Towards Correct-by-Construction SDN

Leonid Ryzhyk[1], Nikolaj Bjørner[2], Marco Canini[3], Jean-Baptiste Jeannin[1], Nina Narodytska[1], Cole Schlesinger[1], Douglas B. Terry[1], and George Varghese[2]

[1]Samsung Research America [2]Microsoft Research [3]Université catholique de Louvain

High-level SDN languages raise the level of abstraction in SDN programming from managing individual switches to programming network-wide policies. In this talk, we present Cocoon (for Correct by Construction Networking), a language designed around the idea of *iterative refinement*. The programmer starts with a high-level functional description of the desired network behavior, focusing on the service the network should provide to each packet, as opposed to how this service is implemented within the network fabric. The developer then iteratively refines the top-level specification, adding details of the topology, routing, fault recovery, etc., until reaching a level of detail sufficient for the Cocoon compiler to generate an SDN application that manages network switches via the southbound interface (we currently support P4 [3]). We designed Cocoon with the following goals in mind:

Correctness Cocoon uses the Corral model checker [6] to establish that each refinement correctly implements the behavior it refines, ensuring that behaviors specified at any refinement step hold on the resulting SDN application.

Generality Cocoon enables a wide range of SDN applications, ranging from network virtualization, through software-defined IXPs, to home networks.

Dynamism A Cocoon program specifies both data and control plane behavior, akin to languages like FlowLog [7], Maple [9], and VeriCon [2]. This is in contrast to languages such as NetKAT [1], which specify a snapshot of data plane behavior but rely on a general-purpose programming language to implement the control plane by emitting a stream of snapshots in response to network events.

Flexibility Existing high-level languages rely on fixed compilation strategies in mapping the high-level network program to a switch-level implementation. Cocoon allows the programmer to specify how each high-level component is implemented and deployed, while automatically verifying the correctness of the implementation.

Refinements are performed either manually or automatically. In the latter case, the user picks one of an extensible set of *refinement tactics*. A tactic can be as simple as instantiating shortest-path routing within a segment of the network, or as complicated as the global NetKAT compilation algorithm [8]. The user is free to apply different tactics to different parts of the network. Whenever no existing tactic matches application requirements, the user can implement their own custom solution via a manual refinement.

Refinements are performed in a modular way, with every refinement confined to a single component of the network. Modularity is enforced at the language level: a Cocoon program defines a number of *roles*, where each role models a group of similar entities, e.g., top-of-rack switches, virtual network function instances, or a segment of the switching fabric. A refinement replaces one or more roles with a more detailed implementation, possibly splitting them into multiple roles.

The modular refinement process facilitates clean separation of concerns and ensures that each individual refinement is amenable to automatic formal verification. The Cocoon verifier proves for each refinement that the refined specification is functionally equivalent to the role it refines. Correctness of each refinement in the chain guarantees that the final specification is functionally equivalent to the top-level specification.

In this talk, we will present the Cocoon's design philosophy, outline its syntax and semantics, as well as report on several case studies where we used Cocoon to synthesize and verify a range of SDN applications, including network virtualization, service chaining, a B4-style WAN [5], and a software-defined Internet exchange [4].

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 *POPL'14*, San Diego, CA, USA, 2014.

[2] T. Ball, N. Bjørner, A. Gember, S. Itzhaky, A. Karbyshev, M. Sagiv, M. Schapira, and A. Valadarsky. VeriCon: Towards verifying controller programs in software-defined networks. In *PLDI'14*, Edinburgh, United Kingdom, 2014.

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[4] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever. An industrial-scale software defined internet exchange point. In *NSDI'16*, Santa Clara, CA, USA, 2016.

[5] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined WAN. In *SIGCOMM'13*, Hong Kong, China, 2013.

[6] A. Lal, S. Qadeer, and S. K. Lahiri. A solver for reachability modulo theories. In *CAV'12*, Berkeley, CA, USA, 2012.

[7] T. Nelson, A. D. Ferguson, M. J. G. Scheer, and S. Krishnamurthi. Tierless programming and reasoning for software-defined networks. In *NSDI'14*, Seattle, WA, 2014.

[8] S. Smolka, S. Eliopoulos, N. Foster, and A. Guha. A fast compiler for NetKAT. In *ICFP'15*, Vancouver, BC, Canada, 2015.

[9] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak. Maple: Simplifying sdn programming using algorithmic policies. In *SIGCOMM'13*, Hong Kong, China, 2013.