test: A Semantic Approach to Modularizing SDN Software

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, 5], 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, 7] 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 [1, 6, 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 — semantics layers, in which each layer enriches and depends on the properties maintained by the ones beneath it.

1. 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] 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.

[6] 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.

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