NFD: Using Behavior Models to Develop Cross-Platform NFs

Hongyi Huang, Wenfei Wu
Tsinghua University

CCS CONCEPTS

• **Networks → Middle boxes / network appliances**.

KEYWORDS

Network Function(NF), NF Abstraction, Cross Platform

ACM Reference Format:
Hongyi Huang, Wenfei Wu. 2019. NFD: Using Behavior Models to Develop Cross-Platform NFs. In *SIGCOMM '19: ACM SIGCOMM 2019 Conference (SIGCOMM Posters and Demos '19), August 19–23, 2019, Beijing, China.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3342280.3342342

1 INTRODUCTION

With the capability to enhance network performance and security, the ecosystem of network function virtualization (NFV) has gradually matured over the past few years. Various network users (e.g., cloud tenants or enterprise network users) have requirements in network flow processing (e.g., filtering, caching, load balancing); network operators (e.g., cloud providers and enterprise network admin) would set up network runtime environments[1] (e.g., AWS Nitro, Azure VFP[3], OpenNF[5], Metron[7], ResQ[13] and CHC[8]), and there are *NF vendors* who deliver software network functions (NFs, a.k.a. middleboxes). These NF vendors could be traditional network device vendors like CISCO and Juniper or software companies like Microsoft and Oracle.

We define that an NF's logic consists of *flow processing logic* and *environmental adaptation logic*. While StatelessNF[6] prefers detaching state layer from NF logic, we propose this for the sake of cross-platform property. Throughout this paper, flow processing logic represents general functionalities of NFs without deployed infrastructure involved; environmental adaptation logic refers to pieces of codes in order to integrate NFs with either software or hardware environments. For *NF vendors*, two further questions may be proposed in NF development. (1) "Can we get an NF with-/without a feature X in flow processing?". For example, can a load balancer support blacklisting? If an IDS is configured to filter traffic by IP header only, can the TCP processing logic be removed (for the purpose of performance)? (2)"Can the NF be deployed in the environment Y?" For example, how does an IDS support SGX? How can a firewall be accelerated by GPU?

[1]"Platform" and "environment" are inter-changeable in this paper.

SIGCOMM Posters and Demos '19, August 19–23, 2019, Beijing, China

ACM ISBN 978-1-4503-6886-5/19/08…$15.00
https://doi.org/10.1145/3342280.3342342

Referring to current progress in NF development, most efforts are put in building NF programming abstractions[4, 10, 11], which improves programming NF flow processing logic. But environment adaptation is often neglected, and NFs are often tightly coupled with one specific runtime environment. For NF vendors, they would face the difficulty in rapidly releasing an NF product to fragmented market with diverse runtime network environments.

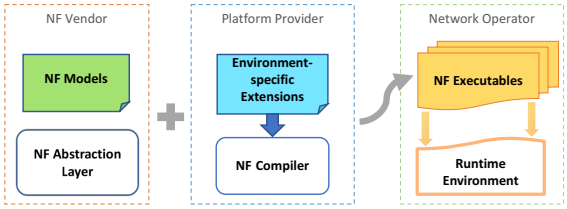

Figure 1: NFD overview.

We propose a new NF development framework named NFD. *NFD consists of an NF abstraction layer to develop platform-independent behavior models and a compiler to adapt NF models to specific runtime environments.* In practice, NF vendors develop platform-agnostic NF models on the NF abstraction layer, and platform providers[2] build compiler extensions; by enabling an extension in the compiler, the platform specific enhancement would be applied to NFs, leaving out the trouble in integrating environmental logic with each individual NF (Figure 1). Thus, NF vendors can build NFs that are "written once and run anywhere".

2 DESIGN

NF Behavior Models. We inherit and extend existing SDN switch and NF modeling language[1–3, 9] and define a platform-independent language in Figure 2. This language can express many commonly used network programming abstractions (i.e., program elements expressing NF processing semantics) including packet processing abstraction, bytestream processing abstraction, state-processing abstraction, timer-logic abstraction, and user-defined abstraction which allows users to implement their own programming abstractions.

An NF behavior model is defined as *an explicit and structured organization of NF programming abstractions* using NFD language. For example, a Stateful Match-Action Table (SMAT) can represent a large variety of NFs including firewall, NAT, load balancer, IDS, etc. A SMAT consists of multiple entries, and each entry has four fields: flow/state match and flow/state action. Each incoming packet is matched entry by entry - if the packet header matches the flow-match field and the NF internal states match the state-match field, the flow-action field is applied to the packet (i.e., send, drop, modification) and the state-action field is applied to NF states; if multiple

[2]They can be network operators who integrate various hardware/software in the platform; or they can be NF vendors who would like to sell NFs to a new platform; or they can be platform hardware/software vendors such as SGX, GPU, etc.

Basic types and expression

value	v	::=	$(0\|1)^+$
header field	h	::=	$sip\|dip\|sport\|dport\|proto\|\ldots$
state	s		
expression	e	::=	$v\|h\|s\|\vec{e}\|Expr\_Op(e_1, e_2, \ldots)$

Predicates

flow predicate	x_f, y_f	::=	$\epsilon \mid * \mid h = v \mid \neg x_f \mid x_f \wedge y_f \mid x_f \vee y_f$
state predicate	x_s, y_s	::=	$* \mid Rel\_Op(s, e) \mid \neg x_s \mid x_s \wedge y_s \mid x_s \vee y_s$

Policies

flow policy	p_f, q_f	::=	$h := e \mid p_f; q_f$
state policy	p_s, q_s	::=	$s := e \mid p_s; q_s$

Model

model	$model$	::=	$stmts$
statements	$stmts$	::=	$stmt \mid stmt; stmts$
statement	$stmt$	::=	$p \mid if \mid loop$
if statement	if	::=	if (x) then $stmts$ else $stmts$
loop statement	$loop$	::=	while(x) then $stmts$

SMAT (an example model)

entry	$entry$	::=	if $(x_f \wedge x_s)$ then $(p_f; p_s)$ else $\perp$
SMAT	$smat$	::=	$entry \mid entry; smat$

Figure 2: NFD language for NF models.

	Match		Action	
	Flow	**State**	**Flow**	**State**
Stateful Firewall	**Configuration:** OK={r1, r2, ...}			
	f∈OK	-	f[output]:=IFACE	seen:=seen∪{f}
	f	f∈seen	f[output]:=IFACE	-
	f∉ OK	f∉seen	f[output]:=ϵ	-

Figure 3: An example of NF models.

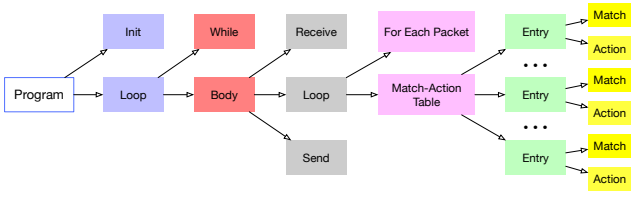

Figure 4: Syntax tree of an NFD program.

entries match the packet and the current states, the first match applies. For example, a stateful Firewall can be expressed in Figure 3.

NFD Compiler. The basic compiler framework compiles a model to a program. Currently we use C/C++ as the target language. During the compilation, control flows (i.e., if and while) are kept as the same in both NFD and C/C++. NFD specific programming abstractions such as flow/bytestream, states, and timers are implemented as libraries to link with the compilation output program.

During the compilation from NF model to NF program, the compiler keeps the syntax tree of the NF program (Figure 4). The syntax tree describes how the NF program is derived according to the syntax of NFD language. For example, the root "program" can derive the "init" block and the "loop" block; the "Match-Action Table" block would derive multiple entries. In the syntax tree, all the non-leaf nodes are symbols that can derive other symbols in the NFD language, and all the leaf nodes are basic symbols, which are the basic elements in NFD language, typically flow, state, operators, and keywords (e.g., if, while).

NFD provides the following interfaces to the environmental extension developer: (1) a syntax tree visitor, which could traverse a syntax tree and allow the programmer to transform the structure of the tree, (2) prototypes of all basic symbols, which allow the programmer to replace the implementation of the leaf node in the tree. We give three examples of using these interfaces to adapt an NF to a specific environment.

(1) Replacing implementation of basic symbols. Basic symbols are leaf nodes in a syntax tree, and the compiler would not translate them to detailed implementation in target NF program (i.e., they are function prototypes). Instead, the compiler would link the program with external libraries which contain their implementation to NF executables. NFD has its own default library to implement the basic symbols, but platform providers could re-implement them based on the function prototype. For example, we can use DPDK I/O to replace current libpcap I/O; or we can use GPU accelerated operator (e.g., Encrypt/Decrypt, or PatternMatch) to replace CPU-based implementations.

(2) Adding new logic to the syntax tree. Adding new logic can be implemented by traversing the syntax tree and inserting new branches on the tree. For example, the behavior model describes the data plane logic, while some platforms contain logic from network control plane, in which case, an integration of both is needed. Taking OpenNF[5] as an example, it requires each NF to have an agent to communicate with the network controller, which can be implemented by adding a piece of agent code as library and inserting the agent execution right after the initialization (i.e., "init" in Figure 4).

(3) Generating new configurations by traversing the tree. Some platform integrations do not change the workflow in NFs, but they collect extra information and generate platform specific configurations. For example, recent solutions[12] propose to protect NF sensitive states using Intel SGX. In NFD, this can be implemented by traversing the syntax tree and recording all state variables, and generating an SGX configuration file which claims all these variables sealed in SGX enclave.

3 PROGRESS AND PLAN

We have prototyped NFD. We developed 14 NFs, spanning security-featured NFs (e.g., Firewall, heavy hitter detector, and flood detector), LBs (layer-3 and layer-4), NAT, monitors, and rate limiters. The basic compiler framework can generate NFs supporting native Linux runtime environment (compatible with Linux Container and KVM), and we also build compiler extensions to support OpenNetVM[14], GPU acceleration, SGX protection, DPDK I/O acceleration, and OpenNF management.

We have evaluation about NFs' functionality and performance (omitted here) and NFD-based NFs shows comparable processing throughput with some de facto NFs. Our future plan includes enriching the language feature and discovering its semantic completeness, supporting more environments, and building network management framework (e.g., verification) based on NFD models.

REFERENCES

[1] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. Netkat: Semantic foundations for networks. In *ACM SIGPLAN Notices*, volume 49, pages 113–126. ACM, 2014.

[2] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker. Snap: Stateful network-wide abstractions for packet processing. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 29–43. ACM, 2016.

[3] D. Firestone. VFP: A virtual switch platform for host SDN in the public cloud. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 315–328, Boston, MA, 2017. USENIX Association.

[4] K. Gao, T. Nojima, and Y. R. Yang. Trident: toward a unified sdn programming framework with automatic updates. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 386–401. ACM, 2018.

[5] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. Opennf: Enabling innovation in network function control. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, 2014.

[6] M. Kablan, A. Alsudais, E. Keller, and F. Le. Stateless network functions: Breaking the tight coupling of state and processing. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 97–112, 2017.

[7] G. P. Katsikas, T. Barbette, D. Kostić, R. Steinert, and G. Q. M. Jr. Metron: NFV service chains at the true speed of the underlying hardware. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 171–186, Renton, WA, 2018. USENIX Association.

[8] J. Khalid and A. Akella. Correctness and performance for stateful chained network functions. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, Boston, MA, 2019. USENIX Association.

[9] H. Kim, J. Reich, A. Gupta, M. Shahbaz, N. Feamster, and R. Clark. Kinetic: Verifiable dynamic network control. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 59–72, 2015.

[10] B. Li, K. Tan, L. L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 1–14. ACM, 2016.

[11] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker. Netbricks: Taking the v out of nfv. In *OSDI*, pages 203–216, 2016.

[12] M.-W. Shih, M. Kumar, T. Kim, and A. Gavrilovska. S-nfv: Securing nfv states by using sgx. In *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 45–48. ACM, 2016.

[13] A. Tootoonchian, A. Panda, C. Lan, M. Walls, K. Argyraki, S. Ratnasamy, and S. Shenker. Resq: Enabling slos in network function virtualization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 283–297, 2018.

[14] W. Zhang, G. Liu, W. Zhang, N. Shah, P. Lopreiato, G. Todeschi, K. K. Ramakrishnan, and T. Wood. Opennetvm: A platform for high performance network service chains. In D. Han and D. Raz, editors, *Proceedings of the ACM SIGCOMM Workshop on Hot topics in Middleboxes and Network Function Virtualization, HotMiddlebox@SIGCOMM 2016, Florianopolis, Brazil, August, 2016*, pages 26–31. ACM, 2016.