PREPRINT

Control under Intermittent Network Partitions

Shaoteng Liu
shaoteng.liu@ri.se
RISE SICS

Rebecca Steinert
rebecca.steinert@ri.se
RISE SICS

Dejan Kostić
dejan.kostic@ri.se
RISE SICS/KTH Royal Institute of Technology

Abstract— We propose a novel distributed leader election algorithm to deal with the controller and control service availability issues in programmable networks, such as Software Defined Networks (SDN) or programmable Radio Access Network (RAN). Our approach can deal with a wide range of network failures, especially intermittent network partitions, where splitting and merging of a network repeatedly occur.

In contrast to traditional leader election algorithms that mainly focus on the (eventual) consensus on one leader, the proposed algorithm aims at optimizing control service availability, stability and reducing the controller state synchronization effort during intermittent network partitioning situations. To this end, we design a new framework that enables dynamic leader election based on real-time estimates acquired from statistical monitoring. With this framework, the proposed leader election algorithm has the capability of being flexibly configured to achieve different optimization objectives, while adapting to various failure patterns. Compared with two existing algorithms, our approach can significantly reduce the synchronization overhead (up to 12x) due to controller state updates, and maintain up to twice more nodes under a controller.

I. INTRODUCTION AND BACKGROUND

The concept of programmable networks based on a logically centralized control plane offers more flexibility, controllability and observability than traditional architectures in this domain. In the context of both wired and radio access networks (RANs), logically centralized control generally provides effective abstractions and techniques for coordinating infrastructure resources [1].

However, despite the advantages of a logically centralized control plane, the control service availability and stability becomes a new problem. When a controller becomes unreachable, portions of the network may lose service. Previous studies indicate that wireless backhaul networks are vulnerable to radio induced interference [2]–[5], which may cause network partitions that isolate portions of the network from the controller. Also, even for networks with wired backhaul, during emergency situations such as an earthquake or hurricane, network partitions could also arise due to link failures, network device crashes, and power supply fluctuations. Moreover, what complicates the situation is that such network partitions may not be permanent, due to the existence of repair mechanisms [2], [3]. Thus, links or network devices may alternate between the crashed and recovery state. Hence, the entire network may undergo repeated partitioning and merging, which is referred to as *intermittent network partition*.

The standard approach to deal with the loss of the controller is using a leader election algorithm to elect a new leader node to resume the control function, which acts as the new controller [6], [7]. Traditionally, a category of leader election algorithms called Ω are utilized in crash-prone distributed systems and

can deal with crash-recovery type of failures. The property of Ω was originally defined in [8]:

Property (Ω): Eventually, if all correct processes can synchronously communicate with each other, every correct process always trusts the same correct process.

However, in intermittent network partitioning situations, when networks can split and merge into partitions of arbitrary sizes at any point in time, the Ω property is hardly satisfiable and not suitable for maintaining stable control service. This situation requires a leader election algorithm that satisfies the following requirements: when the network splits, multiple leaders in different partitions must be elected immediately to guarantee the service availability; when partitions merge, according to the network splitting and merging patterns and intensities, the algorithm should also guide the unification process based on a certain optimization policy. The application of such a leader election algorithm would immediately lead to improved control service stability at a lower cost in terms of synchronization overhead.

To this end, we propose a new leader election algorithm which aims at maintaining stable control services available during intermittent network partitioning situations. The new algorithm can be applied to arbitrary topologies and tolerates arbitrary ways of partitioning. It has a novel framework that enables flexible decision making based on estimates from statistical monitoring. Based on this framework, we implement a configurable decision making unit that adapts the algorithm to the failure patterns. The algorithm can be configured for either optimizing group size, or stability and cost. Compared with existing algorithms, under the same stability requirement, our algorithm can produce larger groups (up to twice the size) with lower synchronization overhead (up to 12x reduction) between leaders and its newly joined group members.

II. RELATED WORK

Prior works that address controller failures in programmable networks are [6], [7], [9], [10]. In [9] and [10], each node keeps querying a data store behind a server to find out which node is the current leader, but in neither contribution it is considered that also the server could fail. Katta et al [6] uses Zookeeper to elect a new leader in the case of failure. Nevertheless, quorum based algorithms such as Zookeeper [11] or Paxos [12] cannot tolerate more than $N/2$ node failures. They can not be applied in the situation when a network splits into partitions of arbitrary sizes, which requires tolerating $N-1$ node failures. Desai et al [7] solves this issue by utilizing an Ω leader election algorithm [8]. As mentioned in Section I, the Ω algorithm [8] is still not a suitable choice.

Fig. 1. Illustration of a network partition.

We also briefly review the relevant leader election algorithms in existing literatures. Works like [13]–[16] extend the original Ω property [8] to tolerate intermittent node failures, but still under the assumption that the communication is eventually synchronous among all the correct processes. Similarly, [17] aims at tolerating intermittent node and omission failures, but requires that eventually a majority of processes remains up and communicate synchronously without omitting messages. In literatures such as [18] and [19], algorithms are proposed to tolerate intermittent link failures. Nevertheless, they assume that all the nodes are completely connected and there is only a limited number of intermittent links. Also, in [20]–[22] and [23], algorithms are designed with considerations about the leader stability. But, they require that eventually at least one node is correct and can synchronously communicate with other nodes.

The above mentioned algorithms can be viewed as extensions of the original Ω algorithm [8]. Generally speaking, their assumptions and requirements do not hold during intermittent network partitioning situations. Thus, their design objectives (e.g., choosing the relatively stable nodes as the leaders [22], [23]) can not be guaranteed. Furthermore, they give no overall considerations about the leader availability, the group size of a leader, the stability and the synchronization efforts, which are essential in maintaining stable control service.

In addition, we want to mention that an old but infrequently used algorithm [24] can work in the intermittent network partition situation and guarantee leader availability in partitions. However, it lacks the ability to adaptively choose leaders based on network failure patterns and intensities.

III. SYSTEM MODEL AND MOTIVATION

A. System model

In a programmable network with logically centralized control, we define the *leader* as the node which runs the control function. A node that is controlled by the leader is referred to as a *follower*. Both the leader and its followers form a *group*. Fig. 1 (left part) shows a group of 6 Radio Transceiver (RT) nodes, with the central node as the leader.

We can describe a programmable network as a graph $G(V = M \cup N, E)$, where V and E denote nodes and links, respectively. Moreover, let N denote the set of nodes holding programmable data forwarding devices (e.g., switches, routers, access points, etc.) and M a candidate set of nodes eligible for hosting controller instances. In this paper, we assume $N = M$ to simplify our discussion. We also assume a node may fail

by stopping and possibly may recover later. We assume that a node has a local clock that can accurately measure intervals of time. The clocks of the nodes might not be synchronized. Additionally, we assume that links are bi-directional and cannot create or alter messages, but are not assumed to be first-in-first-out (FIFO). Concerning synchronous or asynchronous properties, we consider the following two types of links, which are either (a) synchronous or eventually synchronous: after time T , where there is a bound δ on message delays such that if a message is sent at a time $t \geq T$, then this message is received by time $t + \delta$, or (b) intermittent, where the behaviour of a link can alternate between synchronous and lossy. Intermittent links can be caused by link failures due to interference, power failure, or errors in the link maintenance mechanisms [3], [4]. Link failures may later be repaired. According to [2], [3], intermittent links can be modelled as a two-state Markov diagram, where one state represents the link being up, while the other represents the link being down. The failures of an operational link $e \in E$ are modelled according to an exponential distribution with a failure rate parameter λ_e . Similarly, the repair rate is also assumed to be exponentially distributed with a rate parameter μ_e .

Network partition means the graph $G(V, E)$ splits into disconnected sub-graphs of connected nodes, named as partitions, as shown in the right part of Fig. 1. A network partition can appear due to either node failures or link failures. In this work, we mainly focus on the latter more generic case, since the failure of a node can be treated as the case that all the links to and from the node failed simultaneously. Partitions can merge when failed nodes/links have recovered.

Controlling a programmable network can be viewed as a distributed application. Any distributed application can at most satisfy two out of *Consistency* (C), *Availability* (A) and *Partition Tolerance* (P), as the CAP theorem [25]–[27] states. Traditional distributed algorithms or applications, such as Paxos [12] or Zookeeper [11], prioritize C and A. However, regarding the programmable network control service, we think that A and P should be prioritized. It is vital to offer network service availability during scenarios that cause massive failures and partitions, e.g., for the purpose of search and rescue missions during natural catastrophes.

In this paper, we assume a simple leader-follower synchronization model with weak consistency: a leader always asks a newly joined follower to report its current state information. Later on, any change to a follower's state needs its leader's acknowledgement or approval, so that the leader can also update its knowledge base. The failure of such a leader-follower transaction will cause the leader to remove the follower from its group, or the follower to consider its leader is lost.

B. Motivation

Considering a programmable network, nodes inside a group can readily communicate or coordinate with each other (such as building connections, balancing loads and etc.), since the leader of the group can efficiently perform scheduling, optimization and configuration tasks among its group members.

However, cross-group communication or coordination between nodes, even if possible, often requires complicated synchronization/configuration effort among leaders and between leaders and nodes [28]–[30]¹. Thus, unifying nodes into as large groups as possible facilitates the control and configuration tasks of leaders by reducing cross-group messaging overhead, and enhances the capability of the overall control plane in coordinating nodes and other networking resources. Nevertheless, in the situation of intermittent partition failure, group stability and merging cost must also be considered.

Therefore, apart from prioritizing A and P as guided by the CAP theorem, the new leader election algorithm considers the following optimization objectives:

- *Group size* refers to the number of nodes that can co-exist within a group. We use group size as a measure for the *control plane efficiency* in this paper.
- *Group stability* refers to the period of time that a set of nodes can remain in a group. Group stability is important because nodes are required to remain controlled by the same leader for a while in order to complete coordination, configuration or communication tasks.
- *Merging cost* is the cost to join a new leader. A leader usually needs to know the state of its followers in order to perform proper scheduling, optimization and coordination tasks among them. For example, the UEs (user equipments) associated, the channel situation etc., of each follower needs to be synchronized with the leader. Thus, when groups are merging, state information about the newly joined nodes need to be submitted to the leader. Such submitting and updating process often costs additional time and communication effort, which is generally referred to as merging cost.

Ideally, we aim to achieve larger groups with high group stability at low merging cost. In practice we may sometimes need trade-offs between the objectives, since the effort for increasing the group size may cause lowered group stability and increased merging cost. For example, letting nodes with less reliable connections to join a group may actually degrade the overall group stability, possibly leading to additional merging costs as new groups will reform more often.

IV. THE APPROACH AND IMPLEMENTATION

In this section we describe our leader election algorithm, including its new properties, framework and implementation.

A. Properties of the algorithm

Our leader election algorithm satisfies the three properties:

Property 1: (Non-overlapping). At any time, a node can have at most one leader.

Property 2: (Availability). When a node detects that its leader is unreachable, it will have a new leader within a bounded time.

Property 3: (Convergence). If a partition becomes stable, then eventually all the correct nodes in this partition select the same correct node as the leader.

¹Notice that we assume that the network can be controlled with one controller. Scalability issues are beyond the scope of this paper.

Fig. 2. The framework of the new leader election algorithm

The non-overlapping property eliminates the control conflicts due to multiple leaders. The availability property mandates the quick recovery of the control service by providing guarantees on the service continuity. Although our algorithm focuses on the unstable intermittent condition, we still offer guarantees on the stable condition. The convergence property guarantees that if a partition becomes stable, then the group size inside is maximized since all correct nodes eventually join one group. Here, a stable partition is defined as a partition which ceases to split into smaller partition(s) or merge with other partitions. In other words, a stable partition consists of a fixed set of correct nodes that are synchronously connected. In the following sections, we show how we design the algorithm to satisfy the above three properties, and in line with the optimization objectives and considerations in Section III-B.

B. The framework

We propose a new framework for the leader election algorithm, as is depicted in Fig. 2. We consider each node as a Finite State Machine (FSM), where a node can operate in two states, either as a leader or as a follower.

The transitions between the two states can be decided by a component called Local Monitoring and Decision (LMD) module. The module LMD, as its name suggests, has two main functionalities: monitoring the system and making state transition decisions. A follower-to-leader state transition decision is required when a follower node loses its leader; a leader-to-follower decision is required when two or more leaders need to be merged.

In this paper we focus on the LMD implementation on the leader-to-follower transition. We let the follower to leader transition to be directly triggered by a leader crash event in order to reduce the leader election time. In this case, the node directly elects itself as the leader.

C. Overview of the algorithm

In general, the algorithm inside each node consists of three modules: a Failure detector (FD), an LMD and a main body which contains the FSM. We assume that each node has a failure detection (FD) module for detecting whether other nodes are reachable or not. The specific implementation of the FD is independent from the leader election algorithm design and can be chosen arbitrarily (e.g. simple pings, gossip messages [31], etc). We also assume that the FD has high accuracy (i.e. low false positives in crash detections) and periodically (with a period T_{FD}) assesses the (suspected) crash

or recovery of nodes as in [32]. The FD generates events "crash(p)" if a node p is unreachable, and "recover(p)" if a previously unreachable node p becomes reachable.

D. The LMD module implementation

The LMD implements a set of monitoring functions as well as a decision making procedure for directing the group merging behaviours.

1) *Monitoring functions*: The LMD of a node p maintains two metrics: average failure rate and MTBF (Mean time between failure) of every node $q \in V$, based on the reports from its local FD. The failure rate (F_q^p) of a node q observed by p is measured as the number of failures (or repairs) over a fixed length time window (T_{est}). Thus, the $F_q^p(t_n), F_q^p(t_{n-1}) \dots$ is a regularly spaced discrete time series with interval T_{est} , where $F_q^p(t_n)$ denotes the failure rate measured in the n th estimation period. To capture changes and variations, the average failure rate $\bar{F}_q^p(t_n)$ is estimated by using exponentially weighted moving average (EWMA) method [33].

The MTBF is defined as the average up-time between two failure states of a repairable system during operation. Here $MTBF_q^p$ denotes the MTBF of q observed by p . Since failure/recover events of q are stochastic, up-time observations $UP_q^p(t_n), UP_q^p(t_{n-1}) \dots$ (where t_n is the start time of the n th up period) are irregularly spaced time series. The latter is accounted for in the MTBF estimates by the use of inhomogeneous exponentiation moving average (iEMA) method [34].

2) *Decision making procedure*: Based on metrics provided by the monitoring functions, as well as the received group size information periodically broadcasted by each leader, the LMD decides whether a leader gives up its leadership and merges its group with another leader. Such decision is noted as a *merge*.

Suppose the set of leaders that can be merged together is Q . Suppose the group currently controlled by a leader $p \in Q$ is G_p ($|G_p|$ denotes the group size). To make the merge decision, p first computes a gain function $Gain_p(q)$ for each $q \in Q \setminus p$, as Equation (1) describes.

$$Gain_p(q) = \begin{cases} A - R - C, & \text{if } |G_q| \geq |G_p| \\ -1, & \text{otherwise} \end{cases} \quad (1)$$

$$\begin{aligned} A &= cg(1 - \exp(-|G_q|)) \\ R &= cr(1 - \exp(-\frac{2T_{FD}}{MTBF_q^p})) \\ C &= cc(1 - \exp(-|G_p|\bar{F}_p^q T_{est})) \end{aligned}$$

The gain $Gain_p(q)$ takes into account the group size gain A ; the group stability gain R ; and, the merging cost C . $cg, cr, cc \in \mathbb{R}^+$ are the weights of each goal. The importance of each term can be weighed to reflect the operator's requirements such that $cg + cr + cc = 1$. If $Gain_p(q) \geq 0$, then q is a candidate for leader p to merge with. We observe that p never merges to q if $|G_q| < |G_p|$, since the $Gain_p(q)$ is always -1 . This is an effective way for merging cost reduction and stability enhancement, referred to as **bigger-group-win** rule.

If there is more than one leader available for merging, p selects the leader that yields the $\max(Gain_p(q) > 0, \forall q \in Q \setminus p)$. In the case of equal gains, the IDs of the leaders will be used to break the tie (the biggest ID wins). More details are described in lines 37-46 of Algorithm 1. To avoid trivialities, the value of $Gain_p(q)$ is rounded to a fixed precision.

E. The overall algorithm implementation

The pseudo code of the proposed algorithm running in each node is described in Algorithm 1. Essentially the algorithm works as follows: each node p has a *leader* variable to record its current leader. If *leader* == p , the node is a leader. Otherwise, it is a follower. Periodically, each follower checks the liveness of the *leader* with its FD module (line 5-11). A leader also periodically (with period *lePeriod*) sends leader-heartbeat (LHB) messages to its group members (line 19). Received LHB messages at a follower indicates that the leader recognizes the node as a member (line 14-15). If no LHB is received during a period (*flPeriod*) or if its leader crash is detected, the node claims leadership immediately (line 16-18).

If p is the leader, it also periodically sends *Invitation* messages to non-group nodes (line 20). The LMD inside p also periodically (with period *dcPeriod*) checks the received invitations and decides on whether to give up its leadership and select which leader to merge with (line 43-46). If a merge decision is made, then p will become the follower of the new leader, informing its members with message *changeLeader* (line 26-29). Upon receiving message *changeLeader*, a follower of p will also change to the new leader (line 33-36).

To properly run the algorithm, we set $T_{FD} \leq lePeriod < flPeriod$ and $lePeriod \leq dcPeriod$. We assume the communication delay between synchronously connected nodes are smaller than *lePeriod*.

F. Proof of the properties

The proofs of the property 1 and property 2 are straightforward. Since in the algorithm the *leader* variable can at most record one node ID at any given time, then property 1 is proved; since each node immediately elects itself as the new leader upon leader crash detection, the property 2 is also proved. We briefly prove the property 3 (convergence) here:

Lemma 1 Suppose there is a set of L leaders in a stable partition which contains a set of correct nodes P , then for every $p \in P$, their computed gain on any $q \in L$ eventually becomes $Gain_p(q) = A = cg(1 - \exp(-|G_q|))$ (Equation (1)).

Proof For any node $p \in P$, its local MTBF estimation of any other node $q \in L$ eventually approaches ∞ , due to that no crashes and recoveries will be detected by the FD. For the same reason, its local $\bar{F}$ estimation of any other q eventually approaches 0. As a result, stability gain R and merging cost C in Equation (1) always approaches 0. Thus, $Gain_p(q) = A = cg(1 - \exp(-|G_q|))$.

Lemma 2 Suppose at any time t in a stable partition P , the biggest group size equals to $Gmax(t)$. Then, if $Gmax(t) < |P|$, then $\exists \delta t > 0$ that $Gmax(t + \delta t) = |P|$.

Algorithm 1 CODE FOR EACH NODE p :

Uses: FD with events $\langle crash|q \rangle$ and $\langle recover|q \rangle$

On initialization:

- 1: $alives = \emptyset, G_p = \{p\}$
- 2: $leader = p, LHB = True$
- 3: $fmTimer \leftarrow lePeriod$
- 4: **for** $q \in \emptyset$ **do** $invites[q] = [0, 0]$

//general tasks:

- 5: **Upon event** $\langle crash|q \rangle$ **do**
- 6: $alives = alives - \{q\}$
- 7: **if** $leader == p$ **then** $G_p = G_p - \{q\}$
- 8: **if** $q == leader$ **then**
- 9: $leader = p, G_p = \emptyset \cup \{p\}$
- 10: **reset** $fmTimer \leftarrow 0$
- 11: **Upon event** $\langle recover|q \rangle$ **do** $alives = alives \cup \{q\}$
- 12: **Upon event** timeout of $fmTimer$ **do**
- 13: **if** $leader \neq p$ **then**
- 14: **if** $LHB == True$ **then** $LHB = False$
- 15: **reset** $fmTimer \leftarrow flPeriod$
- 16: **else** $leader = p, G_p = \emptyset \cup \{p\}$
- 17: **reset** $decisionTimer \leftarrow dcPeriod$
- 18: **reset** $fmTimer \leftarrow 0$
- 19: **else trigger** $\langle send|G_p, [LHB] \rangle$
- 20: **trigger** $\langle send| \emptyset - G_p, [Invitation, |G_p|] \rangle$
- 21: **reset** $fmTimer \leftarrow lePeriod$

//tasks in leader state:

- 22: **Upon event** $\langle receive|q, [JoinAck, G_q] \rangle$ **do**
- 23: **if** $q \in alives \ \& \ leader == p$ **then** $G_p = G_p \cup G_q$
- 24: **Upon event** $\langle receive|q, [Nack] \rangle$ **do**
- 25: **if** $leader == p$ **then** $G_p = G_p - \{q\}$
- 26: **Upon event** $\langle decision|newLeader \rangle$ **do**
- 27: $leader = newLeader, LHB = False$
- 28: **trigger** $\langle send|G_p, [ChangeLeader, leader] \rangle$
- 29: **trigger** $\langle send|leader, [JoinAck, G_p] \rangle$

//tasks in follower state:

- 30: **Upon event** $\langle receive|q, [LHB] \rangle$ **do**
- 31: **if** $q == leader$ **then** $LHB = True$
- 32: **else if** $q != p$ **then** **trigger** $\langle send|q, [Nack] \rangle$
- 33: **Upon event** $\langle receive|q, [ChangeLeader, newLe] \rangle$ **do**
- 34: **if** $newLe \in alives \ \& \ leader == q$ **then**
- 35: $leader = newLe$
- 36: $fmTimer \leftarrow flPeriod$

//LMD:

- 37: **Upon event** $\langle receive|q, [Invitation, |G_q|] \rangle$ **do**
- 38: $invites[q] = [currentTime, |G_q|]$
- 39: **Upon event** Timeout of $decTimer$ **do**
- 40: $newLe = DECISIONPROC(invites)$
- 41: **if** $newLe \neq \perp$ **then** **trigger** $\langle decision|newLe \rangle$
- 42: **else** $decTimer \leftarrow dcPeriod$
- 43: **procedure** $DECISIONPROC(invites)$
- 44: $valids = \{q | [t, |G_q|] \in invites[q], t - currentTime < lePeriod, \forall q \in alives\}$
- 45: $newLe = le \text{ such that } (Gain_p(le), le) = \max((Gain_p(q) > 0, q), \forall q \in valids)$
- 46: **return** $newLe$

Proof Suppose at a time t , $Gmax(t) < |P|$. Further, suppose that $\exists t'$ that during time interval $(t, t + t')$, $l_b \in L'$ is the leader with the biggest ID among the set of leaders $L' = \{l | |G_l| = Gmax(t)\}$ (L' denotes the leaders that have the group size equal to $Gmax(t)$), then either one of the following two cases apply:

Case 1: If $t' > 2 \times dcPeriod$, there is at least one group

merged to l_b . Stable partition means that an invitation by l_b is guaranteed to reach every other correct nodes in the partition within one decision period ($dcPeriod$). Then, within another decision period, every leader other than l_b has to make its own merge decision. However, they have no reason to refuse l_b 's invitation: l_b has the biggest group size and thus yields the biggest gain according to Lemma 1. Even if another leader l'_b has the same group size, the bigger ID of l_b secures the win. As a result, $Gmax(t + t') > Gmax(t)$.

Case 2: If $t' < 2 \times dcPeriod$, since there is no crash event, only the following two causes could be valid:

- 1) at time $t + t'$, another leader l''_b has $|G_{l''_b}| > |G_{l_b}|$. If this is the cause, we already have $Gmax(t + t') > Gmax(t)$.
- 2) at time $t + t'$, $\exists l''_b$ whose ID is bigger than l_b but has group size equal to $|G_{l_b}|$. In this situation $Gmax(t + t') = Gmax(t)$. However, this situation has finite re-occurrence times, as the number of nodes is limited. Since there are $|P|$ nodes, the maximum re-occurrence time is $|P| - 1$. Combine with $t' < 2 \times dcPeriod$, we have $Gmax(t + 2|P|T_{decision}) > Gmax(t)$.

Therefore, as long as $Gmax(t) < |P|$, $\exists \delta t$ satisfies that $Gmax(t + \delta t) > Gmax(t)$. Lemma 2 is proved. According to Lemma 2, we can eventually get the group size of a stable partition equals to $|P|$, which means that all the nodes joins one leader, hence property 3 is proved.

V. EVALUATION

In this section, we evaluate our algorithm in Mininet [35] with varying test conditions, and compare the results with two reference leader election algorithms. Since we have proven the three properties of our algorithm, the evaluation examines the performance of our algorithm in terms of the three optimization objectives in Section III-B.

A. Experiment set-up

1) *Test settings and metrics*: In a test, we randomly select a portion $fl \in (0, 1]$ of all the links as the intermittent links. For each intermittent link, we independently generate its failure rate $\lambda_e \in (1/450, 1)$ according to a truncated normal distribution $N(u = \bar{\lambda}, \sigma = 1/2\bar{\lambda})$. Similarly, the repair rate μ_e is sampled from a truncated normal distribution $N(u = \bar{\mu}, \sigma = 1/2\bar{\mu})$. Thus, by varying $pl, \bar{\lambda}$ and $\bar{\mu}$, we can control the *failure intensity* of a test scenario. To emulate the changes in the failure patterns, we re-select the intermittent links and re-generate their failure and repair rates after a certain time interval (around 900 s) in a test. Each test lasts 3600 s, which is long enough to get statistical significance. We also repeat each test 10 times. The results showed in the figures are the mean of all the tests.

As mentioned in Section III-B, the group size affects control plane efficiency. To evaluate the group size under a certain stability requirement, we propose a metric $gpSize_p(t, T_{stab})$, which denotes the number of nodes that a node p can stay in a group during the time interval $[t, t + T_{stab}]$. Then we use $ndsInGP(T_{stab})$ to denote the average number of nodes that can remain in a group for at least a period of time T_{stab} . $ndsInGP(T_{stab})$ is calculated as

(a) Group size versus stability in 3 test scenarios.

(b) Merging cost in 3 test scenarios.

Fig. 3. Results in 3 test scenarios with different failure intensities (high to low): (a) $\bar{\lambda} = 1/5, \bar{\mu} = 1/5, fl = 0.7$, (b) $\bar{\lambda} = 1/10, \bar{\mu} = 1/10, fl = 0.7$, (c) $\bar{\lambda} = 1/20, \bar{\mu} = 1/20, fl = 0.5$. In all cases, our algorithm can produce stable groups at lower merging costs compared to the reference algorithms.

$$ndsInGP(T_{stab}) = \frac{\int_0^T \frac{1}{|V|} \sum_{p \in V} gpSize_p(t, T_{stab}) dt}{T} \quad (2)$$

Here, T is the entire simulation time of a test. The merging cost are evaluated as the number of node merges per second.

2) *Algorithm settings and configurations*: The primary parameter in our algorithm is the failure detection period T_{FD} of the FD module. In our evaluation, we set $T_{FD} = 2.0$ s. Other parameters of the algorithm can be set in relation with T_{FD} . We set $lePeriod = T_{FD}$, $flPeriod = 2T_{FD}$ and $dcPeriod$ randomized in $[T_{FD}, 3T_{FD}]$. For the parameters used in statistical monitoring functions, we set $T_{est} = 20T_{FD}$.

To show the flexibility of our algorithm in decision making, we tested two configurations of the decision making procedure (described in Section IV-D2), favouring either group size, or merging cost:

- *large gp.* (group size) configuration: $(cg, cr, cc) = (0.50, 0.25, 0.25)$.
- *Low cost* configuration: $(cg, cr, cc) = (0.33, 0.33, 0.33)$

3) *Reference algorithms*: We implement two algorithms which have been mentioned in Section II, named as *Invitation* algorithm [24] and *Accusation* algorithm [22], [23], respectively. The invitation algorithm uses an a priori agreed node ordering to make leader merge decisions. With the Accusation algorithm, each node counts the number of times it was suspected of having crashed by other nodes, called accusation

(a) Group size versus stability in 3 topologies.

(b) Merging cost in 3 topologies.

Fig. 4. Results in 3 topologies under test scenario $\bar{\lambda} = 1/10, \bar{\mu} = 1/10, fl = 0.7$. Our algorithm, regardless of topology, produces stable groups at lower merging costs compared to the reference algorithms.

time. A node selects its leader among a set of nodes which have the smallest accusation time.

B. Experiment results

Results for four different topologies are shown in Fig. 3 and Fig. 4. Fig. 3 shows the results over a random tree topology with 15 nodes under different test scenarios. Fig. 4 illustrates the results under the test scenario $\bar{\lambda} = 1/10, \bar{\mu} = 1/10, fl = 0.7$ over three real network topologies from [36].

We see that, as the required stability T_{stab} increases, fewer nodes can remain inside a group for a period of time T_{stab} . Nevertheless, in both evaluated configurations, our algorithm can provide stable groups at lower cost compared to the baseline. For example, compared with the reference algorithms, under the same T_{stab} , the large group configuration of our algorithm can generally maintain 5% to 50% more nodes inside a group, with 30% to 50% lower merging cost in all test cases. Also, the low cost configuration can achieve up to 10x and 12x in cost reduction when compared with the Invitation and the Accusation algorithm, respectively. The comparison between different configurations of our algorithm reflects the trade-offs between optimization objectives, as discussed in Section III. The results suggest that at low stability, the large group configuration offers larger group size. However, at high stability, the low cost configuration outperforms the large group configuration. For example, when $T_{stab} = 60$ s, it can achieve a 60% larger group size than the large group configuration. These results indicate that the proposed algorithm offers a significant improvement in group stability to the network operator. It is also configurable with different optimization

goals. More results in other topologies and test scenarios are omitted here due to similar findings.

VI. CONCLUSION

We have proposed an approach to leader election addressing control service availability and stability issues of programmable networks in intermittent network partitioning situations. Our algorithm offers the capability of performing leader election and node clustering in line with tunable optimization objectives related to group stability, size and merging cost, leading to substantially more robust solutions than what is possible with the state-of-the-art. Evaluation results indicate that larger groups (up to twice the size) with the same stability requirement, with up to 12x reduction in merging costs, can be achieved compared to the reference approaches.

Our solution can be practically applied by operators for managing robust networking and control plane services in line with configurable deployment policies. Finally, the overall approach is agnostic to topology and failure models and hence is generically applicable to similar problems.

ACKNOWLEDGMENT

This work was funded in part by the Swedish Foundation for Strategic Research and by the Commission of the European Union in terms of the 5G-PPP COHERENT project (Grant Agreement No. 671639).

REFERENCES

- [1] A. Gudipati, D. Perry, L. E. Li, and S. Katti, "Softran: Software defined radio access network," in *HotSDN*. ACM, 2013, pp. 25–30.
- [2] G. Egeland and E. E. Paal, "The reliability performance of wireless multi-hop networks with apparent link-failures," in *Local Computer Networks (LCN)*. IEEE, 2010, pp. 72–79.
- [3] G. Egeland and P. E. Engelstad, "The availability and reliability of wireless multi-hop networks with stochastic link failures," *Journal on Selected Areas in Communications*, vol. 27, no. 7, pp. 1132–1146, September 2009.
- [4] M. Gerharz, C. de Waal, M. Frank, and P. Martini, "Link stability in mobile wireless ad hoc networks," in *Local Computer Networks*. IEEE, 2002, pp. 30–39.
- [5] F. Tobagi and L. Kleinrock, "Packet switching in radio channels: part ii—the hidden terminal problem in carrier sense multiple-access and the busy-tone solution," *Transactions on communications*, vol. 23, no. 12, pp. 1417–1433, 1975.
- [6] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: Controller fault-tolerance in software-defined networking," in *SIGCOMM Symposium on Software Defined Networking Research*. ACM, 2015, p. 4.
- [7] A. Desai and W. Zheng, "Building reliable and performant software defined networks." [Online]. Available: <http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/>
- [8] T. D. Chandra, V. Hadzilacos, and S. Toueg, "The weakest failure detector for solving consensus," *Journal of the ACM (JACM)*, vol. 43, no. 4, pp. 685–722, 1996.
- [9] F. A. Botelho, F. M. V. Ramos, D. Kreutz, and A. N. Bessani, "On the feasibility of a consistent and fault-tolerant data store for sdn," in *European Workshop on Software Defined Networks*. IEEE, 2013, pp. 38–43.
- [10] F. Botelho, A. Bessani, F. M. Ramos, and P. Ferreira, "On the design of practical fault-tolerant sdn controllers," in *European Workshop on Software Defined Networks*. IEEE, 2014, pp. 73–78.
- [11] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems," in *USENIX ATC*, vol. 8, 2010, p. 9.
- [12] L. Lamport *et al.*, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.
- [13] C. Martín, M. Larrea, and E. Jimenez, "On the implementation of the omega failure detector in the crash-recovery failure model," in *Conference on Availability, Reliability and Security*. IEEE, 2007, pp. 975–982.
- [14] C. Martín and M. Larrea, "Eventual leader election in the crash-recovery failure model," in *Pacific Rim International Symposium on Dependable Computing*. IEEE, 2008, pp. 208–215.
- [15] C. Martín, M. Larrea, and E. Jiménez, "Implementing the omega failure detector in the crash-recovery failure model," *Journal of Computer and System Sciences*, vol. 75, no. 3, pp. 178–189, 2009.
- [16] M. Larrea, C. Martín, and I. Soraluze, "Communication-efficient leader election in crash-recovery systems," *Journal of Systems and Software*, vol. 84, no. 12, pp. 2186–2195, 2011.
- [17] C. Fernández-Campusano, M. Larrea, R. Cortinas, and M. Raynal, "Eventual leader election despite crash-recovery and omission failures," in *Pacific Rim International Symposium on Dependable Computing*. IEEE, 2015, pp. 209–214.
- [18] G. Singh, "Leader election in the presence of link failures," *Transactions on Parallel and Distributed Systems*, vol. 7, no. 3, pp. 231–236, 1996.
- [19] H. Abu-Amara and J. Lokre, "Election in asynchronous complete networks with intermittent link failures," *Transactions on Computers*, vol. 43, no. 7, pp. 778–788, 1994.
- [20] M. K. Aguilera, W. Chen, and S. Toueg, "Failure detection and consensus in the crash-recovery model," *Distributed computing*, vol. 13, no. 2, pp. 99–125, 2000.
- [21] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg, "On implementing omega with weak reliability and synchrony assumptions," in *Symposium on Principles of distributed computing*. ACM, 2003, pp. 306–314.
- [22] —, "On implementing omega in systems with weak reliability and synchrony assumptions," *Distributed Computing*, vol. 21, no. 4, pp. 285–314, 2008.
- [23] N. Schiper and S. Toueg, "A robust and lightweight stable leader election service for dynamic systems," in *Dependable Systems and Networks With FTCS and DCC (DSN)*. IEEE, 2008, pp. 207–216.
- [24] H. Garcia-Molina, "Elections in a distributed computing system," *Transactions on Computers*, vol. 100, no. 1, pp. 48–59, 1982.
- [25] E. A. Brewer, "Towards robust distributed systems," in *PODC*, vol. 7, 2000.
- [26] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51–59, 2002.
- [27] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker, "Cap for networks," in *HotSDN*. ACM, 2013, pp. 91–96.
- [28] A. Bianco, P. Giaccone, R. Mashayekhi, M. Ullio, and V. Vercellone, "Scalability of onos reactive forwarding applications in isp networks," *Computer Communications*, vol. 102, pp. 130–138, 2017.
- [29] A. S. Muqaddas, A. Bianco, P. Giaccone, and G. Maier, "Inter-controller traffic in onos clusters for sdn networks," in *Proc. ICC*. IEEE, 2016, pp. 1–6.
- [30] S. Aurox and H. Karl, "Flow processing-aware controller placement in wireless densenets," in *Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC)*. IEEE, 2014, pp. 1294–1299.
- [31] R. Van Renesse, Y. Minsky, and M. Hayden, "A gossip-style failure detection service," in *Middleware98*. Springer, 1998, pp. 55–70.
- [32] X. Défago, P. Urbán, N. Hayashibara, and T. Katayama, "Definition and specification of accrual failure detectors," in *International Conference on Dependable Systems and Networks*. IEEE, 2005, pp. 206–215.
- [33] A. Ingolfsson and E. Sachs, "Stability and sensitivity of an ewma controller," *University of Alberta School of Business Research Paper*, no. 2013-174, 1992.
- [34] G. Zumbach and U. Müller, "Operators on inhomogeneous time series," *International Journal of Theoretical and Applied Finance*, vol. 4, no. 01, pp. 147–177, 2001.
- [35] "Mininet network emulator." [Online]. Available: <http://mininet.org/>
- [36] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.