第 8 部

リアルタイム通信実験バックボーン: RT-Bone

第 1 章

はじめに

1.1 RT-Bone WG

本章では、WIDE Internet を利用してリアルタイム通信の広域実験およびその研究を行うことを目的とした RT-Bone ワーキンググループの研究動向、およびインターネットにおけるリアルタイム通信の現状について述べる。

パケット交換の best effort 通信を提供するインターネットにおいて，近年，新しい技術が急速に発達し，既に実験的，場合によっては実践的に導入されつつある．仮想チャネル技術をベースとした ATM との統合を始めとして，IP スイッチ，またはラベルスイッチングという技術が現在注目されている．このような環境においては当然ながら，ATM の回線指向型通信を利用した通信相手同士で資源を専有する通信，他の通信に影響されないリアルタイムの通信の実現が考える．このように資源予約をルータなどで行うための方法として，現在 IETF などでは，RSVP [?] が注目されている．RSVP のプロトコル仕様はまだ RFC として確定していないが，既にインターネットドラフトに基づく製品化も行われていて，研究および開発に利用できる実装なども提供されている．

リアルタイム通信を実現するためには，上記で述べた RSVP のような資源予約を行うためのセットアッププロトコル，および ATM などの CBR service やルータにおける特殊なキューイング機構，または帯域確保などの技術などが必要である．

そこで，RT-Bone WG では，現在利用できる技術を WIDE バックボーンの一部に導入し，広域でさらにライブトラフィックのもとで，RSVP やリアルタイム通信に伴って必要な技術の現状を調査するところから実験を始めた．具体的にいうと，図 1.1 に示す各 WIDE の NOC と実験に参加する組織において RSVP が実装されているルータ (まずは，CISCO) を導入することである．現在では，全ての接続ノードがまだ RSVP 対応にはなっていない．

本文では，まず RSVP の最近の動向，および RSVP の実装について簡単に紹介する．その後，広域環境における CISCO RSVP の評価実験について述べる．これは NTT-Data と奈良 NOC と東京 NOC を利用した実験である．

後半においては，トラフィック制御の研究および技術についてまとめてある．まずは，今後実験で導入しようと考えている PC ルータでは必要は不可欠な技術となる代替キューイング機構について述べる．最後に，トラフィック制御およびリアルタイム通信における理

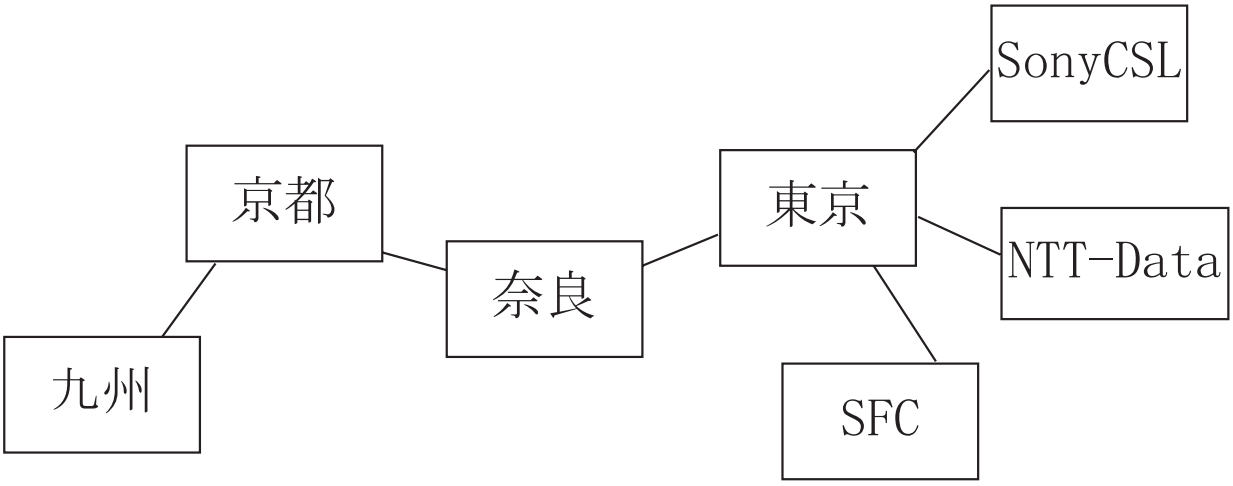

図 1.1: RT-Bone 構成図

論的実証可能で，かつ実用生の高い技術を生み出すための検討について述べる．

1.2 RT-Bone の今後

今後の計画としては，近いうちには 各 NOC の CISCO の RSVP を PC ベースのルータに徐々に置き換えて実験することである．PC ベースのルータのソフトウェアは，自由に変更することが可能なので，RSVP またはトラフィック制御に対して独自の改良点を生み出すことが可能であるし，独自のプロトコルおよび機構の開発のもとにもなる．PC ルータを用いた閉じた環境では高速シリアルや ATM のハードウェアを用いて RSVP やトラフィック制御機構は動作することは実証されているので，広域環境で発生するトラフィックまたはこれらの機構のスケーラビリティを調査する予定である．

さらなる発展としては，現状では RSVP 対応ルータは専用回線で接続されているが，ATM 技術，および cell switch router との組合せも考えられる．このような場合，NOC の構成の再検討，さらに技術の統合化が必要とされる．

その他に評価実験以外にもリアルタイムトラフィックを発生するアプリケーションの開発などが考えられる．現状では，RSVP 対応しているものは少ない．また，これらのアプリケーションを起動するのは当然末端のユーザになるわけだが，資源予約をどの程度個人に許すか，組織に許すか，というポリシに関する研究開発の基盤にも利用していく．

第 2 章

RSVP の現状

RSVP[?] は、Resource ReSerVation Protocol の略であり、ISI および PARC によって提案されているインターネット用の資源予約シグナリングプロトコルである。現在、Version 1 Functional Specification がインターネットドラフトとして提出されている。この 1 年間はプロトコル自体についてはあまり大きな変更点はなく、ようやく仕様が細部まで確定してきたという状況である。そのためさまざまな組織から実装が出てきており、これからは徐々に普及していく段階に入ると思われる。

しかし RSVP 自体は単にシグナリングをおこなうだけであり、実際に資源予約を実現するためには、ルータが RSVP のメッセージを受けて、きちんと帯域を確保できなければならない。よってルータには、RSVP メッセージを処理できることと、帯域を確保するためのトラフィック制御機構を実装することの両方が要求される。こういった点が RSVP の普及が遅れている原因であると考えられる。

2.1 インターネットドラフト

現在、RSVP の仕様のもっとも新しいインターネットドラフトは、draft-ietf-rsvp-spec-14 である。1996 年の 11 月に提出されたもので、この数ヵ月間 RSVP の仕様のドラフトは更新されていない[1]。

rsvp-spec-14	Nov 6, 1996
rsvp-spec-13	Aug 13, 1996
rsvp-spec-12	May 6, 1996

表 2.1: RSVP のドラフト

1997 年の 4 月上旬に IETF の IntServ(Integrated Services) WG と RSVP WG からの要望を受けて、IESG から表 2.2と表 2.3 に示す多数のドラフトに対して Last Call がかけら

[1] 本原稿の提出直前になって spec-15 が発表された

れた。それを受けて本稿執筆時の 5 月頭に draft-ietf-rsvp-spec-15 が発行されたが、以前の版のドラフトから更新された点は細かい部分に限られている。また他のいくつかのドラフトも更新される予定はあるが、最終的には大半のドラフトが、近いうちに RFC として発行されることが予想される。

rsvp-spec-14	Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification
rsvp-md5-02	RSVP Cryptographic Authentication
rsvp-mib-05	RSVP Management Information Base
berger-rsvp-ext	RSVP Extensions for IPSEC Data Flows
intserv-rsvp-use-01	The Use of RSVP with IETF Integrated Services
intserv-ctrl-load-svc-04	Specification of the Controlled-Load Network Element Service
intserv-guaranteed-svc-07	Specification of Guaranteed Quality of Service
intserv-mib-05	Integrated Services Management Information Base
intserv-guaranteed-mib-03	Integrated Services Management Information Base Guaranteed Service Extensions

表 2.2: Proposed Standard RFC になるドラフト

rsvp-intserv-analysis-00	Resource ReSerVation Protocol (RSVP) – Version 1 Applicability Statement
rsvp-procrules-00	Resource ReSerVation Protocol (RSVP) – Version 1 Message Processing Rules
intserv-svc-template-03	Network Element Service Specification Template
intserv-charac-02	General Characterization Parameters for Integrated Service Network Elements

表 2.3: Informational RFC になるドラフト

2.2 ドラフトの変化

表 2.1に示したドラフトの変更点は、ほとんどが細かい修正や用語の変更であり、プロトコル自体の仕様の変更は殆んど見られない。しかし RSVP の仕様に関して、昨年度からの最も大きな変更点としてあげられるのは、IntServ WG による intserv-rsvp-use というイン

ターネットドラフトの発行である。IntServ WG はさまざまな QoS のパラメータを、ネットワーク機器などに依存しない形で統一的に取り扱うための形式を規定している。RSVP を用いてネットワーク帯域を予約することを考えた場合、当然ながらさまざまな局面で QoS パラメータを扱わなければならない。つまりこのドラフトでは RSVP と IntServ WG の定める QoS を統一的に扱うための仕様が規定されている。具体的には、RSVP のオブジェクトのうち、FLOWSPEC と ADSPEC、そして SENDER_TSPEC の 3 つのオブジェクトにおいて、IntServ WG の定める Controlled-Load サービスと Guaranteed サービスを取り扱うためのフォーマットの詳細を規定している。このドラフトの最新版は intserv-rsvp-use-01 であるが、これは以前の版である intserv-rsvp-use-00 からオブジェクトのフォーマットが大幅に変更された。このことから、どちらのドラフトに基づいて実装されたかによって、互いに RSVP による通信できないという混乱が一時的ではあったが発生した。しかし最近では、殆んどの実装が intserv-rsvp-use-01 に基づくように修正されてきたので、このような互換性に関する問題はなくなりつつある。

その他の大きな変更点としては、ドラフトの RSVP メッセージ処理規則に関する章全体が、rsvp-procrules-00 という別のドラフトに分離したことと、rsvp-spec のドラフトの付録に用語集が新たに加えられたことがあげられる。

2.3　RSVP の実装

本節では RSVP の 1997 年 5 月時点での実装の状況をまとめておく.

RSVP のプロトタイプ実装は ISI(USC Information Sciences Institute) から無償配布されている. 最新のバージョンは 4.1a4 で, RSVP の WWW ページから入手可能である (`http://www.isi.edu/dev7/rsvp/`).

この ISI のバージョン 4.1 の実装は rsvp-spec-14 と intserv-rsvp-use-01 という最新の仕様に基づいている。そのため intserv-rsvp-use-00 という古い仕様に基づいたバージョン 4.0 の実装とは互換性がない。

ISI の実装には RSVP プロトコルを解釈するデーモンプログラム (rsvpd) と、アプリケーションから RSVP プロトコルを利用するためのライブラリ、そしてテスト用に rsvpd と通信するツールが含まれている。ここで注意しなければいけないのは、ルータにおいて実際にトラフィックを制御して帯域を確保するための機構 (パケットスケジューラ) は含まれていないという点である。

RSVP のプロトタイプ実装は SunOS 4.x が動作する Sun ワークステーション上で開発されたが、最近ではさまざまなプラットフォームに移植されており、現時点では以下にあげるプラットフォームが正式にサポートされている。

- PC 互換機 FreeBSD 2.x
- Sun ワークステーション SunOS 4.x

- Sun ワークステーション Solaris 2.4
- SGI ワークステーション IRIX 6.2
- SGI ワークステーション IRIX 5.3(ホストサポートのみ)
- DEC Alpha ワークステーション Digital Unix(旧称 OSF/1) 3.0(ホストサポートのみ)

また以下のバージョンの IP マルチキャストに対応している。

- ルータとして利用する場合

  pruning に対応した IP マルチキャストをサポートしている。具体的には PARC の IP マルチキャスト 3.5 に、95 年 6 月のアップデートを含んだカーネルおよび、バージョン 3.6 以上 (最新のバージョンは 3.8) の mrouted が必要である。

- ホストとして利用する場合

  どのリリースの IP マルチキャストもサポートしているが、IP マルチキャスト 3.3 以前のリリースの場合には、UDP によるカプセル化が必要である。

以下に ISI 以外の実装、もしくは ISI の実装を基にしているが、ISI から配布されていないものをあげる。

Solaris 2.4 以上の RSVP は Sun から CBQ(Class Based Queueing) というパケットスケジューラとともに無償配布されている。現在のバージョンは 0.4.9 で ISI の実装のバージョン 4.1a3 を基にしており、`ftp://playground.sun.com/pub/rsvp/SolarisRSVP.0.4.9.tar.gz` から入手可能である。なおこのパッケージは基本的に ISI の実装のバージョンがあがる度にリリースされている。ただし CBQ 関連のカーネルモジュールについては Solaris 2.5 以上しかサポートしておらず、CBQ のソースコードも以前は公開されていたが、最近はバイナリのみが配布されている。また現在のところ CBQ が対応しているネットワークデバイスは、10M もしくは 100M のイーサネット (be, le, qe, hme) と、Sunlink 3.0 による同期シリアルインターフェイスのみである。

また、WIDE プロジェクトが開発した、4.4BSD 系の OS 用にトラフィック制御機構を実現するための枠組を、FreeBSD 2.2.1-RELEASE 上に実装したものが配布されている。これについての詳細は 4節に記されている。サンプルとして実装された CBQ が動作し、ISI の RSVP との連携も確認されている。

Linux 2.0.18 上に ISI の実装のバージョン 4.1a3 を移植したものが、`http://mrpark.kyungpook.ac.kr/rsvp.html` から入手可能である。Linux にはトラフィック制御の機構が実装されていないため、現在のところは rsvpd が動作するだけである。しかし現在開発中の Linux バージョン 2.1 上には単純なトラフィックシェイパーが実装されており、さらに CBQ も移植される予定がある。

他にも DEC 社は独自に自社の Digital UNIX 4.0 用に、ISI の実装に基づかない RSVP とパケットスケジューラの実装を進めている。パケットスケジューラの機構に関しては明らかにされていない。この実装は IPv4 と IPv6 の両方をサポートしていて、近い将来 DEC 社から公式にアナウンスされる予定である。

Windows 系の実装においては、まず Winsock 2.0 に"Generic QoS Mapping"が制定された。これは主にアプリケーションと RSVP を連携して利用するための仕様である。開発中の Winsock 2.0 については `http://www.stardust.com/`を参照されたい。RSVP の実装としては FTP Software 社が ISI のバージョン 4.1a1 を基にして Winsock 2.0 上に実装したものがあり、Windows 95 上で動作する。さらに Precept Software 社が全く新しく実装したものもあり、rsvpd に相当するエージェントと RPC ベースの API を有し、さらにエージェントは GUI で操作できるというものがある。これは Windows 95 と Windows NT 上で動作する。詳細は `http://www.precept.com/`から辿ることができる。またインテル社も Winsock 2.0 ベースの RSVP 開発キットの評価版 (1ヵ月利用可能) を配布しており、こちらの詳細は `http://www.intel.com/ial/rsvp/`から辿ることができる。

CISCO 社製のルータにおいては、IOS のリリース 11.2 から RSVP がサポートされている。IOS の 11.2(1) から 11.2(5) までは rsvp-spec-14 と intserv-rsvp-use-00 に基づいているため、現在のところ最新の ISI の実装であるバージョン 4.1 とは互換性がなかったが、11.2(5.1) 以降の IOS は intserv-rsvp-use-01 に準拠したため、ISI の実装と相互運用できるようになった。なお帯域を確保するためのトラフィック制御機構には WFQ(Weighted Fair Queueing) が用いられているが、WFQ が対応しているネットワークデバイスは現時点では同期シリアルインターフェイスのみである。

Bay Networks 社のルータでは現在のところ rsvp-spec-08 に基づいたものがサポートされているらしいが、近い将来には最新の仕様である rsvp-spec-14 に対応する模様である。残念ながら詳細については不明である。

2.3.1　ISI の実装の今後

今後の ISI のプロトタイプには以下の機能の実装が予定されている。

- IPv6 のサポート
- ネットワーク I/O 部分の分離
- rsvpd とデータリンク層のインターフェースの抽象化
- 診断 (diagnostic) メッセージのサポート
- IP セキュリティのサポート

2.4 RSVP 対応アプリケーション

ISI の RSVP パッケージには、プログラミングインターフェースに関する文書も同梱されている。これは Sun OS および BSD 系の OS のための文書である。この RSVP API は 1997 年 4 月の IETF 会議において、RSVP WG としては RFC 化を目指さないことが合意された。RSVP API(通称 RAPI) に基づいたサンプルアプリケーションは、RSVP パッケージとは別に ISI から配布されている。サンプルとして配布されているのは vic と tkrsvp であり、これらのサンプルには rsvpd と連動して RSVP を用いて帯域を予約する機能が付加されている。

vic とは主に MBone 上で広く利用されているマルチキャストに対応したビデオ会議ツールである。tkrsvp とは RSVP に対応していないアプリケーションに対して、RSVP を利用して帯域を予約する機能を提供するためのフロントエンドプログラムである。現在のところ、この tkrsvp と連携して RSVP を利用することができるアプリケーションは vat というマルチキャスト対応の音声会議ツールだけである。

RSVP が普及するためには、RSVP による資源予約を必須とする魅力的なアプリケーション、いわゆるキラーアプリケーションが登場する必要があるが、残念ながら現時点では、RSVP に対応したアプリケーション自体が殆んど存在していないという状況である。RSVP 対応のルータや高帯域のネットワークの普及に伴い、将来的には RSVP に対応したアプリケーションは増えてゆくと思われる。

2.5 IETF では

IETF における RSVP WG の活動は現状では一段落しているといえる．個々数回の IETF の集会では近いうちに RFC 化されると思われる draft の新しいバージョンの変更点についてしつこいように発表されていた．IETF の集会は年に 3 回行われるのだが，ここでは前回の報告書からの IETF の RSVP WG の活動，および RSVP に関連する WG の活動について簡単に述べる．

前述もしたが，RSVP WG では今年度も ID に変更点を加え，RFC になるような文書構成がやっと整ったようだ．大きい変更点としては，主となる Functional Specification ドラフトから RSVP のメッセージ処理機構の文書が取り除かれたことがあげられる．またその前は Transport Area director からの要請で，現在どの IETF WG で取り上げられているセキュリティサポートに関する ID の Functional Specification と一緒に last call しなければいけなくなったこともある．

1996 年の 6 月 Montreal で開催された集会においては RSVP Version 2 という話題も発表されていたが，結果的にはここで挙げられていた新たな項目は現状で採用されているものは多い．ただ，IPv6 との統合化の話題に関してはまだあまり取り挙げられてはいない．また実質的に int-serv WG は休止することも決まった．committed rate service という新た

な service の提案も見られたが，既に提出されている guaranteed service と controlled load service の ID および RSVP との統合化用 ID を実現した段階で一旦休止となった．実際に，このときから IETF では int-serv WG の時間枠は用意されていない．

また，12 月の IETF では，はじめて QoS routing の BoF が開催され，その後 WG として活動していくことが決まった．RSVP WG では，ドキュメントに対する変更点などに加えて，新たに整理されたポリシマネージメント関連の ID の発表が行われた．また，この他にも issll WG は ATM，802 系，低速リンクと個々のデータリンクに分かれてミーティングは行われた．

1997 年 4 月に Memphis で行われた IETF における RSVP WG のミーティングでは，やはりポリシまわりの発表があった．この結果からは，ポリシに関する話題は資源予約のみに当てはまるものではなく，他の WG との深い関係もあるので，RSVP WG 内の作業ではなく，今後新たな WG を作り，別の場で活動すると思われる．

また，現在一番注目されていることは，ルータはフロー情報を管理する必要があり，広域で RSVP などを利用する場合，この情報のスケーラビリィは維持することができないという問題点である．そこで，何とか flow aggregation を行うことができないか議論が行われている．最後に気になる点としては，今度新たに RSVP の実際的応用に関する ID が提出された．この発表が行われる際には，integrated service が十分にサポートされてない場合などの対応の方法，また RSVP をサポートしてないルータを中継するためのトンネリング文書の復活などが挙げられる．インターネットの全てのノードが RSVP 対応になる可能性は低いと思うが，やはりこのような互換性を考慮した文書が重視されるのは IETF の特徴だと思う．

第 3 章

広域環境における RSVP 実験評価

3.1 はじめに

広域環境では実運用環境下に置かれ，作為的でないトラフィックが予想外の影響を与える可能性がある．本章では，広域環境における RSVP の動作，および他の通信の及ぼす影響について調査した実験結果について述べる．

3.2 評価環境

全ての実験評価環境は以下の構成を基本とする．

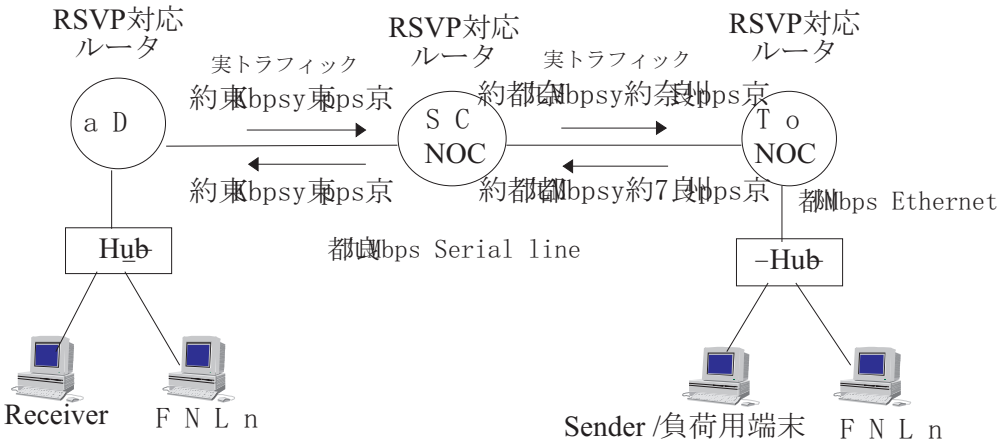

図 3.1: 実験ネットワーク

1. Sender，Receiver 端末

   - Hardware : Sun Sparc 20(Super Sparc 75Mhz)

- Software : Solaris RSVP 0.4.6 (Merge with ISI4.0a8)

2. RSVP 対応ルータ[1]

   - Hardware : Cisco4700
   - Software : IOS v.11.2

3. ルータ間の実トラフィック

   - 奈良 NOC →東京 NOC : 約 1.1 Mbits/sec（約 750 packets/sec）
   - 奈良 NOC ←東京 NOC : 約 1.4 Mbits/sec（約 450 packets/sec）
   - 豊洲→東京 NOC　　: 約 2 Kbits/sec　（2 packets/sec）
   - 豊洲←東京 NOC　　: 約 2 Kbits/sec　（2 packets/sec）

3.3　評価項目

WAN 環境での RSVP の動作と効果を検証するため，実トラフィックある状態で以下の測定を行う.

1. RSVP 動作の検証
   ・RSVP の動作の観察

2. 予約帯域に関する検証
   ・予約帯域を変えたときの RSVP の動作の観察

3. パケット数とトラフィックに関する検証
   ・ルータから送信されるパケット数とトラフィックが変化したときの RSVP の動作の観察

3.4　評価結果

3.4.1　RSVP 動作の検証

1. 測定条件：以下の条件下で RSVP を実行した.

[1]Cisco ルータによる RSVP は，シリアル回線のみの対応となっており，WFQ(Weighted Fair Queueing) というアルゴリズムでスケジューリングを行っている.

- リザーブメッセージのパラメータ
  Service Type : Controlled Load
  Token Bucket Rate : 320Kbps(40Kbyte/sec)
  Token Bucket Size : 40Kbyte
  Smallest Minimal Policed Unit : 20byte
  Largest Maximal Packet Size : 1500byte
- Cisco ルータで確認できた予約パラメータ
  Average Rate : 1064Kbps
  (Bucket Size) : 133Kbyte
- 中継ルータ数: 3 台
- 帯域予約対象のパス種類: TCP
- 帯域予約対象外のパス種類: TCP
- 評価環境概要図は図 3.2に示す.

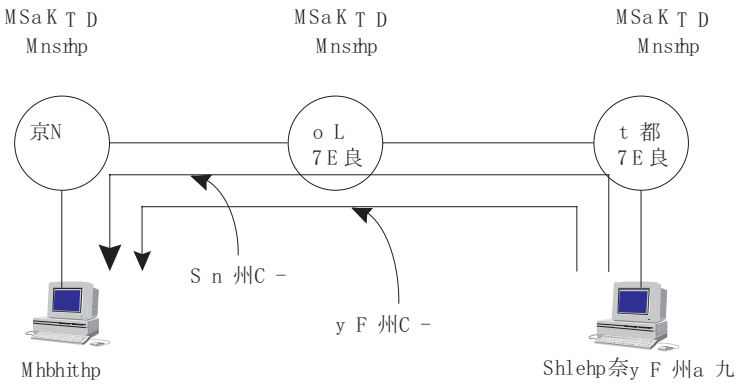

図 3.2: 評価環境概略図

2. 測定方法：ネットワークベンチマークシステム ttcp[2]を用いて，以下の手順でテストデータを送信した．また，tcpdump でパケットをモニタし，そのデータを基にシーケンス番号の変化やジッタを測定した.

   (a) 負荷用パス (Not Reserved Path) を，Sender から Receiver に向かって送信する.

   (b) しばらくして，測定用パス (Reserved Path) を送信する．これにより， Not Reserved Path は Reserved Path からの影響を受ける.

[2]Ballistics Research Laboratory の Mike Muuss によって作成された．TCP/IP および UDP/IP でデータやファイルの転送速度を計測できる．古くから TCP やネットワークの評価に利用されている.

3. 測定結果
   測定対象は負荷用パス (Not Reserved Path) 及び測定用パス (Reserved Path) の送信データである．測定結果は図 3.3, 3.4, 3.5 に示すグラフの通りである．

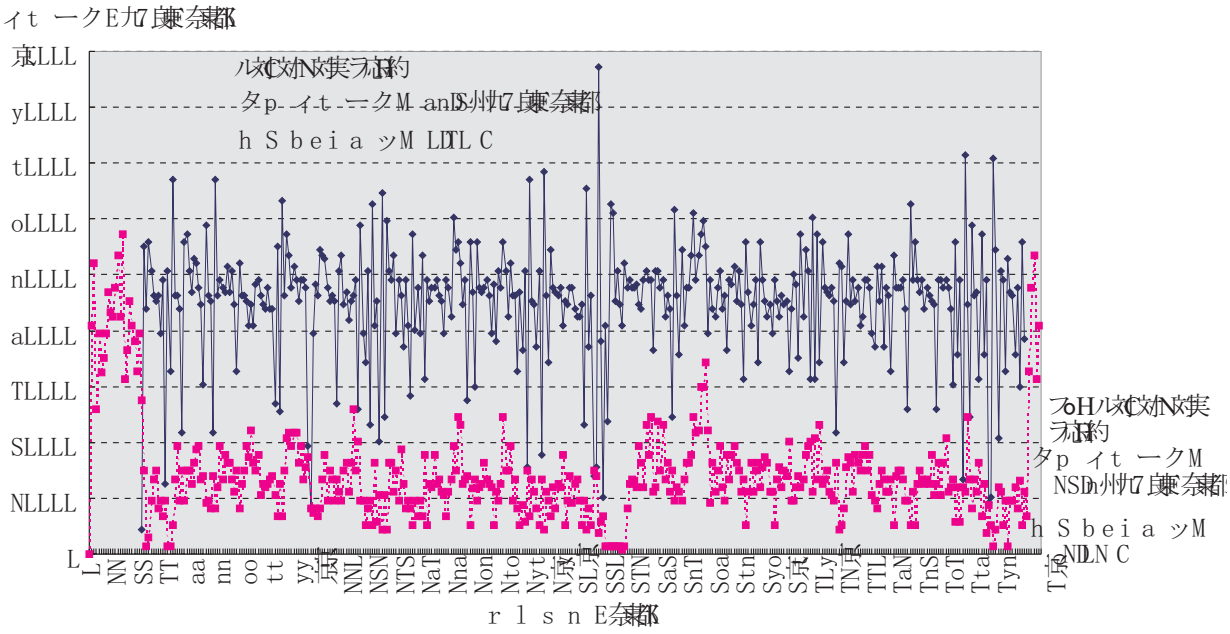

図 3.3: 転送速度の変化 (予約帯域 40KByte/sec)

- 転送速度の変化：1 秒間あたりの転送速度 (Byte/sec) を表す.
- シーケンス番号の変化：Receiver がパケットを受信したときの，時間とそのパケットのシーケンス番号を表す.
- ジッタの分布図：ジッタを {パケットの受信間隔 − パケットの送信間隔} とする．ジッタの分布を相対度数 (%) で表す．[3].

4. 考察

5. 帯域予約
   予約帯域 40Kbyte/sec に対して Reserved Path の平均転送速度が 45.2Kbyte/sec である．また Not Reserved Path より明らかに帯域割当てが優先されているので，きちんとリザーブされていると思われる (図 3.3 と 3.4 を参照).

6. パケットロス
   Not Reserved Path のほうが Reserved Path よりパケットロス率が大きい．このとき

[3](ジッタ, 割合)=(5, 1.25) は，0< ジッタ< 5ms の割合が 1.25%であることを示す.

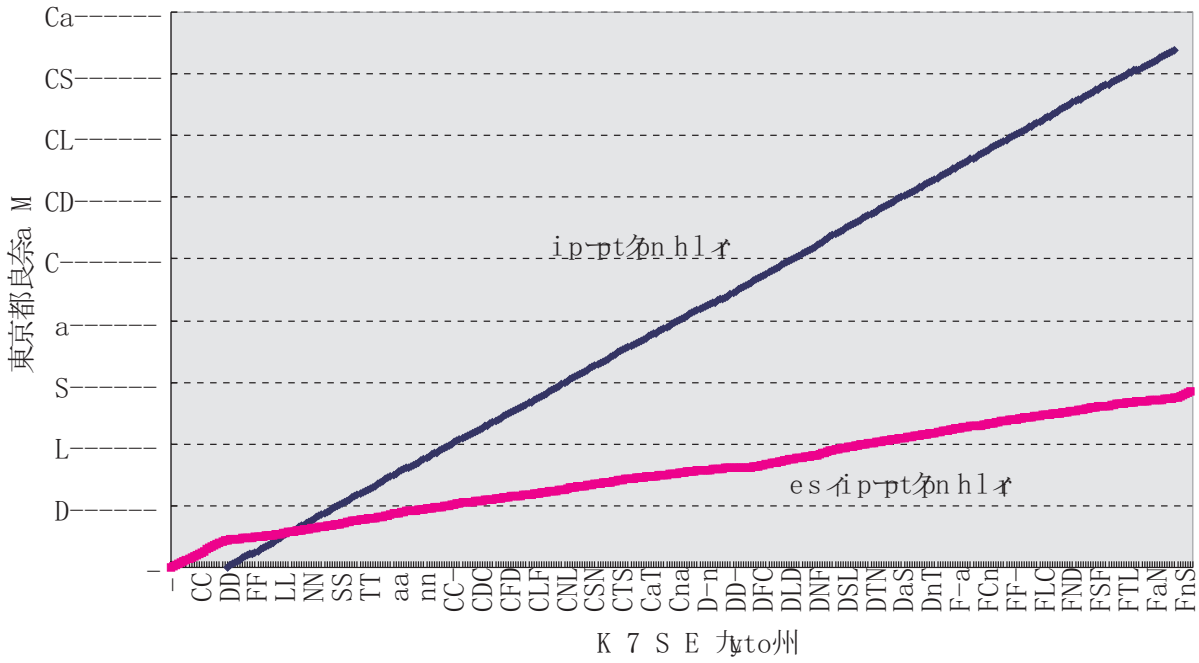

図 3.4: 送信パケットのシーケンス番号の変化 (予約帯域 40KByte/sec)

の Window Size は終始 8760Bytes であった．そのため，ウィンドウ機能による輻輳制御ができず，利用帯域の小さな Not Reserved Path の方がパケットロスが多いと思われる (図 3.3 参照).

7. ジッタ

   Reserved Path のほうが Not Reserved Path よりジッタが小さい．これは Reserved Path のほうはルータ内で WFQ により優先的に配送されるためであると思われる．また，ジッタが－ 60ms 以下や 60ms 以上になるのは，Sender からバースト的にパケットが送信されるので送信間隔が極端に短かったり，長かったりするためである (図 3.5 参照).

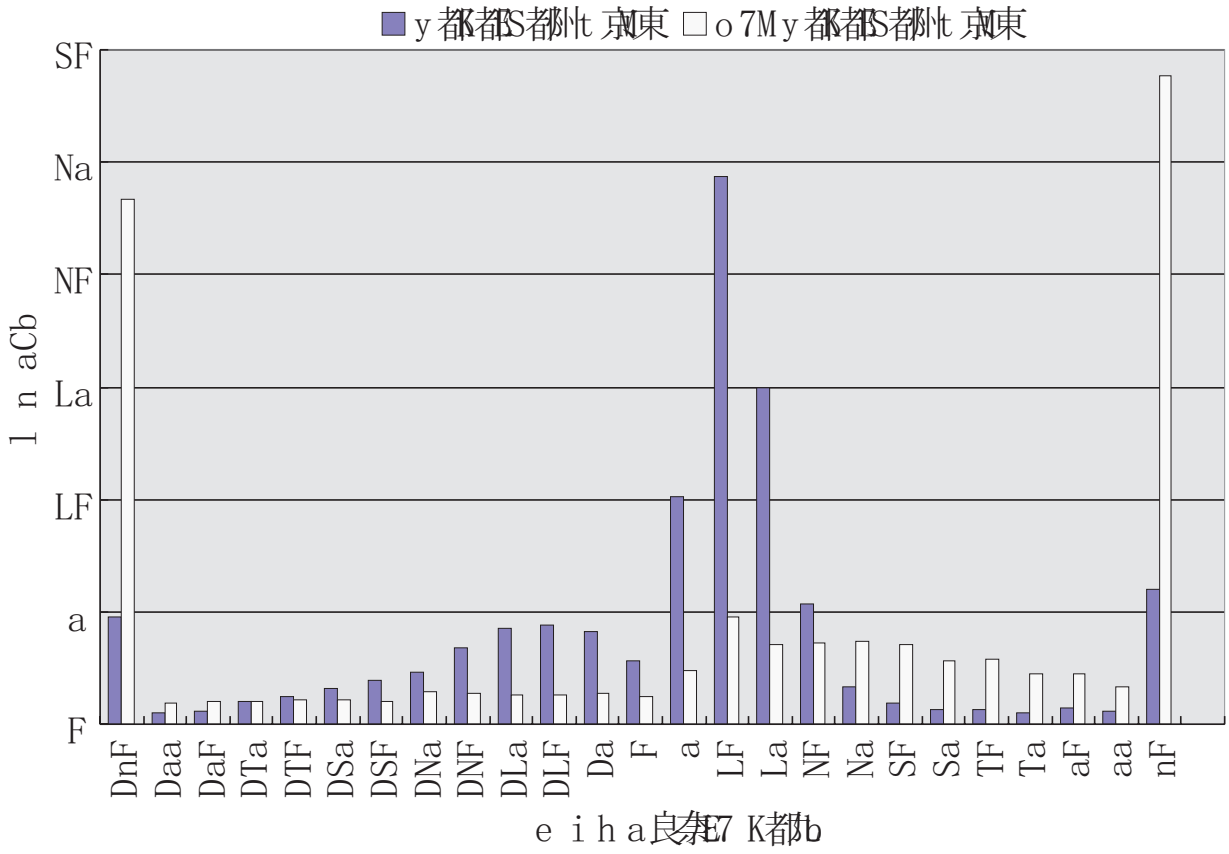

図 3.5: ジッタの分布図 (帯域予約 40KByte/sec)

3.4.2 予約帯域に関する検証結果

1. 測定条件
   以下の４通りをリザーブメッセージのパラメータに設定した.

	Service Type	Token Bucket Rate	Token Bucket Size	Smallest Minimal Policed Unit	Largest Maximal Packet Size
Type1	Controlled Load	1064 Kbps	133 Kbyte	20 byte	1500 byte
Type2	Controlled Load	640 Kbps	80 Kbyte	20 byte	1500 byte
Type3	Controlled Load	320 Kbps	40 Kbyte	20 byte	1500 byte
Type4	Controlled Load	160 Kbps	20 Kbyte	20 byte	1500 byte

   その他の設定は 3.4.1節の 2 と同様

2. 測定方法
   3.4.1 節の 2 と同様

3. 測定結果

4. 考察

   - 平均転送速度の変化
     予約帯域が大きくなるにつれて，実際の平均転送速度も大きくなる．しかし，予約帯域 133Kbyte/sec（Type1）と 80Kbyte/sec（Type2）のときの平均転送速度はほぼ同じである．これは両者の Window Size が終始 8760Byte であることを考慮すると，TCP が Receiver からの ACK 待ち状態よる転送制御機能が働いているためであると思われる (図 3.6 参照).
   - パケットロス率の変化
     予約帯域が大きくなるにつれて Not Reserved Path のパケットロス率が大きくなる．この理由は，3.4.1 節の (r) のパケットロスと同様に Window Size が固定になっているのでウィンドウ制御機能が働いていないためである (図 3.7 参照).
   - ジッタの分布の変化
     予約帯域が大きいほど，ジッタが小さい値のところに集中している．これは予約帯域が大きい方がルータ内で WFQ により優先的に配送されるためであると思われる (図 3.8 参照).

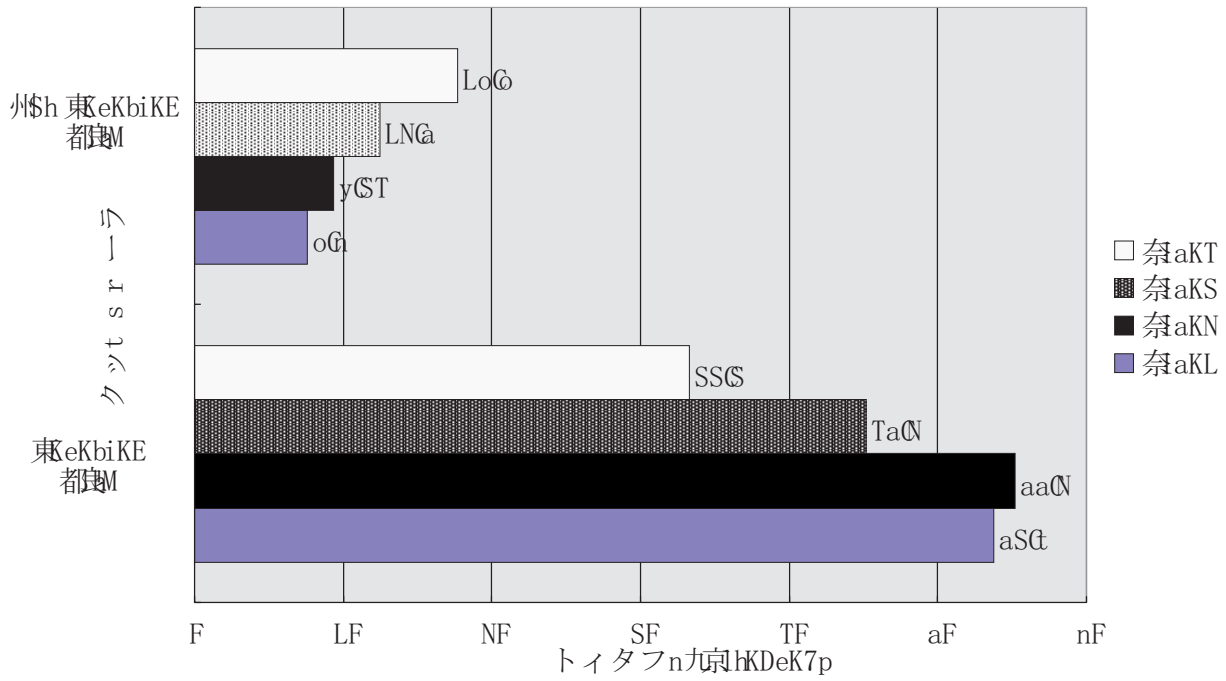

図 3.6: 予約帯域を変えたときの Reserved Path と Not Reserved path の平均転送速度

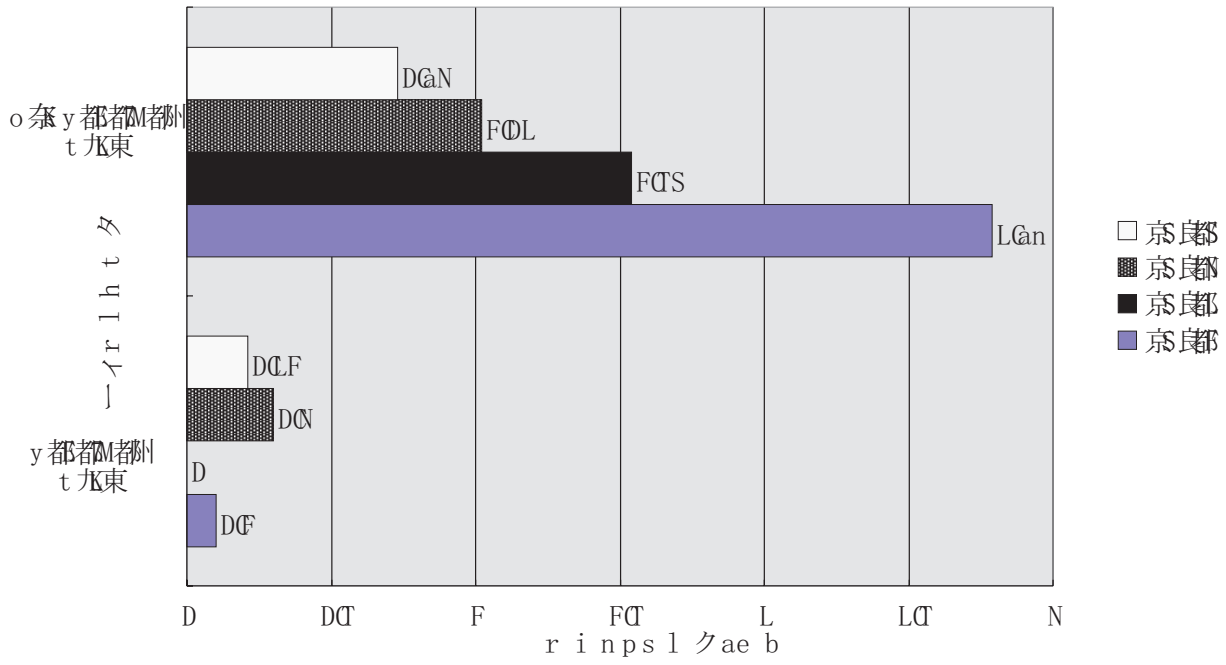

図 3.7: 予約帯域を変えたときの Reserved Path と Not Reserved Path のパケットロス率

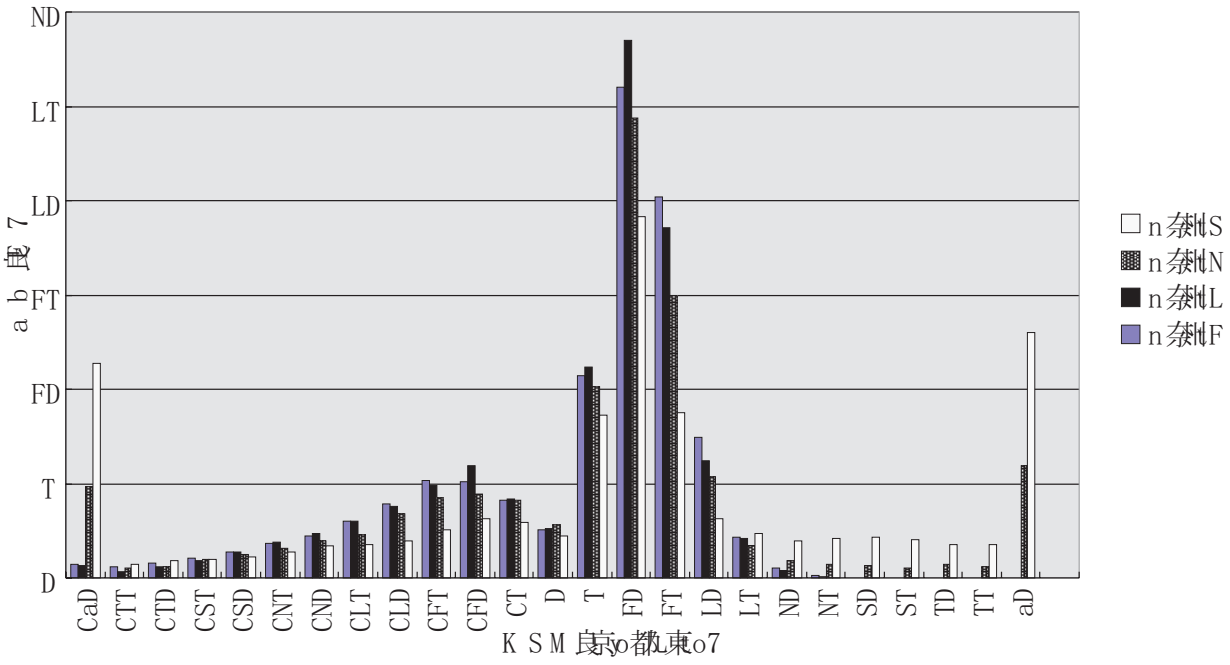

図 3.8: 予約帯域を変えたときの Reserved Path のジッタの分布

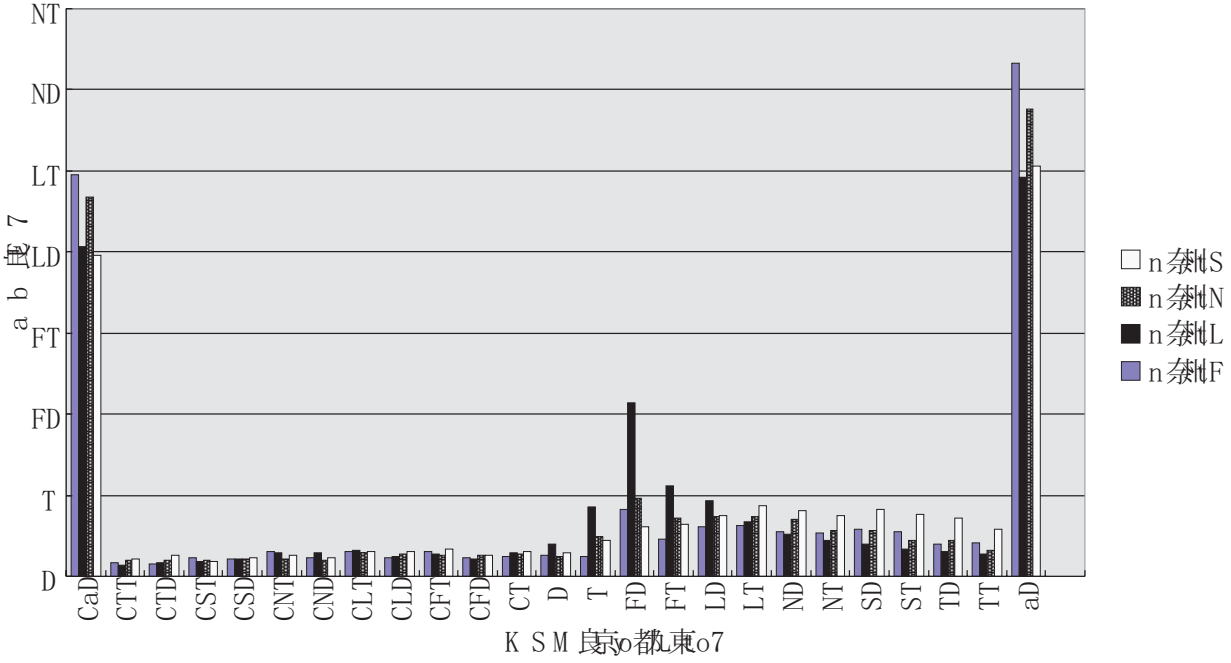

図 3.9: 予約帯域を変えたときの Not Reserved Path のジッタの分布

3.4.3　パケット数とトラフィックに関する検証

1. 測定条件

   - リザーブメッセージのパラメータ
     3.4.2節の 2 と同様
   - 中継ルータ数
     3 台
   - 帯域予約対象のパス種類
     TCP
   - 帯域予約対象外のパス種類
     TCP
   - 評価環境概要図を図 3.10 に示す．今回の測定は WIDE の実験網を使用したので，トラフィックを自由に設定することができなかった．そこで，流れるトラフィック量とパケット数の異なる「東京 NOC →奈良 NOC」と「奈良 NOC →東京 NOC」の実トラフィックを測定の対象にした.「東京 NOC →奈良 NOC」間の単位時間あたりのトラフィック量は多く，パケット数は少ない．逆に,「奈良 NOC →東京 NOC」の単位時間あたりのトラフィック量は小さく，パケット数は多い[4].

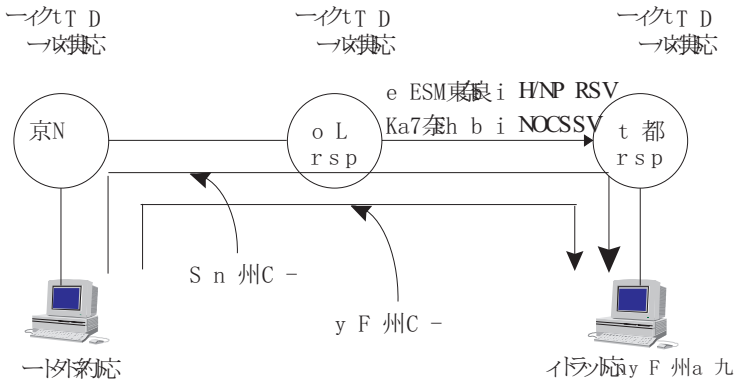

図 3.10: 評価環境概要図

2. 測定方法
   ・3.4.1 節の 2 と同様

[4]「奈良 NOC →東京 NOC」のトラフィックを負荷として利用した結果は 3.4.1 節に示した.

3. 測定結果

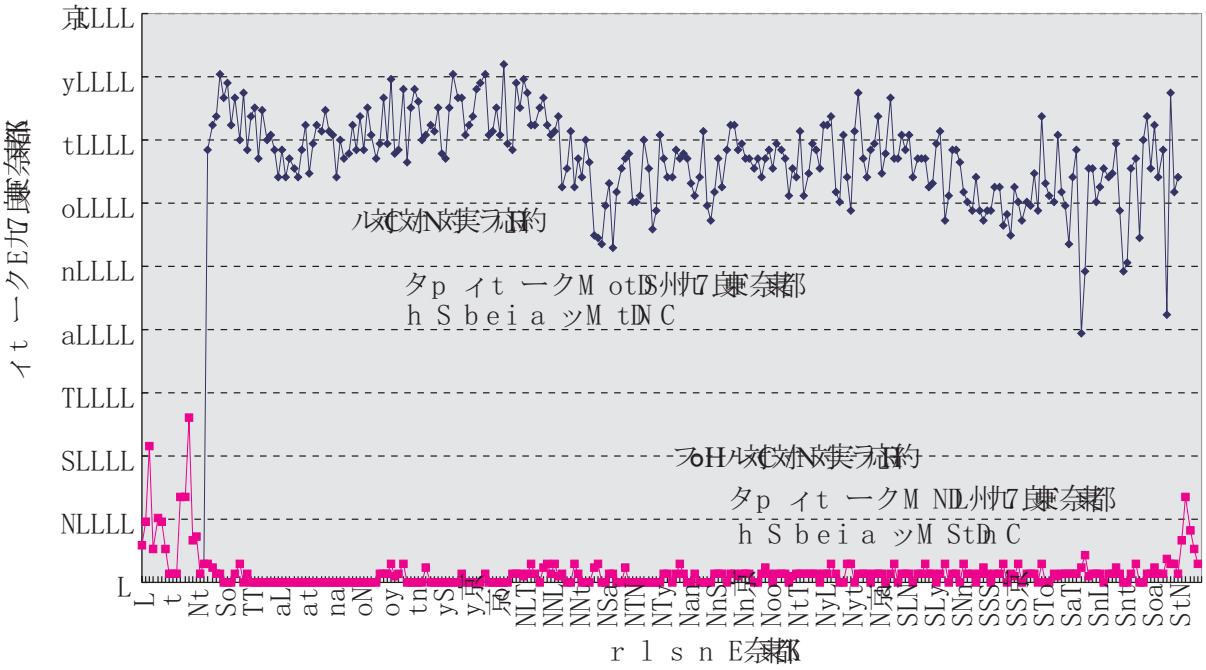

図 3.11: 「豊州→奈良 NOC」に通信した場合の転送速度の変化 (予約帯域 80Kbyte/sec)

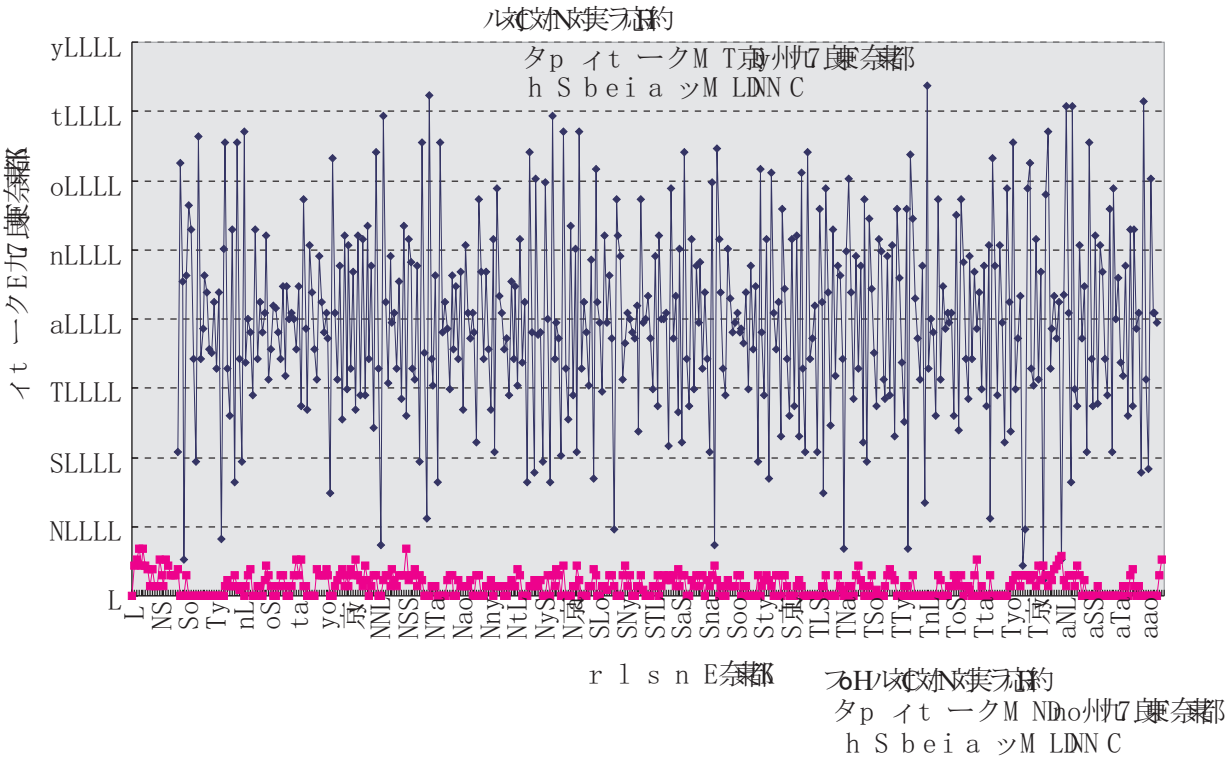

図 3.12: 「豊州→奈良 NOC」に通信した場合の転送速度の変化 (予約帯域 40Kbyte/sec)

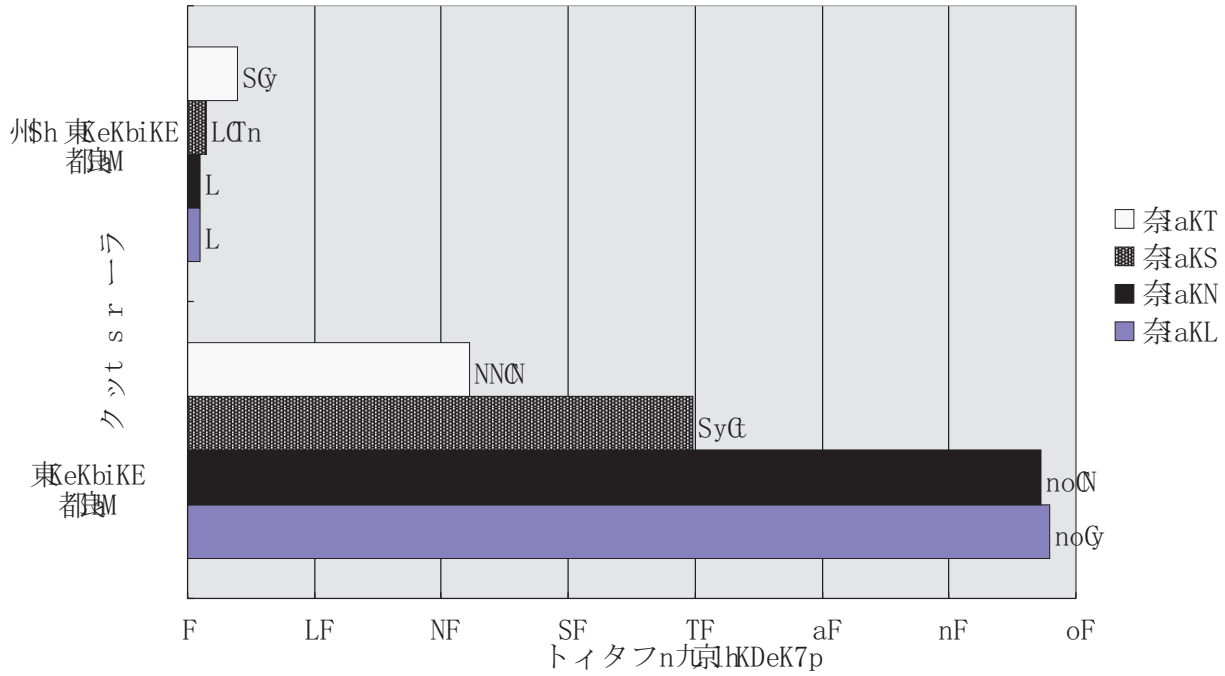

図 3.13: 「豊州→奈良 NOC」に通信した場合の Reserved Path と Not Reserved Path の平均転送速度

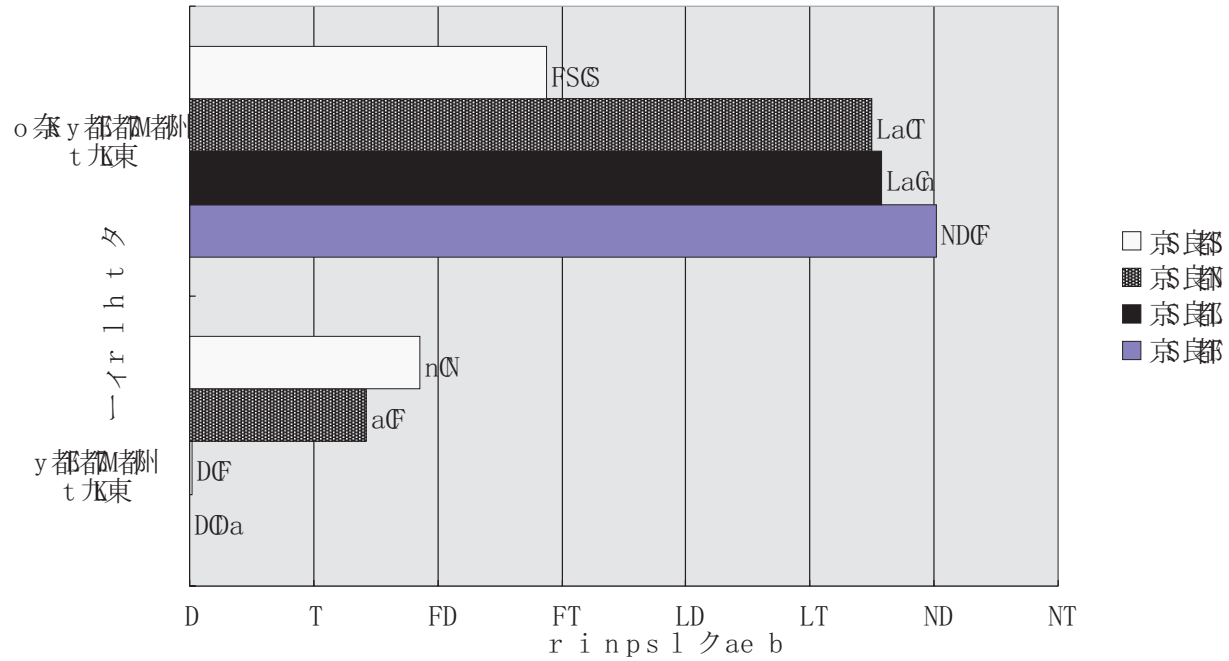

図 3.14: 「豊州→奈良 NOC」に通信した場合の Reserved Path と Not Reserved Path のパケットロス率

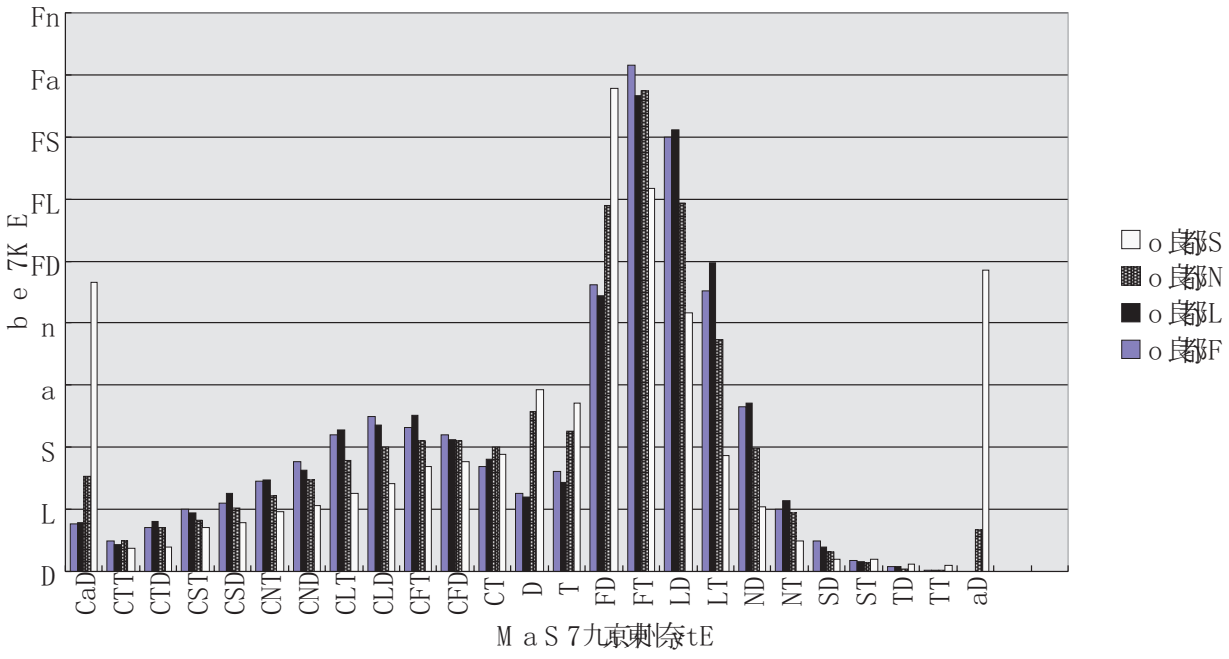

図 3.15: 「豊州→奈良 NOC」に通信した場合の Reserved Path のジッタの分布

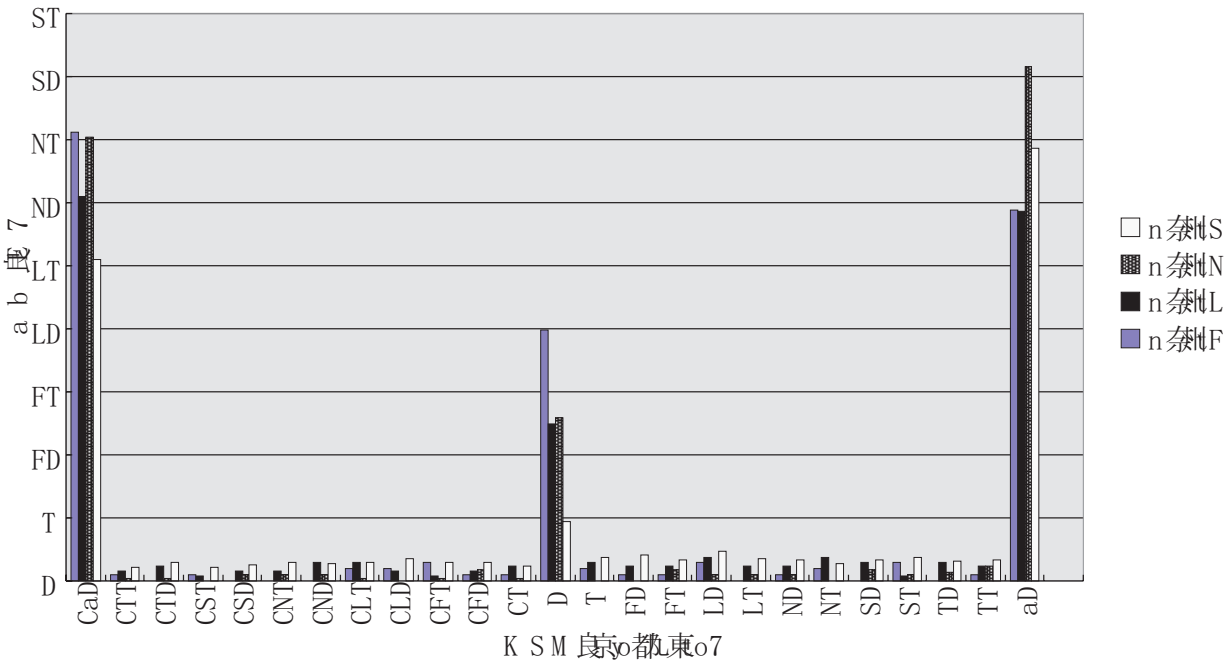

図 3.16: 「豊州→奈良 NOC」に通信した場合の Not Reserved Path のジッタの分布

4. 考察

- 平均転送速度の変化：Reserved Path の平均転送速度が上がっている (図 3.6と 3.13 参照)．このことに関して以下のような理由が考えられる．
  - インターフェースから送信されるパケット数が多いと，WFQ の特徴より，優先的に転送される確率が小さくなる．よって，単位時間あたりに転送されるパケット数の多い「奈良 NOC →豊洲」間の方が転送速度が小さくなる．
  - 端末の送信速度は，Window Size が変わらない場合 ACK が返ってくる時間（RTT）に依存する．RTT は（パケットが Receiver に到着するまでの時間）と（Receiver から ACK が Sender に到着する時間）の合計である．リザーブされているパスの（パケットが Receiver に到着するまでの時間）が同じである場合は,（Receiver から ACK が Sender に到着する時間）が短い方が転送速度は速くなる．よって，リザーブされているときは「豊洲→奈良 NOC」に通信する方が転送速度が速くなると考えられる．
- パケットロス率の変化：Not Reserved Path のパケットロス率が大きくなっている (図 3.7と 3.14 参照)．TCP の Window Size は「奈良 NOC →豊洲」に通信し

た場合と同じなのに，通信帯域は小さくなっているからである[5].

- ジッタの分布の変化：「奈良 NOC →豊洲」に通信した場合とほぼ同じである．(図 3.8,3.9, 3.15,3.16 参照).

[5]「奈良 NOC →豊洲」に通信した場合の方がトラフィック量が多いので，Not Reserved Path の通信帯域は小さくなる．

第 4 章

BSD Unix における代替キューイング機構

4.1 はじめに

ネットワークにおけるフェアな帯域共有、あるいは帯域予約等の実現には、帯域割り当て機能を持つキューイング機構が必要になる。キューイング機構には WFQ[?, ?, ?]、CSZ[?] や CBQ(Class-Based Queueing)[?] 等さまざまなものが提案されている。

しかしながら、現状の BSD 系の Unix の実装には単なるテイル・ドロップの FIFO キューイング以外のキューイングをサポートする機構がなく、このことが新しいキューイング機構を組み込む障害となっており、実際に利用できる代替キューイングがないのが現状である。

そこで、ALTQ という BSD 系 Unix への汎用キューイング・インターフェイスを設計し、実装を行なった。また、実際のキューイング方式の実装として CBQ の移植と WFQ の実装をおこなった。CBQ は RSVP[?] と連動可能であり、また、静的な帯域割り当てによる帯域共有と、RSVP による動的な帯域割り当てが共存可能である。

本プロジェクトは、

1. 種々のキューイング機構実装のフレームワークを提供する。
2. 帯域共有を利用したネットワーク運用のテストベットを提供する。
3. RSVP のためのトラッフィックコントロール・カーネルを提供する。

という３つの側面を持つ。

本稿では、キューイング機構の概要を紹介し、今回実装を行なった代替キューイング機構 ALTQ の設計と、FreeBSD 上のプロトタイプの実装状況を報告する。

プロトタイプはすでに公開されており、WIDE RT-Bone の実験にも用いる予定である。

4.2 キューイング機構

リアルタイム通信あるいは QoS 保証を実現するエンジン部分にあたるのがパケット・スケジューラである。そしてすべてのパケット・スケジューラはなんらかのキュー構造を中心に構成される。厳密にいえばパケット・スケジューラ全体は、キュー構造以外の構成要

素も持つが、一般にネットワークでキューイング機構といえば、キューイング以外の構成要素も含めたパケット・スケジューラの機構をさす。逆に、キューイング機構がパケット・スケジューラ全体を指す場合は、その中のパケット・スケジューリングのみを行なう部分をパケット・スケジューラと呼ぶこともある。本稿ではパケット・スケジューラ全体をさしてキューイング機構と呼ぶ。

キューイング機構にはいくつかの異なる、また、相反する機能が求められる。代表的なものは公平性の実現と帯域保証、遅延保証である。

4.2.1　公平性

現状のベスト・エフォット型のインターネット通信で、キューイングに一番求められるものは公平性の実現である。

通常の FIFO キューのテイル・ドロップによるパケット廃棄では、TCP のような適応型のトランスポート機構は、よりアグレシブな通信と競合すると利用帯域を奪われてしまう。また、TCP 同士が競合した場合も、ラウンドトリップ時間が大きいフローが不利になる問題や、TCP の転送サイクルが同期して不公平が生まれるフェーズ同期問題 [?] が指摘されている。

公平性を実現する手段としては、最後尾のパケットを廃棄する代わりに、ランダムな廃棄を行なう手段 [?] やフェア・キューイング [?, ?, ?] が提案されている。

フェア・キューイングは、Nagle によって提案され、Demers、Keshav、Shenker によって改良された方式で、フローごとに独立したキューを割り当て、それらをラウンド・ロビン・スケジューリングすることによって、公平な帯域割り当てを行なうことが可能である。

フェア・キューイングは、現在までにキューイング関連の多くの研究が参照している重要な方式である。中でも、Parekh はフェア・キューとリーキーバケットを併用すれば、パケットの遅延時間の上限が計算できる事を証明し [?]、その後のリアルタイム通信理論の出発点となった。

4.2.2　QoS 保証

フェア・キューイングのような、特定のフローを他のトラフィックの影響から分離可能なキューイング方式と、アドミッション・コントロールやシェーピングを併用すると、帯域を保証したり、遅延時間を保証することが可能となる [?]。

通常、このような保証を行なうためには、アドミション・コントロールを行なって、アクティブなフローの数を制限する必要がある。また、遅延を制御するためには、送信側でリーキー・バケット等を使ってトラフィック・シェーピングを行なわなければならない。

トラフィック・シェーピングは単に固定レートのフローを作るだけでなく、フローの特性を表すパラメータを与えるという意味で重要である。このパラメータによって、ネット

ワークはフローの特性を知り、アドミッションの判断を行なうことができる。また、実際にフローがこの特性を守っているかどうかポリシングすることができる。

QoS の保証には、決定的保証と統計的保証がある。決定的保証がワーストケースの保証を行なうのに対して、統計的保証は平均値等を取り扱う。

4.2.3 キューイング方式

キューイングの方式には現在までに様々な方式が提案されており、また、それらを組合せること可能である。

キューの構造に関しては、複数のキューを持つもの、プライオリティを持つもの、階層構造を持つものなどがある。

パケット廃棄の方法も、キューの末尾を廃棄するテイル・ドロップ、キューの先頭を廃棄するヘッド・ドロップ、ランダムな廃棄をするランダム・ドロップや、確率的な廃棄をする RED[?] などがある。

また、キュー内部にパケットがある間は送出を続けるワークコンサービングなキューと、そうでない非ワークコンサービングなキューがある。シェーピングを行なうものは非ワークコンサービングである。

さらに、帯域保証、遅延保証に応用可能なもの、そのなかでも決定的保証が可能なもの、統計的保証が可能なもの等あるが、これらは単にキュー構造の問題だけでなく、入力パターンをいかに仮定するかという部分に依存する。

トラフィック制御を行なうキューイング機構を構成する部品には、フローを分類するクラス分け機構、次に送り出すパケットを決定するパケットスケジューラ、バッファ等の資源管理機構、パケット廃棄機構等が必要である。

4.2.4 キューイング理論

ここで、キューイングの実装とキューイング理論の関係について触れておく。ネットワーク・トラフィックを理論解析する際に基本となるのがキューイング理論である、キューイング理論の確立は、コンピュータ・ネットワークの普及と深く関わっている。キューイング理論を体系づけた Kleinrock は、初期の ARPA ネットのトラフィック解析等、広くネットワークへの応用に関する研究も行なっている [?, ?]。

キューイング理論では、パケットの到着を統計的なモデル化を行ない系の動作を解析する。一般に使われるのは、M/M/1 と呼ばれるモデルでポアソン分布に代表される指数的分散を持つ到着分布と処理時間分布を仮定した単一キューのモデル、あるいは、処理時間を一定として簡略化した M/D/1 モデルである。

キューイング理論では、このようなモデルをもとに、キューの長さや遅延を統計的に解析する手法を与える。より高度なキュー構造やより複雑なパケット到着に応用する際にも、このモデルが基本となる。

本稿ではキューイング理論の詳細には触れないが、キューイングが理論的な背景を持ち、理論解析できることは、研究分野として考える上で非常に重要である。

キューイング理論から引き出される重要な結論は、キューの性能が入力量が最大容量に近付くに従って急速に低下するという事実である。つまり、負荷が 90%を越えるあたりから到着レートのゆらぎによってキューの長さの期待値が急増する。このことは、ネットワーク帯域の 90%程度以上を使ってしまうと急速に性能低下する可能性を示唆する。

また、近年、Leland らは、実際のトラフィックが自己相似的な分布を持つことを指摘し、従来のパケット到着モデルに疑問を投げて話題となっている [?]。特にキューイングに与える影響が大きい問題は、従来のモデルではパケット廃棄率を一定以下に押えるためのバッファサイズが計算できるのに対して、自己相似なパケット到着モデルでは、いくらバッファサイズを大きくしてもパケット廃棄を押えることが不可能である点でる。実際のトラフィックは、これらのモデルの両側面を持つと思われ、今後の理論的な解析の進展と実際のネットワークへの応用が期待される。

キューイング理論を実際のネットワークに応用する場合重要なのは、パケット到着のモデル化が妥当なものでなければ、理論的に導きだされたものが意味をなさないことである。

ところが、実際のトラフィックをモデル化するのは極めて難しい。一般にネットワーク・トラフィックが非常にバースト的に発生することは良く知られているが、これをモデル化する手段で広く認知されているものが存在しない。とくに、TCP が作り出すトラフィック・パターンは、モデル化が容易ではない。

また、実際のベスト・エフォット型の通信を行なうネットワークでは、ボトルネックとなる部分に容量をはるかに上回るトラフィックが集中する輻輳状態が頻発する。ところが、キューイング理論が主に取り扱うのは平均入力量が出力容量より小さい場合である。平均入力量が出力容量を上回る場合は処理は破綻し、理論的に解析する余地があまりない。

しかしながら、QoS 保証を行なうようなネットワークでは、入力トラフィックが管理されるため、キューイング理論が直接応用できる。したがって、今後はキューイング理論の応用面での重要度が増してくる。

4.3　ALTQ: 代替キューイング機構

近年、インターネットの拡大にともなってトラフィックが急増し、なんらかのトラフィック制御が必要がとなってきている。さらに、マルチメディア、リアルタイム通信への期待も大きく、高度なキューイング機構の必要が増している。一方で、プロセッサの高速化に伴い、複雑なキューイング処理をパーソナル・コンピュータに組み込むことも可能となってきた。

ところが、現在まで研究レベルではさまざまキューイング方式が提案されているが、ほとんどが理論的な研究やシュミレーションをもとにしたものである。現状、ほとんどのシステムでは単なるテイル・ドロップの FIFO キューイングしか実装されていない。また、

それに代わるキューイング機構を実装しようにも、そのためのフレームワークがないため実装が困難であり、一般に利用できる実装が存在しないのが現状である。そこで、本研究は BSD 系の Unix に、代替キューイングを実装するための汎用フレームワークの提供を試みた。

対象として考えるキューイング機構は、なんらかのパケット・スケジューリングを行なって、帯域割り当てや遅延の制御を実現するものであり、様々なキューイング機構をサポートできるよう考慮されている。

4.3.1 設計

トラフィック制御を行なうのは出力キューであり、入力キューはあまり影響がない。したがって、キュー構造を工夫するのは出力キューのみと考えてよい。BSD 系 Unix では、送出パケットのキューイングは抽象化されたインターフェイス構造体 (struct ifnet) を介して行なわれる。キューイングには、ifqueue という FIFO キュー構造体が使われ、IF_ENQUEUE と IF_DEQUEUE というマクロで操作される。キュー操作を行なうのは、インターフェイス構造体内の、if_output と if_start に登録された関数であり、リンク・タイプ毎に用意される if_output 関数がキューへの格納を、デバイス毎に用意される if_start 関数がキューからの取り出しを行なう。

ここで、代替キューイングを導入する際に問題となるのが、キューに対する操作が単にキューへのパケットの格納、パケットの取り出しだけでないこと、さらに、それらの操作がカーネルないのいくつかの部分で現在のキュー構造を前提にして書かれていることである。入出力以外のキューに関連した操作には、

- キューが飽和するとパケット廃棄する
- パケット送出のためデバイス・ドライバを呼び出す
- キューの先頭にあるパケットを覗き見る

等がある。これらの操作は FIFO キューを想定して作られている。

以下に、一般的な if_output 関数がキューへの格納を行なう部分のコードを示す。

```
s = splimp();
if (IF_QFULL(&ifp->if_snd)) {
        IF_DROP(&ifp->if_snd);
        splx(s);
        m_freem(m);
        return(ENOBUFS);
}
IF_ENQUEUE(&ifp->if_snd, m);
```

```
if ((ifp->if_flags & IFF_OACTIVE) == 0)
        (*ifp->if_start)(ifp);
splx(s);
```

この一連のコードはキューに関連した３つの操作を行なっている。

- IF_QFULL マクロによってキューの飽和をチェックし、飽和している場合はパケット廃棄する。
- IF_ENQUEUE マクロによってパケットをキューに格納する。
- デバイス・ドライバがまだ起動されていなければ、パケット送出のためデバイス・ドライバを呼び出す

現状のパケット廃棄はキューが飽和した場合には、最後に格納しようとしたパケットを廃棄することを前提としているが、一般には、パケット廃棄の判断と、廃棄するパケットの選択はキューイング方式に依存する。

また、非ワークコンサービングなキューでは、キュー内にパケットが存在しても、送出が行なわれるとは限らず、その場合、何らかのタイミングでドライバの送出ルーチンを呼び出す機構を用意する必要がある。

つまり、キューへのパケット格納操作は、パケット廃棄、デバイス・ドライバの呼びだしを伴うが、いずれの動作もキューイング方式に依存するため、これらを統合したインターフェイスを考慮する必要がある。

また、キューの先頭を覗き見る操作は、多くのデバイスドライバがパケットを送出するためのバッファがあるかどうかを確認するために使っているが、その手続きが規定されていないため実装によって手順が異なっていたりする。帯域制御を行なうキューイングでは、キューが複数存在する場合もあるし、また、キューの先頭にあるパケットがかならずしも次に取り出すべきパケットであるとは限らない。したがって、次に送出されるパケットをチェックするための操作も規定する必要がある。

代替キューイング機構の実装は、既存のコードの変更を最小限にする方針をとって検討したが、前記のように現状のキュー操作は十分な抽象化がなされていないため、if_output 関数と if_start 関数の変更が避けられないことが明らかとなった。

とくに、if_start 関数の変更は、デバイス・ドライバの変更を意味するが、既存のすべてのデバイス・ドライバを変更するのは容易でない。そこで、代替キューイングをサポートするデバイス・ドライバとサポートしないデバイス・ドライバが共存できるようにすることによって、必要なドライバへの変更だけですむようにした。

このように、従来の FIFO キューイングと代替キューイングを共存させて、動的に切替可能とすることは、設定の誤り等で代替キューイングに問題が発生しても、従来のキューイングに切替えて使うことができるため、運用上の利便、信頼性の向上、デバッグ効率の向上といった利点もある。

4.3.2 実装

以下に、ALTQ の実装による変更点の概要を示す。変更部分は、ALTQ 定義で識別される。

net/if.h には代替キューイングをサポートするために以下のフィールドが追加された。

```
struct ifnet {
        /*
         * ORIGINAL FIELDS HERE
         */
#ifdef ALTQ
        /* alternate queueing related stuff */
        int     if_altqflags;   /* altq flags (e.g. ready, in-use) */
        void    *if_altqp;      /* queue state */
        int     (*if_altqenqueue)
                __P((struct ifnet *, struct mbuf *,
                     struct flowinfo *, int));
        struct mbuf *(*if_altqdequeue) __P((struct ifnet *, int));
#endif /* ALTQ */
};
```

if_output 関数の変更は 2 箇所。ひとつめは、パケットからクラス分けに必要となるフロー情報を抽出する部分である。この情報はキューへの格納操作に渡される。フロー情報の抽出は、ネットワーク層やキューイング関数の中でもできるが、ネットワーク層で行なうとその情報を if_output 関数に渡すためにインターフェイスの変更が必要となりカーネル内の他の部分への影響が大きい。また、キューイング関数で行なうとリンク・ヘッダを扱わなければならなくなるため、if_output 関数ないでフロー情報を取り出すこととした。

```
#ifdef ALTQ
    struct flowinfo flow;

    /* extract flowinfo before adding link header */
    if (ALTQ_SET_FLOWINFO(ifp))
            altq_extractflow(m, &flow);
#endif /* ALTQ */
```

ふたつめは、実際のキューイングを行なう部分である。ALTQ_IS_ON マクロを使って、代替キューイングが使用中であるかどうか判断し、使用中であれば ifnet 構造体に登録された格納関数を呼び出す。格納関数は、パケット廃棄とデバイス・ドライバの起動も行なう。

```
        s = splimp();
    #ifdef ALTQ
        if (ALTQ_IS_ON(ifp)) {
                error = (*ifp->if_altqenqueue)(ifp, m, flow,
                                               ALTEQ_NORMAL);
                if (error) {
                        splx(s);
                        return (error);
                }
        }
        else {
    #endif
        if (IF_QFULL(&ifp->if_snd)) {
                IF_DROP(&ifp->if_snd);
                splx(s);
                m_freem(m);
                return(ENOBUFS);
        }
        IF_ENQUEUE(&ifp->if_snd, m);
        if ((ifp->if_flags & IFF_OACTIVE) == 0)
                (*ifp->if_start)(ifp);
    #ifdef ALTQ
        }
    #endif
        splx(s);
```

if_start 関数も同様に、ALTQ_IS_ON マクロを使って、代替キューイングが使用中であるかどうか判断し、使用中であれば ifnet 構造体に登録された取り出し関数を呼び出す。

```
    #ifdef ALTQ
        if (IS_ALTQ_ON(ifp))
                m0 = (*ifp->if_altqdequeue)(ifp, ALTDQ_DEQUEUE);
        else
    #endif
        IF_DEQUEUE(&ifp->if_snd, m0);
```

覗き見操作は、第二引数に ALTDQ_PEEK を与えることによって行なう。

```
        /* Sneak a peek at the next packet */
```

```
#ifdef ALTQ
    if (IS_ALTQ_ON(ifp))
            m0 = (*ifp->if_altqdequeue)(ifp, ALTDQ_PEEK);
    else
#endif
    m0 = ifp->if_snd.ifq_head;
```

上記変更を行なったデバイス・ドライバは、以下のようにインターフェイスをアタッチする際に、ALTQF_READY フラグを立てることによって、代替キューイングをサポートすることが識別される。

```
#ifdef ALTQ
    ifp->if_altqflags |= ALTQF_READY;
#endif
    if_attach(ifp);
```

このように比較的簡単な変更でキューイング方式の動的な切替えが可能となる。

また、ユーザレベルからのキュー操作の手順は、キューデバイス (例えば、/dev/cbq) をオープンした後、ioctl を使って、代替キューイングをインターフェイスにアタッチし、 イネーブルする手順をとる。

4.4 キューイング方式のサンプル実装

代替キューイングのフレームワーク上へのキューイング方式のサンプル実装として CBQ[?] と WFQ[?, ?] を実装している。

4.4.1 CBQ

CBQ (Class-Based Queueing)[?] は、ローレンス・バークレイ研究所の Van Jacobson によって提案され、同研究所の Sally Floyd によって解析、シュミレーションが行なわれてきた。

CBQ はその名のとおりクラス構造を基本としている。独立したキューをもつクラスを階層的に構成し、各クラスは上位のクラスから帯域を借りることが可能で、それによってリンク共有を実現できる。つまり、上位のクラスの使用帯域に余裕がある間は自分のクラスの割り当てを越えた帯域使用ができるが、全体の使用帯域が足りなくなってくると自分のクラスの割り当て分まで使用量が制限される。

また、各クラスの使用帯域の最大値の制限や最低値の保証が可能であり、QoS 保証を必要とするサービスに利用できる。

CBQ の実装は以下のような構成要素を持つ。

- クラス分け機構 (classifier) は、入力パケットをパケット・フィルタを使ってクラス分けし、対応するクラスのキューに割り当てる。この際、クラスに設定されたキューのサイズの最大値を越える場合にはパケットは廃棄される。

- パケット・スケジューラは、次にどのクラスからパケットを送出するか決定する。各クラスにはプライオリティが与えられるようになっており、プライオリティの高いクラスからパケット送出が行なわれる。単なるプライオリティ・キューとの違いは、プライオリティが高くても後記の使用帯域制限が効くため、低プライオリティのクラスが枯渇することがないことである。同じプライオリティを持つクラス間では、ラウンドロビン・スケジューリングとなる。

- 推量機構 (estimator) は、各クラスのパケットの送出間隔を測定し、その重み付き平均値を管理することによって各クラスの使用帯域を計測する。クラスの使用帯域が設定された値を越えた場合は使用量超過が検出される。使用量が超過しても借り入れが可能な間は継続してパケット送出ができるが、借り入れができなくなると遅延機構によって使用帯域が制限されることになる。

- 遅延機構 (delayer) は、使用帯域を超過したクラスの使用帯域を制限するために、ある期間パケット送出を差し止める機能を提供する。

CBQ のデザインは実装面を強く考慮してあり、比較的小さいオーバヘッドで機能が実現できる。特に、推量機構とパケット・スケジューラは簡単な計算で、かつ、Unix カーネルのタイマの粒度で各クラスの使用帯域を近似し、送出順序を制御できるよう工夫がされている。

CBQ はローレンス・バークレイ研究所から、NS シュミレータ [?] 用のコードが公開されている。また、UCL がローレンス・バークレイ研究所と共同で Solaris への実装を試み [?]、それを Sun が引き継いで Solaris の RSVP[?] のためのトラフィック制御機構として開発をしている。また、その他のいくつかのベンダが CBQ の実装を試みているようである。

Sun の CBQ の実装は開発初期のものはソースコードも公開されていたが、現在はバイナリによる配布しかされていない。

今回、CBQ の ALTQ への実装は、公開されていた当時のソースコードをもとに行なった。

4.4.2 WFQ

一般に、WFQ(Weighted Fair Queueing)[?, ?] という名前は、Demers らの提案した機構を指す。しかしながら、Demers らの提案のままでは実装が難しい。

オリジナルの WFQ はフローごとのキューを想定しているが、これだとフローの数が増えるとそれに応じて無限のキューが必要となる。実装では、無限のキューを割り当てることは不可能なので、有限のキューにできるだけ異なるフローが同一のキューに割り当て

device	CBQ OFF (Mbps)	CBQ ON (Mbps)
local loop	325.22	310.65
150M ATM	133.76	122.40
100M Ether	32.34	31.54
10M Ether	6.89	6.95

表 4.1: CBQ のスループット

られないようハッシュ関数を使ったマッピングを行なう。この方式は SFQ(Stochastic Fair Queueing)[?] とも呼ばれる。

また、Demers らの提案したパケット・スケジューリングは、ビット単位スケジューリングと等価となるようにキュー内部のパケットを並び替える方法を提案しているが、この方式は実行コストが高く実装に向かない。そこで、パケット・スケジューリングには、Floyd や Shreedhar らが提案した、借り入れ (deficit) を使う方式 [?, ?] をとった。

4.5 現状

代替キューイングの実装は FreeBSD 上に行ない、1997 年 3 月からアルファ・リリースという形で一般に公開している。(`http://www.csl.sony.co.jp/person/kjc/programs.html`)

現在５種類のイーサネットドライバと１種類の ATM ドライバ、また、デバッグ用にローカルループドライバがサポートされている。

キューイング方式では、前述の CBQ と WFQ が実装されている。

表 4.1、4.2に現時点での CBQ の評価データを示す。測定は２台の 200MHz-PentiumPro マシンで netperf ベンチマークを用いて TCP を使ったスループットを測定した。

表 4.1は CBQ を使わない場合と使った場合とのスループットを示す。CBQ には３つのクラスが定義されており、TCP はデフォルト・クラスにマッチする設定になっている。

性能面ではキューイングのオーバーヘッドは思ったより小さい。これは、キューイングの処理が前のパケットの送出時間にオーバーラップするためである。150Mbps の ATM でも MTU が 8K バイトあるため、とんどスループットの低下が見られない。

表 4.2は 10Mbps イーサネット上で CBQ の帯域制限を行なった場合のスループットである。得られる帯域は多少の誤差を含むが、だいたい設定どおりに機能していることが判る。

また、CBQ は ISI による RSVP の実装と連動して動作可能である。そのためのインターフェイスも ALTQ リリースに含まれる。RSVP は、シグナリング部分の実装は ISI から公開されているが、このリリースには実際にトラフィックを制御するトラフィック制御カーネルは含まれない。これまでフリーで公開され利用できるトラフィック制御カーネルが存在しなかったため、今回の代替キューイングを使った CBQ の実装が注目されている。

% bandwidth	10M Ether	150M ATM
10 %	0.96	13.25
20 %	1.98	25.04
30 %	2.87	35.48
40 %	3.69	45.61
50 %	4.42	55.78
60 %	5.28	66.91
70 %	6.08	74.11
80 %	6.91	107.37
90 %	7.55	120.41

表 4.2: CBQ の帯域制限機能

4.6　まとめ

現在まで数多くのキューイング機構が提案、研究されてきたが、利用できる実装が少なく、実際にはあまり利用されていないのが現状である。

そこで、さまざまなキューイング機構を実装するためのフレームワークとして ALTQ という代替キューイング機構を設計、実装し、その上で CBQ と WFQ を実装したプロトタイプを作成した。

今回のプロトタイプで、基本的な機能の動作確認、性能確認、RSVP とのインターフェイス等、当初の目標は達成できたと考えている。今後は、WIDE プロジェクトの RT-Bone における実験等の実際の運用を通じての経験をフィードバックして、より実用的に改良を加えていくつもりである。

また、今回の実装が、WIDE プロジェクト内外で、キューイング、リンク共有、RSVP といった分野での研究の刺激となり、より多くの取り組みが行なわれていくことを期待したい。

第 5 章

実時間通信を実現するための機構の概要 と WIDE から提供できる技術

5.1 Introduction

インターネット等のパケット交換網でマルチメディアコミュニケーションを行うには，さまざまな難しさを伴う．もともとパケット交換網は，実時間性を要しない電子メールやファイル転送などのバッチ形式のデータ指向型トラヒックのために設計されたものであるから，パケット交換網で実時間性を必要とするマルチメディアコミュニケーションを行おうとすること自体がコスト高なことである．

パケット交換網が実時間通信を得意としない大きな原因は，中間ルータの待ち行列長が劇的に変化するので，それに併せてパケット毎の待ち行列遅延が一定にならない，すなわちジター (jitter) が大きいことにある．さらに，その遅延時間の上限を見積もるのが難しく，例えその上限が分かっても遅延時間が大きすぎ，インタラクティブコミュニケーションができないことが多い．

この難しさをさらに上昇させている要因として，End-to-End でみると複数の中間ルータがあって，それぞれの待ち行列遅延が劇的に変化するので，End-to-End の遅延時間の上限と jitter がさらに上昇することが挙げられる．

このように，実時間通信を不得意とするパケット交換網でマルチメディアコミュニケーションを行うには何らかの工夫が必要となる．ここでは，パケット交換網でマルチメディアコミュニケーションを実現する二つの方式に注目し，それぞれの概要をまとめる．最後に，このような状況の中で RT-Bone がどのような研究をすべきかについてまとめる．

5.2 Reactive Control

Reactive Control (Adaptive Control) は，ネットワーク内の交換機メカニズムを変えずに End-to-End のレート制御だけで転送遅延の変化を抑える手法である．この手法は，現在までに幅広く実装/実験されている方式で，多くの運用実績を持つ．インターネットコミュニティでの Reactive Control の実装は，多くの場合 RTP(Real-time Transfer Protocol) およ

び RTCP(Real-Time Control Protocol) を用いた受信端と送信端とのフィードバックによるレート制御を行う [?, ?, ?, ?]. [1]

この方式は，受信端のアプリケーションの機構だけで実現できるので実装が容易である．逆に，ネットワーク内の交換機機構に一切変更を加えないので，品質の高い通信に不向きである．さらに，シミュレーションによると，混雑を検知した UDP による実時間通信がレートを下げるとデータ通信を行っている TCP がレートを上げるので，実時間トラヒックの通信品質は向上しない [?]．このように，インターネットでは Reactive Control の効果が高いとは言えない．

5.3 Quality of Serice Guarantee

Reactive control よりも品質の高いマルチメディアコミュニケーションを実現する機構として QoS 保証方式がある．これは，それぞれのアプリケーションが望むサービスの品質 (Quality of Service) をネットワークが保証する方式である．特に，CBR(Constant Bit Rate) や VBR(Variable Bit Rate) といった実時間通信トラヒックに対して，End-to-End での帯域，遅延時間，遅延時間のゆらぎ (jitter)，ロス率等の保証を与える．これらを保証するためには，異なる 3 つの機構が必要になる [?, pp. 249–250].

1. Traffic shaping. それぞれのコネクションの送信端でフローの振舞に制限を加えて，ネットワーク内で QoS 保証をすべきトラヒックに与える悪影響を軽減する．それぞれの送信端が決められた QoS パラメータを越えたフローを送出しないようにネットワーク側で監視し，制限を加えることを Traffic Policing と呼ぶ．Policing も Shaping の一種と考えることができる．shaping 手法としては leaky bucket が広く使われている．

2. Special queueing schemes. バースティトラヒックの到着に対して，通常の FIFO 待ち行列では実時間トラヒックに対して QoS 保証を与える事が極めて難しい．そこでルータの待ち行列機構に手を入れ，QoS の保証をできるようにする．

3. A setup mechanism. (Admission control algorithms) コネクションセットアップ時に利用者がある QoS の保証を要求し，ネットワークはその QoS パラメータが保証できるかを判断しそのコネクションを受け入れるかどうか決める．現在標準化が進められ，実用段階に入りつつある RSVP(Resource ReSerVation Protocol)[?] はインターネット上におけるこのメカニズムの一つの実装である．

QoS 保証方式は，一部実用化に入りつつあるがまだ残された研究課題も多い．特に，

1. 大規模でスケールする事.

[1] RTP を用いたアプリケーションで代表的なものには vat, vic, ivs 等がある．

2. マルチキャストが実現できる事.

などの難しい問題を抱えている．さらに，QoS 保証の目標として，以下のような事柄が挙げられる [?].

- 低遅延かつ低い遅延のゆらぎ (jitter)
- 従来からのデータ指向型のサービスと実時間サービスを容易に統合できる事.
- ネットワークおよびトラヒックの状態の動的変化に適応できる事.
- 効率が良い事 (バンド幅利用効率，ヘッダオーバヘッド，中間ルータのバッファサイズ等).

ここでは Special queueing schemes に着目し，現在までの提案を整理する．また，この節のまとめで今後の展望について述べる.

5.3.1 Queueing schemes for QoS guarantee

Deterministic guarantee and statistical guarantee

QoS 保証を行う待ち行列機構は，いくつかの方法で分類できる．ここでは，以下の分類法について説明する [?, ?].

1. Deterministic guarantee (hard real-time) すべてのパケットに対して望まれる QoS パラメータを保証する.
2. Statistical guarantee (soft real-time) 定められたある割合以上のパケットに対して望まれる QoS パラメータを保証する.

一般に Deterministic guarantee は，Statistical guarantee と比べて，対象とするマルチメディアアプリケーションにとって効果的ではない．なぜなら，マルチメディアアプリケーションはすべてのパケットがあるデッドラインに確実に到着する事よりも，ほとんど (例えば 99.9%) のパケットがより速く到着する事の方が重要であるのに，Deterministic guarantee を与えると遅延時間が大幅に増加するからである.

どのコネクションを受け入れてよいかを判断する Admission Conrol 部は待ち行列アルゴリズムに依存する．Admission Control の受け入れ可否の判断においては，(1) 既存のコネクションの QoS 保証を存続できるか，(2) そのコネクションの要求する QoS パラメータを満たす事ができるか，の判定が難しい [?]．この判定基準をを呼受け付けガイドライン (A set of guidelines for call admiccion control) と呼ぶ.

Classical studies

Deterministic guarantee における呼受け付けガイドラインについては広く研究されていて，多くの結果が存在する．古典的な結果としては，2 組の結果が有用である．

1 組目の結果は，送信端において全てのフローが leaky bucket によって shaping されている時に，Packet-by-packet Generalized Processor Sharing(PGPS or Weighted Fair Queueing) と呼ばれる待ち行列機構における最悪の遅延時間を解析により導出した [?, ?]．これによって，leaky bucket shaping における WFQ の呼受け付けガイドラインにが作成できる．

2 組目の結果では，全てのフローが “burstiness constraints” と呼ばれる制限を受けている時の，FIFO 待ち行列の遅延時間の限界が解析により導出されている [?, ?]．この “burstiness constraints” は，leaky bucket によってフローを制限することで実現できる．

これら 2 組の結果は重要で，多くの待ち行列機構の解析で利用されている．また，[?] の “burstiness constraints” の考え方は，指数的にバーストを制限するという Exponentially Bounded Burstines に発展した [?, ?].

Integrated CBR and VBR real-time services

しかし，これまでに説明した方式では TCP 等の UBR(Unspecified Bit Rate) 型のトラヒックと，CBR および VBR リアルタイムを効果的に統合できない．CBR および，VBR を効果的に統合することに主眼を置いた研究としては，2 つの研究が有用である．

1 つ目の手法 (CSZ の方法)[?][2]では，CBR リアルタイムに対して WFQ で QoS 保証をし，VBR リアルタイム及び UBR トラヒックに対して遅延の端 (tail) を縮小させる FIFO+を用いる．このように CSZ の方法は，統一的な手法ではない．

2 つめの手法 [?, ?]Class Based Queueing(CBQ) と呼ばれていて，CBQ は統一的な方法でリンク共有とサービスの統合を実現する．CBQ はこれらの目的のために，組織，プロトコル及び，トラヒックタイプ等によって分類する階層クラス構造を用いる．階層クラス構造のそれぞれのクラスは，プライオリティとリンク中の割当バンド幅が指定される．また，それぞれの末端のクラス (leaf class) は中間ルータにおいて各々の FIFO 待ち行列を持ち，CBQ のスケジューリング順序はどのクラスの queue から先にスケジューリングするかを指定する．単純に書くと，クラスのプライオリティ順にスケジューリングし，同一プライオリティクラス間では Weighted Round Robin によって順序を決める．

QoS 保証を行う待ち行列機構が持つべき性質は二つある．一つは低いプライオリティ(つまり，UBR のような best effort トラヒック) のクラスが全く使用できない状態 (starvation) が起こらないことであり，二つ目は，高いプライオリティ(つまり，マルチメディアコミュニケーション用) クラスの遅延時間が保証され，その遅延時間が小さいことである．CBQ では，一つ目に対しては starvation が起こらないと説明されていて [?]，二つ目に対しては

[2] この論文 [?] によって Integrated Services Packet Network(ISPN) という言葉が作られ，サービス統合の理想像として多くの論文で引用されている．

多くの仮定の下で，最高プライオリティクラスの決定的遅延限界が解析により導出されている [?].

Recent studies

この節では QoS 保証を行う待ち行列機構についての，最近の興味ある研究をまとめる．

CSZ の方法および CBQ の他に，ATM についての最近の研究 [?] によると，VBR,CBR リアルタイムを別々に分けて考える CAC(Call Admission Control) 手法が提案されている．

Earliest Deadline First(EDF) スケジューリングは，他のスケジューリングアルゴリズムに比べて最短の時間で遅延保証ができる [?, ?]．しかしその実装においては，パケットが到着するたびにソーティングせねばならず，そのコストは大きく，実現が難しい．一方 Static Priority(SP) スケジューリングは，EDF スケジューリングと比べて遅延時間が大きくなる [?] が，実装が容易であるので広く使われている．このような状況のなか，EDF スケジューラと SP スケジューラの双方の特徴の中間地点を連続的にとることができる Rotating Priority Queue+(RPQ+) とよばれる待ち行列機構が最近提案された [?, ?].

Future work

これまで見て来たように，Deterministic guarantee についての研究は数多くなされて来ているが，Statistical guarantee を満たすための呼受け付けガイドラインはほとんど出来ていない．マルチメディアコミュニケーションにとって有効な Statistical guarantee を満たすための研究を早期に行う必要があるだろう．また，今後の利用方を考えると CBR 及び VBR リアルタイム双方に対応する方式について検討する必要がある．

5.4 Our research

以上これまで述べて来た状況の中，より良いマルチメディアコミュニケーションを行うためには，RT-Bone でどの様な研究を行うべきであろうか，ここで整理する．

- これまでの理論的研究成果の実証．RT-Bone という実験プロジェクトでのみ検証できる事を実行する．
- これまでに無い解析を行い，それを検証する．特に今後必要となるであろう，Statistical guarantee の実現を理論解析と実験とを並行して行う事によってより良い研究成果が得られるであろう．