Enhancing NF Compatibility by State-Manageable NF Development

Hongyi Huang
Tsinghua University

Wenfei Wu
Tsinghua University

ABSTRACT

Recently various NFV architectures are proposed for NF management and enhancement, however, individual NFs are difficult to be integrated with these architectures seamlessly. The state processing logic in NF programs is not explicitly declared and well organized, so the interaction between NFs and the NFV architecture needs to be managed manually, which is tedious and costly. To this end, we propose a State-MANageable NF programming framework (S-MAN). Developers use an S-MAN model to describe NF logic, where NF state processing logic is explicitly declared. An S-MAN compiler would generate NF programs from the NF models, where states are well organized. And the compiler would automatically add NF state operation interfaces for network controllers and NF state enhancement targeting underlying hardware.

1 INTRODUCTION

In modern computer networks, network functions (NFs, e.g., cache, firewall) play a crucial role in improving network performance and security, and network function virtualization (NFV) is a recent effort to enable the programmable and flexible deployment of NFs in networks. While various commodity NF programs and NF programming frameworks [1, 7] are proposed to cater to this trend, NF vendors and network operators still need to overcome the difficulty to integrate these individual NFs into existing network infrastructures.

As a data plane component, an NF needs to be *compatible* with both the network control plane and the underlying hardware. On the one hand, to enable fine-grained flow control from the control plane, an NF must provide operational interfaces to the network controller (e.g., OpenNF[4]), so that the network controller can flexibly create, migrate, and destroy states in NF instances. On the other hand, NFs are often enhanced by special hardware in the infrastructure (e.g., FPGA, GPU, Intel SGX[3]), where NF state processing logic needs to be customized for the hardware-supported libraries.

These integration processes usually require NF developers to make a careful anatomy of NF programs, identify NF state processing logic clearly, and then make customized modifications. However, the state processing logic in NF programs is often tightly-coupled with other parts of the program, which makes the analysis and modification tedious and costly. One recent solution StateAlyzr[5] can identify NF state variables in the program, but cannot extract the state processing logic.

While identifying state processing logic is difficult in existing NF programs, we rethink managing the state logic in the NF development and propose a State-MANageable NF programming framework (namely S-MAN). In S-MAN, an NF is described by an NF model where state processing is explicitly declared; an S-MAN compiler would translate the NF model to an NF program where states are well organized; finally, operational interfaces to the network controller and hardware-specific enhancement are automatically integrated during the compilation.

S-MAN provides a convenient means to generate customized NFs for NF vendors, and it automatically creates NF programs that are compatible with network management systems and network infrastructures, which eases the management from network operators. S-MAN is compatible with most existing NFV frameworks, and it can be used to develop monolithic NF for VM-based architecture (e.g., OpenNetVM[8]) and also fine-grained NF elements (e.g., ClickOS[6], NetBricks[7]). And we would demonstrate the feasibility and applicability of S-MAN by developing several typical NFs.

2 S-MAN DESIGN

Based on the discussion above, we describe our design of S-MAN in details, which includes state declaration in NF models, state organization in NF programs by the compiler, and state operation and optimization for network controllers and underlying hardware. With state management during development, S-MAN makes NF programs easier to be integrated with existing systems.

Overview. Figure 1 shows an overview of S-MAN. During NF development, S-MAN offers a domain specific language (S-MAN Descriptive Language, SDL) and

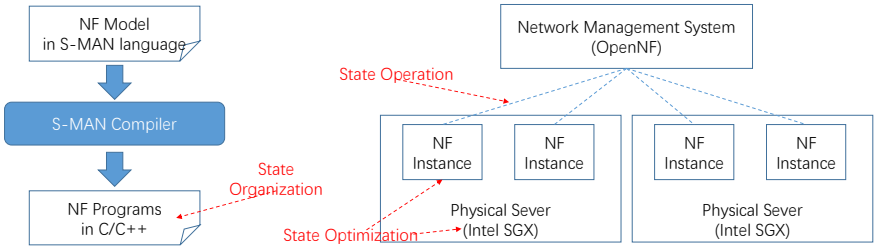

Figure 1: S-MAN Overview

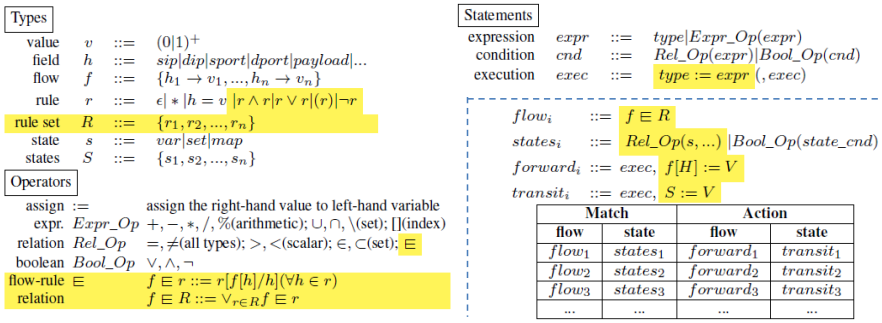

Figure 2: S-MAN NF Model

a programming model to describe NF logic to developers. And then S-MAN uses a compiler to interpret the NF model to a runnable NF program that is written in a target implementation language. In the NF program, states are well-organized for further lookup, overwrite, and deletion, and the compiler would add operational interfaces to network controllers; state processing logic is wrapped up, and the compiler would make optimization targeting special underlying hardware. In this project, SDL is an extension of SNAP language[1], and S-MAN would generate C++ programs; we adopt OpenNF[4] as our network controller and Intel SGX[3] as the underlying hardware.

S-MAN Model. NF states would be updated by NF action primitives and they partially determine where packets are sent [**?**]. In view of this, we take inspiration from existing match-action table within stateless switches and extend it to stateful NFs. In a stateless table, there are match field and action field; a few header fields are matched against a table; the matched entry specifies the corresponding action(s) that are applied to the flow[2]. To cover states, we add the match(es) upon states and the action(s) transiting states to match-action tables, shown as Figure 2. Thus, four pieces of logic compose an entry in S-MAN match-action table, including the match(es) upon flow/state and action(s) concerning flow/state. Each entry in S-MAN table represents an independent branch in the NF program (including conditions and operations on flows and states). With the NF model fixed, NF developers can focus on specifying the model's logic without concerning about the compatibility issue.

S-MAN Compiler. The compiler would translate NF models to corresponding NF programs. The compilation is straightforward, each entry in the model is an execution path with the matches on flow and states as conditions and the actions as branch body.

S-MAN must be skilled in state organization. In particular, each state is bound with a granularity (e.g., 5-tuple), and states are created dynamically as new flows are found to match an entry. All states are registered so that they can be looked-up, overwritten, and deleted.

State Operation and Enhancement. With states well-organized, S-MAN would add state operational interfaces for network controllers. Inspired by OpenNF[4], we would add get/putPerFlow, get/putMultiFlow, and get/putAllFlow interfaces.

S-MAN would add a security enhancement for state processing using Intel SGX. Inspired by current secure middlebox solutions, S-MAN would seal NF state processing using Intel SGX library. State transition logic in NF programs are from the state action field in NF models, thus, they can be wrapped up and declared as secure functions in Intel SGX directly.

REFERENCES

[1] Mina Tahmasbi Arashloo, Yaron Koral, Michael Greenberg, Jennifer Rexford, and David Walker. 2016. SNAP: Stateful network-wide abstractions for packet processing. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 29–43.

[2] Pat Bosshart, Glen Gibb, Hun Seok Kim, George Varghese, Nick Mckeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN. In *Acm Sigcomm Conference on Sigcomm*. 99–110.

[3] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016 (2016), 86.

[4] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. 2014. OpenNF: Enabling innovation in network function control. In *ACM SIGCOMM Computer Communication Review*, Vol. 44. ACM, 163–174.

[5] Junaid Khalid, Aaron Gember-Jacobson, Roney Michael, Anubhavnidhi Abhashkumar, and Aditya Akella. 2016. Paving the Way for NFV: Simplifying Middlebox Modifications Using StateAlyzr. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, Santa Clara, CA, 239–253. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/khalid

[6] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. 2014. ClickOS and the art of network function virtualization. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 459–473.

[7] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. 2016. NetBricks: Taking the V out of NFV.. In *OSDI*. 203–216.

[8] Wei Zhang, Guyue Liu, Wenhui Zhang, Neel Shah, Phillip Lopreiato, Gregoire Todeschi, K. K. Ramakrishnan, and Timothy Wood. 2016. OpenNetVM: A Platform for High Performance Network Service Chains. In *The Workshop on Hot Topics in Middleboxes and Network Function Virtualization*. 26–31.