卒業論文 2008年度(平成20年度)

ARMS:アプリケーション層における複数パスの同時利用

ARMS:Application-level Concurrent Multipath Utilization on Reliable Communication

指導教員

慶應義塾大学環境情報学部

徳田　英幸

村井　純

楠本　博之

中村　修

高汐　一紀

重近　範行

Rodney D. Van Meter III

植原　啓介

三次　仁

中澤　仁

慶應義塾大学 環境情報学部

野澤 高弘

takahiro@ht.sfc.keio.ac.jp

卒業論文要旨 2008年度(平成20年度)

ARMS:アプリケーション層における複数パスの同時利用

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通信効率を向上させる手法，ARMS (Application-level Concurrent Multipath Utilization on 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にする．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 2.0 倍の帯域向上を実現した．

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

慶應義塾大学 環境情報学部
野澤 高弘

Abstract of Bachelor's Thesis

ARMS:Application-level Concurrent Multipath Utilization on 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 Application-level Concurrent Multipath Utilization on 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.

Keyword :
application, congestion window, bandwidth aggregation, multi-homed mobile node, SCTP

Takahiro Nozawa

Faculty of Environmental Information
Keio University

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 [2] のような無線 LAN (Local Area Network) 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 [7] のような無線 MAN (Metro Area Network) や，HSDPA (High Speed Packet Access) [17] などの無線 WAN (Wide Area Network) などによるより広範囲にて利用可能なインターネットサービスも提供され始めている．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フェースを搭載したモバイル端末が登場している．複数のネットワークインタフェースを搭載した端末は，複数の無線接続が可能な場所において複数の通信経路を利用可能であり，マルチホームホスト [22] と呼ばれる．図 1.1 にマルチホーム環境の例を示す．モバイルノードは最初，無線 WAN のみにアクセス可能だが，無線 LAN の通信範囲に移動することで 2 つのネットワークに同時にアクセスできる．

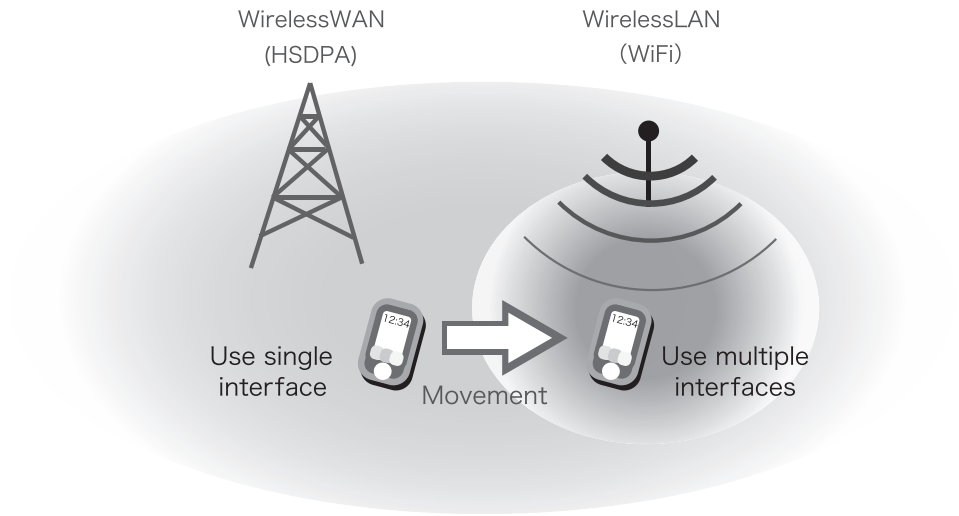

図 1.1　想定環境

また，図 1.2 よりインターネット利用者のうちモバイル端末を利用してインターネットに接続している人が約 81.2% であり，PC を利用してインターネットに接続している人の約 77.4% を上回った．以上より複数のネットワークインタフェースを搭載し高性能化したモバイル端末を用いて，広帯域を必要とするインターネットサービスの需要が高まっている．広帯域を必要とするサービス例として，HD 画質でのビデオチャットやビデオストリーミングサービス，大容量データの短時間でのダウンロードなどが挙げられる．しかし，現在のインターネットにおいて広く用いられているトランスポートプロトコルでは，複数のネッ

トワークインタフェースを同時に利用するという概念がなかったため，マルチホームのホストにおいても単一のネットワークインタフェースのみを利用した通信が一般的である．しかし，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，片方しか通信に用いないのは資源活用の観点から非効率的であり，必要なネットワーク資源の確保が困難となる可能性がある．

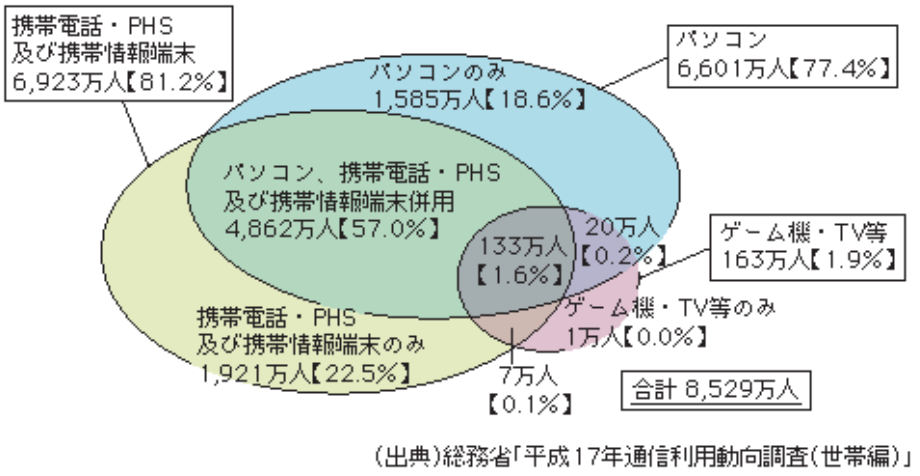

図 1.2　インターネットの利用方法

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，効率的な通信を実現する手法である ARMS (Application-level Concurrent Multipath Utilization on Reliable Communication) を提案する．ARMS の特徴として，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) において両方のネットワークを 1 つのソケット通信にて利用可能であるトランスポートプロトコル SCTP (Stream Control Transmission Protocol) [23] を用いることで，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．また，本研究では複数のキャリアから提供されたネットワーク接続を想定しているので，end-to-end の通信において実現する．この end-to-end の通信における送信側アプリケーションのみの実装で実現可能なことから，段階的に普及できる．

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考察する．3 章では ARMS の設計について記述する．4 章では ARMS 実装について触れ，5 章では評価実験結果と考察について述べる．6 章で本論文をまとめ，今後の展望について論じる．

第 2 章

関連研究

本章では，既存のマルチパス転送の利点と欠点について考察する．

2.1 関連研究

アプリケーション層にて複数のソケットを用いて利用可能な帯域を増大させる手法である PSocket [27]や，Parallel TCP Socket [8] などが提案されている．また，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，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 つ目がソケットを改良することによって実現した SBAM [10] である．2 つ目が TCP を改良して実現している手法の pTCP [11]，mTCP [21]，cTCP [5]，AMS [26]，Multi-path TCP [9]，Stability of end-to-end algorithms [16] などが挙げられる．3 つ目に SCTP を改良して実現している手法であるCMT [12] や BA-SCTP [4], LS-SCTP [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 [19]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 (Parallel Socket) [27] 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，それらのセッションを統合することで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．また，複数のソケットを用いて通信をしているので，受信側アプリケーションにて，トランスポートプロトコルとは別に受信データの再構築が必要となる．

2.3 ソケット改良アプローチ

- SBAM

  SBAM (Socket-level Bandwidth Aggregation Mechanism) [10] 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 [18] 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る欠点が存在する．

2.4 TCP アプローチ

- pTCP
  pTCP (parallel TCP) [11] 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* 命令を出し利用不可能になったアドレスで作成していたTCP-v を削除する．同様に新しいアドレスを発見した際も動的に *open* 命令を出し，新たに TCP-v を作成する．以上より pTCP は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．

- cTCP

  cTCP (Concurrency Handling in TCP) [5] はトランスポートプロトコルの 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． cTCP では MPLB(Multi-Path Load Balancing) [15] に基づいて各パスへの転送レートを決定している．想定環境として，両エンドホストがマルチホームホストであり，通信経路がそれぞれ独立した経路を想定している．送信側ホストに *ACK processor* を実装し，これによりパケットロス情報をデータベース化し，パケットロスが発生していないパスの *cwnd* 値の低下を防ぐ．マルチパス転送を行う際に RTT の差が発生することで送信データの到着順が異なる．このとき通常の TCP 輻輳制御アルゴリズムを用いると，直ちに *ACK* を返し，無駄な再送や *cwnd* の低下が発生する．一方，cTCP を用いた通信では，送信先アドレス毎に *ACK* を管理しているため，RTT の差によって到着順が入れ替わり，無駄な再送や *cwnd* 値の低下を防いでいる．

- AMS

  AMS (Aggregate-bandwidth Multihoming Support) [26] 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．ペアリングとは，送信側アドレスと受信側アドレスを結びつける作業である．データ転送速度

を最大化する為には，帯域の大きさに応じた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が発生した場合は，約 5 秒ほど転送が中断された．以上より AMS では動的なアドレスの変更に適応できていない．

2.5 SCTP アプローチ

- CMT
  CMT (Condurrent Multipath Transfer Using SCTP Multihoming) [12] 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．これは SCTP の仕様によるもので，送信側ホストは *GAP ACK* を受信すると直ちに再送を試みる．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
     必ずデータの送信を行った宛先へ再送する．
  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 (Reliable Multiplexing Transport Protocol) [19] 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．

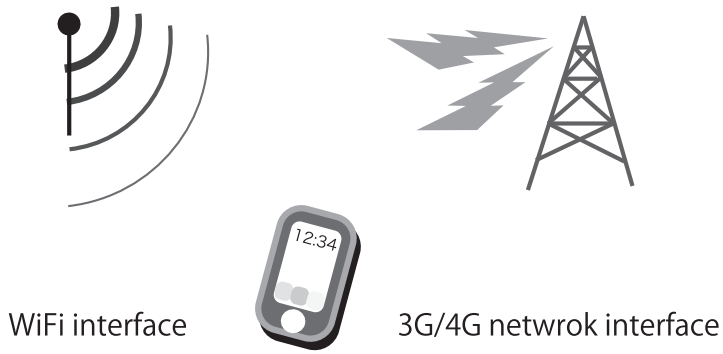

図 2.1　マルチパスの種類

2.7　まとめ

PSocket は，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が，単一アドレスでの通信を想定としているため，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 ネットワークなど複数のネットワークインタフェースを保有し，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 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 にて示された通りオペレーティングシステム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いる既存の研究はこれまでに存在していない．

	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

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

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マルチパス転送を実現する手法，Application-level Concurrent Multipath Utilization on Reliable Communication (ARMS) を提案する．

また，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end-to-end の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．さらにこれまでに提案された手法は，現在まで 1 つも標準化に至っていない．オペレーティングシステムの変更など実現コストが大きいためである．一方で，ARMS は送信側アプリケーションのみの変更で実現可能であるため，実現コストが少なく実現できる．これにより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

本節では，本論文を論じるにあたりマルチパス転送の定義を行う．送信側ノードと受信側ノードのアドレスの関係から，以下の 3 種類に分類できる．

1. 図 3.1 (a) 送信側ノード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，受信側ノードは単一のキャリアによって1 つのアドレスを提供され，それぞれ利用可能な全てのアドレスを通信に用いる場合

2. 図 3.1 (b) 受信側ノード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，送信側ノードは単一のキャリアによって1 つのアドレスを提供され，それぞれ利用可能な全てのアドレスを通信に用いる場合

3. 図 3.1 (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，それぞれ利用可能な全てのアドレスを通信に用いる場合

以上で様々な種類のマルチパス転送を列挙したが，本研究では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ているため，本研究におけるマルチパス転送の定義は，上記 2.(b) ，2.(c) の様な少なくとも受信側ノードは，複数のキャリアから複数のアドレスを割り振られている場合として以降論じる．

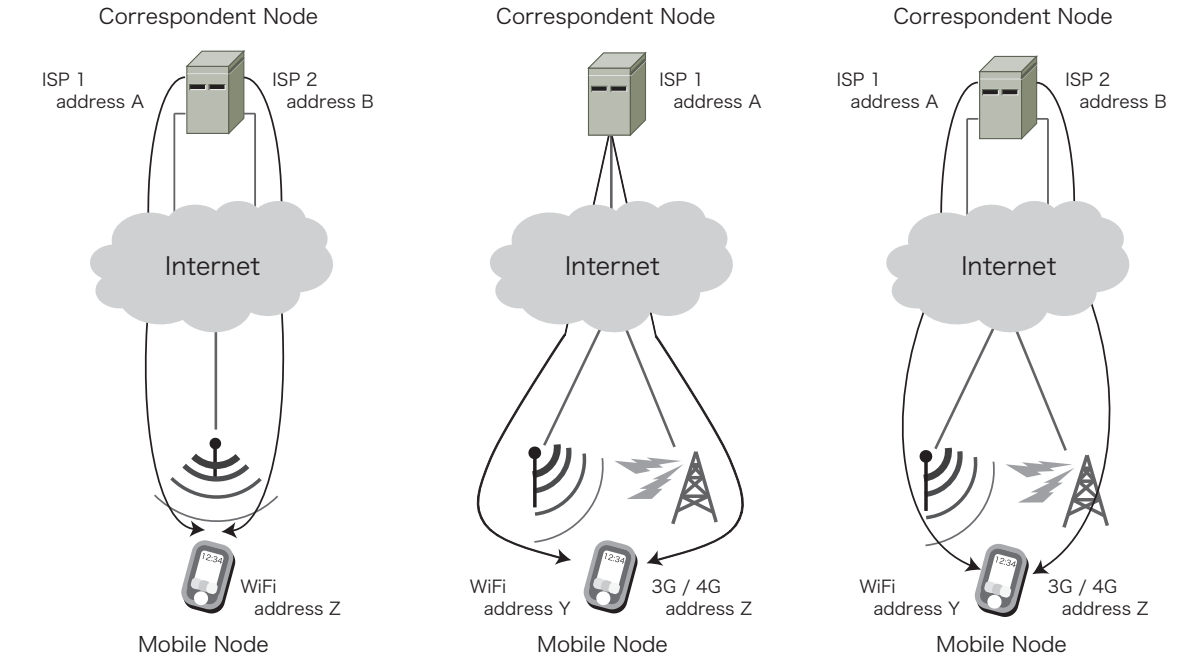

図 3.1　マルチパスの種類

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 のシステム概要を図 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する．以降ではまず，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．次に，複数宛先へのデータ送信方法について触れる．最後に，複数の宛先への最適な送信比率の決定方法を論じる．

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，

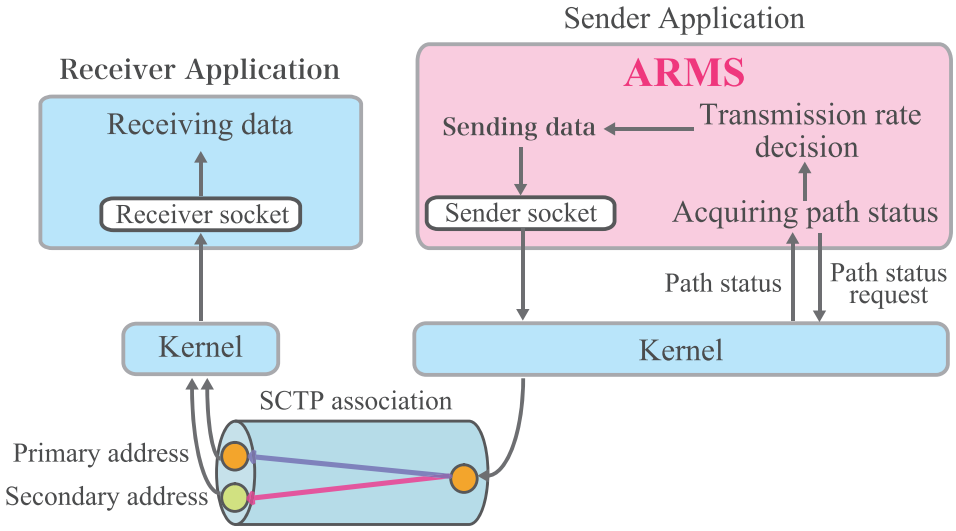

図 3.2　ARMS システム概要

TCP [14], UDP [13], SCTP, DCCP [6] 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ため，輻輳制御機能を持たない UDP や DCCP などのプロトコルで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 [24] 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) [25] 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．

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

	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

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，プライマリパスへの接続性が途切れた場合にのみ，セカンダリパスへデータが転送される．これは SCTP において複数のアドレスは通信の冗長化を目的に設計されているためである．設計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，利用可能な全てのネットワーク資源を利用しきれず，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．これにより利用可能なネットワーク資源を全て利用し，高速なデータ送信を行う．

3.3.3　転送比率決定

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比率で送信を行うと，

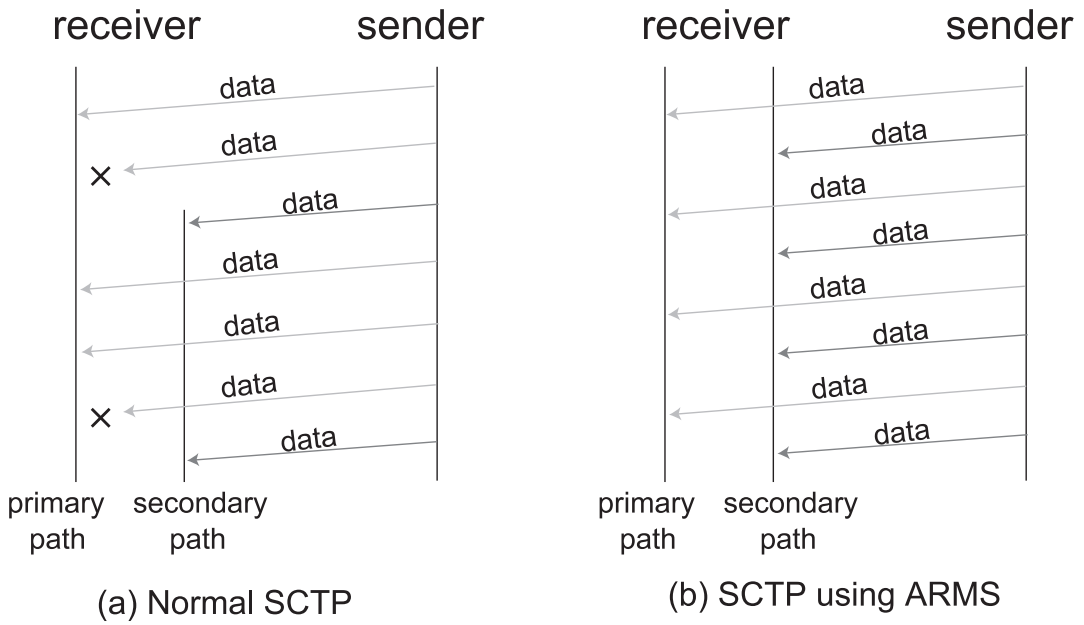

図 3.3　SCTP のマルチパス転送の比較

Link 1 (帯域の小さいリンク) の帯域を利用しきる事はできるが，Link 2 (帯域幅の大きいリンク) の帯域を利用しきる事ができず，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．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，各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)
  異なるネットワークを同時に用いた通信では，各パスの Round Trip Time (RTT) が異なると考えられる．図 3.4 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れるが，これらのパスは遅延が異なるため，帯域遅延積は 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 3.4 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 3.4 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．これは以下の数式により求められる．

$$(Link1RTT) - (Link2RTT)/2 \tag{3.1}$$

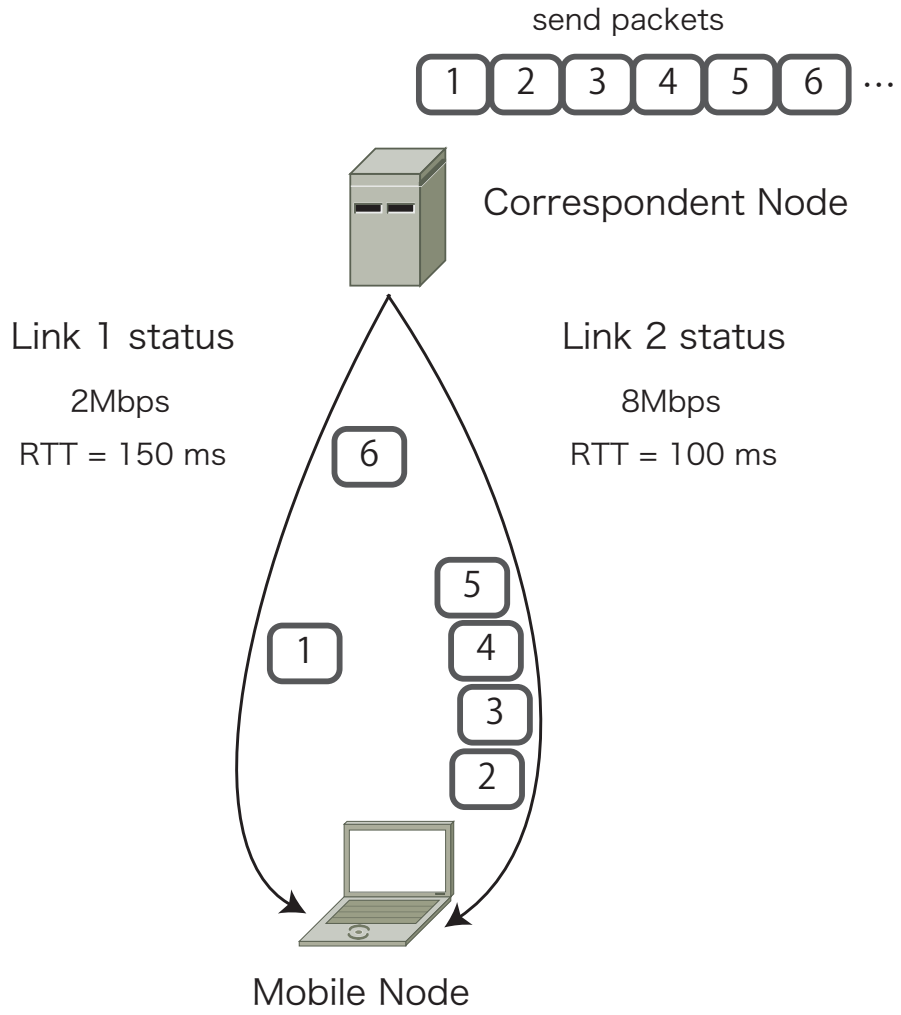

図 3.4　異なる性質のパスにおけるマルチパス転送

RTT は往復遅延時間のため，片道における遅延時間の計算は単純に半分にする．

$$(150 - 100)/2 = 25 \tag{3.2}$$

よって図 3.4 の場合のパケット受信の遅延は 25 ms となる．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．

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パラメータとして Congestion Window Size (*cwnd*) [20] 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，経路の帯域幅および RTT を反映している．理想値としては，帯域遅延積の値になる．具体的な *cwnd* 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次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 が行う処理を述べて行く．

4.1 ARMS 実装概要

ARMS は以下の 5 つの関数に分けられる．

1. *socket()* 関数によりソケットの作成
   SCTP をトランスポートプロトコルとし，*one-to-many* ソケットスタイルでソケットを作成する．

2. 通信に用いるアソシエーションの情報の取得
   get_assoc_info() 関数内で *getsockopt()* 関数を用いて，アソシエーション ID と *cwnd* を取得する．

3. 通信相手のアドレスの取得
   get_peer_addr() 関数内で *get_paddrs()* 関数を用いて通信相手の IP アドレスとその個数を取得する．

4. 状況に応じたデータ転送比率の決定
   get_assoc_info() 関数 と *get_peer_addr()* 関数によって取得したパス毎のパラメータを基に， *rate_decision()* 関数で最適なデータ転送比率を決定する．

5. 複数のアドレスへのデータ転送
   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．

これらの関数は図 4.1のように機能する．(1) *get_assoc_info()* 関数にてアソシエーション ID を取得する．(2) 取得した ID を用いて *get_peer_addr()* 関数にて通信相手の IP アドレスを取得する．(3) 再度 *get_assoc_info()* 関数を用いて，取得したアドレス毎の *cwnd* の値を取得する．(4) アドレス毎の *cwnd* から *rate_decision()* 関数を用いて最適な転送比率を算出する．(5) 算出された転送比率に基づいて *ARMS_send_data()* 関数にて指定した複数の宛先へデータを送信する．

通信が継続されていればアソシエーション ID は変わらないので，上記の (2) - (5) を繰り返し行う．

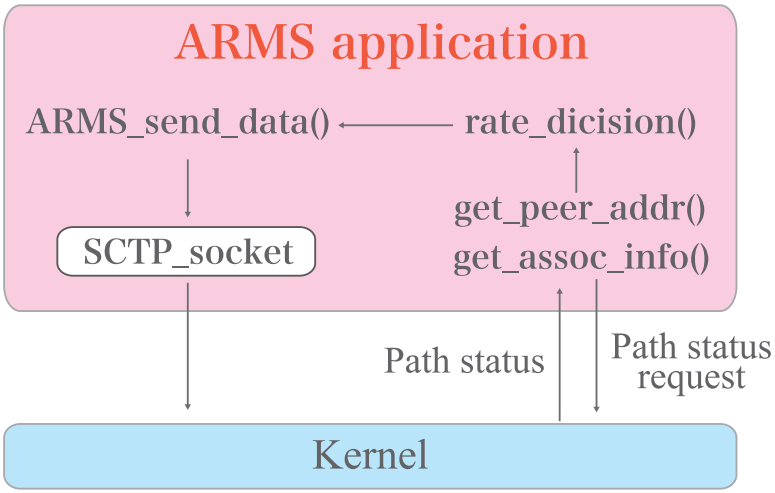

図 4.1　ARMS functions

上記 5 つの関数を以下において実際のソースコードを用いながら実装方法を述べる．

4.2　*socket()* 関数によりソケットの作成

SCTP では，ソケットを作成する際に 2 つのソケットスタイルを選択できる．選択するソケットスタイルの一方が，*one-to-one socket style* であり，他方が *one-to-many socket style* である．選択するソケットスタイルによって SCTP が提供する API の使用方法等が異なるため，用途により使い分ける必要がある．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 を用いた socket() 関数の使い方

```
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

転送比率の決定に必要なアドレス毎の *cwnd* の取得を，SCTP が提供する API を用いて行う．具体的にはソースコード 4.3 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 にて使用するパラメータは，ソースコード 4.2 に示す *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* と受信側ノードのアドレス毎の *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 4.3 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 4.3 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 0 でない整数値を指定することにより取得する．

ソースコード 4.2　sctp_paddrinfo() 構造体

```
1 struct sctp_paddrinfo {
2      sctp_assoc_t             spinfo_assoc_id;
3      struct sockaddr_storage  spinfo_address;
4      int32_t                  spinfo_state;
5      uint32_t                 spinfo_cwnd;
6      uint32_t                 spinfo_srtt;
7      uint32_t                 spinfo_rto;
8      uint32_t                 spinfo_mtu;
9 };
```

ソースコード 4.3　get_asoc_info() 関数

```
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      if(flag == 0){
25          return (sp.spinfo_assoc_id);
26      } else {
27          return (sp.spinfo_cwnd);
28      }
29  }
```

4.4 通信相手のアドレスを取得

通信相手が保持している有効なアドレスや，その個数の取得をアプリケーションから可能とするために，SCTP が提供する API である *sctp_getpaddrs()* 関数を用いた．*sctp_getpaddrs()* 関数の使用方法はソースコード 4.4 の 14 行目にあるように第 2 引数にはソースコード 4.3 にて取得したアソシエーション ID を指定し，第 3 引数に通信相手のアドレスリストを保存するための *sockaddr* 構造体を指定する．また，*sctp_getpaddrs()* 関数を実

行した際の返り値は，通信相手のアドレスの個数である．*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 4.4　get_peer_addr() 関数

```
1  #define REMOTE_ADDRESS_NUM 3
2
3  struct peer_addrs {
4    /* number of remote host addresses */
5    int              num_p_addrs;
6    /* IPv6 addresses held by remote host */
7    char             p_host[REMOTE_ADDRESS_NUM][INET6_ADDRSTRLEN];
8  };
9
10 struct          *peer_addrs
11 get_peer_addr(int sock, int assoc_id)
12 {
13   struct sockaddr *addrs, *addrp;
14   struct sockaddr_in6 *addrp6;
15   struct peer_addrs p_addrs;
16   int              paddr_num;
17   int              counter = 0;
18
19   /* sctp_getpaddrs function */
20   paddr_num = sctp_getpaddrs(sock, assoc_id, &addrs);
21
22   addrp = addrs;
23
24   if (addrp->sa_family == AF_INET6) {
25     for (; counter < paddr_num; counter++) {
26       addrp6 = (struct sockaddr_in6 *) addrp;
27       memset(p_addrs.p_host, 0,
28              sizeof(p_addrs.p_host[counter]));
29
30       if ((inet_ntop(AF_INET6, &addrp6->sin6_addr.s6_addr,
```

```
31                  p_addrs.p_host[counter],
32                  INET6_ADDRSTRLEN)) == NULL) {
33          perror("inet_ntop");
34          exit(1);
35        }
36        ++addrp6;
37        addrp = (struct sockaddr *) addrp6;
38      }
39    }
40    sctp_freepaddrs(addrs);
41    return &p_addrs;
42  }
```

4.5 動的なデータ転送比率の決定

今回の ARMS のプロトタイプ実装では，複数宛先に対し効率的なデータ転送を行うために，宛先毎の *cwnd* を取得し，動的にデータ転送比率を決定している．*cwnd* の取得はソースコード 4.3 で述べたように指定して取得したものを用いる．転送比率の決定は，ソースコード 4.5 に示した．具体的には，2 つの宛先毎の *cwnd* の大きい方を *l_cwnd*，小さい方を *s_cwnd* とすると，以下の計算で算出される．

$$sending\_rate = l\_cwnd / s\_cwnd \tag{4.1}$$

また，*sending_rate* が 1.2 未満である場合，少なくとも一方は帯域を使い切るという目的で，より大きい *cwnd* を保持する宛先に対して，小さい *cwnd* を保持する宛先よりも 1.2 倍多くデータを送信する事とした．

ソースコード 4.5　rate_decision() 関数

```
1  double
2  rate_decision(int prim_cwnd /* primary address's cwnd */ ,
3          int scnd_cwnd /* secondary address's cwnd */ )
4  {
5    int              s_cwnd;
6    int              l_cwnd;
```

```
7     double          sending_rate;
8     int             tmp;
9
10    if (prim_cwnd < scnd_cwnd) {
11      s_cwnd = prim_cwnd;
12      l_cwnd = scnd_cwnd;
13    } else {
14      s_cwnd = scnd_cwnd;
15      l_cwnd = prim_cwnd;
16    }
17
18    sending_rate = l_cwnd / s_cwnd;
19
20    if (sending_rate < 1.2) {
21      sending_rate = 1.2;
22    }
23    tmp = sending_rate * 10;
24    sending_rate = tmp / 10;
25    return sending_rate;
26  }
```

4.6 複数アドレスへのデータ転送とデータ受信

sending_rate_decision() 関数にて算出された値を元に，データを送信する．複数の宛先へデータ転送をするには *sctp_sendmsg()* 関数 [24] 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 4.6　ARMS_send_data() 関数

```
1  void
2  ARMS_send_data(int sock, int cwnd, double sending_rate,
3           struct sockaddr_in6 large, struct sockaddr_in6 small)
4  {
5    int               flag = 10;
6    int               tmp;
7    int               io_len;
8    char              buf[1408];
9
10   tmp = sending_rate * 10;
11
12   if (tmp % 5 == 0) {
13     flag = 2;
14   } else if (tmp % 2 == 0) {
15     flag = 5;
16   }
17   while (1) {
18     for (int num = 0; num < flag; num++) {
19       io_len = sctp_sendmsg(sd, buf, sizeof(buf),
20                             (struct sockaddr *) &small,
21                             (socklen_t) sizeof(small),
22                             0, SCTP_ADDR_OVER, 0, 0, 0);
23     }
24     for (int num = 0; num < sending_rate * flag; num++) {
25       io_len = sctp_sendmsg(sd, buf, sizeof(buf),
26                             (struct sockaddr *) &large,
27                             (socklen_t) sizeof(large),
28                             0, SCTP_ADDR_OVER, 0, 0, 0);
```

```
29          cwnd = cwnd - 1408;
30
31          if (cwnd < 0) {
32            break;
33          }
34        }
35      }
36  }
```

第 5 章

ARMS 評価

本章では，まず評価実験について述べ，その実験の結果を示す．最後に結果から導きだされた考察を述べる．

	Installed operating system	Number of network interfaces
Correspondent Node	FreeBSD 7.0R	1
Mobile Node	FreeBSD 7.0R	2
Dummynet	FreeBSD 6.2R	2

表 5.1　評価用ノード詳細

5.1　評価

本節では前章で述べた ARMS のプロトタイプ実装による評価について述べる．本論文における評価実験環境を図 5.1 に示す．評価用ホストの設定は表5.1 に示した通り，Correspondent Node (CN)，Mobile Node (MN) ともに OS は FreeBSD 7.0R を使用した．また, CN は1つのネットワークに有線にて接続され，MN も2つのネットワークに有線で接続している．また CN と MN が接続している3つのネットワークは全て異なるセグメントである．本プロトタイプ実装における実験では 2 つのパスを同時に利用し，各パスの帯域幅とRTT を変化させて実験を行った．Dummynet を用いて MN が接続する *Network 1* および *Network 2*の帯域と遅延を変化させる．また，CN が接続する *Network 3* は 100Mbps である．今回の実験では，帯域は 1Mbps または 2Mbps に設定し，RTT は 30ms または 60ms と設定した．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．これらにシングルパスまたは，マルチパスの組み合わせを含めた全14パターンにて，20MByte を転送するときに要した時間を計測した．全14パターンのネットワーク環境について，それぞれ5回づつ実験を行い，その結果を表 5.2 にまとめた．表 5.2 の縦軸 (*path1)* が図 5.1 の Network2 の設定を，表 5.2 の横軸 (*path2)* が の図 5.1 の Network 3 の有無または存在する場合の設定を示す．平均スループットは，データ転送量を総転送時間で割ることによって算出し，図 5.2 に示した．

5.2　評価実験結果

評価実験の結果を *cwnd* と RTT の 2 つのパラメータの値の違いにより以下の 4 つの場合に分類し，実験結果をまとめた．

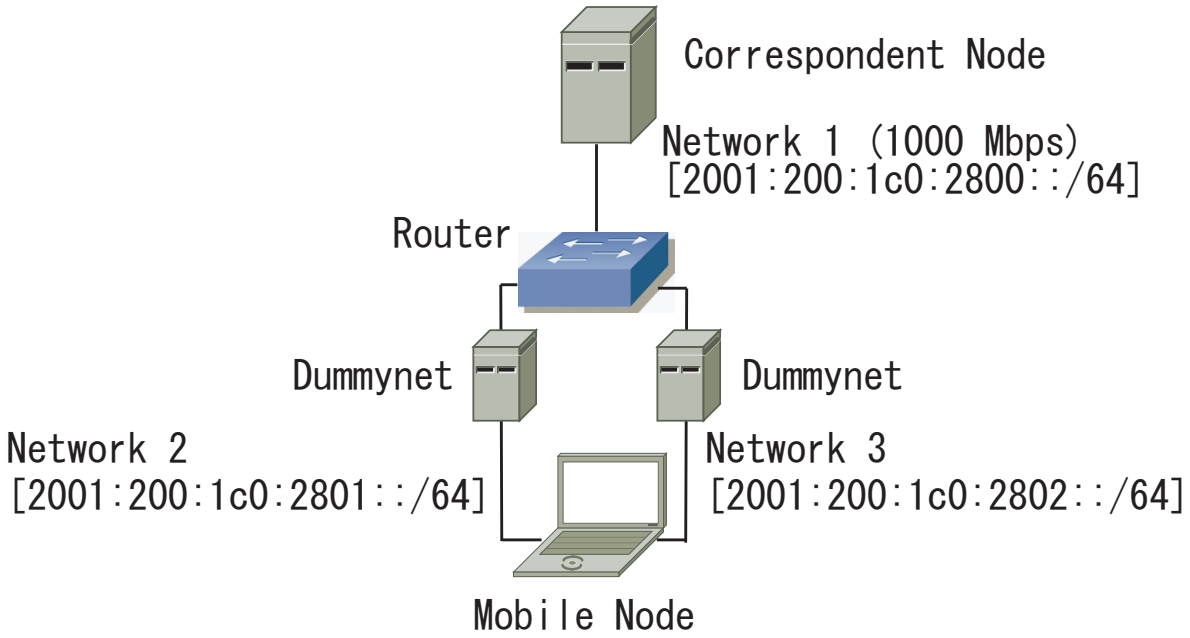

図 5.1　評価環境

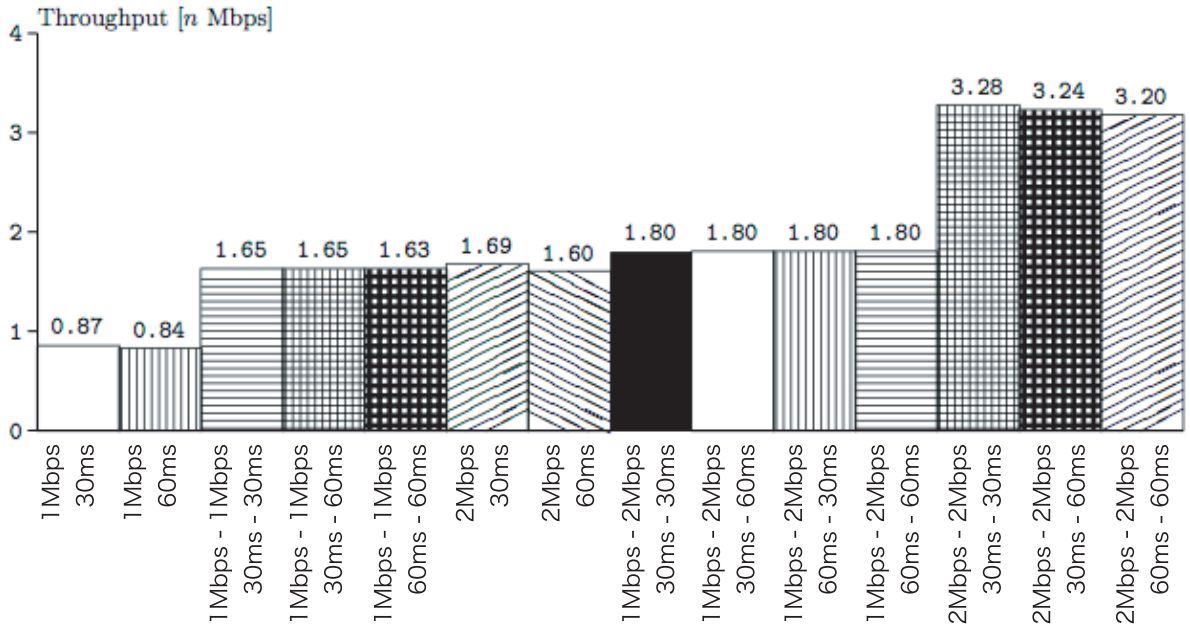

図 5.2　平均スループット比較

$path2$ / $path1$		none	1Mbps		2Mbps	
			rtt : 30ms	rtt : 60ms	rtt : 30ms	rtt : 60ms
1M bps	rtt : 30ms	175.78	92.51	—	—	—
	rtt : 60ms	180.74	92.61	93.82	—	—
2M bps	rtt : 30ms	90.48	84.85	84.78	46.45	—
	rtt : 60ms	95.26	84.81	84.79	47.15	47.61

表 5.2　各パスの組み合わせにおける 20MByte の転送時間 (秒)

5.2.1　同じ性質のパスを用いた場合

path(1Mbps, 30ms)，*path(1Mbps, 60ms)*，*path(2Mbps, 30ms)*，*path(2Mbps, 60ms)* を単独で用いた際のスループットはそれぞれ 0.87Mbps，0.84Mbps，1.69Mbps，1.60Mbps であった．一方で，ARMS によりそれぞれのパスを 2 本同時に利用した際のスループットは，*path(1Mbps, 30ms)* を 2 つ用いた場合が1.65Mbps，*path(1Mbps, 60ms)* を 2 つ用いた場合が 1.63 Mbps，*path(2Mbps, 30ms)* を 2 つ用いた場合が 3.28 Mbps，*path(2Mbps, 60ms)* を 2 つ用いた場合が 3.20 Mbps だった．

単一のパスを用いた場合と ARMS により複数パスを用いた場合のスループットの変化の割合は，*path(1Mbps, 30ms)* の場合が 1.90 倍，*path(1Mbps, 60ms)* の場合が 1.94 倍，*path(2Mbps, 30ms)* の場合が 1.94 倍，*path(2Mbps, 60ms)* の場合が 2.00 倍である．

5.2.2　帯域のみが異なるパスを用いた場合

path(1Mbps, 30ms) と *path(2Mbps, 30ms)* の組み合わせ，*path(1Mbps, 60ms)* と *path(2Mbps, 60ms)* の組み合わせを用いた場合のスループットは，それぞれ 1.80 Mbps, 1.80 Mbps であった．

単一パスを用いた場合のスループットは*path(2Mbps, 30ms)* の場合が 1.69 Mbps，*path(2Mbps, 60ms)* の場合が 1.60 Mbps であったことから，1 割ほどのスループット向上がみられた．

5.2.3 遅延のみが異なるパスを用いた場合

path(1Mbps, 30ms) と *path(1Mbps, 60ms)* の組み合わせ，*path(2Mbps, 30ms)* と *path(2Mbps, 60ms)* の組み合わせを用いた場合のスループットは，それぞれ 1.65Mbps, 3.24Mbps であった．

path(1Mbps, 30ms) のシングルパスと，*path(1Mbps, 30ms)* と *path(1Mbps, 60ms)* の組み合わせのマルチパスではスループットが約 1.90 倍の差となり，*path(2Mbps, 30ms)* のシングルパスと，*path(2Mbps, 30ms)* と *path(2Mbps, 30ms)* の組み合わせのマルチパスではスループットが約 1.92 倍の差となった．

5.2.4 帯域及び遅延が異なるパスを用いた場合

path(1Mbps, 30ms) と *path(2Mbps, 60ms)* の組み合わせ，*path(1Mbps, 60ms)* と *path(2Mbps, 30ms)* の組み合わせを用いた場合のスループットは，それぞれ 1.80Mbps, 1.80Mbps であった．

帯域のみが異なるパスを用いたときと同様に，単一パスを用いた場合のスループットは *path(2Mbps, 30ms)* の場合が 1.69 Mbps，*path(2Mbps, 60ms)* の場合が 1.60 Mbps であったことから，1 割ほどのスループット向上がみられた．

5.3 考察

以上の評価実験結果より今回の *cwnd* を用いて転送比率決定を行ったプロトタイプ実装では，帯域比が同じ場合には，シングルパス転送と比較してほぼ理想通りの転送速度の上昇を実現した．しかし，逆に帯域比が異なる場合にはシングルパスで計測した転送速度を約 1 割ほど改善するという結果にとどまった．この様な結果になった原因を追及する為に，今回のプロトタイプ実装において転送比率を決定したパラメータである *cwnd* の変化を転送速度の向上が見られた場合と，見られなかった場合にて比較し考察を行う．

図 5.3 は，シングルホーム環境にてデータ転送を行った際の *cwnd* の動きである．縦軸が *cwnd* (KByte) を示し，横軸がデータ転送開始時からの時間 (秒) を表す．TCP や SCTP に実装されている輻輳制御アルゴリズムは，一度に送信可能なデータ量，つまり *cwnd* をパケットロスが発生するまで少しづつ上昇させ，パケットロスが何らかの理由で発生した場合に *cwnd* を半減させる．逆に，このように周期的に *cwnd* が増減を繰り返していれば，そのパスは輻輳制御アルゴリズムのもとで効率的に帯域を利用していることを意味する．

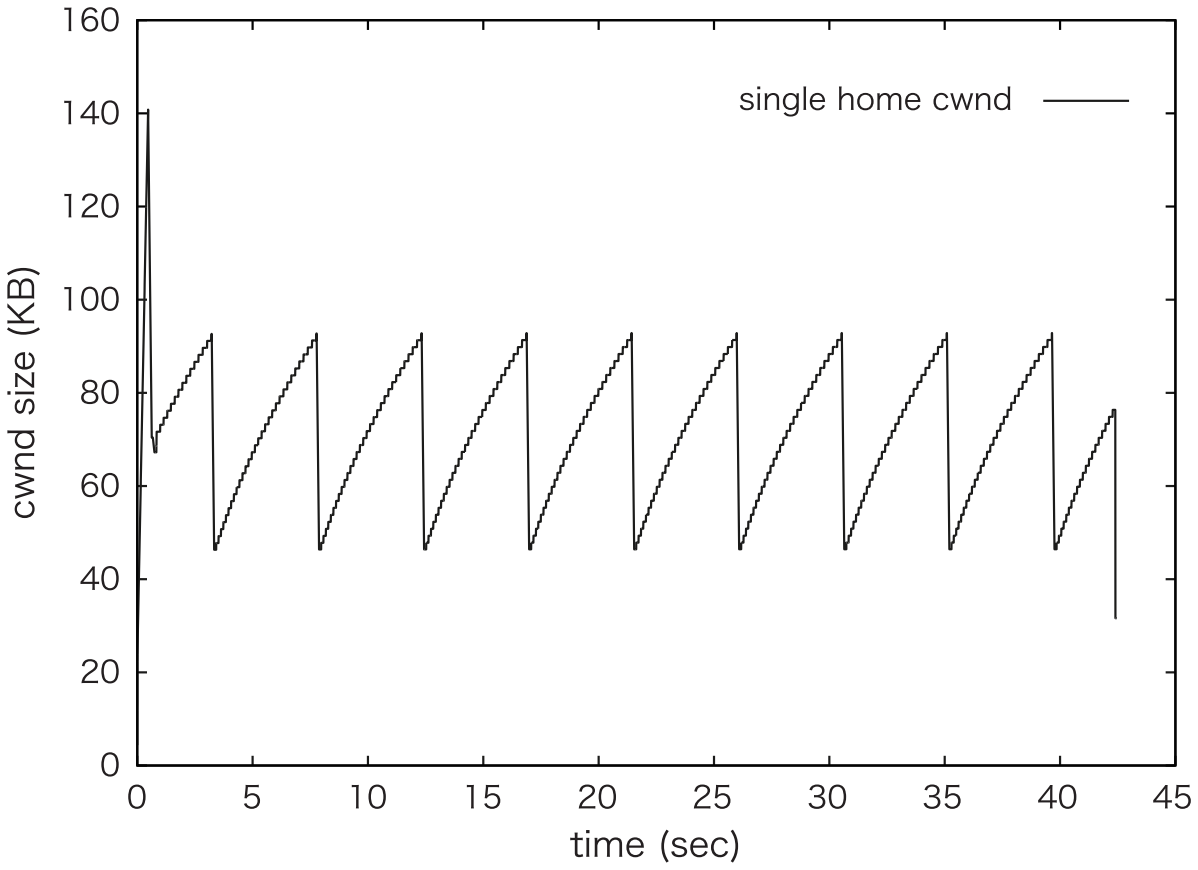

図 5.3　シングルパス転送における cwnd の動き

5.3.1　理想的な転送速度の向上を計測した場合

プロトタイプ実装による評価で理想通りの転送速度の向上を実現した，帯域幅が同じであった場合について考察する．このときの *cwnd* の変動を図 5.4 に示した．図 5.4 の縦軸は *cwnd* (KByte) を表し，横軸はデータ転送開始時からの経過時間 (秒) を表している．図 5.4 は図 5.3 と同じように，両方のパスとも帯域を埋めきることによりパケットロスが発生し，*cwnd* を半減させるのを繰り返している．結果としても理想通りの転送速度の上昇を計測しているので，この場合は効果的に全ての宛先を利用したマルチパス転送を実現したと考えられる．

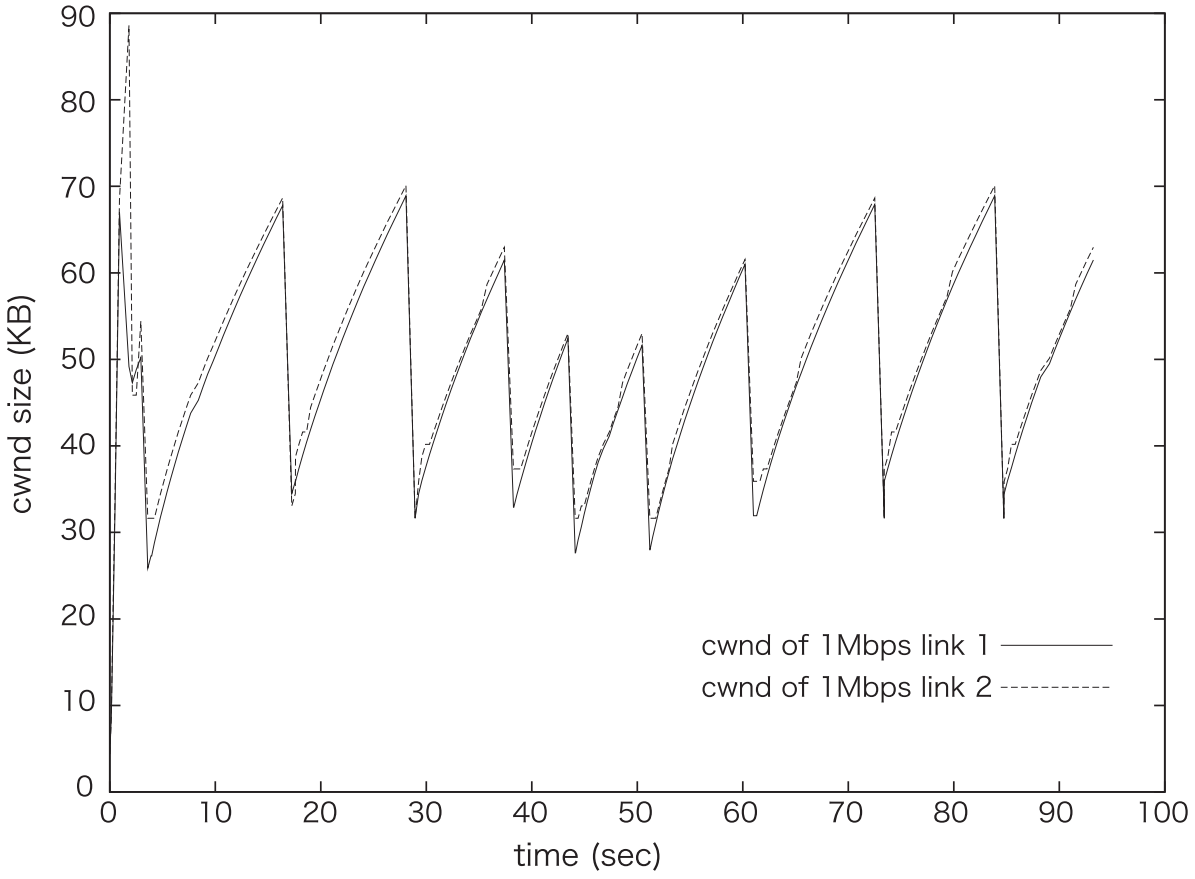

図 5.4　両方のパスの帯域が 1Mbps だった場合の cwnd の動き

5.3.2　理想的な転送速度の向上を計測できなかった場合

プロトタイプ実装による評価で理想通りの転送速度の向上を実現できなかった，帯域幅がパス毎に異なる場合について考察する．このときの *cwnd* の変動を図 5.5 に示した．図 5.5 の縦軸は *cwnd* (KByte) を表し，横軸はデータ転送開始時からの経過時間 (秒) を表している．図 5.5 からわかるように，この場合において *cwnd* はほぼ横ばいの一定値を最長で約 50 秒間も取り続けている．これは図 5.3 の様な増減を繰り返す挙動とは大きく異なる．また，一定の値を取り続けている期間に *cwnd* の値が減っていない事から，輻輳が 1 回も起こっていない事になるが，仮に輻輳が起こっていないのであれば SCTP の輻輳制御アルゴリズムでは *cwnd* の値が上がって行かなければならない．ここに *cwnd* の値の変動に関して矛盾が生じる．この矛盾を持った *cwnd* の値から転送比率を決定してしまった事が，今回理

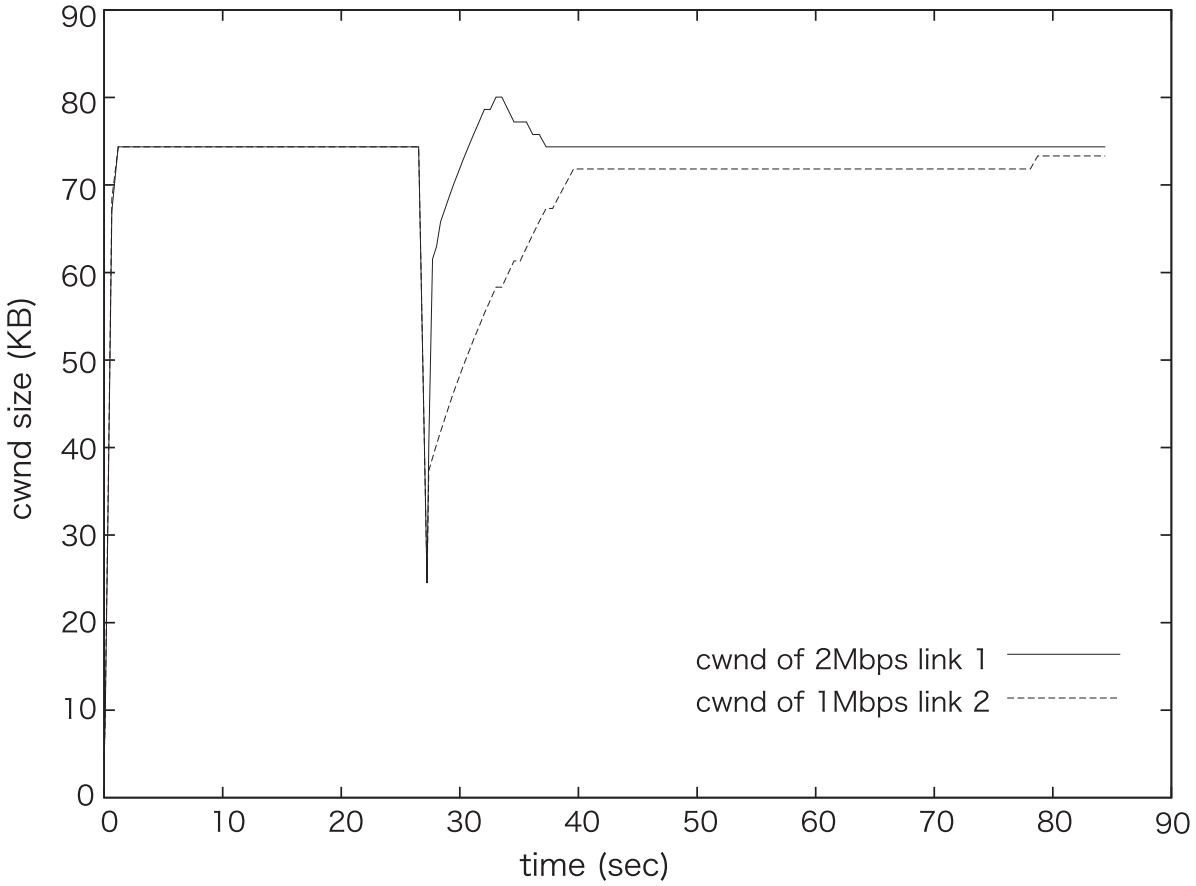

図 5.5　パスの帯域が 1Mbps, 2Mbps だった場合の cwnd の動き

想的な転送速度の向上を実現できなかった理由であると考える．

cwnd の値の矛盾はネットワーク帯域をエミュレートする際に用いたDummynet の設定によるものと考える．TCP にて Dummynet を用いた際に，本研究における結果と同様の症状が報告されている [1]．SCTP の輻輳制御アルゴリズムは TCP の輻輳制御アルゴリズムと類似点が多く存在するため，SCTP を用いた本研究においても TCP と同様に輻輳制御機能が働かなかったと考えられる．これ以降に評価実験を行う際，Dummynet の設定方法の検討や，Dummynet 以外でのネットワーク帯域のエミュレート方法も検討する必要がある．

第 6 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．

6.1 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のネットワーク資源を効率的に利用するため，アプリケーションレベルで複数パス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が，どれも実際には普及していない．しかし，ARMS はアプリケーションレベルで実現することによりプラットフォー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．

アプリケーションでの実現には，受信側アプリケーションにて受信データの再配置という欠点が存在する．しかし，ARMS ではトランスポートプロトコルにマルチホーム環境に対応している SCTP を用いることで解決する．また，効率的な複数パス同時利用に必要となるパス毎の情報は SCTP の API によって取得可能である．よって，TCP を利用した場合に比べ，SCTP を利用する事でアプリケーション開発者は，より容易にマルチパス転送を実現できる．

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用いない．ARMS では宛先を明示的に指定して送信し，同時に宛先毎の最適な転送比率を，宛先毎に SCTP が保持している *cwnd* というパラメータを利用する事にて利用可能な帯域を効率的に利用したマルチパス転送を実現した．

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．その結果，帯域幅が同じ環境下ではほぼ理想的な転送速度の向上を実現した．一方で帯域幅が異なる環境下では，シングルパス転送における転送速度と比較し約 1 割の転送速度の向上にとどまり，理想的な改善には及ばなかった．

6.2 今後の展望

今後の展望として，今回のプロトタイプ実装では最適な転送比率の決定に *cwnd* のみを用いたが，RTT など *cwnd* 以外の各パスの状態を表すパラメータを同時に利用する事で，より最適な転送効率の実現を目指す．このときアプリケーション層からでは情報取得が困難である場合は，SCTP の API の変更や追加を行う．

今回の評価実験では宛先を 2 つに固定して実験を行ったが，今後はより多くの宛先がアソシエーション内に登録されていた場合においても実験を行い，ARMS の宛先の数に対するスケーラビリティを検証する．また，通信の途中にて宛先アドレスの追加，減少，変更と言った動的な変化が起きたときにおいても評価を行い，宛先アドレスの変更や個数の増減

などの動的に変化する環境下においても転送速度の向上を実現させる．

6.3 結論

本研究では，トランスポートプロトコルに SCTP を用い，アプリケーションにて利用可能な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Application-level Concurrent Multipath Utilization on Reliable Communication (ARMS) を提案した．ARMS を用いたマルチパス転送とシングルパス転送とを比較すると，最大で 2.00 倍の転送速度を実現した．

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・メディア研究科准教授高汐一紀博士や，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，慶義塾大学徳田研究室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したことに深く感謝致します．特に政策・メディア研究科博士課程榊原寛氏には，本論文の執筆にあたってご指導いただきましたことを深く感謝致します．

また，私生活，研究生活の両面において支えてくれた両親を始めとする家族，特に同じ大学で学ぶ学友でもあり姉である野澤満恵氏に感謝致します．

大学入学以前から今に至るまでの長い近い付き合いになる福地玲衣氏や吉沢浩太氏を始めとする多くの暖かい仲間に感謝致します．

私が大学に入学してから 4 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，松永明日香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．

同じ研究会として長い時間を共に過ごし，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，島津，研究会とサークル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! 研究グループの偉大な先輩方や，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 3 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．

2009年2月10日

野澤 高弘

参考文献

[1] http://ino-www.jaist.ac.jp/members/isogaki/fbsd-dummynet.html.

[2] IEEE 802.11 WIRELESS LOCAL AREA NETWORKS. http://www.ieee802.org/11/.

[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] Yu Dong, Niki Pissinou, and Jian Wang. Concurrency handling in tcp. *Communication Networks and Services Research, Annual Conference on*, Vol. 0, pp. 255–262, 2007.

[6] E. Kohler. Datagram Congestion Control Protocol. *RFC 4340*, March 2006.

[7] C. Eklund, R.B. Marks, K.L. Stanwood, and S. Wang. Ieee standard 802.16: a technical overview of the wirelessmantm air interface for broadband wireless access. *Communications Magazine, IEEE*, Vol. 40, No. 6, pp. 98–107, Jun 2002.

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

[9] 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.

[10] 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.

[11] Hung-Yun Hsieh, Kyu-Han Kim, and Raghupathy Sivakumar. An end-to-end approach for transparent mobility across heterogeneous wireless networks. *Mob. Netw. Appl.*, Vol. 9, No. 4, pp. 363–378, 2004.

[12] 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.

[13] Jon Postel. User Datagram Protocol. *RFC 768*, Aug. 1980.

[14] Jon Postel. Transmission Control Protocol. *RFC 793*, Sep. 1981.

[15] E.P.C. Jones, M. Karsten, and P.A.S. Ward. Multipath load balancing in multi-hop wireless networks. *Wireless And Mobile Computing, Networking And Communications, 2005. (WiMob'2005), IEEE International Conference on*, Vol. 2, pp. 158–166 Vol. 2, Aug. 2005.

[16] 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.

[17] T.E. Kolding, F. Frederiksen, and P.E. Mogensen. Performance aspects of wcdma systems with high speed downlink packet access (hsdpa). *Vehicular Technology Conference, 2002. Proceedings. VTC 2002-Fall. 2002 IEEE 56th*, Vol. 1, pp. 477–481 vol.1, 2002.

[18] 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.

[19] 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.

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

[21] 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.

[22] R. Braden. Requirements for Internet Hosts - Communication Layers. *RFC 1122*, Oct. 1989.

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

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

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

[26] 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.

[27] 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.