Design and Implementation of Multi-Platform Infrastructure of Extensible Networking Functions

Ryota Kawashima
School of Multidisciplinary Sciences
Department of Informatics
The Graduate University for Advanced Studies (SOKENDAI)
Tokyo, JAPAN
Email: kawa@nii.ac.jp

Yusheng Ji
Information Systems Architecture
Research Division
National Institute of Informatics (NII)
Tokyo, JAPAN
Email: kei@nii.ac.jp

Katsumi Maruyama
Information Systems Architecture
Research Division
National Institute of Informatics (NII)
Tokyo, JAPAN
Email: maruyama@nii.ac.jp

Abstract—Dynamic and flexible composition of higher-level network services, such as security, QoS, or adaptive services are required by future network applications. However, the development of such extensible applications makes them rather complex. In addition, many old applications, which do not support such services, would stick to be used. To solve these problems, we propose a generic and multi-platform infrastructure called FreeNA\(^1\) that extends existing applications by transparently incorporating the services to them. FreeNA offers abstract interfaces such that users can insert the services into each packet flow based on a configuration file. In this paper, we describe the design and implementation of FreeNA including a functionality comparison with relevant systems, and our performance evaluation results. The result shows that FreeNA offers finer configurability, composability, and usability and can be used widely than other similar systems. We also show that overhead of transparent service insertion is about 1–2% at a maximum compared to a method of inserting such services into applications directly.

I. INTRODUCTION

Dynamic and flexible composition of higher-level network services, such as security, QoS, or adaptive services for user requirements and network status are required by future network applications. However, the development of such extensible applications makes them rather complex. In addition, many old applications, which do not support such services, would stick to be used. Therefore, the ease in development or maintaining the interoperability of applications are big problems in current network applications.

In order to develop enhanced network systems and network applications, we have developed a extensible infrastructure called FreeNA, which enables existing applications to be transparently inserted into various network services. FreeNA hides the platform-dependent issues like API and ABI (Application Binary Interface), and also offers a unified abstraction interface to its users. We have developed FreeNA on both Linux and Microsoft Windows operating systems so far. FreeNA enables users to insert arbitrary services into each packet flow and choose their parameters based on a configuration file.

So far, many related systems are proposed for the same purpose as in [1][2][3][4][5], however, these systems have some restrictions on platform, developer-oriented behavior, usability, and flexible service composition. Compared to similar systems, FreeNA is designed to overcome these drawbacks.

In this paper, we describe the design and implementation of FreeNA and also introduce a functionality comparison with other relevant systems, and performance evaluation results. The results show that FreeNA offers finer configurability, composability, and usability and can be used widely than other similar systems. We also show that overhead of transparent service insertion is about 1–2% at a maximum compared to a method of directly inserting such services into applications.

II. RELATED WORK

There have been a number of proposed systems that were related to FreeNA. Here, these systems are categorized by their service insertion types.

• Source code level insertion

Source code level insertion is enabled by Aspect-Oriented Programming (AOP) or similar methods. Such works done by J. Zhang et al.[6] and MetaSockets[1]. However, the users of these systems are largely limited to developers and can only use AOP languages that correspond to the implementation language of the application. Moreover, the users must have the source code of the application.

• Runtime System call Interposition

This type of interposition is done by dedicated system calls, customized linker/loader, and library preloading techniques. Such works done by Interposition Agents[2], DTOOLS[3], TESLA[4], and Mesh[7]. However, their mechanisms are mostly platform-dependent, or do not provide enough functions, such as the fine configurability or arbitrary memory access.

• Service Insertion with Kernel Support

A protocol composable mechanism within the kernel is beneficial in view of the flexibility and efficiency. Such works done by x-Kernel[8] and Streams[9].

\(^{1}\)a FRamework for Extending Existing Network Applications
However, these are not major ones nowadays (they are available on Linux in a limited way and not available on Windows).

- **Others**

Dyninst API[10] can change the runtime process image by dynamically instrumenting/removing the code and provide an unified, and also used by FreeNA internally.

A VTL framework[5] provides a multi-platform transparent network service insertion method, but it only supports applications running on virtual machines.

### III. BRIEF OVERVIEW OF FREE NA

**A. Characteristics**

First, FreeNA is not a proxy server, virtual machine, API, or libraries, but a normal program run on the host machine in which the target application also runs.

- **General-purpose Framework** FreeNA can be used with many different network applications, such as web applications, mobile applications, streaming applications, or network control programs for various purposes.

- **Programming-language Independence** FreeNA does not take into consideration what programming language is used for implementing the target application.

- **Multi-platform** FreeNA is designed to work on various platforms and currently runs on both Linux and Microsoft Windows OSs.

- **Low-overhead** The overhead produced by a service insertion is almost the same as the overhead for processing the service function itself.

- **User-oriented** Users can select the network services being inserted into a desired application by using a configuration file and FreeNA offers several commands for these operations.

- **Flexibility** Users can specify the services to be inserted into the applications, what packet flows which the services are inserted into, and the parameters of each service.

- **Extensibility** Network services can be developed by third parties as libraries, and the users can import the libraries into FreeNA.

**B. Supposed Users**

FreeNA can be used by users who have no profound knowledge on the target applications or networks. Network service components will be treated as only a patch and will be invisible to such users.

System/Network administrators can use FreeNA to extend their systems. Unlike regular users, they have to write a configuration for each application.

Usually, developers will test and modify their programs and parameters. FreeNA supports such processes by inserting the changeable modules and parameters into the application without any modification and recompile/relinking.

The goals of FreeNA are to transparently offer the network services to the applications run on a variety of major platforms, and also to provide unified and abstract interfaces to users.

Above all, the applications or kernels must not be modified at all for this purpose.

### IV. FREE NA ARCHITECTURE

FreeNA can be used to transparently add network service libraries to the communication path between end-applications. For instance, when an application (sender) sends a packet to the receiver, the packet is intercepted and processed by the service libraries inserted by FreeNA, then it is passed to the underlying OS. At the receiver side, the service libraries get the packet from the OS and execute the opposite process; then they pass the result to the receiver application.

Figure 1 shows the overall architecture of FreeNA and FreeNA is composed of several components. The roles of these components are described below.

**A. The FreeNA Client**

The FreeNA client provides an user-interface. The Users can use the client to access and control the FreeNA server that plays the main roles. The current user-interface is a command-based interface like a GNU debugger (GDB) or FTP client.

**B. The FreeNA Server**

The FreeNA server executes various requests from the client, such as run an application and stop the application. The request command is classified by a Command Analyzer, and a Controller executes the command by its type. A Parser parses the configuration file to get information concerning the network service libraries. The Controller instructs an Interposer to start the application and inserts the specified services.

**C. The Configuration File**

A configuration file must be prepared for each application to choose services and parameters. The structure of the file is shown in Fig. 2.

A service tag specifies the service library, its parameters, and the local using rules. A rule tag (local rule) is used to specify the packet flow type which the services are inserted into by a set of conditions, such as the transport protocol, port.
number, and application type (client/server). The rule tags also appear inside the using-rules tag. These rules (global rules) are applied to all services instead of just an individual service. Note that the local rules come before the global rules.

D. Network Service Library

The network service libraries contain service functions; for example, a compression service library may contain a compression send function and a decompression receive function. Although a service library has a special structure, which will be described later, it is basically a normal shared/dynamic library. Therefore, service libraries can be provided by third parties.

V. IMPLEMENTATION

A. Network Service Insertion

As can be seen in Fig. 3, FreeNA inserts hierarchical structures between the application and socket APIs. The Control plane controls the function call graph based on the configuration file. The Service plane offers the network service functions, and the Interface plane bridges the services and socket APIs of the platform.

Figure 4 shows a process image of the application with FreeNA. First, FreeNA launches the application and loads a Control library, then init/exit functions of the library are embedded into the entry function (main) before execution. The config information is also embedded as the argument of the init function. Next, FreeNA changes the function call target from the socket library to the control library. Underlying libraries are dynamically loaded by the control library at runtime and their functions are called via the control library.

In practice, FreeNA uses a Dyninst API[10] for embedding init/exit functions and rewriting function call instructions. The API enables FreeNA to handle the applications on a multi-platform and finely controls them.

At runtime, the control library decides whether the downstream libraries are used or not in view of the global or local rules for each socket. If it decides to use them, the control library calls the downstream service’s function using the service_info’s member function pointers (See Fig. 5).

B. The Network Service Library

The service library is provided as a shared library and implemented in C/C++. Library developers can implement the service functions within the appropriate socket functions, and the proprietary service_info structure is used to call the

```c
/* 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);
  ... 
};
```

Fig. 5. The service_info structure

978-1-4244-2324-8/08/$25.00 © 2008 IEEE.
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE "GLOBECOM" 2008 proceedings.
downstream library’s functions. Library developers also have to prepare an export file that contains the socket function name to be wrapped for each platform. The export file is used by the FreeNA server to compose the function call graph information. Therefore, developers do not have to write the wrapper functions of all the socket calls.

VI. EVALUATION

A. Functionality Comparison

MetaSockets[1], Interposition Agents[2], DITOOLS[3], TESLA[4], and VTL[5] are also general purpose frameworks like FreeNA.

First, FreeNA has no restriction on the programming language used to implement the application, but MetaSockets can only be used in Java applications. In particular, systems that are based on source code level service insertion have similar limitations.

FreeNA is designed as a multi-platform system and now operates on Linux and Microsoft Windows. Although MetaSockets and VTL also work on these platforms, VTL only targets applications run on VM monitors.

It is beneficial that non-developers can use the system by just selecting the component to insert. FreeNA provides a user-oriented client program and configuration files for service insertions, but many other systems do not offer such mechanisms. Although DITOOLS offers the configuration file, configurable items are few and not user-oriented.

A mechanism that allows users to insert network services separately (selective insertion) is imperative for advanced applications. FreeNA enables the selective insertion by using the global/local rules. Other systems, by default, do not support selective insertion. Even though a mechanism can be implemented within the service component itself, it incurs more complexity in the service libraries.

Table I presents a summarization of the functionality of each system. Usability is evaluated on whether the users can use the prepared service components without programming. Configurability denotes whether the system offers easily-configurable features. As you can see, FreeNA has more practical uses in that FreeNA can support many applications run on major platforms and users can specify various items depending on their usage.

B. Overhead Evaluation

We conducted three types of experiments to evaluate the overhead of service insertion by FreeNA. In the experiments, overhead was evaluated on both the dual-installed Linux and Windows OSs. Table II shows the experimental machine’s specifications. An applications that directly calls the service functions is also evaluated for comparison purposes. The test applications were written in C++ and compiled by GNU g++/Visual Studio.NET using the best optimization option.

1) Transmission Overhead with Lightweight Service: In this experiment, the application sends 300,000 user-data with a Null service library, which sends data merely without any processing. Each user-data is 1024 bytes and the time is measured during all the data is transmitted to the receiver.

Table III shows the time of the transmissions with various numbers of null service libraries. Although the times are different for different OSs, the time for FreeNA and for an application directly calls the service are almost the same (performance degradation was less than 2% at the most).

2) Transmission Overhead with Heavyweight Service: This experiment was conducted under the same conditions as the previous one except that a Cryptography service library and a Compression service library were used.

The measurement results are shown in table IV and indicate that FreeNA does not influence the performance of the target
TABLE III
TRANSMISSION TIME WITH LIGHT-WEIGHT SERVICE

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FreeNA</td>
<td>2.635</td>
<td>2.638</td>
<td>2.635</td>
<td>2.637</td>
<td>2.64</td>
</tr>
<tr>
<td>App-direct</td>
<td>2.629</td>
<td>2.635</td>
<td>2.636</td>
<td>2.637</td>
<td>2.636</td>
</tr>
<tr>
<td>Windows</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FreeNA</td>
<td>10.105</td>
<td>10.104</td>
<td>10.12</td>
<td>10.121</td>
<td>10.095</td>
</tr>
</tbody>
</table>

TABLE IV
TRANSMISSION TIME WITH HEAVY-WEIGHT SERVICES

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FreeNA</td>
<td>2.641</td>
<td>26.242</td>
<td>26.785</td>
<td></td>
<td></td>
</tr>
<tr>
<td>App-direct</td>
<td>2.641</td>
<td>26.197</td>
<td>26.721</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FreeNA</td>
<td>10.404</td>
<td>23.574</td>
<td>24.141</td>
<td></td>
<td></td>
</tr>
<tr>
<td>App-direct</td>
<td>10.389</td>
<td>23.625</td>
<td>24.128</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

application when using practical service libraries (performance degradation was less than 1%).

3) Overhead with practical usage: The third experiment was conducted using a file transfer application that uses two connections like FTP. The compression service was applied to the data connection and the cryptography service was applied to both the data and the control connections.

Figure 6 and 7 show the client’s throughput on Linux and Windows. In Fig. 6, the throughputs for FreeNA method and application-direct method were close for each service type. Next, on Windows, there was also no significant differences between both methods. However, there are some substantial performance drop points. This phenomenon may be caused by Windows’ implementation style.

From the experiment, we can say that there seems to be no significant influence of the service insertion by FreeNA for practical usage on both platforms.

VII. CONCLUSION

In this paper, we presented a multi-platform infrastructure called FreeNA, which allows users to insert network services into existing applications, and is currently available on Linux/Windows platforms.

We evaluated the functionality of FreeNA and conducted several experiments. As a result, FreeNA was able to enable more usability, portability, configurability, and practicality than other similar systems, and the overhead of a service insertion using FreeNA was about 1-2% at a maximum on both platforms when compared to that of an application-direct method.

We are going to extend FreeNA in order to dynamically insert or remove network services depending on the network conditions. This approach is expected to enhance the overall performance of the system since the most suitable services and parameters can be used under current network conditions.

REFERENCES