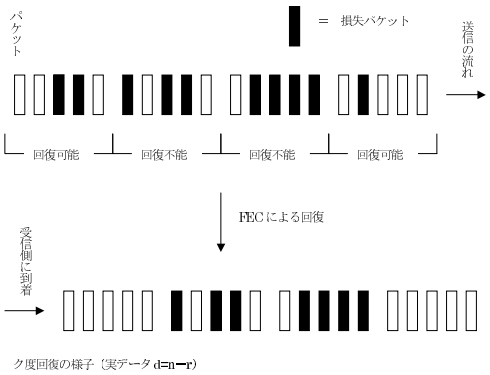


図 1.2.　インターリービング下でのパケット回復

で、処理ブロック長 n の中からブロック内に付加された冗長パケット数 r までの損失に対して回復可能である。通常の誤り訂正の場合は冗長数 r に対して $r/2$ までの回復のみ可能であるが、ここでは損失パケットの位置が分かっている「消失誤り」となり、r までの回復が可能である。全体のブロックで見ると、n パケットからなる処理区間（ブロック）の中で、パケット損失数が冗長パケット数 r を超えなければ、そのブロックに含まれるもとのデータを回復でき、それを超えれば回復できない。

例えば、図 1.2 のような場合、送信パケットを 3 個ずつのブロックに分けたとすると、それぞれに対して 2 個の冗長度（冗長パケット）を加えて 5 個ずつのブロックにする。この場合、受信側では 2 個以下のパケット損失であれば回復が可能なので、図のような損失が起こった場合、回復の様子は損失が 1、2 個のブロックは無事に回復され、3、4 個のブロックは回復されない。

FEC は、リアルタイム転送時に生じるパケット損失を回復するのに効果的な方法である。しかし、パケット損失に対して FEC を用いるにはインターリービングによって誤りを分散させる必要がある。よって、インターリービングの処理ブロック内で回復処理を行うが、回復が可能となるためには、ブロック内での損失数に上限がある。分散させたブロックが上限を超える損失を起こしていた場合、パケット損失の長時間平均値が低くてもそのブロックは回復できない。従って、パケット損失の発生に時間的偏りがあると、回復できないブロックが出てしまう。

輻輳時のパケット損失は経路上のルータの一時的なバッファあふれによるもので、このあふれはある期間の中に集中すると言われている [151, 139]。こ

れがパケット損失発生のバースト性であるが、処理ブロック内のパケット損失が FEC で設定した冗長度を超えた場合、そのブロック内の損失が回復できない。しかも、バースト状態は一定期間続く可能性が高いので、その期間内に含まれる処理ブロックでの損失率が著しく増加する。

一方で、処理ブロック長を n、冗長度を r、実データのパケット数を d とすると、$d = n - r$ であるから、実際の損失パケット数が冗長数 r より小さい場合、必要以上の冗長度を持たせていることになり、回線の帯域資源の使用効率を必要以上に下げてしまう。言い換えると、十分な回復性能を実現しつつ回線使用効率を低下させないためには、冗長度が必要最小限であることが望まれる。

上記のようにインターリーブ処理ブロック内での損失発生数が、時間によって大きく変動する中で、必要最小限の冗長度を与えるためには、ネットワーク上でのパケット損失がどのような傾向を有しているかを分析し、その傾向から、処理ブロック長や冗長パケットの数を決定することを試みる必要がある。

1.3 パケット損失の実測

ストリーム転送時の FEC によるパケット損失回復を実用化するためには、最も効率良く回復を行うための冗長度を決定しなければならない。そのためには、処理ブロック内のパケット損失の発生傾向を正確に把握する必要がある。既存の研究 [151, 139] から、インターネット上のトラフィックモデルとして二項分布やポアソン分布などランダム発生の確率分布が適用できず、バースト的な傾向が見られることが指摘されている。パケット損失もトラフィック量に密接に従うので、バースト的に発生する。この発生状況を、実回線上の測定データから分析し、最適な冗長度や処理ブロック長を決める手がかりにすることが、本研究の狙いである。

実測値との比較対象として、ランダム発生の確率分布を考えておく。もし、パケット損失が常に等確率で発生すると仮定できれば、パケット 1 個を送信した際にパケット損失を起こす確率を p とすると、n 個のパケットを送信してそのうちの k 個が失われる確率の確率分布は、確率 p のベルヌーイ試行を n 回繰り返して、そのうちの k 回失われる確率の確率分布であり、これは二項分布となる。従って、パケット損失が確率 p でランダム発生するとすれば、インターリーブブロック長 n、冗長パケット数 r の時、二項分布の $k < r$ の部分が回復され、$k > r$ の部分が回復できない。

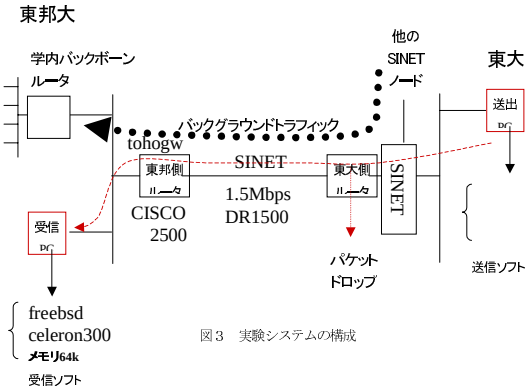


図 3　実験システムの構成

図 1.3.　測定ネットワークの構成

パケット損失がどのように起こるか、その傾向を見るために、パケット損失の発生状況を測定した。インターネット上の 2 地点をそれぞれ送信端、受信端と定め、送信端から等間隔に同一パケット長のパケットを UDP により転送し、受信端での到着時刻、およびパケット損失状況を測定した（到着時刻は損失モデルには直接関係ないが、時間帯ごとのパケット損失状況を比較するために測定したものである）。送信端は東京大学の情報基盤センター内にあるホスト、受信端は東邦大学の情報科学科内のホストを使用しており、両ホストは SINET を通してつながっている。送信端は学内のネットワークを介して SINET バックボーンに接続され、受信端は 1.5 Mbps のアクセス回線で東京大学の SINET 接続点に接続されている（図 1.3）。転送は、UDP ペイロード 130 byte の UDP パケットを 10 ms 間隔で送出した。これは、ヘッダ部がおよそ 30 byte なので、パケットの全サイズは 160 byte となり、約 128 kbps の低速回線を介したビデオトラフィック転送（$160 \times 8\,\mathrm{bit}/10\,\mathrm{ms} = 1280 \times 1000\,\mathrm{bit}/s = 128\,\mathrm{kbps}$）を想定したものである。実測は、転送中に失われたパケットのシーケンス番号を記録することにより、どのパケットが損失を起こったか測定した。送出したパケット数はいずれも 720000（2 時間）である。

実測した結果、表 1.1 のようなデータが得られた。

表 1.1、表 1.2 において冗長度とは、(損失率) × (ブロック長) で計算する。小数点以下は切り上げである。ここでは、ブロック内のパケット数を 100 としている。次節のパケット損失の傾向分析は、この

表 1.1.　SINET 上におけるパケット損失の実測データ（128 kbps、20 byte/パケット）

標本	測定日	時間	総数	損失数	損失率	冗長度 (理想値)
1	10 月 9 日	13:30〜15:30	720000	40972	0.0561	6
2	10 月 10 日	14:30〜16:30	720000	99200	0.1378	14
3	10 月 11 日	15:30〜17:30	720000	53063	0.0737	8
4	10 月 15 日-1	13:30〜15:30	720000	35123	0.0488	5
5	10 月 15 日-2	16:00〜18:00	720000	49715	0.069	6
6	10 月 16 日	13:00〜15:00	720000	6666	0.0093	1
7	10 月 17 日-1	13:00〜15:00	720000	34991	0.0486	5
8	10 月 17 日-2	15:00〜17:00	720000	89162	0.1238	13
9	10 月 17 日-3	17:00〜19:00	720000	10486	0.1456	2
10	10 月 18 日-1	13:00〜15:00	720000	38245	0.0531	6
11	10 月 18 日-2	15:00〜17:00	720000	18868	0.0262	3

表 1.2.　SINET 上におけるパケット損失の実測データ（64 kbps、20 byte/パケット）

標本	測定日	時間	総数	損失数	損失率
12	1 月 25 日-1	14:00〜14:30	720000	18892	0.026239
13	1 月 25 日-2	15:00〜15:30	720000	111077	0.154274
14	1 月 25 日-3	17:00〜17:30	720000	15676	0.021772

実測データをもとに行っていく。

1.4 パケット損失の傾向分析

パケット損失のランダム発生モデルとして二項分布が考えられるので、二項分布と実測データとの比較を行った。処理ブロック長が大きいと、そのブロック分のパケットを蓄積して誤り訂正符号計算を行うため遅延が大きくなる。遅延に対する要求が異なる対話型と非対話型の応用を想定し、それぞれについてブロック長を定めて分析を行った。

対話型は、たとえば IP 電話の応用を想定し、遅延が小さいことを要求する。電話では、遅延が 300 ms を超えると音声の中に不自然な部分が現れると言われている。仮に、音声データ帯域を 64 kbps（無圧縮)、パケット長 20 byte のデータを送信した場合、パケット送出速度は 400 個/秒であり、遅延許容時間の 300 ms の中に 120 パケットが入る。これをすべてインターリーブブロック滞留遅延に費やすとすると、最大の処理ブロック長は 120 個となる。FEC の計算処理に要する時間や回線自体の遅延、またルータでの滞留遅延などが国内では最大で 10 ms から 30 ms 程度とすると、処理ブロック数は 100 個に設定すれば 300 ms の遅延に収めることができるだろう。このことから、64 kbps の帯域で、20 byte のパケットサイズのデータを送った場合を想定して、実測データ（表 1.1 のものとは別に実測したもの、そのデータについては図 1.2 を参照）と二項分布とで比較を行った。ただし、二項分布における確率 p は、各標本において 損失数/全パケット数（損失率）で計算している。二項分布と実測データのそれぞれについて、処理ブロック内の損失数の傾向をグラフにしたところ、図 1.4 から図 1.6 のようなグラフを得た。二項分布のグラフでは、ブロック内の損失数が $100 \times p$ 個のとき、確率のピークを迎えているが、実測データでは 0 個のときが最大となっている。また、損失数が 1 以上の点については、ピークと呼べるものはなく、およそ 40 から 70 までなだらかに値が続いている。これ

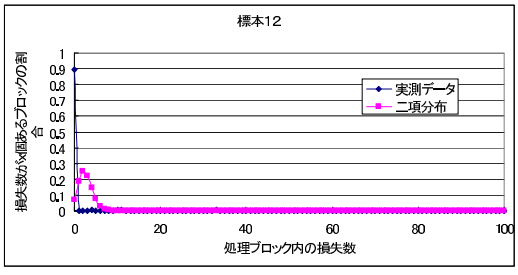


図 1.4.　処理ブロック内の損失数の推移

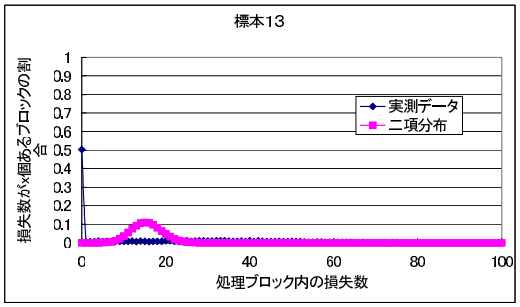


図 1.5. 処理ブロック内の損失数の推移

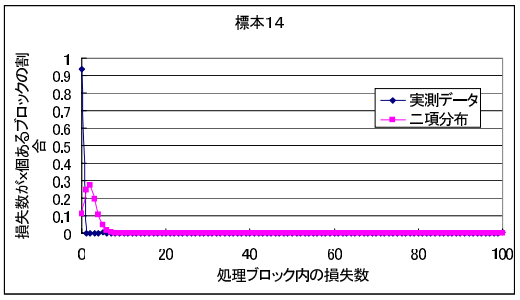


図 1.6. 処理ブロック（100 個）内の損失数の推移

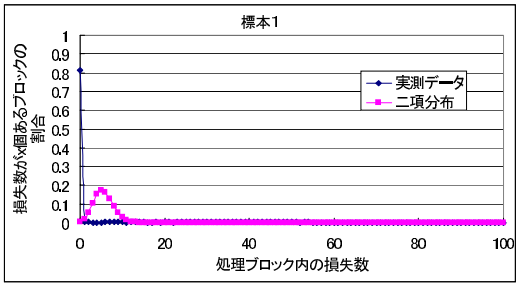


図 1.7. 処理ブロック（100 個）内の損失数の推移

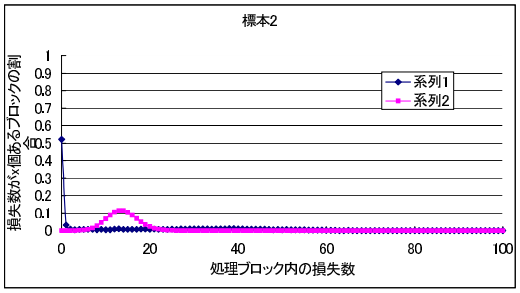


図 1.8. 処理ブロック（100 個）内の損失数の推移

らのグラフから、対話型のケースでは実測データから計算した損失率を用いた二項分布と実測データの分布とは明らかに違うことがわかる。

他方、非対話型の通信は、たとえばビデオ放送（ビデオオンデマンド）などの一方向的なビデオストリーミングのような、応答を伴わない通信である。MPEG での放送、通信を行った場合、128 kbps の帯域で送信を行えば、画像が比較的鮮明な画像を送信することができる。10 ms 間隔でデータを送信する場合、UDP パケットを 1 パケットあたり 130 byte で送信すれば、ヘッダを含めておよそ 160 byte となるので、128 kbps の帯域を占めるデータ送信となる。ビデオオンデマンドの場合、再生開始要求から実際の開始までの遅延の許容範囲は、たとえば 1 秒から 5 秒程度と考えられる。10 ms 間隔で送出した場合、1 秒の遅延はパケット 100 個分に相当するので、処理ブロックを 100 個単位と考えて、実測データ（標本 1〜11）を 100 個ずつの処理ブロックに分けて、ブロック内での損失数を実測データの各標本と二項分布とで比較を行った。二項分布における確率 p は、対話型の場合と同様の計算で求めている。比較の結果、図 1.7〜図 1.8 のようなグラフを得た。二項分布では対話型の場合と同様に、損失数が $100 \times p$ 個の地点で確率のピークを迎えているが、各実測データでは、0 個の地点が最も高くなっている。これらのグラフから、非対話型のケースでも、実測データの分布を二項分布では近似できないことがわかる。

処理ブロック内の損失数について、確率分布と実測データとでは、その傾向に大きなひらきが見られる。この原因の 1 つとして、実際のネットワーク上でのパケット損失の傾向が、バースト性を持つことが考えられる。これは、パケット損失がある一定期間に集中して発生するという性質で、ネットワーク上での輻輳が原因であると考えられる。この性質のため、長時間平均の損失率と、ブロック長のような短い時間区間での損失率で、開きが生じる。これは実データを調べることで検証できる。標本 1 から 3 に関して、1 分間隔で損失数の割合を図示したものが図 1.9 から図 1.11 である。これらの図から、短い時間間隔でパケット損失の傾向を見ると、損失率が 0 に近い値を示す時間帯や 0.3〜0.5 弱の値を示す時間帯があるなど、損失率に変動が見られる。すなわち、パケット損失は一定でなく、バースト性が見られることがわかる。このバースト性のため、長い時間間隔で見た場合に損失率が均等化されることで、短い時間間隔で見た場合の最大損失率と著しく異なるという状態が生じてしまう。

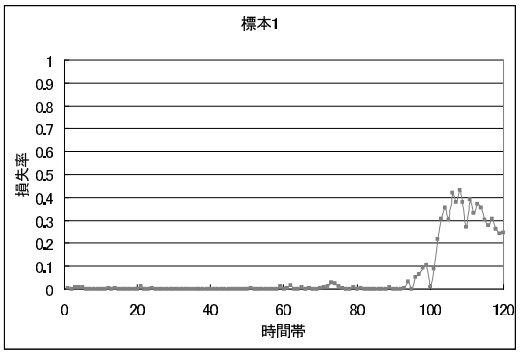


図 1.9.　時間帯ごとの損失率の遷移

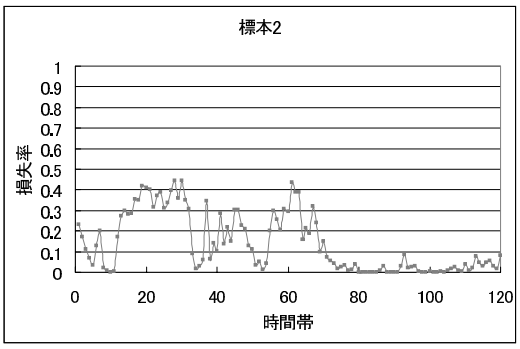


図 1.10.　時間帯ごとの損失率の遷移

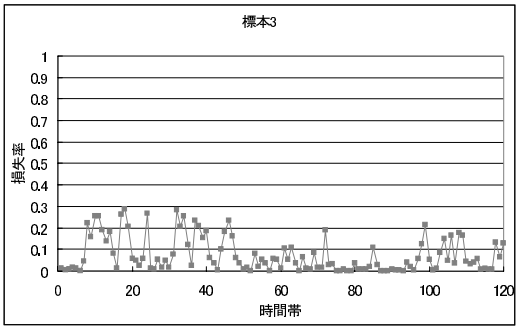


図 1.11.　時間帯ごとの損失率の遷移

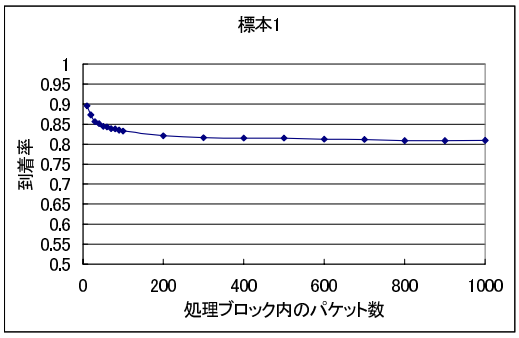


図 1.12.　二項分布から求めた冗長度を用いた場合（$p = 0.0561$）

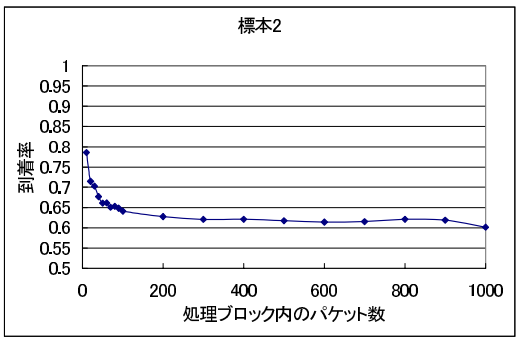


図 1.13.　二項分布から求めた冗長度を用いた場合（$p = 0.1378$）

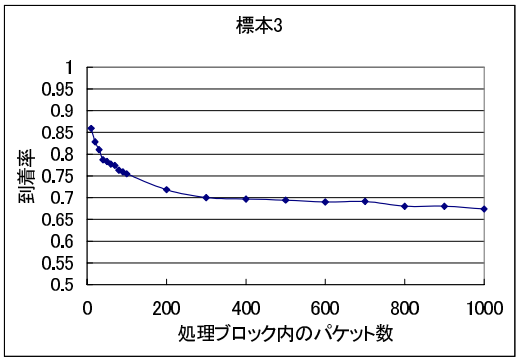


図 1.14.　二項分布から求めた冗長度を用いた場合（$p = 0.0737$）

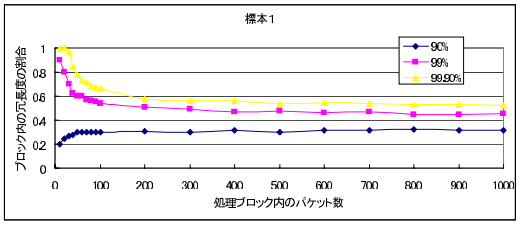
1.5 FEC による損失回復のシミュレーション

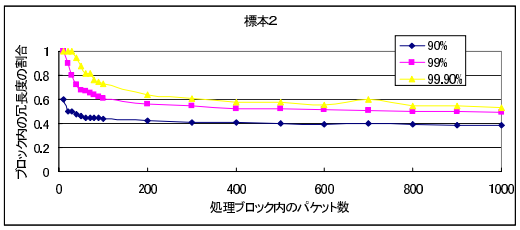
次に、パケット損失状況データに対して、FEC によるパケット回復を適用した場合の回復性能を求める。

損失回復のシミュレーション方法は、

1. それぞれの標本について、処理ブロック単位で分割する。
2. ブロック内の冗長パケット数（以下、冗長度）を与える。
3. ブロック内の損失数が冗長度以下であれば、そのブロック内の損失はすべて回復されたとみなす。
4. 処理ブロック長を、10 個単位で 10 個から 100 個まで、または 100 個単位で 200 個から 1000 個までに設定する。
5. 到着率（※）の規定値を、90%、99.0%、99.9%とする。

但し、到着率とは、送信者が送信した全パケットに対して、受信側に届けられたパケットの割合と定義する。すなわち、{（損失を起こさずに送信されたパ





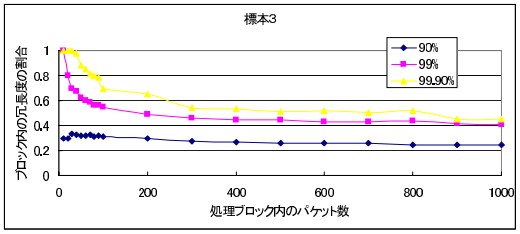
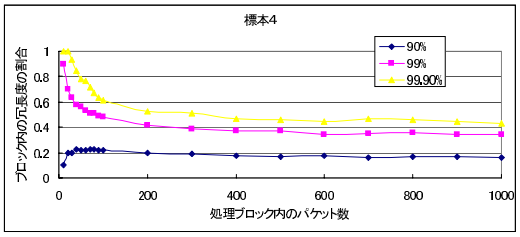
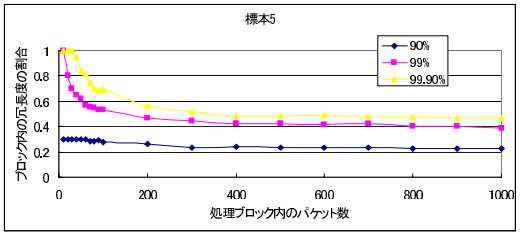


図 1.15. 回復後のブロック内パケット数と冗長度（標本 1–3）





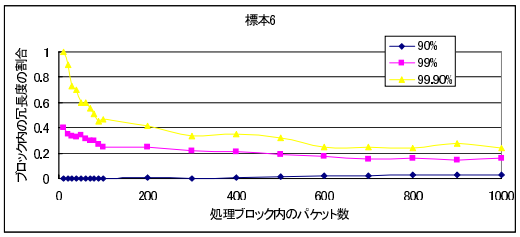
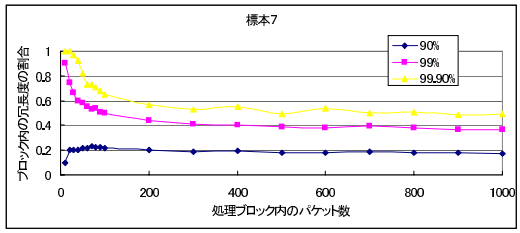
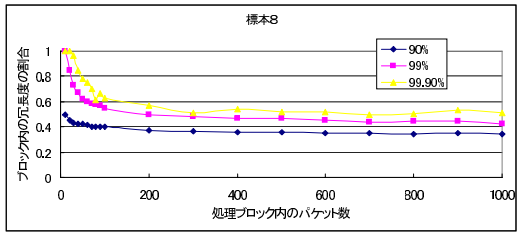


図 1.16. 回復後のブロック内パケット数と冗長度（標本 4–6）





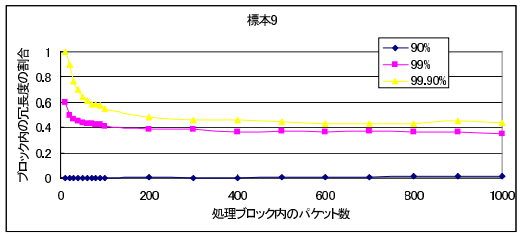


図 1.17. 回復後のブロック内パケット数と冗長度（標本 7–9）

ケット)+（処理ブロック内で損失回復されたパケット）} /(全送信パケット)で計算する。

実測データについて、ブロック内の冗長度を変えて、FEC によるシミュレーションを行った。まず、冗長度をパケット損失率の長時間平均値とした場合の冗長度と到着率との関係を求めた。パケット損失率の長時間平均値は、総損失数/全パケット数×処理ブロック長（小数点以下切り上げ）とした。その結果を図 1.12 から図 1.14 に示す。この図から、確率分布に基づいた冗長度では、損失回復を行っても到着率の規定値に達しないことがわかる。

冗長度をいろいろな値としてシミュレーションした結果、冗長度と到着率との関係は、それぞれの標本について、図 1.15 から図 1.17 のようになった。これらの図から、到着率の規定値を満たすためには、処理ブロック長に対してどれだけの冗長度を加えれば良いかを推定することができる。

また、処理ブロック長を増やせば増やすほど、ブロック内の冗長度の割合を下げることができ、結果として、ブロック内の実データの割合が増加することがわかる。ただし、このグラフでは、それぞれの標本においてブロック長がおよそ 400 から 700 の範囲

で冗長度の割合が増加、すなわち実データの割合の減少が見られる。ブロック長が大きい分、遅延も増加するので、処理ブロック長は 400 から 700 の範囲に設定するのが適当と思われる。また、標本によっては、規定値を 90%と設定すると処理ブロック長を増やすごとに冗長度の割合も増加している現象が見られた。

1.6 考察

本研究から得られた結論は、以下のとおりである。

- 実際のネットワーク上でのパケットの損失率は等確率ではなく、等確率を仮定した従来の確率分布では、損失率を推定できない。
- ネットワーク上でのパケット損失の傾向は、バースト性を有し、損失率の一定にならない。

以上の 2 点から、FEC の冗長度を確率分布から決定した場合、十分な回復性能が得られないことがわかる。また、FEC の処理ブロック内のパケット数と冗長度の決定については、

1. ネットワーク上で FEC によるパケット損失の回復を行う場合、規定された到着率や処理ブロック内のパケット数によって、冗長度が設定できる。
2. ブロック内のパケット数を増加させた分、実データの割合も増加する。ただし、処理ブロック長を 400 から 700 に設定すると実データの割合が減少してしまう場合もある。

と言えることがわかった。この 1)、2) をもとに処理ブロック長や冗長度を設定すると、損失率によって処理ブロック長を 400 から 700 に設定し、到着率 99%を保障するためには 4 割程度の冗長度、99.9%を保証するためには 5 程度の冗長度が必要である。

今回の研究は、大学間ネットワークというある程度の混雑=パケット損失が見込まれる環境を用いてのパケット損失の実測、および分析を行った。ただし、このネットワークは、現存する様々なネットワーク環境のほんの一部に過ぎず、今回の研究による結論が、すべてのネットワーク環境に当てはまるとは考えにくい。従って、より多くの環境に対して同様の実験や分析をすることが今後の課題である。また、今回の研究では、損失率が与えられたときに処理ブロック長や冗長度を設定するための実験式レベルのデータが得られたが、処理ブロック長や冗長度の設計値の損失率による変化をモデル化するにはいたらなかった。既存のバーストパケット損失の発生モデルを元にした、最適処理ブロック長や冗長度を算出するためのモデルの導出、更にそのモデルのパラメタとなる回線上のパケット損失率をどのような時間単位で測定すればよいかを検討することが挙げられる。

第 2 章　Design and Evaluation of Dynamic Protocol Selection Architecture for Reliable Multicast

2.1 Introduction

In recent years, as Internet gains widespread use, many applications have been developed. Some of them aim at group communications such as teleconferencing tool, software distribution tool, and so on. If we apply multicast to these applications, the sender doesn't have to copy the data for each receiver and has only to send one copy of it to the group address because routers duplicate it to the group members.

In case of Internet Protocol (IP) multicast, the sender doesn't need to know who is joining the group, and receivers can join and leave the group whenever they want. Although these aspects make communications scalable, multicast applications use UDP (User Datagram Protocol) for transport protocol and have the responsibility for reliable communications. Then it is difficult to apply these implementations to other applications because implementations of multicast protocols have been pushed into application programs.

2.2 Goal of this research

In chapter 2.1, we described the problem of reliable multicast. To solve this problem, we progress the research for realization of generic reliable multicast architecture[95, 84]. There are three points to realize versatility.

(1) We have proposed the reliable multicast architecture which pulls out the process for reliability from multicast applications.

(2) Multicast applications don't have to select reliable multicast protocol because appropriate protocols to the condition of the network are

selected automatically.

(3) Depending on the network situation, multiple protocols are selected to meet the network which changes dynamically.

(1) means that the process for reliability is separated from multicast applications. We incorporate protocol stacks into network nodes, and retransmission control, flow control, and so on are executed on these stacks. If the process for reliability are pulled out from applications by (1), each application has to select the reliable multicast protocol. Then (2) and (3) provide the mechanism which selects protocols for applications automatically and switches protocols in response to the state of the network.

Section 2.4 and 2.6 describe the procedure of preparation for the data distribution and the mechanism of the dynamic protocol selection to realize the mechanism on (2) and (3).

2.3 Related works

There are some related works which try to provide generic reliable multicast architecture. RFC3048[147] suggests *Building Blocks* which separate the each function of reliable multicast protocols into *building block*. This architecture provides reliable multicast protocols which is appropriate to multicast applications by combining these building blocks.

Also, Tajima, Morikawa and Aoyama add some functions to the routers to support reliable multicast[157]. In this architecture, RMAC (Reliable Multicast Adaptive Caching), routers execute functions to support reliable multicast by discarding unnecessary both NACKs and retransmission packets. In addition, by caching data packets adaptively to congestion and distributing them to multiple routers, RMAC realizes effective retransmission even if cache size is small.

Compared to these related works, our architecture is able to provide the mechanism which switches protocols dynamically. This mechanism can response to changes of the network condition as described above.

2.4 Preparation for the data distribution

Figure 2.1 shows the procedure of the preparation for the data distribution. In the following sections, we explain each item on Figure 2.1.

2.4.1 Acquirement of receiver information

At the beginning, preparatory for the appropriate protocol selection, the sender needs to know the number of receivers because it selects the protocol by considering two points, an application type and the scale of the multicast group.

We use RTCP (Realtime Transport Control Protocol)[55] receiver report packet to acquire receiver information. If the sender get this information, it will be able to select appropriate protocol because it knows the application type in advance.

2.4.2 Acquirement of topology information

Topology information is defined as the connection information of multicast routers. This information is used when the aggregation nodes are decided.

Figure 2.3 shows the format of the topology information for the network on figure 2.2. In this example, the sender's IP address is written on the first line, and the routers are written in order of scanning the multicast tree on depth first.

This information is distributed to receivers using multicast, and all members of the multicast group shares this information.

2.4.3 Allocation of aggregation nodes

If the number of receivers is large, the control packets between the sender and receivers put pressure on the bandwidth of the network and prevent the data transfer which is the intended purpose. We set two types aggregation nodes on the network, *Designate Nodes* and *Intermediate Nodes*. One designate node is allocated per one segment and aggregates the control packets from this segment (Figure 2.4). Intermediate nodes are the subset of designate nodes and aggregate the control packets between designate nodes (Figure 2.5).

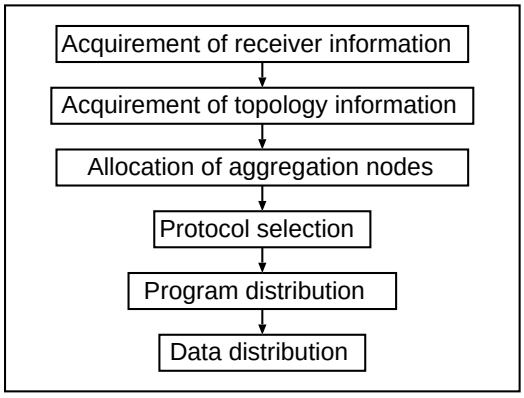


Fig. 2.1. Procedure of the preparation for the data distribution

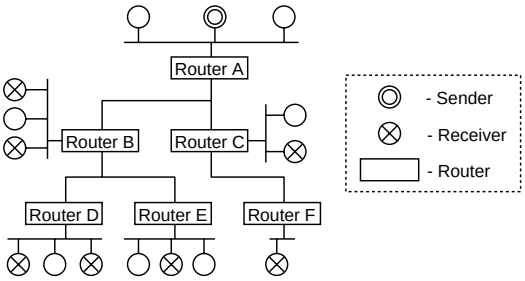
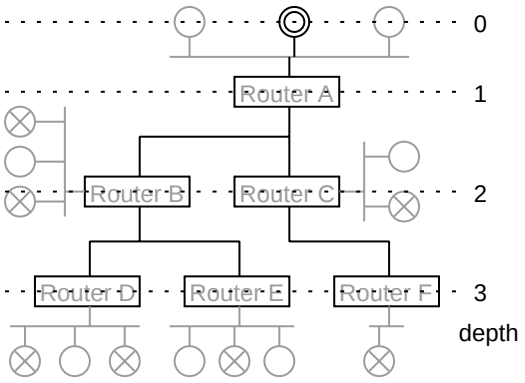


Fig. 2.2. Sample network



topology information

depth	address
0	Sender's IP address
1	Router A's
2	Router B's
3	Router D's
3	Router E's
2	Router C's
3	Router F's

Fig. 2.3. Format of topology information

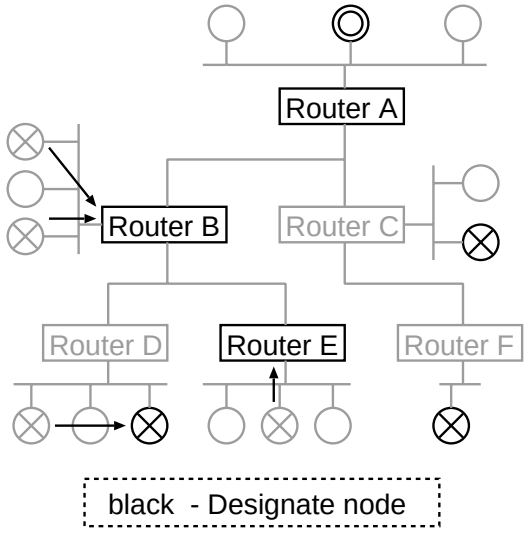


Fig. 2.4. Aggregation of control packets by designate nodes

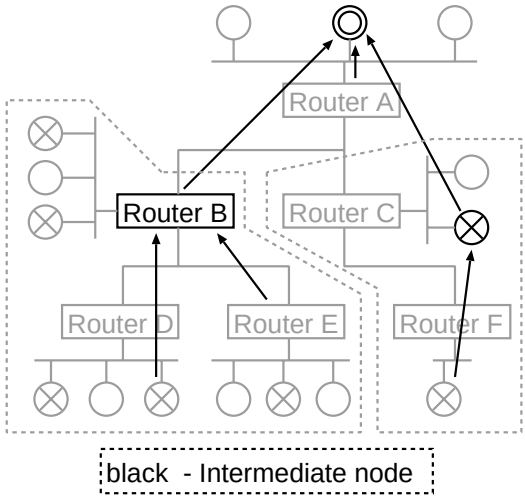


Fig. 2.5. Aggregation of control packets by intermediate nodes

We suggest the way of determining these nodes in section 2.5.

2.4.4 Protocol selection

When data distribution is started, we think over two factors to select reliable multicast protocols as described above. One is the application type, and the other is the scale of the multicast group. To select the appropriate protocol, we need to know an efficient protocol for each application and the threshold of the scale previously. In our architecture, we use simulation to get these information.

We configure the topology on the simulator along the actual network topology, and run the simulation on the various patterns about application type and group scale. It is possible to select protocols to meet the situation of the network by using these thresholds.

2.4.5 Program distribution

We make the process of reliable multicast protocol library functions to separate these process from multicast applications. Consequently, applications don't have to be conscious of reliability, and have only to send or receive the data. However, when the sender decide the protocol to start data distribution, each node aren't ready to execute the process for reliability. Then each node receives programs of the process from the server and they execute these programs. This mechanism uses the concept of active network[91] and is called *programmable switch approach*. This mechanism enables to switch protocols dynamically because each node is able to receive new programs if the state of network changes.

These programs are distributed using multicast, and this architecture needs to guarantee the reliability because all receivers have to receive them. When the recipients get the program from the sender, they return ACK (ACKnowledgement) packet to notify the sender of the reception. Designate nodes and intermediate nodes aggregate this packet because the number of ACK packets becomes large as the scale of the multicast group does.

2.4.6 Data distribution

After each node receives the program, the sender starts data distribution. The receivers who is joining the multicast group are able to join and leave this group all the time. The scale of the multicast group may become larger or smaller and communication efficiency may go down if one protocol is used for a long time. Our architecture enables each node to switch protocols halfway through communications. In section 2.6, we discuss about the mechanism of dynamic protocol switching.

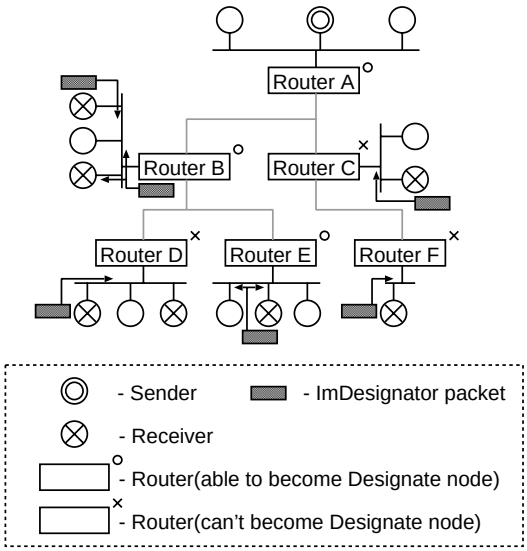


Fig. 2.6. Flow of ImDesignator packet

2.5 Aggregation nodes

2.5.1 Determining the designate node

One designate node is allocated per one segment. If possible, the router of each segment becomes the designate node because the router doesn't leave the multicast group. The receiver who has the smallest IP address among the group member becomes the designate node in case the router doesn't become the designate node.

Firstly, the router sends *ImDesignator* packet to his segment using broadcast if possible (Figure 2.6). Secondly, the receiver who has the smallest IP address sends *ImDesigantor* packet. Both the router and the receiver sends this packet, the router becomes the designator.

2.5.2 Determining the intermediate node

Intermediate nodes are the subset of designate nodes. We use topology information to decide this node. Firstly, each designate node checks topology information and designate nodes of which depth is the even number become intermediate nodes (Figure 2.7). Then all intermediate nodes have to decide the management area. In fact, each designate node needs to know the destination of the con-

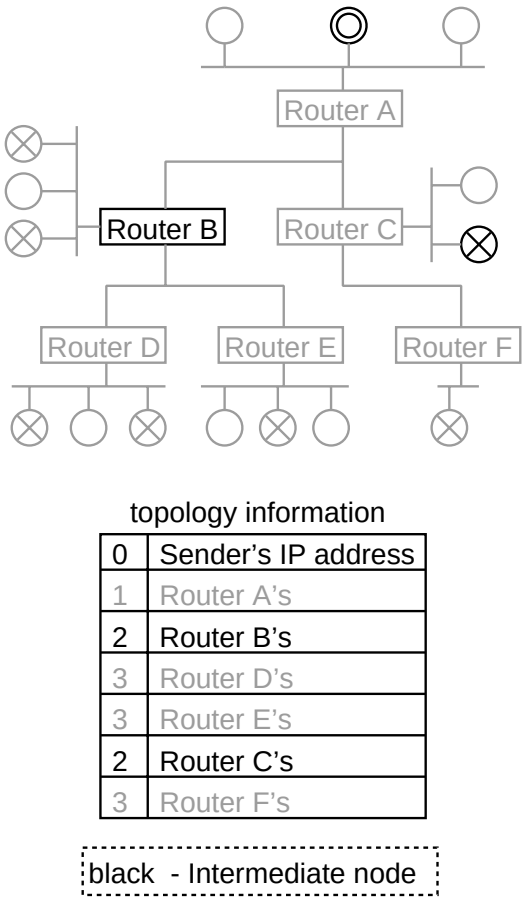


Fig. 2.7. Determination of Intermediate nodes

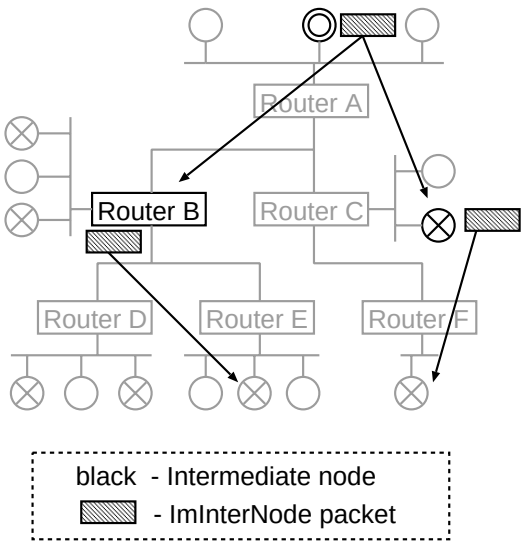


Fig. 2.8. Flow of ImInterNode packet

trol packets. Each intermediate nodes sends *ImInterNode* packet using multicast scoped by the TTL (Figure 2.8). When designate nodes receive this packet, they check the topology information and select the nearest and upstream intermediate node as the manager.

2.5.3 Relation between aggregation and multicast routing protocols

This architecture uses the tree structure to allocate the aggregation nodes. This tree has to correspond multicast routing protocols. There are two major types of multicast routing protocols. One type is the rooted tree. The root of the distribution tree on this type is the sender. Representative example of the rooted tree is DVMRP[141]. The other is the shared tree. Example of this type is PIM-SM[46]. Shared tree type protocol has a core point (for example, Rendezvous Point on PIM-SM) and this point becomes the root of the distribution tree.

In either case, multicast routing protocol has the root and makes distribution tree using this root. Our architecture allocates the aggregation nodes in tune with this tree.

Furthermore, there are two modes in terms of the expanse of receivers, dense-mode and sparse-mode. When the receivers are distributed densely, our aggregation mechanism certainly works. In case the sparse mode, multicast routers which has the receivers on its segment are distributed sparsely. This means that designate nodes are allocated sparsely. Our mechanism also works on the sparse mode because aggregation nodes are allocated only the segments which have receivers. On this situation each intermediate node keeps away from the other intermediate node but making tree is possible.

2.6 Mechanism of dynamic protocol selection

The situation of the network changes every moment during data distribution. For example, there are 200 receivers when the data distribution starts, but the recipients who have finished receiving the data leave the group one after another, then the scale of the group becomes small. On this situa-

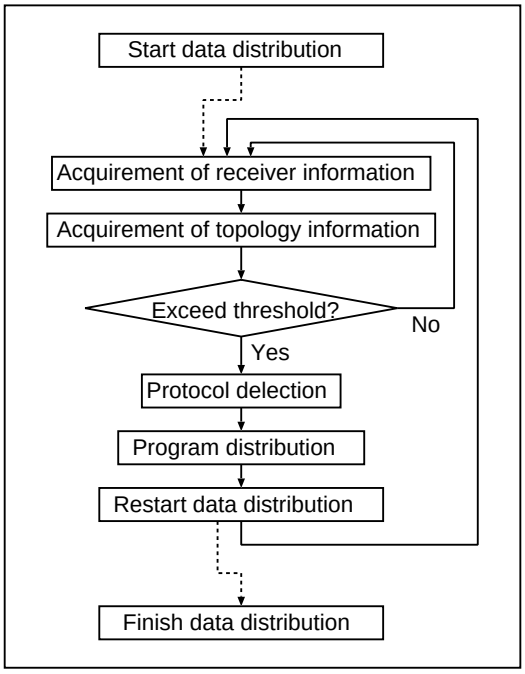


Fig. 2.9. Procedure of the dynamic protocol selection

tion, in case the sender selected protocol of which scalability for communications is better for the first time and continued to use this protocol, we can't able to get the high communication efficiency in the latter part of distribution. As a result, the sender selected a bad protocol for latter part.

We enable our architecture to switch protocols to response the change of network like this example. Figure 2.9 shows the procedure of the dynamic protocol selection. In the following sections, we explain each item on Figure 2.9.

2.6.1 Acquirement of receiver and topology information

It is important for appropriate switching to monitor the network and the scale of the multicast group. On this mechanism, the sender continues to gather RTCP receiver reports periodically . We use aggregation nodes (designate node, intermediate node) like preparation (section 2.4). If the scale of the group becomes small, the number of receiver reports becomes small, too. Consequently, the sender is able to detect network changes.

In addition to the receiver information, the sender continues to monitor the network topology. If the node which is the aggregation nodes leave the group or the multicast distribution tree changes, the new aggregation nodes have to be activated.

2.6.2 Protocol switching

The sender doesn't only monitor the network but also check the threshold by the simulation (section 2.4). When the sender detects the change of the scale, it compares the current number of receivers with the threshold. For example, if the number of recipients exceeds the threshold, the sender decides that each node is switched to the new protocol of which scalability for communications is better.

2.6.3 Program and data distribution

If the sender decides to switch protocol, it distributes the program of the new protocol. The way of distribution is the same as preparation (section 2.4). Receivers send ACK packets to the aggregation nodes when they receive the program, and the aggregation nodes retransmit it if necessary.

Though the sender is able to start distributing the data after program distribution, it is difficult to decide the timing of protocol switching. Because the retransmission process on old protocol still remained even if it stops on the middle of data transfer. Our architecture needs the time both the retransmission on the old protocol and the data distribution on the new one runs simultaneously. We discuss about this timing on section 2.9.

2.7 Evaluation of dynamic protocol selection

In this section, we evaluate the dynamic protocol selection using simulation. We make sure the effect of the dynamic protocol selection by using two protocols in response to the scale of the group.

2.7.1 Environment of experiment

We suppose that the number of receivers changes largely during data distribution. When

the data distribution starts, there are 200 recipients, but the receivers are decreased by 100 on reaching a certain time.

We use two reliable multicast protocols on this experiment, AFDP (Adaptive File Distribution Protocol)[31] and SRM (Scalable Reliable Multicast)[48].

AFDP is a NACK-based protocol. When AFDP recipients can't receive the data packet correctly, they send NACK (Negative ACKnowledgement) packet to the sender using unicast. On AFDP, only the sender is able to retransmit data packets using multicast. Flow control of AFDP is that sender's bandwidth of data transfer increases while recipients receives the data correctly. When the sender receives NACK from recipients, bandwidth of the sender decreases.

SRM is also a NACK-based protocol. When SRM recipients can't receive the data, they send NACK using multicast. SRM enables the sender and receivers to retransmit data packets. They send repair packets using multicast. SRM doesn't have the mechanism of flow control.

According to the experiment which we have done before, AFDP provides more effective retransmission in case of 100 receivers, and SRM provides more effective one in case of 200[83]. Then we assign SRM during 200 receivers, and switch to AFDP when the number of recipients changes from 200 to 100.

We examine the number of repair packets and throughput about the following three patterns.

1. Using only AFDP even if the number of receivers changes
2. Using only SRM
3. Using dynamic protocol switching (AFDP & SRM)

If the number of repair packets increases, they put pressure on the bandwidth of the network. In fact the less the number of packets is, the higher the performance is. And the performance of this protocol is higher when the throughput of this experiment is high.

In case of using dynamic switching, we use the procedure described in section 2.6. Figure 2.10 shows the time chart of protocol switching.

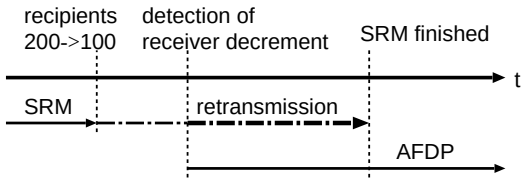


Fig. 2.10. Time chart of protocol switching

1. At the beginning, there are 200 receivers and they use SRM.
2. The number of receivers decreases from 200 to 100.
3. The sender detects decrement by receiver reports.
4. The sender stops data distribution on SRM. SRM only retransmit for recipients who can't receive data packets correctly.
5. The sender starts on AFDP.

On this environment, we configure Network Simulator (ns-2)[112]. The sender of this group sends the following data and we count the number of repair packets and calculate the throughput.

- Sender Bandwidth : 10 Mbps
- Packet size : 1000 B
- Data size (total) : 10 MB

2.7.2 Result of experiment

Figure 2.11 and Table 2.1 shows the result of this experiment. According to this graph, there are many repair packets on *AFDP only pattern*. In addition, this pattern takes more time to finish data distribution than others. In terms of retransmission, *SRM only pattern* and *AFDP & SRM pattern* seem to have the same performance, but *AFDP & SRM pattern* has higher throughput than *SRM only pattern* from Table 2.1. Thus this experiment shows that protocol switching has the highest performance of the three on this situation.

2.8 Imprementation problem of dynamic protocol selection

In section 2.7, We have found that our architec-

Table 2.1. Effect of dynamic protocol selection

	AFDP	SRM	AFDP & SRM
repair packets	577	387	375
throughput [Mb/s]	0.44	0.58	0.70

(repair packets means that the number of repair packets per one receiver (average))

ture contributes to the effective data distribution according to the circumstances. But this architecture has processes on preparation and protocol switching, so this processes may become an overhead.

When our architecture isn't used and a single protocol runs, We don't have to consider about this overhead because a single protocol doesn't need the complicated process which occurs by using multiple protocols. The problem of using a single protocol is that it doesn't correspond to changes of the network as described above.

Considering the advantage of single protocol, there may be a situation that is more effective by using single one. For example, the rate of protocol switching overhead becomes high in case the application which doesn't take much time for data transfer. We suggest the following two solution for this problem.

(1) Our architecture has two mode, *single protocol mode* and *multiple protocol mode*, and the sender selects the mode in tune with the application type.

(2) We adjust the timing of protocol switching to decrease the overhead of protocol switching as far as possible.

It is easy to implement solution (1) because the sender knows the application type. We get ready for the threshold to select single mode or multiple mode using simulation in advance. Then if the application has the large size data to send, the sender selects multiple mode because it takes much time to finish sending data. In section 2.9, we consider about the overhead of protocol switching to evaluate the solution (2) and about whether we get the appropriate timing of switching using simulation.

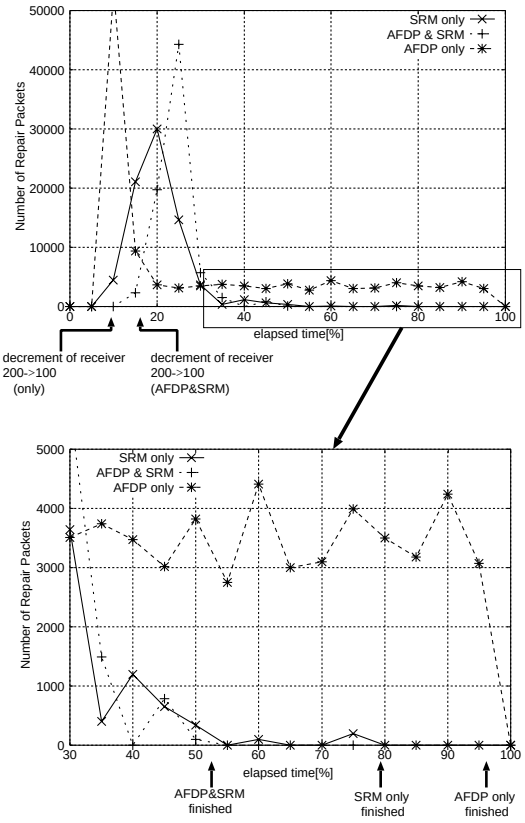


Fig. 2.11. Effect of dynamic protocol selection

2.9 Consideration of the overhead

It is important for our architecture to switch protocols smoothly to get the high throughput. We consider about the timing of protocol switching on this experiment.

2.9.1 Environment of experiment

We suppose the same environment as section 2.7 about the number of receivers. We assign two protocols according to the following time chart (Figure 2.12). T is the parameter which controls the timing of protocol switching. We examine the number of repair packets and throughput on the various value of T. The data from the sender is also the same as section 2.7.

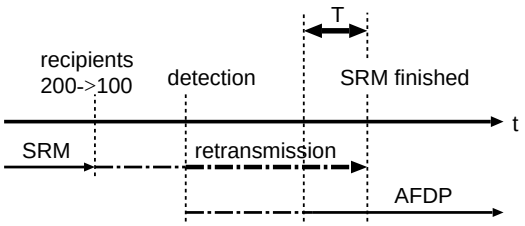


Fig. 2.12. Time chart of protocol switching with parameter T

2.9.2 Result of experiment

Figure 2.13 shows the result of this experiment. According to this graph, when the value of T is small, the throughput is low. Small T means that it takes much time to start data distribution on AFDP. As the sender has to wait for a long time to start sending on AFDP, the throughput isn't high. On the other hand, when T is large, the number of repair packets is large. Large T means that the sender distributes the data using both of the protocols simultaneously for a long time. This situation puts pressure on the bandwidth of the network. On this experiment, We found that the most appropriate value of T for this situation is 6[s].

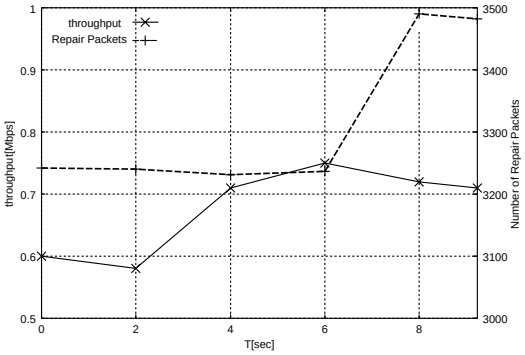


Fig. 2.13. Change of efficiency by parameter T

2.10 Conclusion

Our research aims at generic reliable multicast transport and we have designed the procedure of the dynamic protocol selection. On this procedure, the sender grasps the scale of the multicast group, and then selects appropriate protocols. These mechanisms enable to meet changes of the network dynamically. And we have experimented to evaluate the dynamic protocol selection. Using simulation, we have found that the dynamic protocol selection provides high performance according to the circumstances. Then we have discussed about the problem of the dynamic protocol selection and got the best timing to switch protocols to decrease the overhead.

2.11 Future work

In future, we need to simulate the various network situation, and we are going to implement our architecture on the basis of this experiment.

In this report, we consider about the scale of the multicast group as the factor to switch protocols. We also consider about the congestion as the second factor. More specifically, the sender checks the packet loss field of RTCP receiver report and monitor the receiving conditions. If some receivers are on the bad condition, the sender assigns the other protocol which is more effective to the bad receivers. In fact, multiple protocols run simultaneously from one locality to another. This mechanism needs to the boundary between two protocols. On our architecture, intermediate node plays as this boundary and translate from one protocol to the other protocol as the gateway.

After implementation, we compare single protocol transmission and multiple protocol one, then discuss with the overhead by using multiple protocols.

第 3 章　Study on Merge of TCP Traffic using Reliable Multicast Transport

3.1 Introduction

As the Internet has been developed, various services are provided over the Internet such as WWW, FTP, E-mail and so on. Especially WWW and FTP services are widely used as a method of information provision or data distribution. Also

in recent years, new technologies such as ADSL make extensive improvements in internet access, and this allows users of the Internet to retrieve large amount of information or data using the Internet more easily. Under these circumstances, it is expected that traffic on the Internet involved in the use of the services continues to increase, and overlapped traffic occur in higher probability. Many services over the Internet including the ones described above use TCP as a transport protocol. If multiple receivers attempt to retrieve the same data on a server at the same time, multiple TCP connections are established between the server and each receivers, then the data is duplicated and sent through each connection. Such overlapped traffic degrades the efficiency of networks.

Overlapped traffic described above results from the fact that the services use TCP, i.e. unicast, which is one-to-one communication model. On the other hand, IP also supports multicast (IP multicast), which is one-to-many communication model. Using IP multicast, data sent from a sender to multiple receivers is duplicated only at points in networks to be needed to delivery the data to all receivers. In this communication model, redundant traffic are reduced, and we are able to save bandwidth of networks when we send data to multiple destinations. If we could transfer data of WWW or FTP using IP multicast, it would become possible that we eliminate overlapped traffic and reduce redundant traffic in the Internet. However, WWW and FTP are protocols based on TCP. Even if we add a facility for IP multicast to the protocols, it is impractical to modify existing many systems to use new protocols.

In this report, we propose architecture for merging overlapped TCP traffic by replacing TCP data stream with IP multicast in networks. Moreover, we propose FTP proxy as a instantiation of applying our architecture to FTP traffic.

The remainder of this report is organized as follows. Section 2 describes the goal of our research. Section 3 presents architecture for merge of TCP traffic. Section 4 shows FTP proxy as a instantiation of proposed architecture. Section 5 describes implementation issue of FTP proxy and Section 6 is a conclusion.

3.2 Goal of our research

We have two major goals in designing the architecture proposed in this report. One is transparency to users and the other is scalability. If some modifications are needed to user applications in order to use the architecture, it becomes difficult to apply the architecture to existing systems. Therefore, the architecture must be transparent to users and all processing must be done in networks. Scalability of the architecture is also important. The more networks are covered by the architecture, the more overlapped traffic are able to be merged, and the architecture is able to make networks more efficient.

3.3 Architecture for merge of TCP Traffic

This section describes important components of architecture we propose for merging of overlapped TCP traffic. They are protocol translation server, session and reliable multicast protocol.

Protocol translation servers are the main component of our architecture. They transfer the data of overlapped TCP traffic between TCP and IP multicast at multiple points on networks. Protocol translation servers create a multicast session when they merge overlapped TCP traffic. Information of the created session is announced to all protocol translation servers in order to allow other protocol translation servers to join the session. When protocol translation servers send the data of overlapped TCP traffic using IP multicast, they use reliable multicast protocol. Reliable multicast protocol provide reliability with IP multicast, which is unreliable in data delivery. Each components are explained in the following sections.

3.3.1 Protocol Translation Server

In our architecture, we locates multiple servers at various point on networks. They receive data by TCP and send the data by IP multicast and

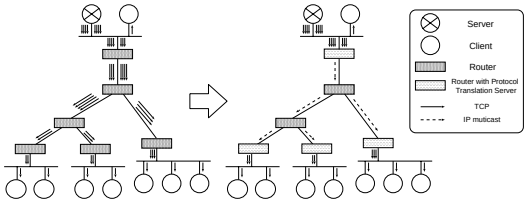


Fig. 3.1. Overview of replacing TCP streams with IP multicast by protocol translation servers

vice versa in order to realize merging of overlapped TCP traffic transparently for end-hosts. In this report, we call such a server as protocol translation server. Protocol translation servers keep watch on TCP streams on networks. When they detect overlapped TCP streams such as multiple transfers of the same file, they start protocol translation. Protocol translation server communicates with end-hosts using TCP and transmits the data received from a end-host to other protocol translation servers using IP multicast. The judgment way of whether multiple streams carry the same data depends on the protocol of the higher layer.

Figure 3.1 shows overview of replacing TCP streams with IP multicast by protocol translation servers.

3.3.2 Session

We describe a multicast session in our architecture.

Multicast Session

When a protocol translation server merges overlapped TCP streams, it creates a multicast session to send the data in the streams using IP multicast. Though it is possible to identify multicast sessions by transport address they use, we use different multicast address and the the same well-known port number for every multicast sessions created by protocol translation servers.

Session Description

We have to decide the way of description of multicast sessions to share session information among protocol translation servers. We use SDP (Session Description Protocol)[60] for that purpose.

SDP defines text format for general real-time multimedia session description purposes. Though SDP is originally defined to describe multimedia sessions, which distribute audio or video streams, SDP provides a method of describing application specific attributes as well. We decided to use SDP because it has enough capability for describing multicast sessions we create.

Session Announcement

A protocol translation server which created a session has to announce the information of the session to other protocol translation servers. We use SAP (Session Announcement Protocol)[61] in order to distribute the session information described using SDP.

In SAP, a host which creates a multicast session periodically multicasts a packet which contains information about the session to well-known multicast address and port. Hosts have only to listen on the multicast address and port to collect information of current active sessions. When a session ended, the announcer of the session is able to send a session deletion packet to inform listeners that the session ended.

Figure 3.2 shows the relationship between SDP and SAP. A protocol translation server which created a session multicasts SAP packets periodically. A SAP packet consists of a SAP header and a payload. The payload is a session description described using SDP. An example of session descriptions is explained in section 3.4.5.

SAP is relatively a simple protocol and does not have good scalability. In future, if new protocol is proposed for session announcement as an alternative to SAP, we will discuss the use of the new protocol.

3.3.3 Reliable Multicast Protocol

Since IP multicast use UDP as transport protocol, IP multicast does not ensure reliability in data delivery. It means that lost packets are not recovered, and receivers may receive duplicated packets, and the order packets arrive may be different from the order the packets were sent. On the other hand, TCP provide reliability. There-

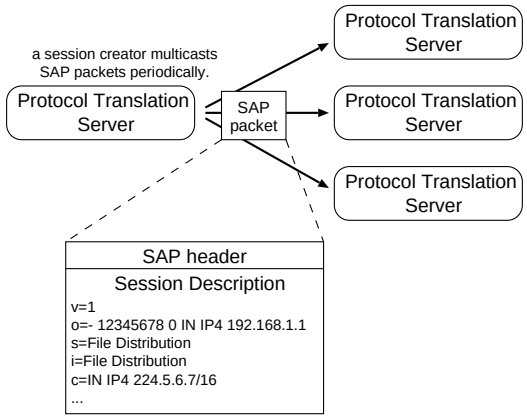


Fig. 3.2. Relationship between SDP and SAP

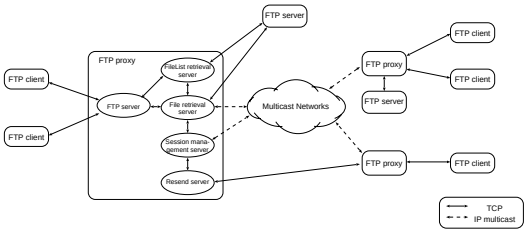


Fig. 3.3. Illustration of the system configuration and components of FTP proxy system.

fore, the communication between protocol translation servers using IP multicast must be reliable in order to ensure reliability of the communication between end-hosts using TCP.

As a method of providing IP multicast with reliability, many reliable multicast protocols have been proposed such as AFDP[32], SRM[49] and RMTP[89]. These protocols provide reliability by implementing facilities of detection and recovery of lost packets and sequencing packets. However, the method of implementing such facilities is different per reliable multicast protocols, and they respectively have different characteristics of scalability and throughput. It is needed that the reliable multicast protocol used in our architecture has good scalability because the more extensively protocol translation servers are deployed, the more efficiently we can merge streams. Though there are many reliable multicast protocols as mentioned above, there is no single reliable multicast protocol which always achieves good throughput for the various receiver set size.

Therefore, we use dynamic protocol selection architecture for reliable multicast[129, 130], which our laboratory are researching. In this architecture, the system selects an optimum reliable multicast protocol using information about the type of the application and the number of receivers in the multicast group and sends data received from the application to multicast group members using the selected protocol. Applications that use this architecture can efficiently multicast with reliability under various circumstances.

3.4 FTP Proxy

In this section, we propose FTP proxy as a instantiation of applying our architecture to FTP traffic.

3.4.1 Target Traffic of FTP Proxy

We suppose to merge download traffic of a file from an anonymous FTP server by FTP proxy. Upload traffic is outside of target of FTP proxy because we consider that there are few opportunity to merge upload traffic. In this report, we assume that anonymous FTP servers allows users to only download files and upload of files to the servers is prohibited.

3.4.2 System Configuration

A system which merges download traffic consists of multiple FTP proxies. Each FTP proxies are located at various point on networks. A FTP proxy has the role of protocol translation server described in section 3.3.1. Though we described that merging of traffic must be completely transparent to users, we let protocol translation server pretend to be FTP server in the system we describe here to simplify the implementation. Namely, FTP proxy has two roles. The first is as a protocol translation server, and the second is as a FTP server. This is why we call protocol translation server FTP proxy here.

FTP clients login to FTP proxy instead of a real FTP server and request files or file lists. FTP proxy itself does not have any files to provide

FTP clients with in initial state. FTP proxy retrieves the file requested by the client from a real FTP server according to need, and at that time, it merges traffic using reliable multicast protocol if possible. When FTP proxy retrieves files or file lists from a real FTP server, it saves retrieved data in a local disk as a cache. The behavior of FTP proxy is described in detail later.

3.4.3 Components of FTP Proxy

A FTP proxy consists of five components. They are FTP server, list retrieval server, file retrieval server, session management server and resend server. FTP server accepts login from FTP clients and provides them with FTP services. List retrieval server retrieves file lists from a real FTP server and manages retrieved file lists. File retrieval server retrieves files from a real FTP server and manages retrieved files. Session management server creates multicast sessions, announces them, and collects session information using SAP. Resend server accepts resend requests from other FTP proxies.

Figure 3.3 shows the system configuration and the relationship between each components of FTP proxy.

3.4.4 Behavior of FTP Proxy

In this section, we explain the behavior of FTP proxy in detail.

Basic Behavior

FTP server listens on port 21 and provides clients with FTP services as a real FTP server performs. When a client requests a file list by a LIST or STAT command, FTP server asks file list retrieval server for the requested file list. The behavior of the file list retrieval server is described in the next section. When a client requests a file by a RETR command, FTP server asks file retrieval server for the requested file. The behavior of the file retrieval server is described in section 3.4.4.3.

Session management server maintains two session lists. One is an active session list and the other is a completed session list. Session management server joins the well-known multicast address defined by SAP and collects session information which other FTP proxies are announcing. The collected session information are added to the active session list. When session management server receives a session deletion packet, it moves the ended session from the active session list to the completed session list.

Process for File list Request

When a client requests a file list, FTP server asks file list retrieval server for the requested file list. The file list retrieval server checks through a local cache. If the requested file list exists in the cache, the file list retrieval server passes it to the FTP server. Otherwise, the file list retrieval server logins to a real FTP server and retrieves the requested file list. In this case, the file list retrieval server uses TCP to connect to the real FTP server. It means that traffic of the file list retrieval is not merged by IP multicast. In general, the size of a file list is not so large, and we consider that it is difficult to merge this kind of traffic. The file list retrieve server passes the retrieved file list to the FTP server and saves it in a local disk as a cache. The FTP server sends the received file list to the client.

Process for File Request

When a client requests a file, FTP server asks file retrieval server for the requested file. The file retrieval server checks through a local cache. If the requested file exists in the cache, the file retrieval server passes it to the FTP server. Otherwise, the file retrieval server sends inquiries to session management server about whether there is an active session for the requested file. If there is a session for the file, the file retrieval server joins the session and receives the file using reliable multicast protocol. At that time, the beginning part of the file has been already sent, and the file retrieval server is not able to obtain them from the session. Therefore, the file retrieve server asks resend server running on the sender of the session to resend the missing data. The resend server sends the requested data using unicast or IP multicast.

The resend server selects protocol in consideration of the size of the requested data. If the size is smaller than a threshold, the resend server chooses unicast, and otherwise it chooses IP multicast. We are going to investigate appropriate value for the threshold in process of employment of this system. The file retrieval server passes the file received from the session and the resend server to the FTP server.

If there is no active session for the requested file, the session management server looks up in the completed session list. If there was a session for the requested file, the file retrieval server asks resend server running on the sender of the completed session to send the file. The resend server asks session management server running on the same host to create a new session. Then, the resend server sends the requested file to the created session. The file retrieval server joins the created session and passes the received file to the FTP server.

If there is no session for the requested file even in past times, the file retrieval server logins to a real FTP server and retrieves the file. In parallel, the file retrieval server asks the session management server to create a new session in order to inform other FTP proxies that it is possible to merge traffic. The file retrieval server passes the received file to the FTP server.

The FTP server send the received file from the file retrieval server to the client. The File retrieval server saves the file in a local disk as a cache.

```
v=1
o=- 12345678 0 IN IP4 192.168.1.1
s=File Distribution
i=File Distribution
c=IN IP4 224.5.6.7/16
t=0 0
m=data 8888 udp 0
a=filename:/ls-lR.gz
a=filesize:2790879
a=start:0
a=end:2790878
```

Fig. 3.4. Example of session descriptions

3.4.5 Session Information

As described in section 3.3, when a new session is created, session management server announces the session information described by SDP using SAP. The session description used in this system contains the following information.

- multicast address, port number and TTL used in the session
- the name of the file transfered in the session (absolute path)
- the size of the file transfered in the session
- the range of the file transfered in the session

When a file is transfered, as the case may be, whole file is not needed. In that case, the session description includes information about which part of the file is transfered through the session.

Figure 3.4 is an example of session descriptions. Each line is one field, and the head character of each line is the field name. “v=” field is the version of SDP, and “o=” field contains information about the originator of the session. “s=” and “i=” field are the session name and the additional session description respectively. “c=” field contains the multicast address and TTL used in the session. “t=” field specify the start and stop times for the session. “0 0” means that the session is not bounded to time. “m=” field is a media description. The first sub-field is the media type whose value is defined by SDP. The second sub-field is the transport port, and the third sub-field is the transport protocol. The fourth and subsequent sub-field are media formats. In figure 3.4, the “m=” field means that bulk-data are transfered in the session, and it uses UDP as a transport protocol, and the used port number is 8888. “a=” field contains the attribute of the session. FTP proxy uses four attributes. “filename” and “filesize” specify the name and the size of the whole file transfered in the session. “start” and “end” specify the start and the end position of the part of the file transfered in the session. In case of figure 3.4, the whole file is transfered in the session because the start position is the head of the file,

and the end position is the tail of the file.

3.5 Implementation Issue of FTP Proxy

In this section, we show implementation issue of FTP proxy described in the previous section.

3.5.1 FTP server

FTP server which is a part of FTP proxy must implement FTP services. However, it is difficult to implement FTP server from scratch. We decided to use the source code of WU-FTPD[56], which is widely used as anonymous FTP servers.

3.5.2 Reliable Multicast Protocol

As described in section 3.3.3, it is ideal to use dynamic protocol selection architecture in order to achieve good throughput under various circumstances. However, the implementation of dynamic protocol selection architecture has not completed yet. We are temporarily going to use an existing implementation of reliable multicast protocol for a while. Though several implementation of reliable multicast protocol are available now, we use libsrm developed by MASH project[117]. This is because the libsrm is provided in the form of C++ library, and it is easy to build into our programs.

3.6 Conclusion

The goal of our research is the realization of efficient use of networks by merging overlapped TCP traffic into a single stream using reliable multicast transport. In this report, we proposed architecture for our goal. In our architecture, multiple protocol translation servers are located at various points on networks. When they detect overlapped TCP traffic, they create a multicast session and transfer the data through the created session. SDP and SAP are used to share the information of created sessions among all protocol translation servers. Moreover, they use reliable multicast protocol in order to ensure reliability of the communication on IP multicast. We also proposed FTP proxy as a instantiation of our architecture for FTP traffic.

In future, we are going to evolve the FTP proxy proposed in this report to transparent FTP proxy. Moreover, we are going to apply our architecture to traffic of other protocols.