システムソフトウェアと
オペレーティング・システム 81－9
（1999．5．6）

適応的アプリケーションのための資源管理機構および資源量通知機構を持つフレームワーク S-MAX の実装

安田 泰勲† 稲村 浩‡
†NTT 情報流通プラットフォーム研究所
‡NTT 移動通信網株式会社マルチメディア研究所

移動ネットワーク環境では，マルチメディアアプリケーションの有用性を向上させるために，切り替わったネットワークインターフェースに合わせて自分自身の QoS を変化させなければならない．本研究では，このようなアプリケーションの適応制御のために，ネットワークインターフェースの切替によるネットワーク資源量の変化をアプリケーションに通知し，資源管理を行なう機構を持つフレームワークである S-MAX の実装について述べる．我々は S-MAX を RT-Mach マイクロカーネル上に実装し評価を行なった．本研究の特徴はインターフェースの切替が起ったときにアプリケーションが適応動作を行うこと，'資源管理機構' および '資源量通知機構' が協調動作することである．

Implementation of A Framework S-MAX using Resource Enforcement and Change Notification for Adaptive Applications

Yasunori Yasuda† Hiroshi Inamura‡
†NTT Information Sharing Platform Laboratories
‡Multimedia Laboratories. NTT Mobile Communications Networks Inc.

In mobile network environment, multimedia applications should adapt its QoS (Quallity of Service) to a network interface change in order to achieve high availability. In this paper, we describe the design and the implementation of *S-MAX* which is a framework using notification of interface change to trigger adaptation in applications and resource enforcing in order to handle a drastic change in network environment. The key insight of our work is using a notification of the interface change to trigger adaptation in applications, and combination *S-MAX* support to enforce bandwidth usage bounds.

1. はじめに

ラップトップコンピュータでは様々なネットワークインターフェースが利用可能になっている．これらのネットワークインターフェースの多くは PC カードで提供されている．ユーザはコンピュータを利用中にインターフェースを交換すること (hot-swapping) が可能であり，ユーザのモビリティは格段に向上した．例えば，オフィスにいるときはラップトップコンピュータを Ethernet につなぎ，会議室に移動する時は無線 LAN に切替えて使い，さらに外出するときには PHS に切り替えることによって，ネットワークを利用するアプリケーションを様々な場所で使い続けることができる．

我々はこのようにモビリティを確保するためにネットワークインターフェースを切り替え，その上でアプリケーションを使い続ける環境を移動ネットワーク環境と呼ぶ．移動ネットワーク環境でマルチメディアアプリケーションを使い続ける場合，マルチメディアアプリケーションはネットワークインターフェースの切替に対して QoS (Quality of Service) を変化させるように振舞わなければならない．なぜならネットワークインターフェースの切替によって利用可能なネットワーク資源の量 (帯域幅や遅延など) が大きく変化するが，これがアプリケーションの QoS に大きく影響するからである．インターフェースの切替が起こった場合，通常の静的な資源予約ではうまくいかない．例えば，Ethernet に接続されたラップトップコンピュータで，インターネット電話アプリケーションとファイル転送アプリケーションを同時に利用する場合を考える．最初は，インターネット電話は 4Mbps の帯域を予約し，ファイル転送アプリケーションは残りの帯域を利用している．ここで通信中にユーザがネットワークインターフェースを Ethernet(10Mbps) から WaveLAN(2Mbps) に切替える．この場合，インターネット電話は帯域を再予約するだけでなく，自分自身の QoS のレベルを落し，利用する帯域を減らす必要がある．このような適応動作を実現するためには，システムがネットワークインターフェースの切替に基づく資源量の変化をアプリケーションに通知しなければならない．

本研究は移動ネットワーク環境においてマルチメディアアプリケーションの有用性を高めることを目的とし，インターフェースの切替が起こったときに

アプリケーション自身が, 自分の利用しているネットワークインターフェースにあわせて QoS を変化させることを支援するフレームワークを実現する. このフレームワークではネットワークインターフェース切替によって変化した利用可能なネットワーク資源量を通知する機構およびネットワーク資源の管理機構の二つの機構が協調動作することの二つを特徴とする. 我々は, 利用可能な資源量を通知する機構, および帯域幅を制御する機構をもつフレームワークである S-MAX を設計, 実装した. このフレームワークの評価として, マルチメディアアプリケーションの一つである NV を用いてそのフレームレートの測定, およびインターフェース切替の際の S-MAX が行う処理に必要な時間の測定を行なった.

まず, 2節ではこのフレームワークの概要について述べ, 3節および 4節でプロトタイプの設計と実装について述べる. 次に 5節で本プロトタイプの評価について述べる. 最後に 6節で関連研究について述べ, 7節でまとめる.

2. 適応的アプリケーションのためのフレームワーク S-MAX の概要

この節では我々が提案する適応的アプリケーションのためのフレームワーク S-MAX の概要について述べる.

S-MAX は二つのメカニズム, 資源管理機構および資源量通知機構から構成される. 資源量通知機構はネットワークインターフェースの切替によって変化した利用可能なネットワーク資源量を通知する機構であり, 資源量管理機構はネットワーク資源の使用量を管理するための機構である.

S-MAX の特徴は二つある. ネットワークインターフェースの切替が起こったときにアプリケーションが適応動作をはじめること, および資源量通知機構および資源量管理機構の二つの機構が協調動作することである.

2.1. 資源管理機構

資源管理機構は, ラップトップコンピュータ上で動作する全てのアプリケーションの使用するネットワーク資源 (帯域幅と遅延) の量を制御する. アプリケーションは利用したいネットワーク資源の割り当て要求を資源管理機構に伝える. 資源管理機構は, アプリケーションからの要求が現在利用可能なネットワーク資源量より少ない場合はネットワーク資源の割り当てを行ない, 不十分な場合には割り当てを行なわない.

資源管理機構を導入する利点は二つある. 一つは他のアプリケーションがネットワーク資源を使い尽くさないようにアプリケーション毎に保護することが可能になることである. 通常のオペレーティングシステムでは全てのアプリケーションに対して暗黙に公平にネットワーク資源を割り当てる. 例えば, マルチメディアアプリケーションとそれ以外のアプリケーションを同時に使う場合を考えると, マルチメディアアプリケーションの方にネットワーク資源を余分に割り当てたいという要求がある. この要求を満たすためには資源管理機構によってネットワークを分割することが必要になる.

二つめの利点として, 適応動作を行なうマルチメディアアプリケーションを実装する場合, 資源管理機構を利用することによってプログラミングが簡単になる. もしこの機構が利用できなければ, アプリケーションプログラマはアプリケーション自身がネットワーク資源の利用を調節する複雑なコードを書かなければならない.

2.2. 資源量通知機構

資源量通知機構は各アプリケーションが利用可能なネットワーク資源の量をそれぞれに通知する. ネットワーク資源の量は帯域幅と遅延である. 資源量通知機構は通知を行う合図, つまりアプリケーションが適応動作を行うための合図としてネットワークインターフェースの切替を用いる. なぜなら, ラップトップコンピュータで利用可能なネットワークインターフェースの帯域幅や遅延は表 1に示すようにその種類によって一桁から二桁のオーダで異なり, 固定的な資源予約だけでは許容できる QoS を達成することは難しい. それゆえ, アプリケーション自身が大幅な利用可能なネットワーク資源量の変化, つまりネットワークインターフェースの切替に適応して QoS を変化させなければならないからである.

device	bandwidth(bps)	latency(msec)
Ethernet	10 M	1 ··· 10
Wireless LAN	1 ··· 2 M	1 ··· 10
serial/PHS	32 ··· 64 k	10 ··· 100

表 1. 移動ホストで利用できるネットワークインターフェース

これらのネットワークインターフェースは PC カード (PCMCIA カード) によって提供されているため, ネットワークインターフェースの切替は PC カードイベントによって知ることができる. そこで資源量通知機構は PC カードイベントからネットワークインターフェースの切替を検知しそれを適応制御の合図として, 資源管理機構とやりとりを行い各アプリケーションの利用可能なネットワーク資源の量を得たあと, 各アプリケーションに通知する. アプリケーションはこの受け取った値を元に自分自身の QoS を変えるなどの適応動作を行う.

3. S-MAX の設計

この節では S-MAX の設計について述べる.

図 1に S-MAX の設計図を示す. S-MAX は三つのコンポーネント 'Arbiter', 'Notifier', 'Library' から構成される.

3.1. Arbiter の設計

'Arbiter' は資源管理機構のすべてと資源量通知機構の一部を実現している. 資源管理機構のネットワーク資源量の制御はパケットスケジューリングによって実現されている. 'Arbiter' はアプリケーションが使用したいネットワーク資源の量を QoS パラメータ (帯域幅, 遅延) の形式で受け取り, それをもとにアプリケーションからの全てのパケットをパケットスケジューリングする. 資源量通知機構における 'Arbiter' の機能については 3.2節で詳しく述べる.

'Arbiter' は, アプリケーションに対して物理的なネットワークインターフェースを抽象化した仮想ネッ

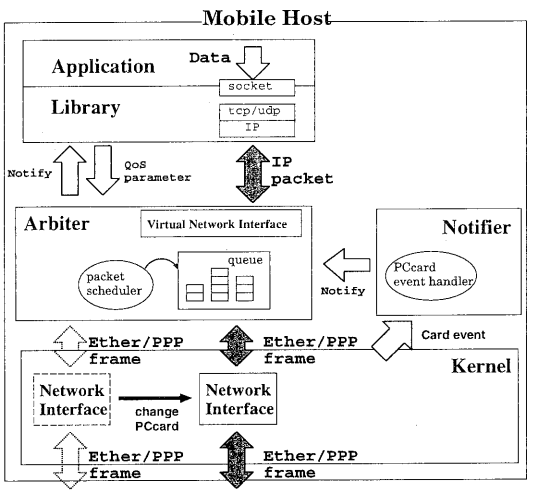

図 1. S-MAX のブロック図

トワークインターフェースを提供する. 仮想ネットワークインターフェースは QoS に直接関わる情報(ネットワークインターフェースの帯域幅, 遅延, MTU など) のみを持つ. アプリケーションはこの仮想ネットワークインターフェースを利用することによって, ネットワークインターフェースの切替を QoS にかかわるパラメータの変化として統一的に扱うことができる. そのため, 'Arbiter' とアプリケーションとのパケットのやり取りはすべて IP パケットで行い, 'Arbiter' は IP パケットのスケジューリングに加えて IP パケットと現在利用しているネットワークメディア用のパケットとの変換を行う.

3.2. Notifier の設計

'Notifier' は 'Arbiter' と連携することで資源量通知機構を実現している. 'Notifier' は PC カードイベントからネットワークインターフェースの切替を検知し, ネットワークインターフェースの設定等を行い, ネットワークインターフェース名とその状態 (disable/enable) を 'Arbiter' に通知する. 'Arbiter' は 'Notifier' からネットワークインターフェースが disable になったという通知を受けとると, 仮想ネットワークインターフェースの利用可能な資源が 0 になったことをアプリケーションに通知し, さらにアプリケーションの送受信をブロッキングする. また, ネットワークインターフェースが enable になったという通知を受け取った場合には, 仮想ネットワークインターフェースに新しい情報を設定し, それを元にアプリケーションがあらかじめ要求しておいた情報をアプリケーションに通知し, 送受信をアンブロッキングする. 現在アプリケーションが要求できる情報は利用可能な帯域幅, 遅延, ネットワークメディア, MTU であり, これらのうち任意のものを選択できる.

3.3. Library の設計

'Library' はアプリケーションと 'Arbiter' との Application Programming Interface (API) を提供しており, 以下の機能をもつ.

- QoS パラメータの設定
- 通知してほしい情報の設定
- 通知された情報とハンドラの起動

アプリケーションは S-MAX を利用するためには 'Library' をリンクする必要がある.

また, 通信のためのプログラミングインターフェースとして UNIX ソケットインターフェースを採用しており, QoS パラメータや通知の設定以外のデータの送受信に関しては通常のソケットプログラミングと同様である. さらにこの 'Library' ではネットワークプロトコルスタック (IP, UDP, TCP...) を実装しており, 'Libary' をリンクしたアプリケーション自身が IP パケットを構築し, 'Arbiter' を経由したデータの送受信を行う.

4. S-MAX の実装

この節では S-MAX の実装について述べる. まず実装のターゲットとしたプラットフォームについて述べ, 次に各コンポーネントの実装について述べる.

4.1. プラットフォーム

我々はプラットフォームとして RT-Mach[9] マイクロカーネル環境を選択し, そのバージョンの一つである Real-Time Mach NTT Release 2.0[3] 上で実装を行った. RT-Mach マイクロカーネルはマイクロカーネルベースの分散リアルタイムオペレーティングシステムであり, マルチメディアアプリケーションを実装するためのいくつかの便利な機能 (リアルタイムマルチスレッド, リアルタイムスケジューラ, ネットワーク透過なプロセス間通信など) を提供している. Real-Time Mach NTT Release 2.0(RT-Mach NR2.0) は, 現在おもに利用されている Real-Time Mach の幾つかの版を統合したものであり, 慶應大学 MKng プロジェクト [6] MKNG008, Carnegie-Mellon University RK98a[7], NTT 2.0 alpha 版をベースとしている. また, RT-Mach NR2.0 は FreeBSD2.2.8-RELEASE のユーザレベルプログラムや インストーラと統合しており, さらに, 慶應大学 MKng プロジェクトで開発された Real-Time Mach 用ソフトウェアを含んでいる.

図 2に RT-Mach マイクロカーネル環境での S-MAX の実装を示す. 以下各々のコンポーネントの実装について述べる.

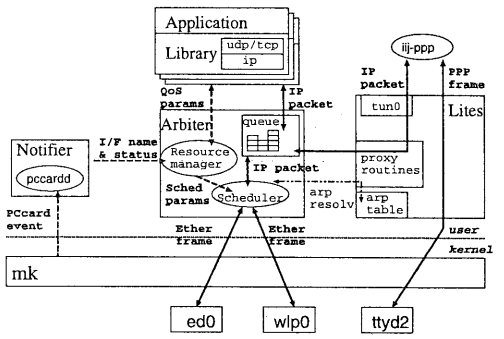

図 2. RT-Mach 上での S-MAX の実装図

4.2. Arbiter の実装

Arbiter はパケットスケジューリングによって帯域幅と遅延を制御する. S-MAX では UNIX ソケット互換のプログラミングインターフェースを採用しており, キューは一つの UNIX ソケットに対して送受信用それぞれ一つずつ用意される. Arbiter はアプリケーションから IP パケットをソケット経由で受けとり, それを一旦キューに入れ, そのキューの QoS パラメータに基づいてスケジューリングを行なう. スケジューリングに用いる QoS パラメータはアプリケーションから渡されたものを Arbiter が各キューに設定する.

本実装ではパケットスケジューリングアルゴリズムとして Weighted Deficit Round Robin (WDRR)[8] アルゴリズムを採用した. このアルゴリズムは Weighted Round Robin (WRR) を改良し, 可変長パケットに対応したものである. WDRR は帯域幅を制御することが可能であり, Max-min fair share に従う. パケットスケジューリングアルゴリズムとして Weighted Fair Queuing (WFQ)[8] アルゴリズムが知られているが, 計算時間の短さ, 実装の容易さから WFQ ではなく, WDRR を選択した. 本実装では遅延の制御に関しては低遅延, 高遅延の 2 レベルをキューに設定可能にしている. 低遅延のキューに入っているパケットは, 常に高遅延のキューに入っているパケットよりも優先して送出されるため, 対話処理を行なうプログラムなどの低遅延が必要なプログラムに対応可能である.

キューはネットワークプロトコル処理部 (IP 層) とデバイスドライバの間に位置する. デバイスドライバの中にキューを作った場合, 切替える可能性のある全てのネットワークインターフェースのデバイスドライバを変更しなければならないため, 実装の手間が大きい. しかし, このように設計・実装するによって, 特定のネットワークインターフェースに依存せずに, ネットワークインターフェースを統一的に扱うことができ, 実装の手間も減らすことができる.

次に Arbiter における IP パケット送受信の方法について述べる. 本プロトタイプでは対応しているネットワークインターフェースは Ethernet とその互換のもの (無線 LAN など) とシリアルポート (PPP) である. この処理は Ethernet (およびその互換) 経由の場合とシリアルポート経由の場合とで異なる (図 2 参照).

Ethernet 経由でパケットを送出する場合, Arbiter は IP パケットに Ethernet ヘッダをつけて Ethernet フレームを作成し, それをネットワークインターフェースに直接書き込む. その際, ARP 解決は Lites 4.4 BSD-Lite UNIX server (以下 Lites) に依頼する. また, それともにその結果を cache しておく. Ethernet 経由でパケットを受信する場合には Arbiter はカーネルからパケットフィルタ経由で直接受け取る. 次に受け取った Ethernet フレームから Ethernet ヘッダをはずし IP パケットにした上で適切な受信用のキューを経由してアプリケーションに渡す. ここで用いるパケットフィルタはアプリケーションがソケットの設定を行った時に Arbiter が作成し, カーネルにインストールする.

シリアルポート経由でパケットを送受信するために Arbiter はユーザレベル PPP (IIJ-PPP) を利用する. この機構のために Lites のトンネルデバイスと直接 IP パケットをやりとりできるインターフェースを Lites に追加した. まず パケットを送信する場合, Arbiter はアプリケーションから受けとった IP パケットを Lites の トンネルデバイス (tun0) 経由でユーザレベル PPP に渡す. ユーザレベル PPP は受け取った IP パケットから PPP フレームを構築しシリアルポートに Lites 経由で送出する. パケットを受信する場合にはユーザレベル PPP が Lites 経由で PPP フレームを受取る. 次にユーザレベル PPP は受け取った PPP フレームを IP パケットにして Lites のトンネルデバイス経由で Arbiter に渡す. Arbiter はそれを受信用のキューを経由してアプリケーションに渡す.

このような実装手法をとった場合にはアプリケーションが Lites を直接利用する場合に比べて IPC の回数が増え, 余計なオーバヘッドが生じることになる. しかし, シリアルポート経由で通信をした場合はそもそも帯域幅が小さくスループットが出ないためこのオーバヘッドが相対的に隠れる, 実装を簡単にすることができるなどの理由によりこのような実装を行った.

4.3. Notifier の実装

Notifier はネットワークインターフェースの切替を検知し, Arbiter にネットワークインターフェース名とその状態 (enable/disable) を通知する機能を持つ. ラップトップコンピュータにおいて切替 (hot-swapping) 可能なインターフェースは PC カードで提供されているため, PC カードイベントをハンドルすることにより, ネットワークインターフェース切替の検知を行なう.

Notifier は, MKng プロジェクトによって RT-Mach 環境に移植された PAO[1] を拡張して実装を行った. PAO は FreeBSD 用のモバイルパッケージであり, PC カードイベントハンドラや各種デバイスドライバなどを持つ. Notifier は主に拡張した pccardd から成る. 拡張 pccardd は PC カードイベントハンドラプログラムであり, Arbiter と通信を行う機能およびネットワークインターフェースを抜く前にスクリプトを実行する機能を追加してある. 後者の機能はネットワークインターフェースを抜く前にネットワーク周りの後処理 (ifconfig down や route delete など) を行わないとそれ自体が失敗するという問題が生じるため pccardd に追加した.

次に拡張 pccardd の振る舞いについて述べる. 拡張 pccardd はカーネルからネットワークインターフェースが挿入されたという PC カードイベントを受けとると, ネットワークインターフェースの設定を行うスクリプトを呼び出し, それが終った後に, Arbiter にメッセージを送る. そのメッセージは有効になったインターフェース名とその状態 (enable) から成る. ネットワークインターフェースを抜く場合には pccardc (PC カードスロットを制御するプログラム) を用いて, その PC カードの電源を OFF にする. それと同時に Arbiter にネットワークインターフェース名とその状態 (disable) から成るメッセージを送る. 次に拡張 pccardd は pccardc の処理が終了すると, まずネットワーク周りの後処理を行うスクリプトを呼び出す. そしてネットワークインターフェースが抜かれたら, 拡張 pccardd はさらに後処理を行うスクリプトを呼び出す.

4.4. Library の実装

Library が提供している Arbiter との通信のための API を以下に示す.

API
int set_qos_request(int sockfd, /* socket descriptor */ resource_class_t *res, /* QoS parameters */ int notify_flags, /* mask for notification */ void(*handler)() /* adaptaion handler */) QoS パラメータおよび通知すべき情報の設定 適応動作用ハンドラの登録
int delete_qos_request(int sockfd, /* socket descriptor */ resource_class_t *res /* QoS parameters */) QoS パラメータの解放
int query_available_qos(int sockfd, /* socket descriptor */ int class_name, /* QoS parameter's ID */ resource_class_t *res /* QoS parameters */) 現在利用可能な QoS パラメータの問い合わせ

データの送受信のための API は通常の UNIX ソケットプログラミングと同様に socket(), bind(), send(), recv() などを用いる. これらの関数もすべて Library で実装しており, 使用方法は従来のものと同じである.

次に Library の利用法について述べる. アプリケーションはソケットの作成など通常の処理を行った後に本 API を用いて様々なパラメータを設定や適応動作のためのハンドラの登録を行う. QoS パラメータのうち帯域幅の設定は kbps を単位として整数値で指定し, 遅延の設定は低遅延, 高遅延のどちらかを指定する. QoS パラメータの設定がない場合には, 割り当てられていない残りの帯域幅を利用することになる. この残りの帯域幅はほかの QoS パラメータの設定を行わなかったもの全てと共有することになり, 遅延は高遅延が設定されているものとする. また, ハンドラを登録する際, そのハンドラを呼び出す合図を設定する必要がある. 設定可能な合図は以下の通りであり, これらの論理和を取ったものを設定する.

- NOTIFY_BW:使用可能な帯域幅の変化
- NOTIFY_DELAY:使用可能な遅延の変化
- NOTIFY_MEDIA:ネットワークメディアの変化
- NOTIFY_MTU:MTU の変化

ハンドラの登録が行われない場合には特別な処理は行われない. また, QoS パラメータの設定が成功した場合, そのキューに設定した QoS パラメータに対してユニークな ID が割り振られ, Arbiter 内でネットワーク資源量の管理 (要求元の識別) に用いられる. 以後アプリケーションから設定の変更や利用可能な資源量の問い合わせなどを行う場合などはその ID を用いる.

本実装は libsockets[4] をベースとしている. libsockets は RT-Mach マイクロカーネル環境において, 通信性能を向上させるために開発されたライブラリであり, ネットワークプロトコル処理 (IP, UDP, TCP, ...) を行う. 'Library' はこの libsockets に Arbiter との通信用 API の追加, およびキューイングを行なうための UNIX ソケットシステムコールの内部処理の変更などを行なったものである.

5. プロトタイプの評価

この節では本プロトタイプの評価について述べる. 本評価ではマルチメディアアプリケーションを用いて, S-MAX の有用性を評価する.

本実験ではマルチメディアアプリケーションである NV(Network Video) とテストアプリケーションとして実装した UDP パケットをバースト的に送信するプログラムを用いる. NV は画像を取り込み, ネットワーク経由で再生するアプリケーション Ron Frederic によって作成された. NV は SunOS 4.x, 5.x をはじめ, HP/UX, SGI, BSDI などさまざまなプラットフォーム上で動作しており, それを今回 RT-Mach NR2.0 + S-MAX 環境に移植した.

これらのプログラムを送信側 (S-MAX が動作) で同時に動かしながらネットワークインターフェースを WaveLAN(2Mbps 無線 LAN) から Ethernet(10Mbps) に, また Ethernet から WaveLAN に切り替え, さらに WaveLAN から PHS(32kbps) に切り替えて, その時の NV のフレームレートを受信側で測定した. NV の画像は IBM Smart Capture Card II を用いてキャプチャしたものを送信している. 画像は我々のオフィスを映したものであり, 全体を通して画像の動きはそれほど多くはない.

実験環境は以下の通りである.

送り側 Toshiba Portege7010 (Mobile Pentium II 300MHz, 160MB memory, Accton EN2216-1, AT&T WaveLAN, Paldio DC-2P), RT-Mach NR2.0

受け側 IBM PC/AT 互換機 (Pentium Pro 200MHz, 64MB memory, DE500A), RT-Mach NR2.0

この実験の結果を図 3に示す.

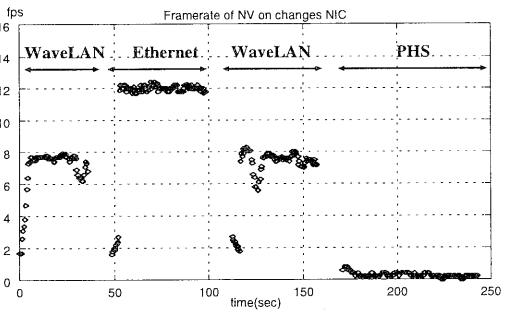

図 3. S-MAX 上で競合するトラフィックがある場合の NV のフレームレート (解像度 320x200, 8bit color).

この図からわかる通り WaveLAN 使用時には約 8fps, Ethernet 使用時には 約 12fps, PHS 使用時には約 0.2 fps のフレームレートがほぼ安定して達成されている. NV は WaveLAN 使用時には 1Mbps, Ethernet 使用時には 4Mbps, PHS 使用時には 25kbps の帯域を割り当て要求を行っており, この結果は競合するトラフィックが無い場合のフレームレートとほぼ同じフレームレートがでている. 切り替えた直後から数秒はフレームレートが低くなるのは NV 自体が RTP を用いた輻輳制御機構を持っており, ネットワークインターフェース切替によってデータの送信ができなかったことを輻輳状態と認識して送信を抑えるように振る舞うからである. また, それとは関係なく WaveLAN を用いているところで一部フレームレートが少し落ち込んでいる部分があるが, これは取り込んでいる画像の変化が大きかったからだと考える. これを確認するために我々は予備実験として NV の代りに 1472 バイトの UDP パケットをバースト送信するプログラムを用いて同様の実験を行い, スループットの計測を行った. その結果, プログラムが指定した帯域幅 (kbps 単位) との誤差は Ethernet では 0% から -0.3%, WaveLAN では ±0.5% とほぼ指定通りのスループットが達成されている. これらの

実験の結果を比較すると, NV を用いた実験でフレームレートが落ち込んだ原因は, その瞬間の画面の変化が大きく送信するデータ量が急激に増えたためだと言える.

また, 我々は S-MAX が切替処理を行った時の所要時間を計測した. その結果を図 4に示す. この実験では, あらかじめラップトップコンピュータに Ethernet カードと WaveLAN カードを同時に差しており, 最初は WaveLAN だけを使用可能にしてある. そしてカードの抜き差しは行わず, コマンドによってイベントを発行してネットワークインターフェース切替の一連の作業を行い, その所要時間を測定した.

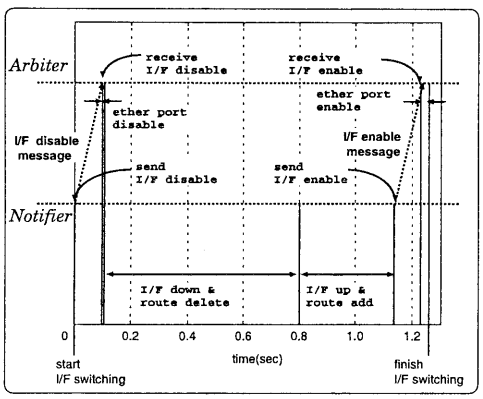

図 4. インターフェース切替のチャート.

まず, 切替コマンドが呼び出されると, Notifier は切替を検知し, Arbiter に interface(I/F) disable メッセージを送る. Arbiter はそのメッセージを受けとると, ネットワーク用のポートを disable し, パケットの送出を止める. ここまでの所要時間は 105[msec] である. 次に, Ethernet インターフェースを disable してからルーティングテーブルを flush し, さらに WaveLAN インターフェースを enable してからルーティングテーブルを更新するのにかかった時間は 701[msec]. 最後に, interface(I/F) enable メッセージを Arbiter に送り, Arbiter がネットワーク用のポートを再設定し, パケットを再び送り出すのにかかった時間は 373[msec]. 合計でインターフェース切替のハンドリングの所要時間は 1.337[sec] であり, これは十分妥当であると考える. なぜなら, すべてのアクティブな TCP のコネクションは, この実験で得られたハンドリングの所要時間ではタイムアウトせずにコネクションを維持できるからである. また PHS を使用する場合にはこれに併せて PPP 接続のための時間がさらに必要になる.

6. 関連研究

移動ネットワーク環境において適応的アプリケーションのためのフレームワークは幾つか提案されているが, 資源管理と通知機構の両方をもつフレームワークは我々の知る限りではまだない.

最も近い関連研究は Jon Inouye' のアプリケーションに特化した適応型システム [2] である. このシステムではビデオプレーヤがネットワークインターフェースの変化に適応的に振舞う。本研究との違いは, このシステムでは帯域幅の実測値をパラメータとしたフィードバックドリブンの適応制御を採用し, さらに資源管理機構をもっていないことである.

他の適応型システムの関連研究として Brian D. Noble と M. Satynarayanan による Odyssey[5] がある. Odyssey は実測ベースの適応制御を採用し, アプリケーションに依存せず, 資源のタイプに特化した QoS の適応制御機構を提供している. システムは資源レベルを観測し, 関係のあるアプリケーションに予約の参考となる値を提供する. 資源管理機構を持たないため, アプリケーションは資源の利用量を自分自身で完全に制御しなければならない. それゆえ, アプリケーションプログラマは我々のものにくらべて複雑なコードを書くことになる.

7. まとめ

本原稿で述べた適応的アプリケーションのためのフレームワークである S-MAX は, アプリケーションが利用するネットワーク資源量を制御する機構およびネットワークインターフェースの切替に伴うネットワーク資源量の変化をアプリケーションに通知する機構をもつ. 我々は S-MAX を RT-Mach マイクロカーネル環境で設計, 実装し, 評価を行なった. 本研究の特徴はインターフェースの切替を適応動作を行う合図としたこと, '資源管理機構' および '資源量通知機構' が協調動作するフレームワークであることである.

本フレームワークでは送信側で QoS 制御を行っており, QoS 要求も送信側のアプリケーションが出しているが, 今後は受信側が送信側に QoS 要求を出すことができるように拡張する予定である. それとあわせて httpd などの良く用いられるアプリケーションを S-MAX に実装し, 効果を確認する予定である.

参考文献

[1] Tatsumi Hosokawa. PAO: FreeBSD Mobile Computing Package; available at http://www.jp.freebsd.org/PAO/, 1997.

[2] Jon Inouye, Shanwei Cen, Calton Pu, and Jonathan Walpole. System Support for Mobile Multimedia Applications. In *Proceedings of NOSSDAV'97*, pages 143–154, 1997.

[3] NTT Information Sharing Platform Laboratories. Real-Time Mach NTT Release 2.0; available at http://info.isl.ntt.co.jp/rtmach/, 1999.

[4] Chris Maeda and Brian N. Bershad. Protocol Service Decomposition for High-Performance Networking. In *In Proceedings of SIGOPS'93*, December 1993.

[5] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. Agile Application-Aware Adaptation for Mobility. In *Proceedings of SOSP-16*, 1997.

[6] Keio-MKng project Keio University SFC. Micro Kernel Next Generation Project; available at http://www.mkg.sfc.keio.ac.jp/, 1997.

[7] CMU Real-Time and Multimedia Laboratory. Resource Kernels; available at http://www.cs.cmu.edu/ rtmach/, 1998.

[8] S.Keshav. *An Engineering Approach to Computer Networking.* Addison Wesley, 1997.

[9] H. Tokuda, T. Nakajima, and P. Rao. Real-Time Mach: Towards a Predictable Real-Time System. In *Mach Workshop, USENIX Association*, October 1990.