卒業論文 2004 年度 (平成 16 年度)

SBAM: ソケット層における帯域統合機構

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

南 政樹

慶應義塾大学 環境情報学部 環境情報学科

榊原 寛

skk@ht.sfc.keio.ac.jp

卒業論文要旨 2004 年度 (平成 16 年度)

SBAM: ソケット層における帯域統合機構

論文要旨

近年の通信技術の発達は様々な無線通信規格を登場させ，複数の無線通信デバイスが搭載された計算機の登場を促している．また，無線通信の発達は，ネットワークを利用しながら移動するユーザを増加させている．そして，広帯域ネットワークを対象としたストリーミングやビデオチャットなどのネットワークサービスの増加と，100Mbps / 1Gbps が一般的な有線通信に対して最大でも 11 / 54 Mbps の無線ネットワーク通信では確保できる帯域の開きによって，無線通信では享受できないネットワークサービスの増加が予想される．このような状況を踏まえて，本論文では計算機上に搭載されている複数の無線通信デバイスを同時に利用し，計算機上で利用可能な帯域を統合するための機構を研究対象とする．

既存の帯域統合機構に関する研究では TCP/IP カーネル層における実現方法が主である．例えばトランスポート層での実現方法やデータリンク層における実現方法が一般的に存在する．しかし，これらの手法では既存のネットワークを利用するアプリケーションを変更しなければ利用できないという問題や，ことなるセグメントのネットワークを同時利用できないと言った問題が発生する．本研究ではこれらの問題を解決するために，ソケット層における帯域統合機構 SBAM (A Socket-level Bandwidth Aggregation Mechanism) を提案し，設計，実装，評価を行なった．ソケット層に帯域統合機構を設置することにより，異なるネットワークを利用した帯域統合が可能となる．また，既存アプリケーションを変更する必要もない．

2 つの無線通信デバイスが搭載された計算機において通信帯域を統合しスループット評価を行なったところ，単一の無線通信デバイスを利用した場合と比べ，約 1.6 倍の帯域向上を実現した．また，送信ホストから受信ホストまでの 2 つのリンクの利用可能ネットワーク帯域が異なる場合でも，packet train 利用可能帯域測定方式を利用することにより，2 つのネットワーク帯域の効率的な利用を確認できた．

キーワード :
帯域統合, モバイルネットワーキング, アルゴリズム, デザイン, パフォーマンス

慶應義塾大学 環境情報学部 環境情報学科

榊原 寛

Abstract of Bachelor's Thesis Academic Year 2004

SBAM: A Socket-level Bandwidth Aggregation Mechanism

Summary

Recent growth of communication technology has caused the appearance of various wireless communication standards and computers with several devices which comply to such standards. Development of wireless communication technology also causes the increase of users who exploit network when they move. Moreover, growth of broadband network and widening of bandwidth gap between wired and wireless networks led to increase of network services which are not exploitable when using wireless networks. This thesis aims to aggergate bandwidth of multiple wireless communication devices on a computer.

Most of existing bandwidth aggregation mechanisms are implemented in network stacks. Transport layer or data link layer are commonly used for the aggregation, however there are problems such as a need for changes to existing software or unexploitability of different networks. In this thesis, we designed, implemented and and evaluated A Socket-level Bandwidth Aggregation Mechanism(SBAM). Bandwidth aggregation of different networks and unnecessity of changes to existing software are achieved exploiting the socket layer for bandwidth aggregation mechanism.

We measured the aggregated throughput of a computer which have two wireless communication devices, 1.6 times increase is confirmed compared to a computer which use only one wireless communication device. We also confirmed efficient bandwidth use for different links with bandwidths using packet train available bandwidth measurement technic.

Keywords :
Bandwidth Aggregation, Mobile Networking, Algorithm, Design, Performance

Faculty of Environmental Information, Keio University

Hiroshi Sakakibara

目 次

第 1 章　はじめに **1**
1.1　本研究の背景 2
1.2　本研究の目的及び意義 3
1.3　本論文の構成 4

第 2 章　移動計算機環境における帯域統合機構 **5**
2.1　本研究の想定環境 6
2.2　本論文の研究対象 6
2.3　複数通信デバイスを用いて帯域を統合する上で発生するシステム上の問題点 7
2.4　帯域統合機構実現レイヤについて 8
2.5　本章のまとめ 11

第 3 章　設計 **12**
3.1　システム概要 13
3.2　動作概要 13
3.3　ポリシー伝達機能 14
3.4　送信データスケジュール機能 14
3.5　送信データ分割機能 16
3.6　ネットワークモニタリング機能 16
3.7　受信データ統合機能 18
3.8　本章のまとめ 18

第 4 章　実装 **19**
4.1　カーネル実装 20
4.1.1　実装環境 20
4.1.2　パケットフォーマット 20
4.1.3　送受信の流れ 22
4.2　アプリケーション実装 23
4.2.1　実装環境 23
4.2.2　データ構造 24
4.2.3　ネットワークモニタリング機能 25
4.2.4　送信データスケジュール機能 27
4.2.5　送信データ分割機能 29

4.2.6 受信データ統合機能 29
4.3 本章のまとめ 29

第 5 章 評価 **30**
5.1 実験環境 31
5.2 スループット 31
5.3 到着パケットシーケンスについて 34
5.3.1 ネットワークの状態と到着パケットシーケンスの関係について . . 34
5.3.2 ネットワークモニタリング機能とパケット到着シーケンスの関係について 36
5.4 本章のまとめ 39

第 6 章 まとめと今後の課題 **43**

付 録 A APP1 のソースコード **50**

付 録 B APP2 のソースコード **70**

図 目 次

1.1 規格の制定年とスループット 2

2.1 本研究が対象とする想定環境 6
2.2 レイヤ図 9

3.1 システム構成図 13
3.2 キュー内のパケット例 15
3.3 帯域遅延積に比例したデータ送信 15
3.4 packet pair 帯域測定方式 17

4.1 パケットフォーマット 20
4.2 送受信における関数の呼び出し 22
4.3 アプリケーション実装のネットワーク構成 24
4.4 session_data{} 構造体 25
4.5 dev_ip{} 構造体 26
4.6 ebuf{} 構造体 26
4.7 利用可能帯域平滑化疑似コード 27
4.8 送信開始時の送信パケット数の決定方法 28
4.9 リンク数が 2 つの場合の比の決定方法 29

5.1 実験ネットワーク 1 の構成図 31
5.2 実験ネットワーク 2 の構成図 32
5.3 各関数の実行時間 33
5.4 NIC1 (11Mbps,0ms), NIC2 (11Mbps,0ms) 35
5.5 NIC1 (11Mbps,50ms), NIC2 (11Mbps,0ms) 36
5.6 NIC1 (11Mbps,0ms), NIC2 (5.5Mbps,0ms) 37
5.7 到着シーケンスの伸び Link1 (11Mbps), Link2 (2Mbps) 38
5.8 到着シーケンスの伸び Link1 (11Mbps), Link2 (4Mbps) 39
5.9 Link1 (11Mbps,0ms), Link2 (2Mbps,25ms) 遅延計測無 40
5.10 Link1 (11Mbps,0ms) Link2 (2Mbps,25ms) 遅延計測有 41
5.11 到着シーケンス番号 Link1 (11Mbps,0ms), Link2 (2Mbps,25ms) 遅延計測無 41
5.12 到着シーケンス番号 Link1 (11Mbps,0ms), Link2 (2Mbps,25ms) 遅延計測有 42

表 目 次

2.1 各ネットワーク層とソケット層における機能要件の定性比較 11

4.1 実装環境 20
4.2 シーケンス番号が周回するまでに要する時間 21
4.3 各ホスト上のプログラム名と機能のまとめ 24
4.4 ABAM 実装環境 25

5.1 SBAM と評価用アプリケーションのスループットの比較 32
5.2 スケジューリングの有無によるスループットの比較 34
5.3 実験環境その 1 34
5.4 実験環境その 2 36

第1章

はじめに

本章では，無線ネットワーク環境の現状を明らかにし，そこから生じる問題点を示す．そして，本研究がそれらの問題点を解決するために必要であることを示す．

1.1 本研究の背景

近年の通信技術とインターネットの普及は，以下に述べる 3 つの特徴を包含していると考えられる．

様々な通信規格の登場

近年の通信技術の発達に伴い，様々な通信規格が制定されている．例えば有線を利用する技術では IEEE 802.3ae [12], IEEE 802.3ab [11] などがあげられ，無線を利用する技術では，IEEE 802.11b [9]，IEEE 802.11a [8]，IEEE 802.11n [16]，Bluetooth [1]，Ultra WideBand (UWB) [22]，2G・3G 携帯電話通信などがあげられる．図 1.1 に，有線・無線デバイスの規格が登場した年とその速度のグラフを示す．図より無線デバイスと比べると有線デバイスの通信速度が速い傾向があることがわかる．また，無線通信規格が少なくとも 1 〜 2 年に一つ登場していることもわかる．このことからラップトップなどの計算機を利用しているユーザが新たにこれらの通信デバイスを購入し，装着する可能性があることが想定される．また，多様な無線機格が登場するに従い，販売している計算機に複数の無線デバイスが搭載されている状況も増えている．例えば，現在販売しているラップトップは IEEE 802.11b/g と Bluetooth などを搭載し販売されているケースが多くなっている．例えば IBM Thinkpad T42 には IEEE 802.11a/b/g を利用可能なネットワークインタフェース (N/I) が一つと Bluetooth，赤外線の 3 つの無線通信デバイスが装着されている．故に一つの計算機上に複数の無線通信デバイスが搭載される状況が増えているといえる．

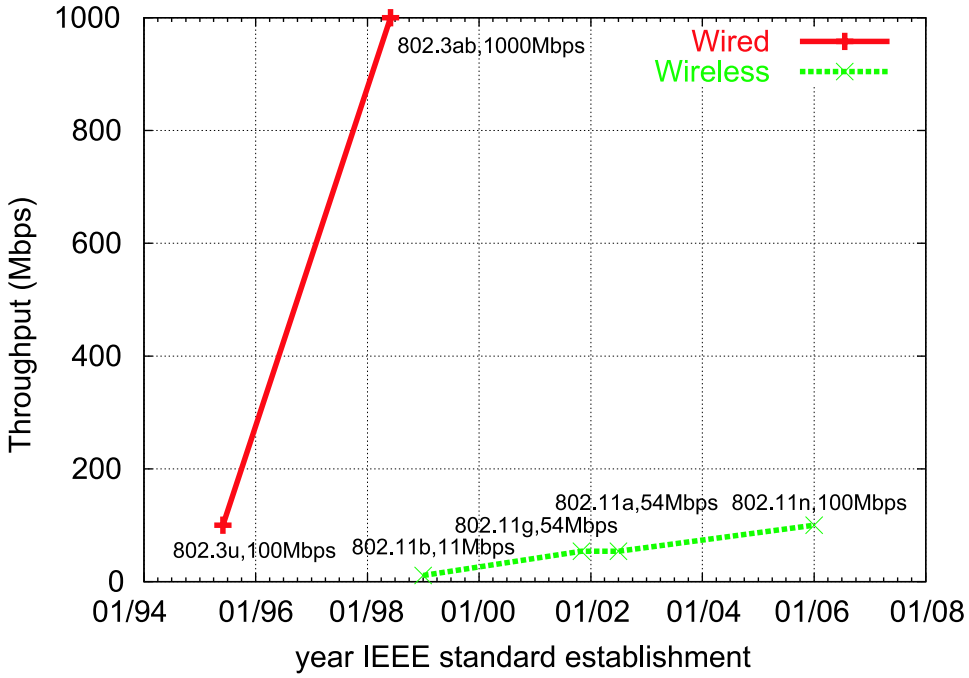

図 1.1: **規格の制定年とスループット**

計算機のモビリティの増大

小型化技術の発達により，様々な小型携帯デバイスが多く登場している．例えば，ラップトップコンピュータや PDA (Personal Digital Assistant)，また携帯電話などが挙げられる．これらのデバイスは数年前と比べ，重量も軽くなりバッテリー駆動時間も大幅に伸びている．小型化技術とバッテリー駆動時間の伸びにより，小型デバイスを携帯し，持ち歩きながらまたは出先で利用できるネットワーク環境が整ってきている．同時に，様々な地域・場所で利用可能なネットワークインフラが整ってきている．例えば，現在広域無線データ通信として PHS 網や 2G，3G 携帯電話網が利用可能である．また，街中ではホットスポットにおいて，IEEE 802.11b を利用した通信も行なえる．また，現状ではラップトップを対象とした場合，ネットワークを移動しながら利用することは少ないという報告が出ている [17] が，VoIP 対応の携帯電話などが登場することにより，移動しながらネットワークが利用される状況が増えると考えられる．

多様なネットワークコンテンツの登場

近年のインターネットの普及に伴い，ネットワーク上に様々なサービスやコンテンツが登場している．例えば，Web はテキストのみではなく，画像や音声を伴ったものが増えている．また，マルチメディアストリーミングサービスも，高品質なサービスを提供するために，広帯域を必要としていることが多い．

同時に，ネットワーク広帯域化の進展により，ネットワークの利用方法が利用可能ネットワーク帯域を気にせず，ユーザに身近な行動によるデータ転送へと変化してきている．例えば，計算機資源が乏しかった時代は，ファイルのやりとりを行なうには ftp に代表されるファイル転送用プロトコルを利用しファイルを効率的に転送していたが，現在では直観的にメールに添付して送ってしまうことが多い．これはネットワークが広帯域化していることによって可能となる．

1.2 本研究の目的及び意義

前節で挙げたネットワークを取り巻く環境の変化により，次のような問題点が考えられる．

- 様々な通信規格の登場により複数の無線デバイスが搭載される機会が増えているが，複数のデバイスを同時に利用することは少なく，多くは排他的に利用されている．例えば，IEEE 802.11b を利用している間は PHS によるデータ通信サービスは利用しない．このことは利用可能な資源を有効活用していないと言える．
- ネットワークコンテンツの品質は，ネットワーク利用の大半を占める有線ネットワークを中心に構成されていくと考えられる．同時に図 1.1 に示した通り，これからも無線デバイスよりも有線デバイスの帯域の方が広い傾向は続くと考えられる．

よって，無線通信では帯域が足りずにネットワークコンテンツのサービスを受けられない状況が発生すると考えられる．

以上のような問題点を解決するために，本研究では計算機上における利用可能帯域を向上させ，接続されている複数の無線通信デバイスを同時に利用可能とする SBAM (A Socket-level Bandwidth Aggregation Mechanism) を提案する．SBAM を利用することにより，ユーザは計算機上に接続されている複数の無線通信デバイスを有効に活用でき，かつ広帯域を確保できることにより，より良いサービスを享受できるようになる．

1.3 本論文の構成

本論文は以下の構成をとる．2 章において，現在の移動計算機環境とそこから発生する問題点について言及して帯域統合機構の必要性について述べ，ソケット層において実現することの利点について論じた後，ソケット層における帯域統合機構 SBAM を提案する．3 章で SBAM の設計を述べ，4 章においてその実装について言及する．5 章において SBAM を用いた評価実験を行ない，その結果について考察し，6 章を本論文のまとめとし今後の課題について言及する．

第2章

移動計算機環境における帯域統合機構

本章ではまず，本論文が対象とする想定環境について述べ，帯域を統合する機構に必要な要件を挙げる．次に機構をオペレーティングシステムのどの層に構築するべきかについて論じ，本研究が提案するソケット層における帯域統合機構 SBAM の特徴について述べる．

2.1 本研究の想定環境

本節では，本研究が対象とする計算機環境について述べる．ユーザは複数の無線ネットワークを利用できる地域を移動すると想定する．また，ユーザが持ち歩く計算機にはそれら複数の無線ネットワークに接続可能な無線通信デバイスが搭載され，ネットワークを利用するアプリケーションを利用し，サービスを受けているとする．例えば図 2.1 のユーザは， Bluetooth と IEEE 802.11b 無線通信が利用できる地域を移動しながら，ストリーミングサービスを利用し動画を見ている．

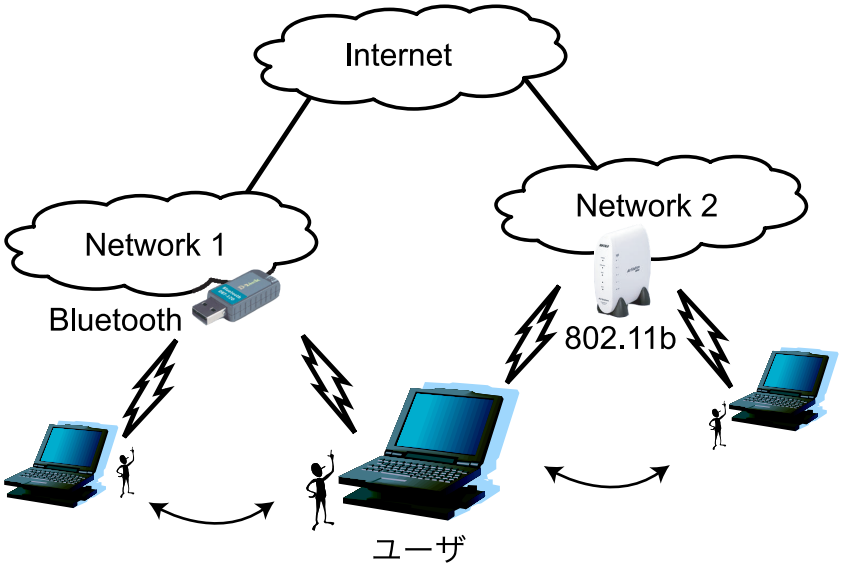

図 2.1: 本研究が対象とする想定環境

2.2 本論文の研究対象

本論文では，図 2.1 のユーザのような無線ネットワークのマルチホーム環境において，利用可能な複数の無線ネットワークを全て利用し，帯域を統合することとする．このことにより，計算機は無線通信デバイスの物理的な帯域に捕らわれることなく，広帯域を確保できる．

以下に 具体的な利用シーンを 2 つ述べる．

街中での利用例

近年，広域ネットワークデバイスが普及している．例えば，AirH” [4] や FreeD [3], また 2G，3G 携帯電話などが挙げられる．これらのデバイスの特徴として，1) 複数所持している場合，2) 同じデバイスを持っている他の人がいる場合などが考えられる．1) の場合，例えば PHS と 3G の携帯電話を持っている場合など，両方を利用することで帯域の

向上をはかれる．また，2) 場合は，他人，例えば友人などが広域通信デバイスを所有している場合，一人のラップトップに 2 つのデバイスを装着し，IEEE 802.11 経由で回線を共有するといった利用例も考えられる．

キャンパス内での利用例

現在，大学のキャンパス内において無線ネットワークを利用できる環境が増えている．本キャンパス [24] においても IEEE 802.11b がキャンパス内のどこにいても利用可能である．それに加え各研究室や活動団体において独自の無線ネットワークを構築し，キャンパス内ネットワークと併用可能な場合が多い．このように複数の無線ネットワークが利用可能な場合，それらに同時に接続し有線ネットワークに接続することなく広帯域を確保できる．

2.3 複数通信デバイスを用いて帯域を統合する上で発生するシステム上の問題点

2.1 節で述べた環境において，計算機に搭載されている複数の無線通信デバイスを同時に利用し，広帯域を確保するための帯域統合機構を構築する場合に発生する問題点を以下に挙げる．

1. 本研究では一つのデータを送受信するのに，複数のリンクを用いるので，リンク毎に送受信データのスケジューリングを行なう必要がある．この時，各リンクの状態は途中経路によってそれぞれ異なる．よって無線通信デバイスが装着された計算機から相手先ホストに対しデータを送信する際，各 N/I に対して単純なラウンドロビンスケジューリングによるデータ送信はできない．例えば，帯域が異なるリンクに対してラウンドロビンスケジューリングでデータ送信を行なった場合，速いリンクは遅いリンクのデータ送信が完了するまで，次の送信データを割り当てられないので，2 つのリンクの帯域を効率的に利用できない．また，各リンクのネットワーク遅延が異なる場合，送信したデータが受信ホストに配送される時間が異なってしまう．このため，受信ホストでは，遅い遅延のリンクからのデータを待つため，データを保持するキューを大量に用意しなければならない．

2. 無線通信デバイスには様々な特性が存在する．例えば，PHS によるデータ通信サービスには，データ転送量に応じて課金されるものがある．このような無線通信デバイスを利用しているユーザにとっては，帯域統合機構により他の課金されない無線通信デバイスと共に大量のデータ送受信をするためのデバイスとして利用されてしまうと，意図しない課金が発生し問題である．

3. 一つの通信デバイスを利用する場合，途中経路のネットワーク状態によりデータの配送順序が変わる場合があるが，基本的に単一の経路を通り送信ホストが送信した

データ順序で受信ホストにデータは配送される．しかし，帯域統合機構を利用することにより，複数の経路を経由してデータが配送されるので，受信ホストでは受信した通りにデータを再構築しても，送信ホスト上におけるデータと同一のデータを得られない場合が多くなる．このことは単一経路において生じるデータ配送順序の乱れよりも，高い確率で発生すると考えられる．

4. 利用していたネットワークや無線通信デバイスが予期せず利用できなくなる場合がある．例えばユーザが移動し無線通信可能範囲から離れた場合や，ユーザがデバイスを計算機から外した場合などが想定される．この時，利用できなくなったネットワークや無線通信デバイスから送信しようとしていたデータが送信できずに破棄されてしまう可能性がある．

3 章においてこれらの問題を解決できる帯域統合機構について詳しく論じる．帯域統合機構の詳細を述べる前に，本機構をオペレーティングシステム上のどのレイヤにおいて実現するべきかについて次節で論じる．

2.4 帯域統合機構実現レイヤについて

帯域統合機構をネットワークスタックにおいて実現する場合，データリンク層，ネットワーク層，トランスポート層，アプリケーション層などが考えられるが，本研究ではネットワークスタックではなく，OS の機能であるソケット層における実現手法を採用した．図 2.2 に OSI 参照モデルと TCP/IP ネットワークスタックの対応を示す．ソケットとは OS が提供するネットワークを利用するための API であり，呼び出す際の引数を変更することによって，利用する下位トランスポートプロトコルを変更する．その他にも，直接 IP プロトコルを呼び出すためや，OS が保持している経路表を書き換えるためにも利用される API である．

以下，各ネットワークスタックにおける利点・欠点を挙げ，ソケット層における帯域統合機構の実現手法の優位性を示す．また，各ネットワークスタックにおいて提案・実現されている関連研究についても随時言及し，その利点・欠点を述べる．

ネットワーク層/データリンク層

2.3 節で問題点として挙げた通り，帯域統合機構を実現するには，エンドホスト間のネットワークの状態を把握し，その情報に応じて各 N/I からデータ送信を行なう必要がある．そのため Hop-by-Hop で処理されるプロトコルではエンドホスト間のネットワーク状態を取得できないので，ネットワーク層やデータリンク層における帯域統合機構の実現は現実的ではない．但し，帯域統合機構の存在する計算機間の帯域や遅延などのリンク状態が全く同一ならば，単純なラウンドロビンスケジューリングを用いることで帯域統合を実現でき，複雑な処理が必要ないという利点が存在する．このような規格として，IEEE 802.3ad [10] がある．これはホスト上に同等の性能の N/I を複数装着し，ネク

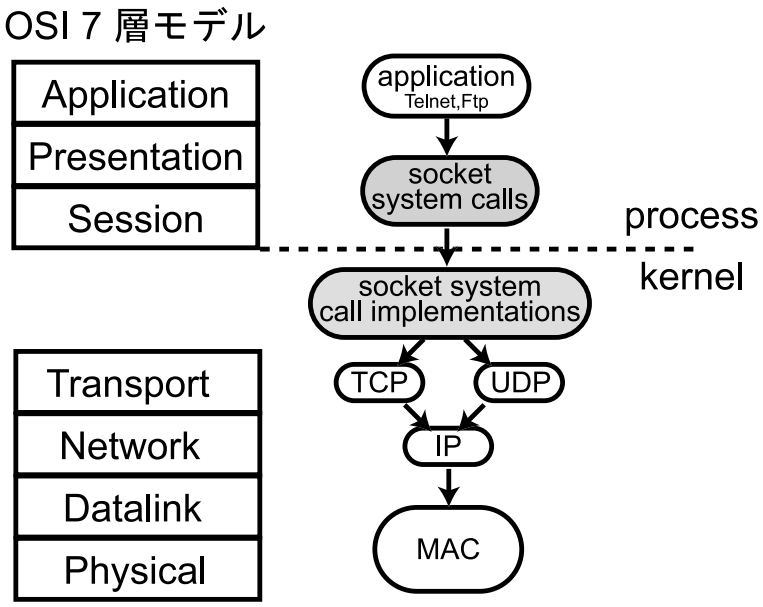

図 2.2: レイヤ図

ストホップまでの帯域を向上させるための仕組みである．この方法は OS やアプリケーションなどに全く影響を与えることなく実現できる点において優れているが，利用できるネットワークの種類は 1 つに限られる．つまり，そのネットワークのアップリンク以上の帯域は出ないことになる．また，802.3ad は有線のための規格なので，無線リンクには対応していない．

本研究で対象とする帯域統合機構を利用する環境では，異なる無線ネットワークに接続することを想定しているので，一つのネットワークへの接続しか想定していない機構では不適切である．また，利用するネットワークメディアを無線としていることから，電波干渉が発生しスループットの向上が見込めないという問題も発生する．

トランスポート層

帯域統合機構を新たなトランスポートプロトコルとして実装した場合，次の 3 つの欠点がある．1) そのプロトコルを利用するために，既存アプリケーションを書換える必要がある．2) 相手ホストでそのプロトコルを利用できない場合，通信は不可能である．この場合，アプリケーションにおいて動的に利用プロトコルを切替える必要がある．3) TCP を改良する場合，TCP の保持している帯域・遅延情報を利用し，効率的に帯域統合機構を実現できるという利点が存在するが，TCP はインターネット上で広く利用されており，全てを新たなトランスポートプロトコルに置き換えるのは現実的ではない．しかし置換を行なわなければ通信できないホストが発生してしまう．

トランスポート層において帯域統合機構を実現している関連研究として，Hung-Yung Hsieh らの PTCP [6] と RCP/R2CP [7] が挙げられる．PTCP は送信ホスト側で輻輳制御や再送制御などを行っているが，RCP では，それらの機能を受信ホストにおいて実現し，

無線メディアに適応した信頼性のあるトランスポートプロトコルを提案している．また，マルチホーム環境の場合には，R2CP というモジュールが N/I 毎に RCP を用意し，効率的な帯域統合機構を実現している．

R2CP はトランスポート層で実現されているため実現が容易という利点があるが，ネットワークの自由度が低くなる欠点がある．例えば，RCP は TCP のような信頼性のあるトランスポートプロトコルであるが，ユーザが UDP のような信頼性のないプロトコルを利用したい場合，新たに RCP に相当するトランスポート層を用意しなけらばならない．また，トランスポート層において送信側と受信側が対応しなければならないので，通信が全く行えない可能性がある．

アプリケーション層

Hung-Yung Hsieh ら [7] によると，アプリケーション層において帯域統合機構を実現した場合，N/I の数に対するスケーラビリティが低いという結果が報告されている．また，既存のプログラムで帯域統合を実現する場合，そのプログラムに帯域統合機構を実装する必要がある．

アプリケーション層において帯域統合を実現している関連研究としては，高速ネットワークにおいて，TCP の Window サイズ限界から生じるスループット上限を回避する，PSocket [5] や XFTP [15] などがあげられる．しかし，これらは全て単一リンクにおいて利用することを仮定しており，我々が調査した限りでは，本研究が対象とするようなマルチホーム環境を想定した機構は存在しない．

ソケット層は OS の機能でありながら，ネットワークスタックではないので，ネットワークスタックにおける既存のプログラムの変更という欠点を回避しつつ，アプリケーション層における欠点も回避できる．表 2.1 に各ネットワークスタックとソケット層における利点・欠点をそれぞれをまとめた．異なる種類のリンクを利用する点に関しては，ソケット層が End-to-End でデータのやりとりを行なうため可能である．新たなトランスポートプロトコルが利用されることになったとしても，ソケット層はトランスポート透過的に利用可能なので，ネットワーク内におけるディプロイは容易であると言える．また，カーネル内での実装であることから，N/I の枚数に対するスケーラビリティは高いと言える．最後に性能は，トランスポート層またはデータリンク層/ネットワーク層における実現方法が，オーバヘッドが少なく効率的であると考えられる．

以上の点から，トランスポート層やデータリンク層/ネットワーク層に対して性能面で劣る点があるが，その他の利点が多いので，本研究では帯域統合機構をソケット層で実現する SBAM (A Socket-level Bandwidth Aggregation Mechanism) を提案する．

表 2.1: 各ネットワーク層とソケット層における機能要件の定性比較

	Data/Network	Transport	Application	Socket
異なるリンクへの対応	×	○	○	○
ディプロイの容易さ	×	×	○	○
スケーラビリティ	○	○	×	○
性能	○	○	×	×

2.5 本章のまとめ

本章ではまず，本研究が想定する環境について述べた．次に，その環境下において帯域統合機構を実現する際に発生する問題点を明らかにした．その後，様々なネットワークスタックにおける帯域統合機構を実現する場合の利点・欠点について検討し，ソケット層において実現することの利点を示した．そして最後に本研究ではソケット層において帯域統合機構を実現することを提案した．

第3章

設計

本章では前章で述べたシステム構築上の問題点と帯域統合機構実現レイヤに関する議論を踏まえ，SBAM のシステム構成とその構成要素について詳述する．

3.1 システム概要

SBAM はソケット実装なので，アプリケーションは特に SBAM を利用していることを意識せず，通常のネットワークプログラミングを行なう際と同様に，socket() システムコールを呼ぶだけで良い．ただし，ポリシーを設定する場合は後述のポリシー伝達機能に対し，ポリシーを設定してから socket() を呼ぶ必要がある．

3.2 動作概要

本節では，ソケット層の内部に存在する SBAM の各機能がどのように連係し動作するかについて述べる．各機能の詳細については後述する．図 3.1 にシステム構成図を示す．

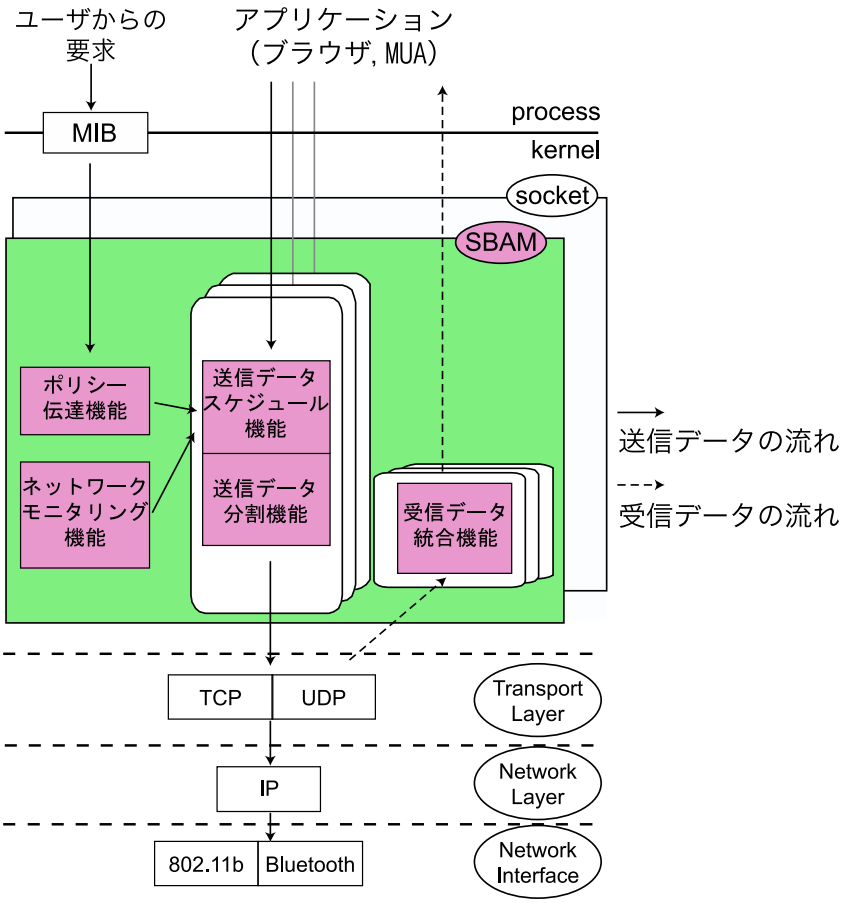

図 3.1: システム構成図

データ送信時にはまず，ポリシー伝達機能が MIB (Management Information Base) [13] から情報を読みとり，その値を送信データスケジュール機能に渡す．また同時に，ネットワークモニタリング機能が通信先ホストまでのネットワーク状態の計測を開始し，その値を送信データスケジュール機能に渡す．計測は定期的に行なわれ，計測結果は適宜送信データスケジュール機能に渡される．ネットワークモニタリング機能とポリシー伝

達機能はそれぞれシステムで一つ動作している機能である．送信データスケジュール機能では，ポリシ伝達機能とネットワークモニタリング機能から得た情報をもとに，各 N/I に対するデータを配分を決定し，その情報を送信データ分割機能に渡す．送信データ分割機能では，渡された値をもとに下位トランスポート層に対する送信データの配分を決定し，SBAM ヘッダを付加した後，下位トランスポート層にデータを渡し，データの送信が行なわれる．送信データスケジュール機能と送信データ分割機能は，アプリケーション毎に動作する機能である．

受信時には，受信データ統合機能において，SBAM ヘッダを取り除き，パケット順の並べ替えを行なった後，データを統合しアプリケーションにデータの到着を通知する．この機能はアプリケーション毎に動作する機能である．

次節より各機能の詳細な説明を行なう．

3.3 ポリシー伝達機能

本機能は，MIB を通してカーネル空間とユーザ空間の間でユーザポリシのやりとりを行う．ユーザポリシとは，ユーザの各 N/I に対する利用要求であり，2.3 節において挙げた，ユーザの意図と反する無線通信デバイスの利用を防ぐための機能である．

3.4 送信データスケジュール機能

送信データスケジュール機能では，送信データ再スケジュールと各 N/I の状態に応じたデータ送信を行なう．本節ではそれぞれについて述べる．

送信データ再スケジュール

何らかの理由である N/I が利用不可能になった場合，他の利用可能な N/I に対して送信データの再スケジュールを行う．また，利用可能な N/I の把握も行なう．例えば，ユーザが N/I をとりはずしたり，電波が届かなくなり通信を行なえなくなった場合，送信データの再スケジュールを行なう．再スケジュールを行う際，図 3.2 のように，キュー内のデータを並べ替える．図 3.2 は N/I-A と N/I-B が存在し，それぞれがキュー内にデータを保持していることを示している．このような状況で N/I-B が利用不可能になったとする．この場合，N/I-B のキュー内の送信データは図のように，N/I-A のキュー内で並べ変えられ送信される．このことにより，受信ホストにおけるデータの並べ替え作業の軽減を行う．

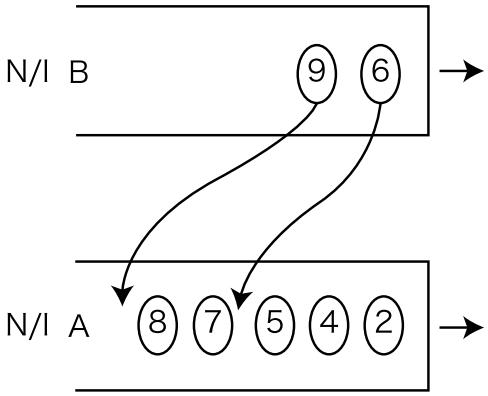

図 3.2: キュー内のパケット例

各 N/I の状態に応じたデータ送信部

後述のネットワーク状態モニタリング機能より，各リンクの帯域・遅延情報を受けとり，MTU (Maximum Transfer Unit) を考慮しつつ各リンクの帯域遅延積を埋めるようにデータ配分量を決定する．以下，帯域と遅延情報から各リンクへ送信するデータ量の決定方法について述べる．

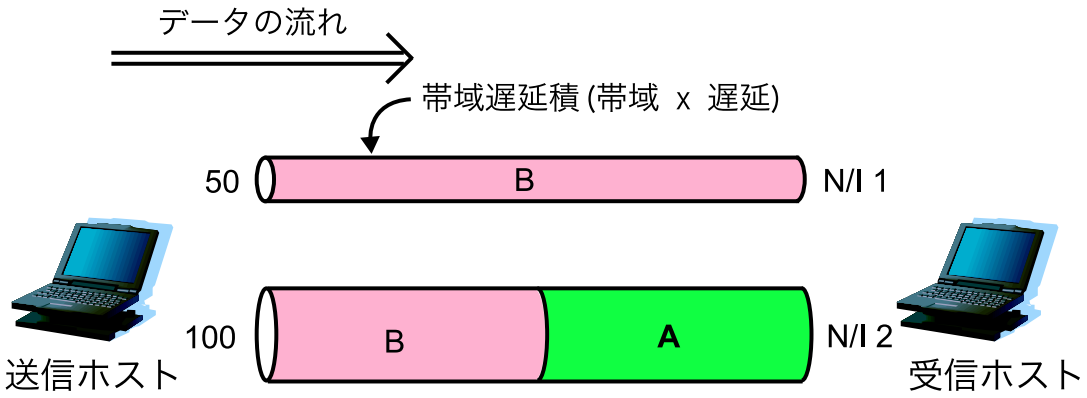

図 3.3: 帯域遅延積に比例したデータ送信

図 3.3 にデータ送信の流れを表す．2 本の円柱は送信ノードから受信ホストまで 2 つのリンクが存在することを表し，円柱の体積は帯域遅延積を示す．帯域遅延積は n をリンクの数とし，遅延を $d_i(sec)$，帯域を $b_i(bps)$ とする時，$d_i \times b_i$ で表される．まず通信開始前に，各リンクの帯域遅延積を求める．図 3.3 ではそれぞれ 50, 100 である．次に，各リンクの帯域遅延積を比較し，その値が最小のリンク以外に対し，式 (3.1) で表される P_i パケットを送信する．m_i は MTU を，α_i は，ポリシー伝達機能によって指定された各 N/I の重みを表す．

$$P_i = \frac{\alpha_i b_i d_i - min(\alpha_1 b_1 d_1, ..., \alpha_n b_n d_n)}{m_i} \ (0 \leq \alpha \leq 1, i = 1...n) \tag{3.1}$$

この操作は図 3.3 中の A 部に相当し，$50 = 100 - 50$ のデータを帯域遅延積が 100 のリンクに対して送信している．各リンク間の帯域遅延積量を一定にすることにより，遅延の差による帯域遅延積の差を埋められる．

次に，各リンクの帯域に比例したデータ量を送信する．この操作は図 3.3 中の B 部に相当する．各 N/I に対して割り当てるパケット数は，各 N/I の MTU を考慮しつつ相手先ホストまでの利用可能帯域に比例させるため，式 (3.2) の P_i で表される．gcd は最大公約数を表し，lcm は最小公約数を表す．

$$P_i = \alpha_i \frac{b_i \cdot lcm(m_1 \cdots m_n)}{m_i \cdot gcd(b_1 \cdots b_n)} \ (i = 1...n) \tag{3.2}$$

例えば，帯域が 2Mbps，MTU が 500 バイト，遅延が 100ms のリンク A と，帯域が 1Mbps，MTU が 1000 バイト，遅延が 50ms のリンク B が存在したとする．この場合，通信開始時はリンク A に対して，式 (3.1) より，$(2000000(bps) \cdot 0.1(sec) - 1000000(bps) \cdot 0.05(sec))/(500 \cdot 8) = 37.5 \fallingdotseq 38$ パケット送信する．次に，各リンクに対する送信パケット数は，式 (3.2) より，リンク A に対して $(2000000 \cdot 1000)/(500 \cdot 1000000) = 4$ パケット，リンク B に対して同様に，1 パケットとなる．このことにより，各リンクの帯域遅延積を埋めることができ，効率的に帯域を使用できる．

3.5 送信データ分割機能

送信データを，送信データスケジュール機能から得られた送信データ量に従い分割し，且つ受信ホストの受信データ統合機能で利用される SBAM ヘッダの付与も行なう．SBAM ヘッダの詳細に関しては 4.1.2 節において詳細に述べる

3.6 ネットワークモニタリング機能

本機能は前節まで述べた送信データ分割機能と送信データスケジュール機能のために受信ホストまでの遅延・利用可能帯域を取得する．遅延データは，定期的に ICMP パケットを受信ホストに送信し取得し，式 (3.3) で表される TCP で利用されている遅延の平滑化手法を利用し保持する．RTT は計測された RTT (Round Trip Time) を表し，$SRTT$ は平滑化された RTT を表す．α は平滑化係数であり，RFC793 [18] では，TCP における推奨値として 0.9 を採用している．本システムではこの値を利用する．

$$SRTT = \alpha SRTT + (1 - \alpha) \times RTT \ (0 \leq \alpha \leq 1) \tag{3.3}$$

帯域情報の取得には，packet pair 帯域測定方式 [19] を利用する．図 3.4 ([19] を基にした) に packet pair のパケットのフロー図を示し，packet pair の動作概要を述べる．x 軸はリンク間に存在するホストやルータを示し，y 軸は時間軸を表す．

まず，時間 s において SOURCE より同じサイズの 2 つのパケットを，連続して SINK ホストへ送信する．途中経路上にはパケットをラウンドロビン処理するルータ SERVER1

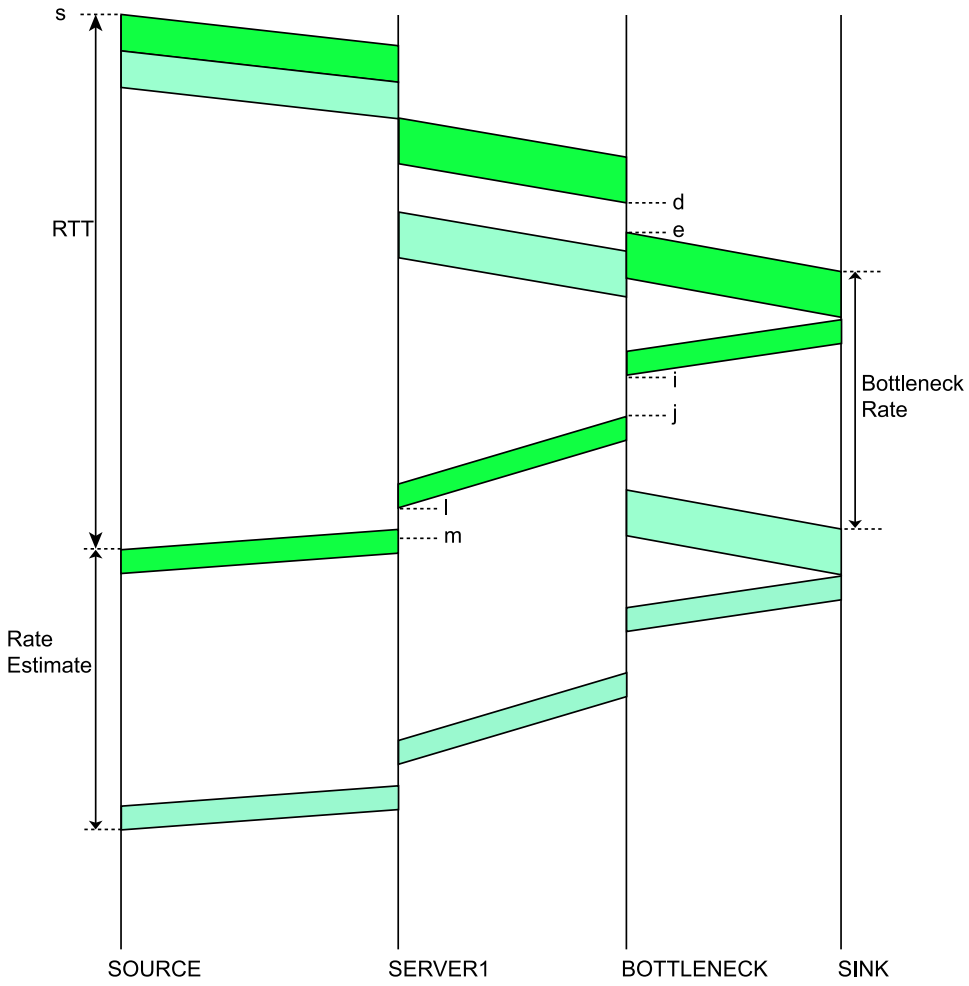

図 3.4: packet pair 帯域測定方式

と BOTTLENECK がそれぞれ存在する．SINK ホスト上において，受けとった 2 つのパケット間ギャップが BOTTLENECK ルータを通り過ぎる際の時間が Bottleneck Rate となる．この 2 つのパケットに対する ack が Bottleneck Rate の通りに SOURCE ホストまで届いた場合，SOURCE において SOURCE - SINK ホスト間の利用可能帯域を以下の式を用いて求められる．P_s はパケットサイズ，T_{gp} は ack の到着時間の差，Bw は利用可能ネットワーク帯域をそれぞれ表す．ack が SINK ホスト上において生じたパケットギャップのまま SOURCE ホストに配送するため，ack パケットのサイズは SOURCE から送信されたパケットよりも小さくする必要がある．

$$Bw = \frac{P_s}{T_{gp}} \tag{3.4}$$

本機能は帯域統合機構が動作するホスト内において，一つだけ動作する機能である．前述の送信データスケジュール機能はアプリケーション毎に動作する必要があるのに対し，本機能は帯域統合機構を利用しているホスト内の，全てのプロセスに対し統一的にネットワーク状態をモニタし，効率的に遅延・帯域情報を提供する．例えば，複数の別

プロセスが同一ホストに対して通信を行なう場合，それぞれのプロセスが個別にが帯域測定を行なうと，測定のためのコントロールパケットが増大し，システムのスケーラビリティ低下が予想される．よって本機構では，ネットワークモニタリング機能をホスト内において一つだけ動作する機能とした．

3.7 受信データ統合機能

本機能は，分割されて送られてきたデータを統合する機能である．後述の SBAM ヘッダを読みとり，パケットの並べ換えを行ない，受信データをアプリケーションに渡す．

パケットの再構築をどのように行なうかは，下位層で利用されているトランスポートプロトコルによって 2 つに分けられる．TCP や STP [21] のように信頼性が求められるプロトコルが利用されている場合は，ヘッダ情報を利用して正確にデータを再構築する．これに対し UDP のような信頼性は必要とされていないが高速なネットワーク転送と少ない遅延が求められるプロトコルが利用されている場合は，SBAM におけるパケットキューの長さを固定し，パケットの並べ換えよりもアプリケーションに対してデータを早く渡すことを優先する．

3.8 本章のまとめ

本章では，まず，本研究が対象とする想定環境について述べ，そこで発生する問題点を整理した．次に，設計方針を述べたあとに，問題点を解決するために SBAM に必要となる機構について論じ，設計とした．

第4章

実装

本論文では，SBAM の構成機能を有する 2 種類の実装を行なった．まず，カーネル内部のソケット層において，送信データ分割機能と受信データ統合機能を有するプロトタイプ実装を行なった．次に，ネットワークモニタリング機能と送信データスケジュール機能の有効性を確かめるためにアプリケーション層においてネットワークモニタリング機能と送信データスケジュール機能，受信データ統合機能を有する評価用実装を行なった．それぞれの実装ではトランスポートプロトコルに UDP を用いて実装を行なった．

表 4.1: 実装環境

マシン	ThinkPad X30, ThinkPad T30
OS	FreeBSD 5.1-Release
NIC	BUFFALO WLI-PCM-L11, Intersil Prism2.5

4.1 カーネル実装

本節ではカーネル内部のソケット層における実装について述べる．また，シーケンス番号を収める空間に関しての議論も行なう．

4.1.1 実装環境

カーネル実装は，表 4.1 に示す環境で実装を行なった．また，ネットワーク層のプロトコルとして IPv4 を利用しているので，受信ホストにおいて指定した N/I からデータを送信するために，送信データ分割機能においてデフォルトゲートウェイを変更している．

4.1.2 パケットフォーマット

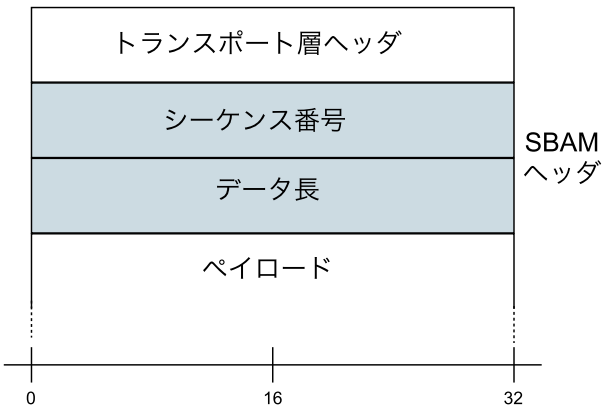

図 4.1: パケットフォーマット

SBAM では，図 4.1 に示されるようなヘッダを各パケットに付与する．トランスポート層ヘッダには，SBAM が利用する TCP や UDP などの下位トランスポートプロトコルのヘッダが入る．シーケンス番号は受信データ統合機能でデータの順序を入れ換えるために必要となる．データ長はそのパケットが SBAM 対応のパケットなのかを確認するためと，ペイロード部のデータ長を把握するために存在する．データ長に収められているデータ長とペイロード部のデータ長が一致しない場合や，下位トランスポートプロト

表 4.2: シーケンス番号が周回するまでに要する時間

bandwidth	time until wrap-around
T1 (1.5Mbps)	6.4hrs
Ethernet(10Mbps)	57min
T3 (45Mbps)	13min
FDDI (100Mbps)	6min
OC-3 (155Mbps)	4min
OC-12 (622Mbps)	55sec
OC-24 (1.2Gbps)	28sec

コルが対応するペイロード長以外の値が格納されている場合，そのパケットは SBAM パケットではないと判断する．

シーケンス番号を収める空間について

実装を簡易にするため，SBAM パケットヘッダに含まれるシーケンス番号空間は 32bit としたが，近年高速なネットワークが普及し，TCP のシーケンス番号周回問題が指摘 [20] されている．表 4.2 に帯域とシーケンス番号が周回するまでに要する時間の関係を表す．1.2Gbps の帯域を持つネットワークでは 28 秒で同じシーケンス番号を持つ別のパケットが存在することになる．

SBAM で利用を想定しているネットワークメディアは無線ネットワークである．よって表中の 10Mbps 〜 100Mbps 程度が現在の対象となる．しかし，IEEE 802.11n や UWB などが登場し利用できる帯域が増えることによって，周回時間が短くなることが想定される．その場合，番号周回問題を解決するには，以下の 2 つの手法が考えられる．

1. ヘッダにタイムスタンプ情報を付与し，PAWS (Protect Against Wrapped Sequences) [23] を適用する
2. シーケンス番号を収める空間を広くする

1 で挙げた PAWS とは，TCP セグメントを送信する場合，送受信側双方でタイムスタンプをヘッダ内に収め，シーケンス番号周回が起きた場合でも，パケット識別ができなくなるという問題を回避する手法である．RFC1323 [23] ではタイムスタンプのためのフィールドとして 32bit を割り当てている．2 は単純にシーケンス番号を収める空間を広くとる手法である．それぞれの手法には以下のような特徴がある．

- SBAM シーケンス番号空間を 64 bit にした場合，シーケンス番号が周回するのに 1.2Gbps の帯域で 1200 億秒かかる計算になる．

- PAWS を実装する場合，タイムスタンプを保持するために 32 bit の空間が必要なのに加えて，時刻を計算するオーバヘッドが生じる．

よって，将来の SBAM ではシーケンス番号空間を 64bit にすることが望まれる．

データ長を収める空間

シーケンス番号を収める空間に関する議論と同様の議論をデータ長を収める空間に関しても行なえる．32 bit の空間には $2 * *32 = 4294967296 \fallingdotseq 43$ 億バイトまでのデータ長を格納できる．現在，UDP でデータ送信を行なう場合，1 セグメントのサイズは 65536 バイトが上限である．よって，32 bit の空間はデータ長を収めるのに十分であると考えられる．

4.1.3 送受信の流れ

本小節では，送受信における処理の流れを示す．図 4.2 に，送受信時にカーネル内でどのような関数が呼び出されるかを示す．背景が灰色になっている関数において実装を行なった．

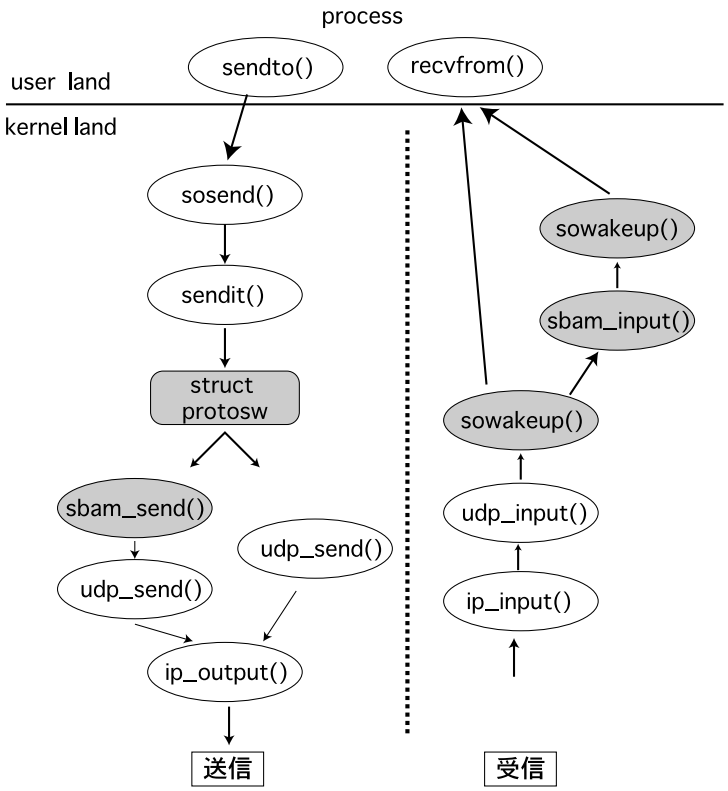

図 4.2: 送受信における関数の呼び出し

送信時

通常のデータ送信の場合，FreeBSD オペレーティングシステムの実装では，sosend 関数内で利用するトランスポートプロトコルに応じた関数を protosw inet 構造体を利用し呼び出す．sosend は下位トランスポートプロトコルが必要とする形に従い，send システムコールや write システムコールの情報を渡す関数である．下位トランスポートプロトコルは TCP/IP プロトコルスタックの場合，protosw inet 構造体に関数ポインタの形で登録されている．SBAM では UDP プロトコルスタックを呼び出す前に，sbam_send という SBAM の処理を行なう関数を呼び出し，udp_send 関数を呼んでいる．sbam_send は 3 章で述べた送信データ分割機能と送信データスケジュール機能にあたる．

受信時

通常，受信したデータは，ip_input，udp_input 関数で処理されソケットバッファにためられる．そして sowakeup 関数により，プロセスはそのデータを読み出す．SBAM では，sowakeup 関数内より input 関数を呼び出し，SBAM ヘッダの処理を行なう．input 関数が 3 章で述べた受信データ統合機能にあたる．

4.2 アプリケーション実装

本節では，SBAM のネットワークモニタリング機能と送信データスケジュール機能，受信データ統合機能を評価するために実装したアプリケーション (ABAM) について述べる．次小節の実装環境において，利用した機器やネットワーク環境について述べた後，実装の詳細について説明する．

4.2.1 実装環境

図 4.3 に，本実装が利用したネットワーク環境を示す．ホスト A とホスト B を，無線ルータ 1 と 2 を利用して接続した．ホスト A は無線ルータ 1 に有線で接続した．また，ホスト B には IEEE 802.11b 無線ネットワークインタフェースが 2 枚装着されており，それぞれ無線ルータ 1 と無線ルータ 2 に接続している．

本実装では，マルチメディアデータをモバイルホストがダウンロードしているのを想定し，ホスト A からホスト B に対してデータを送信するのに必要な機能をそれぞれのマシン上において実装した．ホスト A では，送信データスケジュール機能，送信データ分割機能，ネットワークモニタリング機能を実装した sbam_server を，ホスト B では，受信データ統合機能を実装した sbam_client が実行される．表 4.3 にホスト上のプログラムとそのプログラムに実装されている機能をまとめる．実装に利用した機器を表 4.4 に示す．

次節以降，機能間でデータを交換するのに利用されるデータ構造について述べた後，各機能について述べていく．

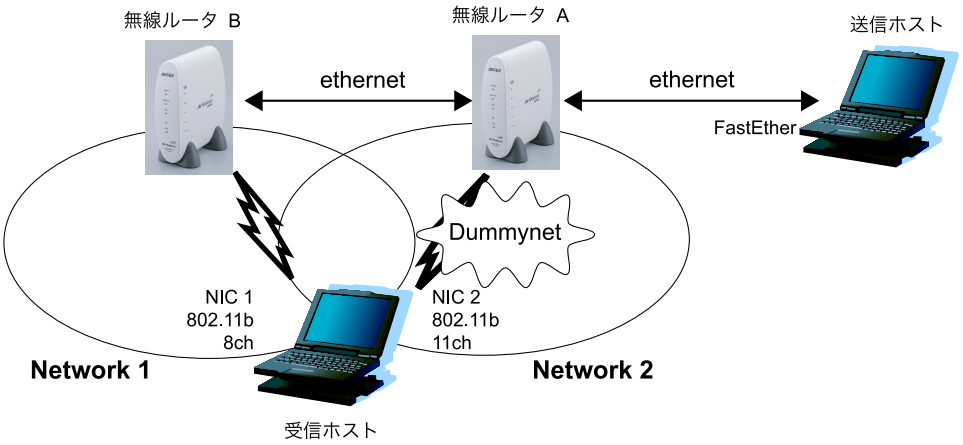

図 4.3: アプリケーション実装のネットワーク構成

表 4.3: 各ホスト上のプログラム名と機能のまとめ

マシン	プログラム名	実装されている機能
ホスト A	sbam_server	ネットワークモニタリング機能 送信データスケジュール機能 送信データ分割機能
ホスト B	sbam_client	受信データ統合機能

4.2.2 データ構造

sbam_server，sbam_client ともに同じデータ構造を利用し，各リンクの情報を保持している．図 4.4 に session_data 構造体，図 4.5 に dev_ip 構造体，4.6 に ebuf 構造体と socket_header 構造体をそれぞれ示す．

session_data 構造体には各セッションに関する情報が収められている．セッションとはホスト A からホスト B に対するデータ送信の開始から終了までを指す．char ipaddr[] は sbam_server が動作しているホスト A の IP アドレスを保持し，int nic_num は sbam_client の動作しているホスト B に装着されている NIC の数を保持している．tq_dvip 構造体は次に説明する，dev_ip 構造体の連結リストへのエントリポイントである．tq_dvip を宣言する時に利用されている TAILQ_XXX とは FreeBSD に付属している双方向連結リストを扱うためのマクロである．

dev_ip 構造体には，エンド間に存在するリンクの情報が収められ，双方向連結リストを用いて任意の数の情報を保持できるようになっている．このことにより，リンクの数に関わらずリンク情報を保持できる．char devname[] には，ホスト B 上の無線のデバイス名が収められており，ip_addr 構造体にはそのデバイスに割り振られている IP アドレスが収められている．int port_to_probe にはホスト A から送られるネットワークモニタリングのための UDP パケットを待つためのポート番号が収められており，int port_to_ack に

表 4.4: ABAM 実装環境

マシン	ThinkPad X30, ThinkPad T30
OS	FreeBSD 5.1-Release
NIC	BUFFALO WLI-PCM-L11, Intersil Prism2.5
無線ルータ 1	BUFFALO AirStation AG54
無線ルータ 2	BUFFALO AirStation G54

```
struct session_data {
        char ipaddr[20]; /* sbam_server IP address */
        int  nic_num;    /* nic_num on sbam_client */
        /* list of struct dev_ip */
        TAILQ_HEAD(tqhead_dvip, dev_ip) tq_dvip;
        /* for making linked-list */
        TAILQ_ENTRY(session_data) f_link;
};
TAILQ_HEAD(tqhead, session_data) tq;
```

図 4.4: **session_data{} 構造体**

は受けとったモニタリングパケットに対する確認応答を投げるポート番号が入る．double smoothed_capacity と double smoothed_rtt にはネットワークモニタリング機能が取得した利用可能帯域と RTT が平滑化され収められる．詳細は 4.2.3 節にて説明する．int bandwidth_ratio には，全てのリンクの利用可能帯域の比が収められている．この値は送信データスケジュール機能によって定められる．詳細は 4.2.4 節にて説明する．tq は次に説明する ebuf 構造体の連結リストへのエントリポイントである．

ebuf 構造体は，パケットデータを収めるためのデータ構造であり，sh 構造体と char *buf にそれぞれ SBAM ヘッダとそのペイロードがそれぞれ格納される．SBAM ヘッダは socket_header 構造体に格納され，メンバの u_int32_t sock_sequence，u_int32_t data_length はそれぞれ SBAM が付与するシーケンス番号と ebuf 構造体のメンバ buf のデータ長である．

4.2.3 ネットワークモニタリング機能

ネットワークモニタリング機能は sbam_server 内に実装されており，3.6 節で述べた通り各リンクの帯域と RTT の計測を行なう．

```
struct dev_ip {
        /* device name */
        char devname[IF_NAMESIZE];
        /* ip address of wireless NIC on host B */
        struct sockaddr_in ip_addr;
        /* wait port on sbam_client */
        int port_to_probe;
        /* wait port on sbam_server */
        int port_to_ack;
        double smoothed_capacity;
        double smoothed_rtt;
        int bandwidth_ratio;
        /* for making linked-list */
        TAILQ_ENTRY(dev_ip) f_link;
        /* receive buffer on sbam_client */
        TAILQ_HEAD(tqhead, ebuf) tq;
};
```

図 4.5: **dev_ip**{} **構造体**

利用可能帯域の計測

本プロトタイプ実装では packet pair 方式を拡張した，packet train [2] 方式を実装し，利用可能ネットワーク帯域の計測を行なった．packet pair 方式が帯域を計測するのに 2 つのパケットを連続して送信するのに対し，packet train は 3 つ以上の複数のパケットを連続して送信する．このことにより途中経路におけるネットワーク負荷の影響が低くなる．また，パケットの到着間隔の計測は packet train の送信側ではなく受信側で行なった．送信側で到着間隔の計測を行なうためには，packet train のパケットを受信し次第，送信側にパケットを送り返さなければならない．しかし，アプリケーション層において UDP パケットの送受信を行なうと，1ms 〜 3ms の時間を要した．よって，受信側での計測は誤

```
struct socket_header {
        u_int32_t sock_sequence;
        u_int32_t data_length;
};

struct ebuf {
        struct socket_header sh;
        char *buf;
        TAILQ_ENTRY(ebuf) f_link;
};
```

図 4.6: **ebuf**{} **構造体**

差が大きくなり過ぎてしまうので，本実装では一方向での計測とした．本実装での packet train 方式のパケット数は 5 とし，1 パケットの大きさは 1400 バイトとした．

RTT の計測

RTT の計測には ping プログラムと同じように ICMP を利用し計測した．

計測した RTT と利用可能帯域はそれぞれ，図 4.5 に示した dev_ip 構造体のメンバ smoothed_rtt，smoothed_capacity に格納することにより，送信データスケジュール機能から参照できるようにしている．格納する際 RTT は 3.6 節にて示した平滑化手法と系数を用いて格納するが，利用可能帯域は，最初の数回の結果が大きく揺れるため，図 4.7 に示す疑似コードに従って平滑化を行なう．最初の結果は無視し，4 回目までの結果は半分ずつ smoothed_capacity に反映していく．5 回目以降の結果は，smoothed_capacity との比較を行ない，挙動を変化させる．もし，結果が smoothed_capacity の 1.5 倍以下ならば RTT と同じように平滑化する．そうでなければ，結果は 1% しか反映しない．

```
struct dev_ip *dip;

if (first probing) {
        dip->smoothed_capacity = 0;
} else if (second probing) {
        dip->smoothed_capacity = probed_capacity;
} else if (until 4th probing) {
        dip->smoothed_capacity
          = (dip->smoothed_capacity + probed_capacity) / 2;
} else {
        if (probed_capacity < smoothed_capacity * 1.5) {
              smoothing same as RTT
        } else {
              dip->smoothed_capacity
                = dip->smoothed_capacity * 0.99
                  + probed_capacity * 0.01;
        }
}
```

図 4.7: **利用可能帯域平滑化疑似コード**

4.2.4 送信データスケジュール機能

設計で述べた通り，送信データスケジュール機能は，複数存在するリンクに対して，どのような割合でデータを投げれば良いかを決定する機能である．本実装では，各リンク経由でデータを投げる際に，dev_ip 構造体に格納された smoothed_capacity と smoothed_rtt

をもとに決定する．計算するべき値は二つある．一つは送信開始時において，帯域遅延積の大きなリンクに対して，その差がなくなるように投げるパケット数の決定であり，もう一つは，各リンクから送信するパケット数の比の値の決定である．以下，それぞれの値の決定部分の疑似コードを示す．

送信開始時の送信パケット数の決定方法

図 4.8 に，送信開始時に投げるパケット数の決定方法を示す．まず，TAILQ_FOREACH ブロック内にでもっとも大きな帯域遅延積を持つリンクを検索する．TAILQ_FOREACH は dev_ip 構造体の双方向連結リストを線形探索するためのマクロである．次に for ブロックにおいて，帯域遅延積を送信パケットサイズで割った数だけ，データの送信を行なう．ここで，送信パケットサイズに足している 28 とは，IP ヘッダの 20 バイトと UDP ヘッダの 8 バイトである．

```
int bdp, big_bdp;
struct dev_ip *dip;
TAILQ_FOREACH(dip, tqp_dvip, f_link) {
        bdp = (dip->smoothed_capacity*1000/8)
                * (dip->smoothed_rtt);

        if (big_bdp < bdp) {
                big_bdp = bdp;
                break;
        }
}

for(i=0; i<big_bdp/packet_size+28; i++)
        send data to dip->ip_addr;
```

図 4.8: **送信開始時の送信パケット数の決定方法**

各リンクへの送信数の比の値の決定方法

送信開始時の処理が終った後は，各リンクより帯域に比例したデータ量を送信する．この時，各リンクより送信するパケット数は全てのリンクの利用可能帯域の比を使用して決定される．リンクの数が 2 つの場合の決定方法を図 4.9 の疑似コードを利用し説明する．

まず，dev_ip 構造体のメンバ double smoothed_capacity の値を整数値に丸め，その二つの値の最大公約数 int gcd_value を求める．次に，丸めた smoothed_capacity を gcd_value で割り，それぞれのリンクの比を求める．この時点で比が 1:1 ならば計算は終了するが，そうでない場合は片方の比が必ず 1 になるようにさらに値を丸める作業を行なう．この作業を行なわない場合，例えば 2:7 のような値が取得された場合に，片方のリンクから

```
redo:
  smoothed_capacities are rounded in int;
  gcd_value = gcd(smoothed_capacity1, smoothed_capacity2);
  link_proportion1 = smoothed_capacity1/gcd_value;
  link_proportion2 = smoothed_capacity2/gcd_value;

  if (link_proportion1 == link_proportion2)
        proportion is 1:1;

  if (not yet redone) {
        tmp = (int)(((float)link_proportion1
                    / link_proportion2) + 0.5);
        tmp = tmp * link_proportion2;
        link_proportion1 = tmp;
        goto redo;
  }
```

図 4.9: **リンク数が 2 つの場合の比の決定方法**

7 パケット送信するまでもう片方のリンクからデータを送信しないことになり，帯域を有効に利用できない．

4.2.5 送信データ分割機能

ABAM 実装では，リンク間の MTU のサイズと送信するパケットサイズを同じ大きさにしている．よって，送信パケットサイズから IP ヘッダの 20 バイト，UDP ヘッダの 8 バイト，ソケットヘッダの 8 バイトを引いただけのデータを各リンクより送信している．

4.2.6 受信データ統合機能

sbam_client では，受信したデータを図 4.6 に示した ebuf 構造体に格納していく．そして，他のアプリケーションにデータを渡す時に，socket_header 構造体に収められている，シーケンス番号が順番になるように渡していく．

4.3 本章のまとめ

本章では，カーネル内部のソケット層において，送信データ分割機能と受信データ統合機能を有するプロトタイプ実装と，ネットワークモニタリング機能と送信データスケジュール機能の有効性を確認するための，ネットワークモニタリング機能，送信データスケジュール機能，受信データ統合機能を有する評価用アプリケーション実装について述べた．次章では本章で述べた実装を用いて評価実験を行なう．

第5章

評価

本章では，SBAM の評価について述べる．カーネル内で実装した送信データスケジュール機能，送信データ分割機能，受信データ統合機能を持った SBAM と，アプリケーションで実装し，カーネル内実装に加えてネットワークモニタリング機能を実装した ABAM を利用し，以下の項目について評価した．

- スループット
- 到着パケットシーケンス
- 受信ホストにおけるキューの長さ

5.1 実験環境

本研究では，図 5.1 と図 5.2 で表される実験ネットワーク 1 と実験ネットワーク 2 を利用して評価を行なった．

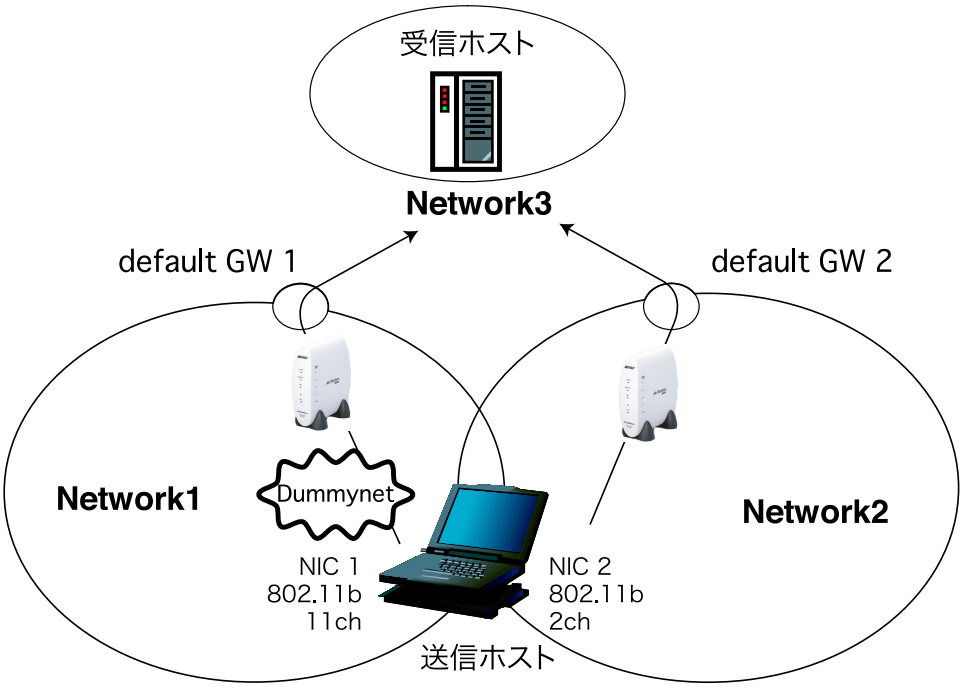

図 5.1: **実験ネットワーク 1 の構成図**

図 5.1 の実験ネットワーク 1 では，送信ホストに 2 枚の IEEE 802.11b の N/I が装着されており，それぞれ別のネットワーク，Network1 と Network2 に別の無線チャネル 2, 11 で接続されている．また，受信ホストはさらに別のネットワークに接続されており，Ethernet で接続されている．

図 5.2 で表される実験ネットワーク 2 では，送信ホストに 2 枚の IEEE 802.11b の N/I が装着されており，それぞれ別のネットワーク，Network1，Network2 に無線チャネル 8, 11 で接続している．また，受信ホストは Network1 に Ethernet で接続されている．

5.2 スループット

N/I を 2 つ利用することにより，データ送信のスループットが向上することを確かめるために，以下のような実験を行なった．図 5.1 に示される実験ネットワーク 1 を利用し，送信ホスト，受信ホストのそれぞれに SBAM 機構を実装した FreeBSD をインストールした．トランスポートプロトコルとして UDP を利用し，送信ホストより，Network1, Network2 に対して交互にデータを送信した．また，比較のために，アプリケーション層において SBAM と同等の機能を持つ APP1 を実装し，スループットの比較を行なった．また，N/I が 1 枚の場合のスループットも APP1 を利用して計測した．APP1 は raw socket

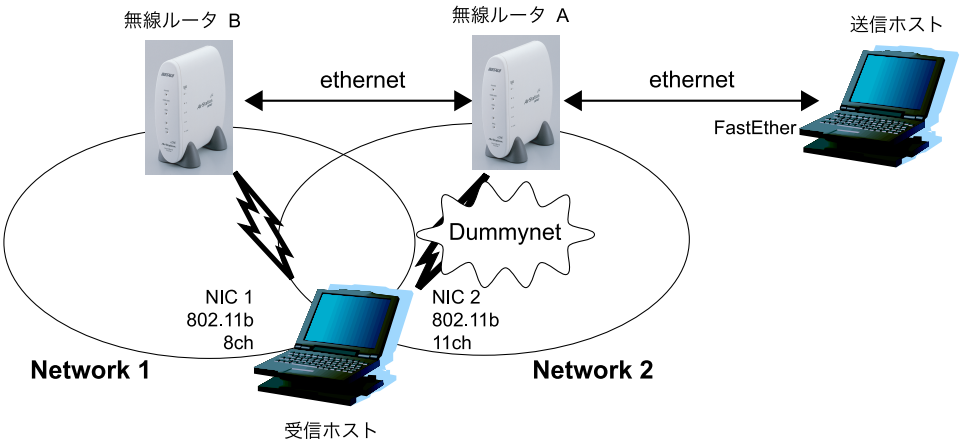

図 5.2: **実験ネットワーク 2 の構成図**

を利用し指定されたデフォルトゲートウェイへの変更，UDP データグラムの送信を行なうアプリケーションである．C 言語のソースコードを Appendix A.1 に掲載した．APP2 はソケットプログラミングを用いた UDP データグラム送信プログラムである．C 言語のソースコードを Appendix B.1 に掲載した．スループットの計測には 5.5MB のメディアデータを 50 回送信し，その計測結果の平均をとった．結果を表 5.1 に示す．

表 5.1: SBAM と評価用アプリケーションのスループットの比較

N/I	実装	スループット
11Mbps + 11Mbps	SBAM	15.0 Mbps
11Mbps + 11Mbps	APP1	15.4 Mbps
11Mbps	APP2	9.2 Mbps
5.5Mbps	APP2	6.1 Mbps

11Mbps の N/I を 1 枚利用した場合と比べ，11Mbps の N/I を 2 枚利用した APP1 も SBAM も約 1.6 倍程度スループットが向上していることが分かる．APP1 と SBAM を比較すると，SBAM のスループットは APP1 に対して 97.4% となっていることが分かる．SBAM では APP1 と比べて，利用可能インタフェースや現在のネットワークトポロジのチェック，ヘッダの付与を行なっているので，APP1 よりも処理が多くなっている．このオーバヘッドの原因を調べるために，N/I を 2 枚利用した場合の sbam_send 関数と，N/I が 1 枚の場合の sbam_send 関数，そして，Pentium Counter を利用して rtalloc 関数の実行時間を計測した．N/I が 1 枚の場合，sbam_send 関数はリンクアップしている N/I の数を調べるために線形探索を 1 度行ない，udp_send 関数を呼び出す．rtalloc 関数とは，ルーティングテーブルを検索するための関数であり，sbam_send 関数の中で 2 回呼び出されている．計測は 1400 バイトのパケットを 50 個送信し，1 パケット毎に sbam_send 関数と rtalloc 関数の実行時間を計測した．その平均結果を図 5.3 に示す．縦軸が実行時間を

各関数の実行時間を表し，N/I 1 枚の場合と N/I 2 枚の場合についてのグラフとなっている．N/I が 2 枚の場合，sbam_send 関数は N/I が 1 枚の場合と比べ 98.1 倍のオーバヘッドが存在することが分かる．また，N/I が 2 枚の場合のオーバヘッドの内，47.7% が rtalloc 関数のオーバヘッドとなっている．よって，SBAM 機構に特化した rtalloc 関数の代替を考案することで，スループット改善の余地があると考えられる．

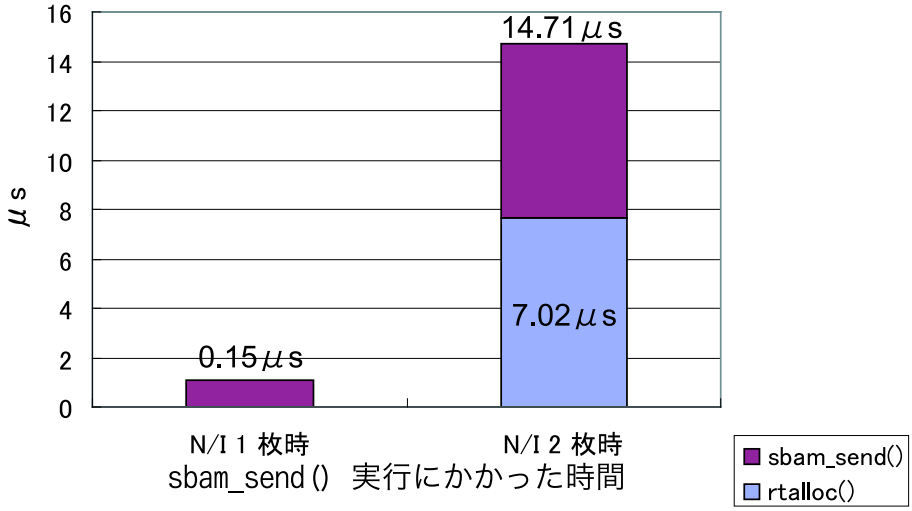

図 5.3: **各関数の実行時間**

実装内容について比較すると，APP1 では raw socket の利用，指定されたデフォルトゲートウェイへの変更，UDP データグラムの送信の 3 機能をアプリケーションにおいて全て実装しなければならず，C 言語で実装したプログラムの行数は 666 行に達した．しかし SBAM では既存のソケットプログラミングによる UDP 送信手法で同等の機能を実現でき，プログラムの行数は 206 行であった．それぞれ Appendix A.1，Appendix B.1 にプログラムのソースコードを掲載した．このことから，アプリケーション層における実装と比べて同等の機能を実現するのが容易であることが分かる．

送信データスケジューリング機能の有無

式 (3.1)，(3.2) に基づいた送信データスケジューリング機能を実装した APP1 を利用し，送信データスケジュール機能がある場合とない場合におけるスループットの比較を行なった．機能送信データスケジュール機能に必要な帯域情報と遅延情報はあらかじめ計測した値を利用し，MTU は 1500 バイトで統一した．また，N/I の重み α も 1 で統一した．結果を結果を表 5.2 に示す．

各リンクの帯域が異なる場合，スケジューリング機能のない APP1 では，11Mbps と 5.5Mbps の N/I のスループットの和と比べて，62.7% のスループットとなっている．これに対し，帯域に比例したデータ送信を行なう APP1 では，76.5% のスループットとなっ

表 5.2: スケジューリングの有無によるスループットの比較

スケジューリング機能の有無	N/I	スループット
なし	11Mbps + 5.5Mbps	9.6 Mbps
あり	11Mbps + 5.5Mbps	11.7 Mbps

表 5.3: 実験環境その 1

	NIC1	NIC2
環境 1	11Mbps, 遅延 0ms.	11Mbps, 遅延 0ms
環境 2	11Mbps, 遅延 50ms	11Mbps, 遅延 50ms
環境 3	11Mbps, 遅延 0ms	5.5Mbps, 遅延 0ms

ている．これは，帯域に比例したデータ送信を行なわない場合，速いスループットを確保できるリンク - この場合 11Mbps のリンク - が遅いスループットしか確保できないリンク - この場合は 5.5Mbps のリンク - と同量のデータ送信しか行なえないからだと考えられる．このことから，リンクの帯域に応じたデータ送信を行なうことで，各リンクの帯域を効率的に利用できることが分かった．

5.3 到着パケットシーケンスについて

5.3.1 ネットワークの状態と到着パケットシーケンスの関係について

ネットワーク環境の変化が受信ホストにおける到着パケットシーケンス番号にどのような影響を与えるかについて確認するための実験を行なった．表 5.3 に今回行なった実験環境を示す．環境 1 は各リンクの状態を帯域 11Mbps，遅延 0ms と同等とした．環境 2 は片方のリンクに 50ms の遅延を発生させ，遅延が異なる環境とした．遅延は Dummynet [14] を Network 1 のリンク上に設置し実現している．環境 3 は片方の IEEE 802.11b N/I を 5.5Mbps モードで動作させて，帯域が異なる環境とした．NIC1，NIC2 はそれぞれ図 5.1 に対応している．実験は全て SBAM を用いて行なった．

各実験とも，5.5MB のメディアデータを各 N/I に対しラウンドロビンで送信した．UDP パケットには，送信ホスト上で，それぞれの NIC において通し番号となるような 4 バイトのシーケンス番号と，4 バイトのデータ部分のデータ長を付与した．パケット到着時間は受信ホスト上において，Pentium Counter を利用して計測した．それぞれの環境における実験結果を図 5.4 〜 5.6 に示す．各図の縦軸は，送信ホストで付与されたパケットシーケンス番号を表し，横軸は送信開始からの経過時間を示す．

図 5.4 より，2 枚の N/I に大きな帯域差がなく，遅延もないネットワークにおいて，デー

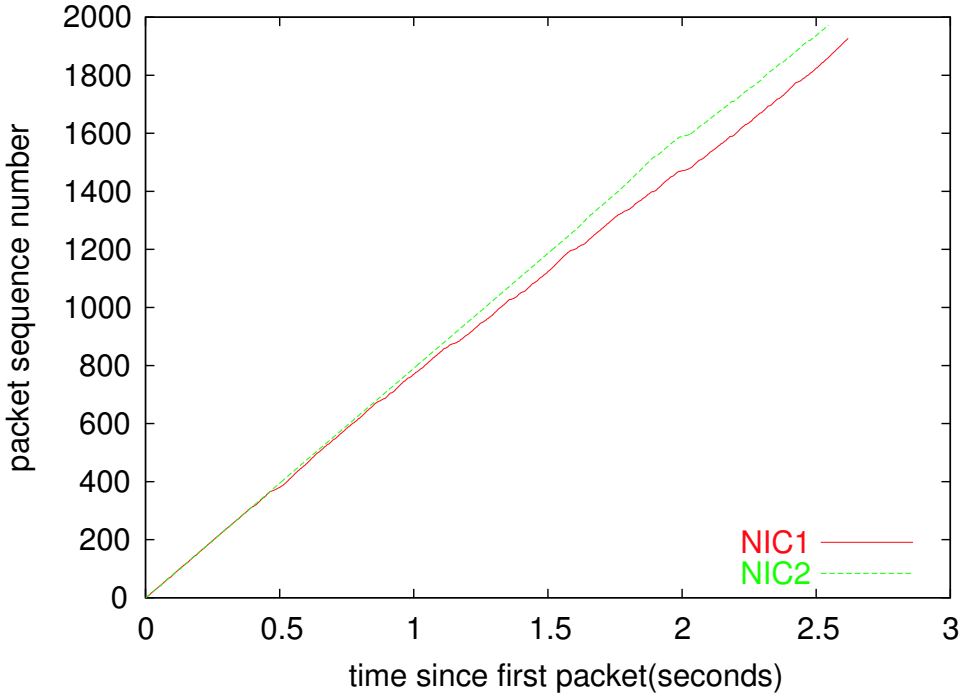

図 5.4: **NIC1 (11Mbps,0ms), NIC2 (11Mbps,0ms)**

タ送信後 1 秒以降，パケットシーケンス番号のずれが目立ちはじめていることが分かる．データ送信開始後 1 秒まではパケットシーケンス番号のずれはほとんど見られない．データ送信開始後 1.7 秒の時点で，100 パケット程度のずれが見られるようになる．しかし，遅延が 50ms ある環境では，送信当初よりパケットシーケンス番号のずれが見られ，データ送信開始後 1.5 秒には 100 パケット程度のずれが生じていることが図 5.5 より分かる．また，NIC 1 の帯域を 11Mbps，NIC 2 の帯域を 5.5Mbps とした実験では，データ送信開始後 0.5 秒後には，100 パケット程度のずれが生じていることが図 5.6 より分かる．

これらの実験より，送信/受信ホスト間の帯域と遅延がパケットの到着順に影響を与えることが判明した．到着したデータをアプリケーションに渡す前に並べ替えない場合，データを構築できないという問題が発生することがわかる．例えば映像ストリーミングを行なっている場合，パケットロスが起きたら，そのパケットを含むフレームを破棄するだけで良いが，結果のように到着パケットシーケンスがずれた場合，フレームを全く構成できない．この実験の結果より，ネットワークモニタリング機能を用いてホスト間の帯域・遅延を計測し，その値に基づいてデータを送信する送信データスケジュール機能が必要なことがわかる．次節において，ネットワークモニタリング機能とその値を基にした送信データスケジュール機能，そして受信データ統合機能を実装したアプリケーション ABAM を用いて，実験を行なった結果を示す．

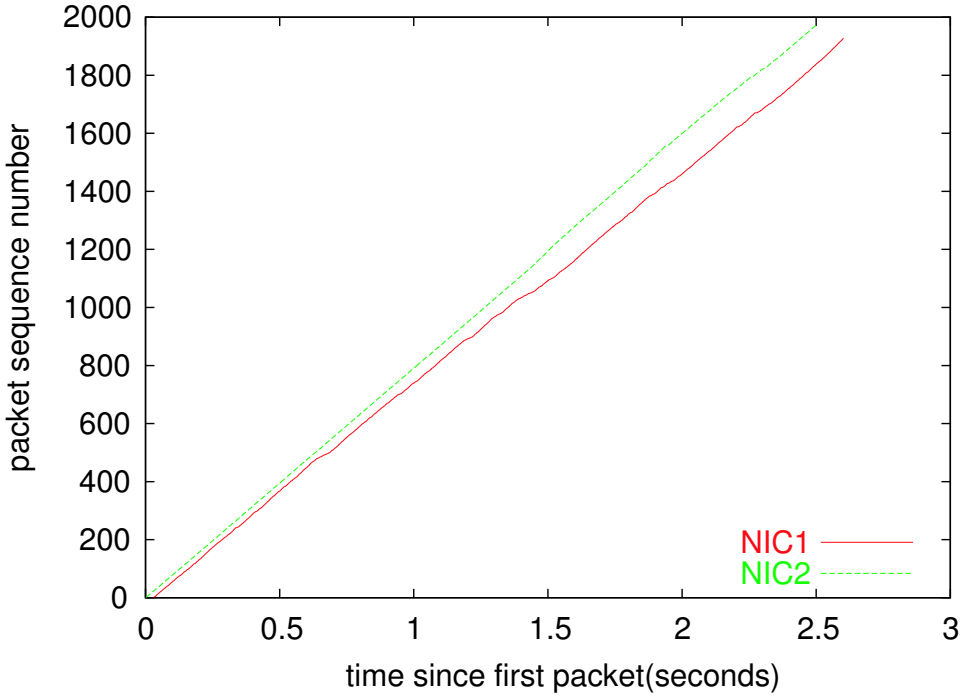

図 5.5: **NIC1 (11Mbps,50ms), NIC2 (11Mbps,0ms)**

表 5.4: 実験環境その 2

	packet train の有無	遅延計測の有無	リンク 1	リンク 2
環境 1	有	無	11Mbps, 遅延 0ms	2Mbps, 遅延 0ms
環境 2	有	無	11Mbps, 遅延 0ms	4Mbps, 遅延 0ms
環境 3	有	無	11Mbps, 遅延 0ms	2Mbps, 遅延 25ms
環境 4	有	有	11Mbps, 遅延 0ms	2Mbps, 遅延 25ms

5.3.2 ネットワークモニタリング機能とパケット到着シーケンスの関係について

packet train を利用したネットワークモニタリング機能を実装したアプリケーション実装を用いて，到着パケットシーケンスがどのように変化するかを確認するための実験を行なった．実験ネットワーク環境は 5.1 節で述べた図 5.2 で表される実験ネットワーク 2 を利用した．表 5.4 に今回行なった実験環境を示す．

環境 1，2 では packet train による利用可能帯域の計測を行ないつつ，リンク 2 の途中経路において Dummynet を利用し帯域をそれぞれ 2 Mbps，4 Mbps にシェーピングした．環境 3 では packet train による計測をし，リンク 2 において Dummynet を用いて遅延を発生させた．環境 4 では packet train による計測をしつつ，遅延の計測も行ない，リンク 2 において環境 3 と同様に遅延を発生させた．それぞれの環境で MTU は 1500 バイトと

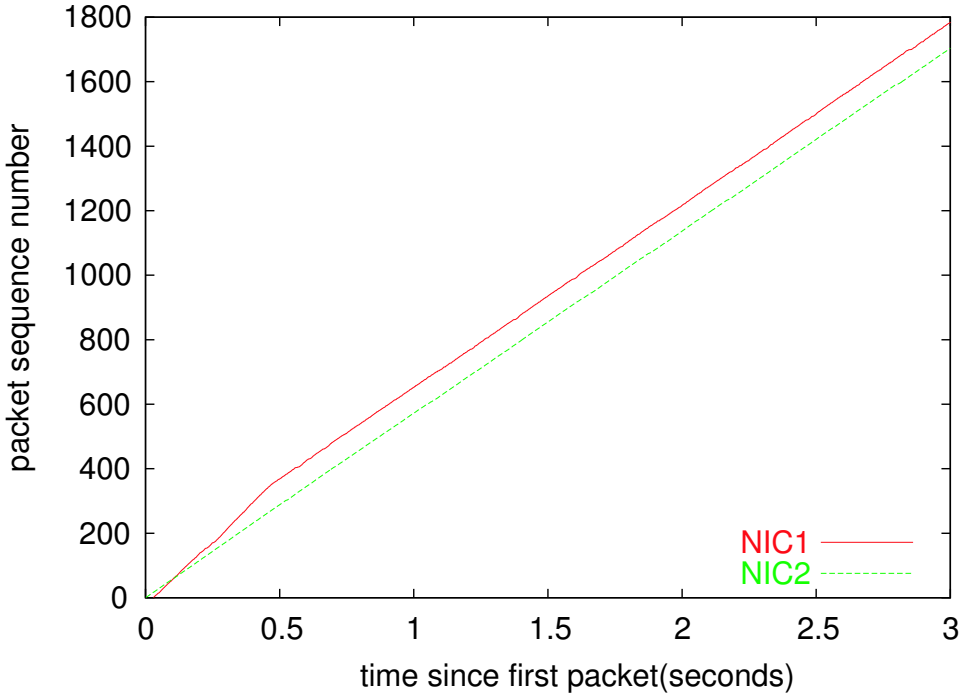

図 5.6: **NIC1 (11Mbps,0ms), NIC2 (5.5Mbps,0ms)**

した．

実験はそれぞれの環境において，27M バイトのマルチメディアデータを，1 パケットあたり 1400 バイトの UDP パケットを利用して送信し，受信ホストにおける到着パケットシーケンスがどのように延びて行くかについて計測を行なった．シーケンス番号はパケットの内部に SBAM と同様，8 バイトの socket_header 構造体を埋め込み，リンク中において一意なシーケンス番号を利用した．また，時間は Pentium Counter を受信毎に利用し測定した．それぞれの環境における実験結果を図 5.7 と図 5.8 に示す．各図の縦軸は，送信ホストにおいて付与されたパケットシーケンス番号を表し，横軸は受信ホストにおける受信開始からの経過時間を示す．

図 5.7 と図 5.8 より，2 つのリンクの帯域が異なる場合でも，ネットワークモニタリング機能から取得された値を利用し送信データをスケジュールすることにより，シーケンス番号がずれることなく，受信ホストに到着していることが分かる．また，Dummynet によって帯域をシェーピングしているリンクとしていないリンクのシーケンス番号の伸びが，帯域値の比と同等になっていることが分かる．このことから，packet train による利用可能帯域の計測が，ネットワークモニタリングを行なう上で有効であることも分かる．

次に，式 3.1 の有効性を確かめるために，環境 3 と環境 4 において，通信開始時に図 4.8 に示した帯域遅延積差を帯域遅延積の小さなリンクに対して先に送信するという実験を行なった．計測は，到着パケットシーケンスの伸びを確かめる実験と同じくそれぞれの環境において，27M バイトのマルチメディアデータを 1 パケットあたり 1400 バイトの UDP データを利用して送信し，受信ホストにおいてデータを並び換える際に必要なキューの長さを計測した．キューはそれぞれの N/I 毎に用意し，長さは 0.1 秒毎に計

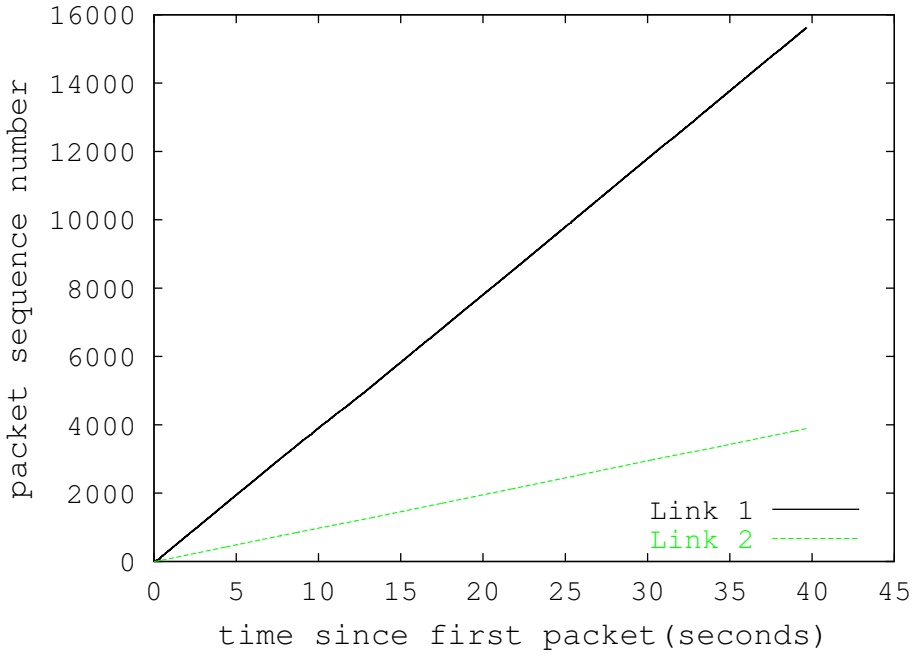

図 5.7: **到着シーケンスの伸び Link1 (11Mbps), Link2 (2Mbps)**

測した.. 図 5.9 と図 5.10 に実験結果を示す．縦軸はキューの長さを表し，横軸は受信開始時からの時間を表す．キューの長さの単位はパケットの数とする．図中の **on Link1** と **on Link2** それぞれ Link1 と Link2 から到着したパケットのキューの長さを表す.

それぞれの図より，遅延の小さなリンク 1 は遅延の大きなリンク 2 のパケットを待ち，リンク 2 の約 3 倍のキューを持たなければならないことが分かる．また，キューの長さが短くなることは少なく，少しずつ多くなっていることも分かる．例えば図 5.9 のリンク 1 では通信開始後 2 秒までのキューの長さは 27 パケット程度であるが，3 秒からは 30 パケットまで増え，キューの長さが短くなることはない．また，26 秒の時点で再度キューの長さが 40 パケットまで増加している．しかし，最後の通信において 2 パケット程落ちていることが分かる．このことから，送信ホストに対してキューの長さを補正するために，何らかのフィードバックをかける必要があることがわかる.

通信開始後 3 秒までの図 5.9 と図 5.10 を見比べると，帯域遅延差の送信がキューの長さに影響を及ぼしていないことが分かる.

そこで図 5.11 と図 5.12 に，通信開始から 100 パケット分のシーケンス番号の伸びを示す．図中の **Link1**，**Link2** はそれぞれ Link 1，Link 2 から到着したパケットを表す．縦軸はシーケンス番号を表し，横軸は通信開始時から計測したパケット到着時間を表す．遅延計測を行なっていない場合，図 5.11 より，Link 2 からのパケット到着は 0.025 秒となっているのに対し，遅延計測を行なっている場合，図 5.12 より，Link 2 からのパケット到着は 0.001 秒となっている．このことより，効率良くデータ送信が行なわれているように見える．しかし，通信開始時に Link 2 に対し数パケット送信することにより，Link 1 に最初に到着するパケットシーケンス番号が 11 からとなっていることが分かる．Link

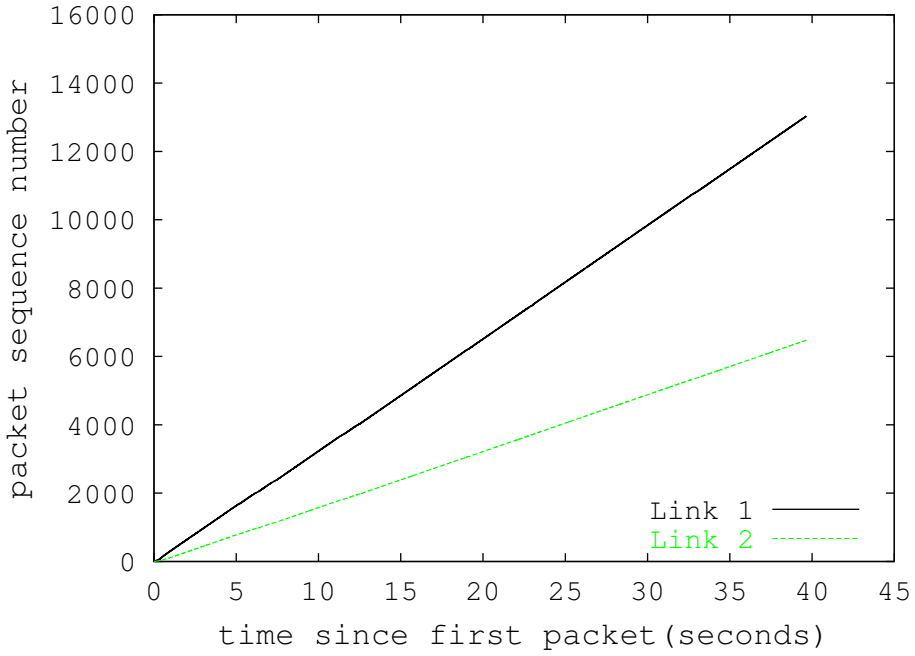

図 5.8: **到着シーケンスの伸び Link1 (11Mbps), Link2 (4Mbps)**

2 からシーケンス番号 10 が到着するのが 0.06 秒となっているので，Link 1 からのパケットはそれまでキューに貯められている．よって，遅延計測を行なっている図 5.10 の方が遅延計測を行なっていない図 5.9 よりもキューが長くなる結果になった．これは次のような原因に基づく現象だと考えられる．まず，本実験では遅延を図 5.2 で表されている Link 2 の途中経路にある Dummynet を用いて発生させている．次に，Dummynet ホストは，送信ホストが Link 2 に対して帯域遅延積差分送信したデータをバッファする．そして，そのバッファが吐き出される前，つまり 25ms 経過することを待たずに送信ホストにおいて Link 1 に対してデータ送信が開始される．このデータが Link 2 に対して送信されたデータよりも先に受信ホストに到着する．よって，受信ホストにおいてキューが長くなってしまう．

このことから，式 3.1 のデータ量の送信手法はこのまま利用しつつ，シーケンス番号を考慮した送信開始時のデータ送信を行なう必要性があることが分かった．例えば，遅延のあるリンクに対して送信するデータをシーケンス番号 1 からではなく 10 から開始する，などの対応が必要であると考えられる．

5.4 本章のまとめ

本章では，送信データスケジュール機能と送信データ分割機能，受信データ統合機能を有する SBAM のプロトタイプ実装と，SBAM プロトタイプ実装に加えてネットワークモニタリング機能を有するアプリケーション層で実装された ABAM を用いて評価を行なった．評価は SBAM を用いたスループットの向上の確認とネットワークモニタリング

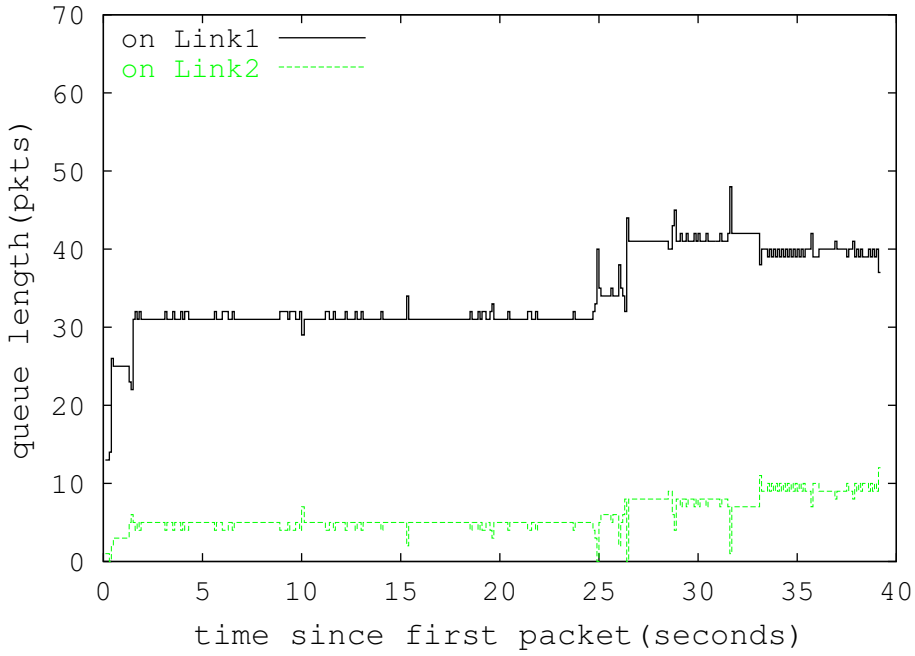

図 5.9: **Link1 (11Mbps,0ms), Link2 (2Mbps,25ms) 遅延計測無**

機能の有無による到着パケットシーケンスの変化と受信ホストにおけるキューの長さについて行なった.

2 枚の N/I を用いたスループットを評価する実験では，1 枚の N/I を利用した場合と比べて約 1.6 倍のスループット向上が確認できた．また，ABAM と比較するとオーバヘッドが確認された.

ABAM を用いたネットワークモニタリング機能の有無による到着パケットシーケンスの変化に対する実験では，ネットワークモニタリング機能を実装しない場合，各リンクの利用可能帯域と遅延によって到着パケットシーケンスが大きくずれてしまうことが分かった．また，ネットワークモニタリング機能を利用した場合，各リンクにおいて利用可能帯域が異なったとしても到着パケットシーケンスのずれの防止を確認できた．しかし，遅延が異なる場合は，現在のアルゴリズムでは受信ホストにおいてキューが長くなってしまうことが分かった.

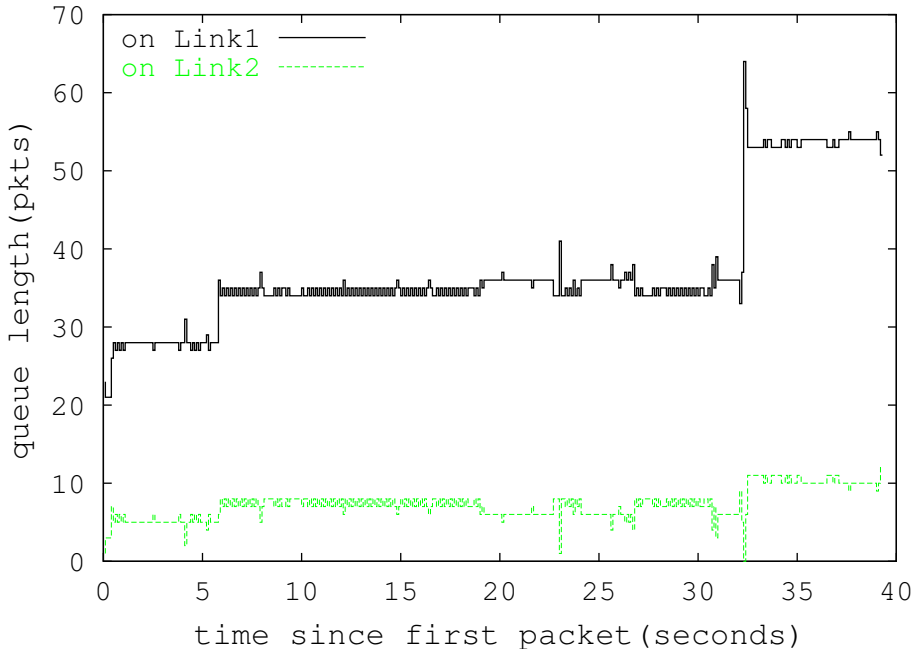

図 5.10: **Link1 (11Mbps,0ms), Link2 (2Mbps,25ms) 遅延計測有**

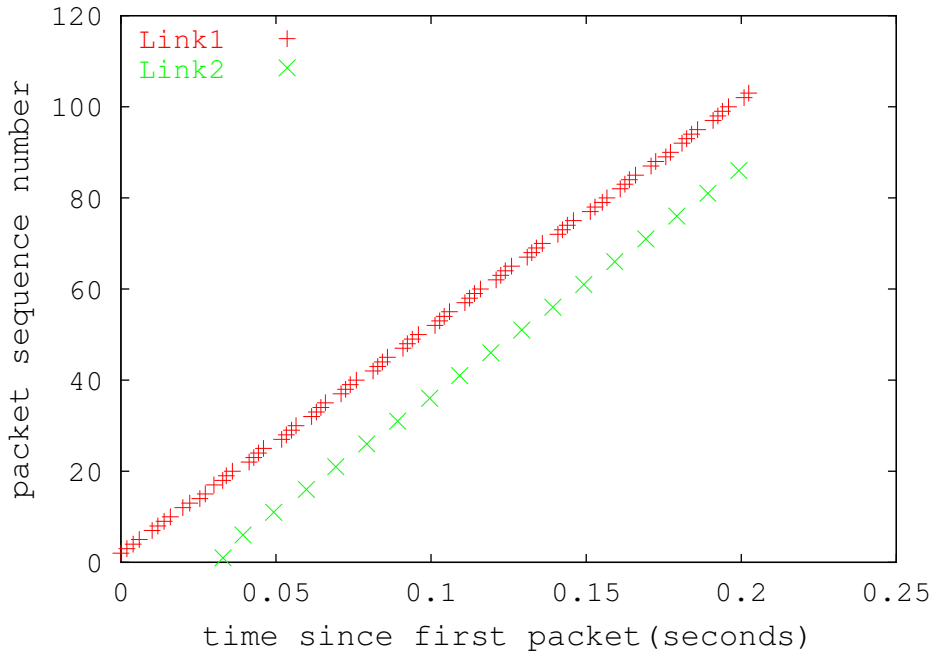

図 5.11: **到着シーケンス番号 Link1 (11Mbps,0ms), Link2 (2Mbps,25ms) 遅延計測無**

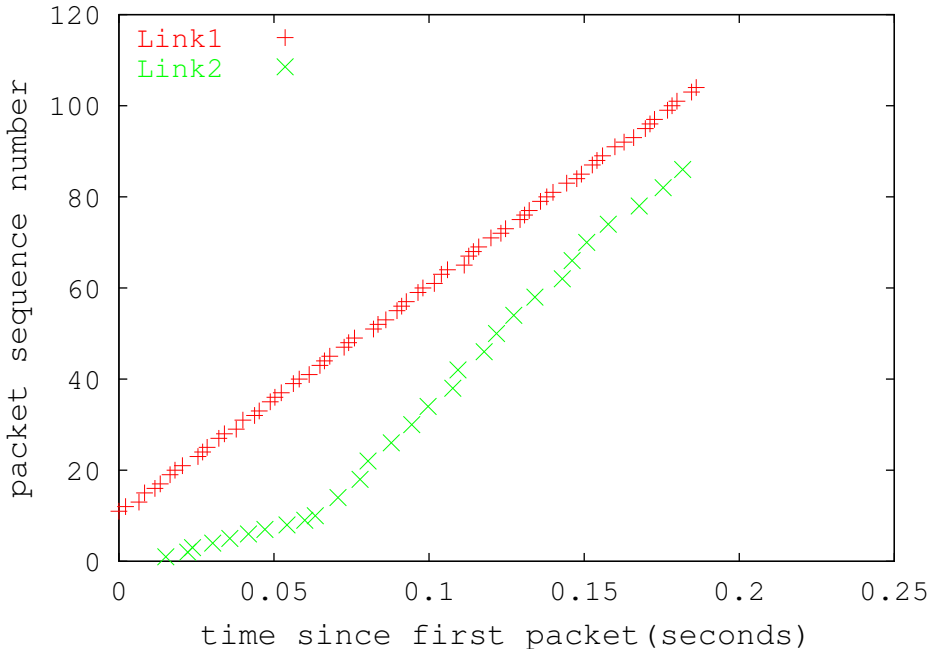

図 5.12: **到着シーケンス番号 Link1 (11Mbps,0ms), Link2 (2Mbps,25ms) 遅延計測有**

第6章

まとめと今後の課題

本章では本論文をまとめ，今後の課題について述べる．

本論文では ソケット層における帯域統合機構である SBAM を提案し，プロトタイプ実装を用いて評価を行った．SBAM は ソケット層において，送信データスケジュール機能，ポリシー伝達機能，送信データ分割機能，ネットワーク状態モニタリング機能，受信データ統合機能を持ち，複数 N/I の帯域を利用できる．本論文では SBAM の設計・実装を述べ，実際に SBAM を用いて複数 N/I を利用しデータ送信を行なった．

評価実験では SBAM を用い，IEEE 802.11b の N/I を 2 枚利用し，UDP のスループットをアプリケーション実装 APP1 と比較した．また，送信ホストにおける到着パケットシーケンスについて評価を行なった．

ソケット層で実装されている SBAM とアプリケーション実装 APP1 のスループット比較では，約 2.6% のスループットの差があることが分かった．しかし，ソケット層で実現しているので同等の機能を持つプログラムを容易に実装できることも分かった．到着パケットシーケンスに関して，11Mbps と 5.5Mbps のリンクよりデータを分割して送信した場合，送信開始後 $1.5s$ の時点で 8% のずれが生じることと，相手先ホストまでの各 N/I 間の遅延の差が $50ms$ の場合，送信開始後 $1.5s$ の時点で約 8% ずれることを観測した．

ABAM を用いたネットワークモニタリング機能の有無による，到着パケットシーケンスの変化に対する実験では，ネットワークモニタリング機能を実装しない場合，各リンクの利用可能ネットワーク帯域と遅延によって到着パケットシーケンスが大きくずれてしまうことが分かった．また，ネットワークモニタリング機能を用いた場合，各リンクにおいて利用可能帯域が異なっても，到着パケットシーケンスのずれの防止を確認できた．しかし，遅延が異なる場合は現在のアルゴリズムでは受信ホストにおけるキューが長くなってしまうことが分かった．

今後の課題として，以下が挙げられる．

- 各機能のカーネル内での実装
  カーネル内において未実装である，送信データスケジュール機能，ネットワークモニタリング機能，ポリシー伝達機能を実装する必要がある．

- アドレス解決
  今回のプロトタイプ実装ではアドレスについての考慮をしていないが，送信先ホストにおいて受信データが同一アドレスから到着しているように見える機構の導入を行なう．

- 他のトランスポートプロトコルでの評価
  今回のプロトタイプ実装ではネットワーク状態の取得，アドレス解決機構が導入されていないので，UDP を用いて実験を行なった．しかし，TCP のような信頼性のあるプロトコルでの評価を行ない，再送機構，輻輳回避機構と SBAM がどのような相関を示すかを評価する必要がある．

- 3 枚以上の N/I を利用した評価
  SBAM の N/I の枚数に対するスケーラビリティを評価し，アプリケーション層における実装に対する優位性を示す必要がある．

- ポリシーエディタの開発
  ポリシーエディタとは，ユーザやプロセスごとに変化するネットワークの利用要求を SBAM 機構に伝えるためのユーザアプリケーションである．ここで設定された値が SBAM のポリシー伝達機能を通して SBAM に伝えられ，指定した N/I を利用状況を変化させる．ユーザポリシを SBAM に伝えるには，ポリシーエディタの開発が必要であると考えられる．

- リンク間の遅延が異なる場合，現在のアルゴリズムでは受信ホストにおけるキューの長さの改善が見られない．今後，送信するパケットシーケンス番号も考慮に入れた送信アルゴリズムを提案する必要がある．

参照論文

- 榊原寛, 守分滋, 斉藤匡人, 徳田英幸.
  “複数ネットワークインタフェースの同時利用機構, ” 日本ソフトウェア科学会 第 3 回 SPA サマーワークショップ ポスターセッション,
  Aug. 2004.
- 榊原寛, 守分滋, 斉藤匡人, 徳田英幸.
  “SBAM: ソケット層における帯域統合機構, ” 情報処理学会 第 12 回 マルチメディア通信と分散処理 (DPS) ワークショップ, Vol. 2004, No.15 pp. 49-54,
  Dec. 2004.

謝辞

研究する機会を与えて頂き御指導を頂きました，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，研究面と共に +精神面+ のサポートをして頂いた，慶應義塾大学大学院政策・メディア研究科の守分滋氏に感謝致します．新天地の就職先でも自分のペースを崩すことなく頑張って下さい．

貴重な御助言を数多く頂きました，慶應義塾大学大学院政策・メディア研究科の斉藤匡人氏及び，NTT DoCoMo ネットワーク研究所の永田智大博士に感謝の意を表します．

また，LaTeX に関して助言を頂きました，慶應義塾大学大学院政策・メディア研究科の由良淳一氏に感謝致します．おかげできれいな C 言語のソースを掲載できました．

慶應義塾大学徳田研究室 ACE グループの出内将夫氏には，様々な助言に加えその体に秘めたるパワーを頂きました．ここに感謝の意を表します．

沼田行雄氏，中井彦一朗氏，本多倫夫氏，船木康平氏を始めとする，慶應義塾大学徳田研究室 move! 研究グループの諸氏，また徳田・村井・楠本・中村・南研究室の方々には，研究会の活動を通じて多くの御意見，御助言を頂きましたこと拝謝します．

最後に，研究の日々を共に過ごした move! 研究グループの小泉健吾氏，米澤拓郎氏を始めとする数多くの友人に感謝し，謝辞と致します．

2005 年 1 月 22 日
榊原 寛

参考文献

[1] BLUETOOTH. http://www.bluetooth.com/.

[2] DOVROLIS, C., RAMANATHAN, P., AND MOORE, D. What do packet dispersion techniques measure? In *INFOCOM* (2001), pp. 905–914.

[3] @FREED. http://www.at-freed.com/.

[4] H", A. http://www.ddipocket.co.jp/p_s/service/air_h/index.html.

[5] H. SIVAKUMAR, S. B., AND GROSSMAN, R. Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks. In *IEEE Supercomputing (SC)* (Nov. 2000).

[6] HSIEH, H.-Y., AND SIVAKUMAR, R. A Transport Layer Approach for Achiving Aggregate Bandwidths on Multi-homed Mobile Hosts. In *Proceedings of ACM MOBICOM'02* (2002).

[7] HUNG-YUN HSIEH, KYU-HAN KIM, Y. Z., AND SIVAKUMAR, R. A Receiver-Centric Transport Protocol for Mobile Hosts with Heterogeneous Wireless Interfaces. In *Proceedings of ACM MOBICOM'03* (2003).

[8] IEEE 802.11 STANDARD (IEEE COMPUTER SOCIETY LAN MAN STANDARDS COMMITTEE). *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: High Speed Physical Layer in the 5 GHz band*, Feb. 2000.

[9] IEEE 802.11 STANDARD (IEEE COMPUTER SOCIETY LAN MAN STANDARDS COMMITTEE). *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Higher speed Physical Layer (PHY) extension in the 2.4 Ghz band*, Feb. 2000.

[10] IEEE 802.3AD STANDARD (IEEE COMPUTER SOCIETY LAN MAN STANDARDS COMMITTEE). *Aggregation of Multiple Link Segments*, 2000.

[11] IEEE STANDARD 802.3AB-1999. *802.3 IEEE Standard for Information technology, Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*, 1999.

[12] IEEE STANDARD 802.3AE-2002. *Amendment: Media Access Control (MAC) Parameters, Physical Layers, and Management Parameters for 10 Gb/s Operation*, Aug. 2002.

[13] K. MCCLOGHRIE, AND M. ROSE. Management Information Base for Network Management of TCP/IP-based internets: MIB-II. RFC 1213, 1991.

[14] LUIGI RIZZO. Dummynet: a Simple Approach to the Evaluation of Network Protocols. In *ACM Computer Communication Review* (1997).

[15] M. ALLMAN, H. K., AND OSTERMANN., S. An application-level solution to tcp's satellite inefficiencies. In *Workshop on Satellite-Based Information Services(WOSBIS)* (Nov. 1996).

[16] M. K. ABDUL AZIZ AND A. R. NIX AND P. N. FLETCHER. *A Study of Performance and Complexity for IEEE 802.11n MIMO-OFDM GIS Solutions*. IEEE Communications Society, 2004.

[17] MAGDALENA BALAZINSKA, AND PAUL CASTRO. Characterizing Mobility and Network Usage in a Corporate Wireless Local-Area Network. In *1st International Conference on Mobile Systems, Applications, and Services (MobiSys)* (San Francisco, USA, May 2003).

[18] POSTEL JOHN. Transmission Control Protocol. RFC 793, September 1981.

[19] S. KESHAV. A Control-Theoretic Approach to Flow Control. *Proceedings of the conference on Communications architecture & protocols* (1991), 3–15.

[20] STEVENS, W. *TCP/IP Illustrated, Volume1 The Protocols*. Addison Wesley, 1994.

[21] T. R. HENDERSON, R. H. H. *Transport Protocols for Internet-Compatible Satellite Networks*, vol. 72 of *IEEE Journal*. Addison-Wesley, Feb 1999.

[22] (UWB)., U. W. http://www.uwb.org/.

[23] V. JACOBSON, R. BRADEN, AND D. BORMAN. TCP Extensions for High Performance. RFC 1323, May 1992.

[24] 慶應義塾大学湘南藤沢キャンパス. http://www.sfc.keio.ac.jp/.

付 録A

APP1 のソースコード

List A.1: APP1 のソースコード

```
#include <sys/socket.h>
#include <sys/time.h>
#include <sys/types.h>

#include <net/route.h>
#include <netinet/in.h>

#include <arpa/inet.h>

#include <errno.h>
#include <fcntl.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

struct link_sched_data {
  char *link_ip;
```

```
  int bandwidth;
  int bandwidth_exp; // value for experiment
  int mtu;
  int mtu_exp; // value for experiment
  int send_count;
  int ref_count; // used by sched();
  int bandwidth_delay_product;
  int pkts_to_send;
  float delay;
  int nobufs;
};

/* function declaration */
struct rt_msghdr * rtm_alloc(char *, int);
void hex(FILE *, char *, int);
int print_route_to(int, char *);
int change_route(int, char *);
void usage(void);
struct rt_msghdr * get_default_gw(int);
int experiment(int, char *, char *, char *, int, int, char *);
int init_udp(int);
int log_start(struct timeval *);
int log_end(struct timeval *);
int log_sub(struct timeval *, struct timeval *, struct timeval *);
int sched(int, struct link_sched_data **);


/* global variable */
int seq; // sequence for query
int opt;

/* definition */
#define CHANGE      0x01
#define PRINT       0x02
#define EXPERIMENT  0x04
#define PORT        0x08
#define TIMES       0x10
#define FILENAME    0x20
#define DSTADDR     0x40
#define SEQ         0x80
```

```
#define D(statement)

int
main(argc, argv)
     int argc;
     char **argv;
{
  int c;
  int route_sock;
  int port;
  int times;
  extern char *optarg;
  char addr[BUFSIZ], addr1[BUFSIZ];
  char dst_addr[BUFSIZ];
  char filename[BUFSIZ];

  /* some initalization */
  if (argc < 2) {
    usage();
    exit(1);
  }

  while((c = getopt(argc, argv, "c:p:e1:2:P:t:f:d:sh")) != -1){
    switch(c){
    case 'c':
      /* change default gateway */
      opt |= CHANGE;
      strncpy(addr, optarg, strlen(optarg));
      break;
    case 'p':
      /* print info to addr */
      opt |= PRINT;
      strncpy(addr, optarg, strlen(optarg));
      break;
    case 'e':
      /* experiment */
      opt |= EXPERIMENT;
      break;
    case '1':
```

```
      strncpy(addr, optarg, strlen(optarg));
      break;
    case '2':
      strncpy(addr1, optarg, strlen(optarg));
      break;
    case 'd':
      /* experiment */
      opt |= DSTADDR;
      strncpy(dst_addr, optarg, strlen(optarg));
      break;
    case 'P':
      /* experiment */
      opt |= PORT;
      port = atoi(optarg);
      break;
    case 't':
      opt |= TIMES;
      times = atoi(optarg);
      break;
    case 'f':
      opt |= FILENAME;
      strncpy(filename, optarg, strlen(optarg));
      break;
    case 's':
      opt |= SEQ;
      break;
    case 'h':
      usage();
      exit(0);
    default:
      usage();
      break;
    }
  }
  if (!(opt & (EXPERIMENT|PORT|TIMES|FILENAME|DSTADDR))) {
    fprintf(stderr,
            "-e, -t, -f, -d and -P must be "
            "specified simultaneously\n\n");
    usage();
    exit(1);
```

```
}

seq = 0;

route_sock = socket(PF_ROUTE, SOCK_RAW, 0);

switch(opt){
case CHANGE:
  if (getuid() != 0) {
    fprintf(stderr, "-c mode should be run as root\n");
    exit(1);
  }
  if (change_route(route_sock, addr) < 0)
    exit(1);
  printf("route changed to %s\n", addr);
  break;
case PRINT:
  print_route_to(route_sock, addr);
  break;
case EXPERIMENT|PORT|TIMES|FILENAME|DSTADDR:
case EXPERIMENT|PORT|TIMES|FILENAME|DSTADDR|SEQ:
  if (experiment(route_sock,
                 addr,
                 addr1,
                 dst_addr,
                 port,
                 times,
                 filename) < 0) {
    fprintf(stderr, "#ERROR: experiment()\n");
    exit(1);
  }
  break;
default:
  /* show all routing table information;
   * not yet implemented */
  ;
}

close(route_sock);
```

```
  return(0);
}


/*
 * experiment(int route_sock, char *addr);
 *  experiment for SBAM
 */
int
experiment(route_sock, gw_addr, gw_addr1, dst_addr, port, times, file)
     int route_sock;
     char *gw_addr;
     char *gw_addr1;
     char *dst_addr;
     int port;
     int times;
     char *file;
{
  int i, j;
  int sock; // for udp socket
  int file_fd;
  int read_num;
  int retry_count = 0;
  int link_num;
  char buf[BUFSIZ*2];
  struct sockaddr_in sin;
  struct hostent *hp;
  struct link_sched_data link1, link2;
  struct link_sched_data *link_data[2];

  /* prepare udp socket */
  sock = init_udp(port);
  if (sock <= 0) {
    fprintf(stderr, "init_udp() error\n");
    exit(1);
  }

  /* file to send */
  file_fd = open(file, O_RDONLY);
  if (file_fd < 0) {
```

```
    perror("#ERROR: experiment(): open()");
    return(-1);
}

/* sockaddr_in for destination host */
if((hp = gethostbyname(dst_addr)) == NULL){
    fprintf(stderr, "%s: unknown host\n", dst_addr);
    return(-1);
}

memset(&sin, 0, sizeof(sin));
sin.sin_family = AF_INET;
sin.sin_port   = htons(port);
memcpy(&(sin.sin_addr), hp->h_addr, hp->h_length);
inet_aton(dst_addr, &sin.sin_addr);

/* set network informaiton */
memset(&link1, 0, sizeof(link1));
memset(&link2, 0, sizeof(link2));
link1.link_ip = gw_addr;
link1.delay = 0.001;
link1.bandwidth = 5910000; // bps (11Mbps)
link1.bandwidth_exp = 3;
link1.mtu = 1400;
//    link1.mtu_exp = 3;
link1.mtu_exp = 1;
link1.pkts_to_send = 3;

link2.link_ip = gw_addr1;
link2.delay = 0.05;
link2.bandwidth = 2640000; // bps (5.5Mbps)
link2.bandwidth_exp = 2; //3;
link2.mtu = 1400;
//    link2.mtu = 1000;
//    link2.mtu_exp = 2;
link2.mtu_exp = 1;
link2.pkts_to_send = 2;

link_num = 2;
link_data[0] = &link1;
```

```
link_data[1] = &link2;

if (link1.delay * link1.bandwidth
    < link2.delay * link2.bandwidth) {
  link1.bandwidth_delay_product
    = (link1.delay * link1.bandwidth
       - link2.delay * link2.bandwidth)
    / (8 * link2.mtu);
  link2.send_count = link2.bandwidth_delay_product;
  link2.ref_count++;
} else if (link2.delay * link2.bandwidth
           < link1.delay * link1.bandwidth) {
  link1.bandwidth_delay_product
    = (link1.delay * link1.bandwidth
       - link2.delay * link2.bandwidth)
    / (8 * link2.mtu);
  link1.send_count = link1.bandwidth_delay_product;
  link1.ref_count++;
}

for (i=0;i<times;i++) {
  int k;
  int packet_sequence;
reread:
  packet_sequence=0;
  if (lseek(file_fd, 0, SEEK_SET) < 0) {
    perror("lseek");
    return(-1);
  }

  for (;;) {


    /* if number of packets to send is not specified, call sched(). */
    if (link_data[0]->send_count == 0 &&
        link_data[1]->send_count == 0) {
      sched(link_num, link_data);
    }

    k = 0;
```

```
while (k < link_num) {
  if (link_data[0]->send_count == 0 &&
      link_data[1]->send_count == 0)
    break;

  if (link_data[k]->send_count == 0) {
    if (k == link_num - 1) {
      k = 0;
      continue;
    } else {
      k++;
      continue;
    }
  }


  /* data send procedure */
  if (change_route(route_sock, link_data[k]->link_ip) < 0) {
    fprintf(stderr, "#ERROR: experiment: change_route()\n");
    return(-1);
  }

  memset(buf, 0, sizeof(buf));
  if (opt & SEQ) {
    memcpy(buf, &packet_sequence, sizeof(int));
    read_num = read(file_fd,
                    &buf[sizeof(int)],
                    link_data[k]->mtu-sizeof(int));
  } else {
    read_num = read(file_fd, buf, link_data[k]->mtu);
  }

  if (read_num < 0) {
    perror("#ERORR: experiment: read()");
    continue;
  } else if (read_num == 0)
    goto reread;
  D(printf("#DEBUG: sock: %d\n", sock););
retry:
  retry_count++;
```

```
          if (sendto(sock,
                     buf,
                     read_num,
                     0,
                     (struct sockaddr *)&sin,
                     sizeof(sin)) < 0) {
            if (errno == ENOBUFS) {
              link_data[k]->nobufs++;
              printf("#DEBUG: ENOBUFS: %s, count: %d\n",
                     link_data[k]->link_ip,
                     link_data[k]->nobufs);
              goto retry;
            } else {
              perror("#DEBUG: experiment(): sendto()");
              return(-1);
            }
          }
          retry_count = 0;
          D(fprintf(stderr, "#DEBUG: sent\n"););
          packet_sequence++;

          link_data[k]->send_count--;

          k++;
        }
      }
    }
    close(file_fd);
    close(sock);

    return(0);
}

/*
 * sched()
 *  set send_count according to bandwidth, mtu, delay ..
 */
int
sched(link_sched_num, data)
      int link_sched_num;
```

```
        struct link_sched_data **data;
{
  int i;

  /* find which data's ref_count is the smallest */
  for (i=0; i<link_sched_num; i++)
    data[i]->send_count = data[i]->bandwidth_exp / data[i]->mtu_exp;

  return(0);
}



/*
 * log family
 *  log execution time between start and end.
 */
int
log_start(start_tv)
        struct timeval *start_tv;
{
  memset(start_tv, 0, sizeof(struct timeval));
  if (gettimeofday(start_tv, NULL) < 0) {
    perror("log_start");
    return(-1);
  }
  return(0);
}

int
log_end(end_tv)
        struct timeval *end_tv;
{
  memset(end_tv, 0, sizeof(struct timeval));
  if (gettimeofday(end_tv, NULL) < 0) {
    perror("log_start");
    return(-1);
  }
  return(0);
}
```

```
int
log_sub(from, to, result)
     struct timeval *from;
     struct timeval *to;
     struct timeval *result;
{
  int usec;
  int sec;

  memset(result, 0, sizeof(struct timeval));

  usec = to->tv_usec - from->tv_usec;
  sec  = to->tv_sec - from->tv_sec;
  if (usec < 0) {
    sec--;
    usec = usec + 1000000;
  }

  result->tv_sec  = sec;
  result->tv_usec = usec;

  return(0);
}

/*
 * get_default_gw();
 *  get current default gateway address; return pointer of rt_msghdr{}.
 *
 */
struct rt_msghdr *
get_default_gw(route_sock)
     int route_sock;
{
  char *buf;
  struct rt_msghdr *rtm;

  rtm = rtm_alloc("1.0.1.0", RTM_GET);
  if (rtm == NULL) {
    fprintf(stderr, "#ERROR: get_default_gw(): rtm_alloc()\n");
    return(NULL);
```

```
  }
  if (write(route_sock, (char *)rtm, rtm->rtm_msglen) < 0) {
    perror("write");
    return(NULL);
  }
  free(rtm);

  buf = (char *)malloc(BUFSIZ); /* XXX */
  memset(buf, 0, BUFSIZ);
  if (read(route_sock, buf, BUFSIZ) < 0) {
    perror("#ERROR get_default_gw(): read");
    return(NULL);
  }

  return(rtm);
}



/*
 * print_route_to(int route_sock, struct sockaddr_in *dst_addr);
 *  print routing information to the dst_addr.
 */
int
print_route_to(route_sock, dst_addr)
     int route_sock;
     char *dst_addr;
{
  int read_num;
  char buf[BUFSIZ];
  struct sockaddr *rti_info;
  struct sockaddr_in *sin;
  struct rt_msghdr *rtm;

  rtm = NULL;
  if ((rtm = rtm_alloc(dst_addr, RTM_GET)) == NULL){
    fprintf(stderr, "#ERROR: rtm_alloc()\n");
    return(-1);
  }
```

```
if (write(route_sock, (char *)rtm, rtm->rtm_msglen) < 0) {
  perror("print_route_to() write");
  return(-1);
}

/* free malloced data in rtm_alloc() */
free(rtm);

/* read buffer from kernel */
memset(buf, 0, sizeof(buf));
read_num = read(route_sock, buf, BUFSIZ);
rtm = (struct rt_msghdr *)buf;

printf("#DEBUG: read done\n");
printf("#DEBUG: RTA_DST: %d, rtm->rtm_addrs: %d\n",
       RTA_DST, rtm->rtm_addrs);
printf("#DEBUG: RTM_GET: %d, rtm->rtm_type: %d\n",
       RTM_GET, rtm->rtm_type);
printf("#DEBUG: getpid(): %d, rtm->rtm_pid: %d\n",
       getpid(), rtm->rtm_pid);
printf("#DEBUG: read_num: %d\n", read_num);
printf("#DEBUG: sizeof(struct rt_msghdr): %d\n",
       sizeof(struct rt_msghdr));
printf("#DEBUG: sizeof(struct sockaddr_in): %d\n",
       sizeof(struct sockaddr_in));
printf("#DEBUG: RTAX_MAX: %d\n",RTAX_MAX);


sin = (struct sockaddr_in *)(rtm + 1);
rti_info = (struct sockaddr *)(rtm + 1);
if ((rtm->rtm_addrs & RTA_DST) != 0) {
  printf("#DEBUG: RTA_DST\n");
  printf("  sa->sa_len: %d\n", (rti_info+RTAX_DST)->sa_len);
  printf("  dest: %s\n",
         inet_ntoa(((struct sockaddr_in *)
                    (rti_info+RTAX_DST))->sin_addr));
}
if ((rtm->rtm_addrs & RTA_GATEWAY) != 0) {
  printf("#DEBUG: RTA_GATEWAY\n");
  printf("  sa->sa_len: %d\n", (rti_info+RTAX_GATEWAY)->sa_len);
```

```
    printf("  gateway: %s\n",
            inet_ntoa(((struct sockaddr_in *)
                    (rti_info+RTAX_GATEWAY))->sin_addr));
}
if ((rtm->rtm_addrs & RTA_NETMASK) != 0) {
    u_char c;
    u_char *ptr;
    char buf[BUFSIZ];
    printf("#DEBUG: RTA_NETMASK\n");
    c = (rti_info+RTAX_NETMASK)->sa_len;
    printf("  (rti_info+RTAX_NETMASK)->sa_len: %d\n", c);
    printf("  (rti_info+RTAX_NETMASK)->sa_family: %d\n",
            (rti_info+RTAX_NETMASK)->sa_family);
    if(c == 5)
      printf("5\n");
    else if (c == 6)
      printf("6\n");
    else if (c == 7)
      printf("7\n");
    else if (c == 8)
      printf("8\n");

    ptr = &(rti_info+RTAX_NETMASK)->sa_data[2];
    snprintf(buf, sizeof(buf), "%d.%d.%d.%d",
              *ptr, *(ptr+1) ,*(ptr+2) ,*(ptr+3));
    printf(" netmask: %s\n", buf);
}
if ((rtm->rtm_addrs & RTA_GENMASK) != 0) {
    printf("#DEBUG: RTA_GENMASK\n");
    printf("  (rti_info+RTAX_GATEWAY)->sa_len: %d\n",
            (rti_info+RTAX_GENMASK)->sa_len);
    printf("  genmask: %s\n",
            inet_ntoa(((struct sockaddr_in *)
                    (rti_info+RTAX_GENMASK))->sin_addr));
}
if ((rtm->rtm_addrs & RTA_IFP) != 0) {
    printf("#DEBUG: RTA_IFP\n");
}
if ((rtm->rtm_addrs & RTA_IFA) != 0) {
    printf("#DEBUG: RTA_IFA\n");
```

```
  }
  if ((rtm->rtm_addrs & RTA_AUTHOR) != 0) {
    printf("#DEBUG: RTA_AUTHOR\n");
  }
  if ((rtm->rtm_addrs & RTA_BRD) != 0) {
    printf("#DEBUG: RTA_BRD\n");
  }

  return(0);
}



/*
 * change_route(int route_sock, struct sockaddr_in *default_addr);
 *  change default route to default_addr.
 */
int
change_route(route_sock, default_addr)
     int route_sock;
     char *default_addr;
{
  struct rt_msghdr *rtm, *gw_rtm;

  if ((gw_rtm = get_default_gw(route_sock)) == NULL) {
    fprintf(stderr, "change_route(), get_default_gw\n");
    return(-1);
  }

  if ((rtm = rtm_alloc(default_addr, RTM_CHANGE)) == NULL) {
    fprintf(stderr, "#ERROR: change_route(): rtm_alloc()\n");
    return(-1);
  }
  D(printf("#DEBUG: change_route(): default_addr: %s\n", default_addr););
  if (write(route_sock, (char *)rtm, rtm->rtm_msglen) < 0) {
    perror("#ERROR: change_route(): write()");
    return(-1);
  }

  free(rtm);
  free(gw_rtm);
```

```
  return(0);
}

/*
 * rtm_alloc(char *, int);
 *  allocate rt_msghdr{} for querying to routing table
 *  valid type are below:
 *   RTM_ADD, RTM_CHANGE, RTM_DELETE, RTM_GET, RTM_LOCK
 *  if type is 0, RTM_GET is used as default
 */
struct rt_msghdr *
rtm_alloc(addr, type)
     char *addr;
     int type;
{
  char *buf;
  struct sockaddr_in *sin;
  struct rt_msghdr *rtm;

  buf = malloc(BUFSIZ);
  if (buf == NULL) {
    perror("malloc");
    return(NULL);
  }

  memset(buf, 0, BUFSIZ);
  rtm = (struct rt_msghdr *)buf;
  sin = (struct sockaddr_in *)(rtm + 1);


  if (type == 0)
    type = RTM_GET;

  if (type == RTM_GET) {
    rtm->rtm_msglen
      = sizeof(struct rt_msghdr) + sizeof(struct sockaddr_in);
    rtm->rtm_flags = RTF_UP | RTF_GATEWAY | RTF_STATIC;
    rtm->rtm_addrs    = RTA_DST;
```

```
    sin->sin_len = sizeof(struct sockaddr_in);
    sin->sin_family = AF_INET;
    inet_pton(AF_INET, addr, &sin->sin_addr);

  } else if (type == RTM_CHANGE) {
    struct sockaddr_in *sin1;

    rtm->rtm_msglen
      = sizeof(struct rt_msghdr) + sizeof(struct sockaddr_in) * 2;
    rtm->rtm_flags = RTF_UP | RTF_GATEWAY | RTF_STATIC;
    rtm->rtm_addrs    = RTA_DST | RTA_GATEWAY | RTA_NETMASK;

    sin->sin_len = sizeof(struct sockaddr_in);
    sin->sin_family = AF_INET;

    sin1 = sin + RTAX_GATEWAY;
    sin1->sin_len = sizeof(struct sockaddr_in);
    sin1->sin_family = AF_INET;
    inet_pton(AF_INET, addr, &sin1->sin_addr);

    sin1 = sin + RTAX_NETMASK;
    sin1->sin_len = sizeof(struct sockaddr_in);
    sin1->sin_family = AF_INET;
  }

  rtm->rtm_version = RTM_VERSION;
  rtm->rtm_type    = type;
  rtm->rtm_pid     = getpid();
  rtm->rtm_seq     = seq++;

  return(rtm);
}

int
init_udp(port)
     int port;
{
  int sock;
  int one = 1;
  struct sockaddr_in server_sa;
```

```
  if ( (sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
    perror("socket()");
    return(-1);
  }

  if(setsockopt(sock, SOL_SOCKET,
                SO_REUSEADDR, (void *)&one, sizeof(int)) <
     0){
    perror("setsockopt");
    return(-1);
  }


  memset(&server_sa, 0, sizeof(server_sa));
  server_sa.sin_family      = AF_INET;
  server_sa.sin_addr.s_addr = htonl(INADDR_ANY);
  server_sa.sin_port        = htons(port);

  if ( bind(sock,
            (struct sockaddr *)&server_sa, sizeof(server_sa)) < 0 ) {
    perror("bind");
    return(-1);
  }

  return(sock);
}

/*
 * Print buffer 'buf' of length
 * 'len' bytes in hex format to file 'out'.
 */
void hex(FILE *out, char *buf, int len)
{
  int i;

  for (i = 0; i < len; i++) {
    fprintf(out, "%02x", (unsigned char)buf[i]);
  }
} /* hex */
```

```
/*
 * usage()
 */
void
usage()
{
  fprintf(stderr,
  "This program is similar to route command.\n"
  "But is for evaluation of application level bandwidth aggregation\n"
  "\n"
  "./route_sock_change_default "
  "  -c <ip address>; change default gateway\n"
  "  -p <ip address>; print routing table info to addr\n"
  "  -e; experiment mode\n"
  "  -t <times>; how many times do you send datagram?\n"
  "  -P <port number>; for experiment; port number to send datagram\n"
  "  -f <file name>; for experiment; file to send \n"
  "  -d <destination addr>; for experiment; destination IP address \n"
  "  -1 <default route addr1>; for experiment; destination IP address \n"
  "  -2 <default route addr2>; for experiment; destination IP address \n"
  "  -s for experiment; add sequence number \n"
  "  -h; print this help\n");
}
```

付 録B

APP2 のソースコード

List B.1: APP2 のソースコード

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/sysctl.h>
#include <sys/time.h>
#include <sys/uio.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <fcntl.h>
#include <netdb.h>
#include <stdio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

```
/* fuction declaration */
void usage(void);
long get_freq(void);
int  udp_recv_init(int);

/* definition */
/* x should be long long int */
#define RDTSC(x) __asm __volatile(".byte 0x0f,0x31" : "=A" (x))
#define OUTPUTBUF 8192

/* global variables */
int argo;

int
main(argc, argv)
        int argc;
        char **argv;
{
        int c;
        int port;
        int sock;
        int recvsize = 0;
        int i, n;
        int output_num;
        int rcvbuf_num;
        socklen_t  sizeof_rcvbuf_num;
        long long int now; // for pentium counter
        socklen_t client_size;
        char *buf;
//      char output_buf[OUTPUTBUF*2];
        extern char *optarg;
        struct sockaddr_in client_sa;

        /* check variable */
        if (argc < 2) {
                usage();
                exit(1);
        }

        argo = 0;
```

```
while((c = getopt(argc, argv, "s:p:oh")) != -1){
        switch(c){
        case 'p':
                port = atoi(optarg);
                break;
        case 's':
                recvsize = atoi(optarg);
                break;
        case 'o':
                argo = 1;
                break;
        case 'h':
                usage();
                exit(0);
        default:
                usage();
                break;
        }
}


if (recvsize != 0)
        buf = (char *)malloc(recvsize);
else
        buf = (char *)malloc(BUFSIZ);
if (buf == NULL) {
        perror("malloc");
        exit(0);
}

if ((sock = udp_recv_init(port)) < 0) {
        fprintf(stderr, "udp_recv_init\n");
        exit(1);
}

rcvbuf_num = 262144;
rcvbuf_num -= 35000;
sizeof_rcvbuf_num = sizeof(rcvbuf_num);
if (setsockopt(sock, SOL_SOCKET, SO_RCVBUF,
```

```
                (void *)&rcvbuf_num, sizeof_rcvbuf_num) < 0) {
            perror("setsockopt");
            exit(1);
        }


        /* main routine */
        i = 0;
        output_num = 0;
        while (1) {
            memset(&client_sa, 0, sizeof(client_sa));
            client_size = sizeof(client_sa);
            if ( (n = recvfrom(sock,
                               buf,
                               recvsize,
                               0,
                               (struct sockaddr *)&client_sa,
                               &client_size)) < 0 ) {
                perror("recvfrom");
                continue;
            }
            memset(&now, 0, sizeof(now));

            if (argo == 0) {
                RDTSC(now);
            } else if (argo == 1 ) {
                if (write(1, buf, n) < 0) {
                    perror("write");
                    exit(1);
                }
                memset(buf, 0, n);
            }
        }

        exit(0);
}


int
udp_recv_init(port)
```

```
        int port;
{
        int sock;
        struct sockaddr_in server_sa;


        memset(&server_sa, 0, sizeof(server_sa));
        server_sa.sin_family      = AF_INET;
        server_sa.sin_addr.s_addr = htonl(INADDR_ANY);
        server_sa.sin_port        = htons(port);


        if ( (sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
                perror("socket()");
                return(-1);
        }

        if ( bind(sock,
                  (struct sockaddr *)&server_sa,
                  sizeof(server_sa)) < 0 ) {
                perror("bind");
                return(-1);
        }

        return(sock);

}


/*
 * long get_freq(void);
 *  Description:
 *   get CPU frequency of the system.
 *  Return value:
 *   frequency in hz.
 */
long
get_freq()
{
        long hz;
```

```
        size_t hzsize = sizeof(hz);

        if (sysctlbyname("machdep.tsc_freq",
                         &hz,
                         &hzsize, NULL, 0)) { // get Hz
                perror("cannot get Hz");
                return(-1);
        }

        return(hz);
}



void
usage()
{
        fprintf(stderr,
                "Description: \n"
                " This program recieves data on specified port\n"
                "\n"
                "./udpinput  "
                "    -p <port number>\n"
                "    -s <datagram size to recieve for 1 read>\n"
                "    -o (if specified, output received data to stdout)\n"
                "    -h (show this usage)\n"
                "\n");
}
```