NTMobile用Javaラッパーの提案と実装

130441077 清水 一輝
渡邊研究室

1. はじめに

モバイルネットワークの普及に伴って，ネットワーク環境にかかわらず通信を開始することができる通信接続性とネットワークが切り替わった際にも通信を継続できる移動透過性が求められている．NTMobile（Network Traversal with Mobility）は両者を同時に実現する次世代の技術である [1].

NTMobile は 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側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 言語によって記述されているため，Java で利用するには変換が必要である．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検討した．提案方式では，Java アプリケーションが NTMobile をほとんど意識することなく，Java 標準 API を使用すると自動的に NTMobile フレームワークが呼び出される．

2. NTMobile

NTMobile は，NTMobile 機能を有する NTM 端末と NTM 端末の端末情報管理やトンネルの経路指示を行う DC（Direction Coordinator）によって構成される．

NTM 端末は，起動時に自身の端末情報を DC に対して登録し，DC から仮想 IP アドレスを取得する．アプリケーションは仮想 IP アドレスを利用して通信を確立する．仮想 IP アドレスにより生成されたパケットは全て実 IP アドレスによってカプセル化される．NTM 端末は IPv4 グローバル/IPv4 プライベート/IPv6 アドレスの違いを意識することなく相互に通信ができ，通信中にネットワークを切り替えることが可能である．

NTMobile フレームワークにて行う処理には `ntmfw_ntm_init` と呼ばれる登録処理，`ntmfw_getaddrinfo` と呼ばれる送信/受信端末間のトンネル構築処理，及びパケット送受信時のトンネル通信の 3 つが存在する．

3. NTMobile用Javaラッパー

図 1 に Java ラッパーを用いた NTM 端末のモジュール構成を示す．Java アプリケーションにて NTMobile 通信を行うには，C 言語で記述された NTMobile フレームワークの関数を呼び出す必要がある．C 言語と Java では引数の型の違いがあるため，Java ラッパーを介することにより型の変換処理を行い，NTMobile 通信を可能にする．

Java アプリケーションでは，Java 標準 API を使用して自動的に NTMobile 通信を実現できるように，Java 標準の通信クラスのメソッドをオーバーライドする．これにより Java アプリケーションが Java の通信に関するメソッドを呼び出した際に Java ラッパーによって NTMobile フレームワークの関数を自動的に呼び出すようになる．

図 1 において `init` と `getByName` の 2 つの処理は，NTMobile フレームワーク特有の処理を行うため，Java 標準 API をオーバーライドすることができない．そこでこの 2 つのメソッドは新たにクラスを作成し，そこに定義する．`init` は `ntmfw_ntm_init` を呼び出し，`getByName` は `ntmfw_getaddrinfo` を呼び出す．

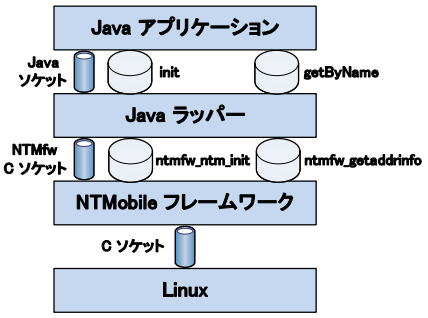


図 1: Java ラッパーを用いた NTM 端末のモジュール構成

4. 評価

Java ラッパーを実装し，NTMobile 通信による，動作検証と処理時間の測定を行った．

送信/受信端末に対して Java アプリケーション，Java ラッパー，NTMobile フレームワークを実装し，Linux 上で動作検証を行った．両端末間で，UDP 通信を行った際に要する時間の測定を 5 回行った．これらを平均した結果を表 1 に示す．

表 1: 送信/受信時の処理時間の平均

測定箇所	送信時 [ms]	受信時 [ms]
Java ラッパー	0.13	0.16
NTMobile fw	2.91	2.37
Linux	0.07	0.01
合計	3.11	2.54

表 1 の Linux は NTMobile 通信を利用しなかった場合の送受信に要する時間である．NTMobile 通信を行うと送信時では約 3.1 ミリ秒，受信時では約 2.5 ミリ秒かかり，多くの時間が NTMobile フレームワークであることが分かった．NTMobile フレームワークに多くの時間を要する理由は，メッセージの暗号化/復号処理に要する時間が含まれるためと考えられる．Java ラッパーでの処理に要した時間は送信時で約 0.13 ミリ秒，受信時で約 0.16 ミリ秒であった．

5. まとめ

本稿では，NTMobile フレームワーク用の Java ラッパーを実現する方式を提案した．UDP 通信部分の実装を行い，Java 標準 API を使用して NTMobile 通信が可能なことを確認した．

参考文献

[1] 上醉尾一真, 他: 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).

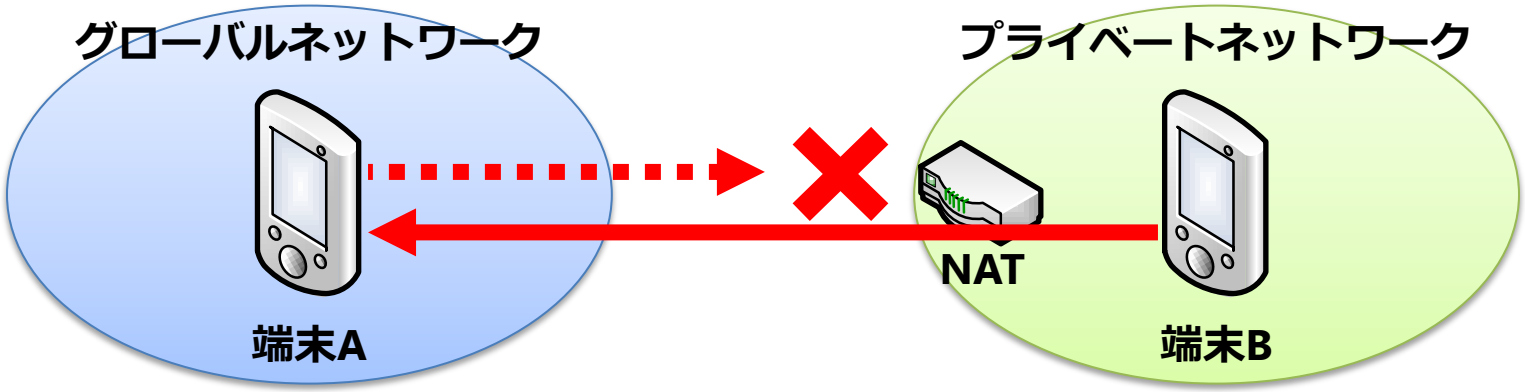
Watanabe Lab.

NTMobile用 Javaラッパーの提案と実装

130441077
渡邊研究室 清水 一輝

研究背景

- インターネット通信の需要増加
  - スマートフォンなどの移動通信端末の普及
- 通信接続性の課題
  - IPv4アドレスの枯渇に伴い，NATによるプライベートネットワークを構築することが一般的
  - NATの外側にあるネットワークから，NATの内側にあるネットワークにアクセスできない（NAT越え問題）



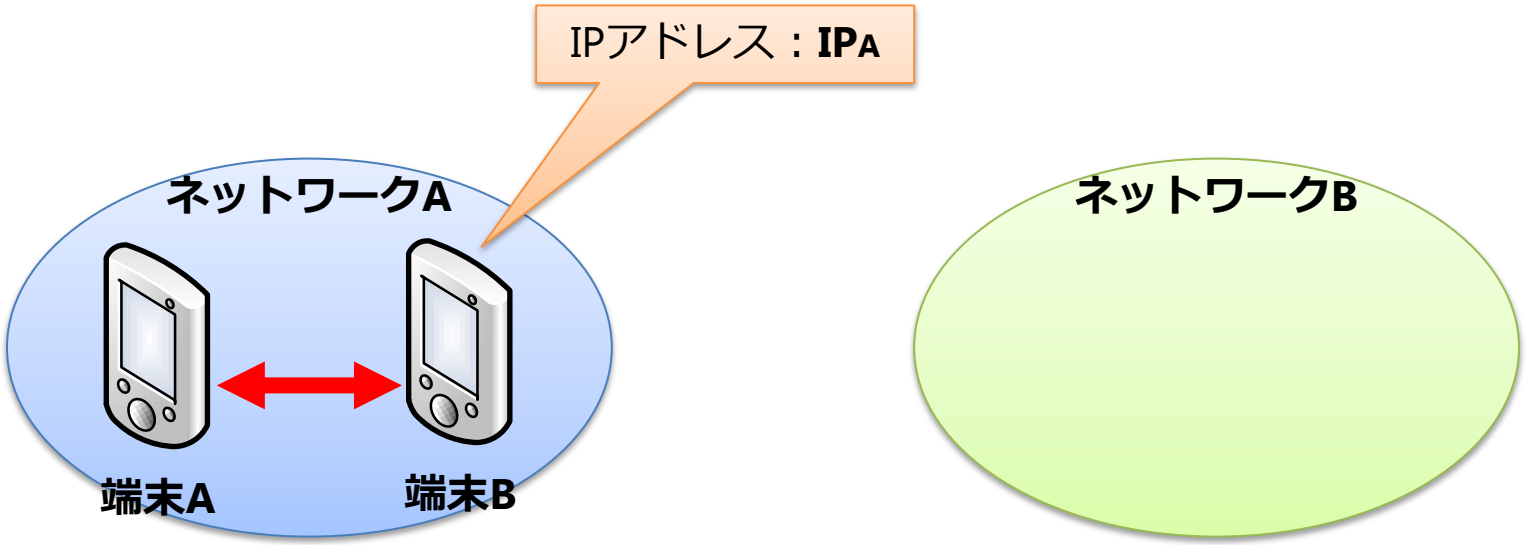
端末間で自由に通信できない

NAT：Network Address Translation

研究背景

移動透過性の課題

- 現在のネットワークでは，IPアドレスを通信識別子としている
- 端末移動時などにネットワークが切り替わると，
  端末のIPアドレスが変化し，通信を継続できない

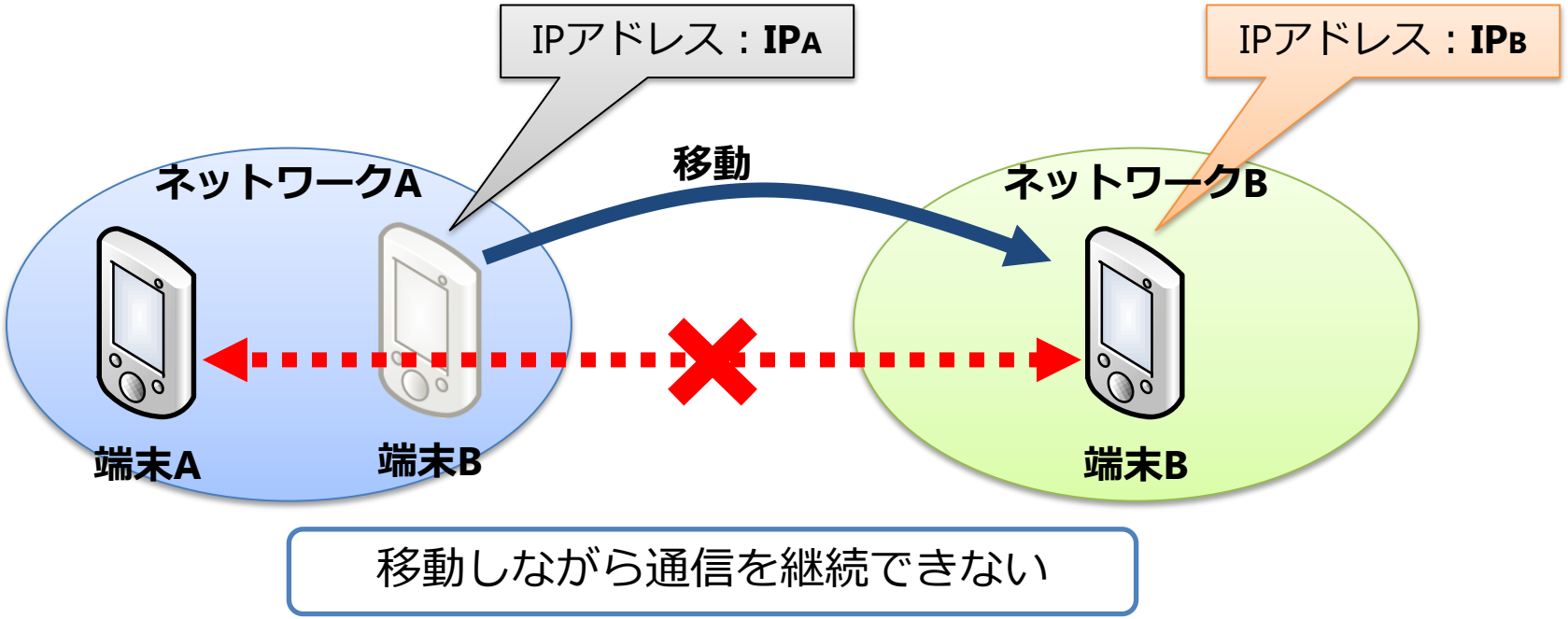


移動しながら通信を継続できない

研究背景

移動透過性の課題

- 現在のネットワークでは，IPアドレスを通信識別子としている
- 端末移動時などにネットワークが切り替わると，
  端末のIPアドレスが変化し，通信を継続できない



Javaの有用性

- 幅広い分野で使用されている
  - 業務システム
  - Webサービス
  - Androidアプリケーション
- 人気のあるプログラミング言語

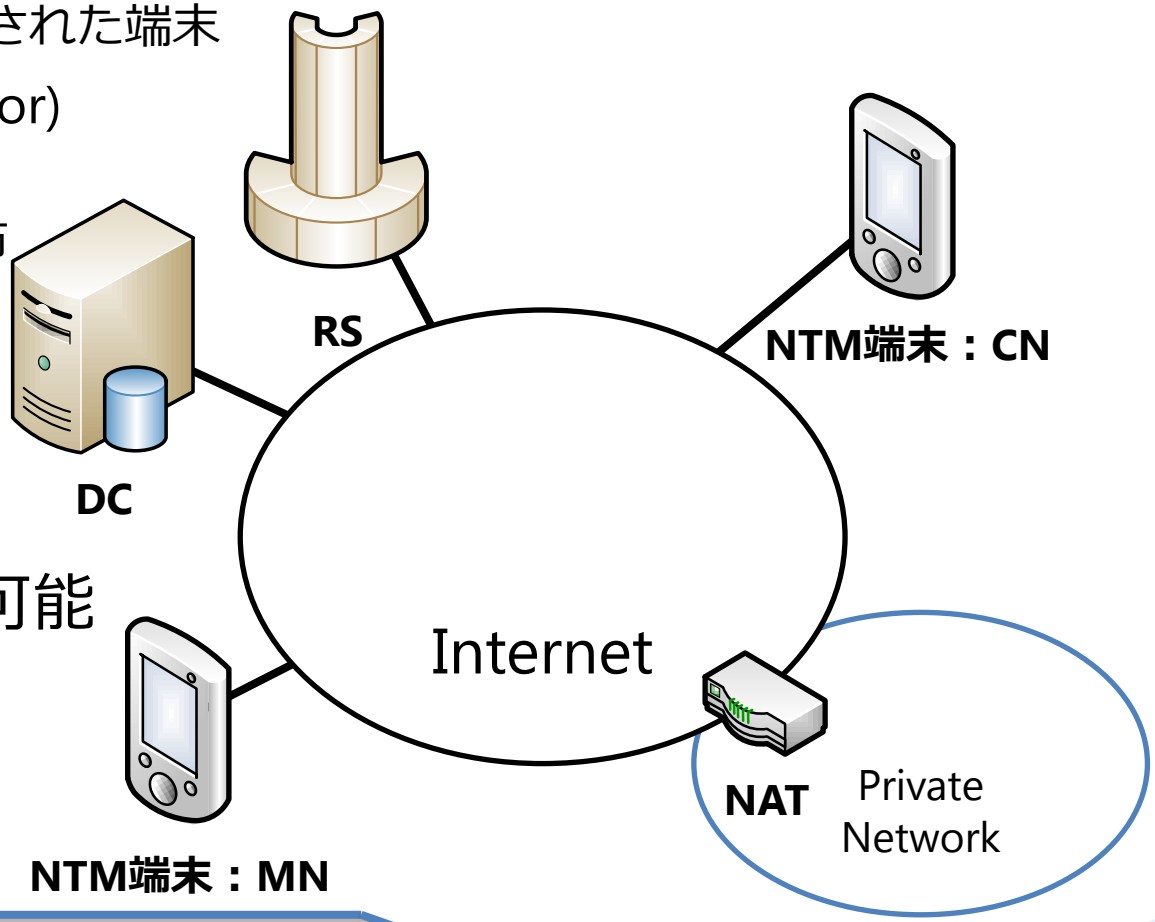
	IEEE	TIOBE	Devpost	GitHub	RedMonk
1位	**Java**	**Java**	HTML/CSS	JavaScript	JavaScript
2位	C	C	JavaScript	**Java**	**Java**
3位	C++	C++	Python	Ruby	PHP
4位	Python	C#	**Java**	PHP	Python
5位	C#	Python	C/C++	Python	C#

各調査機関のプログラミング言語ランキング．Code部．
https://blog.codecamp.jp/programming_language_2015（参照 2016/12/03）

NTMobile (Network Traversal with Mobility)

Watanabe Lab.

- 通信接続性と移動透過性を**同時**に実現する技術
  - NTM端末(NTMobile Node)
    - NTMobile機能が実装された端末
  - DC(Direction Coordinator)
    - 通信経路の指示
    - **仮想IPアドレス**の配布
  - RS(Relay Server)
    - 直接通信不可の際，通信を中継



- DC，RSは複数台設置可能

仮想IPアドレス

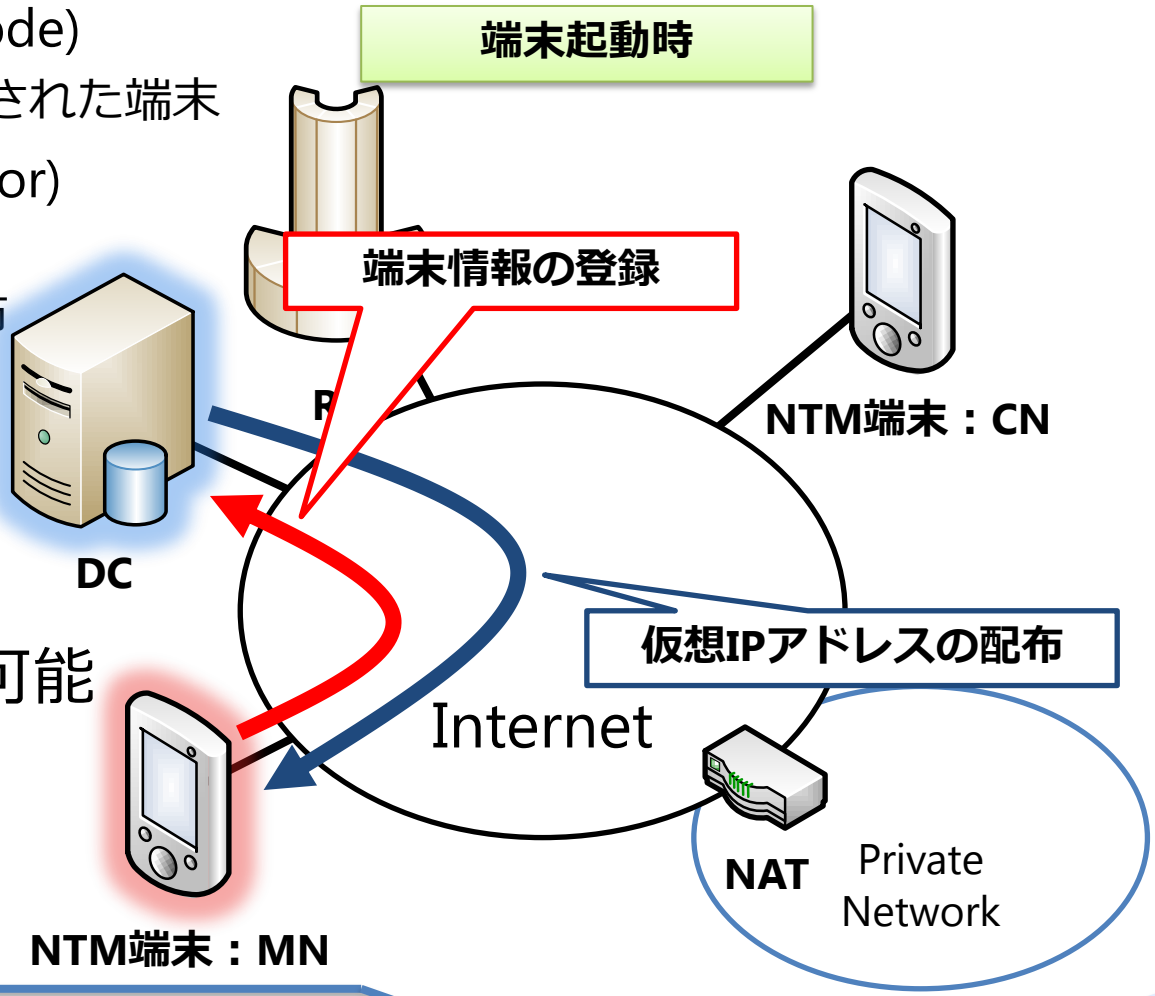
・端末の位置に依存しない
・実IPアドレスの変化を隠蔽

MN：Mobile Node　CN：Correspondent Node

NTMobile (Network Traversal with Mobility)

Watanabe Lab.

- 通信接続性と移動透過性を**同時**に実現する技術
  - NTM端末(NTMobile Node)
    - NTMobile機能が実装された端末
  - DC(Direction Coordinator)
    - 通信経路の指示
    - **仮想IPアドレス**の配布
  - RS(Relay Server)
    - 直接通信不可の際，通信を中継



- DC，RSは複数台設置可能

仮想IPアドレス

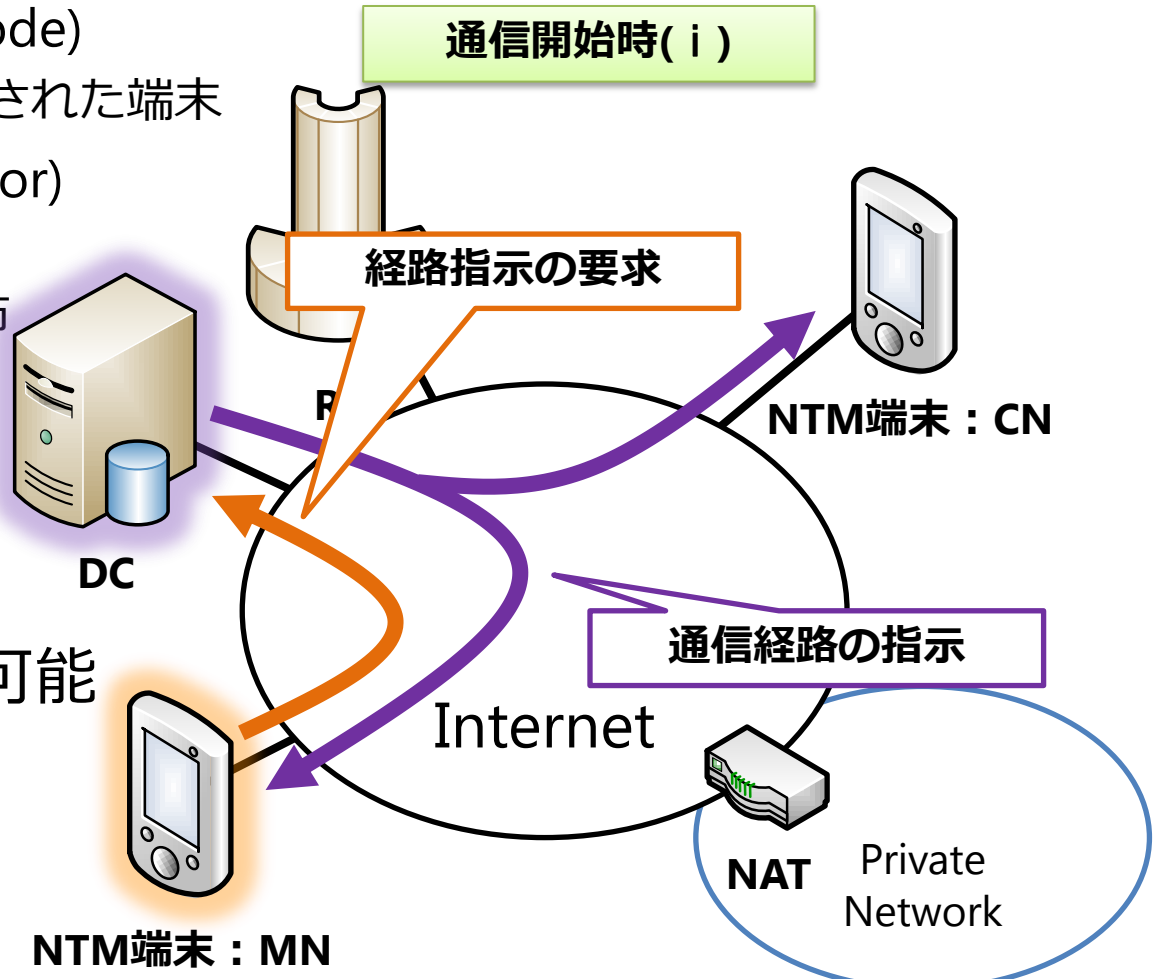
・端末の位置に依存しない
・実IPアドレスの変化を隠蔽

MN：Mobile Node　CN：Correspondent Node

NTMobile (Network Traversal with Mobility)

Watanabe Lab.

- 通信接続性と移動透過性を**同時**に実現する技術
  - NTM端末(NTMobile Node)
    - NTMobile機能が実装された端末
  - DC(Direction Coordinator)
    - 通信経路の指示
    - **仮想IPアドレス**の配布
  - RS(Relay Server)
    - 直接通信不可の際，通信を中継



- DC，RSは複数台設置可能

仮想IPアドレス

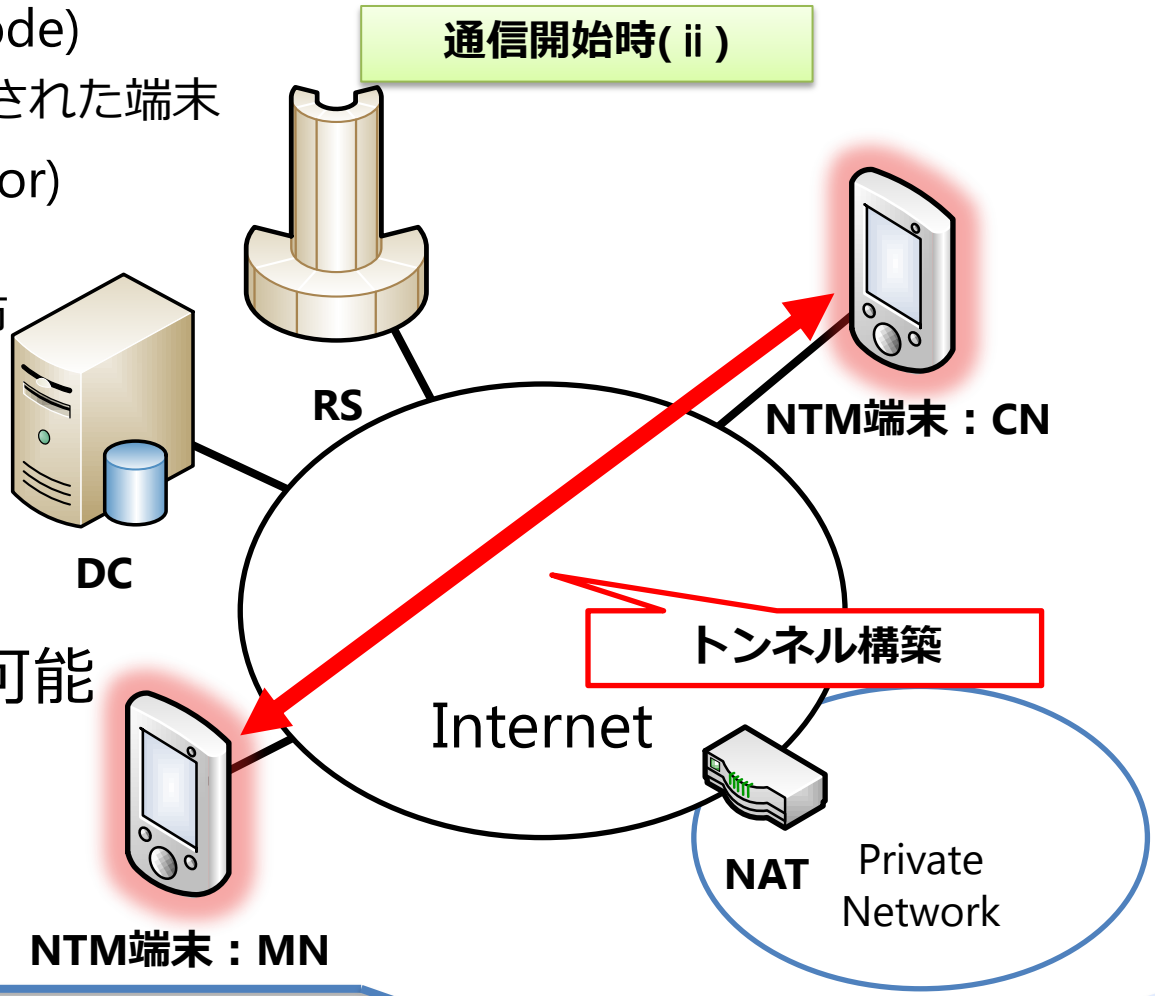
・端末の位置に依存しない
・実IPアドレスの変化を隠蔽

MN：Mobile Node　CN：Correspondent Node

NTMobile (Network Traversal with Mobility)

Watanabe Lab.

- 通信接続性と移動透過性を**同時**に実現する技術
  - NTM端末(NTMobile Node)
    - NTMobile機能が実装された端末
  - DC(Direction Coordinator)
    - 通信経路の指示
    - **仮想IPアドレス**の配布
  - RS(Relay Server)
    - 直接通信不可の際，通信を中継



- DC，RSは複数台設置可能

仮想IPアドレス

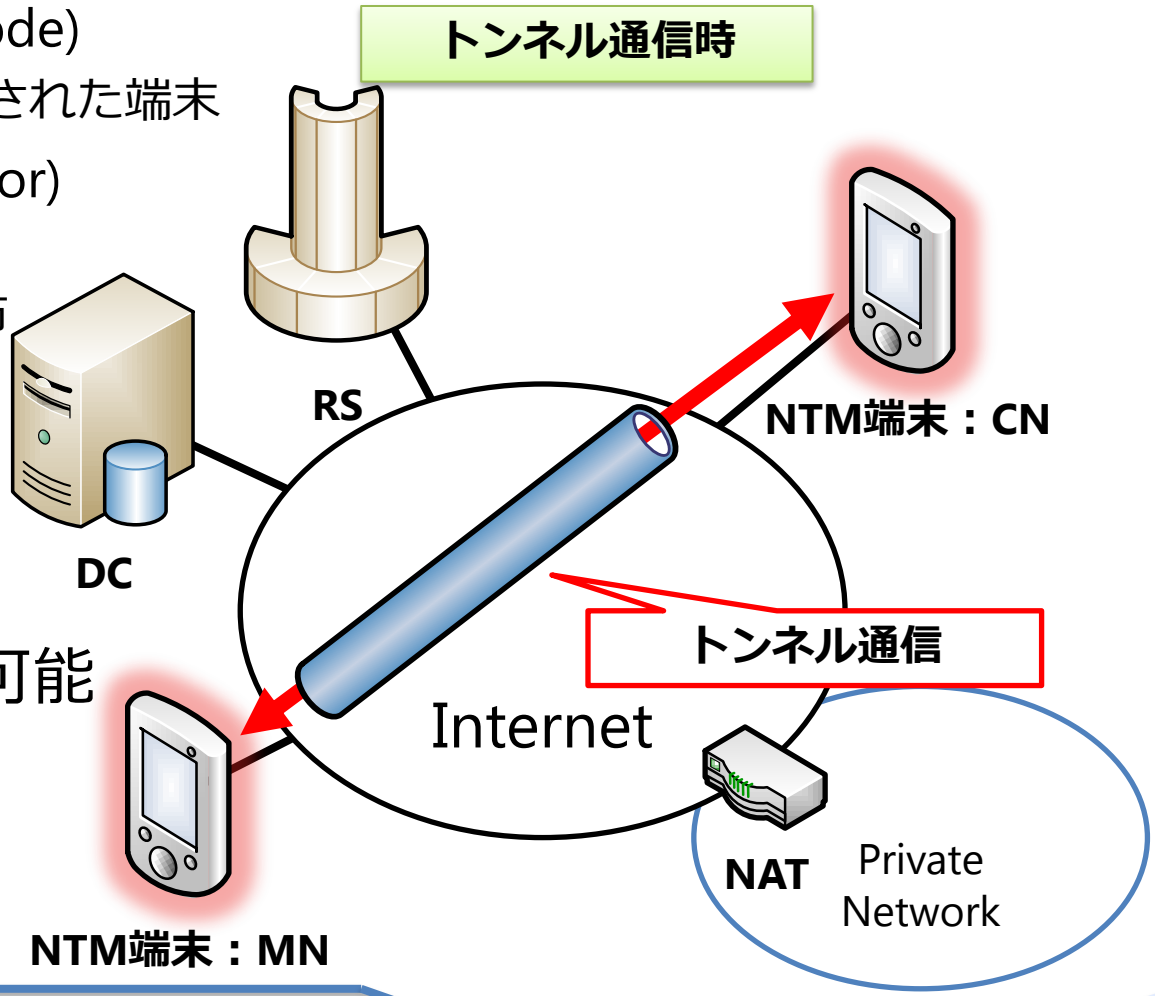
・端末の位置に依存しない
・実IPアドレスの変化を隠蔽

MN：Mobile Node　CN：Correspondent Node

NTMobile (Network Traversal with Mobility)

Watanabe Lab.

- 通信接続性と移動透過性を**同時**に実現する技術
  - NTM端末(NTMobile Node)
    - NTMobile機能が実装された端末
  - DC(Direction Coordinator)
    - 通信経路の指示
    - **仮想IPアドレス**の配布
  - RS(Relay Server)
    - 直接通信不可の際，通信を中継



- DC，RSは複数台設置可能

仮想IPアドレス

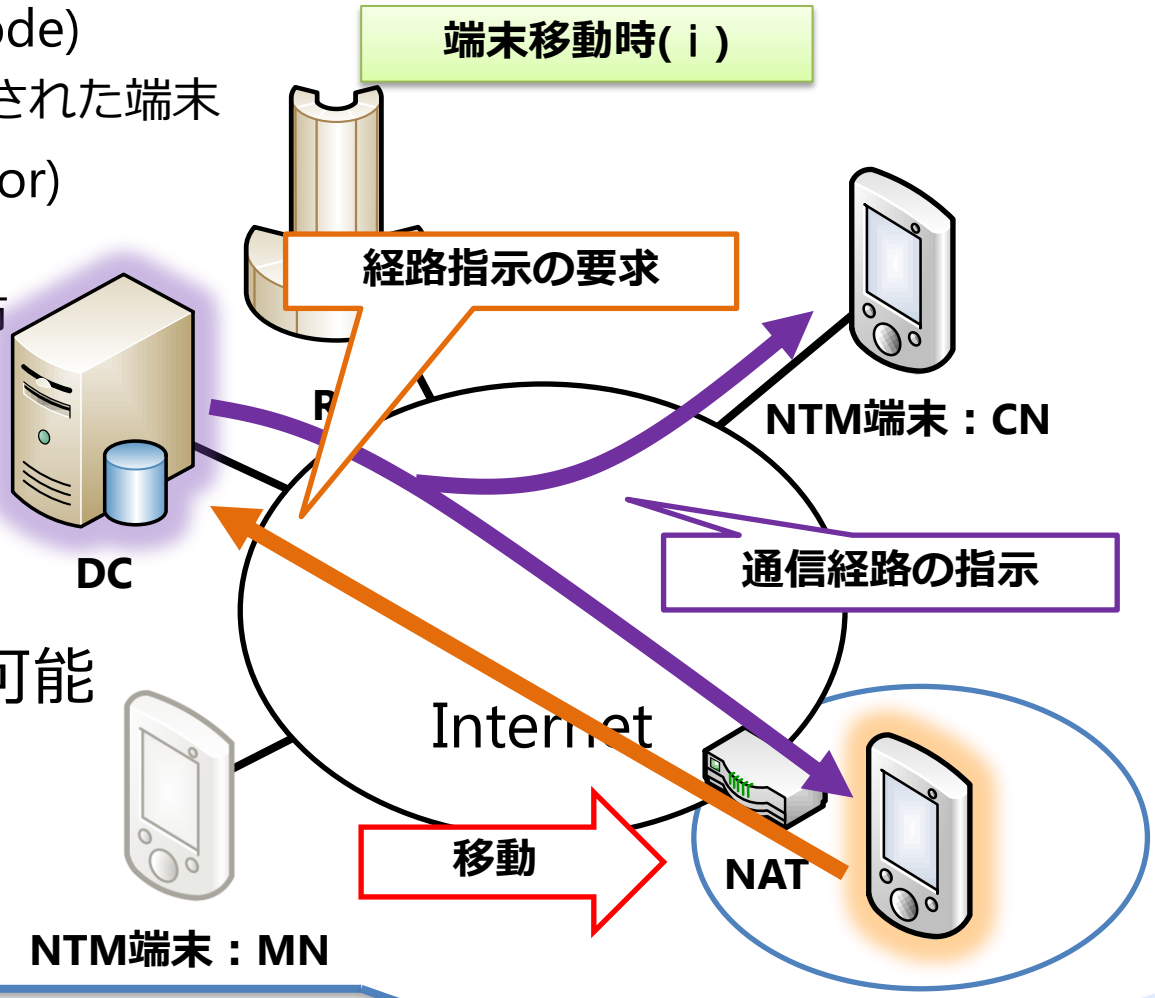
・端末の位置に依存しない
・実IPアドレスの変化を隠蔽

MN：Mobile Node　CN：Correspondent Node

NTMobile (Network Traversal with Mobility)

Watanabe Lab.

- 通信接続性と移動透過性を**同時**に実現する技術
  - NTM端末(NTMobile Node)
    - NTMobile機能が実装された端末
  - DC(Direction Coordinator)
    - 通信経路の指示
    - **仮想IPアドレス**の配布
  - RS(Relay Server)
    - 直接通信不可の際，通信を中継



- DC，RSは複数台設置可能

仮想IPアドレス

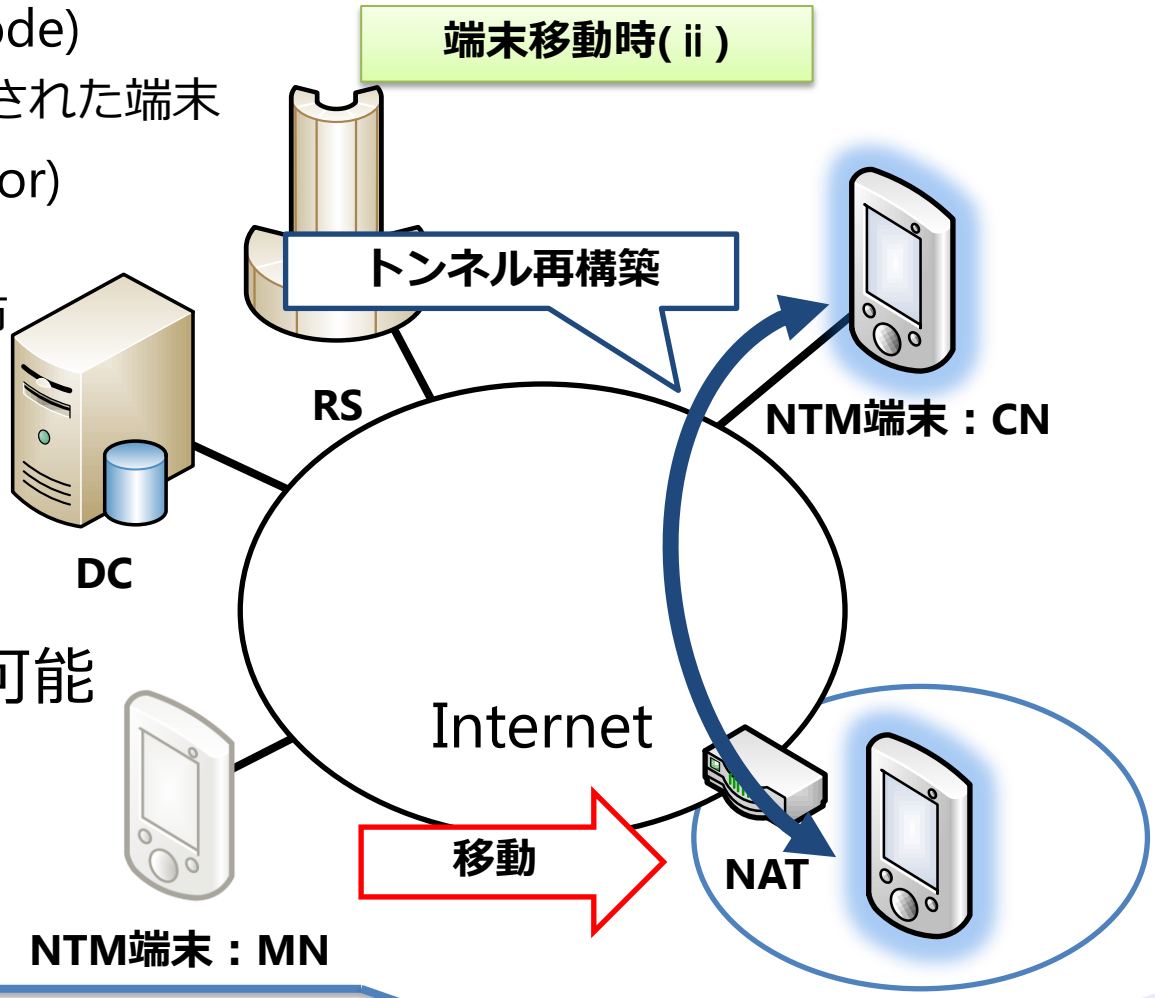
・端末の位置に依存しない
・実IPアドレスの変化を隠蔽

MN：Mobile Node　CN：Correspondent Node

NTMobile (Network Traversal with Mobility)

Watanabe Lab.

- 通信接続性と移動透過性を**同時**に実現する技術
  - NTM端末(NTMobile Node)
    - NTMobile機能が実装された端末
  - DC(Direction Coordinator)
    - 通信経路の指示
    - **仮想IPアドレス**の配布
  - RS(Relay Server)
    - 直接通信不可の際，通信を中継



- DC，RSは複数台設置可能

仮想IPアドレス

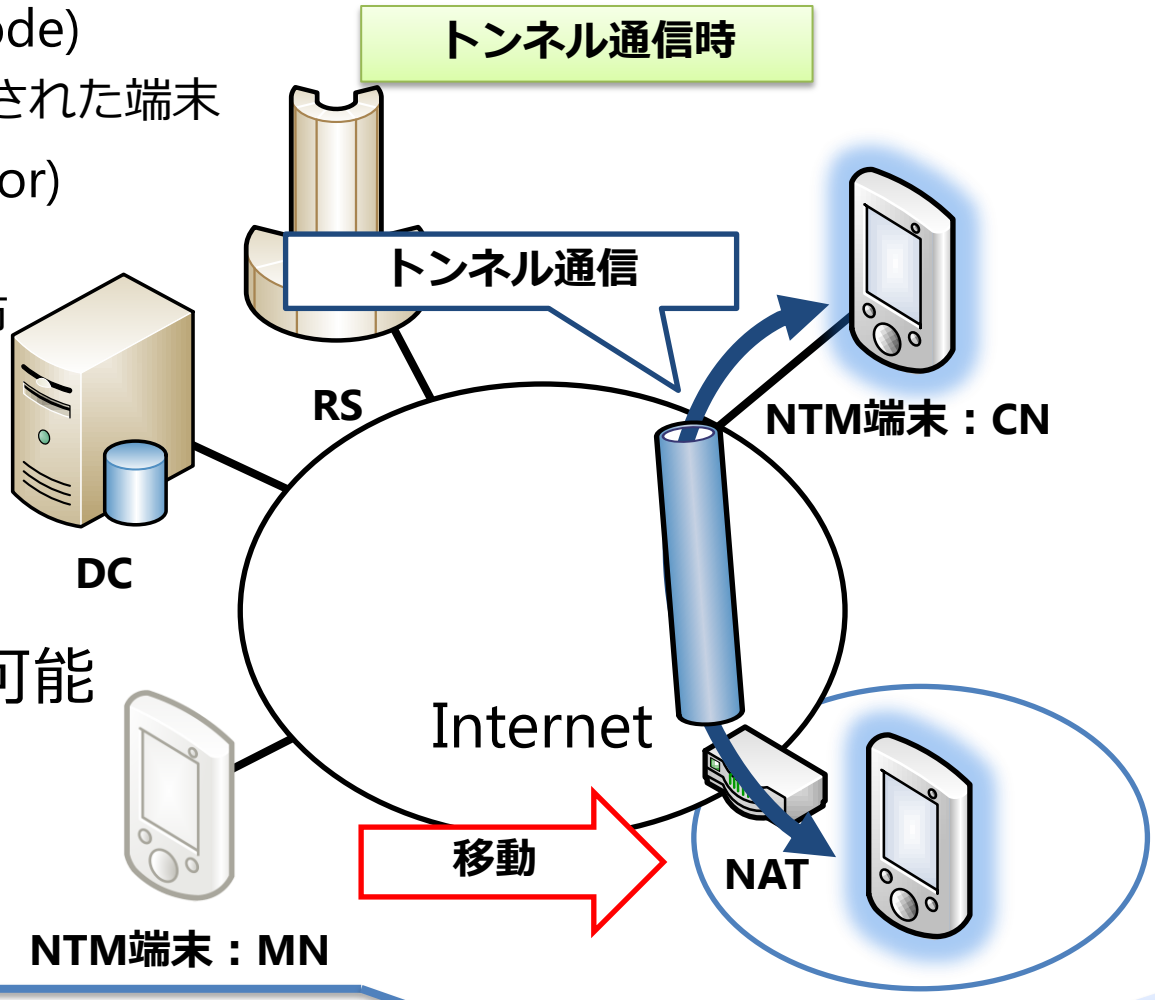
・端末の位置に依存しない
・実IPアドレスの変化を隠蔽

MN：Mobile Node　CN：Correspondent Node

NTMobile (Network Traversal with Mobility)

Watanabe Lab.

- 通信接続性と移動透過性を**同時**に実現する技術
  - NTM端末(NTMobile Node)
    - NTMobile機能が実装された端末
  - DC(Direction Coordinator)
    - 通信経路の指示
    - **仮想IPアドレス**の配布
  - RS(Relay Server)
    - 直接通信不可の際，通信を中継



- DC，RSは複数台設置可能

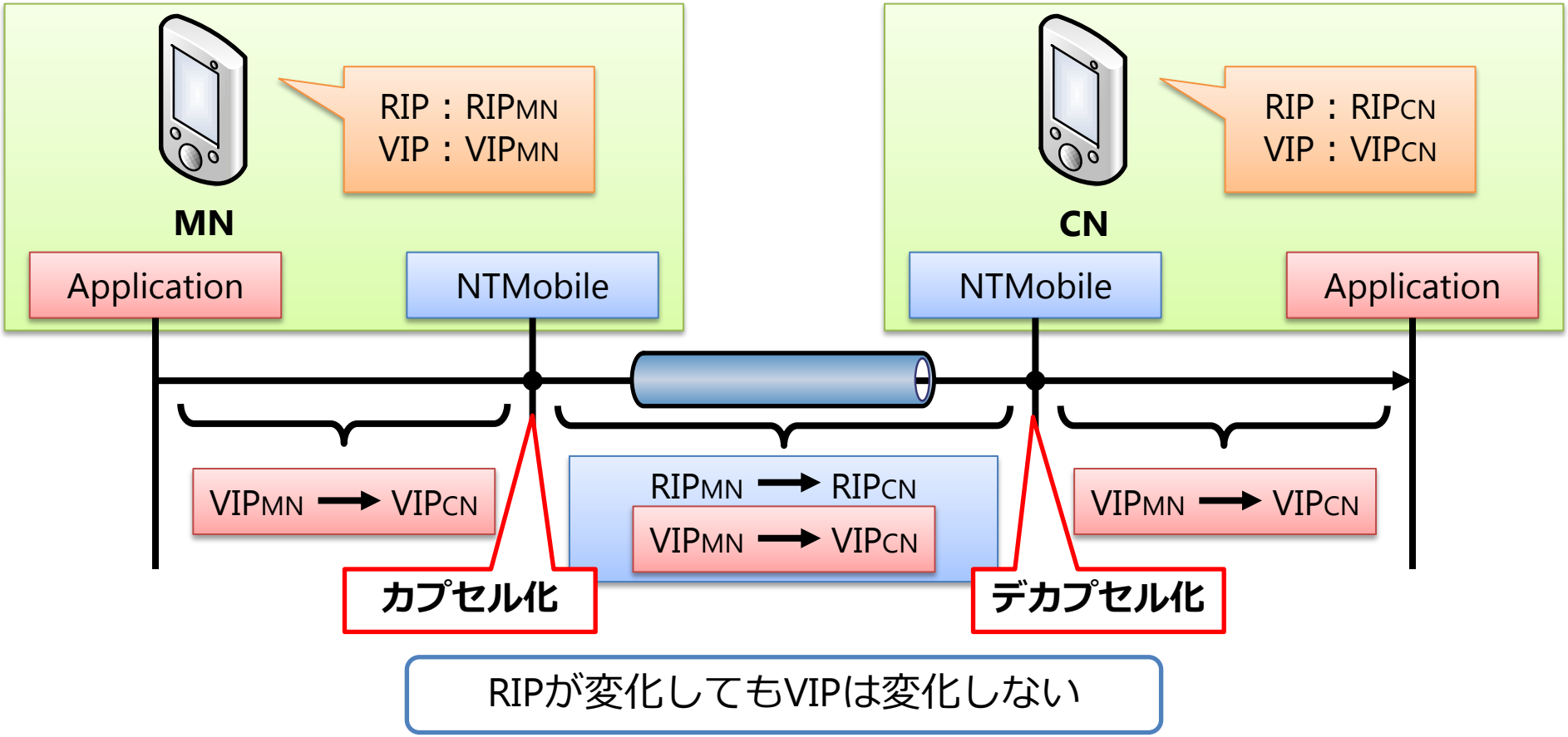
仮想IPアドレス

・端末の位置に依存しない
・実IPアドレスの変化を隠蔽

MN：Mobile Node　CN：Correspondent Node

NTMobileの通信

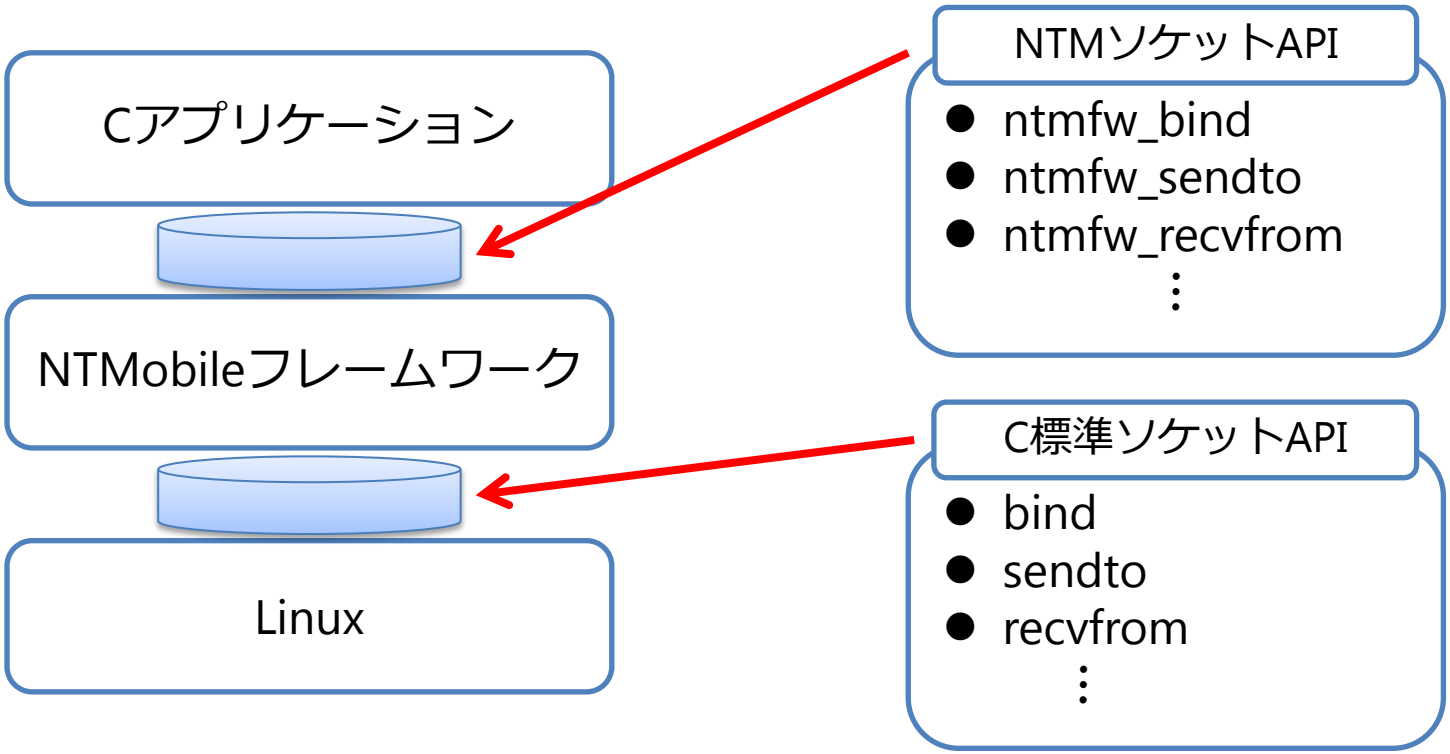
- パケットを実IPアドレスでカプセル化した通信



RIP：実IPアドレス　VIP：仮想IPアドレス

NTMobileフレームワーク

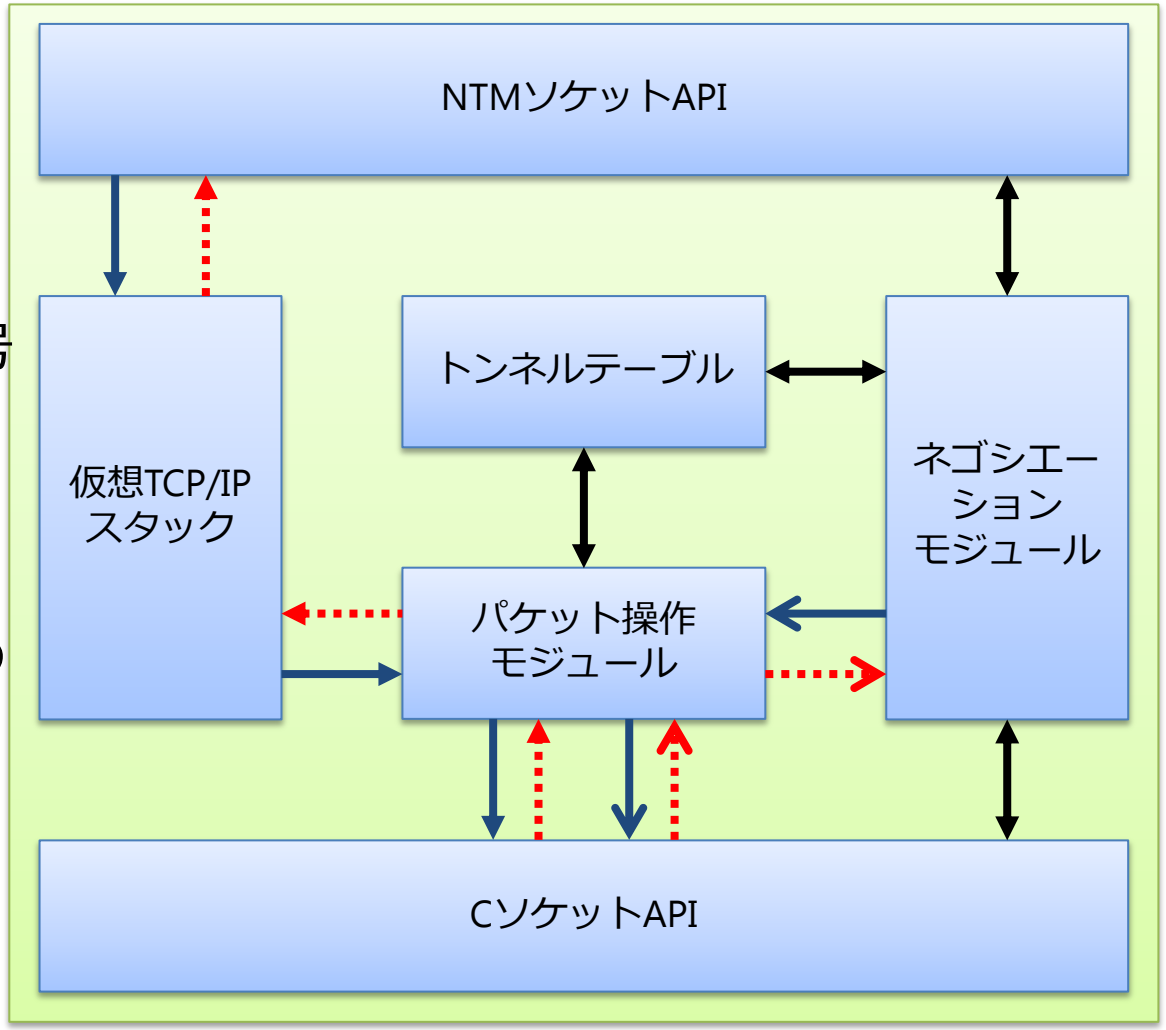
- NTMobile機能をユーザ空間にて実現する実装方式
- アプリケーションはC言語の標準ソケットAPIに代わり，NTMソケットAPIを使用する



API：Application Programming Interface

NTMobileフレームワークの構成

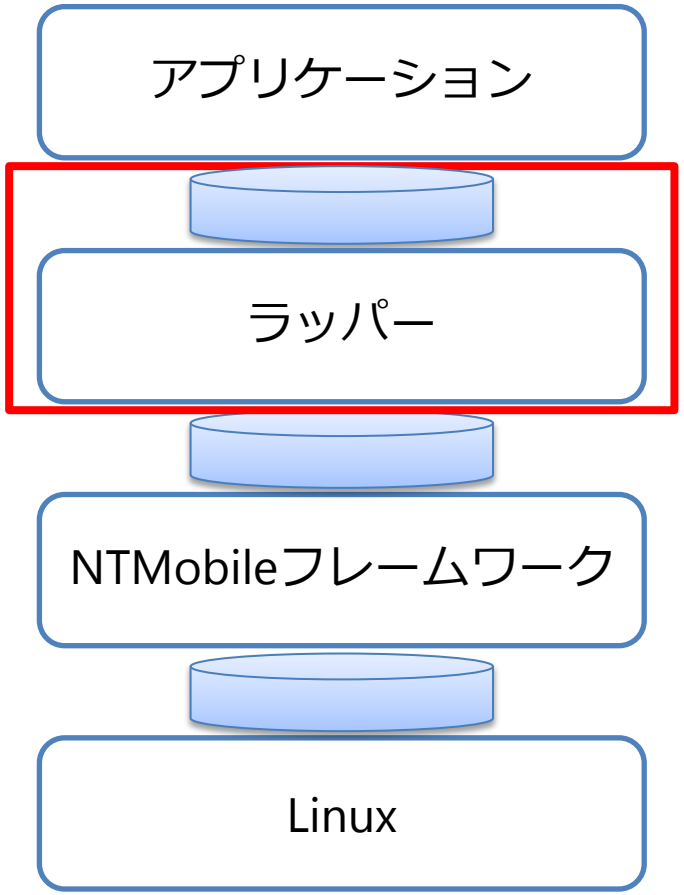
- 仮想TCP/IPスタック
  - 仮想IPアドレスの提供
- パケット操作モジュール
  - パケットの暗号化/復号
  - 改ざん検知のためのMAC付与/検証
- トンネルテーブル
  - 実/仮想IPアドレス等の関係を所持
- ネゴシエーションモジュール
  - 通信経路のやり取り



MAC：Message Authentication Code

NTMobileフレームワーク

- フレームワークはC言語によって実装
  - フレームワークを利用できるアプリケーションはC言語に限定
  - 実用的な利用のためには他のプログラミング言語からフレームワークを利用可能にするための**ラッパー**が必要

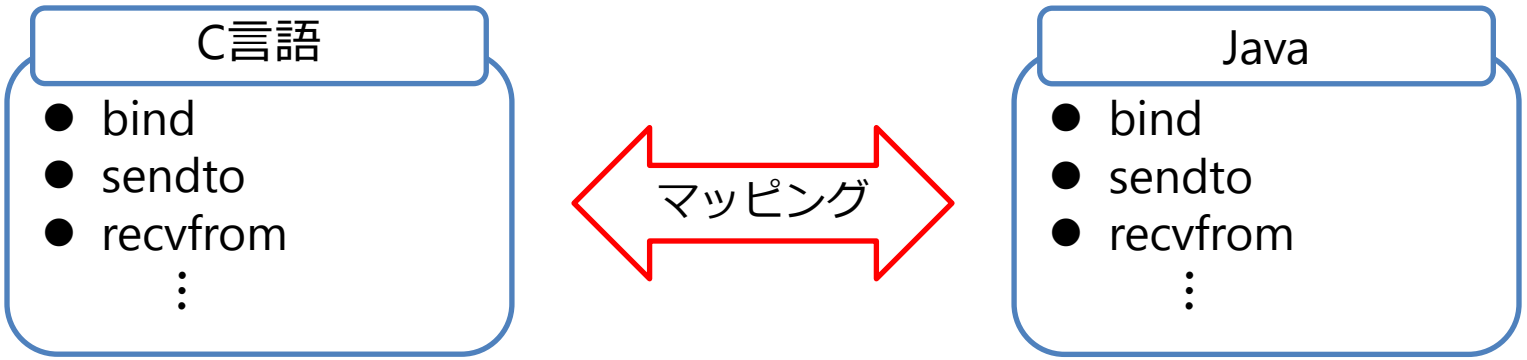


- **ラッパー**とは
  - 他のプログラミング言語にて実装された機能などを利用できるようにするもの

NTMobile用Javaラッパー

- フレームワークのライブラリへJNAを利用しアクセス

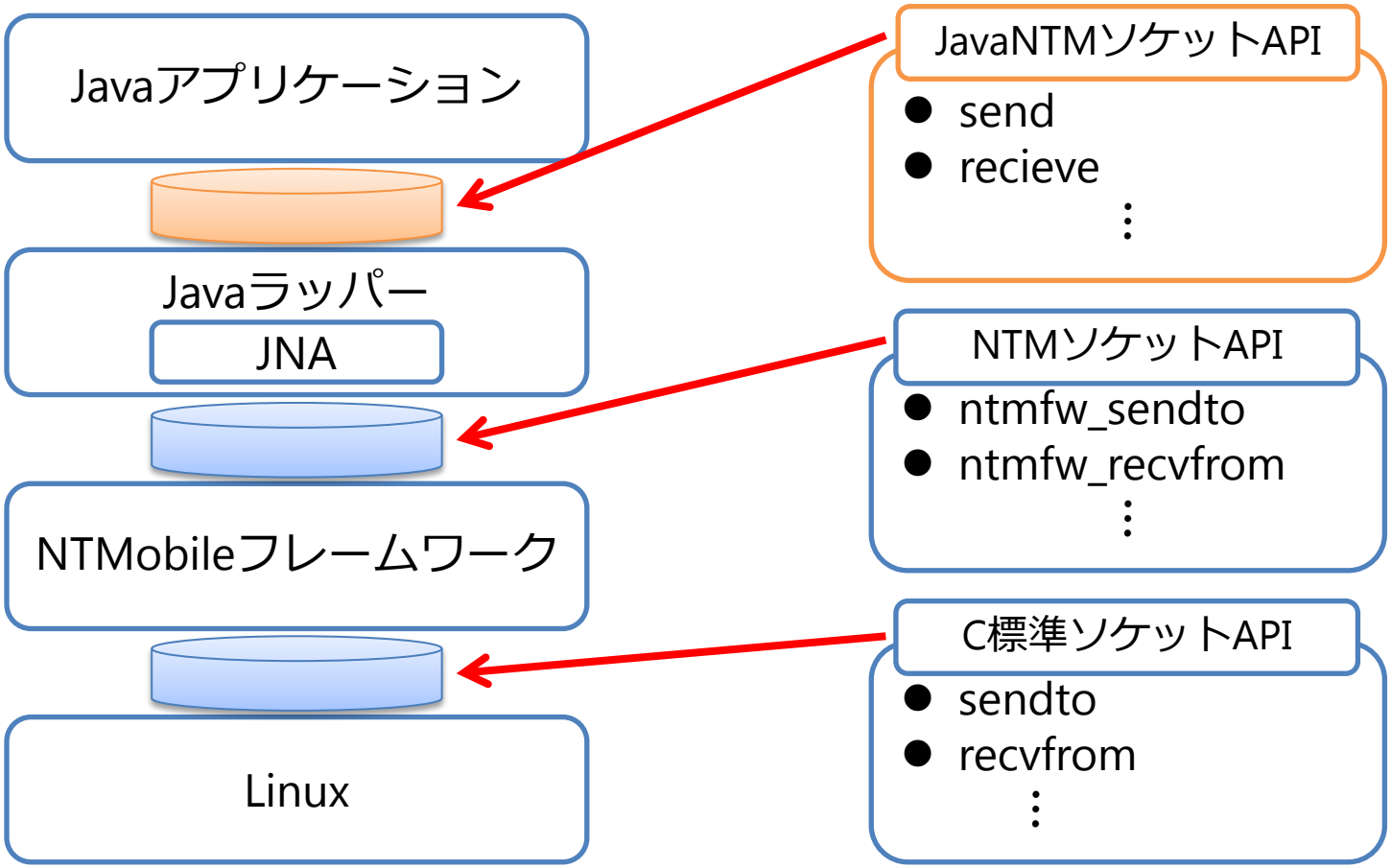
- JNA(Java Native Access)
  - C言語のライブラリにアクセスする方法を提供
  - C言語のソースコードに手を加える必要なし



同じ名前で宣言することで使用可能

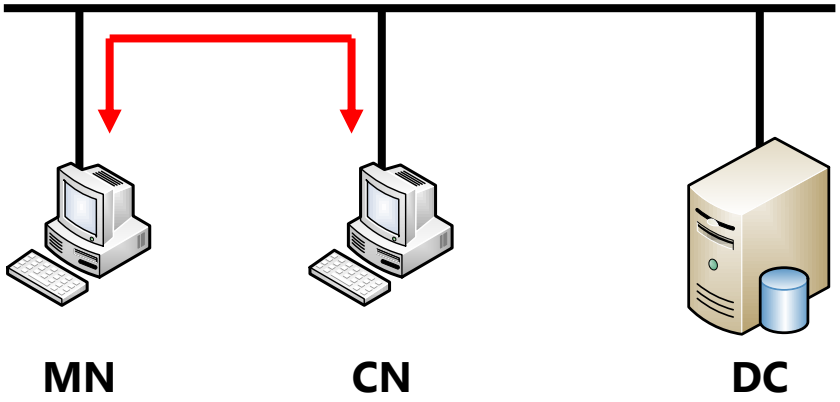
NTMobile用Javaラッパー

- JavaアプリケーションはJavaの標準ソケットAPIに代わり，JavaNTMソケットAPIを使用する



動作検証

- 装置の仕様
  - 全ての装置を1台のホストマシン上に仮想マシンで構築

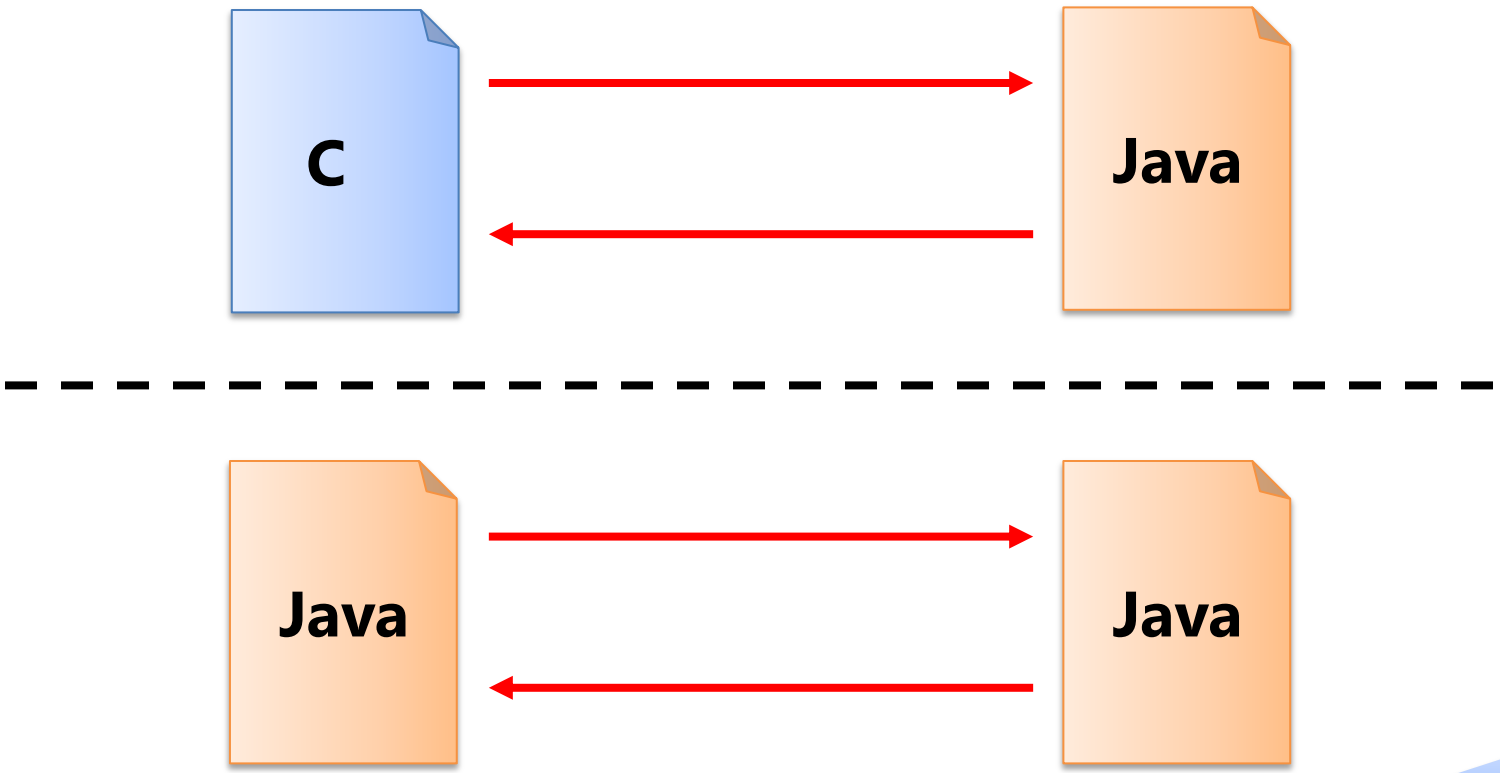


ホストマシン	
OS	Windows 10 64bit
CPU	Intel Core i7-4770 3.40GHz
Memory	8.00GB

仮想マシン	MN，CN	DC
OS	Ubuntu 14.04 32bit	Ubuntu 12.04 32bit
Kernel Version	3.13.0-24-generic	3.2.0-101-generic-pae
CPU割り当て	各1Core	1Core
Memory割り当て	各2.00GB	1.00GB

動作検証

- NTMソケットAPIを使用し，UDPによる任意のメッセージを送受信するアプリケーションをC言語とJavaで作成
- C言語とJava，Java同士の2通りで動作を検証
  - 2通りでUDP送受信の成功を確認

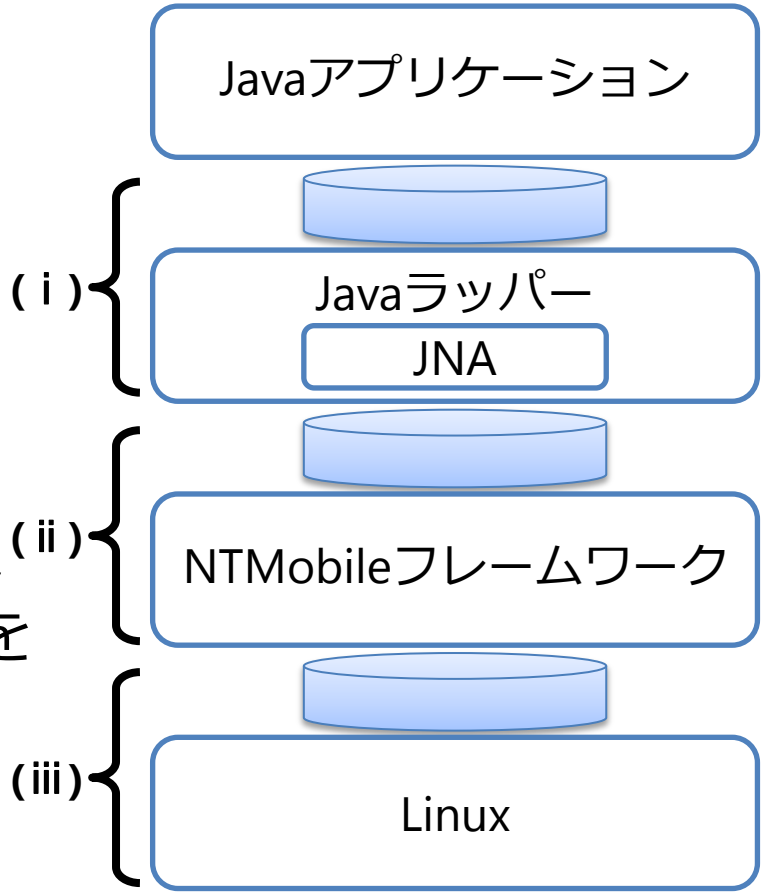


性能評価

- UDP通信時の処理時間を計測
  - 5回の平均を計算

計測箇所	送信時[ms]	受信時[ms]
(ⅰ) Javaラッパー	**0.13**	**0.16**
(ⅱ) NTMobile	2.91	2.37
(ⅲ) Linux	0.07	0.01
合計	3.11	2.54

- NTMobileではパケット暗号化/復号，改ざん検知のためにMAC付与/検証を行うため時間を要する



MAC：Message Authentication Code

まとめ

- NTMobile用Javaラッパー
  - NTMobileフレームワークをJavaにて使用可能にする
- 実装と評価
  - 仮想環境にて正常に動作することを確認
    - JavaでNTMobileを使用可能
  - C言語とJavaでの異なるプログラミング言語による通信も確認
- 今後の予定
  - 既存アプリケーションへの実装