Network Function Modeling and Its Applications

Wenfei Wu • *Hewlett Packard Labs*

Ying Zhang • *Facebook*

The widespread deployment of proprietary network functions (NFs) creates the need for accurate NF models to ensure correctness and reliability. Here, the authors review existing NF models and modeling approaches, propose generic modeling methodologies, and present the models' applications in network management tasks.

Realizing the agility, flexibility, and economy that network function virtualization (NFV) brings into network management and service provision, NFV has received significant attention in recent years. Several NFV platforms — for example, OPNFV (see www.opnfv.org) and OpenStack (www.openstack.org) — are including NFV reference implementations, while standardization bodies such as the European Telecommunications Standards Institute (ETSI), Open Networking Foundation (ONF), and IETF focus on defining the NFV architecture and the interfaces between components. In the NFV architecture, each NF is viewed as a blackbox that processes traffic arbitrarily.

However, as more NFs are developed by diverse vendors and deployed in richer scenarios, the blackbox view of NFs has already hindered its more advanced application and development. For example, if NFs are viewed as boxes without knowing their internal logic, there is no way to reason about their forwarding behaviors; thus, NFV operators can't troubleshoot if they're misbehaving or verify network-wide policy compliance such as reachability.[1] Knowing an NF's behavior can also help monitor the policy compliance by generating meaningful testing traces.[2] Thus, we believe it's time to call for standardizing NF models describing NF internal forwarding logic.

Existing works usually build NF models according to their application scenarios, so the models are tailored specifically to their applications. These approaches fall into two categories. The first category aims to build models used for network management tasks — for example, testing, traffic engineering, and migration;[1–3] and the second focuses on improving NF software, including its performance, reliability, and robustness.[4–11]

Combining these NF applications and development needs, we believe a generic NF model should precisely describe NF internal logic and be interoperable with other NFs. In this article, we summarize existing efforts on NF modeling, show our methodologies to automatically model NFs, and propose several possible application scenarios with NF models.

NF Models in the NFV Ecosystem

Figure 1 shows the important role that NF models play in the whole NFV ecosystem. NF models connect NFV applications and the NF implementation. From the NFV operators' viewpoint, several applications on top of NFV platforms are based on NF models. For example, when an operator needs to generate test traces for deployed NFs, the operator should assume NF behaviors (that is, model) to completely cover the test space;[2] when the operator needs to verify network-wide properties (for example, reachability), whether a flow

1089-7801/17/$33.00 © 2017 IEEE

gets through an NF needs to be inferred from the NF model.[8]

NF vendors also require models to guide the NF program development. For example, NetVM[4] views an NF as a whole function box and focuses on optimizing the input/output performance; NetBricks[5] defines basic processing logical blocks in NF programs, and implements memory isolation between blocks; while Fault-Tolerant MiddleBox (FTMB)[7] focuses on the dependency between packets and internal states so as to correctly rollback in case of failure.

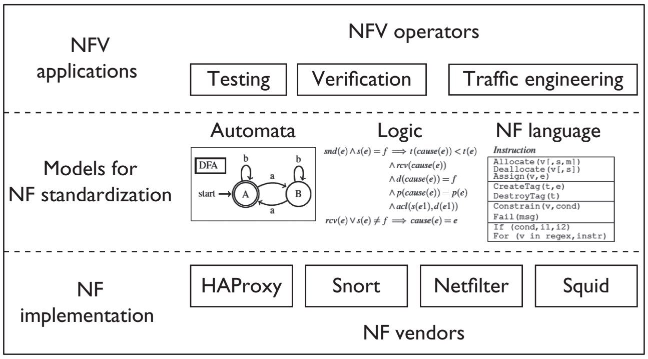


Figure 1. Network function (NF) models in the network function virtualization (NFV) ecosystem. NF models connect NFV applications and the NF implementation. From the NFV operators' viewpoint, several applications on top of NFV platforms are based on NF models.

Existing NF Modeling Approaches

A set of works provide a taxonomy of NFs[9–11] and give definitions of specific NFs; these models are also bound with the specific NF types (for example, firewalls or network address translators). Another set of works model all-type NFs but usually target specific applications (for example, verification or performance optimization) instead of the whole ecosystem. The representative approaches to model all-type NFs are as follows.

Blackbox/Graybox without Modeling Internal Logic

In most existing NFV platforms (OPNFV, OpenStack, and OpenNF[3]) and NF performance optimization solutions,[4] an NF is viewed as a blackbox with traffic traversing it, but the NF's internal logic isn't differentiated. In practice, an NF is implemented as a virtual machine (VM) with the NF software running inside it, and the VM is deployed on general servers, allowing traffic to be routed through. This coarse-grained view only supports simple operations: NF bootup, migration, suspension, and destruction.

OpenNF takes a further step — it recognizes an NF as running logic plus a state blob. The state blob can be created or destroyed, moved between NF instances, and encoded or decoded by NFs. This abstraction allows for flexible flow-level operations (that is, flow migration between NF instances), but still is limited to flow management only.

Hand-Crafted Models with Domain Knowledge

Some NFV applications view NFs as deterministic finite automata (DFA). For example, BUZZ[2] uses DFA to represent each NF in the network, then finds possible violation states in each DFA and uses a network-wide model to generate traces (that might trigger the violation) for testing. In another example, SFC-Checker[8] also views NFs as DFAs; it mainly determines whether given a network with NFs and a series of packet traces, if there's a risk of unexpected reachability/unreachability. The DFA in these works are manually generated and directly used to replace the NFs, which implies possible incompleteness.

Mihai Dobrescu and Katerina Argyraki propose to write NF programs in a style with bounded arrays and loops,[6] and Radu Stoenescu and colleagues propose a symbolic execution friendly language to remove these two features for NF programming.[12] Both works aim to develop NF programs that symbolically can be executed to aid in finding program bugs early in development. These two approaches provide new ways to model NFs, but still require programmers to write NF programs.

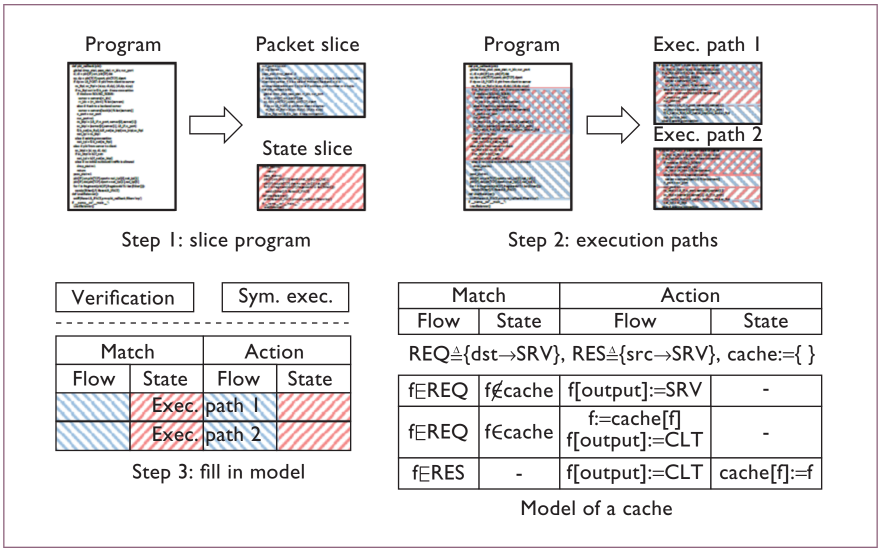
Aurojit Panda and colleagues propose using first-order logic to describe NF behaviors, and give solutions to verify network-wide properties (for example, isolation).[1] The solution is based on SAT solvers. Anthony Joseph and Ion Stoica propose a similar modeling approach,[13] which still requires the programmer with NF knowledge to rewrite NFs in the format of first-order logic.

Automatic NF Model Synthesis

Realizing the incompleteness and manual effort needed to leverage existing NF modeling approaches, we propose building models directly and automatically from actual NFs to get accurate and generic models. There are two approaches: whitebox and blackbox modeling.[14]

Whitebox Modeling

Assuming NF source code is known, we propose a modeling framework called Lancet. An NF program is assumed to have a processing loop and packet input/output functions, and an NF model is predefined like a stateful match action table (SMAT). Lancet first conducts backward



Match		Action	
Flow	State	Flow	State
REQ≜{dst→SRV}, RES≜{src→SRV}, cache:={ }			
f∈REQ	f∉cache	f[output]:=SRV	-

f∈REQ	f∈cache	f:=cache[f]
		f[output]:=CLT
			-

f∈RES	-	f[output]:=CLT	cache[f]:=f

Model of a cache

Figure 2. Lancet steps and output. The first is the slice program; next the execution paths run; and finally, we see the fill-in model. For the cache model, CLT = client; dst = destination; REQ = request; RES = request; Sym. exec. = symbolic execution; and SRV = server.

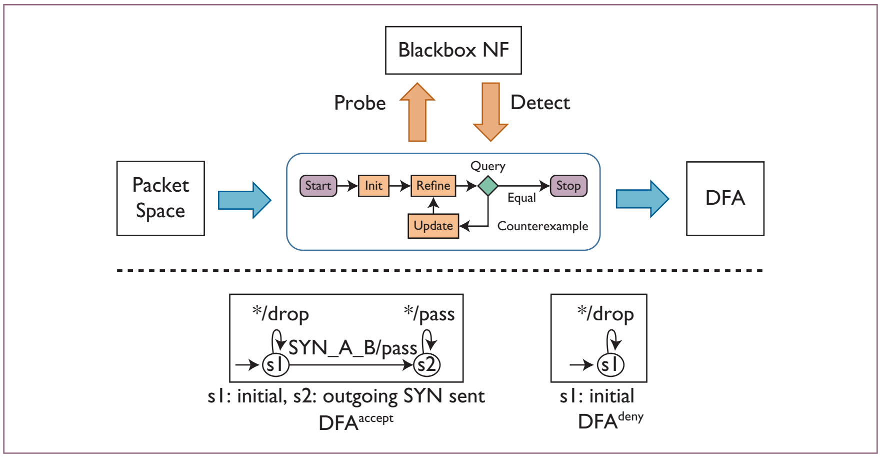


Figure 3. Alembic overview and output. DFA = deterministic finite automata and SYN = synchronize message (SYN). Here, s1 and s2 are two states in DFA.

slicing from the packet output function to get a packet slice, and from the state variable[15] assignment statements to get a state slice. Lancet then symbolically executes the union of both slices and gets multiple execution paths. Finally, each path is filled into the SMAT as a row, conditional statements about packets/states are put into the match column's flow/state fields, and the intersection of an execution path with the packet/state slice is put into the action column's flow/state fields. The output model's completeness can be guaranteed and proved by the theory found in program-slicing techniques.

Figure 2 shows the modeling results of running Lancet on the World Wide Web Offline Explorer (WWWOFFLE) web cache (see www.gedanken.org.uk/software/wwwoffle). The logic in the match action table is described in stateful network-wide abstractions (SNAP)-like language[16] for abbreviation. When the cache observes a request for the first time, it forwards the request to the server; when a response comes back, the cache records the response content and forwards it to its client side. When the cache observes a request whose response has appeared before, it immediately replies to the client with the response without forwarding the request to the server.

Blackbox Modeling

Assuming the source code is unknown (for example, proprietary NFs), we can set up an NF, send packets to it, observe its reactions, and infer its internal logic. We built a system called Alembic. Alembic takes the packet space (filtered by related header fields) as input, interacts with the blackbox NF, and leverages an active learning algorithm proposed by Dana Angluin[17] to generate a DFA model. The main idea of Angluin's algorithm is to iteratively assume a DFA model, compare the DFA with the NF, and refine the DFA until no difference is found between the DFA and the NF. With the algorithm, Alembic completes sequences within the length of the longest probing sequence.

For example, we use Alembic to model a stateful firewall pfSense (see https://pfsense.org), and get the results in Figure 3. If the firewall is configured with an accept rule between the trusted and untrusted zones, it allows an outgoing synchronize message (SYN) to punch a hole, and all following incoming and outgoing traffic of that flow is allowed; without an outgoing SYN, all traffic is dropped. If the firewall is configured with a deny rule, all traffic is simply dropped.

Model Applications

With an accurate NF model, we can design and develop several NFV applications. Example applications include NF program generation, stateful network verification, and network policy composition.

Model-based NF programming is the reverse process of model generation. To generate a new NF program from a customized SMAT (SMAT and DFA are interchangeable; we omit the proof in this article), the program first contains a loop starting with the packet input function. Each row in the SMAT is transformed to a branch in the program. The branch's condition statement is the conjunction (AND) of its state/flow match fields, and the program executes the statements in the branch's state/flow action fields. This program can be viewed as a microservice with the same functionality as the NF in terms of NF logic. The challenge in model-based NF programming is to guarantee other program requirements, including fault tolerance, performance, management API, and so on.

The DFA model can be plugged into several stateful network verification solutions. While we list two existing DFA-based solutions in previous sections (SFC-Checker and BUZZ), there are still challenges. For example, how do we guarantee the verification's completeness and soundness, and how do we efficiently verify large networks with frequent changes? We abstract and solve these problems based on DFA theory.

We can also use NF models in service policy composition. When deploying a network-wide policy composed of several NFs, there might be conflicts between NFs. For example, deploying a cache and a firewall in different orders would lead to different results, such as when one client's content is cached and provided to another client, but was intended to be blocked by the firewall. NF models can be used to determine this order in service policy composition: the operator models the input/output of each NF when chaining them to guarantee end-to-end correctness.

We envision that NF modeling has a number of benefits in long-term NFV evolution and deployment. With NF models, NF interoperability will be improved, the NFV platform will be easier to place onboard and deploy, and debugging and testing will be simplified. The essential step toward this ease of management is NF model standardization, which can occur in three phases.

The first step is to understand and build accurate and representative models from exiting NFs. This step requires developing various modeling techniques to learn about the model out of existing NF boxes. After each individual model is collected, the second step is to identify the commonality across heterogeneous implementations and consolidate a unified model for each NF type. Third, an NF program synthesis method based on unified NF models must be developed to enable NF vendors to customize and implement new NFs.

References

1. A. Panda et al., "Verifying Reachability in Networks with Mutable Datapaths," *Proc. Usenix Symp. Networked Systems Design and Implementation*, 2017, pp. 699–718.
2. S.K. Fayaz et al., "BUZZ: Testing Context-Dependent Policies in Stateful Networks," *Proc. Usenix Symp. Networked Systems Design and Implementation*, 2016, pp. 275–289.
3. A. Gember-Jacobson et al., "OpenNF: Enabling Innovation in Network Function Control," *Proc. ACM Conf. Sigcomm*, 2014; doi:10.1145/2619239.2626313.
4. J. Hwang, K.K. Ramakrishnan, and T. Wood, "NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms," *Proc. Usenix Symp. Networked Systems Design and Implementation*, 2014, pp. 445–458.
5. A. Panda et al., "NetBricks: Taking the V out of NFV," *Proc. Usenix Symp. Operating Systems Design and Implementation*, 2016, pp. 203–216.
6. M. Dobrescu and K. Argyraki, "Software Dataplane Verification," *Proc. Usenix Symp. Networked Systems Design and Implementation*, 2014, pp. 101–114.
7. J. Sherry et al., "Rollback-Recovery for Middleboxes," *Proc. ACM Conf. Sigcomm*, 2015; doi:10.1145/2785956.2787501.
8. B. Tschaen et al., "SFC-Checker: Checking the Correct Forwarding Behavior of Service Function Chaining," *Proc. IEEE Conf. Network Function Virtualization and Software Defined Networks*, 2016; https://users.cs.duke.edu/~btschaen/papers/SFC-Checker.pdf.
9. P. Srisuresh and M. Holdrege, *IP Network Address Translator (NAT) Terminology and Considerations*, IETF RFC 2663, Aug. 1999; www.rfc-editor.org/rfc/rfc2663.txt.
10. I. Cooper, I. Melve, and G. Tomlinson, *Internet Web Replication and Caching Taxonomy*, IETF RFC 3040, Jan. 2001; www.rfc-editor.org/rfc/rfc3040.txt.
11. B. Carpenter and S. Brim, *Middleboxes: Taxonomy and Issues*, IETF RFC 3234, Feb. 2002; www.rfc-editor.org/rfc/rfc3234.txt.
12. R. Stoenescu et al., "SymNet: Scalable Symbolic Execution for Modern Networks," *Proc. ACM Conf. Sigcomm*, 2016; doi:10.1145/2934872.2934881.
13. A.D. Joseph and I. Stoica, "Modeling Middleboxes," *IEEE Network*, vol. 22, no. 5, 2008, pp. 20–25.
14. W. Wu, Y. Zhang, and S. Banerjee, "Automatic Synthesis of NF Models by Program Analysis," *Proc. ACM Workshop Hot Topics in Networks*, 2016, pp. 29–35.
15. J. Khalid et al., "Paving the Way for NFV: Simplifying Middlebox Modifications Using StateAlyzr," *Proc. Usenix Symp. Networked Systems Design and Implementation*, 2016; www.usenix.org/node/194933.
16. M.T. Arashloo et al., "SNAP: Stateful Network-Wide Abstractions for Packet Processing," *Proc. ACM Conf. Sigcomm*, 2016, pp. 29–43.
17. D. Angluin, "Learning Regular Sets from Queries and Counterexamples," *J.*

Information and Computation, vol. 75, no. 2, 1987, pp. 87–106.

Wenfei Wu is a postdoctoral researcher at Hewlett Packard Labs and assistant professor at Tsinghua University. His research interests include network diagnostics, software-defined networking (SDN), and network function virtualization (NFV). Wu has a PhD in computer science from the University of Wisconsin–Madison. Contact him at wenfeiwu@outlook.com.

Ying Zhang is a staff software engineer at Facebook. Her research interests include SDN, NFV, and cloud networking. Zhang has a PhD in computer science and engineering from the University of Michigan. Contact her at winginsky@gmail.com.