RL-009

マルチプラットフォームにおけるネットワーク機能の透過的拡張のためのフレームワークの開発

A Framework for Transparent Network Services Insertion on Multi-Platform Systems

川島 龍太† 計 宇生‡ 丸山 勝巳‡
Ryota Kawashima Yusheng Ji Katsumi Maruyama

1．まえがき

ネットワーク技術が発展・浸透するのに伴い，ネットワークアプリケーションは QoS，セキュリティ，あるいはモビリティなどの付加機能(以下，サービス)をサポートする必要性が生じている．しかし，これらのサービスを実装することによって，アプリケーションの開発が必要以上に複雑化してしまうという問題がある．さらに，これらのサービスをサポートしない旧式システムも未だ数多く利用されているため，新規システム開発の際には既存システムとの互換性にも気を配らなければならない．

セキュリティなどのサービスの多くがアプリケーションのコア機能から独立している点を考慮すると，アプリケーションがこれらを適応的に利用できるような仕組みを持つことが好ましい．つまり，例え開発完了後であっても，再度修正することなくサービス追加・削除・更新などを必要に応じて行えるような仕組みが必要である．

本論文では，既存のネットワークアプリケーションに対して，必要なサービスを透過的に追加するためのフレームワーク(FreeNA)を提案する．FreeNA は API や ABI(Application Binary Interface)といったプラットフォーム依存の詳細を隠蔽し，統一化された抽象インタフェースを提供する．FreeNA により，利用者は単純な設定ファイルを用意するだけで既存アプリケーションを容易に拡張することができる．現在，FreeNA は Windows および Linux 上で利用可能である．また，FreeNA の性能評価を行ったところ，透過的なサービス追加によって生じる性能低下は，アプリケーション内に直接サービスを追加した場合と比較して，最大で 1-2%程度であった．

2．関連研究

これまでにも数多くの類似システムが提案されてきた．例えばソースコードレベルでサービスを挿入するシステムなどが存在する[1][2]．しかし，システムの利用者には対象アプリケーション・システムに対する高度な知識が要求され，さらにはアプリケーションのソースコード一式も必要になる．

専用のローダ／リンカや API を用いてランタイム時にサービスの追加を行うシステム[3][4]も提案されているが，サービス追加の仕組みがプラットフォーム依存であるため，複数のプラットフォーム上にシステムを展開できないという問題がある．ライブラリのプリロード機能を利用することも可能[5]であるが，メモリアクセス等によるアプリケーションの詳細な制御は困難である．

† 総合研究大学院大学 複合科学研究科, SOKENDAI
‡ 国立情報学研究所, NII

3．提案システム FreeNA の特徴

3.1 特徴

まず，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，次のような特徴を備えている．

(1)多目的フレームワーク
Web アプリケーション，モバイルアプリケーション，ストリーミング，ネットワーク制御プログラムといった多種多様なプログラムが様々な目的で FreeNA を利用することができる

(2)プログラミング言語独立
対象アプリケーションの実装言語を問わない

(3)マルチプラットフォーム
FreeNA は様々なプラットフォーム上で動作するように設計されており，現在は Windows および Linux 上で動作する

(4)ユーザ指向
ユーザは，FreeNA が提供する操作用コマンドや設定ファイルを用いて，任意のアプリケーションに対して自由にサービス機能を選択・追加することができる

(5)柔軟性
ユーザは，アプリケーションが使用する通信フローごとに挿入するサービス機能を指定することができる

(6)拡張性
サービス機能は通常の共有ライブラリとして実装されるため，ユーザは，第三者が開発したサービスを自由にアプリケーションに追加することが可能である

3.2 想定利用者

利用者は FreeNA を用いることで，再コンパイル／リンクすることなく，アプリケーションの機能拡張を図ることが可能になる．機能拡張は設定ファイルをベースにして行われるため，開発者や研究者だけではなく，対象システムに関する深い知識を持たない一般の利用者も FreeNA の想定利用者に含まれる．

3.3 サービス機能例

FreeNA によって，以下のようなサービス機能を実現することができる

(a) QoS 制御
フロー制御，ポリシー機能，データ圧縮

(b) セキュリティ
暗号化，認証，パーソナルファイアウォール

(c) モビリティ
セッションマイグレーション，ハンドオフ

(d) オーバーレイ
　アプリケーション層ルーティング
(e) その他
　パケットモニタリング，プロトコル変換

FreeNA の開発目標は，多様なプラットフォーム上で動作するアプリケーションに対して，上記のようなサービス機能を透過的に追加するための仕組みを実現することである．とりわけ，本目的の実現に当たって既存アプリケーションやカーネルの改変は一切行わないようにする．

4. アーキテクチャ

FreeNA は，エンドアプリケーション間の通信パスにネットワーク付加機能を挿入することによって透過的な機能拡張を実現している．例えば，アプリケーションがデータパケットを送り出すと，そのデータパケットは自ホストの OS に渡される前に FreeNA によって追加の処理が施される．同様に受信側では，OS が処理した受信パケットは，アプリケーションに渡される前に FreeNA によって適切な処理が行われる．

4.1 システム構成

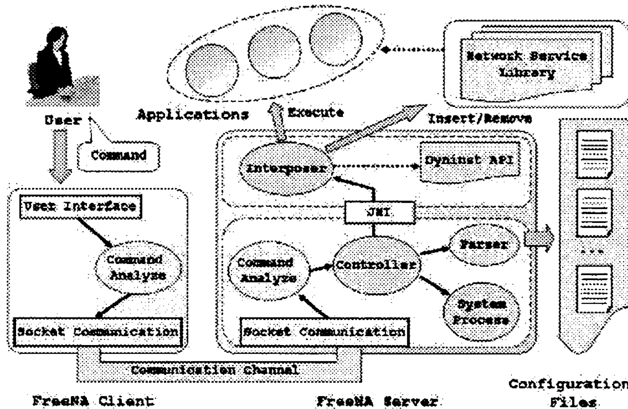

図 1 FreeNA のアーキテクチャ概要図

図 1 は，FreeNA のアーキテクチャの概要図を表している．FreeNA は，サービス挿入を実施する FreeNA Server，サーバに指示を送る FreeNA Client，利用するサービスやパラメタなどを指定する設定ファイル，サービスライブラリによって構成されている．

4.2 設定ファイル

次に，アプリケーションごとに用意される設定ファイルの例を図 2 に示す．Service タグには，アプリケーションに追加するサービスライブラリ，パラメタなどを指定する．Rule タグには，指定したサービスを適用する通信フロー条件(プロトコル，ポート番号，接続方式)を記述する．このサービス挿入ルールには，グローバルルールとローカルルールの二種類があり，ローカルルールは単一のサービスに対してのみ適用され，グローバルルールよりも優先される．

```
<?xml version="1.0" encoding="Shift_JIS"?>
<configuration application="ftp_server">
  <services>
    <service name="crypto" lib="libcrypto.so">
      <parameter name="algorithm" value="AES"/>
      <parameter name="key_size"  value="128"/>
      <parameter name="mode"      value="CBC"/>
      <rule use="true" service="ftp-ctl" transport="tcp"
      port="21" type="client"/>
    </service>
    <service name="compression" lib="libcomp.so">
    </service>
  </services>
  <using-rules>
    <rule use="true" service="ftp-data" transport="tcp"
    port="20" type="client"/>
    <rule use="false" service="*" transport="*"
    port="*" type="*"/>
</using-rules>
</configuration>
```

Service
Local Rule
Global Rule

図 2 設定ファイルの例

4.3 サービスライブラリ

サービスライブラリ内には，ソケット関数に対応するサービス関数群が含まれている．例えば，圧縮サービスライブラリには，圧縮機能付き送信関数と伸張機能付き受信関数が備わっているはずである．また，前述したように，サービスライブラリは通常の共有ライブラリとして作成されるため，第三者が実装したサービス機能をユーザが容易にアプリケーションに取り込むことができる．

5. 実装

5.1 サービス機能の挿入方法

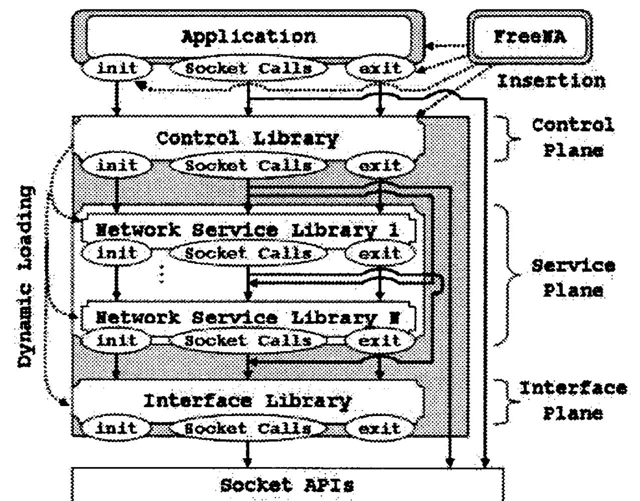

図 3 サービスライブラリの階層化構造

図 3 にあるように，FreeNA は，アプリケーションとソケット API との間に階層化されたライブラリ群を挿入する． Control Plane は，設定ファイルを基にしてライブラリ群の関数呼び出しグラフを構築する． Service Plane は，ネットワークサービス機能を提供する． Interface Plane は，サービスライブラリとプラットフォーム固有のソケット関数との橋渡しを行う．
図 4 は，FreeNA 利用時のプロセスイメージの概略と実行の流れを示したものである．まず，FreeNA は対象アプリケーションを子プロセスとして起動し，次にサービス制

御用の制御ライブラリを設定ファイル情報と共に挿入する．この時，制御ライブラリが提供する init/exit 関数へのCALL 命令がアプリケーションのエントリ関数(main)内に埋め込まれる．これらの関数は文字通りライブラリの初期化・終了処理を行う．次に，アプリケーションのソケット関数呼び出し先を制御ライブラリ関数へと置き換える．これらの処理は Dyninst API[6]を用いて行われる．そして，制御ライブラリは利用するサービスライブラリ群を動的に読み込む．このような処理を行うことによって，マルチプラットフォーム上でサービスを透過的に挿入することが可能になる．

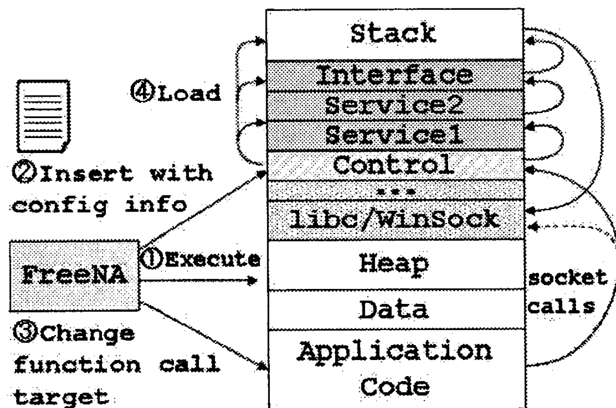

図4 FreeNA利用時のプロセスイメージ概略図

実行時には，制御ライブラリは，設定ファイル内のグローバルルールおよびローカルルールに基づいて，ソケットごとにどのサービス機能を利用するか判断を行う．サービス機能を利用すると判断した場合，制御ライブラリは下位層のサービスライブラリ関数を直接実行する．そうでなければ，インタフェースライブラリが提供するラッパー化されたソケット関数を実行する．下位層ライブラリの関数呼び出しには，図5に示す service_info 構造体を利用する．service_info 構造体は制御ライブラリによって生成され，前述の init 関数を通じて下位層ライブラリに渡される．

```
/* Each library has an instance of service_info */
struct service_info
{
  struct service_info *next;  /* Next node*/
  char **params;        /* Parameter names */
  char **values;        /* Parameter values */
  int num_of_params; /* Number of parameters */

  /* Function pointers of the downstream library */
  void (*service_init)(const struct service_info*);
  void (*service_exit)(void);
  SOCKET (*service_socket)(int, int, int);
  ...
};
```

図5 service_info 構造体

5.2 サービスライブラリの実装

サービスライブラリは，C/C++によって実装された通常の共有ライブラリである．ライブラリ開発者は，ソケット関数と同等のインタフェースを持つサービス関数群を実装する．各関数内では，下位層ライブラリの関数を呼び出すために，service_info 構造体を利用する(図6)．また，ライブラリ開発者は実装したサービス関数名を列挙した export ファイルを作成する必要がある．この export ファイルはサービス関数呼び出しグラフを構築する際に FreeNA Server によって利用される．したがって，ライブラリ開発者は必ずしも全てのソケット関数に対応するサービス関数を実装する必要は無い．

```
struct service_info *info;     /* Global Variable */

void service_init( struct service_info *i )
{
    info = i;  /* Preserve the service_info */
    ...
    info->service_init( info->next );
}
void service_exit(void)
{  ...
    info->service_exit();
}
SOCKET service_socket(int af,int type,int proto)
{  ...
    sock = info->service_socket(af, type, proto);
    ...
    return sock;
}
...
```

図6 サービスライブラリのテンプレートコード

6. 評価

筆者らは，4，5章のアーキテクチャを基に FreeNA の実装および評価を Linux／Windows 上にて行った．筆者らは，FreeNA によるサービス挿入のオーバヘッドを調査するために，以下の三種類の評価実験を行った．実験では，ギガビットイーサに接続された2台のマシン上で，デュアルインストールしてある Linux および Windows の両 OS を用いてそれぞれ行った．また，比較のためにアプリケーション内に直接サービスを実装した場合についても測定を行った．

(1) 軽量サービス挿入時の送信側オーバヘッド

この実験では，1から5個 Null サービスライブラリを挿入したアプリケーションが 30 万個のユーザパケットを送信するのに掛かる時間を評価した．Null サービスライブラリとは，下位層ライブラリ関数呼び出し以外の実際的な処理は何も行わないライブラリである．表1の結果を見ると，OS 間での性能差はあるものの，FreeNA によってサービスを挿入した場合とアプリケーションが直接サービスを呼び出す場合とでは有意な性能差は見られない．

	Number of Null services				
[Linux]	1	2	3	4	5
FreeNA	2.635	2.638	2.635	2.637	2.64
Direct	2.629	2.635	2.636	2.637	2.636

[Windows]	1	2	3	4	5
FreeNA	10.105	10.104	10.12	10.121	10.095
Direct	10.105	10.076	10.122	10.097	10.141

表 1 Null サービス使用時の送信時間 [s]

(2) 重量サービス挿入時の送信側オーバヘッド

次に，暗号化サービスライブラリと圧縮サービスライブラリを用いて(1)と同様の実験を行った．暗号化アルゴリズムには Sosemanuk，圧縮アルゴリズムには Deflate を用いている．表 2 の結果を見ると，やはり FreeNA を用いたサービス挿入に掛かるオーバヘッドは極めて小さいことがわかる．

[Linux]	Crypto	Compress	Crypto+Comp
FreeNA	2.641	26.242	26.785
Direct	2.641	26.197	26.721

[Windows]	Crypto	Compress	Crypto+Comp
FreeNA	10.404	23.574	24.141
Direct	10.389	23.625	24.128

表 2 暗号化・圧縮サービス使用時の送信時間 [s]

(3) 実際面から見た受信側オーバヘッド

次に，FreeNA を用いて FTP アプリケーションに暗号化サービスおよび圧縮サービスを挿入した際の実効スループット結果を図 7，8 に示す．なお，暗号化サービスはデータ用コネクションと制御用コネクションに，圧縮サービスはデータ用コネクションのみに適用している．実験結果を見ると，いずれの OS を用いた場合でも，アプリケーションに直接サービス機能を実装した場合と FreeNA によってサービスを挿入した場合との間に有意な性能差は見られなかった(性能低下は最大で 1-2%程度)．また，Windows 上では所々にスループットが急激に低下するポイント(データサイズ)が見られるが，これは Windows の実装によるものと考えられる．

7. まとめ

本論文では，さまざまなネットワークサービスを既存システムに対して透過的に追加するためのマルチプラットフォームフレームワーク，FreeNA を提案した．現在，FreeNA は Windows および Linux 上で利用可能である．

また，筆者らが実装したシステムを用いて性能評価を行ったところ，FreeNA を用いてサービスを透過的に追加した際に生じる性能低下は，アプリケーション内に直接サービスを追加した場合と比較して，最大で 1-2%程度であった．

筆者らは今後 FreeNA を拡張し，マシン状況やネットワーク状況に応じて動的にサービスを追加・削除するための仕組みを新たに導入する予定である．この機能により，状況に応じた最適のサービスが利用できるようになるため，アプリケーションの実効性能が向上するものと考えられる．

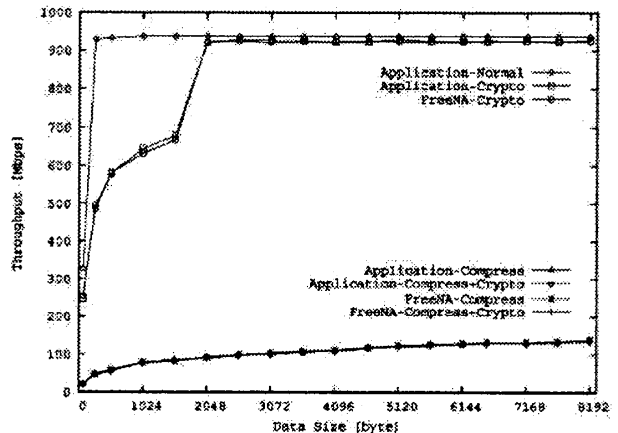

図 7 Linux 上での実効スループット

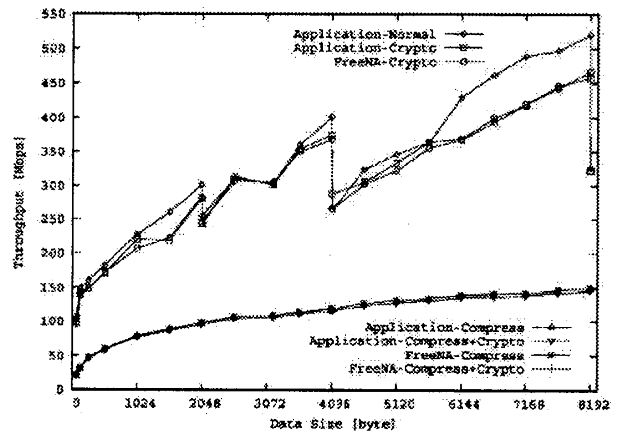

図 8 Windows 上での実効スループット

参考文献

[1] 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.

[2] 柳澤 佳里 他, "OS カーネル用アスペクト指向システム KLASY", 情報処理学会論文誌, Vol.48, No.SIG_10(PRO_33) pp. 176-188, 2007

[3] A.Serra, et al., "DITOOLS: Application-level Support for Dynamic Extension and Flexible Composition", Proc. of 2000 USENIX Annual Technical Conference, CA, USA, 2000.

[4] J.Salz, et al., "TESLA: A Transparent, Extensible Session-Layer Architecture for End-to-end Network Services", Proc. of USITS'03: 4th USENIX Symposium on Internet Technologies and Systems, WA, USA, 2003.

[5] Paul Z, et al., "Mesh: Secure Lightweight Grid Middleware Using Existing SSH Infrastructure", SACMAT'07, pp111-120, Sophia Antipolis, France, 2007.

[6] B.Buck, et al., "An API for Runtime Code Patching", International Journal of High Performance Computing Applications, Vol.14, No.4, pp.317-329, 2000.