International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 3 | ISSN : 2456-3307

Reverse Update : A Uniform Policy Update Theme For Software Defined Networking

D. Banu Priya, T. Sivakumar
Department of Computer Science, Pondicherry University, Puducherry, Tamil Nadu, India

ABSTRACT

Policy and path updates area unit common causes of network instability, resulting in service disruptions or vulnerable intermediate states. during this letter, we tend to propose the Reverse Update, an update theme for software system outlined Networking that guarantees to preserve properties of flows throughout the transition time. we tend to prove through a proper model that the proposal achieves consistent policy updates, during which in-transit packets are continually handled within the next forwarding hops by identical or a newer policy. the most contributions are: (i) a relaxation of the conception of per-packet-consistency within the information plane of software system outlined Networking; and (ii) a policy update theme, evidenced to be consistent and economical. A software system outlined Networking machine was developed and valid. The results of our simulations show that the planned Reverse Update theme is quicker and has lower overhead than the present Two-Phase Update planned within the literature.

Keywords: Software Defined Networking, Consistency, Policy Update, Network Security

I. INTRODUCTION

Software-Defined Networking (SDN) is an idea which has recently reignited the interest of network researchers for programmable networks and shifted the attention of the networking community to this topic by promising to make the process of designing and managing networks more innovative and simplified compared to the well-established but inflexible current approach. Designing and managing computer networks can become a very daunting task due to the high level of complexity involved. The tight coupling between a network's control plane (where the decisions of handling traffic are made) and data plane (where the actual forwarding of traffic takes place) give rise to various challenges related to its management and evolution. Network operators need to manually transform high level policies into low-level configuration commands, a process which for complex networks can be really challenging and error-prone. Introducing new functionality to the network, like intrusion-detection systems and load balancers usually requires tampering with the network's infrastructure and has a direct impact on its logic, while deploying new protocols can be a slow process demanding years of standardization and testing to ensure interoperability among the implementations provided by various vendors. The idea of programmable networks has been proposed as a means to remedy this situation by promoting innovation in network management and the deployment of network services through programmability of the underlying network entities using some sort of an open network API. This leads to flexible networks able to operate according to the user's needs in a direct analogy to how programming languages are being used to reprogram computers in order to perform a number of tasks without the need

for continuous modification of the underlying hardware platform. SDN is a relatively new paradigm of a programmable network which changes the way that networks are designed and managed by introducing an abstraction that decouples the control from the data plane. In this approach a software control program, referred to as the controller, has an overview of the whole network and is responsible for the decision making, while the hardware (routers, switches etc.) is simply responsible for forwarding packets into their destination as per the controller's instructions, typically a set of packet-handling rules.

Software Defined Networking (SDN) is an emerging network architecture where network control is decoupled from forwarding and is directly programmable. This migration of control, formerly tightly bound in individual network devices, into accessible computing devices enables the underlying infrastructure to be abstracted for applications and network services, which can treat the network as a logical or virtual entity. Network intelligence is (logically) centralized in software-based SDN controllers, which maintain a global view of the network. As a result, the network appears to the applications and policy engines as a single, logical switch. With SDN, enterprises and carriers gain vendor-independent control over the entire network from a single logical point, which greatly simplifies the network design and operation. SDN also greatly simplifies the network devices themselves, since they no longer need to understand and process thousands of protocol standards but merely accept instructions from the SDN controllers.

II. LITERATURE SURVEY

Abstractions for Network Update

Configuration changes are a common source of instability in networks, leading to outages, performance disruptions, and security vulnerabilities. Even when the initial and final configurations are correct, the update process itself often steps through intermediate configurations that exhibit incorrect behaviors. This paper introduces the notion of consistent network updates—updates that are guaranteed to preserve well-defined behaviors when transitioning between configurations. We identify two distinct consistency levels, per-packet and per-flow, and we present general mechanisms for implementing them in Software-Defined Networks using switch APIs like OpenFlow. We develop a formal model of OpenFlow networks, and prove that consistent updates preserve a large class of properties. We describe our prototype implementation, including several optimizations that reduce the overhead required to perform consistent updates. We present a verification tool that leverages consistent updates to significantly reduce the complexity of checking the correctness of network control software. Finally, we describe the results of some simple experiments demonstrating the effectiveness of these optimizations on example applications.

A NICE way to test Openflow applications

The emergence of OpenFlow-capable switches enables exciting new network functionality, at the risk of programming errors that make communication less reliable. The centralized programming model, where a single controller program manages the network, seems to reduce the likelihood of bugs. However, the system is inherently distributed and asynchronous, with events happening at different switches and end hosts, and inevitable delays affecting communication with the controller. In this paper, we present efficient, systematic techniques for testing unmodified controller programs. Our NICE tool applies model checking to explore the state space of the entire system—the controller, the switches, and the hosts. Scalability is the main challenge, given the diversity of data packets, the large system state, and the many possible event orderings. To address this, we propose a novel way to augment model checking with symbolic execution of event handlers (to identify representative packets that exercise code paths on

the controller). We also present a simplified OpenFlow switch model (to reduce the state space), and effective strategies for generating event interleavings likely to uncover bugs. Our prototype tests Python applications on the popular NOX platform. In testing three real applications—a MAC-learning switch, in-network server load balancing, and energyefficient traffic engineering—we uncover eleven bugs.

OF.CPP: Consistent packet processing for Openflow

This paper demonstrates a new class of bugs that is likely to occur in enterprise OpenFlow deployments. In particular, step-by-step, reactive establishment of paths can cause network-wide inconsistencies or performance- and space-related inefficiencies. The cause for this behavior is inconsistent packet processing: as the packets travel through the network they do not encounter consistent state at the OpenFlow controller. To mitigate this problem, we propose to use transactional semantics at the controller to achieve consistent packet processing. We detail the challenges in achieving this goal (including the inability to directly apply database techniques), as well as a potentially promising approach. In particular, we envision the use of multi-commit transactions that could provide the necessary serialization and isolation properties without excessively reducing network performance.

A distributed and robust SDN control plane for transactional network updates

Software-defined networking (SDN) is a novel paradigm that outsources the control of programmable network switches to a set of software controllers. The most fundamental task of these controllers is the correct implementation of the network policy, i.e., the intended network behavior. In essence, such a policy specifies the rules by which packets must be forwarded across the network. This paper studies a distributed SDN control plane that enables concurrent and robust policy implementation. We introduce a formal model describing the interaction between the data plane and a distributed control plane (consisting of a collection of fault-prone controllers). Then we formulate the problem of consistent composition of concurrent network policy updates (termed the CPC Problem). To anticipate scenarios in which some conflicting policy updates must be rejected, we enable the composition via a natural transactional interface with all-or-nothing semantics. We show that the ability of an f-resilient distributed control plane to process concurrent policy updates depends on the tag complexity, i.e., the number of policy labels (a.k.a. tags) available to the controllers, and describe a CPC protocol with optimal tag complexity f + 2.

Consistency is not easy: How to use two-phase update for wildcard rules?

The recent proposed two-phase mechanism is a provable theory to achieve consistent updates for SDN. However, how to make it work for practical rules is important yet unsolved-(1) two-phase mechanism requires that rules in the new configuration after an update are assigned with a distinct version number from rules in the old configuration before an update; but (2) setting rules in each configuration with a distinct version number causes serious rule-space overheads in practice due to the sophisticated “covered” relationships between practical wildcard rules. In this letter, we design a simple yet generic solution for the problem. By using well-designed wildcard-based version number matchings, we simplify the update procedure, make a stream of updates easy to be processed in parallel, and avoid all unwanted rule-space overheads. We think that our mechanism bridges the gap between the theory of two-phase consistent update and the practical issue of how to use it for today's networks.

A safe, efficient update protocol for Openflow networks

We describe a new protocol for update of OpenFlow networks, which has the packet consistency

condition and a weak form of the flow consistency condition. The protocol conserves switch resources, particularly TCAM space, by ensuring that only a single set of rules is present on a switch at any time. The protocol exploits the identity of switch rules with Boolean functions, and the ability of any switch to send packets to a controller for routing. When a network changes from one ruleset (ruleset 1) to another (ruleset 2), the packets affected by the change are computed, and are sent to the controller. When all switches have been updated to send affected packets to the controller, ruleset 2 is sent to the switches and packets sent to the controller are re-released into the network.

Incremental consistent updates

A consistent update installs a new packet-forwarding policy across the switches of a software-defined network in place of an old policy. While doing so, such an update guarantees that every packet entering the network either obeys the old policy or the new one, but not some combination of the two. In this paper, we introduce new algorithms that trade the time required to perform a consistent update against the rule-space overhead required to implement it. We break an update in to k rounds that each transfer part of the traffic to the new configuration. The more rounds used, the slower the update, but the smaller the rule-space overhead. To ensure consistency, our algorithm analyzes the dependencies between rules in the old and new policies to determine which rules to add and remove on each round. In addition, we show how to optimize rule space used by representing the minimization problem as a mixed integer linear program. Moreover, to ensure the largest flows are moved first, while using rule space efficiently, we extend the mixed integer linear program with additional constraints. Our initial experiments show that a 6-round, optimized incremental update decreases rule space overhead from 100% to less than 10%. Moreover, if we cap the maximum rule-space overhead at 5% and assume the traffic flow volume follows Zipf's law, we find that 80% of the traffic may be transferred to the new policy in the first round and 99% in the first 3 rounds.

Efficient synthesis of network updates

Software-defined networking (SDN) is revolutionizing the networking industry, but current SDN programming platforms do not provide automated mechanisms for updating global configurations on the fly. Implementing updates by hand is challenging for SDN programmers because networks are distributed systems with hundreds or thousands of interacting nodes. Even if initial and final configurations are correct, naively updating individual nodes can lead to incorrect transient behaviors, including loops, black holes, and access control violations. This paper presents an approach for automatically synthesizing updates that are guaranteed to preserve specified properties. We formalize network updates as a distributed programming problem and develop a synthesis algorithm based on counterexample-guided search and incremental model checking. We describe a prototype implementation, and present results from experiments on real-world topologies and properties demonstrating that our tool scales to updates involving over one-thousand nodes.

A two phase multipathing scheme based on genetic algorithm for data center networking

Data centers for cloud computing should allocate services with different traffic patterns, provide high data transfer capacity and link fault tolerance. Data center network topologies provide physical connection redundancy, which forwarding mechanisms avail to generate multiple paths. In this paper, we divide multipathing into two phases: (i) Configuration phase based on genetic algorithms to minimize path lengths and maximize link usage diversity; (ii) Path selection phase based on heuristics to minimize path reuse. The proposed multipathing scheme implements minimal modification in infrastructure. Our proposal only requires common

network devices features and it avoids any tenant modification. We develop a flow simulator to evaluate multipathing techniques. The simulations model flow behaviors in different data center scenarios and compares the proposed scheme with multipathing techniques in literature. The results show the proposed scheme enhances transmission rates, even in the highest network utilization scenarios.

III. PROPOSED METHODOLOGY

We propose the Reverse Update scheme that guarantees a per-packet consistent policy update for Software Defined Networking. The proposed scheme is based on the relaxation of the per-packet consistency concept and on the installation of policy updates in a sequence that corresponds to the reverse path of the flows.

The proof of consistency of the Reverse Update scheme is performed using a formal model of Software Defined Networking. Reverse Update is a policy update scheme for Software Defined Networking that ensures consistency of policy commitment. The key advantage of our proposal when compared with the Two-Phase Update is the lower overhead for configuring, as it does not depend on packet tagging. The Reverse Update is based on updating flow processing and forwarding rules in the reverse path of the already installed flow, to assure that a flow always reaches the most current network configuration. We relax the concept of per-packet consistency. We assume that a packet may be processed by more than one global network configuration if, and only if, it is always processed by the most recent network configuration. In other words, the relaxed concept of per-packet consistency avoids that a packet, which has already been processed by a recent configuration, be processed by a previous configuration in the next hops. The relaxed concept is important to assure that a packet is never forwarded by unexpected network states.

We guarantee this property because every packet that reaches an already updated switch will always be handled by the most recent network configuration, in which the invariant properties are assured. Moreover, the relaxed concept enables the fast deployment of updates, as it updates even the in-transit packets. A packet that travels from the source to the destination should never traverse a switch that still presents a previous configuration state. This occurs because the configuration updates are applied on the reverse sense, from the destination to the source. When using the Reverse Update scheme, every packet will not be processed by a preceding configuration state.

Flow Diagram

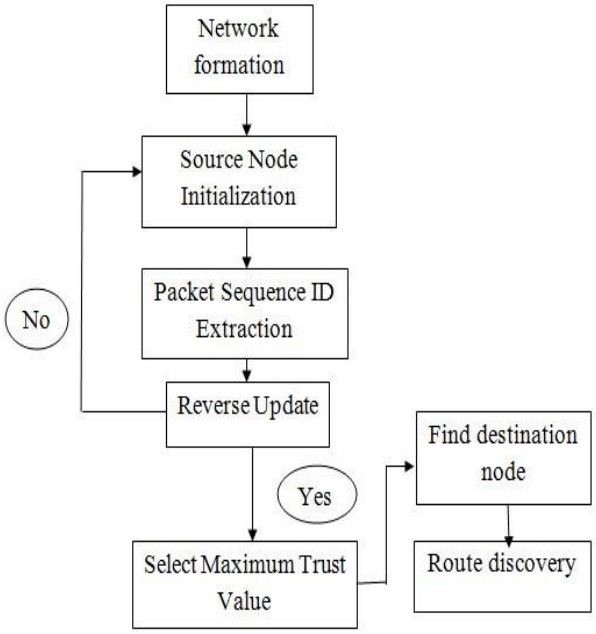

Figure 1 Flow diagram for proposed method

It is worth noting that the definition of a Reverse Update is restricted to policy updates that act on disjoint and loop free paths on the network. Moreover, the composition of new and old paths should be loop-free.

Taking these restrictions into consideration, Reverse Update fits well for path and action updates on the flow paths. Moreover, when considering switches that process packets through multiple tables, the

Reverse Update remains the same, as the processing pipeline acts as a loop-free path inside each switch. The scheme acts in each switch on the reverse path, updating each table in the reverse order of packet-processing pipeline.

Another important consideration is that, as wildcard flow entries have different granularity, policy updates can incur on the definition of overlapping policies. Updating overlapping-policies is a complex challenge. In this paper we assume that updates do not overlap with already defined policies.

IV. EXPERIMENTAL RESULT

We evaluate the proposed Reverse Update scheme by simulating an SDN. In this scheme, the data success route rate is higher, the detection energy consumption is low. We verify that the number of rules installed by the Two-Phase Update is almost eight times higher than other update schemes, due to the addition of new rules on the network-core ports on each switch, for each flow on the network. The Reverse Update only updates the rules, as well as the Ideal Update. We evaluated, for each update scheme, the percentage of forwarded packets that follows the same configuration when compared to the Ideal Update. This metric is important to measure the update proportionality.

V. RESULTS AND DISCUSSION

The reverse update scheme is simulated for 40 nodes spread randomly in a network; transmission range for each node is random. Nodes are positioned randomly on the plane. Nodes start its travel from a random location to a random direction with a random speed.

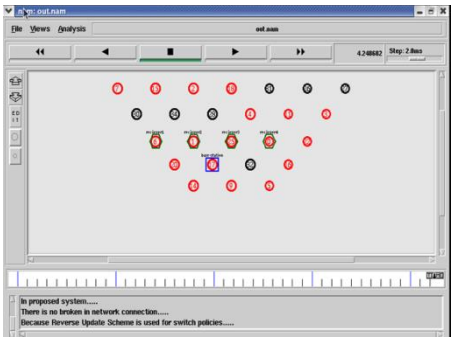

Figure 2. Node Creation

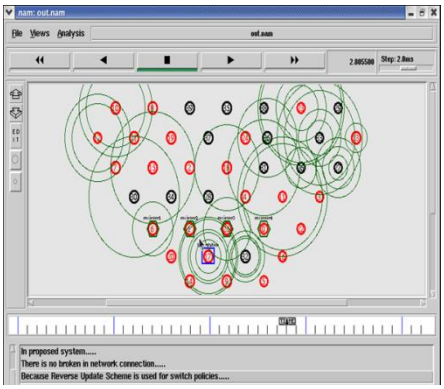

Figure 3. Proposed Scheme

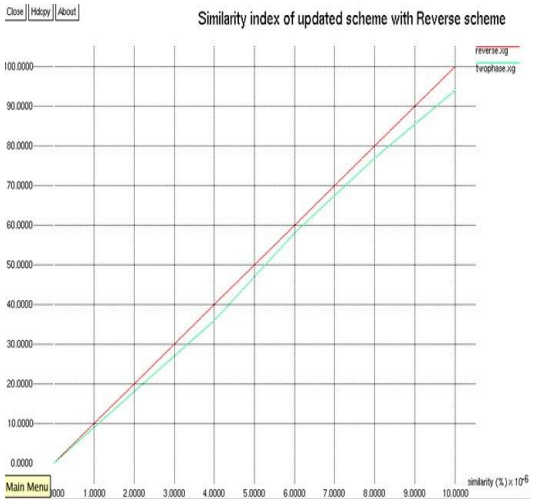

Figure 4. Comparison of Updated Scheme with Reverse Scheme

VI. CONCLUSION

In this letter, we propose the Reverse Update scheme. The proposed scheme updates switch policies, switch-by-switch, on a Software Defined Networking, in the reverse sense of flow paths. We prove that our policy update scheme is per packet consistent and, thus, the flow properties are preserved. It is worth mentioning

that our scheme is simple and does not require packet tagging, which guarantees low processing overhead and reduced number of installed rules on the core of the network. The simulation of Reverse Update scheme in a SDN showed that the configuration overhead is close to an ideal update scheme. Moreover, the Reverse Update promptly updates the rules, presenting a similarity with the Ideal Update of 94%, which is up to four times higher when compared to the Two-Phase Update scheme.

VII. REFERENCES

[1]. M Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in Proceedings of the ACM SIGCOMM 2012. New York, USA: ACM, 2012, pp. 323–334.

[2]. N. C. Fernandes, M. D. D. Moreira, I. M. Moraes, L. H. G. Ferraz, R. S. Couto, H. E. T. Carvalho, M. E. M. Campista, L. H. M. K. Costa, and O. C. M. B. Duarte, "Virtual networks: isolation, performance, and trends," Annals of Telecommunications - Annales des T´el´ecommunications, vol. 66, no. 5, pp. 339–355, 2010.

[3]. M Canini, D. Venzano, P. Pere?s´ini, D. Kosti´c, and J. Rexford, "A NICE way to test openflow applications," in Proceedings of the USENIX NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 127– 140.

[4]. P. Pere?s´ini, M. Kuzniar, N. Vasi´c, M. Canini, and D. Kostiu, "OF.CPP: Consistent packet processing for openflow," in ACM SIGCOMM - HotSDN'13. Hong Kong, China: ACM, 2013.

[5]. M. Canini, P. Kuznetsov, D. Levin, S. Schmid et al., "A distributed and robust SDN control plane for transactional network updates," in The IEEE INFOCOM 2015, Apr. 2015.

[6]. S Luo, H. Yu, and L. Li, "Consistency is not easy: How to use two-phase update for wildcard rules?" Communications Letters, IEEE, vol. 19, no. 3, pp. 347–350, Mar. 2015.

[7]. R McGeer, "A safe, efficient update protocol for openflow networks," in ACM SIGCOMM - HotSDN'12. Helsinki, Finland: ACM, 2012.

[8]. N P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates," in ACM SIGCOMM - HotSDN'13. Hong Kong, China: ACM, 2013.

[9]. J McClurg, H. Hojjat, P. Cerny, and N. Foster, "Efficient synthesis of network updates," in ACM SIGPLAN - PLDI. Portland, USA: ACM, Jun. 2015.

[10]. L. H. G. Ferraz, D. M. F. Mattos, and O. C. M. B. Duarte, "A twophase multipathing scheme based on genetic algorithm for data center networking," in IEEE GLOBECOM 2014, Dec. 2014, pp. 2270–2275.