2006－ＱＡＩ－18（９）
2006／1／20

複数アドレス対応のための Socket API 拡張

丸山 伸† 小塚 真啓†† 中村 素典††† 岡部 寿男†††

† 京都大学大学院情報学研究科
†† 京都大学大学院法学研究科
††† 京都大学学術情報メディアセンター
京都市左京区吉田本町

E-mail: †marushin@net.ist.i.kyoto-u.ac.jp, ††kozuka@wide.ad.jp, †††{motonori,okabe}@media.kyoto-u.ac.jp

あらまし 近年，複数の IP アドレスを同時に利用するトランスポート層のプロトコルが注目を集めている．SCTP はこのようなプロトコルの代表的なもので，「TCP 同様の信頼性」「TCP に類似した API」でありながら，「両端が複数の IP アドレスを利用する」という特徴を持つ．しかし，これまでの Socket API は「1 つの Socket には 1 つの IP アドレス」が割当てられることを前提としているため，1 つの Socket に複数の IP アドレスを割りあてる SCTP とは親和性の悪い部分がある．そこで本論文では，1 つの socket に複数の IP アドレスを割当てるための PF_BUNDLE(AF_BUNDLE) 型を提案し，それに対応するよう Socket API を拡張する手法を提案する．その上で，getaddrinfo(3) のようなライブラリがこれらの PF_BUNDLE 型を利用するようにホスト名の表記を拡張することで，SCTP のような複数アドレスを用いるプロトコルも，従来の「プロトコル独立プログラミング」の枠組みで適切に扱えることを示す．

キーワード ソケット API, SCTP, Protocol 独立プログラミング, 複数アドレス

An Extention of Socket API for Multi-Address Support

Shin MARUYAMA†, Masahiro KOZUKA††, Motonori NAKAMURA†††, and Yasuo OKABE†††

† Graduate School of Informatices, Kyoto University
†† Graduate School of Law, Kyoto University
††† Academic Center for Computing and Media Studies, Kyoto University
Yoshida-honmachi, Sakyo-ku, Kyoto 606-8501, Japan

E-mail: †marushin@net.ist.i.kyoto-u.ac.jp, ††kozuka@wide.ad.jp, †††{motonori,okabe}@media.kyoto-u.ac.jp

Abstract Recently transport protocols which handle multiple IP addresses are focused. SCTP is a representative of such protocol, and has not only features of "TCP like reliability", "TCP like API", but of "Multiple IP addresses on both ends". Since "Socket API" assume that "single IP address for each socket", they are incompatible with SCTP which allocates multiple IP addresses for a single socket. In this paper, we first propose "PF_BUNDLE (AF_BUNDLE)" which are new kinds of address family and protocol family, which handle multiple IP addresses within a single socket, and we also propose extentions to "Socket API" to suite for "PF_BUNDLE". Next , we also improve libraries such as getaddrinfo(3) to be capable of handling "PF_BUNDLE", and show that "Protocol Independent Programming" style is still suitable for protocols like SCTP.

Key words Socket API, SCTP, Protocol Independent Programing, Multi-Address

1. はじめに

近年，複数の IP アドレスを用いたトランスポート層が注目されている．特に SCTP [1] は複数の IP アドレスを利用した信頼性の高い接続を提供するプロトコルとして注目されている．

この SCTP を利用するための SCTP API [2] は IPv6 Socket API Extension [3] の拡張として定義されているものであり基本的なプログラム構造は TCP におけるものと類似する．bind(2) や connect(2) といった API は引数として 1 つの IP アドレスのみを指す 1 つの sockaddr 構造体を受け取る．そのため，SCTP のような複数の IP アドレスを用いるプロトコルにおいて，bind(2) や connect(2) といった API をそのまま利用することは出来ず SCTP 独自の拡張が必要となる．SCTP API においては，bind(2) や connect(2) の SCTP 向けの拡張である sctp_bindx(3)

や sctp_connectx(3) のような関数を用いる SCTP 独特のプログラミングスタイルを採ることが推奨されているが，このようなプログラミングスタイルの変更は非常にコストが高い．

そこで本論文においては，新しいプロトコルファミリー "PF_BUNDLE" を提案し，1 つのソケットで複数アドレスを扱えるようにすることで，SCTP のような複数の IP アドレスを用いるプロトコルにおいても bind(2) や connect(2) を利用出来るようにする．また，ホスト名の表記を拡張して，"{" と "}" で複数のホスト名等が囲まれている時には，getaddrinfo(3) の返り値等で "PF_BUNDLE" 型の情報を含む構造体を利用して複数の IP アドレスの情報をまとめて返すようにする(注1)．

この一連の拡張を行うことで，既存のプロトコル独立プログラミング [4] のプログラミングスタイルを変更することなく，複数の IP アドレスを扱うプロトコルを利用できることを示す．

以下，2 章においては，SCTP の特徴とプロトコル独立プログラミングについて述べた上で，従来の socket API では複数アドレスを利用するプロトコルに対応する事が困難であることを述べる．3 章においては PF_BUNDLE および 構造体 sockaddr_bundle を提案する．4 章において，この提案を実装する．5 章においては，複数のアドレスを扱うプロトコル上での今後の課題を検討する．また，6 章において結論を述べる．

2. これまでの研究

TCP に続くコネクション指向のトランスポートプロトコルとして SCTP [1] のような「複数の IP アドレスを利用するプロトコル」が注目を集めている．SCTP は接続の両端で複数のアドレスを用いることが出来るだけでなく，IPv4 と IPv6 の両方が混在しうる複数アドレス環境を単一の接続で利用することが出来るため，これまでにも増してプロトコルに依存しないプログラミングスタイルの重要性が増している．

2.1 プロトコル独立プログラミング

IPv4 と IPv6 の両方を個別に意識する必要のないプログラミングスタイルとしてプロトコル独立プログラミング "Protocol Indepenedent Programming" [4] が提案されている．

このプログラミングスタイルではサーバ側，クライアント側いずれにおいても，ソケットの作成，アドレスの割当てを順に行い，その後サーバ側では listen(2)，クライアント側では connect(2) を呼び出すことで，クライアント側からサーバへの接続が行われる．

サーバ側では同時に複数のアドレスで待ち受けた上で，クライアント側では複数のアドレスに対する接続を順次試みるという一連の流れを IPv4 と IPv6 との両方のアドレスが混在することを意識することなく行うために，getaddrinfo(3) が利用される．getaddrinfo(3) の出力は addrinfo 型の構造体のリストで与えられ，個々の構造体にはそれぞれプロトコルとアドレスが列挙される．

このようにして列挙された addrinfo 構造体のリストに対して，プロトコル独立プログラミングに従ったサーバは，

（1） getaddrinfo(3) の出力から，(Protocol, sockaddr 構造体) の情報を含む addrinfo 構造体のリスト を得る
（2） 得られたリストのそれぞれに対して，
（a） socket(2) によりソケットを作る
（b） bind(2) によりソケットに sockaddr 構造体を渡す
（c） listen() する

の一連の処理によりソケットの配列を得る．

また，プロトコル独立プログラミングで記述されたクライアントは

（1） getaddrinfo(3) の出力から，(Protocol, sockaddr 構造体) の情報を含む addrinfo 構造体のリスト を得る
（2） 得られたリストのそれぞれに対して，
（a） socket(2) によりソケットを作る
（b） bind(2) によりソケットに sockaddr 構造体を渡す
（c） connect(2) によりサーバに接続を試みる．
（d） 接続に成功したらループを抜ける．

の一連の処理により，1 つの接続済みソケットを得る．

すなわち，サーバ側において getaddrinfo(3) により，複数の addrinfo 構造体のリストを得た時には，そのそれぞれに対して順次 bind(2) を試み，bind できた socket の配列を得た上で，このいずれかの socket にクライアントが接続してくるのを待つ．クライアント側において，接続しようとするサーバのホスト名とサービス名を指定して getaddrinfo(3) を呼び出した際に複数の addrinfo 構造体のリストを得た時には，得られた各構造体に含まれるアドレスとプロトコルの組合せのそれぞれに対して順に接続を試み，最初に接続できたものを利用する．

2.2 SCTP と Socket API の SCTP 拡張

SCTP は同時に複数のアドレスを利用することが出来るトランスポートプロトコルであるため，従来の IPv6 Socket API Extension [3] が前提としている，「1 つのソケットには 1 つのアドレスが対応する」という原則には適応しない．しかしながら，既存のプログラミングスタイルを大きく変更することはコストが高いため，SCTP を利用するための API (SCTP API) [2] は，「1 つのソケットを利用して信頼性のある通信を行う」「getaddrinfo(3) を利用して sockaddr 型のアドレス構造体を用いる」といった基本的なプログラム構造においては既存の IPv6 Socket API Extension に類似するものとなっている．

しかしながら，既存のプログラミングスタイルを尊重しつつ，複数のアドレスを同時に扱えるようにするため，SCTP API は

- ソケットに対応付けられたアドレスを保持するためのバッファをカーネル内に確保し，そこに複数の アドレスを保持できるようにする
- connect(2) の API に 1 つの socket が渡された際には，指定された 1 つの IP アドレスを用いて接続などを行い，接続時に両端の持つ複数の IP アドレスを相互に交換し，上記バッファに保存する

といった方針で設計されている．

(注1)：AF_BUNDLE も PF_BUNDLE と同様に定義することになるが，以下，すべて PF_BUNDLE と記す

2.3 SCTP のプロトコル独立プログラミングに対する問題点

前節で述べた SCTP API の設計方針により，プロトコル独立プログラミングで記述されたアプリケーションであれば，カーネルとライブラリの入れ替えにより getaddrinfo(3) の返り値に PROTO_SCTP を含めるだけで，アプリケーションに一切の変更を加えることなく SCTP による接続が試みられるようになる．また，既存のソケットでは 1 つのアドレスしか指定できないので bind(2) や connect(2) においては 1 つのアドレスしか指定できないが，SCTP の接続の確立時には相互の持つアドレスを交換するため，結果として全てのアドレスを用いて接続しようと試みる．すなわち，一般に両端の持つアドレス全てを用いて接続を確立したい場合には特に問題は生じない．

しかし，この手法による SCTP 対応では，接続元や接続先として複数のアドレスをまとめて指定することが出来ないため，

- クライアント側において，複数のアドレスに対し順次ではなく同時に接続を試みることが出来ない
- サーバ側において，複数のアドレスにあらかじめ bind しておくことが出来ない

といった問題が生じる (それぞれの問題については以下の節において詳細を述べる)．

もちろんこれらの問題は，SCTP API で定義されている sctp_bindx(3), sctp_connectx(3) といった関数を利用することにより解決することも出来るが，そのためにはプログラミングスタイルの変更が必要となりコストが大きく，適切な解決策とは言えない．

2.3.1 クライアント側における問題点

クライアント側のプログラミングにおいては，サーバの名前を getaddrinfo() により addrinfo 構造体のリストに変換し，そのリストの示すアドレスとプロトコルに対して順次 connect(2) を試み，最初に接続できたアドレスとプロトコルの対をその後の通信で用いる．

getaddrinfo(3) は，サーバに複数のアドレスが割当てられている時には個々のアドレス毎に 1 つの addrinfo 構造体を割当てるだけでなく，TCP と SCTP といった複数のプロトコルが利用できる時には，更にプロトコル毎にも addrinfo 構造体を列挙する．そのため SCTP に関する情報を加えて返す際には，addrinfo 構造体の数は 2 倍に増えることになる．この時，SCTP を TCP よりも優先的に利用するためには SCTP による接続を先に試すことになるが，もしサーバ側が SCTP に対応していない場合には，TCP にフォールバックするまでに複数のアドレスに順次接続を試みるという不必要に長い時間を待たされることになる．

この問題は SCTP では本来 1 つのソケットに複数のアドレスを割当てることが出来，全てのアドレスを一度に connect(2) に渡して同時に接続を試みることが出来るにも関わらず，addrinfo 構造体に 1 つのアドレスしか保持できないため，それぞれのアドレスに順次接続を試みていることに原因の一端がある．

2.3.2 サーバ側における問題点

サーバ側のプログラミングにおいては，サービスをどのアドレスで提供するかという点が問題となる．

サーバ側のプログラムでは，待ち受けをしたいアドレスを順次 getaddrinfo(3) を用いて addrinfo 構造体のリストに変換した上で，そのリストに対して順次 socket(2) 及び bind(2) を呼び出すことでソケットのリストを得る．その後，得られたソケットのそれぞれで，クライアントからの接続を待ち受ける．

すなわち，本来 SCTP は 1 つのソケットに複数のアドレスを割当てることにより通信経路の冗長性を高めることが出来るにもかかわらず，この手法では各ソケットに 1 つずつのアドレスしか割当てられないため，SCTP の利点を活かせていない．

3. 提案手法

IPv4 と IPv6 の両アドレスが混在する環境において，その両者の存在を意識する必要のないプログラミングスタイルとして，プロトコル独立プログラミングが提唱されている．しかしながら，IPv4 と IPv6 の混在する複数のアドレスを扱うトランスポートプロトコルである SCTP のプログラミングについて検討する上で，既存の「1 つのソケットに 1 つのアドレス」という制限が問題となってきた．しかしながらプログラミングスタイルの変更は非常にコストが大きいため，プロトコル独立プログラミングの枠組みを可能な限り崩すことなく 1 つのソケットで SCTP を扱う手法が好ましい．

そこで，IPv4 と IPv6 の混在する複数のアドレスを 1 つのソケットで 1 つの束として取り扱うための新たなプロトコルファミリーとして PF_BUNDLE を定義する手法を提案する．また，このアドレスファミリーに対応したソケットアドレス構造体 sockaddr_bundle を定義する．さらに，getaddrinfo(3) に特別な記法のホスト名を渡した際には sockaddr_bundle 構造体を指す addrinfo 構造体を含むリストを返すようにした上で，socket(2), bind(2), connect(2) 等の関数が sockaddr_bundle 構造体を取り扱えるようにする．この一連の拡張を行うことで，既存のプロトコル独立プログラミングに従うプログラムを，カーネルとライブラリの変更のみで，バイナリの再コンパイルを行うことなく，複数アドレスや複数プロトコルを扱うプロトコルに対応出来るようにする．

3.1 sockaddr_bundle 構造体

IPv4 と IPv6 の混在する複数のアドレスを 1 つのソケットで扱うためのプロトコルファミリーとして PF_BUNDLE を定義する．

このプロトコルファミリーに対応したソケットアドレス構造体 sockaddr_bundle を次のようなに定義する．

```
struct sockaddr_bundle {
    unsigned char      sb_len;
    sa_family_t              sb_family;
    struct sockaddr          sb_addr[0];
};
```

sockaddr_bundle 構造体の最後の要素 sb_addr には，sockaddr_in や sockaddr_in6 などの構造体が必要な個数だけ連続して並べら

表 1　ホスト名表記の拡張

Table 1　Extention of Hostname Expression

複数アドレスをまとめて取り扱う表記方法: - FQDN や IPv4 や IPv6 などのアドレス表記を ',' で区切り，全体を {} で囲う - FQDN が含まれている場合，DNS を利用してアドレスへと IPv4 や IPv6 などのアドレスに変換するものとする - {www.example.com,fe80::1%fxp0} のように linklocal address も含めてよいものとする 例:　www.example.com が 192.168.0.1 と 10.0.0.1 を返す場合 - {www.example.com} は {192.168.0.1,10.0.0.1} と等価 - {www.example.com,127.0.0.1,::1} は {192.168.0.1,10.0.0.1,127.0.0.1,::1} と等価

れる．また，sb_family には PF_BUNDLE を指定する．sb_len は実装によっては存在しないこともある．

3.2　getaddrinfo(3) の拡張

getaddrinfo(3) 等の出力においても sockaddr_bundle 構造体を利用するために，getaddrinfo(3) に渡されるホスト名ないしはアドレスの列の表記を表 1 のように拡張する．

すなわち，getaddrinfo(3) に対して中括弧 ("{" 及び "}") で囲まれた複数のホスト名ないしはアドレスの列が渡された時には，これらのアドレスを束ねた sockaddr_bundle 構造体が返されるようにする．

また，getaddrinfo(3) が PF_UNSPEC を hints として問合せを受けた場合には，その返り値となる addrinfo 構造体のリストは図 1 のように，単一アドレスを表わす addrinfo 構造体の情報よりも，複数アドレスを束ねた PF_BUNDLE 型の情報を含む addrinfo 構造体をリストの前方に並べることにする．

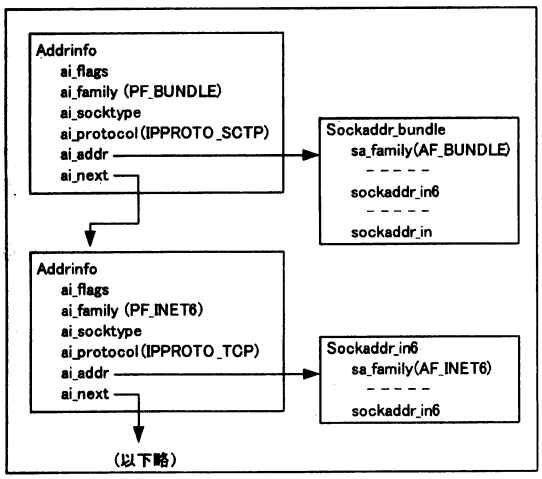

図 1　PF_BUNDLE を含む getaddrinfo(3) の返り値

Fig. 1　Return value of getaddrinfo(3) with PF_BUNDLE

3.3　bind(2), connect(2) の拡張

bind(2) や connect(2) は従来単一の sockaddr 型の構造体を受け取るが，これを sockaddr_bundle 型の構造体も受け渡しが出来るように拡張する．その際，SCTP のような複数アドレスを利用出来るプロトコルの時には渡された複数のアドレスを一度に処理するようにする．

このように sockaddr_bundle 構造体を通じて複数のアドレスをまとめて受け取ることで，指定されたアドレス群だけを使って bind(2) することや，connect(2) が一度に複数のアドレスに INIT を送出したり，次々と異なるアドレスに INIT を再送することが可能となる．

4.　提案手法の実装と評価

本提案の効果を確認するために，FreeBSD Ver.5 上に KAME による IPv6 環境を構築し，そこで提案手法を実装した．SCTP スタックは sctp.org のものをベースに拡張した．

前節で提案した sockaddr_bundle 構造体は SCTP API のライブラリ及び，bind(2) と connect(2) に関するものであるが，ライブラリとカーネルとの受渡しにおいても同一の構造体を用いることにした．

この手法により設計されたライブラリと差し替えることで，プロトコル独立となるように記述された既存のアプリケーションは，再コンパイル等を行うことなく SCTP を利用するようになることを確認した．

この実験には tcpbridge [5] というアプリケーションを用いた．また，従来ホスト名として指定していた文字列の代わりに，適切な書式で記述された複数のアドレスを示す文字列を指定することで，指定された複数のアドレスを利用した SCTP 接続となることを確認した．

5.　議　　論

今回の実装で利用した環境においては提案手法は問題なく動作したが，今後より多くの環境において効果的に実装するには，次のような点について検討する必要がある．

5.1　sockaddr 構造体の列挙のやり方

複数の sockaddr 構造体を 1 つにまとめた sockaddr_bundle 構造体を導入するにあたり，それをどのような構造にするか検討する必要がある．構造体の設計においては，メモリの利用効率とアクセス効率とのトレードオフ等の制約があり，簡単に決めることは出来ない．

まず，sockaddr 構造体は memcpy(3) 等の関数で複製されることが予想されるため，構造体の内部にポインタによるリンクを持つような構造を採ることは好ましくない．そこで本実装においては，sockaddr_bundle 構造体における sockaddr 構造体は，ポインタによるリンク構造ではなく，複数の sockaddr 構造体を次々と並べるものとした．

次に，サイズの異なる複数種の sockaddr 構造体をつぎつぎと並べるものと，一定のサイズ毎に区切って並べるものとを比較する．

前者の方式は FreeBSD のように，sockaddr 構造体にそのサ

イズをあらわす要素 sa_len が含まれる場合には，列挙された sockaddr 構造体の先頭へのポインタさえ渡されれば，2 つ目以降の構造体のアドレスを容易に取得することが出来る．しかし Linux や Windows のように sockaddr 構造体に sa_len が含まれない実装においては，この方式では IPv4, IPv6 等の各プロトコルとそれに対応した sockaddr 構造体のサイズとの対応を事前に知る必要があるという問題がある．また、sockaddr_bundle 構造体が渡された際に，そこにいくつのアドレスが含まれているのかを知るには順次 sockaddr 構造体を辿って数えなければならない．また別の問題として，FreeBSD における sockaddr_in6 構造体のサイズは 28 バイトであり，sockaddr_in6 構造体を列挙した時にはアラインメントの問題も生じる．

sockaddr 構造体を一定の間隔毎に次々と並べる後者の方式としては，sockaddr_storage 構造体を次々と並べてそのそれぞれに sockaddr 構造体を納める方式が検討される．この方式では先の方式で述べた問題は生じないが，その反面メモリ効率が非常に悪い．例えば FreeBSD5 において各構造体のサイズは，sizeof(sockaddr_in)=16, sizeof(sockaddr_storage)=128 である．

5.2 getnameinfo(3) の問題

sockaddr_bundle 構造体を getnameinfo(3) に渡した際にどのような情報を返すべきかが問題となる．

既存の getnameinfo(3) は，渡された sockaddr 構造体に対して，ホスト名の文字列表記ないしはアドレスの文字列表記を返す．それに対し複数のアドレスを示す sockaddr_bundle 構造体ではそれぞれの sockaddr 構造体を逆引きしてテキスト形式ないしは NUMERIC 表記とした上で，本論文の提案に含まれる「ホスト名の拡張表記」の形式で返すことが好ましい．

また，2 つの sockaddr 構造体が指し示すアドレスが同一のものであるかどうかを比較する際に「getnameinfo(3) を NUMERIC で引いて得られた文字列同士を比較する」といった手法 [6] が提案されているので，これとの整合性を取らなければならない．この際，複数のアドレスをどの順序で列挙するかが問題となる．

5.3 複数アドレスに対する connect 時における SCTP の挙動

本提案に従い sockaddr_bundle 構造体を connect(2) に渡すようにすることで，カーネルが複数のアドレスを一度に受け取れるように拡張した．

次に SCTP のような両端が複数のアドレスを用いて接続を行えるプロトコルにおいて connect(2) において複数のアドレスを一度に受け取った場合には，個々のアドレスに接続を試みて待ち時間が生じるのを防ぐために，DoS とならない適切な範囲でアドレス群に対して一度に接続を試みる方が良い．そこで，本実装においては connect(2) に複数のアドレスが渡された時には INIT パケットをまとめて送るようにした．

しかし，このように多数の INIT パケットを送ったとしても，現在の実装では最初に INIT_ACK パケットを送り返してきた相手に対してのみ COOKIE_ECHO パケットを送り，それ以降の INIT_ACK パケットは無視されるようになっている．そのため最初に INIT_ACK パケットを返した相手に対してのみ COOKIE_ECHO パケットが送出される．

この後もし何らかの理由により COOKIE_ACK パケットが返ってこないという状況になった時には，他のアドレスに接続できる可能性が残っているにもかかわらず，接続を確立することは出来ない．この問題は sctp_auth_chunk [7] を利用しているにもかかわらず，auth_chunk の鍵の設定を間違えた時などに発生する問題である．そこで今後，1 つ目の INIT_ACK パケットの処理が ESTABLISHED にならなかった時には，2 つ目以降の INIT_ACK パケットに対しても順次接続処理試みるといった考慮もするべきと思われる．

5.4 ホスト名の拡張表記の汎用性

本論文において提案した「ホスト名の拡張表記」は SCTP 等の複数のアドレスを扱うプロトコルのみならず，複数のアドレスを扱うアプリケーションやトランスポートプロトコルにおいても有益なものであると考える．

そのため，様々なアプリケーションにおいて汎用的に使えるようなものへと修正するべきと考える．今回の実装においては，“{” と “}” とで囲む記法を利用したが，この記法をそのままでは利用できないアプリケーションが存在する．今後，この点についても調査と拡張を行い，可能なかぎり汎用的な定義とするべきと思われる．

5.5 ホスト名拡張表記 と サーバの単一性

本提案の「ホスト名の拡張表記」では，“{” と “}” で囲まれた複数のホスト名を順次 DNS に問合せ，それらに対する回答のすべてを順次列挙して sockaddr_bundle 構造体を作成する．複数のホスト名が列挙されている場合でも，1 つのホスト名に対して複数の A レコードや AAAA レコードの回答がある場合でも，いずれも同等に扱っている．

これは，アドレスやホスト名がどのように列挙されているにせよ，指し示す先のホストが 1 台であるか複数であるのかがは区別されていない．この点を今後さらに考察するべきと考える．

6. おわりに

プロトコル独立プログラミングに従ったプログラミングスタイルで記述されたプログラムは，カーネルと getaddrinfo(3) が SCTP 等の複数アドレスを同時に利用するプロトコルに対応していれば，1 つのソケットにおいて複数アドレスを取り扱うことが出来るが，従来のソケットは 1 つのアドレスのみが割当てられることを前提としていて，接続の確立時には複数のアドレスを順次試みることとなるため，接続に時間がかかる等の問題が生じる．

そこで本論文においては，複数のアドレスを 1 つのソケットで扱うための新たなプロトコルファミリー (PF_BUNDLE) を提案した．また，この PF_BUNDLE を扱う構造体 sockaddr_bundle 型を提案し，これを用いるようにカーネル及びライブラリを拡張し，実装を行い動作を確認した．

今後この方式を実際に利用する上では，この形式で定義されたソケットのアドレスを，どのようにテキスト表記するかという点に関係する問題等が残されている．

また，SCTP のような複数のアドレス同時に利用できるトラ

ンスポートプロトコルは今後モビリティなどを導入する上で有用なものであるため，本提案よりもさらに低コストでこれらのプロトコルを利用する手法を検討する必要がある．

謝辞 本論文の作成にあたっては，Cisco Systems 株式会社の Randall Stewart 氏をはじめ，Cisco Systems 株式会社の SCTP 研究チーム，また KAME チーム等，多くの SCTP 研究者に議論に参加して頂きました．この場をお借りして心よりの感謝を申し上げます．

文　献

[1] R. R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. J. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang and V. Paxson, "Stream Control Transmission Protocol", RFC2960, http://www.ietf.org/rfc/rfc2960.txt, October 2000.

[2] R. Stewart, Q. Xie. ,L. Yarroll, J. Wood, K. Poon and M. Tuexen, "Sockets API Extensions for Stream Control Transmission Protocol (SCTP)", draft-ietf-tsvwg-sctpsocket-11.txt, http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-sctpsocket-11.txt, February 2005.

[3] R. Gilligan, S. Thomson, J. Bound, J. McCann and W. Stevens , "Basic Socket Interface Extensions for IPv6.", RFC 3493, http://www.ietf.org/rfc/rfc3493.txt, February 2003.

[4] Kazu Yamamoto, "Protocol Independent Programming", http://www.mew.org/~kazu/doc/piprog.html, December 2003.

[5] 嶺原 桂二郎, 菊地 高広, 中野 博樹, 能城 茂雄, 勝枝 俊輔, 藤原 和典, 丸山 伸, WIDE 研究会 2002 年度報告書 "tcpbridge: IRC の運用状況とデータ解析" http://www.wide.ad.jp/project/document/reports/pdf2002/part16.pdf, March 2003.

[6] Jun-ichiro Hagino, "[USERS 2622] Re: comparsion of sock-addr_storage", IPv6 Mailing list archive, http://master.buildlab.kame.net/users@jp.ipv6.org/msg02565.html, October 2002.

[7] M. Tuexen, R. Stewart, P. Lei and E. Rescorla, "Authenticated Chunks for Stream Control Transmission Protocol (SCTP)", draft-ietf-tsvwg-sctp-auth-01.txt, http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-sctp-auth-01.txt, October 2005.