修士論文 2010年度(平成22年度)

アプリケーション層における
複数パスの同時利用による高速通信機構

An Application-level Multipath Transfer Mechanism for Fast and Reliable Communication

慶應義塾大学大学院 政策・メディア研究科

野澤 高弘

takahiro@ht.sfc.keio.ac.jp

修士論文要旨 2010年度(平成22年度)

アプリケーション層における複数パスの同時利用による高速通信機構

近年，複数のネットワークインタフェースを搭載したモバイル端末が普及している．しかし現状では，単一のネットワークインタフェースを用いた通信が一般的であり，複数のネットワークインタフェースは効率的に活用されていない．本研究では，複数のネットワークインタフェースを同時に活用し，異なるキャリアが提供するネットワークリソースを効率的に利用することにより広帯域かつ安定した通信を実現する手法，ARMS (An Application-level Multipath Transfer Mechanism for Fast and Reliable Communication) を提案する．ARMS はアプリケーション層においてネットワークリソースの統合を行なうため，既存のオペレーティングシステム及びネットワークプロトコルスタックに変更を加える必要がなく，アプリケーション開発者自身によるマルチパス転送を実現可能である．ARMS ではトランスポートプロトコルにSCTP を用いることで，通信相手のアドレスや各パスの情報を取得し，また異なるパスを流れる複数のフロー間におけるデータの同期を行う．本研究ではプロトタイプ実装による 2 つのインタフェースを利用した評価を行い，単一インタフェースを利用した通信と比べて 最大で 1.97 倍の帯域向上を実現した．また，2 つの帯域幅の差が 5 倍ある環境下においても，ほぼ理想通りの帯域向上を実現した．

キーワード：
帯域統合，マルチホーム，マルチパス転送，小型移動体端末，SCTP

慶應義塾大学大学院 政策・メディア研究科
野澤 高弘

Abstract of Master's Thesis Academic Year 2010

An Application-level Multipath Transfer Mechanism for Fast and Reliable Communication

In this paper, we present a new transport scheme that simultaneously transmits application data over multiple paths between a source and a destination host. Multi-homed hosts are becoming common, hence if communication protocols transmit data to multiple paths, available network bandwidth can be increased. Several such protocols have been proposed, however, they do not get deployed, because they enforce protocol modification with the new standardization. Our proposed An Application-level Multipath Transfer Mechanism for Fast and Reliable Communication (ARMS) is designed as a part of the application, which does not require any change of transport protocols and operating systems, resulting in the maximized chance to benefit for the multi-homed hosts. ARMS allows applications to communicate over a reliable and ordered end-to-end connection along multiple paths, using SCTP. ARMS can effectively utilize multiple distinct paths, because it transmits data along independently congestion controlled paths by SCTP. Our experimental results show that applications which implements ARMS significantly improve the throughput by using spare bandwidth of multiple paths effectively.

キーワード：
bandwidth aggregation, multi-homed mobile node, SCTP

Keio University Graduate School of Media and Governance

Takahiro Nozawa

目次

図目次

表目次

ソースコード目次

第 1 章

序論

本章では，始めに本研究の背景を述べ，次に本研究の概要を述べ，最後に本論文の構成を記述する．

1.1 本研究の背景

無線技術の発展と普及により，公共空間の多くで WiFi のような無線 LAN によるインターネットアクセスが提供されている．さらに，WiMAX のような無線MAN や，HSPA などの無線 WAN などといった，より広範囲にて接続可能なネットワーク接続環境が提供されている．HSPA にいたっては日本国内の人口カバー率 100% であり，場所を選ばずにインターネットへの接続が可能である．このように様々な無線技術の実用化に伴い，複数の無線ネットワークインターフェイスを搭載したモバイル端末が登場している．複数の無線ネットワークインターフェイスを搭載した一例が無線 LAN の WiFi と無線 WAN の HSPA の 2 つのネットワークインターフェイスを搭載している，一般に超小型計算機や高性能携帯電話などである．図 1.1 にマルチホーム環境の例を示す．図 1.1 においてモバイルノードは最初 Wireless WAN のみを用いて通信を行っているが，移動後接続されていた Wireless WAN の他にWireless LAN にも接続し，両方のネットワークを利用した通信が可能となる．このように複数のネットワークを同時に利用可能な場所が増加している

また，図 1.2 が示すようにインターネット上を流れるデータ量は毎年増加を続けている．総務省が毎年行なっている通信利用動向調査によると 2009 年末の時点でインターネット利

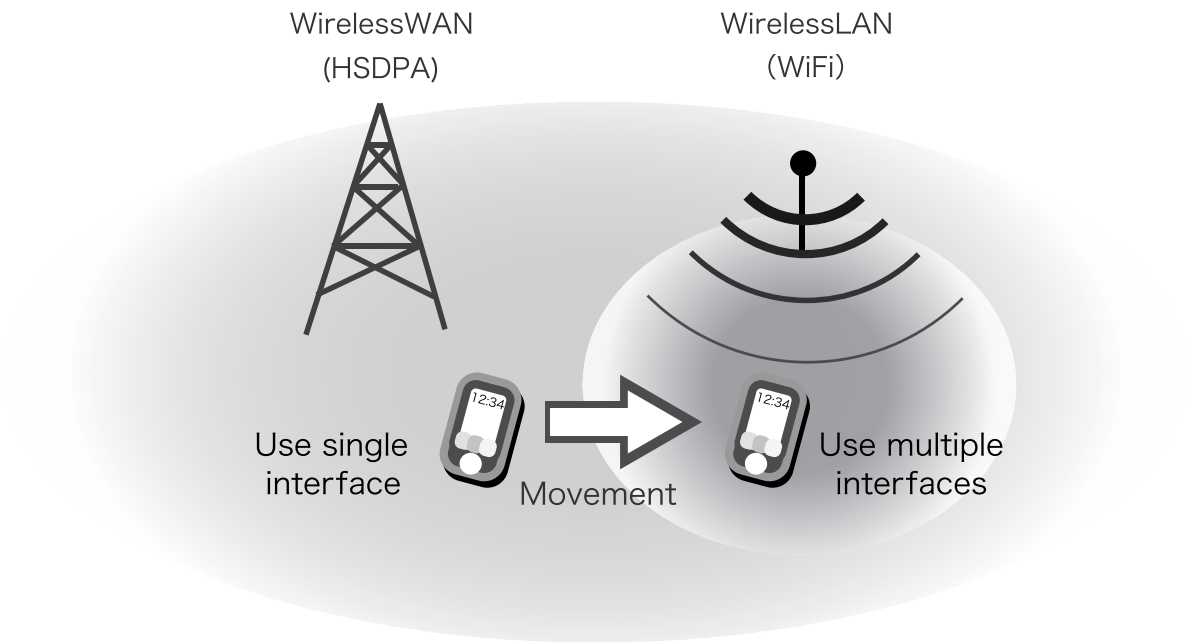


図 1.1　想定環境

用者数が 9,408 万人まで増加しており，人口普及率は 78.0% となっている．インターネットの利用法として，PCからの利用が最も多く全体の 9 割以上を占めている．しかしモバイル端末からのインターネット利用も全体の 85 % が利用しており，年々モバイル端末からのインターネット利用が増加している．さらに，国内だけでなく世界のIPトラフィックも増大傾向にある．この背景には，インターネット上を流れる動画が大きな要因となっている．Cisco Systems 社の試算によると，2014 年のインターネットトラフィックは 2009 年のトラフィック量の4倍以上になると言われている．このトラフィックのほとんどが，インターネットテレビや動画共有サービス，またファイル共有ソフト等によるものと予測されている．さらにモバイル端末からのトラフィックに限ると 2009 年から 2014 年にかけて 39 倍に増大すると予想されている．このようにこれからのインターネットの傾向として，モバイル端末にて広帯域を必要とするコンテンツが増大し，それに伴うインターネットトラフィックが発生すると予想できる．しかしながら現在のインターネットにおいては，アーキテクチャ及びプロトコルの仕様によりマルチホーム環境下にある通信端末においても単一のネットワークインタフェースのみを使用して通信を行うことが一般的である．複数のネットワークインタフェースを利用できるにもかかわらず，片方しか通信に利用しないのは資源活用の観点からは非効率的であり，必要なだけのネットワーク資源を確保する事ができず，結果として帯域不足や通信データの消失による不安定な通信となってしまう．他にも，利用しているネットワークの帯域を多く確保してしまうことにより，他のユーザが確保できるはずの帯域を圧迫してしまう可能性がある．

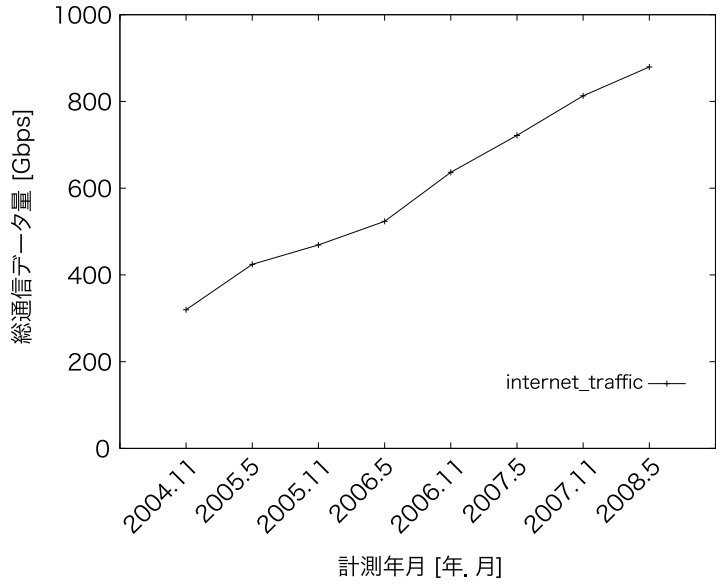


図 1.2　インターネット上のトラフィック量の推移

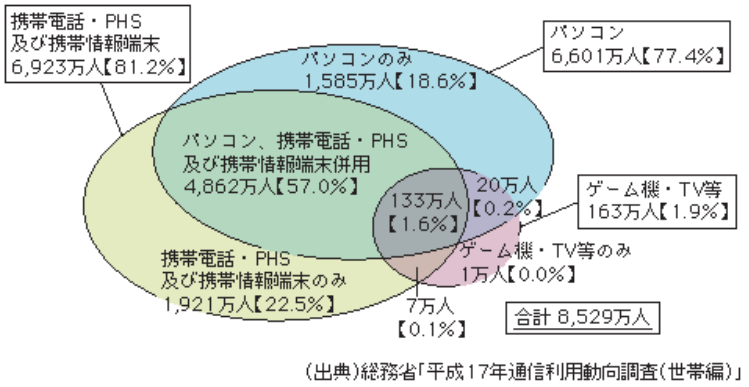


図 1.3　インターネットの利用方法

1.2　本研究の概要

以上の背景より，本研究では複数のネットワークインタフェースを用いて，異なるキャリアによって提供されるネットワークを同時に利用することにより多くのネットワーク資源を確保し，より広帯域かつ安定した通信を実現するAn Application-level Multipath Transfer Mechanism for Fast and Reliable Communication (ARMS) を提案し，様々な環境において本研究の有用性を評価する．

ARMS の特徴として，トランスポートプロトコルに複数のネットワークに接続された環境 (マルチホーム環境) にあらかじめ対応している Stream Control Transmission Protocol (SCTP) [15] を用いることで各パスを流れる複数のフローの間でデータを同期し，信頼性のあるマルチパス転送を実現する．ARMS は現在のインターネット上におけるエンドノードに広く実装されたプロトコルアーキテクチャおよびオペレーティングシステムの変更を必要としないため，アプリケーション開発者が容易に利用できる．

1.3　本論文の構成

本論文は以下で構成される．2 章ではこれまでに提案されてきた複数経路を同時に利用する手法を列挙し，それらの問題点について述べる．3 章では ARMS の設計について述べる．4 章では ARMS 実装について述べ，5 章では複数の宛先への最適なデータの振り分け手法の検討を行う．6 章では ARMS の評価実験結果と考察について述べる．最後に 7 章にて本論文をまとめ，今後の展望について述べる．

第 2 章

関連研究

本章では，これまでに実現されてきたマルチパス転送について述べ，それぞれの実現手法やその効果，特徴について述べる．本研究におけるマルチパス転送の実現手法との比較を行う．

2.1 関連研究

これまでに提案されてきた end-to-end で複数パスを利用して通信速度を向上させる手法として，本研究とは異なり複数のソケットを用いて実現する手法である PSocket [19]や，Parallel TCP [5] などのアプリケーションにて実現する手法が挙げられる．また，オペレーティングシステム内で実現する方法には大きく 4 つの手法に分類できる．1 つ目の手法がソケットを改良することによって実現した SBAM [7] である．2 つ目の手法が現在最も一般的に用いられているトランスポートプロトコル TCP を改良して実現している手法の pTCP [8]，mTCP [14]，AMS [18]， [6]， [10] などが挙げられる．3 つ目の手法に SCTP を改良して実現している手法である，CMT [9] や [4]，[3] などが挙げられる．最後に新たなトランスポートプロトコルにおいて実現している手法である，R-MTP [12]が挙げられる．以上で列挙したものをアプローチ方法から 5 つの場合に分け，それぞれの中でより実現可能生の高いものについて以下に詳しく述べる．

2.2 アプリケーションアプローチ

- PSocket

  PSocket [19] は，オペレーティングシステムの改変を必要とせずに複数のソケットを作成し転送速度の向上を実現している．通常の TCP ソケットを複数用いてデータを送信した場合と比較し，PSocket を用いた転送では最大で約 2 倍の転送速度を計測している．また，ソケットが 1 つの場合と比較し，ソケットが 5 個以下であれば理想に近い転送速度を計測している．しかし，マルチホーム環境には適応しておらず，一つのアドレスで複数のソケットを作成し，複数セッションを張る事で実現している．そのためアドレスの動的な増加・減少・変更のような状況については考慮されていない．

2.3 ソケット改良アプローチ

- SBAM

  SBAM [7] では，オペレーティングシステム内に実装されているソケットを改良することによって複数の帯域統合を実現している．SBAM はソケット内に実装したネットワークモニタリング部において，アドレスの増減の監視をすると同時に packet pair 方式 [11] を用いて，帯域情報を取得している．取得した帯域幅を基として転送データの振り分けを行う．転送データは SBAM システム内にてトランスポートプロトコル

が提供するシーケンス番号とは異なるシーケンス番号を付けられる．データの信頼性を要求される TCP のような通信では，受信側ノードにてシーケンス番号に基づき受信データの再構築を行う．このため SBAM 実装は両エンドホストに必要となる．また，通信に使用していたアドレスが利用不可能になってしまった場合は，他のネットワークインタフェースを用いてデータ送信を行う．この際送信キューに若いシーケンス番号を持ったキューが入る可能性がある為，シーケンス番号の若い順に並び替えを行う．

評価として UDP を使ったデータ転送アプリケーションとの比較を行っている．結果としては，UDP アプリケーションとほぼ同じ転送速度の実現にとどまった．これは，複数アドレスを用いる為，デフォルトゲートウェイを宛先に応じての切り替えが必要となり，転送アルゴリズム以外でのオーバーヘッドが大きいという欠点が存在するからである．また，パケット自体にシーケンス番号をSBAM 内で付ける為，1 パケットあたりの送信可能データ量も小さくなってしまうという欠点も存在する．

2.4 TCP アプローチ

- pTCP

  pTCP [8] はトランスポートプロトコルの TCP を改良する事でマルチパス転送を実現した研究の 1 つである．pTCP ではアプリケーションがソケットを作成すると，pTCP 内部にて自動的に利用可能なネットワークインタフェースの数だけ TCP-v が作成される．pTCP と TCP-v はお互いに *open/close, send/receive* のような 8 つの命令で動いている．また pTCP 内にある send buffer を TCP-v 内にある virtual send buffer が共有して利用している為，*bandwidth-delay product (BDP)* が異なる経路に対しても送信に必要な最適なバッファサイズを要求する事が可能である．通信している最中にアドレスが利用不可能になってしまった際は，pTCP から TCP-v に対して *close* 命令を出す．同様に新しいアドレスを発見した際も動的に *open* 命令を出すので，アドレスの動的な追加／削除にも対応する．

  pTCP の評価として理想値との比較と，帯域情報を取得しその帯域幅に応じて転送比率を決定してデータ転送をするアプリケーションを作成し，比較を行っている．帯域幅が一定を保つとき，pTCP とアプリケーションの両方ともほぼ理想値を計測した．しかし，帯域が大きく揺れる環境下において，アプリケーションはネットワークインタフェースの数が増えるほど理想値とは遠くなり，ネットワークインタフェースが 5 つのときに理想値の約 6 割にとどまった．一方でpTCP はネットワークインタフェースの個数がいくつの時であっても，ほぼ理想の帯域幅を計測した．また，マル

チパス転送を行っている際に一つのネットワークが一時的に使用できなくなった場合，アプリケーションでは全てのネットワークにて一定期間接続性を失ったが，pTCP ではほぼ途切れる事なく通信の継続を実現している．

- AMS
  AMS [18] もまた TCP を改良してマルチパス転送を実現した研究の 1 つである．AMS が追加した TCP オプションは以下の 3 つである．
  1. AMS Permitted
     両エンドホストが AMS をサポートしているかどうかを確認する．
  2. AMS Control
     アドレスの追加や削除などの，動的な変更を相手に通知する．
  3. AMS Common
     送信データを再構築する為の統一シーケンス番号を格納する．

  AMS ではデータ送信を行う前に双方のアドレスのペアリングを行う．データ転送のスループットを最大化する為には，最適なペアリングが必要である．そこで，AMS では通信開始時に全てのアドレスペアに *SYN* パケットを送り，それに対する *SYN/ACK* が返って来た時点の RTT を用いてアドレスペアを行う．
  評価として，スループットの理想値と比較を行っている．コネクションの数が2 つの場合はほぼ理想値を計測したが，3 つに増えると 7 % 程理想値を下回っている．これはアドレスの増加により，送信リストの管理などのオーバーヘッドが増加した為と考えられる．また，動的なアドレスの増加に対しては途切れる事なくスループットの伸びを計測したが，動的なアドレスの減少では，約 5 秒ほど転送が中断された．

2.5 SCTP アプローチ

- CMT
  CMT [9] はトランスポートプロトコルである SCTP を改良する事で実現した手法である．しかし CMT を使用するには以下に示す 3 つのデメリットが存在する．(1) 必要以上に早く再送をしてしまう．これにより経路上に余計なトラフィックが増加してしまいより多くのデータを送信できなくなる．(2) cwnd の伸びが過度に遅い．これによりより多くのパケットを直ちに送信する事が可能であるにもかかわらず，cwnd 分しか送信できなくなってしまう．(3) ack traffic の増加．パケットを受信して直ちに ack を返してしまうと，送信順と到着順が変わってしまった場合に不要な再送が起こってしまう．これらの問題を解決する為に，以下の 5 つの再送ポリシを定義

し，それぞれのポリシを評価している．

1. RTX-SAME
   再送は必ずデータの送信を行った宛先へ行う．
2. RTX-ASAP
   再送するタイミングの際に cwnd に空きのある宛先へ再送される．もし，複数の宛先の cwnd が利用可能である時，ランダムに送信先を選択する．
3. RTX-CWND
   再送パケットは，複数ある宛先の中で再送を行う際に最も大きなcwnd を持つ宛先へ送信する．
4. RTX-SSTHRESH
   再送パケットは，複数ある宛先の中で再送を行う際に最も大きなssthresh (slow start threshold) の宛先へ送信する．
5. RTX-LOSSRATE
   再送パケットは，複数ある宛先の中で再送を行う際に最もパケットロス率の少ない宛先へ送信する．

評価として，まず CMT と SCTP アソシエーションを複数使ったマルチパス転送アプリケーションにおいてデータの転送時間の比較を行っている．結果はアプリケーションよりも約 10 % 程転送速度は向上した．次に，5 つある再送ポリシの比較を行っている．結果は，パケットロス率をベースに含んでいるRTX-CWND, RTX-SSTHRESH, RTX-LOSSRATE の 3 つのポリシがより良い結果を示した．

2.6 新たなトランスポートプロトコルを用いたアプローチ

- R-MTP

  R-MTP [12] は，トランスポートプロトコル層にてマルチパス転送を実現しているが，使用しているトランスポートプロトコルは TCP でも UDP でもSCTP でもない，ユニークなトランスポートプロトコルである．この研究は，各経路の帯域幅に応じて，パケットを転送して行くのが特徴である．帯域幅の推定方法は，*interarrival time*, *jitter*, *long run jitter* の 3 つの数値用いる．*interarrival time* とは，*arrival time* とは異なり，パケットロスなどに左右されず，一定の値である．*long run jitter* とは，全ての*jitter* を合計した値であり，0 付近を変動している．帯域幅を決める為にパケットロスも予測しなければならない，これも *jitter* と *long run jitter* の両方を用いて予測する．発見されたパケットロスも突発的なものは考慮しないが，連続的に発生する場合は最初の転送速度に戻すことでパケットロスを回避する．評価として，シングル

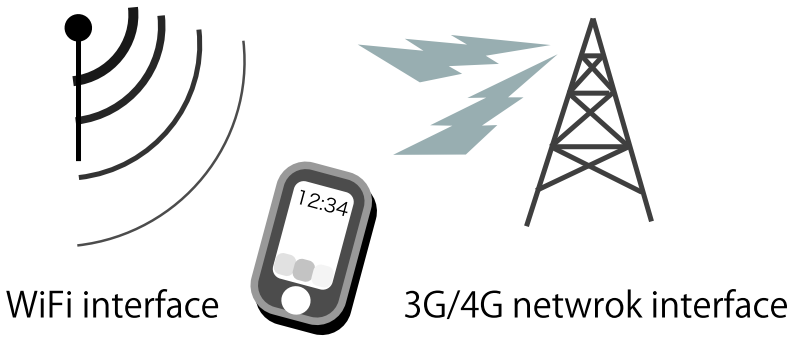


図 2.1　マルチパスの種類

パス転送とマルチパス転送のスループットの比較を行っている．パスの帯域幅の差が大きく，遅延がない場合はほとんど差は出なかったが，帯域幅　の差が小さいときは理想的なスループットを計測した．また，TCP アプリケーションとの比較を行っている．パケットロスが 30 % で起こるネットワークを利用した際，TCP では転送時間が約 250 秒かかったのに対して，R-MTP を用いた場合は約 160 秒で転送を完了した．このようにパケットロスが多く起きる環境下においても高速な転送を実現した．

2.7　まとめ

これまでに挙げてきたそれぞれのマルチパスを用いた既存研究を表 2.1 に挙げている項目を満たしているか否かをまとめていく．アプリケーションを用いて複数のセッションを張る事でマルチパス転送を実現した PSocket は，マルチホーム環境に対応していない．本論文におけるマルチホーム環境とは，図2.1 に示す通り一つのホストが，WiFi と 3G/4G ネットワークなどの複数のネットワークインタフェースを保有し、また利用可能であると定義する． PSocket 以外の全ての手法では，オペレーティングシステム内の実装を必要とし，ユーザ空間のみでは利用する事ができない．また，現在においてどの技術も標準化されておらず，CMT を除きどのオペレーティングシステムにも実装されていない．CMT は唯一FreeBSD のみに実装されているが，その他のオペレーティングシステムでは実装されていない．また，これらの手法は，通常の単一経路を利用した TCP または SCTP のアプリケーションとの共存については触れていない．通常の TCP および SCTP のアプリケーションと複数パス同時利用を利用するアプリケーションが共存するためには，新たなAPI を実装し，各アプリケーションが複数パス同時利用を行うか否かを選択する必要がある．

表 2.1 にて示された通り OS への変更を必要とせずに，動的なアドレスの増減に対応して

	OS の変更	マルチホーム環境の適応	動的なアドレスの増減への対応
PSocket	不要	×	×
SBAM	必要	○	×
pTCP	必要	○	○
CMT	必要	○	○
R-MTP	必要	○	×

表 2.1　既存研究まとめ

いるものはこれまでに存在していない.

第 3 章

ARMS

本章では，始めに本研究が提案する ARMS の概要について述べ，次に本研究におけるマルチパス転送の定義について述べ，最後に ARMS の設計について述べる．

3.1 ARMS 概要

前章にて述べた通り，これまでに様々なアプローチ方法を用いてマルチパス転送は実現されている．しかしアプリケーション層においてマルチホーム環境をサポートしたマルチパス転送は提案されていない．また，広く普及する為には実現コストを低く抑える事が重要である．これまでに多くの研究がなされたにもかかわらず 1 つも標準化に至ってない事から実現コストの重要性が示される．よって本研究ではオペレーティングシステムへ変更を加えずにアプリケーション層において，マルチホーム環境に適応した広帯域かつ高品質なマルチパス転送を実現する，An Application-level Multipath Transfer Mechanism for Fast and Reliable Communication (ARMS) を提案する．本研究手法 ARMS では，既存の研究などとは異なりオペレーティングシステムへの変更を必要としないため，アプリケーションの移植性も高く実現コストは非常に低く抑えることが可能であるため，段階的に普及すると考えられる．

3.2 ARMS におけるマルチパス転送の定義

一般にマルチパス転送という言葉は非常に抽象的である為，本論文で論じるにあたりマルチパス転送の定義を行う．一般に広い意味でマルチパス転送は以下のような場合を指す．

1. 両エンドホスト共に 1 つのアドレスしか保持していない場合
   (a) 図 3.1 (b) に示すように両エンドホスト共に 1 つのアドレスしか保持していないが，セッションを複数張っている場合

2. 両エンドホストのうち少なくともどちらかは複数のアドレスを保持し，その複数のアドレスを通信に用いる場合
   (a) 送信側ノードのみが複数のキャリアにより提供された複数のアドレスを保持し，その複数のアドレスを通信に用いる場合

   (b) 図 3.1 (a) に示すように，受信側ノードのみが複数のキャリアにより提供された複数のアドレスを保持し，その複数のアドレスを通信に用いる場合

   (c) 送信側・受信側ノードの両エンドホストが複数のキャリアにより提供された複数のアドレスを保持し，その複数のアドレスを通信に用いる場合

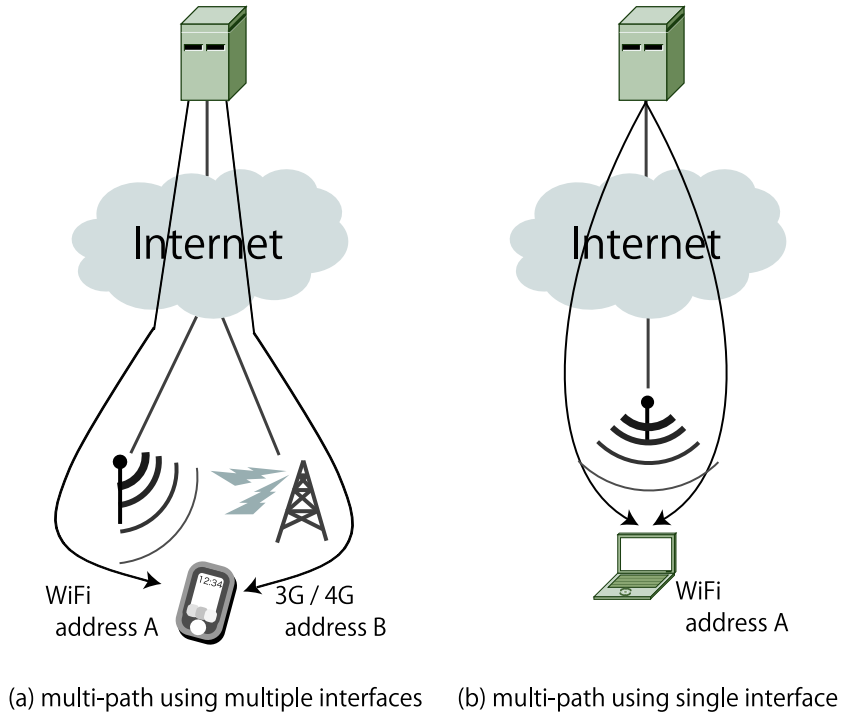


図 3.1　マルチパスの種類

以上のようにマルチパス転送は非常に多くの状況を含んだ言葉である．本研究の想定環境として，受信側ノードが複数のネットワークインタフェースを搭載している移動体通信端末を想定し，送信側ノードは単一のネットワークインタフェースを用いて通信していることを想定する．よって，以降におけるマルチパス転送の定義は図 3.1 (a) のような通信を示すこととして論じる．

3.3　ARMS 設計

ARMS は移動中におけるモバイル端末を用いた通信を想定するため，動的に変化する異なる特性の通信経路を同時に利用した信頼性のある通信を実現する．ARMS の動作概要を図 3.2 に示す．ARMS では複数の宛先を有効に活用するために，アプリケーションにおいてパスの情報を取得し，取得した情報に基づいてデータを各宛先へ転送を行う．以降では第一に，ARMS で使用するトランスポートプロトコルの選択について述べる．第二に ARMS の動作概要を述べ，第三に複数の宛先への最適な送信比率の決定方法を述べる．最後に複数の宛先があるときの宛先の指定方法を述べる．

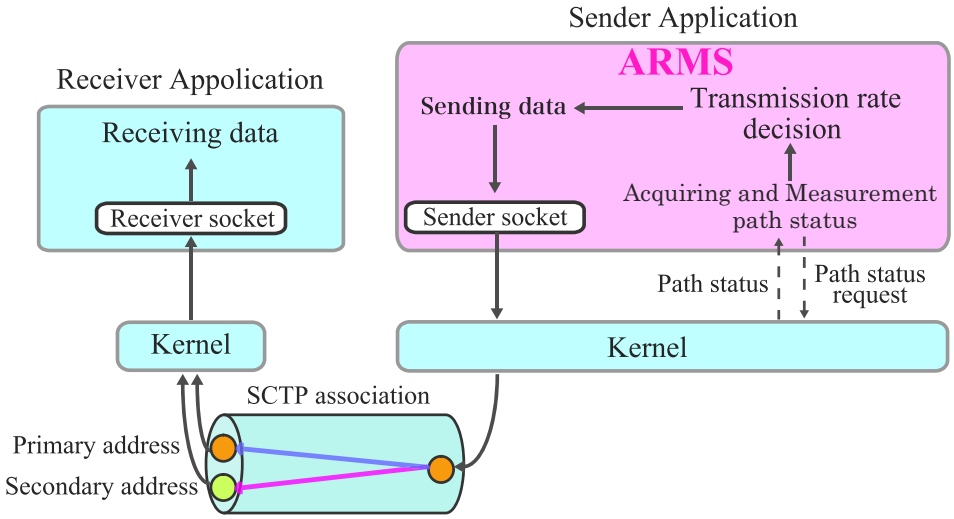


図 3.2　ARMS システム概要

3.3.1　トランスポートプロトコルの選択

アプリケーションにおいてマルチパス転送を行う際に重要なことの一つとして，トランスポートプロトコルの選択が挙げられる．既存のトランスポートプロトコルとして，TCP, UDP, SCTP, DCCP などが挙げられる．しかし本研究では信頼性のある通信の実現を目標とする．従って輻輳制御機能を持たないUDP やDCCP などにおいても実現は可能であるが，自ら輻輳制御機能を実装するコストを考慮し，TCP か SCTP のどちらかを利用した方がより容易に実現が可能であると考える．

- 通信相手のアドレスの取得
  複数パスを利用した通信を行うためには，始めに通信相手がどのアドレスを持つのか検出する必要がある．アプリケーションにおいて通信相手のアドレスを取得するためには，アプリケーションが独自に通信相手とネゴシエーションすることによって通信相手の持つアドレスリストを取得する方法が考えられる．しかしこの手法では，アプリケーション開発者の負担が大きくなる問題がある．そのため，ARMS ではトランスポート層プロトコルである SCTP の機能を用いて通信相手の持つアドレスを取得する．SCTP は TCP と同様に信頼性のある通信を実現するトランスポートプロトコルであり，TCP の 3-Way Handshake にあたる 4-Way Handshake により end-to-end のコネクションを確立する． SCTP におけるコネクションはアソシエーションと呼ばれる．

SCTP では，4-Way Handshake においてお互いが自身の利用可能なアドレスを通信相手に通知する．また，アプリケーションはアソシエーションにおける利用可能な宛先リストを SCTP が提供する API [16] によって取得できる．そのため，アプリケーションは，アドレスリストのシグナリング機能を実装することなく通信相手のアドレスを取得できる．

- データの同期
  複数のパスを用いてデータの同期をとる際に問題となるのは，受信側でのデータの再構成である．TCP では 1 つのセッションにおいて 1 つのアドレスのみしか使用できない．またセッション内ではシーケンス番号によって並び替えられるが，複数のセッションで受信したデータについては，送信側アプリケーションにて独自にシーケンス番号を付加し，それに基づいて受信側アプリケーションでの再構成が必要となる．これでは並び替えがトランスポートプロトコル層とアプリケーション層の両方で行い，かつ同期したいデータとは異なるデータを送信時に付加しなければならないので，効率的な同期とは言えない．SCTP はマルチホーム環境をサポートしているので 1 つのソケットのみで，複数のアドレスに対してデータの送信が可能である．単一ソケットにてデータを送信することにより，トランスポート層にてデータの再構築を全て行う．つまり送信側アプリケーションにて独自のシーケンス番号を付加する事も，受信側アプリケーションでの受信データの再構築も不要となる．以上より，こちらもアプリケーション開発者の負担をより軽減させるには SCTP を用いることが現実的である．

- 動的なアドレスの変化
  本研究では移動体無線通信を前提として考えている．このような環境では端末の IP アドレスが動的に変化する．TCP は永久的に一組の IP アドレスの組を通信の単位として利用するため，端末の IP アドレスが変化した場合は新たなコネクションを再度生成する必要がある．一方で，古いコネクションと新たなコネクションの間でデータを同期することは困難である．SCTP は Dynamic Address Reconfiguration (ADDIP) [17] 機能によりアドレスが変化した際も同一のアソシエーション内で通信を維持できるため，端末の IP アドレスが変化した際も古いアドレスに対するデータ送信あるいは古いアドレスからのデータ送信と新たな IP アドレスを用いたデータ送信との間でデータが同期される．ARMS はデータ転送に SCTP を用いるため，端末の動的なアドレスの変化に適応可能である．

以上の特徴をマルチパス転送を行う観点からの比較を表 3.1 に表した．

	輻輳制御	マルチホーム環境への対応	動的なアドレスの変更
TCP	○	×	×
UDP	×	×	×
SCTP	○	○	○
DCCP	×	×	×

表 3.1　トランスポートプロトコル比較表

各トランスポートプロトコルでマルチパス転送を実現しようとした際，表 3.1 内で“×”となっている部分はアプリケーション開発者が自ら実装しなければならない．アプリケーション開発者の実現コストが最も小さいのは SCTP を用いる事である事から，ARMS ではトランスポートプロトコルに SCTP を用いる事とした．SCTP は既に FreeBSD, Linux, Solaris に標準で搭載されている．また，FreeBSD の実装をベースにしたサードパーティーの実装として，Windows のドライバ実装，Mac OSX カーネルモジュールの実装が存在する．

3.3.2　複数宛先へのデータ送信

本研究ではマルチパス転送を実現するために SCTP を用いるが，通常のSCTP のマルチホーム環境における動作ではマルチパス転送は実現できない．図 3.3 (a) に通常の SCTP のデータ転送と，図 3.3 (b) に ARMS によって行うデータ転送を示す．通常の SCTP のデータ転送ではプライマリパスにしかデータを転送せず，データが到達しなかった場合にのみセカンダリパスへデータが転送される．これはSCTP のデザインが，複数のアドレスは通信の冗長化を目的としているためである．デザイン通りの転送方法では通信の接続性を維持する効果は期待できるが，ネットワーク資源を有効に使いきることはできず，通信速度の向上は見込めない．本研究では同時に利用可能なパスが複数存在したとき，図 3.3 (b) のように全てのパスを同時に用いる．それによりネットワーク資源を最大限に利用し，高速にデータ送信を行う．

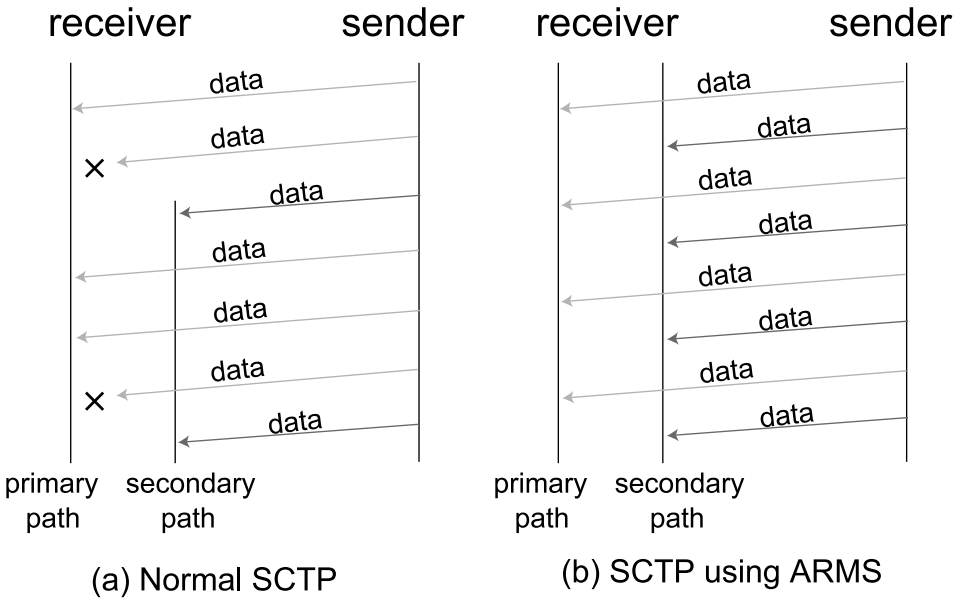


図 3.3　SCTP のマルチパス転送の比較

3.3.3　送信比率決定

本研究は，複数の異なる種類のネットワーク環境に接続されている環境(図 3.4) を前提としているが，この環境下においてパルチパス転送を実現するには以下の問題点が挙げられる．

- 帯域幅
  図 3.4 で示すマルチパス環境下では，Link 1 の帯域幅が 2Mbps，Link 2 の帯域幅が 8Mbps と異なる．このときパケットをそれぞれのパスへ同じ比率で送信を行うと，ソケットは 1 つしかないため，各パスへのパケットは同じ送信キューに入る．その為，Link 1 (帯域の小さいリンク) の帯域を埋めきる事はできるが，Link 2 (帯域幅の大きいリンク) の帯域を埋めきる事ができず，結果として，全てのパスの帯域を最大限に利用する事ができない．これは Link 1 へのパケットが送信キューから出ていかないことに起因している．以上より，利用可能な全てのパスを効率的に利用するためには，パスの帯域幅を考慮した転送比率の決定が必要となる．

- Round Trip Time (RTT)
  異なるパスにを用いた通信では，各パスの Round Trip Time (RTT) が異なると考えれる．図 4 では転送レートを帯域幅に応じて設定すると 1:4 の比率でデータが送信さ

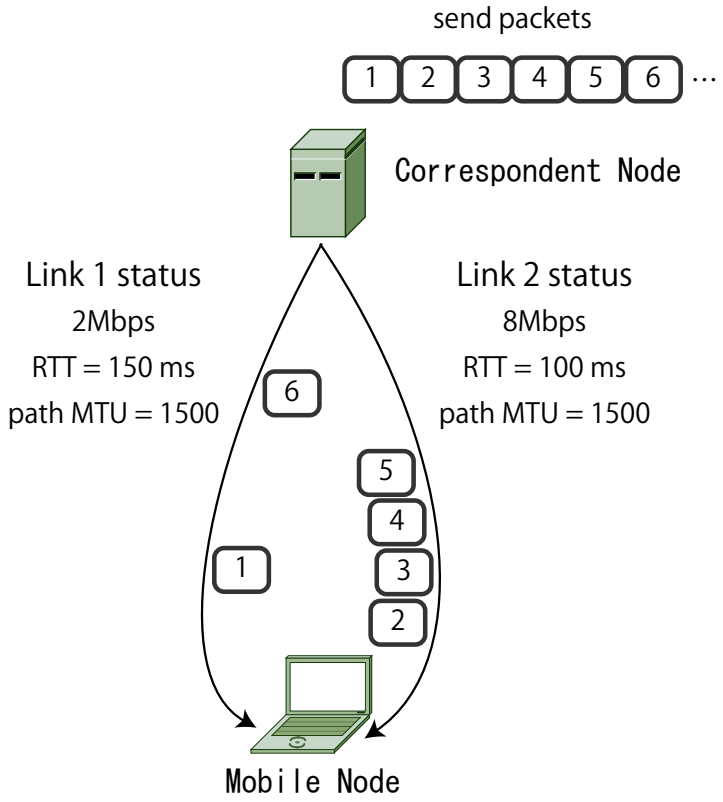


図 3.4　異なる性質のパスにおけるマルチパス転送

れるが，これらのパスは遅延が異なるため，bandwidth-delay product の比は 3:8 となる．そのため，各パスへのデータの送信レートは帯域幅だけでなく，遅延も考慮するべきである．図 4 の場合には，各パスへのデータ送信量は 3:8 の割合になるべきである．また，図4 で Link 1 へパケット 1 が送信され，次に Link 2 へパケット2 - 5 が送信されたとする．この際，それぞれのリンクは遅延が異なるため，Mobile Nodeで受信される順番はパケット2 - 5 が先に受信され25 ms 遅れてパケット1 が到着する．パケットの到着順序の変更は輻輳制御メカニズムにより通信効率の低下を招くため，できる限り避ける必要がある．また，SCTP にはdelayed sack というアルゴリズムが存在する．これは先に到着したパケットがあっても，次のパケットの到着を待ってまとめて ack を返すというネットワークの効率的な使い方である．このアルゴリズムでは 200ms の遅延を待つという設定に初期値では設定されている．よってこの値より大きな遅延差となるマルチホーム環境では，より広帯域なパスのみを利用する．なぜならば遅延が大きなパスを用いて送信したパケットは全てパケットロスとみなされ再送が要求されてしまうからである．以上のことから本研究においてRTT を考慮し

たマルチパスへの転送を行う必要がある．

以上で述べた点を考慮し，ARMS ではアプリケーション層から取得可能なパスのパラメータとして Congestion Window Size (*cwnd*) [13] と *sRTT* を使用する．*cwnd* は TCP および SCTP において送信側で管理される輻輳制御のパラメータで，SCTP において *cwnd* は宛先毎に独立して保持されている．*cwnd* は送信側が ACK の受信を待たずに送信可能なデータの量で，経路の帯域および遅延を反映している．理想値としては，delay-bandwidth product の値になる．また *sRTT* は *RTT* を一定時間取得し，その平均した値である．具体的な *cwnd* や *sRTT* の取得方法や *cwnd* を用いた最適なデータ転送比率の決定方法は第 5 章にて詳しく論じる．

第 4 章

ARMS 実装

本章では，まず始めに SCTP を用いたソケットの作成方法を述べる．次に宛先を指定して送信する為の API の利用方法について述べる，その後 ARMS が行う処理を順序通り述べて行く．

4.1 ARMS アプリケーションの流れ

ARMS は以下の手順に従って複数の宛先へデータの送信を行う.

1. *socket()* 関数によりソケットの作成

2. 通信に用いるアソシエーションの情報の取得

3. 通信相手のアドレスの取得

4. 状況に応じたデータ転送比率の決定

5. 複数のアドレスへのデータ転送

上記項目を以下において実際のソースコードを用いながら述べる.

4.2 *socket()* 関数によりソケットの作成

ARMS はアプリケーションの開始時に MN が 1 つ SCTP を用いたパケットを送信する. このパケットを受け取った CN が MN の情報を取得し，その情報に基づいて最適にデータ転送を行う.

通信を行う際に使用する *socket()* 関数だが、使用プロトコルを SCTP とする際に 2 つのソケットスタイルのどちらかをソースコード 4.1 の様に指定する. 選択するソケットスタイルの 1 つが，*one-to-one socket style* であり，他方が *one-to-many socket style* である. これらの違いは，*one-to-one soket style* では作成したソケットは TCP の様に用い，*one-to-many socket style* では作成したソケットは UDP の様に用いる. また SCTP が提供する API の使用方法等がソケットスタイル毎に異なる. ARMS ではマルチパス転送をより実現しやすい，*one-to-many socket style* を用いる事とした. このソケットスタイルの指定方法はソースコード 4.1 の 5 行目にあるように *socket()* 関数の第 2 引数に *SOCK_SEQPACKET* を入れて指定する.

ソースコード 4.1　sctp_socket.c

```
1 /* one-to-one socket style */
2 int oto_sock = socket(AF_INET6, SOCK_STREAM, IPPROTO_SCTP);
3
4 /* one-to-many socket style */
5 int otm_sock = socket(AF_INET6, SOCK_SEQPACKET, IPPROTO_SCTP);
```

4.3　通信に用いるアソシエーション情報の取得

前章までで述べたように ARMS では，転送比率の決定を CN が MN のアドレス毎に保持している *cwnd* を用いて行っている．このアドレス毎の *cwnd* を取得するにあたり，SCTP が提供する API を用いる．具体的にはソースコード 4.2 の 19, 20 行目にて用いている *getsockopt()* 関数の第 3 引数に *SCTP_GET_PEER_ADDR_INFO* を指定し，実行結果を *sctp_paddrinfo* 構造体に格納する．

ARMS で用いるのは *sctp_paddrinfo* 構造体のメンバの中で，アソシエーション ID が入る *spinfo_assoc_id* と MN のアドレス毎の *cwnd* が入る *spinfo_cwnd* の 2 つである．アドレス毎の *cwnd* の取得は，ソースコード 4.2 の 17 行目にあるように *getsockopt()* 関数の実行前に *sctp_paddrinfo* 構造体の *spinfo_address* に取得したい MN のアドレスをあらかじめ入れておくことにより指定する．

ARMS では，ソースコード 4.2 にて示された *get_assoc_info()* 関数を呼び出す事により，アプリケーションからSCTP アソシエーションの情報を取得する．アソシエーション ID を取得したい場合は，*get_assoc_info()* 関数を呼び出す際の第 3 引数に 0 を指定することにより取得する．また，*cwnd* を取得したい場合は，*get_assoc_info()* 関数を呼び出す際，第 2 引数に取得したいアドレスを，第 3 引数に 1 を指定することにより取得する．最後に *sRTT* を取得したい場合には，*get_assoc_info()* 関数を呼び出す際の第 3 引数に 2 を指定することにより取得する．

ソースコード 4.2　get_asoc_info.c

```
1 static          sctp_assoc_t
2 get_assoc_info(int sock, struct sockaddr *addr , int flag)
3 {
4     struct sctp_paddrinfo sp;
5     socklen_t       siz;
6     socklen_t       sa_len;
```

```
7
8       siz = sizeof(sp);
9       memset(&sp, 0, sizeof(sp));
10      if (addr->sa_family == AF_INET) {
11          sa_len = sizeof(struct sockaddr_in);
12      } else if (addr->sa_family == AF_INET6) {
13          sa_len = sizeof(struct sockaddr_in6);
14      } else {
15          return ((sctp_assoc_t) 0);
16      }
17      memcpy((caddr_t) &sp.spinfo_address, addr, sa_len);
18
19      if (getsockopt(sock, IPPROTO_SCTP,
20                     SCTP_GET_PEER_ADDR_INFO, &sp, &siz) != 0) {
21          return ((sctp_assoc_t) 0);
22      }
23
24      switch(flag){
25      case 0:  // Return association ID
26          return (sp.spinfo_assoc_id);
27          break;
28      case 1:  // Return specified path's CWND
29          return (sp.spinfo_cwnd);
30          break;
31      case 2:  // Return specified path's sRTT
32          return (sp.spinfo_srtt);
33          break;
34      }
35  }
```

4.4 通信相手のアドレスを取得

複数の宛先に対してマルチパス転送を行う際必ず必要となるのが，通信相手が保持している有効なのアドレスの個数や，そのアドレスである．これらの情報を取得する為にSCTP が提供する API を用いる事により，アプリケーションからの情報取得を可能とした．使い方としてはソースコード 4.3 の 14 行目にある通り *sctp_getpaddrs()* 関数を用いる．この *sctp_getpaddrs()* 関数の第 2 引数にはソースコード 4.2 にて取得したアソシエーション ID を指定する．また，実行結果を受け取る為に *sockaddr* 構造体を第 3 引数に指定する．*sctp_getpaddrs()* 関数は使用後に *sctp_freeaddrs()* 関数を行いメモリの解放を行う．

ソースコード 4.3 get_peer_addrs.c

```
1  #define REMOTE_ADDRESS_NUM 3
2
3  void
4  get_peer_addr(int sock , int assoc_id)
5  {
6      struct sockaddr *addrs, *addrp;
7      struct sockaddr_in *addrp4;
8      struct sockaddr_in6 *addrp6;
9      struct peer_addrs p_addrs;
10     int              paddr_num;
11     char             uaddr[REMOTE_ADDRESS_NUM][INET6_ADDRSTRLEN];
12
13     /* sctp_getpaddrs function */
14     paddr_num = sctp_getpaddrs(sock, assoc_id, &addrs);
15
16     addrp = addrs;
17
18     if (addrp->sa_family == AF_INET) {
19         addrp4 = (struct sockaddr_in *) addrp;
20         ++addrp4;
21         addrp = (struct sockaddr *) addrp4;
22     } else if (addrp->sa_family == AF_INET6) {
23         for(int count = 0; count < paddr_num ; count++){
```

```
24                addrp6 = (struct sockaddr_in6 *) addrp;
25                memset(uaddr, 0, sizeof(uaddr[count]));
26                if ((inet_ntop(AF_INET6, &addrp6->sin6_addr.s6_addr,
27                               uaddr[count],
28                               INET6_ADDRSTRLEN)) == NULL) {
29                    perror("inet_ntop");
30                    exit(1);
31                }
32                ++addrp6;
33                addrp = (struct sockaddr *) addrp6;
34            }
35        }
36        sctp_freepaddrs(addrs);
37    }
```

4.5 状況に応じたデータ転送比率の決定

状況に応じたデータ転送比率の決定手法は本研究の中で最も重要なアルゴリズムである．データ転送比率が正しく設定されていない場合では，複数あるネットワークリソースを効率的に利用できないために広帯域な通信や安定した通信を実現することが困難となる．本研究においては，アプリケーション層における効率的なデータ転送比率をアプリケーション層から取得可能である経路情報である *cwnd* と *sRTT*, またアプリケーションから計測可能な帯域幅などを用いて算出する．詳細な転送比率の決定アルゴリズムは次章について深く議論する．

4.6 複数のアドレスへのデータ転送とデータ受信

以上までにより取得・計算された情報を元にデータを送信して行く．複数の宛先へデータ転送をする為に ARMS では SCTP が提供する API である *sctp_sendmsg()* 関数 [16] を用いた．*sctp_sendmsg()* 関数を用いる事で 1 つのソケットから複数の宛先へのデータ転送が可能となる．通常 SCTP は複数の宛先を保持していても，実際に通信に用いるのはプライマリアドレスだけである．このプライマリアドレスが使用不可能になった際に初めてセカンダリアドレスを利用する事となる．しかしARMS では，アソシエーションに含まれる宛先を全て通信に利用する為に *sctp_sendmsg()* 関数の第 4 引数に送信したい宛先アドレスを明

示的に指定する．しかし指定したアドレスがプライマリアドレス以外のアドレス以外の場合，SCTP の仕様によりこの引数は無視され，プライマリアドレスへデータは送信される．これを防ぐ為に*sctp_sendmsg()* 関数の第 7 引数のフラグフィールドに *SCTP_ADDR_OVER* フラグを立てる事でプライマリアドレス以外のアドレスへのデータ転送を可能とした．*sctp_sendmsg()* 関数は *one-to-one socket style* でも *one-to-many socket style* でも使用可能であるが，*one-to-one socket style* では *SCTP_ADDR_OVER* フラグオプションを使用できない．このため ARMS ではSCTP のソケットスタイルを *one-to-many socket style* とした．

ソースコード 4.4　ARMS_send_data.c

```
1  void
2  ARMS_snd_rcv_data(int sock, struct sockaddr_in remote)
3  {
4      int             sent_len;
5      int             recv_len;
6      char            buf[BUFSIZE];
7
8      switch (YOUR_PROGRAM) {
9
10     case SENDER:
11   /************************************************************
12    *ssize_t                                                   *
13    *   sctp_sendmsg(int sockfd, const void *msg, size_t msgsz, *
14    *                const struct sockaddr *to, socklen_t tolen,*
15    *                uinit32_t ppid,                            *
16    *                uint32_t flag, uint16_t stream,            *
17    *                uint32_t timetolive, uint32_t context);    *
18    ************************************************************/
19   sent_len = sctp_sendmsg(sock, buf, sizeof(buf),
20         (struct sockaddr *) & remote,
21         (socklen_t) sizeof(remote),
22         0, SCTP_ADDR_OVER, 0, 0, 0);
23   /* if fail to send */
24   if (sent_len < 0) {
```

```
25          perror("sctp_sendmsg");
26          exit(EXIT_FAILURE);
27      }
28      break;
29
30        case RECEIVER:
31      /*****************************************************************
32       *ssize_t                                                        *
33       *    sctp_recvmsg(int sockfd, const void *msg, size_t msgsz, *
34       *                 struct sockaddr *from, socklen_t *fromlen, *
35       *                 struct sctp_sndrcvinfo *sri, int *msg_flag);*
36       *****************************************************************/
37      struct sctp_sndrcvinfo sri;
38      int              len, flag;
39      recv_len = sctp_recvmsg(sock, buf, sizeof(buf),
40            (struct sockaddr *) & remote,
41            (socklen_t *) & len, &sri, &flag);
42      /* if fail to recreceive */
43      if (sent_len < 0) {
44          perror("sctp_recvmsg");
45          exit(EXIT_FAILURE);
46      }
47      break;
48        }
49  }
```

4.7 アプリケーション内の関数の相関関係

以上までで作成した関数は ARMS アプリケーションの中で図 **??**のように機能する．通信に用いる SCTP ソケットは，*one-to-many* ソケットスタイルを指定してを作成する．次に，*get_assoc_info()* 関数と *get_peer_addr()* 関数によって宛先毎の *cwnd* や相手の宛先アドレスの増減などを取得する．これらの関数によって取得したデータを基に *rate_decision()* 関数にて最適なデータ転送比率を決定する．最適なデータ転送比率の決定手法は本研究に

おいてもっとも重要な部分であるため，次章にて詳細に実験を行い議論を行う．決定された転送比率に基づいて *ARMS_send_data()* 関数にて複数の宛先へデータを送信する．送信時には，宛先を明示的に指定し，かつ*SCTP_ADDR_OVER* フラグを立てる事にて複数の宛先を同時に用いたマルチパス転送を実現する．

第 5 章

複数宛先への動的な振り分け手法

本章では複数パスへの送信比率の決定手法を改良するために行なった事前実験について述べる．事前実験の結果を踏まえて本研究で用いる最適な複数パスへの振り分け手法を検討する．

5.1 複数宛先への動的な振り分けの検討方法

本研究において複数の宛先への振り分け方法は最も重要なアルゴリズムであるため，多くの手法を実際に実装し実験を行なった上で決定した．はじめに，実装した手法を検証するための実験環境を図 5.1 に示す．本実験では，Correspondent Node は単一ネットワークインタフェースのみにて通信を行い，Mobile Node は 2 つのネットワークインターフェイスを用いて通信を行うものとする．また，事前実験では Correspondent Node, Mobile Node 共に PC を用いており，ネットワークはそれぞれ 1Gbps のネットワークに接続されている．本実験環境において，Network 1, 2, 3 はそれぞれ独立したネットワークであり，異なるネットワークアドレスを保持している．これにより，Correspondent Node から Mobile Node への同一ネットワーク内でのデータ転送を起こらないようにしている．本研究では複数のネットワークに接続可能な小型移動体端末を想定しているので，実験環境をよりモバイル環境に近付けることが望ましい．Mobile Node が接続されているネットワークをモバイル環境に近づけるために Correspondent Node と Router の間にDummynet 端末を設置した．Dummynet とは，通過するパケットの送り出す量や時間を変更することにより特定の経路の帯域幅や遅延，パケットロス率などを任意の値に制御するためのブリッジノードのことである．本実験環境ではDummynet を通すことで Mobile Node が接続されているネットワークの通信帯域や遅延などを実際のモバイル環境に近づけて実験を行った．Correspondent Node,Mobile Node, Dummynet 全てのホストは OS にFreeBSD を用いており，ハードウエアスペックがボトルネックにならない程度の性能を搭載したマシンにて実験を行なった．実験中は Dummynet により，Correspondent Node と Mobile Node を結ぶ経路の帯域を 1 - 8 Mbps に，RTT を 30 - 120 ms のい範囲に設定した．尚，以降では帯域が 1 Mbps で RTT が 30 ms のパスを *path(1Mbps, 30ms)*, 帯域が 2Mbps で RTTが 60 ms のパスを *path(2Mbps, 60ms)* と表記する．また以下に挙げる事前実験はベストケースとして実験を行った．よって，本実験環境で利用しているネットワークには自分が送出するパケット以外のトラフィックがない環境を用意した．

図 5.1 に示した環境を用いて，最初に最適なマルチパス転送を行うために必要な要素の検証をするために，2 つの事前実験を行った．この結果を踏まえて，動的に各経路の情報を動的に取得し振り分けアルゴリズムを 2 つ実装した．最後に実験を行った結果から本研究にて使用するアルゴリズムを決定した．

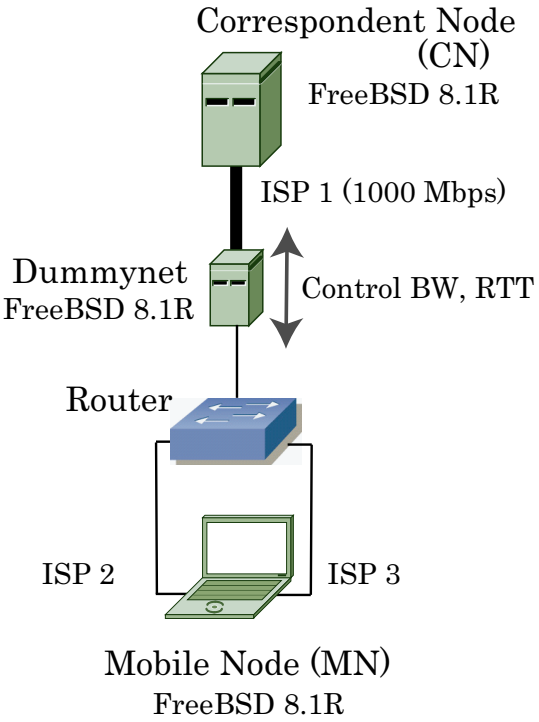


図 5.1　事前実験環境

5.2　事前実験

高速なマルチパス転送を行うために必要な経路情報を特定するために，以下の2 種類の事前実験を行った．

- 各パスへ同じ量のデータを送信
- Dummynet にて指定した帯域比率に応じてデータを送信

これらの実験方法，実験結果などを以下にて詳細に論じる．

5.2.1 各パスへ同じ比率で送信

最初に行った実験は，図5.1 の Network 2 と 3 へ同じデータ量を送信する方法で行った．実験は，それぞれの宛先へ 10Mbyte づつ，合計で 20MByte のデータをCorrespondent Node から Mobile Node へ転送を行った．この実験を行う際に，Dummynet において，帯域幅を 1, 2, 4 Mbps に，RTT を 30, 60, 120 ms のいづれかに設定した．上記のように設定して行なった実験結果を表 5.1 と図 5.2 に示す．

表 5.1 は，それぞれの実験パターンにおいて，転送完了までに要した時間［秒］を示している．表 5.1 内にて “none” と表記されている項目では，シングルパス転送を行った際の結果ということを示す．具体的に *path1(1Mbps, 30ms), path2(none)* となっている場合は，Mobile Node は Network 2 にのみ接続されていることを示す．表 5.1 から，両方の経路の帯域幅が 1Mbps であった時にのみ帯域向上が見られたが，*path2* の帯域幅を 2Mbps, 4Mbps と大きくしても，マルチパス転送に要する時間はほとんど変わらなかった．マルチパス転送を行って最も早く完了するはずの *path1(1Mbps,30ms), path2(4Mbps, 30ms)* にてマルチパス転送を行った際の時間あたりの転送データ量を示したものが図 5.2 である．図 5.2 の横軸は転送開始からの経過時間［秒］を縦軸は転送開始からの受信データ量［MByte］を示している．総転送量と総転送時間から割り出される平均転送速度は，1.80Mbps にとどまりってしまい，*path1(4Mbps, 30ms)* のみを用いたシングルパス転送のほうが高速に転送を行うことができた．

以上の結果からそれぞれの宛先へ同じ量のデータ送信を行うと，帯域幅の小さい方のパスを使い切ることはできるが，帯域幅の大きいパスは使い切ることができないということが検証された．また同時に，帯域幅が同じであればこの手法は有用であることも検証した．これらのことをまとめると，帯域幅の比率に従って送信することが出来れば，より高速なマルチパス転送が行えるという仮説をたてることができる．次に行う事前実験にてこの仮説の検証を行う．

5.2.2 静的に帯域比率で転送

最初の事前実験によりたてられた仮説を検証するために，Dummynet にて設定した帯域比を静的にプログラム内に実装して実験を行った．具体的にはDummynet にて *path1(1Mbps, 30ms), path2(2Mbps, 30ms)* と設定した場合にはプログラム内にて，*path1* へ 1 パケット送る度に *path2* へ 2 パケットを送信するようにプログラムした．Dummynet にて各経路情報を変更しながら静的に指定した比率にてマルチパスん転送を行った際の実験結果を表 5.2 と図 5.3 に示す．

$path1$ / $path2$		none	1Mbps rtt : $30ms$	2Mbps rtt : $30ms$	4Mbps rtt : $30ms$
none		—	169.60	84.64	42.35
1Mbps	rtt : $30ms$	169.60	84.74	84.65	84.67
	rtt : $60ms$	169.52	84.75	84.66	84.66
	rtt : $120ms$	169.40	84.76	84.71	84.72

表 5.1　同比率にてマルチパス転送を行った際の各パスの組み合わせにおける 20MByte の転送時間 (秒)

表 5.2 は，それぞれの実験パターンにおいて20Mbyte のデータすべてを送り終わるまでに要した時間［秒］を示している．最初に行った同じ比率にてマルチパス転送をした際とは *path2* の帯域幅が 2Mbps, 4Mbps と設定した際に大きく変わっている．一例として，*path1(1Mbps, 30ms), path2(4Mbps, 30ms)* と Dummynet にて設定した際の時間あたりの転送データ量の伸びを示したものが図 5.3 である．図 5.3 では 1Mbps, 4Mbps, 5Mbps にてシングルパス転送を行なった際の時間あたりの転送データ量も載せている．*path1(1Mbps, 30ms), path2(4Mbps, 30ms)* の転送速度はほとんど 5Mbps にてシングルパス転送をした際の転送速度と変わらなかった．

5.2.3　事前実験まとめ

以上の 2 つの事前実験より，帯域幅の比率に基づいて転送を行うと最適なマルチパス転送を実現できるという結論に至る．*path1(1Mbps, 30ms), path2(4Mbps, 30ms)* を例にとると，初めに行った同じ比率にて転送した場合は，平均転送速度は，1.80Mbps という結果になった．しかし静的に帯域幅の比率に基づいて転送を行うと，平均転送速度は4.5Mbps まで向上した．参考として，同じプログラムを用いて *path(5Mbps, 30ms), path2(none)* におけるシングルパス転送時の平均転送速度も 4.5Mbps であった．また，僅かな差ではあるが *path1* の RTT を増加させると，そう転送時間も増加する傾向が計測された．RTT の値は帯域幅ほど転送速度に影響を与えない事は示されたが，より効果的なマルチパス転送を行うことを考えるには，考慮するべき値であると言える．

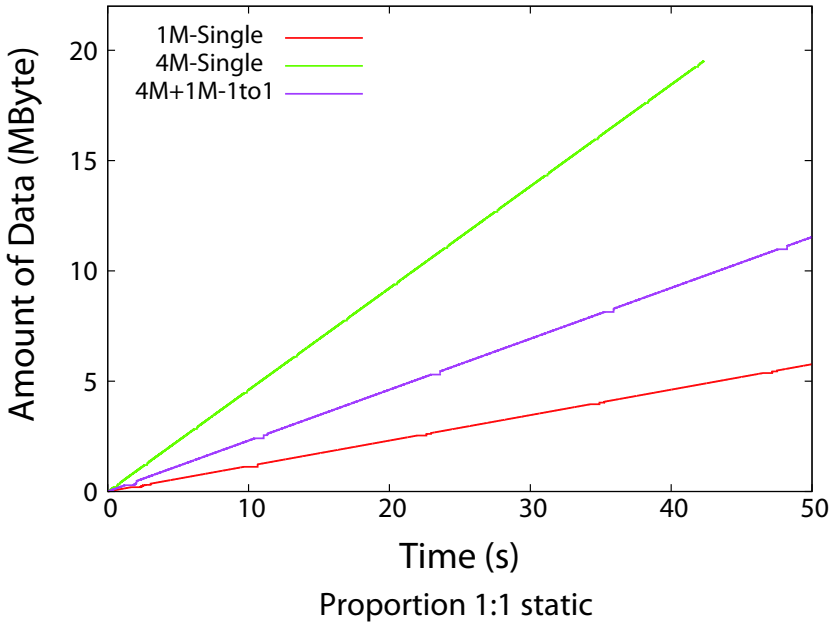


図 5.2　各パスへ同じ比率でマルチパス転送した際の転送量

5.3　*cwnd* を用いた転送比率にて送信

事前実験の結果より，最適な転送比率を導き出すには転送比率を決定する際には帯域幅と RTT を用いる劇であるという結論に至ったが，これら 2 つの値から動的に算出された *cwnd* 値を SCTP は宛先ごとに保持している．理論的に *cwnd* は動的な経路の帯域幅及び遅延を加味した値であるため，本実験ではアプリケーション層から取得できる *cwnd* を用いた転送比率決定手法を試した．この手法のメリットは，細かい頻度で動的に宛先までのパスの状態を取得することが可能なところである．具体的に *cwnd* とは，*socket* に対して送信要求を行うとすぐにパケットが送出される値であり，単位は *Byte* である．よって *cwnd* を取得し，直ちに取得した *cwnd* 分のデータを送信し，送信が完了し次第再度 *cwnd* を取得する．これを繰り返し行えば，理論上ネットワークリソースを使い切ることが可能となる．実際の実装方法としては，ソースコード 4.2 に示した *get_assoc_info()* 関数を用いて *cwnd* を取得し，具体的な比率決定手法は，動的にアプリケーション内にて各宛先への *cwnd* を取得し，そのデータ量分を *socket* に対して送信要求を行うように実装した．以下に実験手法，実験結果，考察の順に述べる．

$path1$ / $path2$		none	1Mbps rtt : $30ms$	2Mbps rtt : $30ms$	4Mbps rtt : $30ms$
none		—	169.60	84.64	42.35
1Mbps	rtt : $30ms$	169.60	84.74	56.76	33.88
	rtt : $60ms$	169.52	84.75	56.46	33.91
	rtt : $120ms$	169.40	84.76	56.55	34.04

表 5.2 静的に比率を指定してマルチパス転送行った際の各パスの組み合わせにおける 20MByte の転送時間 (秒)

- 実験手法

  図5.1 の Dummynet 端末を用いて Network 1 から Network 2, 及び Network 1 から Network 3 への帯域幅を 1Mbps もしくは 2Mbps に，また往復遅延時間を 30ms もしくは 60ms に設定して実験を行なった．このような Dummynet の設定によりシングルパスでの転送は 4 通り，マルチパスでの転送は 10 通りの合計 14 通りについて 20MByte のデータを転送に要した時間を計測した．外れ値が出ることを考慮し，全ての組み合わせにてそれぞれ 5 回づつ転送時間を計測した．実効帯域はデータ転送量を平均転送時間で割ることにより算出した．

- 実験結果

  上記の手法により実験を行った結果が以下の図5.4 の通りである．この実験では結果を 2 種類に分類できる．1 つ目が両方のパスの帯域幅が同じだった場合である．具体的に Network 2 も Network 3 も共に 1Mbps もしくは 2Mbps であった場合である．これらの環境において転送速度を計測した結果はシングルパス転送時の転送速度の 1.9 倍から 2.0 倍の速度で転送を完了した．よって 2 つのパスを効果的に利用し，理想的なマルチパス転送が行われたと言える．2 つ目に両方のパスの帯域幅が異なる場合である．具体的には，1Mpbs のパスと 2Mbps のパスを同時に用いて転送を行った場合である．この環境下における実験結果は，広帯域なパスのシングルパス転送と比較して約 1.1 倍程度の速度の向上しか計測できなかった．

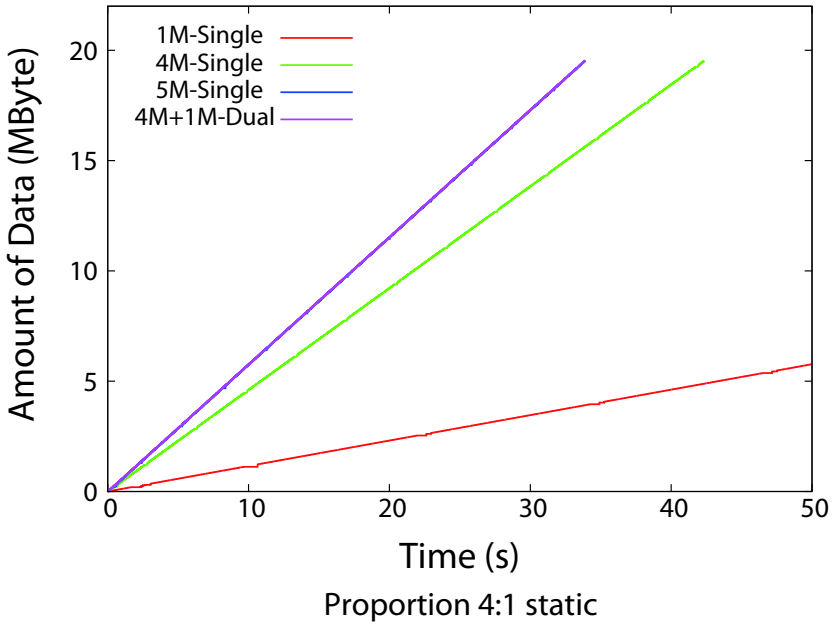


図 5.3　静的に設定した帯域比率にてマルチパス転送を行った際の転送量

以上より，*cwnd* のみで転送比率を決定したアルゴリズムでは，かなり限定された状況のみでしか効果的にマルチパスを使うことができない．

- 考察

本実験の考察をするにあたり，実験中に Correspondent Node 側のアプリケーションが送信比率を決定するために取得した *cwnd* の値をグラフにしたものが図5.5, 5.6 である．図5.5 (a) はシングルパス転送をしたときの *cwnd* の様子である．SCTP は TCP と同様の輻輳制御アルゴリズムに従って *cwnd* を上下させることが見て取れる．図5.5 (b) は両方のパスの帯域幅が同じ値に設定されている時の *cwnd* の動きを示したものである．両方の帯域が同じである場合も図5.5 (a) と同様に正常な輻輳制御アルゴリズムの挙動を示している．このように正常な *cwnd* が取得できている場合ではマルチパスを効果的に利用できていたと言える．

しかし，両方の帯域幅が異なる場合の *cwnd* は図5.6 に示されるように正常な輻輳制御アルゴリズムの挙動を示していない．図5.6 に示されるように *cwnd* が同じ値を撮り続けることはありえない．通常の輻輳制御アルゴリズムはスロースタート時を除

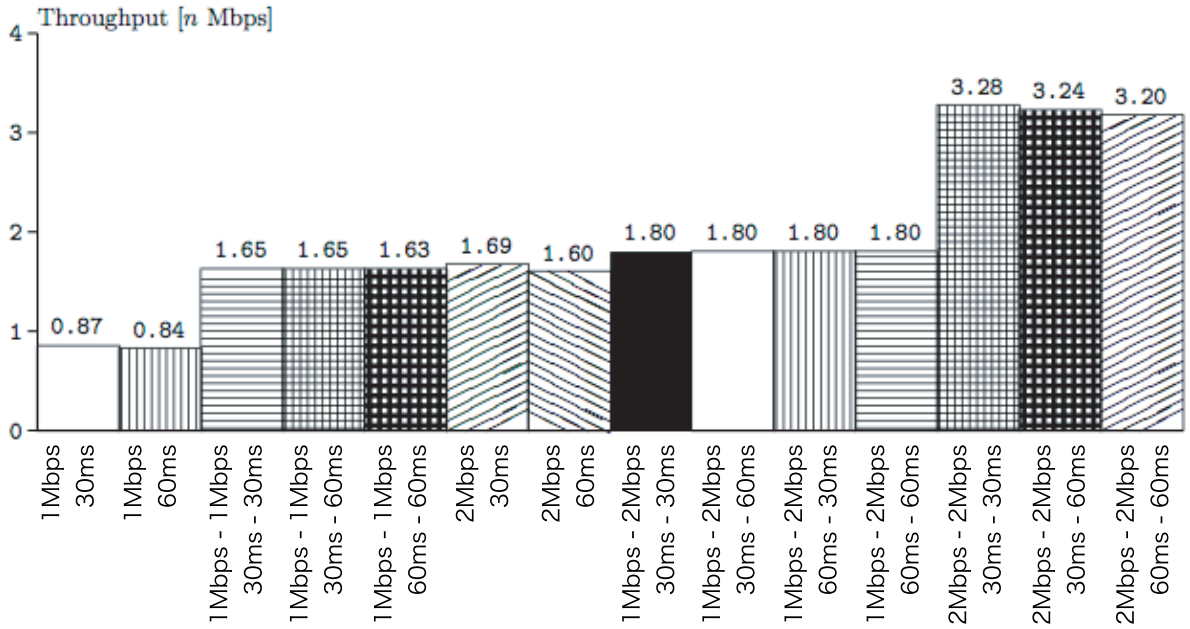
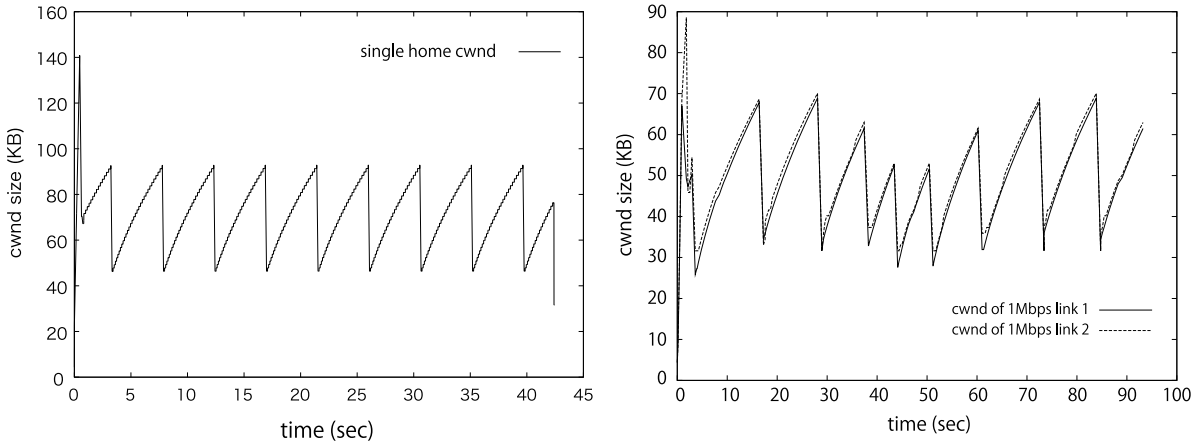


図 5.4　平均スループット比較

き，輻輳が起きていなければ 1RTT 毎に *cwnd* は1MSS(Maximum Segment Size) 分だけ上昇し，輻輳が起きれば *cwnd* は半減する．従って同じ値のまま変動しないということは正常に輻輳制御アルゴリズムが機能していないことになる．そもそも本実験では *cwnd* は正しい値であるという前提のもと行っているので，*cwnd* の値が不確かであればマルチホーム環境を有効にを利用することはできない．SCTP の通常の挙動として，複数の有効なアドレスを保持していても通信に用いるのは 1 つだけであり，他のアドレスは通信の冗長化のためだけに保持されている．従って通常の利用方法では想定されていない利用方法であったため，輻輳制御アルゴリズムが正常に機能しなかったと考えられる．

5.4　帯域幅を用いた転送比率にて送信

事前実験より，正確に帯域幅の比率に基づいてマルチパス転送を行うことが出来れば，ほぼ理想的なマルチパス転送を実現できることが実証されている．SCTP が提供する API の中に *cwnd* 以外に帯域幅を包括した値は本論文執筆時には存在しない．よって，プログラム内にて高い精度で帯域幅を計測し，計測した値に基づいて最適な転送比率を導き出す．



(a) シングルパス転送時の cwnd の挙動　　(b) 両パスの帯域が同じ際のマルチパス転送の cwnd の挙動

図 5.5　正常に cwnd が機能している場合

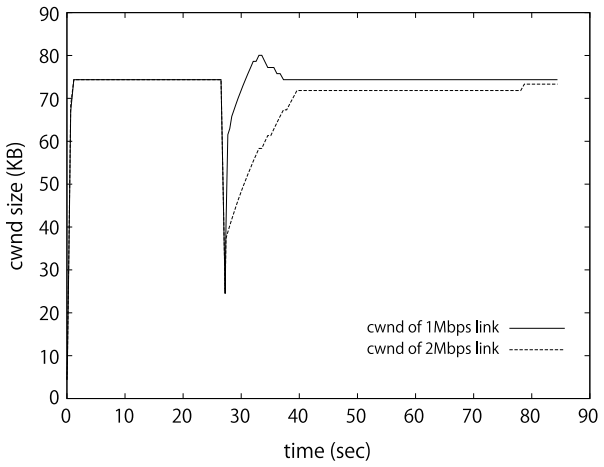


図 5.6　パスの帯域幅に 2 倍の差がある際の cwnd の動き

帯域幅をプログラム内で計測する手法を以下に挙げた．これらの手法の中から最も高い精度かつ，主であるマルチパス転送の転送速度に影響を与えないものを選択し，実装・実験を行う．

- 任意の単位時間あたりのそれぞれの宛先への転送量から算出
  単一アソシエーションからマルチパス転送を行っている際は，*sack* は送信時の宛先アドレスとは関係なくプライマリアドレスから返ってくるため，宛先をセカンダリア

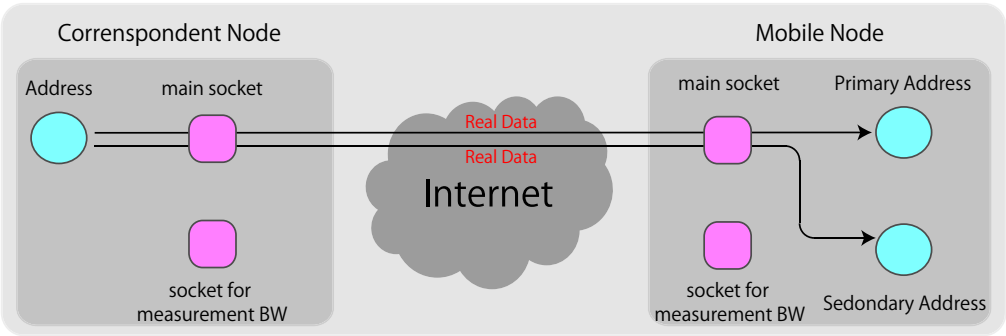
ドレスに指定して送信したパケットをパケットロスや，*RTT* の違いなどからプライマリアドレスから送信する可能性がある．よって，これでは正確にそれぞれの帯域を用いて転送したデータ量とは言えないため，精度は低かった．

- 輻輳が発生してから次の輻輳が発生するまでの転送量から算出
  帯域を最大限以上にデータを転送した際に輻輳は発生する．この事を用いて，輻輳が発生時から次の輻輳までの間に転送した量と時間を，宛先ごとに順に計測を行い，転送量を時間で割ることにより，帯域幅を算出する．しかし，上述の手法と同様に，送信したデータを指定したアドレス以外のアドレスを用いて転送されている可能性がある．実際に実装し，実験してみたところ一定の精度はあったものの，設定した帯域幅の差が大きい時や，遅延時間のさが大きい時などには高い精度を計測することはできなかった．

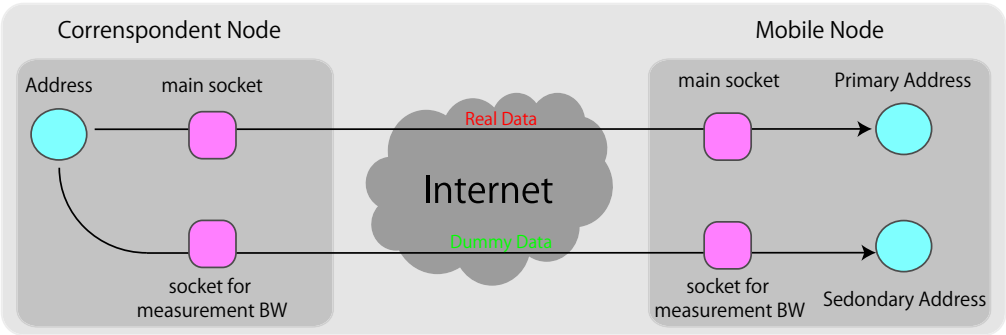
- 異なるアソシエーションを作成し，それぞれの帯域幅を同時に計測
  異なるアソシエーションを作成するとは，図 5.7 (b) に示したように，セカンダリパスへの帯域幅を新たに作成したのソケットを用いて計測を行う．プライマリパスへはデータ転送に用いているソケットをそのまま利用する．この手法では，異なるアソシエーションを用いて帯域計測を行うため，帯域計測中は，ファイル転送をシングルパスで行うことになるというデメリットを含んでいる．これは異なるアソシエーションを用いてデータを転送を行うと，それぞれ独立したシーケンス番号が SCTP パケットヘッダに付加されてしまう．従って，Mobile Node 側のトランスポート層において別々に受信データの再構築が行われることとなる．しかし，指定したアドレスがネットワークから断絶されない限り，指定した宛先に届けられるため，高い精度を計測することができた．

以上の結果を踏まえ，本研究においては異なるアソシエーションを作成し，それぞれの宛先に対してシングルパス転送を行うことにより帯域を計測することとした．帯域計測時のデータ転送は図 5.7 (b) のようになる．図 5.7 (b) でも書いたように帯域測定用に作成したソケットを通るデータは，実際に Mobile Node に対して送信をしている実データではなく，ダミーデータを送信している．帯域の計測が完了し次第，図 5.7 (a) の通常のマルチパス転送を行うように実装した．

2 つのアソシエーションを用いて帯域幅を計測し，その比率に応じた転送を行った結果を図 5.8 に示す．図 5.8 (a) は，当実験における最もマルチホーム環境を有効に活用した組み



(a) 実データ転送時のデータの流れ



(b) 帯域計測中のデータの流れ

図 5.7　実データ転送時と帯域計測時のパケットの経路の違い

合わせによるマルチパス転送時の時間あたりの転送データ量を示している．また，図 5.8 (b) は，当実験における最もマルチホーム環境を有効に活用できなかった際の組み合わせによるマルチパス転送時の時間あたりの転送データ量を示している．図 5.8 (a) では，*path1(2Mbps, 30ms), path2(2Mbps, 30ms)* のマルチパス転送を行い，総転送量から総転送時間を割ることにより算出された平均転送速度は 3.64Mbps であり，*path1(4Mbps, 30ms), path2(none)* のシングルパス転送を行なった際の平均転送速度 3.78Mbps と比較しても 4% 程度の転送効率低下にとどまった．しかし，図 5.8 (b) では，*path1(8Mbps, 60ms), path2(2Mbps, 30ms)* の組み合わせにてマルチパス転送を行い，計測された平均転送速度は 7.39Mbps であり，*path1(10Mbps, 60ms), path2(none)* のシングルパス転送を行なった際の平均転送速度 9.41Mbps と比較すると21% も転送効率が悪化している．さらに，マルチホーム環境の片方の経路である，*path1(8Mbps, 60ms)* のみを用いたシングルパス転送を行なった際の平均転送速度 7.54Mbps よりもマルチパス転送を行なった際の平均転送速度のほうが下回り，マルチホーム環境をうまく利用しきれていない．このように帯域幅の差がある

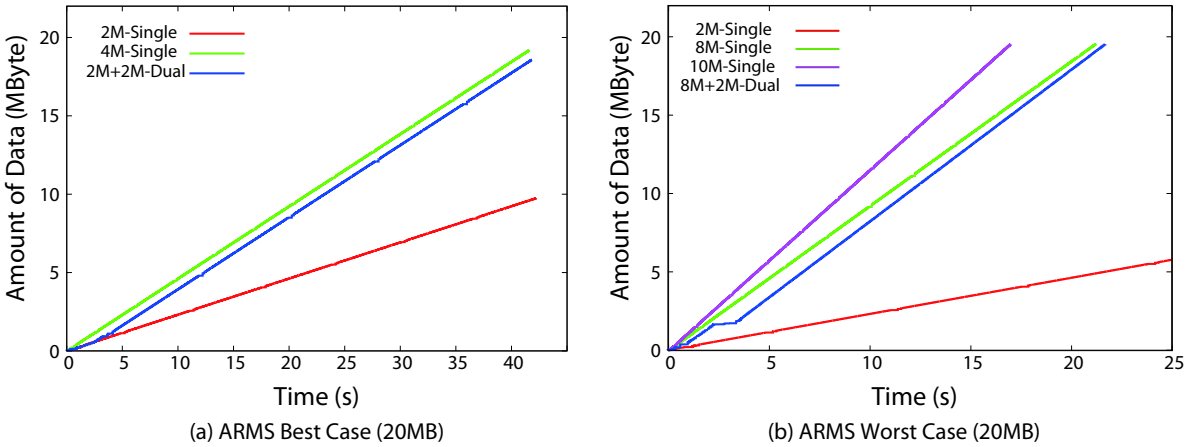


図 5.8　動的に計測した帯域比率に基づいたマルチパス転送を行った際の転送量

ときや遅延の大きさに大きな差があるときにマルチパス転送を行うのは難しい．

5.5　複数宛先への振り分けアルゴリズムの考察

本章では，事前実験により最適な複数宛先への振り分けアルゴリズムを構築する際にもっとも重要な要素は帯域幅であるということを示した．また，振り分けアルゴリズムを検討する際に RTT も重要な要素であることが上記の実験により示された．最適なデータ振り分けアルゴリズムを作成する際に必要である帯域幅や RTT の値から算出された経路ごとに SCTP が保持している輻輳制御用の数値である *cwnd* を用いたデータ振り分けアルゴリズムを実装・実験を行ってみたが，SCTP の設計として同時に 1 つのみを通信に使うという前提があるため，*cwnd* がマルチパス転送を行なった際に正常に機能しなかった．また，RTT も最適なマルチパス転送の振り分けアルゴリズムを検討する上で必要な要素であるが，SCTP の API からプログラム内にて取得した sRTT の値は，Dummynet にて設定した値と大きなさが見られた．sRTT とは一定時間の RTT の平均値であるため，高い精度で取得するためにはある程度の転送時間が必要となる．しかし，帯域測定を行っている際はシングルパス転送となってしまうため，より高速に帯域幅を計測する必要がある．よって本研究内においては RTT を考慮に入れずに計測した帯域幅のみを用いて複数の宛先へ送信データの振り分けを行う．

第 6 章

ARMS 評価

本章では，最初に複数の宛先への送信データの振り分けアルゴリズムに関して評価を行う．次に既存のマルチパス転送を実現する手法である CMT との性能比較を行い，最後に本手法について考察を行う．

6.1 複数宛先への振り分けアルゴリズムの評価

最初の評価として，複数の宛先への振り分けアルゴリズムについて評価を行う．評価実験環境は，前章で用いた図 5.1 と同じものを用いた．

実験は，Network 2, 3 の帯域幅と RTT を Dummynet にて変更しながら行なった．設定した値は，帯域幅が 2, 4, 8 Mbps, RTT が 30, 60ms とした．合計で 24 パターンについてそれぞれ 5 回づつ転送実験を行った．今回の転送実験では，データサイズにてアルゴリズムの性能が変更することを考慮して，20MByte の転送と 200MByte の転送の 2 種類を行なった．以降にて実験結果とその考察を論じる．

- ARMS を用いて 20MByte を転送した場合
  20MByte のデータを ARMS を用いて全ての組み合わせに対して転送した結果を表 5.8 に示した． 表 5.8 では各組み合わせによる 20MByte 分のデータ転送に要した時間［秒］を表している．全ての結果の中で最も転送速度が向上した場合 (*path1(2Mbps, 30ms), path2(2Mbps, 30ms)*) と，最も転送効率が悪かった場合 (*path1(8Mbps, 60ms), path2(2Mbps, 30ms)*) の経過時間あたりの転送量を示したグラフが図 6.1 である．最も転送効率が向上した場合においては，複数宛先への振り分けアルゴリズムが正確に設定した帯域比率通り動作した．逆に，最も転送効率が悪かった場合では，振り分けアルゴリズムが正確に動作していれば *path1* へは *path2* の 4 倍のデータ量が転送されるはずである．しかし，実験を行なった際に *path1* へ実際に転送された量は，*path2* の 3.2 倍であり，*path1* の帯域を埋め切れていなかったことが転送効率が上がらなかった原因の 1 つであると考える．また，この実験では総転送量が 20MByte に設定してあり，比較的短い転送時間で送信が完了してしまう．このため総転送時間に対する，帯域幅の測定時間の割合が大きくなってしまうことも，転送効率が上がらない要因の 1 つとなったと考えられる．

- ARMS を用いて 200MByte を転送した場合
  総転送時間に対する帯域幅の測定時間の割合を小さくするために，データ転送量を先程の実験の 10 倍である，200MByte に設定して実験を行った．今回のプロトタイプ実装では，50MByte のデータを送信する度に帯域幅を計測しなおすように実装した．よってこの実験では，0, 50, 100, 150MByte の合計 4 回帯域幅を計測し宛先ごとの転送比率を更新した．実際の実験ネットワークの設定は転送中に変更は行わなかった．Dummynet によって設定された，帯域と RTT のそれぞれの組み合わせにおける結果を表 6.2 に示した．表 6.2 中の数値はそれぞれの組み合わせに応じた総転送時間

*path*1 / *path*2		BW: 2Mbps		BW: 4Mbps		BW: 8Mbps	
		rtt : 30ms	rtt : 60ms	rtt : 30ms	rtt : 60ms	rtt : 30ms	rtt : 60ms
none		84.64	84.66	42.35	42.35	21.23	21.28
BW 2Mbps	rtt : 30ms	44.07	43.98	30.67	29.97	18.65	21.65
	rtt : 60ms	43.94	44.21	30.60	29.67	18.58	20.31
BW 4Mbps	rtt : 30ms	—	—	22.50	23.11	15.58	18.78
	rtt : 60ms	—	—	23.35	24.12	15.87	15.18
BW 8Mbps	rtt : 30ms	—	—	—	—	12.38	13.78
	rtt : 60ms	—	—	—	—	13.24	13.02

表 6.1　ARMS を用いた 20MByte の転送時間 (秒)

［秒］を表している．この実験において最も転送効率が向上した*path1(2Mbps, 30ms), path2(2Mbps, 30ms)* の組み合わせと，最も転送効率の向上が少なかった *path1(8Mbps, 60ms), path2(2Mbps, 30ms)* の組み合わせにて時間あたりの転送量を示したグラフを図 6.2 に示した．この実験の中で最も転送効率が向上した組み合わせが，*path1(2Mbps, 30ms), path2(2Mbps, 30ms)* であり，20MByte の転送実験を行った際と同様に，複数宛先へのデータ振り分けアルゴリズムが設定した帯域幅の比率通りに機能した．また，他の組み合わせよりも総転送時間が長いため，帯域測定に要した時間が与えた転送時間に対する影響が相対的に小さくなったことも転送効率が向上に付与したと考えられる．具体的に帯域測定時の転送速度は図 6.3 に示される．転送開始から約 3 秒ほどは 2Mbps のシングルパス転送と同じ転送量であるが，転送開始から 3 秒ほどが経過した後は，4Mbps のシングルパス転送とほぼ並行に転送量が増加していることがわかる．よってこの場合は最初の約3 秒間の間にできた 4Mbps のシングルパスとの転送データ量分ロスしていることになる．一方で，最も転送効率の向上が少なかった場合においては，正しい転送比率は 4 倍であるにも関わらず，実際に計測された比率は順に 3.2倍, 4.7 倍, 4.1倍, 4.1倍という値になった．このため最初の 50MByte の転送中はほぼ 4Mbps のシングルパス転送と同じ転送速度であった．転送の後半ではほぼ理

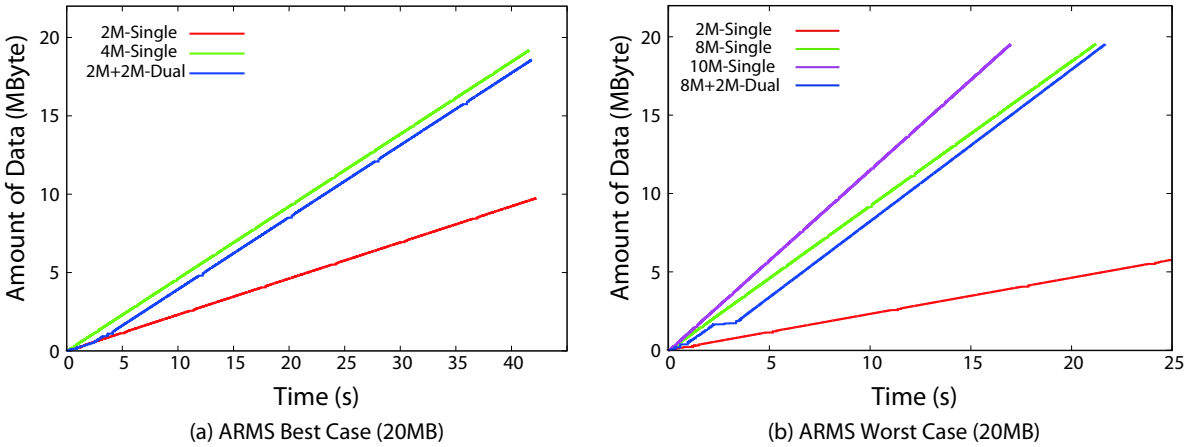


図 6.1　ARMS を用いて 20MByte をマルチパス転送した際の時間あたりの転送量

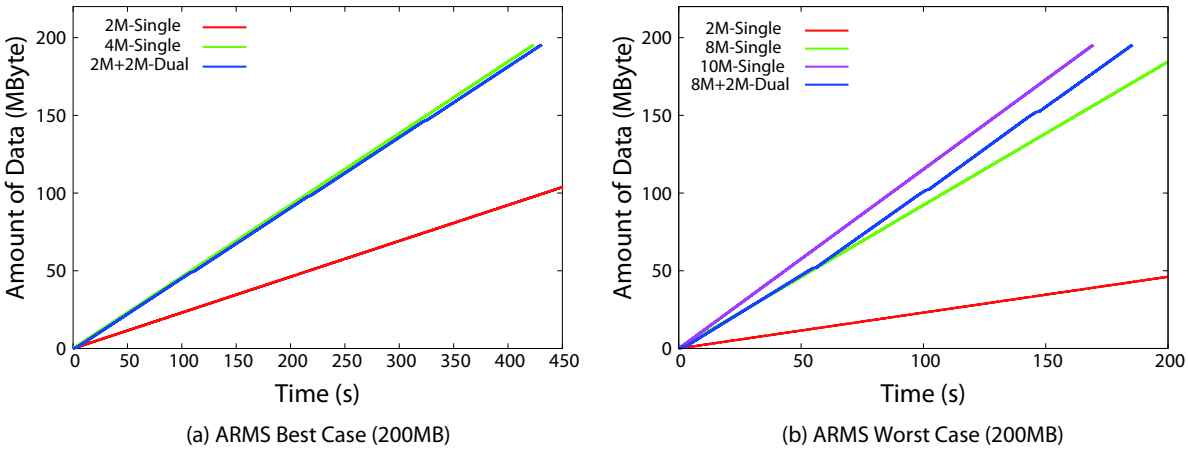


図 6.2　ARMS を用いて 200MByte をマルチパス転送した際の時間あたりの転送量

想的な送信データの振り分け比率に近づいたため，総転送量と総転送時間から求められる平均転送速度は，8Mbps のシングルパス転送より高速な転送速度を実現した．

*path*1		BW: 2Mbps		BW: 4Mbps		BW: 8Mbps	
*path*2		rtt : 30ms	rtt : 60ms	rtt : 30ms	rtt : 60ms	rtt : 30ms	rtt : 60ms
none		846.57	846.63	423.31	423.42	211.75	211.87
BW 2Mbps	rtt : 30ms	430.47	433.30	293.62	289.27	176.03	185.22
	rtt : 60ms	432.67	430.97	289.43	291.38	177.96	184.23
BW 4Mbps	rtt : 30ms	—	—	217.29	224.39	149.32	157.43
	rtt : 60ms	—	—	224.04	232.25	154.79	157.92
BW 8Mbps	rtt : 30ms	—	—	—	—	113.72	121.73
	rtt : 60ms	—	—	—	—	125.64	127.98

表 6.2　ARMS を用いた 200MByte の転送時間 (秒)

6.2　既存研究との性能比較

既存のマルチパス転送を実現する手法の中から，SCTP を拡張して作られたCMT を用いて性能比較実験とその評価を行う．実験環境は引き続き図 5.1 を用いて上述した複数宛先への振り分けアルゴリズムの評価と同じネットワーク環境を Dummynet で作成し，転送を行った．この実験結果を表 6.3 に示した．表 6.3 は左 2 列が Dummynet にて設定されたマルチパスの経路情報を示し，左から 3 番目の列が 8Mbps にてシングルパス転送を行なった際の平均転送速度を示している．右 2 列は，それぞれ ARMS, CMT にて転送を行なった際の平均転送速度を示している．表 6.3 数値データの単位は［Mbps］である．表 6.3 中の数値が **ボールド**体にて書かれているものは，ARMS と CMT の平均転送速度に 1Mbps 以上の差がついたことを示す．この結果から，Mobile Node が接続されている 2 つの帯域幅が等しい時には既存のマルチパス転送手法である CMT も本研究で提案している ARMS も帯域を効率良く埋めていることがわかる．次にCMT の方が ARMS より高速にマルチパス転送を行った *path1(8Mbps, 60ms), path2(4Mbps, 60ms)* の場合では， ARMS の振り分けアルゴリズムの精度が悪かったことにより転送速度に大きな差が出た．最後に *path1(8Mbps, 60ms), path2(2Mbps, 30ms)* や *path1(8Mbps, 60ms), path2(4Mbps, 30ms)* の場

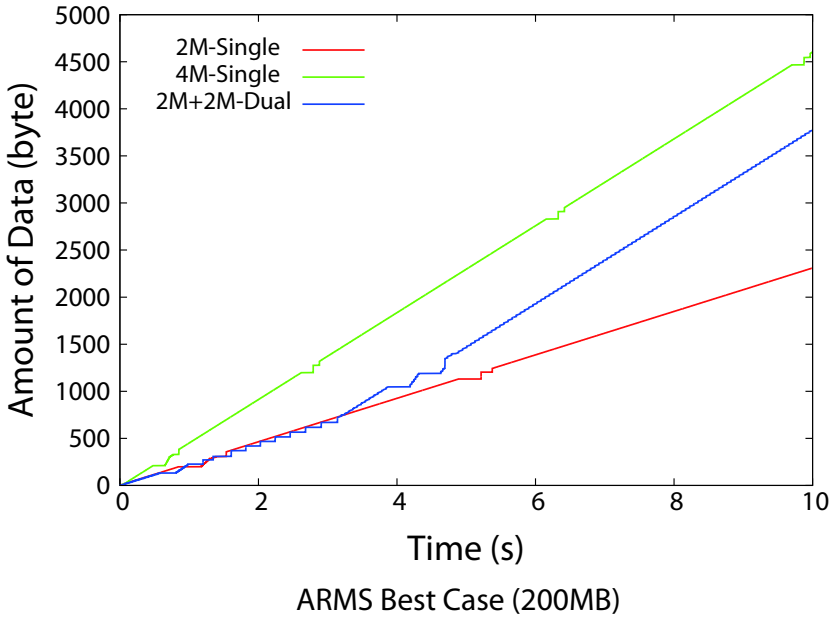


図 6.3　ARMS を用いた帯域幅計測中の時間あたりの転送量

合などでは ARMS はマルチホーム環境のネットワークリソースをうまく活用しきれていたが，CMT においてはシングルパス転送よりも転送速度が遅くなった．

以上より，CMT は大きな帯域の差があるマルチホーム環境においては帯域を効率的に活用できないのではないかと仮説を立て，次の追加実験を行った．Dummynet にて経路情報を *path1(10Mbps, 30ms), path2(2Mbps, 30ms)* に設定した環境にて，ARMS と CMT の両方について 200MByte のデータ転送を行なった．この際の時間あたりの転送量を示したグラフが図 6.4 である．このグラフで示されるように，CMT を使ったマルチパス転送はほぼ直線となり，また 10Mbps のシングルパス転送の転送量を常に下回っている．ARMS は転送中に 4 回帯域幅を測定するためにシングルパス転送となるため，定期的に転送量の傾きが穏やかになることがあるが，常に 10Mbps のシングルパス転送よりも多くのデータを転送していることが見て取れる．

path status (Bandwidth - RTT)		合計帯域幅による	ARMS	CMT
primary status	secondary status	シングルパス転送		
8Mbps - 30ms	2Mbps - 30ms	9.41	9.09	9.09
8Mbps - 60ms	2Mbps - 30ms	9.41	**8.65**	7.37
8Mbps - 30ms	2Mbps - 60ms	9.41	8.99	8.94
8Mbps - 30ms	4Mbps - 30ms	11.27	10.72	11.20
8Mbps - 60ms	4Mbps - 30ms	11.27	10.17	**11.20**
8Mbps - 30ms	4Mbps - 60ms	11.27	**10.31**	7.17
8Mbps - 30ms	8Mbps - 30ms	14.79	14.05	14.29
8Mbps - 60ms	8Mbps - 30ms	14.79	13.14	12.61
8Mbps - 30ms	8Mbps - 60ms	14.79	12.77	13.67

表 6.3　ARMS と CMT の転送速度比較表 (Mbps)

6.3　考察

本研究手法を用いたマルチパス転送では，ワーストケースを除きすべての場合において単一ネットワークを用いた転送よりも高速に転送を完了した．しかし，本手法では転送データが少ない場合や転送データ量に対して保持しているネットワーク帯域が十分に大きい場合などにおいては，総転送時間に対する帯域計測時間の割合が大きくなってしまうため，複数のネットワークリソースを有効に利用しきれないというデメリットも存在する．これは本手法に置いて帯域幅を計測する際に，保持する全てのアドレスに対してシングルパス転送を行うため 1 つの宛先以外には実際の転送とは関係の無いデータを送信していることに起因している．また，本論文における評価用ダウンロードプログラムでは Dummynet にて設定した帯域幅や遅延の値において効果を見ながら帯域測定時間を決定した．しかし実際のネットワークでは動的にネットワークの状態は変化をし続けるため，帯域幅，遅延，パケットロスなどを考慮した帯域計測時間を決定する必要がある．

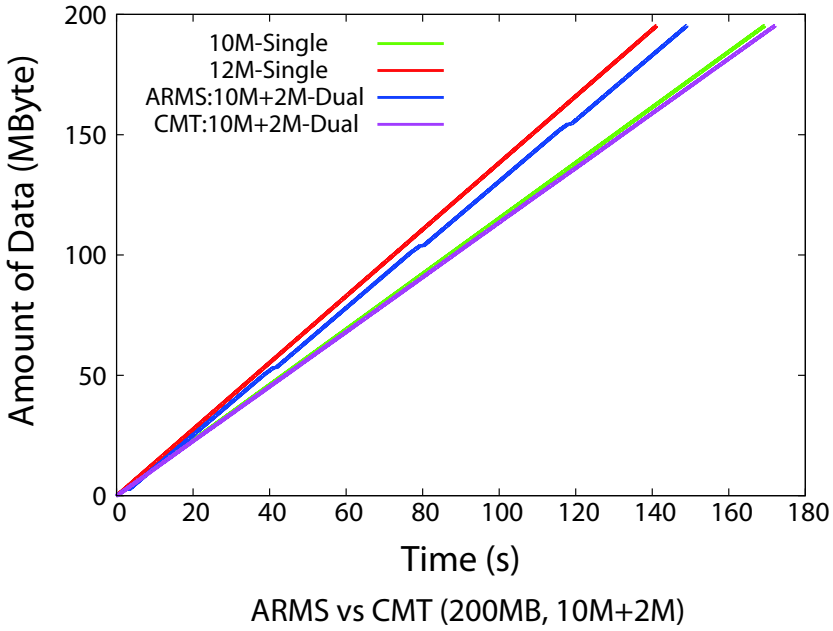


図 6.4　CMT と ARMS 転送速度の比較

現在マルチパス転送において最も標準化に近いとされている，CMT との性能比較も行い，本手法の有用性を実証できた．一般的に複数のネットワークを用いた通信を行う際には帯域幅の差が大きくなるほど，マルチパス転送の効果が薄れることが知られているが，帯域比が 5 倍異なる環境下において，既存研究の CMT はシングルパス転送よりも転送時間が長くなったが，ARMS を用いたマルチパス転送では帯域系素地時間を除けばほぼ理想的にネットワークリソースを利用できたと言える．

また，定性的な評価として，本手法のプラットフォームの移植性と実インターネットにおける懸念を述べる．まずプラットフォームの移植性であるが，実験を行った FreeBSD 以外にも Linux, Mac OS にて本手法を用いたプログラムを正常に動作させることに成功した．Windows にもサードパーティの SCTP スタック実装が存在するので，今後 Windows 上での移植性についても評価を行う予定である．次に実インターネットにおける懸念として，ISP などが設けている Firewall などを通過可能かどうかの懸念が存在する．本研究ではモバイル通信を主眼においていたため日本国内におけるモバイルネットワーク 4 社にて SCTP を用いたデータの送受信が可能かを検証した．検証したキャリアは， NTT Docomo, b-mobile, Emobile, UQ WiMAX である．これら全てにおいてデータの送受信を問題なく行うことができた．よって実際のモバイルネットワークにおいても本手法は利用可能であることが示された．

第 7 章

まとめ

本章では，これまで述べてきた議論をまとめ，本研究の今後の展望と，本論文の結論を述べる．

7.1 まとめ

本研究では複数のネットワークインタフェースを持つ端末においてネットワーク資源を効率的に利用するため，アプリケーション内において複数の宛先を同時に用いた信頼性のある通信を実現する手法，ARMS を提案した．またプロトタイプを実装し，評価実験を行った．現在様々な複数パスを同時に利用する手法が提案されているにも関わらずいずれの手法も標準化にはいたっておらず，日常的にマルチパス転送を行なっている人はほとんどいないのが現状である．ARMS はアプリケーション層にて高速なマルチパス転送を実現するため，マルチホーム環境を持つデバイスに搭載されているオペレーティングシステムに依存せずに利用できるため，容易に普及できると考える．

また，ARMS はトランスポートプロトコルにマルチホーム環境を標準でサポートしている SCTP を用いることにより，異なるパスを流れる複数のフロー間でのデータの同期をアプリケーション内で行う必要がない．また，データを送る宛先アドレスをアプリケーション内部にて明示的に指定することも可能であるなど，豊富な API が用意されているため，他のトランスポートプロトコルよりも SCTP を用いたほうが実現可用性が高いと言える．

しかし，標準の SCTP の仕様では，複数の宛先を保持していてもそれらを通信の冗長化にしか利用しない．しかし ARMS では複数の宛先を同時に利用する必要があるため，SCTP 内に用意されている API である *sctp_sendmsg()* 関数を用いることで複数の宛先へデータ送信することを実現した．データを送信する際の振り分けアルゴリズムは，様々な実験を行った結果，複数のアソシエーションを作成して，それぞれシングルパス環境下において帯域を計測し，計測された比率に応じて単一のアソシエーションからマルチパス転送を行うように設計・実装した．

ARMS の評価として 2 つのパスを利用したデータ転送による実験を行った．実験結果としては，図 6.2 (a) にて示したとおり，最もマルチホーム環境を活用できていた場合には，ほぼ全ての帯域を埋めきることに成功した．しかし，図 6.1 (b) にて示されるように，帯域も遅延も異なる環境下において少量のデータを送信した際にはシングルパス転送よりも転送速度が低下するという結果となった．また，既存研究とも性能比較を行なった．比較対象として本研究に最も近いCMT を選択した． CMT とは，SCTP を拡張したトランスポート層においてマルチパス転送を実現した手法である．この CMT と ARMS を同じ環境下において実験を行った結果，帯域や遅延の差が少ない環境においては CMT の方がかなり高い精度で複数の帯域を埋めきることに成功していた．しかし，帯域や遅延に大きな差があるときは逆に，ARMS のほうが多くの帯域を埋めることに成功していた．よって，小さなデータ量を転送する場合や保持している複数の経路の性質が近い場合は CMT の方がより高速なマ

ルチパス転送を実現できる．しかし，大きなサイズのデータ転送や，帯域ごとに性質が大きく異なるマルチホーム環境下においては ARMS の方が高速なマルチパス転送を実現した．

7.2 今後の展望

今回のプロトタイプ実装では，Mobile Node は 2 つのアドレスを保持した環境下でのみ動作する仕様となっていた．そのため今後の展望として，2 つ以上のいくつでも対応できるように改良を加え，評価実験を行う．また，今回行った実験においてはベストケースにおける評価実験のみを行っていたため，パケットロスが発生する環境や本実験以外のパケットが流れているネットワーク環境を用いて評価実験を行う．そして，本研究の目的でもある小型移動体端末において移動しながらの評価実験を行う．この実験では移動による IP アドレスの増減，変更が発生する．それらのイベントが発生した際に本研究においていかに転送速度を維持するかを議論し，より高速なマルチパス転送を実現する．

7.3 結論

本研究では，トランスポートプロトコルに SCTP を用い，アプリケーション層にて利用可能な複数の帯域を効率的に使用する事で，高速なマルチパス転送を実現する An Application-level Multipath Transfer Mechanism for Fast and Reliable Communication (ARMS) を提案した．ARMS を用いたマルチパス転送を行うことで，より高速な帯域を用いたシングルパス転送と比較して転送速度を最大で 1.97 倍まで増大させることを実現した．

謝辞

本研究の機会を与えてくださり，絶えず丁寧なご指導を賜りました，慶應義塾大学環境情報学部教授徳田英幸博士に深く感謝致します．また，貴重なご助言を頂きました慶應義塾大学政策・メディア研究科委員中村修博士や，慶應義塾大学政策・メディア研究科講師中澤仁博士に深く感謝致します．

また，慶應義塾大学徳田研究室 [1] のファカルティの方々や，諸先輩方には折に触れ貴重なご助言を頂き，また多くの議論の時間を割いて頂きました．特に政策メディア研究科修士課程本多倫夫氏には，本論文の執筆にあたってご指導を頂きました．

また，6年間という本当に長い間安心して学生でいられるよう支え続けてくれた，野澤東氏，野澤弥生氏，また学部時代の 4 年間共に勉学を共にし，常に私の人生における道標となってくれた野澤満恵氏，いつも遠くから応援し続くてくれた祖父母，家族全員に深く深く感謝いたします．

私が大学に入学してから 6 年間もの時間を過ごした，籠球倶楽部Kagers の仲間である，斉藤信登氏や水野千賀子氏等を始めとする同輩たちや松田裕徳氏を始めとする先輩方，また藤岡良子氏など多くの後輩達の支えにより充実した私生活を過ごせた事に感謝致します．

大学院に進学する前後から，早稲田大学の素晴らしいサークル Walkin’ を創ろうと誘ってくれた高校時代の同級生であり親友である吉澤浩太氏に感謝致します．また Walkin’ というサークル活動を通じて大北俊氏や中島基樹氏，高木雄一郎氏，桑野聡氏，脇田あづさ氏などの非常に個性あふれる，ここでしか出会うことのできない仲間に巡り会えたことによる学生生活の充実に，彼らと Walkin’ のメンバー全員に感謝致します．

同じ研究会として長い時間を共に過ごし辛いことや嬉しいことを共有し，研究のみならずそれ以上に深く付き合った同輩である，中津川紘太氏，小川正幹氏，金澤貴俊氏，徳田義幸氏，山本純平氏，研究会とサークル活動の両方で同じ時を過ごした荒木貴好氏，研究の日々を共に過ごした move! 研究グループ [2]の偉大なる先輩方や，兄弟同然に親しくしてくれた後輩である米川賢治氏を始め，多くの後輩達に感謝の意を表し，特に徳田研究会に所属してから今日に至るまでの 5 年間お世話になった本多倫夫氏には多大なる感謝と尊敬の意を表し，謝辞と致します．

2011年2月14日

野澤 高弘

参考文献

[1] Hide Tokuda Lab. http://ht.sfc.keio.ac.jp/.

[2] move! Project. http://www.ht.sfc.keio.ac.jp/move/.

[3] Ls-sctp: a bandwidth aggregation technique for stream control transmission protocol. *Computer Communications*, Vol. 27, pp. 1012–1024, June. 2004.

[4] A. Argyriou and V. Madisetti. Bandwidth aggregation with sctp. *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, Vol. 7, pp. 3716–3721 vol.7, Dec. 2003.

[5] T. Hacker, B. Athey, and B. Noble. The end-to-end performance effects of parallel tcp sockets on a lossy wide-area network, 2002.

[6] Huaizhong Han, Srinivas Shakkottai, C. V. Hollot, R. Srikant, and Don Towsley. Multi-path tcp: a joint congestion control and routing scheme to exploit path diversity in the internet. *IEEE/ACM Trans. Netw.*, Vol. 14, No. 6, pp. 1260–1271, 2006.

[7] Hiroshi Sakakibara and Masato Saito and Hideyuki Tokuda. Design and implementation of a socket-level bandwidth aggregation mechanism for wireless networks. In *WICON '06: Proceedings of the 2nd annual international conference on Wireless internet*, p. 11, New York, NY, USA, 2006. ACM.

[8] Hung-Yun Hsieh and Raghupathy Sivakumar. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, pp. 83–94, 2002.

[9] Janardhan R. Iyengar, Paul D. Amer, and Randall Stewart. Concurrent multipath transfer using sctp multihoming over independent end-to-end paths. *IEEE/ACM Trans. Netw.*,

Vol. 14, No. 5, pp. 951–964, 2006.

[10] Frank Kelly and Thomas Voice. Stability of end-to-end algorithms for joint routing and rate control. *SIGCOMM Comput. Commun. Rev.*, Vol. 35, No. 2, pp. 5–12, 2005.

[11] Kevin Lai and Mary Baker. Nettimer: A tool for measuring bottleneck link bandwidth. In *In Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, pp. 123–134, 2001.

[12] LuizMagalhaes and RobinKravets. Transport level mechanisms for bandwidth aggregation on mobile hosts. In *ICNP '01: Proceedings of the Ninth International Conference on Network Protocols*, p. 165, Washington, DC, USA, 2001. IEEE Computer Society.

[13] M. Allman, V. Paxson and W. Stevens. TCP Congestion Control. *RFC 2581*, Oct. 1999.

[14] Junwen Lai Ming Zhang and et al. A transport layer approach for improving end-to-end performance and robustness using redundant paths. *USENIX 2004 Annual Technical Conference*, pp. 99–112, 2004.

[15] R. Stewart. Stream Control Transmission Protocol. *RFC 4960*, Sep. 2007.

[16] R. Stewart, Q. Xie and et al. Sockets API Extensions for Stream Control Transmission Protocol (SCTP). *Internet Draft*, Jul. 2008.

[17] R. Stewart, Q. Xie, M. Tuexen, S. Maruyama, M. Kozuka. Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration. *RFC 5061*, Sep. 2007.

[18] Shunsuke Saito, Yasuyuki Tanaka, Mitsunobu Kunishi, Yoshifumi Nishida and Fumio Teraoka. AMS: An Adaptive TCP Bandwidth Aggregation Mechanism for Multi-homed Mobile Hosts. *IEICE Transactions on Information and Systems*, Vol. 89, pp. 2838–2847, 2006.

[19] H. Sivakumar, S. Bailey, and R. L. Grossman. Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks. In *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, p. 38, Washington, DC, USA, 2000. IEEE Computer Society.