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

143430018 納堂 博史
渡邊研究室

1. はじめに

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(Internet Protocol version 4) アドレスの枯渇に伴って普及しつつある IPv6(Internet Protocol version 6) 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 [1]．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 (framework) をはじめ，AndroidOS の VPN(Virtual Private Network) サービスを利用する VpnService 利用型，TUN/TAP デバイスを利用する TUN/TAP 利用型の実装がなできていなかった．本稿では，新仕様に基づき framework の実装仕様を検討し，実装及び動作検証並びに評価を行った．動作検証及び評価を行った結果，実装した framework が正しく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 framework を Java[1] 及び Ruby[2] から利用可能にするためのラッパーを設計し，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．

2. NTMobile について

NTMobile は，NTMobile の機能を有する NTM 端末，NTM 端末のアドレス情報等の管理及び UDP トンネルの構築指示を行う DC(Direction Coordinator)，NTM 端末間で直接通信できない場合や，一般端末と通信を行う際にパケット中継を行う RS(Relay Server)，NTM 端末の認証を行う AS(Account Server) で構成される．

NTM 端末は，初回起動時に AS にログインし，DC との通信に必要な共通鍵等の情報を得る．この情報を用いて DC に自端末の実 IP アドレス等の情報を登録し，移動により変化しない仮想 IPv4 アドレスおよび仮想 IPv6 アドレスを取得する．アプリケーションは，この仮想 IPv4/IPv6 アドレスを用いることにより，自端末の属するネットワークに依存せず，IPv4/IPv6 通信を行うことができる．アプリケーションの送信する仮想 IPv4/IPv6 パケットは，暗号化及び MAC(Message Authentication Code) 付与の後，実 IP アドレスでカプセル化されて実ネットワーク上を転送される．UDP トンネルは原則としてエンドツーエンドで構築され，パケット中継が必要な場合のみ RS を経由する．

[1] http://www.java.com/
[2] http://www.ruby-lang.org/

3. framework が提供する機能

framework は，NTMobile のすべての機能をユーザー空間で実現する実装モデルである．アプリケーションは，C 言語の標準ソケット API(BSD ソケット API) の代わりに framework の提供するソケット API(NTM ソケット API) を利用してソケット通信を行う．framework は内部に TCP/IP の実装 (仮想 IP スタック) を持ち，仮想 IP アドレスをアプリケーションに提供する．アプリケーションの送信するデータは，この仮想 IP スタックによって仮想 IP アドレスを用いた IP ヘッダが付与される．このバッファを BSD ソケット API で UDP 送信することで，UDP によるカプセル化を実現する．また，NTM ソケット API は，BSD ソケット API の提供する API のほかに，NTMobile 独自の API を提供する．これらの API は NTMobile の初期化や終了のほか，framework が保持する一部情報をアプリケーション開発者が取得するための機能を提供する．

4. 実装

4.1 framework

図 1 に framework のモジュール構成図を示す．framework は仮想 IP スタックに lwIP(A Lightweight TCP/IP stack)[3] を用い，NTM ソケット API は基本的に lwIP に接続される．lwIP の仮想ネットワークインターフェースには仮想 IPv4/IPv6 アドレスを登録する．このため，アプリケーションが送信するパケットは lwIP により仮想 IP アドレスを用いて TCP/IP または UDP/IP ヘッダが付与され，コールバック関数によって lwIP からパケット操作モジュール (PMM:Packet Manipulation Module) に処理が移る．PMM はこのパケットに対して暗号化及び MAC 付与を行った後，BSD ソケット API を用い，通信相手端末の実 IP アドレス宛に送信する．また，PMM は，カプセル化パケットを BSD ソケット API を用いて受信し，当該パケットを MAC 検証及び復号し，lwIP に処理を渡すことで，lwIP に仮想 IP アドレスによるパケットがプッシュされる．アプリケーションは，NTM ソケット API の recv 関数等によりこのパケットをポップし，データを受信する．以上の処理により，アプリケーションは仮想 IP アドレスを用いてパケットの送受信を行うことができる．

4.2 Java ラッパー

図 2 に Java ラッパーのモジュール構成を示す．NTM ソケット API を Java から呼び出すため，JNA(Java Native Access) を用いることで，C 言語で記述された関数及び構造体を Java で利用できるようにした．Java アプリケーションは，Java ラッパーを通じて NTM ソケット API を利用することにより，NTMobile の機能を利用することができる．

4.3 Ruby ラッパー

図 3 に Ruby ラッパーのモジュール構成を示す．NTM ソケット API を Ruby から呼び出すため，Ruby 拡張ライブラリを設計した．Ruby 拡張ライブラリは，C 言語側で Ruby

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

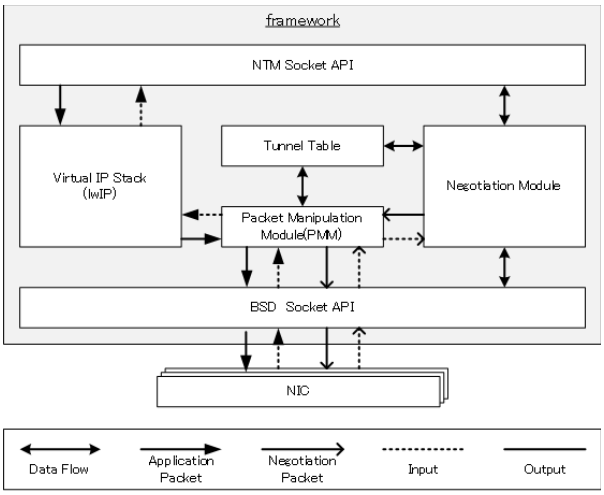


図 1: framework のモジュール構成図

から呼び出し可能な関数を定義し，これを Ruby から呼び出す．また，Ruby の TCPSocket クラス及び TCPServer クラスを継承した NTMTCPSocket クラス及び NTMTCPServer クラスのプロトタイプを実装した．アプリケーションは，Ruby 標準の TCPSocket クラスまたは TCPServer クラスの代わりに，NTMTCPSocket クラスまたは NTMTCPServer クラスのオブジェクトを生成すればよい．

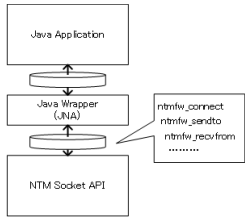


図 2: Java ラッパー

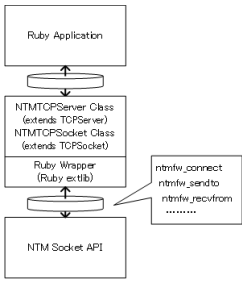


図 3: Ruby ラッパー

5. 動作検証及び評価

IPv4/IPv6 デュアルスタックネットワークに接続したホストマシン上に，DC，RS，AS，NTM 端末 2 台の計 5 台の仮想マシンを構築し，動作検証を行った．NTM 端末は，NAT 接続またはブリッジ接続でネットワークに参加する．その他の機器はブリッジ接続でネットワークに参加する．また，NTM 端末は，IPv4 または IPv6 の有効/無効を切り替えることにより，IPv4 グローバル，IPv4 プライベート，IPv6 の各ネットワークに任意に接続できる．この環境において，NTM ソケット API を利用するテストプログラムを NTM 端末 2 台の間で起動し，相互に IPv4 及び IPv6 で TCP 通信を行った．この結果を表 1 に示す．端末の属するネットワークに依存せず，アプリケーションが IPv4 及び IPv6 で通信を行うことができることを確認した．

また，IPv4 ネットワークに NAT を 2 台接続し，有線ケーブルを差し替えることで 2 台の NAT 間を移動する NTM 端末をシミュレートした．一方の NAT に接続した状態で UDP トンネルを構築し，通信中に移動した時の通信断絶時間を 10 回計測し，その平均値を算出した．試験の結果を表 2 に示す．移動に伴う UDP トンネルの再構築にかかる時間は僅かで，新しい IP アドレスの取得に多くの時間を要することがわかった．インターネット利用時のネットワーク遅延を考慮した場合であっても，新 IP アドレスの取得に要する時間の方が支配的である．また，この結果はカーネル実装型の結果と同等であり，両者で移動処理に係る性能差はないと言える．

表 1: 疎通試験結果

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

◎:end-to-end ○:via RS

表 2: 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

また，実装した Java ラッパー及び Ruby ラッパーを利用し，Java アプリケーションと Ruby アプリケーション間で TCP 通信を行った．この結果，Java と Ruby 間で framework を利用した TCP コネクションを確認した．このコネクションは仮想 IP アドレスにより維持されるため，通信中にネットワークの切替が可能である．

6. 今後の展望

framework の実装は，VpnService 利用型及び TUN/TAP 利用型のベースとなり，両者の実装の加速が期待できる．カーネル実装型を含み，これらの実装は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，カーネル実装型は Linux 系 OS，VpnService 利用型は AndroidOS に利用環境が限定され，TUN/TAP 利用型も TUN/TAP デバイスドライバが提供される OS に限定される．スマートフォンへの適用を考えると，root 権限を要さず，iOS や Android を含む Linux，Windows などに適用可能な framework の有用性は高い．また，実効速度面ではカーネル実装型には及ばないものの，カーネル空間で処理されたパケットをユーザー空間に戻して処理を行い，再度カーネル空間を通す VpnService 利用型及び TUN/TAP 利用型と比して，framework は有利であると推測される．一方で，アプリケーションの改造を要することから，各プログラミング言語の標準的なソケットライブラリのラッパーを用意し，開発者の実装コストを下げる努力が必要である．今後，framework の他言語ラッパーの開発と，TUN/TAP 利用型等のアプリケーション改造が不要な実装モデルの提供により，NTMobile を利用しやすい環境整備を目指す．

7. まとめ

本稿では，NTMobile における framework の実装を行い，動作検証及び評価を行った．この結果，framework を利用する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，通信端末間で IPv4/IPv6 通信を行うことができることを確認した．また，複数のプログラミング言語で framework を利用できることを示し，相互に通信できることを確認した．framework の実装により VpnService 利用型や TUN/TAP 利用型の実装が加速し，アプリケーションの改造が不要な実装の実用化が期待できる．

参考文献

[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，情処研報，Vol.2015-MBL-77,No.20, pp.1-8(2015).

平成28年度　修士論文公聴会
（2017年2月9日）

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

所　　属	名城大学大学院理工学研究科 情報工学専攻渡邊研究室
学籍番号	143430018
氏　　名	納堂　博史

Watanabe Lab.

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

関連研究

- 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

- 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)

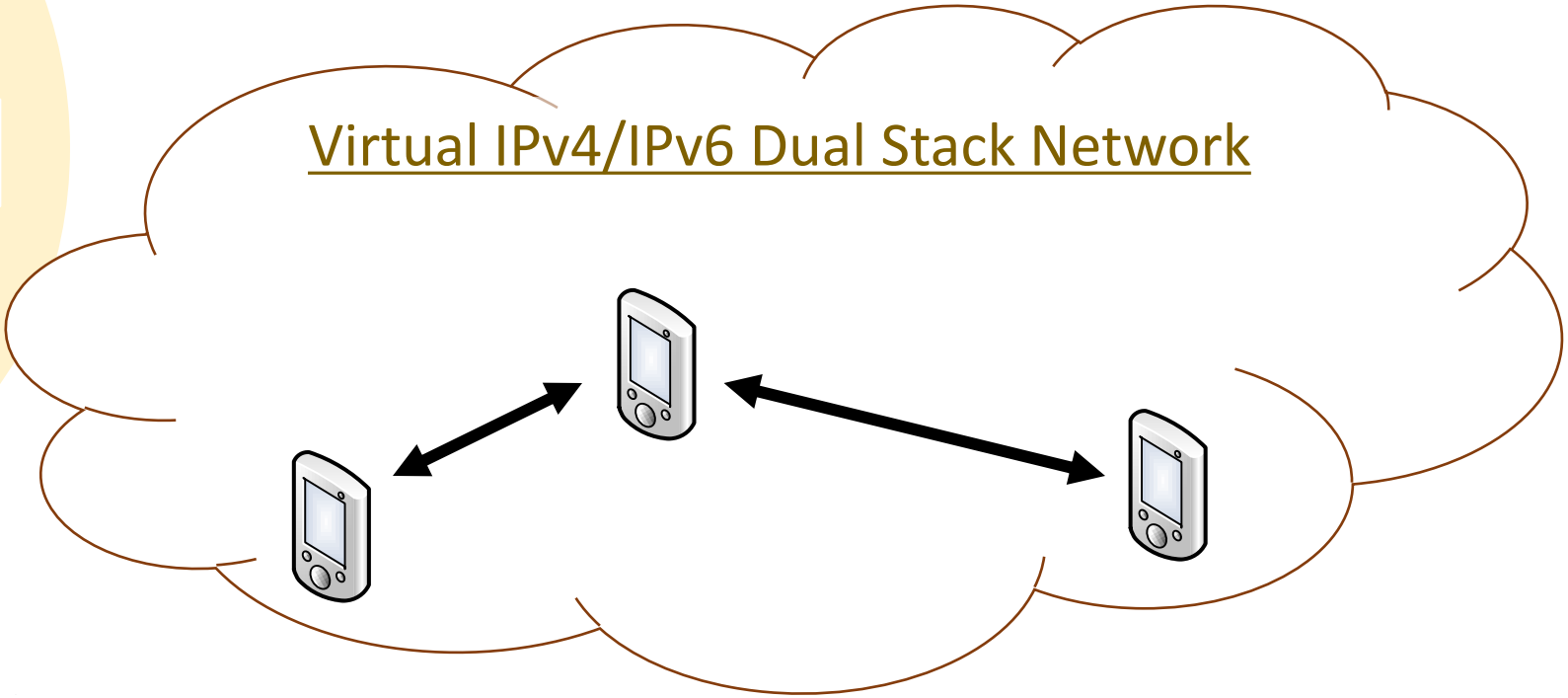
課題

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

いずれの方式もアプリケーションレベルで実現できない

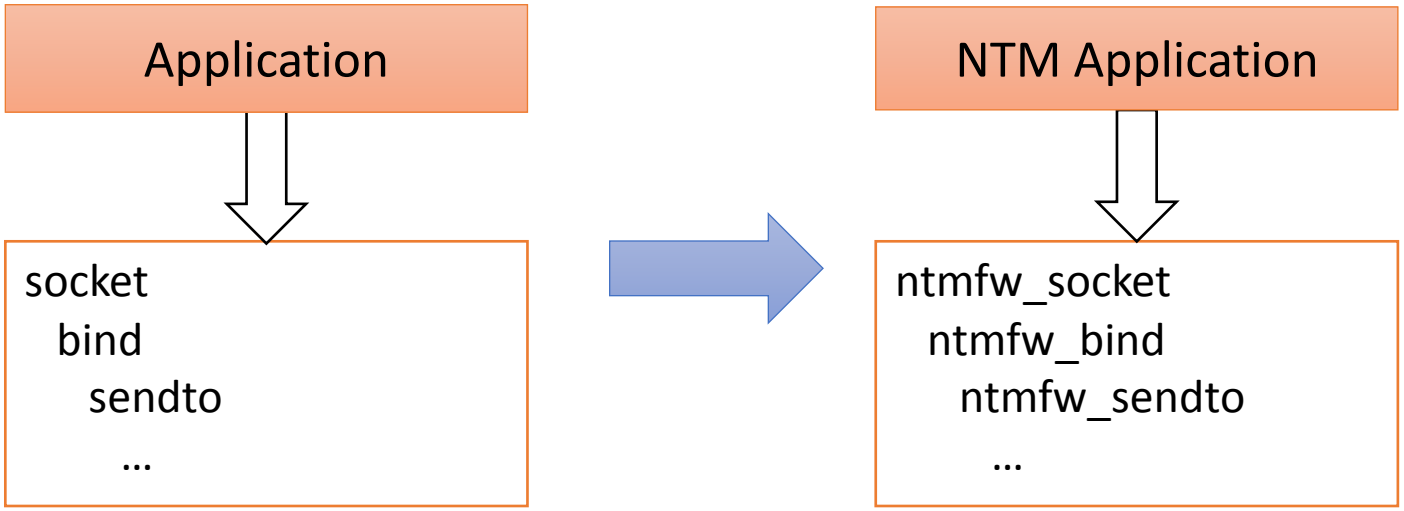
NTMobileの概要

- 仮想的なIPv4/IPv6デュアルスタックネットワークを提供
  - エンド端末への仮想IPv4/仮想IPv6アドレスの提供
  - アプリケーションは変化しない仮想IPアドレスでパケット通信
  - 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
  - 端末移動時は外側のヘッダのみが変化



frameworkの概要

- NTMobileの処理をすべてユーザー空間に実装
  非rootで使用可能
  特定のOSに依存する機能を使わない
- 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

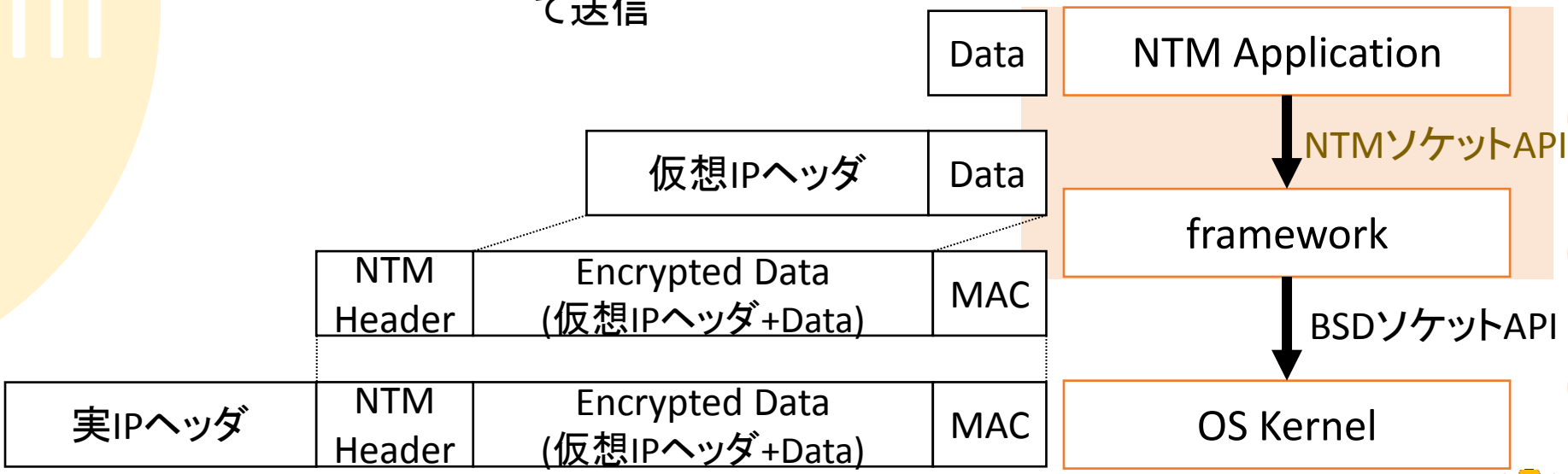
frameworkの動作

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

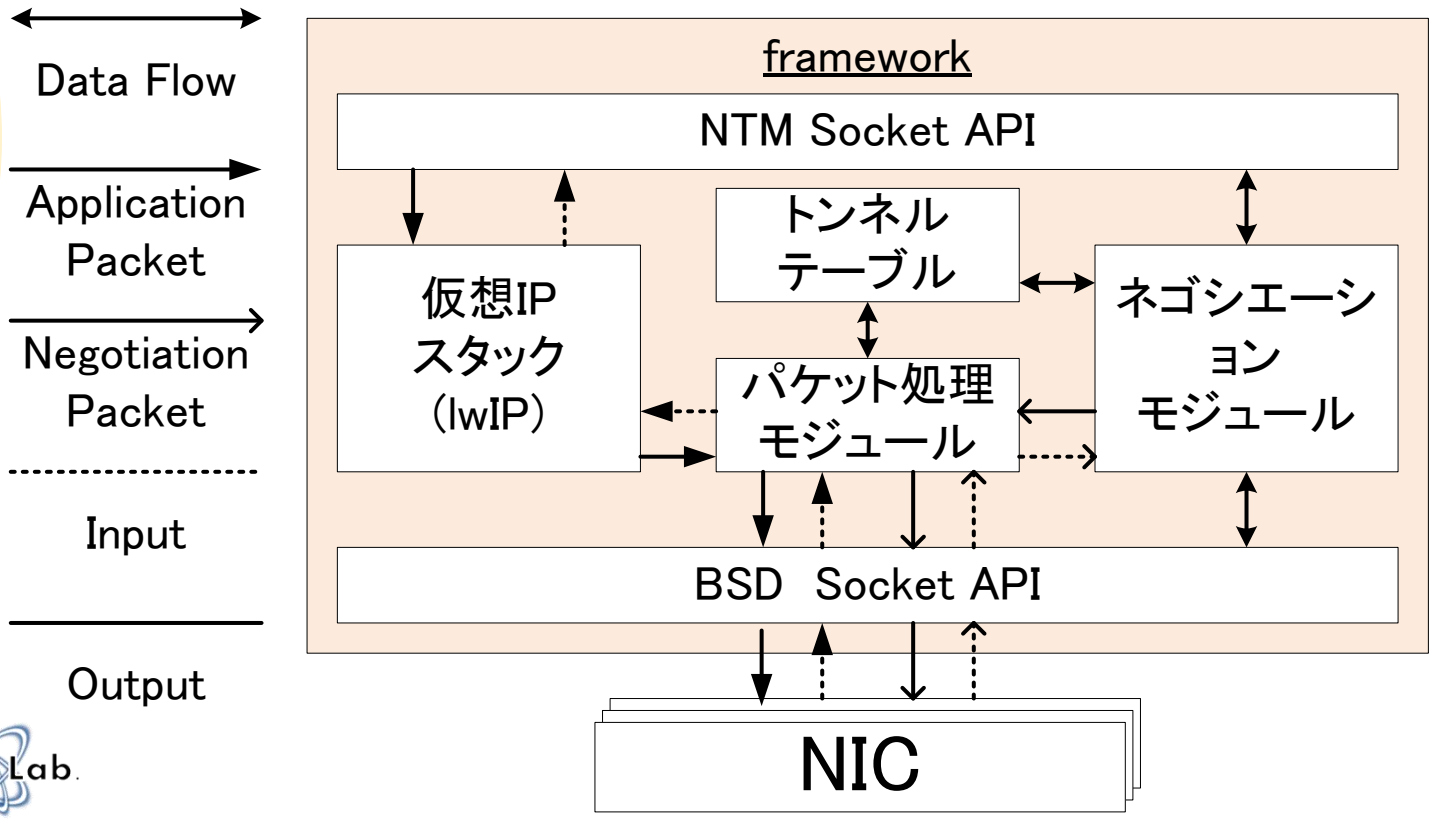
- 実IPアドレスを用いてIPパケット生成・送信



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

- 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



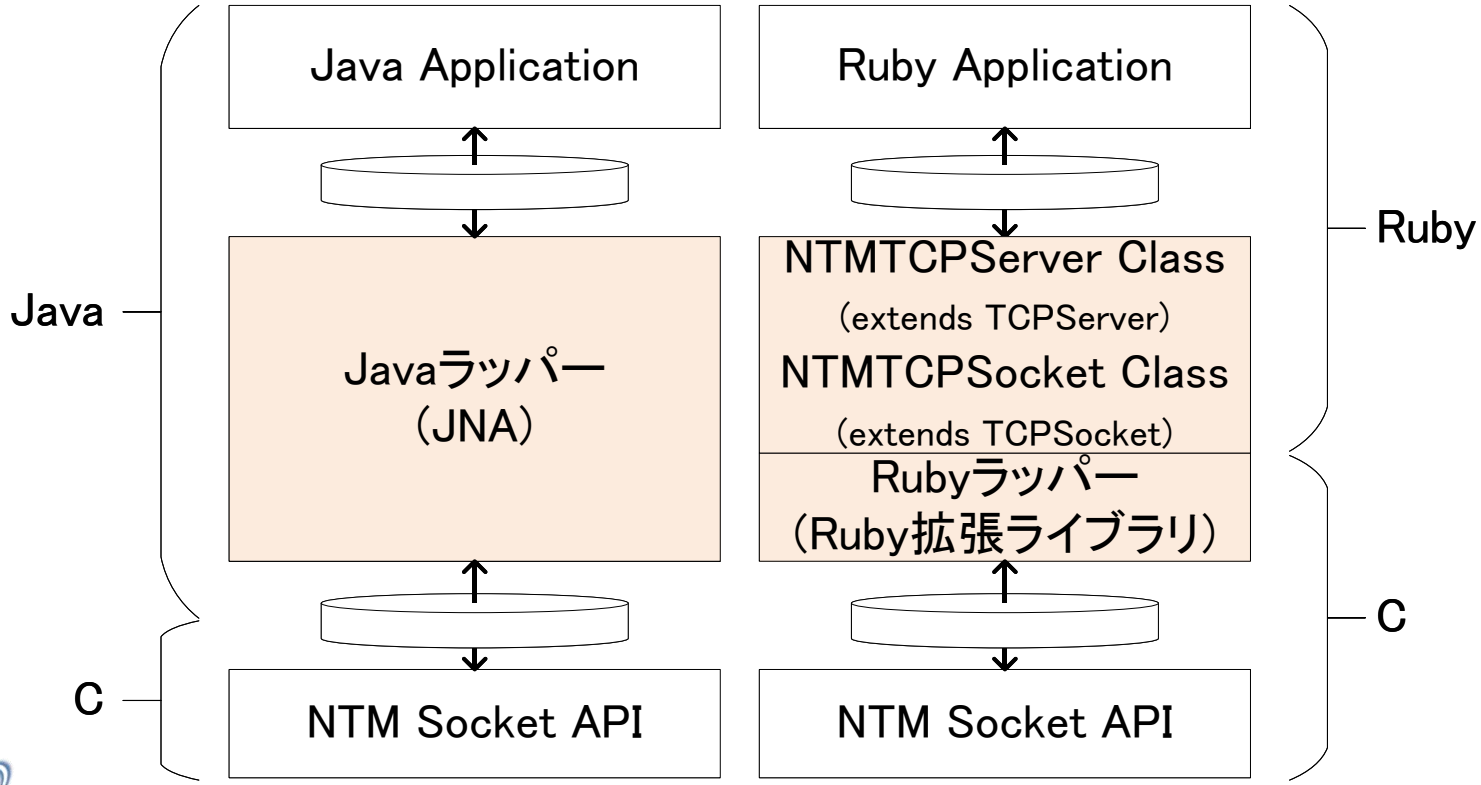
ラッパーの実装

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)

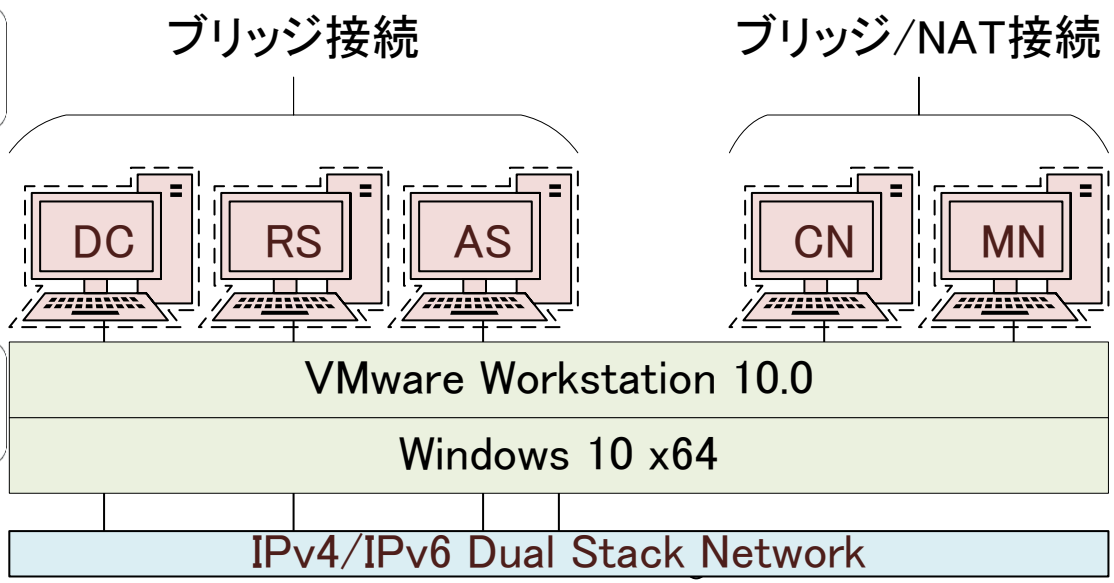
ホストマシン

- 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



MN: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

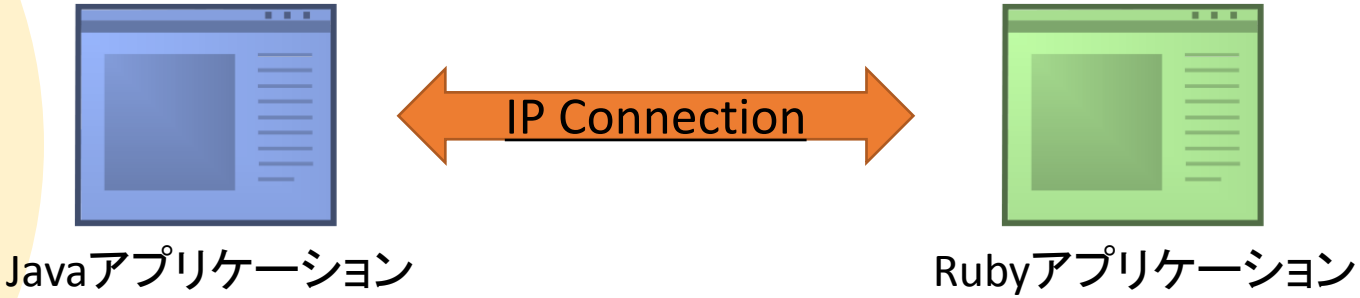
【試験結果】

	MN			
CN		IPv4 Global	IPv4 NAT	IPv6
	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



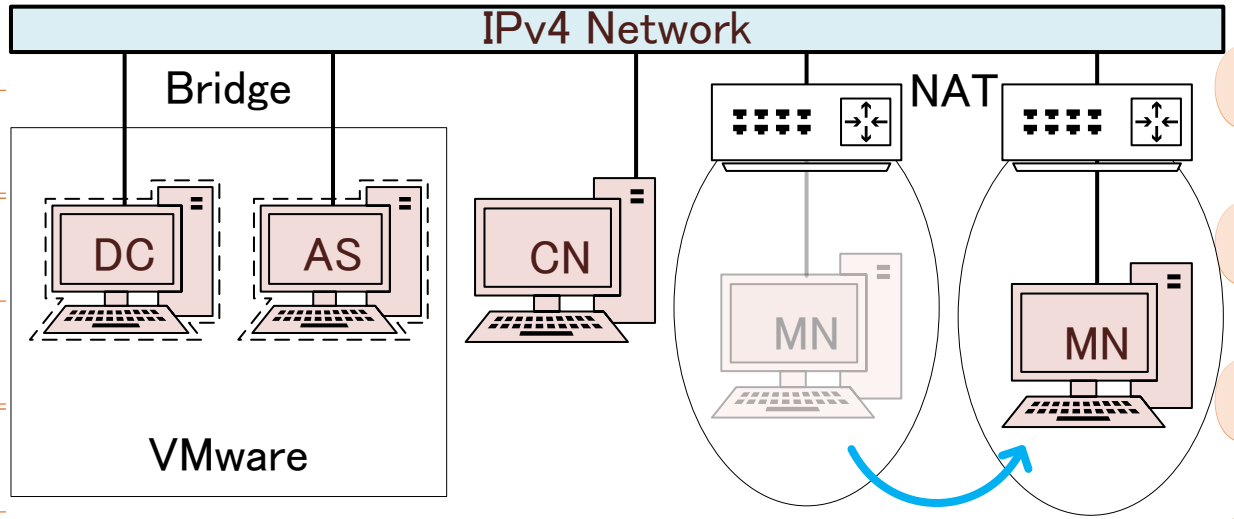
【試験結果】

- Java-Ruby間で仮想IPアドレスによるTCP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

- 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
  - 差し替え後のIPアドレス取得～トンネル構築完了までを測定

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



【試験結果】

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

トンネル再構築に要する時間は僅か

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)

※サービスディーザーサイト

  - frameworkを利用した商用サービスの提供

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
- 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)

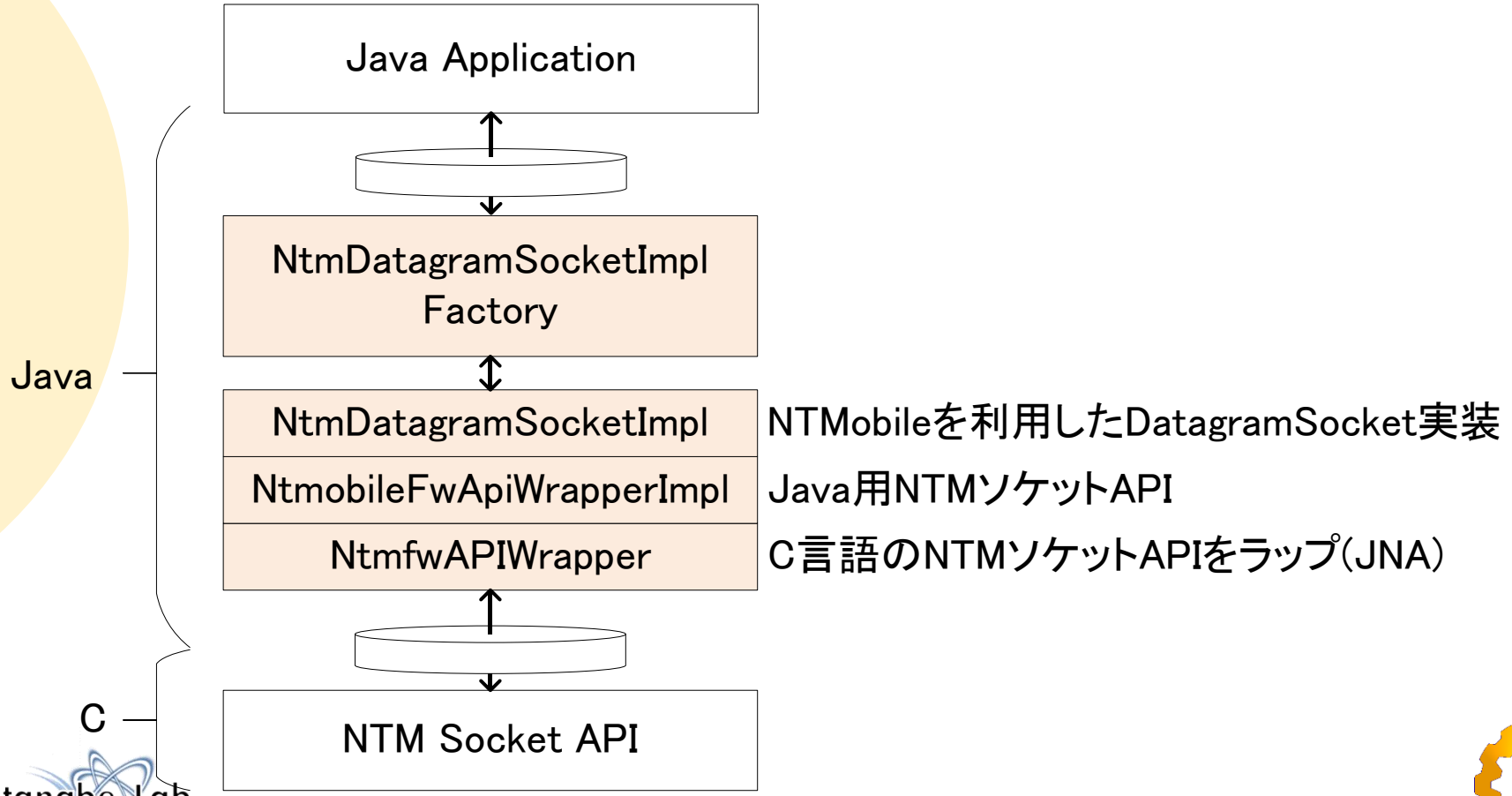
- カーネル実装型
  - カーネル空間に実装
  - 1対1ハッシュテーブル
  - ユーザー空間とNetlinkソケットで連携
    - ユーザー空間にトンネルテーブルとは別にノードテーブルを持つ。
    - トンネルテーブルとノードテーブルで保持する値が異なる。
      - トンネルテーブル：パケットカプセル化に用いるデータ
      - ノードテーブル：制御パケットの送受信に用いるデータ
    - トンネルテーブルとノードテーブルは同期する。
- フレームワーク組込型
  - ユーザー空間に実装
  - 多対1ハッシュテーブル
    - カーネル実装型の全テーブルの要素を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

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。

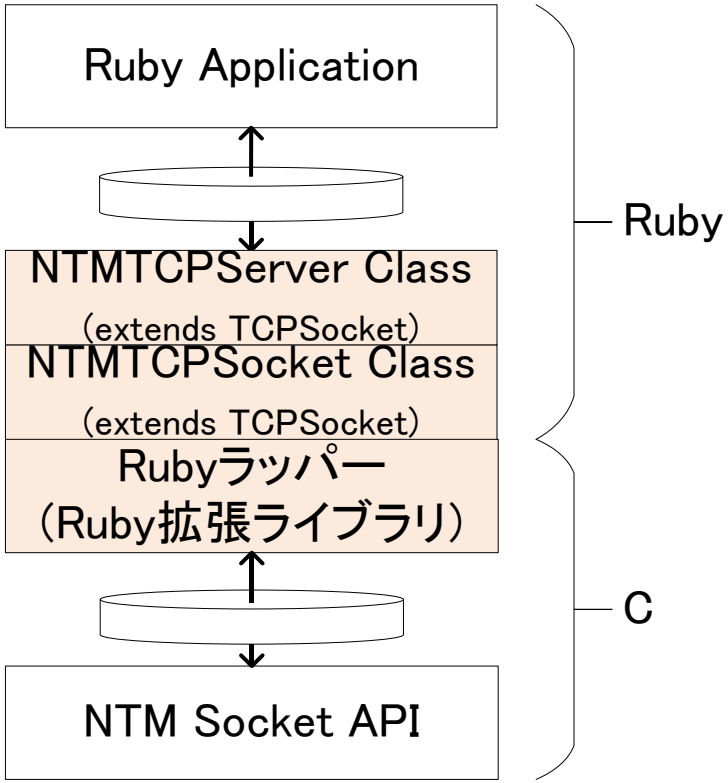
Javaラッパー

- 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)

スループットについて

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