VNF-Consensus: A Virtual Network Function for Maintaining a Consistent Distributed SDN Control Plane

Giovanni Venâncio[*], Rogério C. Turchetti[†], Edson T. Camargo[‡] and Elias P. Duarte Jr.[*]
[*]Federal University of Paraná, Curitiba, Brazil
[†]Federal University of Santa Maria, Santa Maria, Brazil
[‡]Federal Technological University of Paraná, Toledo, Brazil
Emails: gvsouza@inf.ufpr.br, turchetti@redes.ufsm.br, edson@utfpr.edu.br, elias@inf.ufpr.br

***Abstract*—Software Defined Networks (SDN) usually rely on a centralized controller, which has limited availability and scalability by definition. Although a solution is to employ a distributed control plane, the main issue with this approach is how to maintain the consistency among multiple controllers. Consistency should be achieved with as low impact on network performance as possible, and should be transparent for controllers, without requiring any change of the SDN protocols. In this work we propose VNF-Consensus, a Virtual Network Function that implements Paxos to ensure strong consistency among controllers of a distributed control plane. In our solution, controllers can perform their control plane activities without having to execute the expensive tasks required to keep consistency. Experimental results are presented showing the cost and benefits of the proposed solution, in particular in terms of low controller overhead.**

I. Introduction

Software Defined Networks (SDN) separate the control plane from the data plane, which improves their flexibility, programmability, and management [1, 2]. The control plane is usually centralized, consisting of a single controller, while the data plane consists of numerous network devices distributed across the network. While a centralized approach is attractive as it is simpler to operate and manage, it represents a vulnerability as the controller is a single point of failure with a direct impact not only in terms of resilience (i.e. if a controller fails there is another to keep the network running), but also on performance and scalability [3] (e.g. it allows load sharing as the number of flow requests grows). The solution is to distribute the control plane, employing multiple controllers that share responsibilities [4]. Several different strategies have been proposed to distribute the SDN control plane [5, 6].

The advantages of employing multiple controllers should thus be clear. However there is a cost: in order to employ a distributed control plane it is necessary to employ techniques to guarantee the consistency among the multiple controllers [7]. Actions performed on the control plane by multiple SDN controllers need to be synchronized, and this is not a trivial endeavor [5]. An alternative is to employ a straightforward master-slave architecture [8]. Consistency violations can cause problems such as for instance the installation of conflicting forwarding rules on multiple SDN switches, which may result in the creation of loops or routes bypassing important services.

Canini et al. [9] argue that maintaining a consistent distributed control plane is one of the main open problems in SDN. This has to be solved without causing a significant impact on network performance and obviously preserving the correct operations executed on the data plane. Some of the existing solutions for building a distributed and robust SDN control plane are done at the control plane itself. In this case, the controllers themselves incorporate new features in order to synchronize the control plane. The major drawback of this approach is the overhead it represents on the controllers themselves [7, 9, 10]. In contrast, there exist other solutions that avoid increasing controller workload by having switches synchronizing their actions [5, 11]. In general, these solutions have disadvantages as they require modifications of the SDN protocol.

In this work, we propose a solution for keeping the consistency of a distributed SDN control plane that takes advantage of NFV (Network Function Virtualization) [12] technology and which is particularly efficient in terms of the load it imposes on the controllers. NFV enables the implementation in software of network services that run in the network core and can be executed on off-the-shelf hardware. Our solution relies on a Virtual Network Function (VNF) called *VNF-Consensus* that implements Paxos [13] to guarantee the strong consistency of the distributed control plane synchronizing the actions performed by the multiple SDN controllers. In this way, the execution of any operation on different controllers always leads to the same result. Furthermore, our *VNF-Consensus* inherits Paxos properties: safety is guaranteed even if the system is asynchronous and liveness is guaranteed despite controller failures.

In order to synchronize the actions among all controllers our solution enables each controller to access a *VNF-Consensus* instance which is executed on a separate host from the controllers. This instance allows an SDN controller to receive decisions and send actions to be synchronized. Note that all decisions handled by *VNF-Consensus* are systematically performed without the direct participation of the controller. The advantage of the proposed strategy is that *VNF-Consensus* maintains the consistency of a distributed control plane leaving

978-3-903176-23-2 © 2019 IFIP

controllers free to perform their regular control plane activities. As a consequence, the proposed strategy does not increase the load on the controllers and on the computational resources of the controllers, as *VNF-Consensus* is executed on a separate host. We note that *VNF-Consensus* can keep the control plane consistent regardless of the number of controllers. Finally, it is important to note that our strategy can be implemented without any changes to the SDN protocol nor the switches.

Experimental results are reported and show the performance improvements that can be obtained by using *VNF-Consensus* without the direct participation of SDN controllers. In particular, we show that it is possible to synchronize the control plane without increasing the controller load. Results also show that this strategy performs significantly better than having the controllers themselves responsible for synchronizing and guaranteeing the consistency of network operations.

The rest of this work is structured as follows. Section II gives an overview of strategies that enable the synchronization of the control plane in SDN networks. Section III describes the proposed architecture. The experimental results are described in Section IV. Section V concludes the work.

II. Related Work

In this section we describe some of the main strategies previously proposed for the synchronization of multiple SDN controllers. The major difference between these strategies and our solution is that we support consistent synchronization in a distributed control plane using a virtual network function.

A synchronization framework for control planes based on atomic transactions is proposed in [5]. Switches are synchronized in order to guarantee the consistency of network operations. In particular, the authors propose synchronization primitives which allow a controller to represent multiple data plane configuration commands as an atomic transaction. The framework modifies the OpenFlow protocol [16] in order to avoid the controllers to install inconsistent rules on the switches. The authors also report an implementation to show the efficiency of the proposed solution.

Another proposal explores the implementation of the Paxos consensus algorithm on SDN switches [11]. The authors describe two different approaches: the first involves implementing the full Paxos logic, that is, without any optimization; the second implements an optimistic protocol called NetPaxos which does not require the Paxos coordinator. The authors claim that implementing consensus within the switches reduces the complexity, and message latency, and increases transaction throughput. A major disadvantage however is that in order to run Paxos on switches firmware modifications are required.

In order to reach a consistent state among SDN controllers a version of the Paxos consensus algorithm called Fast Paxos-based Consensus (FPC) is proposed in [10]. FPC is implemented in SDN controllers. According to the authors, FPC is less complex than the original Paxos algorithm; FPC does not have a predefined coordinator. Any FPC process can become a leader (called chairman). Once all FPC processes (controllers) have performed an update, the chairman changes its role and finishes the consensus round. The authors compare FPC to the Raft [17] consensus algorithm and conclude that FPC is faster.

Onix [7] is a platform that implements a distributed SDN control plane that maintains a global network view. Onix runs on a cluster of physical servers. A controller stores the network state in a data structure called NIB (Network Information Base). Network control applications are implemented by reading and writing to the NIB. Onix replicates and distributes the NIB among multiple running network instances. Onix employs ZooKeeper [18] to synchronize the multiple instances.

Another proposal [9] defines a formal model to describe the communication between the data plane and a distributed control plane. The distributed control plane consists of a set of controllers which can fail by crashing. The authors address the consistency problem which occurs when network polices are updated at one or more switches. Informally, they guarantee that every packet traversing the network must be processed by exactly one global network policy, even when the network policy itself is updated. An algorithm allows the controllers to directly apply their updates on the data plane and resolve conflicts. A protocol based on the state machine approach is proposed in order to implement a total order on policy updates.

The OpenDayLight (ODL) [19] control plane allows the synchronization of multiple SDN controllers. A “clustering” strategy is defined that allows each controller to store data locally. Data replication is based on the concept of distributed caches. The Raft consensus algorithm is employed to ensure consistency across multiple controllers. Another clustering strategy has also been proposed for the ONOS SDN controller [14]. The controllers themselves implement the consensus algorithm.

An adaptive strategy that employs on line clustering techniques to allow tunable consistency levels is presented in [15]. The strategy receives as input an indicator of application performance and then selects a consistency level indicator. Empirical results show that the strategy is feasible, an experiment is described that maps performance indicators of a load balancing application to different consistency levels.

Table I compares the multiple strategies described in this section. A major difference between our proposal and the other strategies is *where* the tasks for keeping the consistency of the distributed control plane are executed: as presented in the next section, we decouple the consensus algorithm from the controllers to avoid extra controller overhead. With this approach, consistency is provided as a service to the controllers.

III. A VNF to Keep the Control Plane Consistent

In this section, we first give a brief overview of NFV, after which we define the problem of building a consistent SDN control plane. Finally, we describe the architecture of our VNF-based solution to solve the problem.

A. Network Function Virtualization

Currently, most networks employ multiple types of middleboxes which are implemented in hardware, e.g. gateways, firewalls, intrusion detection and prevention systems, traffic

TABLE I
COMPARISON OF THE STRATEGIES.

Related Work	Algorithm for Maintaining the Consistency	Algorithm Executed by
[5]	Atomic transactions using *compare-and-set* (CAS)	SDN switch
[11]	NetPaxos consensus algorithm	SDN switch
[10]	Fast Paxos-based consensus algorithm (FPC)	Controller
[7]	Zookeeper tool	Controller
[9]	Policy Serialization algorithm based on state machine replication	Controller
ODL Platform	Raft consensus algorithm	Controller
[14]	ONOS Clustering	Controller
[15]	Apache Cassandra	Controller

shapers, among several others. Middleboxes represent an important fraction of network OPerational EXpenditures (OPEX) and CAPital EXpenditures (CAPEX), being usually expensive to deploy, manage, and complex to troubleshoot [20].

Network Function Virtualization (NFV) has been proposed as a way to address these challenges by leveraging virtualization technologies to offer a new way to design, deploy and manage networking services [12]. In particular, NFV employs virtualization to deploy network services as software that runs on commodity hardware. A middlebox or any network service is implemented as a software instance which is called a VNF (Virtual Network Function). NFV technology is usually implemented taking advantage of another related technology: Software Defined Networking (SDN) [21]. Using NFV it is possible to create, deploy and manage middleboxes in a fraction of the time required for hardware counterparts.

B. Problem Description

A consistent distributed SDN control plane guarantees that a given sequence of operations issued by a set of controllers are eventually executed in the same order. Controllers issue network operations concurrently and must synchronize the network operations executed. Operations change the network state, for example by installing rules that establish new routes or rules for packet filtering. Keeping a distributed control plane consistent is thus essential to avoid pathological scenarios that result from inconsistent configurations.

Traditionally, an SDN switch communicates with a controller and the controller manages the switch. Decisions taken at the control plane often imply changes on the data plane. The controller adds a flow entry to the switch flow table both reactively, in response to a message from the switch, and proactively [16]. Two types of communication strategies are described for SDN components [22]: Switch-to-Controller and Controller-to-Controller, both are described next.

Switch-to-Controller communications support the interaction between a switch and a controller. This occurs for example when an OpenFlow switch forwards a *packet_in* message to the controller when there is no match in the switch's flow table. In response, the controller returns a *flow_mod* message to allow or deny the installation of a new flow entry. When multiple controllers are employed on a distributed control plane, a decision to add a new flow entry must be synchronized among all controllers. This is necessary to avoid network misconfigurations, such as inconsistent routes that are only partially defined and may cause packets to be dropped or result in loops. It is of course also necessary that switches install *flow_mod* messages in their flow tables in a consistent way. However, it can be complex and expensive to guarantee that updates of the data plane are always consistent. Furthermore several issues have to be dealt with, such as the fact the real networks are not synchronous, in the sense that an upper bound on message transmission delay cannot be always forecast. In this paper we propose a strategy to build a consistent control plane that guarantees the synchronization of the data plane.

Controller-to-Controller communications allow the direct interaction among controllers. To achieve controller-to-controller consistency, controllers are required to share the same network state view. The consistency can be either strong or eventual [22]. Both guarantee the consistency of write operations, which alter the state. On the other hand, strong consistency implies that a given read operation executed by any controller always leads to the same result, while eventual consistency allows different results (for a short period of time). In the proposed architecture we use the Paxos consensus algorithm which provides strong consistency.

C. VNF-Consensus

VNF-Consensus implements Paxos, which is briefly described next. Paxos is a distributed consensus algorithm designed for state machine replication [13]. Informally, consensus allows a set of processes that propose different initial values to decide (i.e., agree on) one of those values. Paxos is safe under asynchronous assumptions, live under weak synchronous assumptions, ensures progress with a majority of non-faulty processes, and assumes a crash-recovery failure model.

Paxos distinguishes the following roles that a process can play: *proposers* propose a value, *acceptors* choose a value, and *learners* learn the decided value. A single process can assume any of those roles, and multiple roles simultaneously. Paxos is resilience-optimum: to tolerate f failures it requires $2f + 1$ acceptors – that is, to ensure progress, a quorum of $f + 1$ acceptors must be non-faulty. To solve consensus, an instance of Paxos proceeds in rounds of two phases each. One of the proposers is elected as the coordinator, which receives a value and submits that value on a consensus round. In the first phase the coordinator proposes a unique round number. When a quorum of acceptors accept that round number, this means that they will not accept any proposal with a lower round number. Consensus is reached in the second phase as the value associated with the largest round number is accepted by a quorum of acceptors. After consensus is reached, learners get to know the decided value.

In our proposed strategy, consensus is executed for each OpenFlow rule. After a decision is reached by *VNF-Consensus*, each controller receives the decided value and installs the corresponding rule on the data plane. This can be done by letting each SDN controller behave as a *learner* of the Paxos algorithm. As Paxos provides strong consistency, all controllers eventually will have the same set of rules.

In order to synchronize the network state, each controller communicates with a *VNF-Consensus* instance. A controller both receives decisions from *VNF-Consensus* and sends actions to be synchronized. Note that all decisions taken in the context of *VNF-Consensus* are made *outside* the controller, i.e. the process does not use controller resources as it runs as an independent VNF. After a decision is reached, the controller executes the action received from *VNF-Consensus*.

Figure 1 shows the interaction of controllers with *VNF-Consensus*. When a new network rule needs to be synchronized, the controller forwards it to *VNF-Consensus*. Meanwhile, the controller can keep on handling requests from the SDN network. After a decision has been taken, all controllers receive the final result and then update the state as required. Although Figure 1 shows several VNF instances, a single instance can be employed by all controllers, or more instances as dictated by scalability requirements.

Consider a host in Figure 1. Assume that this host has just started sending a packet flow which reaches an OpenFlow switch. The OpenFlow switch maintains a flow table and all packets it receives are compared against the flow table entries. When a switch receives the first packet of a given flow, if a matching entry is not found in the flow table, a *packet_in* message is sent to the controller. In the proposed strategy, after the controller receives this message from the switch, the corresponding rule is forwarded to *VNF-Consensus* before it is installed. A *VNF-Consensus* instance receives the rule, executes the consensus algorithm and sends the decision back to the controllers, which install the decided rule.

Each time a new *packet_in* message arrives at the controller, it must be redirected to *VNF-Consensus*. In order to do check packets and do the redirection when required, we employ a Filter module to classify and forward packets. The classification is done by matching each packet against a set of rules that determine the destination, e.g. *VNF-Consensus* (Figure 2). The *Filter* consists of a *Classifier* and a *Forwarder*. *Classifier* is a module that performs packet classification using the stored rules. After the classification, the traffic can be forwarded along the corresponding path by the *Forwarder* module. Figure 2 shows the interaction of these modules: packets arrive at the *Filter* (step 1) from the controller. If a match is found (step 2) then the packet is sent to the *Forwarder* module (step 4). Otherwise the packet is returned to the controller to take another action (step 3). The *Forwarder* module is responsible for delivering traffic to a specific VNF, in this case VNF-Consensus (step 5). Finally the traffic is forwarded from the VNF to the controller (step 6).

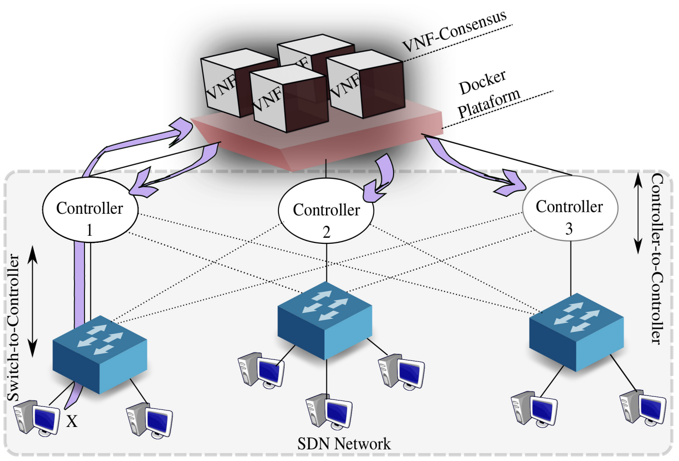


Fig. 1. A SDN with *VNF-Consensus*.

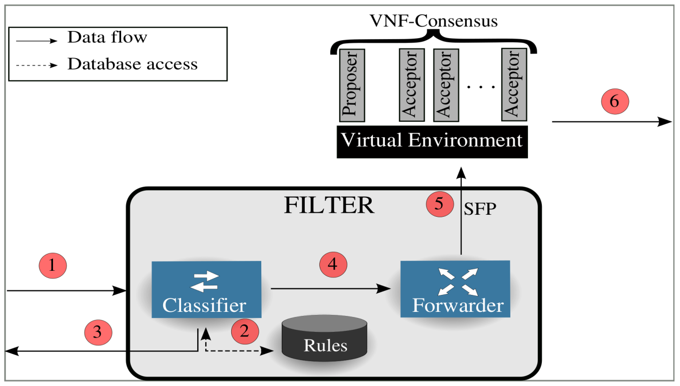


Fig. 2. Architecture: synchronization of the control plane.

In *VNF-Consensus* the Paxos proposer is also the coordinator. Remember that consensus is executed to guarantee the consistency of rules in the network. The coordinator receives a rule from the *Forwarder* module and starts the execution of consensus, which consists of two phases as described next.

In the first phase, the coordinator selects a unique round number and sends a *prepare* request with this round number to the acceptors. Upon receiving a prepare request with a round number that is larger than any round number previously received, the acceptor sends a reply to the proposer promising that it will reject any future requests with smaller round numbers. However, if the acceptor had already accepted a rule, this rule is returned to the proposer. After the coordinator has received positive replies from a quorum of acceptors, it proceeds to the second phase.

In the second phase, after the coordinator has received responses to its prepare requests from a quorum of acceptors, it sends an *accept* request to each of those acceptors. Each of the acceptors then sends an acknowledgement to the coordinator and to the learners, unless the acceptor has already received yet another request with an even higher round number in its first phase. When a quorum of acceptors confirm the *accept* request, consensus is reached.

After *VNF-Consensus* decides on a rule, that rule is delivered to the SDN controllers, which are the learners of our Paxos implementation. Each controller can then issue the network updates accordingly.

IV. Experimental Evaluation

In this section we report results of several experiments executed to evaluate *VNF-Consensus*. The experiments were executed on 32 Intel Core i7 processors at 2 GHz each with 4 cores and running Ubuntu 18.04. Ryu[1] controllers were employed. libPaxos[2] was the Paxos library employed. *VNF-Consensus* uses a REST interface (*REpresentational State Transfer*) to communicate with the controllers.

Each *VNF-Consensus* instance is hosted on a container using the Docker platform. Note that each rule causes an individual instance of Paxos to be executed. As result, we have totally independent and isolated VNFs. We also hosted the Ryu Controller on a container.

Most of the experiments in this section compare the VNF approach (*VNF-Consensus* curve) with the execution of consensus on the SDN controller itself (Consensus on Controller curve). The consensus on controller approach was implemented by having the Paxos algorithm run as a process directly accessed by the controller, both in the same container.

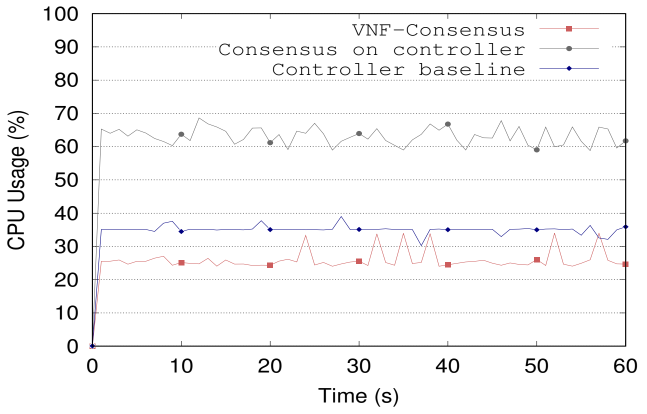
Results are reported for four sets of experiments. The first set was executed to evaluate the cost of executing consensus for keeping the distributed SDN control plane consistent. The second set of experiments was designed to evaluate the consensus throughput, i.e. the number of consensus decisions per time instant. The third set of experiments was executed to evaluate the performance of our proposed solution as the number of SDN controllers grows. Finally, the last set of experiments evaluate *VNF-Consensus* in the presence of VNF faults. Comparisons are presented with an alternative implementation in which the controllers themselves are responsible to run consensus and keep a consistent control plane.

A. The Cost to Keep the Distributed Control Plane Consistent

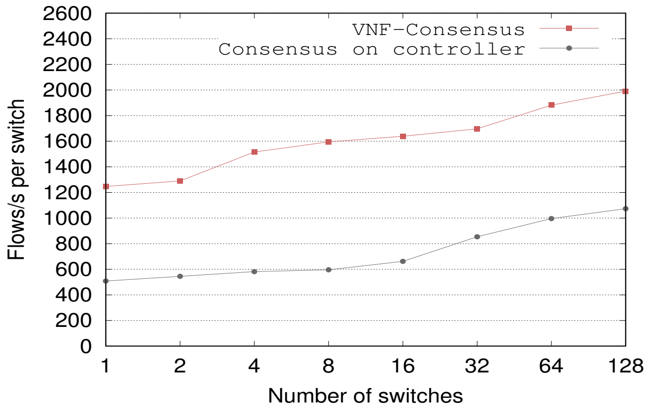
In the first set of experiments, we compare the performance *VNF-Consensus* with that of controllers which are themselves in charge of maintaining the consistency of the distributed control plane. Figure 3 shows results for three metrics: CPU usage, the number of data flows per second handled by the controller, and the time it takes for a controller to install a set of rules on switches.

Figure 3(a) consists of three curves that show: (1) the controller CPU usage as it executes its regular operations while *VNF-Consensus* is responsible for maintaining the control plane consistency (Controller baseline curve). In this case the controller just forwards rules to *VNF-Consensus*. It is important to note that the controller is not blocked while waiting for replies from *VNF-Consensus*. The second curve (2) shows controller CPU usage as it runs Paxos besides its regular activities (Consensus on controller curve). The third curve shows (3) *VNF-Consensus* CPU usage (VNF-Consensus curve). Each experiment lasted 60 seconds and was repeated three times. The network topology consists of three controllers and three *VNF-Consensus* instances.

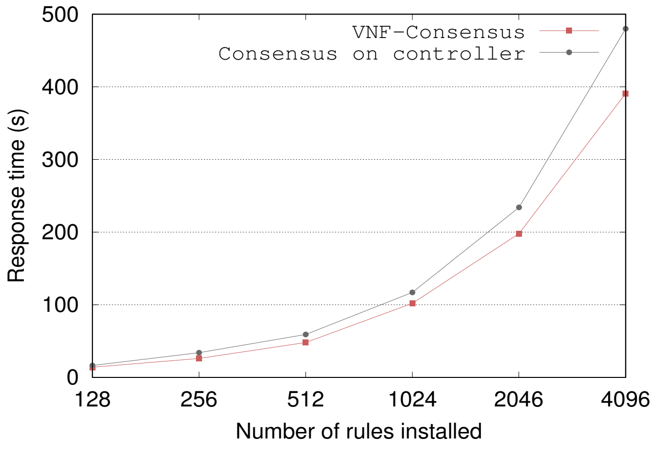
Note that when the controller executes the consensus algorithm, the CPU usage is on average 62.1% (Consensus on



(a) CPU usage.



(b) Flows/second (controller).



(c) Time for a controller to install a set of rules on a switch.

Fig. 3. Keeping the consistency of the control plane: performance evaluation.

[1]https://osrg.github.io/ryu

[2]https://bitbucket.org/sciascid/libpaxos

controller curve). On the other hand, by using *VNF-Consensus* the controller load drops to around 34.4%. The CPU usage of *VNF-Consensus* is on average 25.5%. This experiment clearly shows the advantage of uncoupling the execution of consensus from the controller. As a consequence, the controller does not have any extra overhead and is free to execute regular control plane tasks, as the tasks for keeping the consistency are executed by *VNF-Consensus*.

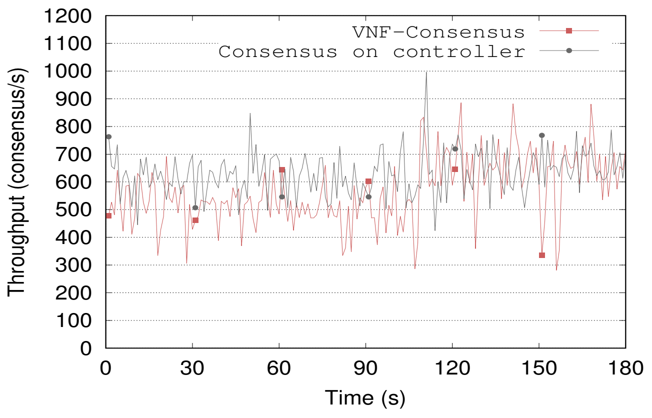
Figure 3(b) measure the overhead for maintaining strong consistency on a controller. The network consisted of three controllers and three VNFs and the number of switches increased from 1 up to 128. Each experiment was executed for 1 minute. As the number of switches increases, a large number of data flows is created. As shown in Figure 3(b), when *VNF-Consensus* is employed, the controller is able to handle a larger number of packets. The reason is that the controller has a lower load in comparison to when it is also performing in parallel the tasks for keeping the consistency of the distributed control plane. The difference is significant, close to 53% and thus remains along the x-axis.

In the third experiment shown in Figure 3(c), a script was employed to spawn 128 jobs in parallel that continuously creates random requests to the controller, simulating a scenario under a heavy load. We measured the time taken to install a set of rules, taking into consideration the whole path traversed (i.e. Host –> Switch –> Controller –> Switch –> Host). In parallel, flows are generated requiring the execution of consensus to guarantee the consistency of the control plane. In this experiment we employed three controllers and three VNFs. Each controller manages exactly one switch. In Figure 3(c), the *VNF-Consensus* curve shows a reduction of up to 18.5% of the response time, outperforming other strategies that execute consensus on the controller itself.

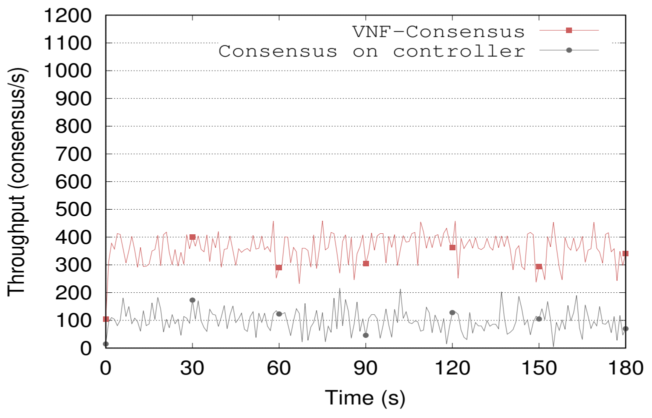
B. *Consensus Throughput*

In this set of experiments we measured the throughput of consensus. In Figure 4(a) requests are continuously submitted to the controller. However, the controller is not performing any other task in parallel, i.e., it only runs consensus for after each update request. The *Consensus on Controller* curve shows the consensus throughput in this case. The *VNF-Consensus* curve shows the throughput when the VNF is in charge of executing consensus. Upon starting the consensus execution, the controllers send requests to and receives decisions from *VNF-Consensus*. Thus, in this case, the *VNF-Consensus* throughput is lower due to this extra communication steps between controller and *VNF-Consensus*. Paxos based on controller has a throughput 11.4% higher than that of *VNF-Consensus*.

In contrast, in the experiment shown in Figure 4(b), the controller both handles data flows and performs the tasks required to keep the distributed control plane consistent. That is, as the controller is executing more tasks in parallel it presents a higher load. As a result, note that the throughput drops significantly in both cases. However the *VNF-Consensus* throughput is 3.6 times higher than that of the alternative of doing everything in the controller. Thus it can be concluded that the proposed *VNF-Consensus* provides an efficient solution for dealing with scenarios under a heavy load.



(a) Throughput without extra tasks on the controllers.



(b) Throughput including parallel tasks on the controllers.

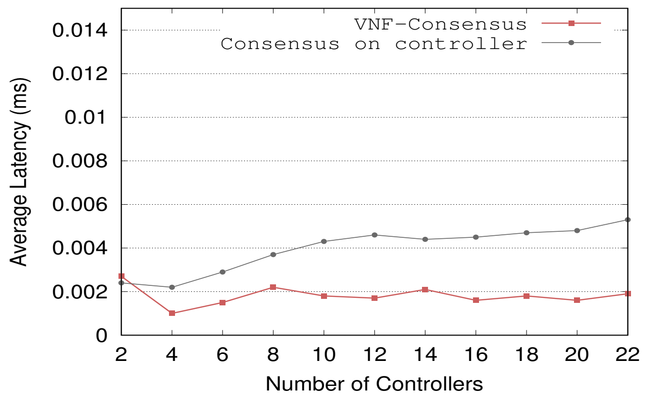
Fig. 4. Comparing the Paxos throughput.

C. *Increasing the Number of Controllers*

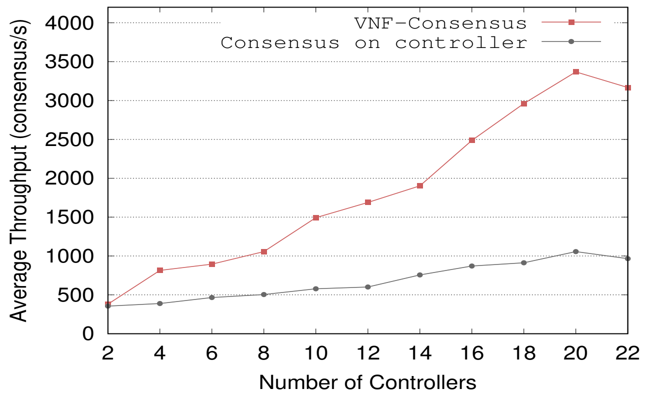
The experiment shown in Figure 5 evaluates the latency and throughput of consensus as the number of SDN controllers varies. Initially, the network topology consists of one switch per controller and three *VNF-Consensus* instances. Then, the number of controllers increases up to 22. This value was determined experimentally as the point from which the throughput drops significantly in both cases. In the *Consensus on Controller* curve the number of consensus instances is exactly the number of controllers. This is required because when consensus is executed on the controller, each controller participates as an instance of the consensus.

Figure 5(a) shows the Paxos latency when it is running on the controllers and on *VNF-Consensus*. While the execution of consensus on the controllers has an average latency of about 0.003 ms, the latency of *VNF-Consensus* is close to 0.001 ms. Thus *VNF-Consensus* reduces 67% of the Paxos latency in comparison with the execution on the controller.

Figure 5(b) shows a comparison between *VNF-Consensus* and *Consensus on Controller* in terms of the number of consensus executions completed per second while the number



(a) Average Paxos latency.



(b) Average Paxos execs/second

Fig. 5. Comparing the scalability.

of controllers increases up to 22. As can be observed *VNF-Consensus* presents a throughput about 2.6 times higher than Consensus on the Controller as the number of controllers increases. It is important to note that with more than 20 controllers, the performance is degraded due to the limitations of the environment in which we executed the experiments.

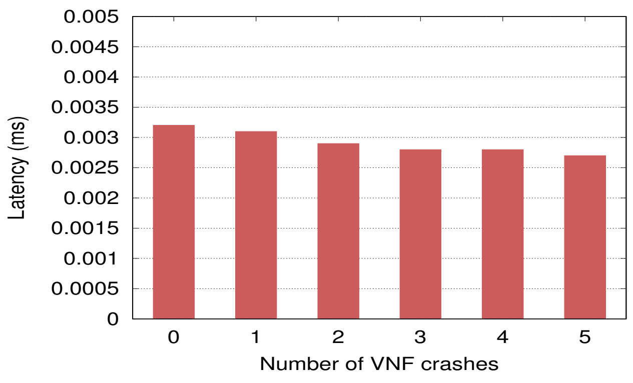
D. Robustness of VNF-Consensus

VNF-Consensus is a robust solution in the sense that the control plane remains consistent even after VNF instances crash. Since *VNF-Consensus* implements Paxos, $2f+1$ acceptors are required to tolerate f crashes. Thus, this experiment was executed to measure the impact of VNF crashes on the throughput and latency of consensus. We measured the impact considering one up to five VNF crashes ($f = 5$).

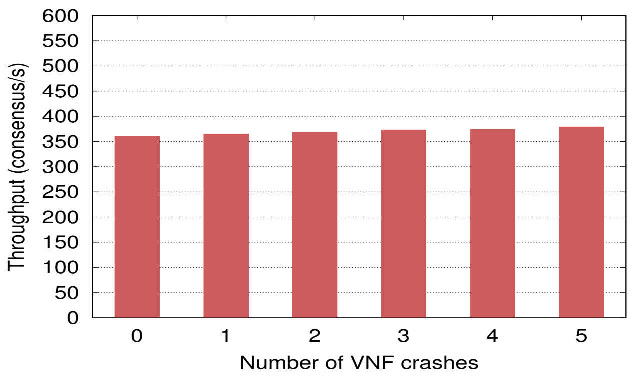
The experiment was executed on three controllers and three switches. To tolerate up to five failures, *VNF-Consensus* was configured with 11 VNF instances: one of the instances is configured as proposer and acceptor, the 10 other instances are acceptors only. VNF failures are simulated by destroying the container which runs a particular instance of the VNF.

Figure 6(a) shows the variation of the latency as VNFs crash. Note that for up to five failures, the latency decreases by 12.9%. This variation happens because the number of acceptors decreases due to the failures. Therefore, the number of consensus participants also decreases, reducing the number of messages transmitted in the network, which consequently decreases the latency.

The throughput variation is shown in Figure 6(b). The throughput increases as VNFs crash, but the impact is low: the difference for a single crashed VNF instance and five crashed VNF instances is only 3.8%. The increase is a consequence of the variation of the latency, as explained above.



(a) Latency of *VNF-Consensus* as VNF instances crash.



(b) Throughput of *VNF-Consensus* as VNF instances crash.

Fig. 6. Performance evaluation in the presence of VNF crashes.

Overall the results clearly indicate that running the tasks to keep the consistency of a distributed control plane using *VNF-Consensus* is significantly more efficient than having the controllers implement this task. In particular, the VNF approach is more scalable, as there is a limit to the amount of tasks a controller can handle. Moreover, the results shows that *VNF-Consensus* keeps the performance levels nearly unaltered in the presence of failures.

V. Conclusion

Much attention has been given recently to the design of distributed SDN control planes, which solve problems related to the dependability and performance that appear when a single controller is employed. However, a distributed control plane that consists of multiple controllers requires consistency guarantees. In this work we proposed a solution to this problem that is based on NFV technology. *VNF-Consensus* is a VNF that implements the Paxos algorithm to guarantee the strong consistency of network operations on a distributed control plane. Using the proposed approach, controllers do not execute the tasks for keeping the consistency, and this has an obvious impact on the performance and in particular the scalability of the system. Furthermore, *VNF-Consensus* does not require any change of the SDN protocol neither of the SDN switches. Experimental results are presented, comparing *VNF-Consensus* with the alternative of having controllers themselves being responsible for keeping their consistent actions. *VNF-Consensus* was shown to improve the performance in terms of resource requirements, Paxos throughput and scalability.

Future work includes the definition of a fail-over mechanism to be executed by the controllers to re-allocate switches after failures. As the number of requests grows substantially, adopting a load balancing strategy as well as optimizing the placement of *VNF-Consensus* instances will become important. Another future work is the evaluation of other consensus algorithms such as Raft [17], to check how they compare with Paxos in this setting. Investigating effective strategies for the synchronization of the data plane in SDN networks is also another possible strategy, i.e. instead of controllers, the consistency is guaranteed among switches.

Acknowledgments

This work was partially supported by CAPES Finance Code 001 and CNPq grant 311451/2016-0.

References

[1] F. Bannour, S. Souihi, and A. Mellouk, "Distributed sdn control: Survey, taxonomy, and challenges," *IEEE Communications Surveys Tutorials*, vol. 20, no. 1, pp. 333–354, 2018.

[2] J. A. Wickboldt, W. P. D. Jesus, P. H. Isolani, C. B. Both, J. Rochol, and L. Z. Granville, "Software-defined networking: management requirements and challenges," *IEEE Communications Magazine*, vol. 53, no. 1, pp. 278–285, 2015.

[3] F. Benamrane, M. Ben mamoun, and R. Benaini, "An east-west interface for distributed sdn control plane," *Comput. Electr. Eng.*, vol. 57, pp. 162–175, 2017.

[4] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (sdn)," *Computer Networks*, vol. 112, pp. 279–293, 2017.

[5] L. Schiff, S. Schmid, and P. Kuznetsov, "In-Band Synchronization for Distributed SDN Control Planes," *SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 1, 2016.

[6] E. Sakic and W. Kellerer, "Response time and availability study of raft consensus in distributed sdn control plane," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 304–318, 2018.

[7] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A Distributed Control Platform for Large-scale Production Networks," in *OSDI*, 2010.

[8] T. Hu, P. Yi, Z. Guo, J. Lan, and Y. Hu, "Dynamic slave controller assignment for enhancing control plane robustness in software-defined networks," *Future Generation Computer Systems*, vol. 95, pp. 681–693, 2019.

[9] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid, "A distributed and robust SDN control plane for transactional network updates," in *INFOCOM*, 2015.

[10] C. C. Ho, K. Wang, and Y. H. Hsu, "A fast consensus algorithm for multiple controllers in software-defined networks," in *ICACT*, 2016.

[11] H. T. Dang, D. Sciascia, M. Canini, F. Pedone, and R. Soulé, "Netpaxos: Consensus at network speed," in *SOSR/SIGCOMM*, 2015.

[12] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[13] L. Lamport, "The Part-time Parliament," *ACM Transactions on Computer Systems (TOCS)*, vol. 16, no. 2, 1998.

[14] A. S. Muqaddas, P. Giaccone, A. Bianco, and G. Maier, "Inter-controller traffic to support consistency in onos clusters," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1018–1031, 2017.

[15] M. Aslan and A. Matrawy, "A clustering-based consistency adaptation strategy for distributed sdn controllers," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018, pp. 441–448.

[16] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[17] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *USENIXATC*, 2014.

[18] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-free Coordination for Internet-scale Systems," in *USENIXATC*, 2010.

[19] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a model-driven sdn controller architecture," in *2014 IEEE 15th International Symposium on*. IEEE, 2014, pp. 1–6.

[20] L. Bondan, M. F. Franco, L. Marcuzzo, G. Venancio, R. L. Santos, R. J. Pfitscher, E. J. Scheid, B. Stiller, F. De Turck, E. P. Duarte *et al.*, "Fende: Marketplace-based distribution, execution, and life cycle management of vnfs," *IEEE Communications Magazine*, vol. 57, no. 1, pp. 13–19, 2019.

[21] M. S. Bonfim, K. L. Dias, and S. F. Fernandes, "Integrated nfv/sdn architectures: A systematic literature review," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, p. 114, 2019.

[22] T. Zhang, A. Bianco, and P. Giaccone, "The role of inter-controller traffic in sdn controllers placement," in *IEEE NFV-SDN*, 2016, pp. 87–92.