Efficient Consensus Between Multiple Controllers in Software Defined Networks (SDN)

Stavroula Lalou
Department of Digital Systems
University of Piraeus
Piraeus, Greece
slalou@unipi.gr

Georgios Spathoulas
Dept. of Inform. Sec. and Comm. Techn.
NTNU
Gjøvik, Norway
georgios.spathoulas@ntnu.no

Sokratis Katsikas
Dept. of Inform. Sec. and Comm. Techn.
NTNU
Gjøvik, Norway
sokratis.katsikas@ntnu.no

***Abstract*—Software Defined Networking (SDN) has emerged as a popular paradigm for managing large scale networks. The traditional single controller architecture has limitations in managing the entire network: it can become a bottleneck when it comes to exchanging large volumes of data and it implies overhead as the number of user increases. Additionally, the single controller acts as a single point of failure because all the forwarding decisions depend directly on the controller. Once the SDN controller or the switches-to-controller links fail, the entire network may collapse. Therefore, scalability, reliability, interoperability, and fault tolerance remain as challenges in centralized network architectures. On the other hand, multiple controller architectures exhibit faster response and a more flexible network structure. Additionally, they can improve scalability and they avoid a single point of failure. In order to synchronize the network state between different controllers, a consensus protocol is required. In this paper, we propose a consensus mechanism, based on the Raft algorithm, which provides a stable, consistent, and efficient network in which all the controllers have the same network state. The proposed mechanism supports high throughput, dynamic view changes, fault tolerance, and controller synchronization. The performance of the proposed mechanism has been experimentally assessed and found to be very satisfactory compared to existing alternatives.**

***Index Terms*—Software Defined Networking; multiple controllers; Consensus Algorithm; fault-tolerance.**

I. INTRODUCTION

Single controller approaches are the main paradigm used to support SDN networks, but it fails to serve a number of critical domain requirements. Firstly, the efficiency of such centralized approaches is limited upon the resources of the single controller. Scalability is an issue, as is high availability, and security is of high importance as, if an attacker compromises the controller, management capability over the network is completely lost. Redundancy is one of the most significant aspects of any design. One controller could fail at anytime and, leave the network without a control plane. Multiple controllers can minimize the consequences of such a situation. Controllers operating normally could even collaborate to detect that another one is misbehaving and even isolate it from the network. Thus, having multiple controllers running at the same time and collaborating with each other enables the network to improve in terms of scalability, persistency, workload sharing and availability.

Consensus is the central protocol behind services replicated for fault tolerance. Consensus protocols are the foundation for building many fault tolerant distributed systems and services. A number of solutions have been proposed in this context.

In this paper, we introduce a novel mechanism that supports the operation of multiple controllers in an SDN network. The mechanism achieves network flexibility and enhances network management; it also synchronizes the network state between different controllers, while addressing single point of failure, fault tolerance and scalability issues. To demonstrate the practicality of the proposal, we present an implementation with the Raft algorithm [1] for state machine replication, whose performance we evaluated and compared to that of an existing alternative by means of experimentation.

The contribution of this paper is:

- The analysis of the existing mechanisms and protocols for SDN networks
- The definition of the consensus problem for distributed SDN controllers.
- The introduction of a mechanism that supports high throughput, dynamic view changes, fault tolerance, and controller synchronization in multiple SDN controllers setups.

The remaining of the paper is structured as follows: In section II, we briefly review necessary background knowledge on SDN and on the Raft consensus algorithm. In section III, we discuss related work. In section IV, we present our proposal for a mechanism supporting multi-controller SDN architectures. In section V, we present the experimental setup that we used for evaluating the performance of the proposal and we discuss the results. Finally, section VI summarizes our conclusions.

II. BACKGROUND

This section presents a review of the architecture of SDN and Raft Algorithm.

A. SDN Architecture

SDN has emerged as a new networking paradigm for implementing flexible network management solutions. Figure 1 depicts SDN architecture [2]. SDN is a network architecture where the forwarding state in the data plane is managed by a remote control plane decoupled from the former. The key principle of SDN is the separation of the control and data planes. The control plane is logically centralized and provides programmable application programming interfaces (APIs) for managing the physical layer. The data plane specializes in forwarding packets according to the instructions received from the controllers. In SDN, the controllers enable flexible management and unified control via programmable interfaces with a global view of the network status.

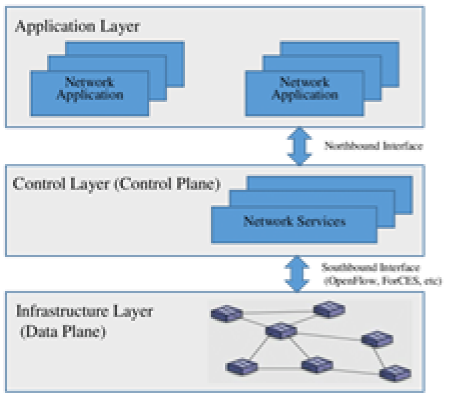

Fig. 1. Software Defined Networking.

The SDN is also defined as a network architecture:

- The control and data planes are decoupled. Control functionality is removed from network devices that will become simple (packet) forwarding elements.
- Forwarding decisions are flow-based, instead of destination-based. A flow is broadly defined by a set of packet field values acting as a match (filter) criterion and a set of actions (instructions). In the SDN/OpenFlow context, a flow is a sequence of packets between a source and a destination. [2].
- Control logic is moved to an external entity, the so-called *SDN controller* or *Network Operating System (NOS)*. The NOS is a software platform that runs on commodity server technology and provides the essential resources and abstractions to facilitate the programming of forwarding devices based on a logically centralized, abstract network view. It is similar to that of a traditional operating system [2]. The network is programmable through software applications running on top of the NOS, that interacts with the underlying data plane devices. This is a fundamental characteristic of SDN, and is considered to be its main value proposition.
- The network is programmable through software applications running on top of the NOS that interacts with the underlying data plane devices. This is a fundamental characteristic of SDN, considered as its main value proposition.

B. Raft Algorithm

Consensus is a fundamental problem for distributed systems. It pertains to getting a group of participants to reliably agree on some value used for a computation. Several protocols have been proposed to solve the consensus problem [3], and these protocols are the foundation for building fault tolerant systems, including the core infrastructure of data centers [1]. For example, consensus protocols are the basis for state machine replication [4], which is used to implement key services.

The Raft algorithm, depicted in Figure 2 [1] is a significant consensus algorithm for managing a replicated log. At the core of Raft lies a replicated log that is managed by a leader. Writes are funneled to the log and replicated throughout the cluster, through the leader. A leader election algorithm is integrated into the Raft algorithm to ensure consistency.

Raft separates the key elements of consensus, such as leader election, log replication, and safety, and it enforces a stronger degree of coherency to reduce the number of states that have to be considered in order to reach consensus. It also includes a mechanism for changing the cluster membership, which uses overlapping majorities to guarantee safety. There are three different node states, namely *leader*, *candidate*, and *follower*. Raft divides time into *terms* with arbitrary duration. Terms are monotonically increasing integers, where each term begins with an election. If a candidate wins an election, it serves as the leader for the rest of the term. Terms allow Raft servers to detect obsolete information such as stale leaders. Current terms are exchanged whenever servers communicate. When a leader or a candidate learns that its current term is out of date, then it immediately reverts to the follower state. Servers reject vote requests from the leader and replicated log entries with a stale term number.

A leader sends periodical heartbeats to all followers. If a follower receives no heartbeat messages over a predefined period of time (election timeout), it assumes there is no leader and starts a new election. It increments its current term, votes for itself, and moves to candidate state. Then, it sends request-to-vote RPCs to other servers. If it receives votes from a majority, it sends heartbeats to all servers to prevent new elections and establish its authority for its term. While waiting for votes, the candidate server may receive a heartbeat message from another server claiming to be the leader.

Raft is typically used to model replicated state machines. Leaders receive state machine commands and write them to a local log which is then replicated to followers in a batching approach. Once a command submitted to a leader has been logged and replicated to a majority of nodes of the cluster, the command is considered committed, and the leader applies the command to its own state machine and responds to the client with the logs. In the event of a server restart, the server replays the committed entries in its logs to rebuild the state of the server state machine.

According to the Raft algorithm a set of nodes can maintain a consistent shared data record. Each node can be a Master or a Candidate, and it sends messages to system nodes. If the Master fails, a new Master controller is chosen, following the process prescribed by Raft. Data records are funneled to the memory and then replicated throughout the cluster, and then through the leader. The leader checks all the data records

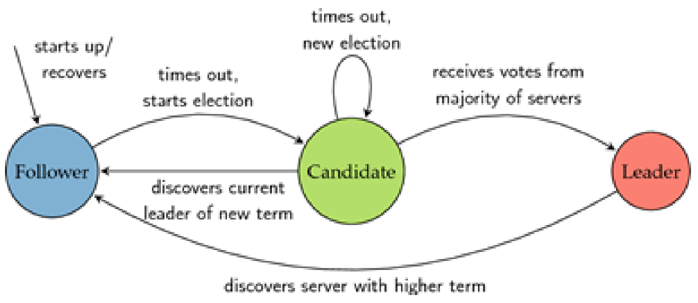

Fig. 2. Raft Consensus Algorithm.

and uses the election algorithm to ensure consistency.In our network we use Raft, so each node has a role either Master, Controller or Worker.

III. Related Work

The use of a single SDN controller offers flexibility and efficiency in network management, but leads to problems such as single points of failure and scalability issues. Existing studies propose multiple SDN controller architectures to address the above issues. The synchronization of the network state information among controllers is a critical problem, known as the controller consensus problem. To synchronize the network information between controllers, a proper consistency model should be chosen. *Strong consistency* and *eventual consistency* are two consistency models commonly used in distributed systems.

Many papers have proposed systems for SDN. The authors of [5] introduce a multicontroller SDN architecture, which employs a Fast Paxos based Consensus (FPC) algorithm to handle the consensus between multiple SDN controllers. The concept of leader election is also supported, as three roles, namely *Listener*, *Proposer*, and *Chairman* are applied to different controllers. The proposal was tested on a small-scale multicontroller architecture. In this research the evaluation of the performance was conducted in an ODL (OpenDaylight) Clustering on 3-node clustering. The FPC is composed of four phases, namely *Propose*, *Accept*, *Update*, and *Adjust*. A controller maintains a table that records its current controller state.

Hyperflow [6], [7] is described as a flat design. It is a distributed event-based control plane for OpenFlow. HyperFlow is logically centralized but physically distributed: it provides scalability while keeping the benefits of network control centralization. By passively synchronizing network-wide views of OpenFlow controllers, HyperFlow localizes decision making to individual controllers, thus minimizing the control plane's response time to data plane requests. HyperFlow is resilient to network partitioning and component failures. It also enables interconnecting independently managed OpenFlow networks, an essential feature missing in current OpenFlow deployments. The network is structured into several domains, where each domain is controlled by a controller situated within its own local network view. Controllers communicate with others through their east-westbound interfaces to get the global view of the network. Each controller only processes flow requests sent from the switches that belong to its local domain. Network events (e.g., flow information, routing information) are transmitted based on specific publish/subscribe mode among controllers [6].

Onix uses Paxos Consensus [8]. It is a multiple SDN controller architecture that provides a control application with a set of general APIs to facilitate access to the network state. It adopts a distributed architecture approach to offer the programmatic interface for the upper control logic and uses Network Information Base (NIB) to maintain the global network state. In Onix, the controller stores network information in key value pairs by utilizing the NIB, which is the core element of the model. It synchronizes the network state by reading and writing to the NIB, thus it provides scalability and resilience by replicating and distributing the NIB across multiple NIB instances. Once a change of a NIB on one Onix node occurs, the change will be propagated to other NIBs to maintain the consistency of the network.

ONOS [9] stands for Open Network Operating System. It uses Raft, it provides the control plane for a SDN, managing network components such as switches and links, and running software programs or modules to provide communication services to end hosts and neighboring networks. ONOS applications and use cases often consist of customized communication routing, management, or monitoring services for SDNs. [9].

Kandoo [10] is a typical hierarchical controller structure. The root controller communicates with multiple domain controllers to get the domain information, but the domain controllers do not contact each other.

DISCO (DIStributed multi-domain SDN COntroller) [11] is a distributed SDN controller scheme, implemented on top of Floodlight. It was introduced to partition a wide area network (WAN) into constrained overlay networks. A DISCO controller manages its own domain and communicates with other controllers via a lightweight and manageable control channel to provide end-to-end network services.

Akka [12] is a toolkit used in ODL Clustering and is responsible for communication and notification among controllers. In the default clustering scheme, switches connect to all controllers, and these controllers coordinate among themselves to choose a master controller. The ODL uses the Raft algorithm to reach controller consistency [2]. The Raft consensus algorithm periodically elects a controller as a leader controller, and all data changes will be sent to the leader controller to handle the update.

As observed from the existing distributed controller architectures, the problem of single point of failure of the SDN controller was solved using multiple distributed controllers. The Copycat project is an advanced, feature-complete implementation of the Raft consensus algorithm that diverges from recommendations. For instance, Raft dictates that all reads and writes are executed through the Master Controller (node), but Copycat's Raft implementation supports per-request consistency levels that allow clients to sacrifice linearizability and read from followers. Similarly, the Raft literature recommends snapshots as the simplest approach to log compaction, but Copycat prefers an incremental log compaction approach to promote more consistent performance throughout the lifetime of a cluster. Copycat's Raft implementation extends the concept of sessions to allow server state machines to publish events to clients.

IV. Our proposal

The proposed mechanism implements a novel network of multiple controllers using the Raft consensus algorithm. It supports the connection and coordination of multiple distributed SDN controllers to serve as backup controllers in case of a failure. Moreover, multiple controllers allow data load sharing when a single controller is overwhelmed with numerous flow requests. In general, our approach can reduce latency, increase scalability, and fault tolerance, and provides enhanced availability in SDN deployments.

The proposed mechanism, as shown in Figure 3, consists of a set of independent controllers (nodes), each one of which stores the required data in its memory. Each controller (node) is assigned with a unique id. In our implementation and test scenario we used a number of nodes in different states. Each one can be in one of three different states, namely *Master*, *Candidate* or *Worker*. In the *Master* state the node manages and controls the network while it can also process data and send update information to the other controllers. In the *Candidate* state the node can send and receive data to/from the other nodes. A Candidate node with the updated data can potentially transit to Master state if the current Master node fails. Finally in the *Worker* state the node passively receives data from Master or Candidate nodes.

If a Master node fails, the Raft election process is initiated to elect a new Master node and avoid single point of failure effects. In this process a node in Candidate state will be elected and will act as Master node, while the previous Master node will switch to Candidate state. Through this process the system maintains its stability and fault tolerance.

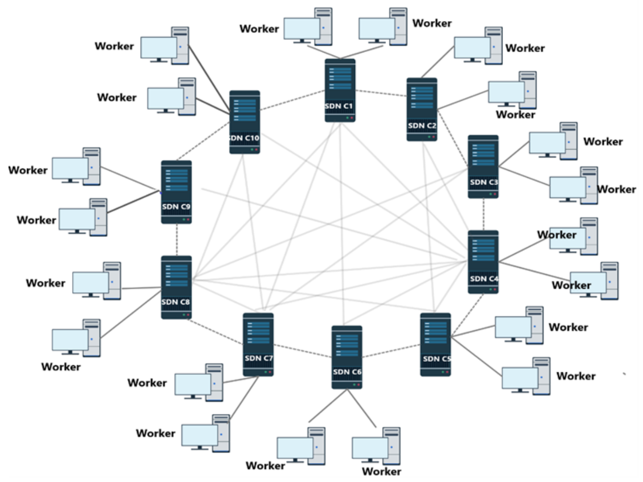

Fig. 3. Proposed mechanism.

Inputs are introduced to the SDN multiple controllers network by clients (nodes). When new data are introduced in the mechanism by a node, according to the Raft protocol, such data is forwarded to the Master Controller. Once the Master Controller receives a series of information, it logs and replicates these to all the mechanism controllers, which store such data in their memory.

When the Master Controller receives a series of data, a broadcast process is initiated to send such data to all nodes in the network and update information in all controllers accordingly. Information is stored in the memory of the Master controller and its initial state is defined as not read. The Master node sends data to all Candidate nodes and the latter forward such data to their neighboring Worker nodes. The Master node monitors if all Candidate nodes have received and stores the new data to ensure that all of them have an updated memory. Each Candidate node makes sure that it records newly sent data, while it also monitors data records to avoid duplicate ones. Consequently, each Candidate node sends data to attached Worker nodes. During this process the same approach is followed to ensure successful delivery of data to all nodes.

When all nodes have successfully forwarded all data, the mechanism transits to an ”OK” state when all controllers have the same data stored in memory. If a controller receives new data, then it stops being in the “OK” state and data shall be sent to the other controllers and workers (neighbor nodes) of the mechanism according to the procedure which has been described previously.

In the proposed mechanism the main entity is the Raft node which shall be deployed along with each SDN controller in an SDN setup. Each node keeps in its memory a set of records which adhere to the structure *record (data, send)*. The variable *data* holds the information to be exchanged and the variable *send* is Boolean and is used to flag whether a specific record has been successfully forwarded to the network. Nodes are identified by a unique id.

The Raft consensus algorithm is used to coordinate the sharing of information between nodes. It defines the creation of a group of controllers (candidates) and the required processes to elect one leader the Master controller. The Master is the one who manages the data flow in the mechanism and leads the group if it is active.

If the Master node fails, then a new Master node must be elected through the mechanism process. Specifically, a time is defined in which the Master sends a message to the other controllers. If the message does not arrive in time, a Controller node sends a message requesting to become the Master node. In this case the other controllers respond, and the specific node is designated as the Master node.

The main processes that nodes operate upon to maintain consistency, stability, and availability are the following:

- The *read* process, that reads from the mechanism memory and checks records and if those have been successfully distributed to others.
- The *send* process that sends data to other controllers.

- The *send-to-all* process, that is responsible for iteratively sending data to all neighboring controllers.

V. PERFORMANCE EVALUATION

To evaluate the performance of the proposed mechanism we conducted a network simulation. Also we compare it with other mechanisms in terms of consensus time, distribution time, data access time and presenting test results.

A. Experimental Setup

Table II shows the main simulation parameters.The system on which the simulation was performed was based on an AMD Ryzen 5, 4500 U CPU and the goal was to evaluate the time required for the main functionalities of the proposed design.

TABLE I
SIMULATION PARAMETERS

Parameters	Value
Consensus algorithm	Raft
Data	In all node types
Number of controllers	10
Number of Workers per Controller	2
Test environment	Windows 10
Hardware	AMD Ryzen 5, 4500 U
Compiler Code	NetBeans 8.2 Java

To evaluate the proposed approach, we have run an
periment to assess the time response of the algorithm.
first run an experimental simulation of a failure scenaric
which the proposed algorithm is executed for 100 sec fc
mechanism with 30 controllers, 10 of which run as Ma:
nodes and 20 run as Worker nodes. We check that datε
being transferred correctly between nodes. In this scenario
assume that at a specific time point, around 20 sec after
start, the master node fails. The main objective is to maint
mechanism stability at all times and avoid the effects o
single point of failure.

To extend the initial scenario, another test was also e
cuted, in which the newly elected Master nodes drop at ti
points around 20 sec, 30 sec and 60 sec respectively. All no
have been monitored to test the read and write performa
in each node (in Master, Controller or Worker states).

In the tests, all the nodes except those of the original Master node and the Worker nodes connected to it have the same data after the initially set Master node crashes. Another node is elected as the new Master node. This is repeated a number of times during the execution of the experiment.

The proposed mechanism reduces the network overhead. Moreover, it maintains the proper network operation, even when there is a controller failure. Moreover, it offers controller synchronization, as all network controllers have the same data. In addition, it preserves it's reliability, scalability, fault tolerance, and interoperability. In the event of Master node failure, a new master controller would be selected to take the control of the network. The mechanism of choosing a new master is through the Raft consensus algorithm process, so the proposed mechanism enjoys high availability rates. The conducted experiment simulates the operation of a distributed mechanism consisting of remote computers or systems.

B. Results

During the tests we compare the proposed mechanism and the Paxos-based mechanism [8], in terms of consensus time, distribution of normalized consensus time, and data access time. The consensus time is the time that elapses from when a transaction is created to when the transaction is committed.According to the research these are closest to our approach and are implemented according to distributed SDN multiple controller architecture and consensus algorithms.

TABLE II
COMPARISON OF PROTOCOLS/ALGORITHMS

Feature	Proposed	CopyCat Raft	Paxos
Consensus algorithm	Raft	Raft	Paxos
Controllers	10	1	3
Workers	2/controller	1 client	6 End Users
Time to read data	10 ms	0.20 s	6.425 ms
Time to write/send data	10.4 ms	0.40 s	17.814 ms
Time to elect a Master	10.06 ms	0.060 s	28 ms

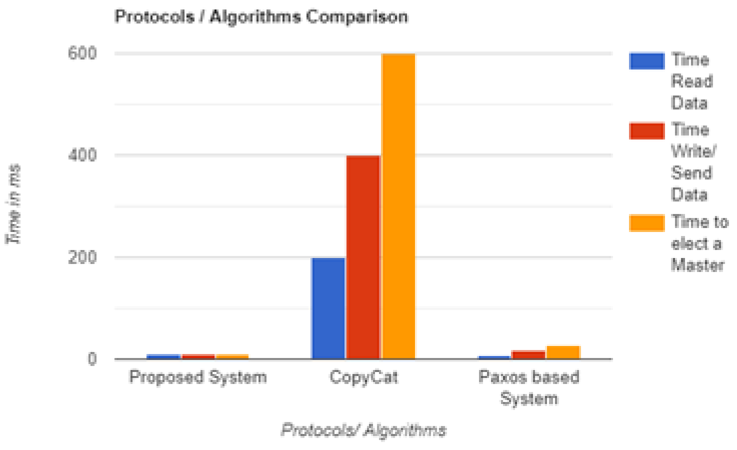

Fig. 4. Comparison of Protocols/ Algorithms.

Table II shows the basic features of the proposed mechanism in comparison to the CopyCat project and a Paxos-based system [8], that we have tested. All models are using consensus algorithms and a distributed SDN multiple controller architecture. As is shown in Table II, through the comparison between the Copycat project and the proposed mechanism it is clear that Copycat requires more computing resources than the proposed mechanism and this makes Copycat less reliable for large scale networks. Also, the time required for reading, writing and sending data is higher than the other two mechanisms. The average time that the Copycat system needs to start and elect a Master Controller is 23.23 seconds.

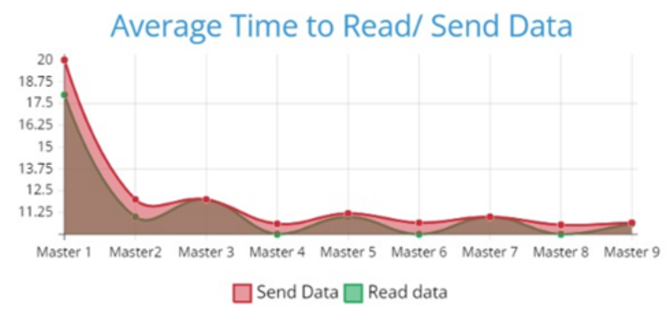

Fig. 5. Read and Write Average Time.

The consensus time of the proposed mechanism is stable. Also, the Write/ Send Data time is stable and low (see Figure 5); the proposed protocol needs 10.4 ms. Low and stable times for reading and sending data can improve the mechanism performance and offer a stable and functional mechanism architecture. Furthermore, all controllers had the same data even after a controller drop in our simulation environment of 100 seconds; the network maintains its stability.

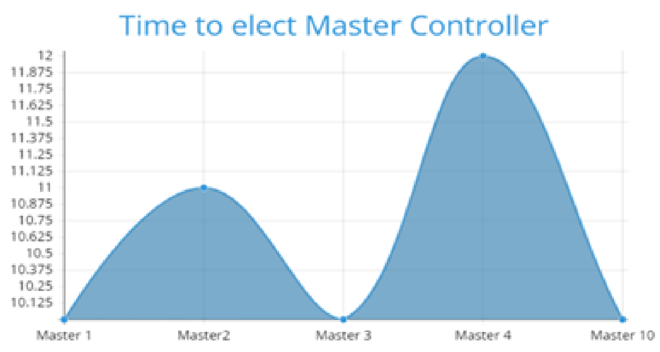

Fig. 6. Controller election Time with Raft.

The test results have shown that for the proposed mechanism the average required time for Master node election is 10.06 ms (see Figure 6). It is stable and low, as shown in Fig. 4 and described in Table II.

VI. CONCLUSION

SDN is a promising paradigm for network management because of its centralized network intelligence. However, the centralized control architecture of SDNs raises challenges regarding reliability, scalability, fault tolerance and interoperability. The existing solutions which were analyzed in literature are not offering high-throughput, fault-tolerance, and controller synchronization. We proposed a novel implementation, based on the Raft algorithm, that can efficiently synchronize the network state information among multiple nodes, thus ensuring good performance at all times irrespective of the traffic dynamics. Further, the proposed mechanism supports high-throughput, fault-tolerance, and controller synchronization. Our simulation results have shown that the proposed mechanism can support Multiple Controllers, as it maintains stability (all nodes have the same data, after a Master node failure) and the average required times are low. The average time it takes to read, write in memory, and send data to neighbor controllers is low and stable. Also, the time it takes to elect a new controller is also low. In our proposal, multiple controllers maintain a consistent global view of the network. This is achieved by employing the Raft consensus protocol to ensure consistency among the replicated network states maintained by each controller.

ACKNOWLEDGMENTS

This work has been partly supported by the University of Piraeus Research Center and also was funded in part by the Research Council of Norway under project nr. 310105 ”Norwegian Centre for Cybersecurity in Critical Sectors”.

REFERENCES

[1] D. Ongaro and J. Ousterhout, ”In search of an understandable consensus algorithm”, in Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference (USENIX ATC’14), USENIX Association, USA, 2014, pp. 305-–320.
[2] D. Kreutz, et al., ”Software-Defined Networking: A Comprehensive Survey”, in Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, Jan. 2015, doi: 10.1109/JPROC.2014.2371999.
[3] L. Lamport, ”The part-time parliament”, ACM Trans. Comput. Syst., vol. 16, no. 2, pp. 133-–169, May 1998, https://doi.org/10.1145/279227.279229.
[4] A. Abdelaziz et al, ”Distributed controller clustering in software defined networks”, PLoS ONE, Vol. 12, no. 4: e0174715, April 2017, https://doi.org/10.1371/journal.pone.0174715(2017).
[5] C. -C. Ho, K. Wang and Y. -H. Hsu, ”A fast consensus algorithm for multiple controllers in software-defined networks,” 2016 18th International Conference on Advanced Communication Technology (ICACT), pp. 1–1, 2016, doi: 10.1109/ICACT.2016.7423293.
[6] D. Dotan and R. Y. Pinter, ”HyperFlow: an integrated visual query and dataflow language for end-user information analysis”, 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC’05), 2005, pp. 27–34, doi: 10.1109/VLHCC.2005.45.
[7] A. Tootoonchian and Y. Ganjali, ”HyperFlow: a distributed control plane for OpenFlow”, in Proceedings of the 2010 internet network management conference on Research on enterprise networking (INM/WREN’10), USENIX Association, USA, 2010.
[8] R. Y. Shtykh and T. Suzuki, ”Distributed Data Stream Processing with Onix,” 2014 IEEE Fourth International Conference on Big Data and Cloud Computing, 2014, pp. 267–268, doi: 10.1109/BDCloud.2014.54.
[9] P. Berde et al, ”ONOS: towards an open, distributed SDN OS”, in Proceedings of the third workshop on Hot topics in software defined networking (HotSDN ’14), Association for Computing Machinery, New York, NY, USA, pp. 1-–6, 2014, https://doi.org/10.1145/2620728.2620744.
[10] S. H. Yeganeh and Y. Ganjali, ”Kandoo: a framework for efficient and scalable offloading of control applications”, in Proceedings of the first workshop on Hot topics in software defined networks (HotSDN ’12), Association for Computing Machinery, New York, NY, USA, 19-–24, 2012, https://doi.org/10.1145/2342441.2342446.
[11] K. Phemius, M. Bouet and J. Leguay, ”DISCO: Distributed multi-domain SDN controllers”, 2014 IEEE Network Operations and Management Symposium (NOMS), 2014, pp. 1–4, doi: 10.1109/NOMS.2014.6838330.
[12] W. Xia, Y. Wen, C. H. Foh, D. Niyato and H. Xie, ”A Survey on Software-Defined Networking”, in IEEE Communications Surveys & Tutorials, vol. 17, no. 1, pp. 27-51, Firstquarter 2015, doi: 10.1109/COMST.2014.2330903.