A Correct, Zero-Overhead Protocol for Network Updates

Rick McGeer
HP Enterprise Services, Palo Alto, CA
Rick.McGeer@hp.com

ABSTRACT

In this paper, we describe a new protocol for the safe update of OpenFlow networks. This protocol meets the packet consistency and weak flow consistency conditions, requires neither on-switch resources nor the diversion of packets to refuges during updates, and alls into the family of Trace-based update protocols. The feature of this protocol is a sequence of per-switch rule updates. We derive a logic circuit for the update sequence, such that there exists a consistency-preserving update for the switch network if and only if the circuit is satisfiable subject to unsatisfiability of invariant violations; further, each satisfying minterm of the circuit yields a consistency-preserving update sequence.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Network Protcols

Keywords

Complexity, Network Update, Network Verification

1. INTRODUCTION

The introduction of the OpenFlow protocol[1, 6] has led to fresh approaches static network verification and network update. *Network verification* is the problem of ensuring that a given static network configuration meets global correctness conditions. The problem of *network update* is ensuring that the correctness conditions are preserved throughout a transition from one network configuration to another.

HIstorically, both problems have been extremely difficult. Traditional Layer-2 networks run a distributed, Turing-complete, real-time continuous computation. Verification of the results of this computation can be undecidable. OpenFlow changed this dramatically, by radically simplifying the nature of a network of switching elements. OpenFlow factors a switching network into a graph of simple, table-driven table-driven forwarding engines and a centralized controller. Verification of the switching graph is isomorphic to verification of a combinational logic network[5, 9], a problem known to be in co-$\mathcal{NP}$. This result is similar to [3], and has been implicitly exploited for network update[4, 8, 7].

In [8], the basic conditions for safe network update were enumerated. These are:

- **The Packet Consistency Condition**. Each packet flowing through the network is processed by a single ruleset.
- **The Flow Consistency Condition**. All packets in the same flow must be handled by a single network configuration (**Strong Consistency**), or the *prefix* of a flow must be handled by one configuration and the *suffix* of the flow must be handled by the successor configuration (**Weak Consistency**, introduced in [4])

[8] derived an elegant method for ensuring both packet consistency and flow consistency. Both rulesets were loaded into each switch, with an unused bit in the header field used to indicate which ruleset was desired. Edge nodes would then set the bit, as appropriate, in order to indicate which ruleset should be used for a particular packet.

This method consumed significant on-switch resources: both configurations would have to be represented in each switch's rule table. Effectively, the method of [8] cuts the available TCAM in half. [4] sought to remedy this by loading an intermediate ruleset onto the switches. In this method, packets whose handling was modified by the change were sent to the controller (or other packet refuge) by the intermediate ruleset; once the intermediate ruleset had been completely loaded onto all switches, the final ruleset was sent to the switches and the held packets were sent to their destinations (or any switch along their final path). This method conserved TCAM space (only one ruleset was present at a time), but increased bandwidth to the controller, increased latency, and resulted in possibly out-of-order packet delivery.

[7] extended the static verification method of [3] to incorporate dynamic network conditions, expressing the properties to be preserved in computation tree logic (CTL)[2].

Our work extends this in two significant areas. As we did for static verification, we demonstrate that one need not use CTL to verify network properties, but the computationally-simpler verification of Boolean networks. CTL verification is known to be in EXPTIME, whereas Boolean verification is in co-$\mathcal{NP}$. We show that safe network update is in co-$\mathcal{NP}$. We derive a polynomial-checkable algorithm to find a safe update sequence for an arbitrary network.

HotSDN'13, August 16, 2013, Hong Kong, China.
ACM 978-1-4503-2178-5/13/08.

2. MODEL, PROBLEM, SOLUTION

Following [5, 9], we model the network as implementing a transfer function on the Boolean spaces $\mathcal{T} : \mathbf{B}^{r \log p} \to \mathbf{B}^{rp}$, where r is the number of header bits and p is the number of network terminii[1]. Any boolean function on $\mathbf{B}^{r \log p} \to \mathbf{B}^{rp}$ gives rise to a boolean function $\phi : \mathbf{B}^{r \log p} \times \mathbf{B}^{rp} \to \{0, 1\}$: $\phi(x, y) = 1$ iff $\mathcal{T}(x) = y$. For details on construction of the network transfer function $\mathcal{T}$ and its associated function ϕ, see [5]. Informally, $\phi = 1$ only if the set of header bits and output places y is the result of transferring x through the network. The correctness condition for static verification is achieved by specifying two correctness functions, α, β over the same space as ϕ. α represents desired packet handling, and β forbidden packet handling. Correctness of the network is then given by the proposition $(\phi \supseteq \alpha)(\overline{\phi} \supseteq \beta)$; the actual transfer function contains all desired packet handling and contains no undesired packet handling.

Our goal is to derive a sequence of updates such that each meets the correctness conditions; we will update one switch at a time, ensure that the correctness conditions are respected, then update the next until all switches have been updated. Each update will perturb a network transfer function ϕ_i into a new network transfer function ϕ_{i+1}.

The network transfer functions are constructed using a composition of the transfer functions at each switch, in an algorithm given in [5]. Denoting the transfer function at switch k as F^k and the composition operator as $\mathcal{C}$, we write $\phi = \mathcal{C}(F^k)$. At time i, the transfer function of switch k is F_i^k, so $\phi_i = \mathcal{C}(F_i^k)$. We must capture three conditions: each switch changes its transfer function from the first to the second configuration, exactly one switch changes at each time i, and each switch changes transfer function exactly once. In order to capture these conditions, we denote the first transfer function at k as F^k, the second as $F^{k\prime}$. If there are n switches, we introduce n^2 new variables z_{ij}, with the property that $z_{ij} = 1$ iff $F_i^j = F^{j\prime}$ at time i. We then can write the switch transfer function through all updates as $F_i^j = z_{ij}F^{j\prime} + \overline{z_{ij}}F^j$. The condition that switch j starts in the initial configuration and ends in the final configuration is is captured by the condition $\overline{z_{0j}}z_{nj}$. The condition that each switch changes value exactly once is given by $z_{ij} \subseteq z_{i+1j}$. The condition that exactly one switch changes value at each time is given by ensuring that at each time i, exactly i of the z_{ij} variables are 1. This is captured in an $O(n)$ logic circuit, which we denote as $\psi_i(z_{ij})$.

Correctness remains. There are two fundamental choices: to insist that the updates respect the invariant pair (α, β); or to insist that they meet the flow correctness conditions. Given that both the initial transfer function ϕ_0 (all switches in the initial configuration) and the final transfer function ϕ_n (all switches in the final configuration) respect the invariants $\overline{\phi_n}\alpha = \overline{\phi_0}\alpha = \phi_n\overline{\beta} = \phi_0\overline{\beta} = 0$, it is fairly easy to show that the packet and weak flow consistency conditions are stronger; any update sequence that respects packet and weak flow consistency will also respect the invariants. Indeed, this is somewhat trivial; packet and flow consistency require that each intermediate transfer function be contained in the disjunction of the initial and final transfer functions; since the initial and final transfer functions respect the invariants, so too must the intermediate transfer functions. This opens the possibility of correct updates which respect the invariants but violate packet or flow consistency.

Packet consistency requires that a packet be handled by either the initial or final transfer functions: $\phi_i \subseteq \phi_0 + \phi_n$. Weak flow consistency requires that once one packet in a flow is handled by the final transfer function, so are all subsequent packets: $\phi_i\phi_n \subseteq \phi_{i+1}\phi_n$. The final formulation is:

$$\phi_i = \mathcal{C}(F_i^k) \quad (1)$$

$$F_i^j = z_{ij}F^{j\prime} + \overline{z_{ij}}F^j \quad (2)$$

$$\prod_j (\overline{z_{0j}}z_{nj})(z_{ij} \subseteq z_{i+1j}) \quad (3)$$

$$\psi_i(z_{ij}) \quad (4)$$

$$\phi_i \subseteq \phi_0 + \phi_n \quad (5)$$

$$\phi_i\phi_n \subseteq \phi_{i+1}\phi_n \quad (6)$$

Relaxing to respecting the invariants replaces (5)-(6) with

$$\overline{\phi_i}\alpha = \phi_i\overline{\beta} = 0 \quad (7)$$

THEOREM 1. *A satisfying assignment to the logic network (1)-(6) yields an update schedule satisfying flow consistency and weak packet consistency, and the invariants. A satisfying assignment to the logic network (1)-(4), (7) yields an update schedule satisfying the invariants. Both logic networks are polynomial in the size of the switching network.*

3. REFERENCES

[1] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In *Proceedings of ACM SIGCOMM*, August 2007.

[2] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8:244–263, 1986.

[3] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *Proceedings of the Usenix Conference on Network Design and Implementation*, April 2012.

[4] R. McGeer. A safe, efficient update protocol for openflow networks. In *Proceedings of HOT SDN*, August 2012.

[5] R. McGeer. Verification of switching network properties using satisfiability. In *Proceedings of ICC WSDN*, June 2012.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *ACM SIGCOMM CCR*, 38(2):69âĂŞ74, 2008.

[7] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. In *SIGCOMM*, pages 323–334, 2012.

[8] M. Reitblatt, N. Foster, J. Rexford, and D. Walker. Software updates in openflow networks: Change you can believe in. In *Proceedings of HotNets*, 2011.

[9] S. Zhang, S. Malik, and R. McGeer. Verification of computer switching networks: an overview. In *Proceedings of the 10th international conference on Automated Technology for Verification and Analysis*, ATVA'12, pages 1–16, Berlin, Heidelberg, 2012. Springer-Verlag.

[1] $\mathbf{B}^k$ is the boolean space of k bits