卒業論文 1998 年度

MIBsocket: 移動型計算機環境におけるネットワークエンティティの状態変化に対応する管理機構の設計と実装

慶應義塾大学 総合政策学部
湧川隆次
ryuji@sfc.wide.ad.jp

指導教官
徳田 英幸, 村井 純, 楠本 博之, 中村 修

MIBsocket: 移動体計算機環境におけるネットワークエンティティの状態変化に対応する管理機構の設計と実装

計算機は移動することにより, ネットワークエンティティが動的に変化する. これらネットワークエンティティは, ネットワークソフトウェア及びカーネル間で共有されている情報である. ネットワークソフトウェアは, ネットワークエンティティの情報を知る必要があり, またネットワークエンティティの一貫性を保ち管理する必要がある.

本研究の目的は, ネットワークエンティティの動的な状態変化に対応するネットワークエンティティの管理機構の実現である. 管理機構は, ネットワークエンティティの状態をメッセージの交換を用いて取得あるいは設定を行う. メッセージ交換を行うため, 管理機構はソケット (MIBsocket) を用い, FreeBSD2.2.7 + PAO-0930 上で実装した. MIBsocket は, 移動等により起きるネットワークエンティティの動的な状態変化を検知しネットワークソフトウェアに通知する. また, 要求と応答メッセージを交換することによりネットワークエンティティの情報を取得することができる. 移動型計算機は, ネットワークエンティティの状態を動的に計算機環境に最適にすることができる. MIBsocket により, ネットワークエンティティを統一的に管理することができる.

実装した MIBsocket のオーバヘッドを測定した. MIBsocket の利用例として, インタフェース切替え機構に応用した. 測定の結果, MIBsocket のオーバヘッドは無視できる範囲であった. また, インタフェース切替え機構は, MIBsocket によりネットワークエンティティの状態を取得できた. ネットワークエンティティの状態変化を MIBsocket から受け取ることによりインタフェースを切替えることができた.

慶應義塾大学 総合政策学部
湧川隆次
ryuji@sfc.wide.ad.jp

MIBsocket: Integrated Control Mechanism for Network Entity in Mobile Network

Network softwares need to know the states of Network Entity, which are changeable in the mobile computing, and to control them consistency one another. The Network Entity states should be adapted to the external changes dynamically such as the relocation of computers.

The purpose of this research is to enable the mobile computer to integrate the control mechanism for Network Entity. The states of Network Entity are passed between kernel and network softwares by messages. The mechanism is implemented as the MIBsocket (a socket) on FreeBSD2.2.7 augmented by the September 1998 release of the PAO patch kit. MIBsocket detects the Network Entity changing and informs it to network softwares. MIBsocket pass the request and reply messages of the Network Entity states to network softwares. The mobile systems are able to reconfigure and adapt the Network Entity states dynamically. We measured, the overhead of MIBsocket, and applied the MIBsocket to switch network interfaces.

As a result, the overhead time of the MIBsocket was reasonable, and we have successfully applied the MIBsocket so that the network software acknowledges the changes, and switches to the available interface.

Keio University
Faculty of Policy Management
Ryuji Wakikawa
ryuji@sfc.wide.ad.jp

目次

図目次

表 目 次

第 1 章

序論

本研究の想定している移動型計算機の背景, 及び目的について述べる. 最後に, 本論文の構成について示す.

1.1 研究背景

小型化及び高性能化, 様々な通信デバイスの登場, そして移動を支援する研究の充実により, 移動型計算機が広く使われるようになった. PDA やラップトップコンピュータは, ワークステーション並の CPU パワーとメモリを持ち, また携帯に十分なほど小型化している. 移動型計算機は, 無線通信やアメリカの Ricochet Wireless Network[1]等の無線型高速データ通信網, 小型で小電力の赤外線, 携帯電話や PHS の様に様々な通信デバイスが利用できる. 今後も次世代携帯電話の IMT2000 や衛星通信が新たな通信デバイスとして注目されている. 移動型計算機を支援するインターネット技術としては, 移動先での IP アドレス取得のための DHCP[1] 移動の透過性を保証する MobileIP[4], 無線環境下における効率の良い TCP[14] の SACK[2], FACK[3] 等といったものがある. 現在 IETF[2]で議論されている Mobile Ad-hoc Network(MANET)[12][13] の実現により移動型計算機で作る自律ネットワークも実現されつつある. 利用者はこれらネットワークソフトウェアや通信デバイスを用いて, World Wide Web や電子メールなどを利用する. しかし, 通信中にネットワークの切断や計算機環境の変化により, 利用者は自ら計算機の設定などを行わなくてはならない.

移動型計算機環境の充実により, 利用者が計算機の移動を気にすること無く常にネットワークを利用できることが望まれる. 計算機自身が移動を検知し, 自動的に移動先の通信環境に最適な設定を行わせるシステムが必要となる.

1.2 研究目的

移動型計算機環境では, IP アドレスや通信デバイスの切替え等の通信環境が変わるような状況が発生する. 移動型計算機環境の最大の特徴は, ネットワークエンティティの状態が頻繁に変わる事である. ネットワークエンティティの状態とは, リンク, IP アドレス, 経路情報, ネットワークインタフェース等のネットワークを利用するために必須なエンティティの状態を指す.

---

[1] www.metricom.com に情報がある

[2] www.ietf.org 参照

特に移動型計算機環境では, ネットワークエンティティの状態を変化させる要素が多い. 例えば, DHCP がインタフェースにアドレスを割り当てた時点でネットワークエンティティの状態は変化する. この他に, 利用者の手動設定, MobileIP, 経路制御デーモン, インタフェース切替えデーモン等もネットワークエンティティの状態を変化させる. 計算機環境が変わることにより, ネットワークエンティティの状態も同時に変化する. また, IP アドレスが変われば経路も同時に変化するというようにネットワークエンティティ自体に相互関係があるため, ネットワークエンティティの状態が連鎖的に変わる. ネットワークエンティティを統一的に管理することは移動型計算機での通信にとって重要である .

移動型計算機は, 移動先での通信環境に適応しなくてはならない. イーサネット, 無線や携帯電話等の複数の通信デバイスを用いて通信を行う. また, 移動透過性を実現するために MobileIP を利用することもある. 移動対計算機では, システム内で自動的に移動を検知して適応されなくてはならない. 現在では, Multihomed Host[5] や MANET の様に, ネットワークエンティティの情報が必要で, ネットワークエンティティの状態や状態変化の情報必要とする研究もある.

現在ネットワークエンティティの管理は, 主に移動しない固定された計算機を想定している. 例えば, Simple Network Management Protocol(SNMP)[11] などのネットワーク管理プロトコルがある. しかし, ネットワークエンティティの頻繁な状態変化は想定されていない等, 移動型計算機で利用するには問題がある. 移動型計算機環境では, 既存の計算機環境とは違いネットワークエンティティの状態を固定できない. 既存の固定された計算機では比較的ネットワークエンティティの状態変化は起きない. ネットワークエンティティの状態変化が起きるのはシステム起動時, 終了時がほとんどである. このため, 管理者が IP アドレスの取得やインタフェースの選択といったネットワークエンティティの管理を行うことができる. しかし, 移動型計算機環境ではネットワークエンティティの状態変化は予想できないため, 管理者による管理は限界がある. 移動型計算機環境に対応したネットワークエンティティの統一的管理は, 既存のネットワークエンティティ管理システムでは不十分である.

これらの問題を解決するため, 本研究では移動型計算機を想定したネットワークエンティティの動的な管理機構を設計・実装・評価を行い実現する. 計算機は, 移動することによるネットワーク環境の変化を, ネットワークエンティティの状態変化を監視することにより知ることができる. ネットワークエンティティの状態を管理することにより, ネットワークエンティティの状態に応じた計算機全体のシステムを動的に構築することができる. 計算機自身が自動に計算機環境に最適なネットワークエンティティの状態にすることができる. 具体的には, 動的に変化するネットワークエンティティの状態を取得, ネットワークエンティティの設定, そして状態変化を通知するシステムを構築する. また, 取得するネットワークエンティティの情報のフィルタリングやネットワークエンティティの情報の変更に対する許可を与えることにより, 利用者は特定のネットワークエンティティの情報のみを取得, 設定を行える. ネットワークエンティティの状態はカーネル内で Management Information Base(MIB)[9] のデータ構造に基づいて保持する. MIB に準拠することにより汎用性が増す. 動的にシステムの設定を変更する機会の多い移動型計算機を想定して, システム構成の把握及び変更が可能となる. 本研究では, カーネルとユーザ間のネットワークエンティティの状態のやりとりに, ソケットを用いて実現する.

1.3　本論文の構成

本論文は以下のように構成される. 第2章で, 移動型計算機及びネットワークエンティティ管理の現状と問題点について述べ, 第3章では, それら問題点を既存のインターネットでどのように解決しているのかを示す. ネットワークエンティティの管理について述べる. 第4章では問題点をシステムモデルを用いて示し, 第5章で問題解決のためのシステム設計について述べる. 第6章で本システムにより問題を解決することにより実現可能なことあるいは本システムが応用できる事について示す. 第7章では本システムの具体的な実装方法について述べる. 第8章で, 実装を行った本システムが実際に実現できているのかを評価し, 本システムの応用としてインタフェース切替えを用いてインタフェース切替え時の処理をICMPとTCPを用いて評価する. 最後に, 第9章で, まとめと今後の課題についての考察を行う.

第 2 章

移動型計算機におけるネットワークエンティティ管理の現状と問題点

本章では, 始めに移動型計算機環境及びネットワークエンティティ管理の現状について述べる. その後, 移動型計算機におけるネットワークエンティティ管理の問題点について述べる.

2.1 移動型計算機環境におけるネットワークエンティティ

移動型計算機環境は, ハードウェアである通信デバイスからソフトウェアの面からも進歩している. しかし, 移動することに起きるネットワークエンティティの状態変化に対応する機構は未解決のまま残されている. そこで, 移動型計算機での通信の現状の問題点について述べ, その後ネットワークエンティティの状態変化に対するネットワークエンティティ管理の現状について述べる.

2.1.1 ネットワークエンティティとは

ネットワークエンティティとは, ネットワークに関する機能 (プロセス) をモデル化したもので一つ以上存在する. 既存のインターネットにおけるネットワークエンティティは, インターネットの 4 階層モデルの TCP/UDP[15],IP, インタフェース層にある. またその他に, 経路情報や通信に必要なネームサーバアドレス等もネットワークエンティティの情報である.

TCP/UDP 層のネットワークエンティティ

TCP は, ポート番号と IP アドレスをもとにコネクションを維持する. ポート番号や IP アドレスはネットワークエンティティの状態である. TCP は, コネクションの輻輳制御やフロー制御, パケット再送機構があり, 信頼ある通信を保証する. UDP は, TCP とは違い信頼性を保証しない. また, 再送やフロー制御機構も無い. TCP のコネクションの状態のコネクション数はネットワークエンティティの状態である. UDP の宛先のポート番号やアドレスもネットワークエンティティの状態をである.

IP 層のネットワークエンティティ

IP 層では IP アドレスを計算機の識別子として用いる. IP アドレス, ネットマスクアドレス, ブロードキャストアドレスはネットワークエンティティの状態である. また, IP 層で処理したデータ

グラム数, エラーのあったデータグラム数, TTL(Time To Live) の数も状態である. その他にも, IP データグラムをフォワードするか等の情報や IPsec 等のセキュリティーに関する情報も IP 層のネットワークエンティティの状態である.

インタフェース層のネットワークエンティティ

インタフェース層の MAC アドレスや, インタフェースのリンクの状態, MTU, インタフェースのタイプ等は全てネットワークエンティティの状態である. また, 他と同じくインタフェースで処理したパケット数やエラー数などもネットワークエンティティの状態である.

経路のネットワークエンティティ

経路のネットワークエンティティの状態は, 宛先の計算機と通信するために宛先アドレスとデータを転送してくれるネクストホップルータの対応などの情報である. また, 経路制御機構の経路情報の獲得は, ネットワークエンティティの状態を変える. 新しい経路により通信経路が変わるためである.

2.1.2 ネットワークエンティティ管理の現状と問題点

既存の固定された計算機のネットワークエンティティの管理は, SNMP 等を用いて管理者が行っている. SNMP はネットワーク管理が主目的であり, 遠隔 ・分散管理を得意とする. 計算機内のネットワークエンティティの管理はその一貫として行われている. SNMP に関しては第 3 章で詳しく述べる. また, WEB を用いたシステムの集中管理も行われている.

移動型計算機では SNMP 等を用いたネットワークエンティティの管理はされてない. 移動型計算機では移動する特性上, 遠隔 ・分散管理は難しく, 単一の計算機自身の管理が求められる. ネットワークエンティティの変化を知らせる SNMP のトラップ機構は, トラップに関する規格以外にベンダーが独自に追加しているものが多く利用することができない. また, トラップ機構が無いデバイスも多い. 移動型計算機環境は, 時と場所により通信デバイスも使う技術も異なる. 動的なネットワークエンティティの状態変化に対応し, これらを自動的に選択し利用できるネットワークエンティティの状態管理機構が必要である. 現在の移動型計算機環境では, 利用者がネットワークエンティティを管理している. 移動の後, 利用者は移動先の環境に適した通信デバイスを決定する. IP アドレスを DHCP 等で取得し, ネームサーバのアドレスの変更を行う. つまり, 利用者がネットワークエンティティの状態把握及び管理を全て行わなければならない. 移動型計算機は, 様々な選択肢から環境に最適なシステム構成を選択しなくてはならない.

会議室, 外出先, あるいは乗物移動等の移動型通信環境には, 統一性が無い. 利用者は複数のネットワークインタフェースやネットワークソフトウェアを用いて通信を行う. それらの決定は全て利用者が行う. 利用者は決定後, それらをシステムに反映させるために各種設定を行い通信を始める. 利用者は通信が切れたなどを検知すると, 切断を回避するためにシステム構成の再決定を行い設定をする. ネットワークエンティティの状態を管理するために利用者が, ネットワークエンティティの状態を常に監視する必要がある. しかし, 利用者の判断が正確である保証もない. 正確な判断を行うためには, インタフェースからネットワークソフトウェアへの十分な知識が利用者の求められる.

2.2 移動型計算機における具体的な問題点

上述の移動型計算機環境で解決すべき問題について具体例をあげる.

2.2.1 既存の複数の通信デバイスに関する問題

移動型計算機では通信デバイスとして無線, 赤外線, 携帯電話あるいは有線を用いる. 環境に応じてそれら通信デバイスを使い分ける. これら通信デバイスは, ハードウェア的な特性がそれぞれ違うため, 扱い方も異なる. 例えば, 無線では無線基地局が無ければ基本的に通信は行えない. また高遅延である. 赤外線は直行性の光で障害物に弱い. 携帯電話では課金制度があり低帯域である. 移動型通信ではこれら複数の通信デバイスを使い分ける. 現在, それら通信デバイスを管理するのは利用者である. 利用者が通信デバイスを変えるたびに設定を行っている. つまり, 通信デバイスの選択は基本的に利用者が決定している. デバイスの選択決定の基準はそれぞれのデバイスによって異なる. 例えば, 無線では伝播強度, 赤外線では通信範囲の制限が基準となる. これらは全てユーザが監視して選択決定の判断を下している.

インタフェースの選択をシステムで動的に行う事はできない. なぜなら通信デバイスの状態及び計算機環境の情報を取得し管理するための手段が無いからである. 既存の固定された計算機の管理は, SNMP である程度行える. しかし, 移動型計算機における頻繁なネットワーク環境の変化, そしてオーバヘッドがあるため実際は難しい. 計算機が常にネットワークへの接続性を維持するためにリンクの状態等のネットワークインタフェースの状態に応じて切替えることが望まれる. 移動型計算機が不安定な環境下での通信を更に広げるためには, 複数デバイスを環境に応じて動的に管理する機構が必要である.

2.2.2 移動型技術の分散に関する問題

移動を支援する技術のほとんどがInternational Standards Organization(ISO) で示されたOpen Systems Interconnection(OSI) 階層参照モデル[6] のそれぞれの層で独自に研究され, またそれら技術は階層間で独立しているためシステム全体として分散されている. 図2.1が移動型計算機技術とOSI 階層参照モデルとの組合せである.

OSI 階層参照モデルはそれぞれの層で機能を切り分ける事により通信プロトコルを容易に設計し実装することを可能にしている. しかし, 場合によってはそれぞれの層で協調して動作しなくてはならない. その場合, 各層にまたがるコントロールブロックが必要である. コントロールブロックは, 各層を統合する. 現在, コントロールブロックは無く各層は互いに独立している. 既存のネットワークソフトウェアはOSI の各層に対応して作られているものが多い. しかし, これらのネットワークソフトウェアは, ネットワークソフトウェア同士で協調した動作が必要である. 技術を組み合わせることにより動体計算機環境をさらに支援する事ができる. また, 通信環境の変化に応じて必要なネットワークソフトウェアが異なるため環境に応じた組合せを選択しなくてはならない. しかし, 動的に組み替えを行うためにはソフトウェア間での協調動作が不可欠である. 既存のネットワークソフトウェアを協調動作させることができる機構が必要である.

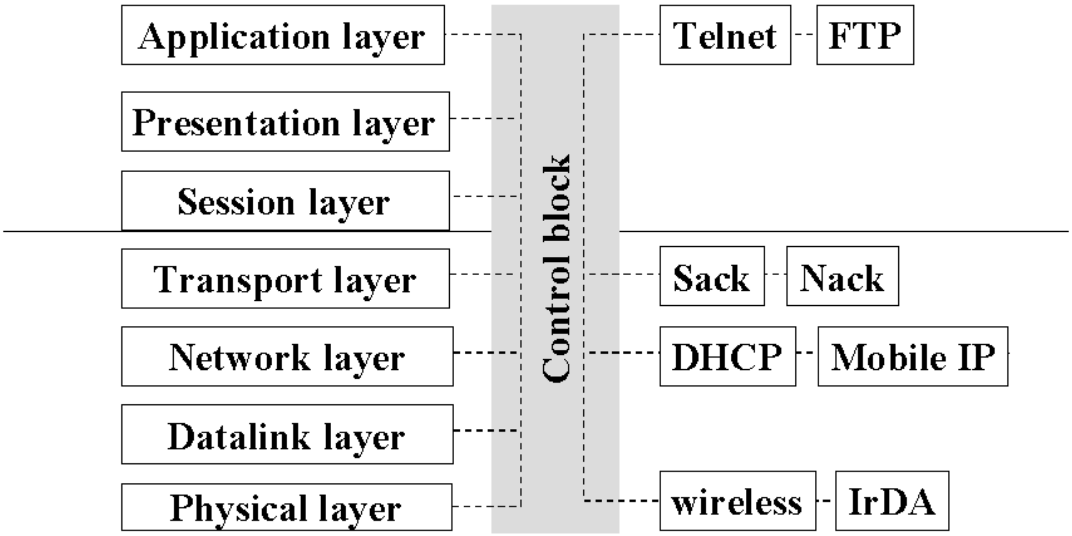


図 2.1: OSI 階層参照モデル

第 3 章

関連研究

本章では, ネットワークエンティティの状態管理に関する研究について述べる. ネットワークエンティティ管理の研究事例としてSNMPを取り上げる. また, インターフェース切替えの研究についても述べる.

3.1 ネットワークエンティティの統一的管理に関する研究

ネットワークエンティティの統一的な管理の研究は以下のようなものがある. また, これらの研究ではインタフェース切替えの研究を行っているものもある.

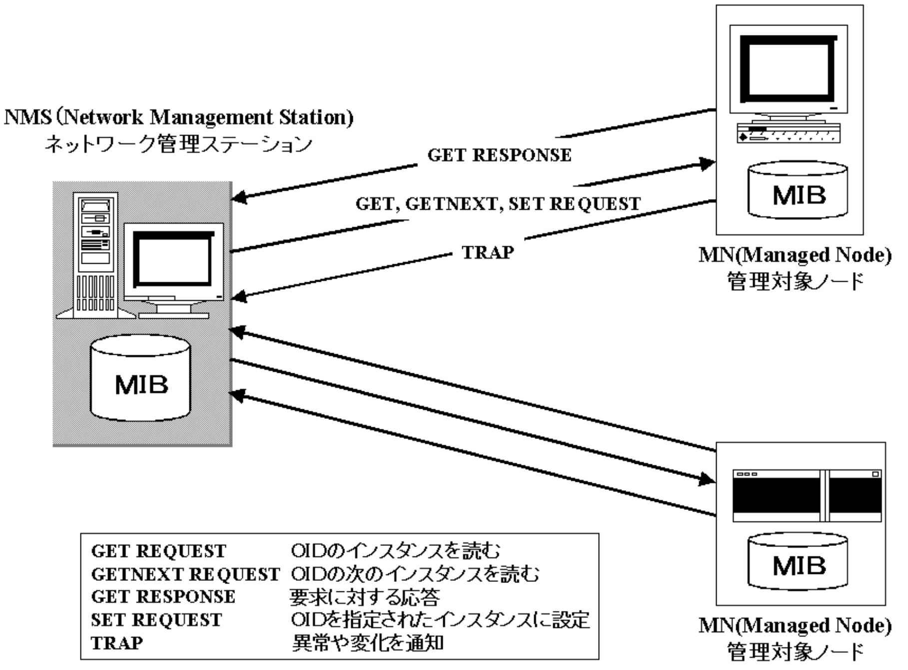
Oregon Graduate Institute で行われているQuasarプロジェクト[21]では，利用インタフェースをネットワーク環境に応じて切替え, インタフェースの帯域に応じたデータ量を制御する研究を行っている. ネットワークエンティティの状態をICMPのエコーリクエストを用いて把握する為，ネットワークエンティティの頻繁に変化する移動体通信環境では，ネットワークエンティティの状態を知るのにコストが掛かる.

Stanford大学のMosquitoNetプロジェクト[19][20]では移動体通信環境におけるオペレーティングシステムとアプリケーションの問題について研究している. 特にMobileIPを用いた移動透過性の研究を中心に行っている. また, 通信環境の変化に応じたシームレスなインタフェース切替えの研究も行っている. インタフェースを切替えることにより起きるトランスポート層での切断も回避する研究も行われている. これらを実現するために, アプリケーションとネットワークエンティティの間で透過的なシステムを作ることを目標としている. アプリケーションはネットワークエンティティの変化を気にすること無く動作することができる. しかし, 移動型計算機の常にネットワーク接続を目標としているため, ネットワークエンティティの管理機構に汎用性が欠ける.

CMU大学のOdysseyプロジェクト[25]では, アプリケーションとオペレーティングシステムの間が協力して動作するシステムを研究している. [24]では, Odysseyのデザイン及びプロトタイプの実装を行って評価している.

3.2 ネットワークエンティティ管理に関する研究

ネットワークエンティティ管理に関する研究はOSIやIEEEでも行われているが, 今回はインターネット環境で広く使われているSNMPについて説明する. また, SNMPで用いられているMIBについても説明をする.



GET REQUEST	OIDのインスタンスを読む
GETNEXT REQUEST	OIDの次のインスタンスを読む
GET RESPONSE	要求に対する応答
SET REQUEST	OIDを指定されたインスタンスに設定
TRAP	異常や変化を通知

図 3.1: SNMP のシステム概要

3.2.1 Simple Network Management Protocol(SNMP)

現在インターネットで広く使われているのが SNMP である. SNMP は TCP/IP プロトコルを用いて動作する標準化されたネットワーク管理プロトコルである. ネットワーク管理プロトコルとは, 標準化したプロトコルによって, ネットワーク機器にアクセスし状態を取得, 設定するものである. これまで, SNMP 以外にネットワーク管理プロトコルとして CMIS/CMIP[1], CMOT[2]等が開発された.

SNMP にはネットワーク管理ステーション (NMS) と管理対象ノード (MN) で構成される. ネットワーク管理ステーションはネットワークの制御 ･監視する計算機である. SNMP を用いて管理対象ノードと通信を行う. 管理対象ノードはネットワーク管理ステーションから要求を受けそれを実行する計算機あるいはデバイスを指す. 2 つの計算機間では MIB と呼ばれる同一データベース構造を用いる. MIB については次に説明する. ネットワーク管理ステーションは管理対象ノードが持つ変数を取得したり設定する事ができる. また, 管理対象ノードはシステム内で起きた状態変化をネットワーク管理ステーションに通知するための機構としてトラップがある. 具体的なシステムイメージは図 3.1である.

NMS は, 近隣の MN の管理を行う. NMS は MN の MIB 情報の値を取得したり設定したりする事によりネットワークを管理する. それぞれ図 3.1にあるように要求 ･応答で取得や設定が行われる. また, ネットワークエンティティの状態変化にはトラップを用いる. 例えば, ルータである MN のインタフェースが落ちた時 MN は, NMS に対してインタフェースのリンクが落ちたという情報をトラップとして送る. このトラップにより NMS はルータである MN のインタフェースが

---

[1] Common Management Information Service/Common Management Information Protocol

[2] CMIP over TCP

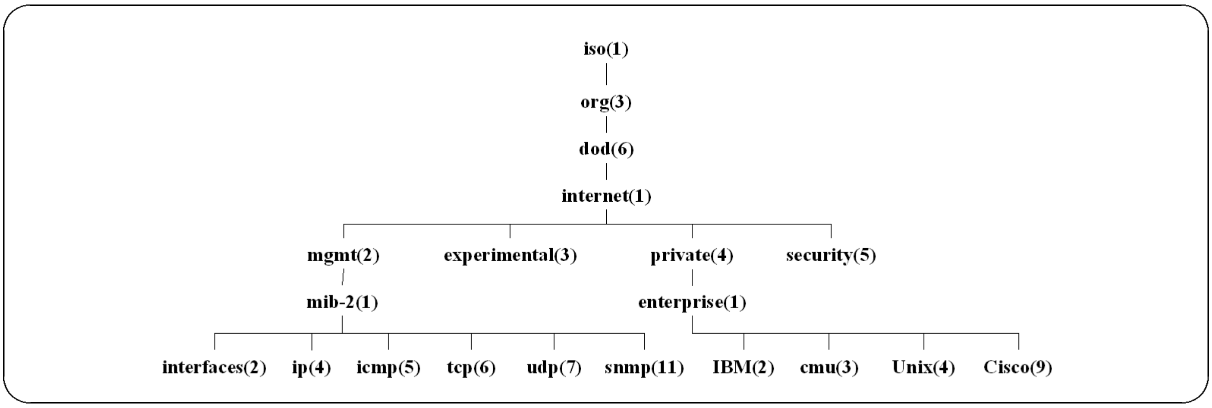


図 3.2: MIB の木構造

落ちたという事を知る事ができる. この様にして, NMS は複数の MN を遠隔から管理する事が可能である. また, NMS を複数台設置する事で管理の分散もできる.

具体的には, SNMP は UDP を用いたアプリケーション層のプロトコルとして定義されている. ポート番号は, 161, 162 が使われる. 現在 SNMPv2 が最も新しい. SNMP と SNMPv2 の違いはセキュリティーの強化, バルク (一括) データ転送が可能になった事などである.

3.2.2 Management Infomation Base(MIB)

MIB とは, 様々な情報を管理するデータベースの構造である. MIB の特徴は, 木構造であり, どんなタイプの情報でも一貫した形式で表す事ができることである. MIB では全ての情報にオブジェクト識別子が割り当てられ, その識別子と情報の値は必ず一致している. 図 3.2でインターネットに関連した主な MIB の木構造を示している.

木構造の根は iso で始まる. ルートの iso からそれぞれ枝が伸びる. インターネット環境で重要であるのは internet 以下にまとめられている. 特に mib2 の下にインターフェース, IP, TCP, UDP 等の情報がある. また private 以下は MIB で定義されていなく独自に定義した MIB を表すためである. よって, 今後必要な情報で MIB で定義されていない場合は private 以下で割当を行う. internet 以下のサブツリーは IANA が管理を行っている. MIB の更新は RFC を用いて適宜行われている.

オブジェクト識別子

オブジェクト識別子とはオブジェクトを識別するための物である. オブジェクト識別子は, ドットで区切られている整数を用い, 上述のように木構造である. ある情報のインスタンスは必ず特定のオブジェクト識別子で識別できなくてはならない. しかし, あるオブジェクト識別子に複数のインスタンスが存在するのは許されている. 経路情報などのオブジェクトは, 一つのオブジェクトに対して複数のエントリーができる可能性があるからである.

オブジェクト識別子に対する複数インスタンスの特定

MIB 構造で, オブジェクト識別子に対して複数のインスタンスが存在する場合がある. この時, オブジェクト識別子から特定のインスタンスを得るための仕組みについて説明する. 例え

ばインターフェースの MTU を知りたい時, ifMtu に割り当てられているオブジェクト識別子 (1.3.6.1.2.1.2.2.1.1) を指定するとシステムにあるインタフェース全て (ループバックやイーサリンク等) の MTU の値が得られる. イーサリンクの MTU のみが知りたければ, オブジェクト識別子の後に特定のインターフェースのインデックスを付加する事により特定の値のみを参照できる. インターネットに関係する, 主なオブジェクトインスタンスの特定の方法を表 3.1にした. この特定の方法については後で説明する SMI 形式でかかれた文書により定義されている. 文書は RFC として配布されている.

表 3.1: オブジェクトインスタンスの識別

オブジェクト識別子	インスタンス名	サフィックス
1.3.6.1.2.1.2.2.1.1	ifEntry	ifindex
1.3.6.1.2.1.4.20.1	ipAddrEntry	ipaddress
1.3.6.1.2.1.4.22.1	ipNetToMediaEntry	ifindex + ipaddress
1.3.6.1.2.1.4.24.2.1	ipForwardEntry	ipaddress + proto + policy + nexthop ipaddress
1.3.6.1.2.1.6.13.1	tcpConnEntry	localipaddress + localport + remoteipaddress + remoteport
1.3.6.1.2.1.7.5.1	udpEntry	localipaddress + localport

SMI

SMI とは RFC1155 で定義されている管理情報構造であり, 管理情報の編成, 名前付けそして記述を行う. MIB の情報は全て, SMI で記述されている. SMI では, 管理対象オブジェクトに対して, 名前, 書式, エンコーディングの定義が必須とされている. 名前はオブジェクト識別子で, 書式はデータタイプ, エンコーディングとは管理対象オブジェクトのインスタンスをネットワークを介して参照する時の方法を指す. SMI では, 抽象化書式のために ASN.1[16] を用いる.

第 4 章

システムモデル及びアプローチ

本章では, 我々が前提としているシステムモデルと問題解決のアプローチについて述べる.

4.1 システムモデル

移動体計算機に必要なのは, ネットワークエンティティの変化を検知し通知するメカニズムである. また, ネットワークエンティティの情報はネットワークソフトウェアとカーネルで共有し, 一貫性を保ちながら動的に変更できなくてはならない.

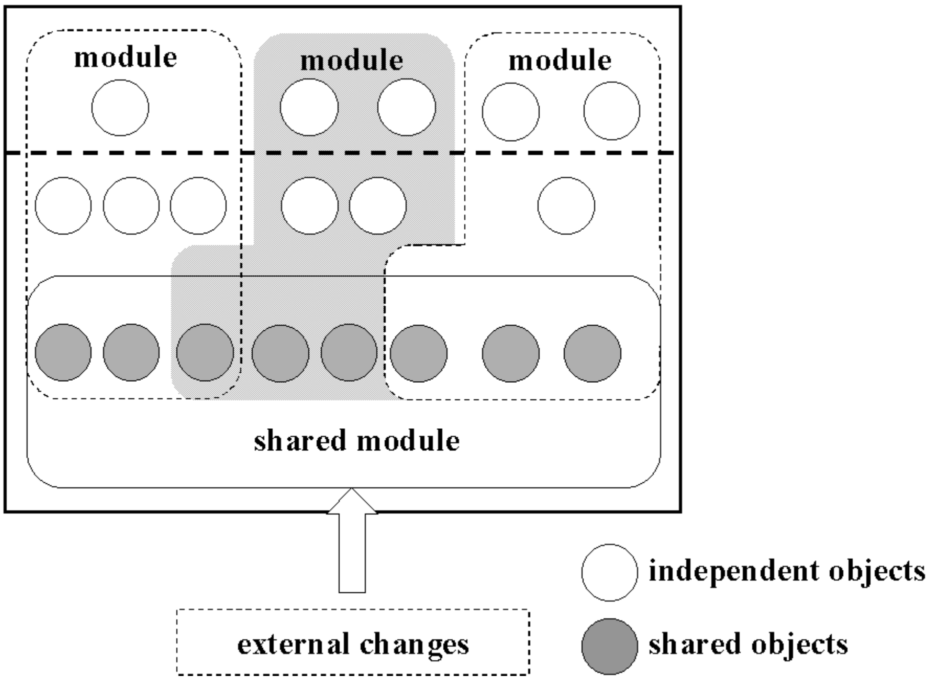


図 4.1: システムモデル

計算機システムは, ネットワークソフトウェアやカーネルのような多くのモジュールで構成されている. モジュールは, 自身の処理のためにオブジェクトが必要である. オブジェクトは, ネットワークエンティティやモジュールの情報を指す. 図 4.1は, モジュールとオブジェクトの関係を

示している. モジュールは 2 種類のオブジェクトを持っている. 一つは independent object でモジュールに固有なオブジェクトである. もう一つはシステム内及びネットワークソフトウェア間で共有している shared object である. shared object は, 例えばネットワークエンティティなどのシステムの情報である. 現在のシステムでは, shared object はネットワークソフトウェアからは参照できない情報である. また, 既存の計算機では shared object がインターネットに最適化され固定されているものが多い. なぜなら, 固定されたシステムでは shared object は動的に変化しないという前提で扱われているからである. しかし, 移動体計算機では shared object が動的に変化するため, ネットワークソフトウェアは shared object の情報を参照でき変化したという情報を検知しなくてはならない.

ネットワークソフトウェアは現在の計算機環境において最適なネットワークエンティティの状態を望む. もし計算機環境が変われば, ネットワークエンティティを最適な状態にそして計算機環境に適応するためにネットワークエンティティの状態を変更する. しかし, ネットワークソフトウェアにとっても計算機システムにとっても最適なネットワークエンティティの状態や適応の仕方が違うこともある. このように, ネットワークソフトウェアが動的にネットワークエンティティの状態を計算機環境に最適にする要求は高まっている. 例えば, 4.4BSD における *setsockopt()* システムコールを考える. このシステムコールはソケットのオプションを設定するものである. 現在, オプションの種類が増加している. これは, ソケットを介してネットワークソフトウェアが計算機環境に適した通信を行うために追加されている. ソケットは, 図 4.1のモジュール内の真中位置するオブジェクトにあたる. しかし, ソケットだけでネットワークエンティティをコントロールするには限界がある. ソケットはネットワークエンティティの一部しか参照できないからである.

以上をうまく解決するために統一したネットワークエンティティの状態管理を行う.

4.2 アプローチ

第 2章で述べた問題を解決するためにはネットワークエンティティの統一された管理機構が必要である. ネットワークエンティティを管理するため, ネットワークエンティティの状態を取得する機構が必要である. また, この状態を取得後, ネットワークエンティティの状態に応じた設定を統一的に行える機構も必要である. アプローチとして以下の 3 つの方法が考えられる.

- **Shared memory**
  複数のネットワークソフトウェアとカーネルが直接ネットワークエンティティのメモリにアクセスする
- **Polling**
  ネットワークエンティティをファイルシステムとして提供し, ネットワークソフトウェアがそれらをポーリングする
- **Message Passing**
  カーネルとネットワークソフトウェアの間でネットワークエンティティに関する情報をメッセージでやりとりする

ネットワークエンティティが共有できていれば一貫性は保証できる. 一方で, 同時にネットワークエンティティの変更を行う場合に競合が起きる.

表 4.1は, 3 つのアプローチの長所と短所を示している. ネットワークエンティティは **Shared Memory** を用いることにより一貫性を保ちながら共有できる. しかし, ネットワークエンティティ

が変化した場合, ネットワークソフトウェアに変化を通知することができない. **Polling** ではCPUやメモリ等のシステムの資源を無駄に消費する. また, 一貫性を保障することが難しい. **Message Passing** は, ネットワークエンティティの変化や変更をメッセージのやりとりで行うため競合が起きにくい. メッセージは到着順に処理されるからである. また, ネットワークエンティティの一貫性もネットワークエンティティの状態変化をメッセージで知らせることにより保つことができる. このため, **Message Passing** がネットワークエンティティの状態に一貫性を保ち管理する目的には最適である. 本システムでは, **Message Passing** を採用する.

表 4.1: それぞれのアプローチの特徴

method	consistency	up-down call	overhead	ap plicable
Shared Memory	○	×	×	×
Polling	×	×	○	○
Message Passing	○	○	△	△

4.4BSD[8] では, ネットワークエンティティの情報はカーネル内にある. そのため, **Message Passing** を実現するためにはカーネルとのインタフェースを決定しなくてはならない. 現在の4.4BSD では, ソケットとキャラクタデバイスが, メッセージをやりとりするカーネルとユーザ空間のインタフェースとして考えられる. ソケットは, 複数で書き込みも読む事も可能である. また, フィルタリングや許可といった拡張も容易である. デバイスは, 複数でデバイスに書き込む場合の処理に注意する必要がある. 結局ソケットが本システムを実現するための仕組みを持ち, かつ容易に実現できるため採用した.

第 5 章

システム設計

第4章でのシステムモデルをもとに，ネットワークエンティティの状態変化に対応する管理システムの設計について述べる．

5.1 システム (MIBsocket) の概要

ネットワークソフトウェアの状態や状態変化をネットワークソフトウェア等に通知するために，ソケットを用いる．ソケットを用いて，ネットワークエンティティの状態を通知する．このネットワークエンティティとソフトウェアとのインタフェースであるソケットを MIBsocket と名付ける．

MIBsocket では以下のことができなくてはならない．

- MIBsocket は複数のネットワークソフトウェアで利用できる．
- ネットワークエンティティの一貫性を保つ．一貫性は MIBsocket に許可を与えることにより可能になる．
- メッセージのフィルタリングが可能である．ネットワークソフトウェアは，必要なメッセージのみを受け取ることができる．

MIBsocket は，フィルタリングと許可という機能がある．MIBsocket では，ネットワークエンティティの情報を取得するためのメッセージ以外は，全てを受け取る事が可能である．フィルター機能は，MIBsocket から必要な情報のみを受け取るために用いる．許可の機能は，MIBsocket の処理に制限を与えることができる．ネットワークエンティティは通信に関して重要な情報を含むため，全ての MIBsocket が設定を行えるのは問題がある．また，複数の MIBsocket が同時にネットワークエンティティの設定を行った時に起こる競合を，許可の機能を用いて回避することができる．これは，一つの MIBsocket のみに許可を与えることにより実現できる．

図 5.1は，OSI の 7 階層モデルにおける MIBsocket の役割を示している．OSI の 7 階層モデルのコントロールブロックとして MIBsocket を用いる．MIBsocket はネットワークエンティティ及びネットワークソフトウェアの唯一のインタフェースとして用いる．それぞれのネットワークソフトウェアはネットワークエンティティの状態を介して互いに協調動作することができる．例えば，図 5.1で DHCP がネットワークエンティティに対して行った処理はネットワークエンティティコントロールデーモンが知ることができるため DHCP が動作するとネットワークエンティティコントロールデーモンも同時に動作することができる．

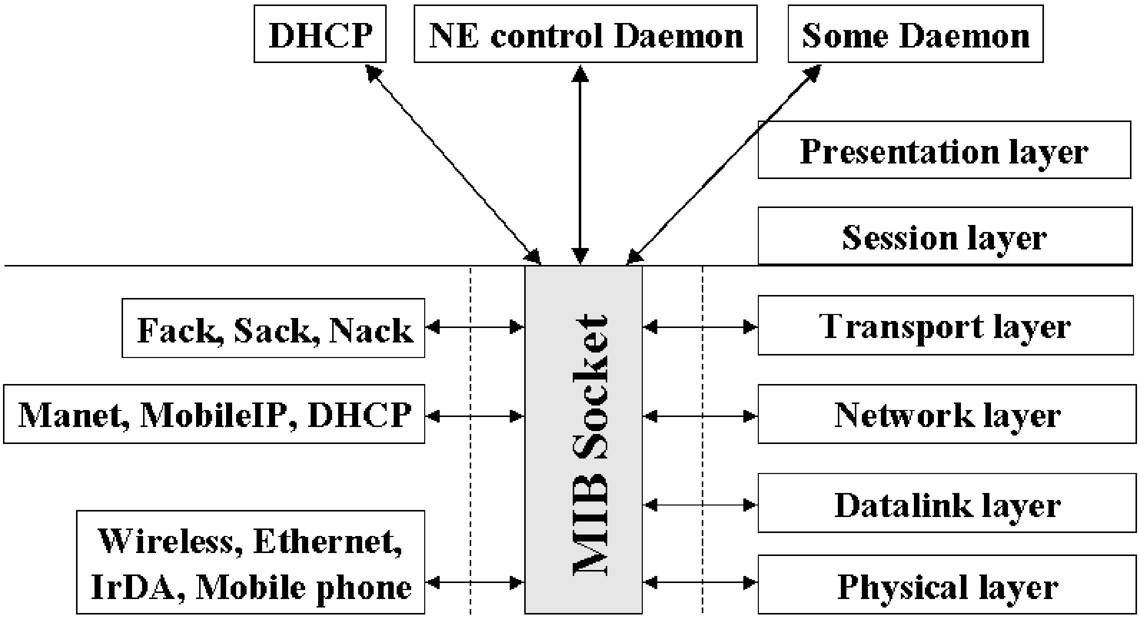


図 5.1: OSI モデルにおける MIBsocket

本システムの, 全体像を図 5.2に示す．MIBsocket を, ネットワークエンティティに対する唯一のインタフェースとして用いる事により, 移動体技術を統合する事ができる．ネットワークソフトウェアが互いに協調してネットワークエンティティを共有することができる．通信環境変化に応じて様々な技術を組み合わせて計算機システムを構成できる．

MIBsocket で扱うネットワークエンティティの情報は, MIB のデータ構造に準拠している．MIB で管理されたオブジェクトの情報は, MIBsocket を用いて取得, 設定する事ができる．ネットワークエンティティの状態変化を検知すると MIBsocket は, 状態変化を検知しトラップとしてメッセージを送る．これらを一つの MIBsocket で行うため, 3 つのモードを定義した．“set mode”, “get mode”, そして “trap mode” である．”set mode” はネットワークエンティティを設定する時に使われる．”get mode” はネットワークエンティティの情報を取って来る時に使う．最後の”trap mode” は, ネットワークエンティティの変化をネットワークアプリケーションに通知する時に用いられる．これらのモードについては後で詳しく述べる．

図 5.2では 2 つのネットワークエンティティの状態の取得と設定が必要なデーモンがいる．それぞれ MIBsocket を開いてネットワークエンティティの状態を取得している．しかしデーモン 2 は IP のみの情報が必要なためフィルタリングを行っている．よってデーモン 2 は MIBsocket から IP に関する情報しか得る事ができない．また許可が Get と Trap のみに与えられているのでデーモン 2 は全ての設定に関する権限は持っていない事になる．この場合, MIBsocket に設定の要求をするとエラーが返る．同時に他の MIBsocket は, 設定を行ったという情報が送られる．

DHCP のようなインターフェースのアドレスを取得するデーモンがネットワークエンティティの状態を直接設定するのではなく, アドレスをもらったという情報を MIBsocket を開いている管理デーモンに通知し, そこで判断して設定を行わせたい時に有効である．もし, 図 5.2のデーモン 1 が管理デーモンだとすると, デーモン 2 があるネットワークエンティティのオブジェクトに対して設定しようとするとデーモン 2 にはエラーが返る．管理デーモンにはデーモン 2 が設定したメッ

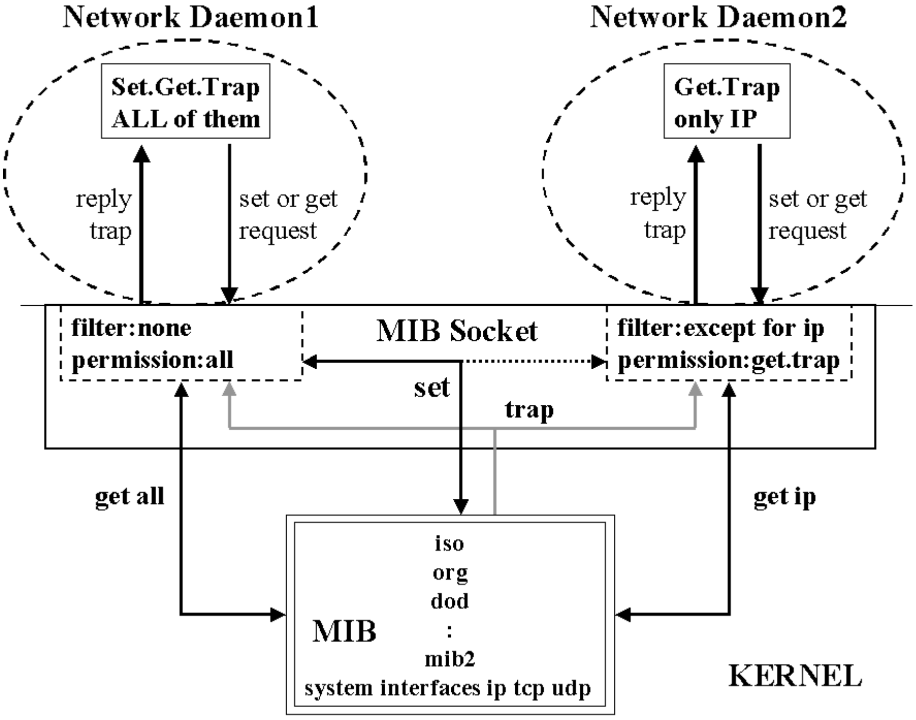


図 5.2: MIBsocket を用いたシステム

セージが送られるので知る事ができる. この管理デーモンに送られるメッセージには設定に関する全ての情報を含んでいるので, 管理デーモンはデーモン 2 の代わりに設定を行える.

5.2 MIB を用いたデータ構造

MIBsocket は, ネットワークエンティティの情報を SNMP 等で用いられている MIB で管理している. MIB を用いる事によりネットワークエンティティのオブジェクトの情報を抽象化することができ, 汎用性も高まる. また, 本機構の拡張として扱う情報を追加する場合でも MIB に準拠しているため容易に行える. MIB ソケットでは, MIB で定義されているオブジェクト識別子を用いてインスタンスの取得・設定・トラップを扱う. 例えば, interface の MTU は, ifMTU のオブジェクト識別子である 1.3.6.2.1.2.2.2.1.4 を使って実際のインスタンスを参照できる.

MIBsocket で設定できるオブジェクトは, MIB で定義されている各々のアクセス許可を参照している. つまり MIB の定義で”read-write” だと取得と設定ができ, ”read only” だと取得しか行えない. 利用者が, ”write” の許可が無いオブジェクトに設定を行いたい時は, プライベート MIB にそのオブジェクトを新たに定義して追加すれば良い.

5.3 MIBsocket とメッセージの仕組み

図 5.3が, MIBsocket で扱うメッセージである. オブジェクトのインスタンスを取得する場合はメッセージヘッダに続いてデータが入る. 基本的に一つのインスタンスであればサイズの制限は

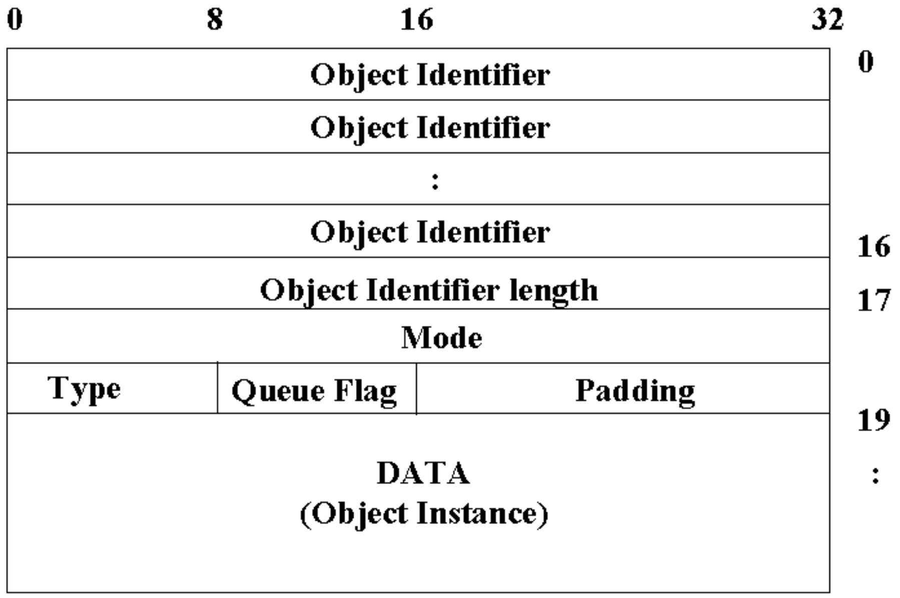


図 5.3: MIBsocket のメッセージ形式

無い. メッセージヘッダには順にオブジェクト識別子, オブジェクト識別子の長さ, モード, qflag が続く. qflag はメッセージが複数続く場合はフラグが立ち, メッセージが次に続かない場合はフラグが立たない. qflag により MIBsocket は, 要求に対する複数の応答を要求者に届けることができる. オブジェクト識別子の書式フォーマットは, ASN.1 のエンコーディングのフォーマットに従う.

MIBsocket が扱うメッセージは, 3 つのモード (“Set mode”, “Get mode”, “Trap mode”) を持つ. “Set mode” はメッセージのオブジェクト識別子で示されたインスタンスを, 計算機に設定すると言う事を意味する. この場合, 設定が成功されれば設定はカーネル内に反映される. “Get mode” はメッセージのオブジェクト識別子で示されたインスタンスを取得するという意味である. “Trap mode” は, MIB で管理しているネットワークエンティティの状態が変化したとき, それに対する通知メッセージに使われる. 表 5.1にそれぞれのモードの主な特徴をまとめた.

表 5.1: MIBsocket のモード

モード	意味	対象オブジェクト
Set mode	オブジェクトインスタンスの設定	write 許可があるもの
Get mode	オブジェクトインスタンスの取得	read 許可があるもの
Trap mode	オブジェクトインスタンスの変化の通知	リンクの状態, 経路情報の変化 インターフェースの IPaddress の状態

5.4 MIBsocket のコントロール

MIBsocket のコントロールは基本的に ioctl を用いて行われる．具体的なコントロールとしては，メッセージのフィルタリングとモードに対する許可のコントロールを行う．メッセージのフィルタリングを用いる事により MIBsocket は，トラップのメッセージのみ，あるいは IP と TCP[14] 等の指定したメッセージのみ受け取る事ができる．これにより，MIBsocket を開くアプリケーションは必要な情報のみを受け取れる．

許可は，モードに対して行われる．“Set mode” の許可が無いと，MIBsocket は “Set mode” を扱えない．“Get mode” に対して許可が無いと，MIBsocket を介してのオブジェクトインスタンスの取得は行えない．しかし，“Set mode” の許可が無くて設定を行うと，設定はカーネル内に反映されず，エラーが戻る．設定を失敗した以外の MIBsocket にも失敗したという”Set mode” のメッセージが届く．

以上のフィルタリングと許可を組み合わせて MIBsocket はコントロールする．表 5.2に，組み合わせ例を示す．

表 5.2: MIBsocket のコントロール例

フィルタリング	Set	Get	Trap	可能な事
none	○	○	○	全て可能
ip	○	○	○	IP に関する処理は行えない
none	×	○	○	Set はできない
ip	×	○	○	Set 及び IP に関する情報は取得できない

5.5 トラップ機構

図 5.4がトラップ機構である．つまり，ネットワークエンティティの状態が変化した時に変化した事をユーザに知らせるための機構である．システムからの割り込みである．トラップでは，インターフェースのリンクの情報，インタフェースの持つ IP アドレスの情報，経路情報に関する変化を取る．つまり，リンクが落ちた場合にはリンクが落ちたというトラップが全ての MIBsocket に通知される．各々の変化に対する情報は，デバイスドライバやルーティグソケットから情報を獲得する．トラップは，全ての MIBsocket に通知される．トラップメッセージはネットワークエンティティのオブジェクトの状態変化を通知する物であり，トラップを受けっとた後の処理は MIBsocket からトラップメッセージを受け取る側に任されている．

5.6 ネットワークエンティティの情報の取得機構

ネットワークエンティティの情報の取得はカーネル内に持っている MIB のデータから情報を取得する．MIB のデータへのアクセスは MIB のオブジェクト識別子を用いて行われる．オブジェクト識別子を指定したメッセージを MIBsocket に書き込むと，オブジェクトのインスタンスが含まれたメッセージが返ってくる．もし，指定したオブジェクト識別子に対して複数のインスタンスが

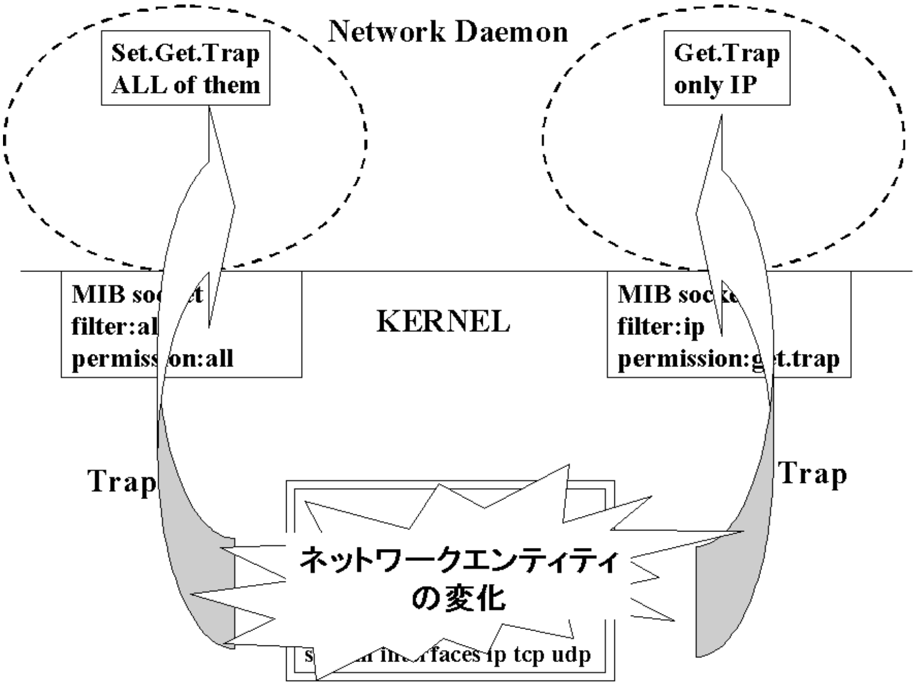


図 5.4: MIBsocket のトラップ機構

存在する場合は全てのインスタンスが返ってくる. その場合はメッセージがインスタンスの数だけ戻ってくる. もし, オブジェクト識別子に対するインスタンスが存在しないあるいはサポートしていない時はエラーが返る.

5.7 ネットワークエンティティの情報の設定機構

設定は, 目的のオブジェクト識別子とそのインスタンスをメッセージとして MIBsocket に書き込む. もし, 設定の許可がある MIBsocket からのメッセージなら設定が行われる. それ以外は, 設定を行った MIBsocket にはエラーが返り, 設定を行おうとしたというエラーメッセージが全ての MIBsocket に送られる. その際, そのメッセージには設定の際のインスタンスも含まれる.

第 6 章

本システムにより実現可能な応用例

本研究によって実現できる事についての説明を行う. 主に, MIBsocket を用いた移動体計算機環境でのネットワークエンティティの管理である. 本論文では, ネットワークエンティティの管理の例としてインタフェース切替えを用いる.

6.1 ネットワークエンティティのコントロールと管理

ネットワークエンティティの管理機構は, インターフェースやIP 等のネットワークエンティティごとに用意する事もできるし, 全てのネットワークエンティティを一括して管理する機構を用いる事もできる. これは許可とフィルタリングを用いる事により実現可能になる. 重要なのは, 管理機構の必要な機能はMIB ソケットの操作とMIBsocket から得られる様々な情報に対するポリシによりネットワークエンティティを管理する事である. そのポリシが経路制御なら, それはgated 等の経路制御機構であるし, インターフェース選択のポリシならインターフェース切替え機構となる. この様に, 単一ホストを想定して管理機構を実現するためのシステムである. 図6.1に計算機の管理機構を示す. ネットワークエンティティとのインタフェースはMIBsocket である.

6.2 インタフェース切替え

ここでは, インターフェースを切替え機構について説明する. 環境の変化により起こる通信切断に対応し, ネットワークインタフェースを切替えて通信を継続する. 図6.2にその概要を示す.

インタフェース切替え機構は, MIBsocket からインタフェースの情報を取得し, システム内に保持する. また, MIBsocket からトラップメッセージを受け取り, ネットワークエンティティの状態変化を動的に知る. それらの情報をインタフェース切替え機構が持つポリシによりインタフェースを選択する. もしインタフェースを切替えるという判断が得られれば, 実際に切替えるためにMIBsocket を通じて設定を行い. また, IP アドレスの取得のためDHCP クライアントを用いる. DHCP クライアントはMIBsocket からのリンクの状態によって動作を決定する. リンクが確立しアドレスを取得後は, MIBsocket を介してIP アドレスをインタフェースに設定する. ネットワークエンティティの状態の競合を防ぐため, DHCP のMIBsocket にIP アドレス設定の許可が与えられていないので直接計算機には反映されない. しかし, インタフェース切替え機構がDHCP クライアントの設定要求メッセージを受け取り, IP アドレスの取得を知る.

インタフェース切替え機構で用いる切替え決定の判断のポリシは以下の様なものがある. 以下

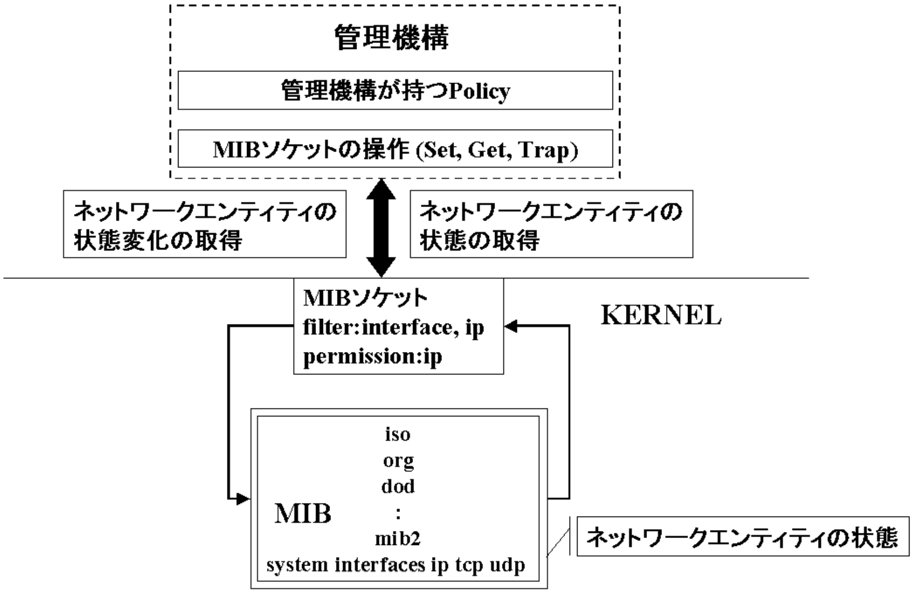


図 6.1: 管理機構の想定図

のポリシを用いて, インタフェース切替え機構は複数のインタフェースから通信に利用する特定のインタフェースを決定する. これらのポリシは移動体計算機の利用者が自分の環境に応じて追加 ･削除できなくてはならない.

- インタフェースの優先度
- リンクの状態
- IP アドレスの有無
- IP アドレスの有効無効
- 経路の有無
- 経路の有効無効

6.3 既存の技術の統合

ネットワークエンティティの管理だけでなく既存 (ネットワークソフトウェア) の技術の統合という点において MIBsocket は利用できる. 既存の様々なネットワークソフトウェアはシステム内のネットワークエンティティを共有している. よって, これらを統合する役目が必要である. MIBsocket をネットワークエンティティとネットワークソフトウェアとの唯一のインタフェースとする. MIBsocket により, 自分以外のネットワークソフトウェアがネットワークエンティティの設定を行いネットワークエンティティの状態を変化させたという情報を得ることができる.

例えば, インタフェース切替え, MobileIP と DHCP の統合を考える. 図 6.3に MIBsocket を用いた場合を示す. この場合移動後, インタフェース切替え機構はインタフェースを再選択する. 移動先での IP アドレスの取得は DHCP が行う. DHCP から得た情報をもとにインタフェース切替

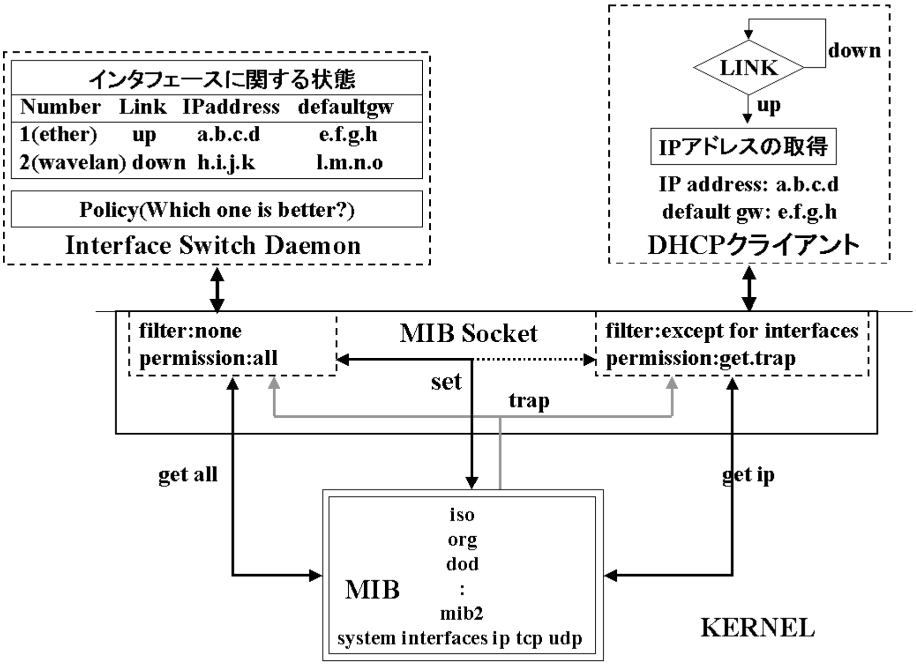


図 6.2: インタフェース切替え

え機構は最適なインタフェースを決定し設定する. インタフェースを切替える等により IP アドレスが変わると MobielIP としては Home Agent(HA) に Care of Address(CoA) の登録をしなおさなくてはならない. インタフェース切替え機構がインタフェースにアドレスを設定した時点でインタフェースに IP アドレスが付いたというトラップを受け取り MobileIP は, HA に新しい CoA の登録を始める. 一方, DHCP はリンクの情報 (主にトラップ) を MIBsocket から得てアドレスを取得あるいは開放を決定する.

MobileIP は, それ自体がネットワークエンティティの一部でもある. よって, MobileIP に対する処理も MIBsocket を介さなくてはならない. そこで MobileIP の MIB を新たに加えることにより, MIBsocket で扱う事が可能になる. 例えば, 移動体計算機環境に応じて利用するしないを決定できる. プライベートアドレスのみで形成されたネットワークに移動した時 MobileIP は利用できない. HA から MN への通信ができないからである. よってこの場合 MobileIP は利用できない. よって, プライベートアドレスネットワーク内で IP アドレスを取得後 MobileIP を MIBsocket を通して MobielIP を OFF に設定する.

6.3.1 MANET の自律ネットワークにおけるネットワークエンティティ管理

本管理機構を構築する事により, 自律ネットワーク等の頻繁にネットワークエンティティの状態が変化する場合にも適応できる. ネットワークエンティティの状態が頻繁に変化する環境下ではネットワークエンティティの状態の検知, 及び管理が必要である. ネットワークエンティティの状態の検知においては, 単純でしかも明確な機構が必要である. なぜなら, 移動体通信の計算機でネットワークエンティティのポーリングを行って状態を監視するなら, ネットワークエンティティの変化を検知するのに時間がかかる. また, 移動体計算機の電力及び CPU を消費してしまう. 移動体計算機にとって電力は無限ではないので電力の節約は重要である. よって, MIBsocket のよう

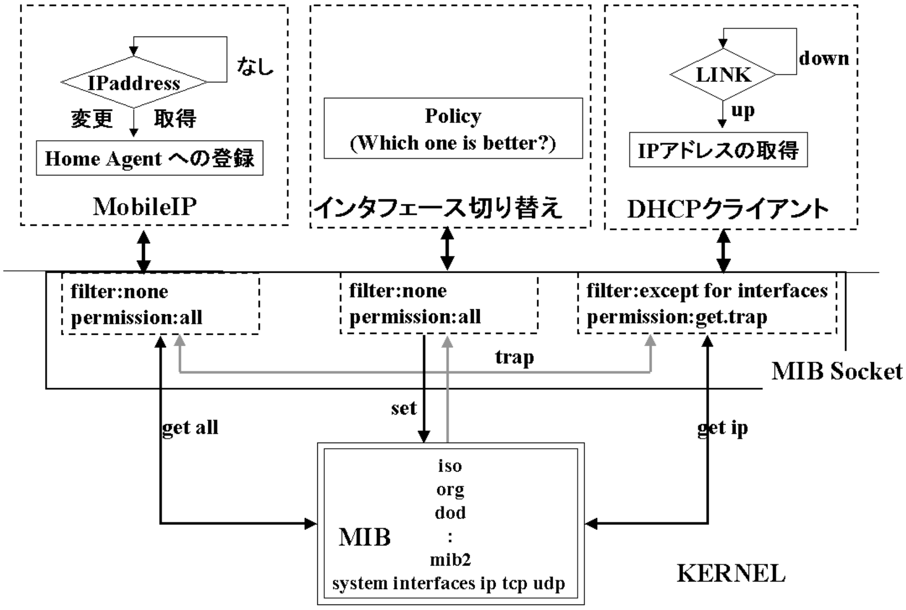


図 6.3: 既存の技術の統合

な, ネットワークエンティティの状態をソケットと MIB で管理する機構は不可欠である.

6.3.2 SNMP の構築

MIBsocket は, MIB を用いた情報の取得あるいは設定を行うことができるため, SNMP で求められる機能を持っている. よって, SNMP の管理対象ノードあるいはネットワーク管理ステーションが, MIBsocket を介して MIB のデータにアクセスする事ができる. SNMP のデータベース部は MIBsocket を用いて管理できるので, 通信に関するプロトコルの部分を実装できれば実現できる. MIBsocket を用いる事により, 容易に SNMP 管理を実現できる.

第 7 章

システムの実装

本章では始めに, MIBsocketの実装について述べ, MIBsocketを扱うシステムコマンド *mibconfig* の実装について述べる. 最後に, MIBsocketを用いた管理機構の例としてインタフェース切替え機構の実装について述べる.

7.1 実装環境

オペレーティングシステムは, FreeBSD2.2.7とPAO-980430を用いた. PAOは, FreeBSD PCカードドライバやAPMドライバの拡張等が含まれている移動型計算機のための総合的なパッケージである. 通信デバイスは, AT&TのWaveLANカード, Megahertz X-Jackカード, 3Com 3C589Dカード, Xircom Airsurferカード等を用いて実装を行った. また, ソフトウエアとしてWIDE[1]のDHCPクライアント(version 1.4.0p1)及びCMUのSNMPライブラリ(version 3.1.3)を改良して用いた.

7.2 MIBsocket

図7.1がMIBsocketの処理の流れである. ソケットを介してユーザ空間とカーネル間で要求・応答を行う.

MIBsocketは大きく分けて3つの部分からなる. ACTIVEモードの処理, PROACTIVEモードの処理, そしてREACTIVEモードの処理の部分の3つである. これらはそれぞれ第5章の, ”set mode”, ”trap mode” そして”get mode” を意味する. ユーザからMIBsocketへのコマンドは, *mib_output()* が受け取り許可のフラグをチェックする. 許可が与えられているコマンドであれば, コマンドに応じた処理を行う.

ACTIVEモードの場合(つまり設定)は, *mib_set()* がまずオブジェクト識別子を調べ, どのオブジェクトに対する設定要求かを調べる. ACTIVEモードで設定できるオブジェクトは, IPアドレスに関するオブジェクト及び経路情報に関するオブジェクトである. 設定が成功すれば, ネットワークエンティティの状態が変わったというトラップが発生し, 失敗した場合は要求を行ったMIBsocketに対してエラー番号が返る. REACTIVEモードの場合(オブジェクトのインスタンスの取得)は, *mib_get()* 及び *mib_get_instance()* で処理される. *mib_get()* が要求のオブジェクト識別子を調べ,

---

[1] WIDE(Widely Integrated Distributed Environments)は, 新しいコンピュータ環境の確立を目指す研究プロジェクトである.

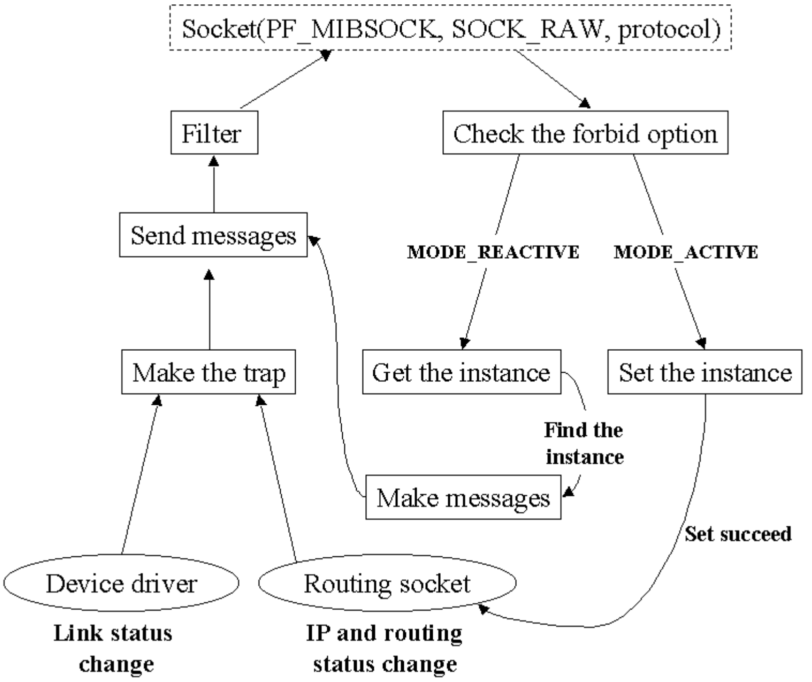


図 7.1: MIBsocket の処理

mib_get_instance() がオブジェクト識別子のインスタンスを調べる. その後, *mib_buildmsg()* が要求に対する応答メッセージを作り MIBsocket に送信する. PROACTIVE モードの場合 (トラップ) は, *mib_trap()* が処理を行う. *mib_trap()* はトラップメッセージを作り, MIBsocket に対して送信する.

mib_msg() は, *mib_raw_input()* を呼び MIBsocket へ実際に送信を行っている. *mib_raw_input()* では, フィルタリングが設定されている場合その処理を行う. フィルタリングされているメッセージは, *mib_raw_input()* が破棄する.

7.3 MIBsocket で新たに追加及び変更した構造体

MIBsocket の実装において, 既存の FreeBSD のカーネル内の構造体を変更あるいは追加した. それら構造体をどのように, そして何のために変更したかについて述べる.

7.3.1 mib_msghdr 構造体

図 7.2が, MIBsocket で扱うメッセージである. oid メンバは, MIB を参照するためのオブジェクト識別子が入る. oidlen メンバは, オブジェクト識別子の長さを表す. qflag は, このメッセージが複数に跨る場合用いられる. もしこのフラグが立っているとメッセージが続く事を示す. xid メンバは, メッセージを特定するための特定の数が入る. type メンバは, トラップ及び設定のタイプを示す. これは, オブジェクト識別子とモードだけで特定できないメッセージを特定するためである.

```
struct mib_msghdr {
  u_long oid[MAXOIDLEN];
  short  oidlen;
  short  mode;
  short  type;
  char   qflag;
  long   xid;
  char   data[MAXDATASIZE];
};
```

図 7.2: mib_msghdr 構造体

7.3.2 rawcb 構造体

図 7.3は, raw protocol control block(rawcb) である. rawcb は raw ソケットの状態を保持する構造体であり, ダブルリンクドリストで保持されている.

```
struct rawcb {
  struct rawcb *rcb_next; /* double linked list */
  struct rawcb *rcb_prev;
   /* 途中省略 */
  int rcb_mib_filter;
  int rcb_mib_mode;
};
```

図 7.3: rawcb 構造体

MIBsocket のために追加したメンバは, rcb_mib_filter と rcb_mib_mode である. rcb_mib_filter は, メッセージのフィルタリングを行うためのものである. *ioctl()* により MIBsocket に対してフィルタリングを設定すると, rcb_mib_filter にその値が設定される. メッセージを送信する時に, この値を参照する. rcb_mib_mode はモードに対する許可である. モードに対する許可の情報を持つ. これも同様に, 利用者が ioctl により設定する事ができる.

7.4 MIBsocket のコントロール

MIBsocket のフィルタリングや許可に関連する部分の実装について説明する. 表 7.1が, MIBsocket で用いる *ioctl* である.

ioctl によって呼ばれる, *mib_ioctl()* が図 7.3にある rcb_mib_filter と rcb_mib_mode にそれぞれ値をセットする.

表 7.1: MIBsocket で用いる *ioctl*

Command	Description	引数
SIOCMIBFILTER	フィルタに関する設定	MIB_FILTER_{IF, IP, TCP, UDP, OTHER}
SIOCMIBMODE	許可に関する設定	MIB_FILTER_{ACTIVE, PROACTIVE, REACTIVE}

7.5 MIBsocket のトラップ

ネットワークエンティティの状態変化は, 表 7.2の方法で検知する.

表 7.2: ネットワークエンティティの状態変化の検知

トラップ	トラップの対象	検知方法
リンク	リンクの状態	インタフェースのデバイスドライバ
IP アドレス	IP アドレスの設定や削除	Routing Socket (*rt_newaddrmsg()*)
経路	経路の追加や削除	Routing Socket (*route_output()*)

リンクの状態は, デバイスドライバが検知する. デバイスドライバは, トラップメッセージをMIBsocket に対して送信し状態変化を通知する. デバイスドライバは, インタフェース毎に実装依存であるが, リンクの状態をドライバの変数の中に持つ物もある. 無線デバイスの場合は, 電波強度を基準にリンクの状態を決定する. デバイスドライバにより通信デバイスからの割り込みによりリンクの変化を検知するものとリンクの状態をポーリングする事により検知するものがある. ポーリングは *mib_slowtimo()* を用いる事により可能である. *mib_slowtimo()* は watchdog タイマをスタートさせタイマが切れるとカーネルは watchdog 関数を呼ぶ. watchdog 関数は *mib_add_slowtimo()* により登録できる. タイマは 500 ミリ秒に設定されている.

IP アドレスについては, インタフェースに IP アドレスを設定したあるいは削除したというトラップが送信される. 既存のルーティングソケットにある *rt_newaddrmsg()* が IP アドレスの変化を検知し, *mib_trap()* を呼ぶ事により送信される. 経路に関するトラップも同様に, ルーティングソケットが検知し送信される.

トラップを送るには, *mib_trap()* 関数を呼ぶか, MIBsocket のメッセージを用意し *mib_msg()* 関数を呼ぶ事により送信される.

7.6 MIBsocket によるネットワークエンティティの情報の取得

MIBsocket に, REACTIVE モードのメッセージが来た時の処理は図 7.4である. 図 7.1も参照して頂きたい. *mib_output()* が MIBsocket からメッセージを受け取ると許可とメッセージのチェックを行う. その後, *mib_get()* で指定されたオブジェクト識別子のチェックを行う.

mib_get_instance() は, オブジェクト識別子に応じたインスタンスを取得し送信する関数を呼ぶ. *mib_get_instance()* で呼ばれた関数は, インスタンス取得をして *mib_buildmsg()* で応答のメッセー

ジを作り送信する．もし, オブジェクト識別子に対して複数のインスタンスが存在する場合は存在するインスタンス全てを送り返す．この時, 一つのメッセージには一つのインスタンスしか送れないので, インスタンスの数だけのメッセージを送る．このとき, MIBsocket のメッセージの qflag メンバが, 最後のメッセージのみ 0 に設定される．一つのインスタンスしか無い時は常に 0 である．つまり qflag が 0 以外の場合は, その後にメッセージが続く事を示す.

```
mib_output(m, so){
        送られて来たメッセージのチェック
        if(mode == REACTIVE) err = mib_get();
        return(err); /* 失敗したら-1 が返る */
}
mib_get_instance(){
        オブジェクト識別子のインスタンスを探す
        switch(object ID){
        case interfaces:
                get_if_instance();
        case ip:
                get_ip_instance();
             /* 以下同様 */
        }
}
mib_buildmsg(){
        メッセージを作る
        m_gethdr(M_DONTWAIT, MT_DATA); /* mbuf を取得する */
        mbuf にメッセージをコピーする
        mib_msg();
}
```

図 7.4: ネットワークエンティティの情報 (MIB) の取得

表 7.3は, ネットワークエンティティの情報がカーネル内で主にどこに保持されているかを示したものである．例えば, ifnet 構造体にはインタフェースに関する情報が含まれている．ifnet 構造体はインタフェースの数だけあり, それらはリンクドリストで連結されている．MIBsocket ではこれらの情報と MIB のオブジェクト識別子とを対応づけして利用している．インタフェースの名前のオブジェクト識別子である 1.3.6.2.1.2.2.2.1.1 では, ifnet 構造体の if_name の情報をインスタンスとして用いる.

7.7 MIBsocket によるネットワークエンティティの情報の設定

現在 MIBsocket から設定できるのは IP アドレスに関する設定と経路情報の設定である．まず, *mib_output()* が取得の場合と同様にメッセージをチェックしモードが SET の場合は, *mib_set()* を

表 7.3: カーネル内のネットワークエンティティの状態を持つ構造体

情報の種類	カーネル内での保持場所
interfaces	ifnet 構造体, if_data 構造体
ip	ipstat 構造体, in_ifaddr 構造体, llinfo_arp 構造体, radix_node 構造体
tcp	tcpstat 構造体, tcpcb 構造体, inpcb 構造体
udp	udpstat 構造体, inpcb 構造体

呼ぶ.

図 7.5が設定の処理で使う主な関数である.

```
mib_output(m, so){
        送られて来たメッセージのチェック
        if(mode == ACTIVE) err = mib_set();
        return(err); /* 失敗したら-1 が返る */
}
mib_set(){
        switch(object ID){
        case ipAddrTable:
                設定に必要な処理を行う
                in_control(); /* 実際に IP アドレスを設定する関数 */
        case ipForwardTable:
ルーティングソケットのメッセージを作る
route_output();
        }
}
```

図 7.5: ネットワークエンティティの情報 (MIB) の設定

mib_output() で送られて来たメッセージをチェックしその MIBsocket が設定の許可を与えられているかを調べる. そして, モードが ACTIVE であったら *mib_set()* を呼ぶ. *mib_set()* は, オブジェクト識別子と設定するためのインスタンスをチェックする. 問題が無ければ, 設定に必要な処理を行い実際の設定を行う関数に渡す. 以下がそれぞれの設定に関する詳しい説明である.

7.7.1 IP アドレスに関する設定

IP アドレスかネットマスクあるいはブロードキャストアドレスのどれに設定すればいいかをオブジェクト識別子から知る. 例えばIP アドレスの場合, 指定されたオブジェクト識別子のサフィックスを用いて設定したいインタフェースの ifnet 構造体を得る. もし, ここで対応する ifnet 構造体

が無ければエラーナンバが返る．その後，実際に設定を行う関数である *in_control()* の引数[2]を作る．最後に，*in_control()* を呼んで実際の設定を行う．

7.7.2 経路情報に関する設定

経路の追加削除は，既存のルーティングソケットを用いる．MIBsocket は経路に関する要求メッセージを受け取るとメッセージのチェック及び MIBsocket の設定許可を調べる．問題が無ければ，ルーティングソケットが処理できるメッセージを MIBsocket の要求メッセージを元に作りルーティングソケットの *route_output()* に渡す．*route_output()* の返り値は MIBsocket の値として返す．

7.8 *mibconfig* 及び利用したライブラリ

mibconfig 及び MIBsocket で用いるライブラリについて述べる．ライブラリは既存のライブラリに機能を追加した．

7.8.1 *mibconfig* コマンド

mibconfig は MIBsocket を用いてシステムの情報やトラップのメッセージを受けるシステムコマンドである．*mibconfig* では，インタフェース，IP，TCP と UDP の情報を取得及び設定，あるいはトラップメッセージの受信ができる．引数で指定されたオブジェクト名をオブジェクト ID に変換し MIBsocket へ要求メッセージを送る．MIBsocket から得た応答メッセージを出力する．

7.8.2 SNMP ライブラリ

MIBsocket では，MIB のデータ構造を用いるためオブジェクト識別子と名前のマッピングが必要になる．今回，CMU の SNMP ライブラリを改良して用いた．この SNMP ライブラリは，CMU[3]が UNIX 環境で SNMP を実現するために開発したフリーソフトウェアの一部である．現在では，UCDavis[4]で維持されている．MIBsocket で使う SNMP ライブラリ (libsnmp.a) は，MIB の操作に関するものである．表 7.4が MIBsocket のために追加した関数と利用する関数の一覧である．

表 7.4: SNMP ライブラリに追加，変更した関数

関数名	引数	処理内容
get_node()	char *objname,u_long oid,int len	オブジェクト名から識別子へのマッピング
get_nodetype()	char *objname,int *type	オブジェクト名からタイプへのマッピング
sprint_objid()	char *objname,int oid,int len	オブジェクトの表示

---

[2] 引数は，socket 構造体，cmd，ifreq 構造体,ifnet 構造体

[3] Carnegie Mellon University

[4] the University of California at Davis

7.9 インターフェース切替え機構: MIBsocket による管理機構の構築例

MIBsocket による管理機構の構築の例としてインタフェース切替えを実装した. 以下にインタフェース切替え機構の実装について説明する.

インタフェース切替え機構は以下の 3 つの機能からなる.

- システム内のインタフェースの情報データベース
- インタフェースの選択の決定
- インタフェースの設定

システム内のインタフェースに関するデータベースは, MIBsocket からの情報を用いる. 起動時にデータベースを構築し, その後トラップなどの情報を元に随時更新される. インタフェース選択の決定機能は, インタフェース切替え機構の持つデータベースの情報を元にどのインタフェースを選択するのが適切かを判断し決定する. 判断基準としては, リンクの状態及び IP アドレスの有無が主に用いられる. インタフェースの決定の後, インタフェースの設定機能により通信が行えるように設定を完了する. インタフェースの切替えは, IP アドレス及び経路の設定を更新する事により実現する. 図 7.6にインタフェース切替え機構のフローチャートを示す.

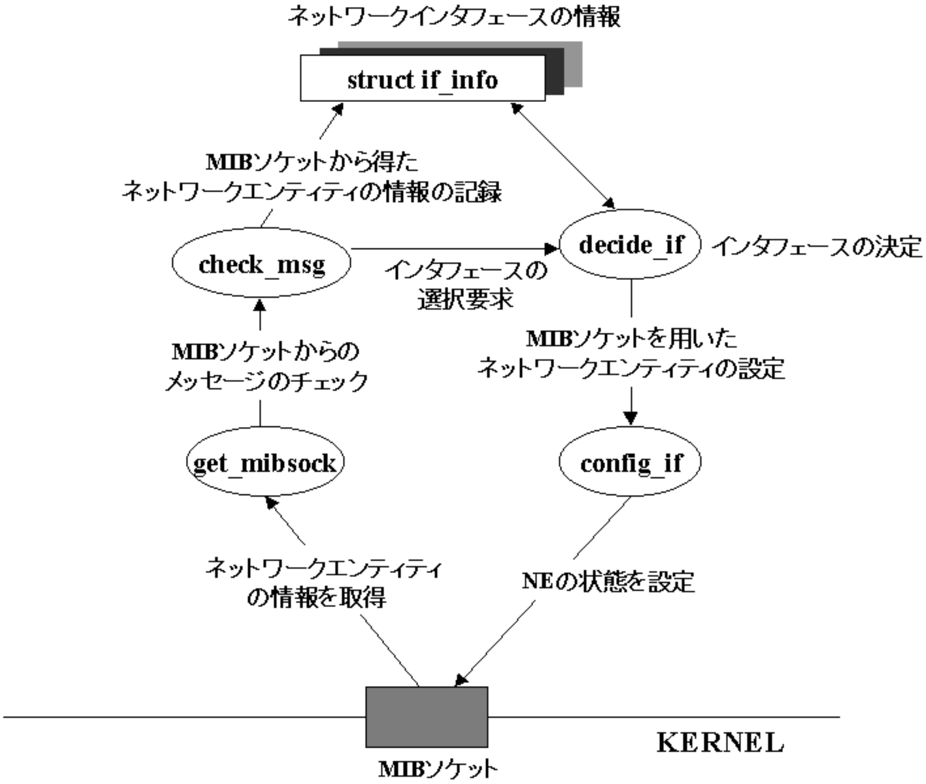


図 7.6: インタフェース切替え機構フローチャート

7.9.1 インタフェースの情報データベース

インタフェースの情報データベースはインタフェース切替え機構の設定ファイルに指定のあるインタフェースの情報を格納する. リンクに関する情報, IP アドレスに関する情報, 経路に関する情報を持つ. 図 7.7の if_status 構造体で情報を保持する. link_status, ip_status, gw_status とは MIBsocket から得られるトラップ情報をもとに保持される. それぞれリンクの有無, IP アドレスの有無及び有効無効, 経路の有無及び有効無効という状態がある. in_addr 構造体には実際の IP アドレス, ネットマスクアドレス, ブロードキャストアドレスの値が入る.

```
struct if_status {
  int ifindex;
  int link_status;
  int ip_status;
  int gw_status;
  struct in_addr dfgw;
  struct in_addr ipaddress;
  struct in_addr netmask;
  struct in_addr broadcast;
}
```

図 7.7: if_status 構造体

インタフェース切替え機構は if_status 構造体以外に if_info 構造体を持つ. if_info 構造体には, 利用者のインタフェース切替えの判断となる優先度や IP アドレスの取得法等の情報がある. 図 7.8 が実際の構造体である. if_info 構造体はインタフェース切替え機構の起動時に設定ファイルから読み込まれる.

priority は優先度が入る. これは複数のインタフェースが同時に有効になった時この優先度によってインタフェースの選択が行われる. ipassign は IP アドレスの取得法を設定する. DHCP[5] と Static のどちらかが選べる. DHCP の場合は, インタフェース切替え機構と一緒に起動される. DHCP がアドレスを取得し MIB ソケットを用いて設定を行っても, MIBsocket の ACTIVE モードへの許可が無いため設定はシステムに反映されない. しかし, インタフェース切替え機構は DHCP が設定を行った事が分かるので (フィルタをかけない限り REACTIVE 以外のメッセージは全ての MIBsocket に情報が送られる), そのメッセージからアドレスなどの情報を if_status 構造体に保持し必要時に設定する. DHCP から取得したアドレスは一定時間を過ぎると破棄される. Static の場合, in_addr 構造体に指定してある値を用いる.

7.9.2 インタフェースの選択の決定

インタフェースの選択はネットワークエンティティの状態変化によって行われる. 状態変化は, MIBsocket の PROACTIVE モードを用いて監視する. インタフェースの選択は以下の条件で起きる.

1. インタフェースのリンクが変わる

---

[5] MIBsocket を用いた DHCP クライアントで, WIDE DHCP を改良した

```
struct if_info {
  char ifname[NAMESIZE];
  int priority;
  int ipassign;
  struct in_addr dfgw;
  struct in_addr ipaddress;
  struct in_addr netmask;
  struct in_addr broadcast;
}
```

図 7.8: if_info 構造体

2. IP アドレスの有無
3. 経路情報の変更 (主にデフォルト経路)

これらの状況は MIBsocket のトラップメッセージを監視する事により知る事ができる. もし, インタフェースを切替える判断を下せばインタフェースを切替え必要が無ければ何も行わない. また, これらトラップのメッセージは随時 if_status 構造体に記録される.

7.9.3 インタフェースの設定

インタフェースの設定は, MIBsocket の ACTIVE モードを用いる. 設定が成功すると PROACTIVE モードのメッセージ (トラップ) が MIBsocket から送られるので, インタフェース切替え機構は切替えが成功したかどうかを知る事ができる.

第 8 章

評価

本章では, MIBsocket の処理にかかわる評価及び MIBsocket を用いたインタフェース切替えを用いて評価を行う.

8.1 MIBsocket のネットワークエンティティの取得機構の評価

MIBsocket を用いて取得できるネットワークエンティティの状態を示す. ネットワークエンティティの状態は, MIB の書式になっていることが分かる. 今回実装した, TCP 層, UDP 層, IP 層そしてインタフェース層の情報を *mibconfig* を用いて取得したものである.

8.1.1 TCP に関するネットワークエンティティの取得

TCP に関する情報は, 図 8.1の情報が取得できる. 図 8.1では, 2 つのコネクションしか示していない. 実際は計算機内のコネクションの数だけ取得できる. コネクションの情報は,tcpcb 構造体からの情報である. コネクションの情報を取得できることにより, 現在計算機が誰と TCP のコネクションを張っているかを知ることができる. TCP に関するネットワークエンティティの情報を, MIBsocket を用いて取得することができた.

8.1.2 UDP に関するネットワークエンティティの取得

UDP に関する情報は, 図 8.2の情報が取得できる. UDP で用いる inpcb 構造体の情報は全て取得できる. UDP に関するネットワークエンティティの情報を MIBsocket を用いて取得することができた.

8.1.3 IP に関するネットワークエンティティの取得

IP に関する情報は, 図 8.3の情報が取得できる. IP 層全体の情報はデータのフォワードについてや IP データグラムの受信数やエラー数等の情報が取得できる. また, ip.ipNetToMediaTable にあるようにインタフェースの MAC アドレスの情報も取得できる. これは, ARP テーブルの情報を用いている. ip.ipAddrTable ではインタフェース毎の IP アドレスの情報を扱う. これは, インタフェースに設定されている IP アドレス全てを取得できる. ip.ipForward では, 経路に関する状態を取得できる. 経路制御表にあるエントリは全て取得可能である.

IP 及び経路に関するネットワークエンティティを取得することができた.

```
tcp.tcpRtoAlgorithm = 4
tcp.tcpRtoMin = 1000
tcp.tcpRtoMax = 0
tcp.tcpMaxConn = -1
tcp.tcpActiveOpens = 0
tcp.tcpPassiveOpens = 5
tcp.tcpAttemptFails = 0
tcp.tcpEstabResets = 0
tcp.tcpCurrEstab = 0
tcp.tcpInSegs = 947
tcp.tcpOutSegs = 172
tcp.tcpRetransSegs = 0
/*      TCP のコネクションテーブルの情報       */
tcp.tcpConnTable.tcpConnEntry.tcpConnLocalAddress = 0.0.0.0
tcp.tcpConnTable.tcpConnEntry.tcpConnRemAddress = 0.0.0.0
tcp.tcpConnTable.tcpConnEntry.tcpConnState = 1
tcp.tcpConnTable.tcpConnEntry.tcpConnLocalPort = 31767
tcp.tcpConnTable.tcpConnEntry.tcpConnRemPort = 0
tcp.tcpConnTable.tcpConnEntry.tcpConnLocalAddress = 133.27.221.83
tcp.tcpConnTable.tcpConnEntry.tcpConnRemAddress = 203.178.141.163
tcp.tcpConnTable.tcpConnEntry.tcpConnState = 4
tcp.tcpConnTable.tcpConnEntry.tcpConnLocalPort = 0
tcp.tcpConnTable.tcpConnEntry.tcpConnRemPort = 129526
```

図 8.1: TCP 層のネットワークエンティティの状態

```
%UDP で用いる inpcb エントリは全て取得可能
udp.udpInDatagrams = 37
udp.udpNoPorts = 36
udp.udpInErrors = 0
udp.udpOutDatagrams = 1
udp.udpTable.udpEntry.udpLocalAddress = 0.0.0.0
udp.udpTable.udpEntry.udpLocalPort = 0
udp.udpTable.udpEntry.udpLocalAddress = 0.0.0.0
udp.udpTable.udpEntry.udpLocalPort = 1538
```

図 8.2: UDP 層のネットワークエンティティの状態

```
/*      一般的な IP に関する情報      */
ip.ipForwarding = 0
ip.ipDefaultTTL = 64
ip.ipInReceives = 148
ip.ipInHdrErrors = 0
ip.ipInAddrErrors = 84
ip.ipForwDatagrams = 0
ip.ipInUnknownProtos = 40
ip.ipInDelivers = 148
ip.ipOutRequests = 1
ip.ipOutDiscards = 0
ip.ipOutNoRoutes = 84
ip.ipReasmTimeout = 60
ip.ipReasmReqds = 0
ip.ipReasmOKs = 0
ip.ipReasmFails = 0
ip.ipFragOKs = 0
ip.ipFragFails = 0
ip.ipFragCreates = 0
/*      以下は，デフォルト経路ホストの MAC アドレス      */
ip.ipNetToMediaTable.ipNetToMediaEntry.ipNetToMediaIfIndex = 6
ip.ipNetToMediaTable.ipNetToMediaEntry.ipNetToMediaPhysAddress =
00:60:09:ed:20:62
/*      インタフェースに設定されている IP アドレスの情報      */
ip.ipAddrTable.ipAddrEntry.ipAdEntIfIndex = 6
ip.ipAddrTable.ipAddrEntry.ipAdEntBcastAddr = 133.27.221.255
ip.ipAddrTable.ipAddrEntry.ipAdEntAddr = 133.27.221.83
ip.ipAddrTable.ipAddrEntry.ipAdEntNetMask = 255.255.255.0
/* 経路情報は，経路制御表にあるエントリ全てを取得することが可能．以下
は，デフォルト経路の情報． */
ip.ipForward.ipForwardTable.ipForwardEntry.ipForwardDest = 0.0.0.0
ip.ipForward.ipForwardTable.ipForwardEntry.ipForwardMask = 0.0.0.0
ip.ipForward.ipForwardTable.ipForwardEntry.ipForwardNextHop = 133.27.221.3
ip.ipForward.ipForwardTable.ipForwardEntry.ipForwardIfIndex = 6
ip.ipForward.ipForwardTable.ipForwardEntry.ipForwardType = 67587
ip.ipForward.ipForwardTable.ipForwardEntry.ipForwardMetric1 = -1
ip.ipForward.ipForwardTable.ipForwardEntry.ipForwardMetric2 = -1
ip.ipForward.ipForwardTable.ipForwardEntry.ipForwardMetric3 = -1
ip.ipForward.ipForwardTable.ipForwardEntry.ipForwardMetric3 = -1
ip.ipForward.ipForwardTable.ipForwardEntry.ipForwardMetric3 = -1
```

図 8.3: IP 層のネットワークエンティティの状態

8.1.4 インタフェースに関するネットワークエンティティの取得

インタフェース層に関する情報は, 図8.4の情報が取得できる. 計算機内のネットワークインタフェースの数とインタフェース毎の情報を扱う. インタフェース毎の情報は計算機内の全てのインタフェースに関して取得できる. 実際の通信デバイス以外にもループバックデバイスのような仮想デバイスも扱える. 図8.4には, ループバックとイーサネットカードのMegahertz X-Jackカードの情報がある.

ネットワークインタフェース及び仮想インタフェースに関するネットワークエンティティの情報を図8.4の通り取得できた.

8.2 MIBsocketにおけるネットワークエンティティ状態の取得機構の評価

測定環境は, 移動対計算機のオペレーティングシステムがFreeBSD-2.2.7 + PAO-0913である. CPUはpentium200MHzでメインメモリは96MBである.

MIBsocketを用いてカーネル内の情報を取得するのにかかる時間を測定した. 測定は, オブジェクトを見つけて実体を取得する時間, MIBsocketに流すためのメッセージの作成, 及びMIBsocketに送信するのにかかる時間である. 送信にかかる時間にはフィルタリングする時間も含まれる. インタフェースのインデックス及び計算機内のIPアドレスの情報を取得した. 測定した計算機では, インタフェースのインデックスは5つのメッセージが戻り, IPアドレスの情報は2つのメッセージが戻る. 表8.1が測定結果である. MIBsocketは, オブジェクト識別子からオブジェクトの実体へは線形探索で行っている. そのためのオーバヘッドがある.

MIBsocketの取得機構を用いて, ネットワークソフトウェアはネットワークエンティティの情報を取得できる. また, MIBsocketのオーバヘッドも無視できる範囲であるため, ネットワークソフトウェアは十分MIBsocketを介してネットワークエンティティを取得できる.

表8.1: REACTIVEモードにおけるMIBsocketのオーバヘッド

	Times (μ sec)	
Operation	ifIndex	ipAdEntAddr
オブジェクトの取得	104	109
メッセージの作成	733	296
メッセージの送信	778	346
total	1615	751

トラップに関するMIBsocketのコストはインタフェース切替えを用いた実験で紹介する.

8.3 インターフェース切替えを用いた評価

TCPとICMPを用いた通信中にインタフェースを切替えて評価した. インタフェースは, リンクの状態変化を受けて動的に切替える.

```
interfaces.ifNumber = 6
/*      ループバックインタフェースの情報     */
interfaces.ifTable.ifEntry.ifIndex = 4
interfaces.ifTable.ifEntry.ifDescr = lo0
interfaces.ifTable.ifEntry.ifType = 24
interfaces.ifTable.ifEntry.ifMtu = 0
interfaces.ifTable.ifEntry.ifSpeed = 0
interfaces.ifTable.ifEntry.ifPhysAddress =
interfaces.ifTable.ifEntry.ifAdminStatus = up
interfaces.ifTable.ifEntry.ifOperStatus = up
interfaces.ifTable.ifEntry.ifInOctets = 40
interfaces.ifTable.ifEntry.ifInUcastPkts = 1
interfaces.ifTable.ifEntry.ifInNUcastPkts = 0
interfaces.ifTable.ifEntry.ifInDiscards = 0
interfaces.ifTable.ifEntry.ifInErrors = 0
interfaces.ifTable.ifEntry.ifOutOctets = 40
interfaces.ifTable.ifEntry.ifOutUcastPkts = 1
interfaces.ifTable.ifEntry.ifOutNUcastPkts = 0
interfaces.ifTable.ifEntry.ifOutDiscards = 0
interfaces.ifTable.ifEntry.ifOutErrors = 0
interfaces.ifTable.ifEntry.ifOutQLen = 0
/*      Megahertz X-Jack カードの情報      */
interfaces.ifTable.ifEntry.ifIndex = 6
interfaces.ifTable.ifEntry.ifDescr = sn0
interfaces.ifTable.ifEntry.ifType = 6
interfaces.ifTable.ifEntry.ifMtu = 1500
interfaces.ifTable.ifEntry.ifSpeed = 10000000
interfaces.ifTable.ifEntry.ifPhysAddress =   00:00:86:05:68:28
interfaces.ifTable.ifEntry.ifAdminStatus = up
interfaces.ifTable.ifEntry.ifOperStatus = up
interfaces.ifTable.ifEntry.ifInOctets = 17537
interfaces.ifTable.ifEntry.ifInUcastPkts = 200
interfaces.ifTable.ifEntry.ifInNUcastPkts = 199
interfaces.ifTable.ifEntry.ifInDiscards = 0
interfaces.ifTable.ifEntry.ifInErrors = 0
interfaces.ifTable.ifEntry.ifOutOctets = 758
interfaces.ifTable.ifEntry.ifOutUcastPkts = 5
interfaces.ifTable.ifEntry.ifOutNUcastPkts = 0
interfaces.ifTable.ifEntry.ifOutDiscards = 0
interfaces.ifTable.ifEntry.ifOutErrors = 0
interfaces.ifTable.ifEntry.ifOutQLen = 0
```

図 8.4: インタフェース層のネットワークエンティティの状態

8.3.1 評価環境

図 8.5が, インタフェース切替えの測定におけるネットワークのトポロジである. AT&T の WaveLAN カードと Megahertz の X-Jack カードを用いて行う. MIBsocket 及びインタフェース切替えが動く MH は, OS が FreeBSD-2.2.7 のラップトップである. pentium200MHz でメインメモリは 96MB である. TCP を用いた場合のみ, MobileIP を用いた. Home Agent はルータの機能もする. Home Agent は pentiumII の 450MHz でメモリが 128MB である.

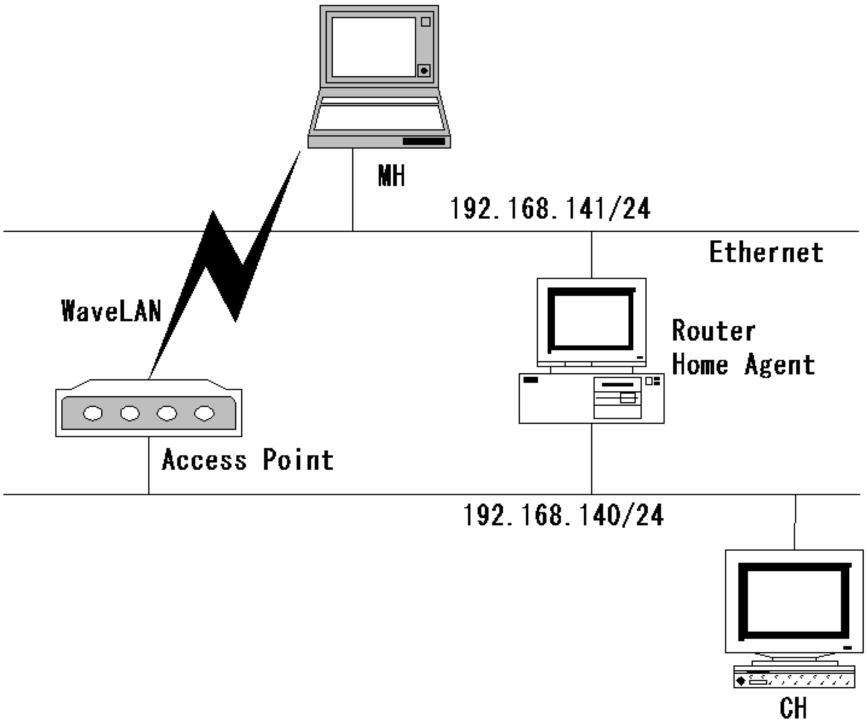


図 8.5: 実験のネットワークトポロジ

8.3.2 TCP で通信上でのインタフェース切替え

図 8.6は, TCP のセッションを張りながらインタフェースを切替えた場合のパケットの到着時間とシーケンス番号である. 測定は *ttcp* を用いてデータを送り, *tcpdump* でログを取った.*tcpdump* は受信側のホストで行った. 送信ホストはインタフェース切替えが行うホストである. インタフェースは 2 度切替えており, 始めイーサネットから無線 LAN へと切替え, 次に無線 LAN からイーサネットへと切替えている.

TCP のセッションは, アドレスの変更により切断されてしまう. なぜなら, TCP のプロトコルコントロールブロック内で IP アドレスを相手計算機の情報として使っているためである. よって, インタフェースを切替えることで TCP のセッションは切断されてしまう. この切断を防ぐために, MobileIP を用いた. MobileIP を用いることにより, 計算機は home address でセッションを張る. home address は MobileIP が動いている限り変更が無いため切断を防ぐことができる. TCP には輻輳制御とキープアライブタイマを持っているため, インタフェースが切替えることにより起き

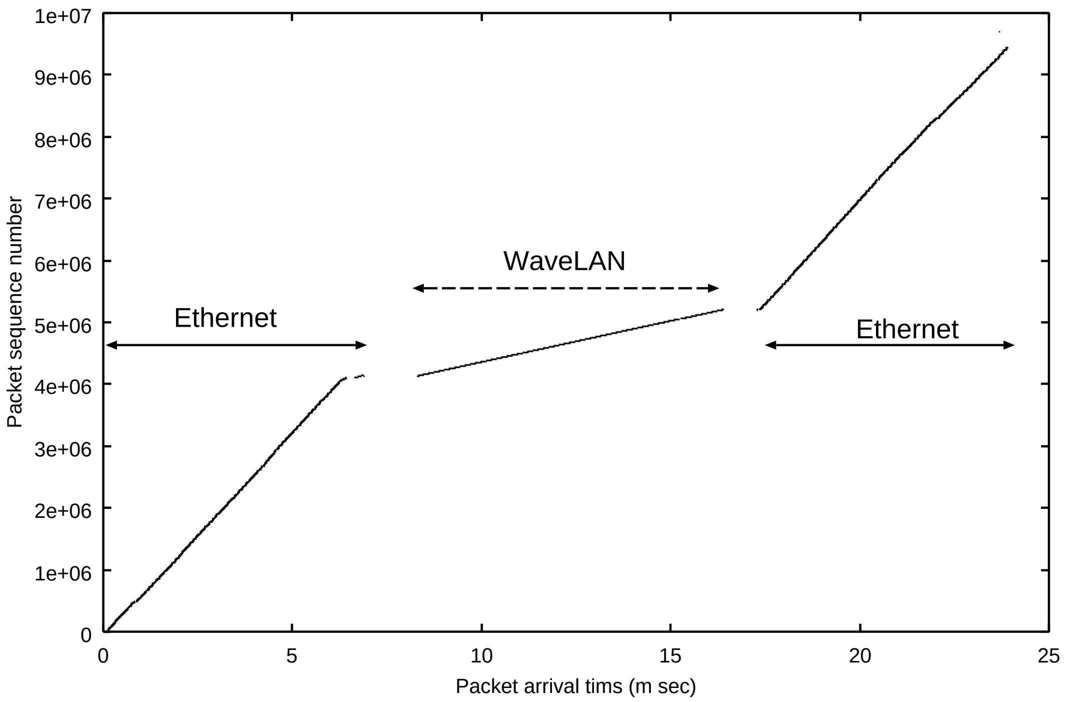


図 8.6: TCP で通信中でのインタフェース切替え

るパケットのロスを検知しスロウタイマが起動され, 再送を試みる. イーサネットから無線 LAN への切替えの方が時間がかかっているのは, 帯域が 10M から 2M へと急激に狭まったため TCP が輻輳が起きたと判断したためである. しかし, TCP のセッションがタイムアウトする前にインタフェース切替え機構はリンクの変化を検知しインタフェースを再選択し切替えることができた. また, インタフェース切替え機構は IP アドレスの情報やデフォルト経路の情報を DHCP から得ることもできている. これは, DHCP クライアントを MIBsocket を用いて動作するものに変更した. DHCP クライアントの MIBsocket には ACTIVE モードの許可が無いため, 設定を行っても実際にシステムには反映されない. しかし, インタフェース切替え機構が DHCP クライアントから MIBsocket への ACTIVE モードのメッセージを受け取り, そのインタフェースが取得した IP アドレスや経路の情報を知ることができるのである. つまり, インタフェース切替え機構は DHCP クライアントと協調して動作することができている.

MIBsocket を用いたインタフェース切替え機構は, MIBsocket を介して DHCP と協調動作することができた. また, MIBsocket を用いることによりネットワークエンティティの状態変化をトラップで知ることができるため, シームレスなインタフェースの切替えが行えた. MIBsocket を用いて, ネットワークエンティティの管理を行ったため TCP を用いた通信でもコネクションに影響を与えること無くインタフェースを切替えられた.

8.3.3 ICMP のエコープロトコル上でのインタフェース切替え

図 8.7は, ICMP を用いた場合のエコーリプライの到着時間とシーケンス番号である. また, MIBsocket がリンクの状態を検知しインタフェースを切替えた時間も示してある. 図 8.7内の表は, トラップメッセージを送るたのにかかる MIBsocket のオーバヘッドである. ICMP のエコーリク

エスト・リプライを用いた. ICMP を用いた理由は TCP の測定では MobileIP と TCP の処理自体のオーバヘッドがあるためインタフェース切替えによるオーバヘッドが分からないためである. ICMP では, プロトコル処理にかかるオーバヘッドが少ないため採用した. 測定には pentium カウンタ[1] を用いた.

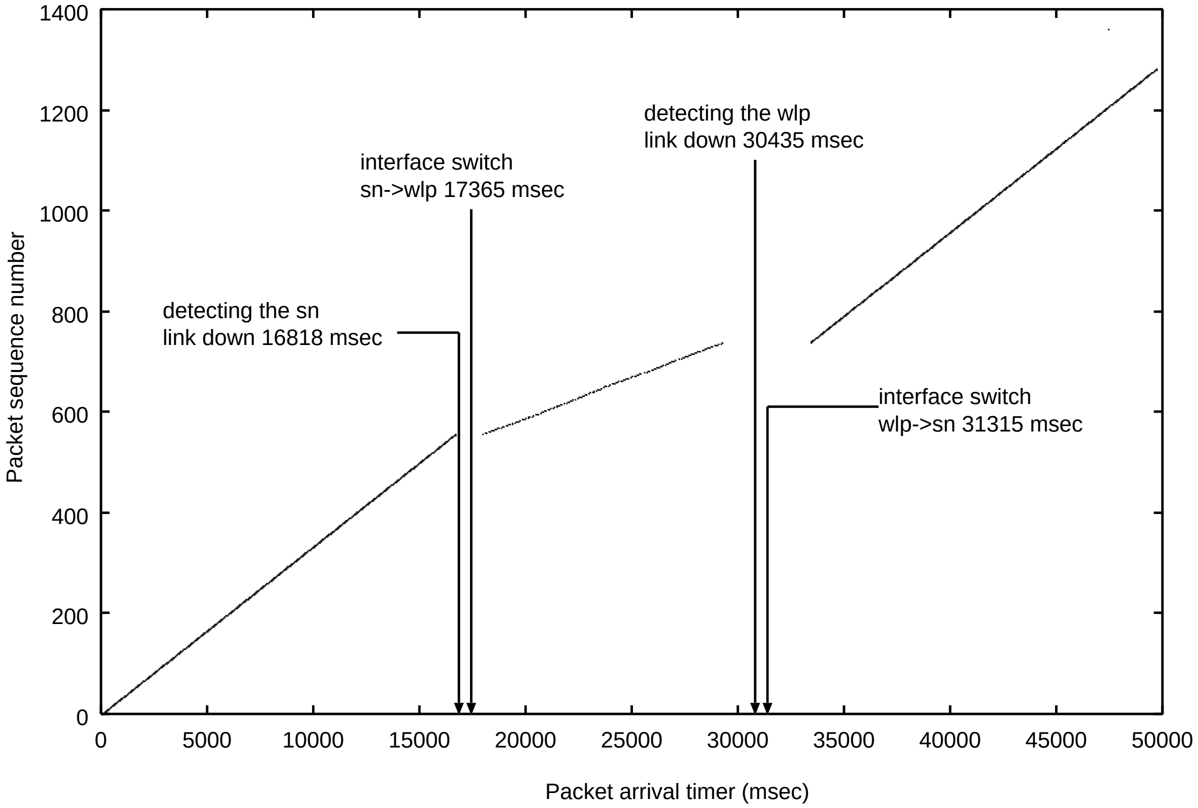


図 8.7: ICMP のエコープロトコル上でのインタフェース切替え

表 8.2: MIBsocket のトラップ機能の性能

MIBsocket の Operation	Time (usec)
リンクの変化の検知	ドライバに依存 (sn: 9)
mbuf の取得	96
メッセージの作成	137
メッセージの送信	182

リンクの変化を検知する時間はそれぞれのネットワークインタフェースのデバイスドライバに依存しているため一概には評価できない. 表 8.2が MIBsocket のトラップ機能の性能評価である. リンクの状態を検知できるイーサネットの場合, インタラプトがかかるのため約 10usec で検知で

---

[1] pentium カウンタは CPU の中の内部レジスタである. 一クロック毎にカウンタが上がっていく.

きる. 無線 LAN の方で時間がかかっているのは, デバイスドライバがシグナルを何秒毎かに調べているためである. しかし, 検知から MIBsocket にトラップが送信されネットワークソフトウエアが受け取るまでの時間は約 500usec である. これは, ネットワークソフトウエアに取って動的にネットワークエンティティの変化に対応できる時間である. インタフェース切替え機構も図 8.6で見たように TCP のセッションに影響を及ぼさない時間無いにインタフェースを切替えて通信を再開できている.

MIBsocket のトラップ機構は, ネットワークエンティティの状態変化を検知しネットワークソフトウェアに通知することができた. ネットワークエンティティの状態変化を検知し通知するオーバヘッドもソケットにメッセージを送るなどの時間だけである. 図 8.7で分かるように, ネットワークソフトウェアはトラップを受け取って, それに対応する処理を行えている. ネットワークソフトウェアは, MIBsocket を用いることによりネットワークエンティティの状態変化に対応することができる.

第 9 章

結論

9.1 まとめ

MIBsocket を用いることにより, 動的なネットワークエンティティの変化に対応する機構が実現できた. インタフェースのリンクの状態や IP アドレスの取得あるいは削除をトラップとして受け取ることによりネットワークアプリケーションはそれに応じた動作が可能となった. また, MIBsocket を介しての設定を行うことにより, ネットワークエンティティに対する競合を防, 一貫性を保つことが可能となった. また, ネットワークソフトウエアはネットワークエンティティの状態を MIBsocket から取得できる. また, MIB でネットワークエンティティを管理したことにより汎用性が高まり全てのネットワークエンティティの情報をオブジェクト識別子で示せることができた.

ネットワークエンティティの状態の取得あるいはトラップにかかる MIBsocket のオーバヘッドも最小にした. これにより, ネットワークソフトウエアは十分にネットワークエンティティの変化に動的に対応できることが可能である.

ネットワークエンティティの状態を介してそれぞれのアプリケーションが協調動作することもできた. これは, インタフェース切替え機構と DHCP クライアント間で実際に協調動作できた. インタフェース切替え機構は DHCP が取得したアドレスをもとにインタフェースの選択をすることができる. DHCP がアドレスを取得できるまでは, インタフェース切替え機構はそのネットワークインタフェースを無効であると判断し利用しない. 取得後は有効になったと判断し, インタフェースの再選択を行う.

MIBsocket を用いてネットワークエンティティを統一的に管理することは実現できた.

9.2 今後の課題

今後の課題として, MIBsocket を用いた管理機構の実現が必要である. そのためにも, DHCP や MobileIP 等のネットワークソフトウエアを MIBsocket を用いて実装する. また, FreeBSD 上の実装だけでなく他の OS 上でも動作させる. また, MIBsocket を介した設定できるネットワークエンティティを増やすことである. 現在は, IP アドレスや経路しか MIBsocket からは設定できないがインタフェースの MTU などの値も MIBsocket から動的に設定できるようにする.

また, リンクの変化をトラップとして送信できるデバイスドライバを増やす. リンクの状態を検知できないデバイスへの対応を今後行う. 例えば NE2000 互換のイーサネットカード等のように

デバイスでリンクの状態を検知できないネットワークインタフェースのリンクをどのように扱うかを解決する.

その他にも, 移動体計算機では, 本論文で取り上げた以外に様々な問題点がある. 例えば, 電力管理等もそれである. 電力が無ければ, システムは動かない. 電力の有効利用や移動体計算機環境では重要である. 例えば, ネットワークインタフェースを使わない時は電力を供給せずに一時適に休止させたりするということである. 今後も, 本システムを応用して移動体計算機環境の支援を行っていく.

謝辞

本論文の作成にあたり, 御指導頂きました慶應義塾大学環境情報学部徳田英幸教授並びに同学部村井純教授に感謝致します.

絶えず御指導と御助言を頂きました慶應義塾環境情報学部の植原啓介助手, 同学部助手西尾信彦助手, 慶應義塾大学政策メディア研究科 2 年の田村陽介氏, 同博士課程 1 年の渡辺恭人氏に感謝致します.

Mobile-IP の利用に関して多大なる御助言と御協力を頂きました株式会社東芝の石山政浩氏に心からお礼を申し上げます.

本研究を進めていく上で様々な助力を与えて頂きました R3 プロジェクトの皆様及び奈良先端科学技術大学院情報科学センター砂原秀樹助教授をはじめとする rover プロジェクトの皆様に感謝致します.

本論文の作成にあたり, 多大なる御協力をして下さった慶應義塾大学環境情報学部 3 年の川喜田佑介氏, 佐藤雅明氏, 西野秀幸氏に感謝します.

本研究を進めて行く上でご支援下さった慶応義塾大学徳田・村井研究室並びに同研究室 NACM 研究グループ及び MCNG 研究グループの皆様に感謝の念を表します.

参考文献

[1] R.Droms.: “Dynamic Host Configuratin Protocol,” Request for Comments: 2131 March 1997.

[2] V. Jacobson. and R. Braden.: “TCP Extensions for Long-Delay Paths,” Network Working Group, Request for Comments:1072 Oct. 1988.

[3] M. Mathis and J. Mahdavi.: “Forward Acknowledgment: Refinding TCP Congestion Control.” In *Proceedings of ACM SIGCOMM ’96* Aug. 1996.

[4] C.Perkins.: “IP Mobility Support,” Request for Comments: 2002 Oct. 1996.

[5] 砂原秀樹, 比良木貴志, 植原啓介, 尾家祐二: “移動体端末装置における通信インターフェースの自動選択機能の実現,” DiCoMo, 1997.

[6] Piscitello, D. M., and Chapin, A, L.: “Open Systems Networking: TCP/IP and OSI.” AddisonWesley, Reading, Mass. 1993.

[7] McGregor, G.: “PPP Internet Protocol Control Protocol(IPCP),” Request for Comments: 1332 May 1992

[8] Mckusick, M. k., Bostic, K., Karels, M.j.,: “The Design and Implementation of the 4.4BSD Operating System.,” Addison-Wesley, 1996.

[9] McCloghrie,k., and Rose, M. T.: “Management Information Base for Network Management of TCP/IP-based Internets: MIB-II,” Request for Comments: 1213 Mar 1991.

[10] Jon Inouye, Jim Binkley, Jonathan Walpole.: “Dynamic Network Reconfiguration Support for Mobile Computers” In *Proceeding of MOBICOM ’97*, Sep. 1997.

[11] Jeffrey D. Case, Mark S. Fedor, Martin L. Schoffstall, and James R. Davin.: “A Simple Network Management Protocol,” Request for Comments: 1157 May 1990

[12] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, Jorjeta Jetcheva.: “A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocol s” In *Proceeding of MOBICOM ’98* Oct. 1998.

[13] V. D. Park and M. S. Corson.: “A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks,” In *Proceedings of INFOCOM ’97*, Apr. 1997.

[14] J.Postel.: “Transmission control protocol,” Request for Comments: 793 Sep. 1981.

[15] J. Postel.: "User Datagram Protocol," Request for Comments: 768 Aug. 1980.

[16] : "Information Processing - Open Systems Interconnection - Abstrace Syntax Notati on One(ASN.1) - Addendum 1: Extensions to ASN.1," International Organizetion for standardization and International Electrotechnical Committee, 1987. International Standard 8824/AD 1.

[17] INTEL CORPORATION.: "Advanced Power Management (APM) BIOS Interface Specification.," February 1996.

[18] J.Postel.: "Internet Protocol," Request for Comments 791, Sep. 1981.

[19] Mary G. Baker, Xinhua Zhao, Stuart Cheshire and Jonathan Stone.: "Supporting Mobility in MosquitoNet" In *Proceedings of the 1996 USENIX Technical Conference* Jan. 1996.

[20] Mary G. Baker.: "Changing Communication Environments in MosquitoNet," In *Proceedings of 1994 IEEE Workshop on Mobile Computing Systems and Applications* Dec. 1994.

[21] Jon Inouye, Shanwai Cen, Calton Pu and Jonathan Walpole.: "System Support for Mobile Multimedia Applications," In *Proceedings of the 7th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '97)* June 1997.

[22] By Wright, Gary R. Stevens, W. Richard,: "TCP/IP Illustrated, Volume 2", Addison Wesley; 10/94; ISBN: 020163354X

[23] E. M. Gafni and D. P. Bertsekas,: "Distributed algorithms for generating loop-free routes in networks with frequently changing topology," IEEE Transactions on Communications, vol.COM-29, pp. 11-18, 1981.

[24] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tiltom, Jason Flinn, Kevin R. Walker,: "Agile Application-Aware Adaptation for Mobility," In *Proceedings of the 16th ACM Symposium on Operating System Principles, Oct. 1997*

[25] M. Satyanarayanan, Brian Noble, Puneet Kumar, Morgan Price,: "Application-Aware Adaptation for Mobile Computing," In *Proceedings of the 6th ACM SIGOPS European Workshop, Sep. 1994*