レイヤ間情報伝達機構 LIES

LIES:A inter Layer Information Exchange System for Mobile Communication

渋井 理恵　　神谷 弘樹　　寺岡 文男
慶應義塾大学理工学研究科

2004 年 10 月 1 日

概 要

モバイル環境などの変化する環境に対して、ノードがその挙動の最適化を図るためには、他レイヤの情報を取得・利用して環境の変化を知り、動的に対応する必要がある。しかし、現在の OSI 基本参照モデルでは各レイヤは独立して設計されており、レイヤ間での情報交換は重要視されていない。現在、ユーザランド上のアプリケーションがカーネル内のレイヤと情報交換をするための方式はいくつか提案されているが、ネットワークレイヤの高速ハンドオーバや TCP の最適化の実現などのためには、ユーザランドとカーネル間だけではなく、カーネル内のレイヤ間においても情報交換を行なうことのできる統一的な機構が必要となる。そこで本論文では、各レイヤのプロトコルエンティティの情報を抽象化する AE(Abstract Entity) と、AE を仲介し情報を他レイヤに伝達する ILS(Inter Layer System) を備えた、任意のレイヤ間での動的な情報伝達機構である LIES を設計、実装、評価する。

1　はじめに

近年、コンピュータのおかれる環境は静的なものではなく動的に変化するものとなってきている。デバイスやコンピュータリソースが変化していくのに伴い、OS やアプリケーションはその変化に対応していかなければならない。特に、モバイルノードが異なるネットワークへ移動する場合にはそれを取り巻く環境は大きく変わる。これらの変化する環境へ動的に対応し最適化を図るためには、各レイヤの情報を他レイヤで利用することが必要となるが、現在のインターネットプロトコルではレイヤ毎に独立した階層構造をなしているため、レイヤの内部情報を他レイヤが取得することはできない。

特に、モビリティサポートプロトコルである Mobile IP[3] を改良した高速ハンドオーバを提案している Low Latency Handoffs in Mobile IPv4[4] および Fast Handoverfor Mobile IPv6[5] では、モバイルコンピューティングにおけるネットワークハンドオーバにおいて、リンクレイヤ (L2) の情報をネットワークレイヤ (L3) にて利用することにより遅延の少ないハンドオーバが実現できるとしており、変化する環境に対応するためにはレイヤ間での動的な情報交換が必須であるが、レイヤ間での統一的な情報交換機構は未だ実現されていない。

そこで本論文では、任意のレイヤ間で動的な情報伝達を行なうための機構 LIES を提案し、設計・実装する。LIES を用いることにより、レイヤの各プロトコルエンティティ (PE) は他レイヤに存在する PE の情報を取得することができ、また、PE で発生したイベントを他 PE に動的に通知することもできる。本論文では評価のため、LIES を用いた情報伝達に要する時間を測定し、変化する環境に対して動的に対応するに十分有効な機構であることを示す。

本論文は次のような構成である。2 章ではレイヤ間情報伝達機構の必要性と具体例を示し、3 章では関連研究として Odyssey と MIBsocket を説明する。4 章、5 章で LIES のアーキテクチャと設計・実装について述べ、6 章にて LIES を利用する応用例として高速ハンドオーバについて説明する。そして 7 章で LIES の評価を行ない、8 章で本論文の結論について述べる。

2　レイヤ間情報伝達機構の必要性

現在、ISO(International Organization for Standardization) で定義されている OSI(Open Systems Interconnection) 基本参照モデルは各レイヤが独立して設計されており、基本的には他レイヤの情報は取得できない構造となっている。この基本参照モデルはプロトコルに独立性を持たせプロトコルの管理を容易にするが、一方で他のレイヤで発生したイベントや状態を得ることができないという欠点が存在する。他レイヤの情報を取得できることにより変化する環境に動的に対応する例として、次の 2 つの例を示す。リンクレイヤ (L2) 情報を利用したネットワークレイヤ (L3) の高速ハンドオーバの実現と、リンクレイヤ (L2) 情報を利用したトランスポートレイヤ (L4) の最適化である。

2.1　ネットワークレイヤ (L3) における高速ハンドオーバの実現

リンクレイヤ (L2) とネットワークレイヤ (L3) 間での情報伝達の例として、ある無線 LAN の基地局に接続しているモバイルノードが、他の基地局に接続を切り替えるというモバイルシステムにおけるハンドオーバを取り上げる。ここで接続している基地局を peer と呼ぶことにする。こ

のハンドオーバはL2ハンドオーバとL3ハンドオーバに分けられる。L2ハンドオーバは現在のリンクの接続先 (peer) を物理的に切替えることであり、このとき一切の通信が途絶える。L3ハンドオーバは、現在のサブネットから異なるサブネットへ接続することであり、これに伴いIPアドレスも変わるため、新しいIPアドレスを取得するまで通信を開始することはできない。Mobile IP[3] による移動透過性を保証した通信を行なうためには、この後さらに、新しいIPアドレスを位置管理サーバに登録する必要がある。

モバイルシステムにおけるL3ハンドオーバは、L2ハンドオーバが完了してから開始される。しかし、L2ハンドオーバが完了したことをL3では検知することができないため、L2ハンドオーバが完了してからL3ハンドオーバが開始されるまでに遅延が生じてしまうという問題がある。この様子を図1に示す。

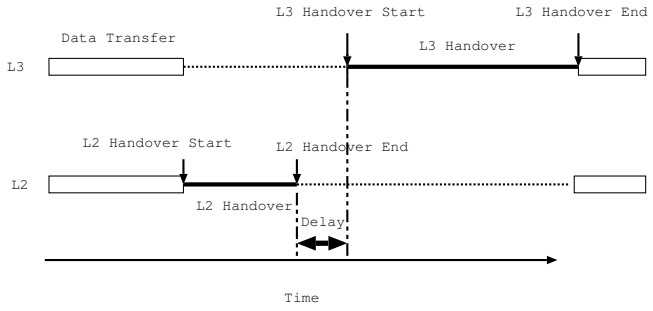

図 1: L2 and L3 Handover

ここでL3がL2に関する情報を得ることができれば、図2のようにL2ハンドオーバが完了した直後にL3ハンドオーバを開始することができ、L3ハンドオーバ開始までに生じる遅延を解消することができる。ここでいうL2の情報とは、L2ハンドオーバが完了したことを通知する情報を意味する。さらに、L2の情報をL3が得ることにより、L3主導の高速ハンドオーバが可能となる。この様子を図3に示す。

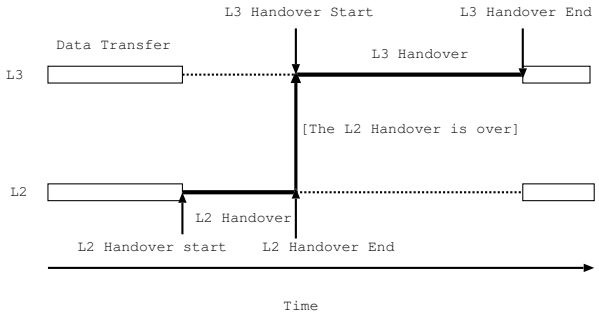

図 2: Optimized Handover

図3では、L2の現在接続しているリンクがじきに切断される可能性が高いことを意味する情報がL3へ通知されている。この通知を受けて、L3はL3ハンドオーバに移行するための準備処理に入る。L3ハンドオーバの準備処理を終了すると、L3はL2へL2ハンドオーバを開始するよう指示を送る。その指示によりL2はL2ハンドオーバを開始し、そして完了した時点でL2ハンドオーバの完了通知をL3に送る。L2ハンドオーバの完了通知を受けてL3は最終的なL3ハンドオーバを開始する。このとき前もってL3ハンドオーバの準備処理がなされているため、より少ない時間でL3ハンドオーバを完了してすぐに通信を開始することができる。L2の情報をL3が得ることができれば、このようにL3主導でハンドオーバを行なうことが可能となる。

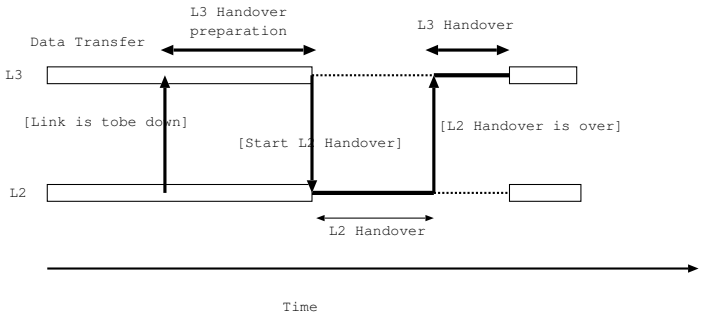

図 3: Fast Handover

2.2 ハンドオーバ時のTCP(L4)の最適化

次に、リンクレイヤ (L2) からトランスポートレイヤ (L4) への情報伝達の例を示す。モバイルノードが通信相手のノードからTCPパケットを受信しているときに、モバイルノードの移動によりハンドオーバが起こるとする。このハンドオーバ時には通信が途切れる時間が存在する。送信側ノードのTCPは確認応答ACKが返ってこない場合、タイムアウトするまで一定時間待機するが、一度タイムアウトするとタイムアウトまでの時間をさらに長くして待つ。このような方法では、もしモバイルノードのハンドオーバが完了してもTCPの通信を迅速に開始することはできない。ここで、ハンドオーバ完了時に、モバイルノードのL2のモジュールからL4内のTCPモジュールに、ハンドオーバが完了したという情報を伝達することができれば、モバイルノードのTCPモジュールが通信相手ノードに制御メッセージを送信し、その制御メッセージを受信した通信相手ノードはTCP通信を即座に開始することができる。

3 関連研究

モバイル環境に対するアプリケーションの挙動の最適化を焦点に当てた、レイヤ間の情報伝達手法はいくつか提案されている。本節では代表的な例として、Odyssey[1] とMIBsocket[2] という2つの手法を紹介する。

3.1 Odyssey

Odysseyは、アプリケーションがモバイル環境へ適応するための最適な手法であるapplication-aware adaptationを実現するプロトタイプである。Odysseyは、帯域、CPUサイクルやバッテリなどのリソースを監視し、各アプリケーションと連係して最適化を行なう。ユーザはアプリケーションの挙動の変化を認識はするが、最適化のための影響はなく、また何かの操作を開始する必要もない。

Odyssey では、図 4 に示すように、ユーザランド上に存在する *viceroy* と *warden* から構成されている。Odyssey に関する操作命令はカーネル内の interceptor モジュールにより *viceroy* へ送られるが、他の全てのシステムコールは NetBSD により処理される。*warden* はデータタイプ毎に存在し、そのデータをシステムレベルまでカプセル化する。*viceroy* はデータタイプに非依存であり、全体のリソース管理を行なう。*warden* がカプセル化した情報を *viceroy* が集約し、アプリケーションはそれらの情報を用いて挙動を制御することができる。サーバとのやりとりおよびデータのキャッシュについての全責任は *warden* が負っており、アプリケーションは直接サーバとやりとりをすることはない。

Odyssey はユーザスペースに実装されているが、機能性や敏捷性を考えると今後カーネル内に実装する必要がある。またこの Odyssey は、ユーザランド以外のレイヤに存在するエンティティの挙動の最適化を行なうことはできない。なぜならこの機構はユーザランドとカーネル間のやりとりのみに焦点に当てており、カーネル内のやりとりについては考えていないからである。

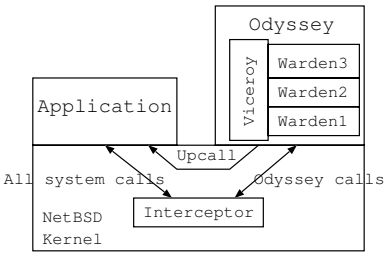

図 4: Odyssey Client Architecture

3.2 MIBsock

MIBsocket は、モバイルノード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である。アプリケーションおよびユーザは、MIBsocket を用いてネットワークエンティティの情報を取得または設定できる。カーネルとユーザランド間のネットワークエンティティのやりとりにソケットを用い、そのネットワークエンティティ情報を MIB(Management Information Base) を用いて表している。この MIBsocket には以下の操作がある。

・*Get*: ネットワークエンティティの情報をアプリケーションが取得する

・*Set*: ネットワークエンティティをアプリケーションから設定する

・*Trap*: ネットワークエンティティの変化をアプリケーションに通知する

3 番目の *Trap* 操作により、アプリケーションは下位レイヤの情報を動的に得ることができ、これにより変化するネットワーク環境に適応することができる。また、この他にフィルタリングの機能と、アプリケーションのパーミッションを設定する機能とがある。フィルタリングとは、各アプリケーションが取得したい情報の種類を限定するための機能であり、これにより不必要な情報をアプリケーションが取得してしまうことはない。また、パーミッションを設定することにより、複数のアプリケーションが同時にネットワークエンティティを設定しようとした場合などにも、矛盾を含むことなくネットワーク情報を保持することができる。

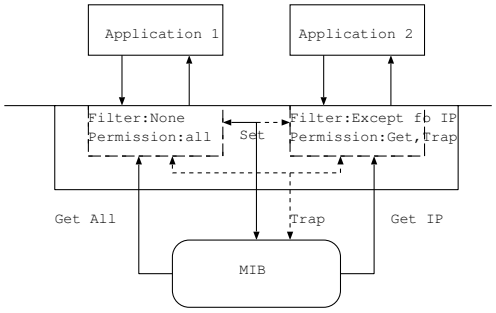

図 5: The design of MIBsocket

図 5 はこの機構の例を表している。各アプリケーションは MIBsocket を開いて、それぞれネットワークエンティティを取得/設定する。またこの例では、アプリケーション 2 はフィルタリング機能により IP に関する情報のみを取得し、パーミッション設定機能により *Get* および *Trap* しか行なえないことを表している。この MIBsocket もユーザランド上のアプリケーションとカーネルとの間での情報交換に限定されて設計されており、その他のカーネル内のレイヤ間での情報伝達は行なうことができない。

上述した 2 例の他にも、ユーザランドとカーネル間での情報交換についてはいくつか研究がなされているが、モバイル環境への最適化などには、リンクレイヤとネットワークレイヤ間やリンクレイヤとトランスポートレイヤ間など、カーネル内のレイヤ間での情報交換が必須となる。よって、ユーザランドとカーネル内のレイヤ間だけでなく、任意のレイヤ間で動的な情報交換を行なうことができる、統一的な手法が必要であるといえる。

4 LIES のアーキテクチャ

本節ではプロトコルレイヤ間の情報伝達機構である LIES の基本的なアーキテクチャについて述べる。

4.1 プロトコル階層の視点でのアーキテクチャ

プロトコルレイヤ間情報伝達機構 LIES は、図 6 に示すように、AE(Abstract Entity)、CIEP(Control Information Exchange Point)、ILS(Inter-Layer System) を持ち、OSI の階層構造を破壊することなく任意のレイヤ間での動的な情報交換を行なうことができる。AE は PE(Protocol Entity) と対になっており、AE は各 PE 依存の情報を PE に独立な情報 (Abstracted Information) へと抽象化する。ILS は全レイヤにまたがった形で構成されており、任意のレイヤの AE は抽象化された情報を ILS や CIEP を通して交換する。ここで CIEP は ILS と AE の境界に位置し、交換を補佐するモジュールである。

抽象化された情報の交換を行なう際、各 AE は図 7 に示した request, confirm, indication, response の 4 種類のプリミティブを用いてこれを行なう。request は他のレイヤの

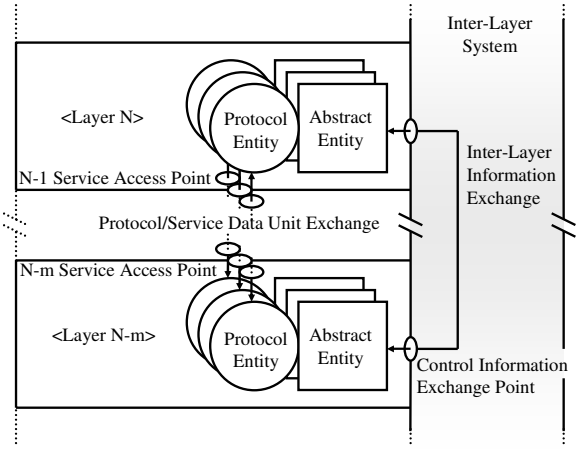

図 6: Layering Model of LIES

情報を取得したい AE が送るものであり、confirm はその request を受け取った AE が返す応答である。indication は AE への動的なイベント発生の通知であり、response はその indication に対する確認応答である。しかし、indication を受け取るためには、あらかじめ request により indication を要求しておく必要がある。

これら 4 種類のプリミティブは次の 5 つのフィールドから構成される。プリミティブが送られるべきレイヤを示す送信先プロトコルレイヤ ID、プリミティブが送られるべき PE を示す送信先プロトコル ID、(request,confirm,indication,response などの) プリミティブクラス、プリミティブタイプ、そしてパラメータである。各 AE は、これらのフィールドで構成されたプリミティブを用いて全ての情報交換を行なうことができる。

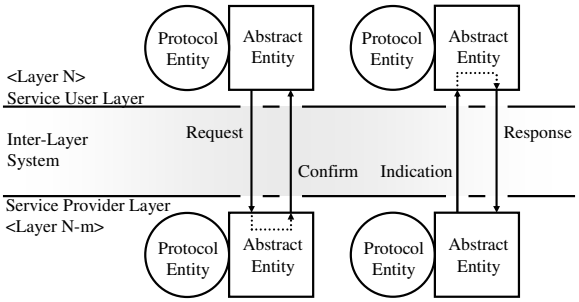

図 7: Primitives of AE

4.2 OS の構造の観点でのアーキテクチャ

上述したプロトコル間の情報交換アーキテクチャをインターネットプロトコルスタックに適用したものが図 8 である。トランスポートレイヤでは TCP-PE と UDP-PE、ネットワークレイヤでは IPv4-PE および IPv6-PE の 2 つの PE がある。リンクレイヤは、LLC(Logical Link Control) サブレイヤと MAC(Media Access Control) サブレイヤの 2 つに分割され、図 8 においては Ethernet と CDMA の 2 種類の通信方式が示されている。Ethernet-PE の下位のレイヤには有線や無線などのデバイスが存在する。また、図 8 からも分かるように、すべての各 PE は対応する AE(TCP-AE や IPv6-AE など) と対になって構成されている。

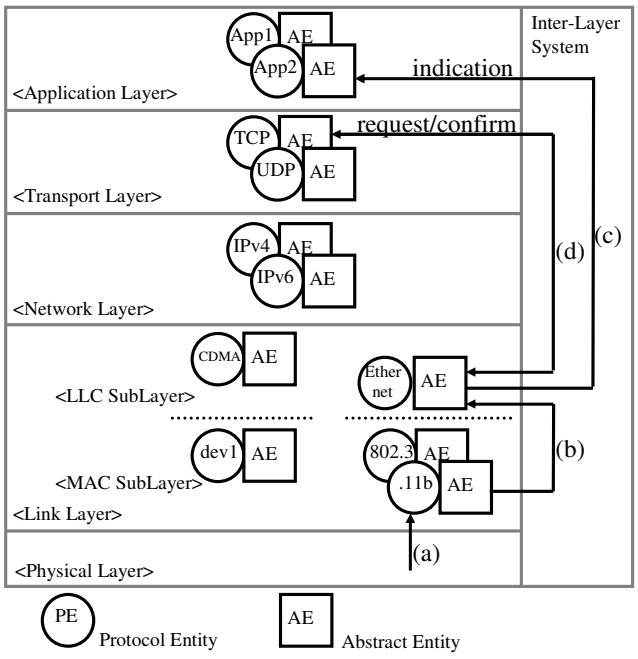

図 8: Layering Model for Internet Protocol Stack

図 8 では、4 つのプリミティブを用いた情報交換の例も示してある。まず、図 8 において、物理的なデバイスカードからデバイス依存のドライバへ、IEEE802.11b がリンクアップしたというイベントが通知される (図 8-(a))。このイベントは Ethernet-AE へ通知される (図 8-(b))。ここで図 8 中の application-AE が、あらかじめリンクアップというイベントの発生の通知を request プリミティブを用いて要求していた (図 8-(c)) とすると、Ethernet-AE は application-AE へそのイベントの発生を indiation プリミティブを用いて通知する (図 8-(d))。このようにして、他レイヤの情報を得たいレイヤの AE が request を送り、confirm が返され、取得したいイベントが発生した場合には indication により動的に通知されることになる。

5 LIES の設計と実装

本章ではプロトコル情報伝達機構 LIES の設計および実装について述べる。

5.1 LIES の設計

以下で LIES の主な機能を担う AE(Abstract Entity) と ILS(Inter Layer System) の設計について述べる。

5.1.1 AE の設計

プロトコル間情報伝達機構 LIES における AE は、abstract function, request handler, indication handler の 3 つの主な機能を持つ。abstruct function は、その AE と対をなす PE の情報を抽象化し、PE 依存な情報から PE 非依存な共通の情報 (Abstracted Information) へと変換す

る。request handler は、他の AE からの request を受け取りその処理を行なう。indication handler は、通知の要求がされているイベントの発生を検知して indication を作成し、その要求をしていた AE に indication を送ることによりイベント発生を通知する。ここで、AE が indication により特定のイベント発生の通知を受け取るためには、あらかじめ request によりその要求をしておかなければならないため、各 AE には、イベント発生通知の indication を要求する request を保持するために request table が存在する。indication handler はイベント発生毎にこの request table を検索し、発生したイベントの通知が他の AE から要求されていれば indication を作成し、要求元である AE に indication を送る。

5.1.2 ILS の設計

ILS は、request を適切な AE に割り振るモジュールである dispatcher、および、作成された indication を保持するための indication table を、(トランスポートレイヤ、ネットワークレイヤ、リンクレイヤなどの) 各プロトコルレイヤ毎に持つ。

ある AE(送信元 AE) から他レイヤの AE(送信先 AE) に request が送られるとき、その request はまず、request の送信先プロトコル ID フィールドに記されているレイヤの dispatcher へ送られる。その後その dispatcher から、同じく request の送信先プロトコル ID フィールドに示されている AE へと適切に送られる。また confirm も、request が経由した dispatcher を経由して request 送信元 AE へと送られる。

indication table は、送られるべき indication を保持しておくためのテーブルである。この indication table の各エントリは、ターゲット ID、indication タイプ、パラメータの 3 つのフィールドから構成される。ターゲット ID はその indication を送るべき送信先 AE を示し、indication タイプはその indication がどのイベントの発生を通知するものかを示す。パラメータは発生したイベントの詳細を表している。この indication table は、ユーザランドとカーネルの間での情報交換時にのみ必要となり、カーネル内での情報交換には必要ない。この理由については後の章で詳しく述べる。

5.2 LIES の実装

今回提案するプロトコルレイヤ間情報伝達機構 LIES を、NetBSD-2.0 上に実装した。LIES は任意のプロトコルレイヤ間で情報交換を可能にするが、本論文では、特にリンクレイヤと他レイヤでの情報伝達に焦点をあてた。そのため、TCP-PE、IP-PE、Ethernet-PE、そして有線 Ethernet ドライバに対応する AE、および ILS をカーネル内に実装した。以下で各プリミティブの説明をした後、具体的な情報交換の流れを 2 つの例を挙げて説明する。

5.2.1 Request/Confirm プリミティブ

これまでも述べてきたように、LIES において request プリミティブにはいくつかの種類があり、それらは 3 つに分類することができる。ひとつはその時点での他レイヤの情報を得るためのものであり、この場合、request を受け取った AE(request 送信先 AE) が、要求された情報を含む confirm を作成する。そしてこれを request への応答として、request を送った AE(request 送信元 AE) へ送る。

もうひとつは indication による特定のイベント発生の通知を要求するためのものである。この場合、request はまず request 送信先 AE の request table に登録される。その後 request 送信先 AE は確認応答としての confirm を作成し、request 送信元 AE へと送る。request table はイベント発生毎に indication handler に参照され、もし発生したイベントに対する request があれば indication が作成される。

最後の request は、特定の挙動の実行を命令するためのものである。request 送信先 AE は request に対する確認応答として confirm を作成して request 送信元 AE へ送り、さらに要求された挙動の実行のための処理を行なう。ここで確認応答として送られる confirm は、命令を受け取ったことを示す確認応答であり、命令の実行が成功したことを意味するものではない。

また、request と confirm の送信方法は、request 送信元 AE がカーネル内にある場合とユーザランドにある場合とで異なる。カーネル内からカーネル内へ request を送る場合は、request 送信先となるレイヤの dispatcher を直接呼び出せるが、ユーザランドからカーネル内へ request を送る場合は、ioctl システムコールを用いて dispatcher へ送る必要がある。

5.2.2 Indication プリミティブ

indication によるイベント発生の通知方法は、indication の送信先となる AE(indication 送信先 AE) がカーネル内であるかユーザランドであるかによって異なる。indication 送信先 AE がカーネル内に存在する場合、indication 送信元 AE は indication 送信先 AE の indication handler を直接呼び出して、indication を渡すことができる。

しかし indication 送信先 AE がユーザランド内に存在する場合は、多少複雑になる。ユーザランドはカーネル内からの情報を直接受け取ることはできないため、いくつかのシステムコールを用いて情報を交換する。まずユーザランドは ioctl システムコールを用いて request/confirm の交換を行なう。そして、カーネルからの indication を取得するために、kqueue というカーネルイベントキューを用いてイベント発生の情報を待つ。kqueue とはカーネル内でのイベント発生をユーザランドが検知するためのキューであり、ユーザランドはこのキューに入った情報を kevent システムコールを用いて取り出すことができる。ここでユーザランドはあらかじめ kqueue システムコールを用いて kqueue を作成しておく必要がある。

カーネル内にある indication 送信元 AE はイベント発生を検知して indication を作成した後、それを ILS 内の indication table に登録する。そして、knote 関数を用いて、作成した indication のタイプを表す情報を kqueue に入れる。ユーザランドは kevent システムコールを用いて

kqueue に入れられた情報を取り出し、イベントが発生したことを検知する。しかし、kqueue により伝達することのできる情報量は限られているため、ユーザランド上の indication 送信先 AE は別途 ioctl システムコールを用いて、残りの indiation 情報を取得する必要がある。

また、通知すべき発生イベントがハードウェアインタラプトによるものかどうかにより、イベント発生の通知方法が異なる。例えば、リンクアップなどのイベントは、ネットワークインタフェースカードなどのハードウェアインタラプトにより通知されるイベントであるが、この場合、デバイスの interrupt handler が実行され、この handler がデバイス非依存な PE の interrupt handler を呼び出す。この interrupt handler は、発生したイベントをそのデバイス非依存な PE とそれに対応する AE との境界に位置するキューにいれて、ソフトウェアインタラプトを発生させる。このソフトウェアインタラプトはキューからイベント情報を取り出し、さらにそのデバイス非依存な AE の indication handler を呼んで処理をする。イベントがハードウェアインタラプトによって発生するものでない場合は、AE の indication handler が直接呼ばれることになる。次の 2 節で具体的な情報交換の流れを説明する。ひとつはカーネルとユーザランド間の情報伝達、もうひとつはカーネル内のレイヤ間の情報伝達である。

5.2.3 リンクレイヤからアプリケーションレイヤ上のデーモンへの情報伝達

カーネルとユーザランド間の情報交換の例として、リンクレイヤからユーザランドのアプリケーションプロセスへの情報伝達を挙げる。アプリケーションプロセス (L7) がリンクレイヤ (L2) の情報を取得したい場合の情報伝達の様子を図 9 に示す。まず、L7 のアプリケーションプロセスが request を作成し、ioctl システムコールを用いて ILS 内の L2-dispatcher へ送信する。L2-dispatcher は request のフィールドを読み取り、それが Ethernet-AE へのものであると判断してそれを送る。Ethernet-AE の request handler は request を受け取り、それがその時点での L2 情報を取得するためのものであった場合は、要求された情報を含む confirm メッセージを作成し、アプリケーションプロセスへ送る。もし request が indication 要求のためのものであった場合は、Ethernet-AE がその request を request table に登録し、その後確認応答のための confirm を作成して送る。ここでアプリケーションプロセスは、新しい基地局を示す peer の出現に対して indication を要求する request を送ったとする。これはつまり、新しい peer の強度がアプリケーションが指定した閾値を越えた時に、indication による通知を要求するものである。

まず 802.11b のデバイスが、ある peer の強度が変化した、といったイベントの発生を検知する。すると、802.11b デバイスの AE 内の abstract function がその情報を抽象化し、Ethernet-PE に渡す。Ethernet-PE は、その抽象化されたイベント情報を Ethernet-AE との境界にある LIES-queue に入れる。さらに Ethernet-AE の indication handler が LIES-queue からイベント情報を取り出し、現在の peer の状態を表す peer table と比較し、さらに request table から request を検索してそれが request に合致するイベントであれば indication を作成する。この例において request に合致するということは、デバイスにより検知された peer の強度が閾値より上回ったことを意味する。前節で説明したソフトウェアインタラプトは Ethernet-AE が LIES-queue からイベント情報を取り出すときに発生する。Ethernet-AE は必要に応じて indication を作成した後、indication 送信先を示す送信先 ID(target ID) と共に ILS 内の L2-indication table に登録する。そしてユーザランドアプリケーションにイベントの発生を知らせるため、knote 関数を用いてアプリケーションが予め作成しておいた kqueue に indication の種類を表す情報を入れる。アプリケーションは kqueue/kevent システムコールにより、kqueue に入れられたイベント情報を取り出してイベントの発生を検知した後、別途 ioctl システムコールを用いて残りの indication 情報を取得する。

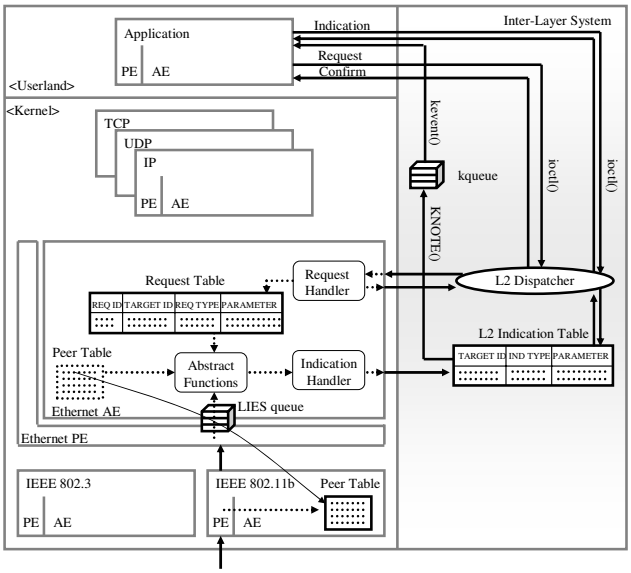

図 9: Example between L2 and L7

5.2.4 リンクレイヤから TCP への情報伝達

次に、カーネル内の情報交換の例として、リンクレイヤ (L2) からトランスポートレイヤ (L4) への情報伝達を挙げる。TCP(トランスポートレイヤ) が L2(リンクレイヤ) に関する情報を取得したいものとする。図 10 はその様子を表している。

TCP-AE は request を作成して ILS 内の L2-dispatcher へ送り、L2-dispatcher は Ethernet-AE へ送る。作成した request がその時点での L2 情報を取得するためのものである場合は、Ethernet-AE の request handler は必要な情報を含んだ confirm を作成し、直接 TCP-AE へ送る。作成した request が indication 要求のためのものである場合、Ethernet-AE の request handler は request table にそれを登録し、イベントの発生を待つ。イベントが発生するとそのイベントは抽象化されて LIES-queue に入れられ、それを Ethernet-AE の indication handler が取り出し、さらに request table と peer table を比較／検索し、必要で

あれば indication を作成する。そして request table に共に記されている送信先 ID(target ID) をもとに、作成した indication を TCP-AE に送る。TCP-AE を示すこの送信先 ID(target ID) は通常、その indication を渡すべき関数のポインタを指している。これらの処理はカーネル内での情報交換であるため、ユーザランド-カーネル間での交換のように ioctl や kqueue/kevent システムコールを用いる必要がない。また、作成した indication を全て直接関数に渡すことができるため、再び全情報を取得するための動作も ILS 内の indication table も必要ない。indication を受け取った TCP-AE 内の関数は、その情報を元に TCP の挙動を最適化することができる。

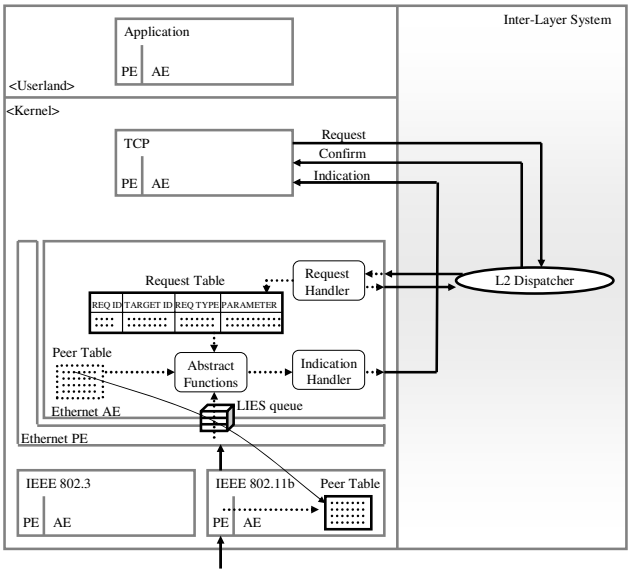

図 10: Example between L2 and L4

6 高速ハンドオーバサポート

本章では、LIES を利用する応用例として、Horizontal ハンドオーバと Vertical ハンドオーバの 2 種類のハンドオーバについて述べる。

6.1 Horizontal Handover

Horizontal ハンドオーバは、モバイルノードが持つ単一のネットワークインタフェースが、異なる peer に接続を切替えるハンドオーバのことであり、2.1 章で述べたハンドオーバを指す。このハンドオーバ形態では、L2 ハンドオーバが完了した後、L3 ハンドオーバ処理開始までの遅延が生じ、この間通信が遮断される。L2 の情報を L3 にて利用することにより、この遮断される時間を少なくし、遅延の小さいハンドオーバを実現することができる。また Horizontal ハンドオーバには、peer が切断されてから接続するハンドオーバと、peer が切断することを予想してあらかじめハンドオーバのための準備をする L3 主導のハンドオーバとがある。この L3 主導 Horizontal ハンドオーバは Low Latency Handoffs in Mobile IPv4[4] や Fast Handovers for Mobile IPv6[5] でも提案されている。これらについては 2.1 章で示した通りである。

6.2 Vertical Handover

Vertical ハンドオーバは、モバイルノードが現在通信しているネットワークインタフェースから異なるネットワークインタフェースへ通信を切替えるハンドオーバのことを指す。このハンドオーバ形態には、L2 が複数存在する中、L3 の判断で L2 を選択する機能が必要である。Vertical ハンドオーバの処理手順を図 11 に示す。L3 はあらかじめ、L2 選択の判断に必要な、ユーザの要求にあったポリシを持っているとする。図 11 において、ノードには a,b,c の 3

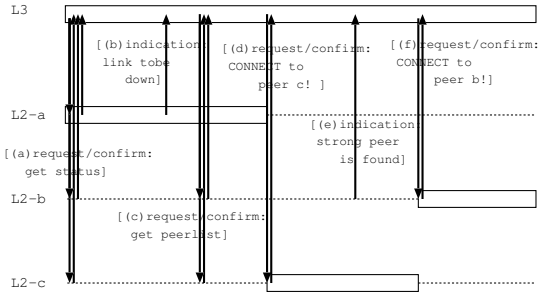

図 11: Vertical Handover

つのネットワークインタフェースが存在している。まず最初に、abc 各リンクの状況を取得するための request を各リンクに送り、confirm によりその時点でのリンクの情報を取得する (図 11-(a))。また、このとき後に発生するであろうイベントの indication を受け取るための request を送っておくこととする。図 11 では最初リンク a に接続しているが、ネットワーク環境の変化に伴い、リンク a の電波強度が弱まり切れそうであるという K2-a からの indication を L3 が受け取る (図 11-(b))。この通知を受け、リンク a に代わるリンクを判断するため、L3 はリンク b,c における現在の peer 情報を取得するための request を送り、confirm により情報を受け取る (図 11-(c))。その結果、L3 は L2-c が最適なリンクであると判断し、リンクを c に切替えるための request/confirm によりリンクを c に切替える (図 11-(d))。しばらく経つとリンク b の peer が見つかったという L2-b からの indication を L3 が受け取る (図 11-(e))。この通知を受けて L3 は L2-b が最適なリンクであると判断し、リンクを b に切替えるための request/confirm を L2-b に送ることによりリンクを L2-b に切替える (図 11-(f))。このように、L3 は複数の L2 を L3 ハンドオーバによって切替えることで、単一のネットワークインタフェースに縛られることなく通信を継続することができる。図 11 以外にも、Vertical ハンドオーバと Horizontal ハンドオーバが同時並行的に発生する場合や、複数の L2 を同時に利用するような場合も想定できる。これらの事例は今後の課題とする。

7 LIES の評価

本論文で提案したレイヤ間情報伝達機構 LIES を用いた情報の伝達に要する時間を測定した。カーネルとユーザランド間での情報交換と、カーネル内のレイヤ間での情報

交換に要する時間を測定し評価を行なった。

7.1 カーネルとユーザランド間の情報交換

表 1 および図 12 は、リンクレイヤ (L2) がイベントの発生を検知してからユーザランドアプリケーション (L7) が indication を受け取るまでに要した時間を表している。これはカーネルとユーザランド間の情報交換であり、indication を受け取るためには kqueue/kevent や ioctl といったシステムコールや knote のような関数も必要となる。評価のために、次の 6 点において時間を測定した。

・(A1). デバイスが peer 情報を検知する
・(A2). デバイスが抽象化された情報を Ethernet-PE に渡す
・(A3). Ethernet-AE が LIES-queue から情報を取り出す
・(A4). Ethernet-AE が indication 作成後、情報を伝達するために knote 関数を読む
・(A5). アプリケーションが *kevent* システムコール によりイベント発生を検知する
・(A6). アプリケーションが *ioctl* システムコールにより、ILS 内にある indication table から全情報を取得する

(A1)-(A2) 間は、デバイスが受け取った peer 情報を抽象化するための時間である。(A2)-(A3) 間は、LIES-queue から抽象化情報を取り出すために要する時間である。(A3)-(A4) 間は、indication 作成に要する時間である。(A4)-(A5) 間は、Ethernet-AE が knote 関数を実行してから、ユーザランドがイベント発生を検知するまでに要した時間である。(A5)-(A6) 間は、ioctl システムコールによりユーザランドが indication の全情報を取得するのに要した時間である。この結果、LIES のオーバヘッドは十分に小さく、従ってモバイル環境など変化する環境に動的に対応するために、十分有用な機構であると言える。

表 1: Overhead of indication

(A1)-(A2) μsec	(A2)-(A3) μsec	(A3)-(A4) μsec	(A4)-(A5) μsec	(A5)-(A6) μsec
8.2	8.5	2.0	11.4	0.7

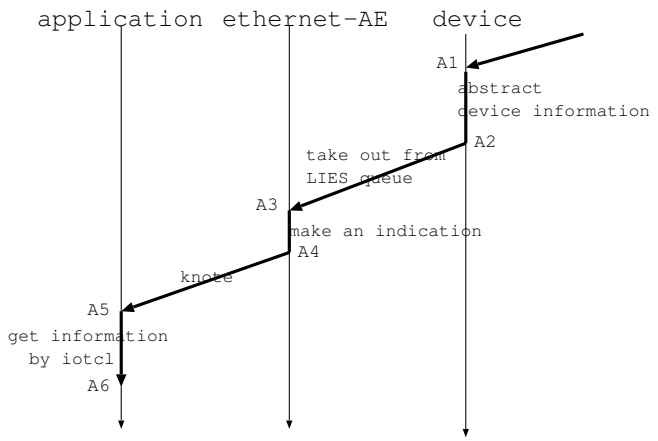

図 12: Overhead of getting indication

7.2 カーネル内での情報交換

続いてカーネル内のレイヤ間での情報伝達に要する時間であるが、リンクレイヤ (L2) がイベントの発生を検知してから TCP(L4) が indication を受け取るまでの時間を測定した結果、情報伝達に要する時間は μsec 以下であった。ユーザランドとカーネル間での情報伝達と異なり、カーネル内で indication を伝達するときは、kqueue/kevent/ioctl システムコールや knote 関数を使う必要はなく、また indication table を使用する必要もない。request を受け取った AE は request table を元に indication を作成し、request table に共に記された送信先 ID(target-id) の指し示す関数にその indication を直接渡すことになる。従って、indication を要求した AE は、indication 送信元 AE から直接 indication を受け取るだけであり、LIES による情報伝達に関するオーバヘッドはほぼないに等しいということができる。

8 結論

本論文では、任意のレイヤ間における動的な情報伝達機構である LIES を設計、実装し、評価した。LIES では、AE(Abstract Entity) が各 PE(Protocol Entity) の情報を抽象化し、それをプロトコルレイヤにまたがって設置されている ILS(Inter Layer System) が中継することにより、任意のレイヤのプロトコルエンティティ間で統一的に情報交換を行なうことができる。リンクレイヤと他レイヤとの情報交換に焦点を当てて実装を行い検証を行なった結果、LIES による情報伝達に要する時間は十分に小さく、高速ハンドオーバなどの変化する環境への最適化に有効な機構であると結論づけられた。

参考文献

[1] Brian D.Noble and M.Satyanarayanan and Dushyanth Narayanan, *Agile Application-Aware Adaptation for Mobility*, In Sixteen ACM Symposium on Operating Systems Principles, pages 276–287, France, Oct. 1997.

[2] R. Wakikawa, K. Uehara, F. Teraoka, and J. Murai, *MIBsocket: An Integrated Mechanism to Manipulate General Network Information in Mobile Communications.*, In IEICE transactions, The Institute of Electronics, Information and Communication Engineers, Aug.2001.

[3] D. Johnson and C. Perkins and J. Arkko, *Mobility Support in IPv6* , RFC3775. June. 2004.

[4] K. El-Malki, *Low Latency Handoffs in Mobile IPv4*, IETF Internet Draft (work in progress). June. 2004.

[5] R. Koodli, *Fast Handovers for Mobile IPv6*, IETF Internet Draft (work in progress). July. 2004.