NTTソフトウェア

Security Human Mobile

IPv6 アプリケーションサービスの作り方

IPv6普及・高度化推進協議会
アプリケーションのIPv6対応検討SWG
エヌ・ティ・ティ・ソフトウェア株式会社
高宮紀明

Asteriskは米国Digium社の登録商標または商標です。
そのほかの記載の会社名、製品名は、それぞれの会社の
商標もしくは登録商標です。

IPv6とその必要性

- 1990年代よりインターネットが流行した
  - IPが多数使われるようになった
  - IPを使う端末が増えた
- IPアドレスの枯渇
  - IANA での在庫枯渇（2011年2月）
  - APNIC での在庫枯渇（2011年4月）
  - RIPE NCCでの在庫枯渇（2012年9月）
- 新しいIPアドレスを持つIPにしよう
  - IPv4（従来型）からIPv6（次世代型）へ
  - IPv6 対応製品が増えている今でしょ。

IPv6の対応状況

- iDC/ホスティング、ISP→続々対応中
- 端末OS→PCは対応済み（随時機能向上中）

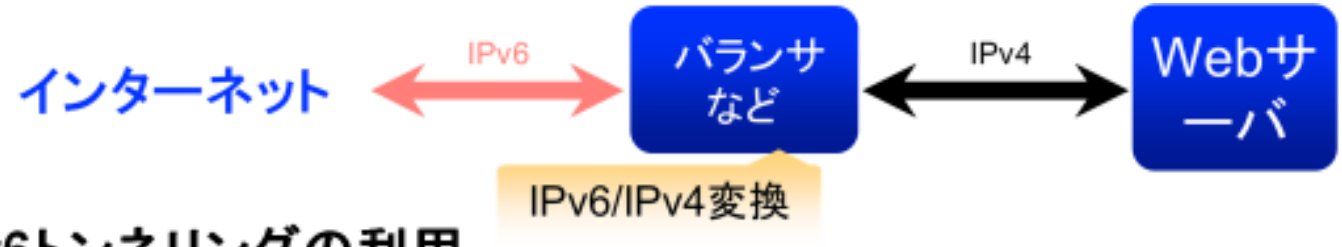
- …じゃぁ、アプリケーションソフトウェアは？
  - ApacheやBINDなど、公共性の高いものは対応済み
- …じゃぁ、私たちが作るアプリは？
  - 私たち自身がこれからがんばらないと！

- アプリケーションソフトウェアがIPv6対応するにはどうしたらいいか

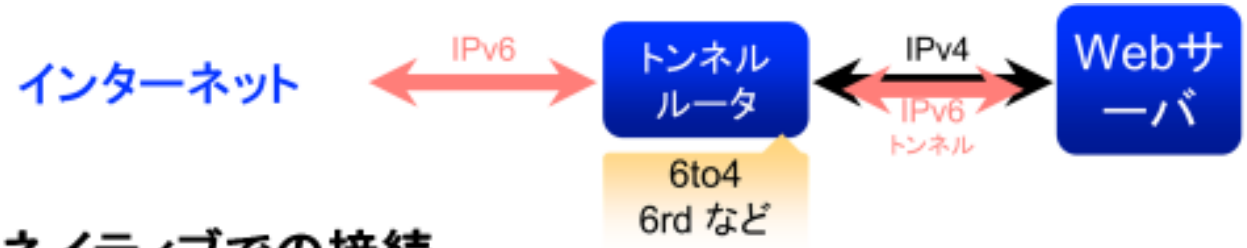
IPv6対応環境について

■ 3つの選択肢

- トランスレータの利用
  - 動向: 一部の企業での導入やパブリッククラウドでの提供が行なわれはじめている。
  - 利用: IPv6対応の先送りには便利だが、過渡期にしか使えない可能性がある。



- IPv6トンネリングの利用
  - 動向: IPv6ネイティブサービスの増加により今後は使われなくなっていく可能性がある。
  - 利用: 検証には使えるが、サービス本番での利用は避けるべき。



- IPv6ネイティブでの接続
  - 動向: 主にパブリッククラウドやVPSサービスでIPv6ネイティブ接続の提供が出揃いつつある。
  - 利用: 利用環境については比較的低コストで容易に手に入るようになったので、早めに利用経験と実績を積み上げていくべき。

IPv6プログラミングの情報について

- 今回の解説で参考にしている書籍
  - IPv6ネットワークプログラミング ASCII社刊
    - http://ascii.asciimw.jp/books/books/detail/4-7561-4236-2.shtml
  - 著者は萩野純一郎（itojun）氏
  - 今回参考にしたプログラムもこの本を参考
    - itojun氏が製作し、パブリックドメインで公開
- IW2012のT7「IPv6実践講座～トラブルシューティング、セキュリティ、アプリ構築まで～」セッション
  - https://www.nic.ad.jp/ja/materials/iw/2012/proceedings/t7/

IPv6普及・高度化推進協議会での活動

- IPv6/IPv4共存WG アプリケーションIPv6化検討SWG
  - http://www.v6pc.jp/jp/wg/coexistenceWG/v6app-swg.phtml

- Socketアプリケーションの IPv6化
  - http://www.v6pc.jp/jp/entry/wg/2012/12/ipv610.phtml
  - 手法本文/サンプルプログラム/ソリューションサンプル
    - http://www.v6pc.jp/jp/upload/pdf/socket-20121203.pdf
    - http://www.v6pc.jp/jp/upload/pdf/socket-sample-20121203.pdf
    - http://www.v6pc.jp/jp/upload/pdf/about_asterisk_ipv6v5-9.pdf
- WebアプリのIPv6化
  - 近日中にパブコメ募集中、ご協力をお願いします。_O_

今回の説明の概要

- BSD Socket APIを使用したアプリケーションソフトウェアのIPv6化を説明

- クライアントプログラムのIPv6対応
  - 具体的な手順
- サーバプログラムのIPv6対応
  - 手法の分類
  - 具体的な手順は割愛（すみません）
- 名前解決の問題と解決案
- 組み込みの話

BSD Socket による クライアントアプリケーションの IPv6化

IPv6プログラミングとは

- IPv4対応プログラム（シングルスタック）
  - ひとつのプロトコルに対応していた

- IPv6/IPv4両対応プログラム（デュアルスタック）
  - 複数のプロトコル・複数アドレスの中のどれでサーバに接続するか選ばなければならない

ただ単に関数を変更するだけではだめ
どのプロトコル・アドレスを使うか選択する機構が必要

シングルスタック・デュアルスタックそれぞれの流れ

シングルスタックの流れ

- ホスト名解決
- サービス名解決
- Socket生成
- Connect実行
- デスクリプタによる入出力
- クローズ

デュアルスタックの流れ

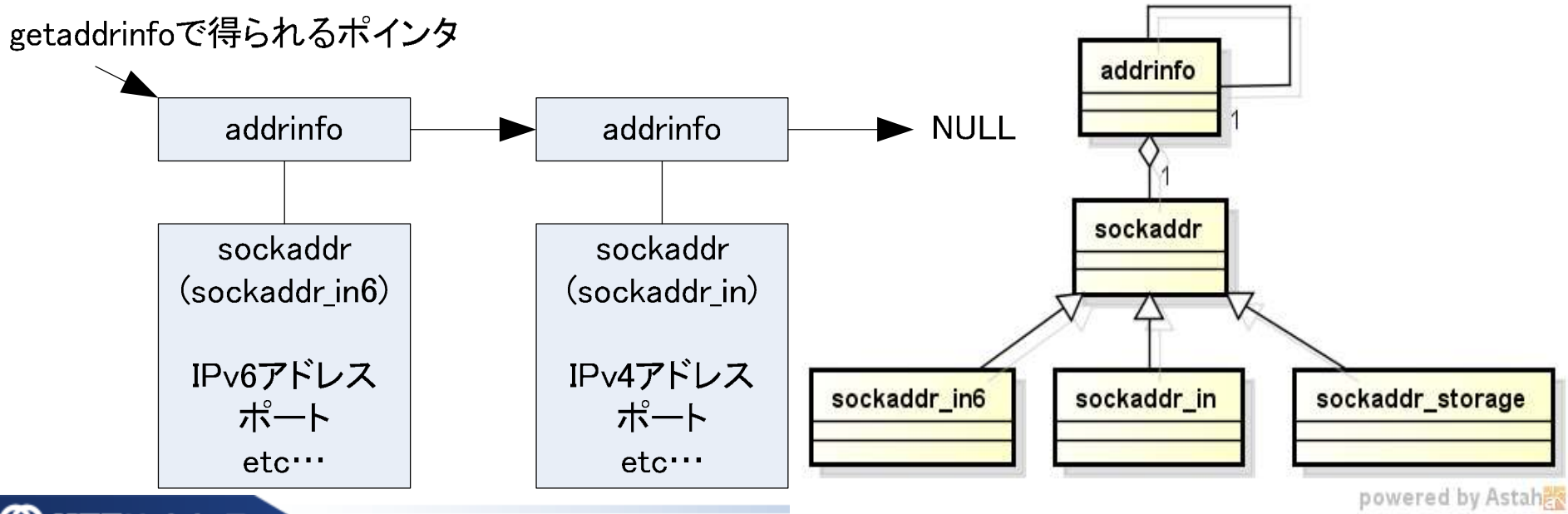
- ホスト名解決
- サービス名解決
- 得られた複数のプロトコル・アドレスでループ
  - Socket生成
  - Connect実行
  - 接続失敗したら
    - 次のプロトコル・アドレスを選択
  - 接続成功したら
    - デスクリプタによる入出力
    - クローズ

ホスト名・サービス名解決

- IPv4
  - ホスト名 : *gethostbyname()*でhostent構造体を得る
  - サービス : *getservbyname()*でservent構造体を得る
- デュアルスタック
  - *getaddrinfo()*を呼ぶとaddrinfo構造体のリストが得られる
    - リストの開放は*freeaddrinfo()* 関数に引数でそのリストを与える
- 注意
  - *gethostbyname2()* はIPv6を扱えるが使うべきではない

addrinfo構造体とsockaddr構造体

- addrinfo**構造体**
  - **1インスタンスで1アドレス情報を持ち、リストを構築している**
  - **内部でアドレスを保持する**sockaddr**構造体へのリンクを持つ**
- sockaddr**構造体**
  - **IPv4やIPv6など各種アドレス情報を汎化した構造体**
  - **実体は**sockaddr_in6（v6）**や**sockaddr_in（v4）
  - **どのアドレスが入るかわからない場合は**sockaddr_storage**で定義**



逆引き

- IPv4
  - アドレス：*gethostbyaddr()*でhostent型構造体を得る
  - サービス：*getservbyport()*でservent構造体を得る
- デュアルスタック
  - *getnameinfo()*にsockaddr構造体を与えるとホスト名やサービス名の文字列を取得できる
    - 文字列領域は呼び出し元が引数で与える
  - いろいろなオプションが指定可能
    - NI_NOFQDN …FQDNではなくホスト名だけ
    - NI_DGRAM …UDPのポート情報を得る
    - NI_NUMERICHOST…逆引きせずアドレスの文字列表現を返す
    - etc...

ソケット生成・コネクト

- デュアルスタックの場合addrinfo構造体を参照する
  - 例：addrinfo ai；
  - プロトコルファミリ：　　ai->ai_family
  - ソケットタイプ：　　ai->ai_socktype
  - プロトコル：　　ai->ai_protocol
  - アドレス：　　ai->ai_addr
  - アドレス長：　　ai->ai_addrlen
- 実例

```
s=socket(ai->ai_family, ai->ai_socktype, ai->ai_protocol);
connect(s, ai->ai_addr, ai->ai_addrlen);
```

クライアントのコード概要

```
struct addrinfo hints, *res, *resall;
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
getaddrinfo(“www.v6pc.jp”,”http”, &hints, &resall);

for (res = resall; res; res = res->ai_next) {
        s=socket(res->ai_family, res->ai_socktype, res->ai_protocol);
        if (s<0) continue;
        if (connect(s, res->ai_addr, res->ai_addrlen) < 0){
                close(s);
                continue;
        }
        /*読み書き*/
        close(s);
        break;
}
```

デュアルスタッククライアントのまとめ

- *getaddrinfo()*で接続先IPアドレスのリストを得る
  - addrinfo構造体のリスト
- リストの順にソケット生成・接続を行い、成功したら通信して終了する

- サンプルプログラム
  - http://www.v6pc.jp/jp/upload/pdf/socket-sample-20121203.pdf

BSD Socketによる サーバアプリケーションのIPv6対応

サーバのデュアルスタック化

- いくつか手法がある
  - inetdを使用する
  - 自身のプロトコル・アドレスの数だけ*socket()*を生成して、全てのFDに対して応答処理をする
  - シングルスタック仕様のプログラムを複数プロセス走行させる
  - IPv4 マップドアドレス（IPv4 Mapped IPv6 address）を使用して、v6ソケットで通信する

サーバアプリ実現手法の分類

手法	利点	欠点
【手法1】 inetdを使用する	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
【手法2】 複数のsocketを生成する	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、それらを同時に待つため、プログラムが複雑になる
【手法3】 シングルスタックプログラムを複数プロセス走行させる	プログラム構成の変更なしにIPv6に対応できる	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
【手法4】 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、プログラム構成の変更が必要ない	IPv4とIPv6の処理が混在する。アドレスを扱う際にはIPv4マップドアドレスかどうかの判定が必要となる場合もある

inetdによるデュアルスタックサーバ

手法1

- 通信部分は変わらない
- 通信相手アドレスを取得する部分で注意が必要
  - *getpeername()* FDとsockaddr構造体を引数で渡すとsockaddr 構造体に相手ピアアドレスを書く
  - 引数は sockaddr構造体ではなく、sockaddr_storage構造体を使用する
    - sockaddr_storage構造体はどんなプロトコルのアドレスでも記憶できる領域を持つ

```
sockaddr_storage from;
getpeername(0,(sockaddr*)&from,sizeof(from));
```

  - あとは*getnameinfo()*で文字列化

複数socketで待ち受けるサーバ

手法2

- もっとも典型的で理想的な対応
- プログラム構成が変化する
  - 複数のデスクリプタを同時待ち受けする機構
- 完全な新規で設計する通信プログラムはこの構成が望ましい

複数ソケットを処理するサーバの流れ

- *getaddrinfo()* で自身のプロトコル・アドレスを*addrinfo*構造体のリストで得る
  - hintsパラメータでAI_PASSIVEを指定するとIN_ADDR_ANYとIN6ADDR_ANY_INITが得られる
- リストで得られたプロトコル・アドレス個別に下記を実施
  - *socket()*、*bind()*、*listen()*
- 得られた複数のFDをfd_set構造体に保存
- 以降はループ
  - fd_set構造体を引数に*select()*で接続の待機
  - *select()*を抜けてきたfdに対して*accept()*
  - 読み書き処理
  - クローズ

複数プロセスで待ち受けるサーバ

手法3

- シングルスタックアプリを複数走行させる
  - *getaddrinfo()* で AF_INETとAF_INET6のどちらかを設定する
- forkでひとつのプログラムがv4/v6に分離するようにすればリソースの節約も可能
  - Copy on Write機能による

自分自身のIPアドレスを得るには？

- 環境依存
  - getifaddrs()
  - UNIXライクOSならioctl関数を利用するのが一般的
- オープンソースOSの場合はifconfigのソースを参照するとよい
  - FreeBSDの場合、/usr/src/sbin/ifconfig/*
  - Linux（CentOS/Debian系）の場合、net-tools/ip-routeパッケージ

デュアルスタックサーバのまとめ

- いくつか手法があるので、メリットとコスト・リスクを比較して適切な選択をしましょう

- サンプルプログラム
  - http://www.v6pc.jp/jp/upload/pdf/socket-sample-20121203.pdf

サーバAsterisk の IPv6 対応について

Asterisk について

- Asterisk とは
  - オープンソースの IP-PBX ソフトウェア
    （IPネットワーク内で、IP電話端末の回線交換を行う装置およびソフトウェア）
  - http://www.asterisk.org/
  - 複数のバージョン（1.4.x、1.8.x、10.x、11.x、12.x）
    - 1.8.x、11.x はLTS（Long Term Support）としてリリース
- Asterisk の IPv6 対応について
  - バージョン 1.8 系より対応（最新版は、11.7（2014/1/31現在））
  - IPv6 対応箇所
    - 呼制御（SIP/IAX）
      SIP は、UDP/TCP/TLS に対応
    - 管理機能（設定用 Web インタフェース、AMI:Asterisk Management Interface）
    - メディアトランスポート（RTP/SRTP）
    - IPv4/IPv6 の相互接続について
      - B2BUA （ゲートウェイ）の接続形式で、IPv4 端末と IPv6 端末との相互接続が可能（この場合、Asterisk が動作している計算機の OS がデュアルスタックで動作していることが前提）

Asterisk の主要機能と IPv6対応

大項目	中項目	概要	IPv6 対応
呼制御	SIP	SIP による呼制御機能	○
	IAX	IAX による呼制御機能	○
	H.323	H.323 による呼制御機能	×
	Websocket	Websocket による呼制御連携	○
メディア処理	RTP	音声/映像ストリーム	○
暗号化	SIPS 対応	SIP over TLS	○
	SRTP	暗号化 RTP	○
管理機能	AMI	Asterisk Management Interface	○
	Web インタフェース	ブラウザからの設定機能	○
PBX 間連携	DUNDi	Asterisk 間の相互接続機能	×

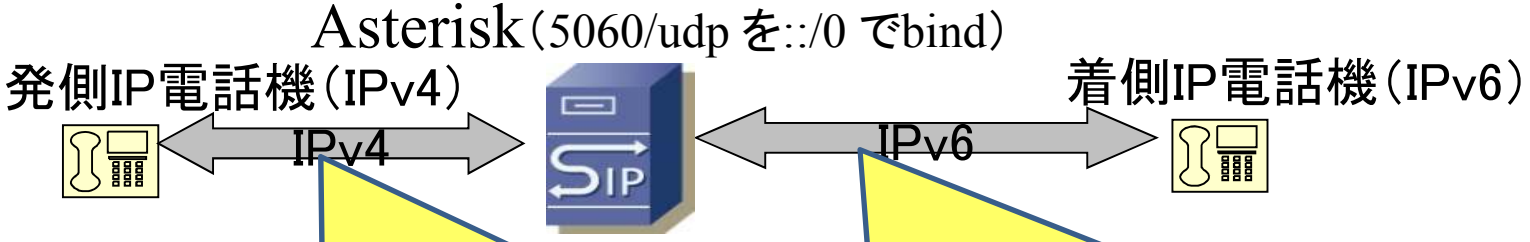
2013/11月 現在（Asterisk 11）

Asterisk の動作環境

- ここでは、Asterisk のデュアルスタック環境での動作について説明します。
  - Asterisk の接続形式
  - Asterisk の設定
  - Asterisk が出力するログファイル

Asterisk の接続形式

- 発信側 IP 電話機は REGISTER 送信時に指定したプロトコルファミリに従い、発信を行う(INVITE の送信)。
- Asterisk は、着側が REGISTER 送信時に指定したプロトコルファミリにより、INVITE を転送する。




```
INVITE sip:6001@192.168.1.212:5060;...
From:
<sip:6000@192.168.1.214>;tag=as3350c74c
To: <sip:6001@192.168.1.212:51784>
Contact: <sip:6000@192.168.1.214:5060>
Call-ID: 4c9d4cb63@192.168.1.214:5060
CSeq: 102 INVITE
Content-Type: application/sdp
Content-Length: 442

v=0
c=IN IP4 192.168.1.214
:
```

```
INVITE sip:6001@[2001:db8:cafe:babe::2] SIP/2.0
From: <sip:6000@[2001:db8:cafe::babe::1]>;tag=623328519
To: <sip:6001@[2001:db8:cafe:babe::2]>
Contact: <sip:6000@[2001:db8:cafe:babe::1]>
Call-ID: 134945149
CSeq: 20 INVITE
Content-Type: application/sdp
Content-Length: 384

v=0
c=IN IP6 2001:db8:cafe:babe::1
:
```

設定ファイル

- 待ち受けに使用される IP アドレスは IPv4/IPv6 の両方が記述可能（設定ファイルは sip.conf）
- IPv4 example: bindaddr=0.0.0.0:5060
  bindaddr=0.0.0.0
- IPv6 example: bindaddr=[::]:5060
  bindaddr=::
- bindaddr=0.0.0.0 とした場合、IPv4 のみ受信可能
- bindaddr=<特定の IPv4/IPv6 アドレス>と指定した場合は、そのプロトコルファミリのみ受信可能
- デュアルスタックにする場合は、bindaddr=:: とするが、挙動はOS依存である
  - Linux の場合、デュアルスタックで運用するためには /proc/sys/net/ipv6/bindv6only=0 であることを確認すること。

ログファイル

- ログファイルでは、IPv6 アドレスが出力される。
  - アドレス文字列の実体は getnameinfo() により得られるアドレス文字列が出力される。
  - RFC5952（A Recommendation for IPv6 Address Text Representation）に従うかどうかは実行環境での getnameinfo() の実装に依存する。

```
[Nov 16 22:50:02] VERBOSE[6103] chan_sip.c: Peer audio
RTP is at port [2001:db8:cafe:babe::2]:7078
[Nov 16 22:50:02] VERBOSE[6103] chan_sip.c: Peer video
RTP is at port [2001:db8:cafe:babe::2]:9078
```

ソケット関連の構造体・関数の扱いについて

- BSD ソケットの扱いは、Asterisk の内部でライブラリ化されており、SIP の処理の中で直接呼び出すことはほとんどない。
- ここでは、Asterisk が保持しているアドレス情報や内部関数の中で、BSD ソケットで使用される構造体・関数がどのように使用されているかを解説する。
  - アドレス情報の保持
  - IPv6 汎用関数
  - 実際の接続について

アドレス情報の保持(1)

- IPv6対応前では *struct sockaddr_in* が使用されていたが、IPv6対応後は *struct sockaddr_storage* が使用される

IPv6 対応前
(asterisk 1.4.40)

channel/chan_sip.c より

```
struct sip_pvt {
:
struct sockaddr_in sa;
};
struct sip_peer {
:
struct sockaddr_in addr;
};
```

IPv6対応後
(asterisk 11.6)

include/asterisk/netsock2.h より

```
struct ast_sockaddr {
struct sockaddr_storage ss;
socklen_t len;
};
```

channel/sip/include/sip.h より

```
struct sip_pvt {
:
struct ast_sockaddr sa;
};
struct sip_peer {
:
struct ast_sockaddr addr;
};
```

一部のOS(MacOS 等)では構造体の長さが必要なため、移植性を考慮して追加されている

アドレス情報の保持(2)

・ アドレスの文字列を格納する文字配列の要素数を *INET_ADDRSTRLEN* (16)から *INET6_ADDRSTRLEN* (46)へ変更している

IPv6 対応前
(asterisk 1.4.40)

channel/chan_sip.c より

```
static void realtime_update_peer(…)
{
:
char ipaddr[INET_ADDRSTRLEN]

};
```

IPv6対応後
(asterisk 11.6)

channel/sip/include/sip.h より

```
static void realtime_update_peer(…)
{
:
char ipaddr[INET6_ADDRSTRLEN]

};
```

IPv6 対応汎用関数(1)

【main/netsock2.c】

- int ast_sockaddr_split_hostport(char *str, char **host, char **port, int flags)
  - IP アドレスとポートを分離する
  - ex) [2001:db8:cafe:babe::1]:5060 => 2001:db8:cafe:babe::1 と 5060 に分離する

```
if (*s == '[') {
        *host = ++s;
        for (; *s && *s != ']'; ++s) {
        }
        if (*s == ']') {
                host_end = s;
                ++s;
        }
        if (*s == ':') {
                *port = s + 1;
        }
} else {
    IPv4 文字列を想定した処理(省略)
}
```

IPv6アドレスは"["と"]"でアドレス文字列がくくられているため、その文字列チェックを行っている（rfc3261）

IPv6 対応汎用関数(2)

【main/netsock2.c】

- int ast_sockaddr_resolve(struct ast_sockaddr **addrs,
  const char *str, int flags, int family)
  - アドレス解決を行う。引数 family にプロトコルファミリーが入る
  - 内部で呼ばれる getaddrinfo() の結果が、引数 addrs に保存される
  - 名前解決のみで使用され、getaddrinfo() の結果はチェックされない。

```
memset(&hints, 0, sizeof(hints));
hints.ai_family = family;
hints.ai_socktype = SOCK_DGRAM;
if ((e = getaddrinfo(host, port, &hints, &res))) {
    (エラー処理、省略)
}
res_cnt = 0;
for (ai = res; ai; ai = ai->ai_next) { res_cnt++; }
i = 0;
for (ai = res; ai; ai = ai->ai_next) {
    (*addrs)[i].len = ai->ai_addrlen;
    memcpy(&(*addrs)[i].ss, ai->ai_addr, ai->ai_addrlen);
    ++i;
}
```

getaddrinfo()の検索条件を設定する

getaddrinfo()の検索結果の数を計上し、関数の引数にコピーしている。

IPv6 対応汎用関数(3)

【channel/chan_sip.c】

- static int ast_sockaddr_resolve_first_af(struct ast_sockaddr *addr, const char* name, int flag, int family)
  - 指定したプロトコルファミリでホスト名の解決を行い、先頭の1つだけを返却する
  - 内容は前ページの ast_sockaddr_resolve() を呼び、その一つだけを返す実装となっている。
- static int ast_sockaddr_resolve_first(struct ast_sockaddr *addr, const char* name, int flag)
  - 設定ファイルで指定されたデフォルトのプロトコルファミリでホスト名の解決を行い、先頭の1つだけ返却する
  - 上記の ast_sockaddr_resolve_first_af() の最後の引数 family を、設定ファイルで記述されているサーバの受信アドレスに従って設定される。
    →[::] となっていれば、PF_UNSPEC が指定される。この場合は DNS の返却結果と getaddrinfo() が struct addrinfo 構造体に結果を返す実装に依存する。

IPv6 対応汎用関数(4)

【main/netsock2.c】

- int ast_sockaddr_is_ipv4_mapped(const struct ast_sockaddr *addr)
  - 引数のソケットアドレス構造体が IPv4 mapped address かどうかをチェックする。

```
int ast_sockaddr_is_ipv4_mapped(const struct ast_sockaddr *addr)
{
        const struct sockaddr_in6 *sin6 =
                (struct sockaddr_in6 *)&addr->ss;
        return addr->len && IN6_IS_ADDR_V4MAPPED(&sin6->sin6_addr);
}
```

<netinet/in.h> **で定義されている、アドレスの種類を判定するマクロ**

IPv6 対応汎用関数(5)

【main/netsock2.c】

- int ast_sockaddr_ipv4_mapped(const struct ast_sockaddr *addr, struct ast_sockaddr *ast_mapped)
  - IPv4 mapped address(実体は struct sockaddr_in6)が格納されたソケットアドレス構造体を、IPv4 アドレスが格納された形式(実体は struct sockaddr_in)に変換する

```
if (!ast_sockaddr_is_ipv4_mapped(addr)) {
    return 0;
}
struct sockaddr_in6 *sin6 = (const struct sockaddr_in6*)&addr->ss;
struct sockaddr_in sin4;
memset(&sin4, 0, sizeof(sin4));
sin4.sin_family = AF_INET;
sin4.sin_port = sin6->sin6_port;
sin4.sin_addr.s_addr = ((uint32_t *)&sin6->sin6_addr)[3];
ast_sockaddr_from_sin(ast_mapped, &sin4);
```

ast_sockaddr_from_sin() **内で、**ast_mapped **に** struct sockaddr_in **構造体がコピーされる。**
ast_mapped->len **は、**sizoef(struct sockaddr_in) **が設定される**

実際の通信について

- Asterisk 内部では、これまでに説明した汎用関数を使用して名前解決を実施するが、たいていの場合はgetaddrinfo() の最初のエントリのみを参照する仕様となっている。プロトコルファミリについては、REGISTER 受信時で使用されるプロトコルファミリをもとに決定しているため、Asterisk と IP 電話機間では使用するプロトコルはあらかじめ決定されている。

名前解決の問題と最近のテクニック

getaddrinfoの並びは？

- *getaddrinfo()* は名前解決
  - 出力されるaddrinfo構造体のリストはどういう順序になるのか？
- 長らく RFC3484で定義されていた
- RFC6724 がRFC3484をObsoleteした
  - デフォルトポリシーテーブルの修正
  - アドレス選択ルールの修正
  - フォールバック問題の記述
  - etc...

フォールバック問題

- フォールバックとは
  - クライアントが接続先IPアドレスのリストを得る
  - リストの先頭にあるIPアドレスに接続しようとして、失敗すると次のリスト要素のIPアドレスを試す
- 原因
  - サーバがそのプロトコル・IPアドレスでアプリサービスをしていない
  - ネットワークの接続性が失われている
- 問題
  - タイムアウトを繰り返すので、接続まで時間がかかる
  - 環境によっては数十秒かかる場合も見込まれている

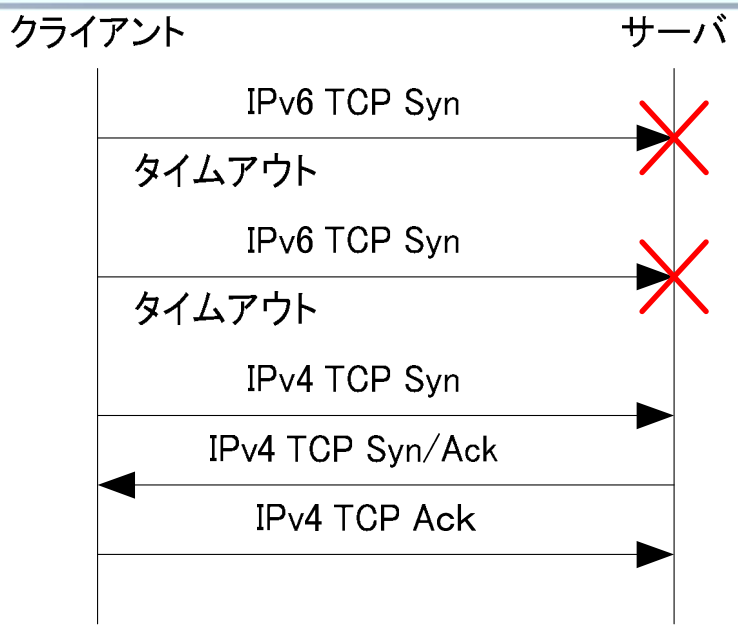
フォールバックの回避

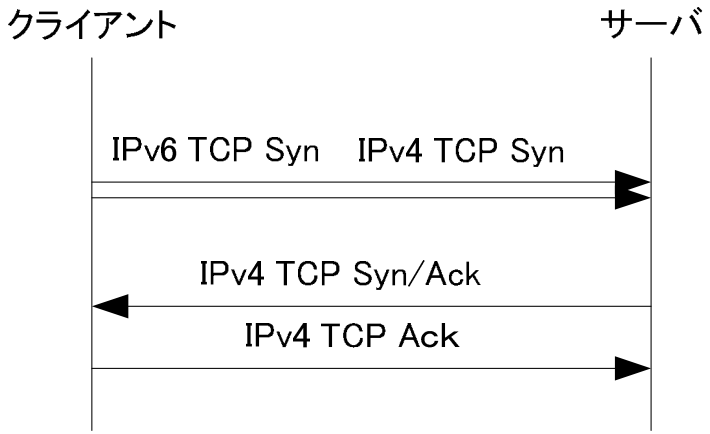
- サーバがサービスしていないIPアドレスはDNSに登録しない
- IPの接続性を健全に保つ
- ポリシーテーブルを変更する
  - Windows: netsh interface ipv6 show prefixpolicies
  - Linux: ip addrlavel show
  - FreeSBD: ip6addrctl show
- サーバサイドだけでの対応では完全には解決できない

Happy Eyeballs

- RFC6555、RFC6556で定義
- 従来はTCP Synの接続失敗を受けて次のTCP Synを送っていた

- Happy EyeballsはIPv6 / IPv4アドレスに両方に対して一気にTCP Synを送る
  - Syn/Ack応答が帰ってきたIPにTCP Ackを送って接続
- 実装依存性が非常に高い
  - 複数I/Fの場合やIPv6アドレスが複数ある場合は？
  - IPv6 Syn/Ackが遅れて届いたら？
  - 今後の展開や運用を注視すべき





組み込み用途でのIPv6対応

組み込み機器へのアドレス埋め込み

- 組み込みではSocketを使うことが多い

- 組み込み機器はいろいろ大変
  - お客様の環境でDNSが使えない
  - 名前解決処理に必要なリソースが不足している
- 組み込み機器でIPアドレスをハードコーディングしたいこともある
  - しかしやめたほうがいい
  - RFC4085でこの問題が記述されている

アドレスハードコーディングの問題

- そもそもIPアドレスは借り物
  - リナンバリングのリスクが伴う
- ホスト名は名前指定して、名前解決には*getaddrinfo()*を使うべき
  - RFC6724やRFC3484 に従った適切な処理が約束されている
  - これを自作するということはRFCに従うコストやそれから外れるリスクを自己負担するということ
- これらのリスク・コスト・インターネットの道義的責任がハードコーディングしないリスク・コストを上回ったとき
  - 十分な注意・サポートとともにやむをえずハードコーディングすることは考えられる

最後に

- IPv6はどんどん浸透してきている
  - アプリでIPv6を先取りして、時代もお客様も先取りしよう
- **WebアプリのIPv6化について、近日中にパブリックコメントを募集する予定ですので、ご協力をお願いします。_O_**
- その他、何かありましたらいつでもこちらまで
  - IPv6普及・高度化推進協議会の連絡先
    - v6info@v6pc.jp
    - https://www.v6pc.jp/jp/info/inquiry_web.phtml

参考文献(Webサイト)

- SIP IPv6 関連
  - SIP FORUM IPv6 task group
    http://www.sipforum.org/content/view/398/286/
- Asterisk 関連
  - Asterisk wiki
    http://wiki.asterisk.org
- IPv6 普及・高度化推進協議会 IPv4/IPv6共存WG
  - http://www.v6pc.jp/jp/entry/wg/2012/12/ipv610.phtml
- IPv6 関連
  - Internet Week 2011より 事例から学ぶIPv6トラブルシューティング
  - http://www.nic.ad.jp/ja/materials/iw/2011/proceedings/t2/t2-02.pdf

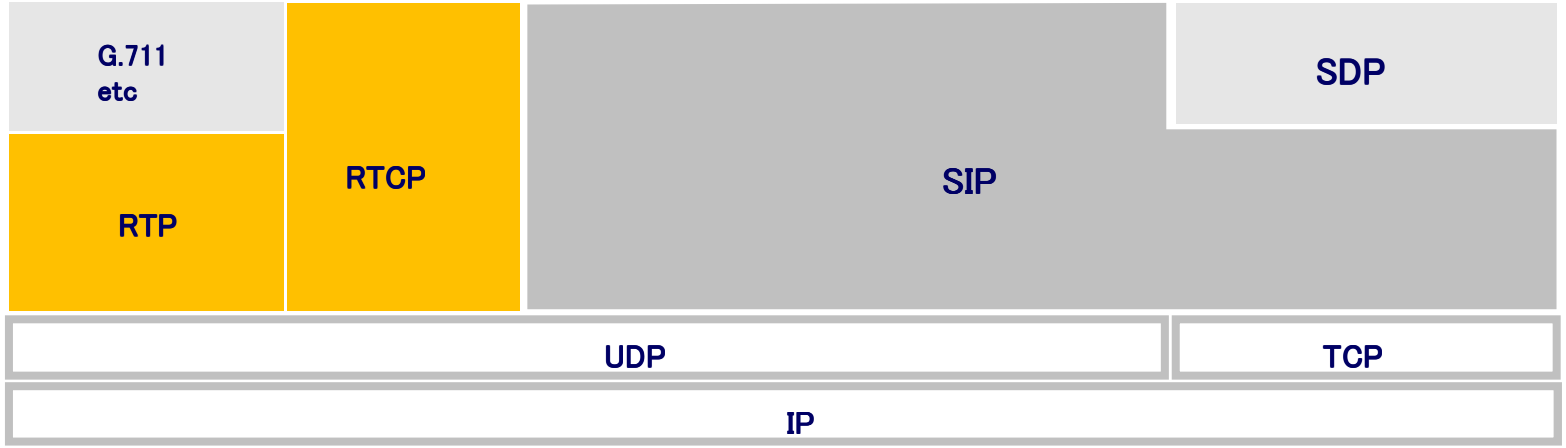
＜参考＞SIPについて

SIP について

- SIP プロトコル概要
- 接続シーケンス

SIP プロトコル概要

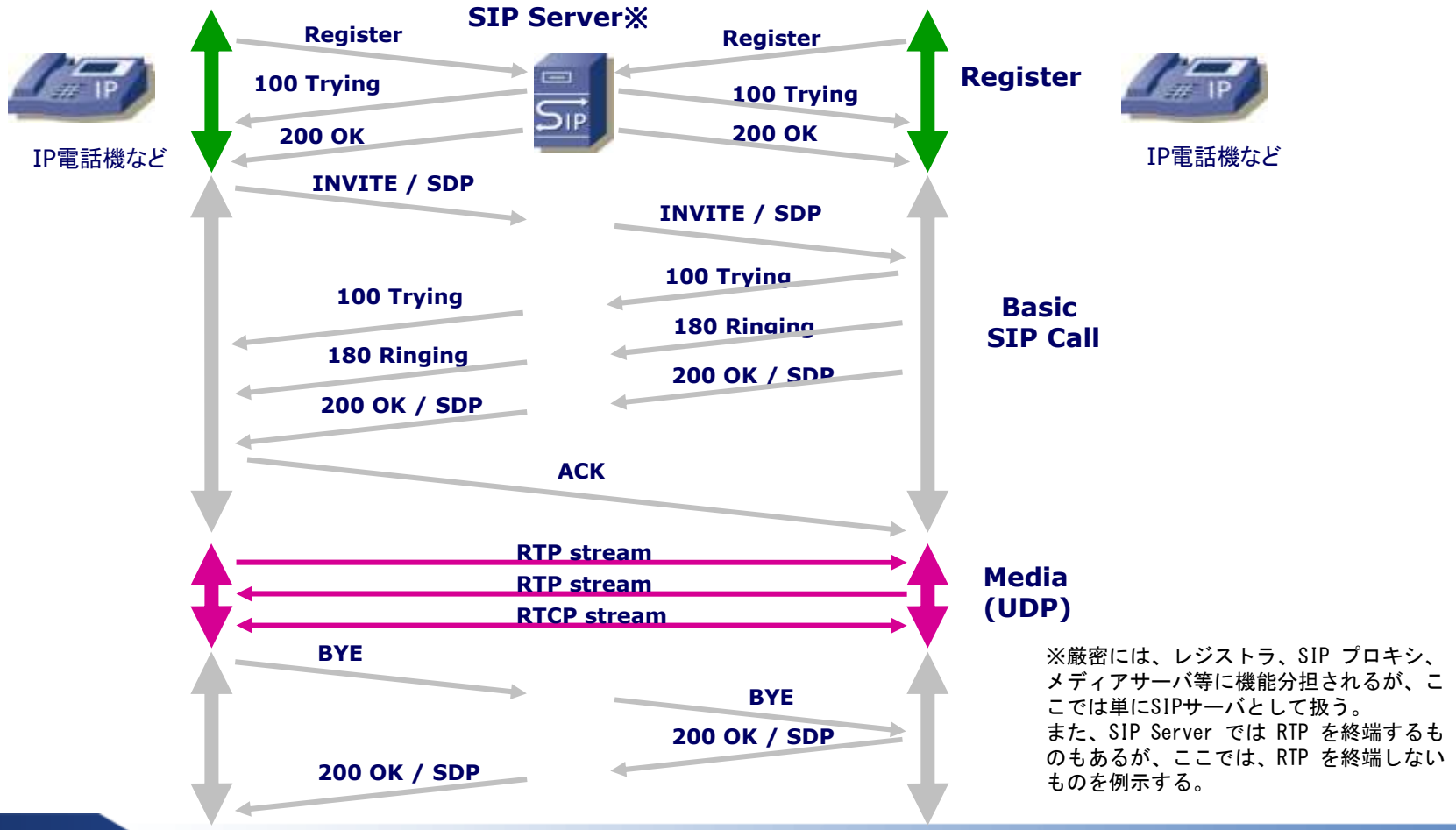
- SIP(Session Initiation Protocol)とは、2つ以上のクライアント間でセッションを確立するためのIETF標準の通信プロトコル
- IETFにて、汎用のセッション制御プロトコルとして開発された
- 特徴として、HTTPに似た、テキストベースのリクエストとレスポンスによって通信を行い、相手先(通話先) はURI(Uniform Resource Identifier)を指定する。



G.711: ITU-T standard for audio companding.
SDP: Session Descripton Protocol
RTCP: RTP Control Protocol
SIP: Session Initiation Protocol
RTP: Realtime Transport Protocol
TCP: Transmisson Control Protocol
UDP: User Datagram Protocol
IP: Internet Protocol

SIP の一般的な接続シーケンス

- IP電話機などのSIP端末は、SIPサーバにREGISTERを送信することで発着可能な状態となる。この時に、IP電話機とSIPサーバ間で使用するプロトコルファミリが決まる。



※厳密には、レジストラ、SIP プロキシ、メディアサーバ等に機能分担されるが、ここでは単にSIPサーバとして扱う。
また、SIP Server では RTP を終端するものもあるが、ここでは、RTP を終端しないものを例示する。