Declarative Verifiable SDN Specification

Rick McGeer

Bio

Rick McGeer is a Principal Investigator at the Communications and Design Group, the Chief Scientist of US Ignite, Adjunct Professor of Computer Science at the University of Victoria. Previously, as a Distinguished Technologist at HP Labs, he served on the Steering Committee of the PlanetLab Consortium and was on the original GENI Design Team. He co-led the team at HP Labs which did the implementation of OpenFlow on a commercial switch, and led multiple GENI projects, including GENI Cloud, the InstaGENI GENIRack project, and the GENI Experiment Engine. At CDG and US Ignite, he continues to lead the GEE project and also built the Ignite Distributed Collaborative Scientific Visualization System. He is the author of over 100 refereed technical papers and one book.

Contribution and Issues

Software-Defined Networking, in particular OpenFlow, has garnered much attention, because it promises developers control over networking akin to their control over other IT resources. Such control is vital to distributed systems, where the network is an integral part of the application. Critically, OpenFlow has been implemented on commodity switches, permitting its rapid deployment on existing networks.
OpenFlow networks are also much easier to verify than classic networks. The classic model of a network is a graph of Turing Machines. Verification of such a network is undecidable, which means, in practice, that debugging the network is similar to debugging a program – with less visibility than one usually has as a program. However, a number of researchers in both the networking and formal verification communities have observed that an OpenFlow network is *state-free*, and with reasonable technological assumptions (notably, a fixed packet time-to-live) it was mathematically equivalent to a combinational logic network. Verification of such networks is in the class *NP*. Better, this specific problem has been extensively studied in the context of the verification of VLSI circuits, and has by now become a matter of industry practice. Best, early experimental evidence on OpenFlow networks has indicated that the instances of verification problems generated by actual network descriptions are amenable to rapid solution, primarily because they generate a small number of equivalence classes of cases.
Effective solution of the network verification problem will yield a number of benefits, in addition to the obvious one (answering the question: “is it possible for my network to do the following bad thing?”). For example, it has been shown that transitioning a network from one configuration to another, while preserving desired properties, is an instance of the verification problem. This variant introduces a number of variables, such that a safe-update schedule is contained in the value of those variables.
The obvious benefits of verifiable, safely-updatable networks will only be achieved, however, if a network specification language can be devised which is based on the state-free semantics of OpenFlow and which incorporates the implementation semantics of network primitives. Verifiability of logic networks in VLSI was retarded for years because of the poor semantics of the primary modeling languages, Verilog and VHDL; the semantics of these languages were based on event-driven simulators, not logic circuits, making the derivation of logic-circuit models a difficult challenge for both synthesis and simulation tools. Similarly, programming models that are too powerful retard verification. Weak models make for strong verification. It is a common error for language designers to base a language on Turing-complete semantics, because it makes implementation easy and it is easy to incorporate ancillary computation. But this makes extraction of the description of the artifact difficult, and verification problematic.
It is therefore critical to have network descriptions which:

1. Can be used by a network compiler to embed an idealized network onto a physical network, preserving topology and delays
2. Maintain verifiable semantics through the use of declarative primitives