アプリケーションの IPv6 対応ガイドライン
基礎編
第 1.0 版

2012 年 12 月 3 日

IPv6 普及・高度化推進協議会
IPv4/IPv6 共存 WG
アプリケーションの IPv6 対応検討 SWG

目次

変更履歴

版	改版日	摘要
0.9	2012 年 5 月 1 日	パブリックコメント版
1.0	2012 年 11 月 19 日	第 1.0 版

はじめに

2011 年の IPv4 アドレス在庫枯渇を機に、通信事業者や ISP 各社から IPv6 対応のサービス展開が進み、インターネットのユーザフロント、サーバフロント共にこれまでの IPv4 での運用から IPv6 が混在する IPv4 と IPv6 の共存期に入りつつあります。一方で、スマートフォンに代表されるスマートデバイスが普及し、ネットワークを使うアプリケーションは一般的になっています。こうした状況の中、アプリケーションの開発者が目にする情報は、これまでの IPv4 で運営されているネットワークを前提としたものが多く、共存状況で注意すべきことやするべきことが整理されていませんでした。

IPv6 普及・高度化推進協議会 IPv4/IPv6 共存 WG では、2011 年 9 月に「アプリケーションの IPv6 対応検討 SWG」を発足し、共存期を前提としたアプリケーション開発についての情報整理を行い、アプリケーション開発者に向けた情報発信と情報共有の活動を始めました。

本文書は、この「アプリケーションの IPv6 対応検討 SWG」の活動によるもので、BSD Socket を用いたプログラムの IPv6 対応方法についてまとめた資料です。最近は直接 BSD Socket を用いずにクラスライブラリなどを利用して簡単にネットワークを利用できる環境も多くなっていますが、ネットワークプログラミングの基礎として BSD Socket によってどのような処理やデータのやり取りが行われ、その際 IPv4 と IPv6 では何が違うのかをまとめました。本文書を通じてご確認いただき、アプリケーション開発にお役立ていただければ幸いです。

検討メンバー

会務担当者以外のメンバーは、氏名の 50 音順に従っています。

氏名	所属
荒野 高志（WG 主査）	IT ホールディングス株式会社
廣海 緑里（chair）	株式会社インテック
波田野 裕一（co-chair）	日本 UNIX ユーザ会（jus）
藤崎 智宏（co-chair）	日本電信電話株式会社
新 善文	アラクサラネットワークス株式会社
大平 浩貴	株式会社リコー
佐藤 良	株式会社コナミデジタルエンタテインメント
高田 美紀	株式会社 NTTPC コミュニケーションズ
高宮 紀明	NTT ソフトウェア株式会社
谷口 寛季	富士ソフト株式会社
花山 寛	ネットワンシステムズ株式会社
渡辺 露文	富士ソフト株式会社

1 本文書について

本文書は添付資料とともに BSD ソケットによる IPv6/IPv4 デュアルスタックプログラムの作成を題材にアプリケーションの IPv6 対応の基礎的事項について説明する。

2章では本文書が対象とするソケット API の概要について説明している。
3章ではソケットによって作られているアプリケーションプログラムを IPv6 に対応させるための基本的な方針を示している。
4章では具体的なプログラミングの方法について解説している。なお4章は本文書の添付資料であるアプリケーション IPv6 化の例示プログラムと共に参照できるものとしている。
5章では DNS の名前解決などを中心に、ソケットプログラミングに限らずアプリケーションの IPv6 対応において注意すべき事項をまとめている。

本文書は参考文献[1]およびその著者である萩野純一郎（itojun）氏によるパブリックドメインソフトウェアを基にしており、そのソフトウェアのソースコードを解説している。各プログラムの詳細解説を添付資料「アプリケーション IPv6 化例示プログラム集」に記載した。また、IPv6 対応の具体例として「Asterisk の IPv6 対応について」の資料を添付している。

2 BSD ソケットによるプログラミングの流れ

通信プログラムを実装するために、BSD ソケットやそれに類似したソケット API を用意している処理系は多い。BSD ソケットは BSD 系 UNIX を起源としており、もともとは C 言語の API である。TCP を利用する場合、ソケットとして提供される関数を図 1の順に呼び出すことで、サーバでの接続待ち受けやクライアントからの接続を実現できる。

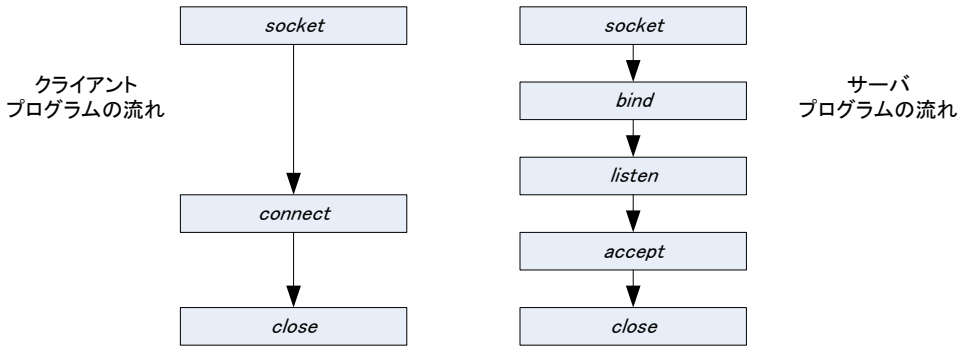

図 1　　ソケット API 呼び出しの基本的な流れ

アプリケーションを IPv6 に対応させる場合でも、上記のソケット API の役割や呼び出し順序は変わらない。しかし、IPv4 のみに対応したプログラムと IPv4/IPv6 に対応したプログラムでは下記の部分が異なる。

- サーバが待ち受ける IP アドレスやクライアントが接続する先の IP アドレスを得る手段
- IP アドレスからホスト名を得る手段
- 文字列表現の IP アドレスを得る手段
- 接続を待ち受ける処理を複数プロトコル・複数アドレスで並行処理する手段

3 IPv6 対応ソケットプログラミング

3.1 基本的方針

特別な環境を除いて一般的に、従来から IPv4 で提供されてきたサービスは、今後もしばらく IPv4 でサービスし続けることが求められる。つまりアプリケーションを IPv6 に対応させるとしても、IPv4 でも継続して動作しなければならない。このため、既存アプリケーションの IPv6 対応は、IPv4/IPv6 両方、すなわちマルチプロトコルに対応することになる。

クライアントアプリケーションは、名前解決の結果得られたアドレスに接続して通信を行う。IPv6 に対応する場合、名前解決で複数のプロトコルの複数のアドレスが得られることになり、それらそれぞれに対して順に接続を試みて初めてつながったものと通信を行う[1]。

[1] IPv4 のみの環境においても、複数のアドレスが得られる場合も少なからずある。通常それらのアドレスの接続性にはほとんど違いはないことから、従来、複数のアドレスがリスト形式で得られた場合、リスト先頭のアドレスに対してのみ接続を試みるものもあった。

サーバアプリケーションは、従来は単一のプロトコル（IPv4）のみを監視し、そこに到着したアプリケーションデータに応答していた。これが IPv6 対応となった場合、複数のプロトコルを監視し、そのいずれかに到着したアプリケーションデータに応答を返すことになる。

参考文献[1]では、サーバアプリケーションの IPv6 対応手法について表 1の手法が挙げられている。

しかし、IPv6 の対応では、IPv4 アドレスと IPv6 アドレスの接続性に違いが想定されるため、得られたアドレスに対して順に接続を試みて初めてつながったものと通信を行う。

表 1　サーバプログラミングの手法と特徴

手法	利点	欠点
複数のソケットをオープンする	● ひとつのプロセスでマルチプロトコルに対応できる	● 複数ソケットを生成し、それらを *select* で待つため、プログラムが複雑になる
IPv4 対応と IPv6 対応の二つのプロセスを動作させる	● アプリケーションの構成はシングルスタック構成のままでデュアルスタックに対応できる	● 複数プロセスで共有リソースを扱う場合、プロセス間で排他制御する必要がある
inetd から呼び出し可能なサーバにする	● 通信部分を inetd が代行するため、通信の IPv6 化を意識しなくてよい	● inetd を必要とする
IPv4 マップドアドレスを使用する	● ひとつのソケットで IPv4/IPv6 両方を処理でき、プログラミングパラダイムの変更が必要ない	● 場合によっては IPv4 と IPv6 の処理が混在することになり、アクセスコントロールやアドレスを表示する際には IPv4 マップドアドレスかどうかの判定とそれに応じた処理が必要となる場合もある ● 一部 OS で利用できない場合がある

3.2 参考となる RFC

3.2.1 ソケットの定義

現在、BSD ソケットを IPv6 に対応させる API を定義した RFC は 2 系統ある。一方はソケット API IPv6 化のベースとなる文書であり、最新は RFC3493 [2]である。もう一方は詳細かつ高度なソケット API の IPv6 化を定めた文書であり、最新は RFC3542 [3]である。(図 2)

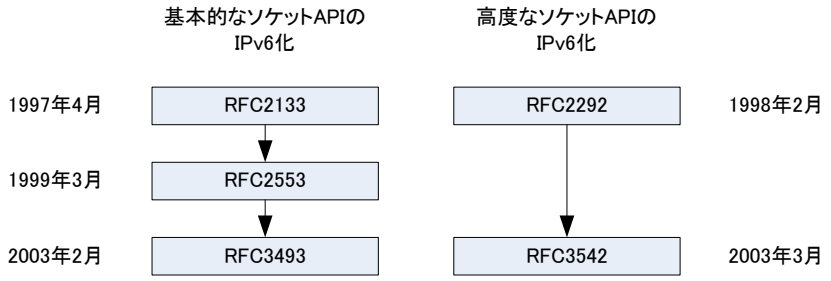

図 2　　RFC 改訂の流れ

前者の RFC はプロトコルファミリの追加、IPv6 アドレス構造体の定義、ソケット関数のシンタックスには変更のないこと、IPv4 マップドアドレスによる IPv4 との互換性、各種アドレス表記、インターフェース ID などの基本的事項について述べている。

後者の RFC は IPv6 ヘッダの詳細、ICMPv6 ヘッダの詳細、Raw ソケット、IPv6 拡張ヘッダ、ソケット付随情報の拡張、パケットの情報（始点アドレスや終点アドレス、ホップリミットなど）経路制御ヘッダオプション、その他のオプション、MTU 関連機能などについて述べられている。

3.2.2 RFC4038 における記述

また、RFC4038 [4]ではアプリケーションプログラムを IPv6 に対応させる際の問題、移行シナリオ、アプリケーションの各レイヤそれぞれのポーティングに関する検討、IP のバージョンに依存しないアプリケーションの作り方について述べている。

4 ソケットプログラミングの実際

本文書の添付資料「アプリケーション IPv6 化 例示プログラム集」に記載されているサンプルに基づいたソケットプログラミングの概要を下記で解説する。

4.1 IPv6 対応クライアントプログラム

従来のクライアントアプリケーションは *gethostbyname* 関数や *getservbyname* 関数を使用して接続先ホストの名前解決を行っていた。IPv6 対応クライアントでは、これらの関数を使用せず、*getaddrinfo* 関数を使い、*addrinfo* 構造体のリストの形で相手先ホストの IP アドレス列を取得する。

また、従来はひとつの IP アドレスに接続してアプリケーションデータの送信と応答を行っていた。IPv4 および IPv6 の両方に対応するためにはこの構造を改め、*addrinfo* 構造体のリストを順に辿りつつ接続を繰り返し、接続に成功したらアプリケーションデータの

送信と応答を行う構造にする。

この様子を添付資料「アプリケーション IPv6 化 例示プログラム集」の「■IPv6 対応クライアントプログラミング」で示している。

4.2 inetd を使用した IPv6 対応サーバプログラム

inetd を使用することにより、サーバプログラムの通信処理は標準入出力に対する読み書きとして実装できる。このため、IPv6 に対応してもプログラムの主たる流れには影響がない。ただし、ロギングなどの用途で IP アドレスを処理する部分には修正が必要となる。具体的には、*getpeername* 関数で接続相手に関する情報を得ることができ、そこで得られた *sockaddr* 構造体を *NI_NUMERICHOST* 引数とともに *getnameinfo* 関数を呼び出して相手ホストの文字表記 IP アドレスを得る手順となる。*getnameinfo* 関数を用いることで、接続相手との通信プロトコルを意識することなく、プロトコルに合った文字列表現の IP アドレスを得ることができる。

この様子を添付資料「アプリケーション IPv6 化 例示プログラム集」の「■ inetd を使用した IPv6 対応サーバプログラミング」で示している。

4.3 複数のソケットを使用した IPv6 対応サーバプログラム

従来のサーバアプリケーションでは *hostent* 構造体および *gethostbyname* 関数や *getservbyname* 関数などが使用されていた。IPv6 対応プログラミングではそれらの関数の使用をやめ、IPv4/IPv6 双方を扱えるように汎化した *addrinfo* 構造体および *getaddrinfo* 関数や *getnameinfo* 関数を使用する。

getaddrinfo 関数を使うことで、自身の使用可能なプロトコルすべてをリストの形で得られる。このプロトコルひとつひとつについてソケットを生成し、*bind*、*listen* する。その結果得られた複数のソケットについて、*select* で待ち合わせる。それらのソケットへの接続が検出されしだい、該当ソケットに *accept* してアプリケーションデータを読み込み、応答を返す。この様子を添付資料「アプリケーション IPv6 化 例示プログラム集」の「■複数のソケットを使用した IPv6 対応サーバプログラミング」で示している。

なお、サンプルプログラム内に *IPV6_V6ONLY* マクロで定義されている部分がある。このマクロが定義されている環境では、*IPV6_V6ONLY* オプションの指定により IPv4 マップドアドレスの使用を抑制するようなコードとなっている。この部分が存在しない場合、「処理系・環境により IPV6_V6ONLY 指定のデフォルト値が異なり、無指定の場合に挙動が異なる」というソースコードのポータビリティが低下するリスクと、「IPv4 マップドア

ドレスが有効となると IPv4 と IPv6 という異なるプロトコルを同一として扱い、IPv4 と IPv6 のセキュリティポリシーを明確に分離しにくい（意図しない通信を透過してしまうなど）」というセキュリティ上のリスクが生じる問題がある（参考文献[5]）。以上から今回のサンプルでは IPv4 マップドアドレスを使用しないようにしている。

4.4 マルチプロセスによる IPv6 対応サーバプログラム

サーバアプリケーションが起動時の引数によって、IPv6 による *listen* を行うか、IPv4 による *listen* を行うかを排他的に選択できるようにし、ひとつのソフトウェアを IPv4 対応プロセスと IPv6 対応プロセスの二つのプロセスとして走行させる。なお、そのプログラムが自ホストのプロトコル情報を得る際には *getaddrinfo* 関数を使用する。4.3節の手法とは違い、プログラムの構造は従来の IPv4 のみに対応したシングルスタックプログラムとあまり変わらず、そのままソケットの生成、*bind*、*listen*、*accept* を行う。ただし、このプロセスが並行して複数走行することになるため、複数のプロセス間でひとつのリソースへのアクセスが競合しないように排他制御する機構が別途必要となる。

この様子を添付資料「アプリケーション IPv6 化例示プログラム集」の「■マルチプロセスによる IPv6 対応サーバプログラミング」で示している。

5 名前解決についての議論

IPv6 アプリケーションを構築するにあたって、名前解決は重要である。デュアルスタック環境でプログラムが正しく動作するには、アプリケーションがデュアルスタックに対応していることだけでなく、名前解決の結果が正しくアプリケーションに与えられている必要である。

5.1 RFC6724 の記述とその実現

IPv6 環境では、自ホストおよび相手ホストが複数の IP アドレスを持つことがある。その場合、自ホストから相手ホストに対して接続する際に、複数ある自分のアドレスのうちどれを始点アドレスとしてパケットに記載し、複数ある相手アドレスのどのアドレスを終点アドレスとするかが問題となる。

このアドレス選択手法が RFC6724 [6]で定義されている。この RFC では、ホストが IP アドレスで接続する際に、始点アドレスと終点アドレスをどのように選択するべきかを記載している。具体的には 2.1 節にポリシーテーブルと呼ばれるアドレス候補順序リストと

そのデフォルトポリシーテーブルが記述されており、9 つのエントリが定義されている。また、始点アドレス選択ルールについては同 RFC の 5 章でルール 1 からルール 8 まで、終点アドレスの選択ルールについては同 RFC の 6 章でルール 1 からルール 10 まで記載されている。

IPv6 対応ソケットプログラミングにおける接続先ホストの名前解決は前述のとおり、主に *getaddrinfo* 関数で行う。*getaddrinfo* で名前解決を行うと、終点アドレスの候補となる IP アドレスがリストの形でユーザに返される。この *getaddrinfo* 関数が返すアドレスリストは RFC6724 に従った順序でソートされる[2]。

また、アプリケーションによる始点アドレスの選択については RFC5014[7]で定義されている。

5.2 フォールバックとその解決

ホスト名を *getaddrinfo* で名前解決すれば IP アドレスがリスト形式で得られるが、この結果は IPv4 と IPv6 のアドレスが混在しており、また特に IPv6 アドレスについては複数のアドレスが得られることも見込まれる。そして、クライアントアプリケーションは得られたアドレスリストの先頭から順に接続を試す。しかし、リストの先頭の IP アドレスが接続可能とは限らない。例えばネットワークの接続やサーバが提供するアプリケーションサービスに不具合があり、その IP アドレスへの到達性がない場合がある。そのような場合、クライアントアプリケーションは接続に失敗したことを検出し、*getaddrinfo* が返したリストに記載されている次の IP アドレスでの接続を試す[3]。本文書ではこれをフォールバックと呼ぶ。

フォールバックが生じると、その分接続成功（アプリケーションサービスの開始）までの時間が長くなり、ユーザの使用感が低下することがある。フォールバックは、ICMP などによる経路不達通知の認識、TCP RST によるセッション切断、TCP SYN に対する応答のタイムアウトなどによって発生し、実際に接続成功までに相当な時間が（場合によっては数十秒以上）かかってしまうことも考えられる。

フォールバックが発生する要因は様々であり、サーバ側の対策だけでは万全ではない。フォールバックをできるだけ発生させないためには、IP の接続性を健全に保つ、アプリケーションサービスを行っていないサーバアドレスを DNS に登録しない、ポリシーテーブルを制御して安定した接続を優先するなどに留意する必要がある。

---

[2] RFC6724 は 2012 年 9 月に RFC 化したため、それ以前の環境では不十分な対応であることが多い。

[3] RFC6724 では IPv6 の接続性にこのような問題が出ることを指して”Broken IPv6”と呼んでいる。

5.3 フォールバック挙動を最適化する Happy Eyeballs

5.2節のフォールバックをクライアントサイドで解決するために 2010 年末頃より IETF で議論されてきたのが Happy Eyeballs という方法であり、2012 年 4 月に策定された RFC6555 [8]に記述されている。

例えば TCP の動作は、通常、接続する側が TCP SYN パケットを送信し、それに他方が TCP SYN,ACK パケットを返す。その後もう一度接続する側から TCP ACK パケットを送って接続完了となる。しかしこの RFC の記述では、名前解決後に IPv6 で TCP SYN パケットを送信するだけでなく、同時に IPv4 でも TCP SYN パケットを送るというものである。そして、TCP SYN,ACK が応答されたプロトコルに対して TCP ACK を送って TCP コネクションを張るという通信方法である。これにより TCP コネクションが張られるまで数十秒かかっていたものが数百ミリ秒で接続できるようになったという実験結果も挙がっている。IPv4 から IPv6 への移行期において、十分な IPv6 接続が望めない環境ではこの Happy Eyeballs のアルゴリズムは有効に動作すると考えられる。

しかしながら、Happy Eyeballs には下記の課題がある。

- Happy Eyeballs は実装依存性が高い。複数の I/F を持っている場合、複数の IPv6 アドレスが名前解決された場合の対応について今後の実践と知見の蓄積が望まれる。
- キャッシュの構成や IPv6 SYN,ACK が遅れて到着する場合などの挙動に応じた動作について検討や実践が望まれる。

以上のような課題があるため、Happy Eyeballs を実装する場合は十分な考慮と動作実験が必要である。Happy Eyeballs については、今後の実装や展開を注視していくべきである。

5.4 組み込み環境でのホスト名利用

組み込みシステムも近年は他の機器と通信する機能を持つことが多い。一方で組み込みシステムではリソースが限られている場合が多く、抽象度の高い言語環境が使えず、本文書で扱っている BSD ソケットを直接扱う可能性が高いと考えられる。組み込みシステムが動作する環境はさまざまであり、DNS が利用できないケースも十分にありえる。このような環境に対応するため、従来、接続先ホストを特定するために内部 ROM に接続先 IPv4 アドレスを直接記憶することもあった。しかし、IPv6 環境においては、IPv6 アドレスを直接 ROM 内に記憶することは避けるべきである。この理由は下記の 2 点による。

第一に IP アドレスはユーザに貸し付けられるものであり、リナンバリングの可能性がある。IP アドレスを組み込み機器の ROM 内に記載すると、接続先アドレスがリナンバリン

グされた場合に対応することができず、許可されていないアドレスを不当に使用することになり、それにより動作不良を引き起こすことになる。このことから IP アドレスの指定は DNS のような外部データベースを使用するべきである。IP アドレスを組み込み機器の内部にハードコーディングした場合の問題と、IP アドレスのハードコーディングを回避する手段について RFC4085 [9]で言及されている。

第二に稼働環境が DNS などのサービスを利用できない場合、前記の IP アドレスの公共性やリナンバリングなどのリスクをやむをえず受容することが考えられる。その場合であっても、IPv6 アドレスを直接使用せず、ホスト名の名前解決を hosts（OS 管理下のホスト名管理データベース）に任せ、プログラム側では *getaddrinfo* などの標準的関数に基づいて IP アドレスを利用できるようにすべきである。これは前述の RFC6724 でアドレス選択順序が定義されており、これらの RFC の仕様が正しく実装された OS 環境を利用することで、適切な名前解決・IP アドレス選択ルーチンを利用できるためである。それを利用しない場合、アドレス選択を組み込み環境のコード自身が行うことになる。これは、アドレス選択を実装するコストや、RFC を逸脱するリスクをメーカー自身が負うことになる。

上記 2 点のコストとリスクが、通常の名前解決や IP アドレス選択方法を使用する場合のリスクを下回った場合にのみ、IP アドレスの公共性に十分に配慮したうえで、IP アドレスを直接扱うものとする。通常は、DNS による名前解決やそれを利用する *getaddrinfo* などの標準関数を採用するべきである。

6 おわりに

本文書は IPv6 対応のソフトウェア・ファームウェアを開発する際の注意点をまとめたものである。ただし、今後も新たな問題や課題・実装提案がされることもあるため、IPv6 ソケットプログラミングを行うソフトウェア・ファームウェア技術者・企画担当者はインターネットの標準化動向を常に注視する必要がある点についてはご留意頂きたい。

ソフトウェア・ファームウェアにとって、IPv6 対応は従来なかった新しいプロトコルに対応することであり、相応にコストがかかる。IPv4 のみに対応していたアプリケーションを IPv6 に対応させるということはシングルプロトコル構造をマルチプロトコル構造に転換することであり、ソフトウェア・ファームウェアの修正・追加量は少なくないものと予測される。しかし、ここで当座の IPv6 対応を先送りにしてしまうと、将来の対応量も期間に比例して増える可能性があるため、早い段階で取り組まれる事をお勧めする。IPv6 の普及が進んだ時点での突貫工事的な対応を考えると、普及が緩やかで局所的な現段階での対応コストは一時的かつそれほど多くないと推測できる。IPv6 普及・高度化推進協議会はその助けになればと考え、またインターネットの健全な発展をめざし、関係諸氏のご協

力を得て本文書を作成した。本文書を今後のアプリケーション開発にお役立ていただければ幸いである。

7 参考文献

[1]
萩野純一郎（itojun）氏著
『IPv6 ネットワークプログラミング』
ASCII 社刊 ISBN4-7561-4236-2 (978-4-7561-4236-8)
2003 年
http://ascii.asciimw.jp/books/books/detail/4-7561-4236-2.shtml

[2]
RFC3493
Basic Socket Interface Extensions for IPv6.
R. Gilligan, S. Thomson, J. Bound, J. McCann, W. Stevens.
February 2003.

[3]
RFC3542
Advanced Sockets Application Program Interface (API) for IPv6.
W. Stevens, M. Thomas, E. Nordmark, T. Jinmei.
May 2003.

[4]
RFC4038
Application Aspects of IPv6 Transition.
M-K. Shin, Ed., Y-G. Hong, J. Hagino, P. Savola, E. M. Castro.
March 2005.

[5]
白畑真氏著
『T2:事例から学ぶ IPv6 トラブルシューティング～サーバ編～』
http://www.nic.ad.jp/ja/materials/iw/2011/proceedings/t2/t2-02.pdf
InternetWeek 2011

JPNIC

[6]
RFC6724
Default Address Selection for Internet Protocol Version 6 (IPv6).
D. Thaler, Ed., R. Draves, A. Matsumoto, T. Chown.
September 2012.

[7]
RFC5014
IPv6 Socket API for Source Address Selection.
E. Nordmark, S. Chakrabarti, J. Laganier.
September 2007.

[8]
RFC6555
Happy Eyeballs: Success with Dual-Stack Hosts.
D. Wing, A. Yourtchenko.
April 2012.

[9]
RFC4085
Embedding Globally-Routable Internet Addresses Considered Harmful.
D. Plonka.
June 2005.