C言語の通信ライブラリを呼び出すJavaラッパーの実現と評価

清水 一輝[1]　八里 栄輔[2]　納堂 博史[1]　鈴木 秀和[1]　内藤 克浩[3]　渡邊 晃[1]

概要：モバイルネットワークの普及に伴い，あらゆるネットワーク環境においても通信を開始することができる通信接続性と通信しながらネットワークを切り替え可能な移動透過性が求められている．NTMobile(Network Traversal with Mobility) は通信接続性と移動透過性を同時に実現する次世代の技術である．NTMobile にはアプリケーションで実現する NTMobile フレームワークという通信ライブラリが存在する．この通信ライブラリは C 言語にて実装されているため，使用可能なアプリケーションが C 言語に限られていた．そこで本稿では，NTMobile フレームワークを Java にて利用可能にするラッパーを実現し，評価したので報告する．2 種類の Java ラッパーを実現し，両者を比較した．

Realization and Evaluation of Java Wrapper which calls C-language Communication Library

KAZUKI SHIMIZU[1]　EISUKE YASATO[2]　HIROSHI NODO[1]　HIDEKAZU SUZUKI[1]
KATSUHIRO NAITO[3]　AKIRA WATANABE[1]

1. はじめに

スマートフォンのような移動通信端末や無線通信技術の普及に伴って，ネットワーク利用の需要が増加している．IPv4 ネットワークでは，グローバルアドレスの枯渇が深刻な問題となっている．この問題に対する短期的な解決策として NAT(Network Address Translation) を利用することが一般的である．しかし，NAT にはグローバルネットワーク側から NAT 配下のプライベートネットワーク側に対して通信を開始することができないという NAT 越え問題が存在するため，双方向通信の妨げとなっている．IP アドレス枯渇の長期的な解決策として，IPv6 ネットワークへ移行する必要がある．しかし，IPv6 ネットワークと IPv4 ネットワークとの互換性がないため，普及が進んでいない．そこで，IPv4 アドレスと IPv6 アドレスが混在した環境は長期に渡り続くと考えられる．このような現状から，接続しているネットワークの環境を問わず通信を開始することが可能な通信接続性が求められている．また，通信端末が移動しネットワークが切り替わると，IP アドレスが変化するため通信を継続することができない．従って，ネットワークが切り替わった場合にも通信を継続できる技術である移動透過性が求められている．

通信接続性と移動透過性を同時に実現する技術として，DSMIP(Dual Stack Mobile IPv6)[1] や HIP(Host Identity Protocol)[2]，NTMobile(Network Tracersal with Mobility)[3], [4], [5] がある．これらの技術は，NAT 越え問題や IPv4/IPv6 間の相互通信といった通信接続性と移動透過性を同時に実現することを目的とした技術である．

DSMIP は IPv6 に対応した移動透過性技術 Mobile IPv6[6] をベースとし，IPv4 が混在した環境に拡張した方式である．しかし，DSMIP は IPv4 ネットワークにおける Mobile IPv4[7] の課題をそのまま引き継いでいる．例えば，全ての移動通信端末に IPv4 グローバルアドレスが必要となり，IP アドレス枯渇問題に逆行するという課題が存在する．

HIP は IP アドレスから通信識別子の役割を分離し，

[1] 名城大学
Meijo University
[2] バレイキャンパスジャパン
Valley Campus Japan Inc.
[3] 愛知工業大学
Aichi Institute of Technology

HI(Host Identity) と呼ばれる新たな通信識別子を導入することにより，通信接続性と移動透過性を実現する．HIP は NAT 越え技術として ICE(Interactive Connectivity Establishment)[8], [9] を利用しているが，ICE はもともと移動を考慮した技術ではないため，NAT を跨ぐ移動が複雑になり，シグナリングに要する時間が大きくなるという課題が存在する．また，HIP は TCP 層と IP 層の間に HIP 層を定義することにより実現されているため，カーネルを改造する必要がある．そのためスマートフォン等への適用が困難である．

NTMobile は，システム内において一意となる仮想 IP アドレスを各通信端末に割り当て，全ての通信パケットを実 IP アドレスでカプセル化する方式である．NTMobile には DSMIP や HIP で述べた課題は存在しない．NTMobile の実装モデルの 1 つとして NTMobile フレームワーク [10], [11] と呼ぶアプリケーションレベルの通信ライブラリがある．アプリケーションは，この通信ライブラリを使用することにより，通信接続性と移動透過性を同時に満たす通信を実現することができる．しかし，NTMobile フレームワークは C 言語にて実装されているため，この通信ライブラリを使用可能なアプリケーションが C 言語に限られていた．

そこで本稿では，C 言語の通信ライブラリを呼び出す Java ラッパーについて検討を行い，NTMobile フレームワークを Java にて使用可能とする Java ラッパーを 2 通りの方法で実現した．実現した 2 種類の Java ラッパーを使用し，Java アプリケーションにて NTMobile 通信を行うことを確認した．2 種類の Java ラッパーで，性能には差異がないが，両者に使用する場面で一長一短があることが分かった．

以後，2 章では通信接続性と移動透過性を同時に実現する NTMobile について，3 章では NTMobile フレームワークについて，4 章では，Java ラッパーの 2 種類の実装方法について，5 章では，Java ラッパーの動作と詳細について，6 章では，動作検証と性能評価について述べ，最後に 7 章でまとめる．

2. NTMobile

本章では，NTMobile の構成と必要な動作の概要について述べる．

2.1 NTMobile の構成

NTMobile の構成を図 1 に示す．NTMobile は下記の機器により構成される．

- DC(Direction Coordinator)
  NTM 端末の仮想 IP アドレスや位置情報を管理し，NTM 端末に対して UDP トンネルの構築指示を出す機器．通信相手の DC を探索するために，DNS サーバーとしての機能を有する．インターネット上に分散配置することが可能である．
- RS(Relay Server)
  NTM 端末間でエンドエンドの通信ができない場合にパケットを中継する機器．IPv4/v6 間の通信であったり，異なる NAT 配下での通信，一般端末 (GN:General Node) との通信にて使用される．インターネット上に分散配置することが可能である．
- NTM 端末 (NTMobile Node)
  NTMobile 機能を有する端末．

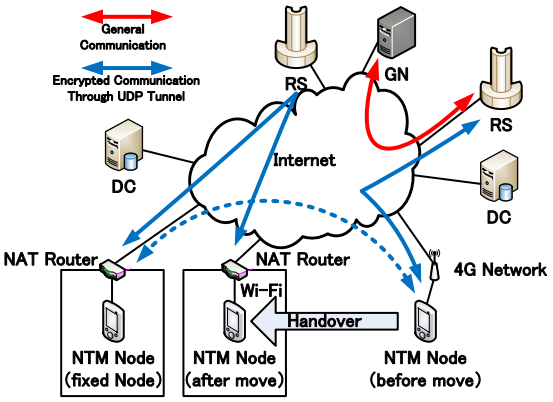

図 1 NTMobile の構成

2.2 NTMobile の原理

NTMobile は，DC が NTM 端末に対して位置に依存しない仮想 IP アドレスを割り当て，アプリケーションは仮想 IP アドレスに基づいた通信を行う．仮想 IP アドレスに基づいたパケットは，端末の実 IP アドレスでカプセル化を行い送信される．NAT 配下に存在する端末は DC との間で常時通信経路を確保しておき，プライベートネットワークとの接続性を維持する．仮想 IP アドレスは，端末の移動によって変化することがない．そのため移動により，端末の実 IP アドレスが変化してもアプリケーションは影響を受けることなく移動透過性を実現できる．

2.3 NTMobile の動作

NTMobile では，通信を開始する前に端末情報の登録を行い，その後，通信開始時のネゴシエーションを行う必要がある．図 2 に端末情報の登録処理を，図 3 に通信開始時のネゴシエーションを示す．実際には通信経路上に NAT が存在する場合があるが，簡単のため省略する．通信開始側の NTM 端末を MN(Mobile Node)，通信相手側の端末を CN(Correspondent Node) とする．

端末情報の登録では，MN と CN はそれぞれ自身の実 IP アドレス ($RIP_{MN/CN}$) を DC に登録する必要がある．各端末は実 IP アドレスを登録後，DC から自身の仮想 IP アド

レス ($VIP_{MN/CN}$) が配布される．仮想 IP アドレスを取得後は，DC と定期的に Keep Alive を行い，DC との通信経路を確保する．

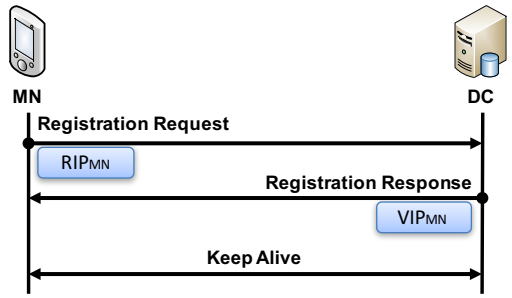

図 2　端末情報の登録

通信開始時のネゴシエーションでは，MN は DC に対して CN の名前解決及びトンネル構築の指示を依頼する．DC は最適な通信経路を判断し，MN と CN に対してトンネル構築を指示する．これにより，MN と CN はお互いの間でトンネル経路を構築する．

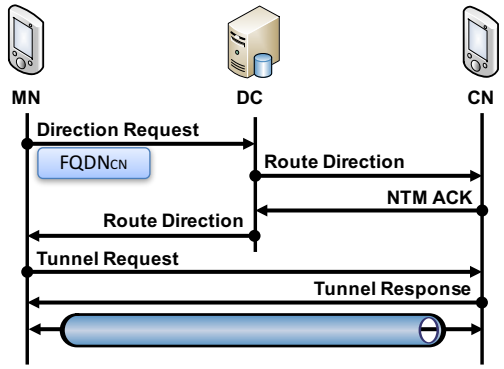

図 3　通信開始時のネゴシエーション

3.　NTMobile フレームワーク

本章では，NTMobile フレームワークの概要とその動作や構成，提供される NTM ソケット API について述べる．

3.1　フレームワークの概要

NTMobile フレームワークは NTMobile をアプリケーションライブラリとしてユーザーに提供する実装方式である．アプリケーションは C 言語による Linux 標準の通信ライブラリを利用するのと同様に，フレームワークを呼び出すことにより NTMobile 通信を利用することができる．そのため，アプリケーションは NTMobile をほとんど意識することなく利用することができる．

3.2　フレームワークの動作

フレームワークのトンネル通信の実現方法を図 4 に示す．アプリケーションが送信したデータは，仮想 IP スタックにより仮想 IP アドレスを用いて仮想 IP ヘッダが付与される．このパケットは，NTMobile 通信であることを示す NTM ヘッダが付与され，暗号化や MAC(Message Authentication Code) の付与が行われた後，C 標準ソケット API にて OS に渡される．ここまでの処理がフレームワークで実現される．このパケットは Linux カーネルにより UDP でカプセル化されて送信される．パケット受信時は上記と逆の動作により実現される．

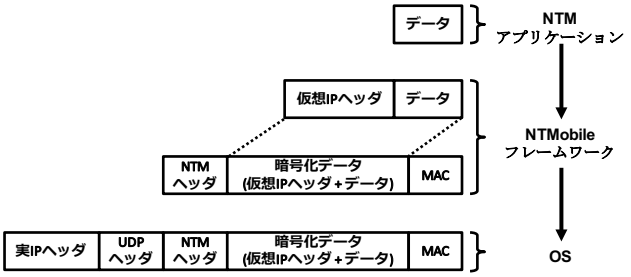

図 4　トンネル通信の実現方法

3.3　フレームワークの構成

フレームワークのモジュール構成を図 5 に示す．フレームワークは下記のモジュールにより構成される．

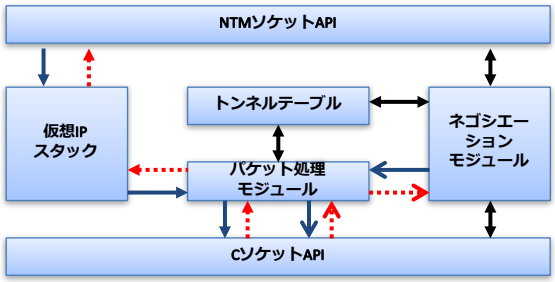

図 5　フレームワークのモジュール構成

- C ソケット API
  フレームワークがパケットの送受信を行うための Linux が指定する標準 API．制御メッセージやカプセル化パケットはこの API を通して送受信される．
- NTM ソケット API
  C ソケット API に代わり，アプリケーションに提供する NTMobile 用のソケット API．フレームワーク独自の API も含まれる．詳細は次節で述べる．
- ネゴシエーションモジュール
  NTMobile の制御メッセージの処理やアドレス情報の監視を行う．名前解決を行う NTM ソケット API が呼び出された場合や他の端末から通信要求があった場合，このモジュールによりトンネル構築処理が行われ，

トンネルテーブルが更新される．また，端末の IP アドレスを確認し，IP アドレスに変化があった場合は DC に対してアドレス情報の更新を行い，構築済みの全トンネルを再構築する．

- パケット処理モジュール
  パケットに対して改ざん検知のための MAC の付与や検証，暗号化，復号を行う．また，パケットの種類に応じてネゴシエーションモジュールと仮想 IP スタックに処理を割り振る．
- 仮想 IP スタック
  アプリケーションが送受信するデータの TCP/IP 処理を行う．アプリケーションが送信するデータはこのモジュールにより仮想 IP ヘッダが付与され，パケット処理モジュールに処理を渡す．
- トンネルテーブル
  通信相手毎の FQDN や仮想 IPv4/v6 アドレス，実 IPv4/v6 アドレス，RS の実 IPv4/v6 アドレス等をメンバとするエントリを持つ．複数のキーを持つハッシュテーブルとなっており，ハッシュキーとして FQDN や仮想 IPv4/v6 アドレス等を利用できる．一定時間参照されなかったエントリは自動的に削除される．

3.4 NTM ソケット API

アプリケーションが NTMobile をできるだけ意識しなくて済むように，C ソケット API に対応したインターフェースを持つ．NTM ソケット API と C ソケット API の対応付けを表 1 に示す．NTM ソケット API の引数は C ソケット API と互換性を持つ．

表 1 NTM ソケット API と C ソケット API の対応付け

NTM ソケット API	C ソケット API
ntmfw_socket	socket
ntmfw_getaddrinfo	getaddrinfo
ntmfw_bind	bind
ntmfw_connect	connect
ntmfw_sendto	sendto
ntmfw_recvfrom	recvfrom
ntmfw_send	send
ntmfw_recv	recv

上記とは別に，NTMobile 特有の動作のために提供する API が存在し，これを表 2 に示す．`ntmfw_ntm_init` は図 2 の端末情報の登録処理，`ntmfw_getaddrinfo` は図 3 の通信開始時のネゴシエーションに用いられる．

表 2 フレームワーク特有の API

API	機能
ntmfw_ntm_init	端末情報の登録 (図 2) を行う．
ntmfw_getaddrinfo	通信開始時の処理 (図 3) を行う．

4. Java ラッパー

本章では，Java ラッパーとして考えられる 2 通りの方式について述べる．

4.1 Java ラッパーの種別

模倣型 Java ラッパーとファクトリ型 Java ラッパーと呼ぶこととする．模倣型 Java ラッパーは，Java のソケットクラスを模倣して作成したクラスから，C 言語の通信ライブラリを呼び出す．ファクトリ型 Java ラッパーは，アプリケーションにソケット実装ファクトリを設定することにより，Java の標準ソケット API で C 言語の通信ライブラリを使用するよう再定義を行う．いずれも，Java から C 言語の通信ライブラリを呼び出すために JNA(Java Native Access) を利用する．JNA は，Java 以外のコーディングをすることなく，C 言語の共有ライブラリに動的にアクセスする方法を提供するオープンソースのライブラリである．C 言語のライブラリにアクセス後は，C 言語の関数を Java のメソッドにマッピングをすることにより，Java から C 言語の関数を使用できる．

4.2 模倣型 Java ラッパー

模倣型 Java ラッパーのモジュール構成を図 6 に示す．

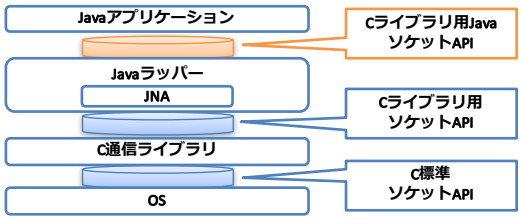

図 6 模倣型 Java ラッパーのモジュール構成

模倣型 Java ラッパーでは，ラッパーにおいて JNA を使用し，C 言語の通信ライブラリで記述されたソケット API を呼び出すラッパークラスを定義する必要がある．Java アプリケーションはラッパークラスに定義された C ライブラリ用 Java ソケット API を使用することにより C 通信ライブラリのソケット API を使用できる．C 通信ライブラリ用のソケット API を使用すると，Java と C 言語とでの言語間の違いを除去した後に，C 標準のソケット API が使用され，パケットの送受信が行われる．

4.3 ファクトリ型 Java ラッパー

ファクトリ型 Java ラッパーのモジュール構成を図 7 に示す．

JNA を使用して C 言語のソケット API を呼び出すラッパークラスを定義するまでは，ファクトリ型と模倣型 Java ラッパーは同様である．ファクトリ型ではラッパークラス

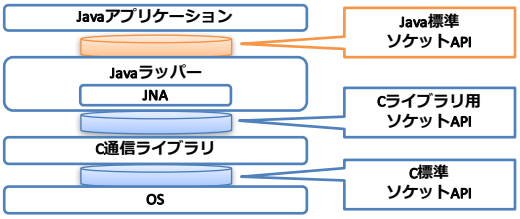

図 7 ファクトリ型 Java ラッパーのモジュール構成

にて提供する API を利用して，Java の標準ソケットクラスを継承するサブクラスを作成し，サブクラスをアプリケーションのソケット実装ファクトリと設定する．この設定を行うことにより，Java 標準のソケットクラスに属する API は C ライブラリ用ソケット API を使うよう再定義される．これにより，C 言語の通信ライブラリのソケット実装ファクトリを設定されたアプリケーションは，Java 標準のソケット API を使用すると，代わりに C ライブラリ用ソケット API を使用して通信を行うように変更される．

4.4 模倣型/ファクトリ型ラッパーの違い

模倣型 Java ラッパーでは，クラス名は Java の標準ソケットクラスと異なるが，その代わりに Java の標準ソケットクラスを使用することができる．それに対し，ファクトリ型 Java ラッパーでは，クラス名は Java の標準ソケットクラスと同じだが，その代わりに Java の標準ソケットクラスを使用することができないといった違いがある．

5. NTMobile 用 Java ラッパーの動作と実装

本章では，Java ラッパーの動作の詳細と構成するクラス，実装方法について述べる．

5.1 Java ラッパーの動作

NTMobile 用 Java ラッパーで行う処理内容を以下に示す．主な処理内容は以下の 3 つである．

- NTM ソケット API のマッピング
- C 言語と Java での違いの除去
- Java の標準ソケット API を NTMobile 用に再定義

NTM ソケット API のマッピングは，JNA を用いて行う．JNA を使用することにより NTMobile フレームワークのソースコードを一切編集する必要がなくなる．これにより，呼び出すライブラリに変更が生じた場合にも最小限の更新をラッパーのみに行うだけでライブラリの変更に対応可能である．

C 言語と Java での違いの除去はプログラミング言語が異なることにより生じる型名の違いであったり，同等の機能を持つ API の引数に関する違いである．型の違いは，型のサイズに応じて型名を変更することで対応可能である．また，API の引数の違いは，引数にて得られたデータを C 言語の API の引数に合うように細分化し，渡すことで対応可能である．

Java の標準ソケット API を NTMobile 用に再定義することは，ファクトリ型 Java ラッパーを作成する上で必要な内容である．あらかじめソケット実装ファクトリを生成できるクラスを用意しておくことにより，Java アプリケーションが最初に NTMobile のソケット実装ファクトリを設定することにより対応可能である．

5.2 Java ラッパーの実装

図 8 に模倣型，図 9 にファクトリ型の Java ラッパーを示す．Java ラッパーは主に下記のモジュールにより構成される．

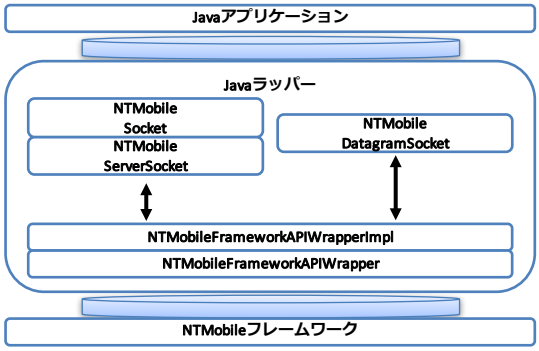

図 8 模倣型 Java ラッパーの詳細

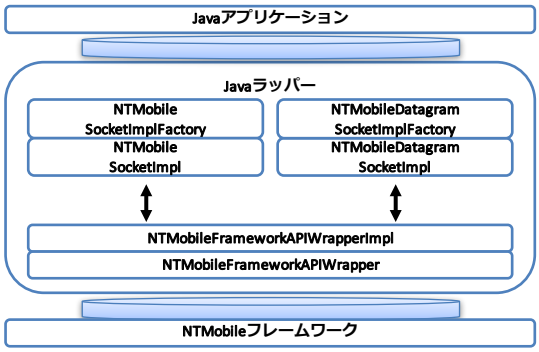

図 9 ファクトリ型 Java ラッパーの詳細

- NTMobileSocket クラス
  NTMobile を使用する Socket クラスの実装．
  Socket クラスを模倣．
- NTMobileServerSocket クラス
  NTMobile を使用する ServerSocket クラスの実装．
  ServerSocket クラスを模倣．
- NTMobileSocketImplFactory クラス
  NTMobileSocketImpl クラスのファクトリ．
  SocketImplFactory クラスを継承．
- NTMobileSocketImpl クラス
  NTMobile を使用する Socket の実装を定義．
  SocketImpl クラスを継承．

- NTMobileDatagramSocket クラス
  NTMobile を使用する DatagramSocket クラスの実装．DatagramSocket クラスを模倣．
- NTMobileDatagramSocketImplFactory クラス
  NTMobileDatagramSocketImpl クラスのファクトリ．DatagramSocketImplFactory クラスを継承．
- NTMobileDatagramSocketImpl クラス
  NTMobile を使用する DatagramSocket の実装を定義．DatagramSocketImpl クラスを継承．
- NTMobileFrameworkAPIWrapperImpl クラス
  Java の標準 API と同じ使い方で使用可能な NTM ソケット API を定義．
- NTMobileFrameworkAPIWrapper クラス
  C 言語で記述された NTMobile フレームワークの NTM ソケット API を JNA を使い定義．

模倣型 Java ラッパーでは，NTMobileSocket/NTMobileServerSocket/NTMobileDatagramSocket クラスのインスタンスを生成後，Socket/ServerSocket/DatagramSocket クラスの API と同じ使い方で使用できる．

ファクトリ型 Java ラッパーでは，Java アプリケーションは始めに，NTMobileSocketImplFactory/NTMobileDatagramSocketImplFactory クラスをアプリケーションのソケット実装ファクトリとして設定する必要がある．それ以降は，Socket/DatagramSocket クラスを使い，NTMobile を用いた TCP/UDP 通信を行うことができる．

但し，両ラッパーとも端末情報の登録と通信開始時のネゴシエーションを行う API を NTMobileFrameworkAPIWrapperImpl クラスから呼び出す必要がある．

6. 評価

6.1 性能測定

模倣型とファクトリ型の NTMobile 用 Java ラッパーを実装し，UDP でメッセージを送信する Java アプリケーションに適用した．1 台のホストマシン上に DC と NTM 端末 2 台を仮想マシンとして構築し，これら 3 台の仮想マシンを同一 IPv4 プライベートネットワークに接続し，動作検証及び処理時間の測定を行った．動作検証及び処理時間の測定を行った際のネットワーク構成及び各仮想マシンの構成を図 10，表 3 に示す．

表 3 各仮想マシンの構成

	DC	MN/CN
OS	Ubuntu 12.04	Ubuntu 14.04
CPU 割り当て	1Core(3.40GHz)	2Core(3.40GHz)
Memory 割り当て	1.00GB	2.00GB

測定して得られた処理時間の 100 回を平均した結果を以下に示す．表 4 は模倣型，表 5 はファクトリ型の Java ラッパーの測定結果である．また，図 11 に測定箇所の範囲を示す．

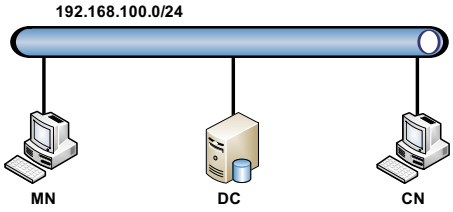

図 10 ネットワーク構成

表 4 模倣型 Java ラッパーの処理時間の測定結果

測定箇所	送信時 [ms]	受信時 [ms]
Java ラッパー	0.83	0.17
NTMobile フレームワーク	0.41	1.20
合計	1.24	1.37

表 5 ファクトリ型 Java ラッパーの処理時間の測定結果

測定箇所	送信時 [ms]	受信時 [ms]
Java ラッパー	0.87	0.19
NTMobile フレームワーク	0.45	1.25
合計	1.32	1.44

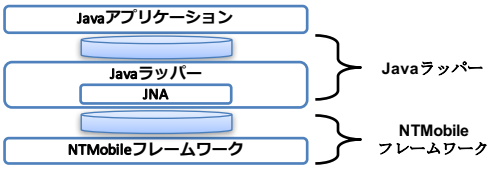

図 11 処理時間の測定箇所

模倣型 Java ラッパーでは送信時に約 1.2 ミリ秒，受信時に約 1.4 ミリ秒の時間を要した．また，ファクトリ型 Java ラッパーでは送信時に約 1.3 ミリ秒，受信時に約 1.4 ミリ秒の時間を要した．これらの結果より，模倣型とファクトリ型の Java ラッパーにおける処理時間の差はほとんどないことが分かった．

6.2 比較

模倣型とファクトリ型の Java ラッパーを比較した結果を表 6 に示す．

表 6 2 種類のラッパーの比較

	項目 (1)	項目 (2)
模倣型 Java ラッパー	○	×
ファクトリ型 Java ラッパー	×	○

評価項目の内容は以下の 2 項目とした．

(1) アプリケーション作成時にラッパーを使う場合

(2) UDP または TCP による通信を全て C 言語の通信ライブラリを使う場合

模倣型 Java ラッパーは一般通信と C 言語の通信ライブラリを用いた通信の使い分けができるのに対し，ファクトリ型 Java ラッパーは通信は全て C 言語の通信ライブラリを用いた通信しかできない．そのため，新規アプリケーション作成時では模倣型 Java ラッパーの方が優れている．一方で，アプリケーションが全て C 言語の通信ライブラリを用いる場合は，ファクトリ型 Java ラッパーの方が優れている．ファクトリ型 Java ラッパーはソケット実装ファクトリを 1 度設定するだけで UDP または TCP による通信を全て C 言語の通信ライブラリを用いるように変更できる．それに対し，模倣型 Java ラッパーは独自に定義したクラス名を使用するため，ヒューマンエラーを引き起こす可能性が考えられる．よって，このような用途には適していない．

7. まとめ

本稿では，C 言語でしか使えなかった NTMobile フレームワークを Java から利用できるような Java ラッパーを検討した．2 通りの方法を提示し，両ラッパーを実現して動作検証を行った．両者は性能に差異はないが，使用される場面に応じて一長一短があることを示した．今後，他のプログラミング言語のラッパーを検討する予定である．

参考文献

[1] H. Soliman. Mobile IPv6 Support for Dual Stack Hosts and Routers. RFC 5555, IETF, 2009.

[2] R. Moskowitz, T. Heer, P. Jokela, and T. Henderson. Host Identity Protocol Version 2(HIPv2). RFC 7401, IETF, 2015.

[3] 鈴木秀和, 上醉尾一真, 水谷智大, 西尾拓也, 内藤克浩, 渡邊晃. NTMobile における通信接続性の確立手法と実装. 情報処理学会論文誌, Vol. 54, No. 1, pp. 367–379, 2013.

[4] 内藤克浩, 上醉尾一真, 西尾拓也, 水谷智大, 鈴木秀和, 渡邊晃, 森香津夫, 小林英雄. NTMobile における移動透過性の実現と実装. 情報処理学会論文誌, Vol. 54, No. 1, pp. 380–397, 2013.

[5] 上醉尾一真, 鈴木秀和, 内藤克浩, 渡邊晃. IPv4/IPv6 混在環境で移動透過性を実現する NTMobile の実装と評価. 情報処理学会論文誌, Vol. 54, No. 2013, pp. 2288–2299, 2013.

[6] C. Perkins, D. Johnson, and J. Arkko. Mobility Support in IPv6. RFC 6275, IETF, 2011.

[7] C. Perkins. IP Mobility Support for IPv4. RFC 5944, IETF, 2010.

[8] J. Rosenberg. Interactive Connectivity Establishment(ICE): A Protocol for Network Address Translator(NAT) Traversal for Offer/Answer Protocols. RFC 5245, IETF, 2010.

[9] M. Westerlund and C. Perkins. IANA Registry for Interactive Connectivity Establishment(ICE) options. RFC 6336, IETF, 2011.

[10] K. Naito, K. Kamienoo, H. Suzuki, A. Watanabe, K. Mori, and K. Kobayashi. End-to-end IP mobility platform in application layer for iOS and Android OS. In *Proc. of IEEE CCNC*, 2014.

[11] 納堂博史, 八里栄輔, 鈴木秀和, 内藤克浩, 渡邊晃. 実用化に向けた NTMobile フレームワークの実装と評価. 情報処理学会研究報告, 第 82 回 MBL・第 53 回 UBI 合同研究発表会, 2017.