Decoupling of Distributed Consensus, Failure Detection and Agreement in SDN Control Plane

Ermin Sakic*†, Wolfgang Kellerer*
*Technical University Munich, Germany, † Siemens AG, Germany
E-Mail:*{ermin.sakic, wolfgang.kellerer}@tum.de, † ermin.sakic@siemens.com

***Abstract*—Centralized Software Defined Networking (SDN) controllers and Network Management Systems (NMS) introduce the issue of controller as a single-point of failure (SPOF). The SPOF correspondingly motivated the introduction of distributed controllers, with replicas assigned into *clusters* of controller instances replicated for purpose of enabling high availability. The replication of the controller state relies on distributed consensus and state synchronization for correct operation. Recent works have, however, demonstrated issues with this approach. False positives in failure detectors deployed in replicas may result in oscillating leadership and control plane unavailability.**

In this paper, we first elaborate the problematic scenario. We resolve the related issues by decoupling failure detector from the underlying signaling methodology and by introducing event agreement as a necessary component of the proposed design. The effectiveness of the proposed model is validated using an exemplary implementation and demonstration in the problematic scenario. We present an analytic model to describe the worst-case delay required to reliably agree on replica failures. The effectiveness of the analytic formulation is confirmed empirically using varied cluster configurations in an emulated environment. Finally, we discuss the impact of each component of our design on the replica failure- and recovery-detection delay, as well as on the imposed communication overhead.

I. Introduction

In recent years, distributed controller designs have been proposed to tackle the issue of network control plane survivability and scalability. Two main operation models for achieving redundant operation exist: i) the Replicated State Machine (RSM) approach, where each controller replica of a single cluster executes each submitted client request [1]–[3]; ii) the distributed consensus approach, where particular replics execute client requests and subsequently synchronize the resulting state updates with the remaining cluster members [4]–[10]. The latter model necessarily leverages the *distributed consensus* abstraction as a mean of ensuring a single leader and serialized client request execution at all times.

For example, in Raft [11], the leader is in charge of serialization of message updates and their dissemination to the followers. In the face of network partitions and unreliable communication channels between replicas, the simple timeout-based failure detector of Raft cannot guarantee the property of the stable leader [8]. Zhang et al. [8] demonstrate the issue of the *oscillating leadership* and *unmet consensus* in two scenarios involving Raft. To circumvent the issues, the authors propose a solution - redundant controller-to-controller connections with packet duplication over disjoint paths. This solution is expensive for a plethora of reasons: a) additional communication links impose significant communication overhead and thus negatively impact the system scalability; b) selection of the paths that would alleviate the failures is not a trivial task, especially when the locality and the expected number of link/node failures is unknown; c) certain restrictions on topology design are imposed - their solution requires disjoint paths between any two controller replicas. In another recent evaluation of Raft in ONOS [6], Hanmer et al. [12] confirm that a Raft cluster may continuously oscillate its leader and crash under overload. This behavior may arise due to increasing computation load and incurred delays in heartbeat transmissions in the built-in signaling mechanism. This issue is unsolvable using disjoint flows alone [8].

In contrast to [8], we identify and associate the core issue of unreliable Raft consensus with the lacking design of its failure detector. In fact, the issue is not limited to ONOS alone - state-of-the-art control plane solutions, i.e., OpenDaylight [4], Kubernetes [13], Docker (Swarm) [14] all tightly couple Raft leader election procedure with the follower-leader *failure detection*, thus allowing for false failure suspicions and arbitrary leader elections as a side-effect of network anomalies.

A. Our contribution

We solve the issue of oscillating consensus leadership and unmet consensus as follows: We alleviate the possibility of reaching false positives for suspected controller replicas by disaggregating the process of distributed agreement on the failure event, from the failure detection. Namely, we require that an agreement is reached among active controller processes on the status of the observed member of the cluster, prior to committing and reacting to a status update.

To this end, we make the following contributions:

- We propose a model for realizing robust distributed event detection, comprising four entities: signaling, dissemination, event agreement, and failure detection.
- We validate correctness of the proposed model using implementation of two instances of each of the four entities in a problematic scenario [8].
- We evaluate all instance couplings with varied parametrizations. Metrics of failure and recovery detection, as well as the per-replica communication overhead; are presented and discussed.
- We formulate a model for computation of worst-case agreement period for an observed replica's status, based on incurred per-entity delays. The correctness of the formulation is confirmed by empirical measurements in an emulated environment.

To our best knowledge, this paper is the first to raise the issue of close coupling of Raft consensus and underlying failure detection.

Paper structure: Sec. II motivates this work with an exemplary scenario of the oscillating Raft leader election in the face of link failures. Sec. III presents the assumed system model. Sec. IV presents the reference model of the reliable event detection framework and details the designs of two exemplary instances for each of the four entities. Sec. V discusses the evaluation methodology. Sec. VI presents the empirical results of our event detection framework for varied control plane configurations. Sec. VII presents the related work. Sec. VIII concludes the paper.

II. Background and Problem Statement

With single-leader consensus, e.g., in Raft-based clusters, the controller replicas agree on a maximum of a single *leader* during the current round (called *term* in Raft). The leader replica is in charge of: i) collecting the client update requests from other Raft nodes (i.e., *followers*); ii) serializing the updates in the underlying replicated controller data-store; iii) disseminating the committed updates to the followers.

In the case of a leader failure, the remaining active replicas block and wait on leader heartbeats for the duration of the expiration period (i.e., the *follower timeout* [11]). If the leader has not recovered, the follower replicas announce their candidature for the new leader for the future term. The replicas assign their votes and, eventually, the majority of the replicas vote for the same candidate and commonly agree on the new leader. During the re-election period, the distributed control plane is unavailable for serving incoming requests [10]. However, not just the leader failures, but also unreliable follower-leader connections as well as link failures, may lead to arbitrary connection drops between members of the cluster and thus the reinitialization of the leader election procedure [8]. Realizations of the Raft consensus protocol [4], [6] implement the controller-to-controller synchronization channel as a point-to-point connection, thus realizing $\frac{|\mathcal{V}|\cdot(|\mathcal{V}|-1)}{2}$ bidirectional connections per replica cluster.

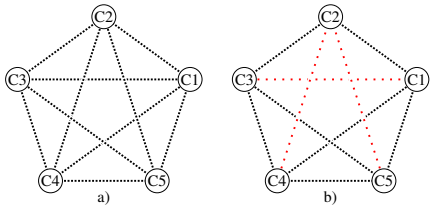

Fig. 1. *Non-failure* (a) and exemplary *failure case* (b) scenarios with injected communication link failures (loosely dotted). In the non-failure case, direct any-to-any connectivity between the replicas is available. The depiction is agnostic of the physical topology and represents a logical connectivity graph.

To demonstrate a possible issue with the consensus design, consider the task of reaching consensus in the multi-controller scenario comprising $|\mathcal{V}| = 5$ controllers, depicted in Fig. 1 a):

1) Assume all depicted controller replicas execute the Raft [8] consensus protocol. Let replica `C2` be selected the leader in term `T1` as the outcome of the leader election procedure.
2) Assume multiple concurrently or sequentially induced link failures, i.e., on the logical links `(C2, C4)` and `(C2, C5)`. Following the expiration of the *follower timeout*, `C4` and `C5` automatically increment the current term to `T2` and initiate the leader re-election procedure by advertising their candidate status to `C1` and `C3`. Since a higher-than-current term `T2` is advertised by the new candidate(s), `C1` and `C3` accept the term `T2`, eventually vote for the same candidate, resulting in its election as leader. Depending on whose follower timer had expired first, either `C4` or `C5` is thus elected as the leader for `T2`.
3) After learning about `T2`, `C2` steps away from its leadership and increments its term to `T2`. As it is unable to reach `C4` and `C5`, `C2`'s *follower timeout* eventually expires as well. Thus, `C2` proceeds to increment its term to `T3` and eventually acquires the leadership by collecting the votes from `C1` and `C3`. Term `T3` begins. The **oscillation of leadership** between `C4`/`C5` and `C2` can thus repeat indefinitely.
4) Should either `C4`/`C5` or `C2` individually apply a client update to their data-store during their leadership period, the replicas `C1` and `C3` will replicate these updates and thus forward their log correspondingly. This in return leads to the state where only replicas `C1` and `C3` are aware of the most up-to-date data-store log at all times. As per the rules of Raft algorithm [15], from then onwards, only `C1` and `C3` are eligible for the leadership status.
5) Since the link/connection `(C1, C3)` is also unavailable (ref. Fig. 1 b)), the leadership begins to oscillate between replicas `C1` and `C3`, following the logic from Steps 1-3.

Hence, in the depicted scenario, the Raft cluster leadership oscillates either between `C2` and `C4`/`C5`, or `C1` and `C3`. Zhang et al. [8] have empirically evaluated the impact of the oscillating leadership on system *availability* (the period during which a leader is available for serving client requests). During a 3-minute execution of an oscillating leadership scenario using LogCabin's Raft implementation and 5 replicas, the authors observed an *unavailability* of $\sim$58% and 248 leadership shifts, even though the network itself was not partitioned.

We introduce a generic design that alleviates these issues completely. We ensure the above Step 2 never occurs, by requiring all active replicas to reach agreement on the inactive replica prior to restarting the leader election.

III. System Model

We assume a distributed network control model (e.g., an OpenFlow [16] model), where controller replicas synchronize and serialize their internal state updates to keep a consistent view of the replicated data-store. Following a data-store change the controller initiating the state update propagates the change to the current Raft leader. The leader is in charge of proxying the update to all members, and committing the update into the persistent store after at least $\lceil\frac{|\mathcal{V}|+1}{2}\rceil$ controllers have agreed on the update ordering.

The communication topology between replicas is modeled by a connectivity graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of controller replicas and $\mathcal{E}$ is the set of active *communication links* between the replicas. An active link (i, j) denotes an available point-to-point connection between the replicas i and j. The communication link between two replicas may be realized using any available mean of data-plane forwarding, e.g., provisioned by OpenFlow flow rule configuration in each hop of the path between i and j, or by enabled MAC-learning in non-OpenFlow data plane. The controllers are assumed to synchronize in either *in-band* or *out-of-band* [17] manner. We assume non-Byzantine [2], [18] operation of network controllers.

Let $\mathcal{D}$ contain the guaranteed worst-case delays between any two directly reachable replicas. In the *non-failure case* (ref. Fig. 1 a)), $\forall(i, j) \in \mathcal{E} : \exists d_{i,j} \in \mathcal{D}$. In the *failure case*, depicted in Fig. 1 b), we assume that partitions in the connectivity graph have occurred (e.g., due to flow rule misconfigurations or link failures). Connectivity between the replicas i and j may then require message relaying across (multiple) other replicas on the path $\mathcal{P}_{i,j}$, with $\mathcal{P}_{i,j} \subseteq \mathcal{E}$. In both *non-failure* and *failure* cases, direct communication or communication over intermediate proxy (relay) replicas, respectively, is possible between any two active replicas at all times.

Replicas that are: a) physically partitioned as a result of one or more network failures, and are thus unreachable either directly or over a relayed connection by the replicas belonging to majority partition; or are b) disabled / faulty; are eventually considered *inactive* by all active replicas in the majority partition.

Each controller replica executes a process instance incorporating the components for *failure detection* and *event agreement*, together with *signaling* and *dissemination* methods (ref. Sec. IV). Any *correct* proposed combination of these methods must fulfill the properties of:

- *Strong Completeness*: Eventually all inactive replicas are suspected by all active replicas.
- *Eventual Strong Accuracy*: Eventually all active replicas (e.g., still suspected but already recovered) are not suspected by any other active replica.

TABLE I
NOTATION USED IN SECTION IV

Notation	Definition
$\mathcal{V}$	Set of controller replicas
$\mathcal{E}$	Set of active communication links
$\mathcal{D}$	Set of unidir. delays between *directly* reachable replicas
t_S	Fixed signaling interval
ϕ	Suspicion threshold for φ-FD failure detector
t_T	Timeout threshold for T-FD failure detector
l_M	Confirmation multiplier for list-based L-EA event agreement

IV. REFERENCE MODEL OF OUR FRAMEWORK

The abstract reference model depicted in Fig. 2 portrays the interactions between the four core entities:

- *Failure Detection (*-FD)*: Triggers the suspicion of a failed replica by means of local observation - e.g., an *active* replica may suspect a remote replica *inactive* following a missing acquisition of replica's heartbeats.
- *Event Agreement (*-EA)*: Deduces the suspected replica as *inactive* or recovered following the collection and evaluation of matching confirmations from at least $\lceil \frac{|\mathcal{V}|+1}{2} \rceil$ *active* replicas. It uses the underlying Failure Detection to update its local view of other replicas' state.
- *Signaling (*-S)*: Dictates the semantics of the interaction between processes, e.g., the protocol (ping-reply or periodic heartbeat exchange), and configurable properties (e.g., periodicity of view exchange). Signaling ensures that the local view of one replica's observations is periodically advertised to all other reachable replicas.
- *Dissemination (*-D)*: Dictates the communication spread (e.g., broadcast/unicast or relay using gossiping). As depicted in Fig. 2, the Dissemination is leveraged for periodic signaling of replica's status view (by the Signaling entity), as well as for triggering asynchronous updates on newly discovered failures (by the Failure Detection).

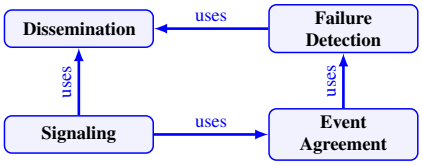

Fig. 2. The proposed event detection reference model.

We next identify two exemplary instances for each of the four entities and provide an analytic expression for the worst-case delay, specific to that instance. The sum of the delays introduced by each of the components denotes the predicted waiting period to reach agreement on an observed remote replica's status. Note that the worst-case convergence time can be determind only in scenarios where all active replicas can communicate either directly or through relay replicas.

A. Dissemination

In our model: i) replicas learn about the status of other members (active or inactive) using local Failure Detection; and ii) confirm those assumptions by collecting information from active remote members in the Event Agreement module. Dissemination governs how the information containing replicas' local view of other cluster members' state is exchanged. We exemplarily distinguish Round Robin Gossip (RRG-D) and Best Effort Broadcast (BEB-D) dissemination:

1) Round Robin Gossip (RRG-D) Dissemination: In RRG-D, message transmissions occur periodically on a per-round basis. During each round, the gossip sender selects the receiver based on the current round number. This ensures that in a fault-free setup, given periodic heartbeat message exchange, each state view update is propagated to all cluster members in a maximum of $\lceil log_2(|\mathcal{V}|) \rceil$ subsequent rounds. The heartbeat destination identifier ID_{Dst} is selected based on current round R_i as:

$$ID_{Dst} = ID_{Src} + 2^{R_i - 1} \mod |\mathcal{V}| : 1 \leq R_i \leq \lceil log_2(|\mathcal{V}|) \rceil.$$

Non-Failure case: The worst case incurred by the gossip transmission between i and j in the *non-failure case* corresponds

to the sum of delays on the longest directional gossip path (but limited by $log_2(|\mathcal{V}|)$) and the signaling interval T_S:

$$T_D(i,j) = \sum_{l=1}^{log_2(|\mathcal{V}|)} (h_l + T_S) : h_l \in \bigcup_{k=1}^{log_2(|\mathcal{V}|)} \mathcal{H}_k^{i,j},$$

where set $\mathcal{H}_k^{i,j}$ contains the k-th largest element of delay set $\mathcal{D}^{i,j} \subseteq \mathcal{D}$ (union of delays for all links of all unidirectional gossip paths between i and j), determined using induction:

$$\mathcal{H}_k^{i,j} = \{a \in \mathcal{D}_{k-1}^{i,j}; a \geq b \forall b \in \mathcal{D}_{k-1}^{i,j}\}$$
$$\text{with } \mathcal{D}_0^{i,j} = \mathcal{D}^{i,j}; \mathcal{D}_k^{i,j} = \mathcal{D}_{k-1}^{i,j} \setminus \mathcal{H}_k^{i,j}$$

Failure case: For the *failure case*, we assume an availability of only one path between the replicas i and j, hence the worst-case dissemination delay corresponds to the sum of all delays across all replica pairs on the longest gossip path $\mathcal{P}_{i,j}$ and the periodic signaling interval T_S:

$$T_D(i,j) = \sum_{(k,l) \in \mathcal{P}_{i,j}} (d_{k,l} + T_S) : d_{k,l} \in \mathcal{D}$$

2) Best Effort Broadcast (BEB-D) Dissemination: With BEB-D, heartbeats are propagated from source replica to all remaining cluster members concurrently in a single round. In contrast to RRG-D, in the *non-failure case* messages must not be relayed, and each message transmission incurs an overhead of $\mathcal{O}(|\mathcal{V}| - 1)$ concurrent transmissions. The worst case delivery time between replicas i and j in the *non-failure case* corresponds to the sum of the worst-case uni-directional delay $d_{i,j}$ and the signaling period T_S. In the *failure case*, intermediate replicas must relay the message, hence the worst-case equals the gossip variant:

$$T_D(i,j) = \begin{cases} d_{i,j} + T_S : d_{i,j} \in \mathcal{D} & \text{if } \textit{non-failure case} \\ \sum_{(k,l) \in \mathcal{P}_{i,j}} (d_{k,l} + T_S) : d_{k,l} \in \mathcal{D} & \text{if } \textit{failure case} \end{cases}$$

B. Signaling

We leverage the unidirectional heartbeats and ping-replies as carriers for transmission of the Event Agreement payloads.

1) Uni-Directional Heartbeat (UH-S): In UH-S, communicating controllers advertise their state to the cluster members in periodic intervals. The periodic messages are consumed by the failure detectors (φ-FD and T-FD, ref. Sec. IV-C) to update the liveness status for an observed replica. The parametrizable interval t_S denotes the round duration between transmissions.

2) Ping-Reply (PR-S): With PR-S signaling, transmissions by the sender are followed by the destination's reply message containing any concurrently applied updates to the destination's Event Agreement views (i.e., the *local* or *global* agreement matrix or the agreement list).

Incurred waiting time for both UH-S and PR-S equals the configurable waiting period between transmissions $T_S = t_S$.

C. Failure Detection

1) φ-Accrual Failure Detector (φ-FD): We next outline the worst-case waiting time for a successful local failure detection of an inactive replica using the φ-FD [19].

To reliably detect a failure, the φ-FD must detect a suspicion confidence value higher than the configurable threshold ϕ. The function $\varphi(\Delta_T)$ is used for computation of the confidence value. To guarantee the trigger, it must hence hold $\varphi(\Delta_T) \geq \phi$, where Δ_T represents the time difference between last observed heartbeat arrival and the local system time. The accrual failure detection $\varphi(\Delta_T)$ leverages the probability that a future heartbeat will arrive later than Δ_T, given a window of observations of size ϕ_W containing the previous inter-arrival times:

$$\varphi(\Delta_T) = -log_{10}(P'_{\mu,\sigma}(\Delta_T))$$

Assuming normally distributed inter-arrivals for previous inter-arrival observations, $P'_{\mu,\sigma}(\Delta_T)$ is the complementary CDF of a normal distribution with mean μ and standard deviation σ. We are particularly interested in the earliest time difference Δ_T^F at which the failure suspicion is triggered, i.e., the time difference for which it holds $\varphi(\Delta_T^F) = \phi$. From here we can directly expand the expression to:

$$\begin{aligned} \varphi(\Delta_T^F) &= \phi \\ -log_{10}(P'_{\mu,\sigma}(\Delta_T^F)) &= \phi \\ -log_{10}(1 - P_{\mu,\sigma}(\Delta_T^F)) &= \phi \\ 1 - P_{\mu,\sigma}(\Delta_T^F) &= 10^{-\phi} \\ 1 - \frac{1}{2}(1 + erf(\frac{\Delta_T^F - \mu}{\sqrt{2} \cdot \sigma})) &= 10^{-\phi} \end{aligned}$$

where $erf()$ is the error function and $erfinv()$ its inverse. Resolving after Δ_T^F evaluates to:

$$\Delta_T^F = \sqrt{2} \cdot \sigma \cdot erfinv(1 - 2 \cdot 10^{-\phi}) + \mu \qquad (1)$$

Note, however, that the recalculation of $\varphi(\cdot)$ executes in discrete intervals. Thus, to estimate the worst-case waiting period after which φ-FD triggers, we must also include the configurable recalculation interval φ_R: $T_{FD} = \Delta_T^F + \varphi_R$.

2) Timeout-based Failure Detector (T-FD): T-FD triggers a failure detection after a timeout period t_T has expired, without incoming heartbeats transmissions for the observed replica. Compared to the accrual φ-FD detector, it is less reliable and prone to false positives in the case of highly-varying network delays. The worst-case waiting period introduced by T-FD corresponds to the longest tolerable period without incoming heartbeats $T_{FD} = t_T$, where t_T is the configurable timeout interval. In contrast to φ-FD, its parameter is intuitive to select. Furthermore, its analytical solution does not require collection of current samples for μ and σ estimation (ref. Eq. 1).

D. Event Agreement - Reaching Consensus on Replica State

The replica i can become suspected as inactive by replica j, following either the failure of the observed replica i or the failure of the communication link (i,j). Since individual replicas may lose connection to a particular non-failed but unreachable replica (e.g., as a consequence of failed physical

link / node undermining the logical link (i, j)), a subset of replicas may falsely consider an active replica as inactive. To alleviate this issue, we introduce the Event Agreement entity.

To reach agreement on a replica's state, the observing replicas must first acknowledge the failure of the suspected replica in an agreement phase. We distinguish two types of event agreement - "*local*" and "*global*" agreement:

- *Local Agreement* on failure (recovery) is reached when a replica observes that at least $\lceil \frac{|\mathcal{V}|+1}{2} \rceil$ *active* replicas have marked the suspected replica as `INACTIVE` (`ACTIVE`).
- *Global Agreement* is reached when a replica observes that that at least $\lceil \frac{|\mathcal{V}|+1}{2} \rceil$ *active* replicas have confirmed their *Local Agreement* for the suspected replica.

The global agreement imitates the semaphore concept, so to ensure that active replicas have eventually reached the same conclusion regarding the status of the observed replica. We assume that physical network partitions may occur. To this end, the majority confirmation is necessary in order to enable progress only among the active replicas in the majority partition. Reaching the event agreement in a minority partition is thus impossible. We propose two Event Agreement instances:

1) List-based Event Agreement (L-EA): With L-EA, reaching the agreement on the status of an observed replica requires collecting a repeated unbroken sequence of a minimum of $l_M \cdot (|\mathcal{V}| - 1)$ matching observations from active replicas. l_M is the confirmation multiplier parameter allowing to suppress false positives created e.g., by repeated bursts of same events stemming from a single replica.

L-EA maintains a per-replica local and global counter of matching observations in the local and global failure (and recovery) lists of length $|\mathcal{V}| - 1$, respectively. On suspicion of a failed replica, the observer replica increments the counter of suspected replica and forwards its updated local failure list to other active replicas. After receiving the updated list, the receiving replicas similarly update their own counter for any replica whose counter is set to a non-zero value in the received list. The active replicas continue to exchange and increment the local failure counter until eventually $l_M \cdot (|\mathcal{V}|-1)$ matching heartbeats are collected and thus the local agreement on replica's failed status is reached. The suspected replica's counter in the global failure list is then incremented as well, and process continues for the global agreement. If any active replicas identify the suspected replica as recovered, they reset the corresponding counter in both local and global lists and distribute the updated vector, forcing all other active replicas to similarly reset their counters for the suspected replica.

After simplification, the worst-case delay of global agreement on the monitored replica's state can be expressed as:

$$T_C = 2 \cdot l_M \cdot \mathcal{O}(C) \cdot (\max_{i,j \in \mathcal{V}} T_D(i, j) + 1)$$

where $\max_{i,j \in \mathcal{V}} T_D(i, j)$ corresponds to the worst-case dissemination time between any two active replicas i and j and C is the computational overhead of list processing time in the source and destination replicas (merger and update trigger).

2) Matrix-based Event Agreement (M-EA): Another realization of the Event Agreement entity is the matrix-based M-EA. Our design extends [20] to cater for the recovery of failed replicas and to feature the global agreement capability.

In short, all replicas in the system maintain a status matrix, and periodically inform all other active replicas of their own status matrix view. The status matrix contains the vector with the locally perceived status for each observed replica, as well as the vectors corresponding to other replicas' views. Thus, each replica maintains its own view of the system state through interaction with local failure detector instance, but also collects and learns new and global information from other active replicas. The status matrix is a $|\mathcal{V}|x|\mathcal{V}|$ matrix with elements corresponding to a value from set {`ACTIVE`, `INACTIVE`, `RECOVERING`}. `RECOVERING` state is necessary in order to consistently incorporate a previously failed but recovered replica, so that all active replicas are aware of its reactivation.

Following a failure of a particular replica or a communication link to that replica, the Failure Detection of an active neighboring replica initiates a trigger and starts suspecting the unreachable replica. The observing replica proceeds to mark the unreachable replica as `INACTIVE` in its perceived vector and subsequently asynchronously informs remaining replicas of the update. The remaining replicas store the state update in their own view matrix and evaluate the suspected replica for failure agreement. Agreement is reached when all active replicas have marked the suspected replica as `INACTIVE`.

In M-EA, each replica maintains two matrix instances:

1) *local* agreement matrix, where each locally perceived suspicion results in a state flip from `ACTIVE` $\rightarrow$ `INACTIVE`, and where a newly seen heartbeat for a previously failed replica leads to a state flip `INACTIVE` $\rightarrow$ `RECOVERING`. As soon as all active replicas have marked the suspected replica as `INACTIVE` (`RECOVERING` for a recovered replica), the local agreement has been reached (state flip `RECOVERING` $\rightarrow$ `ACTIVE` occurs for recovered replica).
2) *global* agreement matrix, where state flip from `ACTIVE` $\rightarrow$ `INACTIVE`, and `INACTIVE` $\rightarrow$ `ACTIVE`, occurs only if the local agreement was previously reached for the updated state of the observed replica.

Dissemination triggers: Dissemination of the matrices is triggered periodically (according to the Signaling entity). If a replica has, however, observed a failure using its local failure detector or has received a heartbeat from a replica considered inactive, it triggers the matrix dissemination asynchronously.

The worst-case global agreement duration using M-EA equals the time taken to exchange the perceived failure updates and reach global and local agreement on the target's state:

$$T_C = 4 \cdot (\max_{i,j \in \mathcal{V}} T_D(i, j) + \mathcal{O}(C))$$

where $\max_{i,j \in \mathcal{V}} T_D(i, j)$ corresponds to the worst-case dissemination time between any two active replicas i and j, and C is the computational overhead of matrix processing time in the source and destination replicas (merger and update trigger). Two rounds are required to synchronize the update in the local agreement matrix between: i) the replica that most-recently lost the connection to the failed replica; and ii) the most remote

other active replica. Correspondingly, global agreement matrix views are exchanged only after the local agreement is reached, thus adding additional two delay rounds to T_C (total of four).

E. Worst-Case Convergence Time

Upper bound event detection convergence time corresponds to the sum of the time taken to detect the failure and time required to reach the global agreement across all active replicas:

$$T_{WC} = T_C + T_{FD} \tag{2}$$

T_C and T_{FD} are both functions of T_D, thus signifying the importance of evaluation of the performance impact in decoupled manner. In empirical evaluation in Sec. VI, we conclude that the presented worst-case analysis is pessimistic and may be hardly reachable in practice. Hence, we also highlight the importance of evaluation of the average case in experimental evaluation of different combinations.

V. Evaluation Methodology

To evaluate the impact of different instances of the four entities of our event detection framework, we implement and inter-connect each as a set of loosely coupled Java modules. We vary the configurations of particular instances as per Table II so to analyze the impact of parametrizations.

TABLE II
PARAMETERS USED IN EVALUATION

Param.	Intensity	Unit	Meaning	Instance
$\|\mathcal{V}\|$	$[4, 6, 8, 10]$	N/A	No. of controller replicas	ALL
l_M	$[2, 3]$	N/A	Confirmation multiplier	L-EA
t_S	$[100, 150, 200]$	[ms]	Signaling interval	UH-S, PR-S
t_T	$[500, 750, 1000]$	[ms]	Timeout threshold	T-FD
ϕ	$[10, 15, 20]$	N/A	Suspicion threshold	φ-FD
φ_W	$[1000, 1500, 2000]$	N/A	Window Size of Inter-arrival Time Observations	φ-FD
φ_R	$[100, 150, 200]$	[ms]	ϕ Recalculation time	φ-FD
$\mathcal{O}(C)$	1	[ms]	Processing overhead constant	L-EA, M-EA

We have varied the measurements in two scenarios, the first comprising 5, and second comprising $[4, 6, 8, 10]$ controllers:

Scenario 1: We realize the connectivity graph depicted in Fig. 1 and gradually inject the link and replica failures as per Fig. 1 b). We inject three link failures at runtime. We then inject a failure in `C2` and eventually recover it so to evaluate the correctness of both failure and recovery detection.

Scenario 2: For the second scenario, we vary the cluster size between 4 and 10 controllers. We inject a failure in a randomly selected replica, and subsequently measure the time required to reach the local and global agreement on the injected failure. After a fixed period, we recover the failed replica so to measure the necessary time to reach the agreement on recovery. Here, we omit link failures so to measure the raw event detection performance in average case.

We repeat the measurements 20 times for each of the 2^4 couplings, and for each parametrization extract the metrics:

1) empirically measured time to reach the *local agreement* on a remote controller's *failure*;
2) empirically measured time to reach the *global agreement* on a remote controller's *failure* and *recovery*;
3) the *average* communication overhead per replica; and
4) analytical worst-case failure detection time (per Eq. 2).

We have used `iptables` to inject communication link failures, i.e., to block communication between replicas and `cpuset` to attach the processes to dedicated CPU cores. Replica failures were enforced by sending `SIGKILL` signal.

VI. Discussion of Evaluation Results

A. Reaching agreement on failure

To demonstrate the correctness of the agreement-enabled consensus, we first evaluate the Scenario 1 case (ref. Fig. 1).

The upper-left subfigure of Fig. 3 depicts the behavior of replicas during the link failure injections for the BEB-D scenario. After the missing signaling heartbeats were observed by the impacted replicas, failure suspicion is triggered for the unreachable neighbors. Since six unidirectional link failures were injected (i.e., three bidirectional link failures), six local suspicions are respectively triggered across the cluster (twice by `C2` and once each by `C1, C3, C4` and `C5`). In the case of RRG-D dissemination (lower-left subfigure), the local failure detector never triggers suspicions in any of the five controllers. This is due to propagation of heartbeats using gossip, where only a subset of all replicas must be directly reachable in order to disseminate the heartbeat consistently to all members.

The suspicions in the local failure detectors are insufficient to agree on the unreachable replicas as inactive, i.e., only direct neighbors on the failed link start suspecting unreachable replicas as inactive. The inactive replicas are correctly marked as such *only after* an actual failure in `C2` (ref. center subfigure of Fig. 3). All active replicas eventually agree on `C2`'s failure.

In the right-most subfigure, we recover `C2` by restarting the replica and depict the time when the local and global agreement on its recovery are reached in the active replicas.

We observe that no replicas ever falsely identify any other replicas as inactive, even when direct connectivity between replicas is unavailable. RAFT's leader election can trigger only whenever global failure agreement is reached, thus solving the issue of oscillating leader.

B. Impact of Event Agreement Algorithm Selection

Based on Scenario 2, we next evaluate the matrix-based M-EA and the list-based L-EA Event Agreement. The results are depicted in Fig. 4. We vary all other parameters apart from those of Event Agreement instances and aggregate the results in the box-plots, hence the large variability in distributions.

M-EA has the advantage of faster agreement on a replica's failure and recovery. The increase of L-EA's multiplier l_M from 2 to 3 showcases the importance of correct parametrization of the selected agreement instance. With the higher multiplier, the probability of detecting false positives with L-EA decreases, at expense of requiring a higher number of matching messages to confirm the status of an observed

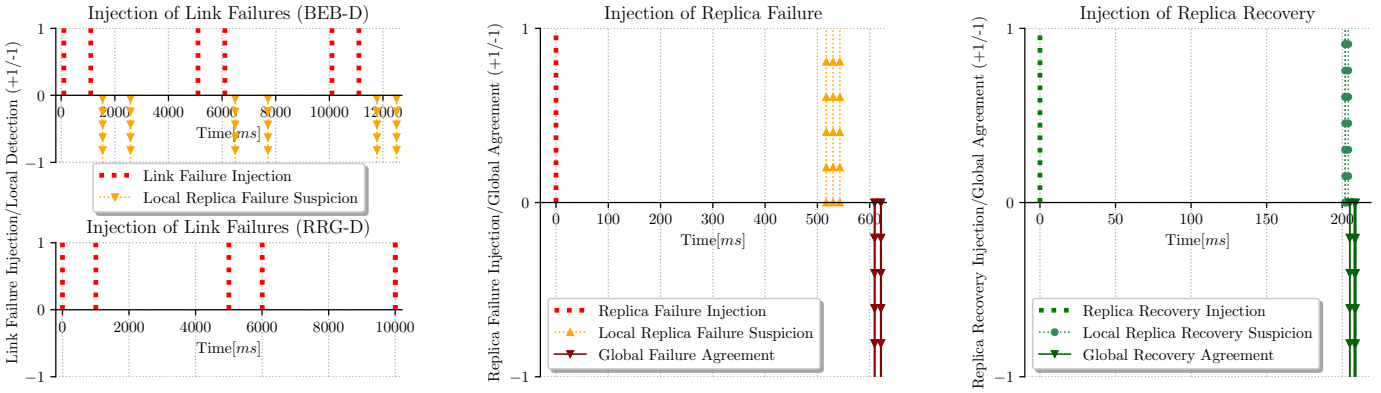

Fig. 3. Link failure injections and the suspicions (left); the `C2` replica failure injection (center) and; replica `C2` recovery injection (right) for evaluation Scenario 1 (ref. Fig. 1). Subfigures on the left denote the distinct Failure Detection behavior for BEB-D and RRG-D implementations. The center subfigure depicts the timepoints where the local and, eventually, global agreement are reached among the active replicas. The subfigure on the right depicts the timepoints where the local and eventually global agreement are reached for the recovered replica `C2`.

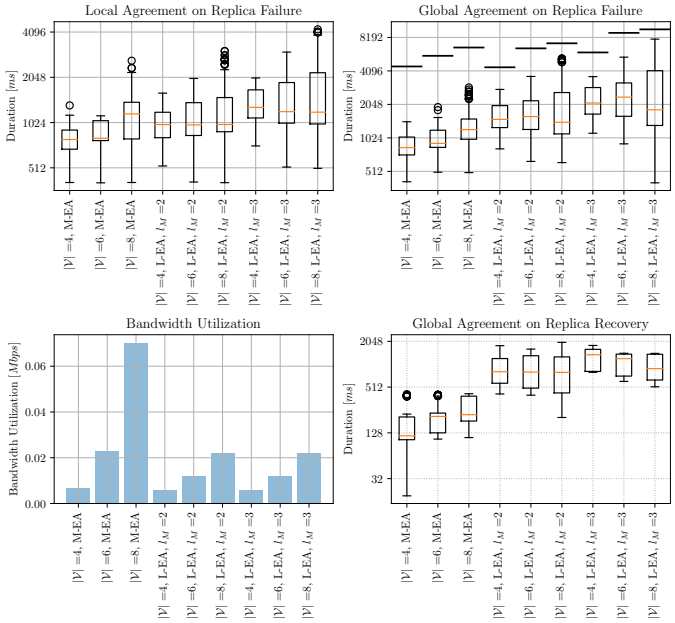

Fig. 4. Impact of selection of the Event Agreement method. Compared to M-EA, L-EA takes longer to converge both for failure and recovery agreement. L-EA, however, offers a lower total communication overhead, computation and memory footprint. Its detection performance scales inversely proportional with l_M multiplier. None of the depicted combinations result in false positives, hence the lower values of l_M are practical and can be tuned further.

replica. In contrast, M-EA does not have this drawback as its agreement requires a single confirmation from other replicas.

Theoretical worst-case agreement time bounds for detecting replica failures (ref. Eq. 2) are depicted as horizontal black lines for each corresponding configuration in the upper-right Fig. 4. The measured empirical detection delays have always stayed below this bound, thus showcasing the correctness and pessimism of the analytic approach.

Notably, we observe that L-EA converges faster to the agreement with the higher number of deployed controllers, but only if BEB-D is used as Dissemination method. M-EA's performance decreases with larger cluster sizes.

The payload size of matrix view grows quadratically with controller cluster size. Compared to L-EA, this makes M-EA less efficient in terms of communication overhead, as can be confirmed by the per-controller loads depicted in Fig. 4.

C. Impact of Failure Detection Selection

Fig. 5 portrays the performance of the adaptive φ-FD and the timeout-based T-FD failure detectors. We have evaluated φ-FD with varying observation window sizes φ_W and suspicion thresholds ϕ, but did not observe important gaps compared to the presented cases. The depicted φ-FD parametrization corresponds to $\varphi_R, \varphi_W, \phi = (150ms, 1500, 15)$.

For T-FD, we varied the timeout threshold t_T. We observe that for networks inducing transmission delays with little variation, T-FD provides similar performance as φ-FD, given a low t_T (i.e., the $t_T = 500ms$ case). For more relaxed parametrizations of t_T, active processes take longer to reach agreement on the updated replica's status. Hence, failure detection agreement time of T-FD is proportional to t_T. The performance of both φ-FD and T-FD suffers for larger clusters.

We note that the advantage of φ-FD lies in its adaptability for networks with large delay variations, as it suppresses false positive detections better than T-FD does. The communication overhead, as well as the time to reach agreement on replica's recovery were not influenced by the failure detector.

D. Impact of Dissemination Selection

Fig. 6 compares the implications of using either BEB-D or RRG-D as the Dissemination method. BEB-D ensures faster agreement for inactive replica discovery, at expense of a larger bandwidth utilization. This is due to the design of RRG-D that propagates the messages in a round-wise manner, thus strictly guaranteeing the convergence time of replicas' states only after the execution of $\lceil log_2(|\mathcal{V}|) \rceil$ rounds (in the *non-failure case*). With BEB-D, however, the time required to agree on replica status scales better with the higher number of controllers.

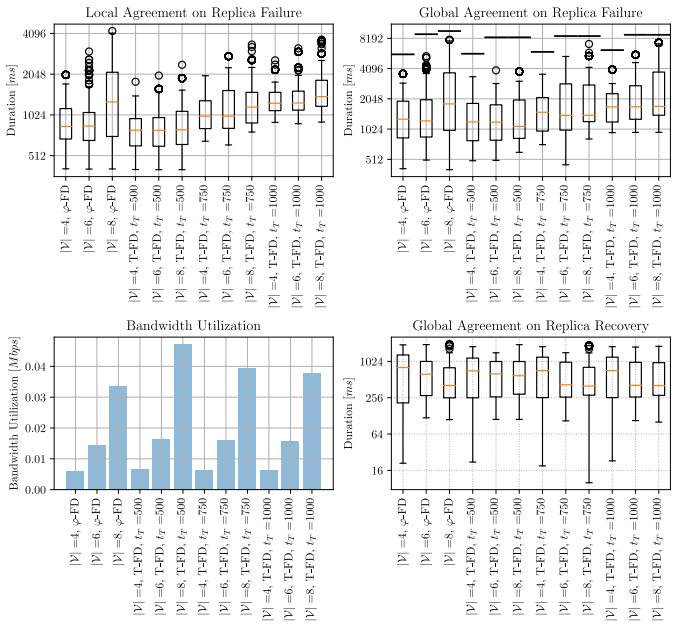

Fig. 5. Impact of selection of the Failure Detection method. T-FD's performance is inversely proportional to the timeout threshold t_T, and given a low t_T, it provides similar performance as φ-FD accrual detector. However, φ-FD is better suited for networks with control channels of high variance latency.

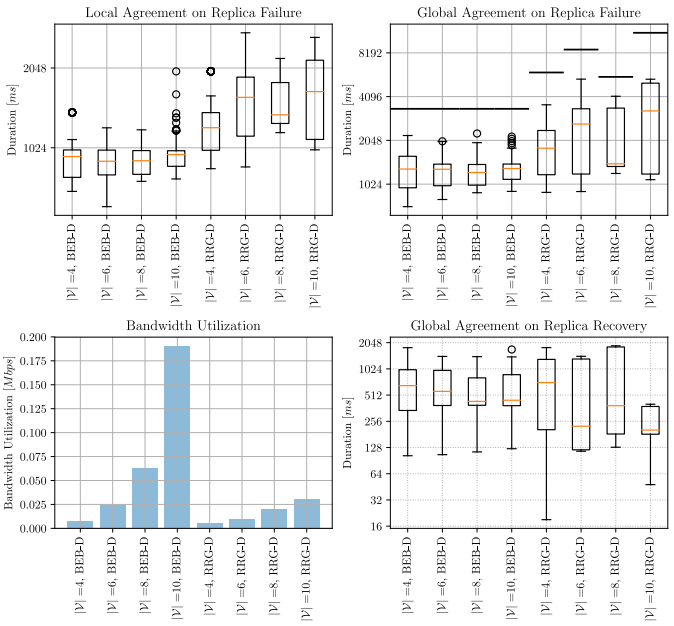

Fig. 6. Impact of Dissemination method selection. The agreement time of gossip-enabled RRG-D lacks compared to the BEB-D broadcast-based dissemination. However, the gossip approach comes with the benefit of a smaller communication overhead. BEB-D scales better with the higher number of cluster members, due to the direct relation between number of rounds required to converge on an update with RRG-D and the size of the cluster.

E. Impact of Signaling Selection

We next vary the signaling methods and the corresponding heartbeat inter-arrival times, but due to space constraints, omit the visual results. We observe no critical advantage of the PR-S over the periodic UH-S design. In fact, PR-S comes at expense of a relatively large communication overhead, as bidirectional confirmations are transmitted on each transmitted heartbeat. For both PR-S and UH-S, we note that the heartbeat periodicity largely impacts the required time to discover and reach agreement on the replica failure and recovery. The highest sending rate (at $100ms$ frequency) shows the best performance for both above metrics, but clearly comes at expense of the largest communication overhead.

VII. Related Work

Hayashibara et al. [19] introduce the φ-FD. While similar in performance to other well-known failure detectors at the time [21], [22], φ-FD was shown to provide a greater tolerance against large delay variance in networks. A minor variant of φ-FD using a threshold-based metric for higher tolerance against message loss was proposed in [23]. OpenDaylight [4] uses Akka Remoting and Clustering for remote replica operation execution as well as its failure detection service. Akka implements the φ-FD [19], hence we focus on the original φ-FD variant in this work as well. Another failure detector type, relying on randomized pinging and distributed failure detection was proposed by Amazon Dynamo [24], but is not investigated here as it cannot guarantee the property of strong completeness in bounded time.

Yang et al. [20] motivate the usage of a matrix-based approach for reaching agreement on SDN controller failures. The authors base their evaluation on a binary round robin gossip variation [25]. We reproduce the same gossip variation as it allows for deterministic estimation of the worst-case convergence time. In contrast to [20] and [25], we also vary the coupling of other components of the event detection, evaluate the failure detection in failure scenarios, provide for its analytic evaluation and extend the matrix-approach with global agreement and replica recovery. Katti et al. [26] introduce a variation of a list-based detector in order to decrease the footprint of matrix-based agreement. They, however, do not provide for a method to converge on replica recovery nor do they provide for analytical bounds on the expected system performance. Suh et al. [27], [28] evaluate the Raft leader re-election time in the scenario of OpenDaylight leader failures. The authors did not consider data-plane (i.e., link/switch) failures or partition occurrence between the cluster members. Similarly, only the built-in variant of OpenDaylight's failure detection (with non-gossip dissemination) is considered there.

In [29], Van Renesse et al. propose one of the earliest failure detectors using gossip dissemination. Katti et al. [26] propose and evaluate a list-based agreement algorithm with random-destination gossip dissemination technique. However, their approach cannot guarantee the upper bound of number of gossip rounds required to converge updates in all replicas.

Recently, implementations of consensus algorithms in networking hardware (e.g., those of Paxos [30], [31], Raft [32] and Byzantine agreement [33], [34]) have started gaining

traction. Dang et al. [30], [31] portray throughput, latency and flexibility benefits of network-supported consensus execution at line speed. We expect to observe similar advantages in event detection time if our framework was to be realized in network accelerators and programmable P4 [35] switches.

VIII. Conclusion and Outlook

We have showcased the limitations of tightly coupled failure detection and consensus processes by reflecting on the example of non-reachable agreement in a Raft-based SDN controller cluster. In contrast to existing works, the proposed failure detection framework considers the possibility of limited knowledge of occurrence of network partitions in the controller replicas. We have solved the leader oscillation issue by introducing agreement as a necessary first step to confirming a particular replica's status, thus effectively ensuring no false positive failure / recovery detection ever arises, independent of the cluster size and the deployed failure detector.

We expect that this work motivates the future evaluations of distributed failure detectors in combination with consensus protocols as a set of loosely coupled but co-dependent modules. Furthermore, we consider partial or full offloading of distributed failure detection to hardware-accelerated data plane as future research. Exposing *event detection as a global service*, as well as enabling faster convergence times (e.g., matrix- / list-based merge procedure in hardware) could lead to a better performing detection and lowered overhead in the end-host applications, compared to the current model where each application implements its detection service independently.

Acknowledgment: This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 780315 SEMIOTICS.

References

[1] P. M. Mohan *et al.*, "Primary-Backup Controller Mapping for Byzantine Fault Tolerance in Software Defined Networks," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–7.

[2] E. Sakic *et al.*, "MORPH: An Adaptive Framework for Efficient and Byzantine Fault-Tolerant SDN Control Plane," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2158–2174, 2018.

[3] H. Li *et al.*, "Byzantine-resilient secure software-defined networks with multiple controllers in cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 4, pp. 436–447, 2014.

[4] J. Medved *et al.*, "Opendaylight: Towards a model-driven SDN controller architecture," in *2014 IEEE 15th International Symposium on*. IEEE, 2014, pp. 1–6.

[5] N. Katta *et al.*, "Ravana: Controller fault-tolerance in software-defined networking," in *Proceedings of the 1st ACM SIGCOMM symposium on software defined networking research*. ACM, 2015, p. 4.

[6] P. Berde *et al.*, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

[7] E. Sakic *et al.*, "Towards adaptive state consistency in distributed SDN control plane," in *Communications (ICC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–7.

[8] Y. Zhang *et al.*, "When Raft Meets SDN: How to Elect a Leader and Reach Consensus in an Unruly Network," in *Proceedings of the First Asia-Pacific Workshop on Networking*. ACM, 2017, pp. 1–7.

[9] E. Sakic *et al.*, "Impact of Adaptive Consistency on Distributed SDN Applications: An Empirical Study," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 12, pp. 2702–2715, 2018.

[10] E. Sakic and W. Kellerer, "Response time and availability study of RAFT consensus in distributed SDN control plane," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 304–318, 2018.

[11] H. Howard *et al.*, "Raft refloated: do we have consensus?" *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 12–21, 2015.

[12] R. Hanmer *et al.*, "Friend or Foe: Strong Consistency vs. Overload in High-Availability Distributed Systems and SDN," in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2018, pp. 59–64.

[13] H. V. Netto *et al.*, "State machine replication in containers managed by Kubernetes," *Journal of Systems Architecture*, vol. 73, pp. 53–59, 2017.

[14] N. Naik, "Applying computational intelligence for enhancing the dependability of multi-cloud systems using Docker swarm," in *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*. IEEE, 2016, pp. 1–7.

[15] D. Ongaro, "Consensus: Bridging theory and practice," Ph.D. dissertation, Stanford University, 2014.

[16] N. McKeown *et al.*, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[17] E. Sakic *et al.*, "Automated bootstrapping of a fault-resilient in-band control plane," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '20. Association for Computing Machinery, 2020, p. 1–13.

[18] E. Sakic *et al.*, "BFT Protocols for Heterogeneous Resource Allocations in Distributed SDN Control Plane," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–7.

[19] N. Hayashibara *et al.*, "The φ accrual failure detector," *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004.*, pp. 66–78, 2004.

[20] T.-W. Yang *et al.*, "Failure detection service with low mistake rates for SDN controllers," in *Network Operations and Management Symposium (APNOMS), 2016 18th Asia-Pacific*. IEEE, 2016, pp. 1–6.

[21] W. Chen *et al.*, "On the quality of service of failure detectors," *IEEE Transactions on computers*, vol. 51, no. 5, pp. 561–580, 2002.

[22] M. Bertier *et al.*, "Implementation and performance evaluation of an adaptable failure detector," in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*. IEEE, 2002, pp. 354–363.

[23] B. Satzger *et al.*, "A new adaptive accrual failure detector for dependable distributed systems," in *Proceedings of the 2007 ACM symposium on Applied computing*. ACM, 2007, pp. 551–555.

[24] G. DeCandia *et al.*, "Dynamo: Amazon's highly available key-value store," in *ACM SIGOPS operating systems review*, vol. 41, no. 6. ACM, 2007, pp. 205–220.

[25] S. Ranganathan *et al.*, "Gossip-style failure detection and distributed consensus for scalable heterogeneous clusters," *Cluster Computing*, vol. 4, no. 3, pp. 197–209, 2001.

[26] A. Katti *et al.*, "Scalable and fault tolerant failure detection and consensus," in *Proceedings of the 22nd European MPI Users' Group Meeting*. ACM, 2015, p. 13.

[27] D. Suh *et al.*, "On performance of OpenDaylight clustering," in *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*. IEEE, 2016, pp. 407–410.

[28] D. Suh *et al.*, "Toward highly available and scalable software defined networks for service providers," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 100–107, 2017.

[29] R. Van Renesse *et al.*, "A gossip-style failure detection service," in *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer-Verlag, 2009.

[30] H. T. Dang *et al.*, "Paxos made switch-y," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 2, pp. 18–24, 2016.

[31] H. T. Dang *et al.*, "Network hardware-accelerated consensus," *arXiv preprint arXiv:1605.05619*, 2016.

[32] Y. Zhang *et al.*, "Network-Assisted Raft Consensus Algorithm," in *Proceedings of the SIGCOMM Posters and Demos*, ser. SIGCOMM Posters and Demos '17. ACM, 2017, pp. 94–96.

[33] E. Sakic *et al.*, "P4BFT: Hardware-Accelerated Byzantine-Resilient Network Control Plane," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–7.

[34] E. Sakic *et al.*, "P4BFT: A Demonstration of Hardware-Accelerated BFT in Fault-Tolerant Network Control Plane," in *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, ser. SIGCOMM Posters and Demos '19. Association for Computing Machinery, 2019, p. 6–8.

[35] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.