実用化に向けたNTMobileフレームワークの実装と評価

納堂　博史[1,a)]　八里　栄輔[3]　鈴木　秀和[1]　内藤　克浩[2]　渡邊　晃[1,b)]

概要：
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 は，様々な新機能が提案されたことや，複数の実装モデルが提案されたこともあり，仕様の把握が困難であった．そこで実用化に向けて仕様が統一され，実装モデルとしてカーネル実装型と，これをアプリケーションで実現するフレームワーク組込型が検討されてきた．カーネル実装型は性能が高いものの root 権限が必要という課題があり，モバイル端末への適用が難しい．一方，フレームワーク組込型はモバイル端末や様々な OS への移植が容易であり，実用化に適しているという特徴がある．本稿ではフレームワーク組込型の実装を実現し評価したので報告する．フレームワークは C 言語で記述されているが，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 NTMobile 通信を行えることを確認した．

Practical Realization of NTMobile Framework that makes Flat Networks

Nodo Hiroshi[1,a)]　Eisuke Yasato[3]　Suzuki Hidekazu[1]　Naito Katsuhiro[2]　Watanabe Akira[1,b)]

1. はじめに

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，IP ネットワークの通信量は増加の一途を辿っている [1]．IP ネットワークは互いに互換性のない IPv4 と IPv6 が利用されており，両者が混在しているのが現状である．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 割に達する．このような状況において，IPv4 グローバルアドレスが枯渇し，IPv6 への移行が必須と言われている．しかし，小規模サービスプロバイダにおいては，当面 IPv6 へ対応しないことが決定されたとの報告もあり，IPv6 の普及には時間を要すると見込まれる [2]．また，スマートフォンにおいても IPv6 への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 ネットワークが中心のネットワーク環境が続くことが想定される．

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 アドレスが端末識別子と位置識別子を兼ねているため，ネットワークを切り替えて IP アドレスが変化すると，構築済みの通信セッションが切断されるという課題である．特に IPv4 ネットワークでは NAT がパケットのアドレス変換を行うので，アドレスの変化を隠蔽するのは容易ではない．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ため，IPv4 アドレスしか持たない通信端末は IPv6 ネットワークに接続できず，IPv6 アドレスしか持たない通信端末は IPv4 ネットワークに接続できないという課題である．また，IPv4 ネットワークにおいて，NAT が通信経路上に存在すると，NAT の外側から内側に向けての通信を開始できないという課題があり，NAT 越え問題と呼ばれている．

移動透過性の課題に関しては，従来より様々な研究があるが [3–5]，多くの技術は IPv6 ネットワークを前提にしており，IPv4 ネットワークには適用できない．Mobile IPv4

[1] 名城大学
Nagoya-shi, Aichi-ken 468–8502, Japan
[2] 愛知工業大学
[3] バレイキャンパスジャパン
[a)] hiroshi.noudou@wata-lab.meijo-u.ac.jp
[b)] wtnbakr@meijo-u.ac.jp

のように IPv4 ネットワーク対応の技術もあるが，NAT が介在する場合は移動先が限定されたり，冗長経路となるなどの課題が解決できていない．

通信接続性の課題については，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いるが [6–9]，これらの技術は通信端末の移動を考慮しておらず，移動透過性は実現できない．

IPv4 ネットワークと IPv6 ネットワーク間での通信を可能とする技術も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ことや，移動透過性を考慮していないなどの課題がある [10]．

移動透過性と通信接続性の課題を同時に解決する技術として，DSMIPv6(Dual Stack Mobile IP version 6) [11] と，HIP(Host Identity Protocol) [12] が検討されている．DSMIPv6 は Mobile IPv6 をベースに，IPv4 が混在する環境に拡張した方式である．しかし，IPv4 ネットワークにおいては Mobile IPv4 がそのまま使われており，NAT が介在する場合の当該技術の課題をそのまま引きずった形となっている．HIP は IP アドレスから端末識別子の役割を分離し，新たな端末識別子を導入することによって，通信接続性と移動透過性を確保することができる．しかし，NAT 越え技術として ICE [6] を利用しているため，NAT が存在するときのシグナリング時間が大きく，また NAT を跨る移動が難しいという課題がある．さらに DSMIPv6，HIP 共通の課題として，両者ともカーネル空間における実装が必要であり，スマートフォンなどへの適用が困難という課題がある．

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し，かつ既存技術の課題を解決できる有用な方式である [13–16]．NTMobile では，エンド端末がネットワークに依存しない一意の仮想的な IPv4 アドレスと IPv6 アドレスを保持する．実際の通信は通信端末が接続するネットワークの実 IP アドレスでカプセル化され，最適経路で通信相手に転送される．

NTMobile ではアプリケーションが IPv4 対応であっても IPv6 対応であってもかまわない．また，物理ネットワークも IPv4/IPv6 ネットワークが混在し，かつ NAT が存在していてもかまわない．通信中の移動先も制約がないという特徴がある．アプリケーション開発者は実ネットワークの制約を一切意識する必要がない．

NTMobile は，様々な機能が提案されその仕様の把握が困難な時期があったことから，実用化に向け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の実装が可能となることを強く意識している．本論文では，この枠組みに基づいて NTMobile をアプリケーションで実現するための基本技術としてフレームワーク組込型 (framework) の検討を行い，実装及び評価を行った．framework は通信ライブラリとして，アプリケーションから呼び出すことにより利用する．カーネルの改造が不要なことから，スマートフォンなどでの利用が期待できる．framework は C 言語で記述されており，これを異なるプログラミング言語において利用できるようにするため，Java と Ruby のラッパーを開発した．

今回実現した framework は，本来の意味での移動透過性と通信接続性を，アプリケーションレベルで実現した初の事例となる．今後の展開として，Android や iOS が提供する VPN サービスを利用する実装が可能で，既存のアプリケーションをそのまま利用できるようにできる．

以降，第 2 章において，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，第 5 章で動作検証と評価についてで述べる．最後に第 6 章でまとめる．

2. 関連研究

本章では，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同時に実現する既存技術について述べる．

2.1 DSMIPv6

DSMIPv6 は，Mobile IPv6 [18] を IPv4/IPv6 混在環境に適用したものである．DSMIPv6 の構成を**図 1** に示す．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．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

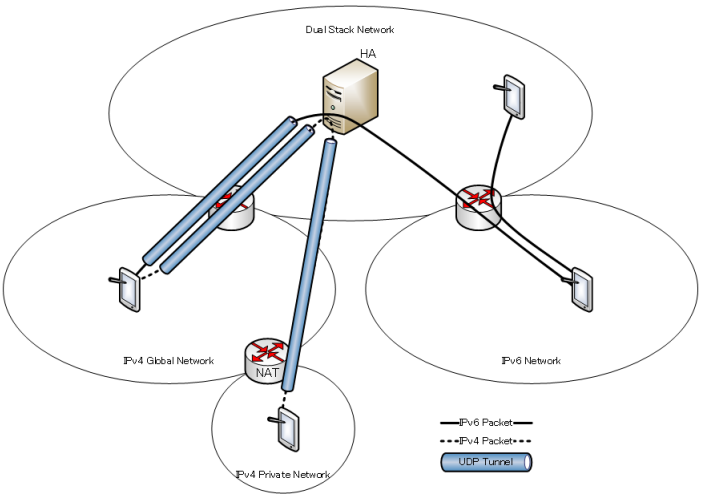


図 1 DSMIPv6 の構成

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，トンネルを通して MN へ届けられる．MN から CN 宛てのパケットはトンネルを通して HA に転送され，HA から CN に送信される．

DSMIP の基本は Mobile IPv6 であり，HA を経由して Mobile IPv4 の技術を適用できるようにしたものである．そのため，DSMIPv6 で利用できるアプリケーションは IPv6 対応のものに限られる．また，Mobile IPv4 の技術に関わる課題はそのまま継承される．すなわち，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り，IPv4 グローバルアドレスが枯渇した現在の状況においては現実的ではない．また，通信経路が必ず HA を経由した冗長経路となるなどの課題がある．

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，端末識別子を分離し，端末識別子として HI(Host Identifir) を用いる．エンド端末における HIP のレイヤーモデルを**図 2** に示す．IP 層と TCP/UDP 層との間に新たに HIP 層を定義する．HIP 層においては，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うので，移動によって IP アドレスが変化しても，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 NAT 越え技術として，ICE を利用するが，シグナリングに要するオーバーヘッドが高い [19]．また，ICE は移動透過性を考慮していないため，通信経路上に NAT が介在する場合は移動透過性が実現できない．HIP は IP 層とトランスポート層の間に HIP 層を設ける構造上，カーネルの改造が必須である．HIP の利用には root 権限を要するため，スマートフォンへの適用は難しい．さらに，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難しいという課題がある．

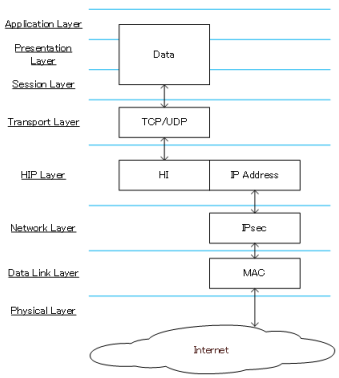


図 2 HIP のレイヤーモデル

3. 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 の構成を示す．NTMobile は下記に示す機器で構成される．

- 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．インターネット上に分散配置することができる．通信相手の DC を探索するために，DNS サーバとしての機能も持っている．
- AS(Account Server)
  ユーザの登録と認証を行う機器．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．
- 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．インターネット上に分散配置することができ，DC の判断により RS を選択できる．
- 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軽減するためのオプション機器．
- NTM 端末
  NTMobile の機能を有するエンド端末．

NTM 端末を除く装置群は，公開鍵証明書を所持する．NTMobile の制御メッセージに用いる共通鍵は，公開鍵証明書を用いて安全に共有される．

NTM 端末は，あらかじめ AS にユーザ ID と認証情報を登録しておく必要がある．基本機能としてメールアドレス及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．

NTM 端末は，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．次に，DC に対して実 IP アドレスを登録し，DC から仮想 IPv4/IPv6 アドレスの配布を受ける．接続ネットワークが切り替わったときは，その都度実 IP アドレスを DC に報告する．以降，定期的に DC とキープアライブを実行することにより，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．ここで NTM 端末がスマートフォンであれば APNS(Apple Push Notification Service) や GCM(Google Cloud Messaging) のサービスを利用することができ，DC とのキープアライブを省略できる．

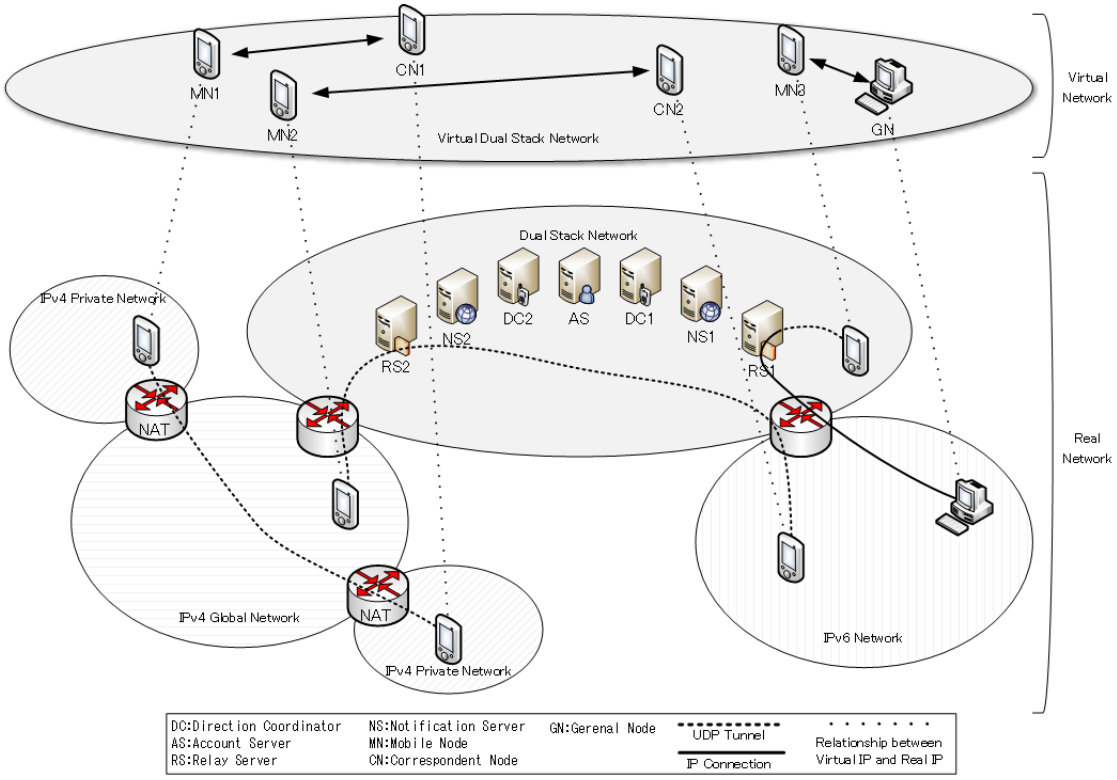


図 3 NTMobile の概要

通信を開始するときは，MN が実行する CN の名前解決がトリガとなり，NTMobile のシグナリング処理が開始される．MN は内部で DNS 要求をフックし，MN を管理する DC(DCmn) に経路指示要求を送信する．DCmn は，DNS サーバ機能により 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 アドレスと仮想 IP アドレスを取得する．DCmn は MN と CN の実 IP アドレスの内容から通信経路を決定し，MN と CN に対して経路指示を行う．CN に対する経路指示は DCcn 経由で行う．MN と CN は指示に従って MN-CN 間でエンドツーエンドの UDP トンネルを構築する．MN は DNS 応答として，CN の仮想 IP アドレスを返すので，MN と CN のアプリケーションは仮想 IP アドレスを用いたセッションを確立する．なお，MN と CN が直接通信できない場合は，DCmn は RS へ中継指示を行い，MN と CN は RS 経由の UDP トンネルを構築する．このとき，MN と CN の位置に応じて適切な RS が選択される．その後さらに MN と CN 間で経路最適化機能が働き，直接通信が可能であればエンドツーエンド通信に切り替わる．

3.2 NTM 端末の実装モデル

NTM 端末の実装モデルとしては，通信パケットのカプセル化処理を Linux カーネル空間で行うカーネル実装型と，カプセル化をアプリケーション層で処理するフレームワーク組込型がある．カーネル実装型は，既存のアプリケーションを一切変更する必要がないことと，スループットが高いという利点がある．その反面，OS が限定されることや，OS のバージョンアップに追随するのが容易ではないという課題がある．また，スマートフォンなどでは root 権限が必要になるため，一般ユーザが使うことは難しい．フレームワーク組込型は，カーネル実装型ほどのスループットは期待できないが，カーネル実装型の欠点を克服し，スマートフォンでの利用が可能である．

NTMobile は UDP カプセルを基本としており，アーキテクチャ的にカーネルを改造しない方法による実装が可能であるという特徴がある．フレームワーク組込型は，NTMobile をアプリケーションライブラリとしてユーザに提供するものである．アプリケーションが C 言語を用いて一般の通信ライブラリを利用するのと同じ要領で framework を呼び出すことにより利用する．このため，アプリケーションは少なからず NTMobile を利用することを意識する必要がある．フレームワーク組込型の応用として，スマートフォンが提供する VPN(Virtual Private Network) サービスを利用する VpnService 利用型への展開がある．VpnService 利用型が実現した場合，既存のアプリケーションをそのまま利用できるという利点がある．

4. framework の動作と実装

本章では，本論文で取り扱う framework の動作を詳しく記述し，その実装方法について述べる．

4.1 framework の動作

framework のトンネル通信の実現方法を図 **4** に示す．framework は，仮想 IP プロトコルスタック [*1] の機能を包含しており，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/UDP ヘッダ及び 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，MAC(Message Authentication Code) 付与等の処理を経て，ソケット API を用いて OS に渡される．この処理により，パケットは UDP でカプセル化されてネットワーク上に送信される．パケットの受信処理は上記と逆の手順により実現される．

framework の初期化処理はアプリケーション側から指示する必要がある．初期化処理において，framework は AS との認証，及び DC への登録処理を行う．通信開始時は，アプリケーション側からの名前解決をトリガとして，DC との間のシグナリング処理を経て，エンドツーエンドのトンネル経路を生成する．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し，トンネル再構築処理を行う．仮想 IP プロトコルスタックは，実 IP アドレスの変化に気づくことなく通信が継続される．

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．

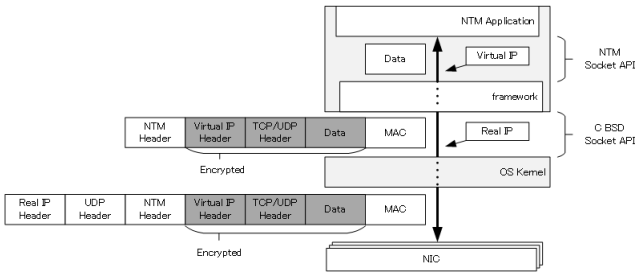


図 **4** framework のトンネル通信の実現方法

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，構築されているすべ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) 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[*2]，NodeID[*3]，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

[*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] 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．

[*3] 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．

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．

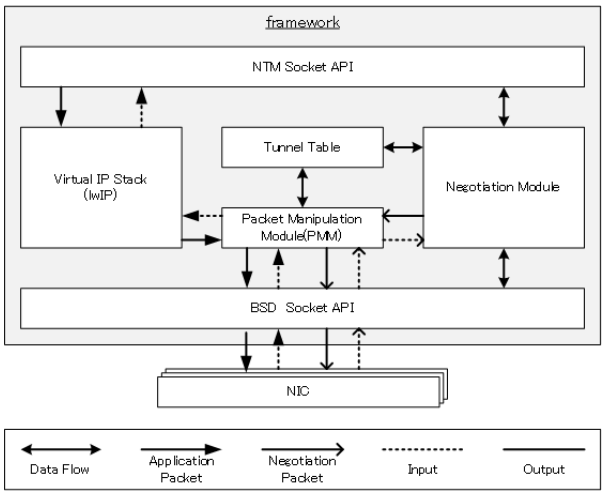


図 5　framework のモジュール構成

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した．

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) 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した．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．

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 拡張ライブラリを作成し，C 言語で記述された NTM ソケット API を Ruby から利用できるようにした．また，Ruby で TCP 通信を行う際に利用されるクラス [*4] 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した．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 [*5] において，framework の初期化に必要な引数が必要である．

[*4] 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．
[*5] 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．

5.　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．Windows10 マシン内に，仮想マシンにより NTMobile のネットワークを構築した．仮想マシンはすべて Ubuntu14.04LTS とした．

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，メールアドレス，及び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，すべ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 経由の接続とした．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 **1** 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．

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．試験ネットワーク及び各機器の諸元を図 **8** 及び表 **2** に示す．1 台のホストマシン

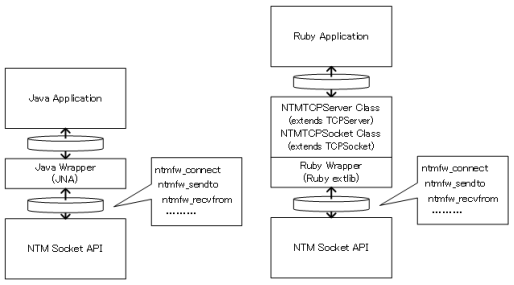


図 6　Java ラッパー

図 7　Ruby ラッパー

表 1　疎通試験結果

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

◎:end-to-end　○:via RS

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 を用いて MN 及び CN のパケットをキャプチャした．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．

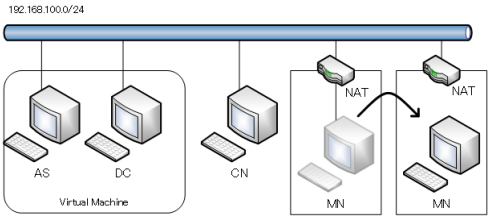


図 8　試験ネットワーク

試験結果を**表 3** 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ることがわかる．この時間には，NTM 端末が IP アドレスを取得するまでの時間と，アドレスの変化を OS が認識するまでの時間が含まれる．NTMobile に関わるシグナリング処理の時間はこれらに比べるとわずかである．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 と仮定しても十分高速に実現できると言える．

表 2　諸元

	AS/DC	MN/CN
OS	Ubuntu12.04	Ubuntu14.04
CPU	VM 1core	2core4thread
	3.40GHz	2.80GHz
Memory	2GB	8GB

表 3　移動試験結果

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

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った．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 Ruby 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った．

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．

5.2　framework の課題と対策

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) 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高くなる．framework のラッパーは，JNA の実装のほか，JNI(Java Native Interface) 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．

framework の利用を促進する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．また，VpnService 利用型への移植により，既存のアプリケーションをそのまま利用できる環境が実現できる．VpnService 利用型の動作は基本的に framework と同様であることから，早期の移植が望まれる．

6. まとめ

本論文では，統合的枠組みに基づく実装として，NTMobile の framework を実現し，動作することを確認した．動作検証により，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を確認した．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により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った．framework は，ほぼ制約のない形での移動透過性と通信接続性をアプリケーション上で実現した初の事例となる．framework は，VpnService 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予想される．

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．

参考文献

[1] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, Cisco Systems, 2013-2018, http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html
[2] 総務省: 平成 28 年版情報通信白書, http://www.soumu.go.jp/johotsusintokei/whitepaper/ (2016)
[3] D. Johnson, C. Perkins and J. Arkko : Mobility Support in IPv6, RFC3775, IETF (2004).
[4] M. Ishiyama, M. Kunishi, K. Uehara, H. Esaki and F. Teraoka: 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] 國司光宣，石山 政浩，植原啓介，寺岡文男 : 移動体通信プロトコル LIN6 の性能評価，情報処理学会論文誌，Vol. 43，No. 2，pp. 398–407，2002 年 2 月．
[6] J. Rosenberg,: Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols, RFC5245, IETF (2010).
[7] J. Rosenberg, R. Mahy, P. Matthews and D. Wing: Session Traversal Utilities for NAT (STUN), RFC5389, IETF (2008).
[8] R. Mahy, P. Matthews and J. Rosenberg: Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN), RFC5766, IETF (2010).
[9] S. Salsano, M. Bonola, and F. Patriarca: The UPMT solution(UPMT technical report version 2.0.2, (2016).
[10] M. Mawatari, M. Kawashima, and C. Byrne: 464XLAT: Combination of Stateful and Stateless Translation, RFC6877, IETF (2013).
[11] H. Soliman: Mobile IPv6 Support for Dual Stack Hosts and Routers, RFC5555, IETF (2009).
[12] R. Moskowitz, T. Heer, P. Jokela, and T. Henderson: Host Identity Protocol Version 2 (HIPv2), RFC7401, Updated by RFC8002, IETF (2015).
[13] 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)．
[14] 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)．
[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] C. Perkins, D. Johnson, and J. Arkko: Mobility Support in IPv6, RFC6275, IETF (2011).
[19] J. Maenpaa, V. Andersson, G. Camarillo, and A. Keranen: Impact of Network Address Translator Traversal on Delays in Peer-to-Peer Session Initiation Protocol, Proc. of IEEE GLOBECOM2010(2010).

実用化に向けた NTMobileフレームワークの 実装と評価

納堂 博史[1)] 八里 栄輔[2)] 鈴木 秀和[1)] 内藤 克彦[3)] 渡邊 晃[1)]

1)名城大学 2)バレイキャンパスジャパン 3)愛知工業大学

Watanabe Lab.

目次

- はじめに
- 関連研究
- NTMobileについて
- 実装
- 動作検証及び評価
- 今後の展望と課題
- まとめ

研究背景(1)

- 移動通信の増加
  モバイルデバイス(スマートフォン等)の普及
  ウェアラブルデバイスの一般販売

  固定通信網(IPネットワーク網)への誘導

  → IP網では移動が考慮されていない
  (ネットワークが切り替わると通信セッションが切断)

- IPv4とIPv6の混在ネットワーク
  スマートフォンのIPv6対応の遅れ
  小規模ISPにおけるIPv6非対応

  IPv4のみ対応するユーザー/IPv6のみ対応するユーザーの出現

  → IPv4-IPv6間で通信を行うことができない
  IPv4同士でもNAT外部から通信を開始できない

あらゆるネットワークにおいて通信接続性と移動透過性の要求

ISP:Internet Service Provider NAT(NAPT):Network Address (Port) Translation

研究背景(2)

- スマートフォンへの対応
  - マルチプラットフォーム化
    - Android™・iOS・Windows®
  - 非特権アカウント要件
    - root権限を使わずに利用可能

アプリケーションレベルで

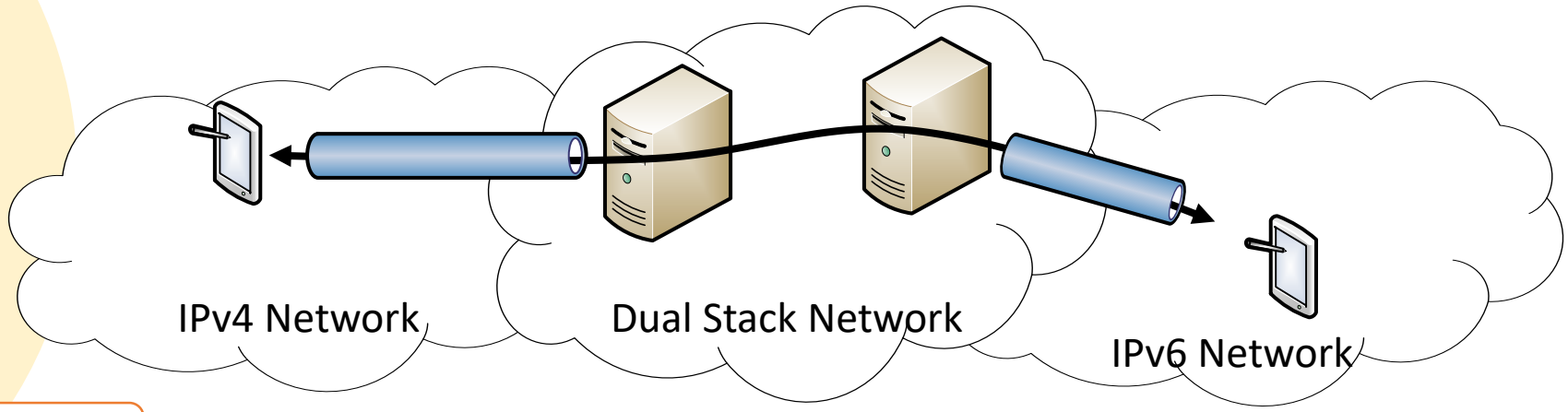
あらゆるネットワークにおいて通信接続性と移動透過性の要求

あらゆる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をアプリケーションレベルで実装

NTMobile: Network Traversal with Mobility

関連研究(1)

- DSMIPv6(Dual Stack Mobile IP version 6)
  - IPv4/IPv6デュアルスタックネットワークにHA(Home Agent)を配置
  - 移動端末はHAに最新のIPアドレスを通知
  - 通信は基本的にHA経由で行う(端末は移動してもHAは移動しない)



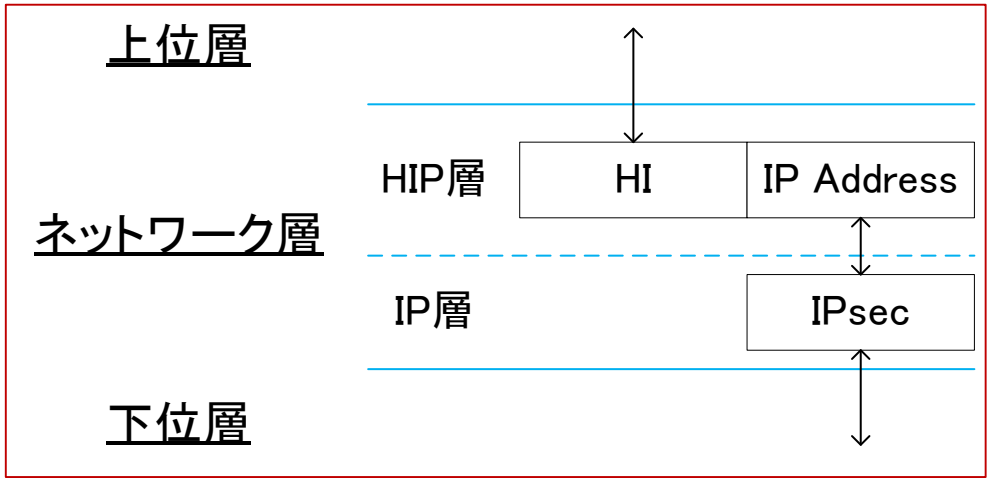
課題

IPv4では必ずHA経由の冗長経路となる
移動端末にIPv4グローバルアドレスを割り当てる必要がある

アプリケーションレベルで実現できない

関連研究(2)

- HIP(Host Identifier Protocol)
  - IPアドレスのほかにHI(Host Identifier)を定義
  - 端末の位置をIPアドレス、識別をHIで行う
  - アプリケーションはHIを識別子に通信を行う(移動してもHIは変化しない)



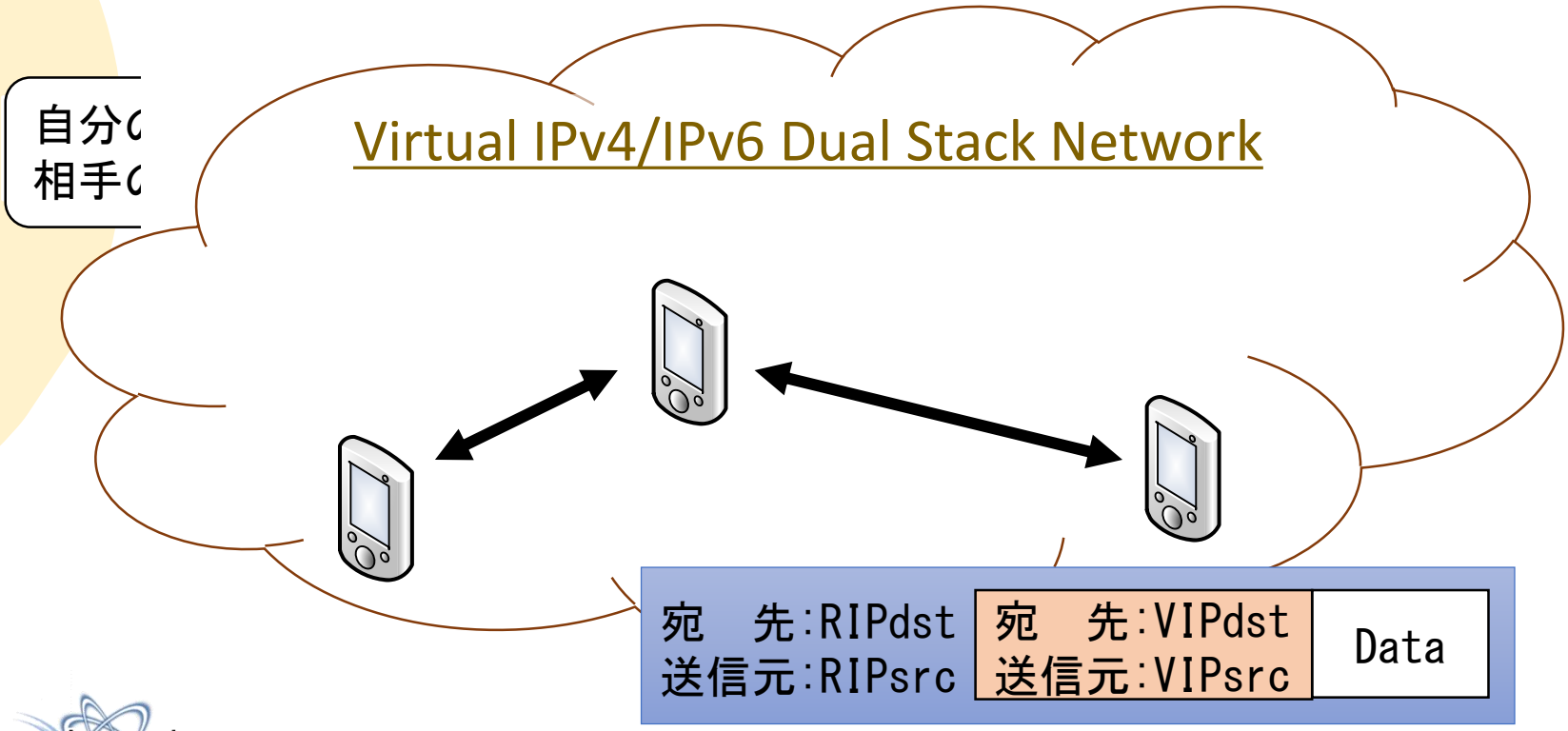
課題

最適経路探索のため通信開始時のオーバーヘッドが大きい
多くの既存技術を前提としており、これらが使えないと利用できない
(センサデバイス等の小型デバイスに向かない)

アプリケーションレベルで実現できない

NTMobileの概要

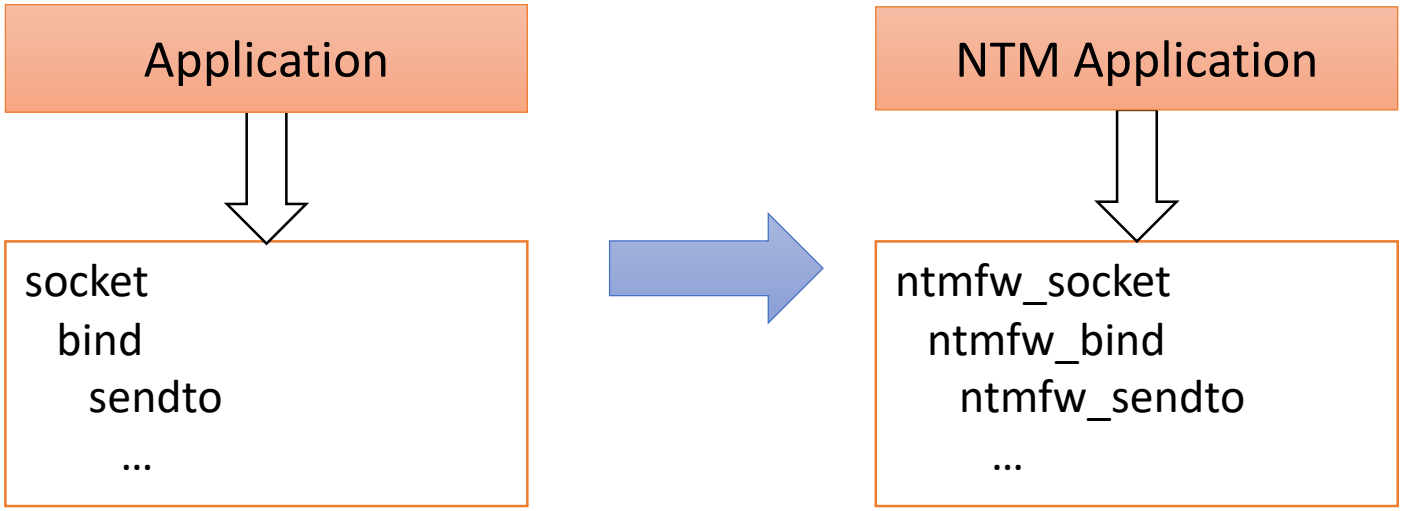
- 仮想的なIPv4/IPv6デュアルスタックネットワークを提供
  - 仮想IPv4/仮想IPv6アドレスを提供し、このアドレスを用いてパケット通信
  - 実際のパケットは実IPアドレスでカプセル化されて転送
    - 常に最新の実IPアドレスを管理装置に通知
    - 通信開始時等に通信相手の実IPアドレスを管理装置から取得
  - 端末移動時は外側のUDP/IPヘッダのみが変化(UDPトンネル)



VIP: Virtual IP Address　RIP: Real IP Address

frameworkの概要

- NTMobileの処理をすべてユーザ空間に実装
  - 非rootで使用可能
  - 特定のOSに依存しない実装が可能
- アプリケーションの利用するソケットAPIを置換
  - BSDソケットAPIの代替ソケットAPI(NTMソケットAPI)を提供
  - アプリケーション開発者はNTMソケットAPIを利用する

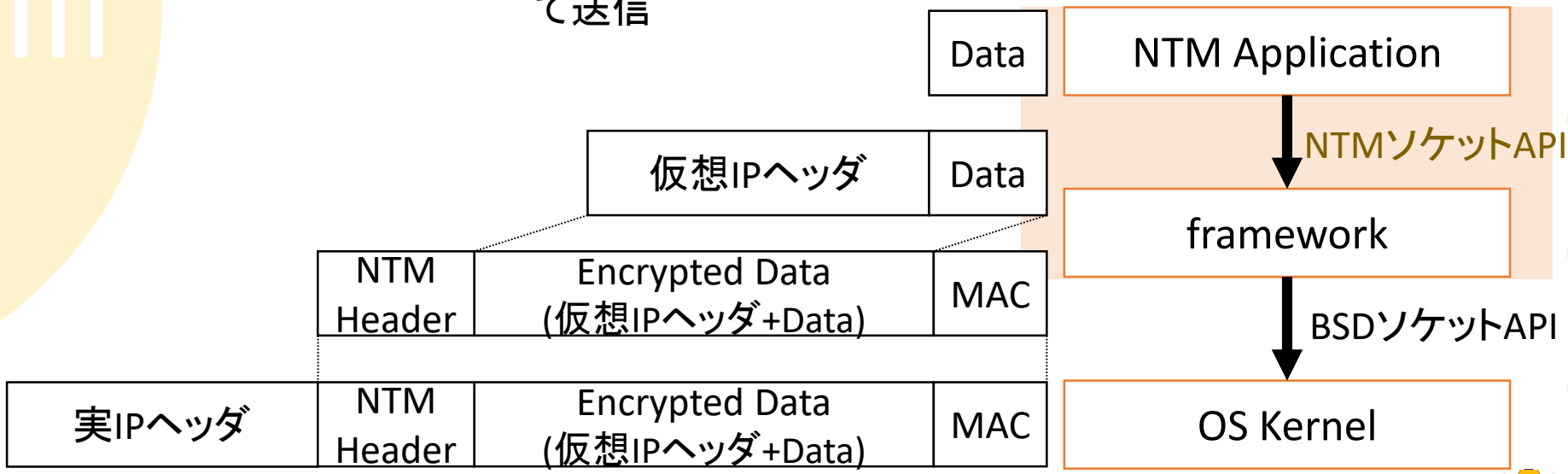


BSDソケットAPI: ネットワーク通信を行うC言語の関数ライブラリ

frameworkの動作

アプリケーション → framework → OSカーネル

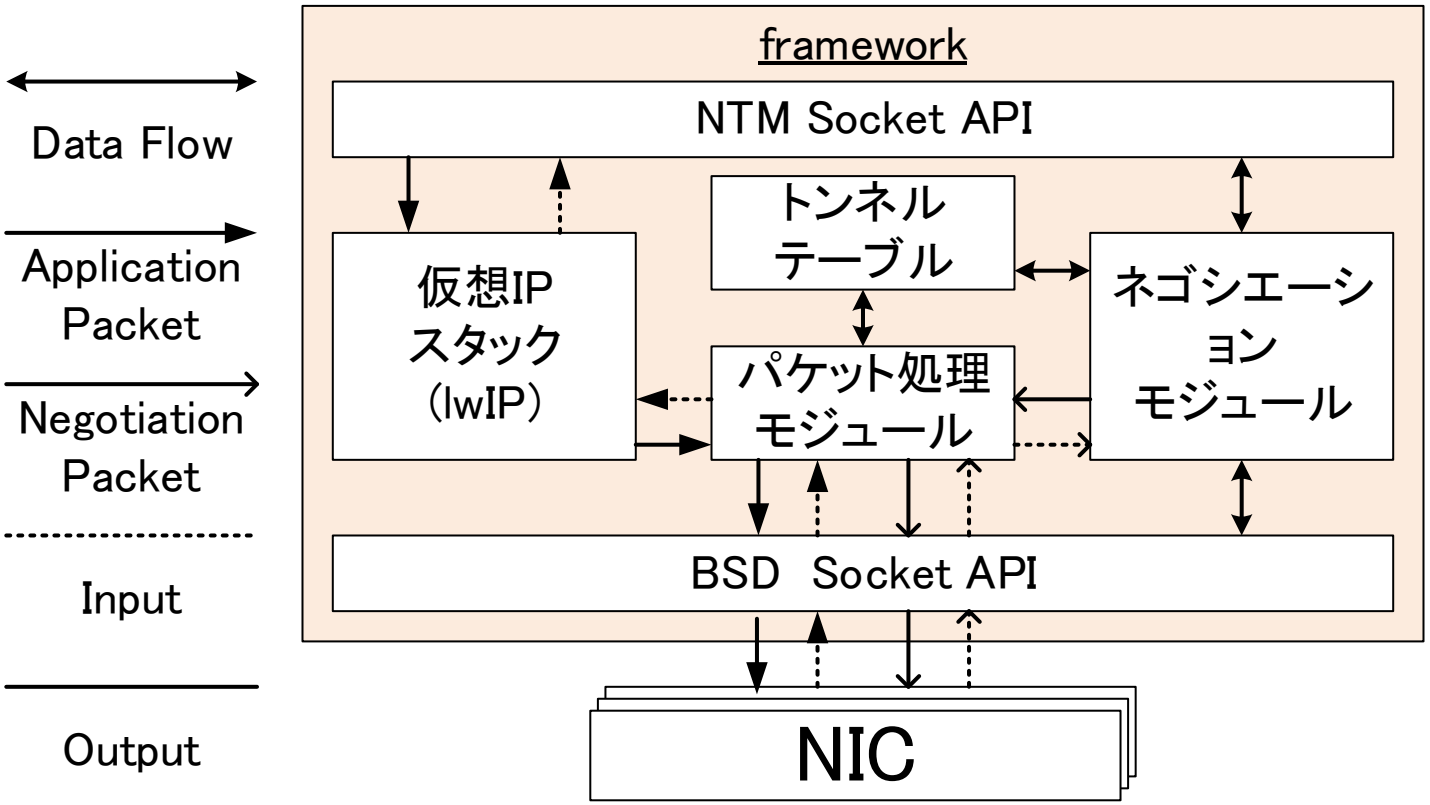
- NTMソケットAPIを利用してデータ送信
- 仮想IPアドレスを用いてIPパケット生成
- IPパケットの暗号化/MAC付与等
- BSDソケットAPIを利用して送信
- 実IPアドレスを用いてIPパケット生成・送信



MAC:メッセージ認証符号(メッセージが改竄されていないことを認証)

frameworkの実装(1)

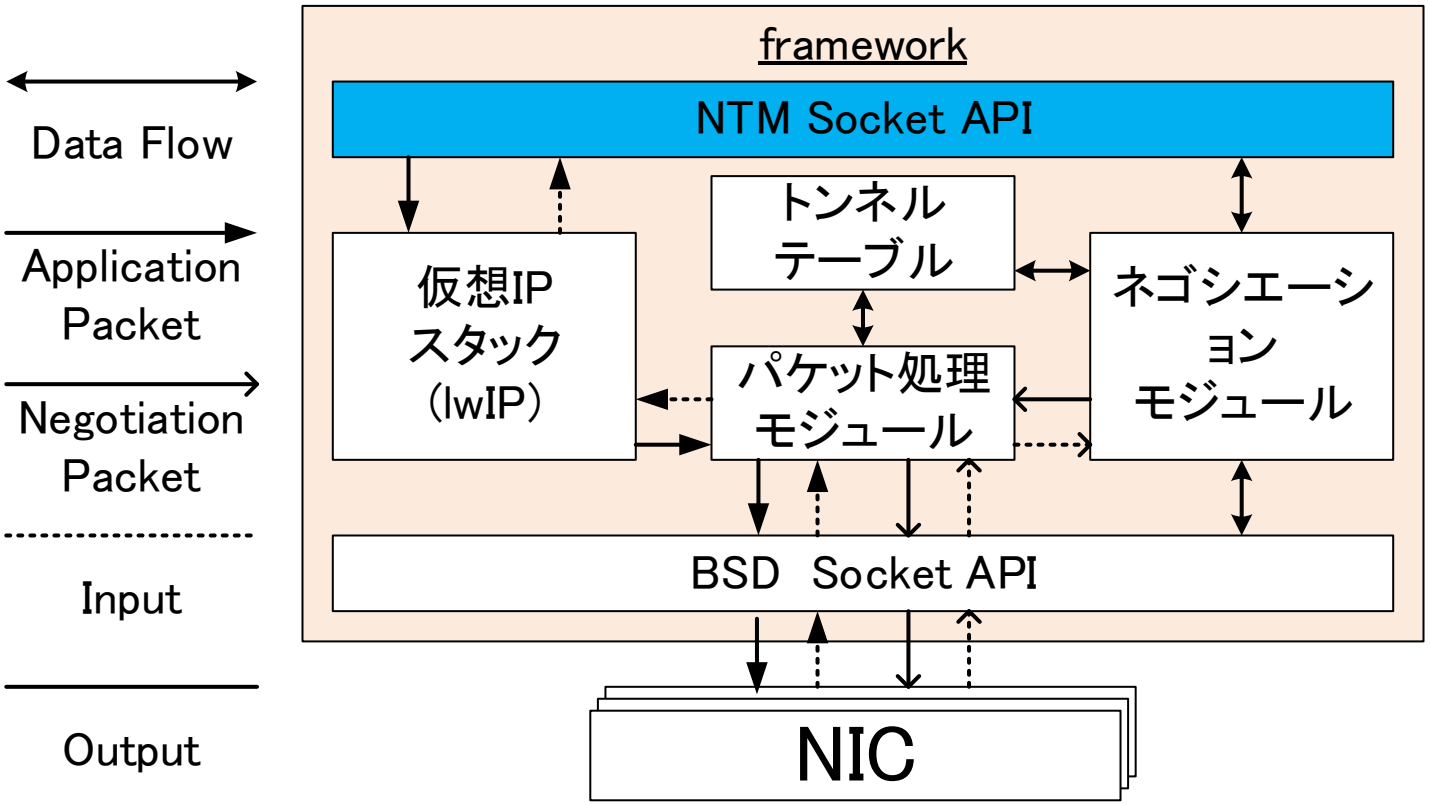
- framework自身に仮想IPスタックを実装
  - NTMソケットAPIで送受信するパケットを処理
    →(仮想IPアドレスに基づくIPヘッダ等が生成される)
- framework自身はBSDソケットAPIでパケットを送受信
  - 仮想IPスタックで生成されたパケットのカプセリングの実現



frameworkの実装(2)

NTM Socket API

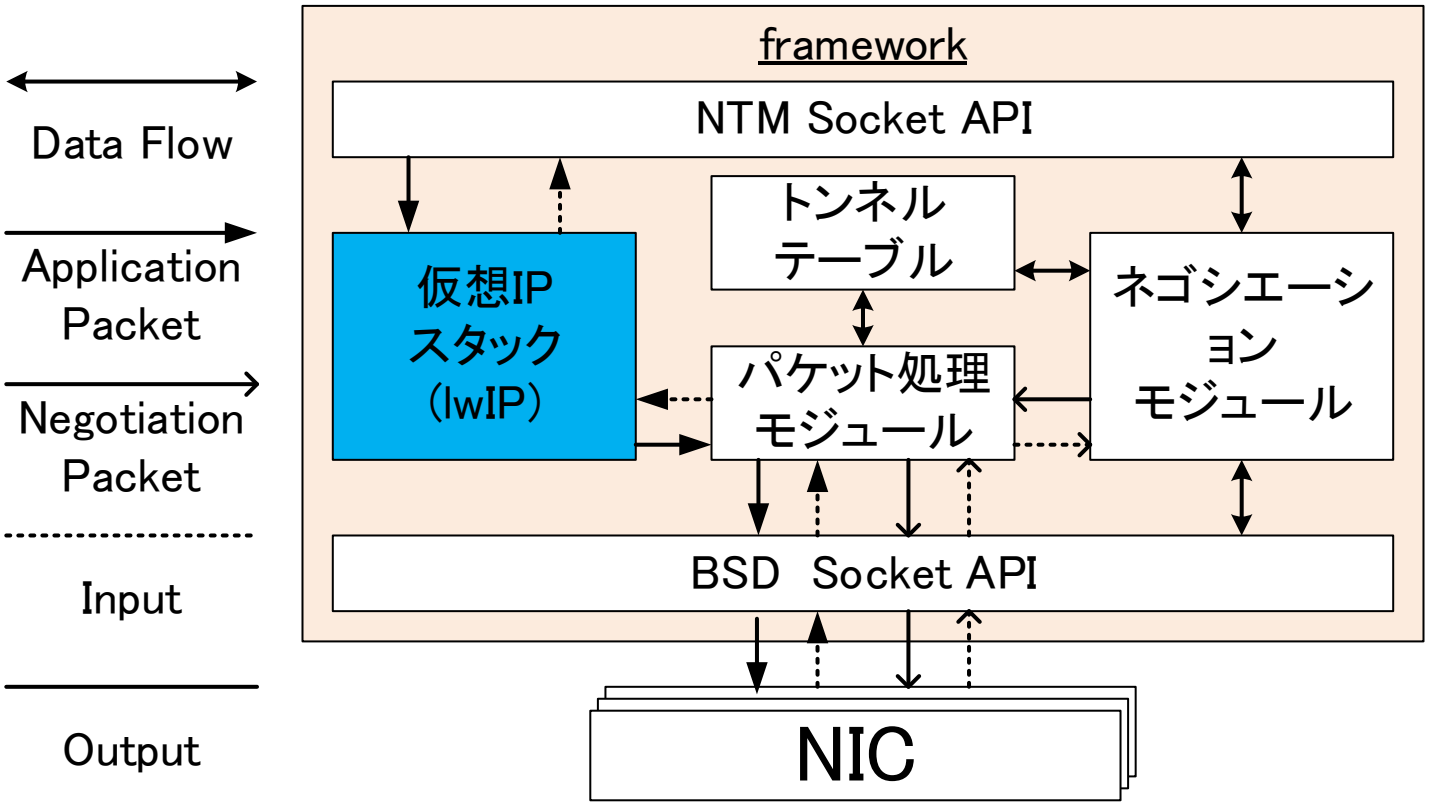
- BSDソケットAPI互換のソケットAPI
- 関数名の接頭語に「ntmfw_」を持つ（Ex.ntmfw_bind）
- 基本的に仮想IPスタックに処理をそのまま渡す
  (名前解決に関するAPIは、仮想IPアドレスが返り値となる)



frameworkの実装(3)

仮想IPスタック

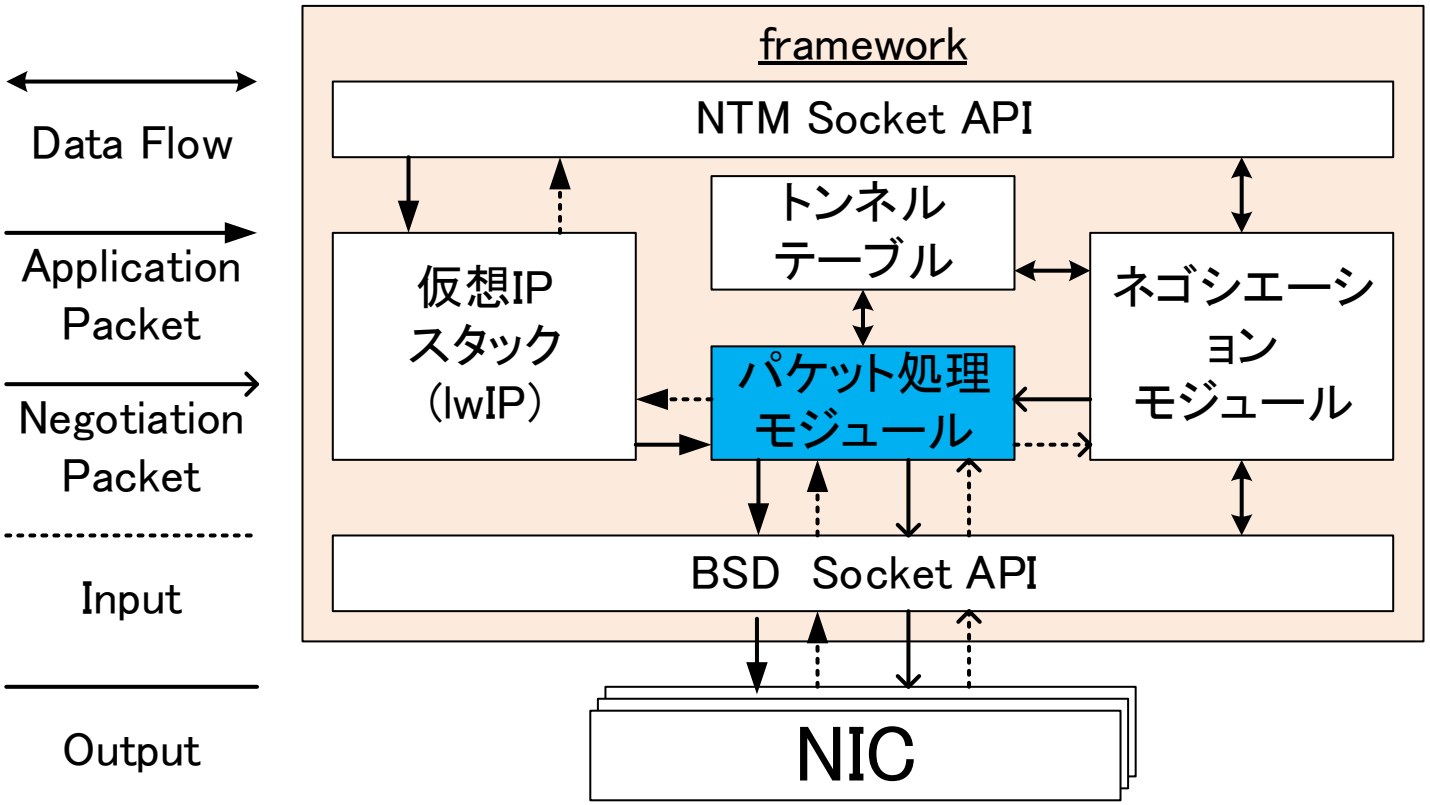
- A lightweight TCP/IP(lwIP)によるIPスタック
- コールバックの仕組みを用いてIPパケットのバッファを取得(カプセル化処理)
- 仮想IPパケットのバッファをinput関数に渡す(デカプセル化処理)



frameworkの実装(4)

パケット処理モジュール

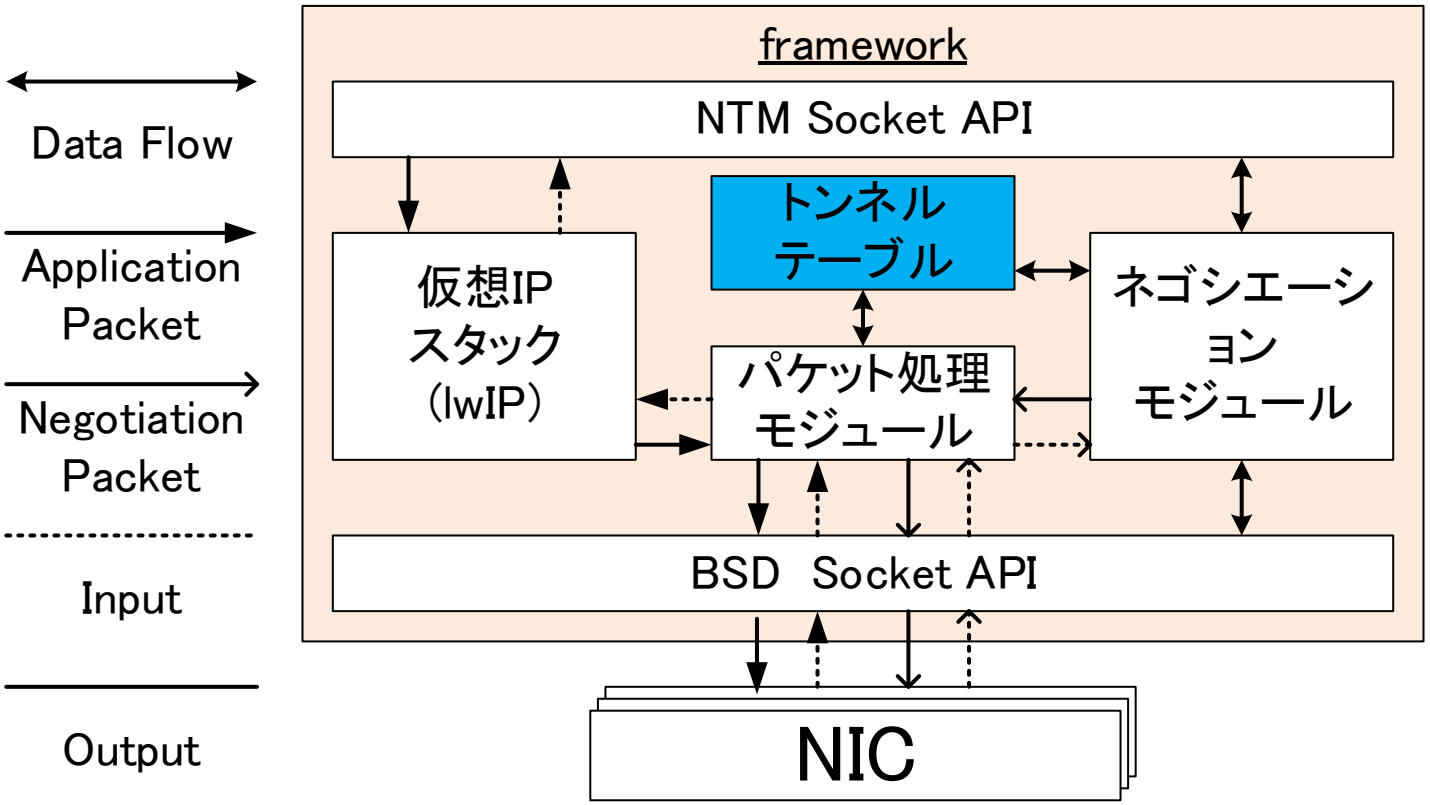
- 制御メッセージとカプセル化パケットの振分け
- 送信データパケットの暗号化/MAC付与
- 受信データパケットのMAC検証/復号等



frameworkの実装(5)

トンネルテーブル

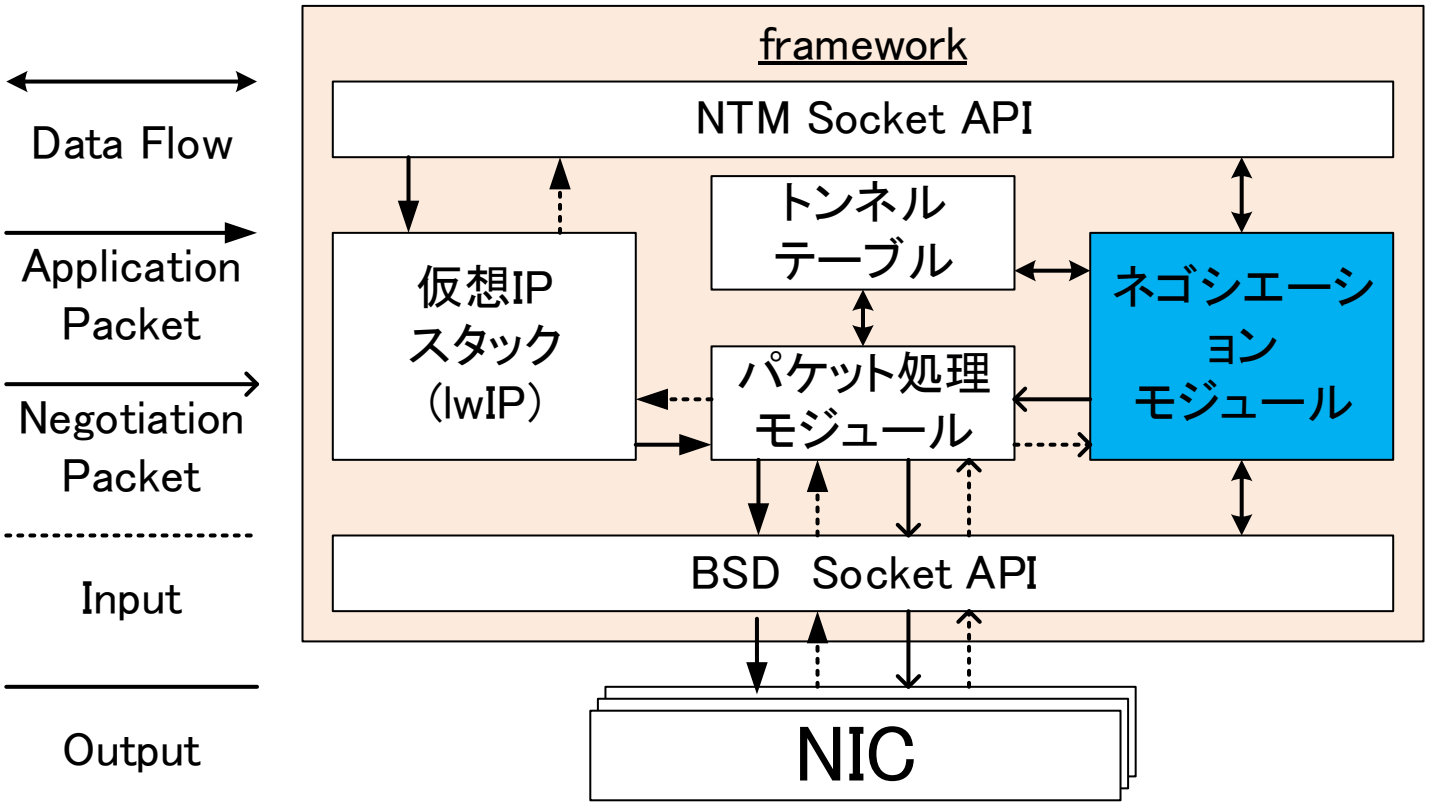
- 通信相手ごとにカプセル化に必要な情報を保持
- ハッシュテーブルとして実装(仮想IPアドレス等Hash Keyを5つ持つ)
- 一定期間参照が無い場合削除される



frameworkの実装(6)

ネゴシエーションモジュール

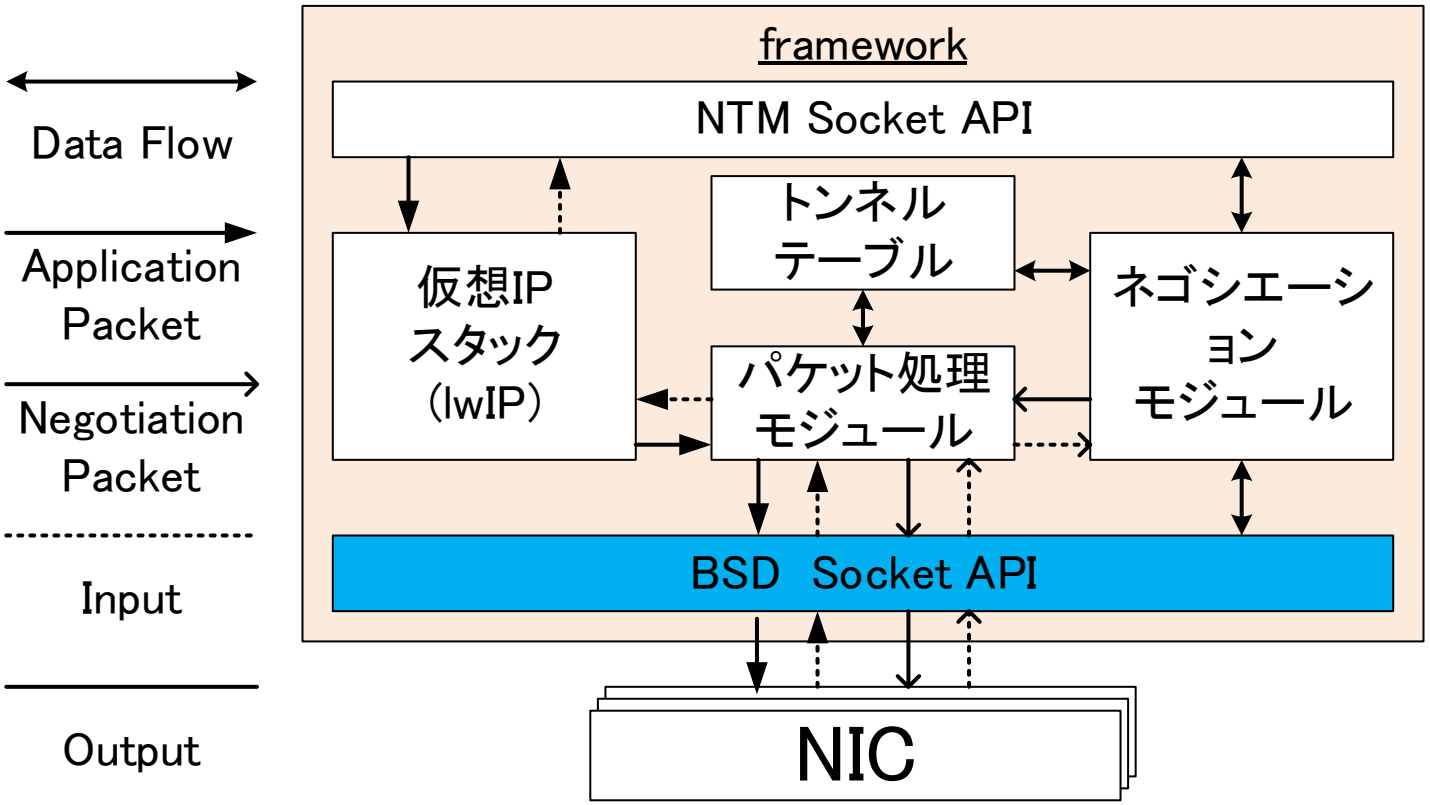
- NTMobileの処理に必要な制御メッセージの生成・処理
- 実IPアドレスの監視
- トンネルテーブルの管理(利用されていないエントリの削除等)
- DCや通信相手端末とのキープアライブ等



frameworkの実装(7)

BSD Socket API

- C言語のソケットAPI
- 制御メッセージやデータパケットの送受信
  （カプセル/デカプセル）



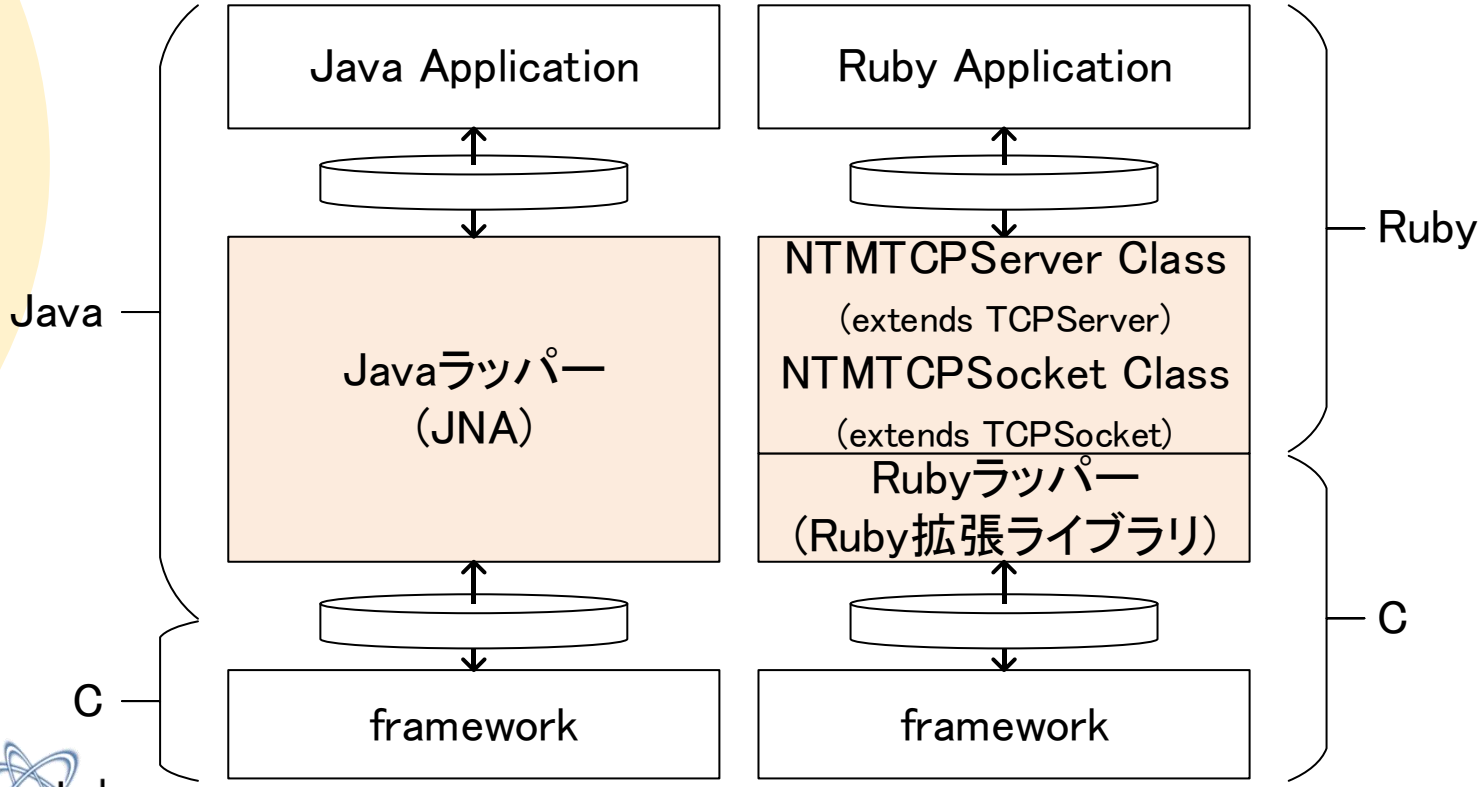
ラッパーの実装

Javaラッパー

- JNAを利用したラッパークラス作成
- NTMソケットAPIをJavaから呼び出し可能

Rubyラッパー

- Ruby拡張ライブラリを作成
- Ruby標準のソケットクラスを継承したクラスを生成



JNA:Java Native Access

試験概要

通信接続性試験

- 2台のNTM端末間でパケットの疎通確認
  - 実装したframeworkの動作確認
  - 異なるネットワーク間における通信の確立を実証

ラッパー試験

- NTM Javaアプリケーションと NTM Rubyアプリケーション間でパケットの疎通確認
  - 実装したラッパーの動作確認
  - 異なるプログラミング言語間でframeworkの利用が可能なことを実証

移動透過性試験

- 2台のNTM端末間で通信中に片方の端末のネットワークを切替
  - 確立した通信セッションが切断されず、継続されることを確認

通信接続性試験(1)

- 2台のNTM端末間でIPv4/IPv6パケットを送受信
  - NTM端末の属する実ネットワークのすべてのパターンで施行

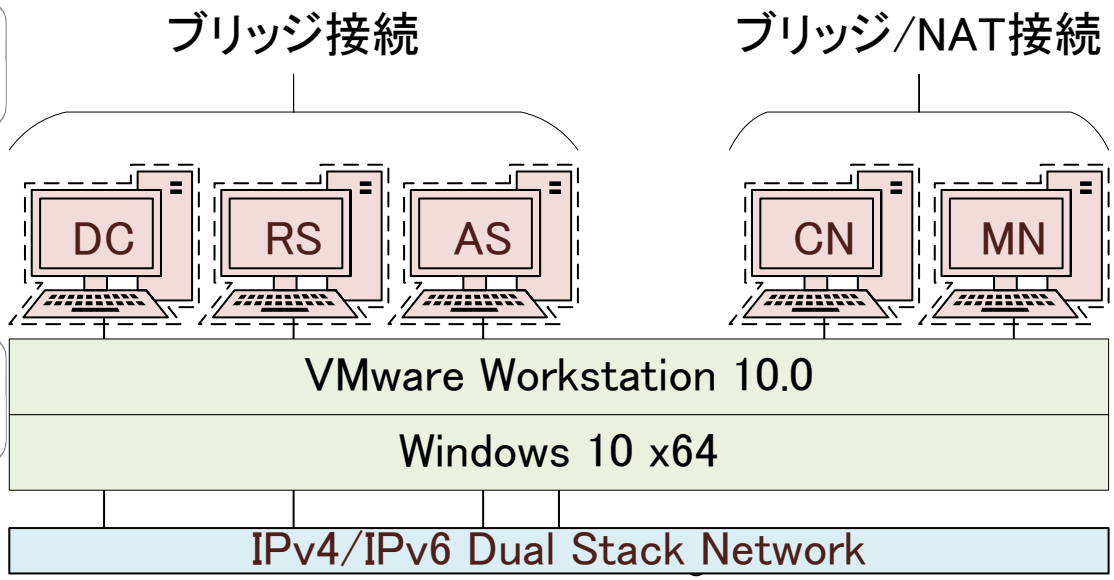
(NTM端末のIPv4またはIPv6の有効・無効により疑似的にネットワーク環境を再現)

ホストマシン

- Windows 10 Pro x64
- CPU:3.5GHz(4C8T)
- RAM:16GB

仮想マシン

- Ubuntu 14.04LTS x86
- CPU:2C2T割当
- RAM:2GB割当



※4C8T=4core8threads
2C2T=2core2threads

DC:Direction Coordinator RS: Relay Server
AS: Account ServerMN:Mobile Node CN:Correspondent Node

通信接続性試験(2)

- すべてのパターンでIPv4及びIPv6通信可能なことを確認
- 原則としてエンドツーエンドの最適経路による通信を確認
  - 異なるNAT間においても経路最適化機能により直接通信可能
  - IPv4-IPv6間のみRSを経由する通信経路となる

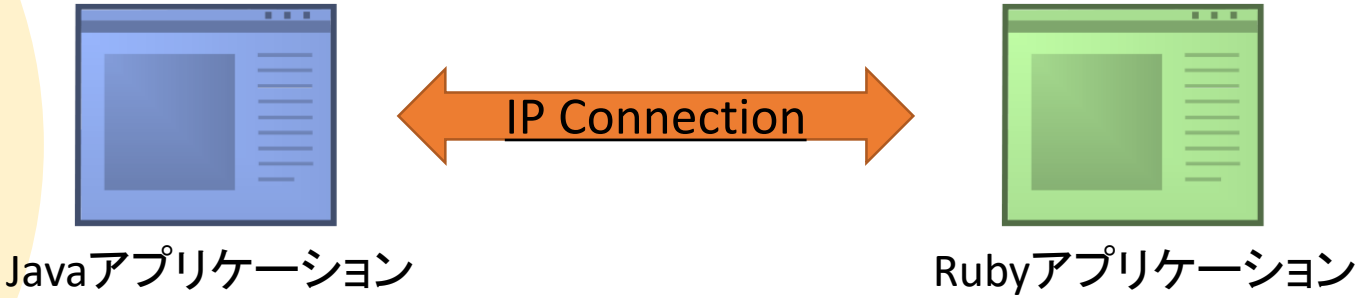
【試験結果】

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

◎:End-to-End　○:via RS

ラッパー試験

- JavaラッパーとRubyラッパーを用いた
  アプリケーション間で通信を施行
- 諸元は通信接続性試験と同一

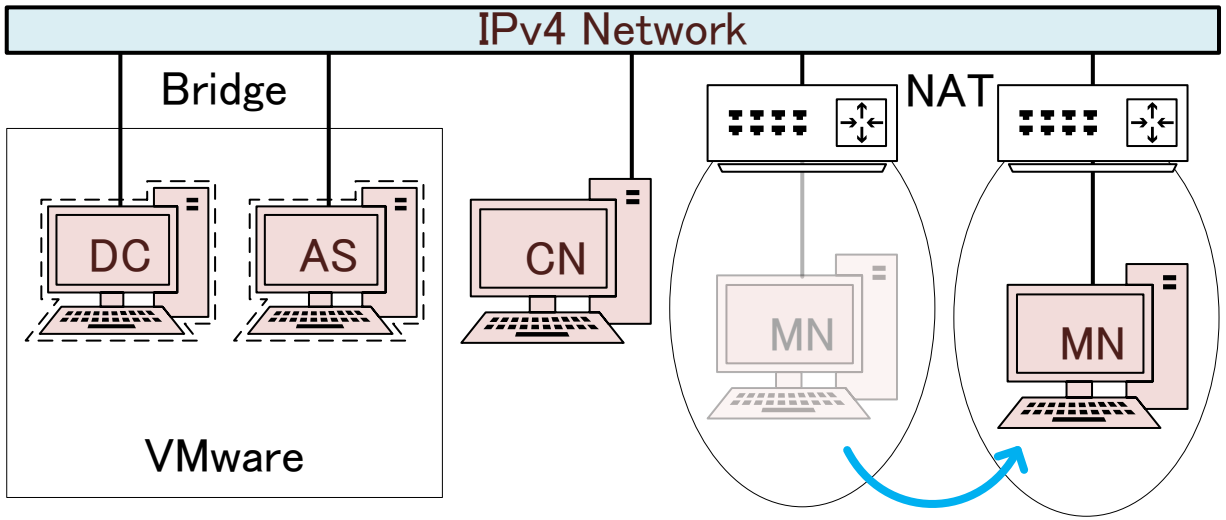


【試験結果】

- Java-Ruby間で仮想IPアドレスによる通信を
  確認した
  →異なるプログラミング言語による実装を確認

移動透過性試験(1)

- 通信中に片方のNTM端末の有線ケーブルを差し替える
  - MNの通信パケットをキャプチャ
  - 移動に要した時間を計測
    (トンネル再構築時間)
  - 試行回数　10回



【諸元】

AS/DC	NTM端末
• Ubuntu12.04 LTS x86 • CPU:3.4GHz,RAM:2GB(AS)/3GB(DC)	• Ubuntu14.04LTS x86 • CPU:3.4GHz(MN)/2.8GHz(CN)(2C4T),RAM:8GB

移動透過性試験(2)

【試験結果】

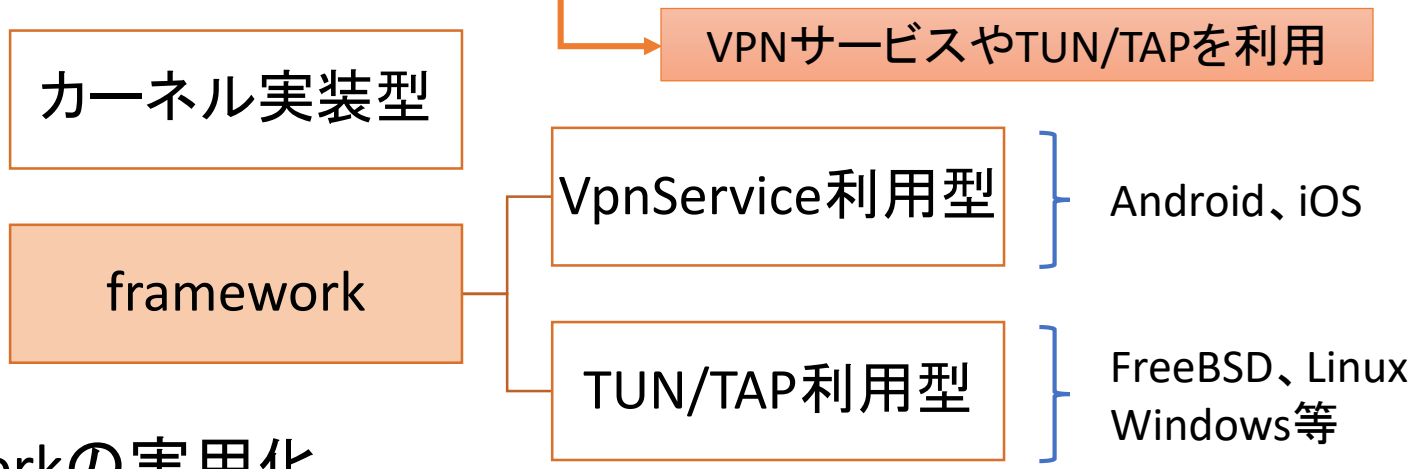
区分	処理時間(ms)	標準偏差
IPアドレスの更新に要した時間	5,789	2.564
トンネルの再構築に要した時間	83	0.030

- IPアドレスの更新に要する時間が支配的
  - DHCPによるIPアドレス取得
  - frameworkによるIPアドレス変化チェック
- トンネル構築に要する処理時間は僅か
  - インターネット(RTT30ms程度)を想定しても数百ミリ秒で処理が終わる
- 従前の実装においても1秒～8秒程度の通信断絶時間が発生

frameworkによる移動処理は従前実装と同等レベルで実現

frameworkの利活用・実用化

- frameworkはVpnService利用型及びTUN/TAP利用型のベース
  - ユーザー空間におけるネゴシエーション処理(カーネル実装型共通)
  - ユーザー空間のトンネルテーブル
  - ユーザー空間における移動検知
  - ユーザー空間のカプセル/デカプセル化処理等



- frameworkの実用化
  - iOSやAndroidへの適用(スマートフォンでの利用)
  - JavaやRubyを含む他プログラミング言語のソケットAPIラッパーの開発
  - frameworkを利用した商用サービスの提供

実装方式別の対比

	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	◎	○	△	△
保守ホスト	×	○	○	○
クロスプラットフォーム化	×	◎	×	○
シームレスハンドオーバー	○	×	×	×
root権限不要	×	○	○	×

Linux等(組込システム含)

センサデバイス

スマートフォン(Android/iOS)

Windows

まとめ

- 新仕様に基づくframeworkの実装
  - アプリケーションのみで移動可能なIPv4/IPv6通信を実現
    - NTMソケットAPIの提供
    - 市販のスマートフォンへ適用可能
- frameworkの動作検証及び評価
  - 接続ネットワーク問わずIPv4/IPv6通信可能なことを確認
  - Java及びRubyからframeworkを利用可能なことを確認
- framework実装完了に伴い実用化が加速
  - 各種スマートフォンへのNTMobileの適用の推進
  - VpnService利用型及びTUN/TAP利用型の実装
    - アプリケーションの改造が不要な方式
  - frameworkを用いた商用サービスの構想
    - 学民共同研究

[使用素材] ニコニ・コモンズ : nc80887,nc81017

ご清聴ありがとうございました

トンネルテーブル(1)

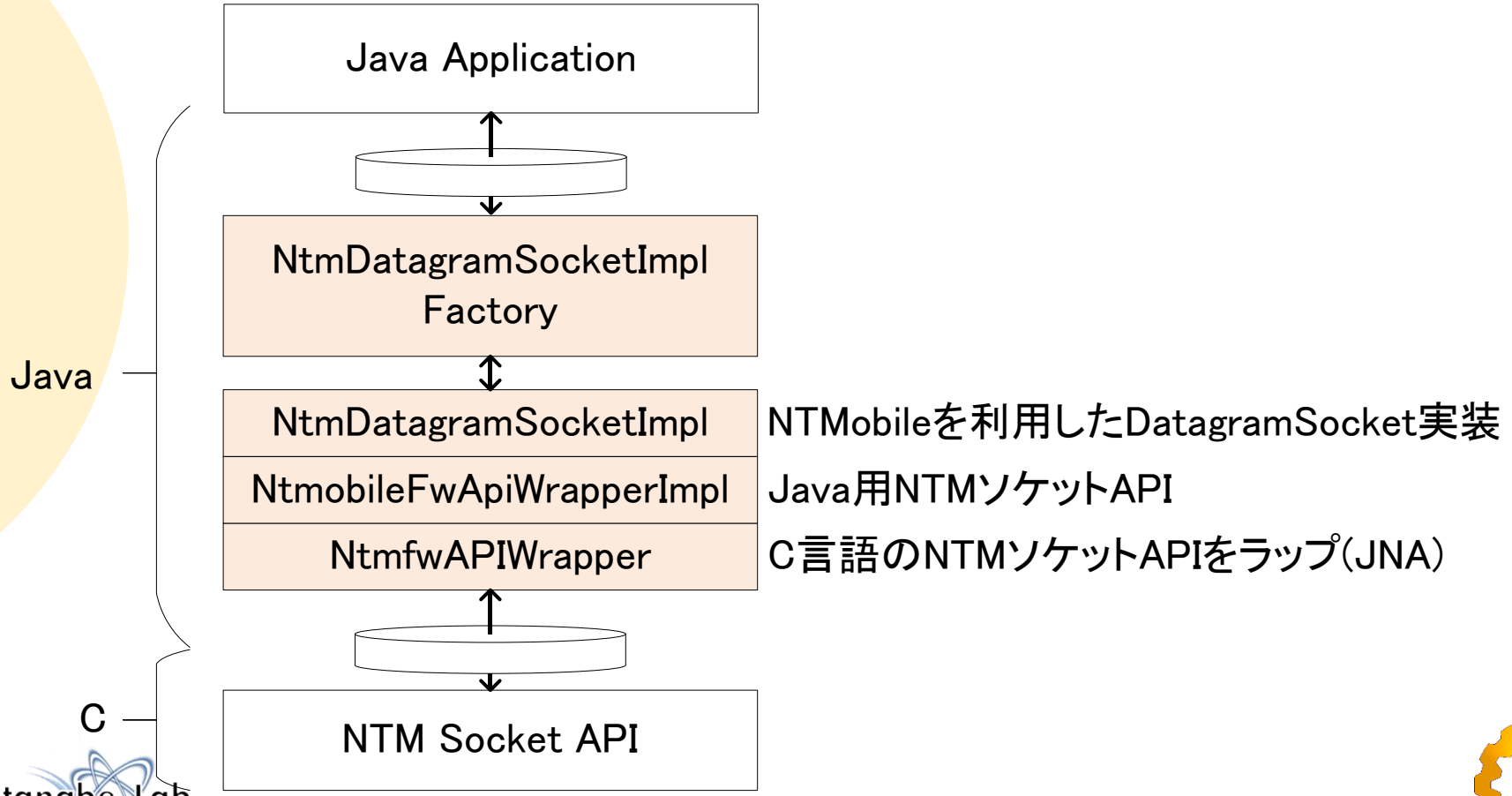
- カーネル実装型
  - カーネル空間に実装
  - 1対1ハッシュテーブル
  - ユーザー空間とNetlinkソケットで連携
    - ユーザー空間にトンネルテーブルとは別にノードテーブルを持つ。
    - トンネルテーブルとノードテーブルで保持する値が異なる。
      - トンネルテーブル：パケットカプセル化に用いるデータ
      - ノードテーブル：制御パケットの送受信に用いるデータ
    - トンネルテーブルとノードテーブルは同期する。
- フレームワーク組込型
  - ユーザー空間に実装
  - 多対1ハッシュテーブル
    - カーネル実装型の全テーブルの要素を複合して持つ。

トンネルテーブル(2)

メンバ	説明
PathID	通信ペアごとに生成される一意の値。ハッシュキー。
NodeID	通信相手端末のID。NTMobileにおいて端末ごとに一意の値を取る。ハッシュキー。
FQDN	通信相手のFQDN。ハッシュキー。
Virtual IPv4	通信相手の仮想IPv4アドレス。ハッシュキー。
Virtual IPv6	通信相手の仮想IPv6アドレス。ハッシュキー。
Real IPv4	通信相手の実IPv4アドレス。通信相手がIPv4未取得であればNULL。
Real IPv6	通信相手の実IPv6アドレス。通信相手がIPv6未取得であればNULL。
NAT IPv4	通信相手端末の接続するNATのIPv4アドレス。NAT配下でない場合はNULL。
RS IPv4	中継通信する際に経由するRSのIPv4アドレス。直接通信の場合はNULL。
RS IPv6	中継通信する際に経由するRSのIPv6アドレス直接通信の場合はNULL。
Dst IPv4	トンネル構築先のIPv4アドレス。DstIPv6と排他の関係にある。
Dst IPv6	トンネル構築先のIPv6アドレス。DstIPv4と排他の関係にある。
Data Key	カプセル化パケットの暗号化に用いる共通鍵。
Nego Key	制御メッセージの暗号化に用いる共通鍵。
Mutex	トンネルテーブルエントリの排他制御に用いられるMutex。
Timestamp	トンネルテーブルエントリの最終参照時間。一定時間経過後当該エントリは削除される。
Count	トンネルテーブルエントリを参照している変数の数。この値が1以上であれば削除されない。

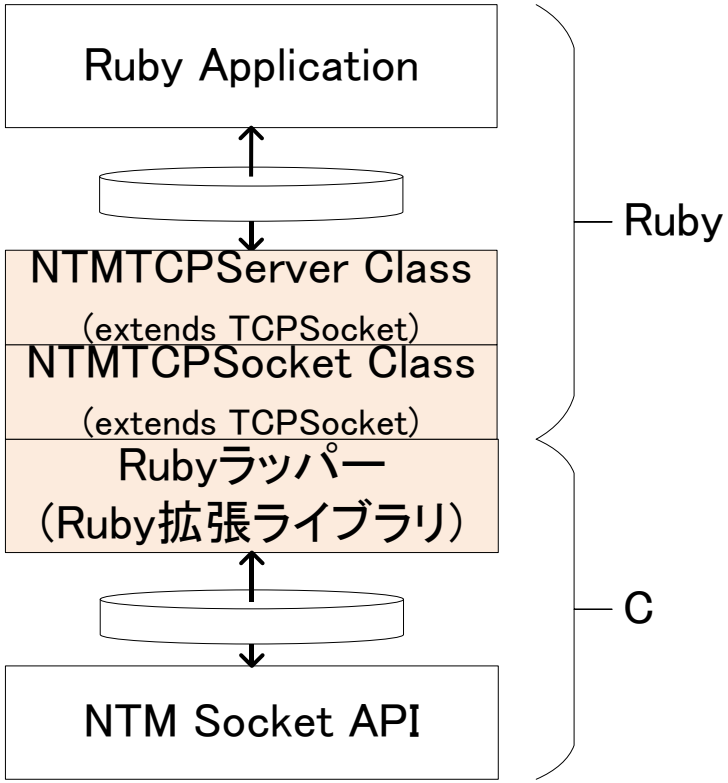
Javaラッパー

- Javaソケットのラッパークラスを実装中
  - UDP用ラッパーの開発完了
  - Javaのソケットクラスと同等の使い方が可能



Rubyラッパー

- C言語でRuby拡張ライブラリを作成
  - Rubyで呼び出し可能な関数を定義
    - NTMソケットAPI
- RubyからNTMソケットAPIを呼び出し既存クラスを上書き



セキュリティについて(1)

通信ペア	鍵の種類	説明
AS-MN(CN)	片方向公開鍵	ASの公開鍵証明書によるTLS通信
AS-DC	共通鍵	初回接続時にAS及びDCの公開鍵証明書によるTLS相互認証により共通鍵を共有
DC-RS	共通鍵	初回接続時にAS及びRSの公開鍵証明書によるTLS相互認証により共通鍵を共有
DC-MN(CN)	共通鍵	初回ログイン時にAS→DC及びAS→MN(CN)へ共通鍵を配送
MN-CN	共通鍵	トンネル構築時にDCからTempKeyとNegoKeyを取得。 MN(CN)がDataKeyを生成する。TempKeyで暗号化し、MN-CN間で共有。制御パケットはNegoKeyで暗号化 ※DataKeyは2重暗号 ※RS経由の場合、DCはNegoKeyのみRSへ配送 →DCやRSの管理者であってもDataKeyは知り得ない
MN(CN)-RS	共通鍵	DCから配布されるNegoKey

AES-CFB128bit/HMAC-MD5

※鍵長及びアルゴリズム可変可能

TempKey:一時鍵　NegoKey:制御メッセージ暗号化鍵　DataKey:カプセル化パケット暗号鍵

セキュリティについて(2)

攻撃手法	対策
リプレイ攻撃	すべてのパケットに付与するシーケンス番号チェック
パケット改竄	HMACによるメッセージ認証 (NTMヘッダ・仮想TCP(UDP)/IPヘッダ・ペイロードを認証)
パケット盗聴	全パケット暗号化
DDoS	全パケットUDPのためSYN Floodは成立せず。 UDP Floodは復号化処理前のHMACにより破棄 正規のHMAC(リプレイ攻撃)であってもPathID等により状態管理を行い、当該PathIDの処理が終了していれば復号化前に破棄
その他サーバー機器への攻撃	既存の技術による防御がそのまま利用可 (IPS/IDS/ApplicationFirewall等)

通信断絶時間について

- カーネル実装型における通信断絶時間
  - 移動に伴う通信断絶時間はOSに依存
  - IPv4ネットワークへの移動時

端末	通信断絶時間(ms)
Android 2.3.7	1,840
Android 4.2	7,754
iPhone 5	623

3G Network

- DHCPによる制約
  - IPアドレス取得後1～10秒の待機後に処理を開始
    (RFCによるDHCPの技術仕様)

カーネル実装型と同等の通信断絶時間

通信断絶時間の大半はNTMobile以外の要因

スループットについて

- カーネル実装型及びVPNサービス利用型におけるスループット
  - 統合的枠組みによる新仕様適用前(旧仕様)に基づく実測値
  - カプセル化部分は新旧で仕様変更がない
  - 通信を行う2端末は同一IPv4ネットワークに接続
  - 測定はAndroid 4.2.2を利用(CPU:1.2GHz 1GB RAM)

【測定結果】

実装モデル	スループット(Mbps)
NTMobile未実装	15.35
カーネル実装型	14.21
VPNサービス利用型	13.95

frameworkはVPNサービス利用型以上カーネル実装型未満想定

カーネル実装型：メモリコピー1回(カーネル)
framework: メモリコピー2回(ユーザー→カーネル)
VPNサービス利用型: メモリコピー3回(カーネル→ユーザー→カーネル)

新仕様とframeworkについて

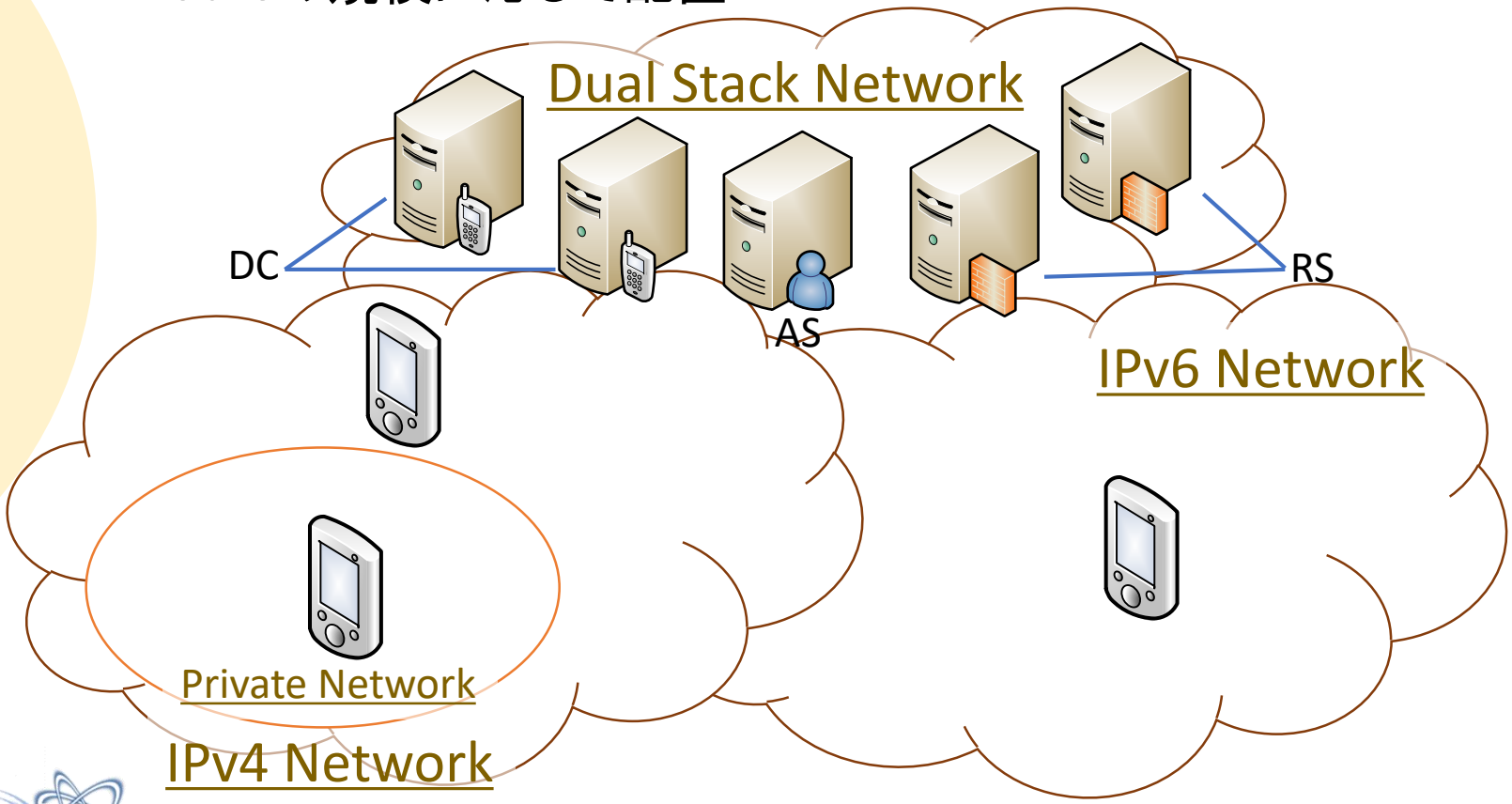
- カーネル実装型とフレームワーク組込型の混在
  - NTMobileはカーネル実装型から発展
    - カーネル実装型の仕様は据え置く
    - 新しい機能等はカーネル実装型と整合性を取るべく場合によって回りくどい仕様
  - スマートフォンに適用するためにプッシュ通知(GCM/APNS)への対応
    - キープアライブやDCとの通信に独自の動作仕様を定義
- 端末の位置等に応じて異なる動作仕様
  - 2台のNTM端末がNAT配下と片方がグローバル空間で異なるシーケンス
    - 送受信するメッセージが同じでも送信元や格納される情報に差異
- OpenIDや公開鍵認証等の新たな要求仕様

仕様を統合的枠組みとして再定義

カーネル実装型の仕様変更も容認して再定義
プッシュ通知等も標準機能として包含
→アプリケーションレベルで動作するframeworkの標準提供を意識

NTMobileのスケーラビリティ

- ASは唯一の認証装置
  - OpenID等を利用することで認証機能を外部へ出せる
- DC、RSは分散配置可能
  - NTMobileの規模に応じて配置

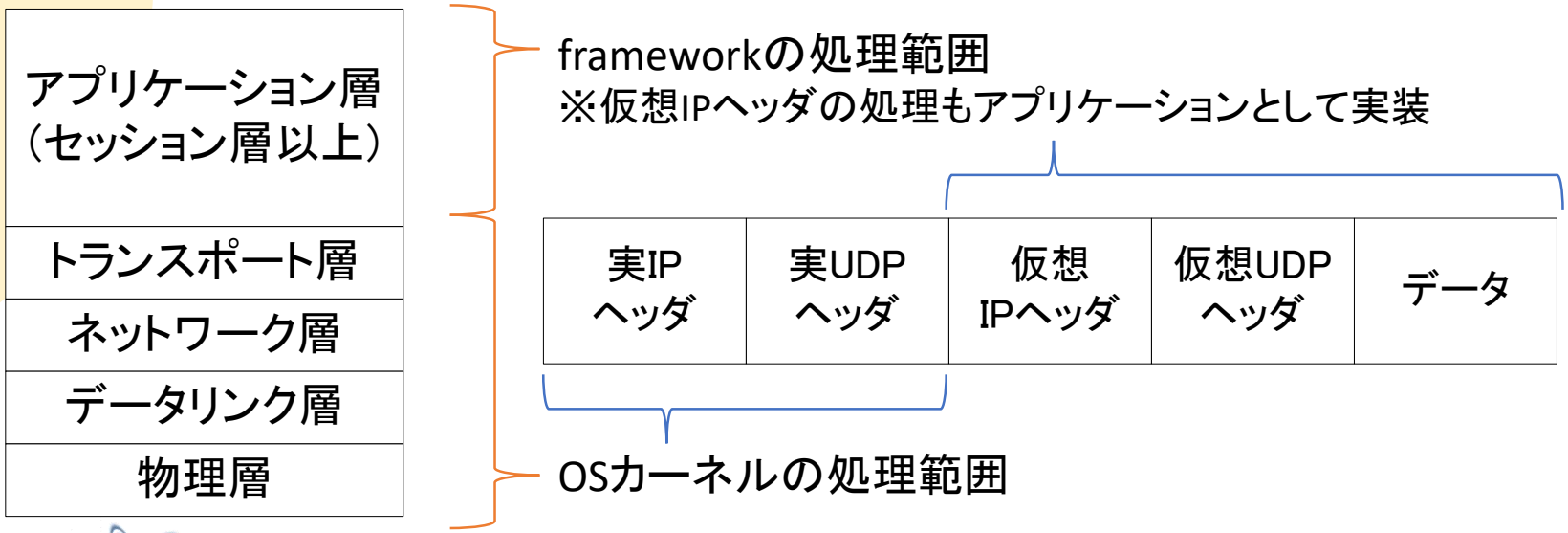


アプリケーション化するための検討

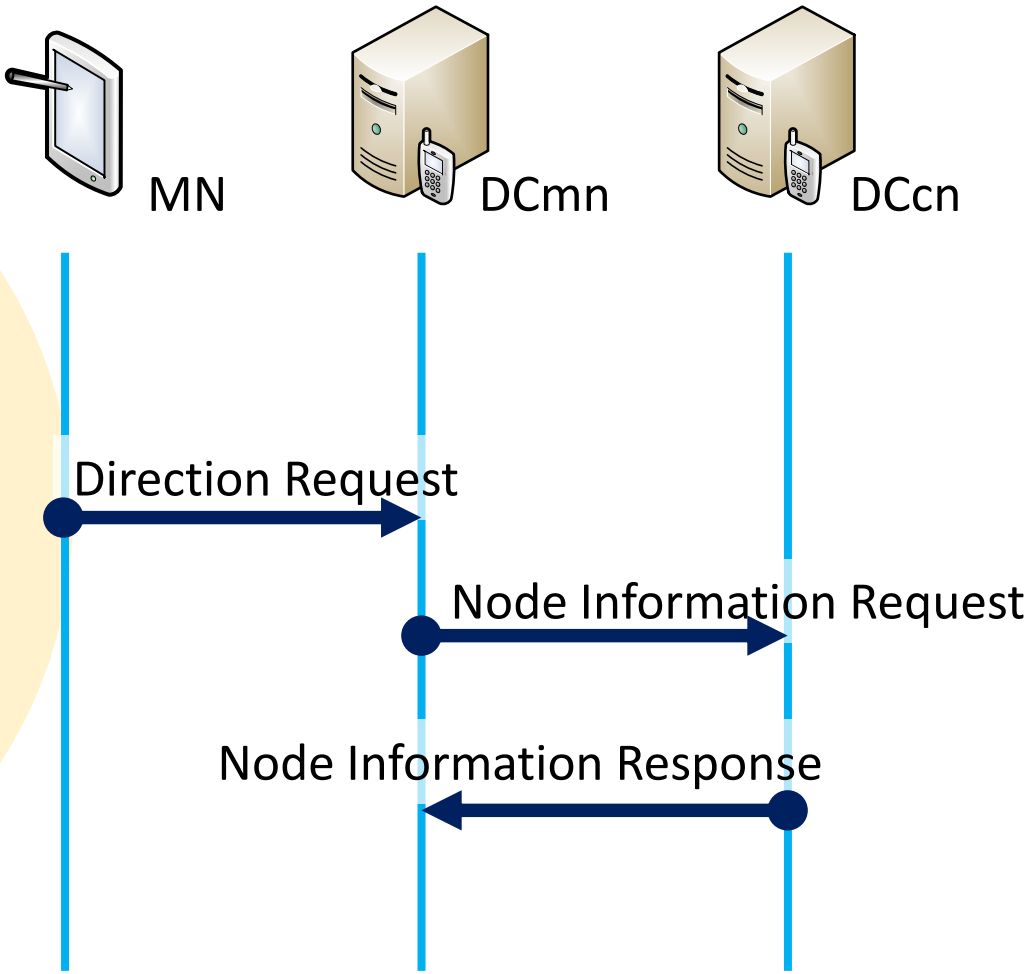
- カプセル化処理は、仮想IPヘッダをデータとしてUDP送信
  - 仮想IPパケットには一切手を加えない
  - 実IPヘッダ及び実UDPヘッダには一切手を加えない

→ トランスポート層以下のデータを処理しない

→ アプリケーションのみで処理可能



NTMobileのシーケンス(1)



FQDNcnの名前解決をフック

DNS要求をフック

DCmnへ経路指示要求

トンネル構築の指示要求

FQDNcnからFQDNdccn取得

FQDNcnのNSレコードを探索

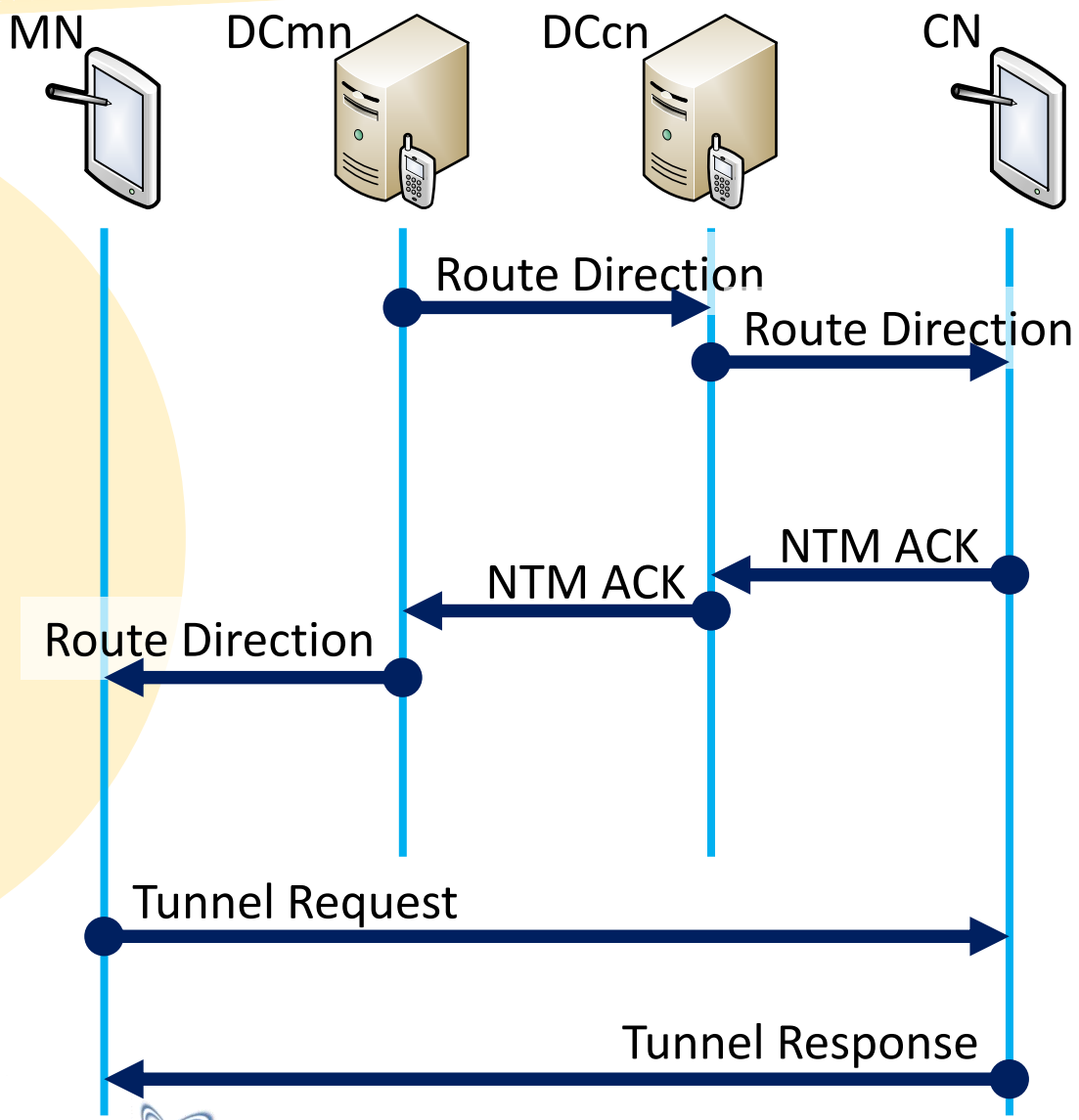
DCcnへCNの端末情報要求

CNの情報を要求

DCmnへCNの端末情報応答

CNの仮想IPアドレス等を応答

NTMobileのシーケンス(2)



経路生成用の共通鍵生成

CNへ経路指示

確認応答

MNへ経路指示

通信用の共通鍵生成

トンネル要求

トンネル応答