LODGE: LOcal Decisions on Global statEs in programmable data planes

LODGE: LOcal Decisions on Global statEs in programmable data planes

German Sviridov*, Marco Bonola†, Angelo Tulumello†, Paolo Giaccone*, Andrea Bianco*, Giuseppe Bianchi†
* Politecnico di Torino, Torino, Italy
†University of Rome Tor Vergata, Rome, Italy

***Abstract*—In stateful Software Defined Network (SDN) data planes, network switches hold some local flow-related states thanks to which they are able to perform decisions by locally executing simple algorithms. While stateful data planes provide better reactiveness in respect to vanilla SDN, states are still bounded to single switches which represents a significant restraint for network-wide applications.**

To deal with the current limitations of stateful data planes we introduce and provide design guidelines for LODGE, a model according to which distributed network applications are able to make local decisions at each switche based on some global variables shared across other switches. We describe the implementation of LODGE with a basic application providing support for the detection of Distributed Denial of Service (DDoS) attack in a scenario of stateful data planes involving P4 and Open Packet Processor (OPP) enabled switches. We validate the two implementations in a small emulated testbed and we show the beneficial effects on the reduction of the total network traffic.

I. Introduction

Programmable *stateful* data planes offer an additional level of programmability with respect to the traditional Software Defined Networking (SDN) paradigm. Indeed, OpenFlow, the most widely known protocol for SDN, is stateless at switch level and keeps all the network states in the controller.

Stateful SDN enables the possibility of embedding network applications in the form of elementary states directly inside the forwarding devices, thus allowing them to take local decisions. This approach greatly improves the reactivity for the vast majority of network applications by removing latency overhead previously caused by the interaction with the controller. Recent proposals for stateful SDN have been assuming unique states' location in the network which inevitably leads to: (1) scalability restraints in case of network applications with a global scope; (2) risk of failures that can jeopardize the integrity of the state variable.

With LODGE, LOcal Decisions on Global statEs, we grant the possibility of defining multiple replicas of the same state across different switches, thus providing support for network-wide applications without incurring into drawbacks of classical approaches. Indeed, LODGE prevents losses of state variables in case of isolated faults and at the same time provides better resource utilization by allowing load balancing among different replicas of the same state. Coherently with classical approaches, in LODGE decisions are still *local* to each switch but they are based on a *global* information which is evaluated by combining the state values of all replicas. In order to provide line-rate reaction times, we define and implement a replication scheme that maintains continuous synchronization of all replicas, allowing to evaluate the global information locally at each switch at any given time.

978-1-5386-4633-5/18/$31.00

In our work we consider two different architectures for stateful switches, namely P4 [1] and Open Packet Processor (OPP) [2]. In this context, we provide the following main contributions:

- we propose the state replication in stateful SDN and discuss the main design issues;
- we describe the implementation of an application targeting the detection of a Distributed Denial of Service (DDoSD) attack, through P4 and OPP, leveraging LODGE;
- we experimentally show the beneficial effect of adopting multiple replicas of the states on reducing the total network traffic.

The paper is organized as follows. Sec. II contains an overview of the stateful approaches and introduces P4 and OPP. In Sec. III we discuss the related works. Sec. IV contains the design challenges related to state replication, which are then experimentally validated in Sec. V through a DDoSD application. Finally, we draw our conclusions in Sec. VI.

II. Stateful approaches for SDN

In stateful data planes switches are provided with the possibility of taking local decisions based on internal states without any interaction with the SDN controller. This enhancement enables a wider level of programmability of the network with respect to standard SDN paradigms such as the ones based on OpenFlow. For the purpose of this work we consider two recent proposals for stateful SDN: P4 and OPP.

P4 [1] is a novel data plane programming language which aims to achieve both target and protocol independence, in-field reprogrammability while providing also stateful operations thanks to the presence of persistent memories. Similarly to OpenFlow, P4 exploits a match-action pipeline, thus allowing to define multiple packet processing stages. Consequently the language specifications are designed to allow fast and efficient translation of the programmer-defined features in match-action rules, in order to guarantee low computational overhead and line-rate processing speed. P4 is protocol independent thanks to the presence of a programmable parser and deparser placed at the two extremes of the packet processing pipeline. Due to the parser programmability, it is possible to define custom

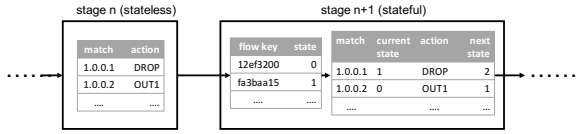

Fig. 1: Example of Open Packet Processor (OPP) pipeline

protocol headers or even extend the parsing/deparsing actions to the packets' payload.

Open Packet Processor (OPP) [2] is a stateful data plane abstraction that allows the definition of per-flow eXtended Finite State Machine (XFSM) directly in the data plane. An OPP pipeline consists of a configurable set of stateful and stateless stages, as shown in the example of Fig. 1. A stateless stage can be seen as a standard OpenFlow/P4 match/action table. A stateful stage instead provides the ability to associate a state shared among all the packets of the same flow, which can be dynamically modified by the data plane itself.

An OPP stage processes each received packet by performing the following operations. First, the packet is associated to a flow key by combining an arbitrary and user configurable set of header fields. The flow key is then used to query a state table and retrieve the so called *flow context*, which consists of a state label and a set of data variables. Second, a set of user-defined algebraical conditions combing the values in the flow context and in the global context is evaluated and an array of the results of such conditions is stored in the packet metadata. The packet header along with the flow context and the condition array is processed by a match/action table that, in addition to a set of packet actions, associates to the packet a new label state and a set of flow context update functions, and thus implements an Extended Finite Set Machine (EFSM).

III. Related work

SNAP [3] is a novel network programming abstraction, which allows to define quite complex network applications for stateful SDN and solves the problem of how to optimally place the states across the network switches, taking into account the dependency between states and the traffic flows. By design, SNAP is limited to just one replica of each state within the network. LODGE, instead, enables multiple replicas of the state, extending the single replica approach in SNAP.

The most relevant work to the state replication in stateful data planes is Swing State [4], which introduces a mechanism providing state migrations entirely in the data plane but, similarly to SNAP, assumes only one state that is on demand migrated across the network.

Regarding the implementation of stateful SDN, OpenState [5] proposed a minimal architectural extension to the Openflow data plane and control plane to identify the flow to which a packet belongs to and to retrieve/update the associated state. Open Packet Processor (OPP) [2] extended OpenState by adding additional features that allow the executions of Extended Finite State Machines (EFSM) directly in the data plane. FAST [1] proposes a switch chip implementation based on the Reconfigurable Match Tables (RMT) model that permits, even if with some limitations indicated by the same authors, to manipulate some state within its pipeline. Finally, Domino/Banzai [6] proposes both a domain specific language and a data plane architecture for designing and implementing line-rate stateful processing. To read/write states, a set of specialized stateful processing instructions are executed within the switch pipeline.

NetPaxos [7] provides application layer acceleration for Paxos protocol by offloading parts of it to the network. Differently from NetPaxos, our work focuses on providing a mean for line-rate state replication directly in the dataplane.

IV. LODGE design and challenges

Assume that S is the number of states replicated in C copies in each of N switches in a network. The overhead of LODGE in terms of per-switch memory occupancy grows as $O(SC)$. The actual amount of memory depends on the adopted data structure.

The implementation of the state replication system needed for LODGE presents numerous design challenges that depend on the employed hardware architecture and on the considered programming model.

A. Consistency requirements

Consistency among replicas belonging to the same global state is required in order to guarantee correct functionality of the network application. The CAP theorem [8] states that for a replication scheme out of Consistency, Availability and Partition tolerance only two properties can be picked at the same time. Considering that network failures may occur, partition tolerance cannot be left out of the design of our replication algorithm, leaving us with two main reference models:

- *Strong Consistency*. This model privileges consistency over availability, meaning that a read operation on any non-faulty replica will return the most recent committed value (same for all replicas) or an error. This property is achieved at the cost of reduced availability due to the requirement of multiple interactions between replicas and is based on complex consensus algorithms [9], [10].
- *Eventual Consistency*. This model privileges availability and results in instantaneous operations on all replicas with a considerably reduced protocol complexity. Although it introduces transient inconsistencies this inconsistency can be seen as an error in the value of a local replica.

The choice among the two models depends on the level of tolerance of the considered application in the presence of inconsistencies, when evaluated and updated in a distributed fashion. Many distributed applications, as the one considered in the following, require small packet processing latencies, thus high availability when state changes occur. Furthermore, they remain robust even in the presence of a (bounded) error for the current value of a local state. Due to the necessity of having latency guarantees, in the following we will consider

specifically the implementation of an eventual consistency model for the design of a state replication scheme.

B. Synchronization traffic generation

In order to provide state synchronization, the switch must generate ex-novo update packets and this is currently not supported in available hardware architectures. Depending on the actual architecture we foresee three different scenarios in order to achieve this goal:

1) Self-triggered updates: The generation of an update packet is triggered periodically by the internal clock of the switch. This capability is typically not available in high-performance switches, since a new packet should be inserted in the internal hardware pipeline, introducing a performance degradation. Indeed, this approach is not available neither in P4 or OPP.

2) Controller-triggered updates: The generation is triggered by the controller. In the case of periodic updates, the controller sends periodic trigger messages that are processed at the switch, and modified to generate the update packets. However the required control bandwidth from the controller to each switch can grow large for small update periods and at the same time the controller is loaded with an additional task, impairing its scalability. Moreover the actual trigger arrival time highly depends on the instantaneous latency from the controller to the switch and thus this approach allows to control only the average trigger arrival time, whenever the network conditions are stationary.

3) Traffic-triggered updates: The update packet generation is triggered directly by the data packet reception at the switch. This allows to self-adapt the amount of generated synchronization traffic based on a temporal window comparable with the states' evolution rate which in turn is proportional to the rate of new incoming packets. We consider two possible approaches to tune the synchronization rate, both based on simple internal states available in the stateful SDN switches:

- packet period p. By keeping a packet counter, a new update packet is generated every p received packets. Thus the update rate is proportional to the traffic. We opt for this approach for P4, as discussed in Sec. V-A, for our DDoSD application.
- time period δ. An update packet is generated at the first packet arrival after δ seconds. This results in a fixed synchronization rate independent from the traffic. It is obtained as follows. After generating an update packet, the switch schedules a change of a binary state at δ seconds in the future. Consequently, the first packet arriving after δ seconds will trigger the generation of the update packet. This solution is adopted in our OPP implementation, as discussed in Sec. V-B.

C. Format for state updates.

The update packet carries the state identifier, the state value and the identifier of the switch originating the update. We assume that all identifiers are pre-established by the controller during network instantiation. This mechanism allows to provide guarantees on the state uniqueness, it provides flexibility in term of coding the state format, and provides support for different switching architectures by imposing a common format. Since the definition of the optimal format is outside the scope of our work, we consider a naïve format with a integer value to identify the state.

D. Routing for update packets

Multiple mechanisms to propagate state updates in networks have been proposed (a comprehensive survey is available in [11]) although they all assume a distributed system without a centralized knowledge of the network and its condition. In the considered SDN scenario, we can exploit the controller's knowledge to install updates' forwarding rules through a multicast distribution tree, either shared across all the states or one specific for each state. Our implementations are adopting this approach.

The overhead in terms of bandwidth needed for state replication grows as $O(SCN)$, which corresponds to $O(SC)$ per node bandwidth. The exact evaluation of the bandwidth highly depends on the network topology, the states' placement and the topology adopted for the replication.

V. DDoSD in P4 and OPP

We validate the effectiveness of the proposed state replication approach in a DDoSD use case. As shown in Fig. 2, we assume that a large network (e.g., an Autonomous System - AS) is connected to other networks (e.g., other ASs) through different edge routers and the attack is targetting a set of internal servers. The main idea of DDoSD is to exploit the typical temporal correlation between the variation of traffic across all the edge routers, due to the distributed nature of the attack. To solve this problem through stateful switches, the total traffic load entering the whole network and directed toward the targeted servers is defined as a *single global state* variable within the AS. This state is distributed across a set of stateful switches (SW1-SW4 in our reference topology) and can be obtained by summing the traffic measurements performed locally at each switch. Notably, DDoSD is robust to possible transient inconsistencies between the values of total traffic estimated at each switch, thus an eventual consistent model is adopted for the state consistency.

Stateful switches are programmed to perform the following tasks: (1) measure locally the data rate of TCP connections toward the server clusters; (2) generate, forward and process the synchronization traffic used to share the local measurement; (3) enforce some mitigation countermeasures if a DDoS attack is detected. Since the definition of a realistic DDoSD algorithm is a well known problem in the literature and it is completely out of the scope of this work, we employ a simple proof-of-concept threshold-based detection scheme, which can be used as a foundation for more sophisticated DDoSD mechanisms. A DDoS attack is detected locally at each switch if $s_G > s_{\text{thr}}$ where s_G is the global state with the total traffic traffic directed to the server clusters in the whole network, and s_{thr} is the traffic

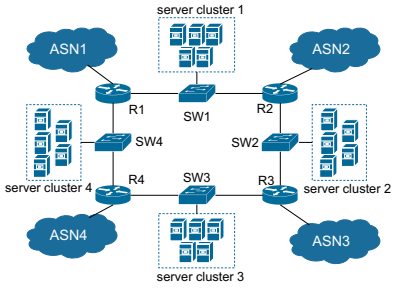

Fig. 2: Reference topology for DDoS Detection use case.

threshold above which the attack is considered as detected, whose value is determined with standard test-based statistical methods. Let C be the number of replicas of the global state. s_G is estimated at each switch by summing s_L, i.e. the corresponding local measured traffic, and all the $C-1$ values of $s_R^{(i)}$, i.e., the remotely measured traffic at the other switches, whose values are obtained through the update packets:

$$s_G = s_L + \sum_{i=1}^{C-1} s_R^{(i)} \tag{1}$$

Although the traffic is routed through one replica of the state, it may occur for it to traverse multiple replicas of the same state. In order to avoid double-counting in our DDoSD application, we flag the data packets once they have been counted at one of the stateful switch. In both implementations we exploit the two unused bits in the IP DSCP header in order to perform such tagging.

The considered DDoSD scheme has been implemented on top of two different programmable data plane platforms: (1) P4 version 1.0.4 ($P4_{14}$); (2) Open Packet Processor (OPP), as described in the following.

A. P4 implementation

Our prototype is developed and tested in a virtual environment using Mininet [12] and P4-enabled virtual switches targeting the *Simple Switch* architecture[1]. Notably, the language specifications do not provide any support for complex operations such as multiplications and divisions.

We assume to know the rate with a sampling window equal to δ_r s. Let r_n be estimated rate in the time interval $[n\delta_r, (n+1)\delta_r)$. The average rate is estimated as

$$s_L(t) = \frac{1}{w} \sum_{i=0}^{w-1} r_{n-i}, \qquad \text{for } t \in [n\delta_r, (n+1)\delta_r) \tag{2}$$

where w is a power of 2 in order to be implemented with a basic shift operation, supported by P4. The w most recent samples of r_n are stored in a circular buffer.

We define an ad-hoc layer-3 packet format to carry the updates in a standalone Ethernet frame, to minimize the synchronization traffic overhead. The Ethernet type field is set to a currently unused one in order to allow unequivocal identification of update packets. This layer-3 packet encloses the identifier of the switch originating the update, the identifier of the state (coded with an integer value) and a variable length field containing the state updated value.

We implement traffic-triggered updates, with packet period p computed by considering only the packets destined to the target servers. This is achieved by defining an additional counter storing the number of matched packets since the last update. The p^{th} packet is then exploited in order to trigger the generation of a clone of the arrived packet. While the original packet is moved to the egress stage, the clone is processed through an additional stage that substitutes the layer-2 payload with the previously defined layer-3 packet containing the most recent value of the local replica $s_L(t)$.

The update distribution is performed through a spanning distribution tree, in a fashion similar to [13]. The distribution tree is shared by all the replicas and programmed by the controller during application instantiation. For each update a multicast is performing on all ports belonging to the distribution tree and the packet is dropped before being emitted if it is destined to the original arrival port. This is made possible by passing ingress port metadata to the output stage and by comparing it with the egress port metadata.

B. Open Packet Processor (OPP) implementation

The DDoSD application implemented in OPP requires a sequence of three stages: stage 0 extracts the state from update messages and detects flagged packets (to avoid double counting); stage 1 stores the state from the metadata notified by the previous table, performs monitoring and detection and generates update messages; stage 2 performs simple L3-forwarding. Stage 0 represents the stateful processing core of local and global states. The processed flows are identified by the IPv4 destination addresses of the target servers. Stage 0 also considers C flow data variables containing the switch's own local state s_L and the $C-1$ remote states to compute (1). Local state s_L is computed by employing of a hardware-implemented Exponential Weighted Moving Average (EWMA) of the number of packets measured in a given preconfigured time window. Whenever an update packet is received carrying $s_R^{(i)}$, the global s_G is computed by (1) and compared with s_{thr}.

As for the state synchronization, an update message is triggered at every computation of the EWMA. This update message is a simple packet with IP and MPLS header where the label contains the state identifier, the state value, and the sender switch, as in the P4 implementation.

C. Experimental evaluation and validation

We implement and evaluate both P4 and OPP solutions for DDoSD. In more detail, we configure a Mininet-based emulation environment deploying the topology shown in Fig. 2, where, for the sake of simplicity, each cluster and each AS is represented by a Mininet host. To simulate the DDoS attack, we use `hping3` tool to send TCP SYN requests from all ASs to all internal servers. In each experiment, during the

[1]https://github.com/p4lang

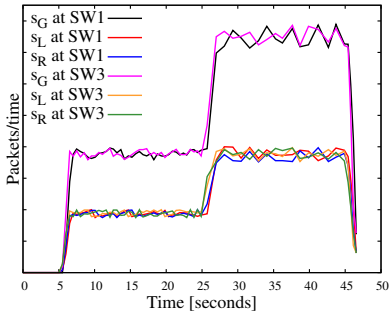

Fig. 3: Temporal evolution of the local, remote and global states for the stateful switches in case of 2 replicas for the global state in P4 implementation.

first 20 seconds, we send the request at a slow rate, and then we increase the rate in a such a way that the global variable s_G exceeds the threshold. We set the replicas of the states: (i) in SW1 for 1 replica ($C = 1$), (ii) in SW1 and SW3 for 2 replicas ($C = 2$), and (iii) SW1, SW2, SW3, SW4 for 4 replicas ($C = 4$).

Fig. 3 shows the evolution of s_G, s_L and s_R for the case of 2 replicas, implemented in P4. Identical results are obtained with OPP and thus are not reported for the sake of space. As expected, the values of s_G evaluated at SW1 and SW3 are coherent, and allow a contemporary detection of the DDoS attack in the two stateful switches, without any interaction with the controller. This experimental result validates our proposed implementation for both P4 and OPP.

In Figs. 4 and 5, instead, we show the utilization of the links present in the ring topology connecting all the stateful switches, for different values of C, under P4 and OPP implementations. Clearly, for one replica the load on the link is greatly unbalanced and in general higher for all the links. The different behavior between P4 and OPP for some links (e.g., SW1-R2) are due to the different routing schemes adopted in the topology. By increasing the number of replicas to 2, the load of the data traffic decreases by a factor of 1.6 both in P4 and OPP and is much better balanced across the links. The slightly different values depends on the different mechanisms adopted for triggering the update event by the incoming traffic: in P4 the update rate depends on the traffic, whereas in OPP it is independent. Adding two other replicas reduces the data traffic by around 20% in both implementations, but now the update traffic becomes more relevant due to the higher number of replicas to synchronize. Indeed, the fraction of update packets increases from 14% (for 2 replicas) to 24% (for 4 replicas) in P4 and from 11% (for 2 replicas) to 23% (for 4 replicas) in OPP. Thus, the two implementations behave very similarly and show the beneficial effect on the overall traffic in the network due to multiple replicas.

VI. Conclusions

We investigated how to enable local decisions in stateful switches based on some global state. Our proposal, named

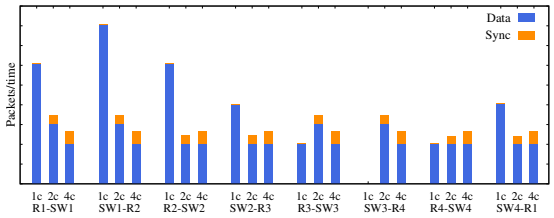

Fig. 4: Link occupation for data and synchronization traffic in case of $1, 2, 4$ replicas for global state in P4 implementation.

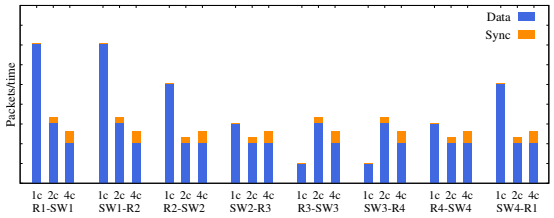

Fig. 5: Link occupation for data and synchronization traffic in case of $1, 2, 4$ replicas for global state in OPP implementation.

LODGE, provides support for distributed applications that requires coordination among switches. In our work we discuss the main practical design issues for state replication, whose implementation was validated using P4 and OPP stateful data planes. Our results show that state replication can be beneficial for the network performance and can be efficiently implemented in high-performance programmable stateful switches.

References

[1] P. Bosshart and al., "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *ACM SIGCOMM CCR*, 2013.

[2] M. Bonola, R. Bifulco, L. Petrucci, S. Pontarelli, A. Tulumello, and G. Bianchi, "Implementing advanced network functions for datacenters with stateful programmable data planes," in *IEEE LANMAN*, 2017.

[3] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker, "SNAP: Stateful network-wide abstractions for packet processing," in *ACM SIGCOMM*, 2016.

[4] S. Luo, H. Yu, and L. Vanbever, "Swing State: Consistent updates for stateful and programmable data planes," in *ACM SOSR*, 2017.

[5] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: Programming platform-independent stateful Openflow applications inside the switch," *ACM SIGCOMM CCR*, Apr. 2014.

[6] A. Sivaraman and al., "Packet transactions: High-level programming for line-rate switches," in *ACM SIGCOMM*, 2016, pp. 15–28.

[7] H. T. Dang, D. Sciascia, M. Canini, F. Pedone, and R. Soulé, "NetPaxos: Consensus at network speed," in *ACM SIGCOMM SOSR*, New York, NY, USA, 2015.

[8] E. Brewer, "CAP twelve years later: How the "rules" have changed," *Computer*, Feb 2012.

[9] L. Lamport *et al.*, "Paxos made simple," *ACM Sigact News*, 2001.

[10] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm." in *USENIX Annual Technical Conference*, 2014.

[11] P. Jesus, C. Baquero, and P. S. Almeida, "A survey of distributed data aggregation algorithms," *IEEE Communications Surveys & Tutorials*, 2015.

[12] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *ACM SIGCOMM HotNets*, 2010, p. 19.

[13] D. Farinacci, C. Liu, S. Deering, D. Estrin, M. Handley, V. Jacobson, L. Wei, P. Sharma, D. Thaler, and A. Helmy, "Protocol independent multicast-sparse mode (pim-sm): Protocol specification," 1998.