IP ソースルーティングを用いた
複数経路同時送信方式

Simultaneous multi-path communication method
using IP source routing

平成 27 年 3 月

岐阜大学大学院
工学研究科博士後期課程
電子情報システム工学専攻

田 中　昌 二

目次

第1章 序論

1.1 研究背景

FTTH を始めとした高速インターネット接続環境や，スマートフォンに代表される高性能モバイル通信端末が一般層にまで普及したことで，インターネットを利用した動画の視聴，OS イメージの配布といった大容量コンテンツの送受信が広く利用されるようになった．Cisco Systems 社による Visual Networking Index 調査 [1]によると，インターネット上の年間データ転送量は 2016 年末には 1 ゼタバイトに達するとの予測が示されている．

大容量コンテンツ転送の増加による影響の 1 つの側面としては，大容量通信のリクエストが集中するサーバにおいて処理パフォーマンスが低下する，というものがあるが，この点においてはコンテンツ配信に特化した CDN（Contents Delivery Network）などの利用が進んでいる．ただし，CDN の構築には巨大なコストが必要であるため，多くの場合は多数の利用者にコンテンツを提供する大規模コンテンツホルダが，CDN 専用サービスを有償で利用するという方式が一般的である．そもそも，個人レベルのコンテンツ転送においては，サーバパフォーマンスに影響が出るほどリクエストが集中することが想定されないため，CDN を利用する必要性がない．

もう 1 つの側面は，そうしたコンテンツの転送による通信パフォーマンスの低下である．大容量コンテンツの転送は広い帯域を要求する上，コンテンツサイズが大きいだけに通信時間も長くなる．サーバにおける処理パフォーマンスがある意味で個々の大規模コンテンツホルダに閉じた問題であったのに対し，インターネットは規模の大小に関わらず共用されるリソースであるため，通信パフォーマンスの低下は個人利用者にも影響を与える問題である．

このような状況下で，ネットワークの通信帯域を圧迫する大容量通信に対応可能な通信方式検討が必要となっている．

1.2 一般的な IP ルーティングの限界

現在インターネットで一般的に利用されている IP ルーティング手法では，複数存在する経路から最適な1本を選択して利用する方式となっている（Fig. 1）．しかも，最適経路を選択する基準にはホップ数（経由ルータ数）やコストメトリック値といったネット

ワークのトポロジに対して固定的なパラメータが用いられており，経路の混雑状況などの動的なパラメータは考慮されない．

そのため，ネットワークの切断などネットワークトポロジが変化した際には最適経路の再選択が行われるが，たとえ最適経路が混雑によって通信パフォーマンスが低下していたとしても最適経路の再選択が行われることはない．

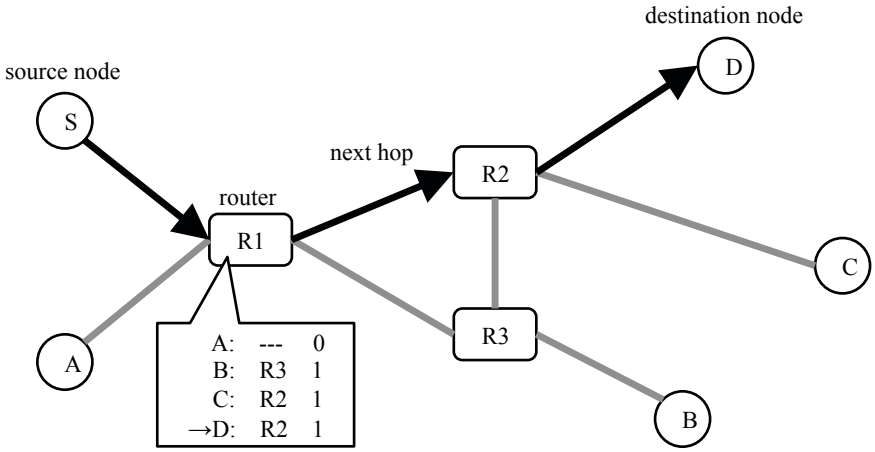


Fig. 1 Next hop selecting on IP routing

一方で，インターネット上を流れる通信のうち 30～80%については，よりパケットロスが少ないか，品質のよい別経路が存在することが Savage らによって明らかにされており [2]，また大規模な ISP の AS 内で，任意の Point-of-Presence（PoP）ペアのうち9割が，途中共有するリンクやノードが存在しない独立した経路を4本以上有する事例が Teixeira らによって確認されている [3]など，インターネット上には実用的な通信経路が複数あることが知られている．

1.3 最適経路以外の利用

最適経路以外の経路を積極的に利用する方式には，最適経路ではリアルタイムストリーミングなどの通信が要求するパフォーマンスを確保できない場合に，必要なパフォーマンスを確保可能な代替経路を利用する QoS ルーティングや，複数の経路を同時に利用して通信を行うマルチパス通信などが存在する．

特に，マルチパス通信を利用した通信帯域の集約はネットワークリソースの有効利用，対障害性，通信パフォーマンス向上という視点で注目を集めているが，現在研究されているマルチパス通信手法は，一度に複数のノードへ信号を送ることが可能な無線インタフェースを備えたモバイルインターネット分野や，エンドノードが複数の異なる

ネットワークインフラへ接続されたマルチホーム環境を前提にしているものが多い.

しかし, モバイル端末の台数が世界的に増加しているとはいえ, 現時点でインターネットに接続している端末は PC が主体であり, 一般的に利用されているデスクトップ PC の多くはネットワークインタフェースを1つしか持っておらず, マルチホーム環境も利用できない.

最適経路以外の有効に利用するためには, こうした単一ネットワークインタフェースしか持たない端末であっても利用可能なマルチパス通信方式が必要不可欠である.

1.4 新たな通信方式に求められる条件

新たな通信方式を提案する際には, その通信方式がネットワークに過大な負荷をかけないという点に注意が必要である. たとえ, その通信方式が高い通信性能を発揮したとしても, その結果としてネットワーク全体が輻輳状態に陥るようなことがあってはならない. そのためには適切な通信制御, 輻輳制御機構の実装が必要不可欠である.

また, その通信方式がどの程度のコストで実現可能かという点も重要である. インターネットは中央集権的な管理下にはなく, 個別の管理下にある AS（Autonomous System, 自律システム）と呼ばれるネットワークが, 相互接続する形で実現されている. そのため, インターネットを構成する全ネットワーク機器に対して新たな処理を必要とする場合, 実現コストは非常に大きなものになる.

1.5 研究目的と本論文の構成

著者はこれまで大容量コンテンツを効率的に送受信するための複数経路同時送信方式 [4] [5] [6] [7] [8]について研究してきた.

本研究では,

- ネットワークインタフェースを1枚しか搭載しておらず
- IP アドレスを1つしか割り振られておらず
- 接続先ネットワークが1つしかない

という条件でも利用可能であると同時に, その利用にあたり既存ネットワーク機器へ与える影響が小さく抑えられた複数経路同時送信方式を考案することを目的とする.

本論文は7章からなり, 本章に続く第 2 章では, 既存のマルチパス通信方式について, その特徴を様々な観点で分類し, マルチパス通信実現における課題を整理する.

第 3 章では, IPv6 経路制御ヘッダによって代替経路を利用し, 通信制御, 輻輳制

御，経路間優先制御機能を有する通信方式 SMPC（Simultaneous Multi-Path Communication）の提案を行う.

第 4 章で，SMPC のシミュレーション実験に使用したネットワークシミュレータ RNS の詳細を述べた後，第 5 章，および第 6 章で，RNS と実機環境を利用した SMPC 通信実験を行い，実験結果から SMPC の評価を行う.

さらに第 7 章で考察を，第 8 章で本研究の結論を述べる.

第2章 マルチパス通信方式

本章ではマルチパス通信方式の特徴と課題について述べる.

2.1 代替経路の選択手法

マルチパス通信では, 一般的な経路制御によって求められた最適経路以外の代替経路を利用する.

2.1.1 経由ルータによる経路制御

通信パケットの転送処理を行うルータが複数の経路を認識し, 通信パフォーマンスが最適となる経路を選択できる場合, エンドノードは代替経路の利用を意識することなく通信パケットをネットワークへ送出することができる(Fig. 2).

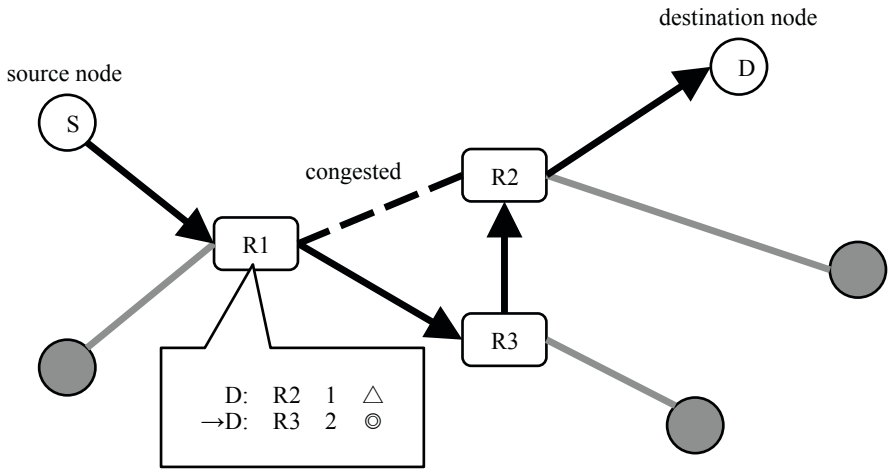


Fig. 2 Multi-path routing by the routers

しかしこの方式には2つの大きな課題が存在する.

1つ目の課題はルータ単体に必要な処理性能が増える点である. 2014 年現在, インターネットには, IPv4 だけで5×10^5を超える経路情報が流通している [9]. 多くのルータはデフォルトの経路情報の保持数を512k(524,288)個と設定しており, 一部にはハードウェア的にこの制限を超える経路情報を扱えない機器も存在している.

この所謂「512k 問題」はインターネット運用上の大きな課題となっている. 実際に2014 年 8 月には, 米国の大手プロバイダによる経路情報の集約ミスによって, 一時的に512kを超える経路情報が流通することになり, 全米のインターネット接続に大きな影響をおよぼす事態に発展した [10].

これまでの経路制御手法にならいルータ主導で代替経路を利用するためには，ルータ上に最適経路以外の経路情報も保持しなければならず，そのためには巨大な経路表が必要となる．この課題に対しては，ネットワークトポロジの解析によって通信の宛先をクラスタリングすることで，必要となる経路情報を集約する方式などが提案されている．

2つ目の課題はインターネット全体での統一的な運用である．インターネットは全体を一元管理する組織が存在せず，インターネットを構成する各 AS がそれぞれのポリシーでネットワークの管理を行っている．そのため，新しい通信方式を採用するようインターネット全体へ強制することは事実上不可能である．ネットワークインフラ上に一部でも代替経路の利用に対応しないルータが存在した場合，マルチパス通信に対応したルータ A が代替経路へ送信しようとルータ B へ転送したとしても，ルータ B がマルチパス通信に対応していなかったためにルータ B にとって目的地への最適経路であるルータ A へ返送されてしまう，といった事態が起こる可能性がある．

2.1.2 マルチホーミング（Multihoming）

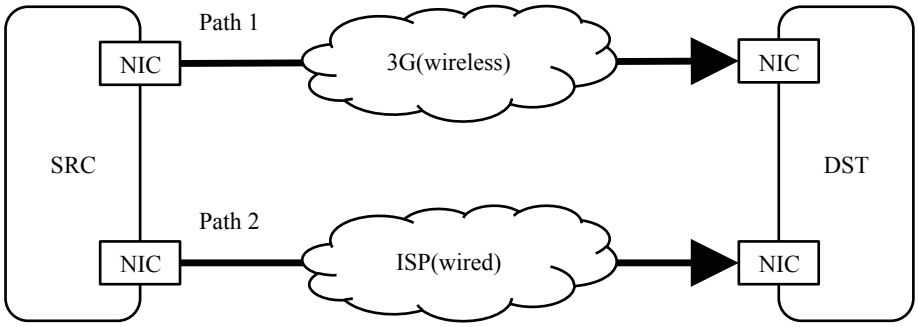


Fig. 3 Multi-homing

Fig. 3 のように，複数のネットワークインタフェースがそれぞれ異なるネットワークに繋がっている場合，パケットの送出先ネットワークインタフェースを選択する事で使用経路を変更することが可能となる．このように複数のネットワークを利用可能な状態をマルチホーム環境と呼ぶ．マルチホーム環境では各ネットワークインタフェースを利用した通信はそれぞれ通常の IP ルーティングに従って転送されればよいため，既存のネットワークインフラへの変更を必要とせず，エンドノードによる処理のみでマルチパス通信が可能となる．

送受信ノード双方それぞれが FTTH を利用した所謂インターネット回線と，LTE などの同一パケット通信網に接続していた場合，それぞれの通信経路に途中共有する

ノードが全く存在しない経路（node-disjoint path）を形成できるという利点もある.

ただし，有線 LAN と Wi-Fi など複数のネットワークインタフェースを搭載している場合が多いノート PC などと異なり，一般的に利用されているデスクトップ PC の多くは有線 LAN インタフェースを1つしか搭載しておらず，接続先ネットワークも ISP が提供する ADSL や FTTH 契約の1種類だけである場合が多い. そうしたデスクトップ PC でマルチホーム環境を構築するためには，PC に新しいネットワークインタフェースを増設し，3G や WiMAX などの回線を新規に契約する必要がある. これらには当然コストがかかる上，ネットワーク構成が複雑化しライトユーザの利用にとってハードルとなる可能性が高い.

また，マルチホーミングの応用として，1枚のネットワークインタフェースに対して複数の IP アドレスを付与することにより，IP アドレスごとに異なる経路を利用可能とする方式も存在するが，やはり現時点のインターネット利用においては一般的な構成とは言えず，利用シーンは限られたものとなる.

2.1.3 ソースルーティング

インターネット通信における転送経路の選択は通常ルータによって行われるため，送信ノードは宛先ノードを指定してパケットを送信することしかできず，途中どのような経路を通って目的のノードに到達するかについてはまったく関与できない.

これに対し，通信の送信元（source）が経路制御（routing）を行う技術をソースルーティングと呼ぶ. ソースルーティングでは，送信元が途中どのような経路でパケットを転送すべきかを予め指定することができるため，ネットワークインタフェースが1つだけしか搭載されていない端末であっても代替経路を利用することが可能である.

なお，ソースルーティングには，1) 途中通過するルータを全て指定するストリクト・ソースルーティングと，2) 最終宛先の前に通過すべき中継ノードを指定するが「送信ノード→中継ノード」間や「中継ノード→最終宛先ノード」間の転送には関与しないルーズ・ソースルーティングの2種類が存在する.

1) ストリクト・ソースルーティング

ストリクト・ソースルーティング（Fig. 4）では送信ノードから宛先ノードまでに経由するルータを全て指定し，経由ルータに対しては必ず指定された順にルータを経由するようにパケットを転送することを要求する. この方式では，パケットの転送経路を厳密に指定することが可能となるが，経路上のルータは必ずソースルーティングで指定された次の経由ルータへ直接パケットを渡す必要があるため，経路上のルータ全てがソース

ルーティングの処理を行わねばならない. このことは, 2.1.1 で述べたインターネット全体での統一的な運用が必要になるという問題に加え, 送信ノードが受信ノードまでに経由するルータを全て把握する必要があるためインターネットを利用した通信においては現実的とはいえない.

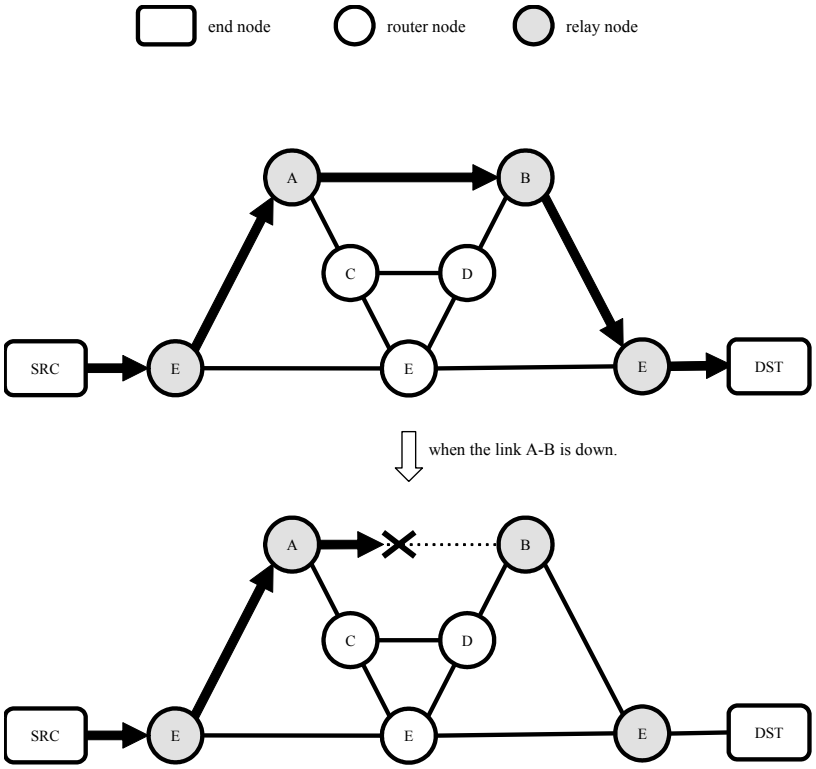


Fig. 4 Strict source routing

2) ルーズ・ソースルーティング

一方ルーズ・ソースルーティング(Fig. 5)では, 最終宛先ノードへ到達するまでに必ず通過しなければならないノードを1つ以上指定する. そのため, 代替経路上に中継ノードが最低1つあれば利用可能である.

この方式では送信ノードから送出されたパケットはまず中継ノードへ向かい, 中継ノードから次の中継ノード(もしくは最終宛先)へ向けて再度送信される. この方式はストリクト・ソースルーティングと異なり, 中継ノード以外のルータは一般的な IP 転送を行うだけで良いため, インターネット上の大半のルータには変更を加える必要がない.

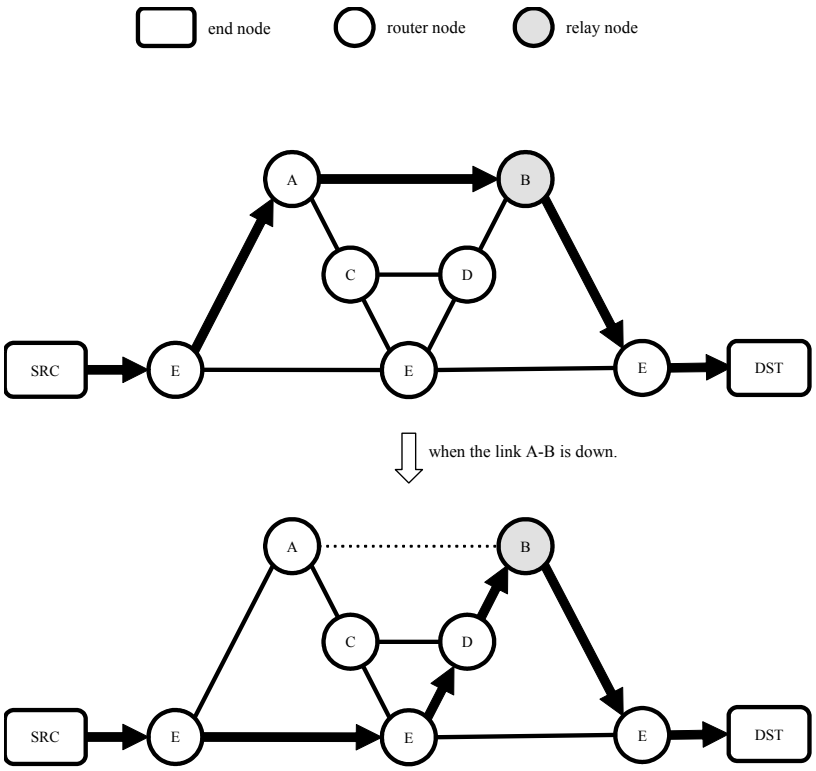


Fig. 5 Loose source routing.

2.2 マルチパス処理レイヤ

複数経路を同時に利用するマルチパス通信では，各経路へのパケットの振り分処理が必要である．ネットワーク通信における様々なレイヤでこうしたマルチパス通信処理行う方式が提案されており，特にエンドノード主体で行われるマルチパス処理の実行レイヤは大きく以下のように分類できる．

2.2.1 リンク層（MAC 層）

OSI 参照モデルの第二層，リンク層が主体となるマルチパス通信は，無線の異なるチャンネルや周波数帯を同時に利用する方式 [11]や，複数のネットワークインタフェースを仮想的に1つのネットワークインタフェースとして認識させる方式 [12]が提案されている．ただし，リンク層は基本的にルータを超えて通信できないためインターネットを介した通信では利用できない．

そのため，この方式は主に同一 LAN 内での利用や，端末側のインタフェース性能向上目的で利用される場合が多い．また，近年ではネットワークインフラを利用せず，モバイル端末がお互いにパケットの中継機能を提供することでネットワークを構築する

モバイルアドホックネットワーク分野での応用が広がっている.

2.2.2 インターネット層

インターネット層によるマルチパス処理は，既に広く利用されているトランスポート層プロトコル（とくに TCP）へ影響を与えない形でマルチパス通信を行う事に主眼を置くものが多い. ただし，TCP 通信は単一経路での利用を想定しているため，マルチパス通信を行う際の課題も多い.

こうした課題の1つは，TCP は送信パケットが送信順に受信ノードへ到着することを想定しているのに対し，マルチパス通信では，使用する経路によって伝送遅延が異なり，パケットが送信順に受信ノードへ到着することを保証できないことに起因する. パケット到達順の入れ替わりが頻発すると，TCP では輻輳状態であると誤検知される可能性がある. いくつかの方式 [13] [14]では，インターネット層で到着パケットの再整列を行うことによりこの問題に対応するが，再整列処理により TCP によるパケットロス検出が遅れるという副作用も存在する.

2つ目の課題は，RTT（Round Trip Time：往復遅延時間）計測への影響である. RTT はあるパケットが送信ノードから宛先ノードへ送信され，宛先ノードから返送された応答パケットが送信ノードへ到着するまでに要する時間であり， TCP では再送タイムアウト（RTO）の算出に RTT を使用している. 複数の経路を使用するとそれぞれの経路によって異なる RTT が計測されるため，RTO を正しく設定することができず，パケットロス時の再送判断をはじめとした輻輳制御に大きな影響が出る. この問題に対しては，異なる RTT の存在が RTO へ与える影響を数値モデル化し，RTO 算出に反映する方式 [15]などが提案されている.

2.2.3 トランスポート層

トランスポート層が行うマルチパス通信では，TCP を始めとした一般的に利用されている既存のトランスポート層プロトコルを拡張する方式が主流である.

TCP は根本的に1経路を使用する前提のプロトコルとなっているが，TCP を拡張することで通信のデータフローをいくつかのサブフローに分割しサブフロー単位で TCP コネクションを生成する方式 [16]などが提案されている. また，単一フローを分割したストリーム単位の通信管理や，マルチホーム環境による複数インタフェースの利用に対応した通信プロトコル SCTP を拡張することで，同時に複数経路を利用する方式 [17]も提案されている.

これらの既存プロトコルを拡張する方式の利点は，上位層アプリケーションとのやり

とりをこれまで通りの TCP, や SCTP として行えるため，上位層アプリケーションに対する影響が小さい点である.

一方，全く新しいトランスポート層プロトコルとしてマルチパス通信を実現する試みも提案されている. 既存プロトコルを利用しないことで TCP における輻輳制御との整合性など，既存プロトコルに由来する制限事項を考慮する必要がなくなり，より効率的なマルチパス通信が可能となる.

2.2.4 アプリケーション層

マルチパス通信機能を単体のアプリケーションとして実装する場合，アプリケーション側で制御が完結するため自由度の高い実装が可能である. P2P ファイル共有アプリケーションなどで，データを一定容量のブロックに分割し，複数のブロックをそれぞれ異なるノードから同時にダウンロードする方式も，広い意味でのマルチパス通信と言える.

一方，上位層のアプリケーションにマルチパス機能を提供するミドルウェアとして実装する場合，上位層アプリケーションに対するインタフェース設計が重要となる. TCP などの既存プロトコルと同様のインタフェースを提供できれば，既存の上位層アプリケーションを最小限のコストで新方式に対応させることが可能となる.

2.3 マルチパス通信における課題

Karim Habak らは，マルチパス通信においてデータパケットの順不同な到着による輻輳制御への影響回避と，パケット送出のスケジューリングが重要な課題であることを示した [18]. 本研究では，上記2点に代替経路を利用することによる通信全体のリソース消費という視点も加えた検討を行った.

2.3.1 パケットの順不同な到着と再整列（reordering）

一般的なシングルパスを用いた通信では，パケットは基本的に送信順に受信ノードへ到着する. まれにパケット到着順の逆転が起きる場合があるが，それは通信中に最適経路が変更された場合など非常にイレギュラーな状況である. しかし，マルチパス通信においては各経路の遅延や可用帯域が異なるため，データパケットが送信ノードから送出された順に受信ノードへ到着することは保証されない. そのため上位層のプロトコルやアプリケーションが順不同なパケットの到着を想定していない場合，マルチパス通信では受信側でパケットを再整列させる必要がある.

例えば，現在主流の TCP はデータパケットが送信順に到着することを想定しており，

データパケットが順不同で到着した場合，パケットロスと誤認識されるおそれがある．パケットロスは経路の輻輳によって発生する場合が多いため，TCP の輻輳制御はパケットロスに対して大きく送信速度を低下させてしまう．順不同なパケット到着によるTCP輻輳制御への悪影響は，TCPとは異なる輻輳制御を行うことや，各経路への通信がそれぞれ個別に輻輳制御を行うことなどにより解決できる．

しかし，一般的なネットワークアプリケーションは，受信データが順序通りに届くことを想定しているため，マルチパス通信においてはいずれかのレイヤで到着データの再整列を行う必要がある．

2.3.2 マルチパス通信におけるスケジューリング

マルチパス通信においては，複数の経路に対してどのようにデータを振り分けて送信するかというスケジューリングが非常に重要である．

マルチパス通信のスケジューリングには，送信パケット単位で使用経路を振り分けるパケットレベルスケジューリング，通信のコネクション毎に使用する経路を振り分けるコネクションレベルスケジューリング，および単一コネクションを幾つかに分割して行うスケジューリングの3種類の考え方が存在する．

1) パケットレベルスケジューリング

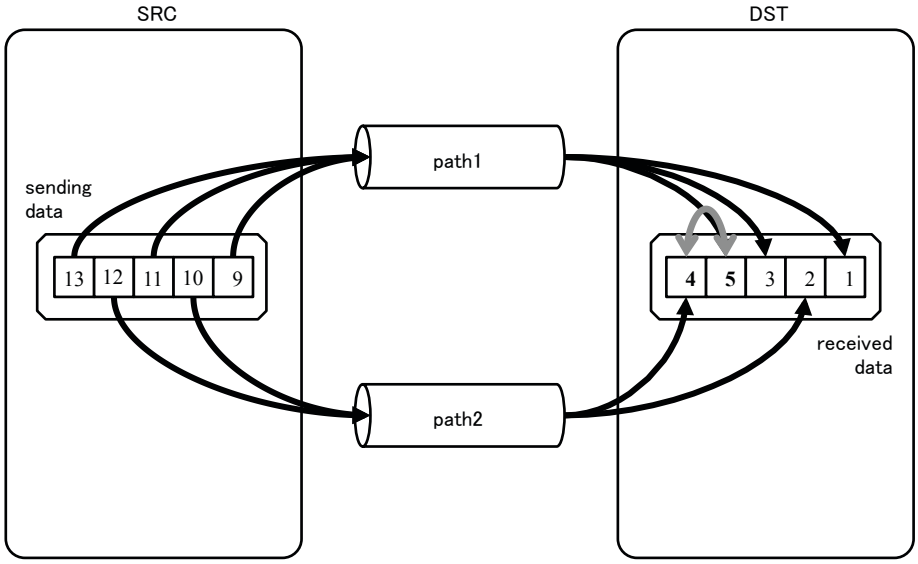


Fig. 6 Packet level scheduling

マルチパス通信が検討される中で，初期に多く検討されたのがパケットレベルスケジューリング方式である．パケットレベルスケジューリングでは，Fig. 6 に示すようにマル

チパス通信で使用する複数経路のうちどちらの経路を使用するかを，個々のデータパケット単位で選択する．この方式では単一コネクションが送信するデータであっても，パケットをそれぞれ異なる経路で送信可能であるため，上位層プロトコルやデータ種別に関わらずマルチパス通信の恩恵を得られる．

しかし，パケットレベルスケジューリングは，他のスケジューリング方式と異なり隣接するデータであっても異なる経路で送信される可能性があり，2.3.1 で示したパケットの再整列処理を始めとして，非常に細かなパケット送受信制御が必要となる．

2) コネクションレベルスケジューリング

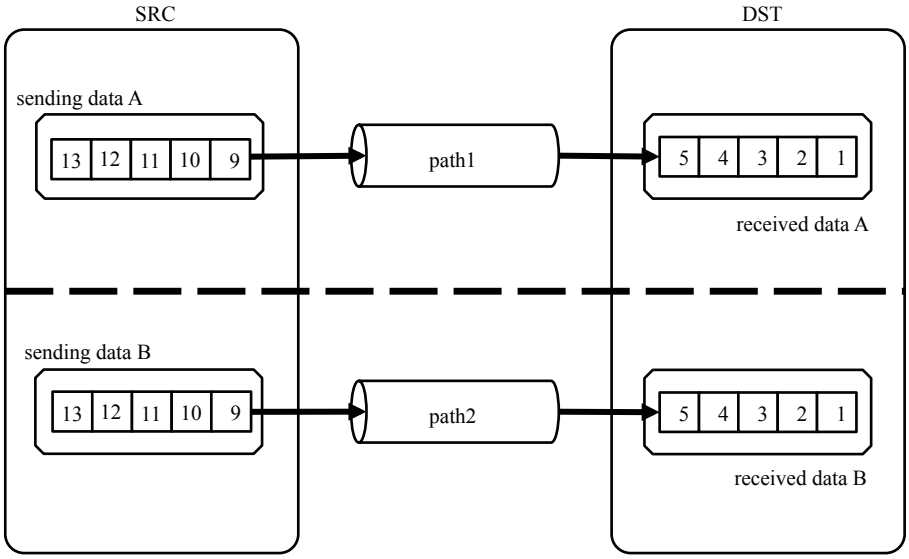


Fig. 7 Connection level scheduling

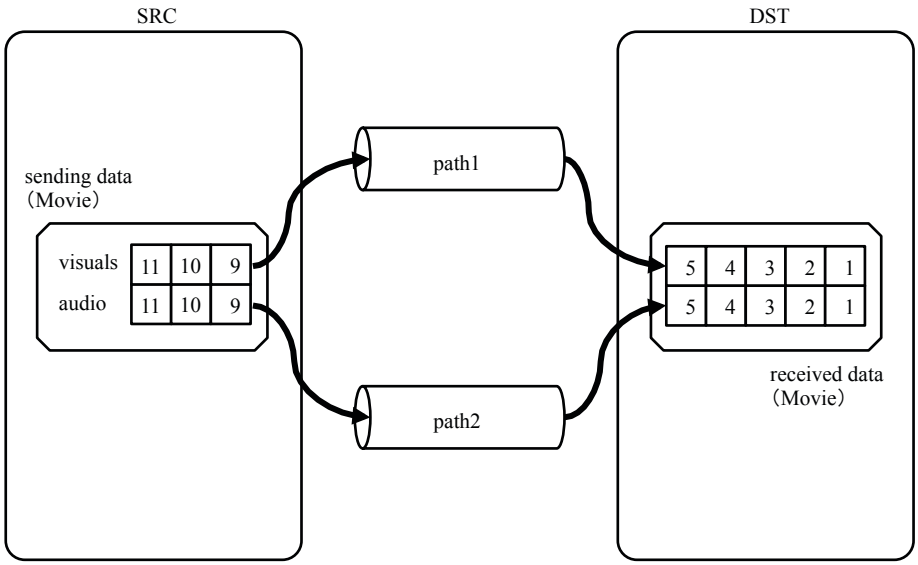
コネクションレベルスケジューリングでは，Fig. 7 に示すようにスケジューリング処理はコネクション単位で行われ，各コネクションは利用可能な経路のどちらか一方のみを利用する．端末が複数のコネクションを同時に利用している場合，端末全体としてみれば複数経路を使用する恩恵を受けることが可能となる．

また，各コネクションは使用する経路が異なる以外は一般的な通信となんら変わりはなく，マルチパス通信特有の変更を必要としない．そのため，既存環境への影響が小さく，コネクション単体としては1経路しか使用しないため 2.3.1 で示したパケット再整列も必要としない．

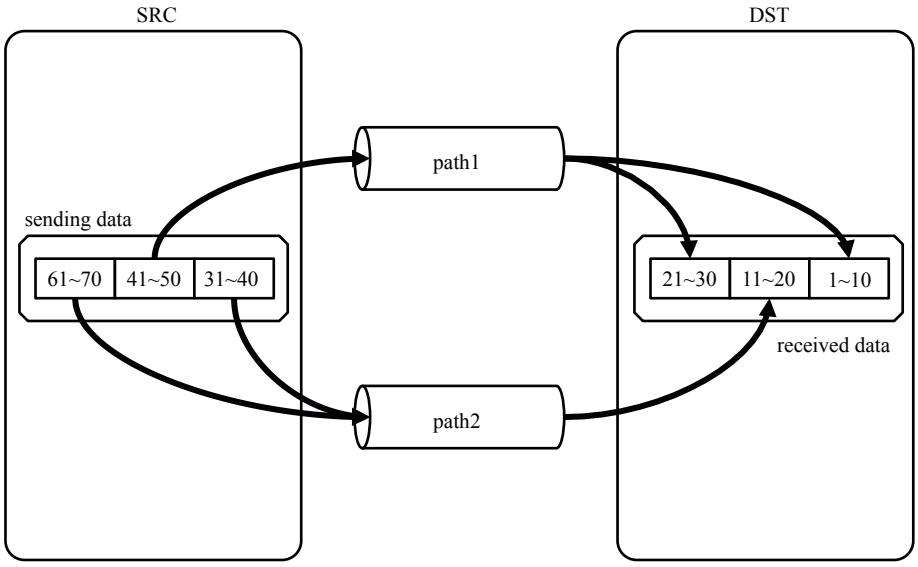
ただし，コネクションレベルスケジューリングでは，単一コネクションの通信パフォーマンスは1経路を利用した場合と変わらず，コネクション単体ではマルチパス通信の恩恵を受けることはできない．

3) 分割スケジューリング

コネクションレベルスケジューリングの応用として，単一コネクションが送信するデータを何らかの形で複数のデータに分割し，分割した部分データ単位でスケジューリングを行う方式も提案されている.



a) Data channel based splitting



b) Data block based splitting

Fig. 8 Splitted connection level scheduling

コネクションの分割方法としては，動画配信における映像と音声など，単一の通信であっても複数のチャンネルを保つ場合にチャンネル毎に経路を振り分ける方式（Fig.

8 の a)や, HTTP の Range ヘッダ等を利用し, 複数の経路を利用して異なるデータブロックを並列取得する方式(Fig. 8 の b) [19]などが提案されている.

2.3.3 マルチパス通信によるネットワークリソース消費

マルチパス通信においては一般的な最適経路以外の代替経路を使用して通信を行う. しかし最適経路は IP ルーティングがホップ数やリンクの帯域幅などなんらかの基準によって最適だとされる経路であるため, 最適でない代替経路を利用することのデメリットも存在する。

例えば, ホップ数(経由ルータ数)を基準に考えた場合, インターネットを構成する組織間の経路制御プロトコル EGP(Exterior Gateway Protocol)は距離ベクトル型アルゴリズムを使用しているため, 一般的に2ノード間の最適経路とは両ノードを繋ぐ最短経路である. 従って, 代替経路は最適経路よりも経路長が長くなることが想定される. そうした場合, 代替経路を使用することによって, 最適経路を使用する場合と比べてネットワークリソースを余分に消費することになる.

ここで, 経由ルータ数Lの最適経路と経由ルータ数$L+\Delta l$の代替経路 1 本を使用するマルチパス通信によって, Q個のパケットが送信された場合を考える. ルータにおける1パケットの転送コストを1, パケット全体のうち最適経路に送信されたパケットの割合をγとすると, 最適経路および代替経路に送信されたパケットの転送コストおよび両者を合計した総転送コストをそれぞれ$C_m(\gamma)$, $C_s(\gamma)$, $C_t(\gamma)$とすると,

$$C_m(\gamma) = \gamma LQ \tag{1}$$

$$C_s(\gamma) = (1-\gamma)(L+\Delta l)Q \tag{2}$$

したがって,

$$\begin{aligned} C_t(\gamma) &= C_m(\gamma) + C_s(\gamma) \\ &= \gamma LQ + (1-\gamma)(L+\Delta l)Q \\ &= LQ + (1-\gamma)\Delta l\, Q \end{aligned} \tag{3}$$

となる. 一方, 最適経路のみを使用した場合(γ=1)の場合の総転送コスト$C_t(1.0)$は

$$C_t(1.0) = LQ \tag{4}$$

であるので, 両者の差 $(1-\gamma)\Delta l\, Q$ がマルチパス通信によって増加する転送コストに相当する.

この値は, γが 1 に近づくほど小さくなり, 最適経路と代替経路へほぼ同数のパケットが送信された場合(γ=0.5)には$0.5\Delta l\, Q$ であるのに対し, 全パケットの 90%が最適経

路を使用した場合（γ=0.9）は$0.1\Delta l\ Q$である．このことは，最適経路を優先的に使用することにより，マルチパス通信に起因するネットワークリソース消費の増加を抑えることが可能であることを意味する．

すなわち，最適経路が混在等により通信性能が発揮できない場合に代替経路を使用することで高い通信性能を発揮するマルチパス通信であっても，最適経路に余剰の帯域が存在する場合には可能な限り最適経路へトラフィックを送信するべきである．

そのためには，マルチパス通信が使用する各経路に優先度を儲け，優先度の高い最適経路の可用帯域を優先的に使用するための経路間優先制御を行う必要がある．

2.4 本研究が提案するマルチパス通信の構成と要件

2.4.1 代替経路選択

経由ルータによってネットワークインフラ側が代替経路の選択を行う方式（2.1.1）は，インターネットを構成するルータ全体が協調して処理する必要があるため，既存ネットワーク環境へのインパクトが非常に大きい．単一組織によって管理される AS 内部での利用は可能であるが，管理主体も管理ポリシーも異なるインターネットを介した利用は非常に困難である．

マルチホーミングを利用する方式（2.1.2）は既存ネットワーク環境へのインパクトはほぼ存在しないが，端末側に複数のネットワークインタフェースと接続先ネットワークが必要となる．

ストリクト・ソースルーティング方式（2.1.3 の 1）には端末側の構成上の制約が存在しないが，既存ネットワーク環境に対してパケットが通過する全ルータでの処理が必要となる．そのため 2.1.1 と同様の理由でインターネットを介した利用は困難である．

ルーズ・ソースルーティング方式（2.1.3 の 2）には端末側に構成上の制約が存在せず，代替経路上に必要となる少数（最低1つ）の中継ノード以外には既存ネットワーク環境への変更も必要としない．そのため，既存ネットワーク環境への影響を非常に小さく抑えた形で実現可能である．

本研究が想定する，「単一ネットワークインタフェースしか搭載されておらずインターネットへの接続口を1つしかもたない端末」による利用を想定した場合に，2.1.1 から 2.1.3 までの各方式について，端末およびネットワークインフラへ想定される影響の大きさを基にした評価を Table 1 に示す．表に示すように，単一ネットワークインタフェースしか搭載されておらずインターネットへの接続口を1つしかもたない端末において最

適経路以外の代替経路を利用する場合には，2.1.3 の 2 に示すルーズ・ソースルーティングが有効である．

Table 1 Characteristics of alternative path selecting method.

		Impact to	
	Type	End-node	Network
2.1.1	Multi-path routing by routers	◎	×
2.1.2	Multi-homing	×	◎
2.1.3 1)	Strict source routing	◎	×
2.1.3 2)	Loose source routing	◎	○

2.4.2 マルチパス処理レイヤ

マルチパス処理レイヤの選択においても，既存環境へのインパクトが重要な要素となる．ここでキーとなる要素はトランスポート層プロトコルである TCP の存在である．に対する対応である．トランスポート層以下による実装の場合，TCP の輻輳制御といかに共存するか，アプリケーション層による実装の場合，TCP での利用を想定しているアプリケーションに対していかにして対応していくか，という点は通信方式毎にスタンスが異なる．

また，エンドノードへの導入を前提とする場合，利用者への導入負荷も検討を要する．その方式が一般的になりOSにデフォルトで実装されるようになるまでは，利用者自身が自身の環境にマルチパス機能を追加して使用することになる．トランスポート層以下の処理は通常OSで行われており，実装レイヤが下位になればなるほど，OSへの影響度も大きくなるため，アプリケーション導入のハードルが高くなる．

こうした状況を考慮すると，マルチパス通信の処理はアプリケーション層で行うことが望ましい．ただし，利用者が直接利用する単体アプリケーションにマルチパス機能を実装した場合，利用シーンが限られてしまう．ある程度の汎用性をもたせるためにはアプリケーション層ミドルウェアという選択が最も有効である．

2.4.3 スケジューリング方式

単一経路を利用した場合，データパケットの順不同な到着は極稀にしか発生しないが，複数の経路を使用するマルチパス通信においては，経路ごとの通信性能や品質の差によって常に起こりうる現象である．

TCP の輻輳制御は単一経路による利用しか想定しておらず，こうした状況下では経

路の輻輳状態を把握することができない．またTCPを用いないマルチパス通信においても，輻輳は経路毎に発生する可能性があり，両経路をまとめた通信状況から個々の経路の混雑状況を切り分けることは困難である．

従って，パケットレベルのスケジューリング方式の実現には高いハードルが存在する．しかし，スケジューリングの単位をコネクションのレベルまで荒くしてしまった場合，単体コネクションではマルチパス通信の恩恵を受けられなくなってしまう．

単体コネクションの送信データを複数に分割し，個々の部分データは常に同じ経路を利用する分割スケジューリング方式は，両者のメリットを集約することができ，最も有効である．

2.4.4 経路間優先制御

2.3.3 に示したとおり，最適経路に余裕があるにも関わらず，代替経路へ多くのトラフィックを送信することは，ネットワーク全体としてみた場合に，無駄なリソース消費を伴う．代替経路にも充分な可用帯域が存在する場合には，その影響は比較的小さいが，マルチパス通信の送信ノードからみた代替経路であっても，当然その経路を最適経路として使用するノードが存在する．

本来，最適経路として使用する通信の帯域を，代替経路として使用するマルチパス通信が奪ってしまう状況は，ネットワーク利用の公平性の点から問題である．

そうした状況を考慮した場合，マルチパス通信を行う方式には，最適経路が混雑していなければ，そちらを優先的に利用して通信を行う経路間優先制御の実装が不可欠である．

第3章 複数経路同時送信方式（SMPC）

3.1 SMPC 概要

第 2 章の議論をふまえて本研究では，複数経路同時送信方式 SMPC（Simultaneous Multi-Path Communication）を提案する．SMPC が想定する基本的なネットワーク構成を Fig. 9 に示す．

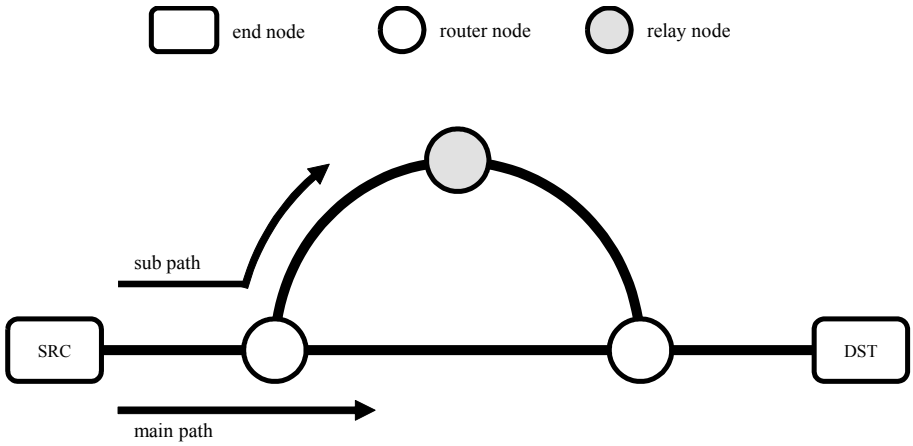


Fig. 9 Concept of simultaneous multi-path communication (SMPC).

SMPC が送信ノード（SRC）から受信ノード（DST）へのデータ送信に利用する経路のうち，1本は一般的な IP ルーティング手法により導かれた最適経路であり，この経路を主経路（main-path）と呼ぶ．また，主経路以外の代替経路を副経路（sub-path）と呼び，SMPC はこの両経路を同時に使用してデータ転送を行う通信方式である．

3.1.1 副経路の生成

代替経路である副経路へデータパケットを送信する方法としては，インターネット層のソースルーティング機能である IPv4 経路制御オプション [20]や IPv6 経路制御ヘッダタイプ 0 [21]を用いたルーズ・ソースルーティングを利用する．副経路上には IP パケットの中継を行う中継ノード（RLY）が存在するが，この中継ノードはインターネット層のソースルーティングを処理できればよく，SMPC 独自の機能は必要としない．

3.1.2 パケットトレイン

SMPC では，送信ノードから同一経路へ連続して送信されるN個のデータパケットか

らなる集合をパケットトレインと呼び，データ送信の処理単位として利用する．ここでNは通信開始前に予め決められた値であり，基本的に通信中は変化しないものとする．また，データパケット1つに含まれるデータサイズS[byte]も通信中は変化しない．

各パケットトレインには，シーケンシャルなトレイン IDXを設定する．Xはランダムに決められた整数のオフセット値X_0から始まり，パケットトレインの生成順に 1 ずつ増加する．ここでランダムなオフセット値を用いる理由は，TCP におけるセグメント値と同様，偽装をしづらくするための措置である．

また，パケットトレインを構成する各パケットには，それぞれトレイン内におけるパケット ID Kを設定する．Kはパケットトレイン毎に独立した 0 から始まる整数でありパケット送信順に 1 ずつ増加する（最大$N-1$）．このトレイン ID Xとパケット ID Kを組み合わせることで，個々のデータパケットを識別する事ができる．

パケットトレインに関連するパラメータS，N，X_0は，SMPC 通信開始時のネゴシエーションにより送信ノードと受信ノード間で共有される．

3.1.3 TACK（Train ACK）

受信ノードがパケットトレインの受信を完了すると，受信ノードは，パケットトレインの受信状況を送信ノードへ返送する．

このとき，受信ノードから送信ノードへ返送されるパケットを TACK パケットと呼ぶ．TACK パケットにはトレインの受信所要時間やトレイン内の個々のパケットが受信ノードへ到達したかどうかを示すビット列が含まれており，送信ノードは TACK パケットの内容から経路の混雑状況を把握することができる．

3.1.4 SMPC の実装レイヤ

SMPC はトランスポート層プロトコルとして UDP を利用し，上位層アプリケーションに対して通信ソケットを提供するアプリケーション層ミドルウェアとしての利用を想定している．ここでトランスポート層プロトコルとして送信制御や輻輳制御を行わない UDP を利用する理由は，SMPC 自身がパケット送信間隔を利用した送信制御と，受信速度計測による輻輳制御を行うためである．

3.2 SMPC のモジュール構成

Fig. 10 に SMPC の送受信ノードにおけるモジュール構成を示す．SMPC は送信ノード上で各経路へのパケット送出を制御する送信モジュール（Sender-Main/Sub）と送信モジュールの送信速度を管理する制御モジュール（Controller），および受信ノー

ド上で動作する受信モジュール（Receiver）から構成され，各モジュールはそれぞれが独立したスレッドとして動作する．

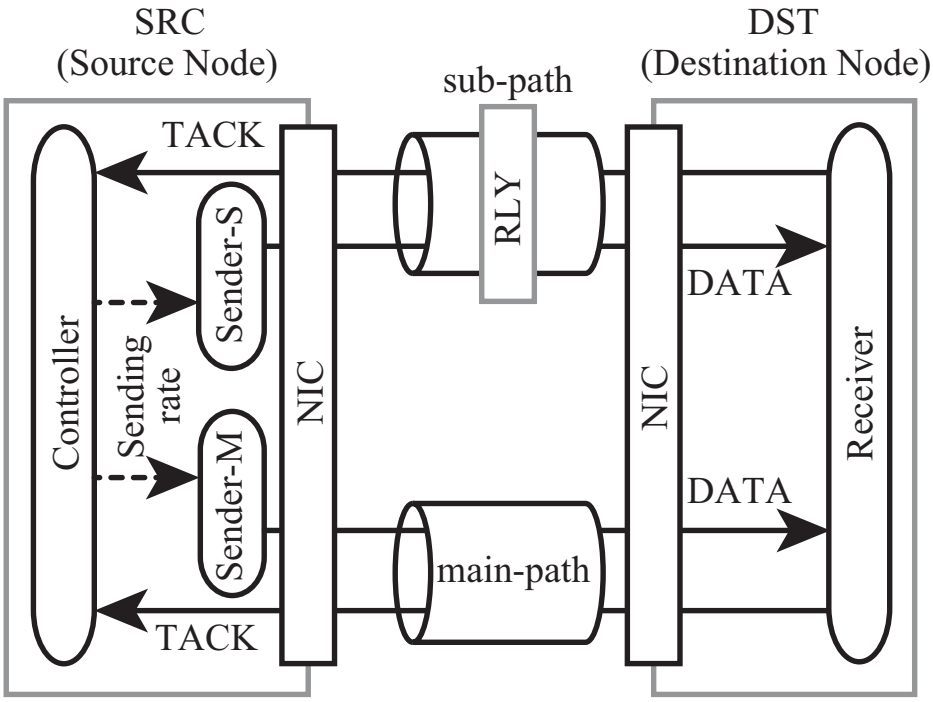


Fig. 10 Module structure of SMPC

3.2.1 送信モジュールの動作

送信ノード上の各経路向け送信モジュール（Sender-M/S）は，自身が受け持つ経路に対するパケットトレインの生成，データパケットのスケジューリングと送信をそれぞれ独立して行っている．

送信モジュールが行う動作の詳細を以下に示す．

3.2.1.1 パケットトレインの生成（送信モジュール）

各送信モジュールは自身が受け持つ経路用の送信データバッファを保持している．送信バッファが空になると，送信モジュールは未送信データの先頭から1トレイン分のデータ（$N \times S$ Byte）を取り出し，送信用データキューに展開する．

SMCP では，各経路の送信モジュールが必要に応じて逐次未送信データを切り出して送信パケットを生成しており，どちらの経路の送信モジュールによって切りだされるかによって，そのデータがどちらの経路を使用して送信されるかが決定される．

3.2.1.2 パケットの送信スケジューリング

送信モジュールは，制御モジュール（Controller）から与えられる送信速度Ws[bps]に対して，データパケットを一定の間隔I[sec]で送出する．ここでIはWsおよび1パケットのデータサイズSから以下のように求められる．なお，式中の 8 はWsが bps（bit per sec）を単位としているため，Byte で表されているSを bit 換算するために用いられる定数である．

$$I = 8\,S/Ws \tag{5}$$

また，パケットトレインの先頭パケットを送信完了した時刻Ts_1および最終パケットの送信完了時刻Ts_nから送信所要時間$Ts[X]$および，実効送信速度$Ws[X]$を求め，経路名$P[X]$とともに記録する．

$$Ts[X] = Ts_n - Ts_1 \tag{6}$$

$$Ws[X] = \frac{(N-1)8S}{Ts}[X] \tag{7}$$

副経路向けパケットのヘッダには中継情報が含まれ主経路側のパケットとヘッダサイズが異なり，パケットサイズでは通信性能の評価ができない．そこで SMPC ではペイロードサイズをもとに計算した送受信速度を用いる．

3.2.1.3 データパケットの送信

SMPC ヘッダに，トレイン ID X，パケット ID K，最終パケット ID $N-1$，経路名$P[X]$，データポジション Dpを記録し，送信用データバッファから1パケット分のデータを取り出してパケットを送信する．ここでDpは，送信されるパケットの SMPC 通信全体でのシーケンシャル番号である．

ここで，SMPC ヘッダは 28bit のトレイン ID，4bit のパケット ID，4bit の最終パケット ID，2bit の経路名，32bit のデータポジションの計 70bit に，2bit のパディングを加えた 72bit のサイズを持つ．

3.2.2 受信モジュールの動作

受信ノード上の受信モジュール（Receiver）はデータパケットを受信すると，トレイン ID およびパケット ID を元に到着データを整列し，受信データを上位層アプリケーションに送る．また，パケットトレイン毎の受信状況を TACK（Train ACK）パケットとして送信ノードへ送信する機能も持つ．

受信モジュールの動作を以下に示す．

3.2.2.1 データパケット受信時の動作

1) パケットトレインの最初に到着したパケットに対する処理

受信パケットがトレイン ID Xのうち最初に到着したデータパケットである場合，トレイン IDX用にサイズがN bit のパケット到達フラグ用ビット列Fを用意し，0 で初期化する．また，このトレインIDXにおける最初のパケットの到達時刻Tr_1に現在時刻を記録する．

2) 全受信パケットに対する共通処理

パケット到達フラグFのKビット目に1を立て，仮のパケット到達時刻Tr_nとして現在時刻を記録する．また，受信データをDpと共に受信バッファへ一時保存する．

3) パケットトレインの最終パケットに対する処理

受信したパケットがパケットトレインの最終パケットだった（$K = N - 1$）場合，1)および 2)で記録したTr_1，Tr_mから受信所要時間Trを算出する．

$$Tr = Tr_m - Tr_1 \qquad (8)$$

その後，TACK パケットに，トレイン ID X，経路名 P[X]，パケット到達フラグ F，受信所要時間Tr[nsec]を記録し，送信時と同じ経路Pを通って送信ノードへ返送する．

TACK パケットは 28bit のトレイン ID，2bit の経路名，16bit のパケット到達フラグ，64bit の受信所要時間の計 110bit に，2bit のパディングを加えた 112bit のサイズを持つ．

3.2.2.2 イレギュラーな TACK 返送処理

トレイン ID Xの最終パケットが到着しないまま，異なるトレイン ID X'をもつデータパケットを受信した場合，その時点で未到着のトレイン ID Xに含まれるデータパケットは経路上で破棄されたものと判断し，トレイン ID X'の最初のパケットに対する処理（3.2.2.1 の 1）に加え，トレイン ID Xに対する TACK パケットの返送を行う．

また，トレイン ID Xを受信中，最後にデータパケットを受信してから予め決められたタイムアウト時間T_{out}を過ぎても次のデータパケットが到着しない場合も同様にトレイン ID Xに対する TACK パケットの返送を行う．

このようにして返送される TACK では最終パケット受信時刻Tr_mは，TACK の送信処理が行われた時刻が使用される．

3.2.2.3 受信データのマージ

受信モジュールは先頭からどこまでのデータを受信完了しているかを示す受信完

了ポジションDpを管理している．そして，データパケットが到着するとDpがDcに連続するかどうか調べ，連続していれば上位アプリケーションへ引き渡してDcをDpに更新し，そうでなければ受信バッファに保持する．このようにして到着データの再整列（re-ordering）およびマージを行う．

3.2.3 制御モジュールの動作

送信ノード上の制御モジュール（Controller）は，受信ノードから返送された TACK パケットを受信すると，TACK に記載された$P[X]$，X，F，Trを元に経路Pの混雑状況を推定し，該当経路の送信モジュール（Sender-M/S）に対して送信速度Wsを通知する．

トレイン IDXの TACK を受信した際の，制御モジュールの動作を以下に示す．

3.2.3.1 受信速度算出

Fig. 11 に受信速度計測の概念図を示す．

制御モジュールは，TACK パケット内のパケット到達フラグFに含まれる 1 の個数から，実際に受信ノードへ到達したパケットの個数Mを取得する．その後，M，受信所要時間Trおよび1パケットのデータサイズSから，以下の式で受信速度Wrを算出する．

$$Wr = (M-1)\,8S\,/\,Tr \quad (5)$$

3.2.3.2 輻輳検知

制御モジュールは，1パケット当たりの送信パケット数N，到達パケット数M，および送信モジュールが送信時に記録した送信速度$Ws[X]$とWrとの比

$$R = Wr/Ws[X] \quad (9)$$

を利用し，送信速度$Ws[X]$に対する経路の混雑状況を以下に示す4パターンに分類する．

1) 混雑している（CONGESTED）

$M<N$の場合，経路$P[X]$上でパケットロスが生じている．こうした経路はすでに重度の輻輳状態にある．このとき，Fで0になっているビットを調べることで，どのパケットがロスしたのかを特定できる．

2) やや混雑している（SLIGHT）

$M=N$，$R<1$の場合は，パケットロスはないが送信速度より受信速度が遅くなっており（$Wr<Ws[X]$），通信路上に少し混雑が発生している可能性がある．そこで，閾値$Rd(<1)$を定め，$R<Rd$の場合にやや混雑していると判定する．

3) 混雑の可能性がある（MAYBE）

混雑が解消したばかりのときや，経路が混雑する中でクロストラフィックのパケットが集中的に破棄されたときなど，$M=N$，$1<R$ という場合が発生しうる．このとき，経路が混雑しているとは断定できないが，その可能性を否定できない．そこで，閾値 $Ra(>1)$を定め，$R>Ra$の場合は混雑の可能性があると判定する．

4) 混雑していない（NONE）

$M=N$，$Rd \leq R \leq Ra$の場合，パケットトレインは送信速度$Ws[X]$と同速度で受信ノードに到達している．したがって混雑は発生していない．

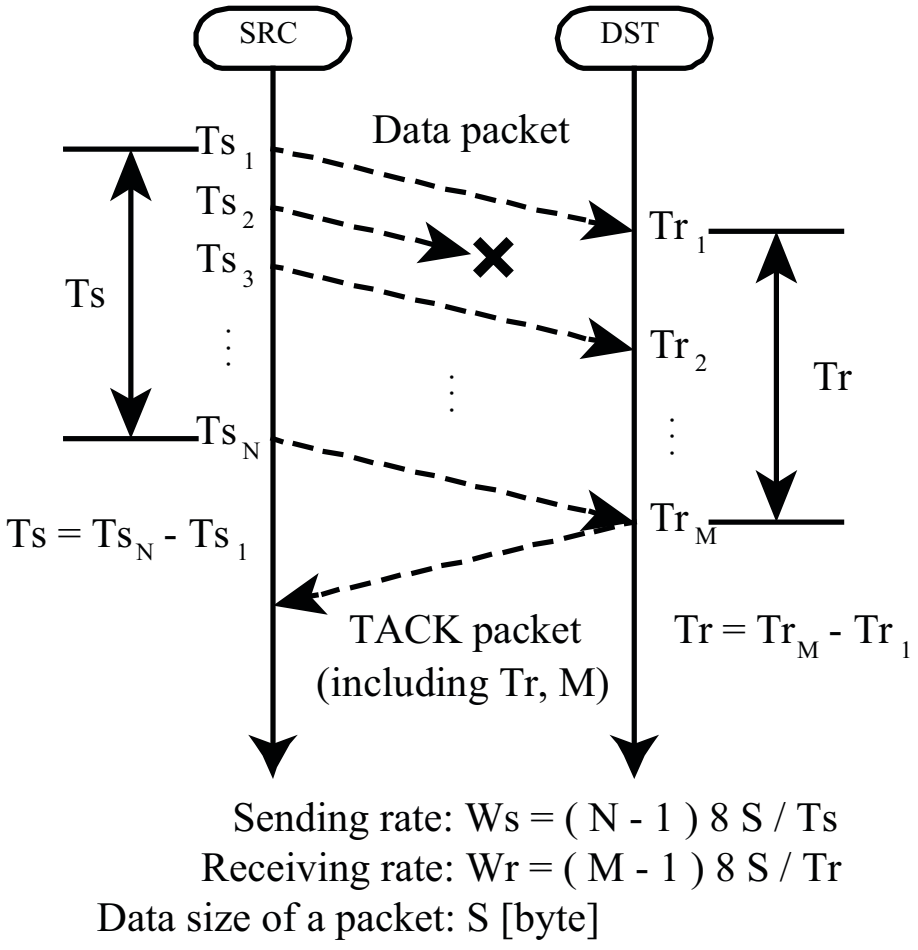


Fig. 11 Measurement of receiving rate with packet trains.

3.2.3.3 送信速度Ws更新（輻輳回避）

送信モジュールは 2)で判定した混雑状況に応じて，経路$P[X]$に対する送信速度WsをWs'へ更新する．

1) CONGESTED

すでに発生している輻輳状態を改善するため，更新後の送信速度Ws'を受信速度WrよりもさらにΔwだけ低い値まで低下させる．

$$Ws' = Wr - \Delta w \quad (10)$$

ここでΔwはSMPCが送信速度を加減速する際の単位量で予め設定された値である．

2) SLIGHT

該当経路の輻輳を回避するためWs'をWrまで低下させる．

$$Ws' = Wr \quad (11)$$

3) MAYBE

予防的にW_S'を$Ws[X]$からΔwだけ減速させる．

$$Ws' = Ws[X] - \Delta w \quad (12)$$

4) NONE

より大きな可用帯域が存在する可能性があるため，送信速度Ws'は$Ws[X]$からΔwだけ増加させる．

$$Ws' = Ws[X] + \Delta w \quad (13)$$

SMPCによる輻輳制御の動作をまとめたものがTable 2である．SMPCでは，広く利用されているTCP Renoを始めとしたパケットロスの検出によって輻輳を検知する方式に対して，受信速度の低下によっても輻輳を検知する．また，SMPCでは各経路が必要最小限の送信速度でデータ送信を開始することで，上記の輻輳回避機構と併せて，他の通信への影響を抑えている．

Table 2 Transmission rate control of SMPC.

Status	Condition	Situation	Ws'
CONGESTED	$M < N$	One or more packets are lost on the path.	$Wr - \Delta w$
SLIGHT	$M = N, R < Rd$	Receiving rate is decelerated obviously.	Wr
MAYBE	$M = N, R < Ra$	Receiving rate is accelerated obviously.	$Ws - \Delta w$
NONE	$M = N, Rd \leqq R \leqq Ra$	Receiving rate is almost the same as sending rate.	$Ws + \Delta w$

3.3 経路間優先制御

2.3.3 で述べたとおり，最適経路である主経路に充分な余裕がある場合，可能な限り主経路を利用することでネットワーク全体のリソース消費を抑制することが望ましい．そこで，SMPC では経路間に優先度を設定し，より優先度の高い経路で充分な帯域が利用可能である場合には，優先度の高い経路へトラフィックを多く配分する経路間優先制御を行う．

3.3.1 経路間優先制御の概要

経路間優先制御を行わず SMPC の送信制御，輻輳制御を利用した場合，主経路と副経路はそれぞれが個別に送信速度を増速させてゆき，その合計送信速度はある値に収束する．これは SMPC がマルチパス通信全体で利用可能な可用帯域を全て使い切っている状態である．

ただし，マルチパス通信では，同じように合計受信速度が収束しているようにみえても，以下の2種類の状況が考えられる．1つは，主経路の独立部分にはまだ可用帯域の余裕があるが，副経路向けの通信と競合することで送信速度が頭打ちになっている状況，もう1つは，主経路の独立部分が混雑しており，主経路側の可用帯域に余裕がない状況である．前者であれば，副経路側を減速させることで主経路側が増速することが可能であるが，後者の場合はたとえ副経路側を減速させても主経路が増速する余地がない．

そこで，SMPC では合計受信速度が一定であれば，副経路の送信を少し減速する，という一時的なアクションを行う．副経路側を減速することにより，主経路に余裕があれば主経路側は増速可能となる．主経路が増速したとしても副経路の減速と相殺されるため，この時の合計送信速度は一定のままであり，さらに副経路側の減速条件が維持される．このように，合計送信速度が一定である間は，主経路の増速と副経路の減速が繰り返され，主経路が優先的に利用されるようになる．

主経路の増速が限界に達すると，副経路を減速させても主経路は増速できないため合計送信速度が低下する．次回の判定では合計送信速度が低下しているため，副経路の減速アクションは行われない．そうすると副経路自身が 3.2.3.3 4)の処理により送信速度を回復させるため，副経路に対する過度の抑制が行われることはない．

その結果，主経路側の送信速度がその可用帯域を使い切って一定になった場合，副経路の送信速度も一定になる．

この方式を用いると，主経路の可用帯域が十分あって送信ノードが最大速度で送

信できる場合は，大部分の通信が最適経路のみで通信が行われ，主経路の可用帯域が不足した分のみ副経路で送信される．

本方式は，送受信インタフェースが各1枚であるようなネットワークで有効なばかりでなく，ネットワーク上に主経路，副経路の共有区間が存在するような場合も有効である．経路が分かれている部分では可用帯域が十分あるが共有区間で混雑が生じている場合，主経路側も副経路側も同じように遅延やパケットロスが発生して通信速度は増加できなくなる．しかし，本研究の優先制御アルゴリズムでは，このような場合でも，分岐している部分については主経路に優先的にパケットが送信される．

3.3.2 合計受信速度の収束判定

経路間優先制御が有効に働くには，合計受信帯域が収束したことを認識しなければならない．SMCP では合計受信速度の単純移動平均（平均受信速度）を監視し，この平均受信速度の変化を利用した収束判定を行う．Fig. 12 に合計受信速度の収束判定方式のイメージを示す．

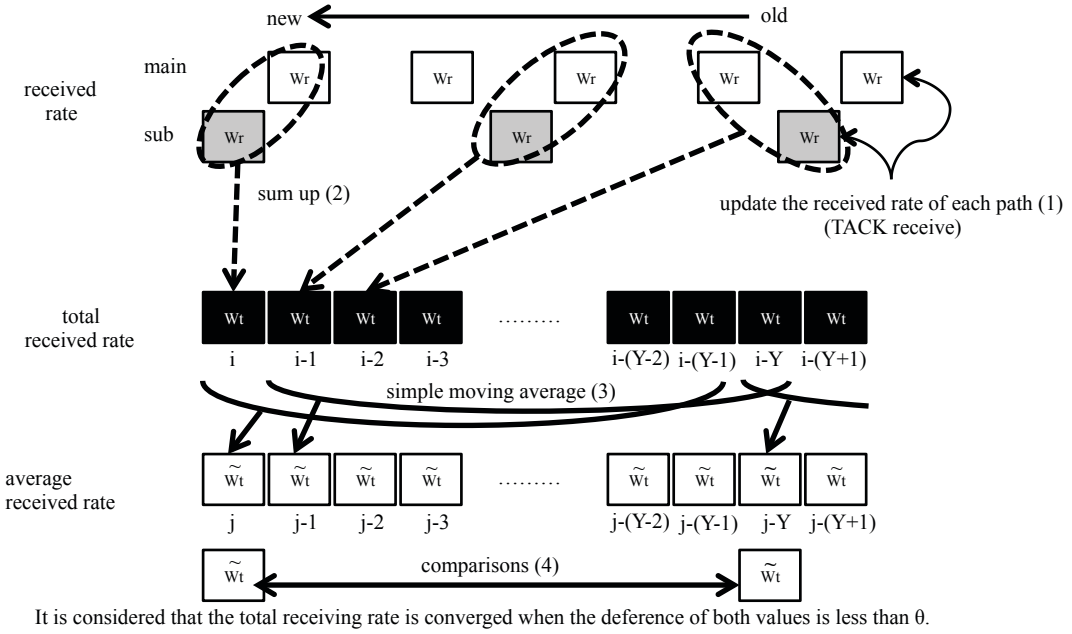


Fig. 12 Converged bandwidth estimation.

1) 経路別受信速度の監視

送信ノードは，経路ごとに直近 1 件の受信速度を保持するための受信速度バッファを持っている．TACK パケットの受信によって当該経路の受信速度バッファの値が更新される．

2) 合計受信速度W_tの算出

また，送信ノードは直近Y件の合計速度を保持する合計速度バッファを持っており，両経路で受信速度バッファが更新された場合に，各経路における直近の受信速度を合計した合計受速度W_tを合計速度バッファに格納する.

ここで，合計速度バッファへの格納を両経路の受信速度バッファが更新された場合に限定する理由は，TACK の受信頻度が通信速度に反比例するため，TACK のたびに毎回合計受信速度を記録した場合，低速側経路の受信速度変化を検出できなくなってしまうためである.

3) W_tの単純移動平均$\widehat{W_t}$の算出

さらに，送信ノードは平均合計速度バッファを持っている. 合計速度バッファに記録された直近Y件のW_tから平均受信速度 $\widehat{W_t}$を算出し平均合計速度バッファに格納する.

$i \geq Y$であるi番目の合計送信速度$\widehat{W_t}(i)$算出時に求められる平均受信速度$\widehat{W_t}(j)$は以下の式で求められる.

$$\widehat{W_t}(j) = \sum_{h=0}^{Y-1} \frac{Wt(i-h)}{Y} \qquad (14)$$

4) $\widehat{W_t}$の収束判定

3)で平均合計速度バッファに記録された直近の$\widehat{W_t}(j)$において$j > Y$であった場合，$\widehat{W_t}(j)$とY件前の平均合計速度$\widehat{W_t}(j-Y)$を比較する. その差が予め与えられた閾値 θ よりも小さければ，合計受信速度$\widehat{W_t}$が収束したと判断し，3.3.1 に示した経路間優先制御による副経路側の一時減速を実施する.

$$\left|\widehat{W_t}(j) - \widehat{W_t}(j-Y)\right| < \theta \qquad (15)$$

なお，合計受信速度$\widehat{W_t}$が収束しない場合や計算に必要なデータが揃わない場合は，経路間優先制御は行わない.

3.4 マルチパス通信としての SMPC

SMPC は代替経路選択に IP 層でのルーズ・ソースルーティング機能である IPv6 経路制御ヘッダを利用し，IPv6/UDP 上で独自に通信制御，輻輳制御を行うアプリケーション層の実装である.

送信データはパケットトレインに分割され，パケットトレイン単位での経路選択（スケジューリング）が行われる. これにより，パケットトレイン内ではデータパケットの順不同

な到着がおきず，輻輳制御機構への悪影響を回避している．

また，合計受信速度が安定した際に一時的に副経路側を減速することで，主経路側を優先的に利用する経路間優先制御機能をもつ．

このように，SMPC は 2.4 の検討結果に基づいて設計された通信方式である．

第4章 ネットワークシミュレータの製作

通信制御方式の評価において，実験環境の構築が大きなハードルとなる．大規模な実験ネットワークを実機で構築する場合，ハードウェア導入や設定の手間などのコストが非常に大きなものになる．そのためネットワーク通信研究の分野ではネットワークシミュレータを用いて通信挙動を確認するのが一般的である．

しかし，著者が IPv6 研究にとりかかった 2006 年時点では，NS-2 や QualNet など一般的に利用されていたネットワークシミュレータには，IPv6 経路制御ヘッダを処理できるものが存在しなかった．2008 年 6 月に NS-2 の後継となる NS-3 が開発されたが，NS-3 に IPv6 経路制御ヘッダ，正確には経路制御ヘッダによって実現されるルーズ・ソースルーティング機能が実験的に実装されるのは 2010 年に公開されたバージョン 3.7 まで待たねばならず，ルーズ・ソースルーティング機能が安定して動作するまでには更に年月を要した．

そこで，著者は IPv6 環境下で経路制御ヘッダを処理可能なシンプルなネットワークシミュレータである NetSim6（Network Simulator for IPv6）を作成した [5]．その後，NetSim6 の実装上の課題解決した RNS（Ruby Network Simulator）を新たに実装し SMPC の評価実験は RNS を用いたネットワークシミュレーションで行った．

4.1 NetSim6 概要

NetSim6 では，ネットワーク上の通信を，一連の IP パケットのライフサイクルとして考える．送信ノード上のアプリケーションによって生成された IP パケットは，ネットワークインタフェースを介してノード間の接続リンクに渡され，対向のネットワークノードに到達する．リンクからネットワークインタフェースを介して IP パケットを受け取ったノードは，それ自身宛の IP パケットであれば受信処理を行い，そうでない場合はしかるべきネットワークインタフェースへパケットを渡す．NetSim6 におけるパケットのライフサイクルを Fig. 13 に示す．

NetSim6 では，このようにネットワークノード，ネットワークインタフェース，リンクによって，IP パケットをバケツリレーすることでネットワーク通信を再現する．

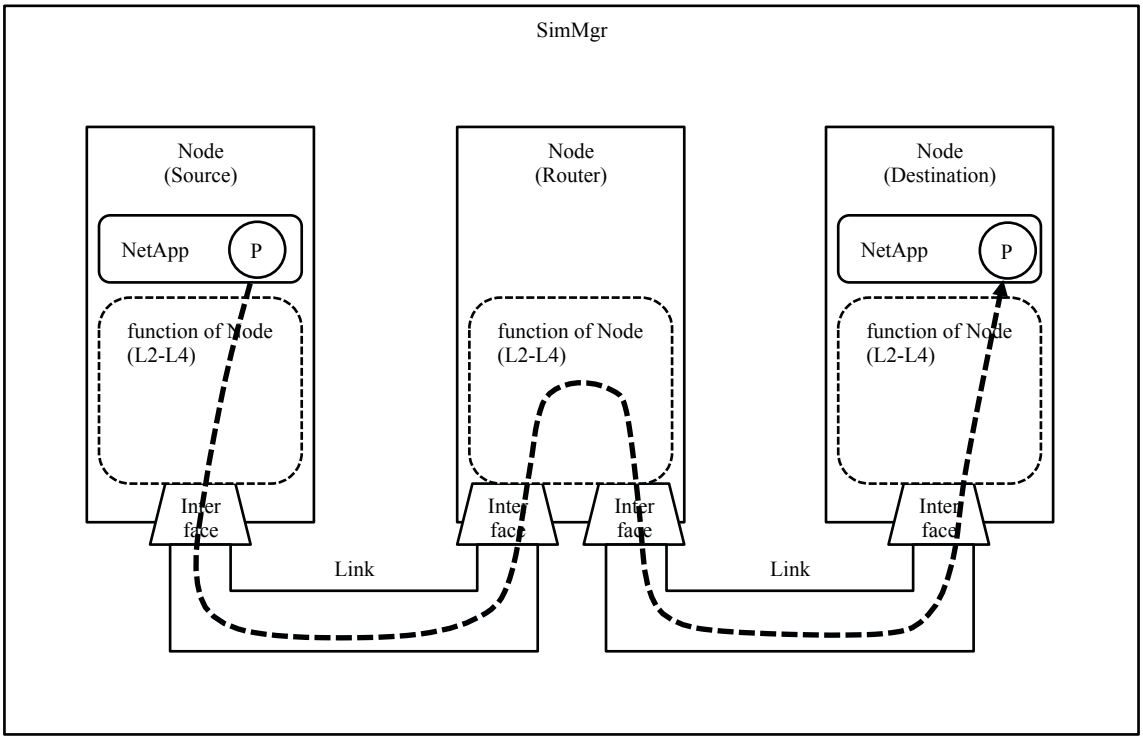


Fig. 13 Life cycle of packets on NetSim6

4.1.1 NetSim6 の構成

Fig. 14 に NetSim6 のクラスツリーを示す.

NetSim6 におけるクラス構成は, 各種エラー関係などを除けば Fig. 13 に登場する各要素をほぼそのままクラス化したシンプルなものとなっている. 各クラスの大まかな機能は以下のとおりである.

1) SimMgr

シミュレーション全体とタイムカウンタの管理.

2) Link

Interface から受け取った IPv6Packet を一定時間保持した後, 対向ノードの Interface へ渡す.

3) Interface

送信キューの管理. Node から IPv6Packet を受け取り Link へ引き渡す.

4) NetApp

IPv6Packet の生成及び受信処理. 実際の動作は NetApp モジュールを mix-in した

サブクラスが行う.

5) Node

IPv6 関連処理その他上記以外一般.

6) IPv6Packet

通信で送受信される IPv6 パケット. 全パケット共通して IPv6Header を内包しており, 副経路へ送信されるパケットには IPv6RtoutingHeaderType0 が追加される.

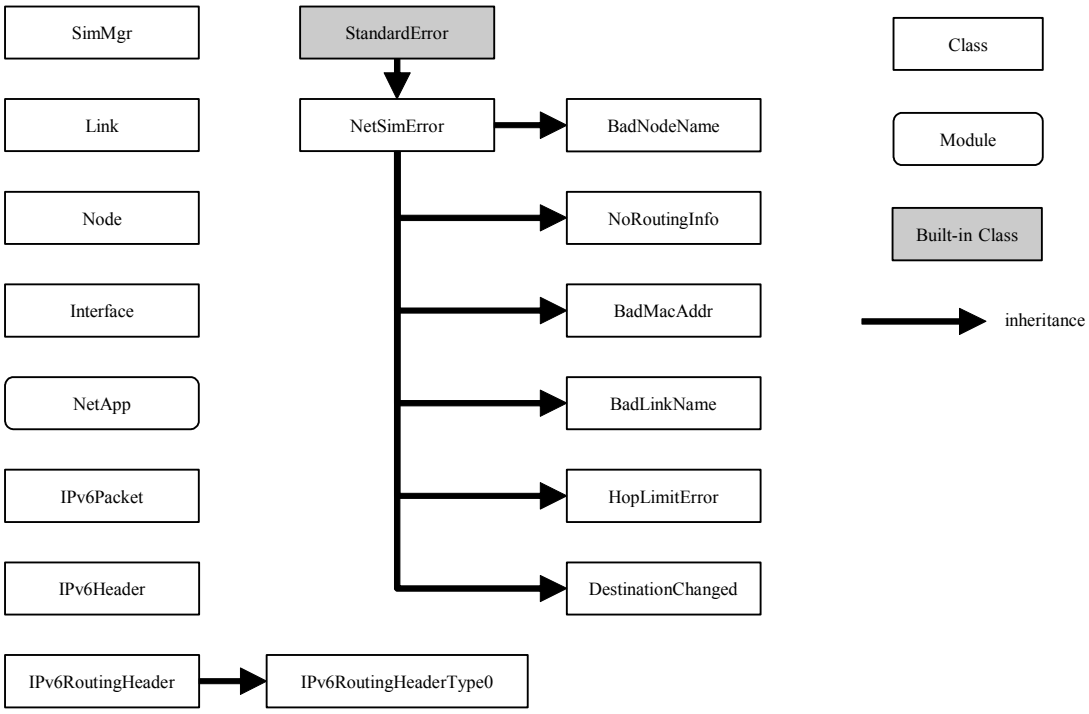


Fig. 14 Class tree of NetSim6

4.1.2 NetSim6 に残された課題

NetSim6 を利用することで IPv6 通信のシミュレーションを行うことには成功したが, 以下の様な実装上の課題が残されていた.

1) シミュレーションパフォーマンスについての課題

100Mbps の通信帯域を実現するため, シミュレーション内の処理時間単位であるタイムカウンタは1カウント=10^{-5}secに設定していた. この値はサイズが 1500byte のパケットによって 100Mbps の通信速度を実現するために必要な時間粒度である.

シミュレーション時間の経過はこのタイムカウンタを上昇させることによって処理して

おり，NetSim6 では1カウント毎にシミュレーション内の各要素をスキャンし，該当カウントでの起動イベント有無をチェックする実装となっていた．より高速な通信シミュレーションを行うには1カウントをさらに細かく設定する必要があり，そうした場合 NetSim6 のような1カウント毎に処理を行う方式ではシミュレーションパフォーマンスが大きく低下してしまうという問題があった．

2) シミュレーション精度についての課題

NetSim6 では，処理の簡略化のため送信される全てのパケットサイズを 1500byte 固定としていた．しかし，SMPC で副経路側へ送信されるパケットには IPv6 経路制御ヘッダが存在するため，同じペイロードを持つパケットであっても主経路を利用する場合と副経路を利用する場合とではパケット長が異なり，リンク通過のための時間が異なる．

大容量の通信により送信されるデータパケット数が多くなることで，こうした誤差がシミュレーション結果に与える影響が大きくなると考えられた．

3) 拡張性/汎用性についての課題

NetSim6 では，オブジェクト指向プログラミングの基本的な考え方である「機能の集約と切り分け」が十分に行われていなかった．

代表的な例が，Node クラスへの機能集中である．Ethernet/IPv6/UDP といった各レイヤの処理は Node クラス内に直接記述されており，IPv4 や TCP その他の通信方式に対応させるためには，Node 自体の実装を大きく変更する必要があった．

4.2 RNS（Ruby Network Simulator）

4.1.2 で述べた NetSim6 の課題を解決するため，NetSim6 の基本コンセプトを受け継ぐ新たなネットワークシミュレータ RNS を実装した．

RNS においてもネットワーク上の通信を一連の IP パケットのライフサイクルとして考える基本的なスタンスは NetSim6 と同様である．ただし，ノード内のネットワークレイヤ処理を独立したクラスとして実装するなど，パケット処理はより細かく定義されている．RNS におけるパケットのライフサイクルを Fig. 15 に示す．

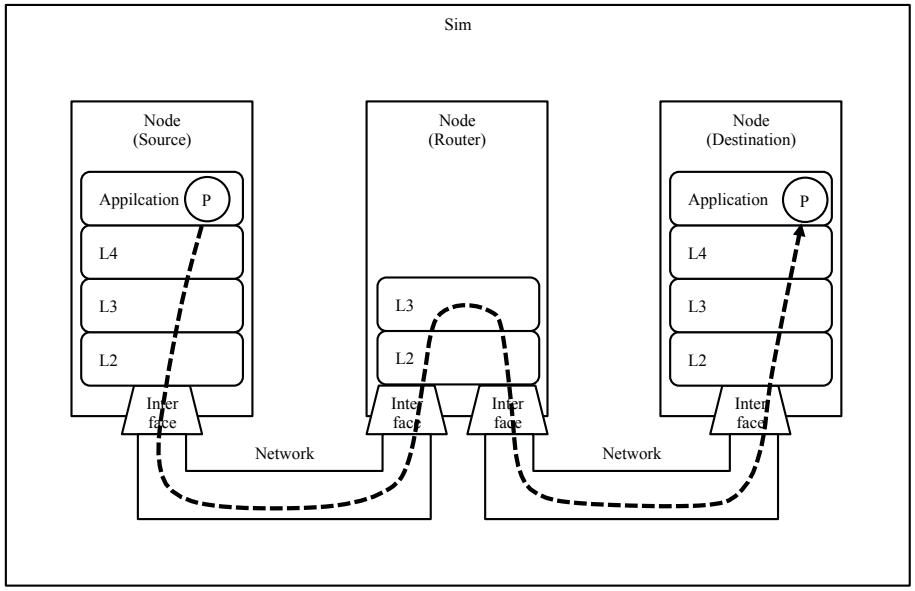


Fig. 15 Packet life cycle on RNS

4.2.1 RNS のクラス構成

RNS におけるクラスツリーを Fig. 16 に示す.

RNS のクラス構成は大きく, Simulator や Node, Interface など SimUnit を基底としたシミュレーション構成ユニット系と, MacAddress や IPv6Address, 各ネットワークレイヤのパケット構造など BitData を基底としたシミュレーションデータ系の2つのツリーに分かれる.

シミュレーション構成ユニット系ツリーの基底モジュールである SimUnit には, 各ユニットの識別子定義や, ユニット間の親子関係定義, シミュレーション時間の経過に対応するトリガ処理, 特定タイミングで起動するイベント処理といった共通機能が集約されている.

シミュレーションデータ系ツリーの基底モジュールである BitData にはデータのビット数を返すだけの機能しか定義されていないが, あるオブジェクトが継承ツリーのいずれかの時点で BitData モジュールを mix-in していることが, RNS が取り扱うデータオブジェクトであることを保証するものとなっている.

DataCapsule モジュールは内部に複数の BitData を収容できるカプセルであり, 自身も BitData の一種である. そうすることで, イーサフレームのペイロードに IP パケットが収容され, IP パケットのペイロードに UDP パケットが収容されるといったデータの包含関係を表現することが可能となっている.

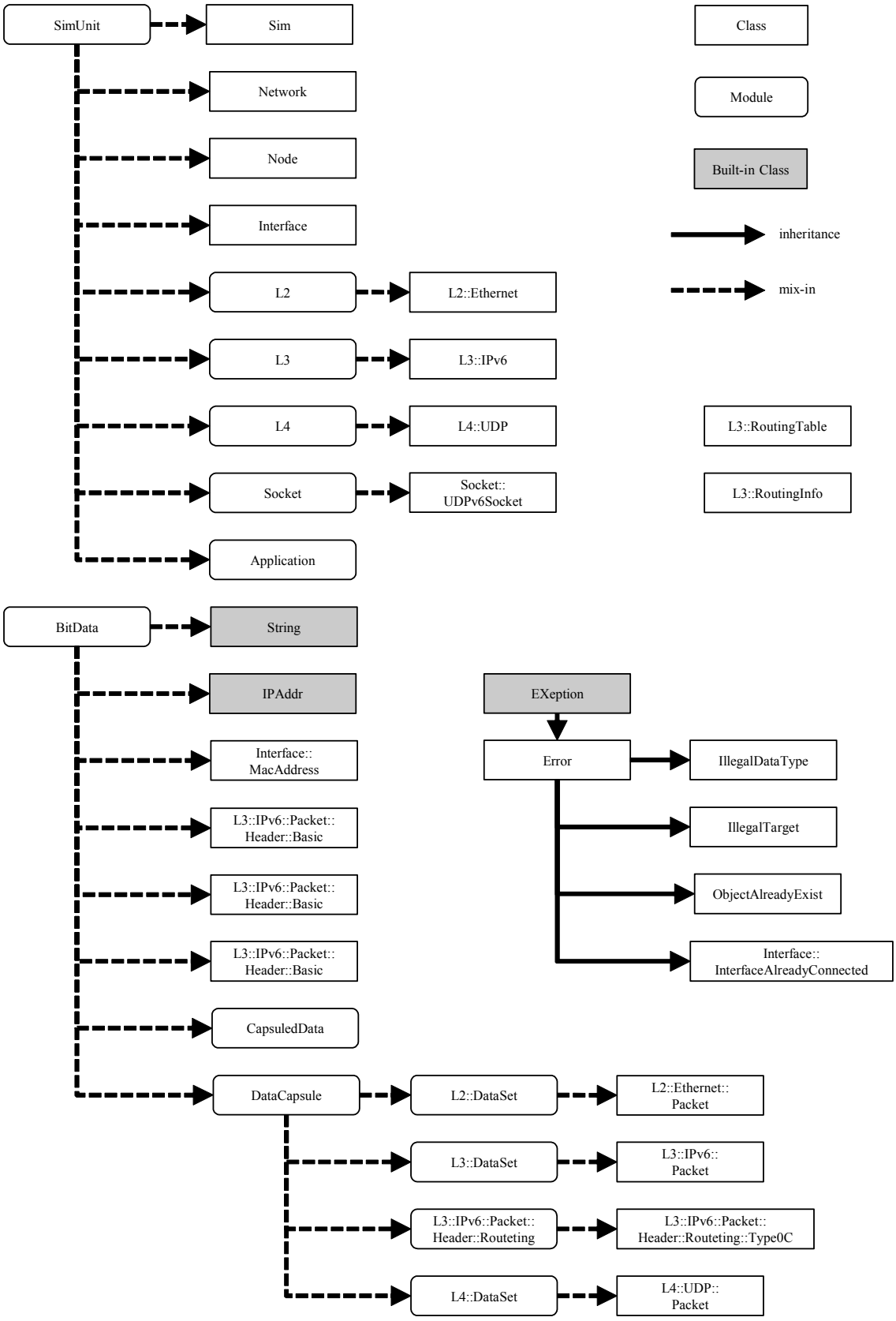


Fig. 16 Class tree of RNS

CapsuledData クラスはあらゆるオブジェクトを BitData として扱うためのラッパであり，BitData としたいオブジェクトとビットサイズを与えることで，DataCapsule に収容可能とする．また，文字列や IP アドレスなど Ruby がもともと持っているデータ形式のうち，ネットワークシミュレーションで頻繁に使用するクラスについては BitData を mix-in させ，直接 DataCapsule に収容可能とした．

また，これら 2 つのツリーの他に，経路表を扱う L3::RoutingTable クラスや，経路情報を扱う L3::RoutingInfo クラスと言ったユーティリティ系クラスや，エラー/例外処理系クラスが独立して存在している．

4.2.2 RNS におけるパケット送受信処理

RNS による UDP/IPv6 パケット送信処理のシーケンス図を Fig. 17 に示す．実際には各オブジェクト内でさらに細かな処理が行われているが，スペースの関係で図中に示すものはオブジェクト間の遷移のみに限定した．

ある Application オブジェクトがパケットを送信しようとする場合，送信データを引数として通信用にオープンした Socket オブジェクトの make_new_paket メソッドを呼び出す．Socket オブジェクトは L4::UDP，L3::IPv6，L2::Ethernet の各レイヤオブジェクトに対して順に new_packet メソッドを呼び出し，送信用パケットオブジェクトを生成して Application オブジェクトへ返す．

Application オブジェクトは受け取ったパケットを送信するため，Socket オブジェクトの send メソッドへ渡す．Socket オブジェクトはパケットの宛先アドレスを元に L3::IPv6 オブジェクトの経路情報を検索し，パケットを送出するための Inteface を特定し，add_output_queue メソッドへパケットオブジェクトを引き渡す．

Interface オブジェクトは自身が接続している Network オブジェクトの put メソッドにパケットオブジェクトを引き渡すが，出力キュー内にパケットが滞留している場合は，Interface オブジェクト内で出力キューサイズを 75 とした RED によるパケットの破棄が行われる．

Network オブジェクトは受け取ったパケットを，パケットサイズに応じた時間だけ保持し，対向側の Iterface オブジェクトの input メソッドに引き渡す．

Interface オブジェクトへパケットオブジェクトが到着すると，各レイヤの input メソッドが順に呼ばれて受信パケットが処理される．このとき一番大きな働きをするのが L3 処理を行う L3::IPv6 オブジェクトである． L3::IPv6 オブジェクトは受け取ったパケットが自身宛ではない，もしくは経路制御ヘッダによって別の最終宛先が指定されている場合，L2 オブジェクトの new_paket メソッドを呼び出して送信用パケットオブジェクトを再構築

したのち, Interface オブジェクトの add_output_queue へ引き渡すことでパケットの転送を行う. 受け取ったパケットが自身宛のものであった場合は, そのまま L4::UDP および Socket オブジェクトの input メソッドを経由して, 最終的に受信 Application オブジェクトの input メソッドに送信データが引き渡される.

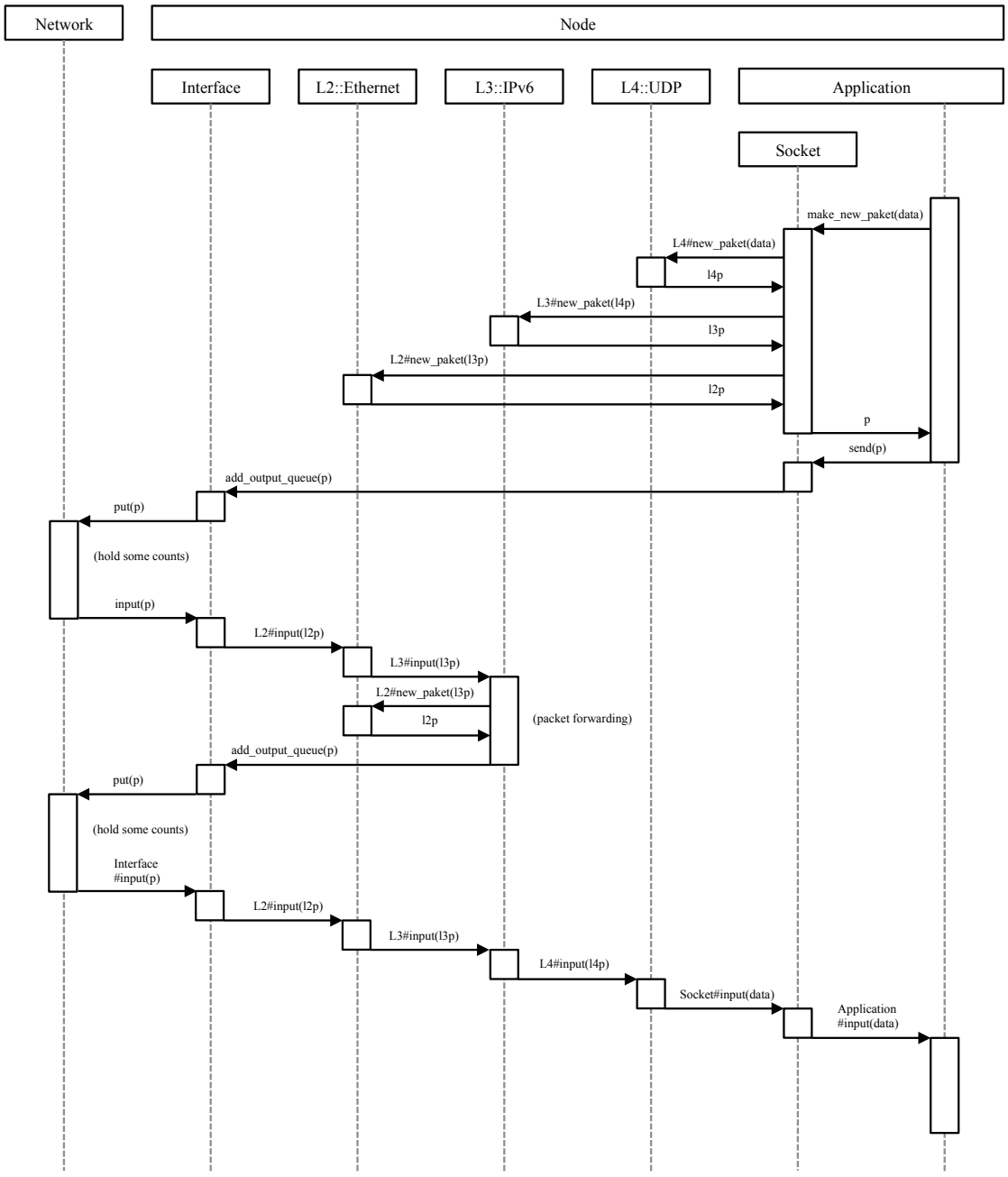


Fig. 17 Sequence diagram of the packet sending on RNS

このように RNS では，様々な階層を処理するオブジェクト群がパケット情報をバケツリレーのように受け渡すことでネットワーク通信を再現している．

4.2.3 NetSim6 からの改善ポイント

1) 高速通信時のパフォーマンス低下対策

より高速のネットワークにも対応できるよう，シミュレーション時間を管理するタイムカウンタの粒度を実行時に指定可能とした．またタイムカウントを1ずつ増加させるのではなく，あるタイムカウントにおけるアクションが完了した時点で各オブジェクトが次にアクションを行うカウントを通知することで，どのオブジェクトもアクションを行わないタイムカウントは一気にスキップさせるよう動作を変更した．このことにより，タイマ粒度を細かく設定したことによるパフォーマンスへの影響を減少させた．

2) 送受信データの詳細な定義

汎用のビットデータを処理するクラスと，ビットデータを包含するコンテナクラスを定義し，IPv6 ヘッダや Ethernet ヘッダ等，送受信データの実際のサイズをシミュレーション上で取り扱えるようにした．その結果，副経路向けのパケットに追加される IPv6 経路制御ヘッダが通信性能へ与える影響をシミュレーション結果に反映させることが可能となった．

3) 各クラスにおける共通機能の集約

NetSim6 における Node や Link，Interface などは，シミュレーションカウントの上昇に対する処理など，共通しする機能をそれぞれが個別に実装していた．RNS では，そうした処理を SimUnit に集約することでコードサイズを削減するとともに保守性を向上させた．

また，Node クラス内で各ネットワーク階層の処理を行う機能の共通部分を L2 や L3，L4 モジュールとして独立させ，Ethernet，IPv6，UDP といった具体的な処理は各レイヤモジュールを mix-in したクラスを用意した．こうすることで，IPv4 や TCP といった異なるプロトコルも実装可能となり拡張性も向上した．

4.3 RNS の評価実験

RNS のシミュレーションを検証するため，実機環境との比較実験を行った．

実験ネットワークの構成を Fig. 18 に示す．

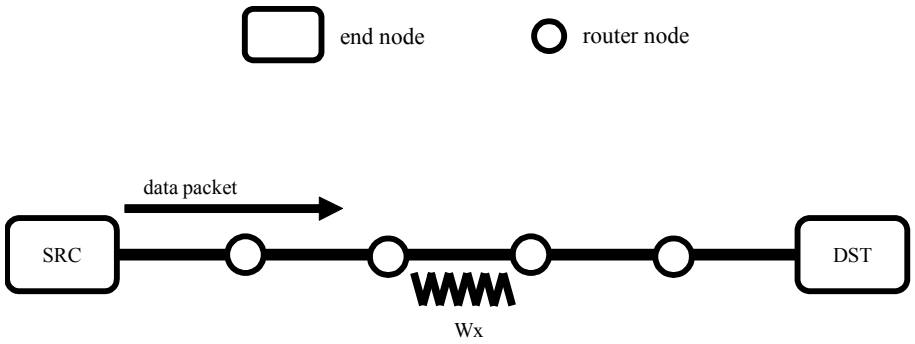


Fig. 18 RNS evaluation network.

実験ネットワークは送信ノード（SRC）と受信ノード（DST）の間にルータが4つ存在するだけのシンプルな構成である．各ノード，ルータをつなぐリンクは物理帯域 100Mbps 全二重の通信性能を持っており，それぞれが独立したセグメントとなっている．

実機環境は，送受信ノードおよびルータに FreeBSD をインストールした PC を利用した．送受信ノードはネットワークインタフェースを1つ，ルータはネットワークインタフェースを複数搭載し，それらを 100Base-TX の UTP で直結している．シミュレーション環境は RNS 上に Fig. 18 と同様のシミュレーションネットワークを設定した．

実機環境，シミュレーション双方に SMPC と同じ送信速度制御，輻輳回避機構を実装した送受信プログラムを実装し，送信ノードから受信ノードへ100MB のデータを送信した．この際，経路上にクロストラフィックとして送信速度Wxの UDP 通信を行い，送信ノードから送られるデータ通信のトータルスループット，および各時点でのデータ受信速度Wrを計測した．

4.3.1 クロストラフィックに対するスループット変化

データ送信の間，一定速度のクロストラフィックが常に存在する場合のトータルスループットを計測した．クロストラフィック送信速度Wxを 0 から 10Mbps ごとに 90 まで変化させた場合の結果を Fig. 19 に示す．

グラフの縦軸はトータルスループット，横軸はWxを表している．実機実験とシミュレーションのいずれも，クロストラフィックの増加に対して，トータルスループットが低下している．SMPC の輻輳回避機構は，競合する UDP 通信が存在する場合には自身の送信速度を積極的に増加させないアルゴリズムとなっており，実験結果からもそうした動作が行われたことが確認できた．

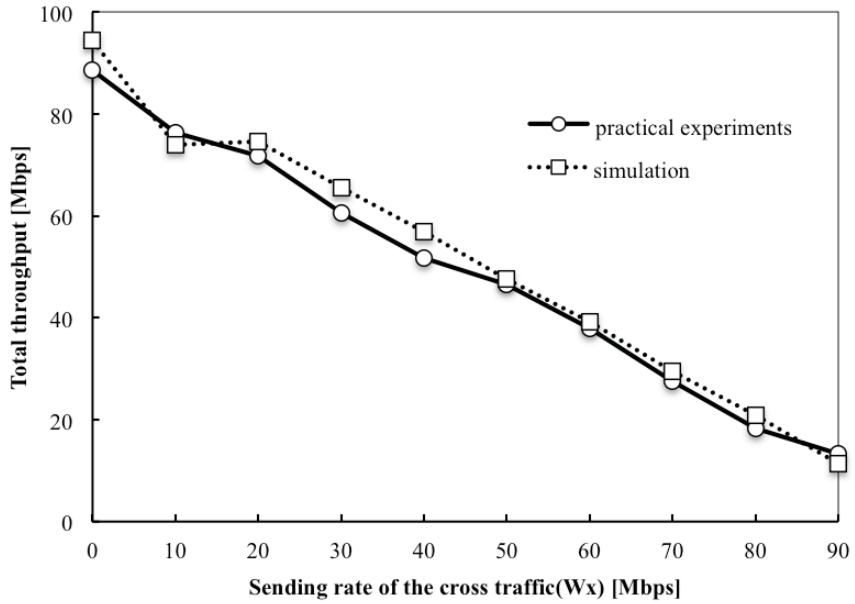


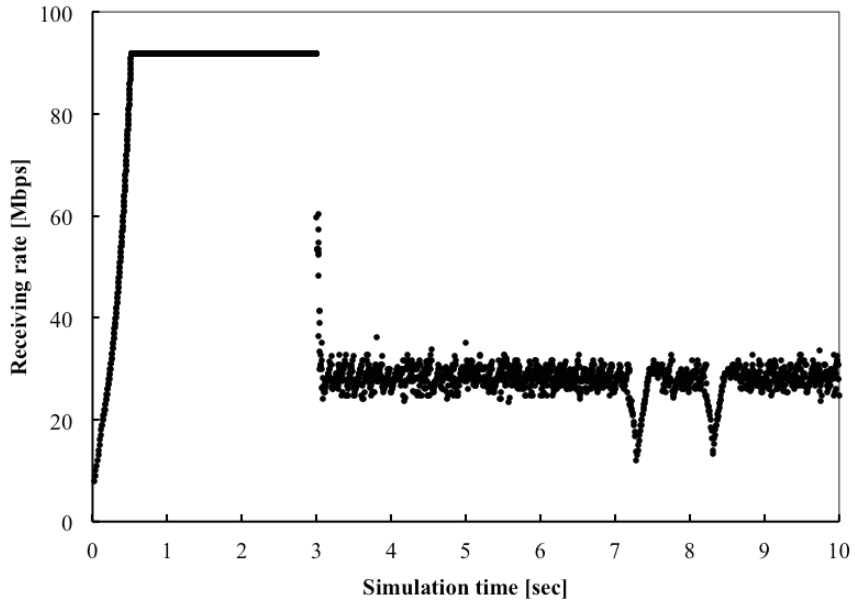
Fig. 19 Comparison total throughputs between simulation and practical experiments with cross traffic.

4.3.2 クロストラフィックの変化に対する受信速度応答

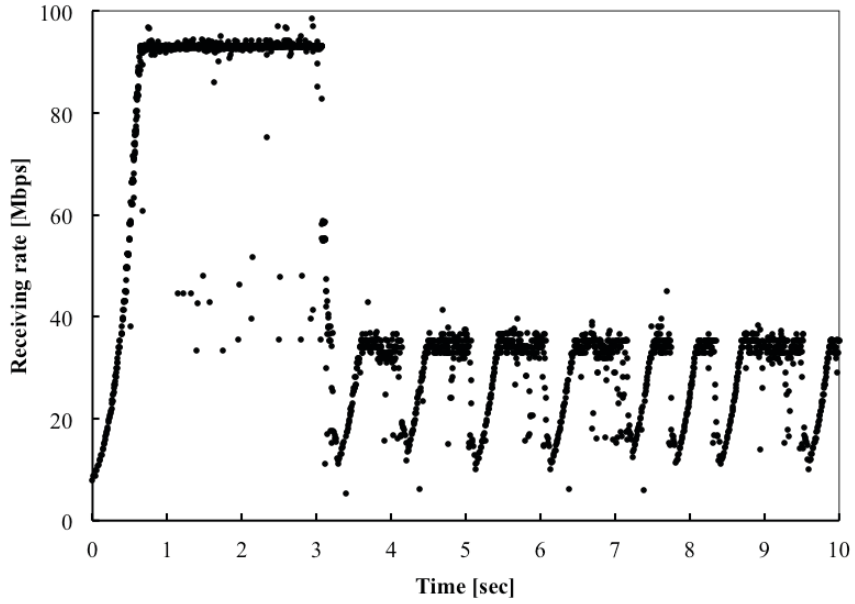
続いて，データ送信途中にスループットを変化させた場合に，受信速度Wrがどのように変化するかを測定した．

データ送信開始時点ではクロストラフィックが流れておらず（Wx=0），約 3 秒後にWx=70 のクロスとラフィックが送信開始された場合の受信速度Wrの変化を Fig. 20 に示す．グラフの縦軸はWr，横軸はWxの値を表している．

グラフより a) シミュレーション実験，b) 実機実験ともに，クロストラフィックの送信開始後にWrが 30Mbps 付近へ低下している．



a) Simulation result



b) Practical experiment

Fig. 20 Changes in receiving rate with cross traffic on single path.
(Comparison between simulation and practical experiment)

第5章 SMPC 通信実験：シミュレーション

SMPC による送信制御，輻輳制御，経路間優先制御の効果を確認するため，RNS を用いて評価シミュレーション実験を行った．

5.1 シミュレーションネットワーク

SMPC 通信シミュレーションを行うシミュレーションネットワークを Fig. 21 に示す．

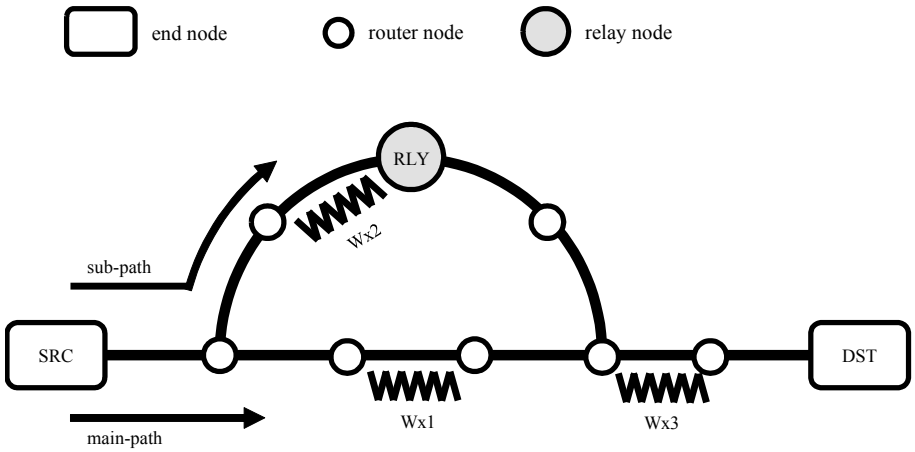


Fig. 21 Simulation network.

各ノードをつなぐリンクは物理帯域として100Mbps 全二重の通信性能を持っており，それぞれが独立したセグメントである．送信ノード（SRC）から受信ノード（DST）へデータを送信するには，ホップ数5とホップ数6の2本の経路が利用可能である．ホップ数の少ない経路が IP ルーティング上の最適経路となるため SMPC はこちらを主経路とし，ホップ数6の経路を副経路として利用する．副経路上には中継ノード（RLY）が存在し，副経路を利用するパケットは，IPv6 経路制御ヘッダに中継ノードの IPv6 アドレスを指定した状態で送信される．なお，シミュレーションにおける各種 SMPC パラメータは以下のとおりとした．

$$S = 1024\text{[byte]},\ N = 16,\ \Delta w = 1\text{[Mbps]},\ Rd = 0.95,\ Ra = 1.05 \quad [1]$$

また，主経路のみ，副経路のみ，主経路と副経路の共有部分に影響するクロストラフィックとして経路上のリンクに対して UDP 通信を行い，この時の各クロストラフィックの送信速度をそれぞれ$Wx1$, $Wx2$, $Wx3$する．

[1] RdおよびRaの値はクロストラフィックの存在しない状況でパケットロス率が 1%以下となる条件でスループットが一番高い組み合わせを選択した．（付録 A 参照）

5.2 経路の混雑に対するトラフィック分配

SMPC の経路間優先制御が，経路の混雑に対してどのような挙動を示すか確認するため，5.1 のネットワーク上で，$Wx1$，$Wx2$，$Wx3$を変化させた場合に，送信ノードから受信ノードへ SMPC を利用して 100MB のデータを送信した際のスループットを計測し，経路間優先制御を行わない場合と行った場合の比較を行った．

5.2.1 主経路混雑時の挙動

主経路上のクロストラフィックに対する SMPC の挙動を確認するため，$Wx2 = Wx3 = 0$とし，$Wx1$を変化させた場合のシミュレーション結果を Fig. 22 に示す．グラフの縦軸はスループット，横軸は$Wx1$を表しており，黒塗り部分が主経路を利用した通信のスループット，白塗り部分が副経路を利用した通信のスループットである．

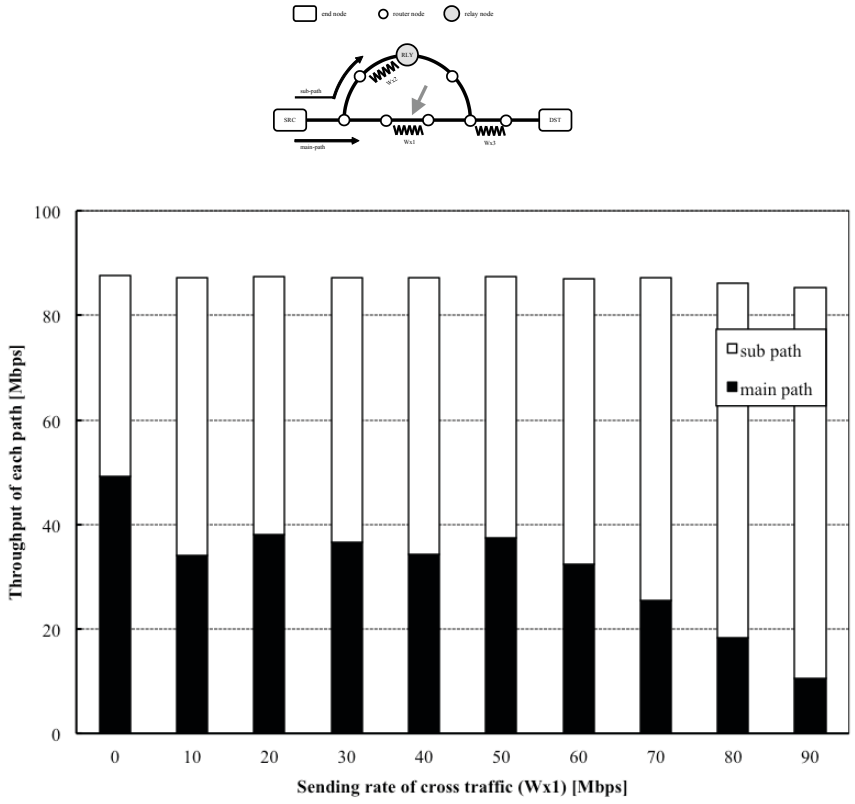
両結果ともに$Wx1$の大きさにかかわらず合計スループットは高く保たれており，単一経路を用いた場合にクロストラフィックによってスループットが低下したのと比べ，SMPC が副経路を利用することで，経路間優先制御の有無に関わらず主経路上のクロストラフィックの増加によるスループット低下を抑制できることが確認できた．

経路間優先制御を使用しない a）では，$Wx1 \leqq 50$ において場合に主経路と副経路のスループットが拮抗している．$Wx1 > 50$ になると$Wx1$の増加とともに主経路スループットが低下するが，副経路のスループットが増加することで合計スループットを高く保っている． 一方，経路間優先制御を使用した b）では$Wx1=0$ の場合に通信の大半が主経路を利用して行われており，$Wx1$の増加に従って主経路のスループットが低下する．ただし，ここでも主経路のスループット低下に応じて副経路のスループットが増加することで合計スループットは高く保たれている．

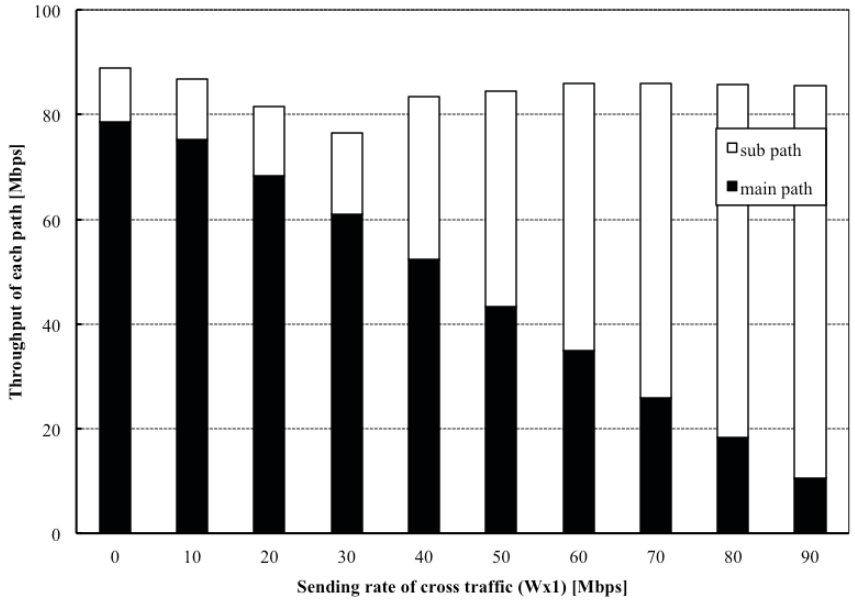
5.2.2 副経路混雑時の挙動

副経路上のクロストラフィックに対する SMPC の挙動を確認するため，$Wx1 = Wx3 = 0$とし，$Wx2$を変化させ，5.2.1 と同様のシミュレーションを行った結果を Fig. 23 に示す．

ここでも，両結果共に$Wx2$の大きさにかかわらず合計スループットは高く保たれており，単一経路を用いた場合にクロストラフィックによってスループットが低下したのと比べ，経路間優先制御の有無に関わらず SMPC のマルチパス通信によって副経路上の



a) SMPC without path priority



b) SMPC with path priority

Fig. 22 Throughput of each path with cross traffic on the main-path. (Comparison by having priority control or not)

クロストラフィックの増加によるスループット低下を抑制できている.

経路間優先制御を行わない a)では, $Wx2 \leqq 50$ で両経路のスループットが拮抗し, $Wx2 > 50$ では$Wx2$の増加に伴い副経路のスループットが低下するのに応じて主経路のスループットが増加している.

一方, 経路間優先制御を行った b)では, $Wx2$の大きさにかかわらず常に主経路のスループットが全体の 9 割近くを占める結果となった.

5.2.3 共有経路混雑時の挙動

両経路共有部分のクロストラフィックに対する SMPC の挙動を確認するため, $Wx1$=$Wx2$=0 とし, $Wx3$を変化させ 5.2.1 と同様のシミュレーションを行った結果を Fig. 24 に示す.

この条件下においては, 両経路共有部分にクロストラフィックが発生しているため通信全体の可用帯域が$Wx3$によって減少する. そのため, 5.2.1 や 5.2.2 と異なり$Wx3$の増加によって合計スループットも低下している.

経路間優先制御を行わない a)では, $Wx3$の大きさに関わらず, 低下する合計スループットを主経路と副経路が均等に分けあっている. 一方, 経路間優先制御を行った b)では, $Wx3$の大きさに関わらずトラフィックの大半は主経路を利用している.

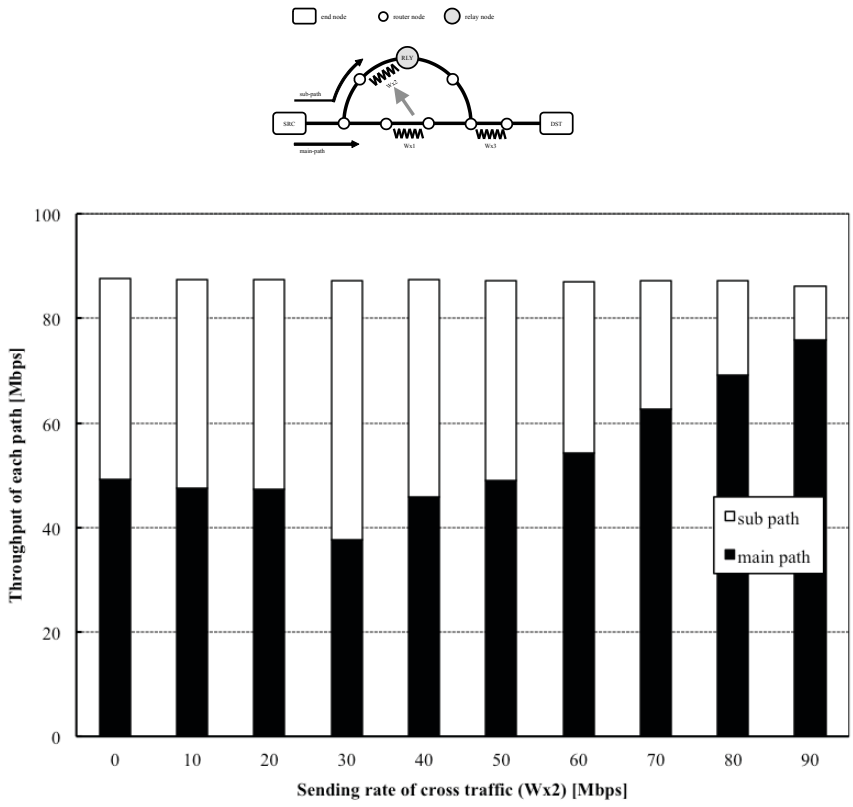
5.3 混雑状況変化への応答

経路上の混雑状況変化に対して, SMPC の送信制御, 輻輳制御, 経路間優先制御により, 受信速度がどのように応答するかについてシミュレーションを行った.

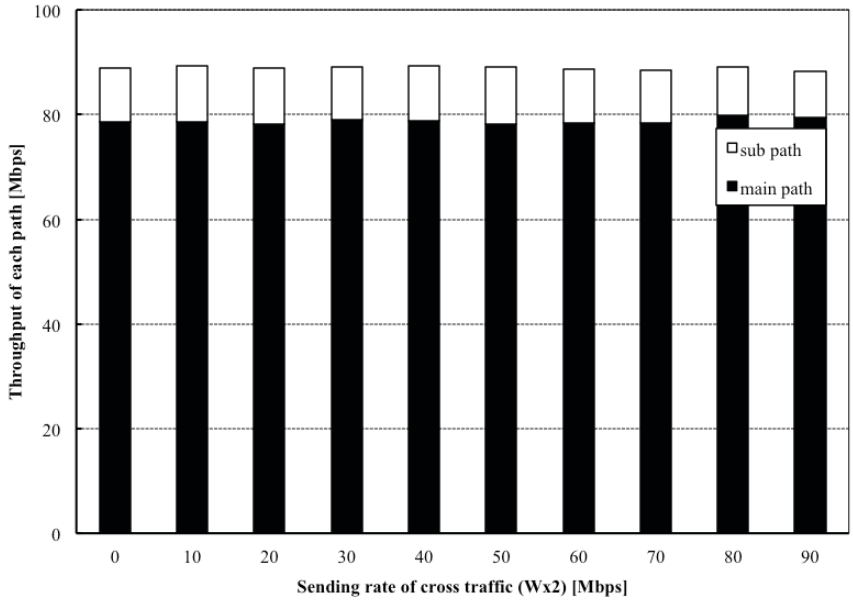
5.3.1 主経路混雑状況変化時

主経路の混雑状況変化への応答を確認するため, いずれのクロストラフィックも送信されていない状態で, SMPC 通信を開始し, その 3 秒後に$Wx1$が 70Mbps へ変化させた. この時の主経路, 副経路の受信速度, および両経路の合計受信速度の時間変化を Fig. 25 に示す. グラフの横軸は SMPC 送信開始から 8 秒後（トラフィック発生から5秒後）までのシミュレーション時間を, 縦軸は受信速度を表している.

クロストラフィックが存在しない SMPC 送信開始後3秒間は, 経路間優先制御を行わない場合と経路優先制御を行った場合の受信速度推移は大きく異なる. 前者 a)では両経路が同様に増速した後も両経路はほぼ同程度の受信速度を発揮する. 一方, 後者 b)は一旦両経路が同様に増速するが, 合計受信速度が最大になると副経路側の受信速度が低下を始め, 代わりに主経路側の受信速度が増速する.

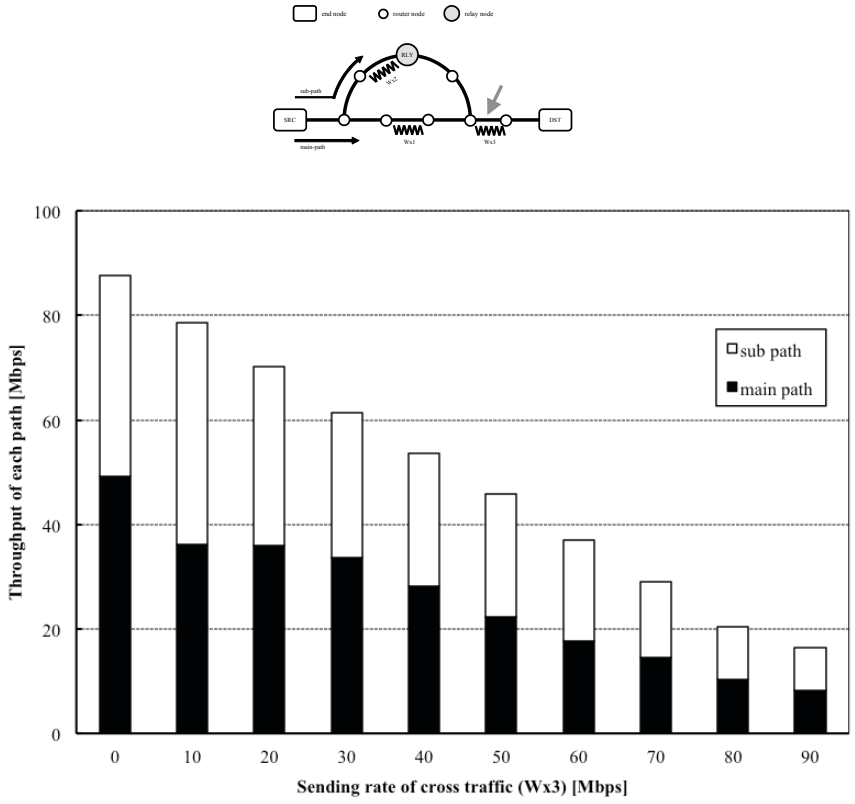


a) SMPC without path priority

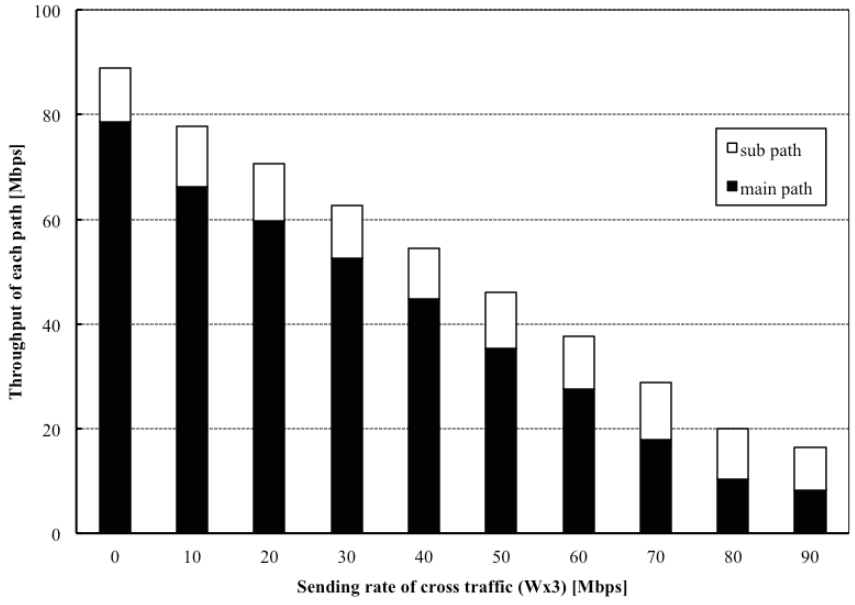


b) SMPC with path priority

Fig. 23 Throughput of each path with cross traffic on the sub-path. (Comparison by having priority control or not)



a) SMPC without path priority



b) SMPC with path priority

Fig. 24 Throughput of each path with cross traffic on the shared link. (Comparison by having priority control or not)

$Wx1$の増加により主経路の可用帯域が減少すると，主経路の受信速度が 30Mbps 近辺まで低下し，代わりに副経路の受信速度が 60Mbps 前後へ上昇することで，主経路が混雑状態にあっても合計受信速度を維持している点は共通している．

ただし，経路間優先制御を行った場合 b）では，$Wx1$の変化直後に合計受信速度が大きく低下し，最終的に元の合計受信速度まで回復するのに約 0.5 秒かかっている．これは，クロストラフィックによって主経路側の受信速度が低下し始めた時点で，副経路側の受信速度が経路間優先制御によって低く抑えられていることに起因している．経路間優先制御を行っていない場合に 45Mbps から 70Mbps へ増速するは 25Mbps 増加すれば良いのに対し，経路間優先制御によって副経路の速度が 10Mbps まで低下していた場合， 70Mbps まで増速するには 60Mbps の増加が必要である．また，受信速度が低いと1トレインを受信完了するのに要する時間が長くなるため，TACK の到着間隔，すなわち送信速度の更新間隔が長くなってしまうことも合計受信速度回復が遅れる要因となっている．

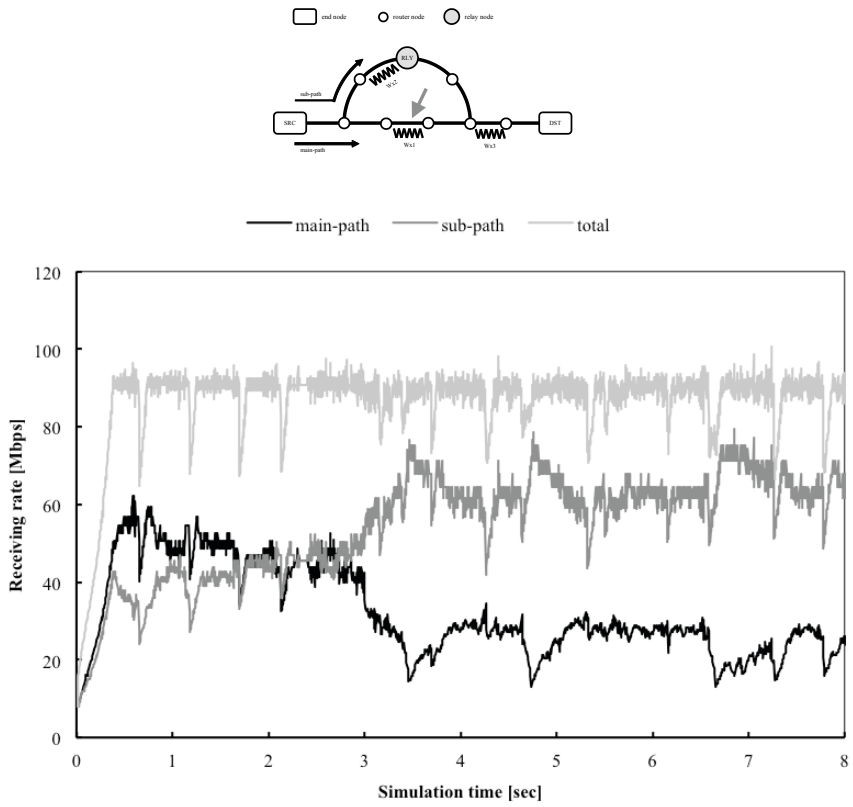
5.3.2 副経路混雑状況変化時

Fig. 26 に，SMPC の通信開始時にはすべてのクロストラフィックが送信されておらず，通信開始 3 秒後に$Wx2$が 70Mbps へ変化した場合の，主経路，副経路が TACK から算出したパケットトレインごとの受信速度，および両経路の合計受信速度の時間変化を示す．グラフの横軸は SMPC 送信開始から 8 秒後（トラフィック発生から5秒後）までのシミュレーション時間を，縦軸は受信速度を表している．

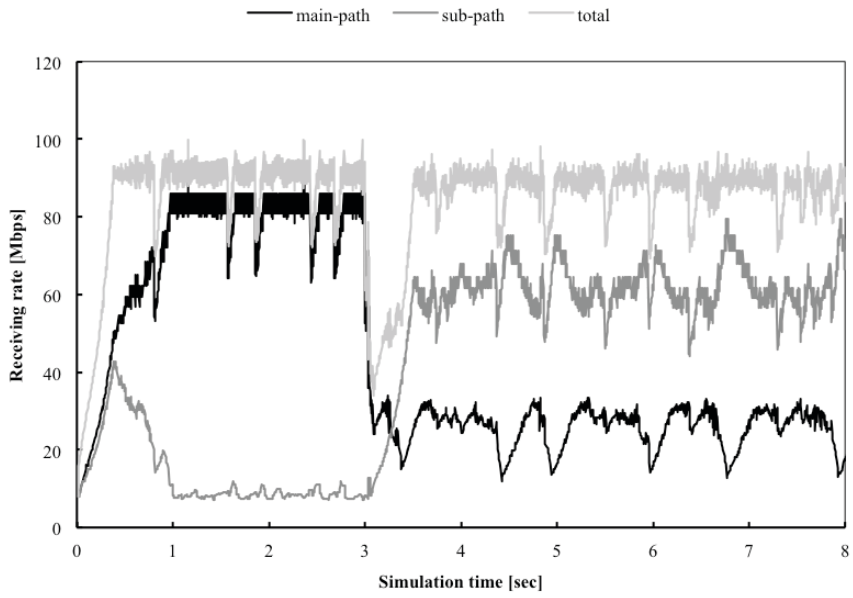
クロストラフィック送信前の状況は，Fig. 25 と同じ条件であるため，経路間優先制御の有無に応じた挙動はほぼ同様である．

$Wx2$の変化に伴い，経路間優先制御を行わない a）において，Fig. 25 a）とは逆に副経路側の受信速度が 30Mbps 近辺まで低下し，代わりに主経路側の受信速度が 60Mbps まで上昇することでトータルスループットを維持している．

一方，経路間優先制御を行った b）では，副経路側には$Wx2$が変化する前から 10Mbps 程度しかトラフィックが流れておらず，$Wx2$が変化しても受信速度に変化は見られない．

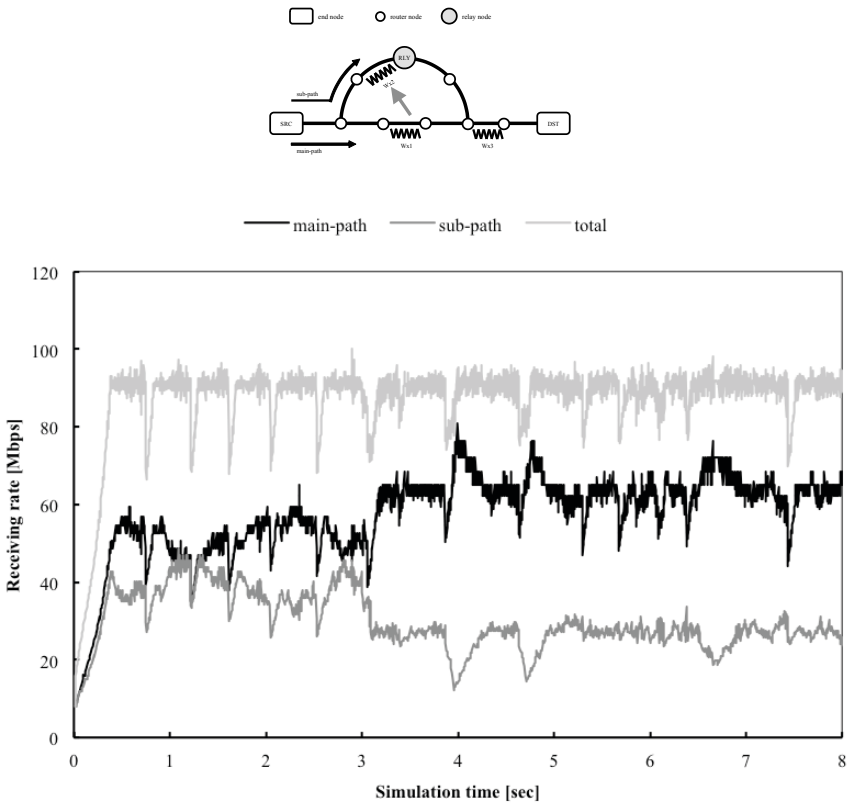


a) SMPC without priority control

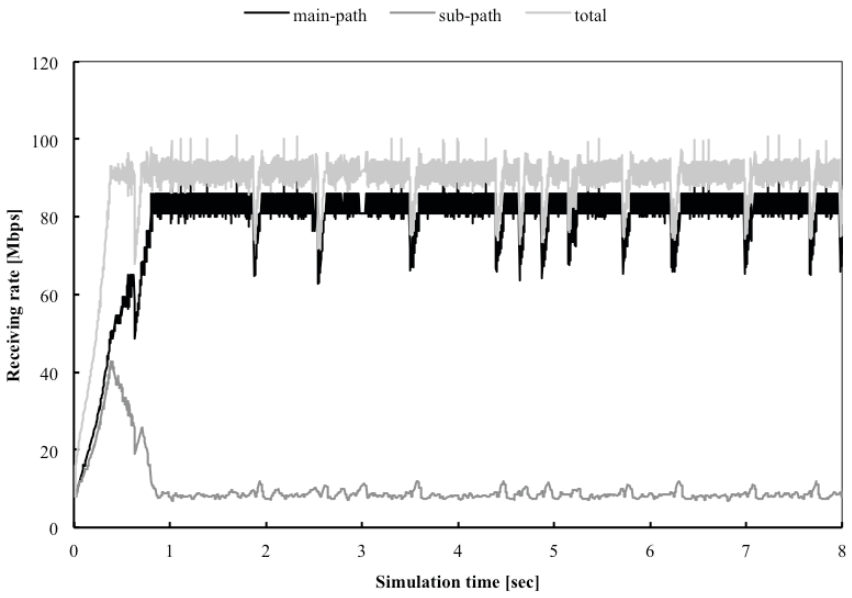


b) SMPC with priority control

Fig. 25 Changes in receiving rate with cross traffic on the main-path. (Comparison by having priority control or not)



a) SMPC without priority control



b) SMPC with priority control

Fig. 26 Changes in receiving rate with cross traffic on the sub-path.
(Comparison by having priority control or not)

5.3.3 共有リンク混雑状況変化時

Fig. 27 に，SMPC の通信開始時にはすべてのクロストラフィックが送信されておらず，通信開始 3 秒後に$Wx3$が 70Mbps へ変化した場合の，主経路，副経路が TACK から算出したパケットトレインごとの受信速度，および両経路の合計受信速度の時間変化を示す．グラフの横軸は SMPC 送信開始から 8 秒後（トラフィック発生から5秒後）までのシミュレーション時間を，縦軸は受信速度を表している．

こちらもクロストラフィック送信前の状況は，Fig. 25 と同じ条件であるため，経路間優先制御の有無に応じた挙動はほぼ同様である．

しかし$Wx3$によって両経路で共有するリンクが混雑するため，クロストラフィックの送信が開始されると，5.3.1 や 5.3.2 とは異なりトータルスループットが 30Mbps 近辺まで低下する．

経路間優先制御を行わない a）ではこの低下した 30Mbps のトータルスループットは両経路によって同じように消費されており，あたかもクロストラフィックが存在しない区間の挙動がそのまま縮小したかのようである．

経路間優先制御を行った b）においては，低下した 30Mbps のトータルスループットの内で主経路側に多くのトラフィックが配分されている．

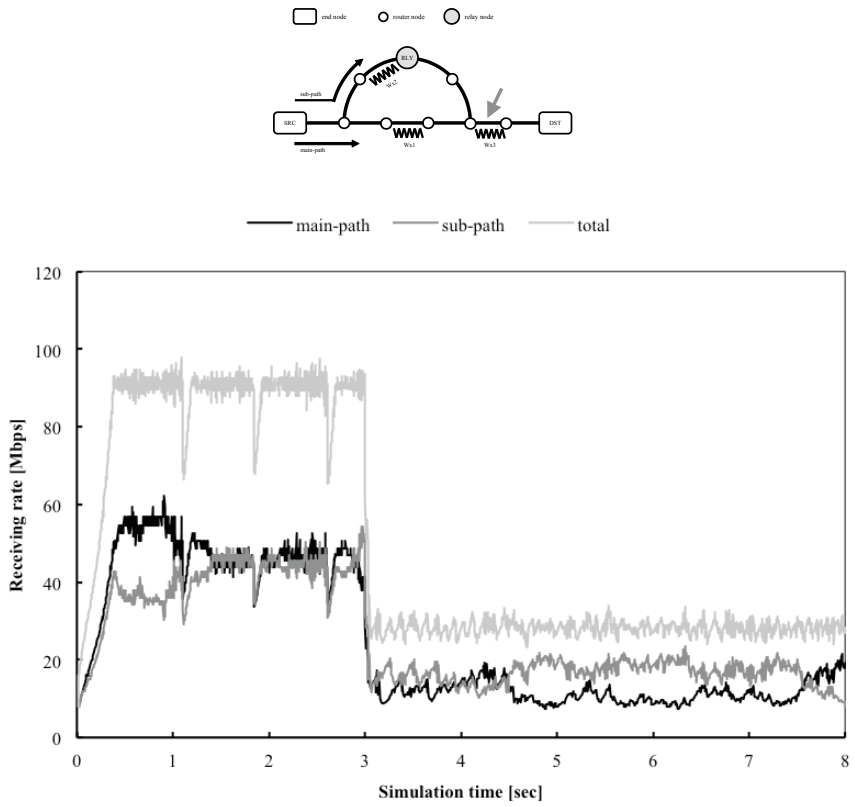
5.4 考察：シミュレーション

5.4.1 経路混雑時の挙動

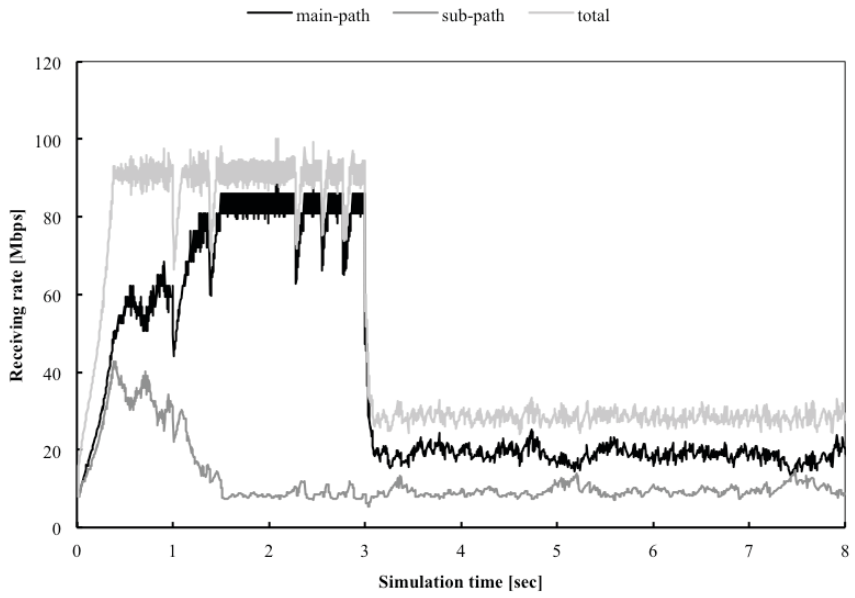
5.2 のシミュレーション結果より，主経路および副経路がそれぞれ単独で混雑した場合は，混雑度に関わらず可用帯域が残っているもう一方の経路を有効に利用してトータルスループットを高く保つという SMPC の動作が確認された．

また，ともに優先制御を行わない Fig. 22　a）と Fig. 23　a）を比較すると，主経路と副経路のスループット配分が逆パターンを示している．経路間優先制御を行わない場合，主経路を使用する通信と副経路を使用する通信は全く同じ送信制御，および輻輳制御を使用しており，主経路混雑時と副経路混雑時で両経路のスループット配分が逆転するのはこのためである．

5.2.3 のように両経路の共有部分が混雑した場合はトータルスループットが下がるが，これは輻輳制御による送信速度の抑制が働いた結果である．このように SMPC は通信のスループットを保証しないベストエフォート型の通信方式だと言える．



a) SMPC without priority control



b) SMPC with priority control

Fig. 27 Changes in receiving rate with cross traffic on the shared link. (Comparison by having priority control or not)

5.4.2 混雑状況変化への応答

5.3.1 および 5.3.2 のシミュレーション結果より，SMPC 通信中に一方の経路が混雑を始めた場合には，いずれの条件においても，もう一方の経路へのトラフィック再配分を動的に実施できている．

ただし，Fig. 25 の a)と b)ではクロストラフィック発生直後の合計受信速度の変化が異なる動きを見せた．これは，クロストラフィックが発生していない状態では，経路間優先制御により副経路側の送信速度が低く抑えられているためである．

SMPC では一定速度で送信されたパケットトレインの受信が完了する毎に TACK によって通信状況が通知され，送信速度の更新が行われる．送信速度が低い状況ではデータパケットの送信間隔が長くなるため，結果的に送信速度更新の間隔も長くなってしまう．こうした傾向は SMPC 送信開始時にも見られるが，送信速度が低い区間で加速に時間がかかる点は SMPC の課題の1つである．

5.4.3 経路間優先制御の評価

Fig. 22 b)では，$Wx1$=0 では大半のトラフィックが主経路を利用して送信されており，$Wx1$の増加に伴い主経路のスループットが低下し，副経路側スループットが増加している．これは，主経路の可用帯域を優先的に利用するという経路間優先制御の挙動が現れたものである．

また，Fig. 23 b)では，$Wx2$の大きさにかかわらず，スループットの配分が変化していない．これは，Fig. 23 の条件では主経路に常に充分な可用帯域が存在するため，$Wx2$の大きさに関わらず経路間優先制御によって主経路が高いスループットを発揮できるためである．

さらに，Fig. 24 b)では，共有経路にかかるクロストラフィック$Wx3$によって通信全体の合計スループットが低下する中であっても，各条件では主経路側のスループットが大半を占める結果になっている．このことから，SMPC の経路間優先制御は，両経路の共有部分が混雑している場合であっても，主経路を優先的に利用する様子が確認できる．

第6章 SMPC 通信：実機実験

SMPC のパラメータ変更が，実機環境において SMPC の送信制御，輻輳制御に与える影響を確認するため，シミュレーションで使用した SMPC 送受信プログラムを実機環境に実装し通信実験を行った．

6.1 実験ネットワーク

実機実験に用いた実験ネットワーク構成を Fig. 28 に示す．

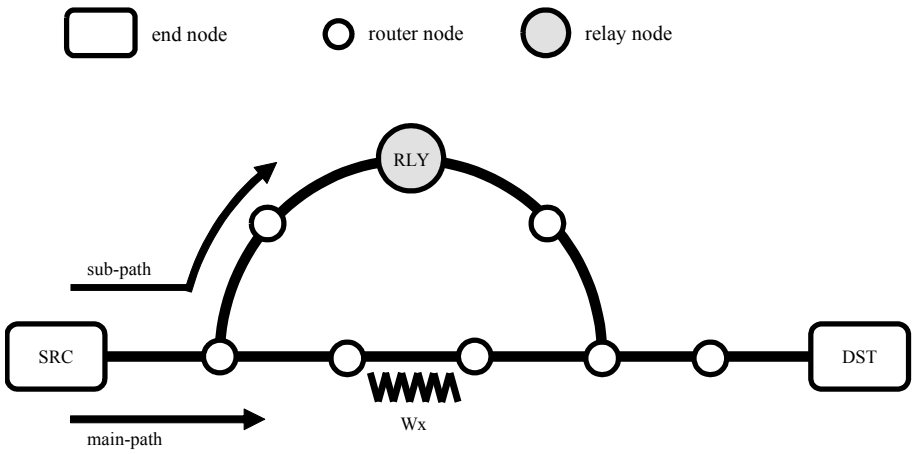


Fig. 28 Assembled network.

送信ノード（SRC）から受信ノード（DST）へデータを送信するために，ホップ数5とホップ数6の2本の経路が利用可能である点は Fig. 21 と同様である．中継ノード（RLY）を含む全てのルータノードには，距離ベクトルアルゴリズムに基づいた経路情報が予め登録されており，特に経路を指定されないデータパケットはホップ数の少ない経路を使用して送信される．SMPC はこちらを主経路とし，ホップ数6の経路を副経路として利用する．

全てのノードは FreeBSD がインストールされた PC　UNIX マシンであり，必要に応じて 100Base-TX 対応のネットワークインタフェースカードがそれぞれ1～3枚搭載されている．

Table 3 に送受信ノードおよび中継ノードのマシンスペックを示す．

Table 3 Machine specification of major nodes.

Node	CPU	MEM	OS	NIC
SRC	Intel Pentium4 3.2GHz	512MB	FreeBSD 4.10-RELEASE	Intel EtherExpress PRO/100B
DST	Intel Pentium4 1.5GHz	256MB	FreeBSD 4.10-RELEASE	Intel EtherExpress PRO/100B
RLY	Intel Pentium4 3.2GHz	512MB	FreeBSD 4.10-RELEASE	Intel EtherExpress PRO/100B × 2

また，主経路の混雑状況を再現するためのクロストラフィックとしてUDPもしくはTCPによる通信を行う．UDP 通信を送信する場合はクロストラフィック送信速度Wxを指定するが，TCP 通信においては TCP 自身が送信制御を行うため特にWxの指定は行わない．

6.2 実験プログラム

実機実験用 SMPC 送受信プログラムには C 言語を利用し，IPv6/UDP を使用する通信アプリケーションとして実装した．

送信ノードにおける各経路への送信モジュールや制御モジュール，受信ノードにおける受信モジュールはそれぞれが独立したスレッドとして並列実行される．

また，主経路へ送信するパケットは通常通りに UDP データグラムとして送信操作を行い，副経路へ送信するパケットについては，パケット生成時に IPv6 経路制御ヘッダタイプ 0 に中継ノードの IPv6 アドレスを指定して送信する．

なお，実機実験用プログラムは SMPC の通信制御についての検証を行う目的で作成されており，経路間優先制御は実装していない．

6.3 予備実験：主経路混雑時の挙動

実験用プログラムの動作を確認するため，主経路に対して送信速度Wxの UDP クロストラフィックが存在する場合に，SMPC により100MB のデータを送信した際の両経路のスループットを計測した．

Fig. 29 にWxを 0Mbps から 10Mbps ごとに 90Mbps まで変化させた 10 パターンについての計測結果を示す．グラフの縦軸はスループット，横軸はWxを表しており，黒塗り部分が主経路を利用した通信のスループット，白塗り部分が副経路を利用した通

信のスループットである.

これはシミュレーションにおける Fig. 22 a)と同様の条件であり, シミュレーション結果 Fig. 22 a)に対して, 実機環境においても同様の結果が得られた.

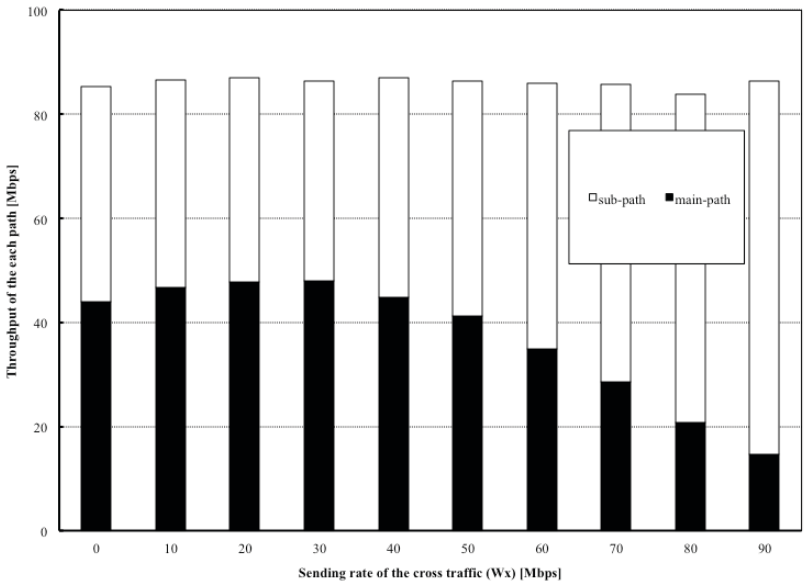


Fig. 29 Throughput of each path with cross traffic on the main-path. (Assembled network)

また, 実機環境で経路上の混雑状況変化に対して SMPC の受信速度がどのように応答するかについても実験を行った. まずクロストラフィックが送信されていない状態で SMPC 通信を開始し, その後Wxとして主経路に 70Mbps の UDP 通信を開始した. この時の主経路, 副経路の受信速度, および両経路の合計受信速度の時間変化を Fig. 30 に示す. ただし, SMPC 送信ノードとクロストラフィック送信ノードが異なるため, 実機実験においてはクロストラフィックの送信開始タイミングを厳密に 3 秒後とすることができず, 約 2.7 秒後よりクロストラフィックの送信が開始されている. こちらはシミュレーションにおける Fig. 25 a)と同様の条件である.

クロストラフィック送信前は主経路, 副経路の受信速度は両者とも 45Mbps 前後となるが, クロストラフィックの送信開始によって主経路の受信帯域が 30Mbps 程度まで低下し, 代わりに副経路の受信速度が上昇して合計受信速度が維持される挙動はシミュレーションと同様である.

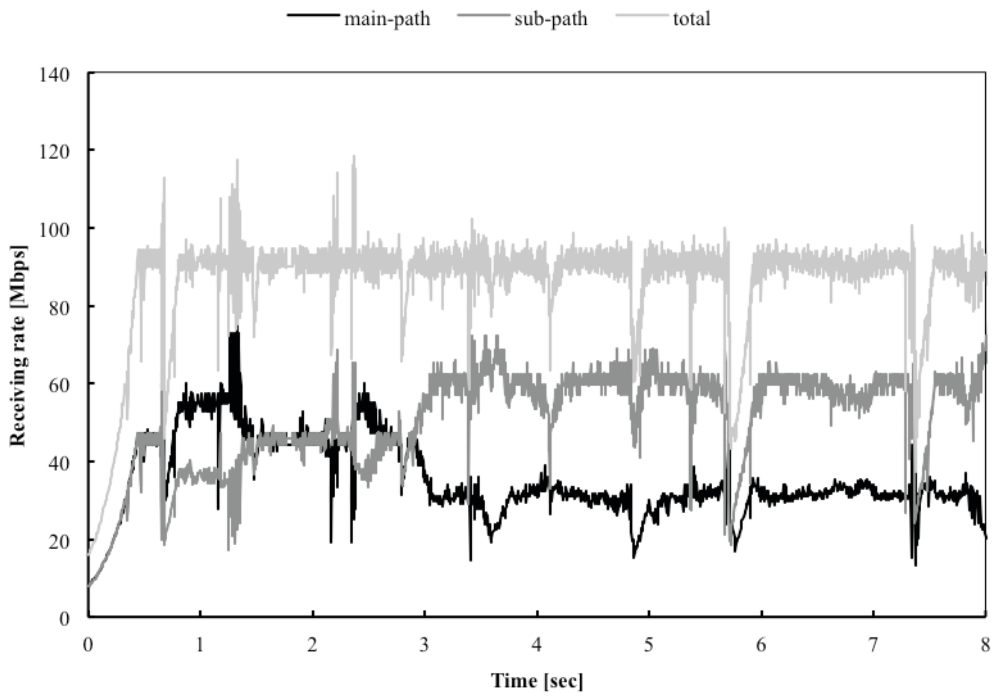


Fig. 30 Changes in receiving rate with cross traffic on the main-path. (Assembled network)

このように, SMPC は実機環境においても主経路の混雑に対して通信速度を維持できている.

6.4 SMPC パラメータの影響

SMPC ではパケットトレイン単位で送信制御を行うため, トレインサイズNや送信速度の増減単位Δwの変化が, SMPC の通信特性に影響を与える.

まず, トレインサイズNの変化が受信速度へ与える影響を確認するため, 送信ノードから受信ノードに対して主経路のみを用いて 100MB のデータを送信した際の各パケットトレインの受信速度Wrを記録した. ここで, Δwは 1.0 であり, また主経路に対してクロストラフィックは発生させていない.

Table 4 はWrが最も安定した部分における連続した 1000 点のWrの平均$Wavg$とその標準偏差σ, およびデータ送信開始から$Wavg$へ到達するまでに要した時間Δtを示している.

Table 4 Effect on the train size on SMPC (1)

Train size	N=4	N=8	N=12	N=16
Wavg [Mbps]	35.20	94.06	93.54	93.04
σ [Mbps]	4.36	2.19	0.89	0.36
Δt [sec]	0.169	0.405	0.536	0.906

Nが 8 より大きい条件では，$Wavg$は 90Mbps を超えた値を示しており，100Mbps の物理帯域を充分使いきれているが，N=4 の場合には35Mbps前後しか利用できていない．σはNが大きくなるに連れて小さくなっており，トレインサイズが大きくなることで受信速度のばらつきが抑えられている．Δtについては，$Wavg$が極端に小さいN=4 の場合をのぞいても，Nが小さいほど短くなる傾向が出ている．

続いて，トレインサイズNの変化がマルチパス通信に対して与える影響を確認するため，データ送信中，主経路に対して常にWxが 90Mbps の UDP クロストラフィックが発生している状態で，送信ノードから受信ノードへ，主副両経路を利用して 100MB のデータを送信した場合の通信スループットを記録した．ここで，Δwは Table 4 と同じく 1.0 である．Table 5 はトレインサイズNに対する主経路，副経路それぞれのスループット，および両経路の合計スループットを示している．

Table 5 Effect on the train size on SMPC (2)

Train size	N=4	N=8	N=12	N=16
main-path [Mbps]	14.50	15.89	15.12	14.56
sub-path [Mbps]	23.11	45.40	61.98	69.76
total [Mbps]	37.61	61.29	77.10	84.32

主経路には90Mbps の UDP クロストラフィックが存在するため，トレインサイズがいずれの場合も主経路のスループットは 15Mbps 前後しか出ていない．N=4 においては副経路のスループットが約 23Mbps までしか上がっていないが，Nが大きくなるにつれて副経路のスループットが大きくなり，N=16 では副経路のスループットが約 70Mbps まで増加している．

最後に，送信速度の増減単位Δwの変化が受信速度へ与える影響を確認するため，クロストラフィックが存在しない条件で，送信ノードから受信ノードに対して主経路のみ

を用いて 100MB のデータを送信した際の各パケットトレインの受信速度Wrを記録した. いずれの場合もNは 16 である.

Table 6 は Table 4 と同様に, Wrが最も安定した部分における連続した 1000 点のWrの平均$Wavg$とその標準偏差σ, およびデータ送信開始から$Wavg$へ到達するまでに要した時間Δtを示している.

Table 6 Effect of Δw on SMPC.

Δw [Mbps]	0.5	1.0	2.0
Wavg [Mbps]	93.04	93.04	93.05
σ [Mbps]	0.33	0.36	0.37
Δt [sec]	1.292	0.906	0.313

Δwの値に関わらず, $Wavg$は高い値を示しており, 100Mbps の物理帯域を有効に利用できている. またσについてもΔwが変化しても目立った影響はでていない. 一方, Δwが小さいほどΔtは長くなる傾向が出ている.

6.5 SMPC 通信が TCP 通信へ与える影響

SMPC の送信制御や輻輳制御が, 競合する TCP 通信に対して与える影響を確認するため, Fig. 28 の実験ネットワークにクロストラフィックとして予め TCP 通信が行われている状況で主経路のみを利用した SMPC 通信を行い, TCP 側スループットの変化を記録した.

SMPC のパラメータΔwを 1.0 で固定し, トレインサイズN=4, 8, 12, 16 の 4 パターンに変化させた場合の TCP スループットの変化を Fig. 31 に示す. グラフの縦軸はデータ 1MB を送信する毎の TCP スループットであり, 横軸は SMPC の送信によりスループットが減少し始める1秒前から 4 秒後までの時間を表している.

また, SMPC によって TCP のスループットが低下した状態での SMPC および TCP の平均受信速度を Table 7 に示す.

いずれの場合も, SMPC 通信が行われていない間は約 90Mps のスループットを記録していた TCP 通信が, SMPC 通信の開始によってスループットを低下させている. ただし低下後の TCP スループットは, N=4 の場合で約 70Mbps, N=8 の場合で約 50Mbps, N=12 の場合で約 40Mbps, N=16 の場合で約 30Mbps となっており, Nが大きくなるほど TCP のスループットが大きく減少する傾向が確認できる.

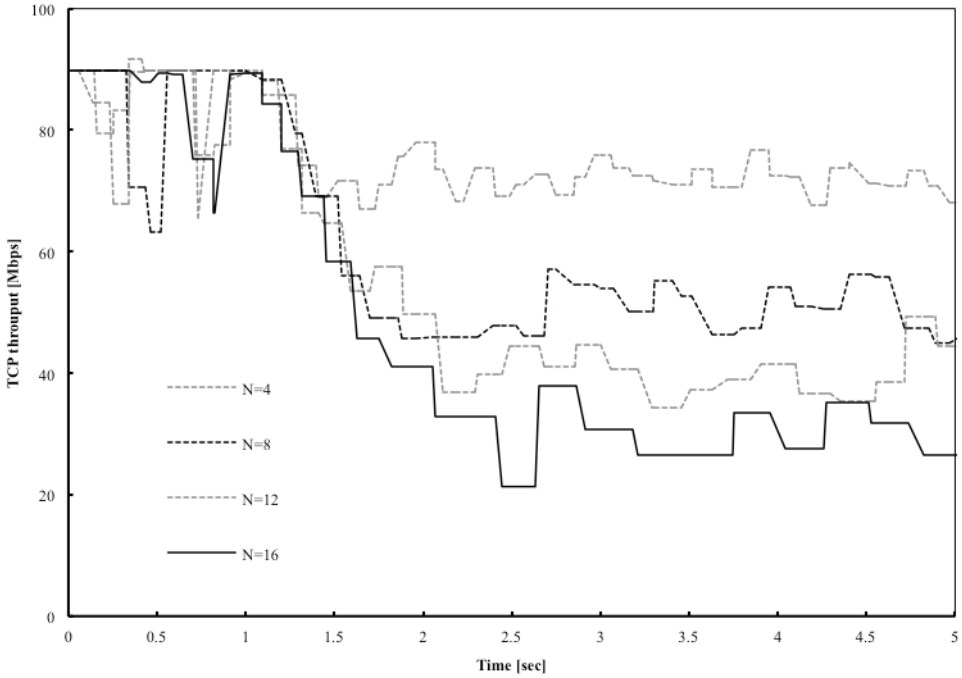


Fig. 31 Influence of SMPC on the TCP traffic.

Table 7 Average throughputs of SMPC and TCP.

Train size	N=4	N=8	N=12	N=19
SMPC [Mbps]	19.06	40.43	53.89	64.37
TCP [Mbps]	71.96	52.36	41.40	29.80

6.6 考察：実機実験

6.6.1 SMPC 通信におけるパラメータ変化の影響

SMPC では，トレインサイズNや帯域増減単位Δwといった SMPC のパラメータを変更することで通信特性が変化する．

Table 6 よりΔwの増加は主にΔtの減少に働く事が確認できるが，これは1度の TACK によって増加する送信速度が大きくなるためである．しかし，受信速度安定時

の受信速度の平均$Wavg$や標準偏差σには大きな変化は見られなかった.

一方, 1つのパケットトレインを構成するパケットの数Nの変化は, SMPC の送信性能に大きな影響を与える. Table 4 では, Nが小さくなるほどΔtが小さくなる結果が確認できる. これは, TACK の返送間隔が短くなり SMPC の送信速度更新が頻繁に発生し, 送信速度がより早く増加するためである. しかし, N=4 では可用帯域が充分存在しているにもかかわらず$Wavg$は 35Mbps 前後で頭打ちになってしまった. また, Table 5 からも主経路にクロストラフィックが存在する状態でマルチパス通信を行った際, Nの減少によってトータルスループットが低下したことが確認できるが, これも十分な可用帯域が存在するはずの副経路側の受信速度が減少した結果であった.

このようにNの減少に伴い$Wavg$やトータルスループットといった通信性能の指標となる数値が低下する原因としては, 通信経路上の遅延時間に対するジッタの影響が考えられる.

経路上のジッタは一定間隔で送信されたパケットにおいて, 受信間隔が伸びる方向に働く場合と縮む方向に働く場合があるため, 長期間パケット到達間隔をサンプリングし平均するとほぼ 0 に収束する [22]. しかし, トレインサイズが小さいとサンプル数が少なくなるため, パケット到達間隔の累積であるパケットトレイン受信所要時間TrのゆらぎJが大きくなる.

一方, SMPC では受信所要時間Trから受信速度Wrを算出し, 該当パケットトレインの送信速度WsとWrとの比Rが一定範囲($Rd \leqq R \leqq Ra$)に収まっている場合にWsを増加させる, という動作をする. このとき, Trに対するジッタはWr, ひいてはRの値に影響与えることになり, 送信速度Wsが定まると, 混雑状況を正しく計測するために許容可能なジッタの大きさJcが定まる.

もし, 経路上の実際のTrに対するジッタJが$J \leq Jc$以下であれば, SMPC が経路の混雑状況を正しく推定でき, SMPC はWsを増加させることが可能である. しかし, $J > Jc$になると混雑していない経路であっても, RがRd以下や Ra以上と算出されてしまい, 結果としてWsの増加が抑制される.

また, 送信データサイズが固定である場合, 受信速度Wsと受信所要時間Tsは反比例の関係にあるため, Wsが大きいほどJcは小さくなる. 一方, 上記の通りNが小さくなるほどJは大きくなるため, Wsが大きくなると, $J \leqq Jc$となるために要求されるNもまた大きくなる. Table 4 ではNが 12 や 16 の場合に 100Mbps という経路の物理帯域をほぼ利用しきれているが, より高速で通信を行うには, より大きなNが要求される可能性がある.

また, 同じWsであってもJが大きなネットワークでは, $J \leqq Jc$となるNはやはり大きくな

る. つまり, 不安定なネットワーク環境で通信を行う場合, やはりより大きなNが必要となる可能性がある.

6.6.2 TCP Friendliness

TCP は経路の混雑状況を悪化させないための輻輳制御を行っている [23]. 上位層で制御されない UDP など輻輳制御を行わない通信と競合した場合, 通信パフォーマンスが低下してしまう. このように競合した通信に悪影響を与える通信方式は TCP Friendly ではない, と表現される.

現在, インターネットを通じて行われる通信の大半は TCP を利用しており, TCP Friendly ではない通信はインターネット通信全体に悪影響を与えることになる. そのため, 新しい通信方式の提案にあたっては TCP Friendly であることが要求される.

SMPC は TCP と同じく輻輳制御を行う通信方式であるため, 輻輳制御を行わない純粋な UDP と異なり競合する通信に対して一方的に帯域を専有することはない. しかし, その輻輳制御の考え方は TCP とは大きく異なっている.

パケットロスをトリガとして輻輳を検知する TCP に対して, SMPC ではパケットロスだけでなく受信速度の低下により, 早期に輻輳の兆候を検知することができる. しかし, 輻輳の検知はパケットトレイン単位で行われており, 輻輳状況の検知にはトレインサイズNに応じたタイムラグが生じる. Fig. 31 および Table 7 においても, トレインサイズNの増加に応じて競合する TCP 通信のスループットが低下している様子が確認された.

TCP Firendliness を維持するためにはNを含む SMPC パラメータの適切な選択が必要となる.

第7章 考察

7.1 SMPC の利点

SMPC のエンドノード側のメリットは，送信ノード，受信ノードが共にシングル NIC，単一の IP アドレスのみの環境であってもマルチパス通信が可能な点である．マルチホーミングや複数 IP アドレスが不要であるため，家庭でブロードバンド環境を利用する一般的なデスクトップ PC 環境でも利用できる．

一方，ネットワークインフラ側のメリットは，中継ノード以外の経由ルータは基本的に通常のパケット転送処理を行うだけでよく，中継ノードについてもインターネット層のソースルーティング処理を行えば良い点である．その結果，マルチパス通信を実現する際にネットワークインフラへ与える影響が小さく，スケーラビリティに優れている．このように，SMPC ではエンドノード側とネットワークインフラ側のメリットを両立させることが可能である．

7.2 SMPC の通信特性

SMPC では，通信制御における輻輳判定や速度調整はパケットトレイン毎に行なっている．1つのパケットトレインに含まれるデータパケットは同じ経路で送信されるため受信ノードに送信順に到着する．そのため SMPC では，二つの経路からデータパケットが順不同に到着しても，輻輳制御へは影響しない．また，パケットの通信帯域を増減させるため，送出間隔による送信速度制御を行なっている．

制御モジュールは，各経路別々に輻輳の検知と送信速度の増減判定を行なっている．そのため，通信が開始されると，各経路の通信速度は等しく増加していき，各経路の送信モジュールは未送信データを均等に奪い合う．その結果，送信元インタフェースのデータリンクや経路の共通部分で最大帯域に達したとき，両経路は最大帯域の 1/2 で送信するようになる．

主経路が混雑してくると制御モジュールは主経路の通信速度を減速するが，空いている副経路は増速させ続けるため，TACK の返ってくる時間は副経路側が相対的に短くなる．その結果，未送信データを副経路の送信モジュールが取出す頻度が高くなり，トラフィックは副経路側に偏っていく．結果として副経路側に十分な可用帯域があれば，主経路が混雑しても通信全体のトータルスループットを維持することができる．

しかしながら，二つの経路で通信速度が異なる可能性があるため，全体としてのパケット到着順は維持されない．

また，Fig. 22 における$Wx1$=0 時の主経路利用率γは，経路間優先制御を行わない場合 a）が約 0.52 であったのに対し，経路間優先制御を行った場合 b）は 0.88 であった．この時，Fig. 21 よりL=5，Δl=1，また，1パケット当たりのデータサイズSが 1,024[byte]であるため 100MB を送信した場合の全送信パケット数Q=102,400 である．これらを式(3)へ代入し，SMPC が副経路を使用することによって増加したパケット転送コストを求めると，経路間優先制御を行わなかった場合が 49,152 であったのに対し，経路間優先制御を行った場合は 12,288 であった．この結果，主経路に充分な可用帯域が存在する場合に，経路間優先制御によってパケット転送コスト増加を約 1/4 に低減可能であることが確認できた．

7.3 中継ノード選択と経路の disjoint 性

マルチパス通信が使用する複数の経路において，経路の disjoint 性という指標が存在する [24]．2 つの経路が共有するリンク/ノードが存在しない場合，それらの経路はリンク/ノードについて独立（link/node disjoint）であり，disjoint 性の高い経路は共有するリンクやノードが少ない．エンドノードが搭載するネットワークインタフェースが 1 枚だけであることが想定される SMPC では，完全に disjoint な経路を使用することを保証はできないが，5.2.3 に示した通り，主経路と副経路が共有するリンクが混雑した場合，通信全体のスループットが低下してしまうため，SMPC においても disjoint 性の高い経路を使用することが望ましい．

一般的な IP ルーティングにより求められた最適経路を主経路として使用する SMPC では，経路間の disjoint 性は副経路の選択によって決まる．しかし，SMPC では IPv6 経路制御ヘッダを利用したルーズ・ソースルーティングにより副経路での通信を行うため，送信ノードから中継ノード，中継ノードから最終宛先ノードへ至る経路をエンドノード側で制御することはできない．SMPC において，副経路の選択はすなわち中継ノードの選択でるため，中継ノードの選択は非常に重要である．

Fig. 32 における A のように，disjoint 性があまりに小さい中継ノードを選択してしまった場合，マルチパス通信の有効性をまったく発揮することができない．一方，disjoint 性を高めるためとはいえ，Fig. 32 における B のように国内通信に対して地球を 1 周させるような中継ノードを選択することもまた適切とは言えない．

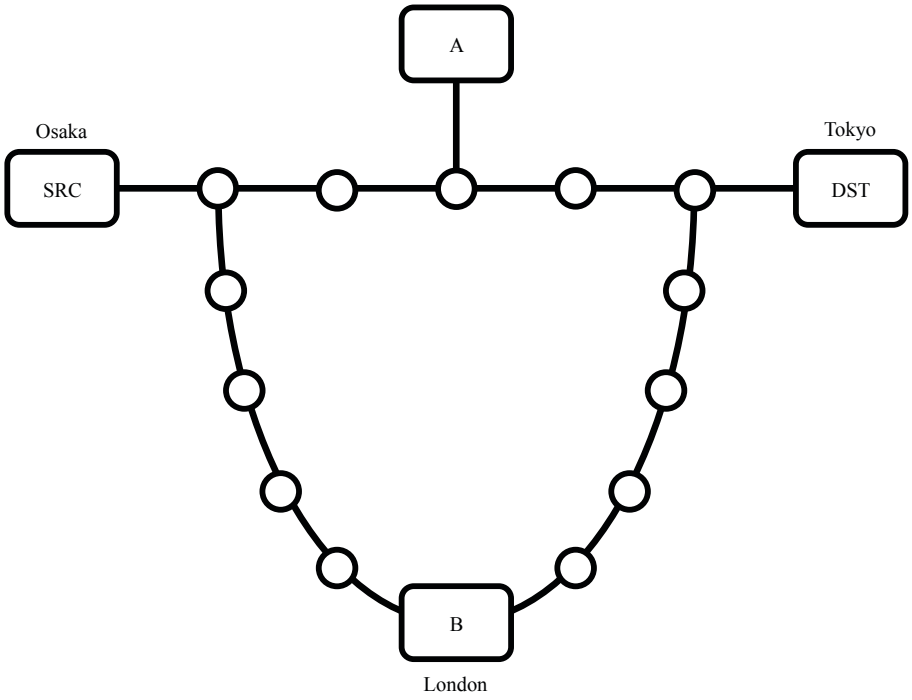


Fig. 32 Inadequate relay nodes.

現時点のSMPC通信実験では中継ノードが既知の情報として与えられているが，実用化に向けては中継ノード情報をどのようにエンドノードへ提供するか，またより望ましい副経路を利用するための中継ノードをどのように決定するか，という課題が残されている.

7.4 ネットワークの「ただ乗り問題」

SMPCでは，各利用者がそれぞれ自分のPCを中継ノードとして提供することも技術的に可能であるが，そうした場合オーバレイネットワークと同様にネットワークの「ただ乗り問題」[25]が生じる.

オーバレイネットワークとは，インターネット上に仮想のネットワークを構成する技術である. Fig. 33 に示すように，ISP A～C が共通の上位 IX に接続している場合，ISP A のユーザ A が ISP B のユーザ B へデータを送信しようとするには，ISP A から IX を通ってそのまま ISP B へ向かう経路が最適経路である. さまざまな理由で，ユーザ A とユーザ B が直接通信できない場合，ユーザ A ともユーザ B とも通信可能な ISP C のユーザ C がいれば，ユーザ C を経由してユーザ A から B へ通信が可能となる. この時，ユーザ A─C─B というオーバレイネットワークが構築されたことになる.

ユーザ C の存在によりユーザ A, B は通信が可能となるが, ISP C にとっては自身のユーザではない A, B のために, ISP C と上位 IX との間の通信帯域を利用されることになる. 一般的な ISP は上位 IX との間でデータ転送量に応じた従量制の契約を結んでいる場合が多いため, このオーバレイネットワークの利用によっても ISP C に運用コストが発生するが, 直接契約を結んでいないユーザ A, B に対して対価を請求することができない. Fig. 33 におけるユーザ C の存在は, SMPC が利用するルーズ・ソースルーティングにおいては中継ノードに置き換えることができる.

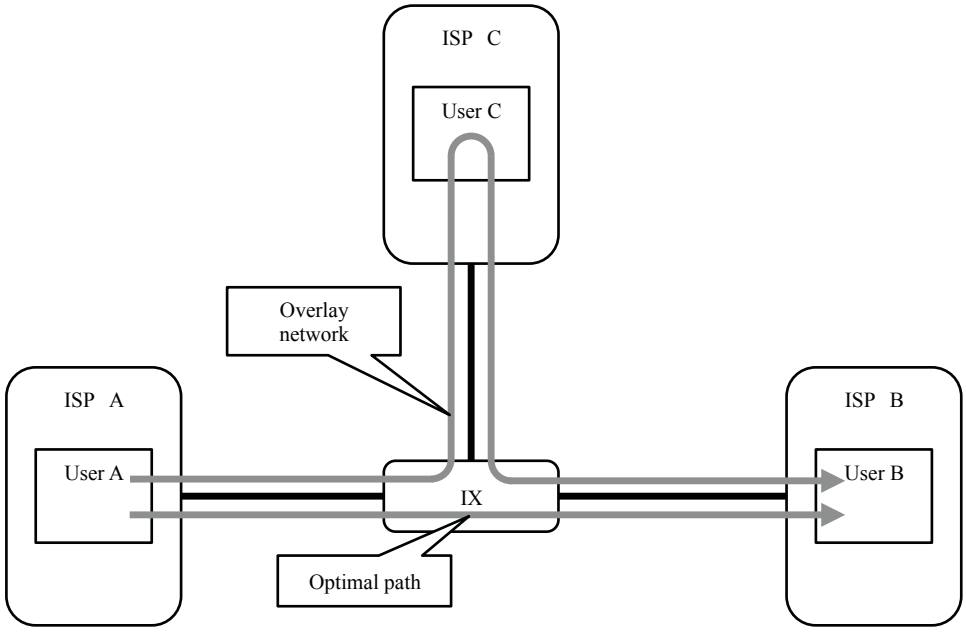


Fig. 33 Overlay network.

ただし, ネットワークの「ただ乗り問題」の本質は技術的な問題ではなく, 経済的な問題というべきものであり, その対策も技術的にではなく, 経済的に行われるべきである.

例えば, 複数の上位 IX（Internet eXchange）と接続している ISP が, 複数ある ASBR（AS border router：AS 境界ルータ）を自身の利用者に対して中継ノード候補として提供する方式や, 異なる IX と接続している ISP 同士が直結回線を持ち, お互いの ASBR を中継ノードとして提供するといった方式が考えられる.

7.5 経路制御ヘッダタイプ 0 のセキュリティ問題

現バージョンの SMPC が使用している IPv6経路制御ヘッダタイプ 0 は, IPv6 規格[21]の一部として定義されているが, セキュリティ上の問題も指摘されている [26].

影響の大きなセキュリティ問題の1つは仕様上の経由ルータ数制限がないため DoS

攻撃に利用可能というものである．インターネット上で一般的な IP パケットサイズ（1500byte）内に，IPv6経路制御ヘッダタイプ0の経由ノードは80回以上記述することが可能である．すなわち，攻撃者は Fig. 34 に示すように1つのパケットを任意の2つのノード間で 40 往復させることができる．もし，こうしたパケットを 25Mbps のスピードで送信できれば，1GB の帯域を飽和させることも可能である．

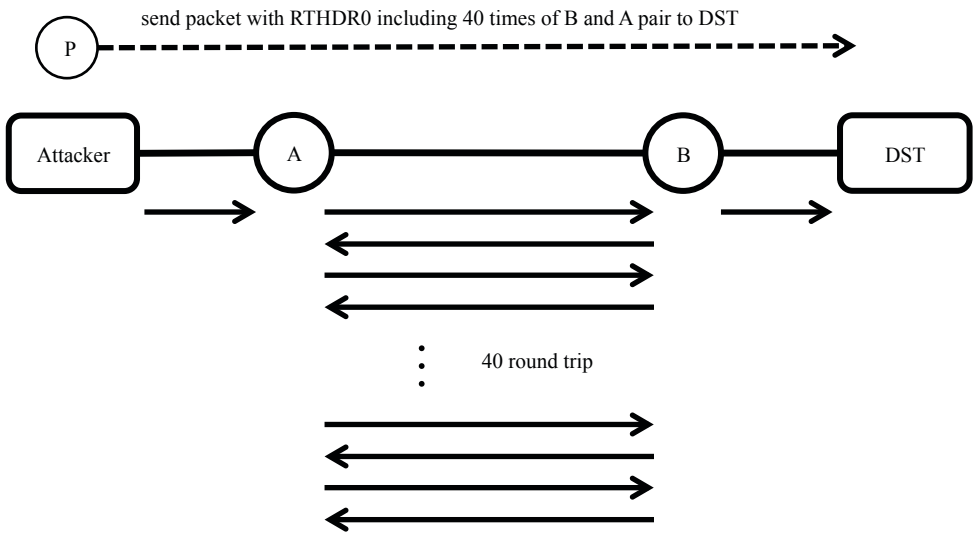


Fig. 34 DoS attack by using IPv6 routing header type0

もう1つの大きな問題は，ファイアウォールなどによるアクセス制御をすり抜けることが可能となる点である．例えば， Fig. 35 のように，あるネットワークに外部と通信可能なノード A とファイアウォールによって外部と遮断されているノード B があった場合に，攻撃者はノード A を中継ノードとして，ノード B 宛のパケットを送ることで，本来外部からは直接通信できないはずのノード B を攻撃可能となる．

このような指摘に応じる形で，2007 年 12 月には，IPv6 経路制御ヘッダタイプ 0 を非推奨とする RFC5095 [27]が公開された．この RFC5095 では，通信機器に対して IPv6 経路制御ヘッダタイプ 0 を含むパケットの遮断を推奨しており，現状 IPv6 経路制御ヘッダタイプ 0 を用いたインターネット通信は困難な状況にある．

ただし，RFC5095 においては「全ての（タイプ 0 位外の）IPv6 経路制御ヘッダタイプを遮断してはならない」とも明示されており，これは IPv6 経路制御ヘッダという仕組み全体について非推奨とするものではない．

SMPC などソースルーティングによるマルチパス通信を行うためには，こうしたセキュリティ問題に対応した新たな経路制御ヘッダタイプの考案が必要である．

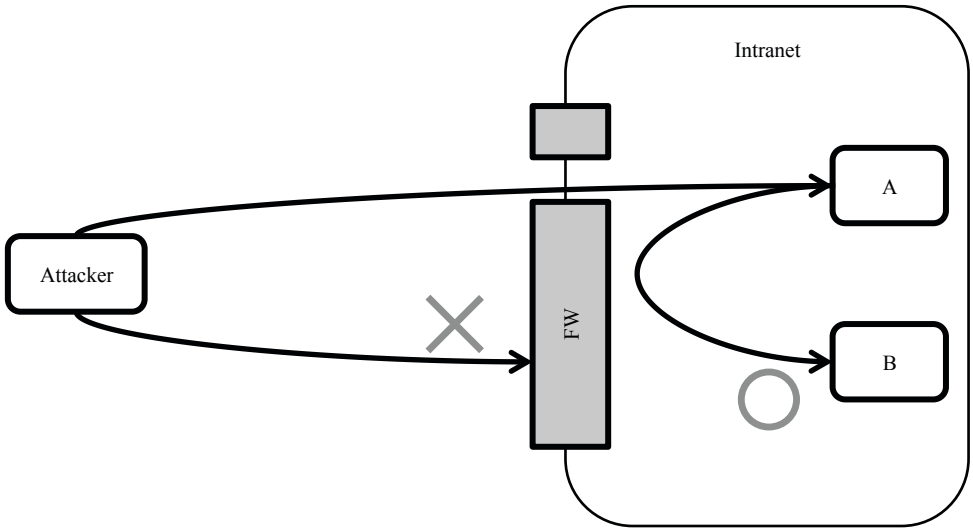


Fig. 35 Loophole in security policy

第8章 結論

本研究では，単一経路の混雑時にも高いスループットを維持可能な複数経路同時送信方式 SMPC を提案し，シミュレーション，および実験より SMPC の効果や通信特性を明らかにした．

SMPC は副経路の利用に IP ソースルーティングを使用するため，エンドノードはシングル NIC，IP アドレス1つという構成でも利用可能である．また，副経路上の中継ノード以外の既存ネットワーク機器に対する機能追加を必要としないため利用におけるハードルが低い通信方式である．また，SMCP はネットワークの各所に発生する混雑に対して，各経路の可用帯域を有効に活用することで高いスループットを維持すると同時に，経路間優先制御によりネットワーク全体の資源消費を抑えることが可能である．

SMPC 通信が採用したパケット送信間隔を利用した受信速度の計測，および送受信速度の変化から輻輳を検知する通信制御手法は，実機環境においても輻輳回避に有効である．ただし，通信経路の品質や競合する通信との公平性の観点から，より適切な SMPC パラメータの選択を行う必要がある．SMPC パラメータが通信特性に与える影響を調査するには，より長距離や低品質環境での実験を行う実環境で特定条件の大規模ネットワークを構築することは困難であり，RNS 等のネットワークシミュレータが有用である．

今後は，SMPC のより詳細な通信特性を解明するとともに，高信頼性通信に必要なロストパケットの再送処理などを組み込み，通信プロトコルとしての完成を目指す．

謝辞

本論文を作成するにあたり，岐阜大学工学部電気電子・情報工学科伊藤昭教授，金子美博准教授，原山美知子准教授よりご指導を賜りました．ここに感謝の意を表します．

また，実機実験環境の構築などでご協力いただいた歴代の原山研究室ネットワークグループのメンバー，および，社会人としての博士後期課程進学を薦めていただき，在学中は業務負荷等の面でご配慮頂いた岐阜大学学術国際部情報戦略課長を始めとする情報戦略課員各位に感謝の意を表します．

参考文献

[1] Cisco Systems. (2014, June) Visual Networking Index (VNI). [Online]. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/VNI_Hyperconnectivity_WP.html

[2] Stefan Savage, Andy Collins, and Eric Hoffman, "The end-to-end effects of Internet path selection," in *ACM SIGCOMM Computer Communication Review*., Oct. 1999, vol. Volume 29 Issue 4, pp. 289-299.

[3] Renata Teixeira, Keith Marzullo, Stefan Savage, and Geoffrey M Voelker , "Characterizing and Measuring Path Diversity of Internet Topologies," in *ACM SIGMETRICS Performance Evaluation Review*., Jun. 2003, vol. Volume 31 Issue 1, pp. 304-305.

[4] 田中 昌二 and 原山 美知子, "IPv6 経路制御ヘッダを利用したソースルーティングによる通信品質向上手法の提案," in *子情報通信学会技術研究報告. IA, 108(74)*., 2008, pp. 55-60.

[5] 田中 昌二 and 原山 美知子, "IPv6 ソースルーティング検討のためのネットワークシミュレータ," in *電子情報通信学会技術研究報告. NS, 108(392)*., 2009, pp. 13-16.

[6] 田中 昌二, 山田 真貴, and 原山 美知子, "IPv6 ソースルーティングを用いたマルチパスデータ転送方式の提案," in *電子情報通信学会技術研究報告. CQ, 109(373)*., 2010, pp. 79-84.

[7] Akiji Tanaka, "Effects of length and number of paths on simultaneous multi-path communication," in *Proceedings of 2011 IEEE/IPSJ 11th International Symposium on Applications and the Internet*., 2011, pp. 214-217.

[8] 田中 昌二 and 原山 美知子, "多経路同時送信方式 SMPC における経路間優先制御手法," in *電子情報通信学会技術研究報告. NS, 113(472)*., 2014, pp. 143-148.

[9] (2014, Oct.) AS6447 BGP Routing Table Analysis Report. [Online]. http://bgp.potaroo.net/as2.0/bgp-active.html

[10] (2014, Aug.) Internet Touches Half Million Routes: Outages Possible Next Week. [Online]. http://research.dyn.com/2014/08/internet-512k-global-routes/

[11] Atul Adya, Paramvir Bahl, Jitendra Padhye, Alec Wolman, and Lidong Zhou, "A Multi-Radio Unification Protocol for IEEE 802.11 Wireless Networks," in *BROADNETS '04 Proceedings of the First International Conference on Broadband Networks*., Oct. 2004, pp. 344-354.

[12] Hsin-hung Lin, Chih-wen Hsueh, and Guo-Chiuan Huang, "BondingPlus: Real-Time Message Channel in Linux Ethernet Environment Using Regular Switching Hub," in *9th International*

Conference, RTCSA 2003., Feb. 2004, pp. 176-193.

[1 Kameswari Chebrolu , Bhaskaran Raman , and Ramesh Rao, "A Network Layer Approach to Enable
3] TCP over Multiple Interfaces," in *Wireless Networks*., Sep. 2005, vol. Volume 11 Issue 5.

[1 Kun-chan Lan and Chen-Yuan Li, "Improving TCP performance over an on-board multi-homed
4] network," in *Wireless Communications and Networking Conference (WCNC) 2012*., 2012.

[1 Dhananjay S. Phatak and Tom Goff , "A Novel Mechanism for Data Streaming Across Multiple IP
5] Links for Improving Throughput and Reliability in Mobile Environments," in *IEEE INFOCOM 2002*., 2002.

[1 Miao Xue, Deyun Gao, Wei Su, Sidong Zhang, and Hongke Zhang, "E2EMPT: A transport layer
6] architecture for end-to-end multipath transfer," in *Broadband Network and Multimedia Technology (IC-BNMT), 2010 3rd IEEE International Conference on. IEEE*., 2010, pp. 37–41.

[1 Samar Shailendra, R. Bhattacharjee, and Sanjay K. Bose, "MPSCTP: A Simple and Efficient
7] Multipath Algorithm for SCTP," in *Communications Letters, IEEE, vol. 15, no. 10*., 2011, pp. 1139–1141.

[1 Karim Habak, Khaled A. Harras , and Moustafa Youssef , "Bandwidth Aggregation Techniques in
8] Heterogeneous Multi-homed Devices: A Survey," in *CoRR abs/1309.0542*., 2013.

[1 Dominik Kaspar, Kristian Evensen, Paal Engelstad, and Audun F. Hansen, "Using HTTP pipelining
9] to improve progressive download over multiple heteroge- neous interfaces," in *Communications (ICC), 2010 IEEE International Conference on*., 2010, pp. 1-5.

[2 Information Sciences Institute University of Southern California. (1981, Sep.) RFC791 INTERNET
0] PROTOCOL - DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION -. [Online]. http://www.ietf.org/rfc/rfc791.txt

[2 IETF Network Working Group. (1998, Dec.) RFC2460 Internet Protocol, Version 6 (IPv6)
1] Specification. [Online]. https://www.ietf.org/rfc/rfc2460.txt

[2 Luiz Magalhaes and Robin Kravets, "Transport Level Mechanisms for Bandwidth Aggregation on
2] Mobile Hosts," in *ICNP '01 Proceedings of the Ninth International Conference on Network Protocols*., 2001, pp. 165-171.

[2 Van Jacobson and Michael J. Karels, "Congestion Avoidance and Control," in *ACM SIGCOMM
3] Computer Communication Review*., Aug. 1988, vol. Volume 18, Issue 4, pp. 314-329.

[2 Sung-ju Lee and Mario Gerla, "Split Multipath Routing with Maximally Disjoint Paths in Ad hoc

4] Networks," in *Communications, 2001. ICC 2001. IEEE International Conference on*., 2001, vol. 10, pp. 3201-3205.

[2 長谷川 剛, 小林 正好, 村田 正幸, and 村瀬 勉, "オーバレイルーティングに起因するネットワー
5] クただ乗り問題に関する一検討," in *電子情報通信学会技術研究報告. IN, 情報ネットワーク 106(420)*., Dec. 2006, pp. 133-138.

[2 Philippe Biondi and Arnaud Ebalard. (2007) SecDev.org. [Online].
6] http://www.secdev.org/conf/IPv6_RH_security-csw07.pdf

[2 IETF Network Working Group. (2007, Dec.) RFC5095 Deprecation of Type 0 Routing Headers in
7] IPv6. [Online]. https://www.ietf.org/rfc/rfc5095.txt

[2 Preethi Natarajan, Nasif Ekiz, Paul D. Amer, Janardhan R. Iyengar, and Randall Stewart, "Concurrent
8] Multipath Transfer Using SCTP Multihoming: Introducing the Potentially-Failed Destination State," in *NETWORKING 2008 Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*., 2008, pp. 727-734.

図索引

表索引

付録 A RdおよびRdの選択

ある経路において送信速度Ws[Mbps]を十分に通過させるだけの可用帯域が存在する場合，理想的には受信速度WrはWsと等しくなる。しかし，実際には十分な可用帯域が存在した場合であっても，経由ルータでのキューイング遅延など，様々な条件によりWrは変化する。そこで SMPC では送受信速度比$R = Wr/Ws$を定義し，ある経路において送受信速度比Rが$Rd \leq R \leq Ra$である場合に，該当経路が混雑していない（混雑度 NONE）と判断する。すなわち，Rd，Raの指定によって混雑度が NONE と判断される許容範囲が定まる。

Rd，Raの選択が SMPC の通信特性に与える影響を確認するため，本論文のシミュレーション実験と同様のシミュレーションネットワーク上で，混雑度が NONE となる R の許容範囲を理想的な条件である 1 に対して±1%（$Rd = 0.99$，$Ra = 1.01$）から±10%（$Rd = 0.90$，$Ra = 1.10$）まで 1%ずつ変化させた場合について，SMPC により 100MB のデータ送信を行うシミュレーション実験を行った。なお，Rd，Ra以外のパラメータは本論文のシミュレーション実験と同様であり，クロストラフィックは送信されていない。また，シミュレーション実験は各条件について 10 回ずつ行った。

Fig. A. 1 は各条件における平均パケットロス率，および平均スループットをプロットしたものである。横軸は横軸はRの許容範囲を，左縦軸がパケットロス率，右縦軸がスループットを表している。

グラフより，Rの許容範囲を狭くとることでパケットロス率を低く抑えられていることがわかる。パケットロス率は通信品質の基準として用いられており，パケットロス率が高い状態は通信品質の悪化を意味する。そのため，通信品質を基準としてみた場合，R 許容範囲は狭くとることが望ましい。しかし，一方でR許容範囲を狭くとることによりスループットも低下しており，通信パフォーマンスの点からはRの許容範囲をある程度高く設定することが望ましい。

通信品質とパケットロス率については，ITU-T SG12 による標準化が行われている[1]が，これは通信経路自体の品質基準であり，通信そのものによって引き起こされるパケットロスについて述べたものではない。

実際の通信中に発生するパケットロス率としては，ストリーミングを想定したものであるが概ね 1%以下であれば良好であるという指針がある[2]。そこで本論文におけるシミュレーション実験および実機実験においては，パケットロス率 1%以下という条件の中で

[1] http://www.itu.int/en/ITU-T/studygroups/2013-2016/12/Pages/default.aspx
[2] http://sdu.ictp.it/pinger/pinger.html

スループットが一番高い許容範囲±5%となる$Rd = 0.95$, $Ra = 1.05$を採用した。

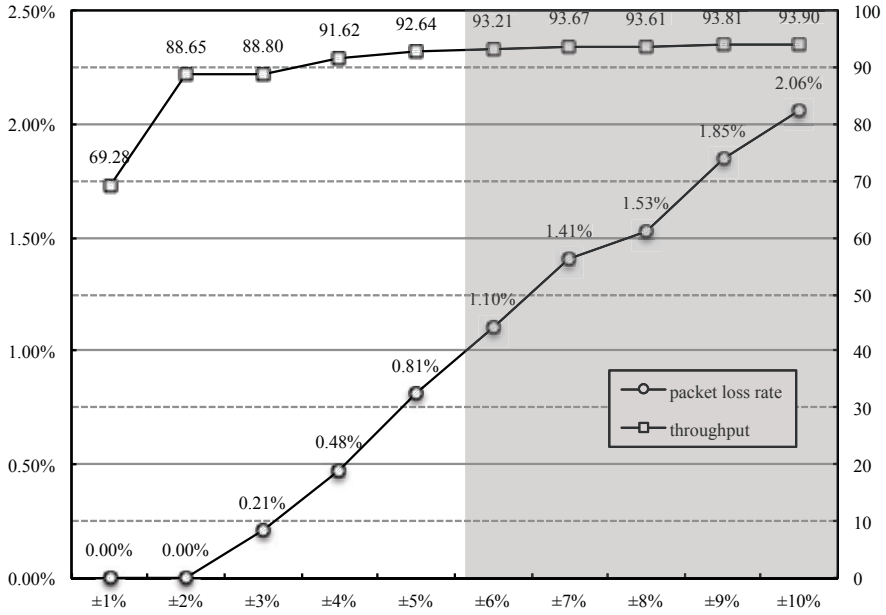


Fig. A. 1 Packet loss rate and throughput.

Fig. A. 2 は，TACK 受信時の混雑度判定結果の割合を示したものである。横軸はR許容範囲を，縦軸は判定結果のパーセンテージを表している。

グラフより，R 許容範囲を広く取ればとるほど，パケットロス検出による CONGESTED 判定が増加しており，この結果は Fig. A. 1 のパケットロス増加と対応するものである。

Fig. A. 1 において，スループットが大きく低下している R 許容範囲±1%では，Wr が Ws よりも大きく計測される MAYBE 判定が全体の 6 割を超えている。混雑度 MAYBE では予防的減速が行われるため， こうした状況下では SMPC が送信速度を十分に上昇させることができない。

±2%の条件では混雑度 NONE 判定が多く検出されている一方で，CONGESTED 判定は検出されていない。CONGESTED 判定が少ないことは良いことではあるが，可用帯域を十分に使い切る前に過剰な減速が行われてしまっていることを表しており，Fig. A. 1 の結果でも平均スループットが 90Mbps に届いていない。

このように，Rd, Raの選択によって SMPC の通信特性が大きく変化する。ここではシミュレーション結果をもとにRd, Raの選択を行ったが，実機環境で最適なRd, Raの選択手法についても検討が必要である。

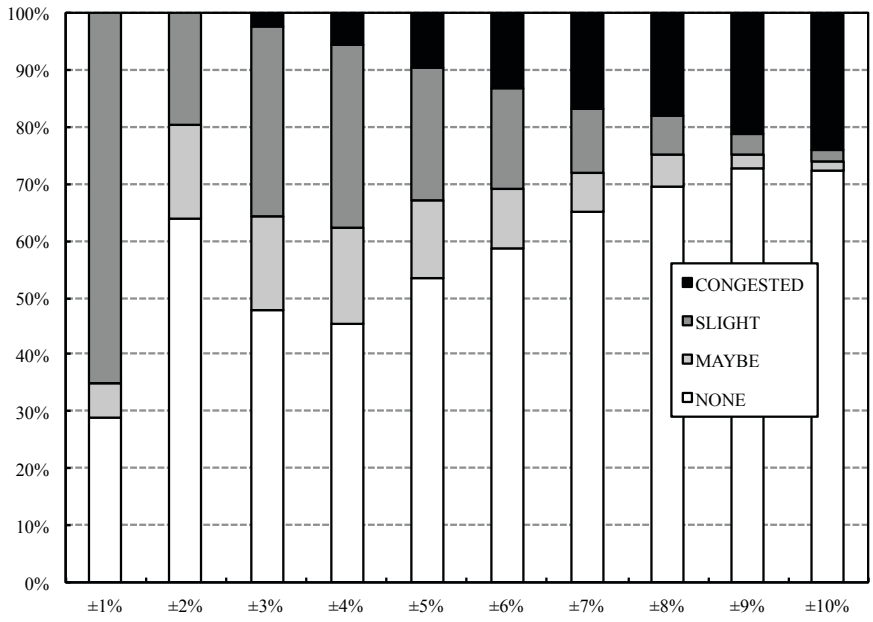


Fig. A. 2 breakdown of the congestion status.

A.4

付録 B RNS ソースコード

○ RNS 本体(rns.rb)

```
=begin
  Ruby Network Simulator ver.1.0.0
  coding start @ 2009/11/13
  by Akiji Tanaka (akiji@gifu-u.ac.jp)
=end

require 'singleton'
require 'ipaddr'

################################################################################
# Ruby Network Simulator Package
################################################################################
module RNS

  class Error < Exception; end
  class IllegalDataType < Error; end
  class IllegalTarget < Error; end
  class ObjectAlreadyExist < Error; end

  def RNS.type_check(_data, _class)
    raise IllegalDataType unless _data.is_a?(_class)
    _data
  end

  ##############################################################################
  # MODULE base simulation unit
  ##############################################################################
  module SimUnit

    UNIT_ID = []

    ############################################################################
    # initialize
    ############################################################################
    def initialize(_parent, _name, _type)
      @name = _name
      @type = _type
      @uid = "#{_type}:#{_name}"
      raise IllegalTarget, "UID: #{@uid} is already exist." if UNIT_ID.include?(@uid)
      UNIT_ID << @uid
      @child = {}
      @parent = _parent.nil? ? nil : RNS.type_check(_parent, SimUnit)
      @next_my_count = nil
      @next_action_count = nil
      @event = {}
      @active = false
    end

    attr_reader :name, :type, :uid, :parent

    ############################################################################
    # add child unit to unit
    ############################################################################
    def add_child(_child, _child_class)
      @child[_child.uid] = RNS.type_check(_child, _child_class)
      return _child
    end

    ############################################################################
```

B.2

```
# proc child unit with specific class
#############################################################################
def each_child(_class = SimUnit)
  @child.keys.sort.each do |key|
    yield(@child[key]) if @child[key].is_a?(_class)
  end
end

#############################################################################
# count up triger method
#############################################################################
def process()
  count = Sim.now
  call_event(count) if @event.has_key?(count)
  if (@next_action_count.nil? || @next_action_count <= count) then
    action() if !@next_my_count.nil? && @next_my_count <= count
    @child.keys.sort.each do |uid|
      @child[uid].process()
    end
  end
end

#############################################################################
# count up action method
#############################################################################
def action()
  @next_my_count = nil
end

#############################################################################
# check next action count
#############################################################################
def next_count()
  counts = [@next_my_count, @event.keys.min]
  each_child do |child|
    counts << child.next_count()
  end
  counts.delete(nil)
  @next_action_count = counts.min
end

#############################################################################
# add child unit to unit
#############################################################################
def set_parent(_parent)
  raise IllegalTarget, "UID: #{@uid} is already a child of #{@parent.uid}" unless @parent.nil?
  @parent = _parent
end

#############################################################################
# convert to string
#############################################################################
def to_s
  to_str()
end

#############################################################################
# recursive call to all children
#############################################################################
def to_str(_indent = 0)
  str = " " * _indent * 2 + to_line() + "¥n"
  @child.keys.sort.each do |key|
    str += @child[key].to_str(_indent+1)
  end
  return str
```

```
  end

  ##########################################################################
  # add log entry
  ##########################################################################
  def logging(_message, _level = Sim::LOG_DEBUG)
    Sim.log(self, _message) if Sim::Mgr.log_level >= _level
  end

  ##########################################################################
  # add event at specific time count
  # * event method name = 'call_' + event name
  #   ex: 'test' event's method name is 'call_test'
  # * _opt is array of remain parameter
  ##########################################################################
  def add_event(_time, _event_name, *_opt)
    count = (_time * Sim::Mgr.granularity).to_i
    @event[count] = [_event_name.to_s, _opt]
  end

  ##########################################################################
  # call event with specific time count
  ##########################################################################
  def call_event(_count)
    __send__('call_' + @event[_count][0].to_s, @event[_count][1])
    @event.delete(_count)
  end

  ##########################################################################
  # convert to one line style
  ##########################################################################
  def to_line()
    @uid
  end

  ##########################################################################
  # convert to one line style
  ##########################################################################
  def any_active?()
    active = is_active?
    each_child do |child|
      active |= child.any_active?()
    end
    return active
  end

  def is_active?()
    false
  end

end

##############################################################################
# CLASS Simulation Manager
##############################################################################
class Sim

  include SimUnit
  include Singleton

  Sim::LOG_ALWAYS = 0
  Sim::LOG_INFO = 1
  Sim::LOG_DEBUG = 2

  ##########################################################################
```

```
# get simulation count just now !
################################################################################
def Sim.now()
  Sim::Mgr.count
end

################################################################################
# output log entry
################################################################################
def Sim.log(_unit, _msg)
  time = Sim::Mgr.count.to_f / Sim::Mgr.granularity
  tform = "%#.#{Math.log10(Sim::Mgr.granularity).to_i}f"
  line = [sprintf(tform, time), _unit.uid, _msg].join(" ")#.gsub(/ /, "¥t")
  Sim::Mgr.log.puts line
end

################################################################################
# initialize
################################################################################
def initialize(_granularity = 1000000000)
  super(nil, 'Mgr', 'Sim')
  @granularity = _granularity.to_i
  @log = STDOUT
  @next_my_count = nil
  @next_action_count = nil
  @count = 0
  @log_level = Sim::LOG_ALWAYS
  @start_time = nil
end

attr_reader :granularity, :log, :count, :log_level

def set_log_level(_level)
  @log_level = _level.to_i
end

################################################################################
# add new node to simulator
################################################################################
def add_node(_name, _router = true)
  add_child(Node.new(self, _name, _router), Node)
end

def each_node()
  each_child(Node) do |node|
    yield node
  end
end

################################################################################
# add new network to simulator
################################################################################
def add_network(_lid = nil, _bandwidth = 100)
  add_child(Network.new(self, _lid, _bandwidth), Network)
end

def each_network()
  each_child(Network) do |network|
    yield network
  end
end

def set_log_file(_filename)
  ps = 0
  begin
```

```
      raise if File.exist?(_filename)
      @log = File.open(_filename, 'w')
    rescue
      ps += 1
      _filename += ".#{ps}"
    end
  end

  def start()
    @start_time = Time.now
    logging('start Simulation', Sim::LOG_ALWAYS)
    logging('set RoutingTable to Simulation', Sim::LOG_ALWAYS)
    RNS::L3::RoutingTable.set_routing_table(self)
    each_node do |node|
      print node.uid
      puts node.l3[RNS::L3::IPv6].routing_table
    end if DEBUG

    @count = next_count()
    begin
      process()
      count = next_count()
      @count = count
    end while any_active?()
    logging("finish Simulation with #{Time.now - @start_time}sec", Sim::LOG_ALWAYS)
  end

  Mgr = Sim.instance

end


################################################################################
# MODULE Capsuled Data Format
################################################################################
module BitData

  def bit_size()
    0
  end

end

################################################################################
# CLASS Basic Capsuled Data Object
################################################################################
class CapsuledData
  include BitData

  def initialize(_data, _bit_size)
    @data = _data
    @bit_size = _bit_size
  end

  attr_reader :data, :bit_size


end

################################################################################
# MODULE Capsuled Data
################################################################################
module DataCapsule

  include BitData
```

```
  def initialize()
    @data = {}
  end

  def add_data(_name, _data, _class = BitData)
    RNS.type_check(_data, _class)
    @data[_name.to_s] = _data
  end

  def [](_key)
    @data[_key]
  end

  def []=(_key, _value)
    @data[_key] = RNS.type_check(_value, BitData)
  end

  def has_key?(_key)
    @data.has_key?(_key)
  end

  def bit_size()
    s = 0
    @data.each_value do |data|
      s += data.bit_size()
    end
    return s
  end

end

################################################################################
# CLASS Node
################################################################################
class Node

  include SimUnit

  ##############################################################################
  # initialize
  ##############################################################################
  def initialize(_parent, _name, _router = false)
    super(_parent, _name, 'ND')
    @inf = {}
    @l2 = {}
    @l3 = {}
    @l4 = {}
    @app = {}
    @router = _router ? true : false
    add_l2(L2::Ethernet)
    add_l3(L3::IPv6)
    add_l4(L4::UDP)
  end

  attr_reader :inf, :l2, :l3, :l4, :app

  ##############################################################################
  # add interface to node
  ##############################################################################
  def add_inf()
    inf_num = @inf.size
    inf = Interface.new(self, inf_num)
    add_child(inf, Interface)
    @inf[inf_num] = inf
```

```
  end

  def each_interface()
    each_child(Interface) do |inf|
      yield inf
    end
  end

  def add_l2(_l2_class)
    l2 =  add_child(_l2_class.new(self), L2)
    @l2[_l2_class] = l2
    return l2
  end

  def add_l3(_l3_class)
    l3 =  add_child(_l3_class.new(self), L3)
    @l3[_l3_class] = l3
    return l3
  end

  def add_l4(_l4_class)
    l4 =  add_child(_l4_class.new(self), L4)
    @l4[_l4_class] = l4
    return l4
  end

  def add_app(_app_class, _opt = {})
    app = add_child(_app_class.new(self, _opt), Application)
    @app[_app_class] = app
    return app
  end

  ##########################################################################
  # default interface address
  ##########################################################################
  def ip()
    ipaddr = nil
    @inf.keys.sort.each do |inf_name|
      inf = @inf[inf_name]
      if inf.connected? then
        ipaddr = inf.ipaddr
        break
      end
    end
    return ipaddr
  end

  def is_my_ip?(_addr)
    RNS.type_check(_addr, IPAddr)
    each_interface do |inf|
      return true if inf.ipaddr == _addr
    end
    return false
  end

  def router?()
    @router
  end

end

##########################################################################
# CLASS Network Interface
##########################################################################
class Interface
```

B.8

```
include SimUnit

class InterfaceAlreadyConnected < Error; end

#############################################################################
# initialize
#############################################################################
def initialize(_parent, _inf_num)
  super(_parent, "#{_parent.name}:eth#{sprintf('%02d',_inf_num)}", 'IF')
  @inf_num = _inf_num.to_i
  @mac = MacAddress.new()
  @ipaddr = nil
  @network = nil
  @queue = []
  @max_threshold = 100
  @min_threshold = 75
  @drop_probability = 0.5
  @queue_size_hist = []
  @queue_size_hist_size = 20
  @l2_type = L2::Ethernet
end

attr_reader :inf_num, :ipaddr, :mac, :network

def action()
  if @queue.empty? then
    @next_my_count = nil
  else
    dst_mac = @queue[0]['header'].dst_mac
    next_count = @network.usage(dst_mac)
    logging("check #{@network.uid} usage #{next_count}")
    if next_count.nil? then
      logging("send #{@queue[0].pid} to #{@network.uid}")
      next_count = @network.put(@queue.shift, dst_mac)
      @next_my_count = next_count if @next_my_count < next_count
    end
  end
end

#############################################################################
# connecting network ?
#############################################################################
def connected?()
  ! @network.nil?
end

#############################################################################
# connect netowrk to interface
#############################################################################
def connect_network(_network)
  raise InterfaceAlreadyConnected if connected?()
  @network = _network
  tmp = @mac.address.split(/:/).insert(3, 'ff', 'fe')
  tmp[0] = (tmp[0].hex ^ 2).to_s(16)
  interface_id = IPAddr.new("::#{tmp[0]}#{tmp[1]}:#{tmp[2]}#{tmp[3]}:#{tmp[4]}#{tmp[5]}:#{tmp[6]}#{tmp[7]}/128")
  @ipaddr = interface_id | @network.netmask
  @network.add_inf(self)
end

#############################################################################
# disconnect network from interface
#############################################################################
def disconnect_network()
  if connected?() then
```

```
      @network.remove_inf(self)
      @network = nil
      @ipaddr = nil
    end
  end

  def add_output_queue(_packet)
    RNS.type_check(_packet, L2::DataSet)
    _drop = false
    _avg_queue_size = avg_queue_size()
    if node.router? && _avg_queue_size && _avg_queue_size > @min_threshold then
      _drop_threshold = (_avg_queue_size > @max_threshold ? 1.0 : (_avg_queue_size - @min_threshold) * @drop_probability /
(@max_threshold - @min_threshold))
      if rand() < _drop_threshold then
        _drop = true
        logging("drop packet #{_packet.pid}", Sim::LOG_ALWAYS)
      end
    end
    @queue << _packet unless _drop
    logging("queue size #{@queue.size}", Sim::LOG_DEBUG)
    @queue_size_hist << @queue.size
    while @queue_size_hist.size > @queue_size_hist_size do
      @queue_size_hist.shift
    end
    if @next_my_count.nil? then
      @next_my_count = Sim::Mgr.count + 1
    end
  end

  def avg_queue_size()
    return nil if @queue_size_hist.size < @queue_size_hist_size
    _total = 0
    _count = 0
    @queue_size_hist.each_index do |_index|
      _count += 1
      _total += @queue_size_hist[_index]
    end
    return (_total.to_f / _count).ceil
  end

  def input(_packet)
    RNS.type_check(_packet, L2::DataSet)
    node.l2[_packet.class::TYPE].input(_packet)
  end

  ##########################################################################
  # parent node
  ##########################################################################
  def node()
    @parent
  end

  ##########################################################################
  # convert to one line style
  ##########################################################################
  def to_line()
    [super().chomp, @mac, @ipaddr].join('  ')
  end

  ##########################################################################
  # CLASS Mac Address(Phisical Address)
  ##########################################################################
  class MacAddress

    include BitData
```

```
      MAC_TABLE = []

      ##########################################################################
      # initialize
      ##########################################################################
      def initialize(_vender_code = '00:00:00')
        @vender_code = _vender_code.downcase
        raise IllegalDataType, "Illegal vender code #{@vender_code}" unless @vender_code =~
/^[0-9a-f][0-9a-f]:[0-9a-f][0-9a-f]:[0-9a-f][0-9a-f]$/
        begin
          @interface_id = ''
          @interface_id << rand(16).to_s(16)
          @interface_id << rand(16).to_s(16)
          @interface_id << ':'
          @interface_id << rand(16).to_s(16)
          @interface_id << rand(16).to_s(16)
          @interface_id << ':'
          @interface_id << rand(16).to_s(16)
          @interface_id << rand(16).to_s(16)
          raise ObjectAlreadyExist if MAC_TABLE.include?(address())
        rescue ObjectAlreadyExist
          retry
        end
        MAC_TABLE << address()
      end

      ##########################################################################
      # mac address by : separated string
      ##########################################################################
      def address()
        "#{@vender_code}:#{@interface_id}"
      end

      ##########################################################################
      # mac address by 6 octets
      ##########################################################################
      def oct()
        address.split(/:/).map! { |hex| hex.hex}
      end

      ##########################################################################
      # size by bit
      ##########################################################################
      def bit_size()
        6 * 8
      end

      ##########################################################################
      # convert to string
      ##########################################################################
      def to_s
        address()
      end

    end

  end

  ##############################################################################
  # CLASS Network
  ##############################################################################
  class Network

    include SimUnit
```

```
NETWORK_ID = []

################################################################################
# initialize
################################################################################
def initialize(_parent, _name, _bandwidth = 100)
  @bandwidth = _bandwidth.to_i
  begin
    @nid = rand(2**16).to_s(16)
    raise ObjectAlreadyExist.new("Network #{@nid} is already exists.") if NETWORK_ID.include?(@nid)
  rescue ObjectAlreadyExist
    retry
  end
  NETWORK_ID << @nid
  @netmask = IPAddr.new("2000:0:0:#{@nid}::/64")
  @connected_inf = {}
  @arp_table = {}
  @data_line = {}
  super(_parent, _name, 'NW')
end

attr_reader :lid, :netmask, :bandwidth, :arp_table, :connected_inf

################################################################################
# main action
################################################################################
def action()
  @data_line.keys.each do |dst_mac|
    next if @data_line[dst_mac][0] > Sim.now
    packet = @data_line[dst_mac][1]
    logging("send #{packet.pid} to #{@connected_inf[dst_mac].uid}")
    @connected_inf[dst_mac].input(packet)
    @data_line.delete(dst_mac)
  end
  count = min_count()
  @next_my_count = count
end

################################################################################
# bandwidth chane event
################################################################################
def call_chbw(_opt)
  bandwidth = _opt.shift
  set_bandwidth(bandwidth)
  logging("change bandwidth to #{bandwidth}", Sim::LOG_ALWAYS)
end

################################################################################
# bandwidth change method
################################################################################
def set_bandwidth(_bandwidth)
  raise IllegalDataType if _bandwidth.to_i == 0
  @bandwidth = _bandwidth.to_i
end

################################################################################
# get minimum count of all data lines
################################################################################
def min_count()
  count = nil
  @data_line.keys.each do |_dst_mac|
    count = @data_line[_dst_mac][0] if count.nil? || count > @data_line[_dst_mac][0]
  end
  return count
```

```
  end

  ################################################################################
  # what time can data line be used
  ################################################################################
  def usage(_dst_mac)
    @data_line[_dst_mac].nil? ? nil : @data_line[_dst_mac][0]
  end

  ################################################################################
  # add interface to network table
  ################################################################################
  def add_inf(_nif)
      @connected_inf[_nif.mac] = _nif
      @arp_table[_nif.ipaddr] = _nif.mac
  end

  ################################################################################
  # remove interface from network table
  ################################################################################
  def remove_inf(_nif)
      @connected_inf.delete(_nif.mac)
      @arp_table.delete(_nif.ipaddr)
  end

  ################################################################################
  # put packet on network
  ################################################################################
  def put(_packet, _dst_mac)
    raise '' unless @data_line[_dst_mac].nil?
    send_count = Sim.now + calc_send_count(_packet.bit_size)
    @data_line[_dst_mac] = [send_count, _packet]
    @next_my_count = min_count()
    ret = Sim.now + calc_send_count(_packet.bit_size + 96)
    return ret
  end

  ################################################################################
  # time count to send data with specific size
  ################################################################################
  def calc_send_count(_bit)
    send_count = _bit.to_f * Sim::Mgr.granularity / (@bandwidth * 1024 * 1024)
    send_count.to_i
  end

  ################################################################################
  # convert to one line style
  ################################################################################
  def to_line()
    line = super().chomp + '  ' + @netmask.to_s + '  '
    @connected_inf.keys.sort{|a,b| a.address <=> b.address}.each do |mac|
      line << '  ' + @connected_inf[mac].node.uid
    end
    return line
  end

end

################################################################################
# MODULE TCP/IP Link Layer 2
################################################################################
module L2

  include SimUnit
```

```
CODE = {L2 => 'L2M'}

def initialize(_parent)
  super(_parent, "#{_parent.name}:#{L2::CODE[self.class]}", 'L2')
end

##############################################################################
# parent of L2 module is Node
##############################################################################
def node()
  @parent
end

##############################################################################
# MODULE Data format for Link Layer
##############################################################################
module DataSet
  include DataCapsule

  def pid()
    @data['data'].pid
  end

  def l3packet()
    @data['data']
  end

  def l4packet()
    l3packet['data']
  end

  def app_data()
    l4packet['data']
  end

end

##############################################################################
# CLASS Ethernet Protocol Stack
##############################################################################
class Ethernet

  include L2

  L2::CODE[Ethernet] = 'ETH'

  ############################################################################
  # make new Ethernet packet(frame)
  ############################################################################
  def new_packet(_inf, _addr, _data)
    src_mac = _inf.mac
    dst_mac = _inf.network.arp_table[_addr]
    Packet.new(src_mac, dst_mac, _data)
  end

  ############################################################################
  # input packet and push it up tu L3
  ############################################################################
  def input(_packet)
    l3p = _packet.l3packet
    node.l3[l3p.class::TYPE].input(_packet)
  end

  ############################################################################
  # CLASS Ethernet Data Packet(frame)
```

```
    ##########################################################################
    class Packet

      include DataSet

      TYPE = Ethernet

      ##########################################################################
      # object initialize
      ##########################################################################
      def initialize(_src_mac, _dst_mac, _data)
        super()
        add_data('header', Header.new(_src_mac, _dst_mac))
        add_data('data', _data)
        add_data('footer', CapsuledData.new(nil, 32))
      end

      attr_reader :l3_packet

      ##########################################################################
      # CLASS Ethernet Frame Header
      ##########################################################################
      class Header

        include BitData

        ##########################################################################
        # object initialize
        ##########################################################################
        def initialize(_src_mac, _dst_mac)
          @src_mac = RNS.type_check(_src_mac, Interface::MacAddress)
          @dst_mac = RNS.type_check(_dst_mac, Interface::MacAddress)
        end

        attr_reader :src_mac, :dst_mac

        def bit_size()
          176 # 22oct = 8oct(preamble) + 6oct(source) + 6oct(destination) + 2oct(type/size)
        end

      end

    end

  end

end

##############################################################################
# MODULE TCP/IP Internet Layer (Network Layer)
##############################################################################
module L3

  include SimUnit

  CODE = {L3 => 'L3M'}

  def initialize(_parent)
    super(_parent, "#{_parent.name}:#{L3::CODE[self.class]}", 'L3')
  end

  def node()
    @parent
  end
```

```
#############################################################################
# MODULE Data format for Internet Layer
#############################################################################
module DataSet
  include DataCapsule

  def pid()
    @data['data'].pid
  end

end

class RoutingTable

  def RoutingTable.set_routing_table(_sim)
    _sim.each_node do |node|
      node.each_interface do |inf|
        node.l3[L3::IPv6].routing_table.set_routing_info(inf.network, RoutingInfo.new(inf, 0, nil))
        RoutingTable.follow_routing_table_network(0, inf, node, inf)
      end
    end
  end

  def RoutingTable.follow_routing_table_node(_hop, _inf, _start_node, _start_inf, _start_addr = nil)
    node = _inf.parent
    _hop += 1
    unless node == _start_node then
      node.each_interface do |inf|
        next if inf == _inf
        network = inf.network
        routing_table = _start_node.l3[L3::IPv6].routing_table
        if !routing_table.routing_info.has_key?(network) || routing_table.routing_info[network].hop > _hop then
          routing_table.set_routing_info(network, RoutingInfo.new(_start_inf, _hop, _start_addr))
          RoutingTable.follow_routing_table_network(_hop, inf, _start_node, _start_inf, _start_addr)
        end
      end
    end
  end

  def RoutingTable.follow_routing_table_network(_hop, _inf, _start_node, _start_inf, _start_addr = nil)
    network = _inf.network
    network.connected_inf.values.each do |inf|
      next if inf == _inf
      addr = _start_addr.nil? ? inf.ipaddr : _start_addr
      RoutingTable.follow_routing_table_node(_hop, inf, _start_node, _start_inf, addr)
    end
  end

  def initialize()
    @routing_info = Hash.new()
  end

  attr_reader :routing_info

  def set_routing_info(_network, _ri)
    network = RNS.type_check(_network, Network)
    @routing_info[network] = _ri
  end

  def search_ri(_addr)
    @routing_info.keys.sort{|a,b| @routing_info[a].hop <=> @routing_info[b].hop}.each do |network|
      return @routing_info[network] if network.netmask.include?(_addr)
    end
    return nil
  end
```

```
  def search_inf(_addr)
    search_ri(_addr).inf
  end

  def to_s()
    str = "[Routing Table]¥n"
    @routing_info.keys.sort{|a,b| @routing_info[a].hop <=> @routing_info[b].hop}.each do |network|
      str << "#{network.name}¥t#{@routing_info[network]}¥n"
    end
    return str
  end

end

class RoutingInfo

  def initialize(_inf, _hop, _addr = nil)
    @inf = RNS.type_check(_inf, Interface)
    @hop = _hop.to_i
    @addr = _addr
  end

  attr_reader :hop, :inf, :addr

  def to_s()
    "#{@hop}¥t#{@inf.uid}¥t#{@addr}"
  end

end

################################################################################
# CLASS IPv6 Protocol Stack
################################################################################
class IPv6

  include L3

  L3::CODE[IPv6] = 'V6'

  def initialize(_parent)
    super(_parent)
    @routing_table = RoutingTable.new()
  end

  def new_packet(_src_addr, _dst_addr, _data)
    if _src_addr.nil? then
      _src_addr = routing_table.search_inf(_dst_addr).ipaddr
    end
    Packet.new(_src_addr, _dst_addr, _data)
  end

  def input(_packet)
    l3p = _packet.l3packet
    begin
      dst_addr = l3p['header']['basic'].dst_addr
      if node.is_my_ip?(dst_addr) then
        l3p['header']['routing'].class.proc_routing_header(l3p) if  l3p['header'].has_key?('routing')
        logging("receive #{_packet.pid} from #{l3p['header']['basic'].src_addr}")
        node.l4[_packet.l4packet.class::TYPE].input(_packet)
      else
        ri = @routing_table.search_ri(dst_addr)
        next_addr = ri.addr.nil? ? dst_addr : ri.addr
        ri.inf.add_output_queue(node.l2[L2::Ethernet].new_packet(ri.inf, next_addr, l3p))
        logging("forward #{_packet.pid} to #{ri.inf.uid}")
```

```
    end
  rescue Packet::Header::Routing::RelayNodeForwarding
    logging("forward #{_packet.pid} by RoutingHeader")
    retry
  end
end


attr_reader :routing_table

#############################################################################
# CLASS IPv6 Packet
#############################################################################
class Packet

  include DataSet

  TYPE = IPv6

  def initialize(_src_addr, _dst_addr, _data)
    super()
    add_data('header', Header.new(_src_addr, _dst_addr))
    add_data('data', _data)
  end

  def set_src_addr(_addr)
    @data['header']['basic'].set_src_addr(_addr)
  end

  #############################################################################
  # CLASS IPv6 Headers
  #############################################################################
  class Header

    include DataCapsule

    def initialize(_src_addr, _dst_addr)
      super()
      add_data('basic', Basic.new(_src_addr, _dst_addr))
    end

    #############################################################################
    # CLASS IPv6 Basic Header
    #############################################################################
    class Basic

      include BitData

      def initialize(_src_addr, _dst_addr)
        @version = 6
        @traffic_class = nil
        @flow_label = nil
        @payload_length = 0
        @next_header = nil
        @hop_limit = 255
        @src_addr = RNS.type_check(_src_addr, IPAddr)
        @dst_addr = RNS.type_check(_dst_addr, IPAddr)
      end

      attr_reader :src_addr, :dst_addr, :hop_limit

      def set_dst_addr(_dst_addr)
        @dst_addr = RNS.type_check(_dst_addr, IPAddr)
      end
```

```
  def set_src_addr(_src_addr)
    @src_addr = RNS.type_check(_src_addr, IPAddr)
  end

  def bit_size()
    40 * 8
  end

end

#####################################################################
# CLASS IPv6 Routing Header
#####################################################################
module Routing

  class RelayNodeForwarding < Error; end

  include DataCapsule

  def initialize()
    super()
    add_data('common', CapsuledData.new(nil, 64))
    @next_header = nil
    @header_length = 0
    @segment_left = 0
    @routing_type = nil
  end

  attr_reader :segment_left

  def set_segment_left(_segment_left)
    @segment_left = _segment_left.to_i
  end

  #####################################################################
  # CLASS IPv6 Routing Header Type0 Custom
  # Type0 routing header with following limitations
  #   * Loose souce routing only
  #   * One relay node only
  #   * Segment Left has only two value,
  #     0(already relayed) or 1 (not relayed yet)
  #####################################################################
  class Type0C

    include Routing

    def Type0C.set_routing_header(_v6header, _relay_addr)
      RNS.type_check(_v6header, Header)
      RNS.type_check(_relay_addr, IPAddr)
      rthdr = Type0C.new()
      raise IllegalTarget, "IPv6 header already has routing header" unless _v6header['routing'].nil?
      _v6header['routing'] = rthdr
      rthdr['type_specific'] = _v6header['basic'].dst_addr
      _v6header['basic'].set_dst_addr(_relay_addr)
    end

    def Type0C.proc_routing_header(_packet)
      RNS.type_check(_packet, L3::IPv6::Packet)
      rthdr = RNS.type_check(_packet['header']['routing'], self)
      if rthdr.segment_left > 0 then
        relay_addr = _packet['header']['basic'].dst_addr
        _packet['header']['basic'].set_dst_addr(rthdr['type_specific'])
        rthdr['type_specific'] = relay_addr
        rthdr.set_segment_left(0)
        raise RelayNodeForwarding
```

```
            end
          end

          def initialize()
            super()
            @routing_type = '0C'
            @header_length = 0
            @segment_left = 1
            add_data('type_specific', '')
          end

        end

      end

    end

  end

end

################################################################################
# MODULE TCP/IP Transport Layer
################################################################################
module L4

  include SimUnit

  CODE = {L4 => 'L4M'}

  def initialize(_parent)
    super(_parent, "#{_parent.name}:#{L4::CODE[self.class]}", 'L4')
  end

  ################################################################################
  # MODULE Data format for Transport Layer
  ################################################################################
  module DataSet
    include DataCapsule

    def pid()
      @data['data'].pid
    end

  end

  ################################################################################
  # MODULE UDP
  ################################################################################
  class UDP

    include L4

    L4::CODE[UDP] = 'UDP'

    def initialize(_parent)
      super(_parent)
      @socket = {}
    end

    def new_packet(_src_port, _dst_port, _data)
      Packet.new(_src_port, _dst_port, _data)
    end
```

```
def open_socket(_port, _socket)
  raise "Port #{_port} already binded to Socket" if @socket.has_key?(_port)
  raise "Socket #{_socket.name} already binded to Port" if @socket.has_value?(_socket)
  @socket[_port.to_i] = _socket
end

def get_empty_port()
  port = nil
  begin
    port = rand(7001) + 10000
    raise if @socket.has_key?(port)
  rescue
    retry
  end
  return port
end

def input(_packet)
  l4p = _packet.l4packet
  dst_port = l4p['header'].dst_port
  socket = @socket[dst_port.to_i]
  socket.input(_packet)
end

###############################################################################
# CLASS UDP Packet
###############################################################################
class Packet

  include DataSet

  TYPE = UDP

  def initialize(_src_port, _dest_port, _data)
    super()
    add_data('header', Header.new(_src_port, _dest_port))
    add_data('data', _data)
    header.set_length(data.bit_size())
  end

  def header()
    @data['header']
  end

  def data()
    @data['data']
  end

  ###############################################################################
  # CLASS UDP Header
  ###############################################################################
  class Header
    include BitData

    def initialize(_src_port, _dst_port)
      @src_port = _src_port
      @dst_port = _dst_port
      @length = 0
    end

    attr_reader :src_port, :dst_port, :length

    def set_length(_length)
      @length = _length.to_i
```

```
      end

      def bit_size()
        64
      end

    end

  end

end

################################################################################
# MODULE Network Socket
################################################################################
module Socket

  include SimUnit

  def initialize(_parent, _name)
     @l2 = nil
     @l3 = nil
     @l4 = nil
     super(_parent, _name, "SC")
  end

  class UDPv6Socket

    include Socket

    def initialize(_parent, _name, _src_port = nil, _src_addr = nil)
      super(_parent, _name)
      @l2 = L2::Ethernet
      @l3 = L3::IPv6
      @l4 = L4::UDP
      @src_addr = _src_addr.nil? ? nil : RNS.type_check(_dst_addr, IPAddr)
      @src_port = set_port(_src_port)
    end

    def app()
      @parent
    end

    def node()
      @parent.parent
    end

    def open()
      node.l4[@l4].open_socket(@src_port, self)
    end

    def set_relay_addr(_addr)
      @relay_addr = RNS.type_check(_addr, IPAddr)
    end

    def bind_addr(_addr)
      @src_addr = RNS.type_check(_addr, IPAddr)
    end

    def set_port(_port = nil)
      @src_port = _port.nil? ? node.l4[@l4].get_empty_port() : _port.to_i
    end
```

```
    def set_relay_addr(_addr)
      @relay_addr = RNS.type_check(_addr, IPAddr)
    end

    def make_new_packet(_data, _dst_port, _dst_addr, _relay_addr = nil)
      l4p = node.l4[@l4].new_packet(@src_port, _dst_port, _data)
      l3p = node.l3[@l3].new_packet(@src_addr, _dst_addr, l4p)
      L3::IPv6::Packet::Header::Routing::Type0C.set_routing_header(l3p['header'], _relay_addr) unless _relay_addr.nil?
      ri = node.l3[@l3].routing_table.search_ri(_dst_addr)
      next_addr = ri.addr.nil? ? _dst_addr : ri.addr
      l2p = node.l2[@l2].new_packet(ri.inf, next_addr, l3p)
      return l2p
    end

    def send_reply(_data, _packet)
      l3p = _packet.l3packet
      dst_addr = l3p['header']['basic'].src_addr
      relay_addr = l3p['header'].has_key?('routing') ? l3p['header']['routing']['type_specific'] : nil
      dst_port = _packet.l4packet['header'].src_port
      packet = make_new_packet(_data, dst_port, dst_addr, relay_addr)
      send_packet(packet)
    end

    def send(_data, _dst_port, _dst_addr, _relay_addr = nil)
      packet = make_new_packet(_data, _dst_port, _dst_addr, _relay_addr)
      send_packet(packet)
    end

    def send_packet(_packet)
      RNS.type_check(_packet, L2::DataSet)
      dst_addr = _packet.l3packet['header']['basic'].dst_addr
      inf = node.l3[@l3].routing_table.search_ri(dst_addr).inf
      logging("send #{_packet.pid} to #{inf.uid}", Sim::LOG_INFO)
      inf.add_output_queue(_packet)
    end


    def input(_packet)
      app.input(_packet)
    end

    def to_line()
      "#{@uid}¥t#{@src_port}¥t#{@src_addr}"
    end

  end

end

################################################################################
# MODULE TCP/IP Application Layer
################################################################################
module Application

  include SimUnit

  CODE = {Application => 'APP'}

  def initialize(_parent)
    super(_parent, "#{_parent.name}:#{Application::CODE[self.class]}", "AP")
  end

  def input(_packet)
  end
```

```
  end

end

################################################################################
# Add size attribute to IPAddr object
################################################################################
class IPAddr

  include RNS::BitData

  ##############################################################################
  # size by bit
  ##############################################################################
  def bit_size()
    s = 0
    s = 32 if self.ipv4?
    s = 128 if self.ipv6?
    return s
  end

end

################################################################################
# Add size attribute to String object
################################################################################
class String

  include RNS::BitData

  ##############################################################################
  # size by bit
  ##############################################################################
  def bit_size()
    size() * 8
  end

end

################################################################################
# load Application files
################################################################################
Dir.glob(File.dirname(__FILE__) + '/Application/*.rb').each do |appfile|
  require appfile
end
```

○ SMPC アプリケーション(smpc.rb)

```
module RNS

  ##############################################################################
  # Namespace for Multi-Path communication
  # save last transmit success/fail flag
  # reimplement old SMPC-P
  ##############################################################################
  module SMPCP13

    ############################################################################
    # CLASS Multi-Path Sender
    ############################################################################
    class Sender

      include Application
```

```
    Application::CODE[Sender] = 'MPS'

    ###########################################################################
    # object initialzie
    ###########################################################################
    def initialize(_parent, _opt = {})
      super(_parent)
      @dst_addr = RNS.type_check(_opt['mp_dst_addr'], IPAddr)
      @dst_port = _opt.has_key?('mp_dst_port') ? _opt['mp_dst_port'].to_i : 8888
      @socket = add_child(Socket::UDPv6Socket.new(self, @name, 8888) ,Socket)
      @socket.open()
      @bw_unit = _opt.has_key?('mp_bw_unit') ? _opt['mp_bw_unit'].to_f : 1
      @platform = []
      @relay_addr = []
      add_route(nil)
      @packet_count = 0
      @data_size_unit = 1024
      @train_id = 0
      @train_count = _opt.has_key?('mp_train_count') ? _opt['mp_train_count'].to_i : 16
      @train_status = {}
      @sent_data_size = 0
      @max_size = 0
      @threshold_delay = 0.95
      @threshold_abnormal = 1.05
      @fix_desired_throughput = _opt.has_key?('mp_desired_throughput') ? _opt['mp_desired_throughput'].to_i : nil
      @desired_throughput = @fix_desired_throughput || (1.0 / 0.0)
      @bw_hist = [0]
      @avg_hist = [0]
      @avg_bw = 0
      @status_hist = [0]
      @continuous_success = 0
      @bw_hist_size = _opt.has_key?('mp_bw_hist_size') ? _opt['mp_bw_hist_size'].to_i : 20
      @stable_threshold = _opt.has_key?('mp_stable_threshold') ? _opt['mp_stable_threshold'].to_f : 3.0
      @force_down_rate = {}
      @force_down_rate[:success] = _opt.has_key?('mp_success_down_rate') ? _opt['mp_success_down_rate'].to_i : 0
      @force_down_rate[:delay] = _opt.has_key?('mp_delay_down_rate') ? _opt['mp_delay_down_rate'].to_i : 1
      @force_down_rate[:loss] = _opt.has_key?('mp_loss_down_rate') ? _opt['mp_loss_down_rate'].to_i : 1
      @force_down_rate[:abnormal] = _opt.has_key?('mp_abnormal_down_rate') ? _opt['mp_abnormal_down_rate'].to_i : 0
      @path_unfair = _opt.has_key?('mp_path_unfair')
    end

    attr_reader :data_size_unit
    attr_reader :dst_port
    attr_reader :dst_addr
    attr_reader :sent_data_size
    attr_reader :train_count
    attr_reader :bw_unit
    attr_reader :socket

    def conf_str()
      _str =
"_DR#{@force_down_rate[:success]}-#{@force_down_rate[:delay]}-#{@force_down_rate[:loss]}-#{@force_down_rate[:abnormal]}"
      _str << "_BU#{@bw_unit}_TH#{@stable_threshold}"
      _str << "_HS#{@bw_hist_size}_TC#{@train_count}"
      _str << "_DB#{@fix_desired_throughput || '-VAL'}"
      _str << "_UF" if @path_unfair
      return _str
    end

     def is_active?()
      ! @next_my_count.nil?
    end

    def force_down(_route, _speed)
```

```
    return unless _speed > 0
    path = nil
    _speed = _speed.to_i
    begin
      ind = @platform.size - 1
      while ind > _route do
        if @platform[ind].bandwidth > @platform[ind].bw_unit * _speed then
          path = @platform[ind]
          break
        end
        ind -= 1
      end
    end
    unless path.nil? then
      _down_speed = @platform[ind].bw_unit * _speed
      logging("force down bandwidth at Path:#{path.route} by #{_down_speed}Mbps", Sim::LOG_ALWAYS)
      path.shift_down(path.bandwidth - _down_speed)
    end
  end

  def avg_total_bw()
    _total_bw = 0
    @bw_hist.each do |_bw|
      _total_bw += _bw
    end
    return _total_bw / @bw_hist.size
  end

  def input(_packet)
    rcv = RNS.type_check(_packet.app_data, Reply).status
    sent = @train_status[rcv.train_id]
    logging("receive #{rcv.count}/#{sent.count} patkets with #{sprintf('%.2f', rcv.throughput)}/#{sprintf('%.2f',
sent.throughput)} Mbps at TRAIN:#{rcv.train_id}/Path#{rcv.route}", Sim::LOG_ALWAYS)

    # transmitting status check
    _status = nil
    if rcv.count == sent.count then
      rate = rcv.throughput / sent.throughput
      if (rate > @threshold_abnormal) then
        logging("found abnormal response at Path:#{rcv.route}", Sim::LOG_ALWAYS)
        _status = :abnormal
      elsif(rate > @threshold_delay) then
        logging("found no congestion at Path:#{rcv.route}", Sim::LOG_ALWAYS)
        _status = :success
      else
        logging("found congestion at Path:#{rcv.route}", Sim::LOG_ALWAYS)
        _status = :delay
      end
    else
      logging("found packet loss at Path:#{rcv.route}", Sim::LOG_ALWAYS)
      _status = :loss
    end

    # priority control
    _forced_down = false
    if _status == :delay || _status == :loss then
      # force down lower path
      unless @platform[rcv.route + 1].nil? || total_bandwidth() < @desired_throughput * 0.95 then
        force_down(rcv.route, @force_down_rate[_status])
        _forced_down = true
      end
    end

    # transmission control
    @platform[rcv.route].set_receive_status(_status, sent.throughput, rcv.throughput, _forced_down)
```

```
  # calc new desired throughput
  _total_bw = 0.0
  begin
    @platform.each do |_platform|
      _total_bw += _platform.get_last_bw
    end
  rescue
    _total_bw = nil
  end
  if _total_bw then
    logging("have total bandwidth #{_total_bw}", Sim::LOG_ALWAYS)
    _old_bw = @bw_hist[@bw_hist_size - 1].to_f
    @bw_hist = @bw_hist.unshift(_total_bw)[0,@bw_hist_size]
    @avg_bw += (_total_bw - _old_bw) / @bw_hist_size
    if @bw_hist.size >= @bw_hist_size then
      @avg_hist = @avg_hist.unshift(@avg_bw)
      logging("average #{@avg_bw}", Sim::LOG_ALWAYS)
      if @avg_hist.size > @bw_hist_size then
        _avg_diff = (@avg_hist.first - @avg_hist.last).abs
        logging("average-diff #{_avg_diff}", Sim::LOG_ALWAYS)
        if @fix_desired_throughput.nil? && _avg_diff < @stable_threshold then
          @desired_throughput = @avg_bw
          logging("change desired throughput to #{@desired_throughput}", Sim::LOG_ALWAYS)
        end
        @avg_hist = @avg_hist[0,@bw_hist_size]
      end
    end
  else
    logging("ignore consecutive ACK on Path:#{rcv.route}", Sim::LOG_ALWAYS)
  end

  # post process
  @train_status.delete(rcv.train_id)
  @next_my_count =  @train_status.empty? ? nil : Sim.now + 60 * Sim::Mgr.granularity
end

def total_bandwidth()
  bw = 0
  @platform.each do |platform|
    bw += platform.bandwidth
  end
  return bw
end

def action()
  @platform.each do |platform|
    platform.finish_send() if platform.is_active?()
  end
  logging("timed out.", Sim::LOG_ALWAYS)
  @next_my_count = nil
end

def send_data(_data)
  logging("send #{_data.pid} to Path#{_data.route}", Sim::LOG_INFO)
  @socket.send(_data, @dst_port, @dst_addr, @relay_addr[_data.route])
end

def set_desired_throughput(_throughput)
  @desired_throughput = _throughput.to_i
end

def rest_data_size()
  @max_size - sent_data_size
end
```

```
def next_data_size()
  sent_finish? ? 0 : (rest_data_size > @data_size_unit) ? @data_size_unit : rest_data_size
end

def make_new_data()
  size = next_data_size
  return nil unless size > 0
  @sent_data_size += size
  Data.new("PK:#{@name}:#{@packet_count += 1}", '0'*size)
end

##########################################################################
# event for testing : send 2 packet to specific route
##########################################################################
def call_send_data(_opt)
  raise unless @max_size == 0
  size = _opt.shift.to_f
  @max_size = (size * 1024 * 1024).to_i
  logging("start MultiPathSending for #{size}MB", Sim::LOG_ALWAYS)
  @platform.each do |platform|
    platform.start()
  end
end

def next_train_id()
  return nil if sent_finish?
  @train_id += 1
end

def set_train_status(_train_id, _route)
  @train_status[_train_id] = TrainData.new(_train_id, _route)
end

def sent_finish?()
  rest_data_size <= 0
end

##########################################################################
# add route with specific relay node
##########################################################################
def add_route(_relay_node)
  relay_addr = _relay_node.nil? ? nil : _relay_node.ip
  @relay_addr << relay_addr
  @platform << add_platform()
end

def add_platform()
  pl = add_child(Platform.new(self, @platform.size, @path_unfair), Platform)
  return pl
end

##########################################################################
# generic 1 line string style
##########################################################################
def to_line()
  @uid + "¥t" +  @relay_addr.join("¥t")
end

##########################################################################
# generic 1 line string style
##########################################################################
class Platform

  include SimUnit
```

```
MODE_NORMAL = 0

attr_reader :treated

def initialize(_parent, _route, _unfair = false)
  super(_parent, "#{_parent.name}:#{_route}", 'PF')
  @shift = 0
  @route = _route
  @current_pos = 0
  @queue = []
  @wait_train = nil
  @sending = false
  @wait = 0
  @wait_init = 1
  @train_status = nil
  @bw_hist = [0]
  @bw_hist_size = 20
  @status_hist = [:success]
  @local_bw_unit = @parent.bw_unit.to_f
  @local_bw_unit /= (2 ** @route.to_f) if _unfair
  @treated = false
  @continuous_success = 0
  @last_success_bw = 0
end

attr_reader :route

def bw_unit()
  @local_bw_unit
end

def set_received_bandwidth(_bw)
  @bw_hist = @bw_hist.unshift(_bw)[0,@bw_hist_size]
  @treated = false
end

def get_last_bw()
  _bw = nil
  unless @treated then
    _bw = @bw_hist.first
    @treated = true
  end
  return _bw
end

def set_receive_status(_status, _sent_bw, _rcv_bw, _forced_down)

  if _status == :success then
    @continuous_success += 1
    @last_success_bw = _rcv_bw
  else
    logging("continuous success #{@continuous_success} / last success bandwidth #{@last_success_bw}", Sim::LOG_ALWAYS)
    @continuous_success = 0
  end
  @status_hist = @status_hist.unshift(_status)[0,@bw_hist_size]
  if _success_rate = status_rate() then
    logging("transmit success rate #{_success_rate} in last #{@bw_hist_size} trains", Sim::LOG_ALWAYS)
  end

  set_received_bandwidth(_rcv_bw)

  case _status
  when :success
    success_throughput(_sent_bw)
```

```
  when :delay
    fail_throughput(_rcv_bw) unless _forced_down
  when :loss
    loss_detected(_rcv_bw)
  when :abnormal
    abnormal_throughput(_sent_bw)
  end
end

def status_rate(_target_status = :success)
  if @status_hist.size == @bw_hist_size then
    return @status_hist.count(_target_status).to_f / @bw_hist_size
  else
    return nil
  end
end

def set_to_treat()
  @treated = false
end

def treat()
  @treated = true
end

def avg_bandwidth()
  if @bw_hist.size == @bw_hist_size then
    _total_bw = 0.0
    @bw_hist.each do |_bw|
      _total_bw += _bw
    end
    return _total_bw.to_f / @bw_hist_size
  else
    return nil
  end
end

def max_received_bandwidth()
  @bw_hist.max
end

def min_received_bandwidth()
  @bw_hist.min
end

def bandwidth_trend()
  if @bw_hist.size == @bw_hist_size then
    return _trend = @bw_hist.first - avg_bandwidth()
  else
    return nil
  end
end

def bandwidth()
  @shift * @local_bw_unit
end

def interval()
  (app.data_size_unit.to_f * 8 * Sim::Mgr.granularity / (bandwidth() * 1024 * 1024)).to_i
end

def app()
  @parent
end
```

```
def set_shift(_shift)
  @shift = (_shift.to_i * @local_bw_unit < 8 ? (8.0 / @local_bw_unit).ceil : _shift.to_i)
end

def set_bandwidth(_bw)
  return unless @sending
  set_shift(_bw.to_f / @local_bw_unit)
  logging("change bandwidth to #{bandwidth()}Mbps with Shift #{@shift}", Sim::LOG_ALWAYS)
end

def shift_change(_throughput, _shift)
  set_shift((_throughput.to_f + @local_bw_unit * _shift) / @local_bw_unit)
  logging("change bandwidth to #{bandwidth()}Mbps with Shift #{@shift}", Sim::LOG_ALWAYS)
end

def shift_up(_bandwidth)
  return unless @sending
  if _bandwidth > bandwidth() then
    set_bandwidth(_bandwidth)
  else
    logging("keep bandwidth to #{bandwidth()}Mbps with Shift #{@shift}", Sim::LOG_ALWAYS)
  end
end

def shift_down(_bandwidth)
  return unless @sending
  if _bandwidth < bandwidth() then
    set_bandwidth(_bandwidth)
  else
    logging("keep bandwidth to #{bandwidth()}Mbps with Shift #{@shift}", Sim::LOG_ALWAYS)
  end
end

def half_down()
  set_shift((@shift.to_f / 2).ceil)
end

def success_throughput(_throughput)
  shift_up(_throughput + @local_bw_unit)
  set_to_treat()
end

def fail_throughput(_throughput)
  shift_down(_throughput)
  set_to_treat()
end

def abnormal_throughput(_throughput)
  shift_down(_throughput - @local_bw_unit)
  set_to_treat()
end

def loss_detected(_throughput)
  shift_down(_throughput - @local_bw_unit)
  set_to_treat()
end

def action()
  send_data()
end

##########################################################################
# send multipath data
##########################################################################
def send_data()
```

```
      data = @queue.shift
      @train_status.add_data(data)
      logging("send #{data.pid} to Path#{data.route}", Sim::LOG_INFO)
      app.send_data(data)
      @next_my_count = @queue.empty? ? make_train() : Sim.now + interval
    end

    def make_train()
      logging("sent TRAIN:#{@train_status.train_id} with #{@train_status.throughput}Mbps", Sim::LOG_ALWAYS) if @train_status
      train_id = app.next_train_id
      return finish_send() if train_id.nil?
      logging("make TRAIN:#{train_id} with #{bandwidth()}Mbps", Sim::LOG_ALWAYS)
      @train_status = app.set_train_status(train_id, @route)
      tmp = []
      pos = 0
      while pos < app.train_count do
        data = app.make_new_data()
        break if data.nil?
        data.set_route(@route)
        data.set_train_id(train_id)
        data.set_pos(pos)
        tmp << data
        pos += 1
      end
      tmp[0].set_train_first()
      tmp[-1].set_train_last()
      @queue += tmp
      return Sim.now + interval
    end

    def start()
      @sending = true
      shift_up(@local_bw_unit)
      @next_my_count = make_train()
    end

    def finish_send()
      @shift = 0
      @sending = false
      return nil
    end

    def is_active?()
      @sending
    end

  end

end

##################################################################################
# CLASS Multi-Path Receiver
##################################################################################
class Receiver
  include Application
  Application::CODE[Receiver] = 'MPR'

  ##################################################################################
  # object initialzie
  ##################################################################################
  def initialize(_parent, _opt = {})
    super(_parent)
    @relay_addr = [nil]
    @rec_port = _opt.has_key?('mp_rec_port') ? _opt['mp_rec_port'].to_i : 8888
    @socket = add_child(Socket::UDPv6Socket.new(self, @name, @rec_port) ,Socket)
```

```
      @socket.open()
      @received_size = [0]
      @train_data = {}
      @packet_count = 0
    end

    ##########################################################################
    # add route with specific relay node
    ##########################################################################
    def add_route(_relay_node)
      @relay_addr << RNS.type_check(_relay_node, Node).ip
      @received_size << 0
    end

    def input(_packet)
      data = RNS.type_check(_packet.app_data, Data)
      if (@train_data[data.route].nil?) then
        @train_data[data.route] = TrainData.new(data.train_id, data.route, _packet)
      elsif (data.train_id != @train_data[data.route].train_id) then
        make_reply(@train_data[data.route]) if @train_data[data.route]
        @train_data[data.route] = TrainData.new(data.train_id, data.route, _packet)
      end
      @train_data[data.route].add_data(data)
      @received_size[data.route] += data.data.size
      logging("receive #{_packet.pid} data #{data.params},rs=#{@received_size.join('/')}", Sim::LOG_INFO)
      if (data.is_train_last?) then
        make_reply(@train_data[data.route]) if @train_data[data.route]
        @train_data[data.route] = nil
      end
    end

    def make_reply(_train_data)
      logging("receive TRAIN:#{_train_data.train_id} by #{sprintf('%.2f', _train_data.throughput)}Mbps/Path#{_train_data.route}
#{_train_data.packets.to_s(16)} #{_train_data.first_include?() ? 'S' : '-'}/#{_train_data.last_include?() ? 'E' : '-'}",
Sim::LOG_ALWAYS)
      reply = Reply.new("PK:#{@name}:#{@packet_count += 1}", _train_data)
      @socket.send_reply(reply, _train_data.packet)
    end

    ##########################################################################
    # generic 1 line string style
    ##########################################################################
    def to_line()
      @uid + "¥t" +  @relay_addr.join("¥t")
    end

  end

  ##########################################################################
  # CLASS Train Status Data
  ##########################################################################
  class TrainData

    def initialize(_train_id, _route, _packet = nil)
      @packet = _packet
      @train_id = _train_id
      @route = _route
      @first_count = nil
      @last_count = nil
      @packets = 0
      @count = 0
      @train_size = 0
      @tp_size = nil
      @first_packet = false
      @last_packet = false
```

```
  end

  attr_reader :packet, :train_id, :route, :packets, :train_size, :count

  def add_data(_data)
    @first_count = Sim.now if @first_count.nil?
    @last_count = Sim.now
    @packets |= 2 ** _data.pos
    @count += 1
    @train_size += _data.data.bit_size
    if @tp_size then
      @tp_size += _data.data.bit_size
    else
      @tp_size = 0
    end
    @first_packet = true if _data.is_train_first?
    @last_packet = true if _data.is_train_last?
  end

  def first_include?()
    @first_packet
  end

  def last_include?()
    @last_packet
  end

  def time_count()
    @last_count - @first_count
  end

  def time()
    time_count.to_f / Sim::Mgr.granularity
  end

  def throughput()
    @tp_size.to_f / time / 1024 / 1024
  end

end

################################################################################
# CLASS Multi-Path Communication Data Format
################################################################################
class Data

  include BitData

  ##############################################################################
  # object initialzie
  ##############################################################################
  def initialize(_pid, _data, _route = nil, _train = nil, _pos = nil)
    @pid = _pid.to_s      # Packet ID (Virtual Data)
    @data = _data.to_s    # Send data (Valiable Size)
    @route = _route.to_i  # Route identify number (4bit)
    @type = 0             # Start/End flag (4bit)
    @train_id = _train.to_i  # Packet Train ID (24bit)
    @pos = _pos.to_i      # Position in Train (8bit)
  end

  attr_reader :pid, :data, :route, :train_id, :pos

  def set_train_first()
    @type |= 1
  end
```

```
      def set_train_last()
        @type |= 2
      end

      def is_train_first?()
        @type & 1 != 0
      end

      def is_train_last?()
        @type & 2 != 0
      end

      def set_route(_route)
        @route = _route.to_i
      end

      def set_train_id(_train_id)
        @train_id = _train_id.to_i
      end

      def set_pos(_pos)
        @pos = _pos.to_i
      end

      def params()
        "r=#{@route},t=#{@train_id},p=#{@pos},f=#{is_train_first? ? 'S' : '-'}#{is_train_last? ? 'E' : '-'},s=#{@data.size}"
      end

      ###########################################################################
      # bit size for BitData
      ###########################################################################
      def bit_size()
        @data.bit_size + 4 + 4 + 24 + 8
      end

    end

    class Reply
      include BitData

      attr_reader :pid, :status

      def initialize(_pid, _train_data)
        @status = RNS.type_check(_train_data, TrainData)
        @pid = _pid.to_s
      end

      def bit_size()
        32 + 4 + 28 + 32 + 32
      end
    end
  end
end
```

付録 C 実機実験用 SMPC 通信プログラム

```
#define DEBUG   // Debug mode(?)
#define END_DEBUG // End of debugmode(?)
#define LOGDISP   // Output detailed log(?)

/*----------------------includes--------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <sys/socket.h>
#include <sys/param.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <unistd.h>
#include <time.h>
#include <pthread.h>

/*----------------------defines--------------------------*/
// Path
#define MAIN 0
#define SUB 1
#define MAXROUTE 2

// Data size
#define BUFF 1024      // SMPC data payload [byte]
#define PACKETSIZE (40 + 8 + (4+2+2+BUFF))  // IPv6/UDP datagram size (w/o rthdr) [byte]
#define ETHERSIZE (18 + PACKETSIZE + 8)   // Ether/IPv6/UDP frame size (w/o rthdr) [byte]
#define SENDDATASIZE 104857600  // 100MB in byte

// Packet train
#define LISTMAX 200
#define TRAIN 16// Packet number of one train
#define SENDPACKETNUM ((SENDDATASIZE >> 10) / TRAIN + 1)  // train number for sending SENDDATASIZE

// Bandwidth
#define MAXWIDTH 96.0     // max bandwidth of one path [Mbps]
#define MINWIDTH 8.0      // minimum bandwidth of one path [Mbps]
#define WIDTH_SHIFT 1.0     // bandwidth unit for up or down sending rate [Mbps]
#define THRESHOLD_MIN 0.95  // sending ratio threshold for delay
#define THRESHOLD_MAX 1.05  // sending ratio threshold for abnormal

//
#define RECVTIMEOUT_SEC 5 // timeout for server [sec]

// Status
#define NORMAL 10
#define SDELAY 11
#define LOSS 12
#define ABNORMAL 13
#define INVALID 14
#define SET 1
#define RESET 0
```

C.2

```
#define min(a,b) ((a) < (b) ? (a):(b))
#define max(a,b) ((a) > (b) ? (a):(b))

/*--------------------structs-------------------------*/
// SMPC data packet
typedef struct  {
    int trid;        // Train ID
    unsigned short int route;   // Path
    unsigned short int id;    // Packet ID
    char msg[BUFF];      // Data payload
} Packetdata;

// TACK Packet
typedef struct  {
    int route;       // Path
    int trid;      // Train ID
    long td1;      // first packet receiving time
    long td2;      // last packet receibing time
    unsigned short int id;  // received packet flag
} Replypacket;

// tarin sending status to cache on client
typedef struct  {
    unsigned int route; // route
    long ts1; // first packet sending time
    long ts2; // last packet sending time
    double widthcache; // sending rate
} Senddata;

//
typedef struct  {
    int trid;      // Train ID
    unsigned short int id;  // received packet flag
} Recvdata;

// parameters of sending module for main path
typedef struct  {
    int s;
    struct addrinfo *res;
    Senddata *sendtimelist;
    long *sleeptimecontrol;
    double *width;
    int *trainid;
} SM_arg;

// parameters of sending module for sub path
typedef struct  {
    int s;
    char relay[NI_MAXHOST]; // relay node name/IP (NI_MAXHOST is defined if netdb.h)
    struct addrinfo *res;
    Senddata *sendtimelist;
    long *sleeptimecontrol;
    double *width;
    int *trainid;
} SS_arg;

// parameters of receiving module
typedef struct  {
    int s;
    struct addrinfo *res;
    Senddata *sendtimelist;
    char relayhost[NI_MAXHOST];
    int cs;
    struct addrinfo *cres;
} RS_arg;
```

```
// parameters of control module
typedef struct  {
    int s;
    struct addrinfo *res;
    Senddata *sendtimelist;
    long *sleeptimecontrol;
    double *width;
} RC_arg;

typedef struct  {
    int average;
    int difference;
} MA_RESULT;

/*------------------------global variables----------------------------*/
int packet_size[MAXROUTE] = { BUFF * 1000 << 3, BUFF * 1000 << 3 };
int train_size[MAXROUTE] = {(TRAIN - 1) * ETHERSIZE << 3, (TRAIN - 1) * (ETHERSIZE + 24) << 3};
char mode;
int error;

/*------------------------declare functions---------------------------------*/

int server_start(char *server_port);
int client_start(char *server_address, char *server_port, char *relay_address, char modeflag);
struct addrinfo *make_socket(char *hostname, char *port, int *s, const int flag);
void *send_main(void *arg);
void *send_sub(void *arg);
void *recv_serv(void *arg);
int nego_server(char *port, char *relayhost, char *clienthost, struct addrinfo **rres1, int *rs1, struct addrinfo **sres1, int *ss1);
int nego_client(char *host, char *port, char *relayhost, struct addrinfo **sres1, int *ss1, struct addrinfo **rres1, int *rs1);
void *recv_client(void *arg);
double bandwidth_calibration(double width);
int bitcount(unsigned short int recv_flag);

/*------------------------implementation------------------------------*/
int main(int argc, char **argv) {
    char port[4], serv_addr[NI_MAXHOST], relay_addr[NI_MAXHOST];

    fprintf(stdout, "argc is %d¥n", argc);
    mode = getopt(argc, argv, "SC12");

    if(argc == 3 || argc == 5)  {
        switch(mode)  {
            case 'S': {   // server
                fprintf(stdout, "run as server¥n");
                strcpy(port, argv[optind]);
                server_start(port);
            } break;

            case 'C':      // client multi path
                fprintf(stdout, "run as client¥tMultipath¥n");
                break;

            case '1':      // client main path only
                fprintf(stdout, "run as client¥tOnly MAIN route¥n");
                break;

            case '2':      // client sub path only
                fprintf(stdout, "run as client¥tOnly SUB route¥n");
                break;

            default: {
                fprintf(stdout, "mode¥n");
                fprintf(stdout, "[-S] : as a server¥n");
```

C.4

```
            fprintf(stdout, "[-C] : as a client multipath  [-1] : use only main route  [-2] : use only sub route¥n");
            exit(0);
        }
    }

    if(mode == 'C' || mode == '1' || mode == '2') {
        strcpy(serv_addr, argv[optind]);
        strcpy(port, argv[optind+1]);
        strcpy(relay_addr, argv[optind+2]);

        client_start(serv_addr, port, relay_addr, mode);
    }
  }
  else  {
    fprintf(stdout, "usage¥n");
    fprintf(stdout, "as a server : %s -S port¥n", argv[0]);
    fprintf(stdout, "as a client : %s mode server_address port relayhost_address¥n", argv[0]);
    exit(0);
  }

  return 0;
}

int server_start(char *server_port) {
  int rs, cs;
  struct addrinfo *rres, *cres;
  Senddata sendtimelist[LISTMAX];
  RS_arg *rs_arg;
  pthread_t rmt;

  char relayhost[NI_MAXHOST];
  char clienthost[NI_MAXHOST];

  // negotiate to client
  if(nego_server(server_port, relayhost, clienthost, &rres, &rs, &cres, &cs) < 0) {
    fprintf(stdout, "cannot negotiation¥n");
    exit(1);
  }

  rs_arg = (RS_arg *)malloc(sizeof(RS_arg));
  rs_arg->s = rs;
  rs_arg->res = rres;
  rs_arg->sendtimelist = sendtimelist;
  sprintf(rs_arg->relayhost, "%s", relayhost);
  rs_arg->cres = cres;
  rs_arg->cs = cs;

  // start receive module
  error = pthread_create(&rmt, NULL, recv_serv, (void *)rs_arg);

  // wait for end of each thread
  pthread_join(rmt, NULL);

  freeaddrinfo(rres);
  freeaddrinfo(cres);

  return 0;
}

//
int client_start(char *server_address, char *server_port, char *relay_address, char modeflag) {
  int ss, rs, trainid;
  struct addrinfo *rres, *sres;
  char recvport[10];
  SM_arg *sm_arg;
```

```
    SS_arg *ss_arg;
    RC_arg *r_arg;
    pthread_t mt, st, rt;

    long sleeptimecontrol[MAXROUTE];
    double width[MAXROUTE];

    Senddata sendtimelist[LISTMAX];

#ifdef END_DEBUG
    Packetdata lastmsg;
    sprintf(lastmsg.msg, "ENDTRIGGER");
#endif

    // negotiate to server
    if(nego_client(server_address, server_port, relay_address, &sres, &ss, &rres, &rs) < 0) {
        fprintf(stdout, "cannot negotiation¥n");
        exit(1);
    }

    // variable initialize
    width[SUB] = MINWIDTH;
    width[MAIN] = MINWIDTH;
    sleeptimecontrol[SUB] = 999999;
    sleeptimecontrol[MAIN] = 999999;

    trainid = 0;

    sprintf(recvport, "%d", atoi(server_port)+1);

    // control module parameters
    r_arg = (RC_arg *)malloc(sizeof(RC_arg));
    r_arg->s = rs;
    r_arg->res = rres;
    r_arg->sendtimelist = sendtimelist;
    r_arg->sleeptimecontrol = sleeptimecontrol;
    r_arg->width = width;

    if(modeflag == 'C' || modeflag == '1')  {
        // when main path is active
        sm_arg = (SM_arg *)malloc(sizeof(SM_arg));
        sm_arg->s = ss;
        sm_arg->res = sres;
        sm_arg->sendtimelist = sendtimelist;
        sm_arg->sleeptimecontrol = sleeptimecontrol;
        sm_arg->width = width;
        sm_arg->trainid = &trainid;
        // start sending module for main path
        pthread_create(&mt, NULL, send_main, (void *)sm_arg);
    }

    if(modeflag == 'C' || modeflag == '2')  {
        // when sub path is active
        ss_arg = (SS_arg *)malloc(sizeof(SS_arg));
        ss_arg->s = ss;
        ss_arg->res = sres;
        sprintf(ss_arg->relay, "%s", relay_address);
        ss_arg->sendtimelist = sendtimelist;
        ss_arg->sleeptimecontrol = sleeptimecontrol;
        ss_arg->width = width;
        ss_arg->trainid = &trainid;
        // start sending module for sub path
        pthread_create(&st, NULL, send_sub, (void *)ss_arg);
    }
```

```
    // start control module
    pthread_create(&rt, NULL, recv_client, (void *)r_arg);

    // wait for end of control module
    pthread_join(rt, NULL);

    if(modeflag == 'C' || modeflag == '1')
        // wait for end of sending module for main path
        pthread_join(mt, NULL);

    if(modeflag == 'C' || modeflag == '2')
        // wait for end of sending module for main path
        pthread_join(st, NULL);

#ifdef END_DEBUG
    sendto(ss, &lastmsg, sizeof(lastmsg), 0, sres->ai_addr, sres->ai_addrlen);
#endif

    close(ss);
    close(rs);

    freeaddrinfo(rres);
    freeaddrinfo(sres);

    return 0;
}

struct addrinfo *make_socket(char *hostname, char *port, int *s, const int flag) {
    struct addrinfo hints, *res, *res0;
    char hbuf[NI_MAXHOST], sbuf[NI_MAXSERV];

    //getaddrinfo
    memset(&hints, 0, sizeof(hints));
    hints.ai_socktype = SOCK_DGRAM; // use UDP socket
    hints.ai_family = AF_INET6;     // use IPv6

    if(flag == 1) { // if server socket
        hints.ai_flags = AI_PASSIVE;
    }

    if (getaddrinfo(hostname, port, &hints, &res0)) {
        fprintf(stdout, "error at getaddrinfo, %s, %s¥n", hostname, port);
        exit(1);
    }

    /*---------------------- create socket -------------------------*/
    for(res = res0; res; res = res->ai_next)  {

        if ((error = getnameinfo(res->ai_addr, res->ai_addrlen, hbuf, sizeof(hbuf), sbuf, sizeof(sbuf), NI_NUMERICHOST |
NI_NUMERICSERV)) != 0) {
            fprintf(stdout, "getnameinfo : %s %s : %s¥n", hostname, port, gai_strerror(error));
            continue;
        }

        fprintf(stdout, "Destination¥t%s¥tPort¥t%s¥n", hbuf, sbuf);

        if ((*s = socket(res->ai_family, res->ai_socktype, res->ai_protocol)) < 0) {
            continue;
        }

        if (flag == 1 && bind(*s, res->ai_addr, res->ai_addrlen) < 0) {
            close(*s);
            continue;
        }
```

```
        return res;
    }
    /*---------------------- create socket -------------------------*/

    fprintf(stdout, "cannot make socket for %s, %s¥n", hostname, port);
    exit(1);
}

// sending module for main path on clinent
void *send_main(void *arg) {
    int packet_id;
    SM_arg *argp = (SM_arg *)arg;
    int s;
    struct timeval now;
    struct addrinfo *res;
    struct timespec sleeptime;
    long *sleeptimecontrol;
    int *train_ptr;
    double *width;
    int array_method_main;
    int trainID_main;
    Senddata *sendtimelist;
    Packetdata sendpacket;

    // parameter extract
    s = argp->s;
    res = argp->res;
    sendtimelist = argp->sendtimelist;
    sleeptimecontrol = argp->sleeptimecontrol;
    width = argp->width;
    train_ptr = argp->trainid;
    free(argp);

    // sending interval initialize
    sleeptime.tv_sec = 0;

    sendpacket.route = MAIN;
    sprintf(sendpacket.msg, "maintestmessage");

    // ----------- train sending loop ------------ //
    while ((trainID_main = (*train_ptr)++) <= SENDPACKETNUM) {

        /* train info buffer index */
        array_method_main = trainID_main % LISTMAX;

        /* Train initialize */
        sendpacket.trid = trainID_main;
        sendtimelist[array_method_main].route = MAIN;
        sendtimelist[array_method_main].widthcache = width[MAIN];
        sendtimelist[array_method_main].ts1 = 0;
        sendtimelist[array_method_main].ts2 = 0;

        /* Packet sending loop */
        for (packet_id=0; packet_id < TRAIN; packet_id++) {
            sendpacket.id = 1 << packet_id;
            sendto(s, &sendpacket, sizeof(sendpacket), 0, res->ai_addr, res->ai_addrlen);
            if (packet_id == 0) {
                gettimeofday(&now, NULL);
                sendtimelist[array_method_main].ts1 = now.tv_usec;
            } else if (packet_id == (TRAIN - 1)) {
                gettimeofday(&now, NULL);
                sendtimelist[array_method_main].ts2 = now.tv_usec;
            }
            sleeptime.tv_nsec = sleeptimecontrol[MAIN];
            nanosleep(&sleeptime, NULL);
```

```
        }
    }
    // ----------- end of train sending loop ------------ //
}

// sending module for sub path on client
void *send_sub(void *arg){
    int packet_id;
    struct timeval now;
    SS_arg *argp = (SS_arg *)arg;
    int s;
    char relay[NI_MAXHOST];
    struct addrinfo *res;
    long *sleeptimecontrol;
    int *train_ptr;
    double *width;
    int array_method_sub;

    struct addrinfo hints, *rres;
    struct msghdr mheader;
    struct iovec message;
    struct sockaddr_in6 *I1;
    struct cmsghdr *cmsgptr;
    void *ptr;

    struct timespec sleeptime;

    int trainID_sub;

    Senddata *sendtimelist;
    Packetdata sendpacket;

    s = argp->s;
    res = argp->res;
    sprintf(relay, "%s", argp->relay);
    sendtimelist = argp->sendtimelist;
    sleeptimecontrol = argp->sleeptimecontrol;
    width = argp->width;
    train_ptr = argp->trainid;
    free(argp);

    message.iov_base = (void *)&sendpacket;
    message.iov_len = sizeof(sendpacket);

    memset(&mheader, 0, sizeof(mheader));
    mheader.msg_name = res->ai_addr;
    mheader.msg_namelen = res->ai_addrlen;
    mheader.msg_iov = &message;
    mheader.msg_iovlen = 1;

    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_INET6;

    if(getaddrinfo(relay, NULL, &hints, &rres)) {
        fprintf(stdout, "send_sub:%s¥n", relay);
        exit(1);
    }

    ptr = malloc(inet6_rthdr_space(IPV6_RTHDR_TYPE_0, 1));
    cmsgptr = inet6_rthdr_init(ptr, IPV6_RTHDR_TYPE_0);

    I1 = (struct sockaddr_in6 *)rres->ai_addr;
    inet6_rthdr_add(cmsgptr, &(I1->sin6_addr), IPV6_RTHDR_LOOSE);
    inet6_rthdr_lasthop(cmsgptr, IPV6_RTHDR_LOOSE);
    mheader.msg_control = ptr;
```

```
    mheader.msg_controllen = cmsgptr->cmsg_len;

    sleeptime.tv_sec = 0;

    sendpacket.route = SUB;
    sprintf(sendpacket.msg, "subtestmessage");

    while((trainID_sub = (*train_ptr)++) <= SENDPACKETNUM) {

        /* train info buffer index */
        array_method_sub = trainID_sub % LISTMAX;

        /* Train initialize */
        sendpacket.trid = trainID_sub;
        sendtimelist[array_method_sub].route = SUB;
        sendtimelist[array_method_sub].widthcache = width[SUB];
        sendtimelist[array_method_sub].ts1 = 0;
        sendtimelist[array_method_sub].ts2 = 0;

        /* Packet loop */
        for (packet_id=0; packet_id < TRAIN; packet_id++) {
            sendpacket.id = 1 << packet_id;
            sendmsg(s, &mheader, 0);
            if (packet_id == 0) {
                gettimeofday(&now, NULL);
                sendtimelist[array_method_sub].ts1 = now.tv_usec;
            } else if (packet_id == (TRAIN - 1)) {
                gettimeofday(&now, NULL);
                sendtimelist[array_method_sub].ts2 = now.tv_usec;
            }
            sleeptime.tv_nsec = sleeptimecontrol[SUB];
            nanosleep(&sleeptime, NULL);
        }

    }
}

// receiving module on server
void *recv_serv(void *arg)  {

    int ok = 0;
    int ng = 0;

    Packetdata recvpacket;
    fd_set fds;
    struct timeval now, timeout;

    Replypacket reply[MAXROUTE];
    Recvdata old[MAXROUTE];

    int on = 1;
    int recvMsgSize;
    struct sockaddr_storage from;
    socklen_t fromlen;

    //for reply
    char relayhost[NI_MAXHOST];
    int cs;
    struct addrinfo *cres;

    //msghdr
    struct addrinfo hints, *replyres;
    struct msghdr mheader;
    struct iovec message;
    struct sockaddr_in6 *I1;
```

C.10

```
struct cmsghdr *cmsgptr;
void *ptr;

//arguments
RS_arg *argp = (RS_arg *)arg;
int s;
struct addrinfo *res;
Senddata *sendtimelist;

s = argp->s;
res = argp->res;
sendtimelist = argp->sendtimelist;
sprintf(relayhost, "%s", argp->relayhost);
cs = argp->cs;
cres = argp->cres;

free(argp);

setsockopt(s, IPPROTO_IPV6, IPV6_RTHDR, &on, sizeof(on));
fromlen = sizeof(from);

message.iov_base = (void *)&reply[SUB];
message.iov_len = sizeof(Replypacket);

memset(&mheader, 0, sizeof(mheader));
mheader.msg_name = cres->ai_addr;
mheader.msg_namelen = cres->ai_addrlen;
mheader.msg_iov = &message;
mheader.msg_iovlen = 1;

//for reply
//getaddrinfo for relay node
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_INET6;
error = getaddrinfo(relayhost, NULL, &hints, &replyres);
if(error) {
    fprintf(stdout, "nego_client:%s¥n", relayhost);
    exit(1);
}

//cmsg
ptr = malloc(inet6_rthdr_space(IPV6_RTHDR_TYPE_0, 1));
cmsgptr = inet6_rthdr_init(ptr, IPV6_RTHDR_TYPE_0);

I1 = (struct sockaddr_in6 *)replyres->ai_addr;
inet6_rthdr_add(cmsgptr, &(I1->sin6_addr), IPV6_RTHDR_LOOSE);
inet6_rthdr_lasthop(cmsgptr, IPV6_RTHDR_LOOSE);
mheader.msg_control = ptr;
mheader.msg_controllen = cmsgptr->cmsg_len;

memset(&reply, 0, sizeof(reply));
memset(&old, 0, sizeof(old));
old[MAIN].trid = -1;
old[SUB].trid = -1;

timeout.tv_sec = RECVTIMEOUT_SEC;
timeout.tv_usec = 0;

while(1)  {
    FD_ZERO(&fds);
    FD_SET(s,&fds);
    recvMsgSize = select(s + 1,&fds,NULL,NULL,&timeout);
    if(recvMsgSize == 0)  {
        reply[recvpacket.route].trid = INT_MAX;
        sendto(cs, &reply[MAIN], sizeof(Replypacket), 0, cres->ai_addr, cres->ai_addrlen);
```

```
            fprintf(stdout, "recv_serv timeout error %d:%d¥n", ok, ng);
            break;
        }

        if(FD_ISSET(s,&fds))  {
            // socket is ready fo reading.
            memset(&recvpacket, 0, sizeof(recvpacket));
            recvMsgSize = recvfrom(s, &recvpacket, sizeof(recvpacket), 0, (struct sockaddr *)&from, &fromlen);
        }
        if(error) {
            fprintf(stdout, "receive: %s¥n", gai_strerror(error));
            exit(1);
        }

        gettimeofday(&now, NULL);

        if(recvpacket.trid != old[recvpacket.route].trid) { // receive new train packet

            if(old[recvpacket.route].trid >= 0) {
                // send back TACK
                reply[recvpacket.route].id = old[recvpacket.route].id;
                if (reply[recvpacket.route].td2 == 0) {
                    // when td2 was not set.(last packet is lost.)
                    ng++;
                    reply[recvpacket.route].td2 = now.tv_usec;
                }

                if(recvpacket.route == MAIN) {
                    sendto(cs, &reply[MAIN], sizeof(Replypacket), 0, cres->ai_addr, cres->ai_addrlen);
                } else {
                    sendmsg(cs, &mheader, 0);
                }
            }

            // new train data initialize
            reply[recvpacket.route].trid = recvpacket.trid;
            reply[recvpacket.route].route = recvpacket.route;
            reply[recvpacket.route].td1 = now.tv_usec;
            reply[recvpacket.route].td2 = 0;
            old[recvpacket.route].trid = recvpacket.trid;
            old[recvpacket.route].id = 0;
        }

        // received flag merge
        old[recvpacket.route].id += recvpacket.id;

        if(recvpacket.id == (1 << (TRAIN - 1))) {
            // for last packet of the train
            ok++;
            reply[recvpacket.route].td2 = now.tv_usec;
        }
    }
    /*---------------------- end of while -------------------------*/

    fprintf(stdout, "Loop Out succeed¥n");
}

// negotiation on server
int nego_server(char *port, char *relayhost, char *clienthost, struct addrinfo **rres1, int *rs1, struct addrinfo **sres1, int *ss1){
    struct sockaddr_storage from;
    socklen_t len;
    int recvMsgSize;

    struct sockaddr_in6 *source;
```

```
    char clientport[NI_MAXSERV];

    *rres1 = make_socket(NULL, port, rs1, 1);

    len = sizeof(struct sockaddr_storage);

    //receive relayhost
    recvMsgSize = recvfrom(*rs1, relayhost, NI_MAXHOST, 0, (struct sockaddr *)&from, &len);
    sprintf(clientport, "%d", atoi(port)+1);

    source = (struct sockaddr_in6 *)&from;
    inet_ntop(AF_INET6, &(source->sin6_addr), clienthost, NI_MAXHOST);
    *sres1 = make_socket(clienthost, clientport, ss1, 0);
    sendto(*ss1, &relayhost, NI_MAXHOST, 0, (*sres1)->ai_addr, (*sres1)->ai_addrlen);

    return 0;
}

int nego_client(char *host, char *port, char *relayhost, struct addrinfo **sres1, int *ss1, struct addrinfo **rres1, int *rs1)  {
    char recvport[NI_MAXSERV];
    int recvMsgSize;
    socklen_t fromlen;
    struct sockaddr_storage from;
    char recvbuff[BUFF];

    struct addrinfo hints;
    char buff[NI_MAXHOST];
    struct msghdr mheader;
    struct iovec message;
    struct sockaddr_in6 *I1;
    struct cmsghdr *cmsgptr;
    void *ptr;

    *sres1 = make_socket(host, port, ss1, 0);

    sprintf(recvport, "%d", atoi(port)+1);
    *rres1 = make_socket(NULL, recvport, rs1, 1);

    //msghdr
    message.iov_base = buff;
    message.iov_len = sizeof(buff);

    memset(&mheader, 0, sizeof(mheader));
    mheader.msg_name = (*sres1)->ai_addr;
    mheader.msg_namelen = (*sres1)->ai_addrlen;
    mheader.msg_iov = &message;
    mheader.msg_iovlen = 1;

    //getaddrinfo for relay node
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_INET6;
    if(getaddrinfo(relayhost, NULL, &hints, rres1)){
        fprintf(stdout, "nego_client:%s¥n", relayhost);
        exit(1);
    }

    ptr = malloc(inet6_rthdr_space(IPV6_RTHDR_TYPE_0, 1));
    cmsgptr = inet6_rthdr_init(ptr, IPV6_RTHDR_TYPE_0);

    I1 = (struct sockaddr_in6 *)(*rres1)->ai_addr;
    inet6_rthdr_add(cmsgptr, &(I1->sin6_addr), IPV6_RTHDR_LOOSE);
    inet6_rthdr_lasthop(cmsgptr, IPV6_RTHDR_LOOSE);
    mheader.msg_control = ptr;
    mheader.msg_controllen = cmsgptr->cmsg_len;
```

```
    //send relay node
    sprintf(buff, "%s", relayhost);
    sendmsg(*ss1, &mheader, 0);

    recvMsgSize = recvfrom(*rs1, recvbuff, sizeof(recvbuff), 0, (struct sockaddr *)&from, &fromlen);

    return 0;
}

// control module on client
void *recv_client(void *arg)  {
    long *sleeptimecontrol; // sending interval [nsec]
    double *width, maxwidth = 0;
    double ws, wr;      // sending & receiving rate
    double delay, r;  // sending time & sending rate ratio
    double recv_packet_rate;
    long ts, td;  // sending receiving time
    int s, on = 1, recvMsgSize;
    int route_state[MAXROUTE] = {};
    int PC_flag = RESET;
    int num_recv_packet;
    int t, sum_offset = 0, ave_offset = 0;

    double calibrated_width;
    double width_cache[MAXROUTE];

    RC_arg *argp = (RC_arg *)arg;
    Replypacket recvpacket;
    Senddata *sendtimelist;

    struct sockaddr_storage from;
    struct timeval timeout;
    struct addrinfo *res;

    socklen_t fromlen;
    fd_set fds;

    struct timeval now;
    struct tm *now_time;

    s = argp->s;
    res = argp->res;
    sendtimelist = argp->sendtimelist;
    sleeptimecontrol = argp->sleeptimecontrol;
    width = argp->width;
    free(argp);

    timeout.tv_sec = RECVTIMEOUT_SEC;
    timeout.tv_usec = 0;

    setsockopt(s, IPPROTO_IPV6, IPV6_RTHDR, &on, sizeof(on));
    fromlen = sizeof(from);

    /*--------------------- TACK receive loop -------------------------*/
    do  {

        // TACL receive
        FD_ZERO(&fds);
        FD_SET(s, &fds);
        recvMsgSize = select(s + 1, &fds, NULL, NULL, &timeout);
        if(recvMsgSize == 0)  {
            fprintf(stdout, "¥nrecv_client timeout error¥n");
            break;
        }
        if(FD_ISSET(s,&fds))
```

```
            recvMsgSize = recvfrom(s, &recvpacket, sizeof(Replypacket), 0, (struct sockaddr *)&from, &fromlen);

        t = recvpacket.trid % LISTMAX;

        // check umber of received packets
        num_recv_packet = bitcount(recvpacket.id);

        // calc sending time & receiving time
        ts = sendtimelist[t].ts2 - sendtimelist[t].ts1;
        if(ts < 0) {
            ts += 1000000;
        }
        td = recvpacket.td2 - recvpacket.td1;
        if(td < 0) {
            td += 1000000;
        }

        // calculate sending and receiving status
        ws = ((double)packet_size[recvpacket.route] * (TRAIN - 1) / (double)ts) / 1000;
        wr = ((double)packet_size[recvpacket.route] * (num_recv_packet - 1) / (double)td) / 1000;
        delay = ts / (double)td;
        r = wr / ws;
        recv_packet_rate = (double)num_recv_packet / (double)TRAIN;

        gettimeofday(&now, NULL);
        now_time = localtime((time_t *) &now);

        // output tarin information before update
        fprintf(stdout, "¥n%2d:%2d:¥t%2d.%06d¥t", now_time->tm_hour, now_time->tm_min, now_time->tm_sec, now.tv_usec);
        fprintf(stdout, "%d¥t%d¥t", recvpacket.trid, recvpacket.route);
        fprintf(stdout, "%ld¥t%ld¥t%ld¥t%ld¥t%ld¥t%ld¥t%.3f¥t", sendtimelist[t].ts1, sendtimelist[t].ts2, ts, recvpacket.td1,
recvpacket.td2, td, delay);
        fprintf(stdout, "%x¥t%d¥t", recvpacket.id, num_recv_packet);
        fprintf(stdout, "%f¥t%f¥t%.3f¥t%f¥t%f¥t%ld¥t%ld¥t", ws, wr, r, width[MAIN], width[SUB], sleeptimecontrol[MAIN],
sleeptimecontrol[SUB]);

        // sending status judge & sending rate update
        if (recvpacket.td2 < recvpacket.td1) {
            route_state[recvpacket.route] = INVALID;
            fprintf(stdout, "invalid¥t");
        } else if (num_recv_packet < TRAIN) {
            route_state[recvpacket.route] = LOSS;
            fprintf(stdout, "loss¥t");
            width[recvpacket.route] = min(width[recvpacket.route],wr - WIDTH_SHIFT);
        } else if(r < THRESHOLD_MIN) {
            route_state[recvpacket.route] = SDELAY;
            fprintf(stdout, "delay¥t");
            width[recvpacket.route] = min(width[recvpacket.route],wr);
        } else if(r > THRESHOLD_MAX) {
            route_state[recvpacket.route] = ABNORMAL;
            fprintf(stdout, "abnormal¥t");
            width[recvpacket.route] = min(width[recvpacket.route],sendtimelist[t].widthcache - WIDTH_SHIFT);
        } else {
            route_state[recvpacket.route] = NORMAL;
            fprintf(stdout, "normal¥t");
            width[recvpacket.route] = max(width[recvpacket.route],sendtimelist[t].widthcache + WIDTH_SHIFT);
        }

        // sending rate normalize
        if (width[recvpacket.route] > MAXWIDTH) {
            width[recvpacket.route] = MAXWIDTH;
        } else if (width[recvpacket.route] < MINWIDTH) {
            width[recvpacket.route] = MINWIDTH;
        }
```

```
        // update sending interval for current path
        sleeptimecontrol[recvpacket.route] = packet_size[recvpacket.route] / bandwidth_calibration(width[recvpacket.route]);

        // output sending rate information after update
        fprintf(stdout, "%f¥t%f¥t%ld¥t%ld¥t", width[MAIN], width[SUB], sleeptimecontrol[MAIN], sleeptimecontrol[SUB]);

    } while(recvpacket.trid < (SENDPACKETNUM - 3));
    /*--------------------- end of TACK receive loop-------------------------*/

}

// bandwidth calibration
double bandwidth_calibration(double width) {
    return 0.0044 * width * width + 1.0246 * width - 0.5922;
}

// cout "1" on bitstream
int bitcount(unsigned short recv_flag)  {
    int count;

    for(count = 0; recv_flag != 0; recv_flag >>= 1) {
        if(recv_flag & 1)
            count++;
    }
    return count;
}
```