平成28年度　修 士 論 文

和文題目

ネットワークをフラット化する NTMobileフレームワークの実用化

英文題目

Practical Realization of NTMobile Framework that makes Flat Networks.

情報工学専攻　渡邊研究室

(学籍番号: 143430018)

納堂　博史

提出日: 平成29年1月30日

名城大学大学院理工学研究科

概要

近年のネットワーク利用の大半はモバイル通信であり，増大するネットワークトラフィックを Wi-Fi をはじめとする固定通信網にオフロードする動きが加速している．これらの固定通信網を用いて移動通信を行うためには，移動透過性技術の利用が必須である．また，IPv4 アドレスの枯渇に伴って普及しつつある IPv6 が IPv4 と互換性が無いため，両者間で通信接続性を実現する技術が求められる．NTMobile (Network Traversal with Mobility) は，IPv4/IPv6 混在環境において，移動透過性と通信接続性を同時に実現できる有用な技術である．NTMobile の実装は複数のモデルが提案されており，仕様の把握が困難であったことから，実用化に向けて仕様が統一された．実装モデルのうち，カーネル実装型については新しい仕様で実装と評価が行われているが，フレームワーク組込型については実装がされていない．本研究では，新しい仕様に基づきフレームワーク組込型の実装仕様を検討する．検討した実装仕様に基づき実装及び動作検証並びに評価を行う．動作検証及び評価を行った結果，実装したフレームワークがアプリケーションに組み込み可能なことを確認した．また，実用的な利用のため，C 言語で実装されたフレームワークを Java 及び Ruby から利用可能にするためのラッパーを設計及び実装し，異なるプログラミング言語間で通信を行うことが可能なことを確認した．

Abstract

The majority of network use in recent years is mobile communication, and the movement to offload increasing network traffic to fixed networks such as Wi-Fi is accelerating. In order to carry out mobile communication using these fixed networks, it is essential to use the IP mobility technology. Also, as IPv6, which is becoming popular due to exhaustion of IPv4 addresses, is incompatible with IPv4, a technology to realize communication connectivity between them is required. Network Traversal with Mobility (NTMobile) is a useful technology that can achieve network mobility and communication connectivity in both IPv4 and IPv6 networks. Several implementation models have been proposed for NTMobile's implementation, and because its specifications were complicated, specifications were unified for practical use. Of the implementation models, kernel module types are implemented and evaluated with new specifications, but framework built-in types are not implemented. In this paper, it was considered that the implementation specification of framework built-in type based on new specification. Perform implementation, operation verification and evaluation based on the examined implementation specifications. As a result of the operation verification and evaluation, it was confirmed that the implemented framework can be installed in the application. Also, it was designed and implemented a Wrapper to make the framework implemented in C language available from Java and Ruby, and confirmed that it is possible to communicate properly between different programming languages.

目 次

第 1 章　序論 **1**

第 2 章　関連研究 **3**

2.1 DSMIPv6 3

2.2 HIP 4

第 3 章　NTMobile **6**

3.1 NTMobile の概要 6

3.2 NTM 端末の実装モデル 8

第 4 章　framework の動作と実装 **9**

4.1 framework の動作 9

4.2 framework の実装 10

4.3 ラッパーの機能と実装 11

第 5 章　評価 **13**

5.1 動作検証 13

5.1.1 通信接続性試験 13

5.1.2 移動透過性試験 14

5.1.3 ラッパー試験 16

5.2 framework の課題 16

第 6 章　今後の展望 **18**

第 7 章　結論 **20**

謝辞 **21**

参考文献 **23**

研究業績 **25**

付 録 A　framework のトンネルテーブル仕様 **27**

付 録 B　NTM ソケット API **29**

付 録 C framework のカプセル化/デカプセル化処理 **31**
C.1 lwIP について 31
C.2 カプセル化/デカプセル化の実装 31

第1章　序論

スマートフォンを筆頭にモバイルデバイスやウェアラブルデバイスの普及に伴い，その通信量は増加の一途を辿っている [1]．今日において，ネットワークの通信プロトコルは専ら IP (Internet Protocol) である．IP はバージョンによって互いに互換性のない IPv4 (Internet Protocol version 4) と IPv6 (Internet Protocol version 6) が利用されており，両者が混在している．IPv4 アドレスの枯渇に伴い，ほぼ無限のアドレス空間を有する IPv6 が徐々に普及してきているが，特に小規模 ISP (Internet Service Provider) においては IPv6 へ対応しないことが決定している等，普及には時間を要すると見込まれる [2]．また，インターネット利用者の 6 割以上がモバイルデバイスによる通信を行っており，特にインターネット利用時間の多い若年層では，この割合が 8 割にも達する．しかし，スマートフォンの IPv6 対応が遅れているとの調査報告があり [2]，今後しばらくは IPv4 ネットワークの利用が続き，IPv4/IPv6 の両者が混在するネットワーク環境が維持されることが推測される．

IP ネットワークにおける課題は移動透過性と通信接続性の 2 つに分類できる．前者は，IP ネットワークにおける端末識別子である IP アドレスが位置識別子を兼ねているため，接続するネットワークを切り替えると IP アドレスが変化し，構築済みの通信セッションが全て切断されるという問題である．後者は，IPv4 ネットワークと IPv6 ネットワークに互換性が無いことに起因する．IPv4 アドレスしか持たない通信端末は IPv6 ネットワークに参加できず，その逆も同様である．このため，IPv4 ネットワークに参加している通信端末と IPv6 ネットワークに参加している通信端末間では通信を行うことができない．また，IPv4 ネットワークにおいては，IPv4 アドレスの枯渇対策として導入された NAT (Network Address Translation)[⋆1] が通信経路上に存在すると，NAT の外側から通信を開始できないという NAT 越え問題が通信接続性の課題として挙げられる．

移動透過性の問題に関しては，従来より多くの技術が研究されてきた [3–5]．しかし，これらの技術は IPv6 ネットワークを前提にしており，通信経路上に NAT がある IPv4 ネットワークには適用できないほか，IPv4 ネットワークで利用できる場合であっても，常に冗長経路となるものや移動先が限定されるといった課題がある．通信接続性の問題については，IPv4 ネットワークにおける NAT 越え問題を解決する技術が研究されてきた [6–9]．しかし，これらの技術では IPv4 ネットワークと IPv6 ネットワーク間の通信接続性を確保できない．近年においては，IPv4 ネットワークと IPv6 ネットワーク間で通信を可能とする技術の研究が進んでいるものの，既存インフラ設備に改造を要するため利用できるネットワークが限定される，通信端末の移動を想定していないといった課題がある [10]．

これらの技術は移動透過性と通信接続性の課題を個別に解決するものであり，同時に解決するこ

[⋆1] NAPT (Network Address Port Translation) を含む．

とはできない．DSMIPv6 (Dual Stack Mobile IP version 6) [11] や，HIP (Host Identity Protocol) [12] は，異なるネットワーク間の通信接続性と移動透過性を実現しているが，通信開始時のオーバーヘッドが大きい，NAT に改造を有するため移動できるネットワークが限定される，通信経路が冗長となるといった課題がある．また，これらの技術はカーネル空間における実装であり，市販のスマートフォンへの適用には root 権限が必要となり現実的でない．

NTMobile (Network Traversal with Mobility) は，IPv4/IPv6 ネットワークにおいて，移動透過性と通信接続性を同時に実現する技術として提案されている [13–16]．NTMobile によると，アプリケーションは移動端末 (以後，NTM 端末) の属しているネットワークに依存せず，移動によって変化しない一意の仮想的な IPv4 アドレスと IPv6 アドレスを取得する．この仮想 IPv4/IPv6 アドレスを用いて通信することにより，アプリケーションは端末の属するネットワークや移動を意識する必要がない．実際の通信は実 IP アドレスでカプセル化され，原則として直接通信相手に転送される．このとき，パケットの盗聴や改竄防止のため，暗号化及びメッセージ認証符号の付与が施される．このように，NTMobile を利用することで，アプリケーション開発者は独自のサーバーを構築することなく，常にセキュアなエンドツーエンド通信を行うアプリケーションを開発することができる．

NTMobile においては，複数の実装方式が混在しており，その仕様の把握が困難な時期があった．そこで，実用化に際してこれらの仕様を統一的な枠組みとして再定義した [17]．このとき，アプリケーションレベルで実装が可能となることを強く意識した．各実装方式で異なる処理を実現するため，NTMobile のメッセージ仕様を拡張し，仕様が確定していなかった NTMobile の装置間の信頼関係を担保する仕組みを定義した．また，OpenID や公開鍵による相互認証といった，NTMobile を実用化するにあたって出された新たな要求仕様を満たすよう留意した．

NTMobile の実装方式のうち，カプセル化処理を Linux カーネル空間で行うカーネル実装型は実装されているが，カプセル化処理をアプリケーション層で行うフレームワーク組込型 (framework) は実装がない．framework を前提とした新たな実装モデルの研究も進んでおり，framework の実装が急務である．そこで，本論文では，統一的な枠組みに基づいて framework の実装を行い，通信を行う NTM 端末のペアが IPv4/IPv6 のどのようなネットワークに属しているかに関係なく，アプリケーションが IPv4/IPv6 の各暗号化通信を行うことを可能とする．このとき，NTM 端末が移動しても通信が継続することを保障し，移動に伴う通信断絶時間が既存の実装と比して同程度となるよう留意する．また，実装にあっては，framework を異なるプログラミング言語において利用できることを示すため，Java と Ruby でラッパーを開発し，互いに通信できることを示す．複数のプログラミング言語によるアプリケーション開発者向けのライブラリを提供可能とすることで，NTMobile の利用に際する敷居を低くし，実用化に供する．

以降，第 2 章において，NTMobile と同じく IPv4/IPv6 ネットワーク間で移動透過性と通信接続性を実現する関連研究について述べる．第 3 章で複数の実装モデルにおいて統合された NTMobile について説明する．第 4 章で framework 動作と実装について説明し，動作検証と評価について第 5 章で述べる．NTMobile の今後の展望について 6 章で述べ，最後に第 7 章でまとめる．

第2章　関連研究

本章では，NTMobile と同様に，IPv4/IPv6 ネットワーク間で通信接続性と移動透過性を実現する研究について述べる．

2.1　DSMIPv6

DSMIPv6 は，Mobile IPv6 [18] を IPv4/IPv6 混在環境に適用したもので，IPv4/IPv6 デュアルスタックネットワークに設置する HA (Home Agent) と移動端末 MN (Mobile Node) がトンネルを構築する．DSMIPv6 の概要を図 1 に示す．MN はホームネットワークで取得する HoA (Home Address) と移動先のネットワークで取得する CoA (Care of Address) の 2 種類の IP アドレスを持つ．MN がどのようなネットワークに移動しても HoA は変化せず，CoA のみが変化する．通信相手端末 CN (Correspondent Node) と MN 間の通信は HoA が用いられるため，CoA の変化を隠蔽できる．CN が送信する HoA 宛てのパケットは HA が受信し，トンネルを通じて MN へ届けられる．MN から

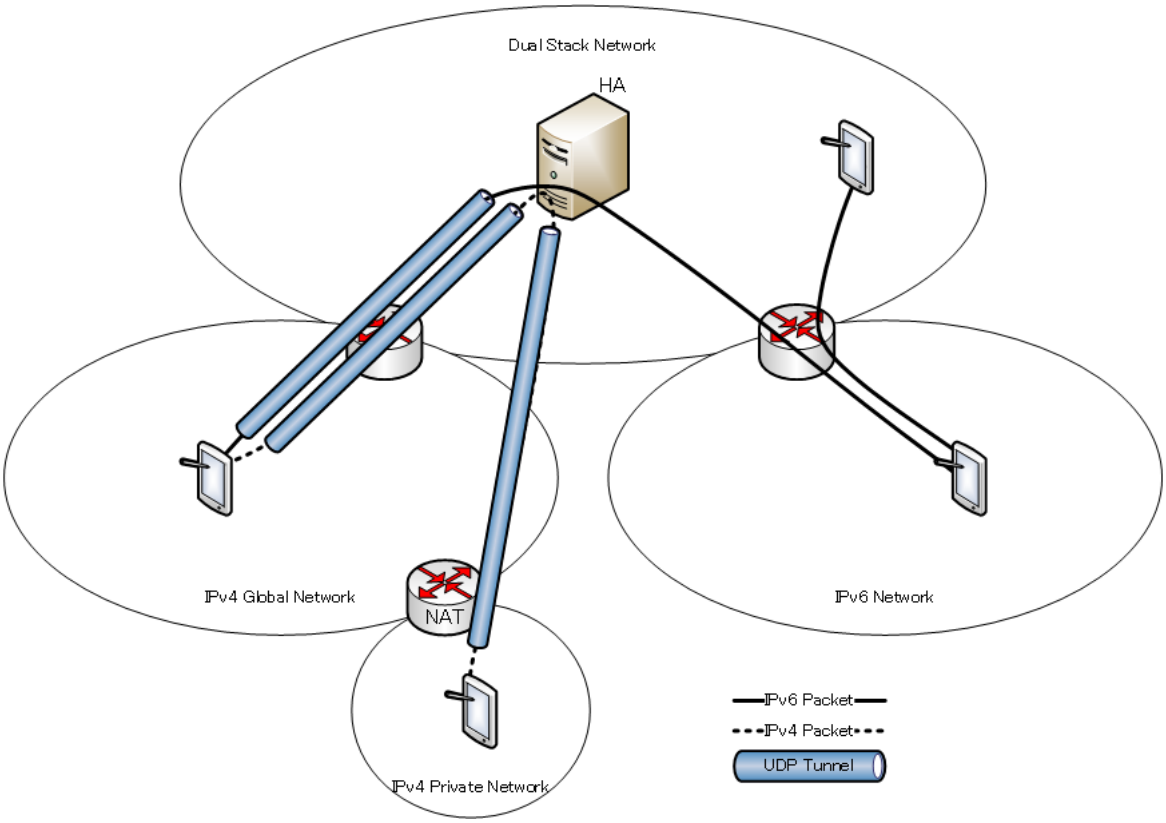

図 1　DSMIPv6 の概要

CN 宛てのパケットはトンネルを通じて HA に転送され，HA から CN に送信される．

以上のように，DSMIPv6 を用いることで IPv4/IPv6 間で通信接続性と移動透過性を実現することができるが，次のような課題がある．

- 通信経路の冗長化
  MN と CN が IPv6 を利用可能な場合を除き，通信経路が HA 経由となり冗長である．また，IPv4 アプリケーションを利用する場合，必ず HA 経由となる．
- グローバル IPv4 アドレスの確保が困難
  MN と CN 間で双方向の通信接続性を維持するため，移動端末にグローバル IPv4 アドレスを HoA として割り当てる必要がある．グローバル IPv4 アドレスが枯渇している今日において，この手法は現実的でない．

2.2 HIP

HIP は，IP アドレスが持つ端末識別子と位置識別子の役割のうち，端末識別子を分離する．エンド端末における HIP のレイヤーモデルを図 2 に示す．OSI 参照モデルにおけるネットワーク層とトランスポート層との間に新たに HIP 層を定義し，端末識別子として HI (Host Identifir) を用いる．HIP 層において IP アドレスと HI のマッピングを管理し，上位層では HI を用いて通信を行う．IP アドレスは位置識別子の役割のみを担い，移動によって変化するが，HI は変化しない．アプリケーションは HI を識別子に通信を行うため，IP アドレスが変化しても通信を継続することができる．IPv4/IPv6 に関係なく HI は同一であることから，IPv4/IPv6 間においても通信接続性と移動透過性を実現できる．

HIP では ICE (Interactive Connectivity Establishment) [6] により常に最適な経路で通信を行うこと

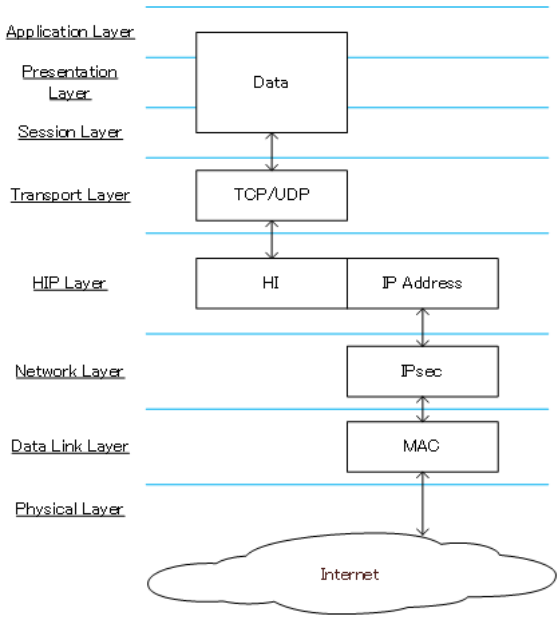

図 2　HIP のレイヤーモデル

が可能であるが，ICE のシグナリングに要するオーバーヘッドが高い問題がある [19]．また，トランスポート層とネットワーク層の間に HIP 層を設ける構造上，HIP の利用に root 権限を要するほか，ICE や IPsec を始めとする関連技術を多く導入する必要があることから，センサデバイス等の小型端末への適用が考慮されていない．特に市販のスマートフォンに適用する場合，root 権限を利用するとメーカーサポートを受けられなくなるため，現実的でない．

第3章　NTMobile

本章では，統合的枠組みとして統一された NTMobile について述べる．

3.1　NTMobile の概要

図 3 に NTMobile の概要を示す．NTMobile は下記に示す機器で構成される．NTM 端末を除く装置群は，予め公開鍵証明書を取得している．NTMobile の制御メッセージに用いる共通鍵は，この公開鍵を用いて安全に共有され，有効期限が切れるまで利用される．共通鍵の有効期限が切れた場合，公開鍵を用いて新しい共通鍵が共有される．NTM 端末が公開鍵証明書を所有することも可能であり，NTM 端末同士で TLS (Transport Layer Security) 相互認証を行うような，高いセキュリティを求められる場合に用いられる．

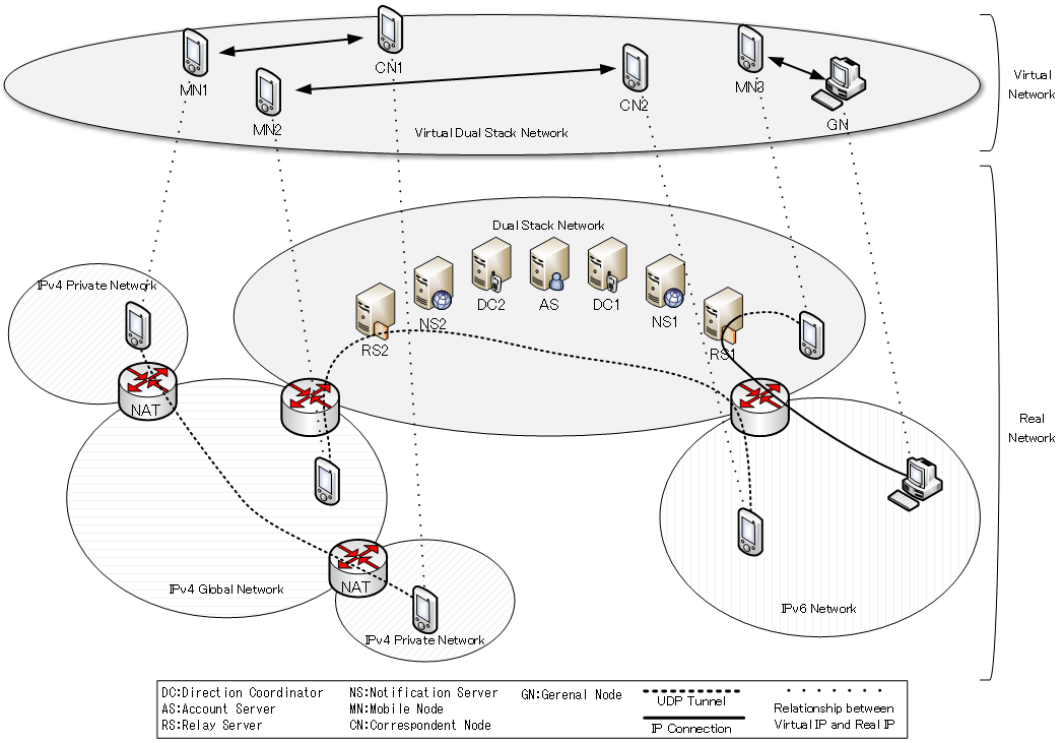

図 3　NTMobile の概要

- DC (Direction Coordinator)
  NTM 端末の仮想 IP アドレスや位置情報等を管理し，UDP トンネルの構築指示を出す機器．

- AS (Account Server)
  NTM 端末の認証や，DC と NTM 端末間等における通信の暗号化に用いる共通鍵の配布を行う唯一の機器.
- RS (Relay Server)
  NTM 端末間でエンドツーエンド通信ができない場合や，一般端末 (GN:General Node) との通信の際にパケットを中継する機器.
- NS (Notification Server)
  NAT 越えを実現するために送受信するキープアライブパケットを劇的に削減するオプション装置.
- NTM 端末
  NTMobile の機能を有するエンド端末.

NTM 端末は，初回起動時に AS にログインし，DC の FQDN 及び DC との通信に用いる共通鍵を取得する．このとき，予め AS に端末情報が登録されている必要がある．基本機能としてメールアドレスおよびパスワード認証を提供するが，公開鍵や OpenID による外部認証もサポートする [20,21]．その後，DC に実 IP アドレス等の情報を登録し，仮想 IPv4/IPv6 アドレスを取得する．以降，定期的に DC とキープアライブを行うことで，NTM 端末と DC 間で常にパケットの送受信を行うことができる状態を維持する．なお，NS や APNS (Apple Push Notification Service)/GCM (Google Cloud Messaging) を利用することにより，DC とのキープアライブを省略することが可能である [22,23].

通信を開始するときは，NTM 端末 MN が行う，通信相手端末 CN の名前解決 (DNS 要求) がトリガとなり，NTMobile の処理が開始される．MN は内部で DNS 要求をフックし，CN の FQDN (Fully Qualified Domain Name) を含めた経路指示要求を MN を管理する DC (DCmn) に送信する．DCmn は，この FQDN を用いて CN を管理する DC (DCcn) を探索し，CN が NTM 端末か一般端末かを判別する．CN が NTM 端末の場合，DCcn から CN の実 IP アドレス等の情報を取得し，経路指示を MN 及び CN に行う．このとき，CN は DCcn と常に通信可能であることから，DCcn 経由で経路指示を行う．経路指示パケットには MN と CN で UDP トンネルを構築するときに用いる共通鍵を生成して格納する．MN と CN は経路指示パケットの指示に従って MN-CN 間で UDP トンネルを構築する．MN は DNS 応答として CN の仮想 IP アドレスを返すことで，アプリケーションは仮想 IP アドレスを用いた通信を行う．なお，MN と CN が直接通信できない場合，DCmn から RS へ中継指示を行い，MN 及び CN は RS 経由の UDP トンネルを構築する．このとき，RS は MN と CN の位置に応じて適切な RS が選択される [24] ほか，直接通信可能であれば経路最適化機能によって即座にエンドツーエンド通信に切り替わる．CN が一般端末の場合は MN と RS が UDP トンネルを構築し，RS は NAT 処理を行って CN との通信を中継する.

3.2 NTM 端末の実装モデル

NTM 端末の実装は，Linux カーネルモジュールで動作するカーネル実装型，アプリケーションレベルで動作するフレームワーク組込型，VPN (Virtual Private Network) サービスとして動作する VpnService 利用型 [25]，TUN/TAP 利用型 [26] がある．これらの実装モデルのうち，カーネル実装型は新しい仕様で実装を終えている．また，市販のスマートフォンに適用できるのは root 権限を要しないフレームワーク組込型と VpnService 利用型に限定される．

- カーネル実装型
  カーネル空間でパケットのカプセル化/デカプセル化処理を行うことにより，高速に動作する．LinuxOS で動作するため，Linux デスクトップのほか，スマートフォンに用いられている AndroidOS に適用でき，アプリケーションの改造が不要という大きな利点がある．しかし，WindowsOS や iOS には適用できず，カーネルモジュールを利用するためには Linux の root 権限を要するため，市販のスマートフォンに適用するための敷居が高い．
- フレームワーク組込型
  NTMobile の機能をすべてアプリケーションレベルで実装する．アプリケーション開発者は，通常のソケット API (Application Programming Interface) を用いる代わりに，framework の提供するソケット API (NTM ソケット API) を利用することにより，NTMobile を利用できる．動作する OS は Linux に限らず，Windows や iOS にも適用でき，root 権限も不要である．しかし，アプリケーションごとにフレームワークを組み込むための改造が必要である．
- VpnService 利用型
  AndroidOS の提供する VPN サービスを利用する実装方式である．framework を VPN サービスとして利用することにより，root 権限を要さず，アプリケーションの改造も不要となる．なお，従来は AndroidOS でのみ利用可能であったが，最新の iOS 端末において VPN サービスを確認できており，今後 AndroidOS 端末に限らず iOS 端末にも適用可能範囲が拡大する見込みである．
- TUN/TAP 利用型
  VpnService 利用型を拡張し，Android の VPN サービスを利用する代わりに TUN/TAP インターフェースを利用する方式である．Linux 系 OS 以外でも利用可能である [*1] が，TUN/TAP デバイスの生成には root 権限を要する．

[*1] Linux のほか，macOS，Windows，FreeBSD，Solaris 等で利用できる．

第4章　frameworkの動作と実装

本章では，本論文で取り扱う framework について詳記し，その実装について述べる．

4.1　frameworkの動作

framework の概要を図 4 に示す．framework は，仮想 IP プロトコルスタック [*1] の機能を内包しており，このプロトコルスタックの持つ仮想ネットワークインターフェースに仮想 IP アドレスが割り当てられる．framework 自身は OS 標準のソケット API(以下，単にソケット API という．) を利用してパケットの送受信を行う．アプリケーションは，NTM ソケット API を利用してデータの送受信を行う．アプリケーションが送信したデータは，仮想 IP プロトコルスタックの処理により，仮想 IP アドレスを用いて TCP (Transmission Control Protocol) または UDP (User Datagram Protocol) ヘッダ及び IP ヘッダが付与される．このパケットは暗号化，MAC (Message Authentication Code) 付与等の処理の後，ソケット API を用いて UDP 送信される．この処理により，NTM ソケット API で送信されたパケットは UDP でカプセル化されて送信される．カプセル化されたパケットを framework がソケット API で受信すると，この時点で外側の UDP/IP ヘッダが外される．受信したパケットの MAC 検証，復号を行い，仮想 IP プロトコルスタックに戻すことで，内側の IP ヘッダ及び TCP または UDP ヘッダが除去され，アプリケーションはデータを受信できる．

framework は，初期化されるときに，仮想ネットワークインターフェースに対して仮想 IPv4 アドレスおよび仮想 IPv6 アドレスを登録する．また，端末の属するネットワークに応じて，ソケット API を用いて IPv4 ソケットまたは IPv6 ソケット若しくはその両方 [*2] を開き，IPv4 または IPv6 で通信可能な状態を維持する．framework は端末の NIC (Network Interface Card) に割り当てられている IP アドレスを監視しており，この IP アドレスの変化を移動として認識する．移動を検知した場合，開いていた IPv4 ソケット及び IPv6 ソケットを一度閉じ，再度開きなおす．その後，新しい実 IP アドレスの登録とトンネル再構築を行う．一方で，仮想 IP プロトコルスタックは，仮想 IP アドレスが変化しないため，実 IP アドレスの変化をアプリケーションに通知しない．

以上の処理により，アプリケーションは NTM ソケット API を用いることにより仮想 IP アドレスによるパケットの送受信が可能であり，実 IP アドレスに依存せずに通信を行うことができる．

[*1] TCP/IP の実装で，UDP や ICMP (Internet Control Message Protocol) の送受信も可能である．

[*2] 端末が IPv4 ネットワークに接続しているときは IPv4 ソケットのみ，IPv6 ネットワークに接続しているときは IPv6 ソケットのみを開き，IPv4/IPv6 デュアルスタックネットワークに接続している時は両方のソケットを開く．

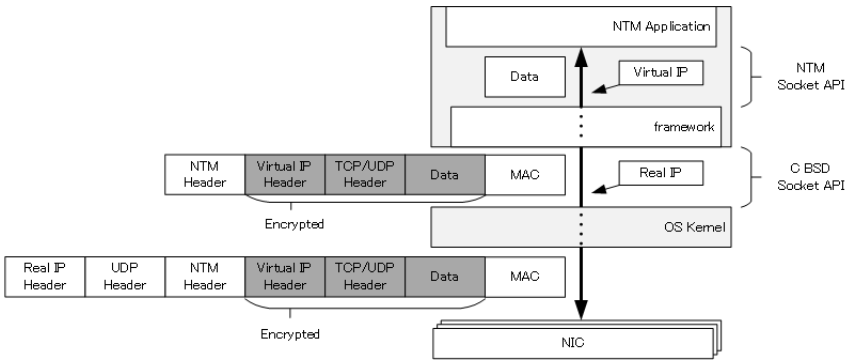

図 4　framework の概要

4.2　framework の実装

framework のモジュール構成を図 5 に示す．framework は次のモジュールから構成される．

- NTM ソケット API
  BSD ソケット API に代わってアプリケーションに提供するソケット API で，framework 独自の API と名前解決を担う API を除き，仮想 IP スタックに処理を渡す．名前解決を担う API は，引数から FQDN を抽出してトンネル構築を行い，当該 FQDN に対応する仮想 IP アドレスを返す．仮想 IP アドレスは，引数に応じて仮想 IPv4/IPv6 アドレスの片方もしくは両方を返す．
- BSD Socket API
  framework がパケットの送受信を行うために用いる C 言語標準ソケット API で，制御メッセージやカプセル化パケットはすべてこの API で送受信される．なお，ネゴシエーションモジュールがアドレス情報の管理を行う際もこの API が用いられる．
- 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．NTM ソケット API の名前解決を担う関数が呼び出された場合や，他端末から通信要求があった場合，このモジュールで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 アドレスを毎秒確認し，IPv4 アドレスまたは IPv6 アドレスに変化があった場合，DC に対してアドレス情報の更新を行い，構築されている全てのトンネルを再構築する．
- パケット処理モジュール
  パケットの MAC 付与/検証及び暗号化/復号を行い，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 スタックとに処理を振り分ける．また，BSD Socket API を用いたパケットの送受信を行う．
- 仮想 IP プロトコルスタック
  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．TCP/IP スタックとして lwIP (A Lightweight TCP/IP stack)[*3] を用いている．アプリケーションが送信するデータは，lwIP によって仮想 IP ヘッダが付与され，アウトプットコールバック関数によってパケット処理モ

[*3] http://savannah.nongnu.org/projects/lwip/

ジュールに処理が移る．また，パケット処理モジュールから受信したパケットは，lwIP のインプット関数に処理が渡される．

- トンネルテーブル
  通信相手ごとに FQDN，仮想 IPv4/IPv6 アドレス，実 IPv4/IPv6 アドレス，共通鍵，PathID[*4]，NodeID[*5]，RS の実 IPv4/IPv6 アドレス等をメンバとするエントリ持つ．複数のキーを持つハッシュテーブルで実装され，ハッシュキーは FQDN，仮想 IPv4/IPv6 アドレス，PathID，NodeID である．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．

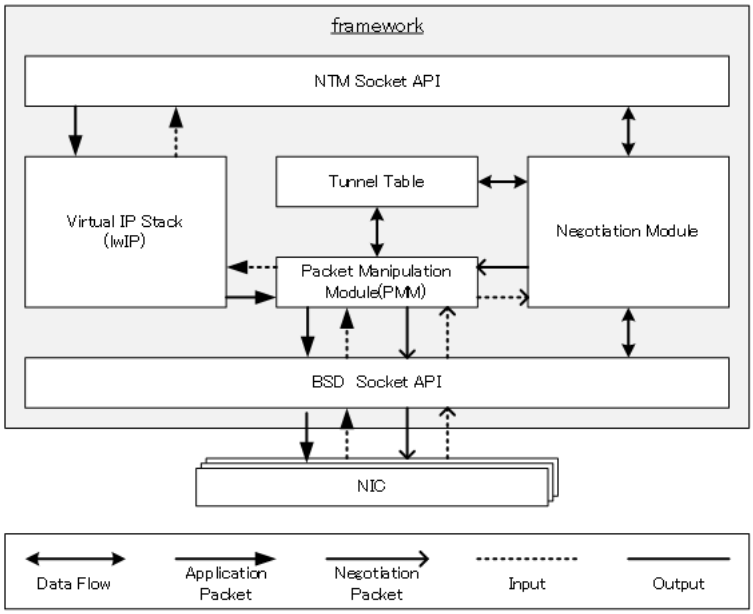

図 5　framework のモジュール構成

4.3　ラッパーの機能と実装

framework を C 言語以外から利用可能であることを示すため，Java ラッパー及び Ruby ラッパーを設計し，実装する．

Java ラッパーの実装モデルを図 6 に示す．Java アプリケーションから framework を利用するため，JNA (Java Native Access)[*6] を用い，C 言語で記述された NTM ソケット API を呼び出すラッパークラスを定義する．開発者は，ラッパークラスのメソッドを利用してデータの送受信を行うことで，NTMobile の機能を利用することができる．また，これらのメソッドを利用して Java のソケットクラスを継承するサブクラスを定義することも可能である．

次に，Ruby ラッパーの実装モデルを図 7 に示す．Ruby アプリケーションから framework を利用するため，Ruby 拡張ライブラリ [*7] を作成し，C 言語で記述された NTM ソケット API を Ruby

[*4]通信相手ごとに生成される一意の値で，カプセル化パケットに格納される．
[*5]NTMobile において端末を識別する一意の値で，各種制御パケットに格納される．
[*6]`https://github.com/java-native-access/jna`
[*7]`https://docs.ruby-lang.org/en/2.1.0/README_EXT_ja.html`

から利用できるようにする．また，Ruby で TCP 通信を行う際に利用されるクラス [*8] を継承し，framework を利用した TCP 通信を行うラッパークラスを定義する．開発者は，TCPServer クラスまたは TCPSocket クラスを用いる代わりに，NTMTCPServer クラスまたは NTMTCPSocket クラスを用いることで，NTMobile の機能を利用することができる．なお，それぞれのクラスのコンストラクタ [*9] において，framework の初期化に必要な引数を必要とする．

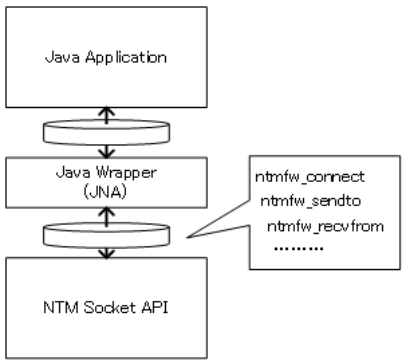

図 6　Java ラッパー

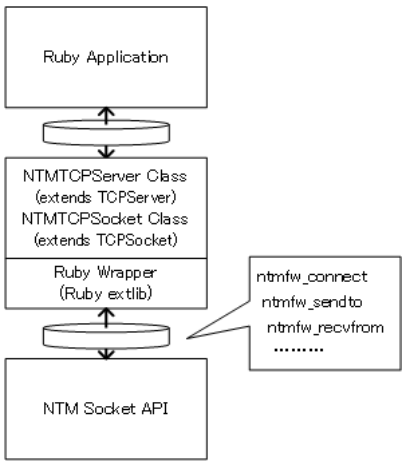

図 7　Ruby ラッパー

[*8] サーバーアプリケーションは TCPServer クラスを，クライアントアプリケーションは TCPSocket クラスを利用する．

[*9] クラスのオブジェクトを生成するときに呼び出される初期化メソッドのこと．Ruby においては initialize メソッドが該当する．

第5章　評価

5.1　動作検証

実装した framework を評価するため，仮想マシンを用いて仮想 IP アドレスによる UDP 及び TCP の疎通試験 (通信接続性試験) 及び移動試験 (移動透過性試験) を行った．また，実装したラッパーを用いて異なるプログラミング言語で実装されたアプリケーション間で疎通試験 (ラッパー試験) を行った．

5.1.1　通信接続性試験

NTM ソケット API を用いた試験プログラムを作成し，NTM 端末 MN と NTM 端末 CN 間でパケットが送受信可能であることを確認した．試験プログラムは，UDP 及び TCP の IPv4 ソケットと IPv6 ソケットを生成し，各ソケットで送受信を行う．試験ネットワークを図 8 に示す．また，試験に用いた機器の諸元を表 1 に示す．試験では，1 つのホストマシン上に仮想マシンを 5 台構築し，それぞれを AS，DC，RS，MN，CN とした．ホストマシンを 1Gbps の IPv4/IPv6 デュアルスタックネットワークに接続し，AS，DC，RS はブリッジ接続する．図 8 では MN が NAT 接続，CN がブリッジ接続しているが，試験区分に応じて MN 及び CN はブリッジ接続または NAT 接続する．なお，NAT 接続する場合，仮想ネットワークは IPv4 のみ有効となる．

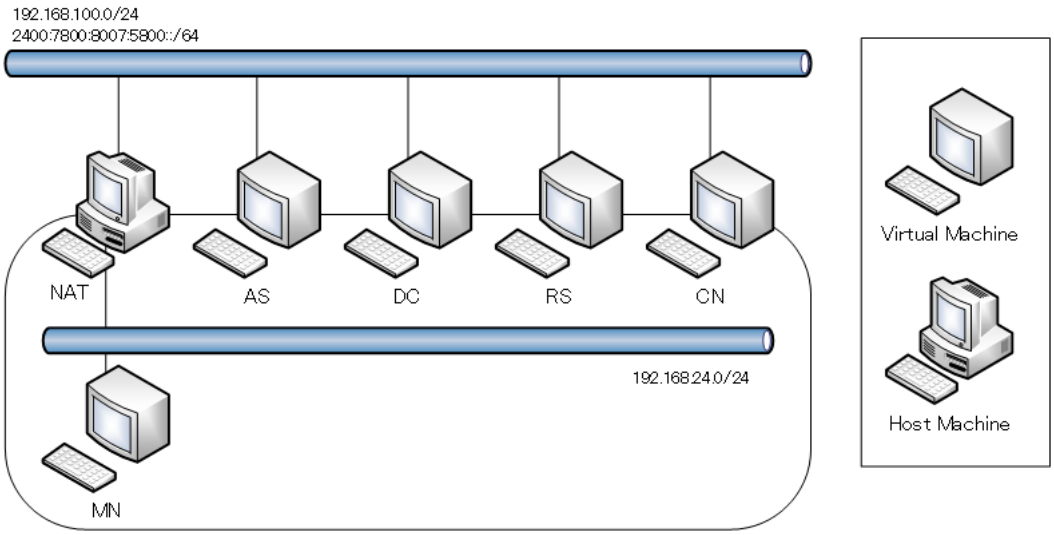

図 8　試験ネットワーク

事前準備として，AS に DNS サーバーを構築し，各機器の A レコード及び AAAA レコードを登

表 1　諸元

	Host Machine	Virtual Machine (MN,CN,RS,AS,DC)
OS	Windows 10 Pro x64	Ubuntu14.04 LTS x86
CPU	intel 4core8threads 3.50GHz	allocate 2core2threads
Memory	16GB	allocate 2GB
Virtualization Software	VMware Workstation 10.0	-

録し，各機器のプライマリ DNS サーバーを AS に設定した．AS には予め MN と CN の FQDN 及びメールアドレス並びにパスワードを登録し，AS，DC，RS には公開鍵証明書を配布した．

試験は，MN 及び CN を IPv4 グローバルネットワーク，IPv4 プライベートネットワーク，IPv6 ネットワークのいずれかに接続し，全ての組み合わせで疎通試験を行った．IPv4 グローバルネットワークに接続する場合は IPv6 を無効化してブリッジ接続し，IPv4 プライベートネットワークに接続するときは NAT 接続した．IPv6 ネットワークに接続する場合は IPv4 を無効化してブリッジ接続した．

試験結果を表 2 に示す．試験の結果，アプリケーションは端末の属するネットワークに依存せず，IPv4 パケット及び IPv6 パケットの送受信を行うことができた．また，構築されたトンネルは常に最適経路であることが確認できた．なお，試験ネットワークでは通信ペアが同一 NAT 配下に属するが，異なる NAT 配下に属している場合でも経路最適化機能によってエンドツーエンド通信が可能である [16].

表 2　疎通試験結果

		CN		
		IPv4	IPv4 NAT	IPv6
MN	IPv4	◎	◎	○
	IPv4 NAT	◎	◎	○
	IPv6	○	○	◎

◎:end-to-end　○:via RS

5.1.2　移動透過性試験

framework による移動透過性の試験のため，MN と CN に実機を用いた移動試験を行った．試験ネットワークおよび各機器の諸元を図 9 及び表 3 に示す．1 台のホストマシン上に仮想マシンとして AS，DC を構築し，MN と CN をそれぞれ別の物理マシンに構築した．試験ネットワークは 1Gbps の IPv4 ネットワークで，AS，DC をブリッジ接続した．また，試験ネットワークに 2 台の NAT 機器を接続し，CN を試験ネットワークに，MN を NAT 配下に接続した．

試験は，MN-CN 間でトンネル通信を行っている途中で，MN の移動を行った．移動は，一方の NAT に有線接続している MN の有線ケーブルを，もう一方の NAT に差し替えることによって行った．移動試験を 10 回繰り返し，wireshark[⋆1] を用いて MN 及び CN のパケットをキャプチャした．

[⋆1] http://www.wireshark.org/

MN はネットワークの変化を検出すると，DHCP (Dynamic Host Configuration Protocol) による IP アドレスの取得を行うことから，移動処理開始時刻を最初の DHCP パケット送信時刻とした．また，本試験における移動後のトンネル構築は，MN が CN へトンネル要求パケットを送信し，CN が MN へトンネル応答パケットを送信することで完了することから，移動処理終了時刻を MN がトンネル応答パケットを受信した時刻とした．ここで，移動処理開始時刻から移動処理終了時刻までに要した時間を移動処理時間とする．なお，framework は IP アドレスの変化を検知すると DC に対して端末情報登録パケットを送信するため，移動処理開始時刻から当該パケット送信時刻までに要した時間を IP アドレスの更新に要する時間とし，当該パケット送信時刻から移動処理終了時刻までに要した時間をトンネル再構築に要する時間とする．

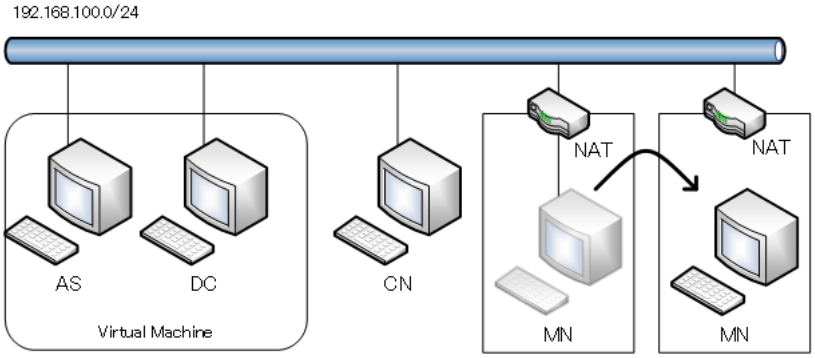

図 9　試験ネットワーク

表 3　諸元

	AS	DC	MN	CN
OS	Ubuntu12.04LTS x86	Ubuntu12.04LTS x86	Ubuntu14.04LTS x86	Ubuntu14.04LTS x86
CPU	VM 1core (Intel) 3.40GHz	VM 1core (Intel) 3.40GHz	Intel 2core4thread 3.40GHz	Intel 2core4threads 2.80GHz
Memory	2GB	3GB	8GB	8GB

試験結果を表 4 に示す．試験の結果，IP アドレスの更新に要する時間が支配的であり，トンネルの再構築に要する時間は安定して少ないことがわかる．従前の実装に係る評価試験から，IP アドレスの更新からトンネル再構築まで約 1 秒〜8 秒を要することがわかっており，今回の結果はこの結果を逸するものではなく，framework の実装は従前の実装と移動処理に係る性能差はないと言える．また，IP アドレスの更新に要する時間は OS に依存することがこれまでの研究から判明しており，OS によっては 1 秒程度の通信断絶時間で通信を再開できることが見込まれる．

なお，今回の試験ネットワークは遅延のほとんどないローカルな環境で行ったものであり，トンネルの再構築にはネットワーク遅延が加算される．しかし，インターネットの RTT を約 25ms と

仮定[*2]しても，IP アドレスの更新に要する時間の方が依然として支配的である．

なお，NTMobile を用いたシームレスハンドオーバーが実現した場合，移動に伴う通信断絶は発生しない [27].

表 4　移動試験結果

区分	時間 (ms)	標準偏差
IP アドレスの更新に要する時間	5,789	2.564
トンネルの再構築に要する時間	83	0.030

5.1.3　ラッパー試験

実装した Java ラッパーと Ruby ラッパー間で通信試験を行う．5.1.1 節の試験環境において，それぞれのラッパーを用いて開発した Java プログラム及び Ruby プログラムを用い，MN で Java プログラムを，CN でプログラムを実行して試験を行った．試験は Java プログラムと Ruby プログラム間で TCP パケットの送受信を行う．

試験の結果，Java プログラムと Ruby プログラム間で TCP コネクションが構築され，データの送受信を確認できた．この結果より，異なるプログラミング言語で実装されたアプリケーション間で framework を用いた通信が可能であることが立証できた．また，Ruby ラッパーに見られるように，既存アプリケーションの大部分の改造を要せずに framework の利用を可能とするラッパーを設計できることが確認できた．

5.2　framework の課題

新しく開発するアプリケーションは，NTM ソケット API を利用することで NTMobile の機能を簡単に利用できる．しかし，既存のアプリケーションを framework に対応させる場合，BSD ソケット API を NTM ソケット API に書き換える必要がある．また，C 言語の BSD ソケット API を利用せず，例えば MFC (Microsoft Foundation Class)[*3] を利用してソケット通信を実装している場合，これを NTM ソケット API に置き換える必要がある．MFC のソケット API を利用しない形にアプリケーションを改造する必要があり，アプリケーションの規模によっては改造コストが非常に高くなる．framework のラッパーは，JNA の実装のほか，JNI (Java Native Interface)[*4] 及び Ruby 拡張ライブラリのプロトタイプ実装のみであり，その他のプログラミング言語で framework を利用する場合，ラッパーの開発が必要であり，その上でアプリケーションのソケットの実装をラッパーに置き換える必要がある．

以上のように，framework の利用には各プログラミング言語で一般に利用されるソケット API またはソケットクラスのラッパーの開発及び普及が急務である．第 4 章で示した Ruby のラッパーの

[*2] 3G 回線による (www.softbank.jp) への 4 日間の Ping による実測値.
[*3] https://msdn.microsoft.com/ja-jp/library/d06h2x6e.aspx
[*4] http://docs.oracle.com/javase/7/docs/technotes/guides/jni/

ように，既存のソケットクラスを継承し，クラス宣言部の置き換えのみで利用可能なラッパーは汎用性が高く，このようなラッパーの実装が好ましいだろう．

第6章　今後の展望

framework は，VpnService 利用型及び TUN/TAP 利用型の実装のベースとなる．framework の実装により，これらの実装の加速が見込まれる．

表 5 に NTMobile の実装方式別の定性的な評価を示す．

アプリケーションの透過性は，フレームワーク組込型以外は既存アプリケーションの改造が不要である．framework はアプリケーション開発者による NTM ソケット API の利用が必須であり，他方式の実装はこの点において大きな利点を持つ．実効速度はカーネル空間で処理をするカーネル実装型が最も良い．その他の方式はユーザー空間でカプセル化/デカプセル化を行うため，カーネル実装型と比較して実効速度の低下が予想される．なお，VpnService 利用型および TUN/TAP 利用型は，アプリケーションの送信するデータをカーネル空間で処理し，一度ユーザー空間に戻して暗号化等の処理を行い，再度カーネル空間へ戻すことから，フレームワーク組込型より実効速度の低下が予想される．VpnService 利用型や TUN/TAP 利用型の実装により，各実装ごとのスループット特性の評価が待たれる．

保守コストは，カーネル実装型以外は同程度と見込まれる．カーネル実装型は Linux カーネルのバージョンアップに NTMobile カーネルモジュールについても追従する必要がある．カーネルの実装はユーザーアプリケーションの実装と比してプログラミングコストが大きく，コストが高い．VpnService 利用型や TUN/TAP 利用型についても AndroidOS のアップデートや TUN/TAP ドライバのアップデートに追従する必要がある．一方でフレームワーク組込型は，基本的にスタンドアロンで動作する仕組みであり，利用している lwIP の更新に対応すればよい．lwIP 自体はフレームワーク組込型に内包されるため，更新前の lwIP が組み込まれた framework も問題なく動作する．つまり，重大なセキュリティ上の問題等が無い場合，必ずしも lwIP の更新に追従する必要はなく，framework 自体の保守コストは低い．しかし，他のプログラミング言語で利用するためのラッパーは，各プログラミング言語のソケット API またはソケットクラス等の更新に追従する必要がある．

クロスプラットフォーム化は，カーネル実装型は Linux のみ，VpsService 利用型は AndroidOS のみで利用可能であり，クロスプラットフォーム化できない．TUN/TAP 利用型も TUN/TAP デバイスドライバが実装された OS 以外では利用できない．一方フレームワーク組込型は，framework が外部との通信のためのソケットを開くことができれば，OS を問わない．framework で利用している lwIP は独立した TCP/IP の実装であり，OS なしでも動作する．このため，framework で利用するタイマーや排他制御のクロスプラットフォーム化により，OS 問わず利用可能である．

以上に示すように，実装方式ごとに特性がある．NTMobile の利用者は利用シーンに応じて最も適した実装モデルを採用することができ，NTMobile 利用の敷居が低くなることが見込まれる．また，市販のスマートフォン等の製品をターゲットとした場合，root 権限が不要な framework は有

力な選択肢となるだろう.

表 5　実装方式別比較

	カーネル 実装型	フレームワーク 組込型	VpnService 利用型	TUN/TAP 利用型
アプリケーションの透過性	◎	×	◎	◎
実効速度	◎	○	△	△
保守コスト	×	○	○	○
クロスプラットフォーム化	×	◎	×	○

第7章　結論

本論文では，統合的枠組みに基づく実装として，framework を実装し，動作することを確認した．動作確認により，アプリケーションは仮想 IP アドレスによって通信を行い，端末の属するネットワークに依存せずに通信を行うことができた．また，framework を複数のプログラミング言語から利用できることを示し，適切なラッパーを用意することでアプリケーション開発者が容易に NTMobile を利用できることを示した．本研究により，C 言語を含み複数のプログラミング言語で framework を用いたアプリケーションの開発環境を提供することが可能となり，実用化に近づいた．なお，framework の実装は，カプセル化/デカプセル化部を除き VpnService 利用型や TUN/TAP 利用型に流用可能であり，今後これらの実装が加速すると推察される．複数の実装方式の実現は，NTMobile ユーザーの選択の幅を広め，より実用的な利用を可能とするだろう．

今後は，Windows や iOS をはじめとしたクロスプラットフォーム化や，様々なプログラミング言語の主要な，そして汎用的なラッパーについて検討と実装を進めていく予定である．

謝辞

本研究を遂行するにあたり，多忙な中御指導と御教示を賜りました，名城大学大学院理工学研究科渡邊晃教授に心より感謝申し上げます.

本論文を執筆するにあたり，快く副査を引き受けて頂きました，名城大学大学院理工学研究科柳田康幸教授に心より厚く御礼申し上げます.

本研究を遂行するにあたり，副査を快諾して頂き，常日頃より的確な御意見及び御助言を賜りました，名城大学大学院理工学研究科鈴木秀和准教授に心より深謝致します.

本研究を遂行するにあたり，本質を捉えた多くの貴重な御意見を賜りました，愛知工業大学大学院経営情報科学研究科内藤克浩准教授に心から感謝致します.

最後に，本研究を行うにあたり，率直な御意見をいただいたほか，動作実験等の稼行に御協力をいただきました，名城大学大学院理工学研究科情報工学専攻渡邊研究室及び鈴木研究室の諸氏に厚く御礼申し上げます.

参考文献

[1] Index, C. V. N.: Global Mobile Data Traffic Forecast Update, Cisco Systems (2013-2018).

[2] 総 務省：平成 28 年版情報通信白書 (2016).

[3] Leung, K., Dommety, G., Narayanan, V.，and Petrescu, A. : Network Mobility (NEMO) Extensions for Mobile IPv4, RFC 5177, IETF (2008). Updated by RFC 6626.

[4] Ishiyama, M., Kunishi, M., Uehara, K., Esaki, H.，and Teraoka, F. : LINA: A New Approach to Mobility Support in Wide Area Networks, *IEICE Transactions on Communications*, Vol. E84-B, No. 8, pp. 2076–2086 (2001).

[5] カフレベド，福島裕介，原井洋明 : Design and Implementation of ID-Layer Multipath Forwarding in HIMALIS Architecture, 電子情報通信学会技術研究報告，Vol. 114, No. 478, pp. 301–306 (2015).

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

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

[8] 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).

[9] Salsano, S., Bonola, M.，and Patriarca, F. : The UPMT solution(UPMT technical report version 2.0.2) (2016).

[10] Mawatari, M., Kawashima, M.，and Byrne, C. : 464XLAT: Combination of Stateful and Stateless Translation, RFC 6877, IETF (2013).

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

[12] Moskowitz, R., Heer, T., Jokela, P.，and Henderson, T. : Host Identity Protocol Version 2 (HIPv2), RFC 7401, IETF (2015). Updated by RFC 8002.

[13] 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. 380–393 (2013).

[14] 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).

[15] 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. 10, pp. 2288–2299 (2013).

[16] 納堂博史，鈴木秀和，内藤克浩，渡邊　晃 : NTMobile における自律的経路最適化の提案，情報処理学会論文誌，Vol. 54, No. 1, pp. 394–403 (2013).

[17] 納堂博史，杉原史人，鈴木秀和，内藤克浩，渡邊　晃 : NTMobile の実用化に向けた統合的枠組の検討，情報処理学会研究報告，Vol. 2015-MBL-77, No. 20, pp. 1–8 (2015).

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

[19] Maenpaa, J., Andersson, V., Camarillo, G.，and Keranen, A.：Impact of Network Address Translator Traversal on Delays in Peer-to-Peer Session Initiation Protocol, *Proc. of IEEE GLOBECOM2010* (2010).

[20] 上野泰輔，大久保陽平，鈴木秀和，内藤克浩，渡邊　晃:OpenIDConnect を用いた NTMobile ユーザ認証スキームの提案，第 13 回情報学ワークショップ WiNF2015 論文集，No. C2-2, pp. 153–158 (2015).

[21] Miyazaki, Y., Sugihara, F., Naito, K., Suzuki, H.，and Watanabe, A.：Certificate based key exchange scheme for encrypted communication in NTMobile networks, *Proceedings of the 12th IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS 2015)*, No. RS8-5, pp. 1–5 (2015).

[22] 杉原史人，内藤克浩，鈴木秀和，渡邊　晃，森香津夫，小林英雄 : NTMobile における組み込み機器向けトラフィック削減手法の提案，マルチメディア分散協調とモバイル (DICOMO2014) シンポジウム論文集，Vol. 2014, No. 1, pp. 1313–1318 (2014).

[23] 内藤克浩，鈴木秀和，渡邊　晃，森香津夫，小林英雄：スマートフォンアプリケーションにおいてエンド間通信を実現可能なプラットフォーム開発，マルチメディア, 分散, 協調とモバイル (DICOMO2014) シンポジウム論文集，Vol. 2014, pp. 836–844 (2014).

[24] 三宅佑佳，鈴木秀和，内藤克浩，渡邊　晃：NTMobile における端末移動後を考慮した最適なリレーサーバ選択手法の提案と実装，情報処理学会研究報告，Vol. 2015-MBL-77, No. 19, pp. 1–9 (2015).

[25] 山田貴之，鈴木秀和，内藤克浩，渡邊　晃：IPv4/IPv6 混在環境に対応した VpnService 型 NTMobile の性能評価，マルチメディア, 分散, 協調とモバイル (DICOMO2015) シンポジウム論文集，Vol. 2015, pp. 1784–1791 (2015).

[26] 鈴木秀和，内藤克浩，渡邊　晃:ユーザ空間における移動透過通信技術の設計と実装，マルチメディア, 分散, 協調とモバイル (DICOMO2014) シンポジウム論文集，Vol. 2014, pp. 1319–1325 (2014).

[27] Yohei, O., Suzuki, H., Naito, K.，and Watanabe, A.：A Seamless Handover Method Using IEEE 802.21 and NTMobile for Android Smartphones, *Proc. of The 9th International Conference on Mobile Computing and Ubiquitous Networking (ICMU2016)*, pp. 59–60 (2016).

研究業績

学術論文（査読あり）

（1）納堂博史，鈴木秀和，内藤克浩，渡邊　晃:NTMobile における自律的経路最適化の提案，情報処理学会論文誌，Vol. 54, No. 1, pp. 394-403, Jan. 2013.

国際会議

（1）Katsuhiro Naito, Fumihito Sugihara, Hiroshi Nodo, Masanori Kako, Tatsuya Hirose, Hidekazu Suzuki, Akira Watanabe, Kazuo Mori, Hideo Kobayashi: Implementation of smartphone applications supporting end-to-end communication, Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MOBIQUITOUS 2014), pp. 393-394, Dec. 2014.

研究会・大会等（査読なし）

（1）納堂博史, 杉原史人, 鈴木秀和, 内藤克浩, 渡邊　晃：NTMobile の実用化に向けた統合的枠組の検討，情報処理学会研究報告，Vol. 2015-MBL-77, No. 20, pp. 1-8 Nov. 2015.

（2）納堂博史, 鈴木秀和, 内藤克浩, 渡邊　晃：NTMobile の経路最適化の検討，情報処理学会研究報告，Vol. 2012-MBL-61, No. 33, pp. 1-8 Mar. 2012.

（3）納堂博史, 鈴木秀和, 内藤克浩, 渡邊　晃：多段 NAT 環境における NTMobile の経路最適化の提案，平成 23 年度電気関連学会東海支部連合大会論文集，Sep. 2011.

（4）赤堀蒼磨，納堂博史，鈴木秀和，内藤克浩，渡邊　晃：NTMobile を Android 端末に実装するための検討，平成 28 年度電気・電子・情報関係学会東海支部連合大会講演論文集，No. B1-1, Sep. 2016.

（5）尾久史弥，納堂博史，鈴木秀和，内藤克浩，渡邊　晃：NTMobile アダプタの実現方式の検討，平成 28 年度電気・電子・情報関係学会東海支部連合大会講演論文集，No. B1-3, Sep. 2016.

（6）菅沼良一，納堂博史，鈴木秀和，内藤克浩，渡邊　晃：NTMobile におけるマルチキャスト機能の実現，平成 28 年度電気・電子・情報関係学会東海支部連合大会講演論文集，No. B2-4, Sep. 2016.

（7）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・電子・情報関係学会東海支部連合大会講演論文集，No. B2-5, Sep. 2016.

（8）鈴木秀和，上醉尾一真，納堂博史，西尾拓也，内藤克浩，渡邊　晃 : IPv4/IPv6 混在ネットワークにおいて通信接続性と移動透過性を実現する NTMobile の研究，マルチメディア，分散，協調とモバイル（DICOMO2012）シンポジウム論文集，Vol. 2012, No. 1, pp. 2391 － 2401, Jul. 2012.

受賞歴

（1）**情報処理学会東海支部　学生論文奨励賞**（2012 年）
納堂博史，鈴木秀和，内藤克浩，渡邊　晃 : NTMobile における自律的経路最適化の提案，情報処理学会論文誌，Vol. 54, No. 1, pp. 394-403, Jan. 2013.

（2）**モバイルコンピューティングとユビキタス通信研究会奨励発表賞** (2011 年)
納堂博史，鈴木秀和，内藤克浩，渡邊　晃 : NTMobile の経路最適化の検討, 情報処理学会研究報告，Vol. 2012-MBL-61, No. 33, pp. 1-8, Mar. 2012.

付 録A　frameworkのトンネルテーブル仕様

表 6　トンネルテーブルの仕様

メンバ名	型	説明
path_id	path_id_t	通信を識別する一意の値
target_fqdn	fqdn_t	通信相手端末の FQDN
target_node_id	node_id_t	通信相手端末を一意に識別する値
target_vip_addr	struct ntm_sockaddr	通信相手端末の仮想 IPv4 アドレス
target_vip6_addr	struct ntm_sockaddr	通信相手端末の仮想 IPv6 アドレス
nego_key	ntm_com_key_t	制御パケットの暗号化及び MAC 計算に用いる共通鍵
data_key	ntm_com_key_t	データパケットの暗号化及び MAC 計算に用いる共通鍵
target_ip_addr	struct ntm_sockaddr	通信相手端末の実 IPv4 アドレス
target_ip6_addr	struct ntm_sockaddr	通信相手端末の実 IPv6 アドレス
target_nat_addr	struct ntm_sockaddr	通信相手端末の属する NAT の IPv4 アドレス
rs_ip_addr	struct ntm_sockaddr	RS の実 IPv4 アドレス
rs_ip6_addr	struct ntm_sockaddr	RS の実 IPv6 アドレス
dst_addr	struct ntm_sockaddr	UDP トンネル構築先の実 IPv4 アドレス
dst_addr6	struct ntm_sockaddr	UDP トンネル構築先の実 IPv6 アドレス
target_port	uint16_t	UDP トンネル構築先のポート番号
timestamp	time_t	このエントリの最終参照日時
mutex	NTM_MUTEX	このエントリの排他制御用 Mutex
ref_count	int32_t	このエントリを参照しているポインタ変数の個数

従来の実装であるカーネル実装型は，パケットのカプセル化/デカプセル化処理をカーネルモジュールとして実装している．ユーザー空間に実装される NTMobile デーモンは制御メッセージの送受信を行い，必要に応じて Netlink ソケットでカーネルモジュールと連携をする．このため，カーネル実装型においては，トンネルテーブルがカーネルモジュールに実装されている．

一方，framework は，NTMobile の全ての機能をユーザー空間に実装している．このため，トンネルテーブルの実装も framework 独自で行っている．ここでは，framework におけるトンネルテーブルの仕様を記す．

表 6 にトンネルテーブルの仕様を示す．各メンバの型は統合的枠組みとして統一された仕様以前の NTMobile と共通であるので，該当する資料を参照されたい．

トンネルテーブルは多対一のハッシュテーブルとして実装されており，ハッシュキーとなるメンバ名に下線が付されている．トンネルテーブルは，ref_count が 0 で，最終参照日時から一定時間経過したエントリが削除される．カプセル化/デカプセル化の際には最終参照日時が更新される．カプセル化/デカプセル化の処理中や経路最適化処理中，移動に伴うトンネル再構築中は ref_count

が 1 以上となる．トンネルテーブルエントリは複数のスレッドから参照されることから，メンバ値の参照や更新時に Mutex による排他制御が行われ，値を参照しない場合等 *1 は排他処理を行わない．

*1 経路最適化処理時における，通信相手端末からのパケット受信待機中等．

付 録B　NTMソケットAPI

framework が提供する NTM ソケット API を表 7 にまとめる．なお，framework は C 言語で実装されているため，対応するソケット API も C 言語のものである．NTM ソケット API 関数の引数は，C 言語の標準ソケット API と互換性を持つ．

また，NTMobile の動作のために独自に提供する API を表 8 にまとめる．この API は framework の初期化や終了等に用いられ，アプリケーションの起動時と終了時に呼び出される必要がある．

表 7　NTM ソケット API 一覧

標準ソケット API	対応する NTM ソケット API
socket	ntmfw_socket
close	ntmfw_close
closesocket	ntmfw_closesocket
shutdown	ntmfw_shutdown
bind	ntmfw_bind
gethostbyname	ntmfw_gethostbyname
gethostbyname_r	ntmfw_gethostbyname_r
getaddrinfo	ntmfw_getaddrinfo
freeaddrinfo	ntmfw_freeaddrinfo
getsockopt	ntmfw_getsockopt
setsockopt	ntmfw_setsockopt
ioctl	ntmfw_ioctl
ioctlsocket	ntmfw_ioctlsocket
fcntl	nrmfw_fcntl
select	ntmfw_select
sendto	ntmfw_sendto
send	ntmfw_send
write	ntmfw_write
listen	ntmfw_listen
recvfrom	ntmfw_recvfrom
recv	ntmfw_recv
read	ntmfw_read
connect	ntmfw_connect
accept	ntmfw_accept
getsockname	ntmfw_getsockname
getpeername	ntmfw_getpeername

表 8　framework 独自の API 一覧

関数名	説明
ntmfw_ntm_init	framework の動作に必要な値を引数に持ち，framework の初期化を行う．
ntmfw_stop	framework の処理を終了する．
ntmfw_getdata	framework が内部に保持している DC との共通鍵等のデータを取り出す．

付 録C　frameworkのカプセル化/デカプセル化処理

framework において，パケットのカプセル化/デカプセル化は lwIP を用いて行っている．これは framework 独自の実装である．ここでは，lwIP を用いた framework のカプセル化/デカプセル化について詳しく述べる．

C.1　lwIP について

lwIP はユーザー空間における TCP/IP の実装であり，framework は仮想 IP スタックに lwIP を利用している．lwIP は，lwipopts.h ファイルに C 言語のプリプロセッサである#define ディレクティブを用いて機能の有効・無効の切り替え等が可能である．framework で IPv4/IPv6 パケットのカプセル化/デカプセル化を行うため，表 9 に示す設定で lwIP をコンパイルし，ライブラリ化した．なお，framework は，このライブラリ化された lwIP をリンクし，framework ライブラリ [*1] を生成する．アプリケーションは，この framework ライブラリをリンクして NTM ソケット API を利用する．

C.2　カプセル化/デカプセル化の実装

カプセル化/デカプセル化時の主要な処理フローを図 10 及び図 11 に示す．

framework 以外の実装方式は，アプリケーションの改造が不要である．アプリケーションは BSD ソケット API を利用してパケットを送信すれば良く，一度カーネルの TCP/IP プロトコルスタック実装で TCP/IP ヘッダまたは UDP/IP を付されたデータのバッファに対して，カーネル空間またはユーザー空間で暗号化等の処理を施し，再度送信する．framework は，内包する lwIP が TCP/IP プロトコルスタック実装であるので，アプリケーションが送信するデータをそのまま lwIP へ処理を渡し，TCP/IP または UDP/IP ヘッダを付する．lwIP にはコールバック関数を利用できるため，これを用いて再度 framework に処理を戻し，暗号化等の処理の後，BSD ソケット API で送信する．BSD ソケット API で送信されたデータは，カーネルの TCP/IP プロトコルスタック実装により再度 UDP/IP ヘッダを付されて送信される．受信の際には，BSD ソケット API でパケットを受信すると外側の UDP/IP ヘッダが除かれたデータを得られるので，これを復号化し，lwIP のインプットコールバックを直接呼び出す．この処理により，lwIP により内側の TCP/IP または UDP/IP ヘッダが除かれてスタックにプッシュされる．アプリケーションが NTM ソケット API の受信関数を呼び

[*1] libntmfw.so 及び libntmfw.a

表 9　lwipopts.h の設定値

定義名	値
IP_FORWARD	1
IP_OPTIONS_ALLOWED	1
IP_REASSEMBLY	1
IP_FRAG	1
IP_REASS_MAXAGE	3
IP_REASS_MAX_PBUFS	4
IP_FRAG_USES_STATIC_BUF	0
IP_DEFAULT_TTL	255
LWIP_ICMP	1
LWIP_RAW	1
LWIP_DHCP	0
LWIP_AUTOIP	0
LWIP_SNMP	0
LWIP_IGMP	0
LWIP_DNS	1
LWIP_UDP	1
LWIP_TCP	1
LWIP_LISTEN_BACKLOG	0
PBUF_LINK_HLEN	16
PBUF_POOL_BUFSIZE	LWIP_MEM_ALIGN_SIZE(TCP_MSS+40+PBUF_LINK_HLEN)
LWIP_HAVE_LOOPIF	0
LWIP_NETCONN	1
LWIP_SOCKET	1
LWIP_STATS	1
PPP_SUPPORT	0
LWIP_IPV6	1
LWIP_IPV6_SEND_ROUTER_SOLICIT	0

出すと，lwIP の受信関数に処理が渡るので，スタックからデータがポップされ，アプリケーションはデータを受信する．

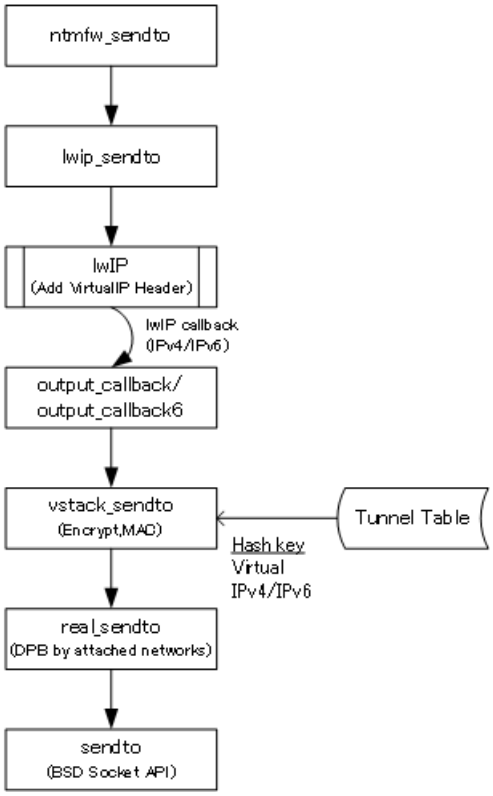

図 **10**　カプセル化

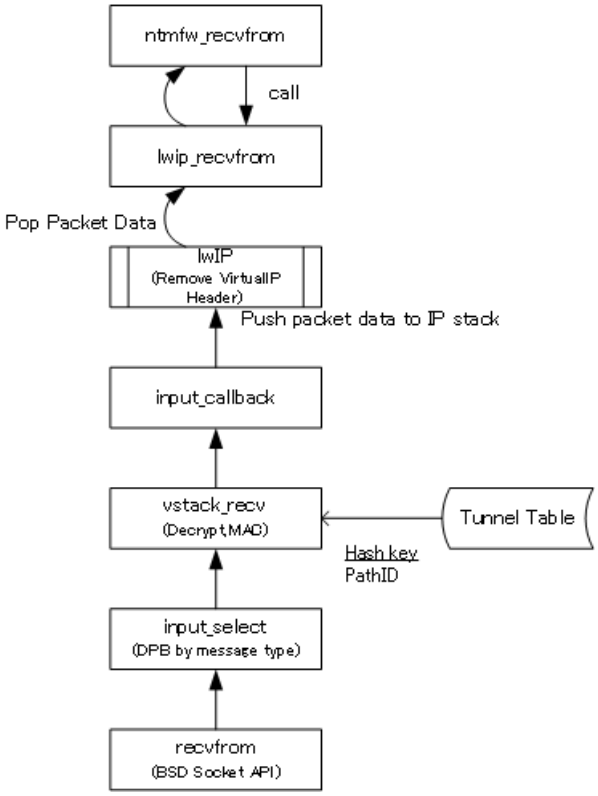

図 **11**　デカプセル化