平成28年度　卒 業 論 文

和文題目

NTMobile用
Javaラッパーの提案と実装

英文題目

Proposal and Implementation of
Java Wrapper for NTMobile

情報工学科　渡邊研究室

(学籍番号: 130441077)

清水　一輝

提出日: 平成29年2月10日

名城大学理工学部

概要

NTMobile（Network Traversal with Mobility）は移動透過性と通信接続性を同時に実現する次世代の技術である．NTMobile はこれまで Linux カーネルに実装を行っていたが，これをアプリケーションに移植を行っている．これを NTMobile フレームワークと呼んでいる．フレームワークは C 言語により記述されているが，アプリケーションは Java によって記述されることが多い．そこで本研究では，NTMobile フレームワーク用の Java ラッパーを実現する方式について提案し，実装した結果を示す．Java ラッパーでは JNA（Java Native Access）を使用し，NTMobile フレームワークのライブラリを使用することが可能である．また，アプリケーション作成者が Java ラッパーを使用することにより，Java 標準 API を使用して NTMobile の通信を実現することが可能である．

Abstract

NTMobile(Network Traversal with Mobility) is a next generation technology which realizes mobility and connectivity simultaneously. We implemented previous Studies about NTMobile to Linux Kernel. Currently, it is being ported to Applications. This is called NTMobile Framework. NTMobile Framework is coded in C-language. However, Application is coded in Java frequently. Therefore, in this study, we propose Java Wrapper for NTMobile Framework and present the results of implementation. JNA(Java Native Access) makes it possible to use NTMobile Framework Library on Java Wrapper. Another thing, an Application developer realizes that NTMobile Communication with Java API by using Java Wrapper.

目 次

第1章　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．現在の IP ネットワークでは，通信端末に割り当てられてた IP アドレスが通信識別子となっている．よって通信端末が移動し，ネットワークが切り替わると IP アドレスが変化する為，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．現在主流である 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）を利用するのが一般的である．NAT は自身の配下に存在する通信端末に対しプライベートアドレスを割り当て，NAT がプライベートアドレスとグローバルアドレスを変換することにより，通信を行う．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為，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 ネットワークは IPv4 アドレスとの互換性がない為，普及が進んでいない．従って，IPv4 アドレスと IPv6 アドレスの混在した環境が長期に渡り続くであろうと考えられる．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．

IPv4 ネットワークで移動透過性を実現する技術として MIPv4（Mobile IPv4）[1]，MIPv6（Mobile IPv6）[2]，DSMIP（Dual Stack Mobile IPv6）[3] が提案されている．これらの技術は通信パケットが HA（Home Agent）と呼ばれるアドレス管理機能とパケット中継機能を持つ通信中継装置を経由する為，冗長な経路になるという課題が存在する．

NAT 越えを実現する技術として STUN（Session Traversal Utilities for NATs）[4]，TURN（Traversal Using Relay around NAT）[5]，ICE（Interactive Connectivity Establishment）[6, 7] 等が提案されている．STUN，TURN，ICE は，NAT に改造を加える必要がないが，アプリケーションがこれらの技術に対応している必要がある．また，これらの技術は移動通信端末について考慮されていない為，移動透過性を実現することができない．

我々は，通信接続性と移動透過性を同時に実現する技術として NTMobile（Network Traversal with Mobility）[8–10] を提案している．NTMobile では，通信端末のアプリケーションは仮想 IP アドレスで通信を識別し，実際の通信は実 IP アドレスでカプセル化する．その為，アプリケーションでは NAT の存在の有無や移動による実 IP アドレスの変化を意識する必要がなく，通信中に自由に IPv4 グローバル/IPv4 プライベート/IPv6 ネットワークを切り替えることができる．

NTMobile は Linux カーネルに実装を行っていたが，スマートフォン等ではルート権限が必要となる為，利用できなかった．そこで NTMobile をアプリケーションに移植し，通信ライブラリを

提供する方法として NTMobile フレームワークを検討している [11]．NTMobile フレームワークは C 言語によって記述されているが，アプリケーションは Java によって記述されることが多い．そこで本論文では，Java アプリケーションの作成者が NTMobile をほとんど意識せずに利用できる NTMobile 用 Java ラッパーを検討した．NTMobile 用 Java ラッパーでは，アプリケーション作成者が Java 標準 API の通信クラスを使用した場合に，NTMobile フレームワークの共有ライブラリから NTMobile による通信をする為の関数を呼び出し，使用する．NTMobile 用 Java ラッパーを実装し，Java アプリケーションにて NTMobile フレームワークを使用し，NTMobile 通信を行うことを確認した．

以後，2 章では一般的な Java ラッパーを使用した Java アプリケーションについて，3 章では NTMobile について述べる．4 章では提案する NTMobile 用 Java ラッパーの実装について，5 章では Java ラッパーの動作検証と性能評価，最後に 6 章でまとめる．

第2章　Javaアプリケーション

本章では，JNA（Java Native Access）を用いたJavaアプリケーションにおけるラッパーについて述べる．

2.1　JNA（Java Native Access）

JNA[*1]とは，共有ライブラリに容易にアクセスする方法を提供するライブラリである．JNAでは複雑なデータ型を使用する一部のAPIや特別な処理が必要となる配列等を除けば，C言語のコードを記述する必要なく，Javaアプリケーションが共有ライブラリAPIを呼び出すことが可能である為，システム開発の生産性が向上する．また，自動的にマルチプラットフォームに対応する為，プラットフォームを意識することなく開発できる．

JNAの主な機能として，大きく以下の2つが存在する．

(i) 共有ライブラリのロード
(ii) メソッドのマッピング

(i)では，共有ライブラリをJavaのインタフェースでマッピングする．図1に共有ライブラリをマッピングする際の流れを示す．共有ライブラリは予めC言語のソースファイルをコンパイルし，作成しておく必要がある．Javaアプリケーションで JNAのNativeクラス `loadLibrary` メソッドを使い，使用する共有ライブラリ名を指定する．JNAは `loadLibrary` メソッドにより指定された共有ライブラリをロードする．これにより，共有ライブラリが使用可能になる．

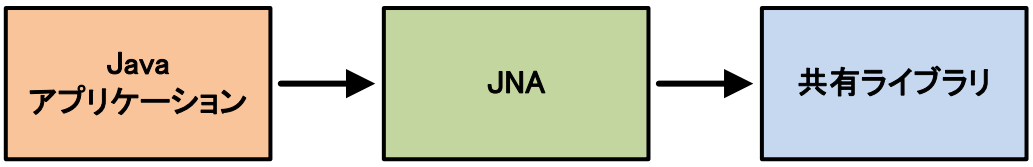

図1　共有ライブラリのロード

(ii)では，共有ライブラリが提供するネイティブAPIとJavaメソッドのマッピングを行う．図2にJNAによりロードした共有ライブラリとメソッドのマッピング例を示す．メソッドをマッピングするには，共有ライブラリにて定義されている関数名すなわちC言語のソースファイルに定義

[*1] https://github.com/java-native-access

された関数名と同名にする必要があり，引数等の型も Java と C 言語で型のデータサイズを合わせる必要がある．

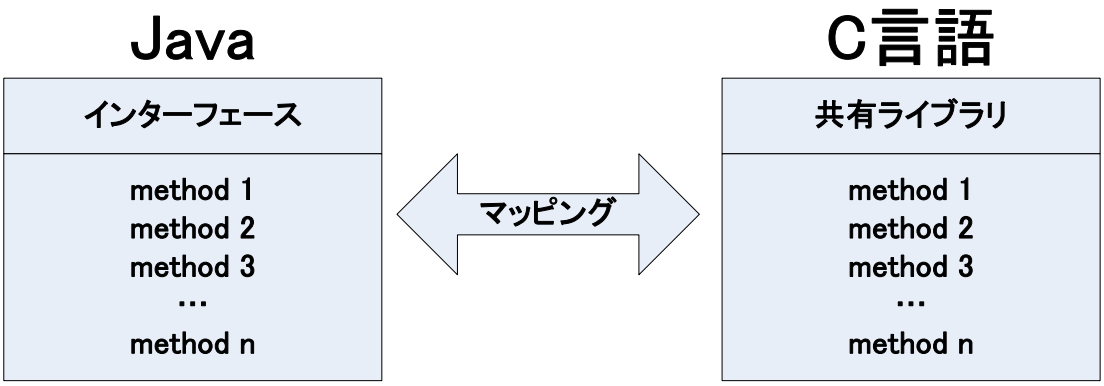

図 2　メソッドのマッピング例

以上の (i)，(ii) により，Java アプリケーションにて共有ライブラリ内の C 言語の関数を自由に使用することが可能になる．

2.2　Java ラッパー

共有ライブラリを使用するための Java ラッパーは，2.1 節の JNA を使用することによる共有ライブラリのロードと関数に渡す引数の型の変換を行い，変換後の値をマッピングしたメソッドに渡すのみである．

ラッパークラスの例として，図 3 にて Java にて標準で用意されている int 型のラッパークラスである Integer クラス [⋆2] を示す．Java にて int 型の変数を String 型の変数に型を変換したい場合，int 型のラッパークラスである Integer クラスにて用意されている `toString(int i)` メソッドを使用する必要がある．図 3 では，int 型の値 4330 を `toString(4330)` として Integer クラスに値を渡すと，ラッパークラスである Integer クラスにて型の変換が行われ，返り値として String 型の変数 4330 が得られる．

[⋆2] https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html

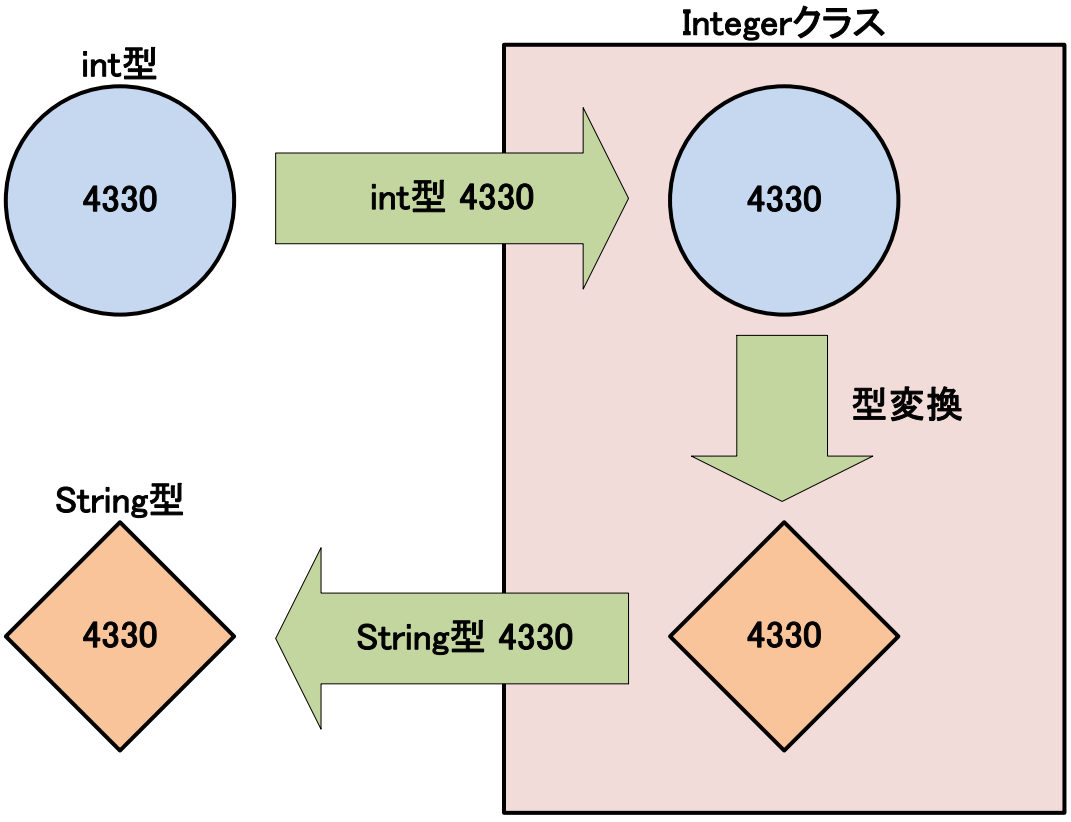

図 3　int 型のラッパークラス

第3章　NTMobile

本章では，提案方式を適用する NTMobile の概要と動作について述べる．

3.1　NTMobile の概要

NTMobile は，ネットワークが切り替わった際にも通信を継続可能な移動透過性とネットワーク環境に関わらず通信を開始することが可能な通信接続性の両者を実現する次世代の技術である．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 アドレスは，端末の移動により変化せず，実ネットワークに依存しない IP アドレスである．その為，通信中に端末のネットワークが切り替わった場合でも通信相手の端末やアプリケーションに対して IP アドレスの変化を隠蔽し，通信を継続することが可能である為，通信接続性と移動透過性を実現する．

3.2　NTMobile の構成

図 4 に NTMobile の構成を示す．NTMobile は，NTMobile 機能を実装した端末である NTM 端末，NTM 端末の端末情報の管理とトンネル経路指示を行う DC（Direction Coordinator），エンドエンドでの通信が行えない場合にパケットを中継する RS（Relay Server）によって構成される．DC 及び RS は，グローバルネットワークに設置し，ネットワークの規模に応じて複数台の設置が可能である．

NTMobile は，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．DC は DNS サーバーの機能を持っており，NTM 端末の通信開始時に通信相手の名前解決を行った後に名前解決により得られた情報から最適な通信経路の指示をする．また，NAT 配下の端末は DC に対して定期的な Keep Alive を行うことにより，端末との通信経路を確保し，NAT が存在するプライベートネットワークにおいても通信接続性を実現する．仮想 IP アドレスは，端末の移動により変化せず，実ネットワークに依存しない IP アドレスである．アプリケーションによって生成された仮想 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，通信相手に送信される．端末同士がエンドエンドで通信できない場合は RS を経由した通信をする．RS を経由する際，複数の RS の中から 1 つ選択し，冗長経路の少ない経路を生成可能である．

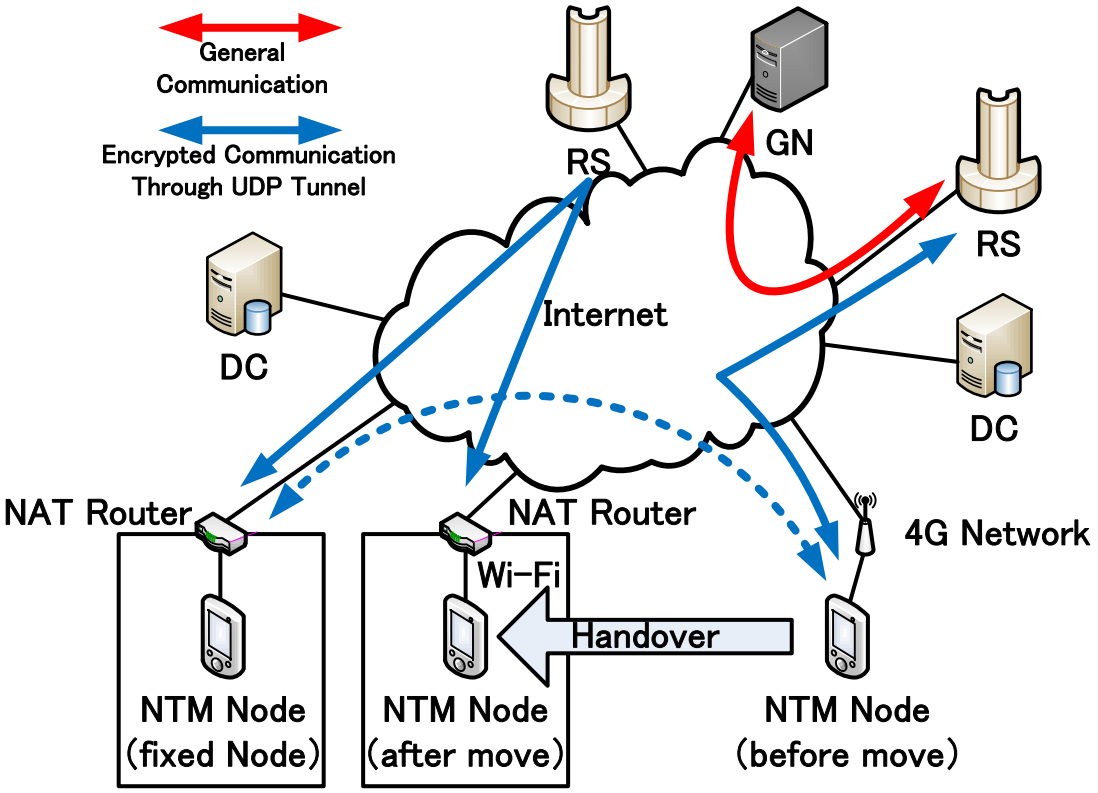

図 4　NTMobile の構成

3.3　NTMobile の動作

以降の説明では，通信開始側の NTM 端末を MN（Mobile Node），通信相手側の NTM 端末を CN（Correspondent Node）として説明する．また，NTM 端末 N の実 IPv4 アドレスを RIP_N，仮想 IPv4 アドレスを VIP_N とし，NTM 端末 N を管理する DC のを DC_N とする．

3.3.1　端末起動時

図 5 に端末起動時の動作を示す．端末起動時に MN は自身を管理する DC_{MN} に対して RIP_{MN} 等の端末情報の登録を行う．DC_{MN} は MN の端末情報をデータベースに登録後，MN に対して VIP_{MN} を配布する．VIP_{MN} を取得後は DC に対して定期的な Keep Alive を行うことにより通信経路を確保を行う．

3.3.2　通信開始時

図 6 に通信開始時の動作を示す．通信開始時に MN は DC_{MN} 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_{MN} は，DNS サーバーの仕組みを利用し，DC_{CN} を探索し，DC_{CN} から CN の端末情報を取得する．その後，DC_{MN} は MN 及び CN の端末情報から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．MN と CN は DC_{MN} の指示に従い，MN と CN

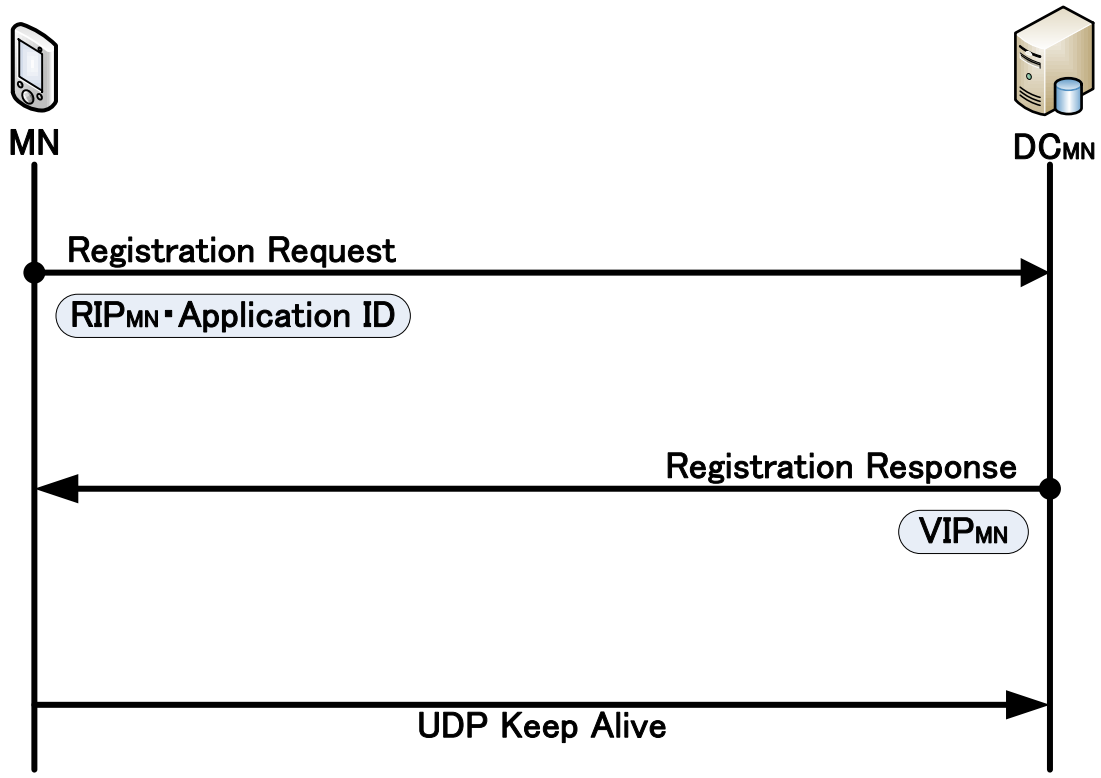

図 5　端末起動時の動作

との間でトンネル経路を構築する．MN または CN が NAT 配下に存在する場合は NAT を経由してトンネルを構築する．RS を経由する場合は，MN/CN と RS との間でトンネルを構築する．

3.3.3　トンネル通信時

図 7 にトンネル通信時の動作を示す．MN 側では，まず仮想 IP アドレスを用いてパケット（送信元 : VIP_{MN}，宛先 : VIP_{CN}）を生成する．その後，仮想 IP アドレスに基づくパケットは NTMobile の機能により実 IP アドレス（送信元 : RIP_{MN}，宛先 : RIP_{CN}）でカプセル化され，CN へ送信される．MN からパケットを受け取った CN は，NTMobile の機能によりパケットをデカプセル化し，仮想 IP アドレスに基づくパケットを取り出す．その後データを取り出し，CN のアプリケーションに渡す．

以上より，MN や CN がネットワークを切り替えて実 IP アドレスが変化した場合においてもアプリケーションが認識している仮想 IP アドレスは変化しない為，通信を継続可能である．

3.4　NTMobile フレームワーク

図 8 に NTMobile フレームワークの構成を示す．アプリケーションは NTMobile フレームワークが提供するソケット API を使用することで NTMobile による通信を行う．NTMobile フレームワークは，NTMobile フレームワークのソケット API にてアプリケーションから渡された値をカプセル

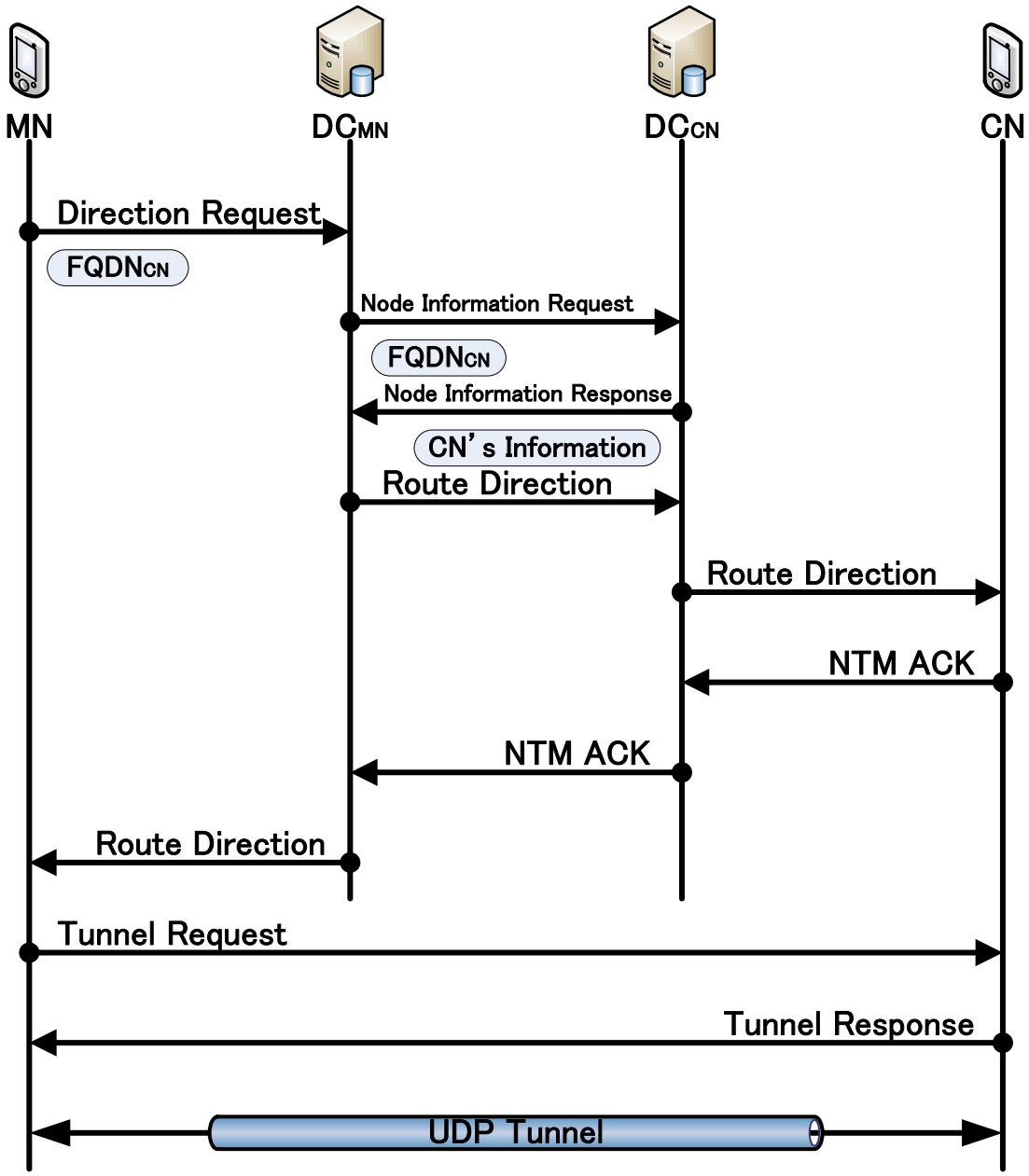

図 6　通信開始時の動作

化やネゴシエーションメッセージ処理，トンネルテーブルの作成等を行い，その後カーネルソケットに値を受け渡すことにより動作する．

フレームワークは，上位アプリケーションに対して ntmfw_ntm_init 関数，ntmfw_getaddrinfo 関数，UDP による送信/受信では ntmfw_sendto/ntmfw_recvfrom 関数，TCP による送信/受信では ntmfw_send/ntmfw_recv 関数を提供する．ntmfw_ntm_init 関数では図 5 に示す端末起動時の処理，ntmfw_getaddrinfo 関数では図 6 に示す通信開始時の処理を行い，ntmfw_sendto/ntmfw_recvfrom 関数や ntmfw_send/ntmfw_recv 関数により通信を行う．

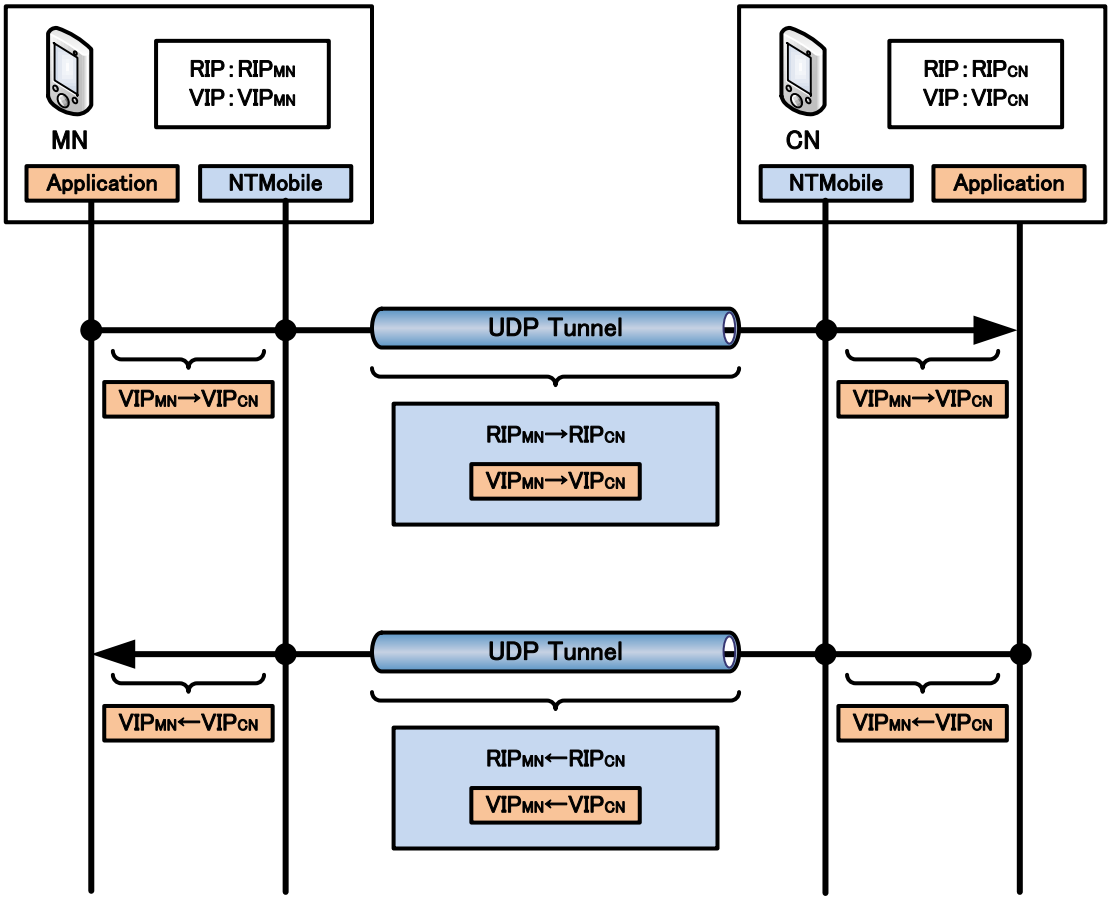

図 7　トンネル通信時の動作

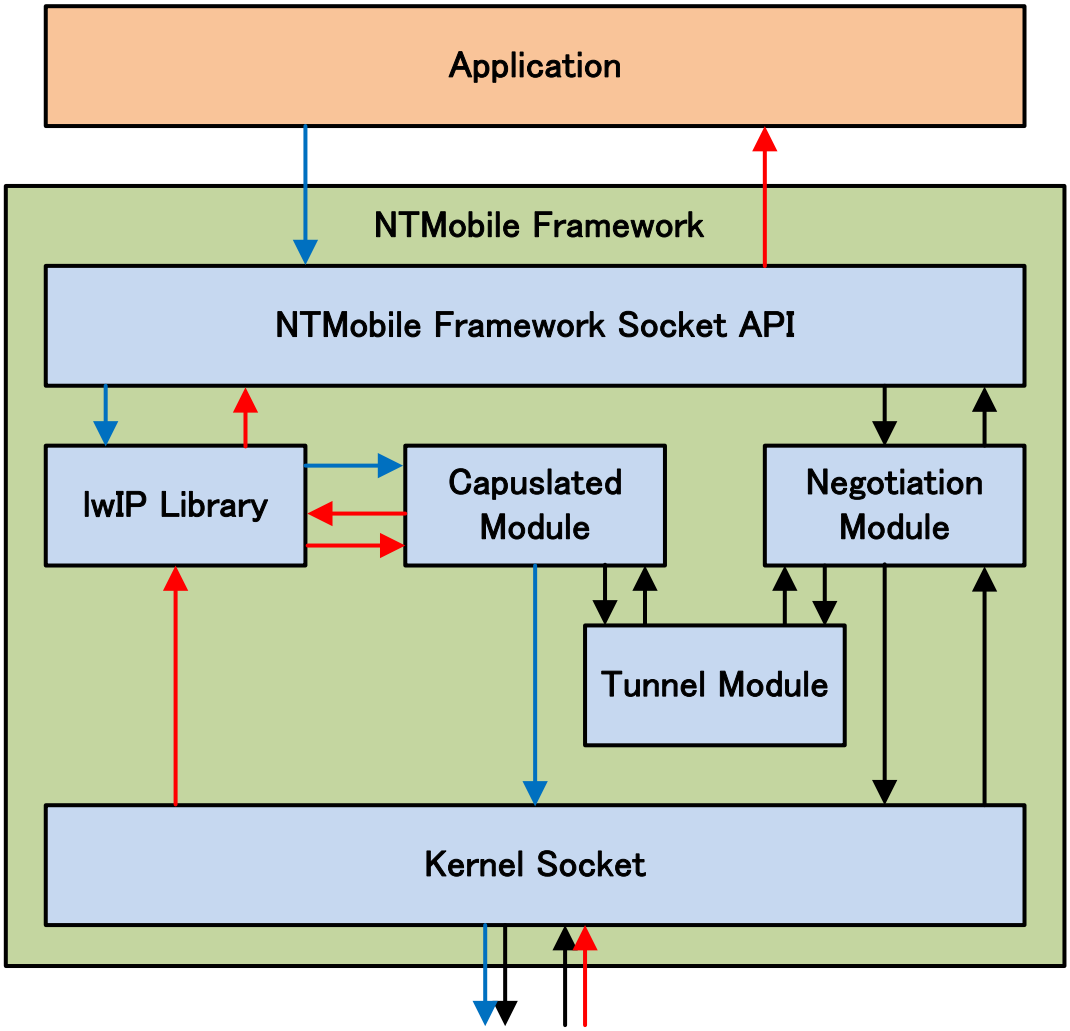

図 8　NTMobile フレームワークの構成

第4章　NTMobile用Javaラッパー

本章では，提案するNTMobile用のJavaラッパーについて述べる．NTMobileフレームワークをNTMobile用Javaラッパーにて JNA（Java Native Access）を使用し，呼び出す．呼び出し後は，Javaラッパー内でC言語とJavaでの変数の型の違いを変換によって処理し，違いを取り除く．アプリケーション作成者が可能な限りNTMobileを意識することなくJavaアプリケーションにてNTMobileを使用する為に，Java標準APIを使用することでNTMobileの通信を実現できるように実装する．

4.1　ラッパー構成

図9にNTMobile用Javaラッパーのパッケージ構成を示す．Javaラッパーは以下の3つのパッケージにより構成される．

(i) api
(ii) net
(iii) structure

(i) のapiパッケージには，NTMobileフレームワークに実装されているAPIをラップしたクラス群が格納されている．3.3.1，3.3.2項で述べた`ntmfw_ntm_init`関数，`ntmfw_getaddrinfo`関数に代替するメソッドはapiパッケージにて提供する．

(ii) のnetパッケージには，Java標準APIのDatagramSocketクラスをNTMobile通信用にするためのクラス群が格納されている．netパッケージに格納されているクラス群は，Java標準APIを使用した際に自動的に呼び出されるようにする為，Java標準APIにて用意されているDatagramSocketクラス[*1]における`setDatagramSocketImplFactory`メソッドで使用する．

(iii) のstructureパッケージには，NTMobileフレームワークで使用されている独自の構造体をJava用に定義したクラス群が格納されている．structureパッケージに格納されているクラス群はNTMobileフレームワーク独自の構造体と使い方が同じである．

4.2　モジュール構成

Javaラッパーを用いたNTM端末のモジュール構成を図10に示す．Javaアプリケーションにて使用するNTMobileフレームワークの機能は，`ntmfw_ntm_init`関数と`ntmfw_getaddrinfo`関数，UDPとTCP通信用の関数である．

[*1] https://docs.oracle.com/javase/8/docs/api/java/net/DatagramSocket.html

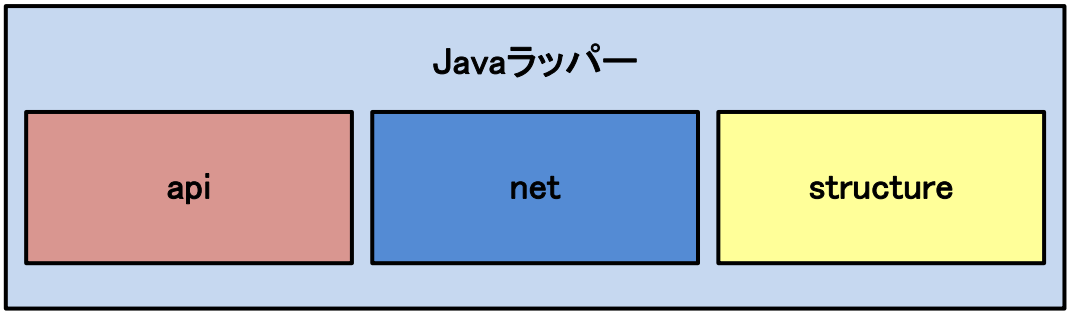

図 9　NTMobile 用 Java ラッパーのパッケージ構成

`ntmfw_ntm_init` 関数は `init` メソッドとして，`ntmfw_getaddrinfo` 関数は `getByName` メソッドとして api パッケージにて提供される．従って，Java アプリケーションが NTMobile フレームワークの `ntmfw_ntm_init` 関数，もしくは `ntmfw_getaddrinfo` 関数を使用したい場合は，Java アプリケーションにて Java ラッパーの api クラスに定義されている `init` メソッド，もしくは `getByName` メソッドを使用することで，必要な Java から C 言語への型の変換が終了後，JNA が NTMobile フレームワークの共有ライブラリをロードし，`ntmfw_ntm_init` 関数と `ntmfw_getaddrinfo` 関数を使用することができる．

通信関連の関数を使用する場合は予め Java 標準 API をオーバーライドすることで，Java アプリケーションが Java 標準 API の通信関連のメソッドを使用すると自動的に Java ラッパーにて既に定義済みの NTMobile 通信用のメソッドが呼び出され，必要な型の変換終了後に NTMobile フレームワークの該当する関数を使用することができる．

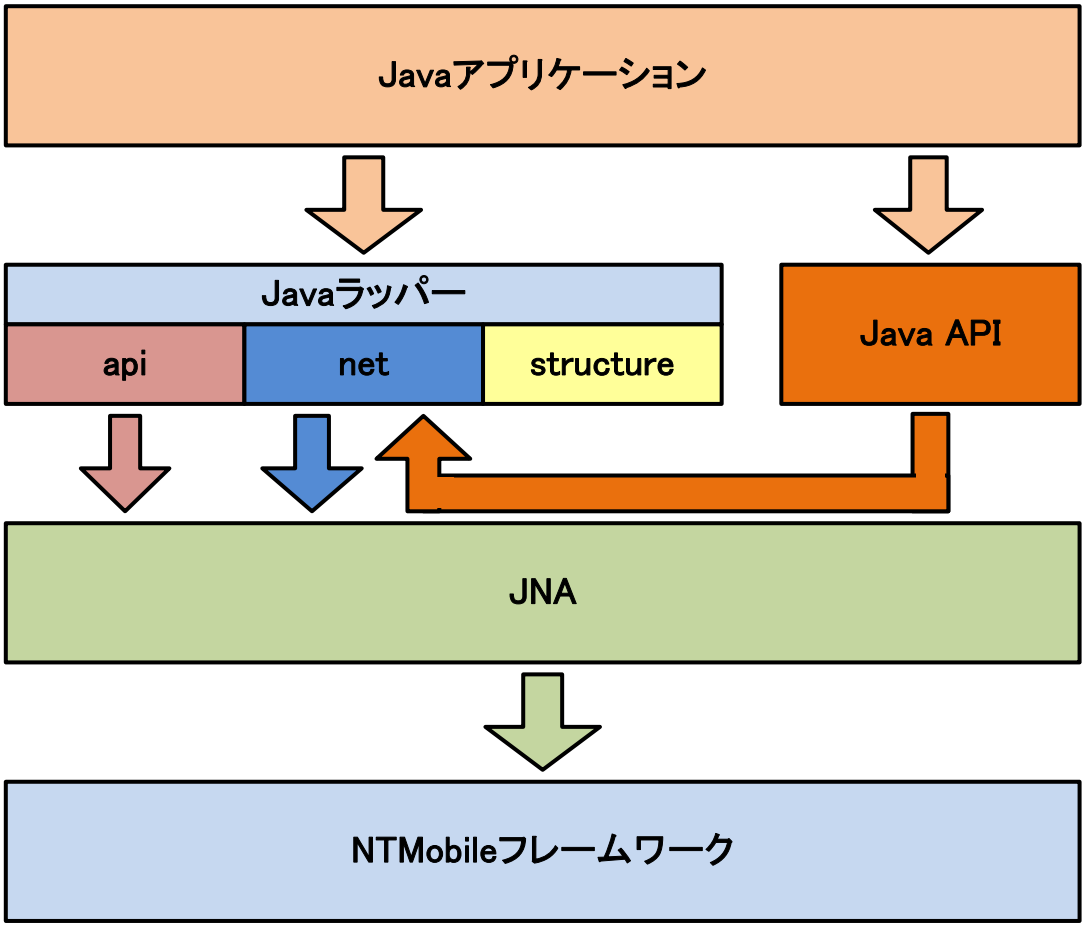

図 10　Java ラッパーを用いた NTM 端末のモジュール構成

第5章　評価

本章では，4 章にて実装した NTMobile 用 Java ラッパーの動作検証及び性能評価について述べる．NTMobile 用 Java ラッパーを NTMobile フレームワークの動作が確認済みの NTM 端末に対して実装を行った．NTMobile 用 Java ラッパーの評価として，NTMobile 通信を使用した場合の処理時間を測定した．

5.1　動作検証

表 1 にホストマシンの仕様，表 2 に NTM 端末の仕様，図 11 に動作検証を行ったネットワーク構成を示す．1 台のホストマシン上に VMware Workstation Player[*1] を用いて DC，MN，CN を構築した．DC，MN，CN を同一 IPv4 プライベートネットワークに接続した．この動作環境において，NTMobile 用 Java ラッパーが正常に動作することを確認した．

表 1　ホストマシンの仕様

	ホストマシン
OS	Windows 10 64bit
CPU	Intel Core i7-4770 3.40GHz
Memory	8.00GB

表 2　NTM 端末の仕様

	MN	CN
OS	Ubuntu 14.04 32bit	Ubuntu 14.04 32bit
Linux Kernel	3.13.0-24-generic	3.13.0-24-generic
CPU 割り当て	1Core	1Core
Memory 割り当て	2.00GB	2.00GB

5.2　性能評価

MN と CN の間でパケット送受信時における処理時間を計測し，性能評価を行った．表 3，4 に図 11 のネットワーク構成において実行した処理時間を示す．表 3 が送信，表 4 は受信に要した処

[*1] http://www.vmware.com

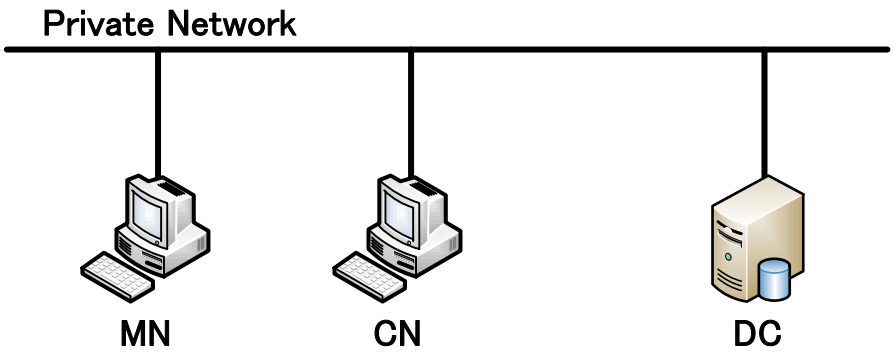

図 11　動作検証を行ったネットワーク構成

理時間の平均である．図 12 に表 3，4 の測定箇所を示す．処理に要した時間は MN，CN にて Java の System クラス [*2] における `nanoTime` メソッドにより取得した．MN から CN に対して通信を 7 回行い，最初の 2 回を除いた 5 回の平均時間を示す．最初の 2 回を除いたのは，Java では動的にクラスをロードする為，初回起動時等では立ち上がりが遅いからである．

表 3，表 4 の Linux は NTMobile を使用せず Java 標準 API にて用意されている UDP 通信用の DatagramSocket クラスの `send/receive` メソッドを使用し，送信/受信するのに要した時間である．この測定時間は，図 12 の Java ラッパーと NTMobile フレームワークを除いた処理と同等である為，Linux の処理時間であるとみなすことができる．

表 3　送信における処理時間の平均

測定箇所	NTMobile フレームワークなし [ms]	NTMobile フレームワークあり [ms]
Java ラッパー	-	0.13
NTMobile フレームワーク	-	2.91
Linux	0.07	0.07
合計	0.07	3.11

表 4　受信における処理時間の平均

測定箇所	NTMobile フレームワークなし [ms]	NTMobile フレームワークあり [ms]
Java ラッパー	-	0.16
NTMobile フレームワーク	-	2.37
Linux	0.01	0.01
合計	0.01	2.54

表 3 より，送信時に NTMobile フレームワークにおいて約 3.1 ミリ秒の時間を要し，NTMobile フレームワークが多くの時間を要していることが分かった．Java ラッパーにおける処理に要した

[*2] https://docs.oracle.com/javase/8/docs/api/java/lang/System.html

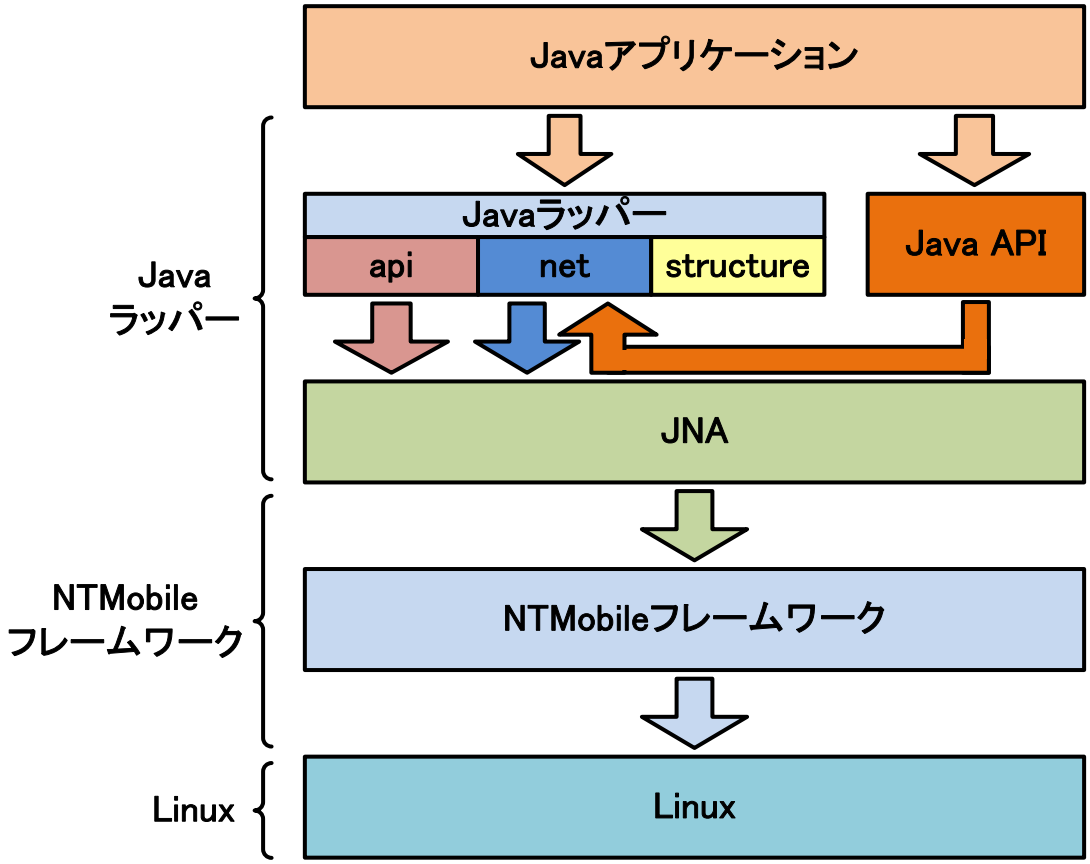

図 12　測定箇所

時間は約 0.13 ミリ秒であった．

表 4 より，受信時に NTMobile フレームワークにおいて約 2.5 ミリ秒の時間を要し，送信時と同様に NTMobile フレームワークが多くの時間を要していることが分かった．Java ラッパーにおける処理に要した時間は約 0.16 ミリ秒であった．

NTMobile フレームワークでの処理に多くの時間を要したのは，メッセージ全体の暗号化/復号処理が含まれる為と考えられる．

第6章　まとめ

本論文では，NTMobile 用 Java ラッパーの提案と実装を行った．Java において NTMobile フレームワークを用いて，Java 標準 API を使用して NTMobile 通信を可能とする Java ラッパーを提案した．これにより，Java において NTMobile 通信を使用可能になった．そして，Linux 上で NTMobile 用 Java ラッパーの実装を行い，動作検証を行った．Java 標準の通信と NTMobile 用 Java ラッパーを用いて通信を行った場合の処理時間の測定を行い比較するとともに，Java 標準 API を使用し NTMobile 通信を使用可能であることを確認した．

謝辞

本研究を進めるにあたり，多大なるご指導とご教授を賜りました，名城大学理工学部情報工学科　渡邊晃教授に心から感謝致します.

本研究を進めるにあたり，様々なご指導を賜りました，名城大学理工学部情報工学科　鈴木秀和准教授に深く感謝致します.

本研究を進めるにあたり，ご意見並びにご助言を賜りました，愛知工業大学情報科学部情報科学科　内藤克浩准教授に深く感謝致します.

本研究を進めるにあたり，様々なご助言並びにお力添えを賜りました，株式会社バレイキャンパスジャパン本社開発部　八里栄輔氏に深く感謝致します.

最後に，本研究を進めるにあたり，数々の有益なご助言を賜りました，渡邊研究室及び鈴木研究室の諸氏に感謝致します.

参考文献

[1] Perkins, C.: IP Mobility Support for IPv4, RFC 5944, IETF (2010).

[2] Perkins, C., Johnson, D. and Arkko, J.: Mobility Support in IPv6, RFC 6275, IETF (2011).

[3] Soliman, H.: Mobile IPv6 Support for Dual Stack Hosts and Routers, RFC 5555, IETF (2009).

[4] Rosenberg, J., Mahy, R., Matthews, P. and Wing, D.: Session Traversal Utilities for NAT(STUN), RFC 5389, IETF (2008).

[5] Mahy, R., Matthews, P. and Rosenberg, J.: Traversal Using Relays around NAT(TURN)): Relay Extensions to Session Traversal Utilities for NAT(STUN), RFC 5766, IETF (2010).

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

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

[8] 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).

[9] 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).

[10] 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).

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

研究業績

研究会・大会等

（1）清水一輝，納堂博史，鈴木秀和，内藤克浩，渡邊晃 : NTMobile で SIP 通信を可能とする仮想 IPv4 アドレス生成方式の検討，平成 28 年度電気関係学会東海支部連合大会論文集，Vol. 2016，講演番号 B2-5, Sep. 2016.