第XI部

信頼性を有するマルチキャスト技術

第 11 部
信頼性を有するマルチキャスト技術

第 1 章　環境と活動概要

1.1 RM-WG の活動

Reliable Multicast WG は今年も昨年と同様に、研究会での研究発表と討論を中心とした研究的な活動を行ってきた。本報告書ではその研究活動を報告する。

1.2 RM の研究環境

Reliable Multicast 関連技術の検討は、IETF の rmt-wg にて行われているが、今年度はあまり目覚しい技術革新は報告されていなかった。取り上げられる話題は、従来からのパケット損失対策技術のほか、M. Ruby らがフロー制御を再度取り上げていたのが目立っている。これは、S. Floyd など RM や E2E の中心となる研究者たちが通信（unicast, multicast を限定せず）の信頼性確保の基本として輻輳制御を重視していることとも合致している。

IETF rmt-wg では従来から、応用場面によって要求因子が大きく異なるので唯一万能の RM 機構を準備することができず、多種類の手法を使い分けられるビルディングブロック式のプロトコルを定義すべきである、という認識を持っていたが、この方向性は変化していない。いくつかの技術手法に基づく、パケット損失回復のためのプロトコルビルディングブロックが提案・議論・改良されてきている。

またフロー制御のためのビルディングブロックとして、WEBRC フロー制御 (Wave and Equation Based Rate Control) が提案されている。マルチキャスト環境でのフロー・輻輳制御自体の必要性は、マルチキャストがインターネット上で広範に利用される場合を想定すると、他の通信を一方的に圧迫しないために、是非必要であることは明白である。この場合、インターネットを利用している他のトラフィックはほとんどがユニキャストの TCP を用いた通信であるから、TCP との整合性 (TCP-friendliness) が大きな要請となる。TCP のフロー・輻輳制御は原理的にパケットの損失を検出して送信制御へフィードバックし、送出量を加減している。この制御系に共存し、お互いにほぼ公平に回線容量を分け合うことが要求されている。

マルチキャストの場合、配信規模のスケーラビリティを要求すると、受信側から送信側へのフィードバックを返すことが困難になる（Ack Implosion 問題）。従って、マルチキャストのフロー・輻輳制御においては、フィードバックを用いない方法でなければならず、かつ上述のように TCP-friendly でなければならない。今までにも、たとえば layered multicast を用いた制御などが議論されてきたが、WEBRC は layered multicast、TCP-friendlyness などの研究を引き継いで、フロー・輻輳制御を行う機構としてまとめられているものである。

本年の研究活動として以下の 2 件を詳細に取り上げる。

- 高信頼マルチキャストでのパケット損失の回復手段として提案されている手法は、マルチキャスト配送規模や配送経路形状などの環境によって効率・性能が異なる。環境に合わせてそれらを選択し、アプリケーションからは分からないように動的に切り替える機構を提案した。特に、円滑な切り替え・引継ぎのための手続きと、切り替え条件の検出について検討・評価し、後述の FTP proxy にて実用化している。
- 高信頼マルチキャスト機構を、FTP proxy アプリケーションへ適用する例を示した。FTP proxy は、インターネット上でのダウンロード等の FTP トラフィックをマージする。このとき、データもとのサーバーから複数の FTP proxy への転送を、高信頼マルチキャストを用いることによって、ネットワークのトラフィックを軽減する。RM プロトコルの 1 つである SRM を用いて FTPProxy を実装し、FTP proxy を実装し、性能を測定・評価している。

第 2 章　汎用性のある高信頼性マルチキャスト通信に関する研究

2.1 概要

インターネット上において 1 対多、多対多型の通信を効率良く行うための技術として、IP マルチキャストが注目され、研究が進められている。IP マルチキャストでのデータ配信は、トランスポート層プロトコルとして UDP(User Datagram Protocol) を使用しており、通信を行う際に信頼性が求められる場合には、高信頼性マルチキャストプロトコルが各アプリケーションで実装されている。本研究では汎用性のある高信頼性マルチキャスト通信の実現を目標とし、再送制御やフロー制御をアプリケーションに依存しない形で行うための手法を提案している。本手法の実現には、状況に応じた適切な高信頼性マルチキャストプロトコルの選択を行うことが必要となる。プロトコルの選択は、アプリケーションが意識する必要はなく、マルチキャストグループの規模や、受信者の分布に応じて通信の途中でも動的にプロトコルを切り替えて常に最適なプロトコルでの通信を提供する。我々はこの方式を動的なプロトコル選択方式と呼んでいる。本報告書では動的なプロトコル選択方式の概要を述べ、プロトコルを切り替える際に必要となる閾値の獲得のために、シミュレーションを用いて既存の高信頼性マルチキャストプロトコルの評価を行っている。

2.2 はじめに

近年の爆発的なインターネットの普及にともない、多くの新しいアプリケーションが開発されている。その中には、遠隔会議や、ソフトウェアの電子的配布などのグループ活動を主眼としたアプリケーションも含まれる。このようなアプリケーションに対してマルチキャストを適用した場合、送信者はデータを複製する必要がなく、送信する際には、グループ毎に割り当てられるマルチキャストアドレスにデータを送信すればよい。また、送信されたデータは、ルータで経路が分岐する場合にのみコピーされるので、冗長なトラフィックの発生を抑えることができ、効率の良い通信が可能となる。

マルチキャスト通信の中で、IP(Internet Protocol) 上でデータをやりとりするのが IP マルチキャストである。IP マルチキャストには、送信者が受信者のリストを維持する必要がなく、受信者はマルチキャストセッションに自由に参加、離脱することができるため、大規模で動的なグループでの通信が可能になるという特徴がある。しかし、IP マルチキャストではトランスポート層プロトコルとして UDP(User Datagram Protocol) を使用しており、信頼性がない。そこで現在は、再送制御やフロー制御を提供する高信頼性マルチキャストプロトコルが数多く開発されている。しかし、現在の高信頼性マルチキャスト通信には以下のような問題点がある。

- 高信頼性マルチキャストプロトコルのプロトコルスタックはアプリケーションに実装されているものが多く、新たなマルチキャストアプリケーションに対して既存のプロトコルを適用することが難しく、汎用性がない。
- 現在、マルチキャストアプリケーションは数多く存在し、それぞれのアプリケーションが求める信頼性、対象としているマルチキャストグループの規模などが異なるため、単独の高信頼性マルチキャストプロトコルで全てのアプリケーションに対応することが困難になっている。

2.3 本研究の目的

現在の高信頼性マルチキャスト通信における問題点を解決するために、我々は汎用性のある高信頼性マルチキャスト通信の実現 [146] を目標として研究を進めている。実現のための方針は以下の 2 つである。

1. 今までアプリケーションで行われていた高信頼性マルチキャストプロトコルの処理をネットワークに組み込む。
2. アプリケーション側で使用するプロトコルを選択する必要はなく、ネットワークが状況に応じた適切なプロトコルを選択する。

(1) は高信頼性マルチキャストプロトコルの処理をアプリケーションから切り離し、アプリケーションとは独立した形で再送制御やフロー制御を行うというものである。(1) により、プロトコルスタックがネットワークに組み込まれると、各アプリケーションはプロトコルを選択しなければならなくなる。そこで (2) によってマルチキャストグループの規模や、

受信者の分布の時間的な変化についても考慮し、適切なプロトコルを動的に選択することを考えている。我々はこのプロトコルの選択方式を動的なプロトコル選択方式 [87][88] と呼んでいる。

現在 IETF の RMT-WG において Building Blocks 方式 [26] が提案されており、高信頼性マルチキャストプロトコルの標準化作業が進められている。Building Blocks 方式では、各高信頼性マルチキャストプロトコルが持つ機能に注目し、これらの機能を組み合わせることによって、各アプリケーションに合ったプロトコルを提供するものである。この方式と本研究が大きく異なる点は、時間的なネットワークの変化に対応できるという点である。単一のプロトコルで信頼性を提供する場合とは異なり、マルチキャストグループの規模の変化や、受信者の分布の変化に対応して、最適な高信頼性マルチキャストプロトコルを動的に割り当てていくことができ、各プロトコルの長所を生かした効率の良い通信を実現できると考えている。

動的なプロトコル選択方式においてプロトコルを切り替える際に必要となるのが閾値である。例えば、受信者数が 100 人以下のマルチキャストグループの場合にはプロトコル A の方が通信効率が良く、100 人以上の場合にはプロトコル B の方が効率がよい場合を考える。この場合、事前に切り替えの目安である 100 人という閾値を獲得しておかなければ、通信の途中でプロトコルを切り替えることができない。そこで、本研究ではプロトコルの切り替えの閾値をシミュレーションを用いて事前に獲得しておく。切り替えの基準とするマルチキャストグループの規模と、受信者の分布状況に関して様々なパターンを想定したシミュレーションを行い、どのような状況においてどのプロトコルを割り当てるのが良いかという情報を獲得する。

2.4 動的なプロトコル選択方式

本節では、提案している動的なプロトコル選択方式の概要について述べる。まず最初に、本研究で対象としている高信頼性マルチキャストプロトコルについて簡単に説明し、次に動的なプロトコル選択方式について説明する。

2.4.1 使用する高信頼性マルチキャストプロトコル

本研究で注目している高信頼性マルチキャストプロトコルは、AFDP(Adaptive File Distribution Protocol)[43] と SRM(Scalable Reliable Multicast)[60] の 2 種類である。AFDP の再送処理は、受信者から送信者へのユニキャストによる NACK に対して送信者からのマルチキャストによる再送という形で行われる。SRM では、受信者からの NACK はマルチキャストでグループの参加者に送信され、パケットを正しく受信できている受信者からもマルチキャストによる再送が可能となっている。この 2 種類のプロトコルの一般的な評価では、SRM の方が AFDP に比べて規模適応性があると言われている。これは、受信者が多くなってきた場合に、送信者への再送要求が集中する AFDP は SRM に比べて通信効率が落ちるためである。

2.4.2 動的なプロトコル選択方式

動的なプロトコル選択方式は、大きく分けて以下の 2 つの処理から成る。

1. ネットワークの状況を把握するために、定期的に行われる制御パケットの送受信
2. プロトコルの切り替え処理

(1) の制御パケットの送受信の流れをタイムチャートにしたものを図 2.1 に示す。図中の左側が送信者の処理、右側が受信者の処理となる。送信者は、プロトコル名パケットを受信者に対してマルチキャストで送信する。プロトコル名パケットには現在使用している高信頼性マルチキャストプロトコルの種類が情報として含まれる。このパケットを定期的に送信することで、途中から参加した受信者が現在使われているプロトコルを知ることができ、ただちに受信を開始することができる。送信者は、プロトコル名パケットよりは低い頻度で、受信者レポート要求パケットをマルチキャストで送信する。受信者レポート要求パケットを受信した受信者は、ランダムなタイマを用いて一定時間待った後、受信者レポートパケットを送信者にユニキャストで返送する。送信者は返ってくるレポートパケットをカウントし、現在の受信者数を獲得する。また、レポートパケットにはタイムスタンプを格納しておき、送信者と受信者の間の RTT を計測する。送信者は得られた RTT から受信者間の RTT の平均値、標準偏差を算出し、受信者の分布に関する情報を獲得する。

受信者の管理のために送受信されるパケット群は、

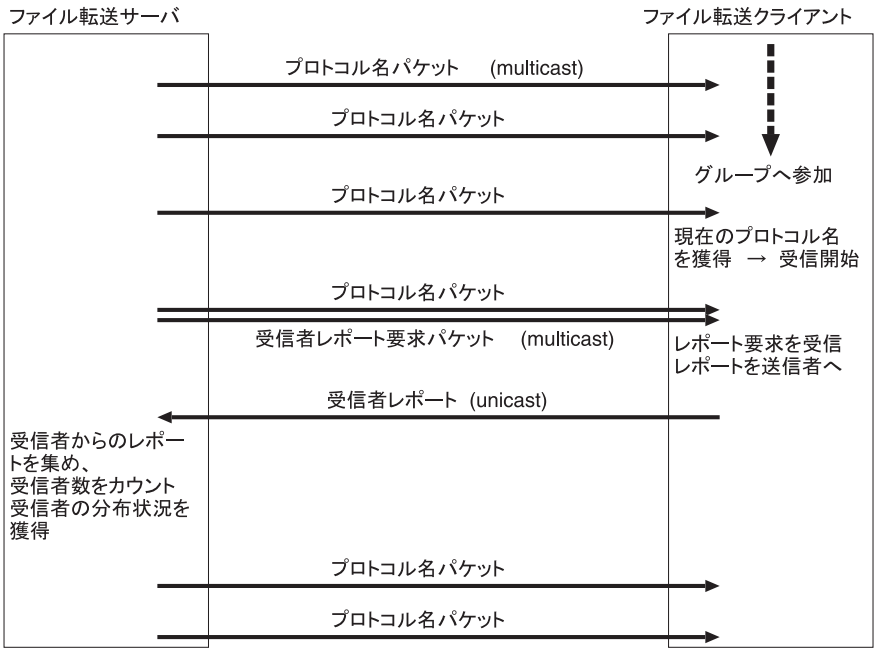

図 2.1.　制御パケットの送受信による受信者管理

全てトランスポートアドレスとして UDP を使用する。また、これらのパケット群に対する信頼性については考慮しない。これは、プロトコルを切り替えるのに必要な情報には 100%の信頼性は必要なく、受信者の人数と分布が大まかに獲得できればよいためであり、制御パケットの信頼性を保証することで生じるオーバヘッドを避ける目的がある。

(2) のプロトコル切り替え処理の流れをタイムチャートにしたものを図 2.2 として示す。(1) の制御パケットの送受信により、送信者は現在の受信者の状況を獲得することができる。この情報と事前に獲得しているプロトコル切り替えのための閾値とを比較し、プロトコルを切り替える必要がある場合には (2) の切り替え作業を行う。

制御パケットの送受信によって、送信者が切り替えの必要があると判断した場合、プロトコルの切り替えを通知するためのパケットをマルチキャストで送信する。この切り替え通知パケットは、受信者全員に確実に届く必要があるため、高信頼性マルチキャストプロトコルの AFDP を使用して送信される。AFDP では、送信したデータに対する NACK が届かなくなってから一定時間が経過すると、全員にパケットが届いたと判断する実装になっている。送信者はこの時間が経過した後、新しい高信頼性マルチキャストプロトコルでのデータ配信を再開する。その後は、(1) と同様に、制御パケットの送受信が行われる。

2.5 シミュレーションによる高信頼性マルチキャストプロトコルの評価

本節では、プロトコルを切り替える際の閾値を獲得するために、シミュレーションを用いて高信頼性マルチキャストプロトコルを評価するための実験を行う。シミュレーションを行うソフトには、UCB で作成されたネットワークシミュレータである ns2 を使用した。

2.5.1 実験環境

想定したネットワークは JGN(Japan Gigabit Network) のギガビットネットワーク通信回線を参考にしたものであり、図 2.3 のようになる。図 2.3 において、日本を縦断しているのが 10Gbps のバックボーンリンクであり、バックボーンルータにはそれぞれ木構造をしたネットワークが接続され、AS を構成している。末端のリンクの帯域は 100Mbps である。各ノードは受信者となることができ、末端以外のノードについては受信者とルータの役割を兼ねることができる。

以上のような環境の下で、1 人の送信者から、複数の受信者 (20–200 人) に対してパケットを送信する。送信されるデータは、ファイル転送アプリケー

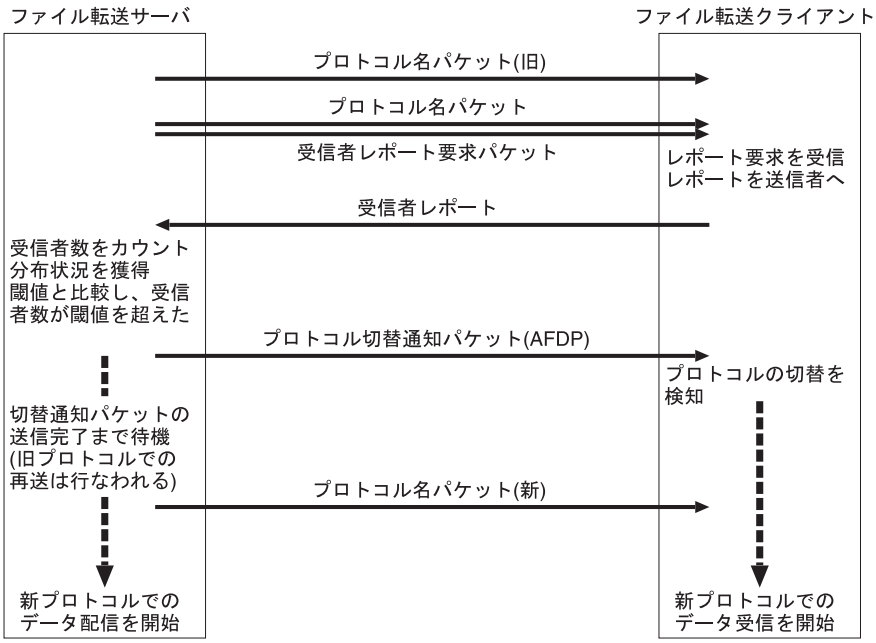

図 2.2. プロトコルの切り替え処理

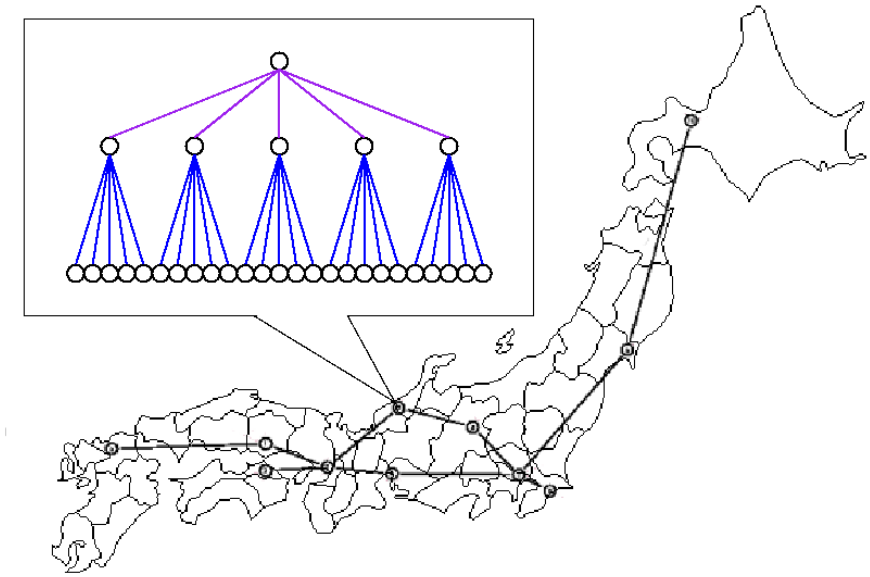

図 2.3. 想定したネットワーク

表 2.1. 各実験における受信者の分布

分布パターン	(1)	(2)	(3)
RTT 平均値 [ms]	52.26	43.87	68.59
標準偏差 [ms]	12.23	8.207	8.845

ションを想定しており、1000 バイトのパケットを 10 Mbps で 1000 パケット送信する。図 2.3 に示したネットワーク上に、パケットをロスするリンクをランダムに 5 ヶ所設定しておき、パケットロス率を 5%としておく。送信者から送られたパケットが欠落すると、高信頼性マルチキャストプロトコルによる再送処理が行われる。実験では、受信者に到着する再送パケットの数をカウントし、より多くの再送パケットが届いているプロトコルほど、冗長な再送パケットが多く、効率の悪いプロトコルであるとして評価する。

実験の際には、受信者数の変化による各プロトコルの通信効率について評価すると共に、受信者の分

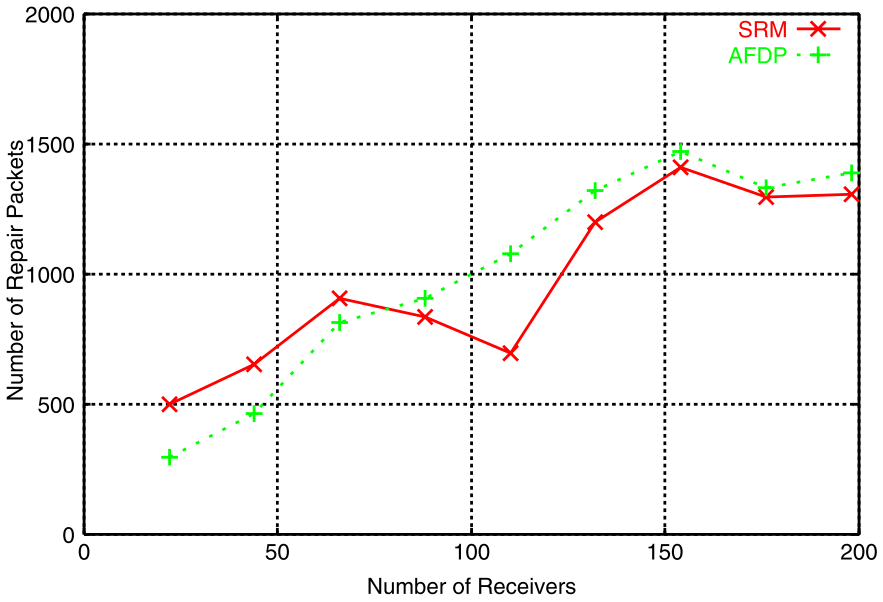

図 2.4.　AFDP と SRM の通信効率の比較 (受信者が各 AS に散在している場合)

布を変化させた場合の通信効率についても調べる。今回の実験では、受信者数は 20 人から 200 人まで変化させ、受信者の分布については以下の 3 種類のパターンを用いて評価した。

1. 受信者が各 AS に散在している場合
2. 送信者に近い AS に集中して分布する場合
3. 送信者から遠い AS に集中して分布する場合

(1) は、例えば受信者数が 20 人の場合には、すべての AS に受信者が 2 人ずつ分布するような場合を想定している。(2) については、20 人の受信者が送信者と同じ AS、もしくは送信者に近い AS に集中して分布する場合、(3) は送信者から離れた位置にある AS に受信者が集中する場合を想定している。

シミュレーション上での受信者の分布を定量的に評価するために、本研究では送信者から各受信者までの RTT(Round Trip Time) を測定し、RTT の平均値と標準偏差によって分布を示すことにする。平均値は送信者と受信者との距離の目安とし、標準偏差は受信者がどの程度分散して分布しているかの目安とする。

2.5.2 AFDP と SRM の評価

まず最初に、それぞれの分布のパターンについての RTT の平均値と標準偏差をまとめたものを表 2.1 に示す。平均値については、送信者に近い AS に受信者が集中している (2) のパターンが最も小さく、逆に送信者に遠い AS に受信者が集中する (3) のパターンが最も大きい。標準偏差については、各 AS に分散して分布するパターンである (1) の値が最も大きくなっており、集中して分布する形の (2)、(3) のパターンは (1) の標準偏差よりも小さくなっている。

次に実験結果を図 2.4〜図 2.6 に示す。図 2.4 は分布のパターンが (1) の各 AS に散在している場合、図 2.5 はパターン (2)、図 2.6 はパターン (3) にそれぞれ対応している。グラフの横軸はマルチキャストグループの規模、縦軸は受信者に届いた再送パケット数の平均をとっている。

図 2.4 の分布パターン (1) の場合、受信者数が 80 人以下の状況では、AFDP の方が若干再送効率が良いことが分かる。80 人以上になると効率が逆転し、SRM の方が効率が良い。受信者が集中して分布する場合は、図 2.5、図 2.6 のように、受信者の数に関らず SRM の方が効率が良いという結果が得られている。

2.5.3 実験の結果に対する考察

まず、分布パターン (1) の結果に関しては、AFDP と SRM の一般的な評価に沿ったものが得られている。AFDP の再送処理は、送信者への NACK と送信者からの再送という形になるため、受信者数が増加するに従って NACK の数も増える。送信者からの再送が増えると、送信者と送信者付近のリンクに負荷がかかり、さらにパケットロスを引き起こす。

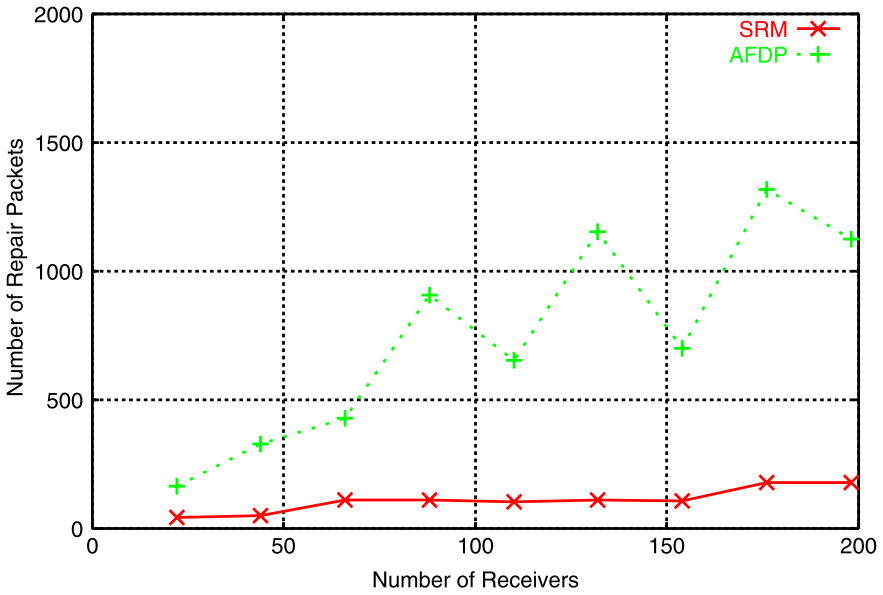

図 2.5. AFDP と SRM の通信効率の比較 (送信者に近い AS に集中して分布する場合)

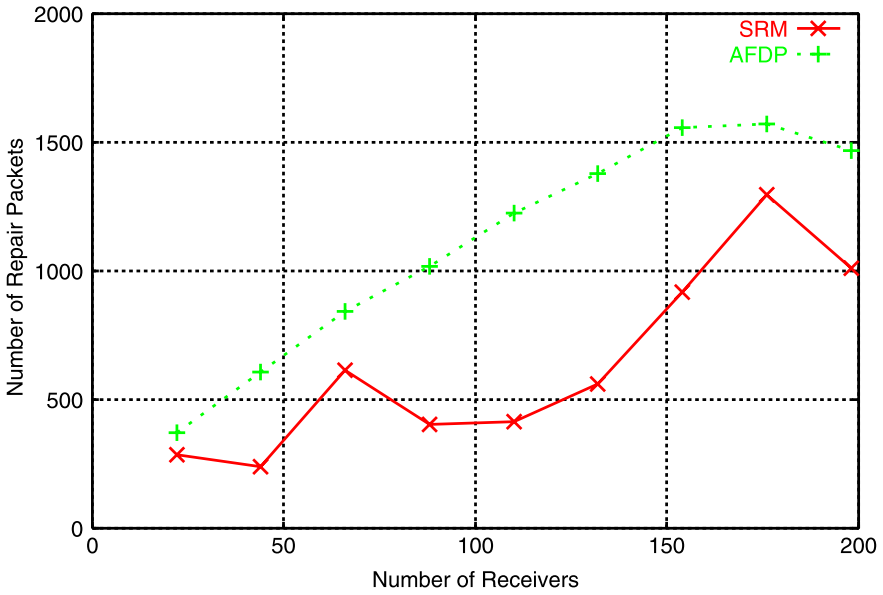

図 2.6. AFDP と SRM の通信効率の比較 (送信者から遠い AS に集中して分布する場合)

これにより再送パケットが欠落してしまい、さらに再送処理が行われるというように、悪循環に陥る可能性がある。

分布パターン (2)、(3) の受信者が AS に集中する場合に関しては、(1) の場合と異なり、受信者数が少ない場合にも SRM の方が効率が良いという結果が得られている。これは、SRM の局所的な修復が関係していると考えられる。SRM の場合、受信者も再送を行うことが可能なため、パケットを正しく受信できなかった受信者 A の近くに再送可能な受信者 B がいる場合には、受信者 B から再送を行うことで受信者 A は欠落したパケットを受信することができる。SRM を用いるとこのような局所的な修復が可能となり、受信者が集中して分布するパターン (2)、(3) の方が局所的な修復が発生する可能性が高い。これに対し、分布パターン (1) の方は、受信者間で再送を行う場合にも、距離の離れた別の AS からの再送となってしまう可能性が高いため、(2)、(3) に比べると効率が落ちていると考えることができる。

2.6 おわりに

本研究では、汎用性のある高信頼性マルチキャスト通信の実現を目指し、動的な高信頼性マルチキャストプロトコルの選択方式を提案している。本方式には、プロトコルの切り替えを行うための閾値が必要となるため、シミュレーションによって高信頼性マルチキャストプロトコルの性能比較を行っている。様々な状況下において有効なプロトコルを調べておくことで、受信者数、受信者の分布が時間的に変化した場合にも、有効なプロトコルを割り当てることができる。本報告書では、AFDP と SRM の 2 種類の高信頼性マルチキャストプロトコルについて、再送処理の効率についての評価を行った。

今後は、受信者の分布のパターンをさらに増やした実験を行うことを考えている。本報告書では、受信者の分布は 3 種類のパターンを想定したが、実際のネットワーク上で考えられるような受信者の推移をシナリオとして、RTT の平均値、標準偏差によって再送制御の効率がどのように変化するのかについて評価を行う。また、実験環境の改善についても考えており、シミュレーション上でのマルチキャスト経路制御プロトコルを実際の広域なネットワークに近いものにする。これにより、経路制御プロトコルを変えることによって高信頼性マルチキャストプロトコルの性能が変化するかどうかについても評価することができると考えている。

第 3 章　Study on Merge of Overlapped 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 through 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 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 eliminated, and it is possible to save bandwidth of networks when data are sent to multiple destinations. If we could transfer data of WWW or FTP using IP multicast, it would become possible to eliminate overlapped traffic and redundant traffic in the Internet. However, WWW and FTP are based on TCP and they are already in widespread use. Even if a new facility for IP multicast is added to them, it is impractical to modify existing many systems to use new protocols.

In this paper, we propose architecture for merging overlapped TCP traffic by replacing TCP data stream with IP multicast in networks. And we propose FTP proxy as a instantiation of applying our architecture to FTP traffic. Moreover we mention the implementation of FTP proxy and an experiment we performed.

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

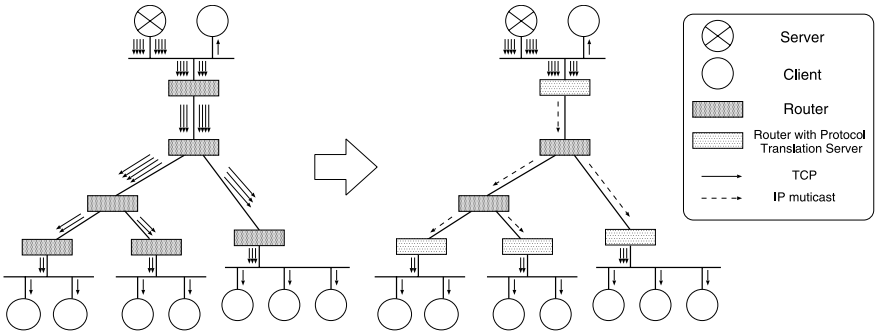

Fig. 3.1. Overview of replacing TCP streams with IP multicast by protocol translation servers

chapter 6 concludes this paper.

3.2 Goal of our Research

We have two major goals in designing the architecture we propose in this paper. 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 Overlapped TCP Traffic

This chapter describes important components of architecture for merge 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 using IP multicast between multiple points on networks. Protocol translation servers create a multicast session when they merge overlapped TCP traffic. Information of a 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 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 following sections.

3.3.1 Protocol Translation Server

In our architecture, we locate multiple servers at various points on networks. They receive data by TCP and send the data by IP multicast and vice versa in order to realize merge of overlapped TCP traffic transparently for end-hosts. In this paper, 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

In this section, we describe a multicast session in our architecture.

Multicast Session

When a protocol translation server merges overlapped TCP streams, it creates a multicast session

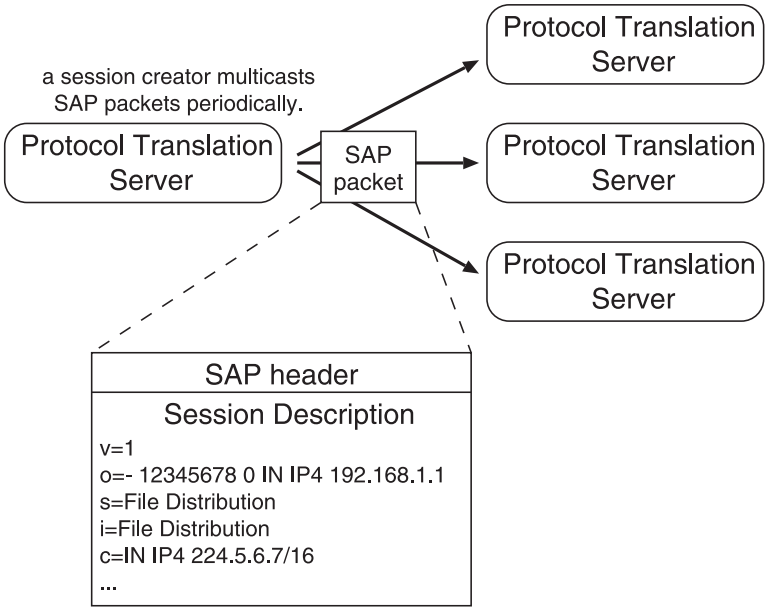

Fig. 3.2. Relationship between SDP and SAP

to send data using IP multicast. Each multicast session uses a different multicast address and the same well-known port number.

Session Description

We need a common format to describe multicast sessions in order to share session information among protocol translation servers. We use SDP(Session Description Protocol)[101] 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)[100] in order to distribute the session information described using SDP.

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.

3.3.3 Reliable Multicast 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 packets were sent. On the other hand, TCP provide reliability. Therefore, communications between protocol translation servers using IP multicast must be reliable in order to ensure reliability of communications between end-hosts using TCP.

As a method of providing IP multicast with reliability, many reliable multicast protocols have been proposed such as AFDP[43], SRM[60] and RMTP[118]. These protocols provide reliability by implementing facilities of detection and recovery of lost packets and sequencing packets. However, methods of implementing such facilities are different per reliable multicast protocols, and they respectively have different characteristics of scal-

ability 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 various receiver sets.

Therefore, we use dynamic protocol selection architecture for reliable multicast[144, 145], which our laboratory are researching. In this architecture, the system dynamically selects an optimum reliable multicast protocol using information about the type of the application and the number of receivers in the multicast group. Then it sends data received from the application to multicast group members using the selected protocol. Applications can efficiently multicast with reliability under various circumstances by this architecture.

3.4 FTP Proxy

In this chapter, 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 overlapped 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 paper, we assume that anonymous FTP servers allows users only to 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 proxy is 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 this system 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 as 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 a 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, file 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. When a client requests a file list by a LIST or STAT command, FTP server asks file list retrieval server for the re-

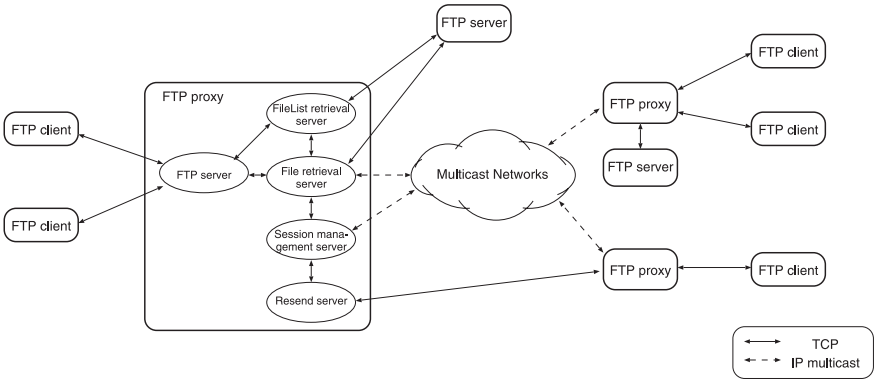

Fig. 3.3. Illustration of the system configuration and components of FTP proxy system

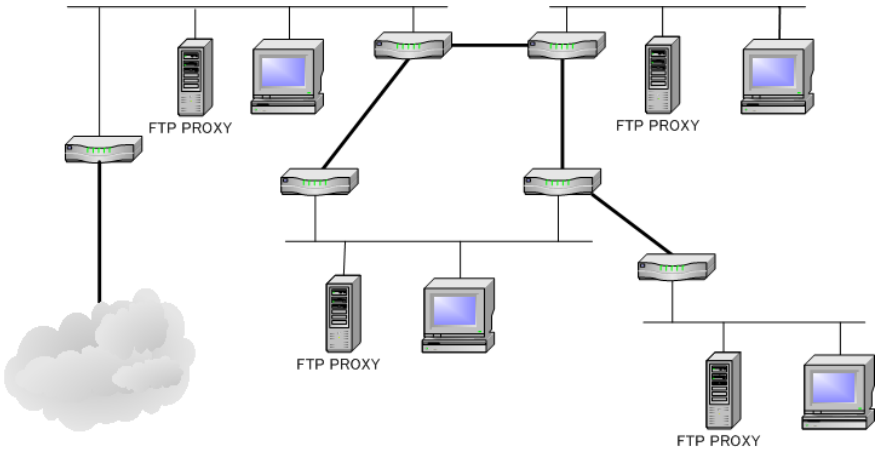

Fig. 3.4. Illustration of the experimental network

quested file list. When a client requests a file by a RETR command, FTP server asks file retrieval server for the requested file. The behaviors of file list retrieval server and file retrieval server are described later in this section.

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.

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
- the size of the file transfered in the session

3.5 Implementation and Experiment

In this chapter, we describe implementation and experiment of FTP proxy.

3.5.1 Implementation

We implemented servers of FTP proxy on Linux. Each server communicates with other servers using UNIX domain socket. As for implementation of FTP server, we leveraged wu-ftpd 2.6.2, which is FTP server widely used on the Internet to provide anonymous ftp service. Main modification applied to wu-ftpd is that codes to read file lists or files are replaced by ones to communicate with file list retrieval server or file retrieval server.

Though we described that our architecture use dynamic protocol selection architecture to multicast with reliability, the implementation of the architecture has not completed yet. Therefore, currently we have employed libsrm, which is an implementation of SRM by MASH project. Libsrm is provided in the form of C++ library and very easy to build into application softwares.

3.5.2 Experiment

Figure 3.4 shows the network for this experiment. We set up four FTP proxies in the experimental network. Each Linux PC is equipped with Pentium III 1GHz processor, 256 MB memory and 100Base-T NIC. The original FTP server is located at outside of the experimental network.

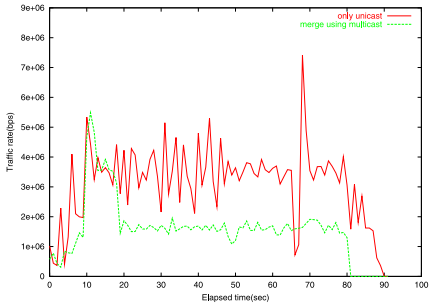

Fig. 3.5. Total traffic rate of backbone links (interval: 3 seconds)

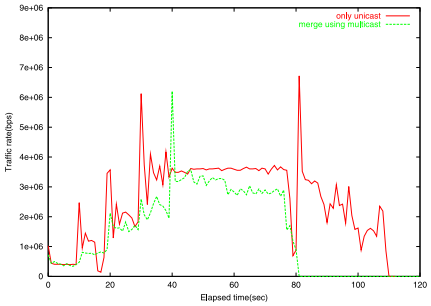

Fig. 3.6. Total traffic rate of backbone links (interval: 10 seconds)

We measured the total traffic rate of backbone links in the experimental network when four clients download the same file from the respectively different FTP proxy. Backbone links are drawn thickly in figure 3.4. Whether overlapped traffic occur depends on how clients access. In this experiment, each clients start downloading one after another with three or ten seconds interval. Though clients access FTP proxies from outside of the experimental network, packets between clients and FTP proxies are not counted into the result. For comparison, we also measured the total traffic rate of backbone links when FTP proxies don't merge overlapped traffic using multicast. At this moment, we are not able to multicast with libsrm at faster than 50 kbytes/s due to a problem on implementation. We limited the transfer speed between FTP proxies and clients to 50 kbytes/s to perform measurements under the same condition.

Figure 3.5 and figure 3.6 show the total traffic rate of backbone links in the experimental network when clients start downloading with three and ten interval respectively. In figure 3.5, the traffic rate is reduced to half by merging overlapped traffic in many portion. However, the effect of merging overlapped traffic is not clear until twenty seconds elapsed. This is because FTP proxies resend missing part of files by unicast immediately after the second or later clients start downloading. On the other hand, in figure 3.6, the traffic rate decreases in fewer degree than in figure 3.5 when FTP proxies merge overlapped traffic. In this case, the time which clients' accesses overlap is shorter and FTP proxies have to resend more by unicast, then FTP proxies communicate using both multicast and unicast while transferring the file. This makes traffic rate larger. However, when FTP proxies merge overlapped traffic, data transfers finish more first than as in only unicast and they eventually save total network bandwidth.

Currently, FTP proxies always resend data using unicast. However, resend traffic may be possible to merge under some situations. For example, if two clients start downloading the same file at once when other client is already retrieving the file, FTP proxy must resend twice and the two resend traffic exactly overlap. We are now considering to use multicast when FTP proxies resend relatively large amount of data.

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 paper, 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. FTP proxies accept users' login and retrieve files from a real FTP server on behalf of users. When multiple FTP proxies are requested the same file at the same time, they transfer the file using multicast between FTP proxies. In this paper, we implemented FTP proxy and performed experiment. Using FTP proxy, we succeeded to save bandwidth of backbone links when overlapped traffic occur.

Our experiment in this paper was done under the condition that overlapped traffic certainly occur. We are currently considering to examine how much FTP proxy can save bandwidth when real ftp users access them.