Received: 16 January 2020 | Revised: 27 March 2020 | Accepted: 6 April 2020

DOI: 10.1002/nem.2124

SPECIAL ISSUE PAPER

WILEY

VNF-Consensus: A virtual network function for maintaining a consistent distributed software-defined network control plane

Giovanni Venâncio[1] | **Rogério C. Turchetti**[2] | **Edson T. Camargo**[3] | **Elias P. Duarte Jr**[1]

[1]Department of Informatics, Federal University of Paraná, P.O. Box 19018, Curitiba, PR, 81531-980, Brazil

[2]CTISM, Federal University of Santa Maria, Av. Roraima, 1000, Santa Maria, RS, 97105-900, Brazil

[3]Coordination of Internet System Technology, Federal Technological University of Paraná, Rua Cristo Rei, 19, Toledo, PR, 85902-490, Brazil

Correspondence
Elias P. Duarte Jr., Department of Informatics, Federal University of Paraná, P.O. Box 19018, Curitiba, 81531-980, Brazil.
Email: elias@inf.ufpr.br

Funding information
Conselho Nacional de Desenvolvimento Científico e Tecnológico, Grant/Award Number: 311451/2016-0; Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, Grant/Award Number: Código de Financiamento 001

Summary
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.

1 | 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 the controller fails the whole network stops running) but also on performance and scalability[3] (i.e., the single controller has to process all requests from all switches). The solution is to distribute the control plane, employing multiple controllers that share responsibilities.[4] Several different distributed SDN control plane strategies have been proposed.[5,6]

Although the advantages of employing multiple controllers should be clear, there is also a cost to pay: in order to employ a distributed control plane it is necessary to employ techniques to guarantee the consistency among the multiple controllers.[7] Therefore, actions performed on the control plane by the multiple SDN controllers need to be synchronized, and this is not a trivial endeavor.[5] If the synchronization is not done properly, consistency violations can occur, causing several problems. 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.[8] 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 in 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.[7-9] In contrast, there are other solutions that avoid increasing controller workload by having the switches synchronize their actions.[5,10] In general, these solutions have disadvantages as they require modifications to the SDN protocol.

In this work, we propose an alternative solution for keeping the consistency of a distributed SDN control plane that takes advantage of network function virtualization (NFV)[11] 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[12] 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 across the network. 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 propose actions to be synchronized and receive decisions. Note that all decisions handled by *VNF-Consensus* are systematically taken 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 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. It is important to note that our strategy can be implemented without any changes to the SDN protocol and the switches. We show that our solution is robust, ensuring correct network operation even in the presence of VNF failures and also controller failures.

VNF-Consensus was implemented and experimental results are reported, with a highlight on the performance improvements in comparison with having the controllers themselves responsible for keeping the consistency of network operations. In particular, we show that it is possible to synchronize the control plane without increasing the controller load. Finally, results show the impact on network performance when both VNFs and controllers fail.

The rest of this work is organized as follows. Section 2 gives an overview of NFV technology, as well as the related work. Section 3 describes the proposed architecture. The experimental results are described in Section 4. Finally, Section 5 concludes the work.

2 | BACKGROUND AND RELATED WORK

In this section, we give an overview of NFV technology and also present work related to our strategy to enable the synchronization of multiple controllers in SDN networks.

2.1 | Network function virtualization: an overview

Networks employ multiple types of middleboxes which are traditionally implemented in hardware, such as firewalls, intrusion detection and prevention systems, traffic shapers, among several others.These middleboxes represent an important fraction of the network OPerational EXpenditures (OPEX) and CAPital EXpenditures (CAPEX), being usually expensive to deploy and manage and complex to troubleshoot.[13]

Network function virtualization (NFV) is a novel technology that addresses these challenges by leveraging virtualization technologies to offer a new way to design, deploy, and manage network services.[11] In particular, NFV employs virtualization techniques to define a VNF that runs on off-the-shelf hardware and is used to deploy network services. NFV technology has brought new alternatives to the network service market, even allowing developers to publish and distribute VNFs through marketplaces.[14]

Figure 1 shows a comparison between traditional networks based on middleboxes and those that rely on NFV. In traditional networks, services run on dedicated hardware; using NFV technology, those services can be implemented on a virtualization layer and are executed on commodity hardware. With NFV, it is possible to create, deploy, and manage virtual middleboxes in a fraction of the time required for the hardware counterparts. These functions are started on demand and removed when they are no longer needed, making optimal usage of system resources. Moreover, they are easier to manage and operate.[15]

We note that NFV technology is usually implemented taking advantage of another related technology: software-defined networking.[16] SDN and NFV are complementary technologies that together provide a software-based approach to networking. While SDN technology decouples the control plane from the data plane, NFV technology has a focus on the services that are provided within the network. Although they can be used independently of each other, SDN technology allows traffic to be redirected to VNFs through a policy-based decision process. Therefore, SDN technology complements NFV mainly in the sense that it facilitates VNF management and orchestration.

2.2 | Related work

Next 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 VNF.

A synchronization framework for control planes based on atomic transactions is proposed by Schiff et al.[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 execute multiple data plane configuration commands as an atomic transaction. The framework modifies the OpenFlow protocol[17] in order to avoid controllers to install inconsistent rules on the switches. The authors also describe an implementation and evaluate the efficiency of the proposed solution empirically.

Another proposal explores the implementation of the Paxos consensus algorithm on SDN switches.[10] 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.

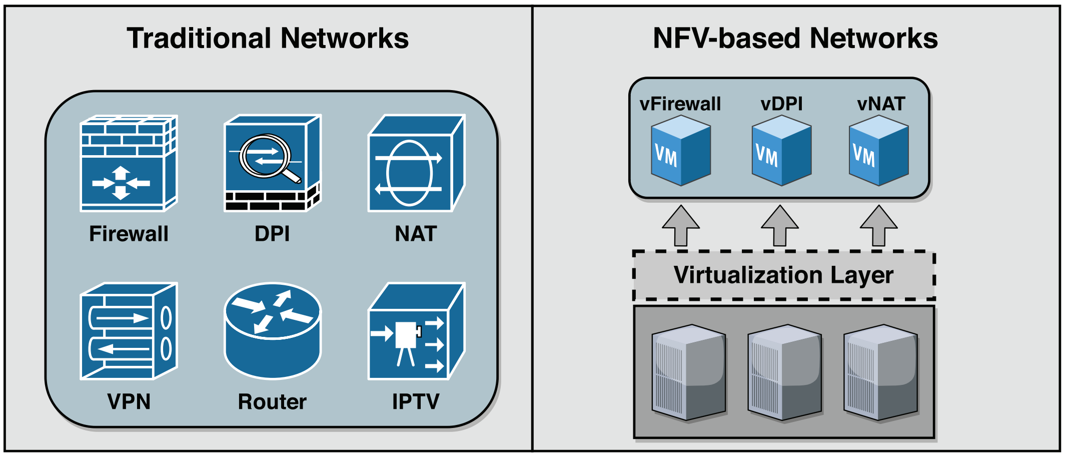


FIGURE 1 Comparison between a traditional and a network function virtualization (NFV)-based network

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 by Ho et al.[9] 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[18] 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 network information base (NIB). 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[19] to synchronize the multiple instances.

Another proposal proposed by Canini et al.[8] 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 guarantee the total order of policy updates.

OpenDayLight (ODL)[20] 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.[21] The controllers themselves implement the consensus algorithm.

An adaptive strategy that employs on line clustering techniques to allow tunable consistency levels is presented by Aslan et al.[22] 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.

Ravana[23] is a fault-tolerant controller platform for SDN. Ravana ensures the all-or-nothing property: a control message is either processed by all controllers and switches, or none of them. In addition, the platform ensures that transactions are totally ordered across replicas, while also ensuring the consistency of switch states. Ravana proposes a two-phase replication protocol that extends replicated state machines for dealing with consistency issues. In order to do this, Ravana uses ZooKeeper for event logging and leader election.

The work proposed by Mahajan et al.[24] deals with the problem of validating controller actions considering that one or more SDN controllers in a high-availability (HA) cluster may be faulty. The authors present a system called JURY that validates controller activities in a clustered SDN deployment. JURY uses a consensus algorithm among cluster nodes to validate controller activities. JURY has three main components: a replicator, a module that is executed on each replica, and an out-of-band validator. Using state machine replication, the validator decides within a specified time interval whether a controller action is valid or not. It is important to note that JURY also implements the replicator out-of-band, similar to our work. However, it requires changes to the SDN controllers while also requiring an SDN cluster to be formed. Furthermore, Katta et al.[23] argue that existing fault-tolerance techniques, such as state machine replication, are not what is needed to ensure correct network behavior under controller failures.

Botelho et al.[25] argue that it is essential for controllers to have a single and consistent view in a distributed SDN network. They also argue that maintaining an eventually consistent view is not enough, possibly leading to network anomalies. Motivated by the need of strong consistency in the control plane, the work proposes a novel and modular SDN control plane architecture. This architecture is supported by a fault-tolerant data store, which is responsible for the coordination of actions, and consequently guarantees a strongly consistent view of the network by using the BFT-SmaRt replication library. However, the authors themselves argue that "a fundamental concern of these systems is their limited scalability and performance overhead." In order to improve the data store limitations, the work proposes a set of optimizations techniques, improving both latency and throughput. Nevertheless, a major drawback is that the centralized data store represents a network bottleneck.

Table 1 compares the multiple strategies described in this section. A major difference between our proposal and the other strategies is *where* and *how* 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.

TABLE 1 Comparison of the synchronization strategies

Related Work	Algorithm for maintaining the consistency	Synchronization executed on
Schiff et al.[5]	Atomic transactions using *compare-and-set* (CAS)	SDN switch
Dang et al.[10]	NetPaxos consensus algorithm	SDN switch
Ho et al.[9]	Fast Paxos-based consensus algorithm (FPC)	Controller
Koponen et al.[7]	Zookeeper tool	Controller
Canini et al.[8]	State machine replication	Controller
ODL Platform	Raft consensus algorithm	Controller
Muqaddas et al.[21]	ONOS clustering	Controller
Aslan and Matrawy[22]	Apache Cassandra	Controller
Katta et al.[23]	Extended state machine replication	Switch and controller
Mahajan et al.[24]	State machine replication	External entities
Botelho et al.[25]	BFT-SMaRt	External entities

3 | A VNF TO KEEP THE CONTROL PLANE CONSISTENT

In this section, initially we define the problem of building a consistent SDN control plane. Then, the architecture of our VNF-based solution is described. Finally, the strategies adopted to tolerate failures of both SDN controllers and VNFs are described.

3.1 | 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 on changes to 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.[17] Two types of communication strategies are described for SDN components[26]: 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. Our strategy is a solution 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.[26] 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 multiple different results for a short period of time. In the proposed architecture, we use the Paxos consensus algorithm which provides strong consistency.

3.2 | VNF-Consensus

VNF-Consensus implements Paxos, which is briefly described next. Paxos is a distributed consensus algorithm designed for state machine replication.[12] 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 nonfaulty 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 nonfaulty. 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 in 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 to install each new 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 in an independent VNF. After a decision is reached, the controller executes the action received from *VNF-Consensus.*

Figure 2 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 2 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 2. 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 new 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

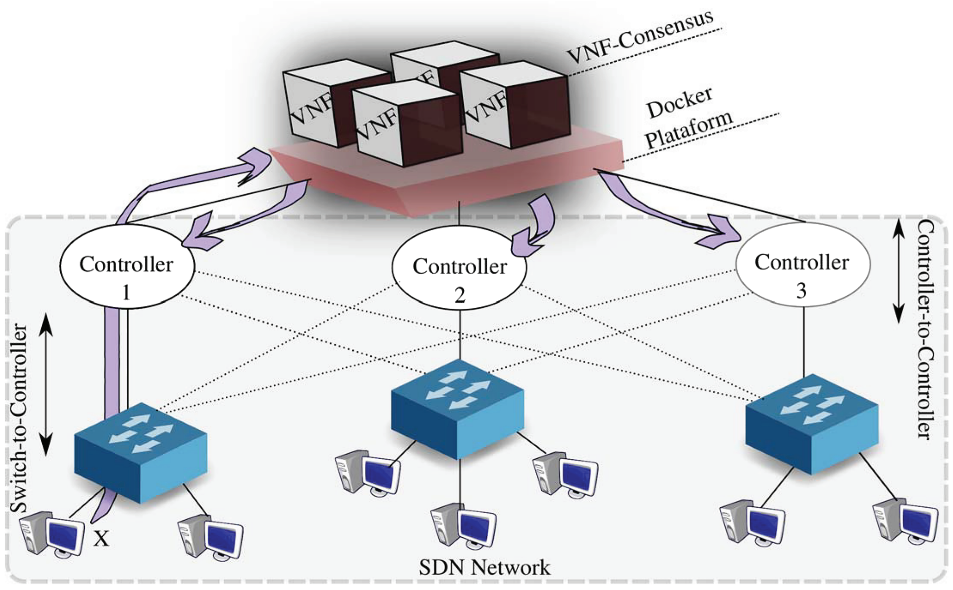


FIGURE 2 A software-defined network (SDN) with *VNF-Consensus*

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 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, for example, *VNF-Consensus* (Figure 3). The *Filter* consists of a *Classifier* and a *Forwarder*. The *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 3 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).

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.

Note that in the second phase, the coordinator may select a rule that is different from the one it had sent in the first phase. If one or more acceptors returned multiple rules in the first phase, the coordinator chooses the rule with the highest round number. Moreover, if no acceptor returned any rules, the coordinator can select which rule to proceed, in our case the rule received from the *Forwarder* module. Then, the coordinator sends an *accept* request containing the selected rule and the round number to the acceptors.

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.

It is also important to note that Paxos solves timing and management issues that may appear in real distributed environments, such as cloud-based platforms. Paxos is able to deal with asynchrony while also always guaranteeing safety: no two controllers will install different rules, even if timing issues become significant. Furthermore, Paxos guarantees progress under weak synchrony assumptions – this means that if the system becomes really slow so that half of

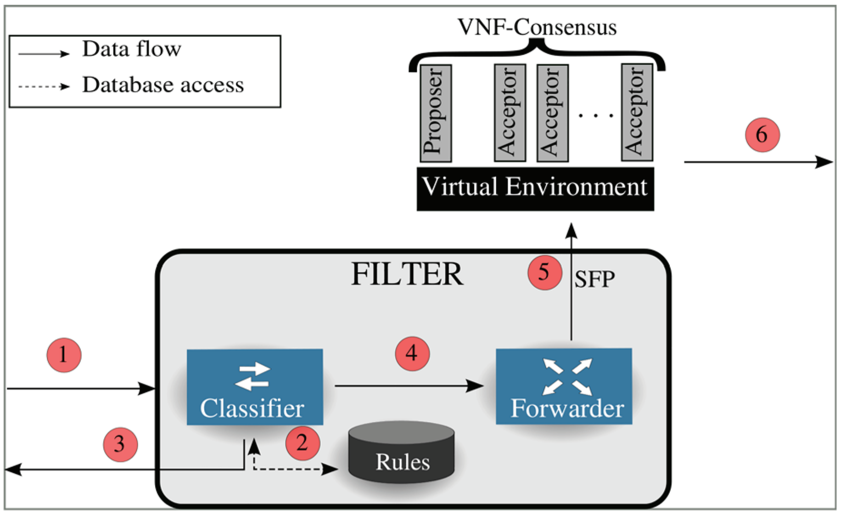


FIGURE 3 Architecture: synchronization of the control plane

the *VNF-Consensus* instances do not reply in time, no decision is reached. This can be dealt with by re-running the algorithm until the network reaches a more stable condition.

3.3 | Tolerating control plane faults

Despite the many advantages offered by SDN, the occurrence of failures of its components, such as controllers and switches, can compromise network dependability, affecting both the availability and reliability of the services offered.[27] In this work, both the SDN controller and *VNF-Consensus* are responsible for handling network traffic. Next, we define fault tolerance and resiliency strategies to ensure correct network execution, even in the presence of failures. The strategies are designed to tolerate faults of both VNF-Consensus instances and SDN controller.

3.3.1 | Tolerating failures of the SDN controller

The main purpose of distributing the control plane is to avoid a single point of failure, preventing a controller failure from compromising all communication on the network. There are several strategies in the literature for tolerating failures in the control plane,[28-30] in the data plane,[5,23,31] and even in applications themselves.[27]

A common strategy to design fault-tolerant SDN controllers[32,33] is to have a primary controller responsible for receiving and managing all data plane requests, while a standby controller is used only to take over in case the primary fails.

Although this is a simple and reliable arrangement, there are drawbacks. The first is that using a single primary controller can lead to performance issues, as the controller can become a bottleneck. The second problem is related to some scenarios that may lead to network inconsistency. Consider the case where the primary controller fails immediately after installing a new rule in the switch. Also consider the standby has not yet been updated. In this case, the control plane will be inconsistent with the data plane, since they have processed different OpenFlow rules.

In order to avoid the aforementioned problems, this work employs a slightly different strategy. As shown in Figure 4, each controller (primary) has a standby. As we are dealing with a distributed control plane, there are *N* primary controllers, each of them responsible for a set of switches. Therefore, the bottleneck issue does not exist here. Next, we describe how to eliminate inconsistencies between the control/data planes.

As described in Section 3.2, *VNF-Consensus* ensures the consistency of the control plane by implementing the Paxos algorithm. In this case, as each controller is a learner, all learned values (i.e., OpenFlow rules) are received by the controller. In order to avoid inconsistent scenarios between the control plane and the data plane caused by failures, the standby controller is also a learner. Therefore, all values learned by the primary controller are also learned by the standby, thus preserving the consistency between them. Upon failure, the standby controller can immediately take over the execution of the primary controller, since they have the exact same set of OpenFlow rules.

Once the resiliency strategies have been defined, procedures must be executed to recover the controller, ensuring the correct operation of the network. In particular, two steps are important during the recovery: switch the standby controller to become the new primary, and create a new replica to tolerate future failures. Both steps are described below.

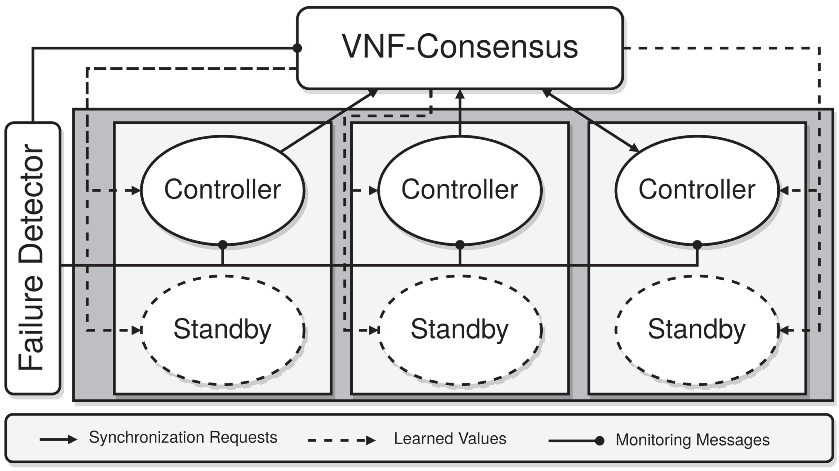


FIGURE 4 Software-define network (SDN) controller—primary and standby

As seen in Figure 4, a failure detector (FD) is employed to monitor all primary controllers. The FD used in this work is based on the polling strategy—queries are periodically sent to each SDN controller. If the controller does not reply to the message within a time interval, the controller is suspected to have failed. If the FD suspects that a controller has failed, it sends a message to the respective standby, which will assume the role of primary. Upon the receipt of this message, the standby controller creates a new connection with its switches. In this stage, the standby controller is already considered the primary and can immediately take over the execution of the failed controller. Next, the recovery process proceeds to the second step.

The second step to recover a controller is to create a new replica to tolerate possible future failures. First, a new controller is created as the standby associated with the new primary. After the creation, it is necessary to synchronize the new standby controller copying all rules of the the primary. Once the synchronization is complete, the standby is ready and will continue updating the set of rules as a learner.

3.3.2 | Tolerating VNF-Consensus failures

In addition to tolerating failures of the SDN controllers, *VNF-Consensus* instances—responsible for synchronizing the control plane—are also susceptible to failures, possibly compromising the network communication.

Failures in a virtualized environment can occur at many levels, such as in the process, in the VNF, and even in the infrastructure itself. A process-level failure occurs when the process that performs the VNF function fails. In the context of this work, a process-level failure occurs when a process that performs one of the Paxos roles fails (i.e., acceptor, learner, and proposer). VNF-level failures occur due to errors directly related to the virtualization techniques (e.g., virtual machine and container), such as lack of virtual resources or even connection failures. Infrastructure-level failures correspond to the failure of the entire server that hosts the VNFs, caused, for instance, by a power outage. This work considers only VNF-level failures.

As described in Section 3.2, *VNF-Consensus* implements Paxos, a fault-tolerant algorithm. Paxos requires $2f+1$ instances of *VNF-Consensus* to allow f of these instances to fail, and still reach consensus. In order to always maintain a minimum number of *VNF-Consensus* instances, whenever a failure is detected by FD, the recovery process is immediately started.

In order to detect failures of the VNF instances, the FD shown in Figure 4, in addition to monitoring the controllers, also monitors the *VNF-Consensus* instances. Thus, when the FD detects a VNF failure, it makes a request to replace the VNF virtual resources, for example, by replacing the virtual machine or container that hosts the VNF. After the VNF recovery, it is necessary to perform a reconfiguration, which consists mainly of obtaining all the values decided by Paxos before the failure. Therefore, all OpenFlow rules decided by consensus until the moment of its failure are restored. After a new VNF is instantiated, it updates the local state by receiving the local list of decided values of another VNF.

4 | 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 controllers[34] were employed and connected with virtual switches (OpenVSwitch[35]). The Paxos library employed was libPaxos.[36] *VNF-Consensus* uses a REpresentational State Transfer (REST) interface to communicate with the controllers.

Unless otherwise specified, the network topology consists of three controllers and three *VNF-Consensus* instances. Each controller is associated with a switch, and each switch connects a host. In all experiments, the hosts are employed to generate TCP traffic through the iPerf3[*]tool.

Each *VNF-Consensus* instance is hosted on a container using the Docker platform.[37] Note that each rule causes an individual instance of Paxos to be executed. As a 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

*https://iperf.fr/

implemented by having the Paxos algorithm run as a process directly accessed by the controller, both in the same container.

Results are reported for five 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, that is, 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 the strategy in the presence of failures. Comparisons are presented with an alternative implementation in which the controllers themselves are responsible to run consensus and keep a consistent control plane.

4.1 | 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. Results are shown 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 5 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 s and was repeated three times.

Note that when the controller executes the consensus algorithm, the CPU usage is on average 62.1% (Consensus on 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 6 measures the overhead for maintaining strong consistency in the controller in terms of the number of control messages per second that the controller handles. The number of switches increases from 1 up to 128 and each experiment was executed for 1 minute. As the number of switches increases, the number of hosts generating traffic is also increases, since each switch connects with a host. Therefore, a large number of data flows is created.

As shown in Figure 6, when *VNF-Consensus* is employed, the controller is able to handle a larger number of packets. The difference is significant, close to 53% and thus remains along the *x*-axis. The reason for this is that the number of flows per second that the controller handles is limited by its capacity. Thus, 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. If the controller itself is running the consensus algorithm, part of its resources are being used to run this task. When *VNF-Consensus* is employed, more controller resources are available to handle more control messages per second.

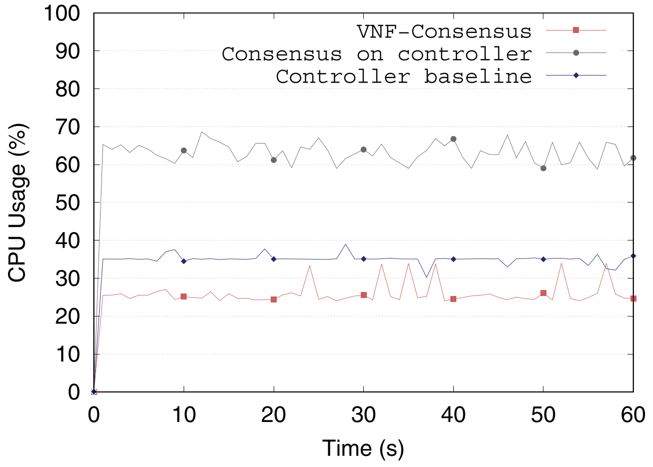


FIGURE 5 CPU usage

FIGURE 6 Flows/second handled by the controller

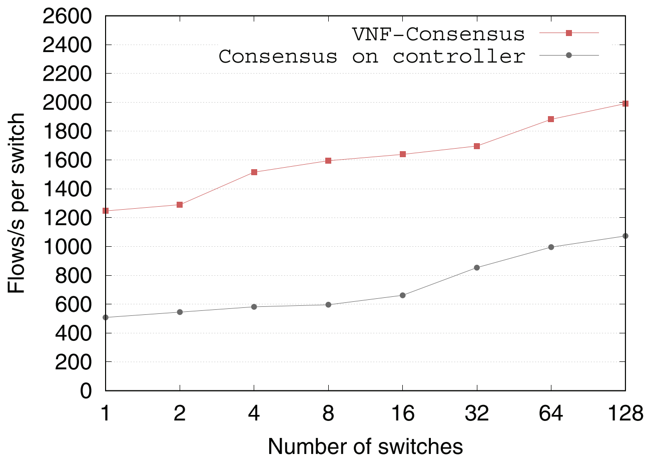
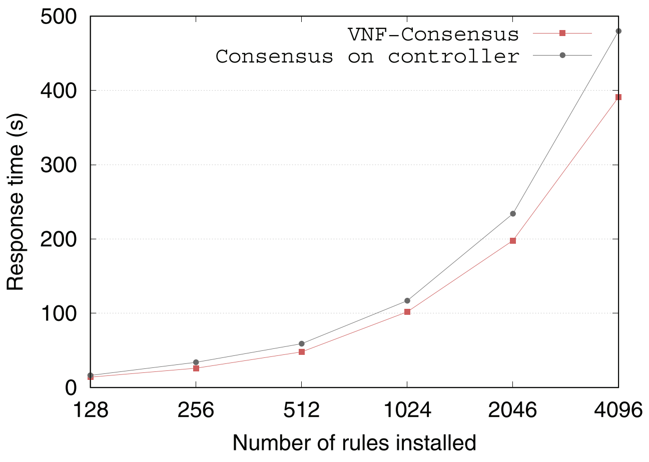


FIGURE 7 Time for a controller to install a set of rules on a switch

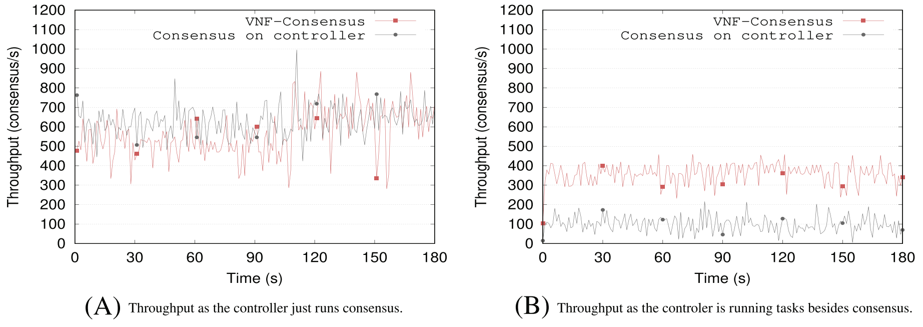


In the third experiment shown in Figure 7, a script was employed to spawn 128 jobs in parallel that continuously creates random requests to the controller, simulating a heavy load scenario. We measured the time taken to install a set of rules, taking into consideration the whole path traversed (i.e., Host –> Switch –> Controller –> Switch –> Host). In parallel, flows are generated requiring the execution of consensus to guarantee the consistency of the control plane. Each controller manages exactly one switch. In Figure 7, 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.

4.2 | Consensus throughput

In this set of experiments, we measured the throughput of consensus. In Figure 8A, requests are continuously submitted to the controller. However, the controller is not performing any other task in parallel; that is, it only runs consensus 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.*

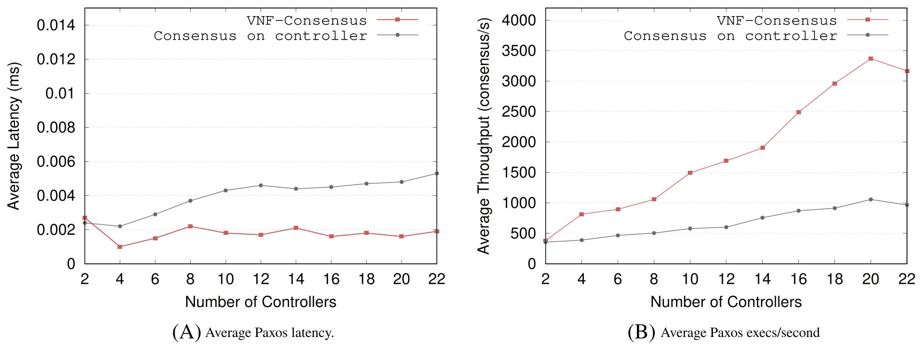
In contrast, in the experiment shown in Figure 8B, the controller both handles data flows and performs the tasks required to keep the distributed control plane consistent. That is, as the controller is executing more tasks in parallel it presents a higher load. As a result, note that the throughput drops significantly in both cases. However, the *VNF-Consensus*throughput is 3.6 times higher than that of Consensus on Controller. Thus, it can be concluded that *VNF-Consensus* provides an efficient solution for dealing with scenarios under heavier loads.



(A) Throughput as the controller just runs consensus.

(B) Throughput as the controler is running tasks besides consensus.

FIGURE 8 Comparing the Paxos throughput



(A) Average Paxos latency.

(B) Average Paxos execs/second

FIGURE 9 Comparing the scalability

4.3 | Increasing the number of controllers

This experiment 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 in every instance of the consensus algorithm.

Figure 9A 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 9B shows a comparison between *VNF-Consensus* and *Consensus on Controller* in terms of the number of consensus executions completed per second while the number 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 degrades quickly due to the limitations of the environment in which we executed the experiments.

4.4 | 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 ($f=5$).

In order 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 10A 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 10B. 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.

4.5 | Controller failures

As seen in Section 3.3.1, in this work, each primary controller has a standby controller, responsible for taking over in case the primary fails. In order to reduce the impact on the network after a controller failure, downtime must be minimized through fast detection and recovery. The experiment shown in Figure 11 shows the total recovery time, from the detection to the complete synchronization of the new standby controller, as the number of rules installed on the switch varies.

The recovery process consists of four main steps. The first step is to reconfigure the standby to become the new primary controller. This step consumes about 9.8% (500 ms) of the total time. The second step is the creation of a new standby controller, which consumes most of the recovery time: 55.5% (2.8 s). This step takes longer because it includes the creation of a new container. Afterwards, it is necessary to initialize the new controller (e.g., initialization of processes and connections), consuming about 19.8% (1 s) of the total time. Finally, the new standby controller is synchronized with the rules that was previously learned, which consumes about 14.9% (0.81 s) of the total recovery time. Note that the recovery time is scalable with respect to the number of rules: the difference between 10 and 10^6 rules is only 3.65 s.

Finally, the last experiment aims to verify the performance in terms of the number of requests handled per second by the controller during failures. For this experiment, shown in Figure 12, two faults were injected at times 20 and 40, causing the controller to fail. In this experiment, the controller installed a set of 1000 OpenFlow rules.

As expected, it is possible to notice the downtime after a failure occurs. As soon as the FD detects the controller failure and triggers the recovery and completes the synchronization of the new standby controller, the situation is back to normal. The total downtime is of 2.7 s.

FIGURE 10 Performance evaluation in the presence of virtual network function (VNF) crashes

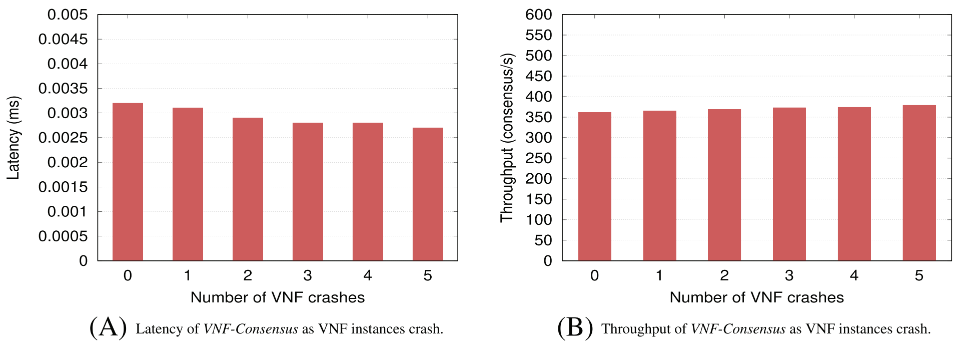
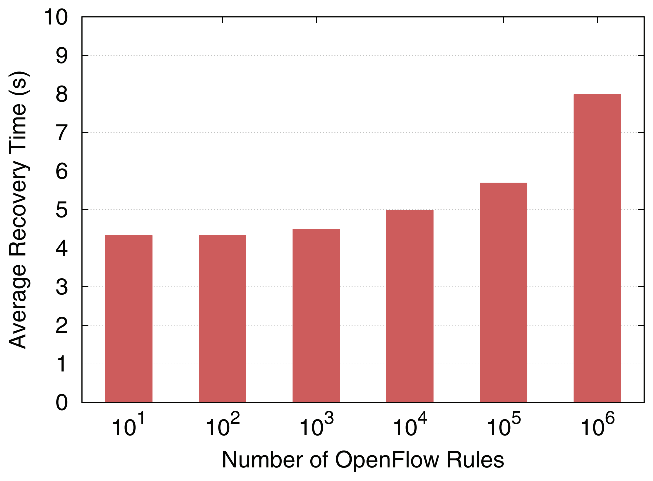


FIGURE 11 Time for a controller to recover



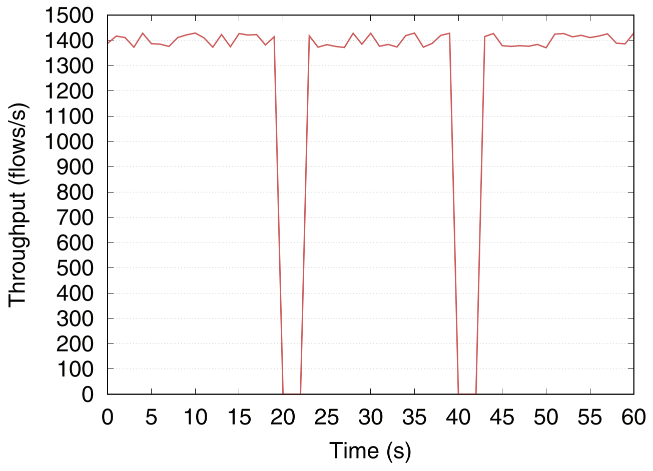


FIGURE 12 Flows/second handled by the controller in the presence of failures

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 on the number of tasks a controller can handle. Moreover, the results show that *VNF-Consensus* keeps the performance levels nearly unaltered in the presence of failures.

5 | 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 to the SDN protocol and 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 failover mechanism to be executed by the controllers to reallocate 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 becomes important. Another future work is the evaluation of other consensus algorithms such as Raft,[18] to check how they compare with Paxos in this setting. The investigation of effective strategies for the synchronization of the data plane in SDN networks is also another possible strategy, that is, the consistency in this case is guaranteed among switches instead of the controllers.

ACKNOWLEDGEMENT

This work was partially supported by the Brazilian Research Council (CNPq - Conselho Nacional de Desenvolvimento Cientéfico e Tecnológico) grant 311451/2016-0, and CAPES (Coordenaçádo de Aperfeiçoamento de Pessoal de Nível Superior) Código de Financiamento 001.

ORCID

Giovanni Venâncio https://orcid.org/0000-0001-7620-3793
Rogério C. Turchetti https://orcid.org/0000-0002-5242-5057
Elias P. Duarte Jr https://orcid.org/0000-0002-8916-3302

REFERENCES

1. Bannour F, Souihi S, Mellouk A. Distributed SDN control: survey, taxonomy, and challenges. *IEEE Commun Surv Tut*. 2018;20(1): 333-354.

2. Wickboldt JA, Jesus WPD, Isolani PH, Both CB, Rochol J, Granville LZ. Software-defined networking: management requirements and challenges. *IEEE Commun Mag*. 2015;53(1):278-285.
3. Benamrane F, Ben mamoun M, Benaini R. An east-west interface for distributed SDN control plane. *Comput Electr Eng*. 2017;57:162-175.
4. Karakus M, Durresi A. A survey: control plane scalability issues and approaches in software-defined networking (SDN). *Comput Netw*. 2017;112:279-293.
5. Schiff L, Schmid S, Kuznetsov P. In-band synchronization for distributed SDN control planes. *SIGCOMM Comput Commun Rev*. 2016;46(1):37-43.
6. Sakic E, Kellerer W. Response time and availability study of raft consensus in distributed SDN control plane. *IEEE Trans Netw Serv Manag*. 2018;15(1):304-318.
7. Koponen T, Casado M, Gude N, et al. Onix: A distributed control platform for large-scale production networks. *9th Usenix Symposium on Operating System Design and Implementation (OSDI)*. 2010;10:1-6.
8. Canini M, Kuznetsov P, Levin D, Schmid S. A distributed and robust SDN control plane for transactional network updates. In: 2015 IEEE Conference on Computer Communications (INFOCOM) IEEE; 2015:190-198.
9. Ho CC, Wang K, Hsu YH. A fast consensus algorithm for multiple controllers in software-defined networks. In: 2016 18th International Conference on Advanced Communication Technology (ICACT) IEEE; 2016:112-116.
10. Dang HT, Sciascia D, Canini M, Pedone F, Soulé R. Netpaxos: consensus at network speed. In: Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research; 2015:1-7.
11. Mijumbi R, Serrat J, Gorricho J-L, Bouten N, De Turck F, Boutaba R. Network function virtualization: state-of-the-art and research challenges. *IEEE Commun Surv Tut*. 2016;18(1):236-262.
12. Lamport L. The part-time parliament. *ACM Trans Comput Syst (TOCS)*. 1998;16(2):133-169.
13. Cotroneo D, De Simone L, Iannillo A, Lanzaro A, Natella R, Fan J, Ping W. Network function virtualization: challenges and directions for reliability assurance. In: Software Reliability Engineering Workshops (issrew), 2014 IEEE International Symposium on IEEE; 2014:37-42.
14. Bondan L, Franco MF, Marcuzzo L, et al. Fende: marketplace-based distribution, execution, and life cycle management of vnfs. *IEEE Commun Mag*. 2019;57(1):13-19.
15. Turchetti RC, Duarte EP Jr. NFV-FD: implementation of a failure detector using network virtualization technology. *Int J Netw Manag*. 2017;27(6):e1988.
16. Bonfim MS, Dias KL, Fernandes SF. Integrated NFV/SDN architectures: a systematic literature review. *ACM Comput Surv (CSUR)*. 2019;51(6):114.
17. McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput Commun Rev*. 2008;38(2):69-74.
18. Ongaro D, Ousterhout J. In search of an understandable consensus algorithm. In: 2014 USENIX Annual Technical Conference (USENIXATC 14); 2014:305-319.
19. Hunt P, Konar M, Junqueira FP, Reed B. ZooKeeper: Wait-free coordination for internet-scale systems. In: USENIX annual technical conference; USENIX (the Web site of this important organization is https://usenix.org). 2010. 1–14.
20. Medved J, Varga R, Tkacik A, Gray K. Opendaylight: Towards a model-driven SDN controller architecture. In: 2014 IEEE 15th International Symposium on IEEE; 2014:1-6.
21. Muqaddas AS, Giaccone P, Bianco A, Maier G. Inter-controller traffic to support consistency in ONOS clusters. *IEEE Trans Netw Serv Manag*. 2017;14(4):1018-1031.
22. Aslan M, Matrawy A. A clustering-based consistency adaptation strategy for distributed SDN controllers. In: 2018 4th IEEE Conference on Network Softwarization and Workshops (netsoft); 2018:441-448.
23. Katta N, Zhang H, Freedman M, Rexford J. Ravana: Controller fault-tolerance in software-defined networking. In: Proceedings of the 1st ACM Sigcomm Symposium on Software Defined Networking Research ACM; 2015:4.
24. Mahajan K, Poddar R, Dhawan M, Mann V. Jury: validating controller actions in software-defined networks. In: 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (dsn) IEEE; 2016:109-120.
25. Botelho FA, Ribeiro TA, Ferreira P, Ramos FMV, Bessani AN. Design and implementation of a consistent data store for a distributed SDN control plane. In: 12th European Dependable Computing Conference, EDCC IEEE; 2016:169-180.
26. Zhang T, Bianco A, Giaccone P. The role of inter-controller traffic in SDN controllers placement. In: IEEE nfv-sdn; 2016:87-92.
27. Chandrasekaran B, Benson T. Tolerating SDN application failures with LegoSDN. In: Proceedings of the 13th ACM Workshop on Hot Topics in Networks ACM; 2014:22.
28. Gonzalez AJ, Nencioni G, Helvik BE, Kamisinski A. A fault-tolerant and consistent SDN controller. In: 2016 IEEE Global Communications Conference (globecom) IEEE; 2016:1-6.
29. ElDefrawy K, Kaczmarek T. Byzantine fault tolerant software-defined networking (SDN) controllers. In: 2016 IEEE 40th Annual Computer Software and Applications Conference (compsac), Vol. 2 IEEE; 2016:208-213.
30. Botelho F, Ribeiro TA, Ferreira P, Ramos FM, Bessani A. Design and implementation of a consistent data store for a distributed SDN control plane. In: 2016 12th European Dependable Computing Conference (EDCC) IEEE; 2016:169-180.
31. Kim H, Schlansker M, Santos JR, Tourrilhes J, Turner Y, Feamster N. Coronet: fault tolerance for software defined networks. In: 2012 20th IEEE International Conference on Network Protocols (ICNP) IEEE; 2012:1-2.
32. Budhiraja N, Marzullo K, Schneider FB, Toueg S. The primary-backup approach. *Distrib Syst*. 1993;2:199-216.

33. Hu T, Yi P, Guo Z, Lan J, Hu Y. Dynamic slave controller assignment for enhancing control plane robustness in software-defined networks. *Fut Gen Comput Syst*. 2019;95:681-693.
34. Ryu S. Ryu SDN framework. Online http://osrggithubio/ryu; 2015.
35. Pfaff B, Pettit J, Koponen T, et al. The design and implementation of open vswitch. In: 12th USENIX symposium on networked systems design and implementation (NSDI 15); 2015:117-130.
36. Primi M, Sciascia D. Libpaxos; 2016.
37. Merkel D. Docker: lightweight linux containers for consistent development and deployment. *Linux J*. 2014;2014(239):2.

AUTHOR BIOGRAPHIES

Giovanni Venâncio is a PhD student in Computer Science at the Department of Informatics of the Federal University of Paraná (UFPR) under the supervision of Prof. Dr. Elias Procópio Duarte Júnior. Giovanni holds an MSc (2017) in Computer Science and a Computer Science degree (2016) at the same institution. His research interests include network function virtualization, high availability, and fault-tolerant distributed systems

Rogério C. Turchetti is a Adjunct Professor at the Federal University of Santa Maria, Santa Maria, Brazil. He received a PhD degree in Computer Science from Federal University of Parana, Brazil, 2017, the MSc degree in Production Engineering with emphasis on Information Systems from the Federal University of Santa Maria, Santa Maria, Brazil, in 2006. His research interests include computer network and distributed systems, their dependability and algorithms. His recent research is focused on the dependability in network function virtualization and software-defined network.

Edson T. Camargo is a Associate Professor at the Federal Technological University of Paraná (UTFPR), Toledo, Brazil. He received a PhD degree in Computer Science from the Federal University of Parana (UFPR), Curitiba, Brazil. During his PhD, he spent 1 year as a PhD student at the Universitá della Svizzera italiana (USI). He received a master's degree in Electrical Engineering with emphasis on Automation Systems from the Federal University of Santa Catarina (UFSC), Brazil, in 2006. His research interests include computer network and distributed systems, their dependability and algorithms. His recent research is focused on the dependability in Internet of things, parallel algorithms, and software-defined network.

Elias P. Duarte Jr. is a Full Professor at the Federal University of Parana, Curitiba, Brazil, where he is the leader of the Computer Networks and Distributed Systems Lab (LaRSis). His research interests include computer networks and distributed systems, their dependability, Management, and algorithms. He has published more than 200 peer-reviewer papers and has supervised more than 130 students both on the graduate and undergraduate levels. Prof. Duarte is currently Associate Editor of the IEEE Transactions on Dependable and Secure Computing and has served as chair of more than 20 conferences and workshops in his fields of interest. He received a PhD degree in Computer Science from Tokyo Institute of Technology, Japan, 1997, MSc degree in Telecommunications from the Polytechnical University of Madrid, Spain, 1991, and both BSc and MSc degrees in Computer Science from Federal University of Minas Gerais, Brazil, 1987 and 1991, respectively. He chaired the Special Interest Group on Fault Tolerant Computing of the Brazilian Computing Society (2005–2007); the Graduate Program in Computer Science of UFPR (2006–2008); and the Brazilian National Laboratory on Computer Networks (2012–2016). He is a member of the Brazilian Computing Society and a Senior Member of the IEEE.

How to cite this article: Venâncio G, Turchetti RC, Camargo ET, Duarte EP Jr. VNF-Consensus: A virtual network function for maintaining a consistent distributed software-defined network control plane. *Int J Network Mgmt*. 2021;31:e2124. https://doi.org/10.1002/nem.2124