Towards Agile Network Operation with Layer 4.5 Protocol Customization

1st Daniel F. Lukaszewski
C.S. Department, Naval Postgraduate School
dflukasz@nps.edu

2nd Geoffrey G. Xie
C.S. Department, Naval Postgraduate School
xie@nps.edu

***Abstract*—Protocol customization takes different forms depending on the protocol specification, implementation layer, and desired end goal. Currently, these customizations are developed in isolation for a single protocol, resulting in duplicate low-level functionality across protocols and more importantly, elevated complexity of their development and deployment.**

We believe the demand for protocol customization will not only continue but escalate, motivating the need for a protocol agnostic approach to customization. We sketch out a Layer 4.5 architecture for systematic customization of host networking protocol behaviors based on a set of standardized, chainable modules. Layer 4.5 is a BPF-like socket layer implementation of protocol customization and aims at requiring no modifications to the original protocols or applications benefiting from it. We finish by exploring potential use cases applicable to network operations.

***Index Terms*—protocol customization, network monitoring, agile development**

I. Introduction

Traditionally, protocol customization is about extending existing protocols with additional features primarily for performance (e.g., MPTCP) and security reasons (e.g., protocol dialecting [1]). These customization needs are *per network* and evolve over time. We argue that continuing with the current approach of modifying code or configuration of each protocol in isolation, while workable in limited scenarios, is unable to fully meet the more stringent per network dynamic requirements, due to its inherent problems of complexity and middlebox interference. Therefore, this talk introduces a new approach to protocol customization. Inspired by the work of x-Kernel [2], our aim is two-fold: (i) raise the level of abstraction for reasoning and deploying customization on a per host or network basis, and (ii) optimize and support re-use of common customization primitives across protocols.

To illuminate the design space of this new approach, we sketch out a Layer 4.5 architecture for systematic customization of host networking protocol behaviors based on a set of standardized, chainable modules. The design exercise shows that Layer 4.5 customization will likely have important advantages over the existing approach: It may be able to realize a variety of customization goals without requiring modifications to the original protocols or applications benefiting from it. Further, it raises the level of abstraction so that a network operator can specify customization behaviors by composing module level semantics, thus potentially easing the burden of deployment.

II. Layer 4.5 Architecture

The Layer 4.5 framework, Figure 1, consists of customization modules that apply or process protocol customizations and a per host customization loader responsible for loading all applicable modules based on the socket context of the connection. It should be noted that we do not introduce new socket API calls to invoke the customization loader. Instead we tap into standard socket calls such as send and receive.

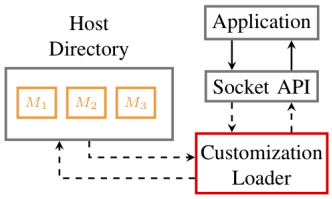

Fig. 1. Layer 4.5 customization architecture. Customization modules (depicted in orange boxes) are stored in a root privilidged customization directory on the host. The customization loader (red box) creates customized socket based on contents of the directory and application requesting socket.

A. Customization Modules

Much like the x-Kernel [2] approach, we raise the level of abstraction of customization requirements into *atomic* customization modules each *serving a specific goal* such as insertion of data to packets, insertion of new packets, and dropping or transformation of packets. Customization modules are the basic building blocks for realizing protocol customization requirements and include standard functions to separate the processing of ingress and egress packets at the sender and receiver, respectively. It is through the chaining of multiple such modules that more complex objectives are realized.

B. Customization Loader

The loader is supplied with a socket context from which it must determine what protocol customizations should be applied. This process could be accomplished by matching socket parameters with available modules located in a secure directory on the host machine. Each module includes necessary metadata to aid the loader in making decisions, such as the target IP addresses, port numbers, and/or application names. After making the determination on which modules should be applied to the given protocol, the loader will begin the process of linking/chaining them together in a specified order

according to the loader configuration. The customizations are then applied to the socket context, much like socket options, and all future communication on the socket will have the appropriate customizations applied.

III. Use Cases

A. Piloting New Services

New network protocols and features need to be tested/piloted prior to system-wide deployment on a given network. This typically leads to independent testbeds that implement the new capability to ensure compatibility with existing applications and network policies. The Layer 4.5 architecture allows for individual hosts to pilot new capabilities on a per-socket, per-application basis. The architecture allows for fine-grained testing of new capabilities and allowing multiple capabilities to be tested on the same machine without interfering with the normal operation of non-customized applications or protocols.

An example customization that could be developed and piloted would be a Forward Error Correction (FEC) module that is applied to UDP based applications. Through configuration of the module and customization loader, the module could be applied either to specific applications or to any application using the UDP protocol. Similarly, a series of modules could be chained together to enable QUIC-like behavior and applied to applications that could most benefit from it.

B. Expanding Monitoring Capabilities

BPF provides Linux administrators the capability to observe kernel level calls and gather information that was previously difficult to monitor [3]. These BPF hook points are based on specified events and are triggered by processes being monitored as well as those that are not. Layer 4.5 differs from current BPF capability by allowing customizations to add passive monitoring to applications on a per-socket basis. When the target application makes a send or receive call, the message buffer passes through the customization module(s), which can inspect the data and store information for user-level analysis. When another non-customized application makes the same send/receive calls, no extra buffer processing overhead is incurred.

Besides passive host-based monitoring, Layer 4.5 can also support network monitors by inserting data in each packet traversing the network. Given a specific network monitoring goal, such as user traffic identification independent of IP address, each packet could be altered to include the desired data prior to leaving the host machine. An example of such data could be the use of the Host Identity Protocol (HIP) version 2.5 implementation that includes identities above layer 4, but requires applications to be updated to invoke the protocol [4].

Introducing data to packets may impact deep packet inspection tools within the network. To address this, a customization server could be deployed on the network to orchestrate customization use. The server acts as an agent to communicate with end-hosts and middleboxes in the network and inform them of the active customization modules. The communication enables several approaches to alleviating middlebox interference. One is to integrate with the OpenBox architecture [5], which leverages the common processing conducted by multiple packet inspection services to reduce redundant processing. Under the OpenBox model, Layer 4.5 customization could be treated as a processing step prior to inspection by the middlebox. An alternative to OpenBox would be to have the customization server provide a standard API to receive and process customized packets from the middlebox, and then supply either the customization to be applied to the packet or a processed version of the packet for analysis.

IV. Concluding Remarks

We have sketched out a Layer 4.5 architecture for systematic protocol customization based on standardized, chainable and protocol-independent modules. We believe that elevating the level of abstraction has major advantages over the existing low level, protocol specific approach. It is our hope that the networking community will join us in refining the approach in order to meet the ever growing demand for efficient, agile and secure protocol customization solutions.

V. Biographies

Daniel Lukaszewski is a Naval Officer and a Ph.D. student at the Naval Postgraduate School. He earned his B.S. in mathematics at Univ. Arizona and a M.S. in computer science at NPS with a thesis investigating a MPUDP implementation in the Linux kernel to support VPN connections. His current research focuses on creating a systematic architecture for protocol customization efforts to reduce repeated efforts and allow for more rapid prototyping.

Geoffrey Xie is a CS Professor with the Naval Postgraduate School. He earned Ph.D. from UT Austin in 1996. He was a visiting scholar with CMU from 2003 to 2004, and with Cambridge Univ. from 2010 to 2011. He has published over 90 papers in data networking. He led the pioneering work on static reachability analysis of IP networks as well as the first work demonstrating feasibility and performance benefits of full-duplex communication in underwater acoustic networks. He was part of a team that won the 2015 SIGCOMM Test of Time Paper Award for proposing and illuminating the design space of centralized network control in 2005. His current research focuses on systematic network design and analysis.

References

[1] M. Sjoholmsierchio, B. Hale, D. Lukaszewski, and G. Xie, "Strengthening sdn security: Protocol dialecting and downgrade attacks," in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*. IEEE, 2021, pp. 321–329.

[2] N. C. Hutchinson and L. L. Peterson, "The x-kernel: An architecture for implementing network protocols," *IEEE Transactions on Software engineering*, vol. 17, no. 1, p. 64, 1991, publisher: IEEE Computer Society.

[3] B. Gregg, *BPF performance tools*. Addison-Wesley Professional, 2019.

[4] X. Gu *et al.*, "Host identity protocol version 2.5," *Master's thesis, Aalto University*, 2012.

[5] A. Bremler-Barr, Y. Harchol, and D. Hay, "Openbox: a software-defined framework for developing, deploying, and managing network functions," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 511–524.