A Fault-Tolerant and Consistent SDN Controller

Andres J. Gonzalez
Telenor Research
Telenor ASA
Trondheim, Norway
andres.gonzalez@telenor.com

Gianfranco Nencioni, Bjarne E. Helvik
Department of Telematics. Norwegian
University of Science and Technology
Trondheim, Norway
{gianfranco.nencioni, bjarne}@item.ntnu.no

Andrzej Kamisiński
Department of Telecommunications. AGH
University of Science and Technology
Kraków, Poland
kamisinski@kt.agh.edu.pl

***Abstract*—Software-Defined Networking (SDN) is a new paradigm that promises to enhance network flexibility and innovation. However, operators need to thoroughly assess its advantages and threats before they can implement it. Robustness and fault tolerance are among the main criteria to be considered in such assessment. The currently available SDN controllers offer different fault tolerance mechanisms, but there are still many open issues, especially regarding the trade-off between consistency and performance in a fault-tolerant SDN platform. In this paper, we describe existing fault-tolerant SDN controller solutions, and propose a mechanism to design a consistent and fault-tolerant Master-Slave SDN controller that is able to balance consistency and performance. The main objective of this paper is to bring the performance of an SDN Master-Slave controller as close as possible to the one offered by a single controller. This is obtained by introducing a simple replication scheme, combined with a consistency check and a correction mechanism, that influence the performance only during the few intervals when it is needed, instead of being active during the entire operation time.**

***Index Terms*—Master-Slave Controller; SDN performance; SDN dependability; Fault Tolerance; Controller Consistency.**

I. INTRODUCTION

SDN is a new paradigm that promises better network flexibility, programmability, and innovation. However, due to the demanding carrier grade requirements, operators must assess the potential risk and the involved drawbacks carefully, as well as explore successful ways to overcome them, before moving on to the implementation phase. In this context, availability and fault tolerance are essential criteria that must be evaluated. This issue has been previously studied in several papers in which the problem has been addressed in two different fault tolerance domains: 1) The data plane and 2) The control plane. The fault tolerance of the data plane includes such proposals as [12], where a mechanism to obtain path failure recovery times below 50 milliseconds is presented. The work presented in [11] is also focused on the data plane by providing fast-failover mechanisms to react to link or switch failures. On the other hand, the basic SDN architecture depicts the control plane as a potential single point of failure. Therefore, the design of a fault-tolerant control plane is a must. The straightforward solution is to have redundant controllers that can take responsibility in the case of a failure. However, as Section II-B will explain, the mechanisms used to implement such a solution are complex and pose challenges that cannot be neglected.

This paper is focused on the fault tolerance of the control plane with an assumption that consistency and performance are important criteria and must be considered. We define the **performance** of a controller according to the following two concepts: 1) Controller Latency: The time that an incoming request has to spend waiting once it is delivered at the ingress switch, until the respective new data plane rules are established in the respective switches. 2) Controller Throughput: The maximum number of new flows handled within a time unit. In addition, we define **consistency** as the ability to have exactly the same view of a network at all controllers, which is an exact mapping of the network state, at any time.

Maintaining a consistent view of a network among all controllers is challenging. The existing mechanisms used for this purpose tend to affect negatively the performance of the controller platform. Thus, having mechanisms that achieve fault tolerance and consistency by keeping the performance consequences low is a clear goal in the research community. In this context, the main questions addressed by this paper are: 1) How to bring the performance of a fault-tolerant controller close to that of a non-redundant (single) controller? 2) How to implement consistency-guard mechanisms without affecting the system during its entire operation time?

A fault-tolerant controller platform may be implemented by distributing the load among separate controller units, which means that the responsibility has to be spread over several domains [8], [13]. On the other hand, there are simpler approaches known as Master-Slave, where a single controller is in charge of all decisions. It is supported by backup controllers having a synchronized view of the network, which can take the responsibility for future decisions in the case of a failure [2], [7], [10]. Further details on the load-distributed and Master-Slave approaches will be provided in Section II-C. This paper is focused on Master-Slave scenarios, since we believe that due to their simpler structure, they offer a better possibility to improve the trade-off between consistency and performance.

To the best of our knowledge, all previous proposals rely on consistency-guard mechanisms that decrease the overall performance during the entire controller operation period. Based on that, there are two important remarks that motivate our paper: 1) Failures are very serious events that should always be addressed and quickly solved. 2) Failures are rare events whose failure handling times represent a very small fraction relative to the entire operation time. In conclusion,

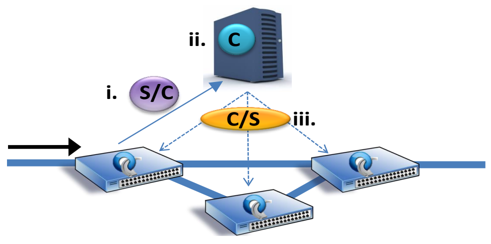


Fig. 1. Operation of a Single SDN Controller.

failures should be handled properly, but at the same time, the mechanisms necessary for the handling should not affect the performance during the entire operation time. Having this as a guideline, the main contributions of this paper are:

- Proposing a Master-Slave fault-tolerant controller that achieves consistency and fault tolerance, with performance close to the single controller scheme.
- Proposing mechanisms that guarantee consistency without affecting the controller performance during ordinary operation.

This paper is organized as follows. In Section II, we present performance-related shortcomings of the single SDN controller and previous works on the design of fault tolerant controllers, in order to define the problem and challenges addressed in this paper. Section III describes our proposed fault tolerant Master-Slave SDN controller, and provides details on the different steps executed under different operation modes. In Section IV, we evaluate the performance (latency and throughput) of the proposed model. Finally, Section V concludes the paper.

II. Performance, Consistency and Fault Tolerance in an SDN controller

Before presenting our Mater-Slave controller, in this section we mention all the concepts needed for its elaboration. We start by describing the main features and concerns of a Single SDN controller, continuing with the description of previous works on the design of a fault tolerant control plane. Finally, we define the open challenges that this paper addresses, as a guideline and motivation for Section III.

A. The Single SDN Controller and its Performance

In a single SDN controller, when a new flow with no specified forwarding instructions comes into an SDN switch, the following actions are performed:

i. A packet representing a new flow is received at a network ingress switch and it is sent to the controller.
ii. The controller computes the forwarding path, depending on the flow request, the network policies and its current network view.
iii. The controller updates the respective switches by sending entries to be added to the flow tables.

After these steps, all subsequent packets of the new flow are forwarded, based on the pre-calculated forwarding decisions and do not need any control plane action. The mentioned steps are presented in Figure 1, and it is the simplest and conventional way to operate new flows in an SDN network. Since the launch of the first SDN solutions, there has been a huge interest in modeling and analyzing the controller performance, motivated by the requirements of a large-scale deployment. For instance, [6] and [9] evaluate how a single controller architecture will perform under certain parameters, in order to foresee potential implementation issues. Their results show that the performance of an SDN network has a strong dependency on the processing speed of the controller. If it is not fast enough, the capability of the network to handle new flows is limited considerably, affecting the overall user experience.

In spite of its potential limitations and consequences, the model presented in this section (Figure 1) is the simplest and best performing SDN architecture. Therefore, the main point of this paper is to develop a consistent and fault-tolerant Master-Slave controller, with performance as close as possible to the one offered by a single controller.

B. Fault Tolerant SDN Controllers

A single controller is a single point of failure, and hence unacceptable. Having a fault-tolerant SDN controller is a well identified need addressed in several previous works. Among them, one of the first and most relevant proposals is ONIX [8]. They elaborate the main concepts and pillars to be considered in the implementation of a general fault-tolerant controller platform, which have been used as a reference in most of the further related works. ONIX also provides a general framework where several important open issues have been identified, such as the trade-off between consistency, durability, and scalability. Although the specific solution remains open and up to the specific implementation needs. One of the most important concepts that our paper takes from ONIX is the Network Information Base **NIB**. They defined the NIB as *a copy of the current network state which contains relevant information from all network entities within the topology*. We follow the same definition, and from now on, we assume the NIB as all information that the control plane must know in order to perform network operations, modifications and all related decisions. Hyperflow [13] is another widely used fault-tolerant controller proposal. It is a distributed event-based control plane for OpenFlow that is logically centralized but physically distributed. The platform localizes decision-making to individual controllers, ensuring that all the instances are synchronized, considering inconsistency problems due to the synchronization speed.

As mentioned in [8], the design of a distributed SDN controller needs to take into account conventional best practices from standard distributed systems. After some years of addressing this issue, one of the solutions that have got good acceptance in real implementations for the coordination of distributed controllers is Zookeeper [5]. It uses state machine replication methods (SMR), where *all write requests are forwarded to a single server, called the leader*. It offers strong consistency and can achieve high throughput and low latency.

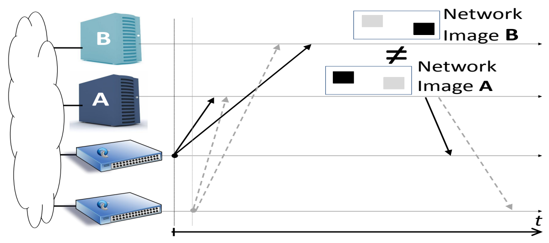


Fig. 2. Potential inconsistency scenario under conventional Active Replication.

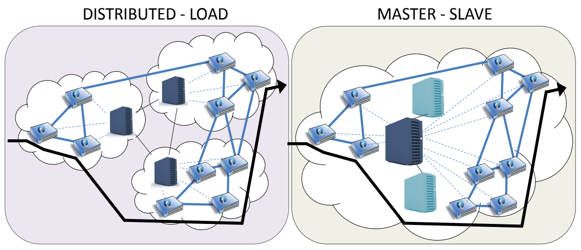


Fig. 3. Two Different Fault-Tolerant Controller Approaches.

On the other hand, in the literature of fault-tolerant SDN controllers, Master-Slave controllers are a subclass that address the problem in a simplified way, since there is one central unit (the master) in charge of taking the decisions to be implemented on the data plane. A simple Master-Slave approach was given in [3], where active and passive replication methods are proposed to provide fault tolerance. However, this approach does not consider consistency issues. On the contrary, two papers on the master-slave architecture that consider the consistency problem are [2] and [10]. Both propose an architecture that uses a shared datastore on top of the controllers, in order to keep a global common NIB. Consistency is kept by replicating every NIB change, but this has a high impact on the controller's performance, given the time overhead produced by each replication. Finally, a recent work [7] (Ravana) proposes a fault-tolerant Master-Slave SDN controller that processes incoming messages transactionally and exactly once. Ravana implements replicated state machine methods by using ZooKeeper, but extends its scope by implementing switch-side mechanisms to guarantee correctness. This work is novel and solid, and hence we will follow some of its approaches. However, we propose some variations in order to address the challenges that will be presented bellow.

C. Problem Definition and Main Challenges

The SDN controller must be fault-tolerant and highly available. In virtualized environments, active and passive replication are two common techniques used to achieve this [4]. Since active replication is a solution that offers high resilience and negligible downtime, its use may be seen as the primary alternative [3]. One intuitive solution is to duplicate all the messages sent by a switch, with the intention that both controllers receive a copy and perform identical operations. This approach may easily create inconsistencies between the network image of the controllers due to reordering of events generated by delay differences, as Figure 2 illustrates. Even if we were able to enforce a strict order, non determinism in the processing poses an additional huge challenge. This is the root problem of the performance/consistency balance, and it leads to the definition of the first challenge:

Challenge 1: Avoid single point of failure, considering synchronization problems between replicas caused by non-deterministic delays and processing.

There are many approaches used to design a fault-tolerant SDN controller, as pointed out in Section II-B. Here, we want to highlight the fundamental difference between a distributed-load and a Master-Slave approach, as illustrated in Figure 3. The distributed-load approach targets large scale networks where a single controller has difficulties to manage the entire load, and hence the operation has to be split into several domains with the corresponding dedicated controllers. In this design, fault tolerance may be provided by making the appropriate load redistribution in the case of controller failures. In this way, the controllers can act independently and in parallel, as long as they share a common network view, i.e., same NIB. For this; shared data stores, state machine replication techniques and consensus protocols are commonly used, such as those provided by Zookeeper [5]. In these scenarios, if consistency is prioritized, the performance degradation is unavoidable [8].

On the other hand, a Master-Slave SDN controller is a simpler concept in which only one controller (the master) is in charge of all decisions, while the backup controllers (slaves) are used to provide fault tolerance. This approach may be implemented in small or medium scale networks. However, one of the main challenges of this kind of design is to guarantee the consistency between the network image (NIB) of the master and slave controllers. Even though this is a much simpler design, the available solutions use similar tools to those used in the distributed-load case, where the performance is drastically affected. For instance, the implementation presented in [2] shows that the controller throughput may be reduced almost five times when the datastore is used regularly. This leads to the second challenge:

Challenge 2: Implement a simpler and less performance-expensive Master-Slave platform that guarantees NIB consistency.

Having a consistent NIB is important to guarantee the appropriate operation of a Master-Slave controller. This is achieved in previous works using two different approaches. The first one proposes that any change in the NIB is processed only by the master controller, but before sending the update to the switches, the NIB-update is replicated and synchronized on all the backup controllers. The second approach consists of the use of a common datastore that imposes a rigorous order on the incoming request messages sent by switches, and after a consistency check, they are processed independently by each replica, assuming deterministic controller processes (such as the one presented in [7]). This brings the third challenge:

Challenge 3: Find which approach is better. Having repli-

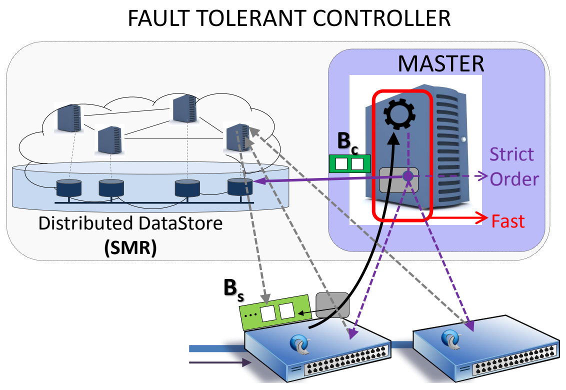


Fig. 4. Proposed Master-Slave SDN Controller.

cation of NIB updates, or having replication of incoming messages.

An important dependability issue may occur during the SDN controller recovery, as some requests may be lost. This will degrade user experience, but also some critical messages such as failure recovery requests may be omitted, creating catastrophic consequences. A solution to this problem has been proposed in [7]. They use a *switch-runtime* buffer on every ingress node in the data-plane, and when a new packet is sent to the controller, the switch buffers the event temporarily. This enables a recapture of events lost during controller recovery. The process includes the following five steps: i.) The packet is buffered in the switch and a copy is sent to the master controller. ii.) The incoming packet is replicated in the datastore. iii.) After the replication, the master controller processes the message. iv.) The master sends a response to each of the involved switches, and the switch-buffered copy is deleted v.) Switches acknowledge the received message. This leads to the fourth challenge:

Challenge 4: Is it viable to simplify the *no-lost-event* mechanisms proposed in [7], without scarifying resilience and consistency?

III. A Fast, Consistent and fault-tolerant Master-Slave SDN Controller

The Master-Slave controller proposed in this paper is presented in Figure 4. It is composed of a master controller and a distributed datastore used by backup SDN controllers to keep the NIB information.

Our work follows a hybrid Master-Slave approach, in the context of the previous works presented in II-B and *Challenges 3 and 4*, where: i.) The controller-NIB-updates are replicated. ii) The switch stores the requests sent to the controller in a buffer named B_S (as proposed in [7]), and NIB-Updates are buffered by the Master in B_C. According to the previously identified challenges, our goal is to keep the processing of the master controller (red box in Figure 4) as fast as possible, without scarifying consistency. Our proposal may be summarized as follows. The master attends messages in the same way that a single controller does it, but with two differences:

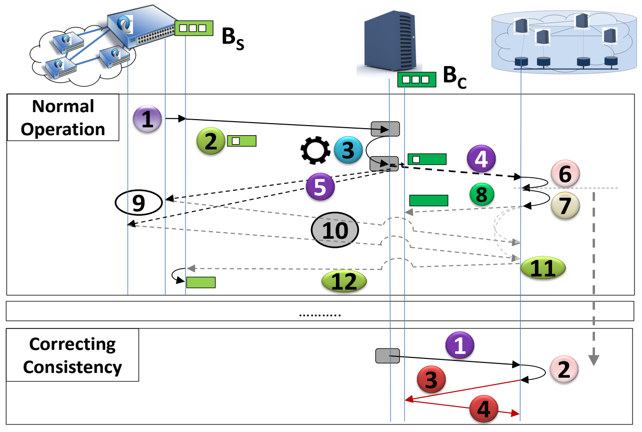


Fig. 5. Sequence Diagram during the normal operation (Fig. 5) of the controller and during a consistency correction routine.

1.) After processing an incoming message, each NIB update is immediately split, stored, and sent to the common datastore with an additional and consecutively-increased ID used for consistency checks. 2.) Consistency assurance procedures rely on the communication between switch buffers B_S and the slave platform, as well as on NIB-updates split and buffered from a unique reference point, providing some degree of Master independence. With the proposed modifications, our intention is to keep the overhead on the master operation as low as possible.

Under normal operation, the following steps are performed (see upper part of Figure 5).

1) A message is sent from the switch to the controller, denoting a new flow arrival, a switch state change, an error, or in general a controller incoming message.
2) A copy of the sent message is kept in B_S.
3) The master controller processes the incoming message (process with expected time duration μ_M).
4) When the master finishes to process the message, a copy of the answer (NIB update) is sent to the datastore and also stored in the controller buffer B_C, in both cases with the respective incremental ID.
5) Consecutive to Step 4, the master sends the answer to the respective network switches.
6) **In parallel** (Steps 6 and 7), the datastore leader receives the message and checks the consecutive ID to prevent against potential NIB update losses.
7) The datastore replicates the NIB update.
8) The slave platform confirms the synchronization of the NIB update, and the respective entry in B_C can be deleted.
9) Consecutive to Step 5, the respective data plane rules are recorded on the switches.
10) A successful data plane change message is sent from the switches to the datastore.
11) The datastore confirms the successful end-to-end data plane modification by comparing the register obtained after Step 7, and it sends a confirmation to the switch.

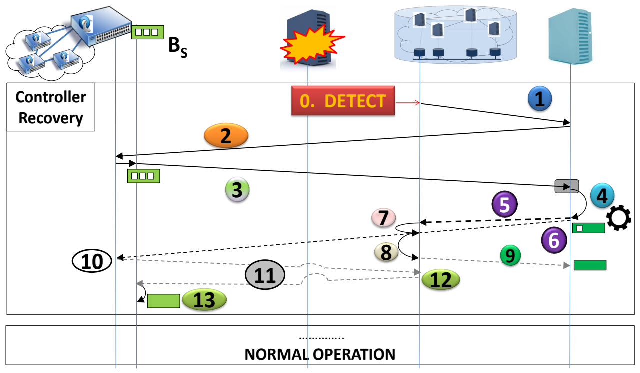


Fig. 6. Sequence Diagram during a master failure recovery routine.

12) The message is removed from the switch buffer B_S.

As presented in *Challenge 1*, the non-determinism in delay and controller processing is the core problem to solve. Our scheme guarantees a strict and universal-controller order of processed messages to be set in the network, by having the master *output-point* as reference, as illustrated in Figure 4 ('*Strict Order*'). Packet losses may generate inconsistencies on the performed ID check (Step 6). When this is detected, the following steps are performed (See lower part of Figure 5).

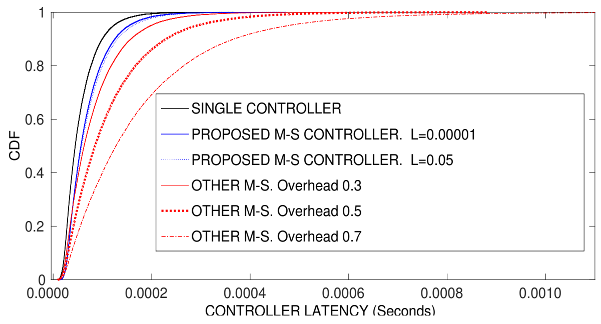
1) The master sends to the datastore a copy of the message with an incremental ID (Same step 4 in Fig. 5).
2) The consecutive ID checking raises a flag of a potential inconsistency.
3) The datastore leader requests again the identified missed messages.
4) The controller buffer B_C provides the requested message.

Previous works agree that the most effective way to detect controller failures is through a distributed datastore system ([1], [2], and [7]). For instance, [1] mentions the use of ZooKeeper [5] to detect and react to controller failures. In this paper, we present the recovery routines performed after the failure is detected, as Figure 6 illustrates.

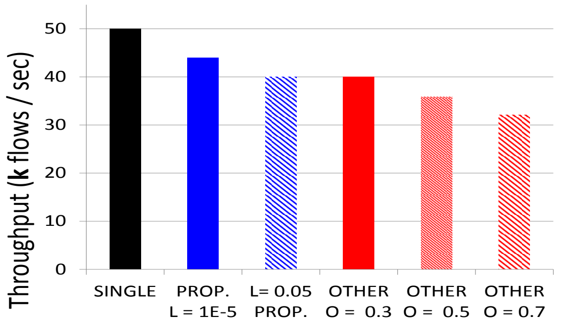
1) A new master is selected.
2) The new master notifies the network switches about its new role.
3) The unattended messages stored in the switch buffers B_S are sent to the new master.
4) Steps 4 to 9 in the recovery process are the same Steps 3 to 8 under normal operation, but here, the incoming requests are the remaining unattended messages on the different B_S's. In Step 10, the respective data plane rules are recorded on the switches. Finally, in Steps 11, 12 and 13 confirms the successful end-to-end data plane modification in order to safely delete the messages from the B_S buffers.

IV. Performance Evaluation

In this section, we evaluate the performance of the proposed controller and compare it with a single controller and other Master-Slave approaches. In order to do this, we use Discrete Event Simulation.



(a) CDF of the Latency of the Controllers.



(b) Throughput of the different Controllers

Fig. 7. Performance evaluation of different existing controllers and the proposed solution.

In all the analyzed controllers (Single controller, the proposed solution, and other Master-Slave controllers), the processing phase of the incoming message (Step 3 in Figure 5) is common, and it is the core process that will affect the performance of all controllers. Here, we assume this as a stochastic process with expected time duration μ_M. For the Single Controller case, we only consider Steps 1,3,5 and 9 of the sequence diagram under normal operations. The simulation of our Master-Slave proposal follows all the concepts previously presented in Section III. Finally, since we lack specific technical details in order to have a fair comparison under exactly the same circumstances with other Master-Slave controllers such as [2], and [7], we use the following approach. The key difference of our proposal is that each new flow that arrives to the controller can be attended directly, being the time used on splitting and buffering routines, the only regular overhead. Consistency check routines may represent an important overhead, but they are not applied on each controller operation. On the other hand, other Master-Slave approaches such as [2], and [7] impose an additional overhead on each incoming request, due to synchronization routines. Therefore, for the simulation of other Master-Slave controllers, we assume that the average *overhead* O_M per request is a percentage of the time needed for only processing that request, i.e., if the expected processing time of a single request is μ_M, and $O_M = 1$ (100% overhead), the expected time needed to process a new request would be $2\mu_M$.

Figure 7(a) shows the Cumulative Distribution Function

(CDF) of the latency, after simulating the three different types of controllers previously mentioned. For the evaluation, we assume the following scenario. For common parameters such as the core controller processing time, we assume a negative exponential distribution (*n.e.d*) with expected value of 20 microseconds. We emulate a network with 12 OpenFlow-compatible switches, each with a *n.e.d* arrival process of new flows (addressed by the SDN controller) with a mean value of 400 microseconds in all the cases. The transmission delay in the OpenFlow channel is assumed to be uniformly distributed with a minimum of 4 microseconds and a maximum of 16 microseconds, based on the results presented in [6]. For the case of our proposal, Figure 7(a) evaluates two different packet loss probabilities L between the master and the datastore, representing the probability that the consistency correction mechanisms is applied. The first is a packet loss probability of 0.00001 and the second one is 0.05 (not realistic in real operational networks, but just for the sake of illustration). Finally, for the evaluation of other Master-Slave controllers, we test different overhead values. We observe that among all the evaluated controllers, the latency of our Master-Slave controller has the closest performance to those from a single controller. We also observe that the probability of losing a packet does not have a strong influence on the controller latency. However, in Figure 7(b) will be shown that its effect on the throughput is more noticeable. Finally, other Master-Slave approaches may operate with latency values close to the single controller, if the overhead possed on each replication is very low. However, obtain small overhead values (e.g. lower than 0.3) poses a huge technical challenge.

Figure 7(b) shows the controller throughput in terms of the number of flows per second attended by the controller. Here, we consider the same controllers, i.e. single, our proposal with a packet loss probability of 0.00001 and 0.05 and other Master-Slave controllers with overhead values of 0.3, 0.5 and 0.7. We observe that under typical conditions, i.e., low packet loss probability, our approach is the one that provides the closest throughput to the one offered by a single controller. The throughput of the proposed Master-Slave can be affected if the packet loss increases to very high values. However, this is not realistic in operational scenarios. On the other hand, the overhead presented in other Master-Slave controllers reduces considerably the throughput that they can offer.

V. Conclusion

Answer to Challenge 1: Our proposal does not have a single point of failure and overcomes synchronization problems caused by delay differences and non determinism in control processes. This is done, by having a unique reference point complemented by additional tools to keep consistency.

Answer to Challenge 2: The proposed solution takes advantage of the potential simplification opportunities of a Master-Slave solution by proposing consistency aware routines, executed when they are needed, instead of having them active during the entire controller operation.

Answer to Challenges 3 and 4: Our final solution is a hybrid approach based on NIB updates replication, but it is complemented by routines proposed by Ravana [7] (such as the use of B_S) that guarantee consistency and allow a higher independence between the master and the slaves. It is important to clarify that our scheme under a controller recovery routine may present a small probability that some messages are attended twice. This implies that some rules may be rewritten, but at the same time, customers will not experience additional downtime.

In order to deliver a competitive performance, our Master-Slave controller needs a very reliable communication channel between the master controller and the datastore. However, this is by default one of the most important criteria for providers when implementing a fault-tolerant controller system. Therefore, we assume that this is a must, regardless of the fault-tolerant controller approach used.

References

[1] Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W., et al. ONOS: towards an open, distributed SDN OS. In *Proceedings of the third workshop on Hot topics in software defined networking* (2014), ACM.

[2] Botelho, F., Bessani, A., Ramos, F., and Ferreira, P. On the design of practical fault-tolerant SDN controllers. In *Software Defined Networks (EWSDN), 2014 Third European Workshop on* (2014), IEEE, pp. 73–78.

[3] Fonseca, P., Bennesby, R., Mota, E., and Passito, A. Resilience of SDNs based On active and passive replication mechanisms. In *Global Communications Conference (GLOBECOM), 2013 IEEE* (2013), IEEE, pp. 2188–2193.

[4] Gonzalez, A. J., and Helvik, B. E. System management to comply with SLA availability guarantees in cloud computing. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on* (Dec 2012), pp. 325–332.

[5] Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *USENIX Annual Technical Conference* (2010), vol. 8.

[6] Jarschel, M., Oechsner, S., Schlosser, D., Pries, R., Goll, S., and Tran-Gia, P. Modeling and performance evaluation of an openflow architecture. In *Teletraffic Congress (ITC), 2011 23rd International*.

[7] Katta, N., Zhang, H., Freedman, M., and Rexford, J. Ravana: Controller fault-tolerance in software-defined networking. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research* (2015), ACM.

[8] Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., et al. ONIX: A Distributed Control Platform for Large-scale Production Networks. In *OSDI* (2010), vol. 10.

[9] Mahmood, K., Chilwan, A., sterb, O., and Jarschel, M. Modelling of openflow-based software-defined networks: the multiple node case. *Networks, IET 4*, 5 (2015), 278–284.

[10] Pashkov, V., Shalimov, A., and Smeliansky, R. Controller failover for SDN enterprise networks. In *Science and Technology Conference (Modern Networking Technologies)(MoNeTeC), 2014 International* (2014), IEEE.

[11] Reitblatt, M., Canini, M., Guha, A., and Foster, N. FatTire: declarative fault tolerance for software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in Software Defined Networking* (2013), ACM, pp. 109–114.

[12] Sharma, S., Staessens, D., Colle, D., Pickavet, M., and Demeester, P. Openflow: Meeting carrier-grade recovery requirements. *Computer Communications 36*, 6 (2013), 656–665.

[13] Tootoonchian, A., and Ganjali, Y. HyperFlow: A distributed control plane for OpenFlow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking* (2010).