Vol.2009-IOT-7 No.6
2009/10/9

既存システムとの親和性を考慮した動的再構成可能な通信コンポーネントの提案

川島龍太[†] 計宇生[††] 丸山勝巳[††]

筆者らはこれまで，既存のネットワーク型アプリケーションに対して一切の変更を加えずに通信機能を拡張するためのソフトウェア実行環境である FreeNA を提案してきた．FreeNA では，アプリケーションのユーザは拡張する機能を事前に設定ファイルに記述しておかなければならず，通信相手となるアプリケーションにおいても，あらかじめ同様の設定を行っておく必要があった．したがって，サーバシステムのように不特定多数のクライアントと通信を行うシステムに対しては，上記のような静的な機能設定を行うことができないという問題があった．

そこで本研究では，FreeNA にネゴシエーション機能を持たせることによって，エンドアプリケーション同士が通信を開始する直前に拡張機能を決定できるようにした．ネゴシエーションのための通信チャネルには，アプリケーション用の内部チャネルおよび SIP プロトコルを用いた外部チャネルのいずれかを用いる．また，ネゴシエーションそのものは SDP プロトコルのオファー／アンサーモデルに基づいて行うため，既存システムとの親和性や運用性を損なわずに動的な機能拡張を行うことができる．さらに，FreeNA によってもたらされる拡張機能のコンポーネント性と，SDP プロトコルの更新メカニズムを応用することによって，通信品質やマシン使用状況などの変化に応じて拡張機能を適応的に再構成することも可能になる。

A Proposal for Dynamic Reconfigurable Communication Components Compatible with Existing Systems

Ryota Kawashima,[†] Yusheng Ji[††] and Katsumi Maruyama[††]

We have proposed a software execution environment system called FreeNA which enables users to extend their existing applications transparently without any modification of them. For both end-applications, the application's user has to decide which extensions will be used during the communication in advance. Therefore, previous system cannot be applied to server-like systems communicate with unspecified client applications.

In this paper, we extend FreeNA to have a negotiation mechanism to decide which extensions will be used before starting the communication. As the negotiation channel, we can use an inner channel which is also used as application channel, or an outer channels which is used with SIP protocol. Moreover, negotiation itself is conducted based on SDP offer/answer model to adapt to existing systems. As a result, users can deploy our system without brand-new knowledge about the negotiation and altering existing system settings. In addition, we propose an adaptive reconfiguration mechanism which evaluates current communication quality or machine resource usage to recompose suitable extensions on-the-fly.

1. はじめに

近年における通信技術やインターネット技術の進展により，有線・無線を問わない多様なネットワーク環境の下でアプリケーションを動作させることが可能になった．当然，ネットワーク環境が異なるとそれに伴って帯域幅や信頼性あるいは機能性などが変わるため，ネットワーク型アプリケーションやその下で動作するOS，ミドルウェアは，そのような環境の変化に対応できるように構成されいている必要がある．例えば映像ストリーミングアプリケーションを考えた場合，帯域幅が十分にあり，かつ通信品質も安定していれば高解像度に対応した符号化方式を用い，逆に帯域幅が限られており，かつ無線環境のようにパケットロスが多発するような場合は，低解像度の符号化方式と共に誤り訂正符号を用いるという使い方ができると便利である．もちろん，上記のような機能をアプリケーション

† 総合研究大学院大学 複合科学研究科
School of Multidisciplinary Sciences, The Graduate University for Advanced Studies (SOKENDAI)
†† 国立情報学研究所
National Institute of Informatics (NII)

自身で実現することも可能であるが，通信に関連する機能だけでも，モニタリング，帯域幅管理，圧縮，セキュリティ，モビリティ管理など多岐に渡るため，単一のアプリケーションでこれらの機能を全てサポートすることは困難である．したがって，利用するネットワーク環境の特性に応じてアプリケーションを拡張するためには，ネットワーク環境をモニタし，適切な機能をアプリケーションに対して動的に追加・削除するための共通プラットフォームが必要になる．

筆者らはこれまで，ネットワーク型アプリケーションの下で動作するネイティブ型のソフトウェア実行環境である FreeNA[1] を提案してきた．FreeNA を用いることにより，ユーザは OS に依存しない抽象化された方法によって，既存のアプリケーションを一切変更することなく先述のようなネットワーク機能を追加することができる．しかし，ユーザは拡張する機能を事前に設定ファイルに記述しておく必要があり，同時に通信相手となるアプリケーションにおいても同様の設定を行っておく必要があった. そのため，サーバシステムのように不特定多数のクライアントと通信を行うシステムに対しては，上記のような静的な機能設定を行うことができないという問題があった．

そこで本研究では，FreeNA にネゴシエーション機能を持たせることによって，エンドアプリケーション同士が通信を開始する直前に，拡張機能を決定できるようにする．FreeNA 同士でネゴシエーションを行うための通信チャネルとして，アプリケーション用の内部チャネルおよび SIP プロトコル[2] を用いた外部チャネルの両方が利用可能である．また，ネゴシエーション自身は SDP プロトコル[3] のオファー／アンサーモデル[4] に基づいて行うため，既存システムとの親和性や運用性を損なわずに動的な機能拡張を行うことができる．さらに，FreeNA によってもたらされる拡張機能のコンポーネント性と，SDP プロトコルの更新メカニズムを応用することによって，通信品質やマシン使用状況などの変化に応じて拡張機能を適応的に再構成することも可能になる。

2. 従来システムの概要

本章では，従来システムである FreeNA について，その概要と課題点について述べる．

FreeNA はプロキシサーバ，仮想マシン，API といった類のものではなく，対象アプリケーションと同一のプラットフォーム上で動作するネイティブのプログラムであり，一種のミドルウェアシステムとして動作する．FreeNA は，プラットフォーム固有の詳細を隠蔽し，抽象化された API/ABI をユーザに提供すると共に，アプリケーションが発行した API をフックすることによって拡張機能を透過的に挿入している．FreeNA は任意数の拡張機能をサポートしており，ユーザはアプリケーションごとに用意された設定ファイルに追加機能名およびパラメタを記述することによってその機能を利用することができる．

このように，従来の FreeNA では，アプリケーションに追加する機能はあからじめ設定ファイルに全て記述しておく必要があり，しかも通信を行うエンド間で同一の機能設定がなされていなければならないという制約があった．文献[5] では，クライアント側のアプリケーションが使用するトランスポート層プロトコルに応じて，サーバ側が使用するトランスポート層プロトコルを変更するという仕組みを実現しているが，より上位層の機能については，依然として静的な設定を行う必要があった．そこで本論文では，エンドアプリケーション同士が通信を開始する際に，FreeNA 同士がネゴシエーションを行うことによって，拡張機能を自律的に決定できるようにした．これにより，FreeNA はユーザが示した拡張機能の一覧を元にして, エンド間で共に利用可能な機能を見つけ出すことが可能になる．また，このネゴシエーション機能と拡張機能の動的構成技術を応用することにより，通信開始後においても，ネットワーク環境の変化などに応じて適応的に拡張機能を再構成することも可能になる．

3. FreeNA によるネゴシエーション機能

本章では，FreeNA によるネゴシエーションの仕組みについて，ネゴシエーション開始のタイミングと，ネゴシエーションの際に使用する通信チャネルについて述べる．

FreeNA によるネゴシエーションを行うことによって，拡張機能を動的に構成することが可能になるが，FreeNA の機能を既存システムへ幅広く導入するためには，FreeNA を導入していない相手ともうまく通信できるようにしなければならない．すなわち，相手が FreeNA を導入している場合はネゴシエーションを行ってアプリケーションの機能拡張を行い，そうでない場合はネゴシエーションを行わず，機能拡張もせずに通常通りの通信を行うという具合である．そこで，アプリケーション用の通信コネクションを確立する前に，通信相手が FreeNA を導入しているか否かを判断し，導入している場合には FreeNA によるネゴシエーションを行うという仕組みにすると都合がよい．

3.1 ネゴシエーション開始のタイミング

通常，TCP を用いるネットワーク型アプリケーションは，通信を開始する際に `connect()/accept()` ソケット関数を呼び出すため，FreeNA によるネゴシエーションはこれらの関数が実行される直前に行えばよい．第 2 章で述べたように，FreeNA はアプリケーションが発行した API/システムコールを捕捉することで機能追加を行っているため，本研究では `connect()/accept()` をフックし，アプリケーションの処理を一時中断させた段階で FreeNA によるネゴシエーションを開始するという方法を取っている．図 1 に，ネゴシエーションを開始するまで，およびアプリケーションによる通信開始までの流れを示す．

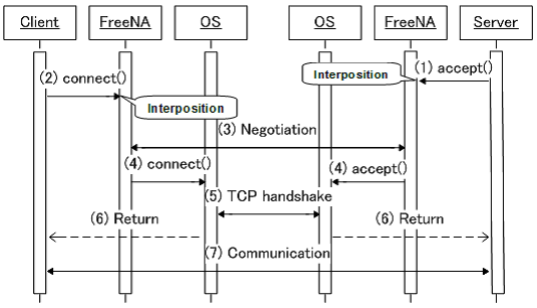

図 1 ネゴシエーション実行のタイミング
Fig. 1 Timing of conducting a negotiation

(1)，(2) では，エンドアプリケーションがそれぞれ `connect()/accept()` を発行し，FreeNA によってそれが捕捉される．そして，(3)FreeNA 同士でのネゴシエーションが行われ，アプリケーションに追加する拡張機能を決定する．次に，(4)FreeNA によって捕捉されていたソケット関数呼び出しが OS に伝わり，(5) アプリケーション間で通信を行うためのコネクションを確立する．そして，(6) コネクション確立の結果が FreeNA を経由して元のアプリケーションへと返る．最後に，(7) 拡張された機能を利用して，アプリケーション同士で本来の通信を行うという流れになっている．

なお，コネクションレス型のアプリケーションを用いる場合は，`connect()/accept()` は通常利用されないため，アプリケーションが最初に実行する送受信関数 (`sendto()/recvfrom()`) 関数を捕捉してから，FreeNA によるネゴシエーションを行えばよい．

3.2 ネゴシエーション用通信チャネル

FreeNA によるネゴシエーションは，エンドアプリケーション同士がコネクションを確立する前に行っておく必要があるため，アプリケーション間通信で使用するのとは別の通信チャネルが必要になる．ネゴシエーション用の通信チャネルとしてまず考えられるのは，アプリケーションが使用するポート番号と異なるポート番号を用いて TCP コネクションを新たに確立し，ネゴシエーションを行うという方法である．しかし，アプリケーションごとにネゴシエーション専用のポートを単純に用意するだけでは，システムの管理性やセキュリティなどの問題が生じてしまう．そこで本研究では，下記に述べる方法によってこれらの問題の解決を図っている．

1．外部チャネルでの SIP の利用

SIP はエンド間でセッションを確立するために使用するプロトコルであり，ユーザ管理機能，ロケーション機能，セッション管理機能，認証機能などを備えている．SIP を利用するノード同士は水平連携していて独立性があるため，他のプロトコルやサービスと連携することによって容易に機能を拡張することができる．例えば，SDP と併用することによって，セッション開始の際にエンド間でアプリケーションの機能確認を行うことが可能になる．

そこで，この SIP を利用して FreeNA のネゴシエーションを行うことにより，SIP URI を用いて相手にアクセスできるようになるため，外部チャネルに関する詳細 (ポート番号など) を知る必要がなく，またセッション用のメッセージ形式も新たに決める必要がないというメリットが生じる．

2．限定された環境下での外部チャネルの利用

SIP はセッション制御用のプロトコルとして既に標準的な地位を占めているが，SIP を利用しているネットワーク型アプリケーションはまだ少ないのが現状である．また，FreeNA を利用するためだけに新たに SIP を導入するのは実際的ではないため，例えばシステムの規模が十分に小さく，通信相手も限定されている場合は，単純に外部ポート番号を指定してコネクションを確立し，ネゴシエーションを行う方法が適している．

3．内部チャネルの利用

上記の二つの方法では外部チャネルを利用するため，FreeNA を利用するアプリケーションごとに別個のポート番号が必要になると共に，コネクションの確立も二重に行なわなければならなかった (コネクション型アプリケーションの場合)．本研究では，本来ならばアプリケーションが利用するポート番号を用いて，ネゴシエーションとアプリケーション通信を一つのチャネルで両立させると共に，FreeNA を導入していない通信相手とも整合性を失わずに通信を行う方法を提案し，3.3 節でその詳細を説明する．

3.3 内部チャネルによるネゴシエーション／アプリケーション通信の両用

内部チャネルを用いてネゴシエーションを行うためには，チャネル内を流れるデータがネゴシエーション用なのかどうかを判別できなければならない．しかしながら，FreeNA を導入していない，すなわちネゴシエーションを全く行わずにアプリケーション通信のみを行う通信相手も存在するため，単純にネゴシエーション専用のプロトコルを TCP/IP にトンネリングさせることはできない．したがって，特別なプロトコルを使用せずに相手が FreeNA に対応しているか否かを判断するための仕組みが必要である．

本研究では，IP プロトコルの IP レコードルートオプション[6] を用いてこの仕組みを実現している．IP レコードルートオプションは，IP パケットが経由したノードのアドレスを順に記録するために用いられる．そこで，FreeNA 自身をネットワーク上の 1 ノードとみなし，FreeNA のアドレスをレコードルートに記録することによって，IP パケットの受信者は送信者が FreeNA に対応しているかどうかを知ることができる．図 2 は，FreeNA が存在する場合におけるレコードルートを表している．TCP コネクション確立の際に，FreeNA は自身のアドレスとしてホストアドレスをレコードルートに記録するため，レコードルートの一番目および二番目のアドレスは同一になる．そこで，このような場合にはパケットの発信者が FreeNA に対応していると受信者が想定することにより，通信相手が FreeNA に対応しているかどうかを透過的に判断することが可能になる．

4. ネゴシエーションによる拡張機能の決定

本章では，FreeNA によるネゴシエーションの内容に焦点をあて，アプリケーションに追加する拡張機能をどのように決定していくかについて説明する．

前章ではネゴシエーションで使用する通信チャネルについて述べ，SIP や内部チャネルを利用する方法を説明したが，ネゴシエーションの内容自体はどの通信チャネルを用いても同じものになる．本研究では，アプリケーションが有する機能性を十分に表現でき，かつ拡張可能な設計になっているセッション記述用プロトコルである SDP を用いてネゴシエーションを行う．元々SDP にはネゴシエーション機能は含まれていなかったが，オファー／アンサーモデルの登場によって，セッションで使用する機能を SDP を用いてネゴシエーションすることが可能になった．現状では，SDP は主にマルチメディアセッションの情報を記述するために

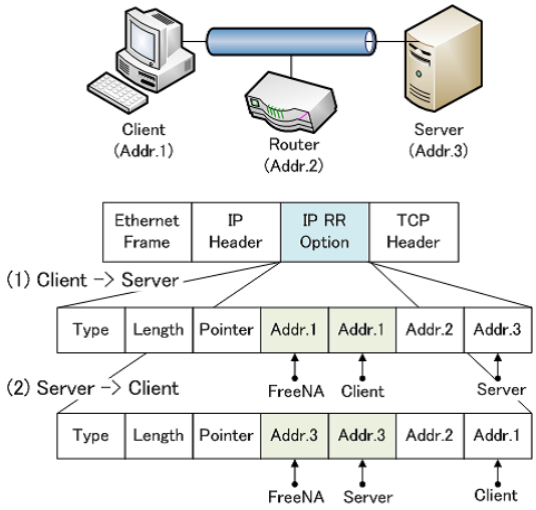

図 2 IP レコードルートを利用した FreeNA 存在性の判別
Fig. 2 Determination of FreeNA existence with IP RR

使用されており，フォーマットも厳密に規定されているが，いくつかの属性値を利用することによってユーザ独自のフォーマットを追加することが可能である．

そこで本研究では，この特徴を活かして SDP を図 3 のように拡張した．

```
(1)v=<version number>
(2)o=<user name> <session id> <session version>
   <nettype> <addrtype> <unicast addr>
(3)s=<session name>
(4)c=<nettype> <addrtype> <connection addr>
(5)t=<start> <stop>
(6)m=application <port no.> SERVICE/<service name>
   <essentiality>
(7)a=fmtp:<parameter name> <parameter value>
```

図 3 FreeNA ネゴシエーションのための拡張 SDP
Fig. 3 Extended SDP for FreeNA's negotiation

(1)-(5) に関しては従来の SDP と変化はないが，(6) ではメディアの種類を表す代わりに，アプリケーションに追加する拡張機能の種類を表すようにした．Essentiality はその拡張機能の必要度を表し，**required**(必須)，**selectively-required**(選択必須)，**option**(任意) を指定できる．(7) は (6) で指定した拡張機能のパラメタを指定し，一行につき一組のパラメタ名，パラメタ値を記述できる．

4.1 拡張 SDP を用いたオファー／アンサーモデル

拡張された SDP においても，オファー／アンサーの形式は従来通りであり，セッションの開始側 (Offerer) が対応可能な機能をオファーメッセージとして

SDP で示し，受け入れ側 (Answerer) がその SDP 中から利用可能なものを選択するというアプローチを取る．FreeNA によるネゴシエーションでは，基本的に先述の essentiality を満足させる機能選択を行うことがネゴシエーションの目標になる．つまり，開始側が **required** と記した機能は受け入れ側も必ずサポートしなければならない．また，**selectively-required** と記された同一名の拡張機能が複数指定されている場合は，受け入れ側は利用可能なものを必ず一つ選択する必要がある．これらの条件が満たされない場合は，FreeNA による機能拡張は行われない．本来の SDP によるオファー／アンサーモデルでは，受け入れ側が満足な回答を行えない場合は，開始側は SDP 記述を変更して再度オファーを試みるが，FreeNA によるネゴシエーションでは，初回のオファーが受け入れられない場合はネゴシエーションを中断し，FreeNA による機能拡張を行わない仕組みになっている．

```
(1) v=0
(2) o=clt 844526 844526 IN IP4 clt.example.com
(3) s=-
(4) c=IN IP4 clt.example.com
(5) t=0 0
(6) m=application 10000 SERVICE/CRYPTO
    selectively-required
(7) a=fmtp:algorithm AES
(8) a=fmtp:key_size 256
(9) a=fmtp:key_size 128
(10)a=fmtp:mode CBC
(11)m=application 10000 SERVICE/CRYPTO
    selectively-required
(12)a=fmtp:algorithm DES
(13)a=fmtp:key_size 128
(14)a=fmtp:mode CBC
(15)m=application 10000 SERVICE/COMPRESSION
    optional
(16)a=fmtp:algorithm Deflate
```

図 4 拡張 SDP によるオファーメッセージ
Fig. 4 Offer message with extended SDP

図 4，5 は，拡張 SDP によるオファー／アンサーの例を表している．オファー時には，(6)(11)(15) でそれぞれ開始側が利用したい拡張機能が記されている．(6) と (11) は同一の拡張機能を指しているが，後続行で示すパラメタ値が異なっている．また，**selectively-required** が指定されているため，受け入れ側は (6) か (11) のどちらかの設定を選択しなければならない．

アンサー時には，オファー内容が受け入れられた場合はその内容をそのまま記述し，オファーを拒否する場合は **m** 属性のポート番号を 0 にする．図 5 では，

```
(1) v=0
(2) o=svr 844564 844564 IN IP4 svr.example.com
(3) s=-
(4) c=IN IP4 svr.example.com
(5) t=0 0
(6) m=application 8000 SERVICE/CRYPTO
    selectively-required
(7) a=fmtp:algorithm AES
(8) a=fmtp:key_size 256
(9) a=fmtp:mode CBC
(10)m=application 0 SERVICE/CRYPTO
    selectively-required
(11)a=fmtp:algorithm DES
(12)a=fmtp:key_size 128
(13)a=fmtp:mode CBC
(14)m=application 0 SERVICE/COMPRESSION
    optional
(15)a=fmtp:algorithm Deflate
```

図 5 拡張 SDP によるアンサーメッセージ
Fig. 5 Answer message with extended SDP

(6) にある通り，AES を用いる CRYPTO 機能は受け入れられており，DES アルゴリズムの使用は拒否されている (10)．オファー時には，AES の鍵長として 256 ビットおよび 128 ビットが提案されていたが (8)(9)，アンサー時には 256 ビットの鍵長が受け入れられている (8)．また，オプションとして圧縮アルゴリズムの使用も提案されていたが (15)，受け入れ側はそれを使用しないと示している (15)．

上記の例では，オファー時における essentiality の条件を満たしているため，ネゴシエーションは成功となり，続くアプリケーション間の通信では 256 ビット鍵, CBC モードを用いる AES 暗号機能を使用することになる．

4.2 通信中における拡張機能の更新

セッション確立後に再度 SDP のオファー／アンサーを行うことにより，セッション内容を更新することができる．本研究では，この特徴を利用して拡張機能を適応的に再構成するための仕組みを提案する．いったんアプリケーション同士の通信が始まったとしても，端末の移動などによってネットワーク環境が変化することが考えられる．そこで，環境が大きく変化した時に再度ネゴシエーションを行うことにより，適応的に拡張機能を再構成を行うことができる．

5. 関 連 研 究

MetaSocket[7] は，既存 Java アプリケーションに対して透過的に機能追加を行うためのフレームワークであり，DM と呼ばれる意思決定コンポーネントがエンド間で協調することにより，例え通信中であっても，

規定の条件を満たした場合に機能の再構成を行うことができる．しかし，再構成の内容や条件判断に関してはユーザ自身がプログラムで記述しなければならないため，提案方式と比較して管理性や拡張性という面で劣っている．

Lee[8] らは，Java のリフレクション機能を活用し，プロトコルスタックの状態を厳密に管理することによって，通信中であっても透過的にプロトコルスタックを再構築することができるフレームワークを提案している．しかし，このフレームワークはエンド間での協調機能を実現しておらず，プロトコルスタックの再構築はローカル内で行われるため，プロトコルの種類そのものを変更することはできず，実装の変更しか行うことができない．

Alonistioti[9] らは，プロトコルスタックを抽象化し，プロトコルの実装をメタデータで管理することによって，無線 LAN や 3G 通信といった通信媒体に応じてプロトコルスタックを切り替えるためのアプローチを提案している．プロトコルの切り替えは，CBCM と呼ばれるコンポーネントがメタデータを検索し, 最適なプロトコルコンポーネントをプロトコルスタックにバインディングすることによって行われる．しかしながら，設計上はプロトコルの実装だけでなく，プロトコルの種類そのものも変更できるようになっているが，エンド間でどのように同期を取るかについては述べられていない．

Rütti[10] らは，分散システム内のあるノードがプロトコルを変更する際には，分散システム全体でグローバル同期を取らなければならないという問題について言及し，その解決方法を示している．当該方式では，変更するプロトコルに対する呼び出しをいったん補足し，グローバル同期を取った状態で変更を行っている．しかし，変更内容についてノード間でネゴシエーションすることはできないため，あらかじめ決められたプロトコルの変更しか行うことはできない．

6. ま と め

本論文では，これまで筆者らが提案してきた透過的なアプリケーション拡張用のソフトウェア実行環境である FreeNA の問題点を示すと共に，FreeNA にネゴシエーション機能を導入して動的な機能選択を行うための仕組みを新たに提案した．

FreeNA によるネゴシエーションはエンドアプリケーション同士による通信が行われる前に実施され，その通信チャネルとしては，SIP による外部チャネルを利用する方法や，IP レコードルートオプションによる内部チャネルを利用することが可能である．ネゴシエーション自身は FreeNA 用に拡張された SDP オファー／アンサーモデルに基づいて行われ，FreeNA 同士がエンド間で自律的に拡張機能を決定することができる．また，ネットワーク環境が変化した場合に再度ネゴシエーションを行うことにより，環境の変化に対して適応的に機能を再構成することも可能になる．

今後の課題として，本方式を用いた場合の既存システムとの相互接続性を調査すると共に，拡張機能の再構成のための環境変化の条件を定量化し，実環境下においてその機能性および性能についての評価を行うことが挙げられる．

参 考 文 献

1) R. Kawashima, Y. Ji, and K. Maruyama, FreeNA: A Multi-Platform Framework for Inserting Upper-Layer Network Services, IEICE Transactions on Communication vol.E92-D no.10, Oct. (2009)
2) J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, SIP: Session Initiation Protocol, RFC3261 (2002)
3) M. Handley, V. Jacobson, C. Perkins, SDP: Session Description Protocol, RFC4566 (2006)
4) J. Rosenberg and H. Schulzrinne, An Offer/Answer Model with the Session Description Protocol (SDP), RFC3264 (2002)
5) 川島 龍太, 計　宇生, 丸山 勝巳, トランスポート層プロトコルフリーな通信環境を透過的に実現するフレームワークの開発, 電子情報通信学会論文誌 B vol.J92-B no.4, April (2009)
6) Postel, J., Internet Protocol, STD 5, RFC 791, Sep. (1981)
7) S.M.Sadjadi, et al., MetaSockets: design and operation of runtime reconfigurable communication services, *Software-Practice & Experience*, Vol.36, No.11-12, pp.1157-1178 (2006)
8) Y. Lee and R. Chang, Developing dynamic-reconfigurable communication protocol stacks using Java, *Software-Practice & Experience*, vol.35 no.16 pp.601-620 (2005)
9) N. Alonistioti, E. Patouni, and V. Gazis, Generic Architecture and Mechanisms for Protocol Reconfiguration, Springer, Mobile Networks and Applications, vol.11 no.6 pp. 917-934, Dec. (2006)
10) O. Rütti, P. T. Wojciechowski, A. Schiper, Structural and Algorithmic Issues of Dynamic Protocol Update, IEEE 20th International Parallel and Distributed Processing Symposium, Rhodes Island, Greece, Apr. (2006)