Policy and Resource Orchestration in Software-Defined Networks

Anduo Wang and Jie Wu
Center for Networked Computing
Temple University, Philadelphia, PA
Email: {jiewu, adw}@temple.edu

***Abstract*—This position paper deals with policy and resource orchestration in large-scale software-defined networks (SDNs) through network function virtualization (NFV). One of the key issues in a large-scale SDN is the automated policy orchestration and efficient resource sharing. The rich set of SDN policies that collectively drive the behavior of a single SDN often interacts in complex ways. With NFV, on the other hand, multiple workloads can co-locate and use a given infrastructure simultaneously. We examine challenges related to the automated orchestration and provisions of SDN-enabled policies and software middleboxes. We propose novel solutions — optimizations algorithms and reasoning procedures — that advance the state of art SDN orchestration service.**

***Index Terms*—Middlebox placement, network function virtualization, network policy, software-defined networks.**

I. Introduction

Software-defined networks (SDN) decouples the control plane of traditional networks devices to a distinct controller, exposing to networks operators a high-level management interface, and connects to the underlying heterogeneous network hardware through standard protocols. As shown in Figure 1, SDN thus provides an extremely flexible centralized alternative to managing the distributed heterogeneous network devices. Network operators enjoyed an unprecedented opportunity to realize their traffic goals — traffic forwarding, fault tolerance, resource provisioning, stateful middleboxes, service chains, and more — by directly programming their own network policies or adding software functions (also called, Virtual Network Functions, VNF).

To facility network operators to create their own policies, much efforts have been on simplifying programming. Sophisticated control platforms [1–5] decouple the complex distributed state management from the decision logic through separate state management service, enabling centralized control that is often drastically simpler than a distributed counterpart. Simultaneously, a variety of network abstractions [6] which raise the level of programming are introduced, further hide the complexity of low-level details. However, how much SDN can realize in terms of taming complexity of integrating network policy still remains an open question [7, 8].

On the other hand, network function virtualization (NFV) leverages virtualization technologies to implement network function (traditionally implemented in hardware) in software middleboxes, or simply *middlebox*. Such functions include firewalls, network address translators, proxies, etc. SDN offers the new opportunity of hosting the software middleboxes, also called VNF, by directing traffic through the VNFs and allowing the software middleboxes to choose service locations from multiple available servers/routers. Therefore, resource orchestration in terms of VNF provisioning and sharing becomes very important in a large-scale SDN.

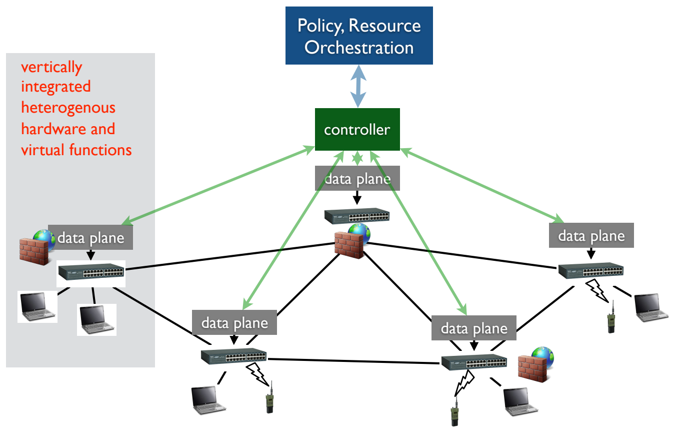

Policy, Resource Orchestration
controller
vertically integrated heterogenous hardware and virtual functions
data plane
data plane
data plane
data plane
data plane

Fig. 1: Software defined network overview.

II. Policy Orchestration

Software-defined networking (SDN) refactors the distributed network protocols in the network into an ensemble of centralized programs running at a server (controller) that is separate from the network, creating a rare opportunity to simplify network management with modern software engineering. Yet the SDN software architecture, which often requires coordination among multiple entities over shared states, remains monolithic. The SDN controllers, or network operating systems [2, 3], while exposing to control software a uniform programming interface that abstracts away details of the network hardware, fall short in providing the operating system functionality of coordination among those software.

The onus of combining multiple control software that collectively drive the behavior of a single network is falling on the admin to write modular programs. Modular programming, though a natural choice at first glance, often prefixed [9, 10] modularization support in the language features tailored to a particular task. The modular composition itself is tightly coupled with the code that achieves the individual target task, and determining the composition requires clear understanding of the joint intent of every components.

Modularization by Semantics layering

To bring modularity to SDN software, in this position paper, we propose a drastically different approach called *semantics layering*. Rather than embedding modularization in user-supplied modular software, semantics layering realizes modularization through a distinct orchestration service implemented at the controller. Semantics layering is a general organizational principle that is decoupled from individual software component, it promotes, enforces, and automatically determines modularization.

Semantics layering is built on the insight that essential to modular composition is not the different form of network abstractions used to realize a semantic property, but the property itself expressible in standard logic. That is, we view the SDN software components as semantic units that operate in a control loop — each control module continuously monitors the network states s against some property i, whenever a violation of i is detected, it reconfigures s to restore the invariant by generating some update u.

Rather than relying on the composition logic explicitly specified in the modular program for instructing the interactions among these semantic units, semantic layering orchestrates the network updates u by automatically inferring their impacts on the semantic properties. The result of such automated reasoning is semantic layering of SDN applications — the upper layers depend on the lower ones for repairing property violations. Whenever an application initiates an update, all lower-layer applications are invoked as well. For example, routing application is located below the firewall application because the firewall, in order to enforce secure end to end connectivity, must *depend* on the routing application to actually remove the switch configurations corresponding to the insecure request.

Determining Semantics layering

The essence of semantics layering is to coordinate SDN modules to respect the semantic properties of every individual modules such that the updates pushed by one will not inadvertly hurt the properties of another. To this end, we introduce the notion of semantics dependency: one module depends on another module if the maintenance of its property *logically implies* the maintenance of the property of the second.

To determine semantics layering, we only need to determine semantic dependency, a problem that can be recasted as the (database) irrelevant update problem. Given two modules x and y, and some shared network states s; we represent s by tables (facts), and formalize x and y as a pair of database programs that continuously query (monitor) and update (reconfigure) the tables s. In the database terms, x depends on y if the output of x's query — a database view — is affected by y's update program, but x's update will never alter y's query result. That is, we only need to check whether x's update is irrelevant of y's query.

Armed with the formulation of semantic dependency as a database irrelevant update problem, we can determine semantic dependency by database irrelevant reasoning [11–13], a satisfiability technique that checks irrelevant updates. Once we generate the dependency graph containing all semantic dependencies among the modules, we can run a topological sort to produce a hierarchy of modules — semantics layers, in which each layer enriches and depends on the properties maintained by the ones beneath it.

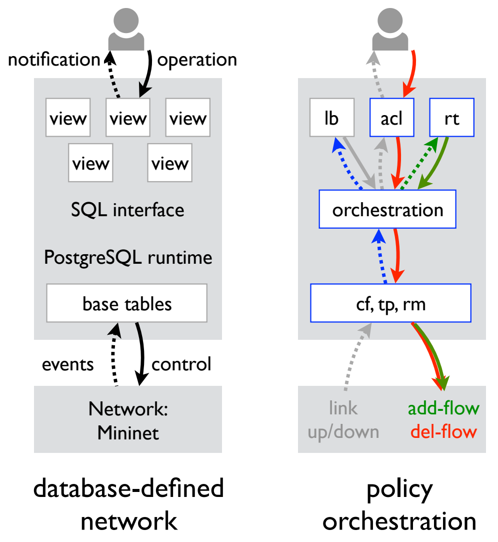


Fig. 2: Policy orchestration, a database implementation.

Database implementation

Having pinned down the semantics layering principle, our goal is to implement an orchestration service that enforces modular composition of disparate control software with correctness guarantee. We leverage our previous work *Ravel* [14], a database-defined network that utilizes a standard SQL database as "the" highly-customizable controller to manage the network. Ravel features a plain control plane abstractions and orchestrates control modules by user-defined priority. Ravel represents all network states as database tables: the dataplane as shared, stored tables, while the modules operate on derived tables (database views); and to reduce a module x's operations — the checking and repairing of invariants — to a database query (view x_v and a database update (view update x_U). We enhanced Ravel with an orchestration service by integrating irrelevant update reasoning, as shown in Figure 2.

As an example, we show the (ir)relevant update reasoning for firewall and routing. The routing update does not affect firewall policy, because either deletion nor insertion on the Ravel configuration table ($cf(F, X, Y)$, meaning that a flow F at switch X will be forwarded to Y) can affect the reachability requirement table ($rm(F, S, D)$ denoting a flow F between nodes S, D) which is monitored by firewall, thus routing update is trivially irrelevant to firewall. The firewall update, however, does affect routing because the firewall deletion condition $rm(F, S, D) \wedge blacklist(S, D)$ and the condition of firewall violation view $rm(F, S, D) \wedge \neg cf(F, X, Y), \ldots$ is jointly satisfiable.

III. Resource Orchestration

One of the key resource orchestration issues in a large-scale SDN is the efficient resource provisioning and sharing. With NFV, multiple middleboxes can co-locate and use a given infrastructure simultaneously. We examine several challenges related to provisioning and sharing software middleboxes, also called *middlebox placement*.

In middlebox placement, we usually place several flows that are required to go through several types of middleboxes, m_1, m_2, ..., in a particular total order (also called *service chain*), partial order, or no order (i.e., independent). We call these requirements *flow-to-middlebox* constraints. The service chain comes from the service requirement in the network. For example, an Internet Protocol Security Decryptor (a type of middlebox) must be placed before a Network Address Translator (NAT, another type of middlebox). The following shows several research threads related to middlebox placement.

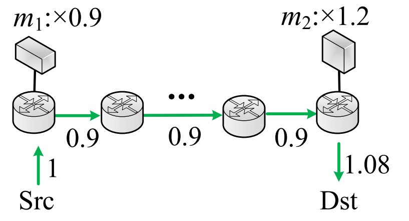
Graph embedding: Various service chains form a *middlebox graph*, G_m, of multiple service chains that needs to be embedded in a given *network graph,* G_n. It is shown in [15] that the embeddability of G_m in G_n is NP-hard under various constraints, including node capacity, link capacity, node placement, link placement, and latency.

Graph flow routing: This thread deals with several classic graph flow routing algorithms such as shortest path and maximum flow in G_n, subject to the given flow-to-middlebox constraint. For example, the classic shortest path algorithm has been extended [16] under the constraint that each path must go through a given service chain that is already assigned in G_n. The classic maximum flow problem has also been extended [16] in such a way that the flow goes through a given middlebox assigned in G_n; however, the hardness of the problem in which the flow goes through several middleboxes still remains open.

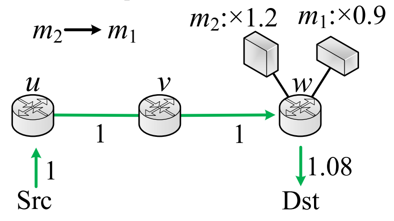
Facility allocation: Location analysis resembles the *facility location problem* that concerns with the optimal placement of facilities to minimize transportation costs. Here facilities are middleboxes and transportation costs correspond to incurred flow costs when flows go through network and are possibly detoured through middleboxes. Middlebox placement can be also related to the generalized assignment problem [17].

Set covering: When the flow assinement is already given in G_n and each middlebox has a capacity limit, the middlebox placement resembles the set covering problem with some twists [18]. The objective is usually to minimize the number of middleboxes used (as shown below). Middlebox placement can be better explained through the *hitting set* formulation which is equivalent to set covering. The hitting set problem is described as a bipartite graph, with right vertices (middleboxes) to be selected to cover left vertices (flows in the network).

Although middlebox placement is similar to several classic graph and combinatoric optimizations, it has some uniqueness. For example, middleboxes may exhibit *traffic-changing effects* [19]. For example, BCH encoder (a type of middlebox) adds



(a) Independent middleboxes.



(b) Dependent middleboxes (m_2 before m_1).

Fig. 3: A traffic-changing middlebox placement example.

31% to traffic volume due to checksum overhead. The Citrix CloudBridge WAN optimizer (another type of middlebox) reduces traffic volume by up to 80% by compressing traffic. One interesting problem is how to place middleboxes that minimizes the total traffic. For example, suppose a flow that goes through u, v, and w with a flow-to-middlebox requirement of m_1 ($\times$ 0.9, traffic-changing ratio) and m_2 ($\times$ 1.2). The optimal assignment would be m_2 to u and m_1 to w as shown in Figure 1 (a). However, if m_1 cannot be placed before m_2, then both m_1 and m_2 need to be placed in w as shown in Figure 1 (b). When a middlexbox has flow capacity limit, the problem becomes more challenging as shown in Figure 1 (c) with flows f_1, f_2, and f_3 having flow rates of 1 Mbps, 4 Mbps, and 1 Mbps, respectively, and capacity for both m_1 ($\times$ 0.9) and m_2 ($\times$ 1.2) being 3 Mbps, but without any service order. In this case, the optimal solution in terms of minimum total traffic is shown in Figure 1 (c), where f_2 is covered twice, one at u for a partial coverage of 3 Mbps and another at v for the remaining coverage of 1 Mbps. Chen and Wu [20] studied an optimal scheduling in a tree-based topology that minimizes the combination of total traffic and total middlebox set-up costs.

Although significant efforts have been made in middlebox placement, there are many other unexplored areas. Many middlebox placement problems are NP-hard, and we can examine special setting where these problems are tractable, such as by considering special network topologies such as trees. Currently, all flow-to-middlebox conditions require that flows must go through certain middleboxes. We can also explore the opposite requirement that flows are forbidden to pass through nodes with certain middleboxes (for security reasons). Lastly, we can look into the relationship between middlebox placement and some other classic scheduling problems, such as *flow shop* scheduling in which each job needs to follow several processing stages in a given sequence.

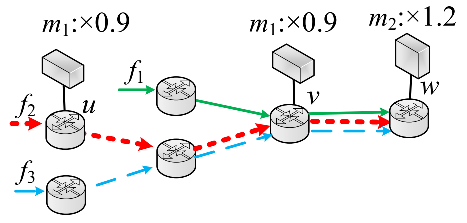


Fig. 4: A traffic-changing middlebox placement example with a flow covered by multiple middleboxes.

IV. Conclusion

In this paper, we develop new solutions to policy and resource orchestration for software-defined network. First, we propose a new organization principle called semantic layering that orchestrates the interactions of policies by their semantics — the properties maintained by component policies and the logic entailment relations among them. We develop a database realization of semantic layering that leverages our previous work on database-defined network, where each SDN policy is formalized by database integrity constraints. We automatically infer the semantic layering of the integrity constraints by database irrelevance reasoning.

We then study the joint VNF deployment and flow allocation problem. We aim at minimizing the total cost of deploying VNF instances when all flows are fully processed. We assume that all flows request the same type of network functions. We study the heterogeneous VNF deployment in tree topologies. We reformulate the deployment of heterogeneous VNFs in a line and propose a performance-guaranteed strategy. An optimal greedy solution is designed for homogeneous VNF deployment in a line.

References

[1] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "Onos: Towards an open, distributed sdn os," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620744

[2] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: a distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI'10, 2010.

[3] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: Towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008. [Online]. Available: http://doi.acm.org/10.1145/1384609.1384625

[4] J. Rexford, A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, G. Xie, J. Zhan, and H. Zhang, "Network-wide decision making: Toward a wafer-thin control plane," in *In Proceedings of HotNets III*, 2004.

[5] P. Sun, R. Mahajan, J. Rexford, L. Yuan, M. Zhang, and A. Arefin, "A network-state management service," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 563–574. [Online]. Available: http://doi.acm.org/10.1145/2619239.2626298

[6] H. Kim, J. Reich, A. Gupta, M. Shahbaz, N. Feamster, and R. Clark, "Kinetic: Verifiable dynamic network control," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 59–72. [Online]. Available: http://dl.acm.org/citation.cfm?id=2789770.2789775

[7] A. Vahdat, D. Clark, and J. Rexford, "A purpose-built global network: Google's move to sdn," *Queue*, vol. 13, no. 8, pp. 100:100–100:125, Oct. 2015. [Online]. Available: http://doi.acm.org/10.1145/2838344.2856460

[8] M. Casado, N. Foster, and A. Guha, "Abstractions for software-defined networks," *Commun. ACM*, vol. 57, no. 10, pp. 86–95, Sep. 2014. [Online]. Available: http://doi.acm.org/10.1145/2661061.2661063

[9] N. Foster, A. Guha, M. Reitblatt, A. Story, M. J. Freedman, N. P. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, D. Walker, and R. Harrison, "Languages for software-defined networks." *IEEE Communications Magazine*, vol. 51, no. 2, pp. 128–134, 2013.

[10] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN Programming with Pyretic," *USENIX ;login*, vol. 38, no. 5, October 2013.

[11] J. A. Blakeley, N. Coburn, and P.-V. Larson, "Updating derived relations: Detecting irrelevant and autonomously computable updates," *ACM Trans. Database Syst.*, vol. 14, no. 3, pp. 369–400, Sep. 1989. [Online]. Available: http://doi.acm.org/10.1145/68012.68015

[12] A. Y. Levy and Y. Sagiv, "Queries independent of updates," in *Proceedings of the 19th International Conference on Very Large Data Bases*, ser. VLDB '93. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 171–181. [Online]. Available: http://dl.acm.org/citation.cfm?id=645919.672674

[13] C. Elkan, "Independence of logic database queries and update," in *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, ser. PODS '90. New York, NY, USA: ACM, 1990, pp. 154–160. [Online]. Available: http://doi.acm.org/10.1145/298514.298557

[14] A. Wang, X. Mei, J. Croft, M. Caesar, and B. Godfrey, "Ravel: A database-defined network," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16. New York, NY, USA: ACM, 2016, pp. 5:1–5:7. [Online]. Available: http://doi.acm.org/10.1145/2890955.2890970

[15] M. Rost and S. Schmid, "Charting the complexity landscape of virtual network embeddings," in *Proc. of IFIP 2018*.

[16] G. Sallam, G. R. Gupta, B. Li, and B. Ji, "Shortest path and maximum flow problems under service function chaining constraints," in *Proc. of IEEE INFOCOM 2018*.

[17] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. of INFOCOM 2015*.

[18] Y. Chen, J. Wu, and B. Ji, "Virtual network function deployment in tree-structured networks," in *Proc. of IEEE ICNP 2018*.

[19] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent nfv middleboxes," in *Proc. of IEEE INFOCOM 2017*.

[20] Y. Chen and J. Wu, "NFV middlebox placement with balanced set-up cost and bandwidth consumption," in *Proc. of ICPP 2018*.