

セッションレイヤにおけるエンドツーエンド型モビリティサポートの実装と評価

金子 晋丈 河内 佑介 森川 博之 青山 友紀 中山 雅哉

東京大学工学系研究科 〒113-8656 東京都文京区本郷 7-3-1

E-mail: {kaneko, nakayama}@cnl.k.u-tokyo.ac.jp, {kochi, mori, aoyama}@mlab.t.u-tokyo.ac.jp

あらまし ネットワーク接続性, コンテンツ, コンピューティングリソースがあらゆる場所に存在する 3C everywhere 環境では, ターミナルモビリティのみならず, デバイスやネットワーク, コンテンツを自由に切り替えるサービスモビリティの実現が望まれる. 筆者らは, ターミナルモビリティとサービスモビリティを考慮したモビリティサポート機構としてエンドツーエンド型モビリティサポート手法を用いる. エンドツーエンド型モビリティサポート手法は, 通信端末間で移動に関するメッセージのやり取りを行い, 通信の維持に関するトラッキング機構をエンドホストで実現するものである. 本稿では, エンドツーエンド型モビリティサポート手法の基本的な機能である, 通信端末の IP アドレスの変更に対応するための実装を行い, その切り替え遅延に関して評価する.

キーワード 3C everywhere, モビリティサポート, サービスモビリティ, エンドツーエンド, セッション層

Implementation and Evaluation of End-to-End Mobility Support at the Session Layer

Kunitake KANEKO Yusuke KOCHI Hiroyuki MORIKAWA
Tomonori AOYAMA and Masaya NAKAYAMA

School of Engineering, The University of Tokyo 7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656 Japan

E-mail: {kaneko, nakayama}@cnl.k.u-tokyo.ac.jp, {kochi, mori, aoyama}@mlab.t.u-tokyo.ac.jp

Abstract In the environment of 3C everywhere in which network Connectivity, Content, and Computing resources are ubiquitous, there can be a new way of using these resources, bringing flexibility that allows users to take most advantage of the resource by choosing (or combining) appropriate ones according to the on-the-spot needs. To realize flexible service, a service mobility support is desired. We implemented end-to-end mobility support which we presented for service mobility in 3C everywhere environments and evaluate its performance from the viewpoint of service disruption time.

Keyword 3C everywhere, mobility support, service mobility, end-to-end, session layer

1. はじめに

筆者らは, インターネットの将来像として, ネットワーク接続性, コンテンツ, コンピューティングリソースがあらゆる場所に存在する 3C everywhere 環境を想定している. 3C everywhere 環境では, ターミナルモビリティのみならず, デバイスやネットワーク, コンテンツを自由に切り替えるサービスモビリティの実現が望まれる. 筆者らは, ターミナルモビリティとサービスモビリティを考慮したモビリティサポート機構のデザインにあたって, アプリケーションの下位レイヤ依存, 端末に依存した認証形態, レイヤ構造の問題, 名前空間と通信空間のバインディングの 4 つ問題点について検討を行い, 通信の維持を困難に

している通信インタフェースの識別子について, デザインを検討し, そのデザインからエンドツーエンド型モビリティサポート手法を導いた [1]. エンドツーエンド型モビリティサポート手法は, 通信端末間で移動に関するメッセージのやり取りを行い, 通信の維持に関するトラッキング機構をエンドホストで実現するものである.

本稿では, エンドツーエンド型モビリティサポート手法の基本的な機能である, 通信端末の IP アドレスの変更に対応するための実装を行い, その切り替え遅延に関して評価した. 以下, 2. でエンドツーエンド型モビリティサポート手法について説明し, その実装設計を行う. 3. で実装システムを示し, 4. で動作確認と評価を行う. 最後に 5. で本稿をまとめる.

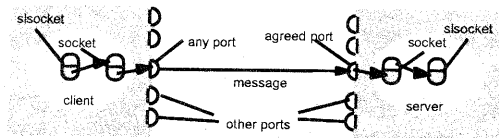


図 1. エンドツーエンド型モビリティサポート

2. エンドツーエンド型モビリティサポート手法

本章では、エンドツーエンド型モビリティサポート手法について説明する。通信端末やアクセスリンクが多様化したインターネット環境におけるアプリケーションは、通信リンクや通信端末を自在に変更することのできる柔軟なサービスになるであろうと考えられる。このようなアプリケーションサービスでは、移動する通信相手の特定が適切に行われ、よりよい通信端末やアクセスリンクが発見されたときにはすみやかに切り替えられるが、通信は維持されることが望まれる。しかしながら、現在のインターネットでは、ロケーティングの問題や通信識別子の問題、認証の問題、レイヤ間インタラクションの問題から、多様化するインターネット環境における柔軟なアプリケーションサービスを実現することができない。エンドツーエンド型モビリティサポート手法は、とくに通信識別子の問題を解決するための、通信の維持に関するトラッキング機構をエンドホストで実現する手法である。

エンドツーエンド型モビリティサポート手法では、アプリケーションの通信インタフェースの識別子を単なる ID とし、ネットワークアドレスに依存するネットワーク通信インタフェースとアプリケーションの通信インタフェースを通信端末内部で動的に接続することで、ネットワークアドレスに依存しない通信の維持に関する機構を実現している。また、このような動的な切断、再接続をアプリケーションサービスの中断なくすみやかに行うために、通信端末間で通信維持に関するメッセージングを行っている。例えばターミナルモビリティ時には、新しく利用する IP アドレスがこのメッセージに記される。しかしながら、双方の通信端末が同時に移動したとき、もしくは通信リンクの不意の切断時など、コントロールメッセージを通知することができないときがある。このような場合は、ロケーティングを行うことによって、ネットワークアドレスの解決を行い、通信を維持する。

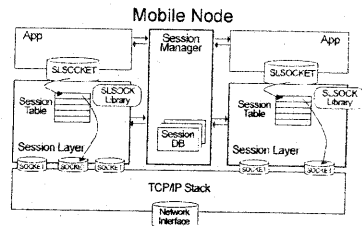


図 2. 実装システムの全体像

次章では、エンドツーエンド型モビリティサポート手法の、アプリケーションの通信インタフェースとネットワーク通信インタフェースの動的なマッピング、およびすみやかな切断、再接続のためのメッセージングシステムの実装について述べる。

3. 実装システム

2. で示したエンドツーエンド型モビリティサポート手法のコンセプトに基づき実装を行った。まず、3.1 で実装する上での設計指針を示し、3.2 で実装システムの全体像を提示する。3.3 で本実装を用いたアプリケーションプログラミングの方法について述べる。実装の詳細は、3.4 以降で示す。

3.1 実装設計

エンドツーエンド型モビリティサポート手法の実装にあたって以下の 4 点を設計指針とする。

- (1) アプリケーションプログラマの利便性を考え、既存の socket API[2] と類似性のある関数群を提供する。
- (2) アプリケーションによって通信の信頼性に関する要求は異なるため、信頼性の有無はアプリケーションが選択する。
- (3) ネットワークインタフェースに依存しないネットワーク状態の検出を行うため、外部モジュールがこれを行う。
- (4) アプリケーションにより利用するロケーティングシステムが異なるため、ロケーションは、外部モジュールが解決する。

これらの設計指針に基づき、エンドツーエンド型モビリティサポートの実装を行った。次節では、その実装システムの全体像を述べる。

3.2 実装システムの全体像

実装システムは、大きく 2 つの機能に分類できる。一つは、動的にアプリケーションの通信インタフェ

<code>s = sl_socket(AF_INET, SOCK_STREAM, 0)</code>	<code>s = sl_socket(AF_INET, SOCK_STREAM, 0)</code>
<code>sl_bind(s, server_addr)</code>	<code>sl_bind(s, client_addr)</code>
<code>sl_listen(s, 5)</code>	<code>sl_connect(s, server_addr)</code>
<code>sl_accept(s, client_addr)</code>	<code>sl_send(s, buf, buf_size)</code>
<code>sl_recv(s, buf, buf_size)</code>	
サーバプロセス	クライアントプロセス

図 3. TCP 型アプリケーションにおける `slsocket` の使い方

<code>s = sl_socket(AF_INET, SOCK_DGRAM, 0)</code>	<code>s = sl_socket(AF_INET, SOCK_DGRAM, 0)</code>
<code>sl_bind(s, server_addr)</code>	<code>sl_bind(s, client_addr)</code>
<code>sl_recvfrom(s, buf, client_addr)</code>	<code>sl_sendto(s, buf, server_addr)</code>
サーバプロセス	クライアントプロセス

図 4. UDP アプリケーションにおける `slsocket` の使い方

ースとネットワーク通信インタフェースを接続する機能であり、もう一つは、他の通信端末と移動に関するコントロールメッセージのやり取りを行う機能である（図 2）。

前者に関して、アプリケーションから見たひとつながりの通信サービスをセッションと定義する。セッションごとに、アプリケーションの通信インタフェースとネットワーク通信インタフェースのマッピング情報や通信相手のロケーションを保持するセッションテーブルが存在する。本稿では、アプリケーションの通信インタフェースを `slsocket` と呼び、ネットワーク通信インタフェースを `socket` と呼ぶ。また、セッションテーブルを含め、ダイナミックな `slsocket` と `socket` の切断、接続を行うレイヤをセッションレイヤと呼ぶ。

後者に関して、他の通信端末と移動に関するコントロールメッセージのやり取りを行い、セッションレイヤに切断、再接続命令を出す機能をセッションマネージャが持っている。セッションマネージャは端末に一つだけ存在し、セッションデータベースを所持している。セッションデータベースは、他の通信端末から届いたコントロールメッセージをセッション毎に振り分けるためのものである。

本稿では、このような機能を提供するライブラリを総称して、`slsock` ライブラリと呼ぶ。

表 1. トランスポートプロトコル

型	トランスポート プロトコル	接続切り替え時 の信頼性
SOCK_STREAM	TCP	○
SOCK_STREAMGRAM	TCP	×
SOCK_DGRAM	UDP	×

3.3 アプリケーションプログラミング

`slsocket` ライブラリを用いたアプリケーションプログラミングは、既存の `socket` ライブラリを用いたプログラミングと類似している。異なるのは、通信相手の指定方法とトランスポートプロトコルの選択である。

まず、通信相手はロケーションで指定される。ここで指定されたロケーションは、通信相手を決定する初期値として用いられる。

アプリケーションプログラマが選択できるトランスポートプロトコルは、3つある（表 1）。TCP[3] による通信を行う `SOCK_STREAM` と `SOCK_STREAMGRAM`、および UDP[4] による通信を行う `SOCK_DGRAM` である。`SOCK_STREAM` は、端末間の通信に TCP を用い信頼性のある通信を実現するとともに、移動に伴うコネクションの切断、再接続時にもアプリケーションデータをバッファリングし、信頼性を保つ。`SOCK_STREAMGRAM` は端末間の通信に TCP を用いるが、コネクションの切断、再接続時に信頼性が保たれない。`SOCK_DGRAM` は端末間の通信に UDP を用い、コネクションの切断、再接続時にも信頼性が保たれない。`SOCK_STREAM` と `SOCK_STREAMGRAM` のアプリケーションプログラミングに違いはない。図 3 に TCP 型アプリケーションにおける `slsocket` ライブラリの使い方を示し、図 4 に UDP アプリケーションにおける `slsocket` ライブラリの使い方を示す。なお、本実装では `SOCK_STREAM` の実装は行っていない。

3.4 `slsocket` の作成と終了

本節では、アプリケーションから `sl_socket` や `sl_closesocket` が呼び出されたときのセッションレイヤにおける処理について述べる。

`sl_socket` が呼び出されると、セッションレイヤでは `socket` を呼び出し `socket` を作成する。`socket` が正しく作成されると、セッションテーブルを作成し、`sl_socket` は終了する。

セッションテーブルは、slsocket descriptor, socket descriptor, 端末のロケーション, 端末の IP アドレスとポート番号, トランスポートプロトコル, サーバプロセスかクライアントプロセス, socket 状態フラグの 7 つの要素で構成されている。slsocket descriptor は `sl_socket` の返り値であり, socket descriptor は `socket` の返り値である。socket descriptor は, ロケーションや IP アドレス, ポート番号の変更によって更新される。ロケーションの初期値は, アプリケーションが指定したロケーションとなる。この値は, ロケーションの変更によって更新される。IP アドレスとポート番号の初期値は, ロケーション解決によって得られた IP アドレスとポート番号となる。この値は, ロケーションや IP アドレス, ポート番号の変更によって更新される。トランスポートプロトコルは, 3.3 で述べた `SOCK_STREAM`, `SOCK_STREAMGRAM`, `SOCK_DGRAM` のいずれかがアプリケーションの指定により与えられる。サーバプロセスかクライアントプロセスの記述については, 4.5 で示す。socket 状態フラグは slsocket の自動クローズに用いる。このフラグの利用については後述する。セッションテーブルは, 4.6 で述べるセッションマネージャからの更新要求メッセージによって更新される。

作成された slsocket は `sl_socket` を呼び出したアプリケーションによって管理される。したがって, アプリケーションは自由なタイミングで slsocket をクローズすることができる。`sl_closesocket` を呼び出すと slsocket がクローズされ該当する slsocket に関するセッションテーブルが消去され, セッションが終了する。

また, ネットワークからの切断, あるいは電源消失などの突然の障害が生じた場合, `sl_closesocket` を呼び出せないため, 利用されていない slsocket を自動的にクローズする機構が必要である。そこで, slsocket にマッピングされた socket がクローズされた後, タイムアウト時間が経過した場合, セッションレイヤが自動的に slsocket を終了させる。タイムアウト時間までに新たな socket が作成された場合は, この限りではない。socket 状態フラグは slsocket と socket のマッピング状態の判断に用いられる。

次節では, `sl_socket` によって作成された slsocket と socket のダイナミックな切断, 再接続機構について述べる。

3.5 セッションの維持

slsocket と socket のダイナミックな切断, 再接続機構について述べる。切断, 再接続機構は, アプリケーションの指定するトランスポートプロトコルによって異なる。ここでは, 実装を行った `SOCK_STREAMGRAM` 型と `SOCK_DGRAM` 型の機構を示す。まず, `SOCK_STREAMGRAM` 型について述べ, 次に, `SOCK_DGRAM` 型について述べる。

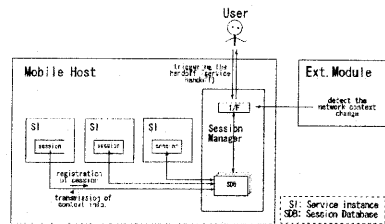


図 5. セッションマネージャ

以下に, `SOCK_STREAMGRAM` 型のセッション維持手法を示す。socket ライブラリにおける `SOCK_STREAM` 型通信では, コネクション確立におけるサーバクライアントモデルが使われている。すなわち, 端末やロケーション, IP アドレスを変更する側がコネクション確立時のサーバ側かクライアント側かにより, 再接続方法が異なる。以下, サーバ側が変更する場合とクライアント側が変更する場合にわけて説明する。ここでは, 実装を行った, ロケーションと IP アドレス変更に関する機構を示す。

まず, サーバ側がロケーションや IP アドレスを変更する場合, サーバ側では, 新しい IP アドレスに基づいて socket を作成し, `listen` 状態となる。変更要求を受けたクライアント側は, 新たな socket を作成し `connect` を用いて再接続する。再接続に成功すると, セッションテーブルの該当する socket descriptor の値を書き換える。サーバ側では, 接続要求を受けつけ (`accept`) ると, 新たな socket が生成される。これを slsocket と接続し, セッションテーブルの該当する socket descriptor の値を書き換える。

次に, クライアント側がロケーションや IP アドレスを変更する場合, クライアント側は新たな socket を作成し `connect` を用いて再接続する。再接続に成功すると, セッションテーブルの該当する socket descriptor の値を書き換える。一方, サーバ側では, 接続要求を受けつけ (`accept`) ると, 新たな socket が生成される。これを slsocket と接続し, セッションテーブルの該当する socket descriptor の値を書き換える。

`SOCK_DGRAM` 型の通信では, クライアントサーバ間のコネクション確立がなく, 直接データの送受信を行う。データ送信のために `sl_sendto` が呼び出されると, セッションテーブルに基づいて送信先 IP アドレスを決定し, `sendto` を呼び出す。

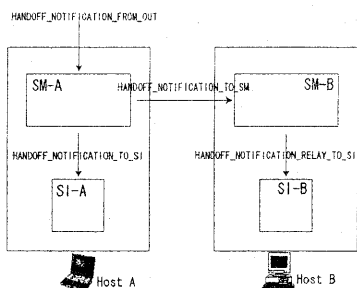


図 6. コントロールメッセージ

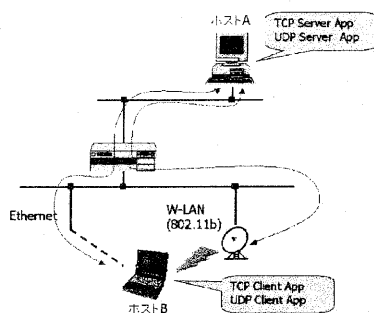


図 7. 動作確認

3.6 セッションマネージャ

本節では、他の通信端末と移動に関するコントロールメッセージをやり取りを行い、セッションレイヤに切断、再接続命令を出す機能をもつセッションマネージャ（図 5）について説明する。セッションマネージャの機能は、大きく 3 つに分類される。自ホストや通信相手の端末から受信した移動に関するコントロールメッセージの受信、コントロールメッセージの受信後の適切なセッションレイヤへのメッセージ通知、および適切なセッションレイヤに通知するためのセッション管理である。

まず、セッション管理について述べる。セッションは、セッションデータベースによって管理されている。セッションデータベースは、ロケーション、ポート番号、ssocket descriptor、アプリケーションのプロセス番号の 4 要素からなっている。ロケーションおよびポート番号によって、セッションを定義する。それぞれのセッションに対して、ssocket descriptor とアプリケーションのプロセス番号の参照を行う。ssocket ライブラリはアプリケーションにインクルードする形で用いられるため、ssocket descriptor は、アプリケーション内では一意性が保たれるが、端末内では一意性が保たれない。そのため、アプリケーションのプロセス番号によって端末内での一意性を保証する。

セッションデータベースへのセッション情報の追加は、アプリケーションが ssocket を作成し、セッションを開始するときに行われる。セッションデータベースのセッション情報は、ssocket がクローズされセッションが終了したとき、およびアプリケーションプロセスが終了したときに削除される。セッションデータベースの参照は、コントロールメッセージの受信時、および送信時に行われる。

次に、コントロールメッセージの受信に関して述べる。コントロールメッセージは、自ホスト、または通信相手端末の IP アドレスが変化したときに受信する。自ホストの

IP アドレスが変化すると、外部検出機構がこれを検知しセッションマネージャにこれを通知する。また、通信相手端末の IP アドレスが変化すると、通信相手端末上のセッションマネージャが、この情報を通知する。

コントロールメッセージの送信は、コントロールメッセージの受信を受けて、セッションテーブルを参照し、該当するセッションレイヤ、もしくは通信相手端末のセッションマネージャに通知される。

これらのメッセージは、共通したメッセージングプロトコルに従って送受信される。メッセージコマンドをいくつか示す（図 6）。

HANDOFF_NOTIFICATION_FROM_OUT コマンドは、外部検出機構がセッションマネージャに送信する。このメッセージには、ホスト A のロケーションとポート番号、新しい IP アドレスが含まれる。HANDOFF_NOTIFICATION_TO_SI コマンドは、A のセッションマネージャがローカルホストのセッションレイヤに対して送信する。HANDOFF_NOTIFICATION_TO_SM コマンドは、A のセッションマネージャが B のセッションマネージャに送信する。HANDOFF_NOTIFICATION_RELAY_TO_SI コマンドは B のセッションマネージャが B のセッションレイヤに対して送信する。

4. 実装システムの動作確認と評価

本章では、エンドツーエンド型モビリティサポートの実装システムの動作確認を行い、その基本性能として ssocket と socket の切断、再接続によるコネクション切り替えに要する時間の測定を行った。

4.1 動作確認

本実装システムの動作確認を行うため、図 7 に示すテストベッドネットワークを構築した。このテストベッドネットワークにおいて、固定的な IP アドレスを持つホスト A と有線ネットワークインタフェースおよび PCMCIA カ

ードスロットを持つホストBを用意した。本システムをホストAとホストBに実装し、動作を確認した。ホストBには、無線LANのインタフェースを検出し自動的にIPアドレスを設定した後、セッションマネージャにIPアドレスに変更があったことを通知するIPアドレス自動設定機構と外部検出機構が実装されている。

動作確認は以下のように行った。ホストBで有線ネットワーク側のIPアドレスを有したまま、無線LANカードをPCMCIAカードスロットに挿入しネットワーク接続を変更した際に、SOCK_STREAMGRAM型のアプリケーションとSOCK_DGRAM型のアプリケーションが通信を維持することができるかを確認した。

次節では、このときのコネクション切り替えに要する時間について述べる。

4.2 コネクション切り替えに要する時間

エンドツーエンド型モビリティサポートの性能評価として、コネクション切り替え時間をテストベッド上で測定した。コネクション切り替え時間とは、外部検出機構がセッションマネージャにHANDOFF_NOTIFICATION_FROM_OUTコマンドを送信してから、アプリケーションが通信可能状態になるまでの時間である。

測定の結果、SOCK_STREAMGRAM型アプリケーションでは1.05 msec、SOCK_DGRAM型アプリケーションでは0.71 msecであった。この測定結果は、30回行った測定の最大値、最小値を除いた算術平均である。

以下では、ユーザが無線LANカードの挿入した後、コネクションの切り替え要求を外部検出機構が発してから、コネクション切り替えが完了して通信可能な状態になるまでの時間について実用性の面から検討する。エンドツーエンド型モビリティサポートでは、通信端末間で切り替えのメッセージが送信されるため、切り替えに要する時間はRTTに依存する。本テストベッドネットワークにおけるホストAとホストB間のRTTは0.62msecであった。なお、このRTTは有線ネットワーク経由の時間である。RTTに依存しない`slsocket`と`socket`の切断、再接続に要する時間を別途測定したところ0.38msecであった。インターネットにおける通信端末間の距離は、本テストベッドネットワークにおけるものよりも長いと考えられるため、本実装では切断、再接続がRTTよりも十分に短い時間で行えており、十分実用にたえるものだと考えられる。

5. おわりに

本稿では、エンドツーエンド型モビリティサポート手法における、通信の維持に関するトラッキング機構の実装を行い、その切り替え遅延に関して評価した。

今後、アドレス変更時にも信頼性のある通信を行うアプ

リケーションに対応するためのSOCK_STREAM型の再送処理に関する実装やセキュリティ機能、アプリケーションへのネットワーク状態通知機能などの実装をすすめていきたい。また、IPアドレスの変更だけではなく、ロケーションの変更や端末の変更にも対応できる用に、本実装の拡張をすすめていきたいと考えている。さらに、本エンドツーエンド型モビリティサポートに対応したアプリケーションの開発もあわせて行いたいと考えている。

文 献

- [1] 金子 晋丈, 森川 博之, 青山 友紀, 中山 雅哉, "多様化するインターネット環境におけるエンドツーエンド型モビリティサポート," 信学技報, MoMuC2002, May. 2002.
- [2] J. Quarterman, A. Silberschatz, and J. Peterson, "4.2BSD and 4.3BSD as Examples of the UNIX System," ACM, Computer Surveys, Vol. 17, No. 4, Dec. 1985.
- [3] J. Postel, "Transmission Control Protocol," IETF, RFC 793, Sep. 1981.
- [4] J. Postel, "User Datagram Protocol," IETF, RFC 768, Aug. 1980.