Internet Week 2004 **チュートリアル** T28

NTT Information Sharing Platform Laboratories

IPv6 Advanced Features

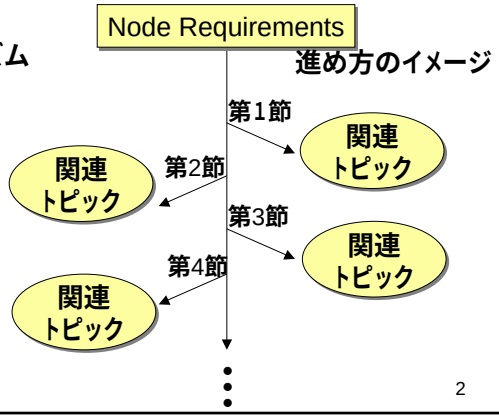
NTT**情報流通プラットフォーム研究所**
加藤淳也 <kato@syce.net>
2004/12/03

1

チュートリアルのアウトライン

NTT Information Sharing Platform Laboratories

- IPv6 quick review
- IPv6 Node Requirements **概観**
  - IPv6**ノードが持つべき機能について網羅したドキュメント**
  - **入門書ではあまり取り上げられていない機能も注目**
    - **アドレス自動設定**
    - **アドレス選択アルゴリズム**
    - DNS
    - MIB
    - MobileIPv6 / NEMO
    - IP security（IPsec）
- **その他のトピック**
  - **ローカルアドレス**
  - **プラグラミング**API



2

IPv6 quick review ー特徴ー

NTT Information Sharing Platform Laboratories

- 広大なアドレス空間
  - IPv4：IPv6 = バケツ1杯：太陽の体積
  - 経路集成と経路膨張の抑止
  - NATのないP2P通信
- プラグアンドプレイ
  - アドレス自動設定機能が標準
- セキュリティ機能の標準搭載
- QoS / マルチキャスト
- 経路最適化をサポートしたモビリティ
- より洗練されたプログラミングAPI

3

IPv6 quick review ー標準化の経緯ー

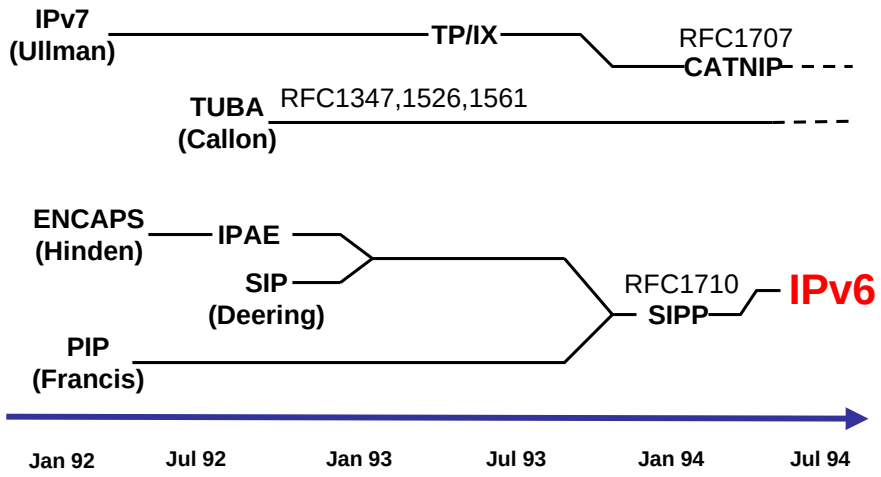
NTT Information Sharing Platform Laboratories

- 1991年7月
  - IPv4アドレスが不足するという研究報告に対するIETFの調査開始
- 1992年11月
  - RFC1380 “IESG Deliberations on Routing and Addressing”において調査結果報告
  - 次世代インターネットプロトコル（IPng）検討開始
- 1993年12月
  - RFC1550 “IP: Next Generation (IPng) White Paper Solicitation” において要求条件提案
- 〜1994年
  - さまざまなプロトコルが提案され、廃案、マージを繰り返し
- 1995年1月
  - RFC1752 “The Recommendation for the IP Next Generation Protocol”においてIPv6へと改名

4

IPngの候補たち (RFC1752)

- 4つの系列



5

Internet Protocolのバージョン番号

http://www.iana.org/assignments/version-numbers

Assigned Internet Version Numbers

Decimal	Keyword	Version
0		Reserved
1-3		Unassigned
4	IP	Internet Protocol
5	ST	ST Datagram Mode
6	IPv6	Internet Protocol version 6
7	TP/IX	TP/IX: The Next Internet
8	PIP	The P Internet Protocol
9	TUBA	TUBA
10-14		Unassigned
15		Reserved

6

IPv6標準化の経緯（基本仕様確定まで）

NTT Information Sharing Platform Laboratories

- 1995年12月
  - RFC1884 “IP Version 6 Addressing Architecture”において128bitアドレス空間の利用方法提案（現在は、RFC3513（2003））
- 1997年4月
  - RFC2133 “Basic Socket Interface Extensions for IPv6”においてプログラミングAPIが定まる（現在は、RFC3493（2003））
- 1998年12月
  - RFC2460 “Internet Protocol, Version 6 (IPv6) Specification”などにおいて標準仕様決定
  - この時点でIPv6の格子がほとんど固まる。
- これ以降
  - 基本仕様の改定と、付加機能の議論が主となる

7

IPv6 Node Requirements

draft-ietf-ipv6-node-requirements-11.txt

NTT Information Sharing Platform Laboratories

- IPv6ノードが満たすべき要件を定義
  - 機能項目が列挙され、それぞれ詳細仕様を参照
    - MUST〜MAYまで網羅的に記載されている
  - アドレス自動設定などの詳細仕様が議論中のため、本ドキュメントも議論中
    - 最新の事情が反映されている
- 入門書ではあまり取り上げられていない項目
  - 踏み込んだ内容
  - 標準化途中の内容

  に注目し、その内容を紹介

8

IPv6 Node Requirements
大まかな節構成

- Sub-IPレイヤ
  - データリンクとの連携
- IPレイヤ
  - 基本プロトコル仕様 RFC2460
  - 近隣探索
  - ICMPv6
  - アドレス自動設定
  - マルチキャスト
  - アドレス選択方式
- DNS／DHCP
- IPv4から移行技術
- モビリティ
- セキュリティ
- ルータ機能
- ネットワーク管理

IPv6 Node Requirements
アドレスの自動設定に関するトピック

アドレス自動設定

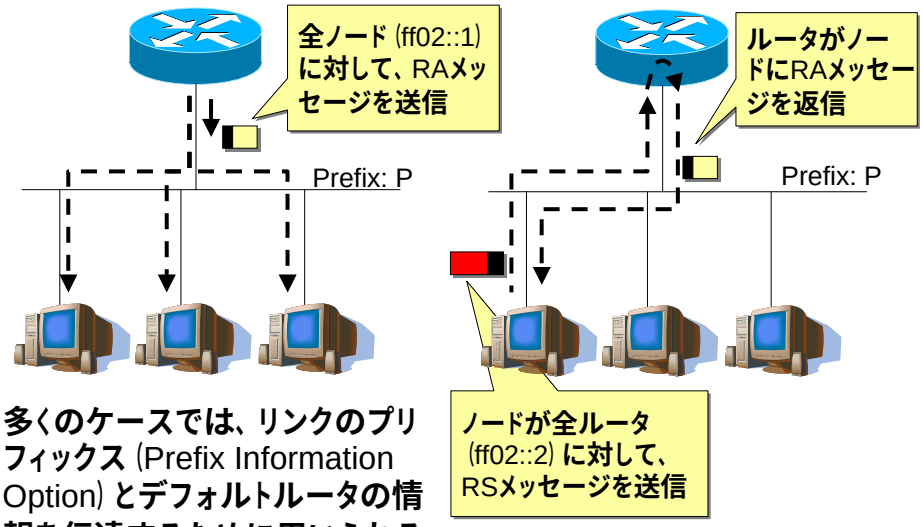
NTT Information Sharing Platform Laboratories

- アドレス自動設定のための2つのモード
  - Stateless Address Autoconfiguration
    - 特徴
      - ノードのアドレスを知ることができない
      - アドレスの割り当てを制限できない
    - 実装例
      - Router Advertisement (RA)
  - Stateful Address Autoconfiguration
    - 特徴
      - ノードのごとに状態を管理できる
      - アドレスの割り当てを制御できる
    - 実装例
      - DHCPv6
- ステートレス, ステートフル間の連携方法も議論中
  - RAにM, Oの2つのフラグを増設し、RAとDHCPv6を連動

11

Router Advertisement (RA) と Router Solicitation (RS)

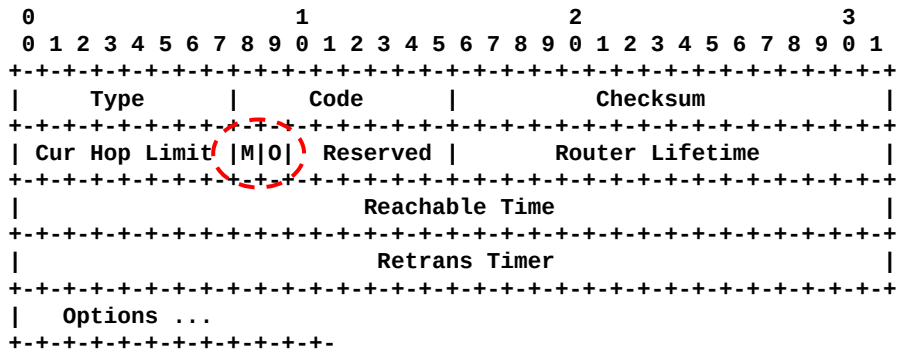
NTT Information Sharing Platform Laboratories



多くのケースでは、リンクのプリフィックス (Prefix Information Option) とデフォルトルータの情報を伝達するために用いられる

12

Router Advertisement メッセージのフォーマット
ManagedFlag (M) と OtherConfigFlag (O)



- M/Oフラグフィールドを設けること自体は、RFC2461, 2462で標準化済
- 詳細な使用方法については議論中

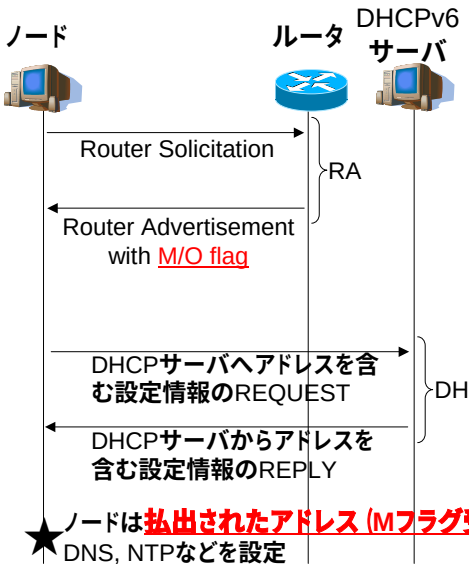
M flag: ManagedFlag
Managed address configuration

- RAのPrefix Informationオプションを用いず、ステートフル手法でアドレスの割当てを要求
  - ステートフルな手法としてDHCPv6が想定される
- 連携方法
  - ノードがRAを受信後、Mフラグを検査。オンならばDHCPv6クライアントを起動しアドレス割り当てを要求する
- RAとの比較
  - ステートフルである (網側からノード管理できる)
  - RAでは渡せない情報 (DNS, NTP) をノードに伝達できる

O flag: OtherConfigFlag
Other stateful configuration

- アドレス以外の情報 (DNS, NTPサーバなど) をステートフル手法により割り当てる
- 連携方法
  - ノードがOフラグつきのRAを受信後、DHCPv6クライアントを起動しサーバから情報を取得
  - アドレスを割り当てる点以外は、動作はMフラグの時とほぼ同じ
- RAとの比較
  - ステートフルである (渡した情報の状態を管理できる)
    - サーバのアドレス変更を通知できる
  - 状態を管理せず情報取得の手段として使うことも可能
    - ステートレスDHCPv6

ステートレス (RA) とステートフル (DHCPv6) 設定の連携シーケンス



- ルータ側の実装状況
  - 多くの実装でraフラグのON/OFFの設定が可能
- ノード側の実装状況
  - DHCPv6クライアントの実装が不十分、連携機能の実装も少ない
  - KAMEでは、Oフラグつきの RAを受信時に任意のスクリプトが起動可能

```
rtsol -O [script_name]
```

ここからDHCPv6を起動

Privacy Extensions for Address Configuration in IPv6, RFC3041

NTT Information Sharing Platform Laboratories

- ［標準化背景］MACアドレスからEUI-64で生成したインターフェイス識別子では、ホストが容易に特定されてしまう
  - Temporary（一時）アドレス
  - Anonymous（匿名）アドレスと呼ばれる
- インターフェイス識別子をランダムに決定し、IPv6アドレスを一定時間で使い捨てる方式
- 短期間でアドレスが変わるのでサーバの運用には不向き
- Node Requirementsでは［SHOULD］と規定

17

匿名アドレスの有効期限の扱い

NTT Information Sharing Platform Laboratories

- RFC3041でのデフォルト値
  - 推奨有効期限が24時間
  - 最大有効期限が7日間

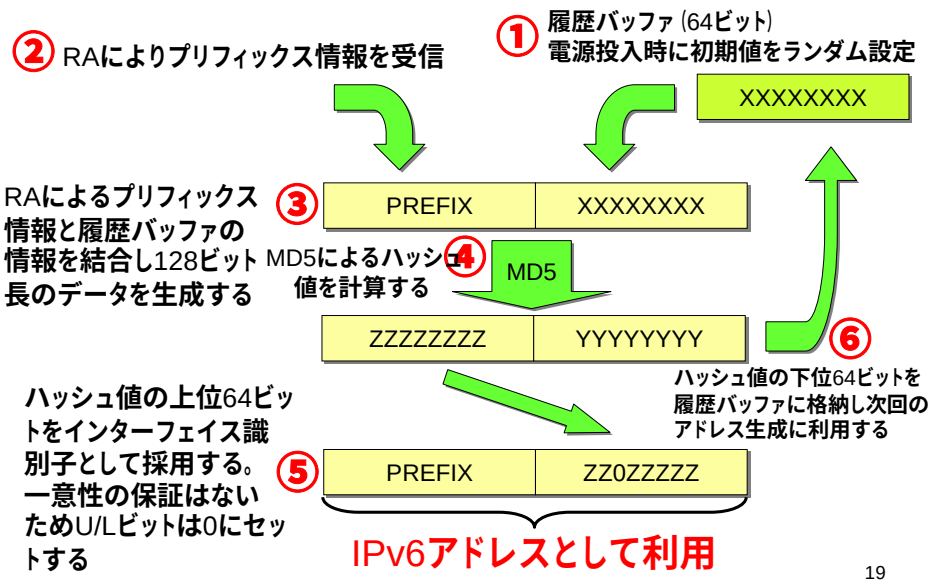
    ※期限の延長は不可
- 参考：EUI-64によるアドレスのデフォルト値
  - 推奨有効期限が7日間
  - 最大有効期限が30日間

    ※期限の延長が可能

18

匿名アドレス生成のアルゴリズム

NTT Information Sharing Platform Laboratories



匿名アドレスの効果

NTT Information Sharing Platform Laboratories

- 履歴バッファとMD5ハッシュを使うことで
  - 現在のアドレスから次に生成されるアドレスの予想が難しい
  - 現在のアドレスから過去に使われたアドレスが推測が難しい
- 履歴バッファを持てない組み込み機器などは単純なランダム生成でもよい
- 改訂版が議論中
  - draft-ietf-ipv6-privacy-addrs-v2-00.txt

Default Address Selection for Internet Protocol version 6 (IPv6), RFC3484

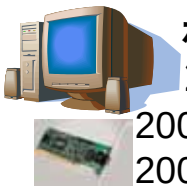
NTT Information Sharing Platform Laboratories

- IPv6ではひとつのI/Fに複数のアドレスを付与できる
  - ＝（自分自身を見たとき）複数のソースアドレスを持つ
  - ＝（外から見たとき）複数のあて先アドレスを持つ
- 機能概要
  - ソースアドレスが複数ある場合、どれを選択するか基準を与える
  - あて先アドレスが複数ある場合、どれを選択するか基準を与える
  - ユーザがポリシーを与えることで任意に制御可能
  - Node Requirementsでは［MUST］とされる
- 実装状況
  - KAME, USAGI, Windows XP/2003で実装済み

21

longest matchによるソースアドレス選択

NTT Information Sharing Platform Laboratories



ホストは1つのインターフェイスを持ち、2つのアドレスが付与されている

2001:db8:aaaa::a/64 ・・・①

2001:db8:bbbb::b/64 ・・・②

- 「2001:db8:abcd::ef」があて先のとき①,②のどちらがソースアドレスになるか？

→プリフィックスの最長一致（longest match）で決定する

2001:db8:
- ab(hex): 1001 1010 ・・・ あて先アドレス
- aa(hex): 1001 1001 ・・・① ➡ ソースアドレスとして選択
- bb(hex): 1010 1010 ・・・②

22

あて先アドレス選択の例

NTT Information Sharing Platform Laboratories

- サーバのホスト名 “www.example.com”を解決したところ、合計4つのアドレスが得られた
  - IPv6
    - ①2001:db8:aaaa:80, ②2001:db8:bbbb::80
  - IPv4
    - ③192.16.178.80, ④192.16.179.80
- このとき、Webブラウザは①～④のどのアドレスへアクセスを試みるのか？

23

あて先アドレス選択の例 (cont.)

NTT Information Sharing Platform Laboratories

- 標準状態での接続試行の順序
  step 1. すべてのIPv6アドレスに接続を試行 (①, ②)
  step 2. すべてのIPv4アドレスに接続を試行 (③, ④)
  step 3. 接続失敗

  ※リゾルバが返却したアドレスのリスト

  ①2001:db8:aaaa:80 → ②2001:db8:bbbb::80 → ③192.16.178.80 → ④192.16.179.80
- 同じプロトコルでの順序はリゾルバに依存。ラウンドロビンで入れ替え

  名前解決 1回目

  ①2001:db8:aaaa:80, ②2001:db8:bbbb::80

  名前解決 2回目

  ②2001:db8:bbbb::80, ①2001:db8:aaaa:80

  名前解決 3回目

  ①2001:db8:aaaa:80, ②2001:db8:bbbb::80

24

アドレス選択のためのポリシーデータベース

NTT Information Sharing Platform Laboratories

- ソースアドレス、あて先アドレスともにユーザが優先させるアドレスを明示的に指定可能
  - IPv6スタック内部に下表のデータベースを構成

Prefix	Precedence	Label	Prefixの意味
::1/128	50	0	ループバックアドレス
::/0	40	1	IPv6アドレス
2002::/16	30	2	6to4アドレス
::/96	20	3	互換アドレス
::ffff:0:0:/96	10	4	IPv4アドレス

- あて先アドレス
  - Precedenceの高いものからマッチ
- ソースアドレス
  - あて先アドレスと同じラベルのものとマッチ

RFC3484が推奨するデフォルトポリシーデータベース

25

Default Address Selection 運用例（1）
あて先アドレス選択の強制

NTT Information Sharing Platform Laboratories

IPv4 → IPv6の順で接続させたい

リゾルバが返すアドレスリストの順序を制御

Prefix	Precedence	Label	Prefixの意味
::1/128	50	0	ループバックアドレス
::/0	40	1	IPv6アドレス
2002::/16	30	2	6to4アドレス
::/96	20	3	互換アドレス
::ffff:0:0:/96	**60**	4	IPv4アドレス

www.example.com IN A 192.16.178.80 ・・①
IN AAAA 2001:db8:aaaa::80 ・・②

リゾルバが返すアドレスリスト

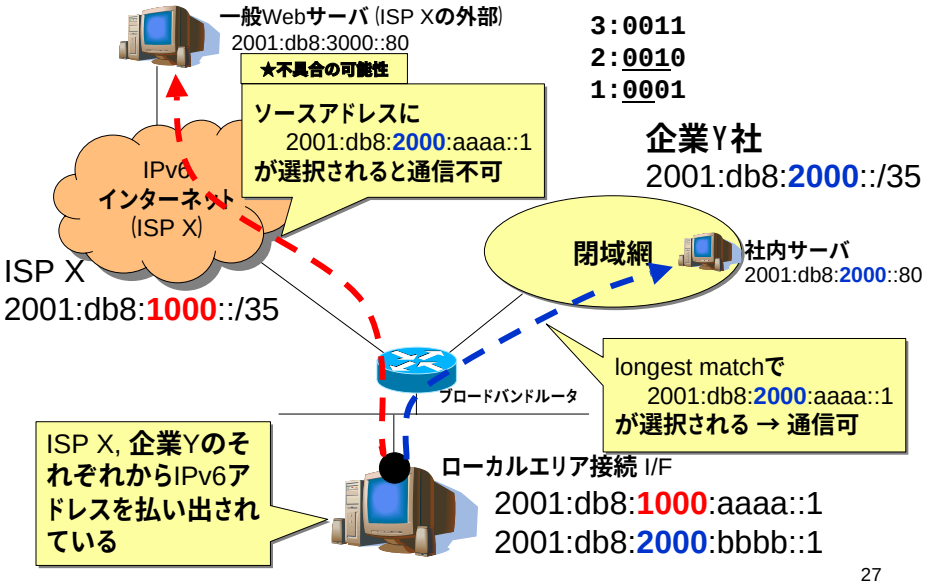
①192.16.178.80, ②2001:db8:aaaa::80

26

Default Address Selection 運用例 (2)

特定ソースアドレスの使用を抑制する

NTT Information Sharing Platform Laboratories



Default Address Selection 運用例 (2)

特定ソースアドレスの使用を抑制する (cont.)

NTT Information Sharing Platform Laboratories

- **インターネットと通信するときは、ISP Xのアドレス (2001:db8:1000::) を必ず用いる**

＝インターネットと通信するときは、企業Yのアドレス (2001:db8:2000::) を絶対使わない

通信相手

2001:db8:3000::80・・①

ソースアドレス

2001:db8:1000:aaaa::1・・②

2001:db8:2000:bbbb::1・・③

Prefix	Precedence	Label
::1/128	50	0
::/0	40	1 ①②
2002::/16	30	2
::/96	20	3
2001:db8:2000::/35	45	5 ③
::ffff:0:0:/96	10	4

Step 1:
あて先①のラベルを求める

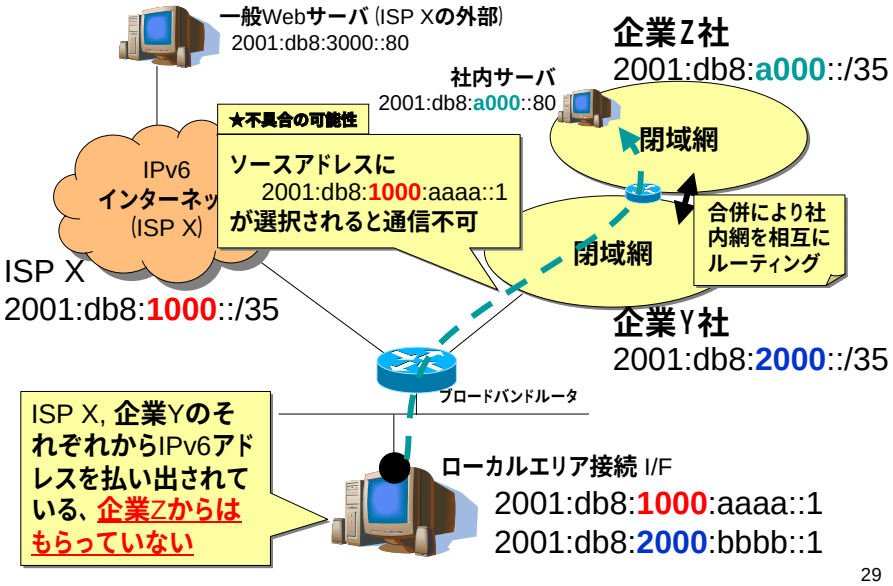
Step 2:
ソースアドレス②,③のラベルを求める

Step 3:
Step 1.とStep 2.でもとめたラベルを比較して、同じラベルのソースをアドレスを選択する

Default Address Selection 運用例（3）

あて先アドレスに応じて特定ソースアドレスの使用する

NTT Information Sharing Platform Laboratories



Default Address Selection 運用例（3）

あて先アドレスに応じて特定ソースアドレスの使用する (cont.)

NTT Information Sharing Platform Laboratories

- 企業Zと通信するときは、企業Yのアドレス（2001:db8:2000::）を必ず用いる
- インターネットと通信するときは、ISP Xのアドレス（2001:db8:1000::）を必ず用いる

通信相手

2001:db8:a000::80・・①

Prefix	Precedence	Label
::1/128	50	0
::/0	40	1 ②
2002::/16	30	2
::/96	20	3
2001:db8:2000::/35	45	5 ③
2001:db8:a000::/35	45	5 ①
::ffff:0:0:/96	10	4

ソースアドレス

2001:db8:1000:aaaa::1・・②

2001:db8:2000:bbbb::1・・③

Step 1：
あて先①のラベルを求める

Step 2：
ソースアドレス②,③のラベルを求める

Step 3：
Step 1.とStep 2.でもとめたラベルを比較して、同じラベルのソースをアドレスを選択する

IPv6 Node Requirements

DNS & DHCP

IPv6 DNSの現状

- **ネームサーバ、リゾルバ** (OS) **ともに**AAAA**に対応した実装がほぼ揃い始めている**
  - AAAA**レコードと**IPv6**アドレスを保持し、クエリに対する返答が可能**
- IPv6**トランスポート上でのクエリはまだ今一歩**
  - Windows XP SP2**は未対応**
  - DNS**サーバの**IPv6**アドレスの特定手法が未定義**

  → DNS**ディスカバリはホットトピックとして議論中**

DNSディスカバリ

NTT Information Sharing Platform Laboratories

- IETFで長らく議論が続き3つの方式に集約
  - Router Advertisement（RA）
  - DHCPv6 option
  - Well-known Anycast Addresses
- 各手法の長短所をまとめたドキュメントを発行し、現在も議論は進行中

  draft-ietf-dnsop-ipv6-dns-configuration-04.txt

33

Router Advertisement（RA）によるDNS発見

NTT Information Sharing Platform Laboratories

- 特徴
  - Router Advertisementにオプション（RDNSS）を増設し、ルータとプリフィックスの通知と同時にDNSサーバの通知を行う
- 長所
  - 既存のRA/NDの方式の上に構築（DNS通知はオプションのひとつ）
  - NTP, SIPなど他のパラメータも（原理的には）通知可能
  - RAが動作するリンクなら、どのメディアでも動作可
  - オーバヘッドが小さい
    - MobileIPv6等のハンドオーバで効き目が出てくる
- 短所
  - カーネル内部の実装を必要とする
    - このレイヤでNTP, SIPなどアプリケーションの設定を扱うべきか疑問視
  - RAを送信するルータの存在を仮定している

34

DHCPv6 optionを用いたDNS発見

NTT Information Sharing Platform Laboratories

- 概要
  - DHCPv6-liteによりDNSサーバのIPv6アドレスを供給する。DHCPv6-liteはサーバから情報の取得のみを行う。
- 長所
  - DHCPv6、DNSオプションの標準化が進み、相互接続可能な実装が多く存在する
  - DNSだけでなく、NTPのような他のネットワークパラメータも設定可能
  - ステートフルならばパラメータを管理可能
- 短所
  - 他の2つの手法に比べオーバヘッドが大きい
  - DNSのためにDHCPv6は重過ぎるという意見もある

35

Well-known Anycast Addresses によるDNS発見

NTT Information Sharing Platform Laboratories

- 概要
  - 規定済みのIPv6アドレスをDNSサーバとする
- 長所
  - トランスポート層に依存しない（L3のみで解決）
  - DNS発見のためのオーバヘッドがない
- 短所
  - IP層（L3）の設定が複雑になる
  - 指定したanycastアドレスで、DNSサーバへの到達性を確保（ルーティング）をする必要がある

36

Well-known**サイトローカルアドレスによる** DNS**との通信**（**参考**）

NTT Information Sharing Platform Laboratories

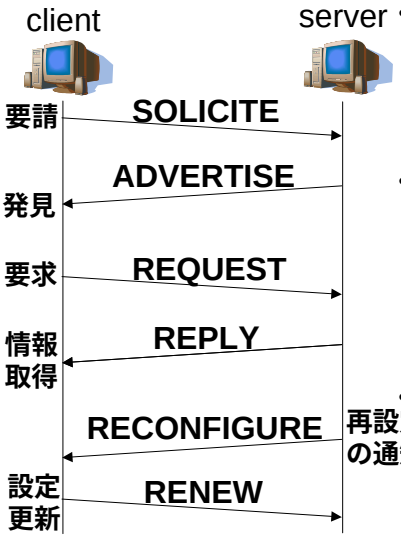
- **サイトローカルアドレスの廃止で、本方式もすでに廃止**
- draft-ietf-ipv6-dns-discovery-08.txt
  - “Well known site local unicast addresses to communicate with recursive DNS servers”
- DNS**用のサイトローカルアドレスをあらかじめ予約する**
  - fec0:0:0:ffff::1, fec0:0:0:ffff::2, fec0:0:0:ffff::3
- Windows XP**などに残骸が見られる**（SP2**でも動作不可**）

```
C:¥>ipconfig /all
Ethernet adapter ローカル エリア接続:
        . . . . . .
DNS Servers . . . . : 192.16.178.230
                      fec0:0:0:ffff::1%1
                      fec0:0:0:ffff::2%1
                      fec0:0:0:ffff::3%1
```

37

Dynamic Host Configuration Protocol for IPv6 DHCPv6, RFC3315

NTT Information Sharing Platform Laboratories

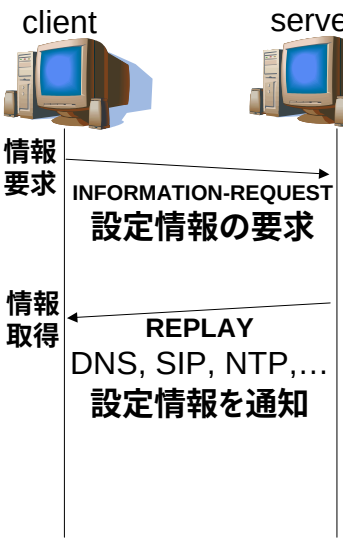


- **ステートフルなアドレス設定方式**
  - **特定のクライアントのみにアドレスを割り当てる**
  - **どのクライアントに、何の情報を割り当てたかサーバで把握できる**
- **クライアントはリンクローカルマルチキャストによりサーバを発見**
  - ff02::2:1（**クライアントが利用**）
    All DHCP Relay Agents and Servers
  - ff05::1:3（**リレーが利用**）
    All DHCP Servers address
- **リンクローカルスコープで**UDP 546/547**番ポートを利用**
  - **すべてレイヤ3**（IPv6）**で完結する**

38

ステートレスDHCPv6

NTT Information Sharing Platform Laboratories



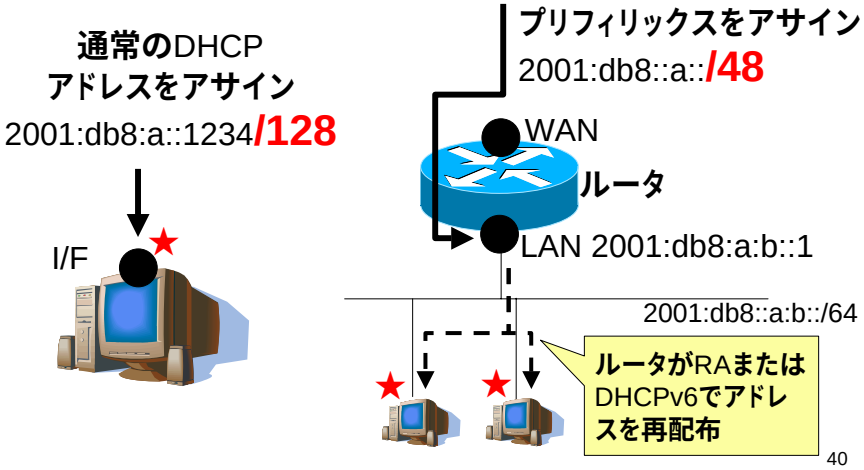
- サーバがクライアントの状態を管理しない方式
- 主にノードに設定情報（DNS, SIP, NTP）を渡すことを想定
  - DNSディスカバリの議論で、軽いDHCPv6が必要とされた
  - DHCPv6-lite とも言われる
- INFORMATION-REQUEST, REPLAYメッセージだけで情報の授受を行う

39

DHCP-PD（Prefix Delegation）

NTT Information Sharing Platform Laboratories

- ホスト（ルータ）にプリフィックスを払い出すオプション



40

国内ISP各社によるコンシューマ向け IPv6サービスの現状

NTT Information Sharing Platform Laboratories

- IPv6ではプリフィックス（/48～/64）を払出す
  - IPv4ではアドレス単体の払い出しが主流
- 接続方式
  - DHCP-PD
    - OCN／アッカ, KDDI, Nifty, BIGLOBEなど
  - IPv6 over IPv4トンネル
    - OCN, IIJ, フリービット, … など多数のISP
  - RA
    - NTT東日本（Flets.NET）

41

DHCP-PDを利用したIPv6 ISPのサービス例

OCN ADSLサービス IPv6デュアル（A）

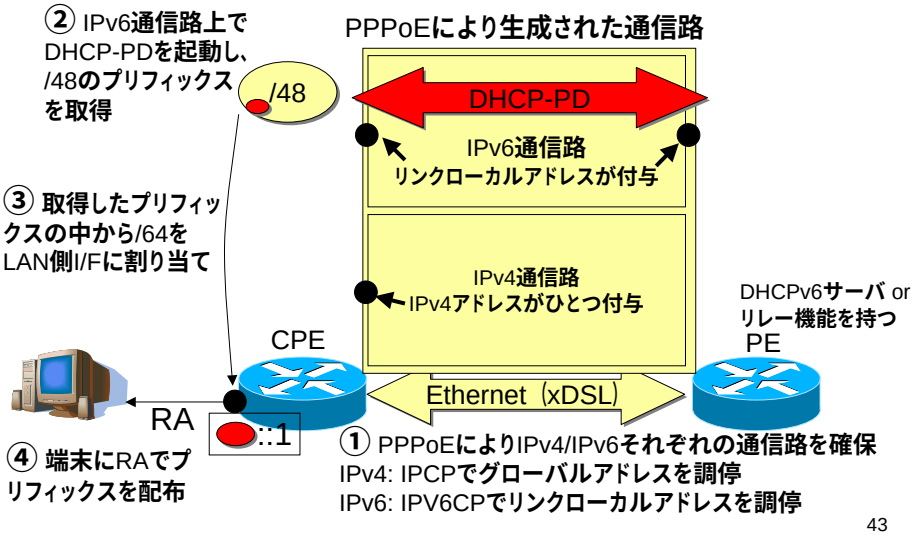
NTT Information Sharing Platform Laboratories

- 概要
  - NTTコム・アッカネットワークスによるIPv4/IPv6コネクティビティが提供されるデュアルサービス
- 顧客の宅内設備（ブロードバンドルータ）に対して
  - IPv4：PPPoE（IPCP）を用いて、IPv4アドレス（ダイナミック）をひとつ払い出す
  - IPv6:PPPoE（IPV6CP）+DHCP-PDを用いて、/48のプリフィックス（固定）をひとつ払い出す
- 仕様
  - 「OCN ADSLサービスIPv6デュアル(A)用 ユーザ網インタフェース仕様書 第2.3版」, http://www.ocn.v6.ntt.net/dual/pdf/ipv6if_ver2.3.pdf

42

DHCP-PDによるプリフィックス割り当ての流れ

PE: Provider Edge
CPE: Customer Premises Equipment



IPv6 Node Requirements

Network Management

ネットワーク管理

NTT Information Sharing Platform Laboratories

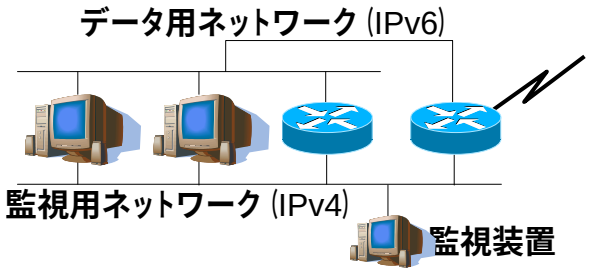
- 概要
  - SNMPによりデバイスの設定／状態取得を行う
  - Node Requirementsで実装は［MAY］とされる
- SNMP対応
  - トランスポートのIPv6対応は容易（実装の問題）
- MIBの標準化
  - 標準化の完了したもの
    - IPv6 MIB, RFC2465
    - IPv6 ICMP MIB, RFC2466
    - IPv6 TCP MIB, RFC2452
    - IPv6 UDP MIB, RFC2454
  - IPv4のMIBドキュメントにIPv6の項目をマージ
    - RFC2011 “IP MIB”, RFC2096 “IP Forwarding MIB” を改定しIPv6対応の記述を盛り込む作業が進行中

45

IPv6ネットワークの監視方法

NTT Information Sharing Platform Laboratories

- デバイスのIPv6対応とは
  - IPv6 MIBを実装する
  - SNMPのトランスポートがIPv6対応している
- 監視システムのIPv6対応
  - トランスポート（情報取得手段）のIPv6対応はまだ少ない
  - 多くのシステムでIPv4で取得した情報を処理できる



IPv6ネットワークの監視のために、監視系にIPv4を使うと効果的

46

ネットワーク監視システムの例
Nagios

- ping, traceroute の監視
- ポート（サービス）の生死
- E-mail, pagerによる管理者への状態通知
- 監視情報の取得は簡易なスクリプトのため、比較的容易にIPv6対応可能
- プラグインによる機能拡張が可能
- 「IPv6実践ガイド IPv6実践ガイド マルチOSで学ぶv4/v6デュアルスタックネットワークの構築・運用方法」翔泳社

http://www.nagios.org/

47

ローカルアドレスに関するトピック

48

サイトローカルアドレス (fec0::/16) の状況

NTT Information Sharing Platform Laboratories

- **サイトローカルアドレス** (fec0::/16) **は廃止が決定**
  - **規格は存在するが使用は推奨されない**
  - "Deprecating Site Local Addresses", RFC3879
- **サイトローカルアドレスの問題**
  - **スコープ** (**サイト境界**) **の定義があいまいになる**
    - **グローバル空間へのリーク**
    - DNS**クエリ・ソースアドレスがサイトローカルのパケット**
  - **サイトの併合時に重複が発生する可能性**
    - IPv4**のプライベートアドレスでもほぼ同じ問題が指摘されている**
    - **同一のアドレス空間を利用していた企業間連携・合併時の問題**
    - NAT**を助長する恐れがある**
- **しかしローカルアドレスは有効かつ必要と言う認識**

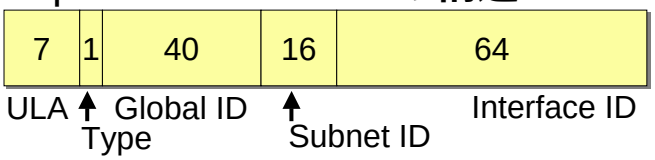
49

Unique Local IPv6 Addresses（1）

NTT Information Sharing Platform Laboratories

- 2003**年末からサイトローカルアドレスの代替手段について議論が始まる**
  - fc00::/8 **管理組織が割当** (Centrally assigned)
  - fd00::/8 **独自割り当て** (Locally assigned)
    - **ローカル割り当てのサンプルとして**Global ID**をMD5で決める方法が示されている。**

      trunc(MD5(local time + EUI-64), 40bit)
- Unique Local Address**の構造**



50

Unique Local IPv6 Addresses（2）

NTT Information Sharing Platform Laboratories

- **提案から1年以上を経過し、必要性も強く認識されているが標準化に至っていない**
  - 2004年9月24日に draft-ietf-ipv6-unique-local-addr-06.txt **が発行され議論が継続中**
- Open Issues
  - **グローバル空間へのリーク問題**
    - **ルーティング**, DNS**クエリ**
  - **グローバル空間とのマルチホーム時の挙動**
    - **ローカルとグローバルの使い分けをどうすべきか？**
    - **アドレス選択が正しく動くか？**
  - two-faced DNS**サーバの返答の問題**

51

文書作成用アドレス空間 (2001:db8::/32)

NTT Information Sharing Platform Laboratories

- APNIC**が文書作成用にアドレス空間を予約**
- **マニュアルや設定サンプルへの利用を想定**
- **明示的な割り当て (申請) は必要ない**
- **ローカルアドレスではないので通信のために利用してはならない**

- **参考ドキュメント**
  - APINIC “IPv6 Documentation Prefix”
    http://www.apnic.net/info/faq/ipv6-documentation-prefix-faq.html
  - “IPv6 Address Prefix Reserved for Documentation”, RFC3849

52

MobileIPv6
NEMO (Network Mobility)

IPv6上でモビリティを実現するプロトコル
MobileIPv6

- Node Requirementsでは [MAY] とされる
- 位置透過性を提供
  1. 接続リンクが変更になっても同一アドレスでノードを参照できる
  2. 接続リンクが変更になっても進行中の通信 (セッション) を維持できる

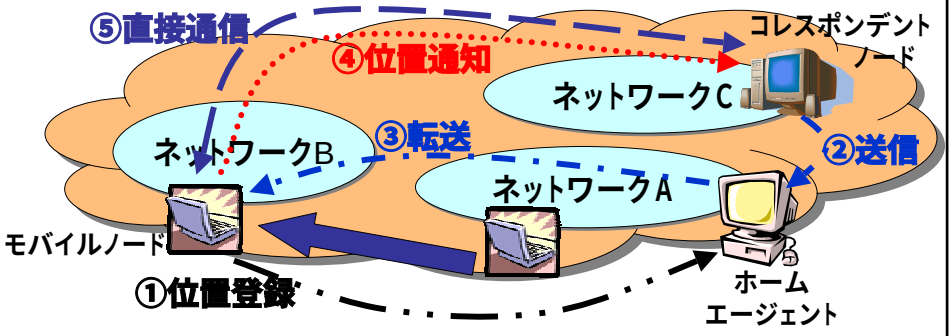
- IPv6レイヤのみで実現
  - アプリケーションおよびL2のサポートが不要
    (アプリケーションの改造もL2の改造も不要)

⇒ IPv6コネクティビティさえあれば位置透過な通信が可能

- Mobile IPv4からの改善点
  - 経路最適化が標準で定義
  - Foreign Agentの廃止
  - ソースアドレス詐称が無い
    ⇒ ファイヤウォールのingress filterに整合

Mobile IPv6 の動作の概略

NTT Information Sharing Platform Laboratories



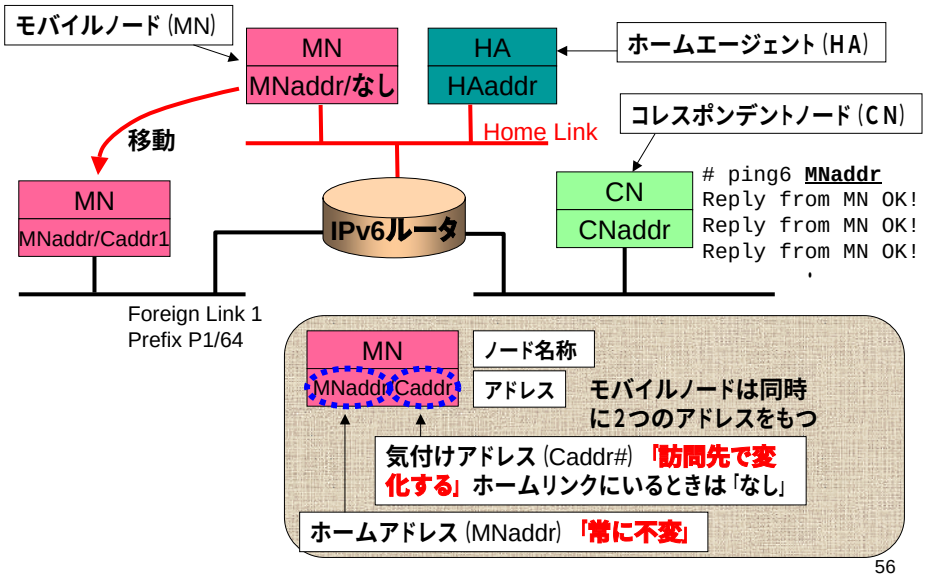
① 移動端末はホームエージェントに**位置登録**
② 相手端末は、通信開始時はホームエージェント経由でパケットを**送信**
③ ホームエージェントが移動端末にパケットを**転送**する
④ 移動端末は相手端末にも**位置通知**する
⑤ 通信開始後は、移動端末と相手端末は**直接通信**

Mobile IPv6の特徴

55

Mobile IPv6を用いた移動通信

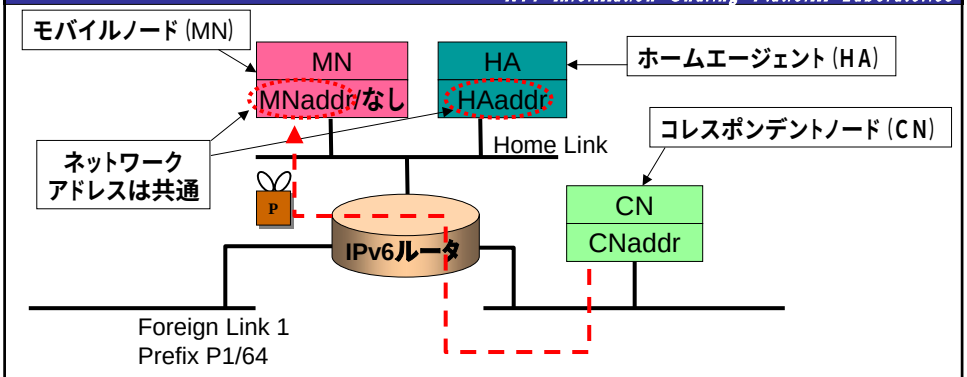
NTT Information Sharing Platform Laboratories



56

MNがHome Linkに存在

NTT Information Sharing Platform Laboratories

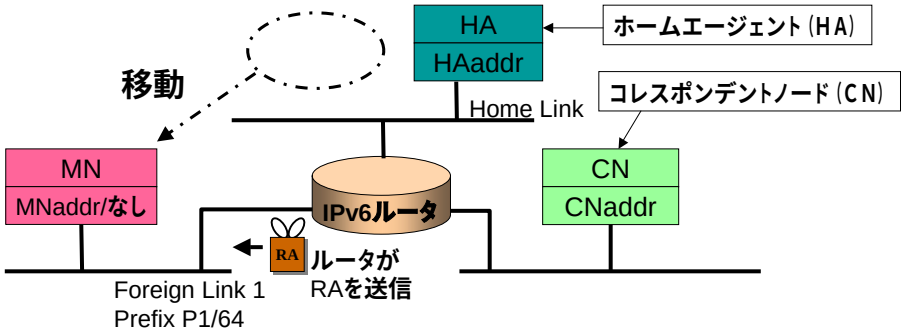


通常のIPv6通信が行われる

57

MNは自分が移動したことを検出する

NTT Information Sharing Platform Laboratories



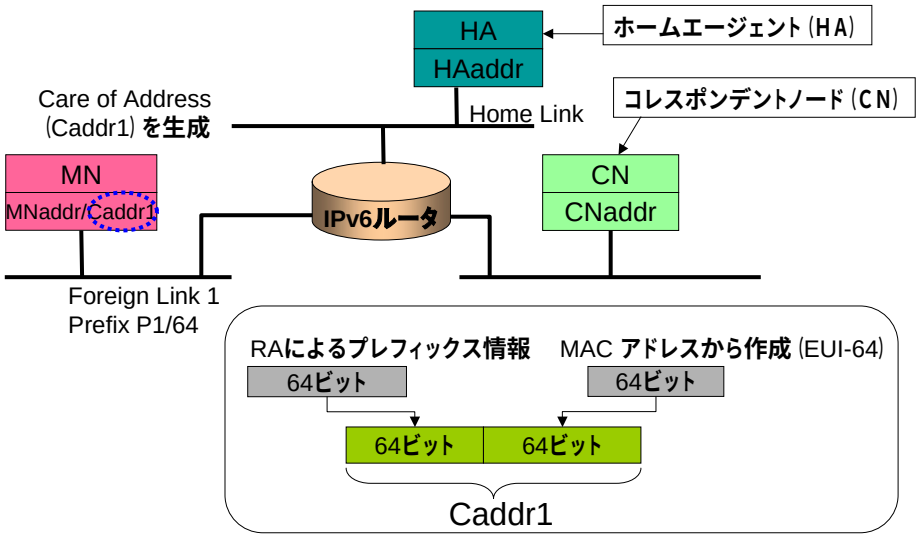
- RAで受信したプリフィックスがホームアドレスのものと一致しない
- 直前に受け取ったプリフィックスと一致しない

→ MNは移動したと判断

58

MNは気付けアドレスを生成する

NTT Information Sharing Platform Laboratories

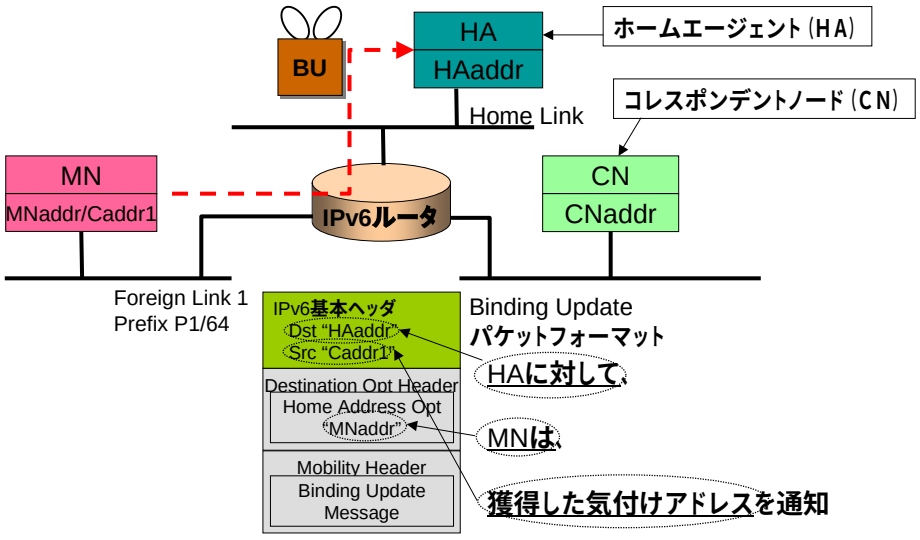


59

MNが気付けアドレスをHAに通知する（位置登録）

HAへのBinding Update

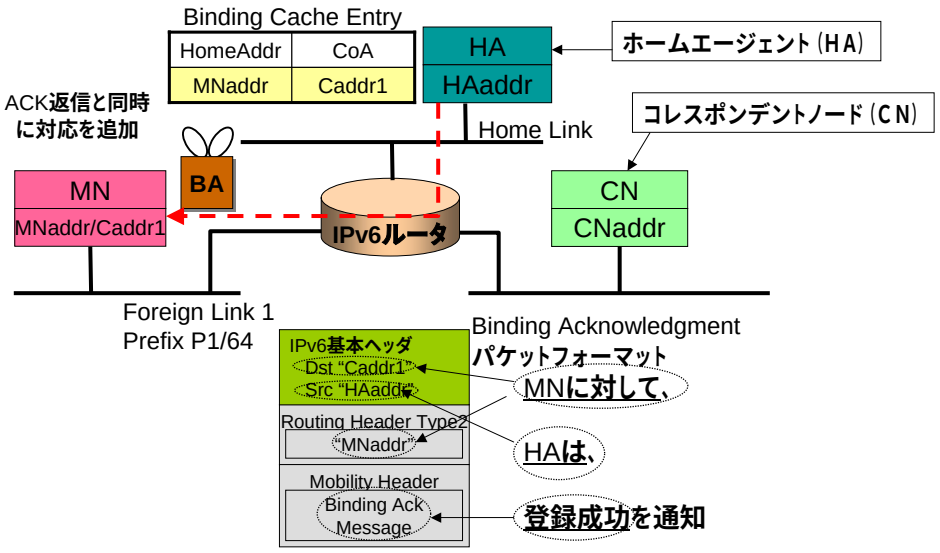
NTT Information Sharing Platform Laboratories



60

HAが気付けアドレスを確認し登録成功を通知
Binding Acknowledgment

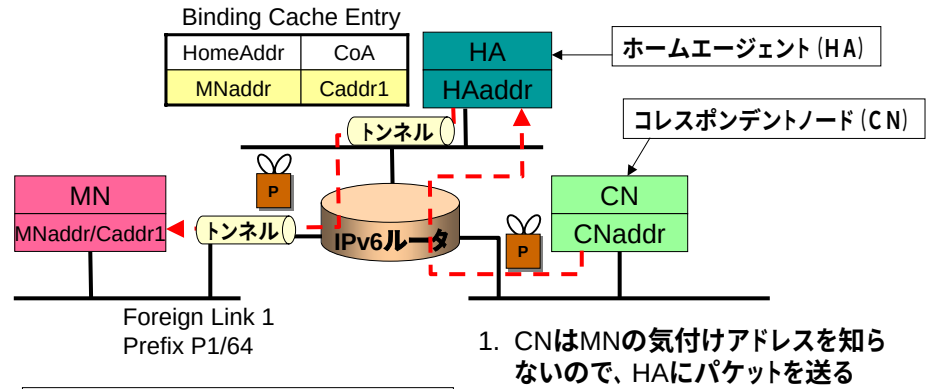
NTT Information Sharing Platform Laboratories



61

CNがMNに向かって通信を開始

NTT Information Sharing Platform Laboratories



1. CNはMNの気付けアドレスを知らないので、HAにパケットを送る
2. HAはBinding Cacheの対応関係を調べ、転送先の気付けアドレス（Caddr1）を発見する。
3. トンネル（カプセル化）によりCNからのパケットを転送する

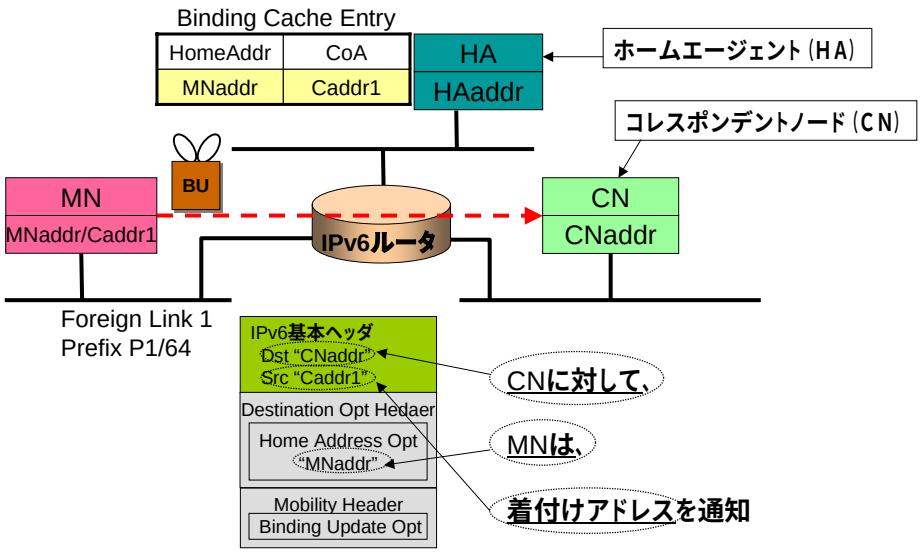
4. MNはCN発のパケットをHA経由で受け取った場合、CNがBinding Cacheを持たないと判断する

62

MNがCNに対して気付けアドレスを通知する

※通知前にReturn Routability手続きが必要だが簡略化のため別途解説

NTT Information Sharing Platform Laboratories

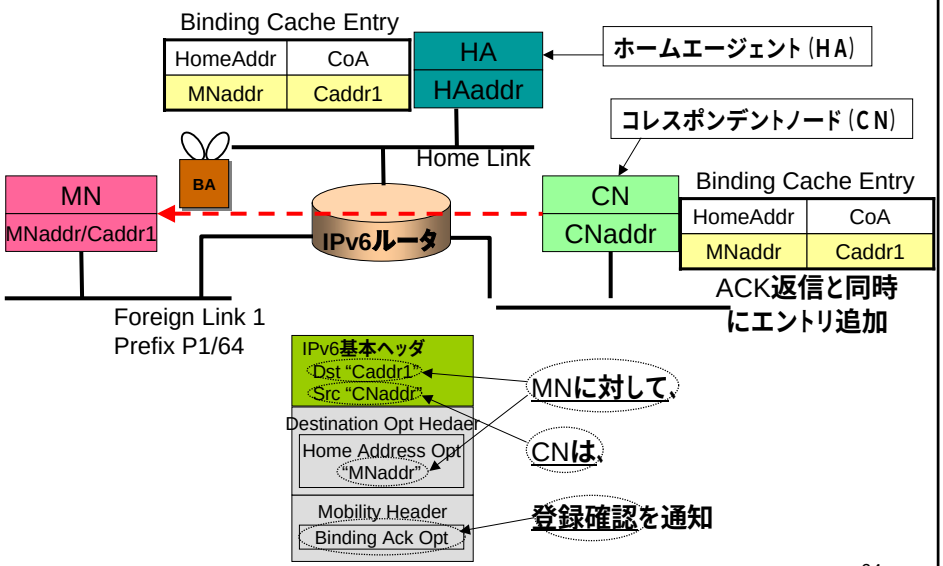


63

CNが気付けアドレスを確認（Acknowledge）する

※Return Routability Procedureは別途解説

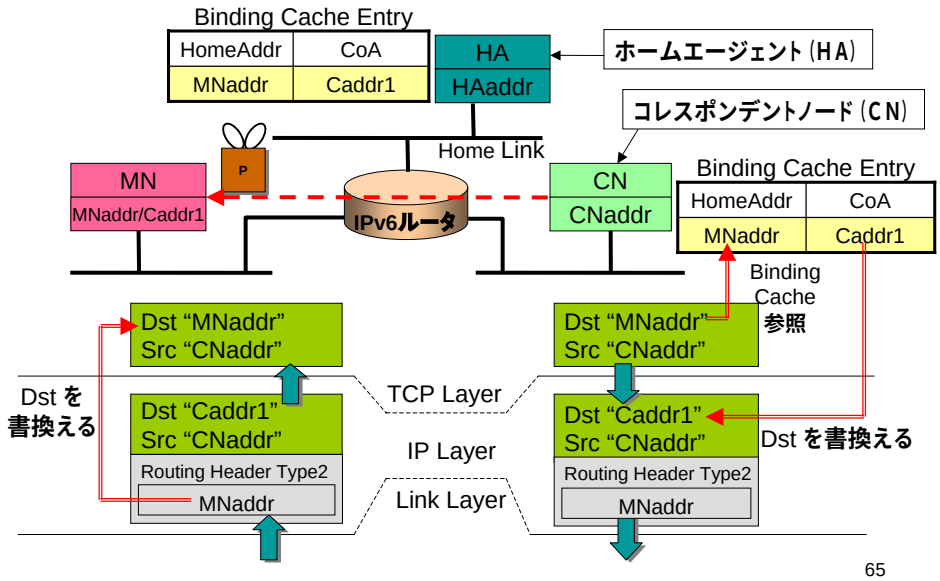
NTT Information Sharing Platform Laboratories



64

CNはBinding Cacheを使いMNにパケットを送信

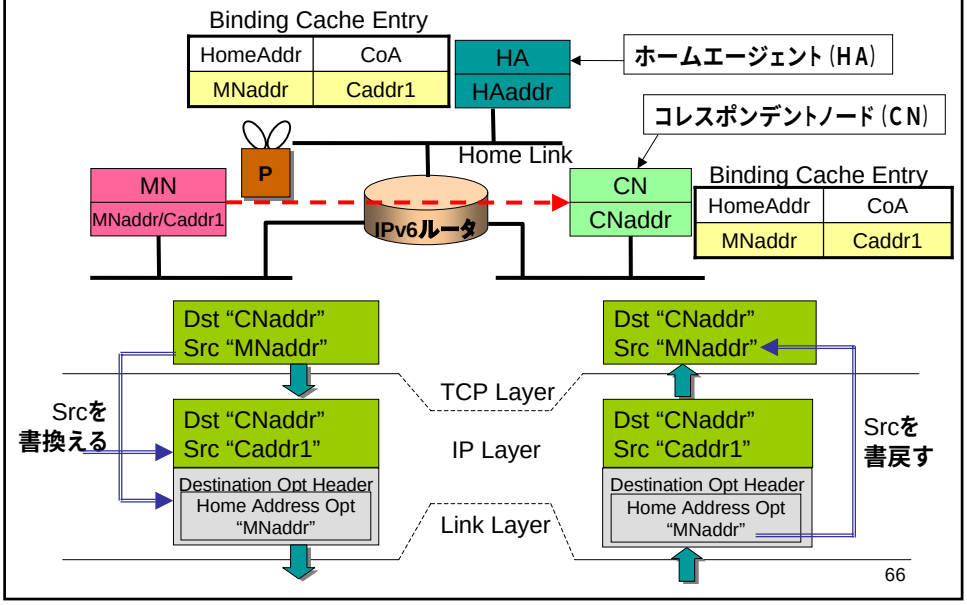
NTT Information Sharing Platform Laboratories



65

MNはHome Addressオプションを使い CNにパケットを送信

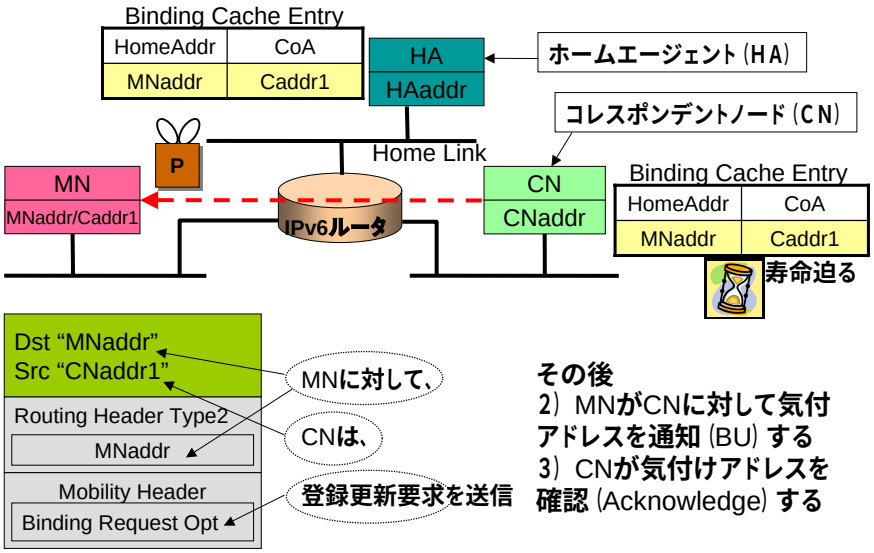
NTT Information Sharing Platform Laboratories



66

CNはBinding Cacheの寿命が切れそうになったらMNにBinding Requestを送信

NTT Information Sharing Platform Laboratories

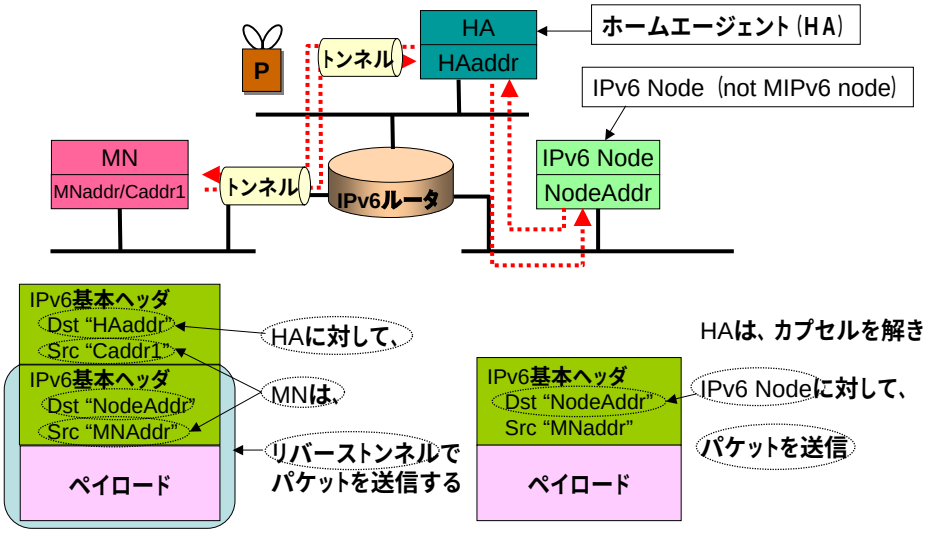


その後
2）MNがCNに対して気付アドレスを通知（BU）する
3）CNが気付けアドレスを確認（Acknowledge）する

67

Mobile IPv6 を実装しないノード（Binding Cacheを持たないCN）との通信

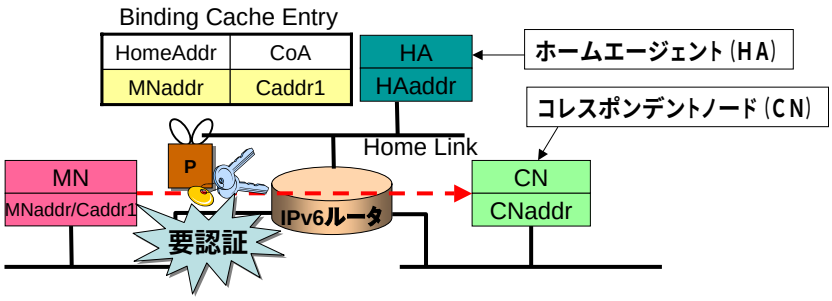
NTT Information Sharing Platform Laboratories



68

MN⇔CN間の認証手続き
Return Routability

CNによるMNの認証の必要性



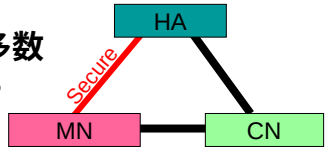
- 成りすまし防止のためCNはMNを認証する
- （現状では）一般にCNはMNの認証鍵を持たない
- HA⇔MN間はIPsecにより安全という仮定を置く

⇒ Return Routability（RR）手続きを発行

Return Routabilityの特徴と問題点

NTT Information Sharing Platform Laboratories

- **特徴**
  - CNにアクセスしてくるMNは不特定多数
    - CNが公開Webサーバだったとしたら・・・
    - あらかじめMNの鍵を持つことは困難
    - PKI + IPsec は現状では非現実的
  - HAがMNを認証していることを前提とする
    - RRはHA⇔MN間の通信が信頼できることを利用
- **問題点**
  - CNの近傍 (non-IPsec区間のCN⇔HA通信) の盗聴には無防備

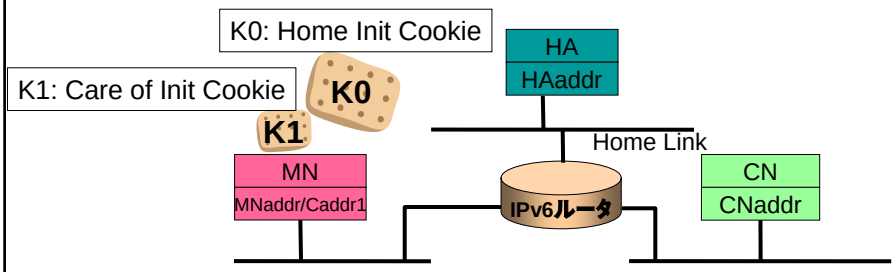


⇒ 不特定多数の認証と不完全な認証の折衷案

71

Return Routability手続き STEP 1
MNはクッキー (K0, K1) を生成

NTT Information Sharing Platform Laboratories

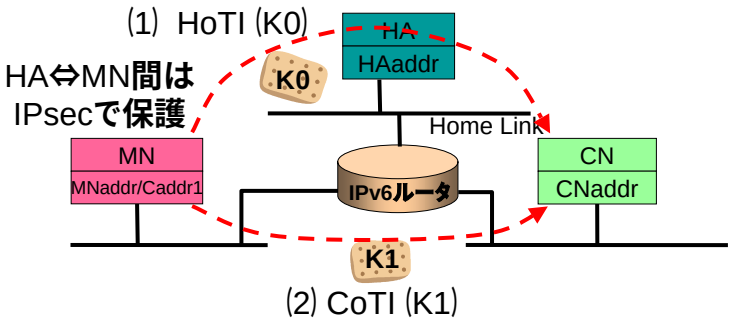


K0: Home Init Cookie
K1: Care of Init Cookie
※いずれも64ビット長の乱数

72

Return Routability手続き STEP 2
MNは2種類のTest Initメッセージを送信

NTT Information Sharing Platform Laboratories

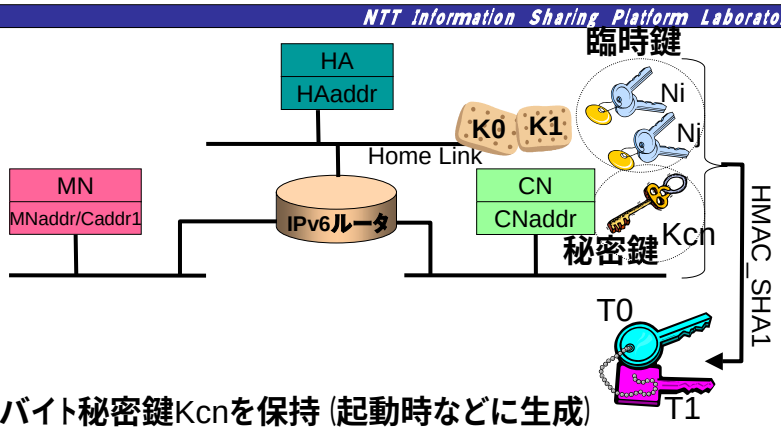


MNはCNへ2種類のメッセージを同時に送信
(1) Home Test Init (HoTI)　(クッキーK0を含む／ホームエージェント経由)
(2) Care of Test Init (CoTI)　(クッキーK1を含む／CNへの直接配送)

73

Return Routability手続き STEP 3
CNは署名用トークン (T0,T1) を生成

NTT Information Sharing Platform Laboratories

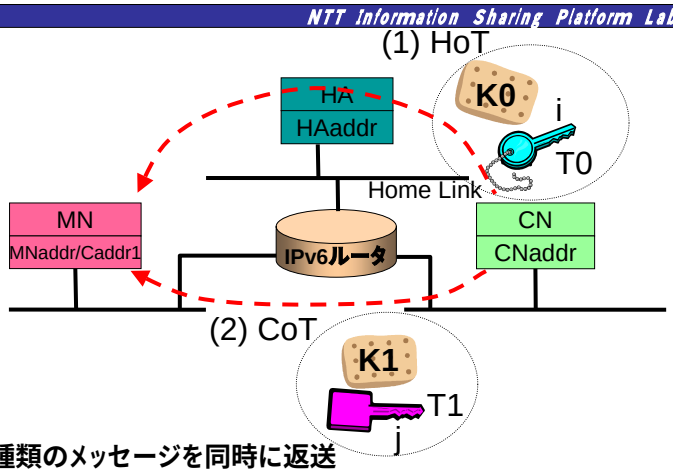


CNは20バイト秘密鍵Kcnを保持 (起動時などに生成)
CNは臨時鍵Ni, Njを乱数で生成 (i, j は鍵を区別する通し番号)
ハッシュにより2つの署名用トークンT0, T1を生成
(1) home keygen:　T0 = HMAC_SHA1(Kcn, (MNaddr+Ni+0x00))
(2) care of keygen: T1 = HMAC_SHA1(Kcn, (Caddr1+Nj+0x01))

74

Return Routability手続き STEP 4
CNはMNへ署名用トークン (T0,T1) を返送

NTT Information Sharing Platform Laboratories



CN はMNへ2種類のメッセージを同時に返送

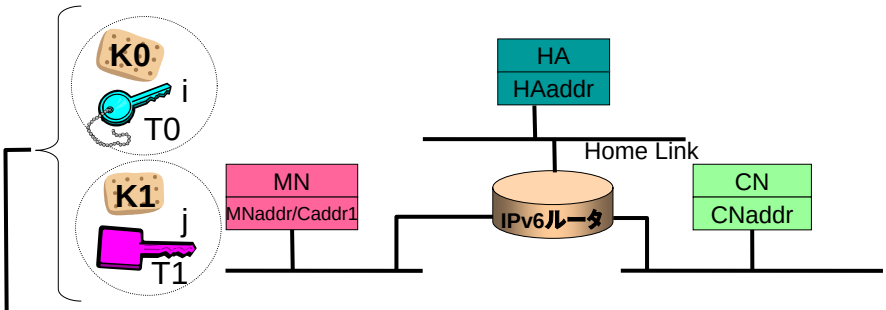
(1) Home Test (HoT)
cookie K0, 署名用トークンT0, 臨時鍵番号 i を含む／HA経由

(2) Care of Test Init (CoT)
cookie K1, 署名用トークンT1, 臨時鍵番号 j を含む／直接配送

75

Return Routability手続き STEP 5
MNは署名鍵 (T0,T1) から署名MACを生成

NTT Information Sharing Platform Laboratories



Binding Key の生成　Kbm = SHA1 (T0+T1)

MNはBinding Updateメッセージを生成し、Kbmで署名(MAC)を生成する

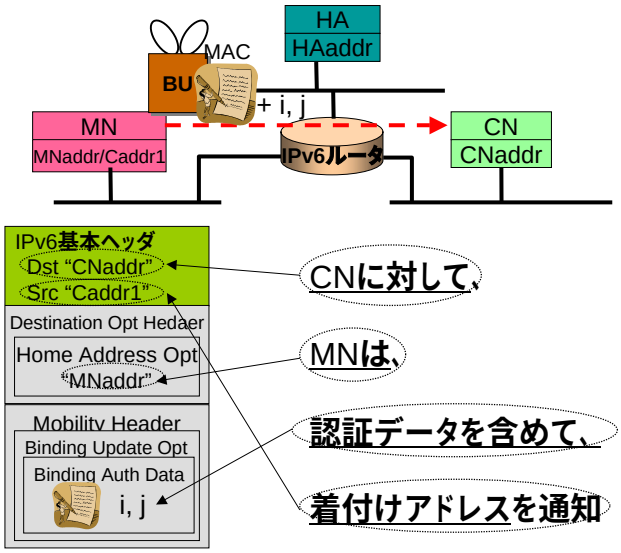
MAC = HMAC_SHA1 (Kbm, (Caddr1+CNaddr+BU))

Binding Updateメッセージ自身のデータ

76

Return Routability手続き STEP 6
MNはCNへ署名つきBinding Updateを送信

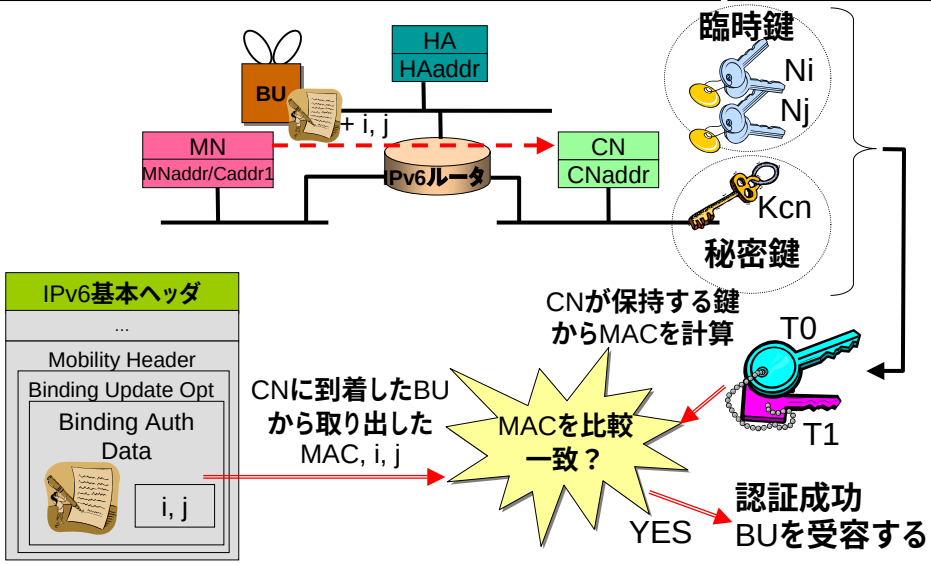
NTT Information Sharing Platform Laboratories



77

Return Routability手続き STEP 7
CNはBUに含まれる署名を検証し正しければ受容

NTT Information Sharing Platform Laboratories



78

Mobile IPv6 がもたらす効果

NTT Information Sharing Platform Laboratories

- IPはメディアに非依存の性質を利用すると
  - アクセス統合／ISP統合が実現
- ホームエージェントサービスの可能性
  - IP層の接続性が特定アクセス網／ISPから分離
  - ISPが変わっても同じアドレスを使い続けられる
  - 考え方はMSP (Mail Service Provider) と類似
    xxxxx@hotmail.com はISPに依存しない

79

Mobile IPv6 の課題

NTT Information Sharing Platform Laboratories

- ハンドオーバ
  - ハンドオーバ時間の短縮
    - FMIP (Fast MIP), HMIP (hierarchical MIP)
  - 異メディア間／異ISP間のハンドオーバ
    - ISPによりアカウントやアクセス方法はまちまち
- ホームエージェントオペレーション
  - 本来のインターネットは自立分散アーキテクチャ
  - Mobile IPv6は超集中型アーキテクチャ
- 認証機構やセキュリティ機能の強化
  - 標準化のホットトピック
- できないこと
  - セッションの保存
  - 端末間のコンテキスト移送

80

レイヤ別 位置透過性確保手法

NTT Information Sharing Platform Laboratories

［レイヤ4以上］
- SIP, Dynamic DNS, 各種VPN
- セッションの移送（例 VNC, Windows Terminal Service, SunRay）

○ セッションが保存できる
○ 端末間移動ができる

× 特定アプリケーションやサーバが必要
× 資源利用がサーバに集中

［レイヤ3］Mobile IP, LIN6 など

○ アプリケーションは変更不要
○ メディアに依存しない（IPはメディア非依存）

× ハンドオーバ時間が長い
× セッション保存や端末間セッション移送は不可

移動に対してシームレス

［レイヤ2］携帯電話, PHS, 無線LAN（L2）ローミング

○ プロトコルスタック／アプリケーションは変更不要
○ ハンドオーバ時間が短い

× メディアが単一
× 他のメディアに比べて低速

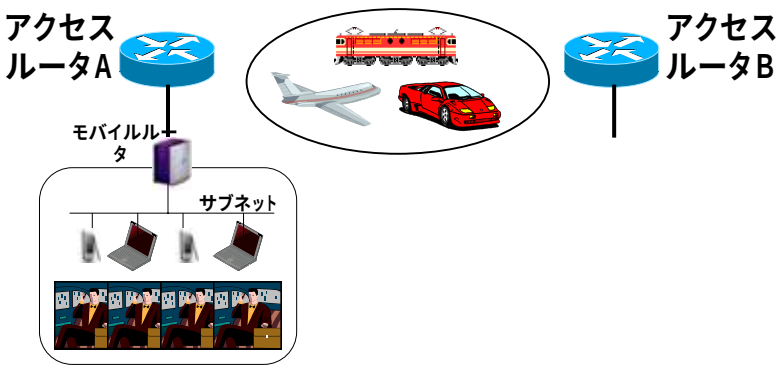
81

ネットワークモビリティ技術

サブネットワークが丸ごと移動

NTT Information Sharing Platform Laboratories

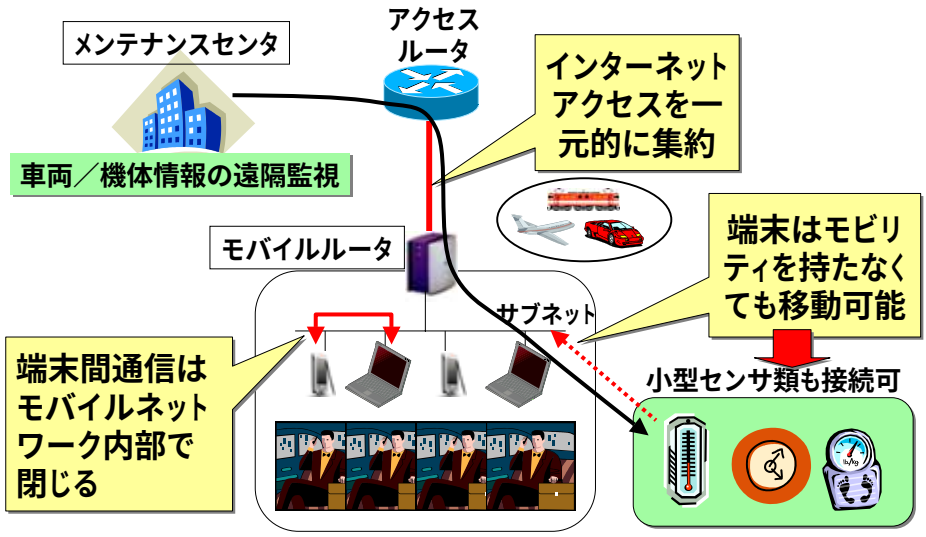
電車、自動車、飛行機など乗り物では
– 乗客 ／ センサーなど
複数のデバイスが同時に移動する



82

ネットワークモビリティの応用例

NTT Information Sharing Platform Laboratories



83

IETF NEMO WGの概要

NTT Information Sharing Platform Laboratories

- MobileIPv6をベースとして、移動ネットワークの実現手法を議論

- 議論中の項目
  - ベーススペックのFIX (NEMO Basic Support)
  - マルチホーミング
  - 利用シーンの検討
  - ホームエージェントのリダンダント化
  - 経路最適化

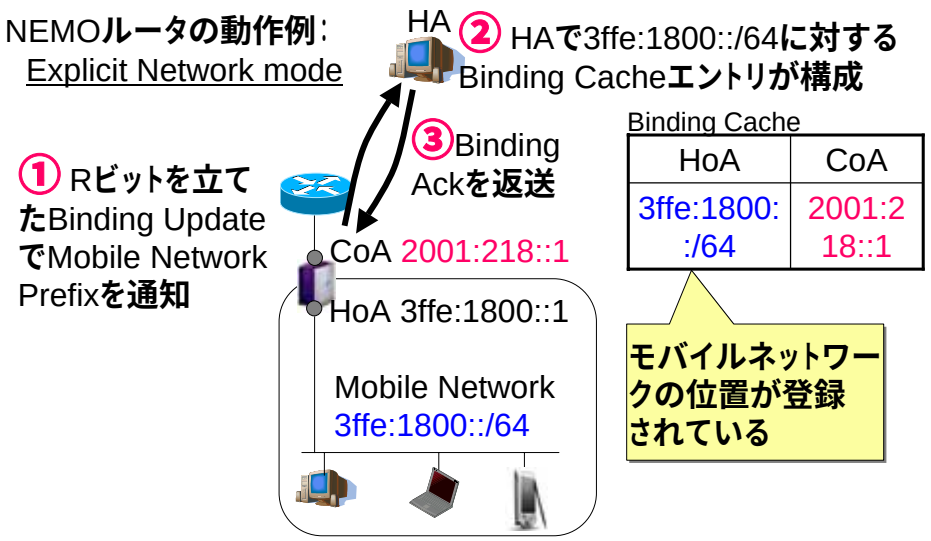
84

NEMO Basic Support Protocol (1)

draft-ietf-nemo-basic-support-03.txt

NTT Information Sharing Platform Laboratories

NEMOルータの動作例:
Explicit Network mode



Binding Cache

HoA	CoA
3ffe:1800::/64	2001:218::1

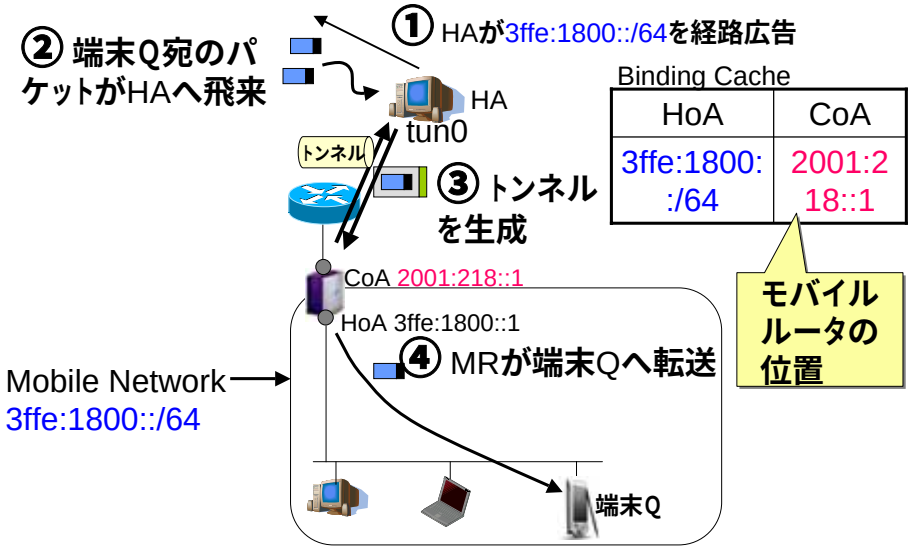
モバイルネットワークの位置が登録されている

85

NEMO Basic Support Protocol (2)

draft-ietf-nemo-basic-support-03.txt

NTT Information Sharing Platform Laboratories



Binding Cache

HoA	CoA
3ffe:1800::/64	2001:218::1

モバイルルータの位置

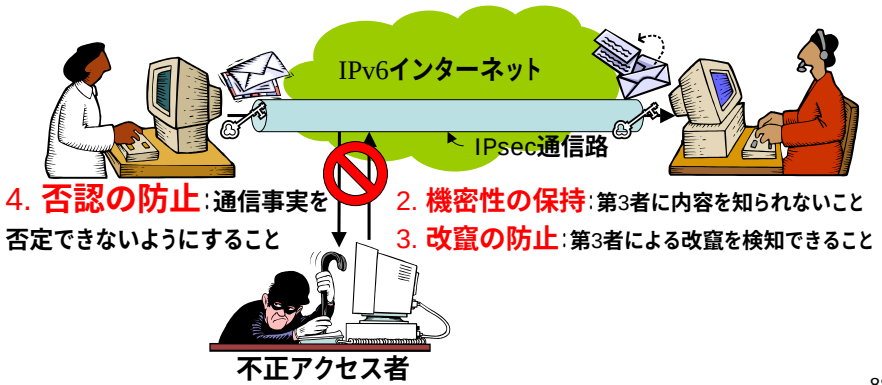
86

IP security（IPsec）

IP Security（IPsec）

- インターネット上にセキュアな通信路を確立する技術
- インターネットの基本プロトコルレベルで**4機能**を実現
- Node Requirementsでは［MUST］と規定

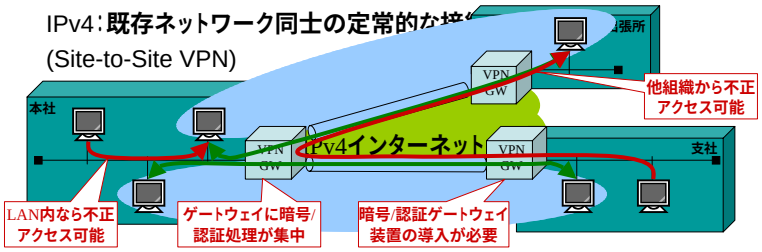
1. **認証**：通信相手を確認できること



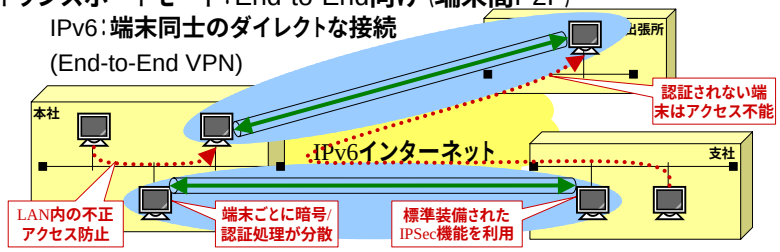
4. **否認の防止**：通信事実を否定できないようにすること
2. **機密性の保持**：第3者に内容を知られないこと
3. **改竄の防止**：第3者による改竄を検知できること

IPsecの2つの通信モード

- **トンネルモード**：Site-to-Site**向け**（**サイト間**VPN）
  IPv4：**既存ネットワーク同士の定常的な接**
  (Site-to-Site VPN)



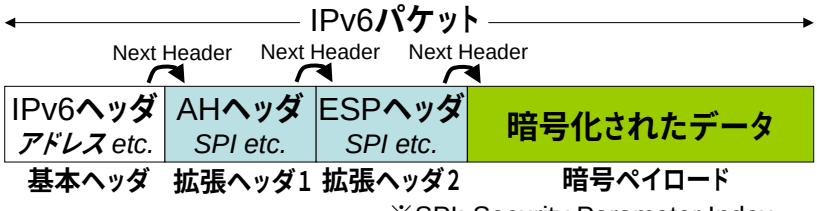
- **トランスポートモード**：End-to-End**向け**（**端末間**P2P）
  IPv6：**端末同士のダイレクトな接続**
  (End-to-End VPN)



89

IPsecの基本機能

- **認証ヘッダ** (AH)
  - **パケット全体の改ざんの防止、認証、否認の防止**
  - **アルゴリズム**：MD5, SHA1 **など**
- **暗号化ペイロードヘッダ** (ESP)
  - **機密性の保持**
  - **ペイロードの改ざん防止**
  - **アルゴリズム**：DES, 3DES, AES **など**



※SPI: Security Parameter Index

90

IPv6でのIPsec標準搭載（MUST）の効果

NTT Information Sharing Platform Laboratories

- End-to-End（P2P）で通信可能
  - ホスト単位でVPN参加者を制御できる
    - ゲートウェイの配置に依存しない柔軟なVPNを構成
  - IPv6 P2Pアプリケーション発展の期待
    - VoIP、IP携帯、テレビ会議、インスタントメッセージなど
- NATの排除
  - NATによりIPsec機能が制約されない
  - NAT traversalなどの補助機能の追加も不要
- 高価なゲートウェイ装置が不要
  - だがIPv4と同様の構成をとることも可能

91

IPv6 IPsecの現状と問題点

NTT Information Sharing Platform Laboratories

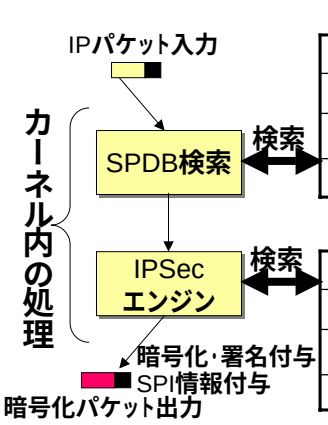
- 現状
  - IPsec（AH, ESP）は安定した実装が多い
    - 処理の大部分はIPv4とほぼ同じ
  - 上位層オフローディング機能搭載のNIC登場
    - IPsecのハードウェア処理機能が載りつつある
- 問題点
  - MUSTだが実装していないOSがある
    - 組み込み系にとってIPsecはまだ重い
  - トランスポートモードはホスト毎の設定が必要
    - インターネット鍵交換プロトコルが不十分

92

IPsec動作のためのデータベース
ホストが持つ2つのDB

NTT Information Sharing Platform Laboratories

- SP (セキュリティポリシ)
  - IPSec通信の適用/不適用を決定するポリシDB
- SA (セキュリティアソシエーション)
  - IPSec通信の暗号化・復号化に必要となる鍵DB



SPDB

Dst	Src	Protocol
2001:db8::1	2001:db8::9	AH
2001:db8::2	2001:db8::9	ESP
…	…	…

SADB

SPI	Dst	Src	Algorism	Key
3001	2001:db8::1	2001:db8::9	MD5	Himitsu
3002	2001:db8::2	2001:db8::9	3DES	Secret
	…	…	…	

93

セキュリティアソシエーションの共有

NTT Information Sharing Platform Laboratories

- 両端でのSA (符号化方式・鍵) の共有が必要
- 事前設定 (pre-shared) ではアプリケーションに限界
  - 公開Webサーバ
  - P2Pアプリケーション (不特定多数から着信するVoIP)
  - IPsec通信ではSPIだけが運ばれる

SADB

SPI	Dst	Src	Algorism	Key
3001	2001:db8::1	2001:db8::9	AES	Himitsu

SPI	Dst	Src	Algorism	Key
3001	2001:db8::9	2001:db8::1	AES	Himitsu

SADB

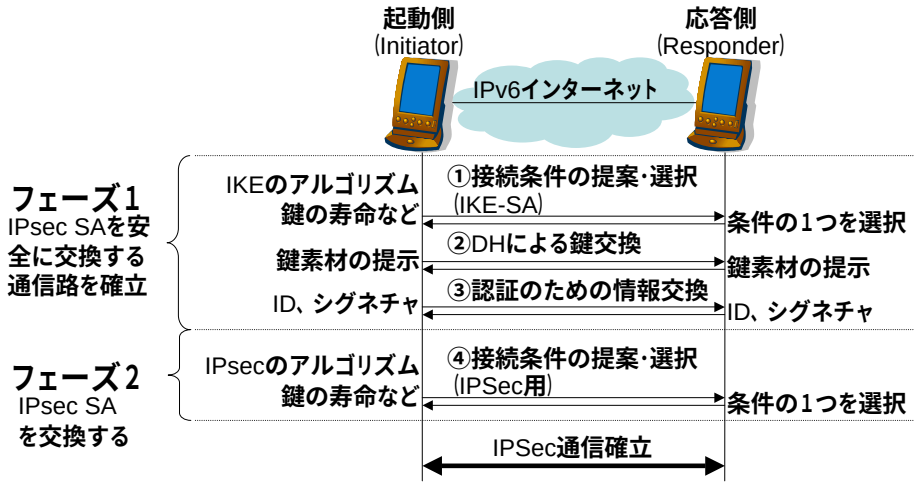
94

インターネット鍵交換プロトコル (IKE)

- 目的
  - 不特定多数のホストと自動的にSAを交換しIPsec通信を行う
- IKEでの相互認証方式 (IPsec SA交換用の通信路を確保)
  - 事前共有秘密鍵 (これはスケールしない)
  - デジタル署名方式 (PKIを利用)
  - 公開鍵を使った暗号文で認証 (RSAの仕組みを利用)
- 現状
  - 仕様も設定も非常に複雑。改善のためIKEv2が策定中
  - IPv6に対応した実装は多くない

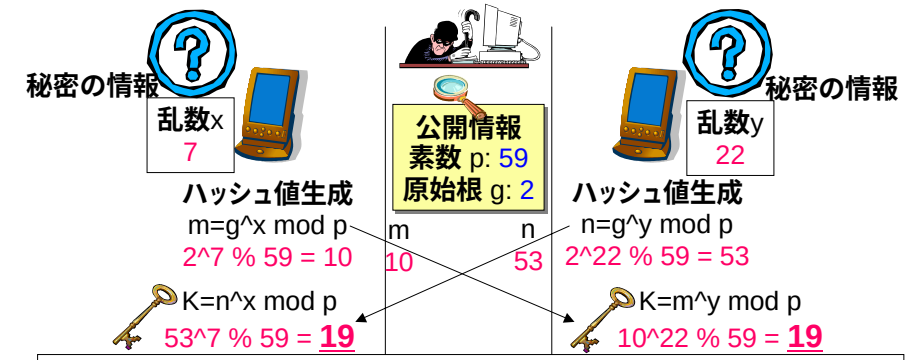
インターネット鍵交換の動作

IKEの2つのフェーズ



※Pre-shared key認証方式を使ったメイン・モード

Diffie-Hellman法による鍵交換アルゴリズム



素数がpが大きくなると p,g,m,n からx,yの計算は数学的に困難

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08
8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B
302B0A6D F25F1437 4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9
A63A3620 FFFFFFFF FFFFFFFF
```

グループ1
768ビット

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08
8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B
302B0A6D F25F1437 4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9
A637ED6B 0BFF5CB6 F406B7ED EE386BFB 5A899FA5 AE9F2411 7C4B1FE6
49286651 ECE65381 FFFFFFFF FFFFFFFF
```

グループ2
1024ビット

IKEで利用される素数
原始根 g = 2

IPv6アプリケーションのプログラミング

プログラミング解説の目的

NTT Information Sharing Platform Laboratories

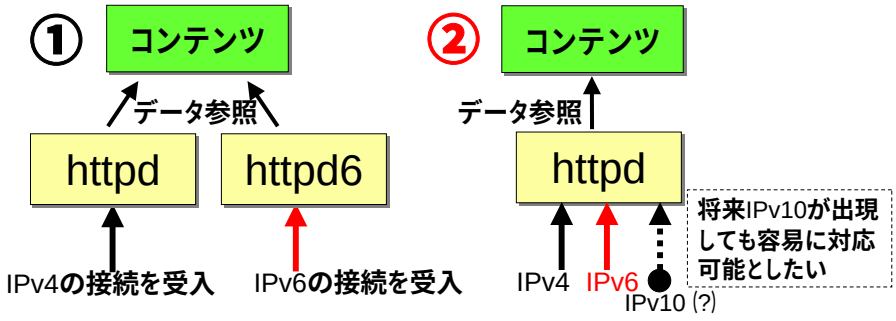
- IPv6アプリケーションを開発するための基本的なパラダイムを理解する
  - IPv6対応アプリケーションの動作を知る
  - プログラミング未経験でも有意な内容とした
  - プログラミングAPIの克明な利用方法ではない
- 既存のIPv4アプリケーションをIPv6対応に書き直すアプローチをとり、比較を行う
- アプリケーションはWebを想定
  - ブラウザ (TCPクライアント)
  - Webサーバ (TCPサーバ)

99

IPv6対応アプリケーションとは？

NTT Information Sharing Platform Laboratories

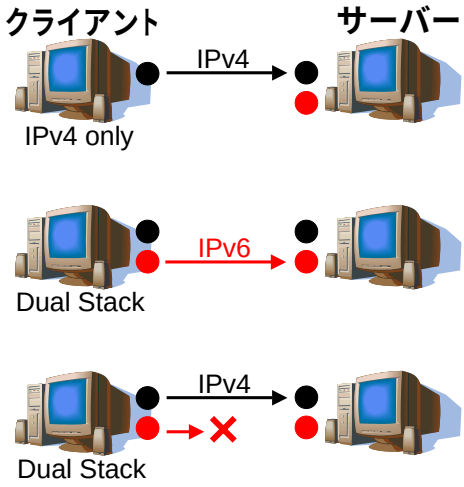
- IPv6専用のアプリケーションではなくIPv4, IPv6いずれでも接続できるものが一般的
  1. IPv6専用のアプリケーションと、IPv4専用のアプリケーションを同時に動作 (並存) させる
  2. プロトコルに非依存なアプリケーション
- IPv6対応のWebサーバの動作例



100

プロトコル非依存のアプリケーションとは？

NTT Information Sharing Platform Laboratories



- クライアントの組み合わせ
  - IPv6 only
  - IPv4 only
  - dual stack
- サーバの組み合わせ
  - IPv6 only
  - IPv4 only
  - dual stack
- DNSの返答内容
- ネットワークの状態

⇒条件分岐が複雑だが、組み合わせによらず同一のコードでプログラムを生成する

:
:多数の接続パタンが存在

101

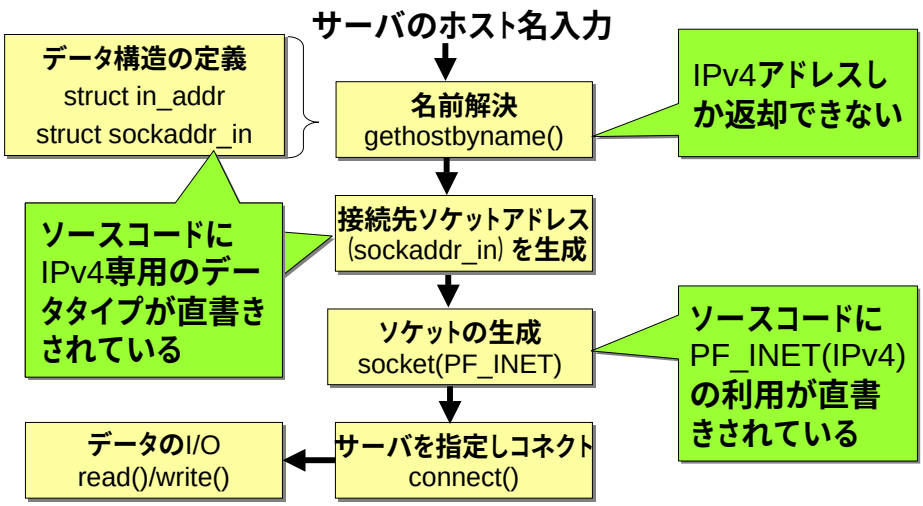
ネットワークアプリケーションのAPI
BSD（バークレー）ソケット

NTT Information Sharing Platform Laboratories

- 概要
  - 4.2BSDへのTCP/IP搭載時に規定
  - WinSockはBSDソケットから派生したもの
- 特徴
  - 仮想的な通信の端点（ソケット）を提供、そのソケットを指定するだけで通信可
    - 個別のプロトコル（TCPのSYNやACKなど）の通信手順を隠蔽
  - 通信内容が「ソケットアドレス」により表現される
    - アドレスやポート番号などを詰め込む構造体を定義
    - `sockaddr_in`（IPv4）, `sockaddr_in6`（IPv6）, … など
  - マルチプロトコルでの利用を前提とした設計
    - IPX , AppleTalk, OSI などIP以外のプロトコルでも利用される

102

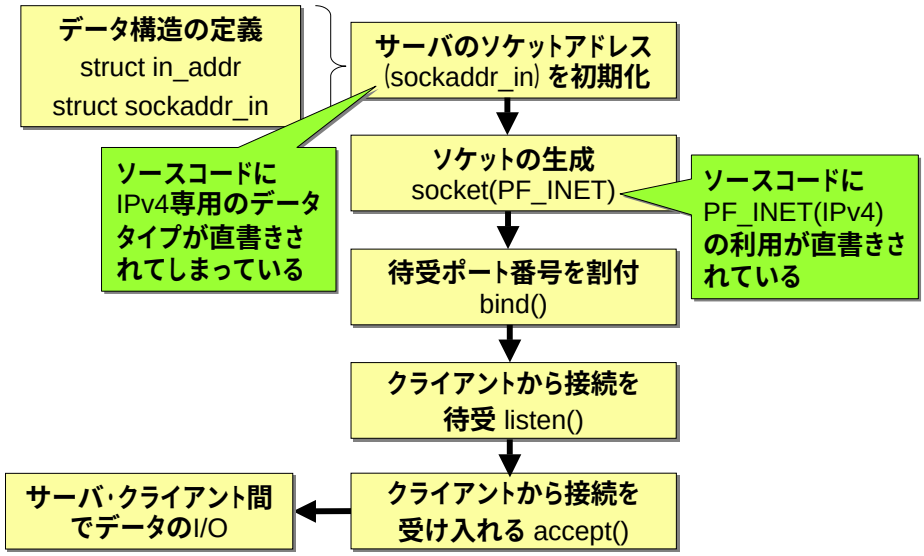
BSDソケットによるIPv4 TCPクライアントの処理フロー



従来のIPv4アプリケーション
サンプルコード(TCPクライアント編)


```
/*
 * サーバに接続し受信した文字列を表示するクライアント
 * IPv4バージョン
 */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

int main(int argc, char *argv[])
{
    char *hostname;          /* サーバのホスト名 */
    char *servname;          /* ポート名 */
    struct hostent *hp;      /* IPv4アドレスの解決結果 */
    struct servent *sp;      /* ポート名の解決結果(ポート番号) */
    unsigned short port;     /* ポート番号を格納 */
    struct sockaddr_in sin4; /* ソケットアドレス */
    int s;

    /* コマンドライン引数のチェック */
    if (argc != 3) {
        fprintf(stderr, "usage: v6-tcp-client <hostname>
  <port>¥n");
        exit(1);
    }

    /* コマンドラインの第一引数(ホスト名)と第二引数(ポート名)をセット */
    hostname = argv[1];
    servname = argv[2];

    /* ホスト名からIPv4アドレスを解決 */
    if ((hp = gethostbyname(hostname)) == NULL) {
        fprintf(stderr, "cannot resolv hostname: %s¥n",
  hostname);
        exit(1);
    }

    /* ポート名からポート番号を解決 */
    if ((sp = getservbyname(servname, "tcp")) == NULL) {
        port = atoi(servname);
        if (port == 0) {
            fprintf(stderr, "cannot resolv servname: %s¥n",
  servname);
            exit(1);
        }
    /* ネットワークバイトオーダーへ変換 */
        port = htons(port);
    } else
        port = sp->s_port;

    /* sockaddr_in 構造体を初期化 */
    memset(&sin4, 0, sizeof(struct sockaddr_in));
    /* アドレスファミリとしてIPv4を指定 */
    sin4.sin_family = AF_INET;
    /* ソケットアドレスの長さを設定 BSD系UNIXではコメントを外す */
/*  sin4.sin_len = sizeof(struct sockaddr_in); */
    /* IPv4アドレスを設定 */
    memcpy(&sin4.sin_addr, hp->h_addr, sizeof(struct in_addr));
    /* ポート番号を設定 */
    sin4.sin_port = port;

    /* IPv4ソケットを生成する */
    s = socket(AF_INET, SOCK_STREAM, 0);
    if (s < 0) {
        perror("socket");
        exit(1);
    }
    /* sin4で指定したサーバのポートへ接続 */
    if (connect(s, (struct sockaddr *)&sin4, sizeof(struct
  sockaddr_in)) < 0) {
        perror("connect");
        exit(1);
    }

    /* サーバとのデータのI/O */
    while (1) {
        /* サーバからデータを受け取り表示 */
        int n;
        char buf[4096];
        n = read(s, buf, sizeof(buf));
        if (n == 0)
            exit(0);
        fputs(buf, stdout);
    }
}
```

BSDソケットによるIPv4 TCPサーバの処理フロー



従来のIPv4アプリケーション
サンプルコード(TCPサーバ編) (1／2)


```
/*
 * 接続してきたクライアントに対して文字列を送信するサーバ
 * IPv4バージョン
 */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    char *servname;          /* ポート名 */
    struct servent *sp;      /* ポート名の解決結果(ポート番号) */
    unsigned short port;     /* ポート番号を格納 */
    struct sockaddr_in sin4; /* ソケットアドレス */
    int s;                   /* ソケット */
    fd_set rfd, rfd0;

    /* コマンドライン引数のチェック */
    if (argc != 2) {
        fprintf(stderr, "usage: v6-tcp-server <port>¥n");
        exit(1);
    }

    /* コマンドラインの第一引数(ポート名)をセット */
    servname = argv[1];

    /* ポート名からポート番号を解決 */
    if ((sp = getservbyname(servname, "tcp")) == NULL) {
        port = atoi(servname);
        if (port == 0) {
            fprintf(stderr, "cannot resolv servname: %s¥n",
  servname);
            exit(1);
        }
        /* ネットワークバイトオーダーへ変換 */
        port = htons(port);
    } else
        port = sp->s_port;

    /* sockaddr_in 構造体を初期化 */
    memset(&sin4, 0, sizeof(struct sockaddr_in));
    /* アドレスファミリとしてIPv4を指定 */
    sin4.sin_family = AF_INET;
    /* ソケットアドレスの長さを設定 BSD系UNIXではコメントを外す */
    /*  sin4.sin_len = sizeof(struct sockaddr_in); */
    /* IPv4アドレスを設定(バイトオーダに注意) */
    sin4.sin_addr.s_addr = htonl(INADDR_ANY);/* ポート番号を設
*/
    sin4.sin_port = port;

    /* IPv4ソケットを生成する */
    s = socket(AF_INET, SOCK_STREAM, 0);
    if (s < 0) {
        perror("socket");
        exit(1);
    }

    /* ポート番号をソケットに結びつける */
    if (bind(s, (struct sockaddr *)&sin4, sizeof(struct
      sockaddr_in)) < 0) {
        perror("bind");
        close(s);
        exit(1);
    }

    /* クライアントからの接続の要求へ待機する */
    if (listen(s, 5) < 0) {
        perror("listen");
        close(s);
        exit(1);
    }

    /* ソケットsへのクライアントからの接続要求をselectにより監視 */
    FD_ZERO(&rfd0);
    FD_SET(s, &rfd0);

    /* サーバとのデータのI/O */
    while (1) {
        int as;
        struct sockaddr_in csin4; /* クライアントのソケットアドレス */
        int len;                  /* ソケットアドレスの長さ */
```

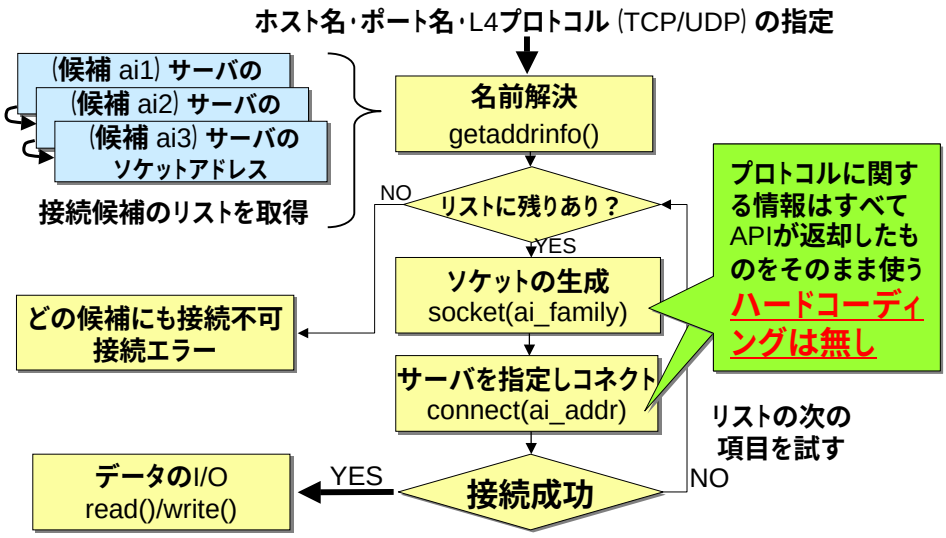
従来のIPv4アプリケーション
サンプルコード(TCPサーバ編) (2／2)


```
        int n;
        int pid;

        /* クライアントから接続が来るまでselectで待つ */
        rfd = rfd0;
        n = select(FD_SETSIZE, &rfd, NULL, NULL, NULL);
        if (n < 0) {
            printf("n = %d¥n", n);
            perror("select");
            close(s);
            exit(1);
        }

        /* クライアントからの接続を受け入れる */
        /* クライアントを示すソケットアドレスがcsin4に書き込まれる */
        as = accept(s, (struct sockaddr *)&csin4, &len);
        if (as < 0) {
            perror("accept");
            close(s);
            exit(1);
        }

        /* サブプロセスを生成し、クライアントとのI/Oを処理させる */
        if ((pid = fork()) < 0) {
            perror("fork");
            exit(1);
        }

        if (pid == 0) {
            /* サブプロセスの処理 */
            char *text = "send string: ¥"server to client¥"¥n";
            write(as, text, strlen(text));
            close(as);
            exit(0);
        } else
            close(as);
    }
}
```


IPv4アプリケーションをIPv6対応にするために

- 基本的な方針
  - IPv4のコードに(やっつけの) IPv6の機能を付け加えない
  - IPv4/IPv6だけに限らずマルチプロトコルが扱えるアプリケーションに書き換える
- プロトコル非依存なプログラミングの推奨
  - APIの規定とその利用方法の解説書
    - RFC3493 “Basic Socket Interface Extensions for IPv6”
    - RFC3542, “Advanced Sockets Application Program Interface (API) for IPv6
    - 「IPv6ネットワークプログラミング」荻野. 2003. アスキー
  - ポイントは新たなリゾルバの導入
- IPv4のプログラムも非常に簡潔なコードになります

プロトコル非依存TCPクライアントの処理フロー



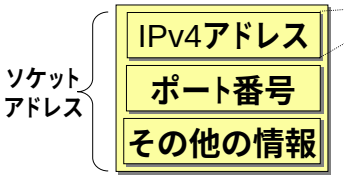
新しい名前解決API getaddrinfo(3) 関数

- 入力
  - ホスト名
  - ポート番号
  - L3プロトコル (IPv4, IPv6 or 問わない)
  - L4プロトコル (TCP or UDP)
- 出力
  - ソケットアドレスと接続に必要な付加情報のリスト
  - 接続の候補となりうるアドレスがリストとして列挙される
- 概念的には

  getaddrinfo()　gethostbyname() /* ホスト名解決 */
  + getservbyname() /* ポート名解決 */
  + ソケットアドレスの初期化
  + バイトオーダの隠蔽
- 標準化されているためほとんどのOSで利用可能

getaddrinfo(3)が生成する情報

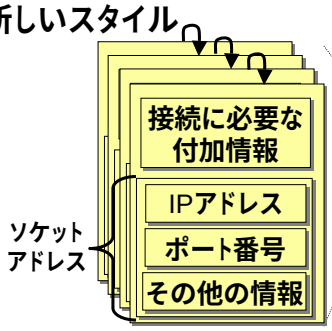
従来のスタイル



gethostbyname()

APIが生成する情報はIPv4アドレスだけで残りの部分はプログラマが組み立てる

新しいスタイル



getaddrinfo()

IPv4/IPv6どちらのアドレスファミリに対してもAPIがソケットアドレス全体を組み立てる。接続に必要な付加情報も提供

異なるプロトコルのソケットアドレスを効果的扱う addrinfo 構造体 (1)

- **このデータ構造の役割**
  1. getaddrinfo()の振舞いを決める「ヒント」を格納する
  2. ホスト名解決の結果を格納する

addrinfoのデータ構造

ai_flags
ai_family
ai_socktype
ai_protocol
ai_addrlen
ai_canonical (ポインタ)
ai_addr (ポインタ)
ai_next (ポインタ)

```
AI_PASSIVE     /* サーバ用のソケットアドレスを生成 */
AI_CANONNAME   /* 別名を格納する */
AI_NUMERICHOST /* ホスト名は生アドレスで与えられる */

PF_UNSPEC      /* IPv6, IPv4 両方を対象とする */
PF_INET        /* IPv4のみ対象とする */
PF_INET6       /* IPv6のみ対象とする */

SOCK_STREAM    /* socketの第2引数 */
SOCK_DGRAM     /* TCP or UDP or any other */
```

```
/* wwwのポートhttp(80)にTCPで接続したい */
/* IPv4, IPv6は問わない */
memset(&hints, 0, sizeof(hints));
hints.ai_family = PF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
/* 名前解決 */
g = getaddrinfo("www", "http", &hints, &ai0);
```

ヒント情報生成

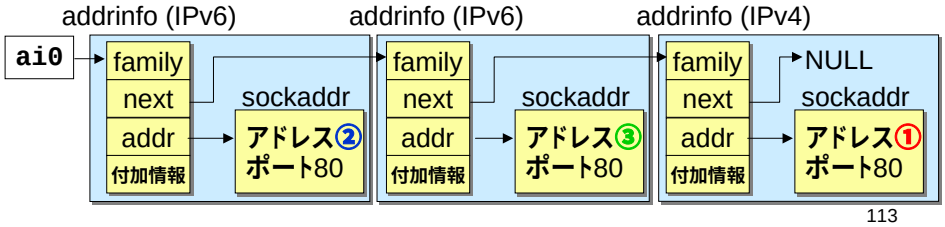
異なるプロトコルのソケットアドレスを効果的扱う addrinfo 構造体（2）－結果の格納方法－

NTT Information Sharing Platform Laboratories

- Webサーバには下記のアドレスが付与されている

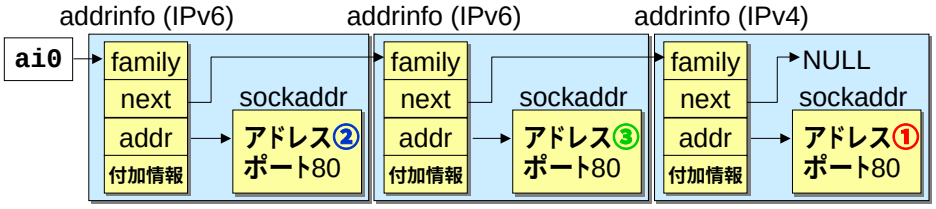
```
---------------------------------------------------------------------------
www.example.com IN A    192.16.178.80 ･･･① IPv4
                IN AAAA 2001:db8::80 ･･･② IPv6
                IN AAAA 2001:db8:aaaa:80 ･･･③ IPv6
---------------------------------------------------------------------------
```

- すべてのアドレスを含んだリスト（ai0）を返す。明示的にアドレスの種類を指定できる
- リストは使い終わったら freeaddrinfo(3)により開放する



プロトコルに依存しないTCPクライアント

NTT Information Sharing Platform Laboratories



- ai0からnextを順にたどり、成功するまで試行を続ける
- すべてのaddrinfoが失敗したら接続失敗
- IPv6の接続が失敗しても、IPv4に自動的にフォールバックする

```
for (ai = ai0; ai; ai = ai->ai_next) {
  s = socket(ai->ai_family,
             ai->ai_socktype, ai->ai_protocol);
  conenct(s, ai->ai_addr, ai->ai_addrlen); /* 接続試行 */
  if (接続成功)
    break; /* ループ脱出：接続試行はこれで終了 */
}
/* ソースコードに特定プロトコルの情報をハードコーディングする必要がない */
```

プロトコル非依存に書き換えたアプリケーション
サンプルコード(TCPクライアント編)


```
/*
 * サーバに接続し受信した文字列を表示するクライアント
 * IPv6/IPv4デュアルバージョン
 */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int main(int argc, char *argv[])
{
    int s;                  /* クライアントと通信を行うソケット */
    struct addrinfo hints; /* getaddrinfoの動作を指定するヒント情報
      */
    struct addrinfo *ai0;  /* addrinfoリストの先頭の要素*/
    struct addrinfo *ai;   /* 処理中のaddrinfoリストの要素 */
    int g;                  /* 名前解決の結果 */
    char *nodename, *servname;

    /* コマンドライン引数のチェック */
    if (argc != 3) {
         fprintf(stderr, "usage: dual-tcp-client <hostname>
      <port>¥n");
         exit(1);
    }

    /* コマンドラインの第一引数(ホスト名)と第二引数(ポート名)をセット */
    nodename = argv[1];
    servname = argv[2];

    /* ヒント情報の初期化 */
    memset(&hints, 0, sizeof(hints));
    /* プロトコルは指定しない: IPv4 or IPv6 */
    hints.ai_family = PF_UNSPEC;
    /* ストリーム型(TCP)による通信を指定 */
    hints.ai_socktype = SOCK_STREAM;

    /* サーバ名とポート名の解決を行い、addrinfoリスト(ai0)を取得 */
    g = getaddrinfo(nodename, servname, &hints, &ai0);
    if (g) {
         fprintf(stderr, "%s", gai_strerror(g));
         exit(1);
    }

    /* addrinfoリストの要素を先頭から接続できるまで順に試行する */
    for (ai = ai0; ai; ai = ai->ai_next) {
        /* ソケットの生成 */
        s = socket(ai->ai_family, ai->ai_socktype, ai-
      >ai_protocol);

        /* ソケットの生成に失敗したらリストの次の項目を試す */
        if (s < 0)
            continue;

        /* ソケットが生成できたらサーバへ接続を試みる */
        /* 失敗したらリストの次の項目を試す */
        if (connect(s, ai->ai_addr, ai->ai_addrlen) < 0) {
            close(s);
            s = -1;
            continue;
        }

        /* ここではすでに接続に成功しているので、forループを脱出 */
        break;
    }

    /* リストのすべての項目が失敗していないかチェック */
    if (s < 0) {
        fprintf(stderr, "cannot connect %s¥n", nodename);
        exit(1);
    }

    /* サーバとのデータのI/O */
    while (1) {
        /* サーバからデータを受け取り表示 */
        int n;
        char buf[4096];

        n = read(s, buf, sizeof(buf));
        if (n == 0)
            exit(0);
        fputs(buf, stdout);
    }
}
```


IPv6対応サーバの基本的な考え方

- **サーバはIPv4, IPv6いずれでも、クライアントからの接続も受け入れる**
  - IPv4, IPv6両方の接続を待ち受ける必要がある
  - = **複数のソケットを開いて待ち受ける**

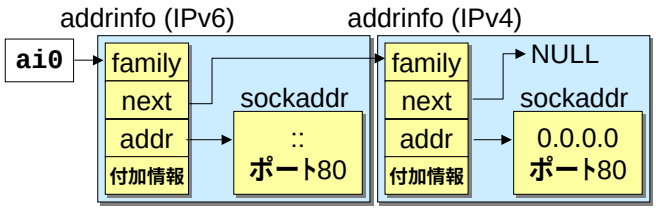
[比較] クライアントが通信に使うソケットは
1つだけ

接続が成功するまでaddrinfoリストを順に試行するが、成功した時点で以降の試行は打ち切り

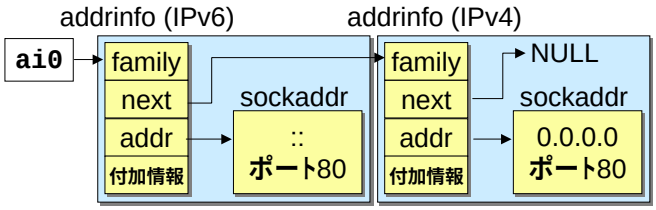
AI_PASSIVEフラグつきのgetaddrinfo(3)

- ヒント情報にAI_PASSIVEを設定すると
  - OSのプロトコル数に応じて、addrinfoのリストを生成
  - 通常、DualStackホストなら [IPv6] → [IPv4] の2項目

```
memset(&hints, 0, sizeof(hints));
hints.ai_family = PF_UNSPEC;
hints.ai_flags = AI_PASSIVE; ←サーバ用ソケットアドレス
hints.ai_socktype = SOCK_STREAM;

g = getaddrinfo(NULL, “http”, &hints, &ai0);
```



TCPサーバの基本ロジック



すべてのプロトコルについてソケットを生成し待受ける (listen)

```
int s[MAXSOCKS];
for (smax = 0, ai = ai0; ai; ai = ai->ai_next) {
    s[smax] = socket(ai->ai_family, ai->ai_socktype,
                                    ai->ai_protocol);
    if (bind(s[smax], ai->addr, ai->addrlen) < 0)
        continue; /* bindに失敗したら次のプロトコルを試す */
    if (listen(s[smax], 5) < 0)
        continue; /* listenに失敗したら次のプロトコルを試す */
    smax++;
} /* addrinfoのリストはすべてなめつくす */
```

クライアントからの接続受け入れ処理

NTT Information Sharing Platform Laboratories

- IPv4, IPv6どちらの接続が来るか不明
  - IPv4, IPv6　2つのソケットを開いて待受けている時、どちらかに接続が来たら受入れ処理を開始
  - 「どちらか」をselectにより判定する
- select()システムコールの役割
  - ファイルディスクリプタの集合を渡し、いずれか一つが読込み可能になるまで待つ
  - listen()により待ち受けているソケットは、クライアントからの接続があるまで待つ
  - 書込・タイムアウト等の条件も設定可

119

クライアントからの接続を受け入れるaccpet()

NTT Information Sharing Platform Laboratories

```
int accept(int s, /* listen()で待つソケット */
           struct sockaddr *saddr, /* OUT */
           int *len                /* OUT */)
    /* OUTはaccept()が情報を渡すという意味 */
```

- 接続要求が来たクライアントとの通信(I/O)用ソケットを返す
- 接続してきたクライアントを示すソケットアドレスを返す
  - saddrにはプロトコル種別に応じたソケットアドレスが返される
    - IPv4の通信の場合 struct sockaddr_in
    - IPv6の通信の場合 struct sockaddr_in6
- プログラマがIPv4, IPv6どちらが返ってきても対応できるように、saddrに十分に大きな領域を確保しておく必要がある

120

select()とaccept()による接続要求処理の多重化

NTT Information Sharing Platform Laboratories

```
struct sockaddr_storage ss;
int sslen = sizeof(ss);

while (1) {
  /* IPv4, IPv6のいずれかの接続が来るまで待つ */
  select(FD_SETSIZE, &rfd, NULL, NULL, NULL);
  s = 接続が来たソケットをrfdから取り出す;

  /* 接続を受け入れる。 ss にはクライアントの情報が書き込まれる */
  as = accept(s,
              (struct sockaddr *)&ss,
              &sslen);

  スレッド or サブプロセス生成(as);
}
```

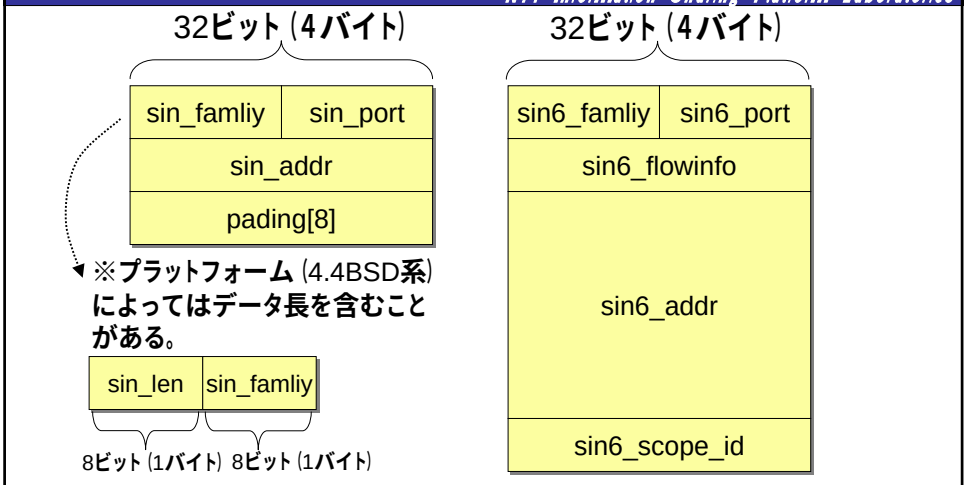
IPv4, IPv6どちらのソケットアドレスでもすっぽり格納できる領域を確保

```
スレッド or サブプロセス(int as){
  ソケットas に対してI/Oを行う
  read(s, ...);
  write(s, ...);
}
```

121

2つの構造体の比較
sockaddr_in (IPv4) , sockaddr_in6 (IPv6)

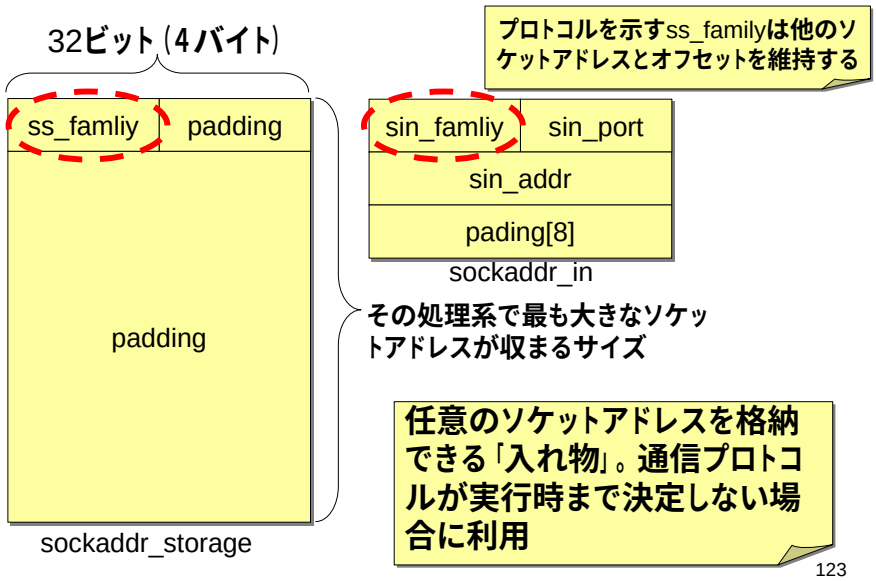
NTT Information Sharing Platform Laboratories



sizeof(struct sockaddr_in) < sizeof(struct sockaddr_in6)

122

任意のプロトコルのソケットアドレスを格納する sockaddr_storage 構造体



プロトコル非依存に書き換えたアプリケーション サンプルコード（TCPサーバ編）（1／2）


```
/*
 * 接続してきたクライアントに対して文字列を送信するサーバ
 * IPv6/IPv4デュアルバージョン
 */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

int main(int argc, char *argv[])
{
    int s[64];              /* クライアントと通信を行うソケット */
    struct addrinfo hints; /* getaddrinfoの動作を指定するヒント情報
     */
    struct addrinfo *ai0;  /* addrinfoリストの先頭の要素*/
    struct addrinfo *ai;   /* 処理中のaddrinfoリストの要素 */
    int g;                 /* getaddrinfoの処理結果 */
    char *servname;        /* ポート名 */
    int smax;              /* 接続を受け入れるプロトコルの数 */
    fd_set rfd, rfd0;
    int i;

    /** コマンドライン引数のチェック */
    if (argc != 2) {
        fprintf(stderr, "usage: dual-tcp-server <port>¥n");
        exit(1);
    }

    /** コマンドラインの第一引数（ポート名）をセット */
    servname = argv[1];

    /** ヒント情報の初期化 */
    memset(&hints, 0, sizeof(hints));
    /* プロトコルは指定しない: IPv4 or IPv6 */
    hints.ai_family = PF_UNSPEC;
    /* ストリーム型(TCP)による通信を指定 */
    hints.ai_socktype = SOCK_STREAM;
    /* PASSIVE型のソケット生成を指定 */
    hints.ai_flags = AI_PASSIVE;
```

```
    /* 名前とポート名の解決 & 接続候補のリストを取得 */
    g = getaddrinfo(NULL, servname, &hints, &ai0);
    if (g) {
        fprintf(stderr, "%s", gai_strerror(g));
        exit(1);
    }

    smax = 0;
    /** addrinfoリストの要素を先頭から接続できるまで順に試行する */
    for (ai = ai0; ai; ai = ai->ai_next) {
        /* ソケットの生成 */
        s[smax] = socket(ai->ai_family, ai->ai_socktype, ai-
      >ai_protocol);

        /* ソケットの生成に失敗したらリストの次の項目を試す */
        if (s[smax] < 0)
            continue;
        /* ポート番号をソケットに結びつける */
        if (bind(s[smax], ai->ai_addr, ai->ai_addrlen) < 0) {
            perror("bind");
            close(s[smax]);
            continue;
        }

        /* クライアントからの接続の要求へ待機する */
        if (listen(s[smax], 5) < 0) {
            perror("listen");
            close(s[smax]);
            continue;
        }

        smax++;
    }

    /** forループで待受けソケットを生成できない場合はエラーとする */
    if (smax == 0) {
        fprintf(stderr, "cannot create server socket¥n");
        exit(1);
    }

    /** s[xx]へのクライアントからの接続要求をselectにより監視 */
    FD_ZERO(&rfd0);
```

プロトコル非依存に書き換えたアプリケーション
サンプルコード（TCPサーバ編）（2／2）


```
for (i = 0; i < smax; ++i)
    FD_SET(s[i], &rfd0);

    /** サーバとのデータのI/O */
    while (1) {
        int as;
        struct sockaddr_storage ss; /* クライアントのソケットアドレス
 */
        int sslen;                  /* ソケットアドレスの長さ */
        int n;
        int pid;

        /* クライアントが接続してくるまでselect()で待つ */
        rfd = rfd0;
        n = select(smax + 1, &rfd, NULL, NULL, NULL);
        if (n < 0) {
            perror("select");
            exit(1);
        }

        /* 接続を受け付けたソケットを求める */
        for (i = 0; i < smax; ++i) {
            if (FD_ISSET(s[i], &rfd))
                break;
        }

        /* クライアントからの接続を受け入れる */
        /* クライアントを示すソケットアドレスがssに書き込まれる */
        as = accept(s[i], (struct sockaddr *)&ss, &sslen);
        if (as < 0) {
            perror("accept");
            exit(1);
        }

        /* サブプロセスを生成し，クライアントとのI/Oを処理させる */
        if ((pid = fork()) < 0) {
            perror("fork");
            exit(1);
        }

        if (pid == 0) {
            /* サブプロセスの処理 */
            char *text = "send string: ¥"server to
client¥"¥n";
            write(as, text, strlen(text));
            close(as);
            exit(0);
        } else
            close(as);
    }
}
```


プロトコルごとに処理を分ける例

- トランスポート層の中にIPアドレスに関する情報が埋め込まれているアプリケーション
  - FTP, SIP, … など

IPv4でのFTP（send-port）

```
USER ftp
PASS xxxx@nttv6.net
PORT 192.168.0.21,14,248
RETR index.txt
```

IPv6でのFTP（send-port）

```
USER ftp
PASS xxxx@nttv6.net
EPRT |2|2001:db8:aaaa::21|3832|
RETR index.txt
```

- ログ生成

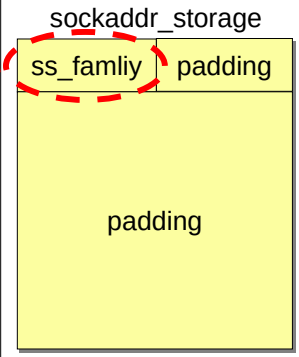
```
192.16.178.192 - - [16/Oct/2004:19:08:39 +0900] "GET /index.html HTTP/1.1" 301
255 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)"
2001:db8:aaaa::1 - - [16/Oct/2004:19:08:50 +0900] "GET /index.html HTTP/1.1"
301 255 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)"
```

プロトコル別の処理が必要になったら

NTT Information Sharing Platform Laboratories

```
struct sockaddr_storage ss;
struct sockaddr_in      *sin4;
struct sockaddr_in6     *sin6;

switch (ss.ss_family) {
case AF_INET:
  sin4 = (struct sockaddr_in *)&ss
   ...
  break;
case AF_INET4
  sin6 = (struct sockaddr_in6 *)&ss;
   ...
  break;
}
```

場合分け



sockaddr_storageのポインタをキャストすれば各プロトコルのソケットアドレスとして参照可能

ホスト名の逆引きと IPアドレスの文字列への変換

NTT Information Sharing Platform Laboratories

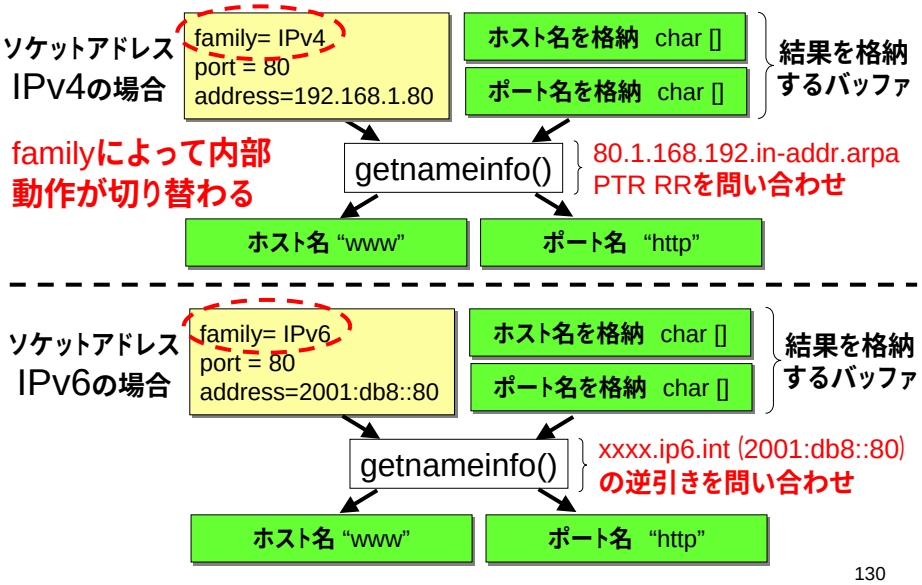
- IPv4では以下の関数が使われていた
  - `inet_ntoa(struct in_addr addr)`
    - バイナリ表現のIPv4アドレスを文字列 (xxx.yyy.zzz.aaa) 形式に変換
    - 入力はIPv4アドレスのみを受け付ける
  - `gethostbyaddr()`
    - IPアドレスからホスト名を逆引きする
    - 引数にプロトコル種別を指定できIPv6にも対応可 ただし機能が不十分のため利用は推奨されない
- プロトコルに依存しないプログラミングのために IPv6/IPv4いずれのアドレスでも、同じ操作で処理が行えるAPIが提供されている

getnameinfo(3)

- 目的
  - プロトコル非依存のプログラム作成のため、プロトコルごとの個別処理を記述させないこと
- 機能
  - IPアドレスからホスト名逆引き
  - IPアドレスから文字列表現 (presentation format) へ変換
  - ポート番号からポート名
  - ポート番号 (binary) から文字列表現へ変換
  - IPv6ではスコープ付きアドレスも正しく扱える
    - ff02::1%eth0
  - 感覚的にはIPv4で利用される複数のAPIの機能が統合されたもの

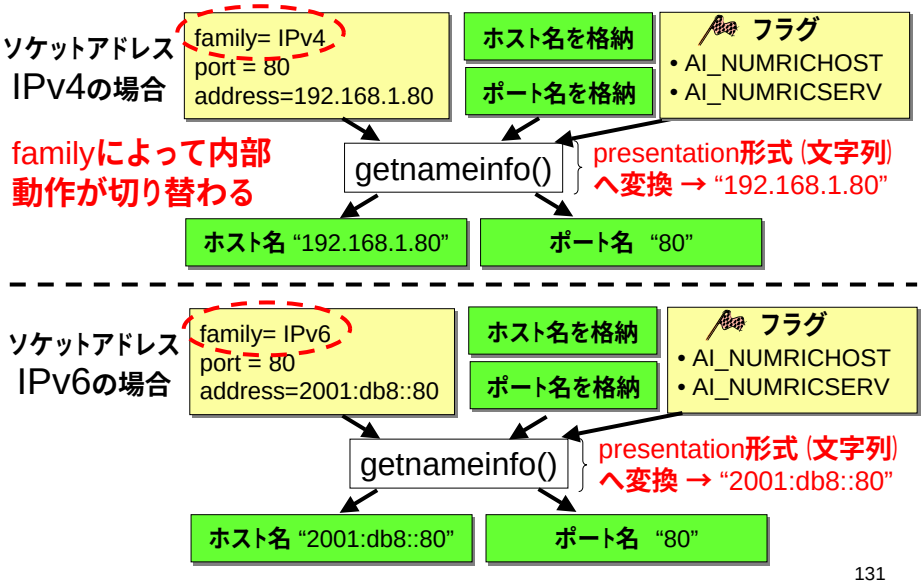
    getnameinfo()　gethostbyaddr()　/* ホスト名逆引き */
    + getservbyport()　/* ポート名逆引き */
    + inet_ntoa()　/* 文字列表現への変換 */
    + バイトオーダの変換

ソケットアドレスからホスト名・ポート名の逆引き



IPアドレス･ポート番号の文字列表現の取得

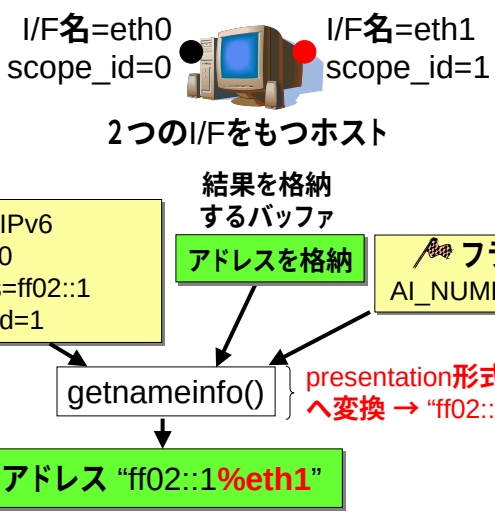
NTT Information Sharing Platform Laboratories



131

スコープを反映したIPv6アドレスの文字列表現

NTT Information Sharing Platform Laboratories



132

IPv6プログラミングのまとめ

NTT Information Sharing Platform Laboratories

- 設計の基本方針
  - プロトコル依存性を排除する
  - 依存性のある部分は極力APIに隠蔽させる
  - プロトコルごとの場合分けは必要最小限に
    - IPv6ではスコープ付きアドレスを扱うことを忘れない
- IPv4のアプリケーションでも新しいAPIは有効
  - 名前解決近辺のコードが非常に簡潔に記述できる
  - 多くの場合、OSがIPv4 onlyのシステム上でも稼動する
  - “PF_INET” を設定すればIPv4の利用を強制できる
  - 利用プロトコルを指定できるようにしおくとよい
    - ssh -6 : IPv6の利用を強制する
    - ssh -4 : IPv4の利用を強制する
    - ssh : IPv6,IPv4両方の接続を試す

    ⇒ IPv4のみの動作としたい場合は “-4” をデフォルトにする

133

チュートリアルのまとめ

NTT Information Sharing Platform Laboratories

- IPv6基本仕様は、本質的にIPv4と同じ
- しかし設計時から洗練された付加機能 (Advanced Feature) が存在
- IPv6をより便利にするために、目的に応じた Advanced Featureの選択を

134

参考文献 (1)

- IPv6の標準化経緯
  - "The Recommendation for the IP Next Generation Protocol", 1995/1, RFC1752.
  - 「WIDEプロジェクト 1995年度研究報告書」, 第6部 IP version 6, http://www.wide.ad.jp/document/reports/pdf1995/index.html
- IPv6基本仕様
  - “Internet Protocol Version 6 (IPv6) Addressing Architecture”, 2003/4, RFC3513.
  - “Internet Protocol, Version 6 (IPv6) Specification”, 1998/12, RFC2460.
  - “Neighbor Discovery for IP Version 6 (IPv6)”, 1998/12, RFC2461.
  - “IPv6 Stateless Address Autoconfiguration”, 1998/12, RFC2462.
  - “Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification”, 1998/12, RFC2463.
  - “IPv6 Node Requirements”, John Loughney , 2004/8, draft-ietf-ipv6-node-requirements-11.txt.

参考文献 (2)

- DHCPv6
  - “Dynamic Host Configuration Protocol for IPv6 (DHCPv6)”, 2003/7, RFC3315.
  - “DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)”, 2003/12, RFC3646.
  - “IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) version 6”, 2003/12, RFC3633.
  - “Requirements for IPv6 Prefix Delegation”, 2004/6, RFC3769.
  - “Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6”, 2004/4, RFC3736.
  - “Simple Network Time Protocol Configuration Option for DHCPv6”, 2003/12, draft-ietf-dhc-dhcpv6-opt-sntp-00.txt

参考文献 (3)

- **アドレス選択方式**
  - "Default Address Selection for Internet Protocol version 6 (IPv6)", 2003/2, RFC3484.
- **匿名アドレス**
  - "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", 2001/1, RFC3041.
- **ネットワーク管理**
  - "Management Information Base for IP Version 6: Textual Conventions and General Group", 1998/12, RFC2465.
  - "Management Information Base for IP Version 6: ICMPv6 Group", 1998/12, RFC2466.
  - "IP Version 6 Management Information Base for the Transmission Control Protocol", 1998/12, RFC2452
  - "IP Version 6 Management Information Base for the User Datagram Protocol", 1998/12, RFC2454
  - "Management Information Base for the Internet Protocol (IP)", 2004/5, draft-ietf-ipv6-rfc2011-update-10.txt
  - **小早川知昭**, **宮本崇之**, **齊藤允**, 「IPv6**実践ガイド マルチ**OS**で学ぶ**v4/v6**デュアルスタックネットワークの構築・運用方法**」, 2003/10, **翔泳社**

参考文献 (4)

- **ローカルアドレス**
  - "Centrally Assigned Unique Local IPv6 Unicast Addresses", 2004/6, draft-ietf-ipv6-ula-central-00.txt
  - "Unique Local IPv6 Unicast Addresses", 2004/9, draft-ietf-ipv6-unique-local-addr-06.txt
  - "Deprecating Site Local Addresses", 2004/9, RFC3879.
- MobileIPv6 / NEMO
  - "Mobility Support in IPv6", 2004/6, RFC3775.
  - "Using IPsec to Protect Mobile IPv6 Signaling Between Mobile Nodes and Home Agents", 2004/6, RFC3776.
  - "Network Mobility (NEMO) Basic Support Protocol", 2004/6, draft-ietf-nemo-basic-support-03.txt

参考文献 (5)

- IPsec関連
  - IPセキュリティ・アーキテクチャ全般
    - “Security Architecture for the Internet Protocol”, 1998/11, RFC2401
    - “IP Security Document Roadmap”, 1998/11, RFC2411
  - IPsecの認証機能、暗号化機能
    - “IP Authentication Header”, 1998/11, RFC2402
    - “The Use of HMAC-MD5-96 within ESP and AH”, 1998/11, RFC2403
    - “The Use of HMAC-SHA-1-96 within ESP and AH”, 1998/11, RFC2404
    - “The ESP DES-CBC Cipher Algorithm With Explicit IV”, 1998/11, RFC2405
    - “IP Encapsulating Security Payload (ESP)”, 1998/11, RFC2406
    - “The NULL Encryption Algorithm and Its Use With IPsec”, 1998/11, RFC2410
    - “The ESP CBC-Mode Cipher Algorithms”, 1998/11, RFC2451
  - 鍵交換機能
    - “The Internet IP Security Domain of Interpretation for ISAKMP”, 1998/11, RFC2407
    - “Internet Security Association and Key Management Protocol (ISAKMP)”, 1998/11, RFC2408
    - “The Internet Key Exchange (IKE)”, 1998/11, RFC2409
    - “The OAKLEY Key Determination Protocol”, 1998/11, RFC2412
    - “Internet Key Exchange (IKEv2) Protocol”, 2004/09, draft-ietf-ipsec-ikev2-17.txt

参考文献 (6)

- プログラミング関係
  - “Basic Socket Interface Extensions for IPv6”, 2003/2, RFC 3493.
  - “Advanced Sockets Application Program Interface (API) for IPv6”, 2003/5, RFC3542.
  - 「IPv6ネットワークプログラミング」,萩野純一郎 著、小川彩子 訳, 2003年2月, アスキー, http://www.ascii.co.jp/books/detail/4-7561/4-7561-4236-2.html
  - “Unix Network Programming Volume 1., 3rd Edition, The Sockets Networking API”, W. Richard Stevens, Bill Fenner, Andrew M. Rudoff, 2003/11, Prentice Hall.
  - “Winsock 2.0”, Lewis Napper, 1997/11, Hungry Minds Inc.