Towards Adaptive State Consistency in Distributed SDN Control Plane

Ermin Sakic*, Fragkiskos Sardis**, Jochen W. Guck*** and Wolfgang Kellerer***
* {Technical University of Munich, Siemens AG}, Munich, Germany, ermin.sakic@{tum.de, siemens.com}
** Centre for Telecommunications Research, King's College London, London, UK, fragkiskos.sardis@kcl.ac.uk
*** Technical University of Munich, Munich, Germany, {guck, wolfgang.kellerer}@tum.de

***Abstract*—State synchronisation in clustered Software Defined Networking controller deployments ensures that all instances of the controller have the same state information in order to provide redundancy. Current implementations of controllers use a strong consistency model, where configuration changes must be synchronised across a number of instances before they are applied on the network infrastructure. For large deployments, this blocking process increases the delay of state synchronisation across cluster members and consequently has a detrimental effect on network operations that require rapid response, such as fast failover and Quality of Service applications. In this paper, we introduce an adaptive consistency model for SDN Controllers that employs concepts of eventual consistency models along with a novel 'cost-based' approach where strict synchronisation is employed for critical operations that affect a large portion of the network resources while less critical changes are periodically propagated across cluster nodes. We use simulation to evaluate our model and demonstrate the potential gains in performance.**

Keywords - SDN, distributed control plane, scalability, QoS, adaptive consistency, RAFT, OpenDaylight, ONOS

I. Introduction

In large networks, the scalability and resilience of the SDN controller becomes important as it poses a single point of failure for the entire network. A single node hosting the controller may not scale well for hundreds of switches and thousands of concurrent OpenFlow events. Similarly, a single node cannot provide high reliability when it comes to hardware failures. To address this problem, SDN controllers are clustered over multiple nodes, where each instance of the controller is responsible for a number of switches while also providing redundant copies of the other instances' state. When a node fails, another controller instance takes over the failed node's tasks and resumes operation with minimal downtime.

In distributed computing, clustering refers to the loose or tight coupling of nodes for purpose of reliability and load balancing. Such systems can be scaled horizontally by adding nodes to the cluster, however, as more nodes are added, the overhead in state synchronisation between nodes increases. There exist two main models for synchronising state across a cluster. The *strong consistency model* [1] requires that the distributed state across cluster members is replicated and, following any single state-update at state leader, propagated using mutual consensus to replicas. In contrast, the *eventual consistency model* [2] omits the consensus procedure and guarantees that the *at least one delivery* invariant holds. However, the advantage of non-blocking operations comes at the expense of sacrificing the total ordering of state updates and sometimes the system correctness. In eventually consistent systems, the convergence to a single state is determined by two factors: *anti-entropy* and *reconciliation*. Anti-entropy ensures that data is synchronised in a timely manner and that the system will not enter a state of complete de-synchronisation between instances [3]. Reconciliation refers to the mechanisms that determine the final system state by resolving conflicting updates from different instances. Typically such conflicts are resolved by the *last-writer-wins* approach, where the most recent change of state is considered final [4].

In this paper, we introduce the concept of runtime adaptation of consistency levels in state synchronisation for a distributed SDN control plane (DCP). A consistency level is assigned to every resource state accessed by an SDN application (e.g. routing, topology manager). The consistency level is adapted based on the experienced effort of state convergence after a non-synchronisation period has expired; and the inefficiencies resulting from operations with stale state as inputs. We define the application inefficiency as a qualitative distance between the optimal and the computed result. The proposed method enables the design of scalable SDN DCPs, since the majority of state updates are executed as local non-blocking, eventually consistent operations. The methodology of changing the level of controller consistency on-the-fly allows for maintaining a scalable system by sacrificing some controllable amount of result optimality - and thus the blocking overhead of cluster-wide synchronisation.

The rest of the paper is structured as follows: Section II describes the problem of state synchronisation in distributed SDN control plane, Section III investigates the state of the art related to state synchronisation in distributed systems and focuses on existing SDN DCP implementations, Section IV presents the proposed solution for adaptive state consistency, Section V presents and discusses the performance of the proposed method using simulation. Finally, Section VI concludes this paper.

II. Problem Definition

In distributed deployments, individual controller instances hold the application state necessary to fulfil the requirements of controller applications (e.g. path finding, policy handler). Assuming a partitioned DCP design, we distinguish between *global* and *local* controller decisions. Global decisions necessitate a response to events where an action modifies the configuration of a switch that is outside the controller's administrative domain. Interaction between controller instances in the DCP is necessary, resulting in additional latency overhead. In state of

978-1-4673-8999-0/17/$31.00 ©2017 IEEE

the art scalable SDN DCPs [5], [6], a network is typically partitioned into multiple administrative sub-domains.To minimize controller-to-controller (C2C) synchronisation efforts in the DCP, our model supports transformation of global controller decisions into local ones, by means of assigning all controllers as masters of all switches and granular per-controller planning of switch notification subscriptions for scalability. A global route configuration that necessitates message passing across the whole DCP in current controllers [5], [6], can be applied by a single controller to all switches on path in our design, since the administrative domain of the controller may stretch across the whole network. The related OpenFlow-based controller role configuration is introduced in Subsection IV-A.

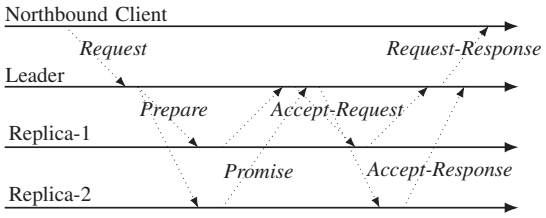

Fig. 1. Paxos workflow where leader requires confirmation only from cluster majority to progress the state. Notice how Replica-2 delays its response.

Strong consistency systems always implement some consensus algorithm to enable conflict-free distribution of state updates. Paxos [7], a popular decentralized consensus algorithm, proposes a four-delay state update method encompassing *Prepare-Request*, *Promise*, *Accept-Request* and *Accept-Response* delays, as depicted in Figure 1. In large-scale SDN deployments, the amount of incoming controller requests can reach up to 11 million requests per second [8]. In the worst case, every state request may necessitate a state change and hence a consensus run, thus preventing fast network reconfigurations and introducing a bottleneck on control and data planes. Although optimizations of Paxos, including the recent RAFT [9] consensus algorithm, were proposed in literature, the main concept of the algorithm remains unchanged.

Recently published industrial SDN use-cases introduce new requirements on global QoS-aware route establishment across WANs [10] and locally administered networks [11]. [10] introduces the requirements of critical infrastructure operators for on-demand service establishment in a software-defined WAN, for purpose of interconnection of a large number of Internet-of-Things (IoT) devices over the service provider's infrastructure. The described IoT use-case assumes QoS guarantees for individual IoT-device-to-Cloud application flows. Coupled with the ever-growing number of IoT devices and need for dynamic resource (de-)allocation for globally computed QoS-enabling paths, current SDN controller solutions are not able to provide the necessary degree of scalability in global configuration. Some 5G use-cases [12] introduce the requirements of connection setup times of $<$15-30ms for low-latency services in converged backbone for arbitrary numbers of end-hosts. Initial performance measurements of an SDN- and OpenDaylight-enabled DCP in the test network of Telecom Italia [13] show that the requirement of low setup time cannot be met in most scenarios. The identified main cause of delay in end-to-end path establishment lies in the C2C interactions, required in order to reach consensus for distributed path establishment. Our approach minimizes this response handling time in critical path by adaptively lowering the frequency of C2C interactions, hence enabling faster connection admissions than possible with current strong consistent DCP models.

Alternative approaches to the strong consistency model assume an eventually consistent state synchronisation where changes made to a controller instance get propagated over time across the cluster, thus solving the issue of blocking during synchronisation period. This allows the active instance to apply changes immediately and synchronise its state over time. Stale updates can have a detrimental effect on the entire network as the consistency of initial state determines the quality of output delivered by the controller's decisions. For example, a path finder application which computes a globally optimal path with regards to currently utilized resources and a given set of constraints (e.g. on delay, bandwidth etc.), may produce a suboptimal result due to stale information on the state of reservations. We consider the amount of observed result suboptimality as an input for our autonomous consistency level adaptation algorithm. This online algorithm outputs the consistency level required in order to provide an *exactly sufficient* amount of experienced result-correctness.

This paper investigates the relation between overhead minimization in SDN DCP and associated system correctness. In particular, we investigate the trade-off between low response-time delays and inefficiencies resulting from operating with stale data. Can strict requirements on low setup times be supported for different topologies and traffic patterns, while ensuring a sufficient degree of system correctness at all times? A strong consistent DCP introduces critical overhead in controller-to-controller synchronisation [13], while an eventually consistent DCP provides no correctness guarantees. We hence introduce a consistency model that provides instantaneous response for most requests, while bounding the observed correctness to a tunable threshold.

III. Related Work

In [14], authors investigate the performance of a strongly-consistent replicated data store implementation in Floodlight. While the results in a four-node cluster show promising transaction throughput for data store requests made for simple host-port mappings, significantly lower data-store performance was measured for load balancing and device management operations. The enforcement delays and blocking times in data-plane were not considered in that study. The evaluated network comprised a single OpenFlow switch, deployed in an out-of-band control channel. A more realistic in-band control network would cause higher variance in request inter-arrival times, higher switch configuration latencies and possibly bottlenecks in case all requests were destined for a centralized DCP leader.

OpenDaylight (ODL) [5] is a popular open-source SDN controller platform that relies on strong synchronisation between all controller instances, where any changes made to a particular instance have to be synchronised with a number of other instances in the cluster before they can be applied to the network. In ODL clustering implementation, a single controller instance is the leader of any state shard at any point in time and only the leader is allowed to make changes to its state. Following a state modification, the leader propagates the update to follower replicas that also hold the shard. The leader and its followers make up a strong consistency cluster.

ODL Clustering implements RAFT [9], a consensus algorithm similar to Paxos [7], which adds a leader election mechanism. Inside a RAFT cluster of size N, each state update is initiated by the leader, and propagated to at least $N/2$ followers. The followers must acknowledge the state update before the leader can continue processing further state changes. This blocking period takes an arbitrary amount of time depending on leader and follower placement, network and processing delays and quorum size. The requirement that at least $N/2 + 1$ cluster participants reflect the leader's state update is a minimum requirement for overlapping reads and writes in a reliable cluster that tolerates $\lfloor N/2 \rfloor$ failures [7]. Stringent deployments may require acknowledgement by a higher number of followers, hence causing additional overhead.

ONOS [6] and ONIX [15] are alternative and more recent controller designs that try to solve the issue of scalability by providing the APIs for selection of either strong or eventual consistency mode for its distributed state primitives. Applications which can operate correctly without strong consistent state updates may synchronise in eventually consistent manner. However, the active state consistency model does not change at runtime and must be hard-coded in the SDN application without knowing the exact constraints of the network it will be deployed in. The application's designer is unaware if the application might run correctly even if its state was eventually synchronised, which is the case when the probability of a state conflict is small. This may lead to pessimistic estimations of an application's requirements in the deployed domain. For example, routing applications may tolerate suboptimality if maximized network utilization is not a concern.

Levin *et al.* [16] show that distributed network functions such as load-balancers can work around eventual consistency and still deliver performance sufficient for production deployments. The *continuous consistency model* for geo-replicated services [17] allows application designers to bound the maximum distance between the local data state and final consistent state. In [17], distance in actual and stale state view is parametrized by the numerical and order error and state staleness. In context of SDN, [1] argues that linearisability is likely an unnecessary property for ensuring correct application of most network policies as the investigated policies often have simple correctness conditions. The authors state that determining a consistency model that is *necessary* and *sufficient* for network policies is an important research problem.

IV. Proposed Solution

In the following section, we introduce our adaptive consistency model for SDN controllers, where state synchronization occurs according to performance and consistency constraints set by the application at runtime. We make use of triggers that allow for dynamic switching of a *consistency level* (CL) on a per-state-fragment basis, based on a defined local threshold. This threshold could, for example, be specified as the allowed suboptimality of path reservations, based on the consistency of the reservation state of accepted paths. The suboptimality of a result may have a source in the fact that concurrently executed path reservations were not propagated to the instance that established the suboptimal path. However, the trade-offs of suboptimality and scalability might be tolerable in systems that require high request throughput and low response time. In our model, emphsynchronisation credits for state modifications are assigned to controller replicas. The credit-based approach bounds the data staleness which may arise as a consequence of concurrent and non-synchronised modification of states in cluster members. Consistency levels control the frequency of DCP-wide state-updates and eventual reconciliation of state-conflicts. Tight consistency levels lead to more frequent synchronisation, hence causing more control plane overhead compared to relaxed consistency levels, where synchronisation overhead is lower but the probability of occurring state-conflicts is raised. Our cost-aware algorithm is used to identify the balance in between the two key performance indicators; the *synchronisation overhead* and *system correctness* of DCP, by adapting the consistency levels based on correctness thresholds.

A. System Model

We model our DCP as a set of N controllers with each switch in the data plane configured to register with all controllers in the administrative domain (i.e. using *OFPCR_ROLE_EQUAL* mode in OpenFlow). Hence, each controller has full access to the switch and is equal to other controllers of the same role. Controllers react to external events (i.e. northbound requests and *packet-ins*) locally without synchronisation. While northbound requests are directed and hence handled by a single controller instance, the invariant of the *exactly-once* response must hold for asynchronous data plane events as well. By default with OpenFlow, all registered controllers receive all switch asynchronous messages. Hence, for scalability, the *OFPCR_EQUAL* mode would be combined with per-controller *Asynchronous Configuration* [18] to expose notifications only to a subset of controllers.

To circumvent the need for a strictly consistent cluster synchronisation, a resource state S in our system is associated with a maximum synchronisation credit amount C^S_{total}. Every credit value C_S represents a smallest non-divisible element of state S, allocated to an SDN controller instance K_N for concurrent and non-synchronised modification of the shared resource state S. By bounding the amount of concurrent modifications for state S per controller, staleness of a concurrently modified state S can be controlled. The total sum of synchronisation credits C^S_{total} represents the maximum number of concurrent modifications to state S during a *non-synchronisation cycle* (introduced below). We distribute the total synchronisation credits across P controllers so that $C^S_{total} = \sum_{K_N=1}^{P} C^S_{K_N}$.

Resource representation S can encompass physical or virtual network resources - e.g. bandwidth or flow table elements available for reservation. Depending on the required granularity of state management, limitations for bounded number of modifications may be configured either per state or per operation which affects multiple states. Hence we distinguish between *resource state credits* C^S_{total} for state *S*, and *execution credits* C^{Op}_{total} for operation *Op* that modifies the state *S*. Isolation of state modifications per controller allows for concurrent, unsynchronised modifications of the state S. In Figure 2, the resource credit $C^{S_{BW}}_{total}$ for bandwidth resource $S^{S1\rightarrow S2}_{BW}$ for edge $S1 \rightarrow S2$, and the execution credit $C^{add-flow}_{total}$ for operation *add-flow* that operates on $S^{S1\rightarrow S2}_{BW}$, are distributed

across all controllers.

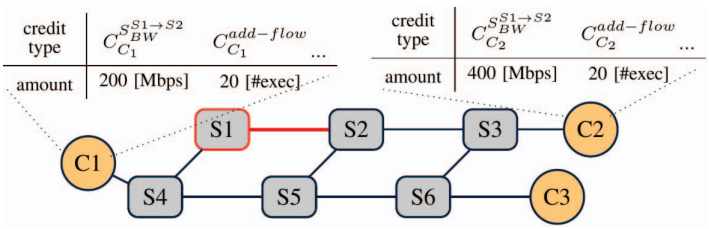

Fig. 2. An exemplary assignment of synchronisation credits to controllers in DCP. As *add-flow* operation modifies the resource state $S_{BW}^{S1\to S2}$. Notice that both the execution credits $C^{add-flow}$, and granular resource credits $C^{S_{BW}^{S1\to S2}}$ limit the maximum duration of the *non-synchronisation period* for the bandwidth resource $S_{BW}^{S1\to S2}$. Hence, it is expected that for state *S* the controller tracks a single type of synchronisation credit - C^S at granularity of state S; or C_{Op} at granularity of operation *Op* that modifies S.

Controller K_N modifies the resource state S in a manner where each update is handled locally for some *non-synchronisation period* T^S, without cluster-wide synchronisation. T^S is thus the time period elapsed in-between cluster-wide synchronisations of controllers' views of state S. In some cases, the strategy of concurrent modifications of a state S may result in global inefficiencies and suboptimality of a result. *Quality of a result* relates to the staleness of the input state S. To limit the effect of asynchronous access to a state, we introduce the notion of *consistency levels*. The choice of a consistency level CL_S for the state S, defines the maximum duration of non-synchronisation period T_{max}^S. In our model, the actual elapsed non-synchronisation period T_i^S is not the same for all synchronisation cycles i and may vary based on the occurrence frequency of synchronisation triggers. Observed system KPIs such as the encountered number of state synchronisation conflicts, are used as triggers for cluster-wide synchronisation procedure. The triggers that lead to adaptation of CL_S are explored further in Subsection IV-C. Consistency model adaptation results in a new active consistency level. In addition to T_{max}^S, a consistency level also governs the maximum number of locally executed, non-synchronised state updates by tightening or relaxing the resource/execution credit set $C_{K_N}^S$ / $C_{K_N}^{Op}$.

For example, SDN applications utilizing the edge-cost state for purposes of routing on a network graph may tolerate low consistency for edge-cost values. By decoupling the synchronisation and routing operations, scalability of routing execution is raised at the expense of result optimality.

B. Allocation of Synchronisation Credits

Manipulation of local resources allows for management of the resource reservations without a cluster-wide state synchronisation required, as long the reservations are conducted within the assigned credit boundaries. We distinguish between *execution credit sets* associated with the operation executions; and *resource state credit sets* associated with actual network resources.

Bounded execution credit sets. Credit set $C_{K_N}^{Op}$ represents the number of executions of operation Op that may run locally on controller K_N, without the cluster-wide synchronisation of states modified by the operation. Following a depletion of execution credits, cluster-wide synchronisation leads to view convergence for all states modified by the operation Op.

Bounded resource state credit sets. Credit set $C_{K_N}^S$ represents the total number of modifications of state S that may occur locally on controller K_N, without the cluster-wide synchronisation of the state. Following a depletion of resource state credits, cluster-wide synchronisation leads to view convergence for the state S on all replicas. All resources whose reservations may be represented as counter objects can also be modeled as resource state credits, an example being the per-edge bandwidth shares.

C. Adaptation of Consistency Levels

The choice of a consistency level governs the maximum non-synchronisation period T_{max}^S for state S. In case of a resource state credit set, it also bounds the number of allowed isolated modifications $C_{K_N}^S$ of state *S* in SDN controller K_N. In case of an execution credit set, it governs both T_{max}^S (where S may be modified by operation Op) and the number of allowed isolated executions $C_{K_N}^{Op}$ of *Op*. Consistency level change is state-/operation-specific, hence $C_{K_N}^{S_A} \neq C_{K_N}^{S_B}$ for resources S_A and S_B, and $C_{K_N}^{Op1} \neq C_{K_N}^{Op2}$ for operations $Op1$ and $Op2$. Following triggers lead to tightened consistency level CL_S, which result in shorter duration of T_{max}^S and a lower-than-current number of allowed isolated state modifications:

- *Cost of result suboptimality* of an eventually-consistent execution of an operation is above a threshold. If a path finding application computes paths which are considerably more suboptimal than would be the case with strong consistency, CL_S is set to a stricter level.
- *Cost of state-update conflicts* is above a specified threshold. If all controllers are able to run operations that modify the shared resource state set S, following a conflict detection (same state modified concurrently), controllers may raise the CL_S to a stricter level. A conflict resolution strategy must be deployed in order to reconcile a diverged state into a consistent state.
- *Setup-failures* wherein the callback associated with resource reservation does not result in a successful configuration of an external network device. Historical information of steps leading to setup failures is required for conflict detection and state reconciliation.

Following triggers lead to relaxation of CL_S , which, depending on type of resource, result in longer T_{max}^S and a higher-than-current number of allowed isolated state modifications:

- *Cost of result suboptimality* of an eventually-consistent execution of an operation is below a specific threshold - the result could on average be close-to-optimal even in case of a more relaxed consistency model.
- *Cost of consistency-related conflicts* is below a specific threshold and the observed probability of incurred state-update-related conflicts is small.

D. State Synchronisation Triggers

Various events may trigger the state synchronisation procedure. Based on the locality of an event, we distinguish local and external synchronisation triggers in SDN controller.

Locally activated Triggers. Following the exhaustion of execution credit set $C_{K_I}^{Op}$ or resource state credit set $C_{K_I}^S$;

or expiration of maximum duration of non-synchronisation period T^S_{max}, the controller K_I broadcasts its state to all other replicas. In case of unsuccessful device configuration or identified divergent state, the controller should assume a conflict has happened. It can then retrieve the current state of other cluster participants to identify and resolve the conflict.

Externally activated Triggers. Following an exhaustion of the execution credit set $C^{Op}_{K_I}$ or resource state credit set $C^S_{K_I}$; or expiration of maximum duration of non-synchronisation period T^S_{max} in controller K_I, all other controller replicas are triggered to update their copy of update history for state S.

E. Algorithm

Algorithm 1 depicts the state synchronisation procedure for state S_A and adaptation of active consistency level CL_{S_A}. The algorithm assumes *resource credits* assigned for isolated modifications of S_A. If instead *execution credits* were assigned for an operation that modifies S_A, the algorithm would adapt the size of the execution credit set. On observed state S_A modification in a remote controller K_{rem}, controller K_{loc} proceeds to adapt the consistency level CL_{S_A} for state S_A based on locally identified conflict-resolve and result suboptimality costs (C_{cflct} and C_{sbptml}, respectively) and given reference CL-cost threshold maps $max_{S_A}[CL^{curr}_{S_A}]$ and $min_{S_A}[CL^{curr}_{S_A}]$. For simplicity, cost thresholds are manually specified by the SDN application which operates on the state. In Algorithm 1, controller K_{rem} triggers a state synchronisation event by sending a state update to controller K_{loc}. K_{loc} then initiates the local state synchronisation as follows:

1) Lines 2-5: Any obvious state-conflicts are detected by controller K_{loc} (e.g. by using version vector [17] comparison to determine state-update causality). If version conflict is identified, conflict resolution cost C_{cflct} is computed depending on conflict-resolve strategy needed to converge the views for S_A.
2) Lines 6-8: The SDN application computes the induced suboptimality of local result in previous non-synchronisation period and outputs the cost of suboptimality C_{sbptml}. Computation of this cost is specific to application logic. We show an exemplary routing application that provides this cost in Section V.
3) Line 9-15: Based on frequency and amplitude of C_{cflct} and C_{sbptml} for observation interval n, the active CL is adapted. The depicted approach assumes a threshold-based assignment of CLs, where the cost thresholds associated with a CL are predefined.

After the state synchronisation procedure has identified the new consistency level, all controllers that hold S_A modify their resource state credit set size $C^{S_A}_{K_{loc}}$ and timers $T^{S_A}_{max}$ accordingly.

Algorithm 1: Pseudocode for state synchronisation procedure and adaptation of active state consistency level CL_{S_A} for state S_A in an SDN controller.

Input :
State update $S^{K_{rem},V_{rem}}_A$ with version vector V_{rem};
Initially active consistency level CL_{S_A};
Mapping of maximum non-synchronisation durations to various consistency levels $mapSyncPeriod[\,]$;
Mapping of synchronisation credit set sizes to various consistency levels $mapResourceCredits[\,]$;
Output : Adapted consistency level $CL^{new}_{S_A}$

1 **upon** *updated(stateUpdateQueue):*
2 $(S^{K_{loc},V_{loc}}_{Anew}) \leftarrow$ merge($S^{K_{loc},V_{loc}}_A$, $S^{K_{rem},V_{rem}}_A$)
3 **if** *conflictDetected*($S^{K_{loc},V_{loc}}_A$) = *true* **then**
4 $C_{cflct} \leftarrow$ handleConflict($S^{K_{loc},V_{loc}}_{Anew}$)
5 **end**
6 **if** *subOptimalityDetected*($S^{K_{loc},V_{loc}}_{Anew}$) = *true* **then**
7 $C_{sbptml} \leftarrow$ handleSuboptimality($S^{K_{loc},V_{loc}}_{Anew}$)
8 **end**
9 $C^{S_A}_{accum}.push(C_{cflct} + C_{sbptml})$
10 $C^{S_A}_{hist} \leftarrow mean(C^{S_A}_{accum}[C^{S_A}_{accum}.size - n : C^{S_A}_{accum}.size])$
11 $CL^{new}_{S_A} \leftarrow adaptCL(C^{S_A}_{hist}, CL_{S_A})$
12 $T^{S_A}_{max} \leftarrow mapSyncPeriod[CL^{new}_{S_A}]$
13 $C^{S_A}_{K_{loc}} \leftarrow mapResourceCredits[CL^{new}_{S_A}]$
14 genClusterEvent($CL^{new}_{S_A}, CL\_MOD$)

15 **Function** *adaptCL* ($measuredCost$, $CL^{curr}_{S_A}$)
16 **if** $measuredCost > max_{S_A}[CL^{curr}_{S_A}]$ **then**
17 return $CL_{curr}.tighten()$
18 **else if** $measuredCost < min_{S_A}[CL^{curr}_{S_A}]$ **then**
19 return $CL_{curr}.relax()$
20 **else**
21 return CL_{curr}

V. Implementation

A. Simulation of Concurrent Path Computations and Resource Reservations with Multiple SDN Controllers

To evaluate the effect of consistency level (CL) adaptation, we have developed a cluster-aware path-finding SDN controller application which measures and logs the suboptimality of routing results, where suboptimality is a function of execution credit set size (active CL), number of controller replicas, traffic model and network topology size. The application utilizes a bandwidth-constrained Dijkstra implementation to identify and reserve paths for uniformly selected source and destination pairs in a variable-size grid network. As the underlying network design and traffic patterns may bias the results experienced in practice, various topology sizes and traffic models were evaluated. The grid topology size was scaled between 5x5 to 25x25 vertices, with directed edges modeled as 1GbE links. The traffic models consider uniform specification of a flow bandwidth requirements in [1,30] Mbps range. We include an access control mechanism for new flow requests, which ensures that the edges whose 1GbE bandwidth capacities are exceeded during the reservation procedure are pruned before the routing procedure execution can take place. Pruning of links is done in order to guarantee that the new flows are may be embedded only on those paths which fulfil the bandwidth requirement. For simplification, the cost function which provides the cost input C_E for an edge E, considers only the sum of flows configured on that edge $C_E = \#flows_E$.

Following an execution of a path finding algorithm in operation *add-flow*, bandwidth resources are reserved at the

edges of the computed path. Whenever bandwidth utilization on an edge exceeds 80%, for every new flow, an existing, uniformly and randomly selected flow is removed, hence allowing for embedding of a large number of sequential flow requests and realistic results. To introduce concurrency in execution, multiple SDN Controller instances execute the path finding algorithm *add-flow* in isolation from other instances. Hence their local values of C_E might differ during the non-synchronisation period. The synchronisation trigger fires after an assigned set of execution credits $C^{Op}_{K_N}$ is exceeded on every controller instance K_N. The controllers then synchronise the costs of edges C_E and converge to the same state. We vary the execution credits $C^{add-flow}_{K_N}$, allocated for the path-finding operation *add-flow* and focus on identifying the trade-off between the frequency of cluster-wide synchronisations and the result suboptimality which is formally defined as the approximation factor $D_{subopt} = \frac{O_{optimal}}{O_{measured}}$, where $O_{optimal}$ is the cost of *true* optimal path (computed as if all reservation updates in system were serialized); and $O_{measured}$ is the actual measured cost of a sampled path identified by an isolated controller instance during the non-synchronisation period. As each path computation in a controller instance K_N only considers the reservation updates made during the non-synchronisation period on the local executor instance, in terms of total path cost, non-optimal paths with $D_{subopt} < 1$ could be determined. The suboptimality D_{subopt} is computed after the synchronisation trigger fires, followed by a cluster-wide synchronisation of C_E. In the simulation, isolated reservations made by different controller instances have often lead to concurrent reservation of bandwidth resources on the same edge, hence sacrificing the optimality of cheapest path finding. For reference, a set of SDN controllers which operate in the strong consistent mode, and hence serialize each resource reservation, would lead to optimal reservations and $D_{subopt} = 0$ in each scenario. While the cluster-wide synchronisation at the end of a non-synchronisation period must be implemented as a blocking task, all intermediate local state changes are instantaneous updates, hence providing obvious response time benefits compared to a strong consistent approach.

B. Experimental results

In this simulation, consistency levels (CLs) are defined by the amount of execution credits assigned to controller instances for isolated execution of operation *add-flow*. To evaluate the effects of different deployed topologies and traffic models on experienced suboptimality, we vary the CL and hence the number of isolated executions $C^{add-flow}_{K_N}$ per controller K_N. An exemplary mapping of amount of isolated executions of *add-flow* operations to CL_i is shown in Table I. By manipulating the active CL, controllers execute a variable number of path finding executions in isolation from other instances.

Consist.Lvl	CL_1	CL_2	CL_3	CL_4	CL_5	CL_6	CL_7	CL_8	CL_9	CL_{10}	CL_{11}
$\mathbf{C^{add-flow}_{K_N}}$	2	3	5	9	17	25	33	41	49	57	65

TABLE I. MANUAL MAPPING OF CLS TO EXECUTION CREDIT SET $C^{add-flow}_{K_N}$, FOR THE OPERATION *add-flow* THAT MAY RUN IN ISOLATION.

According to Figure 3, compared to smaller topology sizes, large topologies lead to higher result suboptimality D_{subopt}. Possible explanation lies in the fact that the state synchronisation is triggered only after the *execution credit set* is exceeded. With large topologies, an *add-flow* execution on

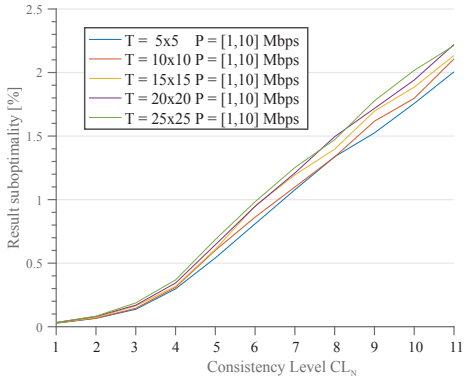

Fig. 3. Measured routing inefficiency for variable topology sizes T, variable consistency levels CL_N (as per Table I), and traffic flows with uniformly distributed bandwidth requirement $P = [1, 10]$ Mbps. The cost suboptimality scales with the strictness of active CL and topology size. Our cost function considers the sum of flows placed on edges as edge cost, and bandwidth capacity is chosen as admission constraint.

average updates a higher number of edge reservations than in the case of smaller topologies. Consistency management using *resource state credit sets* might possibly lead to lower, uniform suboptimality for variable topology sizes. As expected, the D_{subopt} of a routing operation scales linearly with the duration of non-synchronisation period during which the routing operations are executed in isolation. In the 3-controller setup with 65 concurrent flow additions, compared to a strongly consistent setup, mean result suboptimality for the eventually consistent setup measures at 2.22%.

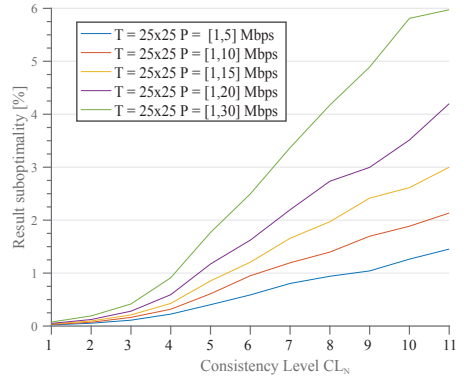

Fig. 4. Measured routing inefficiency for variable traffic models P, variable consistency levels CL_N (as per Table I), and a static 25x25 nodes grid topology. The cost suboptimality scales with the strictness of consistency levels and traffic models. Flows that request higher bandwidth are more often evaluated as cost-inefficient paths, as larger amount of resource is reserved on every admission and the edge capacity invariant is invalidated more frequently.

Figure 4 depicts similar behaviour but also shows how variation in traffic patterns influences performance of an eventually consistent system. By intelligent variation of assignment of execution credit amount and consistency level, bounding of experienced suboptimality to an arbitrary target value is possible, regardless of active traffic patterns.

Concurrent execution of an operation allows for faster handling of a large set of same-type requests. Figure 6 depicts the mean suboptimality when handling a batch of 140k path requests for all combinations of traffic models and topology

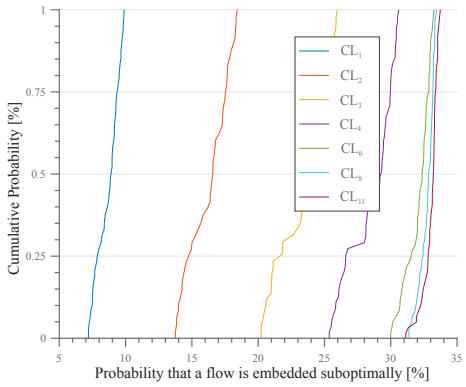

Fig. 5. Probability that during the non-synchronisation period a flow is embedded suboptimally. Cumulative probability is determined over all possible combinations of traffic models and topology sizes. With the relaxation of applied consistency level CL_N (as per Table I), probability rises that a sampled reference path is suboptimal at end of its non-synchronisation period. With more relaxed $CL_N > CL_6$, as many as 34% of sampled paths are marked as suboptimal. Regardless of this probability, the inefficiency of a suboptimal path cost remains low (see Figures 3 and 4).

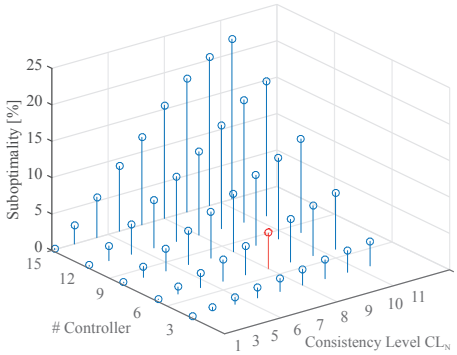

Fig. 6. Load-balancing of path requests over a variable-size cluster of SDN controllers. The average suboptimality of sampled flows is determined over all possible combinations of traffic models and topology sizes. For different consistency levels CL_N (as per Table I), each controller instance executes a variable number of *add-flow* operations in isolation. The state of admissioned flows on every edge E (and hence the cost C_E) is distributed to all controller instances at the end of non-synchronisation period.

sizes. We vary the size of cluster between 3 to 15 replicas. The CL defines the number of concurrent path finding executions at each controller instance. The suboptimality is shown to scale with the number of concurrent path finding executions, and peaks at 25% for the largest cluster size of 15 controllers. This evaluation shows that a cluster of 6 SDN controllers is able to cope with CL_8 (41 isolated requests per instance), while limiting the path cost difference to less than 6% compared to similar but strong consistent setup. In scenarios where flows are short-lived and higher cost inefficiencies can be tolerated, adaptation of the CL assigns higher shares of execution credits to controllers. Results hence show that a CL adaptation mechanism, such as our threshold-based approach (see Algorithm 1), can oscillate the experienced result suboptimality around a target value, while minimizing the flow setup latency independent of network topology and traffic models.

VI. Conclusion and Future Work

In this work, we introduce an algorithm for adaptation of consistency levels that leverages observed frequency and weight of synchronisation conflicts in order to find a consistency level appropriate for targeted system optimality and correctness. In terms of response delay, enabling non-synchronised global switch configurations is especially efficient when working with short-lived flows that require fast response. By not relying on costly consensus after every resource state update, end-hosts benefit from shortened request-handling time in the SDN controller. If state synchronisation conflicts occur and correctness is endangered, our system adapts autonomously to a more appropriate consistency level. We have shown by simulation that adaptive consistency in an SDN DCP can provide acceptable inefficiency compared to its strong consistent counterpart, while keeping the response-time benefits of the eventual consistency model. Hence, we firmly believe that our model can pave the way for a scalable SDN DCP design. While strong and eventually consistent DCPs were compared in terms of correctness metrics, time-response analysis is left for later comparison. To this end, trade-offs between short execution time and blocking period incurred by conflict resolve procedure require further attention.

This work has received funding from the EC's Horizon 2020 programme under grant agreement #671648 VirtuWind.

References

[1] A. Panda, C. Scott, A. Ghodsi *et al.*, "CAP for Networks," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, 2013.

[2] P. Bailis and A. Ghodsi, "Eventual Consistency Today: Limitations, Extensions, and Beyond," *Commun. ACM*, vol. 56, no. 5, 2013.

[3] A. Demers, D. Greene, Hauser *et al.*, "Epidemic Algorithms for Replicated Database Maintenance," in *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, 1987.

[4] W. Vogels, "Eventually Consistent," *Commun. ACM*, vol. 52, 2009.

[5] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a Model-Driven SDN Controller architecture."

[6] P. Berde, M. Gerola, J. Hart *et al.*, "ONOS: Towards an Open, Distributed SDN OS," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, 2014.

[7] L. Lamport, "Paxos Made Simple," *ACM SIGACT News 32*, 2001.

[8] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, 2010.

[9] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *USENIX Annual Technical Conference*, 2014.

[10] G. Petropoulos, F. Sardis, S. Spirou *et al.*, "Software-defined internetworking: Enabling coordinated QoS control across the internet," in *23rd International Conference on Telecommunications*, 2016.

[11] J. Guck *et al.*, "Function Split between Delay-Constrained Routing and Resource Allocation for Centrally Managed QoS in Industrial Networks," *IEEE Transactions on Industrial Informatics*, no. 99, 2016.

[12] "5G Whitepaper: The Flat Distributed Cloud (FDC) 5G Architecture Revolution." University of Surrey - 5G Innovation Centre, 2016.

[13] A. Bianco *et al.*, "The Role of Inter-Controller Traffic for Placement of Distributed SDN Controllers," *CoRR*, vol. abs/1605.09268, 2016.

[14] F. A. Botelho, F. M. V. Ramos *et al.*, "On the Feasibility of a Consistent and Fault-Tolerant Data Store for SDNs," in *Proceedings of the 2013 Second European Workshop on Software Defined Networks*, 2013.

[15] T. Koponen *et al.*, "ONIX: A Distributed Control Platform for Large-scale Production Networks," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, 2010.

[16] D. Levin, A. Wundsam *et al.*, "Logically Centralized?: State Distribution Trade-offs in Software Defined Networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012.

[17] H. Yu *et al.*, "Design and Evaluation of a Continuous Consistency Model for Replicated Services," in *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation*, 2000.

[18] "OpenFlow Switch Specification Version 1.0.0," ONF, 2009.