Adaptive State Consistency for Distributed ONOS Controllers

Fetia Bannour, Sami Souihi and Abdelhamid Mellouk
LISSI/TincNetwork Research Team
University of Paris-Est Créteil (UPEC), France
fetia.bannour@gmail.com, (sami.souihi, mellouk)@u-pec.fr

***Abstract*—Logically-centralized but physically-distributed SDN controllers are mainly used in large-scale SDN networks for scalability, performance and reliability reasons. These controllers host various applications that have different requirements in terms of performance, availability and consistency. Current SDN controller platform designs employ conventional strong consistency models so that the SDN applications running on top of the distributed controllers can benefit from strong consistency guarantees for network state updates. However, in large-scale deployments, ensuring strong consistency is usually achieved at the cost of generating performance overheads and limiting system availability. That makes weaker optimistic consistency models such as the eventual consistency model more attractive for SDN controller platform applications with high-availability and scalability requirements. In this paper, we argue that the use of the standard eventual consistency models, though a necessity for efficient scalability in modern SDN systems, provides no bounds on the state inconsistencies tolerated by the SDN applications. To remedy that, we propose an adaptive consistency model for the distributed ONOS controllers following the notion of continuous and compulsory (per-controller) eventual consistency, where network application states adapt their eventual consistency level dynamically at run-time based on the observed state inconsistencies under changing network conditions. When compared to the ONOS approach to static eventual consistency, our approach proved efficient in minimizing state synchronization overheads while taking into account application state consistency SLAs and without compromising the application requirements of high-availability, in the context of large-scale SDN networks.**

***Index Terms*—Software-Defined Networking (SDN), logically-centralized control, physically-distributed control, scalability, reliability, adaptive consistency, eventual consistency, consistency-based Service-Level Agreement (SLA), ONOS controller,**

I. Introduction

The emergence of Software-Defined Networking (SDN) has aroused great interest in rethinking traditional approaches to network architecture, design and management. By decoupling the control plane logic from the data forwarding hardware, SDN aims for a centralized control (governed by a global network view) and for network programmability, thereby enabling network innovation, simplifying network operations, and easing service deployment. Another core benefit of the SDN paradigm is that it drives network abstraction and automation along with the incorporation of adaptive control aspects in network configuration and management, thus promising to bring agility and to address the exponential growth in network infrastructure and services.

Beyond the hype, there have been serious concerns about the widespread adoption of SDN. For instance, the physical centralization of the control plane in a single programmable software-based controller, shows important limitations in terms of scalability, availability and reliability, especially in the context of large-scale real-world network deployments [1]. As a matter of fact, recent research in SDN perceives the SDN control plane as a distributed system [5], where multiple physically-distributed SDN controllers are responsible for handling multiple network domains, while synchronizing their local state information and coordinating to maintain a logically-centralized network view.

However, when compared to physically centralized SDN designs, distributed SDN controller platforms face an additional set of challenges related to inter-controller communications for achieving network state consistency. In fact, guaranteeing an appropriate level of controller state consistency while sustaining good performance in a fault-tolerant SDN network is a challenging task as stated by the CAP theorem [2]. In this respect, different state distribution trade-offs, mainly between consistency and performance properties, are involved when designing a distributed SDN controller platform.

Current implementations of distributed SDN controller platforms [3, 4] use strong and/or eventual consistency models for their internal and external SDN control applications. These models have different advantages and drawbacks.

The *Strong Consistency* control model ensures a strongly-consistent network state view to ensure the correct behavior of the applications running on top of the distributed controllers. However, it introduces controller-to-controller synchronization overhead, and hinders response-time and throughput, thereby impacting availability and performance as the network scales.

On the other hand, the *Eventual Consistency* control model allows for higher availability, especially in the context of large-scale distributed SDNs supporting scalable applications. However, it may present a temporarily inconsistent network view that leads to incorrect application behaviors, but that becomes *eventually* consistent throughout the network.

In this paper, we introduce the use of an adaptive multi-level consistency model following the continuous consistency concept [5] in the context of distributed SDN controllers. That model presents many advantages when compared to the strong consistency and optimistic (eventual) consistency extremes, especially in large-scale deployments. Our consistency adapta-

tion strategy was implemented for the distributed open-source ONOS controllers; It mainly consists in continuously tuning the application's consistency level (at run-time) based on the observed state inconsistencies with respect to the application requirements (SLAs) under changing network conditions.

The rest of this paper is organized as follows: In Section II, we provide an overview of the consistency models used by state-of-the-art SDN controller platforms. In Section III, we review the consistency problem in SDN and investigate the involved consistency trade-offs. In Section IV, we discuss the consistency models implemented in the ONOS controller platform. In Section V, we present our consistency adaptation approach for the distributed ONOS controllers. Finally, Section V shows and discusses the experimentation results.

II. Related Work

The challenges related to consistency in distributed SDN control have been recently addressed in the SDN literature. Some works focused on the impact of switch-to-controller state consistency *between switches and controllers* on network application performance. Reitblatt et *al.* [6] studied the consistency of controller-driven flow updates in terms of network policy conservation. They proposed a new type of consistency abstractions to enforce a consistent forwarding state at different levels (per-flow consistency and per-packet consistency). Recent approaches [7] focused on efficiently updating the network data plane state while preventing forwarding anomalies at the switches and maintaining desired consistency properties (e.g. loop and black-hole freedom).

Another category of works, falling within the scope of this paper, focused on achieving controller-to-controller state consistency *between the distributed controllers* without compromising application performance.

Current implementations of distributed SDN controller platforms offer different state consistency abstractions. They use static mono-level [4] or multi-level [3] consistency models such as the strong, eventual and weak state consistency levels.

Onix [8] offers two separate dissemination mechanisms for synchronizing network state updates between the NIBs stored at the controller instances. These mechanisms are based on two implemented data-store options: A replicated transactional database designed for ensuring strong consistency at the cost of good performance for persistent but slowly-changing states, and a high-performance memory-only distributed hash table (DHT) for volatile states that are tolerant to inconsistency.

Similarly, ONOS [3] provides different state sharing mechanisms to achieve a consistent network state across the cluster of ONOS controllers. More specifically, ONOS's distributed core eases the state management and coordination tasks for application developers by providing them with an available set of core building blocks for dealing with different types of distributed control plane states, including a consistent primitive for state requiring strong consistency and an eventually consistent primitive for state tolerating relaxed consistency.

On the other hand, ODL [4] supports a strong consistency model in its distributed datastore architecture. In fact, all the data shared across the cluster of controllers for maintaining the logically centralized network view is handled in a strongly-consistent manner using the RAFT consensus algorithm [9].

Recent approaches to handling the issues of controller state consistency [10, 11] recommended the use of adaptive consistency for the distributed SDN controller platforms. Aslan et *al.* [10] attempted to mitigate the impact of controller state distribution on SDN application performance by proposing an adaptive tunable consistency model following the delta consistency model. In their model, the automatic control plane adaptation module tunes the consistency level (the synchronization period parameter) based on an application-specific performance indicator that is measured given the current state of the network. To assess their approach, the authors compared the performance of the distributed load-balancing application running on top of adaptive and non-adaptive controllers.

In the same spirit, the work described in [11] put forward an adaptive consistency model for distributed SDN controllers following the Eventual Consistency level. The main aim of changing the controller consistency level on-the-fly was to maintain a scalable system that sacrifices application optimality for less synchronization overhead. Accordingly, the authors propose a cost-based approach that bounds the correctness to a tunable threshold, where the consistency level is adapted based on the effort of state convergence after the expiration of a non-synchronization period, and the application inefficiencies due to operations with stale state. The performance of the proposed model was evaluated based on a specific routing application.

III. The Consistency Problem in SDN

A. Consistency trade-offs in SDN

In distributed SDN architectures, the SDN control plane supports the interaction between multiple controllers through their "east-west" interfaces. Inter-controller communications are indeed needed to synchronize the controllers' shared data structures to maintain a consistent global network view, and therefore ensure the correct behavior of the network applications running on top of the distributed controllers. Such control traffic can be in-band or out-band.

However, distributing the network control state across the SDN control plane affects the performance objectives of the control applications. In fact, many state distribution trade-offs arise as discussed by Levin et *al.* such as the trade-off between application state consistency/staleness (state synchronization overhead) and application performance (objective optimality), and the trade-off between application logic complexity and robustness to inconsistency.

More generally, Brewer's CAP theorem applied to networks [2] investigates the involved trade-offs between Consistency (C), Availability (A), and Partition-tolerance (P). It states that, in SDN networks, it is generally only possible to achieve two out of the three desirable properties: CA, CP or AP.

However, in the context of modern and scalable distributed database systems (DDBS), Abadi's PACELC theorem [12] is believed to be more relevant, as it combines in a single complete formulation, the CAP theorem trade-offs, and in the

absence of partitions (E), the Latency (L)/Consistency (C) trade-off. Many popular modern DDBSs do not by default guarantee strong consistency, as stated by CAP. Conversely, they come with trade-offs that are better represented by the PACELC alternative. For example, Amazon's Dynamo [13], Facebook's Cassandra [14], and Riak are PA/EL systems, MongoDB is PA/EC, yahoo's PNUTS is PC/EL, and finally BigTable and HBase are PC/EC systems.

In this context, we argue that SDN is bringing the network design much closer to the design of distributed database systems. In the same spirit, we argue that PACELC can apply to large-scale SDN controller platforms, in the same way it applies to modern and scalable NoSQL DDBSs.

B. Consistency Models in SDN

Many architectures have been proposed to support distributed SDN controllers, with the goal of improving the scalability, reliability and performance of SDNs. Two main consistency models are used by current controller platforms:

1) The Strong Consistency Model: In SDN, the strong consistency model guarantees that all controller replicas in the cluster have access to the most updated network information at all times. That comes at the cost of increased state synchronization delay and communication overhead, especially in large-scale deployments. In fact, strong consistency relies on a blocking synchronization process that keeps the switches from reading the data, unless the controllers are fully updated, thereby affecting network availability and scalability.

Strong consistency is a requirement for certain applications that favor consistency and correctness properties over availability. In current controller platforms, strong consistency is usually achieved using Paxos, RAFT [9] and similar protocols.

2) The Eventual Consistency Model: In SDN, the eventual consistency model takes a relaxed approach to consistency by assuming that all controller replicas will "eventually" converge and become consistent throughout the network. That means that controllers may temporarily present an inconsistent network view, allowing for some stale data to be read, and potentially causing a transient incorrect application behavior.

Many applications opt for eventual consistency to guarantee high-availability and performance at scale. Modern DDBSs, including Dynamo [13] and Cassandra [14], support eventual consistency settings by default in exchange for extremely high availability (fast data access) and scalability.

3) Adaptive Consistency Models: Recent research in SDN [10, 11] has introduced the concept of adaptive consistency in the context of distributed SDN control. Unlike static consistency approaches, adaptively-consistent controllers adjust their consistency level at run-time to reach the desired application performance and consistency requirements. That alternative offers many benefits: It spares application designers the task of developing complex applications that require implementing multiple consistency models, it provides applications with robustness against sudden network conditions, and it reduces the overhead of state distribution across controllers without compromising application performance [10].

In the community of modern database systems, the need for adaptable consistency where the consistency level is decided dynamically over-time based on various factors has been recognized. Many adaptive consistency models have been proposed, such as the QUOROM-based consistency [15], RedBlue Consistency [16], Chameleon and Harmony [17], the delta consistency [10], and the continuous consistency [5]. In addition, most of modern database systems such as Cassandra [14] and Dynamo [13] are currently equipped with an adaptive consistency feature, offering multiple consistency options with tunable parameters for application developers.

In our opinion, all the above consistency models could be leveraged by the SDN community to build adaptively-consistent SDN controllers.

IV. Consistency Models in ONOS

In this paper, we are particularly interested in the open-source Java-based ONOS controller [3]. In this section, we describe in detail the ONOS approach to state consistency in a distributed controller setting.

To achieve high-availability, scale-out and performance, the ONOS controller platform supports a physically-distributed cluster-based control plane architecture, where each controller is responsible for handling the state of a subsection of the network. To maintain the logically-centralized network view, local controller state information is disseminated across the cluster in the form of events that are shared via ONOS's distributed core. The latter consists of core subsystems tracking different types of network states being stored in distributed data structures and requiring different coordination strategies. Two main state consistency schemes are implemented in ONOS's subsystem stores to provide two different levels of state consistency: strong consistency and eventual consistency.

A. Strong Consistency in ONOS

To ensure strong consistency among replicated network states, ONOS uses (since version 1.4) the Atomix framework that is based on the RAFT consensus protocol [9]. For instance, the store for switch-to-controller mastership (mapping) management is handled in a strongly consistent manner using that framework. Besides, ONOS exposes a set of core distributed state management primitives that can be leveraged by application developers to implement their application-specific stores. In this respect, applications whose state is maintained in a strongly consistent fashion can leverage the `ConsistentMap` distributed primitive, which guarantees strong consistency for a distributed key-value store.

B. Eventual Consistency in ONOS

For eventually-consistent behaviors, ONOS employs an optimistic replication technique complemented by a background gossip/anti-entropy protocol. For instance, the stores for devices, links, and hosts are managed in an eventually-consistent manner. The distributed topology store is also eventually consistent since it relies on the distributed versions of the device, link and host stores. For the eventual consistency option,

ONOS offers the `EventuallyConsistentMap` distributed primitive for control programs and applications. The latter can create different instances of these primitives for managing their eventually-consistent application-specific states.

(a) *Optimistic Replication:*

Optimistic replication is a key technology that is used in large-scale distributed data sharing systems, meeting the goal of achieving higher availability and scalability as compared to strongly-consistent systems. That strategy for replication propagates changes in the background, and discovers conflicts after they occur. It is based on the ”optimistic” assumption that inconsistencies rarely occur and that replicas will converge after some time, thus providing eventual consistency guarantees.

In ONOS, optimistic replication is used in the distributed maps. Whenever an update occurs in the store managed by one controller, the associated `EventuallyConsistentMap` replicates events immediately to the rest of the controllers. That means that maps on each controller will get closely in sync (apart from a small propagation delay) in case the controllers are functioning properly. On each controller, updates are added to an `EventAccumulator` as they are written.

(b) *Gossip-based Anti-Entropy:*

Controllers that purely rely on optimistic replication might progressively get out of sync, especially in the event of node failures and partitions or in the case updates get missed or dropped. The anti-entropy protocol takes care of ensuring that replicas are back in sync by resolving discrepancies, and that the entire cluster converges fairly quickly to the same state.

In ONOS, the gossip-based anti-entropy mechanism is a lightweight peer-to-peer background process that runs periodically: At fixed intervals (3-5 seconds), each controller randomly chooses another controller, they both exchange information in order to compare the actual content (entries) of their distributed stores (based on timestamps). After synchronizing their respective topology views, the controllers become mutually consistent. This reconciliation approach proves useful in fixing controllers when their state drifts slightly, and also in quickly synchronizing a newly-joining controller with the rest of the controllers in the cluster.

V. The Proposed Adaptive Consistency for ONOS

In this section, we explain our approach which is mainly aimed at optimizing the consistency management in ONOS.

A. A Continuous Consistency model for ONOS

As explained in III-B3, an adaptive consistency model offers many benefits for the distributed SDN controllers. In particular, the ONOS controllers can benefit from the continuous consistency model proposed in [5]. The latter is based on a middle-ware framework (called TACT) for adaptively tuning the consistency and availability requirements for replicated online services, following the continuous consistency concept. In contrast to the strong consistency model (which imposes performance overheads and limits availability), and to optimistic consistency models (which provide no bounds on system inconsistencies), the continuous consistency model explores the semantic space between these two types of traditional models: It offers a continuum of intermediate consistency models (*multi-level consistency*) with tunable parameters. These quantifiable degrees of consistency can be exploited by applications to explore, at runtime, their own trade-offs between consistency and availability, while taking into account the changing network and service conditions. More specifically, TACT bounds the amount of inconsistency and divergence among the replicas in an application-specific manner. Basically, applications specify their consistency semantics through *conits*; a set of metrics that capture the consistency spectrum: *Numerical Error*, *Order Error*, *Staleness*.

Hence, for each conit, consistency is quantified continuously along a three-dimensional vector:

$$Consistency = (NumericalError, OrderError, Staleness) \tag{1}$$

Numerical Order bounds the discrepancy between the value delivered to the application client and the most consistent ”final” value. *Order Error* bounds inconsistency by the number of tentative/unseen writes at any replica. *Staleness* places a real-time bound on the delay for propagating the writes among the replicas.

In our opinion, many features from the discussed continuous consistency spectrum can indeed be incorporated when rethinking the ONOS strategy to state consistency, especially in the context of large-scale deployments.

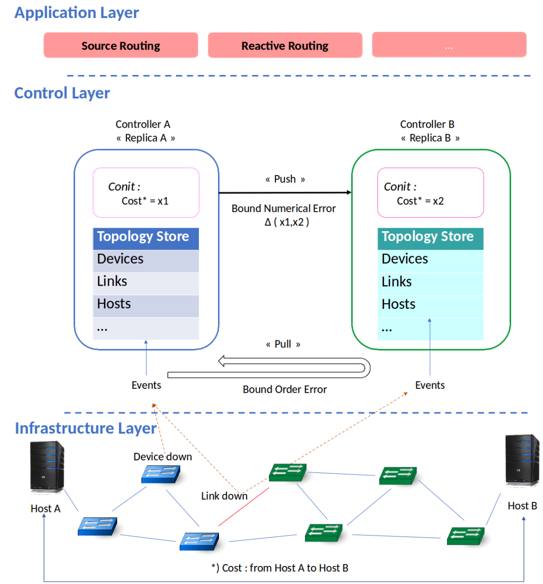


Fig. 1: The proposed Adaptive Consistency Strategy

B. Our Consistency Adaptation Strategy for ONOS

We propose to maintain the strong consistency model implemented in ONOS for applications requiring strict consistency guarantees, but we suggest turning the eventual consistency model into an adaptive tunable model following the concept

of continuous consistency in order to explore the availability, consistency and scalability benefits of such a model.

With this in mind, we adopt the following strategy when reviewing the eventual consistency model:

- We keep the current implementation of the optimistic replication technique used for replicating events and updates among controllers in the cluster.

- We bring significant changes to the anti-entropy protocol used for eventual consistency in ONOS: Instead of running the anti-entropy process for each controller replica periodically at fixed intervals (each 3-5 seconds) (*voluntary anti-entropy*) causing too much overhead and affecting system scalability and performance, we argue that the anti-entropy process should only be scheduled when the system consistency is at risk (*compulsory anti-entropy*). In other words, the choice of the anti-entropy reconciliation period for each controller replica (*per-replica consistency*) should be based on the correctness of the system with respect to the consistency requirements expressed by the applications. That way, at each controller replica and for each application state, the consistency level is dynamically adapted based on the computed values of the consistency metrics (see Equation (1)) capturing the application's consistency semantics with respect to the given thresholds set in advance by the application.

C. Our Implementation Approach

To implement our state consistency adaptation approach, we consider a replicated source routing SDN application running on top of a cluster of distributed ONOS controllers. The cost-based source routing application operates on a distributed topology graph for computing the shortest-path costs between source and destination hosts. Since the topology graph state is handled in an eventually-consistent manner, the application's state is also considered as eventually-consistent.

In our routing application f, the path between the source host A and the destination host B (see Figure 1) may be defined as a *conit*. Besides, we argue that an important consistency requirement for our control application is the result optimality of the instant path computation cost (in terms of hop-count in our case) which is captured by the *Numerical Error* metric. The numerical error of our conit C can be defined as relative difference between the value of the "shortest-path" cost x_{local} as perceived by a local replica, and its "final "optimal" value $x_{optimal}$ at a replica that has reached some "final" consistent state. That error is continuously bounded at run-time using an application-defined threshold $T(f)$ (in percentage) as follows:

$$NumericalError(C_f) = (\frac{|x_{local} - x_{optimal}|}{x_{optimal}}) < T(f) \qquad (2)$$

Finally, it is worth noting that other consistency semantics for the source routing application might be expressed using *Staleness* and *Order Error*.

To implement our approach, we introduce some modifications to the ONOS Java source code. We start by developing our adaptive source routing application (similar to the `Intent Forwarding Application`) for computing the shortest-path cost between host A and host B. Running on each ONOS instance, our application gets information from the in-memory topology cache maintained by each ONOS instance (`DistributedTopologyStore`). Whenever an update occurs in the topology graph (e.g. links/devices failing or joining), our application detects that topology change and updates the "local" shortest-path cost between host A and B accordingly. We also modify the implementation of the `EventuallyConsistentMap` distributed primitive, especially for the eventually-consistent stores that feed the topology store (e.g. link and device stores). We indeed focus on the `BackgroundExecutor` service of the eventually-consistent maps, which runs the background anti-entropy tasks, and we propose a new implementation of the `Runnable` interface used for executing the scheduled anti-entropy thread.

In fact, instead of sending the anti-entropy advertisement messages periodically each 3-5 seconds between the controllers, as it is the case in ONOS, we propose to run, at each replica, a periodic check on the consistency of other replicas with respect to the application's shortest-path cost state, by computing at run-time the relative *Numerical Error* defined in Equation (2). In fact, in the event of a controller failure, the rest of the controllers in the cluster that detect that failure, keep a screen-shot of their own topology graph at the moment of the failure. During the periodic consistency check, they use that stored topology graph to estimate the inconsistency of the failed controller; which is equal to the relative difference between the "local" shortest-path cost (computed based on the current topology graph state) and the shortest-path cost as perceived by the controller after recovery (computed based on the *stale* topology graph state being stored).

When the failed controller recovers, the rest of the controllers make an anti-entropy decision based on the checked numerical error. If the error exceeds an "alarming" consistency threshold set in-advance by the application, then, an anti-entropy process is launched to fix the failed controller state. That is achieved by synchronizing the controllers' eventually-consistent stores that feed the topology view. In the opposite case, the inconsistency is considered as tolerated by the application, and an anti-entropy session might be scheduled afterwards in case the controller state significantly drifts away.

VI. Performance Evaluation

A. Experimental Setup

Our experiments are performed on an Ubuntu 16.04 LTS server using ONOS 1.13. We also use Mininet 2.2.1 and an ONOS-provided script (*onos.py*) to start an emulated ONOS network in a single machine; including a logically-centralized ONOS cluster, a modeled control network and a data network. Wireshark is used as a sniffer to capture the inter-controller traffic which uses TCP port 9876.

To validate our proposed approach on ONOS, which we will refer to as ONOS-WAC (ONOS-With Adaptive Consistency), we have considered many test scenarios. In each scenario, we

run a cluster of N ONOS controller instances, controlling a Mininet network topology of S switches (see Table I).

Test scenarios	N Controllers	S Switches	F Controller Failure scenarios
n°1	3	16	2
n°2	5	36	3
n°3	7	64	4
n°4	9	100	4
n°5	10	121	5

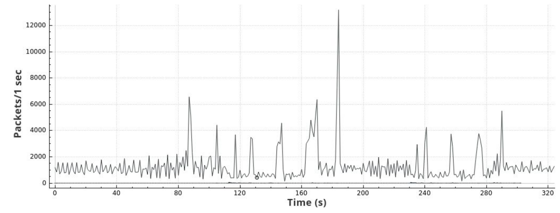
TABLE I: Test Scenarios

In order to create state inconsistencies among the controller instances in the cluster with respect to the shortest-path cost state of our source routing application, we create different controller failure scenarios F. Shortly after a controller failure (in F) in a specific scenario S_i, we consider changing the network topology by taking down network switches and links along the shortest-path (computed by the application) between source host A and destination host B. That way, after recovery, the controller will have an inconsistent network topology view as compared to the rest of the controllers in the cluster. According to our proposed approach, that inconsistency in the network topology view affects the optimality of the shortest-path cost computation performed by the source routing application instance running on top of the recovered controller. The induced *Numerical Error* is likely to trigger a synchronization process achieved by anti-entropy tasks (see Section V-C).

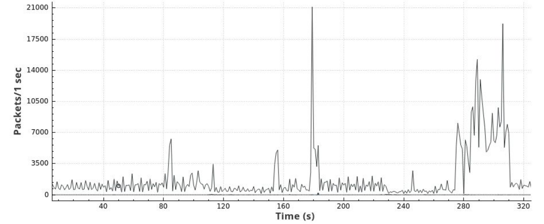
B. Results

In Scenario n°1 (Scenario with three controllers as shown in Table I), we adopt the methodology described in VI-A. In Figure 2, we show the inter-controller traffic captured during the test scenario period in an ONOS cluster (Figure 2(a)) and in an ONOS-WAC cluster (Figure 2(b)). After running the Mininet topology according to Scenario n°1, the same event sequence is performed for both ONOS and ONOS-WAC clusters. For instance, the first traffic peak in both figures (at $t = 90s$) corresponds to a "pingall" Mininet CLI command executed for topology discovery. At $t = 150s$, we simulate a failure scenario by taking down one controller instance. That action is followed by other topology changes (e.g. links down) corresponding to the subsequent peaks in both figures. At $t = 180s$, we bring back the failed controller, resulting in a traffic peak that appears to be more significant in the case of ONOS-WAC. That increase in traffic is due to the anti-entropy process which has been triggered by an inconsistency (*Numerical Error*) value that exceeded the application threshold. Conversely, in the ONOS network, the anti-entropy traffic is generated periodically over the test period regardless of the observed inconsistencies. Likewise, at $t = 280s$, we repeat the same scenario following the same event sequence, but considering the failure of a different controller.

In the test scenario described above, the application inconsistency threshold was set to 0%, triggering the start of anti-entropy sessions for any observed inconsistencies in the considered application state. In the following test experiments, we repeat the same scenario (Scenario n°1), but we consider varying the source routing application's inconsistency



(a) ONOS



(b) ONOS-WAC

Fig. 2: Scenario n°1: Captured Inter-controller traffic (in packets per second) during the test scenario period (using Wireshark)

threshold. As shown in Figure 3, ONOS shows an average inter-controller overhead equal to $315kbps$ regardless of the application inconsistency threshold. On the other hand, ONOS-WAC shows relatively less inter-controller overhead (due to low anti-entropy overhead). The latter is impacted by the application's consistency requirements. For example, in the case of strict consistency requirements (application threshold between 0% and 30%), inconsistencies occurring in the application state are more likely to trigger the anti-entropy reconciliation sessions causing much more overhead, when compared to the case of less strict consistency requirements (application threshold between 40% and 50%).

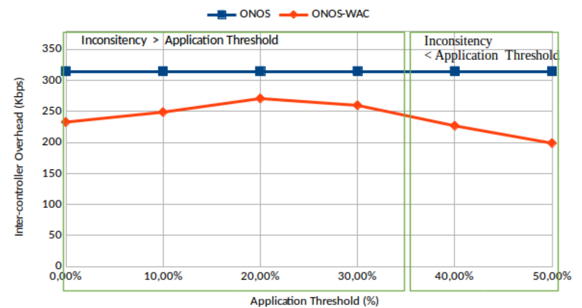


Fig. 3: Scenario n°1: Inter-controller Overhead in ONOS and ONOS-WAC according to the Application Threshold

In order to assess the gain in anti-entropy overhead of our adaptive consistency model implemented on ONOS, as compared to the eventual consistency model of ONOS, we consider estimating the rate ($R(S_i)$) of increase in anti-entropy overhead of ONOS with respect to ONOS-WAC (see Equation (3)) as a function of the number of controllers in the cluster (following Scenarios n°1, n°2, n°3, n°4, n°5) (see Figure 4).

$$R_i(S_i) = 1 - [\frac{A(S_i) - B(S_i)}{C(S_i) - B(S_i)}] \quad (3)$$

-$A(S_i)$: the inter-controller overhead generated by ONOS-WAC *after* the event sequence (VI-B) following Scenario S_i.
-$B(S_i)$: the inter-controller overhead generated by ONOS-WAC *before* the event sequence (the overall inter-controller traffic without the anti-entropy traffic).
-$C(S_i)$: the inter-controller overhead generated by ONOS *after* the event sequence (VI-B) following Scenario S_i.

As shown in Figure 4, the gain in anti-entropy overhead, when adopting ONOS-WAC, grows almost linearly with the number of controllers in the cluster. For example, in Scenario n°5 (corresponding to 10 controllers in the network cluster), the gain in anti-entropy overhead has reached 25%.

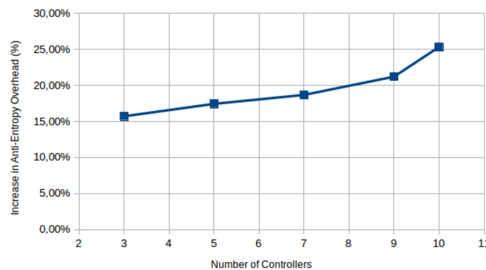


Fig. 4: Gain in Anti-Entropy Overhead of ONOS-WAC with respect to ONOS according to the number of Controllers in the cluster

VII. Conclusion

In this paper, we investigated the use of adaptive consistency for the distributed ONOS controllers. Our approach was aimed at turning the eventual consistency model of ONOS into an adaptable multi-level consistency model following the concept of continuous consistency. The latter delivers the performance and availability benefits of an eventual consistency model, but has the additional advantage of controlling the state inconsistencies in an application-specific manner. Our consistency adaptation strategy was implemented for a source routing application on top of ONOS. Besides ensuring the application's state consistency requirements (specified in the given SLA), our results showed a substantial reduction in the Anti-Entropy reconciliation overhead, especially in the context of large-scale networks. As a future work, we consider extending our adaptive consistency approach to the optimistic replication technique used in ONOS's eventual consistency model by leveraging multiple replication degrees as well as the geo-placement of the controller replicas.

Although the main focus of this work was placed at dynamically adjusting the consistency level of application states (which use controller states), we plan to extend our approach to the controller states (internal controller applications). Indeed, the long-term goal of this work is to design adaptively-consistent controllers that adjust both control and application plane consistency levels under changing network conditions.

References

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

[2] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker, "Cap for networks," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, 2013, pp. 91–96.

[3] ONOS. [Online]. Available: https://onosproject.org/

[4] ODL. [Online]. Available: http://opendaylight.org/

[5] H. Yu and A. Vahdat, "Design and evaluation of a continuous consistency model for replicated services," in *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4*, ser. OSDI'00, Berkeley, CA, USA, 2000.

[6] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, "Consistent updates for software-defined networks: Change you can believe in!" in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, 2011, pp. 7:1–7:6.

[7] T. D. Nguyen, M. Chiesa, and M. Canini, "Decentralized consistent updates in sdn," in *Proceedings of the Symposium on SDN Research*, 2017, pp. 21–33.

[8] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Z. Google, R. Ramanathan, Y. I. NEC, H. I. NEC, T. H. NEC, and S. Shenker, "Onix: a distributed control platform for large-scale production networks," in *9th Conference on Operating Systems Design and Implementation*, 2010, pp. 351–364.

[9] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 2014.

[10] M. Aslan and A. Matrawy, "Adaptive consistency for distributed sdn controllers," in *2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks)*, Sept 2016, pp. 150–157.

[11] E. Sakic, F. Sardis, J. W. Guck, and W. Kellerer, "Towards adaptive state consistency in distributed sdn control plane," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–7.

[12] D. Abadi, "Consistency tradeoffs in modern distributed database system design: CAP is only part of the story," *Computer*, vol. 45, no. 2, pp. 37–42, Feb 2012.

[13] S. Sivasubramanian, "Amazon dynamodb: A seamlessly scalable non-relational database service," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: ACM, 2012, pp. 729–730.

[14] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

[15] S. P. Kumar, "Adaptive Consistency Protocols for Replicated Data in Modern Storage Systems with a High Degree of Elasticity," Theses, CNAM, Mar. 2016.

[16] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues, "Making geo-replicated systems fast as possible, consistent when necessary," in *Conference on Operating Systems Design and Implementation*, 2012.

[17] H.-E. Chihoub, M. Pérez, G. Antoniu, and L. Bougé, "Chameleon:customized application-specific consistency by means of behavior modeling," Research Report, 2013.

View publication stats