社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2007－ＭＢＬ－41（２）
2007／5／17

通信ポリシを利用したマルチベアラ利用制御ミドルウェア

塩田　尚基† 　富森　博幸† 　奥山　嘉昭‡
浅井　伸一† 　佐藤　直樹‡

† 日本電気株式会社サービスプラットフォーム研究所 〒108-8557 東京都港区芝浦 2-11-5(五十嵐ビル)
‡ 日本電気株式会社システムプラットフォーム研究所 〒211-8666 神奈川県川崎市中原区下沼部 1753
E-mail: † n-shiota@ay.jp.nec.com,

あらまし　近年の無線通信技術の発展に伴い、目的や用途の違いに応じた様々な無線通信規格が提案・標準化されている。一方で携帯端末の高性能化も進んでおり、複数の無線通信手段を備える通信端末も一般的になりつつある。このような中で、携帯端末上で動作するアプリケーションから、複数の無線通信手段を容易に切り換えて利用したいという要求が現れている。しかしながら、既存のシステムではこの要求を満たすような機能が提供されているとは言えない。本論文では、アプリケーションの開発負担を増やすことなく、複数の通信手段の中から通信ポリシに従って適切な通信手段を自動的に選択することのできる基盤ソフトウェアを提案し、性能を評価する。

キーワード　マルチベアラ、ミドルウェア、通信制御ポリシ

Middleware for Controlling Multiple Bearer Utilization using Communication Policy

Naoki SHIOTA† 　Hiroyuki TOMIMORI† 　Yoshiaki OKUYAMA‡
Shinichi ASAI† 　and 　Naoki SATOH‡

† Service Platform Lab., NEC Corporation, Igarashi Building, 2-11-5 Shibaura, Minato-ku, Tokyo, 108-8557 Japan
‡ System Platform Lab., NEC Corporation, 1753 Shimonumabe, Nakahara-ku, Kawasaki-Shi, Kanagawa, 211-8666 Japan
E-mail: † n-shiota@ay.jp.nec.com

Abstract　In recent years, there have appeared many radio communication standards. Meanwhile, mobile terminals with multiple communication interfaces have become popular, because of the improvement in the performance and the power efficiency of terminals. Under such environments, there arises a requirement to select and use a bearer easily. But existing systems don't have enough functions to satisfy such a requirement. In this paper, we propose a middleware which automatically selects a bearer using a communication policy. Furthermore, we evaluate the overhead that the proposed middleware causes on an application performance.

Keyword　Multiple Bearer, Middleware, Communication Policy

1. はじめに

近年の無線通信技術の発展と携帯端末の省電力化・小型化により、複数の通信手段を持つ端末が普及し始めている。このような端末では、一般的に利用シーンによって通信手段を切り換えて通信を行う。携帯電話におけるビジネス向けの例としては、社内での通話には無線 LAN で VoIP を使い、外出時はセルラー網を使うというような用途が挙げられる。一般向けの用途では、家にいる間は Web 閲覧に無線 LAN を利用し、外出時はセルラー網のパケット通信を利用するという例が挙げられる。

一方、高性能な通信端末の普及に従って様々な通信サービスが利用可能になりつつあり、複数の通信手段を必要に応じて切り換えたり、同時に利用したりするニーズはこれから爆発的に増加することが予想される。

しかしながら現状の携帯端末のシステムにおいては、通信手段の選択のために容易な方法が提供されているとは言えない。このため、複数の通信手段を切り替えて使うようなアプリケーションは挙動が複雑になり、新規アプリケーションの開発コストや既存のアプリケーションの移植コストの増大が懸念される。

また、複数の通信手段を利用することによるセキュリティ上の新たな問題がある。通信端末が複数の通信手段を持つことで、複数のネットワークと通信を行う

アプリケーションが情報漏えいのようなリスクを発生させることが考えられる。

本論文では、アプリケーションの開発コストの削減とセキュリティ上の対策のために、通信ポリシを用いて利用する通信手段を選択・制限することのできるミドルウェアを提案する。また、試作した提案ミドルウェアが通信性能に与えるオーバーヘッドを測定し、測定結果について考察を行う。

2. 提案ミドルウェア

2.1. 概要

マルチベアラ端末における通信を制御する、Linux上で動作するミドルウェアを提案する。

現在のシステムでは、ベアラ選択のためにアプリケーションは利用するベアラをコネクションごとに明示的に設定したり、ルーティングテーブルに適当なエントリーを追加したりする必要がある。前者の方法は煩雑であり、後者の方法はシステム全体の通信に影響を及ぼしてしまうという問題があった。

また、アプリケーションから複数のベアラを利用できるようになると、一方のネットワークから取得した情報を他方のネットワークへ送信するようなアプリケーションが容易に作成できるようになるため、情報漏えいの原因となる。

提案ミドルウェアは上記の問題を解決するため、ベアラ選択やアクセス制御といった、マルチベアラ端末上のアプリケーションが必要とする機能を一元的に提供する。

2.2. 機能

提案ミドルウェアは以下の３つの機能を提供する。

- ベアラ選択機能
- ベアラアクセス制御機能
- 上記機能の拡張性

以下では、それぞれの機能について説明する。

2.2.1. ベアラ選択機能

ベアラ選択機能は、アプリケーション単位またはアプリケーションが行う通信単位で、使用するベアラを自動的に選択する機能である。この機能により、「キャリアドメインの Web サイトへの接続はセルラー網を利用したい」とか、「ビデオや音声といった広帯域を必要とするデータは無線 LAN を利用したい」といった要求に対し、システム全体の通信に影響を与えることなく、適切なベアラを自動的に割り当てることができる。

2.2.2. ベアラアクセス制御機能

ベアラアクセス制御機能は、アプリケーションが使用したベアラやアクセスしたファイルに応じて、アプリケーションからの特定のベアラ利用を制限する機能である。この機能により、「社内 LAN にアクセスしたアプリケーションは他のネットワークへのアクセスを禁止させたい」といった要求に対し、適切なアクセス制御を行うことができる。

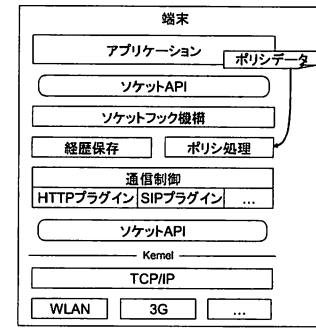

図 1 アーキテクチャ

2.2.3. 機能拡張

ベアラ選択機能とベアラアクセス制御機能は、拡張性を持つ。提案ミドルウェアが動作するシステムの要求内容により、提案ミドルウェアの基本機能では対応できない場合でも、機能拡張により適切なベアラ選択、アクセス制御を可能とする。

2.3. アーキテクチャ

前述の機能を提供するミドルウェアを設計するにあたって考慮したアーキテクチャ上の課題と、実際に設計したアーキテクチャを紹介する。

2.3.1. 課題

先に述べたとおり、提案ミドルウェアは(1)通信に利用するベアラの容易な選択を、(2)開発や移植に際したアプリケーション実装への負担を最小限にして提供することを目的とする。

(1)の目的においては、アプリケーションが明示的にベアラを選択することなく、適切なベアラで通信が行われることが理想的である。全ての通信に対して明示的なベアラ選択を行うことは煩雑であり、この手間を可能な限り低減できることが望ましい。

(2)の目的においては、広く利用されている既存の通信 API を利用できることが理想的である。独自 API による機能提供は移植性を低くする上、既存アプリケーションへの適用を困難にする。また、アプリケーションによって通信ミドルウェアによる機能共通化の程度が異なることも問題となる。例えば、同じ HTTP による通信を行うアプリケーションでも、あるアプリケー

ションは HTTP の全ての処理を自前で用意しており、別のアプリケーションは HTTP のメッセージ生成から送受信処理まで全てをミドルウェアに任せている場合がある。このような状況下でも、全てのアプリケーションに対して機能を提供できることが望ましい。

2.3.2. アーキテクチャの概要

前述の課題を考慮して設計された、提案ミドルウェアのアーキテクチャを図 1に示す。このアーキテクチャは以下の特徴を有する。

- ソケット API による機能提供
- 通信ポリシを用いた通信制御
- プラグイン機構

以下では、これらの特徴について説明する。

2.3.2.1. ソケット API による機能提供

提案ミドルウェアの機能は、UNIX 系 OS をはじめとした多くのプラットフォームで広く使われており、POSIX によって規格が定められている、ソケット API によって提供される。

実際には提案ミドルウェアは、アプリケーションによるソケット API の呼び出しをフックし、ベアラ選択やアクセス制御などの固有の処理を行う。この仕組みはソケットフック機構によって提供される。

ソケットフック機構は、アプリケーションからのソケット API 呼び出しをフックして独自の処理を加えるための仕組みである。特定のソケット API と特定のソケットデスクリプタを指定してコールバック関数を登録しておくことで、該当ソケットデスクリプタに対し該当 API が呼び出されたときにコールバックを受けることができる。

ソケット API をフックするためには、Linux のプリロードの機能(環境変数 LD_PRELOAD を設定してライブラリをアプリケーション実行前にロードさせる)を利用するか、アプリケーションを再メイクして提案ミドルウェアのライブラリをリンクさせる必要がある。

2.3.2.2. 通信ポリシを用いた通信制御

提案ミドルウェアは通信ポリシに従ってベアラ選択やアクセス制御を行う。通信ポリシは、アプリケーションが行う通信を制御するルールを記述した外部データであり、1 アプリケーションに対して 1 つ用意する。通信ポリシを変更することにより、アプリケーション自体には変更を加えることなく、アプリケーションが利用するベアラを制御/変更することができる。

図 2に、提案ミドルウェアで実際に使用した XML 形式の通信ポリシの例を示す。この例では、(A)が「音声メディアは無線 LAN 固定」というベアラ選択上の要求、(B)が「無線 LAN でデータを受信した場合、無線 LAN 以外のベアラを利用禁止」というアクセス制御上の要求を表している。

```
<policies service="VoIP" protocol="SIP">
 <policy>
  <cond>
   <case item="MIME" type="equal">audio</case>   …(A)
  </cond>
  <use>required bearer = wlan</use>
 </policy>
 <policy>
  <cond>
   <default />
  </cond>
  <use>optional bearer = cellular wlan</use>
 </policy>
 <restriction>
  <cond>
   <case item="bearer" type="equal">wlan</case>   …(B)
   <case item="use" type="equal">received</case>
  </cond>
  <use>required bearer = wlan</use>
 </restriction>
</policies>
```

図 2　通信ポリシ例

2.3.2.3. プラグイン機構

ベアラ選択に用いる情報やアクセス制御の粒度などには様々な要求が考えられるため、提案ミドルウェアは基本機能として最小限の機能のみを提供し、プラグインによる拡張機構を用意した。

提案ミドルウェアは基本機能として、ポート番号や IP アドレス、無線電界強度などの情報を用いたベアラ選択機能と、ベアラ単位でのアクセス制限を行うアクセス制御機能のみを提供する。

プラグインによる拡張を行うことで、基本機能より柔軟な通信制御が可能となる。例えば、SIP のセッション情報を参照してメディアセッション単位でベアラ選択を行うようなプラグインや、HTTP のアクセス先ドメイン単位で通信を制限するようなプラグインを作成することが可能である。

プラグインによる拡張もまた、ソケットフック機構を利用する。プラグインはコールバック関数を用意することでソケット API の処理をフックし、固有の処理を追加する。

本研究では、実際に HTTP と SIP の通信内容に応じてベアラ選択を行うプラグインを作成した。それぞれのプラグインの機能についてもここで説明する。

2.3.2.4. HTTP プラグイン

HTTP プラグインは、HTTP のリクエスト URL、リクエストメソッド、取得ファイルの MIME タイプを用いてベアラ選択を行う機能を追加するものである。

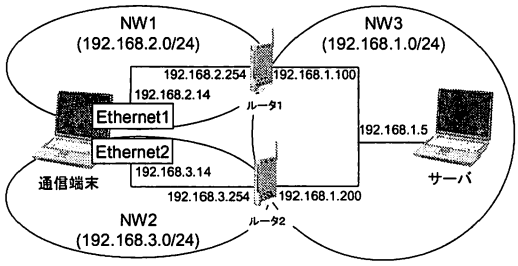

図 3 ネットワーク構成

HTTP プラグインは上記情報に従って、アプリケーションが送信するリクエストパケットを適切なベアラで送信する。

2.3.2.5. SIP プラグイン

SIP プラグインは、SIP リクエストの送信ベアラを選択する機能と、メディアセッションを MIME タイプに応じてベアラを選択する機能を追加するものである。

SIP リクエストには自分の IP アドレスを記述するヘッダが含まれるため、単純にパケットをルーティングするだけではなく、利用するベアラによってこのヘッダを正しく書き換える機能を追加する。更に、メディアセッションのベアラ選択に関しては、セッション開始時にアプリケーションが送受信する SDP の内容を書き換えることで、メディアセッションの送受信に利用するベアラを設定する。

3. 評価

本章では、試作した提案ミドルウェアの評価を行う。

まず、提案ミドルウェアが既存のアプリケーションに対して容易に適用可能であることを確認するため、Opera ブラウザへの適用を行った。前述のプリロードを用いて提案ミドルウェアを適用することで、Opera 自体には変更を加えることなくベアラ選択機能とアクセス制御機能を利用できることが確認できた。

一方、提案ミドルウェアはアプリケーションのソケット API をフックして通信制御を行うため、通信処理のオーバーヘッドが発生する。特にプラグインによる機能拡張を行った場合は、通信パケットの解析や操作を行うため、通信性能の劣化が懸念される。そこで、以降では試作した提案ミドルウェアの性能を評価する。

性能評価では、HTTP、SIP のベアラ選択プラグインを使用し、それぞれのプロトコルを使うアプリケーションに対して提案ミドルウェアが与えるオーバーヘッドを測定する。

3.1. 評価環境

図 3に性能評価に用いたネットワークの構成を、表 1に性能評価に用いた機器のスペックを示す。

通信端末	
CPU	Pentium M 1.3GHz
メモリ	DDR SDRAM 256MB
OS	RedHat Linux 9 (Linux 2.4.20-20.9)

サーバ(HTTP/SIP 共用)	
CPU	Mobile Pentium III 500MHz
メモリ	SDRAM 64MB
OS	RedHat Linux 9 (Linux 2.4.20-20.9)

表 1 評価機器

大項目	小項目
HTTP トランザクション	1 トランザクションの処理時間
	アイテムの多いページの取得時間
	アイテムの少ないページの取得時間
SIP トランザクション	REGISTER の処理時間
	INVITE の処理時間
	BYE の処理時間
メディアセッション	RTP の送受信時間
	MSRP トランザクションの処理時間

表 2 評価項目

通信端末は複数の通信インタフェースを持つ PC であり、マルチベアラ端末に対応する。この通信端末に提案ミドルウェアを適用して性能測定を行った。

NW1、NW2 はそれぞれ通信端末と 1 台のエッジルータからなるネットワークであり、ホームネットワークや社内ネットワークのようなローカルネットワークや、プロバイダのアクセスネットワークに対応する。

NW3 はサーバとルータ 1、ルータ 2 からなるネットワークであり、インターネットのような広域ネットワークに対応する。

ルータでは NAT 機能を利用せず、ルーティングは全て静的に設定した。評価環境はクローズドなネットワークであり、図に示す以外のネットワーク機器は接続されていない。

3.2. 評価項目

表 2に、測定を行った項目の一覧を示す。表のそれぞれの項目について、提案ミドルウェアのある場合とない場合、さらに提案ミドルウェアのある場合についてはデフォルト設定のベアラを利用する場合と利用しない場合について計測した。

ここで、デフォルト設定のベアラを利用する場合とは、常に最初に設定されたベアラが使われる場合であり、提案ミドルウェア利用時のベストケースに相当する。逆にデフォルト設定のベアラを利用しない場合とは、全ての通信において提案ミドルウェアでベアラ選択の処理が発生する場合であり、提案ミドルウェア利用時のワーストケースに相当する。

以下では、それぞれの評価項目について説明する。

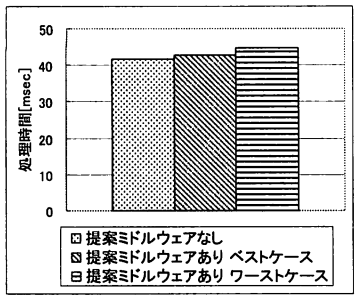

図 4 HTTP トランザクション処理時間

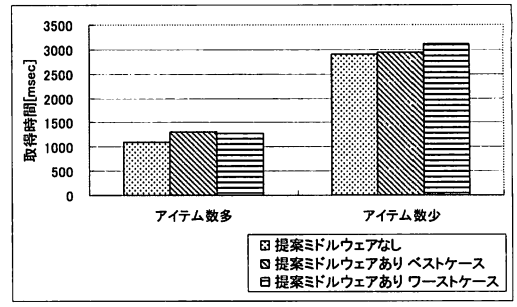

図 5 Web ページ取得時間

3.2.1. HTTP トランザクション

HTTP トランザクションの項目では、1 回の HTTP GET トランザクションにかかる処理時間の計測と、一般的なページの取得時間の計測を行った。HTTP トランザクションを行うクライアントアプリケーションとしては Opera 9.01 を、サーバアプリケーションとしては Apache 2.0.40 を用いた。

1 回のトランザクションにかかる処理時間については、1 バイトのファイルに対する GET リクエストの送信開始からレスポンスの受信完了までにかかる時間を計測し、受信の待受時間を差し引いたものを用いた。

一般的なページの取得時間については、最初の GET リクエストの送信開始から最後のレスポンスの受信完了までにかかる時間を計測した。この計測では、トランザクション回数による変化を見るため、同じデータサイズで構成アイテム数の異なる 2 種類の Web ページについて同様の計測を行った。一方は 1 つの HTML ファイルと 28 個の埋め込みアイテムからなる Web ページ、もう一方は 1 つの HTML のみの Web ページであり、データサイズは両方とも 191.2kB である。

3.2.2. SIP トランザクション

SIP トランザクションの項目では、REGISTER、INVITE、BYE それぞれのトランザクションにかかる処理時間の計測を行った。SIP トランザクションを行うアプリケーションには、RTP セッションと MSRP セッションを利用できる自作のマルチメディア通信アプリケーションを利用した。

REGISTER、BYE トランザクションに関する処理時間については、HTTP トランザクションと同様にリクエストの送信開始からレスポンスの受信完了までの時間から待受時間を差し引いたものを用いた。

INVITE トランザクションにかかる処理時間の計測については、リクエストの送信開始から ACK の送信完了までの時間を計測し、レスポンスの待受時間を差し引いたものを用いた。

3.2.3. メディアセッション

メディアセッションの項目では、RTP、MSRP の各プロトコルの送受信にかかる処理時間を計測した。アプリケーションには、前述のマルチメディア通信アプリケーションを利用した。

RTP については、該当ソケットでの send および recv のそれぞれにかかる時間を計測した。

MSRP はトランザクション形式のプロトコルであるため、リクエスト送信開始からレスポンス受信完了までの時間から待受時間を差し引いた時間を測定した。

3.3. 評価結果

図 4～図 7に評価結果の一覧を示す。

3.4. 考察

3.4.1. HTTP トランザクション

HTTP トランザクションのベアラ選択においては、提案ミドルウェアによるオーバーヘッドは主に HTTP リクエストヘッダのパース処理と、新規にコネクションを確立する処理である。これらの処理はほぼ定数オーダーなので、受信するデータサイズが大きくなるほど提案ミドルウェアによるオーバーヘッドは相対的に小さくなる。

図 5によると、処理時間のオーバーヘッドは最大 200 ミリ秒程度であり、やや大きな影響が見られる。この値は図 4の結果から読み取れるトランザクションあたりのオーバーヘッド(約 3 ミリ秒)から推定される数値に比べると非常に大きい。これは、ソケットフック機構による recv 関数に対するオーバーヘッドが影響していると考えられる。ソケットフック機構によるオーバーヘッドは、単体では軽微だが大きなデータをやりとりする場合には無視できない影響が出ることがわかる。

3.4.2. SIP トランザクション

SIP トランザクションのベアラ選択においては、インタフェース変更時にリクエストおよびレスポンスデ

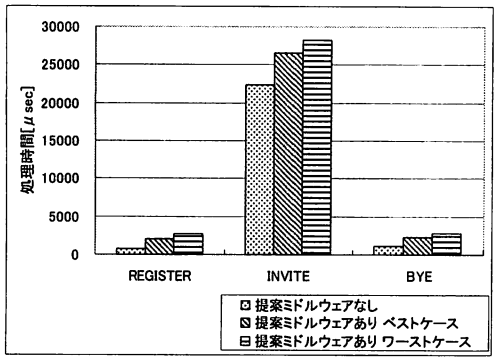

図 6 SIP トランザクション処理時間

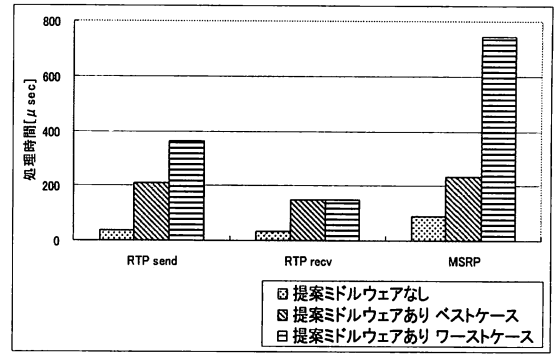

図 7 メディアセッション処理時間

ータの書き換えが発生するため、この処理がオーバーヘッドの最大の要因となる。

図 6によると、書き換え処理の多い INVITE トランザクションにおいてでも最大 6 ミリ秒程度のオーバーヘッドであり、実用上は問題ないと考えられる。特に SIP のトランザクションは 1 クライアントから頻繁に発生するものではない上、ミリ秒オーダーが重要になるケースはほとんど無い。さらに、サーバの処理時間やネットワーク時間による影響もあるため、ミリ秒単位のオーバーヘッドはほぼ無視できる数値である。

3.4.3. メディアセッション

SIP で制御されるメディアのベアラ選択においては、RTP の場合はメディアの判定、MSRP の場合はヘッダの書き換えがオーバーヘッドの最大の原因となる。

図 7によると、特に MSRP はオーバーヘッドが顕著である。リクエストが連続して送信された場合には無視できない影響が予想されるが、テキストデータという特性上、リアルタイムレベルの性能が要求されることは少ないため、1 ミリ秒以下の遅延ならば影響は軽微と考えられる。

一方、RTP ではそれぞれのオーバーヘッドは決して大きくないが、音声データやビデオデータなどのリアルタイムデータをバースト的に送信するため、前述のソケットフック機構によるオーバーヘッドによる性能劣化が懸念される。

4. 関連研究

ソケット以下のレイヤでベアラを切り替える技術としては、Mobile IP[2][3]がある。しかしながら、Mobile IP は 1 つの通信端末に 1 つの固定的な IP アドレスを付与する技術であり、同時に複数のベアラを利用するケースや、アプリケーションごとに使用ベアラを変更するような要求には対応していない。また、Mobile IP の導入には Home(/Foreign) Agent のインフラ整備が不可欠であり、まだ実用化の見通しは立っていない。

一方で、端末ミドルウェアを用いて複数ベアラの利用制御を行う方式が提案されている[1]。この方式では、ミドルウェアが通信先ホストと独自のプロトコルで通信を行うことで、使用するベアラを決定する。このため、通信を行うエンドホストの両方でミドルウェアの導入を必要とする点が本研究のミドルウェアとの最大の違いである。[1]の方式は無線端末間の P2P 的な通信を想定しているのに対し、本研究の方式は無線端末が通信する相手を特に限定していないためのアプローチの違いと考えられる。

5. まとめ

マルチベアラ端末における通信を透過的に制御するミドルウェアを提案し、試作を行った。提案したミドルウェアは、ソケット通信を行う一般的なアプリケーションに対し変更を加えることなく適用可能であり、通信ポリシを用意するだけでベアラ選択とアクセス制御を自動的に行えることを確認した。

また、HTTP および SIP 固有の情報を用いてベアラ選択処理を行うプラグインを試作した。それぞれのプロトコル処理のオーバーヘッドを計測し、多くの場合において性能的に問題が無いことを確認した。

今後の課題としては、フック機構の高速化によるパフォーマンスの向上と、通信ポリシを簡単にすることで、用意に設定が行えるようにすることが挙げられる。また、より実用ケースに近い環境でのテストのために、異種無線環境や 3 つ以上のベアラが利用可能な環境における性能評価を行う。

文 献

[1] 元濱 努, 瀧本 英二, 鈴来 和久, 毛利 公一, 大久保 英嗣, “異種ネットワーク環境における適応的通信デバイス制御,” 情報処理学会研究報告, 2006-MBL-36, pp.281-286, Feb. 2006.

[2] “IP Mobility Support for IPv4,” IETF Request for Comments, http://www.ietf.org/rfc/rfc3344.txt .

[3] “IP Mobility Support in IPv6,” IETF Request for Comments, http://www.ietf.org/rfc/rfc3775.txt .