A Semantic Approach to Modularizing SDN Software

Anduo Wang
Temple University

1. PROBLEM STATEMENT

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 [4, 8], 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 [3, 11] 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. Moreover, modular programming solutions, driven by ad hoc requirements lack clear direction and foresight, cannot anticipate future needs, in brief, will not give true modularity to the extremely flexible and ever evolving SDN.

To bring modularity to SDN software, in this poster, we advocate a drastically different approach, a shift in SDN software architecture. Rather than embedding modularization in user-supplied modular software, we propose to realize modularization through a distinct architectural primitive implemented at the controller that, decoupled from individual software component, promotes, enforces, and automatically determines modularization. Making modularization an architectural primitive yields solutions to multiple problems. *Simplify network management.* A truly modular SDN architecture makes the SDN software easier to understand, reuse, and extend. A highly modular control plane is also more configurable and flexible. *Independent evolution.* Modularization as a distinct primitive frees the component modules from committing to one "right" programming platform that fits all. More importantly, it enables independent evolution of the component software, the conceptual model, and the language abstractions. *Incremental deployment.* New control software or new modularization principle can be introduced without retrofitting the rest of the system.

Of course, modularization as an architectural primitive has been proposed in the past, including layering, software router platforms such as x-Kernel, Click, XORP, and declarative networking [6, 7, 5, 10]; but all these solutions are based on data flow organizational principle that decomposes the software system into parts among which certain data — packets, routes — flow. A data abstraction which, essential to the data flow principle, agreed by every single module in the extremely flexible and evolving SDN, however, is unlikely. To identify a *flexible organizational principle*, then, is the first key barrier to modularizing SDN software. Can we define a simple yet powerful interface along which meaningful composition can be determined without restricting the flexibility in programming individual software component? A second key barrier is *determining the meaningful modularization*. Can we automatically discover the right composition without relying on the internalized knowledge of experienced admin who clearly understands the joint intent of every module? Finally, can we *put the principle into practical use via a controller service*? Can we extend an SDN controller to enable high-level control software for drastically different purposes while at the same time enforcing meaningful composition?

We argue that these barriers, while ambitious, are solvable. Moreover, the solutions will shed light on SDN abstractions, guide future design, and improve understanding of dependencies in SDN networks. As a first step towards this goal, this poster will present the design of *semantics flow*, an organizational principle centered around network semantics — logical properties managed by individual modules. As opposed to data flow principle that breaks the system into parts among which some form of data — packets, routes — flow, semantics flow decomposes the network into semantic units related by logical implication. Leveraging the author's recent work on database defined network [12], a unique SDN architecture that utilizes a standard SQL database as "the" controller to manage the network with highly customizable control plane abstractions and a open controller runtime, making it a convenient platform to implement the modularization service, we discuss the implementation of semantics flow as a controller service.

2. A SEMANTIC APPROACH

Semantics flow

Our first step is to identify a *task-agnostic organizational principle* in which to handle and compose control modules in a uniform manner. We build on the insight that essential

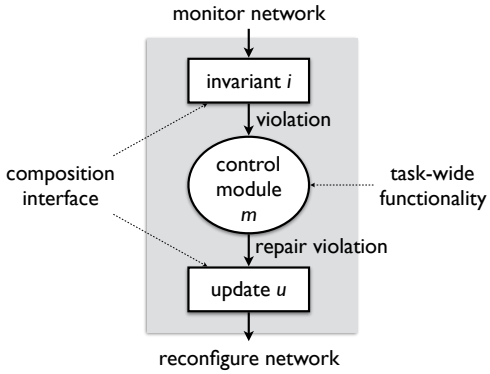

Figure 1: A generic model of SDN control module

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. In other words, we treat the software components as semantic units, and we only need to standardize the way we handle their semantics.

Figure 1 depicts a generic model of a control module m that operates over some network state s, describing a module's semantic property. The composition interface of m consists of two parts: the invariant i describes the semantics — network properties — of the target task managed by the module, is taken by the module as input that (when violated) triggers repairing update; the updates u describe the affects of the module on the rest of the network and are generated as output that restore the invariant property. Both i and m refer to states of the network s. Intuitively, m operates in a control loop — it continuously monitors the network states s through i, whenever a violation of i is detected, it reconfigures s to restore the invariant by generating some update u.

Determining Semantics Flow

In addition to promote and enforce modularity, we develop automated reasoning method that automatically determines meaningful composition. We first observe that the crux to a coherent SDN control plane — a set of well formed modules that collectively maintain a consistent network state — is to coordinate the modules to respect the semantic properties of every individual modules such that the updates pushed by one will not permanently hurt the properties of another. To capture this intuition, we introduce the notion of semantics dependency: a module depends on another if the maintenance of its property *logically implies* the maintenance of the property of the second.

More importantly, we can recast this as the (database) irrelevant update problem. The idea is that given two modules `x` and `y`, and some shared network states `S` (as shown in Figure 1); we represent s as database base tables (facts), and formalize x and y as a pair of database programs that continuously query (monitor) and update (reconfigure) the base tables s. In the database terms, `x` depends on `y` if the output of `x`'s query — a database view — can be affected by `y`'s update program, but `x`'s update will never alter `y`'s query result — it thus suffices to check whether `x`'s update is irrelevant to `y`'s query.

Armed with the notion of semantic dependency and its formulation as a database irrelevant update problem, we can automatically determine semantic dependency by database irrelevant reasoning [1, 9, 2], 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, in such a hierarchy each layer enriches and depends on the properties maintained by the ones beneath it.

Database implementation

Having pinned down the semantics flow principle, our goal is to implement a controller service that enforces modular composition of disparate control software with correctness guarantee. We leverage our previous work *Ravel* [12], 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. We plan to enhance Ravel orchestration service by integrating irrelevant update reasoning, to automatically combine modules by the semantics flow principle.

3. REFERENCES

[1] BLAKELEY, J. A., COBURN, N., AND LARSON, P.-V. Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM Trans. Database Syst. 14*, 3 (Sept. 1989), 369–400.

[2] ELKAN, C. Independence of logic database queries and update. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (New York, NY, USA, 1990), PODS '90, ACM, pp. 154–160.

[3] FOSTER, N., GUHA, A., REITBLATT, M., STORY, A., FREEDMAN, M. J., KATTA, N. P., MONSANTO, C., REICH, J., REXFORD, J., SCHLESINGER, C., WALKER, D., AND HARRISON, R. Languages for software-defined networks. *IEEE Communications Magazine 51*, 2 (2013), 128–134.

[4] GUDE, N., KOPONEN, T., PETTIT, J., PFAFF, B., CASADO, M., MCKEOWN, N., AND SHENKER, S. Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev. 38*, 3 (July 2008), 105–110.

[5] HANDLEY, M., HODSON, O., AND KOHLER, E. Xorp: An open platform for network research. *SIGCOMM Comput. Commun. Rev. 33*, 1 (Jan. 2003), 53–57.

[6] HUTCHINSON, N., AND PETERSON, L. Design of the x-kernel. In *Symposium Proceedings on Communications Architectures and Protocols* (New York, NY, USA, 1988), SIGCOMM '88, ACM, pp. 65–75.

[7] KOHLER, E., MORRIS, R., CHEN, B., JANNOTTI, J., AND KAASHOEK, M. F. The click modular router. *ACM Trans. Comput. Syst. 18*, 3 (Aug. 2000), 263–297.

[8] KOPONEN, T., CASADO, M., GUDE, N., STRIBLING, J., POUTIEVSKI, L., ZHU, M., RAMANATHAN, R., IWATA, Y., INOUE, H., HAMA, T., AND SHENKER, S. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation* (2010), OSDI'10.

[9] LEVY, A. Y., AND SAGIV, Y. Queries independent of updates. In *Proceedings of the 19th International Conference on Very Large Data Bases* (San Francisco, CA, USA, 1993), VLDB '93, Morgan Kaufmann Publishers Inc., pp. 171–181.

[10] LOO, B. T., HELLERSTEIN, J. M., STOICA, I., AND RAMAKRISHNAN, R. Declarative routing: Extensible routing with declarative queries. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (New York, NY, USA, 2005), SIGCOMM '05, ACM, pp. 289–300.

[11] REICH, J., MONSANTO, C., FOSTER, N., REXFORD, J., AND WALKER, D. Modular SDN Programming with Pyretic. *USENIX ;login 38*, 5 (October 2013).

[12] WANG, A., MEI, X., CROFT, J., CAESAR, M., AND GODFREY, B. Ravel: A database-defined network. In *Proceedings of the Symposium on SDN Research* (New York, NY, USA, 2016), SOSR '16, ACM, pp. 5:1–5:7.