Poster: Adding Selection into Protocol Stacks – Communication with Choices

Yu-Shun Wang, *Student Member, IEEE*, Joseph D. Touch, *Senior Member, IEEE*, and John A. Silvester, *Senior Member, IEEE*

I. Introduction

Communication involves many choices. Take person-to-person communication as an example: one can talk to the other person directly, make a phone call on the land line or mobile phones, write a note and send it via email, fax, or postal mail. There are plenty of options. This flexibility is often lost as communication moves from the user layer down protocol stacks. The Flexible Stack Architecture (FSA) [7] aims to retrofit choices back into current protocol stack framework, and investigates how the resulting flexible infrastructure can benefit the overall development of communication architecture. This extended abstract describes the problems with existing protocol stacks, discusses the design principles of the approach, and provides an overview of the FSA framework.

A. Problem Statement

Choices in communication often come from diversity or redundancy in media and protocols. Choices can be utilized to achieve better fault tolerance, implement different types of security or quality-of-service policies, and help the adaptation and evolution of network architecture. The other side of choice is the capability to add, update, or remove entries for selection. The resulting architecture and implementation can then support dynamic, runtime plug-ins and detachments of protocols in a stack, selecting among different resolution mechanisms and the corresponding results. More sophisticated selection policies can support concurrency at both stack levels and protocol levels. This is essential in experimenting and adopting any new communication architecture. In short, various protocols and media present options on how to send a message. Resolutions of identities from one layer to another provide choices on where to send a message. Lastly, contents are another type of choice on what to send in a message, though the selection of content is not considered in this work.

The fundamental problem is that current stack architectures lack the notion of choices. Protocol order is fixed both architecturally and in implementations. There are no choices once a communication enters the stack. Some protocols or mechanisms implement their own selection internally, sometimes at multiple places within the same protocol layer. Those often result in multiple selection points that can interfere with each other. A good example is at the network layer where firewall, routing, and IPsec security policies all operate on the same IP protocol. This rigid architecture is also hard to extend, especially when adding new protocols into a stack or updating existing ones.

There are other flexible-stack-like proposals that break the hard links between protocols in a stack [1][3][4][5]. These systems basically provide mechanisms and the freedom to link one protocol to another, but none provides insight on what really constitutes a "layer" in communication and how protocols in one layer relates to another. FSA adds a structured framework to determine which and how protocols can be linked together, based on a template, MDCM, of what a communication layer does.

Other challenges for choices are performance and complexity. Traversing static links between protocols is always faster and more efficient than dynamic selection. Selection policies increase complexity. This is a tradeoff between capability and performance. Although tuning and caching mitigate the performance issue, minimizing the impact to better justify the new capabilities on choices will continue to be a basic objective of all flexible stack approaches, including FSA.

II. The Flexible Stack Architecture

The goal of FSA is to enable choices in the communication architecture. The methodology is to "retrofit" the existing stack framework based on a new template described below rather than an unstructured approach.

1) Template – MDCM

The framework of FSA is based on the Multi-Domain Communication Model, MDCM [6], which integrates selection and resolution into communication as follows:

(1) Perform protocol-specific processing
(2) Select the next protocol if not at the destination
(3) Resolve the corresponding source and destination in the new protocol
(4) Enter the new protocol

The purpose of the template is to categorize different "protocols." Protocols for message transmission should be separated from protocols for resolution and selection.

Y. Wang and J. D. Touch are with the University of Southern California Information Sciences Institute, Marina del Rey, CA 90292 USA, (Y. Wang, 310-448-8742, fax: 310-448-9300; e-mail: {yushunwa, touch}@isi.edu).

J. A. Silvester is with the Electrical Engineering Department, University of Southern California, Los Angeles, CA 90089 USA, (e-mail: silvester@usc.edu).

2) Resolution – Glue between Layers

Although any two random protocols can be stacked together, the communication can only work if one can resolve source and destination identities from one protocol domain to the other. The resolution function represents the mapping between the entities in the two protocol domains. Without a resolution function, it is impossible to translate the identities from the source domain to the corresponding identities in the target protocol domain; therefore, communication cannot continue in the target protocol domain. This is why the resolution function is the glue that connects (or stacks) one protocol to another. Note that the resolution of identities also determines the semantics of the identities involved in the mapping, but the following discussion focuses on the operational logistics.

3) Flexible Stack Architecture

The reference stack is loosely based on the network stack implementation of FreeBSD [2], an original BSD derivative, while the description will focus on high-level differences and additions only. The changes can be summarized into three parts. First it adds both selection (downstream) and update (upstream) databases for every protocol in the stack. The second is to replace the hard link between protocols with calls to a selection function and the subsequent resolution function to perform selection. Finally, it also needs mechanisms to initialize and maintain the selection and update databases. These will also support dynamically loading and unloading protocols of a stack during runtime. Figure 1 shows a simplified block diagram of relevant data structures and their relationships. The main protocol table represents an entire "stack" in the conventional sense. It is a list of all active protocols, similar to the *protosw* (protocol switch) array in BSD. Each protocol table entry points to a protocol structure that stores the information to perform protocol functions and to facilitate the transition to the next protocol in the table (stack). The main protocol table also provides a repository for instantiating new protocols or updating existing ones.

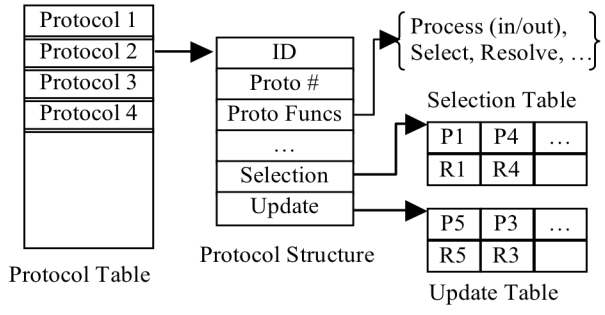

Figure 1. Flexible Stack Architecture

The selection table consists of all available downstream protocols that can be selected by the given protocol. Each selection table entry contains a protocol index, the corresponding resolution function, and flags or metrics (not shown) for selection. The selection table is analogous to a routing table. The update table of a protocol lists the upstream protocol domains that can select the given protocol. The update table contains the same information as the selection table. The protocol functions implement various aspects of protocol processing. FSA adds several functions for selection, resolution, and updating the two tables above.

4) Operations

Operations can be summarized into three parts: protocol instantiation, communication, and update. Instantiation is the same during system startup and loading a new protocol during runtime. The main task is to scan the entire protocol table to establish relationship between protocols by the availability of resolution functions. This must be done on the selection and update tables of both active protocols and the newly loaded one. Once a protocol is instantiated, the communication can utilize that protocol from any of its upstream protocols. If the message has not yet reached the destination after outbound processing, the select function will be invoked to choose the next protocol from the selection table. Once a next protocol is selected, the corresponding resolution function, included in the selection table, is used to resolve the new source and destination addresses in the next protocol. With this information, the communication is ready to enter into the next protocol domain. If the status of an active protocol changes, the framework must update its corresponding upstream and downstream tables of all protocols linked to the one in question.

III. Status and Future Work

The skeleton framework of FSA is implemented on the FreeBSD CURRENT platform. The prototype assumes the INET (IPv4) protocol domain as existing active protocols, and implemented IPv6, UDPv6, and ICMPv6 as kernel loadable modules (KLM). Selection and resolution functions are implemented in very rudimentary form. We are currently modularizing existing IPv4 protocol family, completing TCPv6 module, and adding some new and experimental protocols such as HIP, Shim6, and IPsec. More sophisticated selection functions and the selection of resolution functions are planned; the performance impact of this architecture is also being benchmarked.

References

[1] Braden, R., Faber, T., and Handley, M., "From Protocol Stack to Protocol Heap -- Role-Based Architecture", HotNets-I, October 2002; also in Computer Communication Review, Vol. 33, No. 1, January 2003.

[2] FreeBSD. http://www.freebsd.org

[3] Hutchinson, N. C., and Peterson, L. L., "The x-Kernel: An architecture for implementing network protocols," IEEE Transactions on Software Engineering, 17(1):64#76, Jan. 1991.

[4] Koler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F., "The Click modular router," ACM Transactions on Computer Systems 18(3), August 2000, pages 263-297.

[5] Tschudin, C., "Flexible Protocol Stacks," ACM Sigcomm 1991, pp. 197-205.

[6] Wang, Y, Touch, J, and Silvester, J., "A Unified Model for End Point Resolution and Domain Conversion for Multi-Hop, Multi-Layer Communication," ISI Technical Report ISI-TR-590, June 2004.

[7] Wang, Y., Touch, J., and Silvester, J., "Adding Selection into Protocol Stacks – Communication with Choices," submitted to Infocom 2007.