When Raft Meets SDN: How to Elect a Leader over a Network

Kostas Choumas and Thanasis Korakis
Dept. of ECE, University of Thessaly, Volos, Greece
Email: kohoumas, korakis@uth.gr

***Abstract*—This paper discusses the benefits in the operation of a Raft based SDN controller cluster, when the election of the cluster leader becomes more or less "fair". Raft is a leader based consensus algorithm, which is used by the most popular open-source SDN controllers for replicating the network state. It requires all state changes to be confirmed by the leader, thus the leader election is very crucial for the Raft performance. In case that the inter-controller communication delay is the same for all controller pairs, the election process is absolute fair, meaning that the leadership is shared equally among the controllers. In all other cases, some controllers become leaders more frequently in benefit or at a cost of the average time required for a network state update. In this paper, we model this time as a function of the leadership probabilities of the cluster controllers. We also model these probabilities as a function of the time that each controller is waiting after detecting the current leader failure and before starting its campaign. We configure different ranges for the controller waiting times, adjusting the leadership probabilities and decreasing the average response time. Our model is confirmed by testbed experimentation.**

***Index Terms*—SDN, Raft, clustering, testbed experimentation**

I. Introduction

Software Defined Networking (SDN) is a technology that decouples the control and data planes, using servers for controlling the operation of the forwarding datapaths or switches. This is why these servers are called controllers. At first, there was only one controller for all datapaths of a network, which was not scalable. Soon, multiple controllers were introduced to cooperate over the same network, consisting a cluster. Each controller is responsible for a subset of the network datapaths and should communicate with all other controllers, in order for the whole cluster to share the same network state.

The most recent and well-known protocol for reaching consensus among the controllers is Raft [1]. Raft enhances Paxos, which is assumed as the "gold standard" algorithm in distributed consensus [2]. The most popular open-source SDN controllers, namely OpenDaylight (ODL) [3] and Open Network Operating System (ONOS) [4], utilize Raft for strong consistency. According to Raft, all network updates have to be processed by the cluster leader, even if it is not controlling any datapath related to these updates. Thus, the average time needed for an update is affected by the result of the leader election process.

According to this process, each controller campains for the leadership when it realizes that there is no existing leader, and after waiting for a random time interval. This interval is called election timeout and the controller with the lowest one, assuming that multiple controllers campaign together, probably wins. Raft designers propose the selection of this timeout from the same range for all controllers, which results to identical leadership probabilities, when all inter-controller communication delays are equal. However, these delays are unequal in most cases.

In this paper, i) we model how *the leadership probabilities of the controllers can be adjusted by choosing different ranges for their election timeouts*. The average time needed for a network update can be decreased, by either increasing the leadership probabilities of the most central controllers or the ones receiving more updates. For example, if all controllers receive equal number of network updates, the average response time benefits from a central controller as a leader, whereas if one controller receives the great majority of updates, then this one should be the leader. As a proof of concept, ii) we also show theoretically and experimentally, how we can increase the leadership probability of the central node or share the leadership equally among all controllers in two bus topologies.

The paper is divided in the following Sections. Section II presents related work. Section III highlights insights into Raft, whereas Section IV models Raft performance over a network. Section V shows our experimentation results and Section VI concludes the paper.

II. Related Work

The utilization of multiple controllers was initially proposed by Heller [5], presenting the controller placement problem. In this work, heuristics are given for placing the controllers in a network, focusing on the controller-to-switch communication. There have been numerous publications for improving the controller placement, taking also care of the inter-controller traffic, such as [6]. However, most of these works assume that all controllers have the same role, in contrast to the leader based structure of Raft. Raft is exploited by ODL and the ODL controller placement problem is studied in [7], aiming at minimizing the total control traffic.

In [8], an adaptive consistency model is introduced, which employs concepts of eventual consistency along with a 'cost-based' approach, where the strict Raft synchronisation is employed for critical operations. In this way, they improve the perceived reactivity of the controllers. On the other hand, in [9], authors aim at the same goal by requiring only strong consistency and considering both the delay of the controller-to-switch and inter-controller communications for proposing Raft based controller placement. They build a model for estimating the reaction time perceived at the datapaths, however, they assume a fixed leader (no failures and leader transitions). In [10], authors present the inter-dependency between the controller cluster and the datapath network, as well as the problems caused by this. In [11], Raft performance is evaluated through Stochastic Activity Networks (SAN), in terms of response time and availability. The election timeouts are also modified in [12] for the purpose of reducing the election duration.

III. Insights into Raft

Let's assume a cluster of nodes using the Raft protocol for reaching consensus. Each node is a state machine and all nodes compute identical copies (replicas) of the same state. In this way, they are able to continue operating even if some of them are down. Replicated state machines are typically implemented using a replicated log, which contains a series of commands that have to be executed by each state machine in order. Keeping the replicated log consistent is the job of the Raft algorithm. When OpenFlow controllers use Raft for their clustering, the replicated log is the set of OpenFlow messages coming to the controllers and affecting their state.

Raft operation is based on the election of a node as leader, which is responsible for managing the replicated log. The leader has to be chosen when a cluster starts or when an existing leader fails. At any given time, each node is in one of the three states: *leader*, *follower* or *candidate*. Between the elections, there is exactly one leader and all of the other nodes are followers, while during the elections there is no leader and some nodes become candidates. Followers are passive, meaning that they do not issue requests on their own but simply respond to requests from leaders and candidates. The *leader handles all incoming commands* (a command applied to a follower is redirected to the leader). Moreover, the leader replicates these commands across the cluster, forcing each follower log to agree with its own. Now, let's see in more detail the leader election and the log replication processes.

A. Leader Election

Raft divides time into *terms* of arbitrary length, numbered with consecutive integers, which begin and end with an election. In each election, one or more candidates attempt to become the leader requesting votes from *more than half* of the nodes. In the case of a *split vote*, none of the candidates collect the necessary number of votes, thus a new term with a new election begins.

Raft uses a heartbeat mechanism to trigger the leader election. At the initial phase, all nodes begin as followers. A node remains follower as long as it receives requests from a leader or a candidate. The leader sends periodic *heartbeat requests* to all followers, with a period equal to the *heartbeat timeout*, which is slightly higher than the average time it takes a node to send requests in parallel to all cluster nodes and receive their responses. If a follower receives no communication over a period of time called *election timeout*, then it assumes there is no viable leader and begins an election for a new leader. In particular, it transitions to candidate, votes for itself and issues *vote requests* in parallel to all other nodes. A candidate remains in this state until one of the following conditions are met: (a) it wins the leadership being voted by at least half of the cluster, (b) another node establishes itself as leader and this candidate transits to follower, or (c) another election timeout goes by with no winner because of a split vote.

When the third possible outcome happens, each candidate times out and starts a new election by initiating another round of vote requests. Assuming a fixed election timeout for all nodes, this process could be repeated indefinitely (e.g. all cluster nodes become candidates simultaneously and vote for themselves repeatedly). Thus, Raft uses *randomized* election timeouts to ensure that split votes are rare and resolved quickly. The selection of the range of the randomized election timeout is critical. It has to be an order of magnitude higher than the heartbeat timeout, but as low as possible. The lower bound of the election timeout enables the leader to reliably send the heartbeat messages, while the higher bound is for preventing long periods of unavailability, since there is no leader to handle the commands during the elections.

B. Log Replication

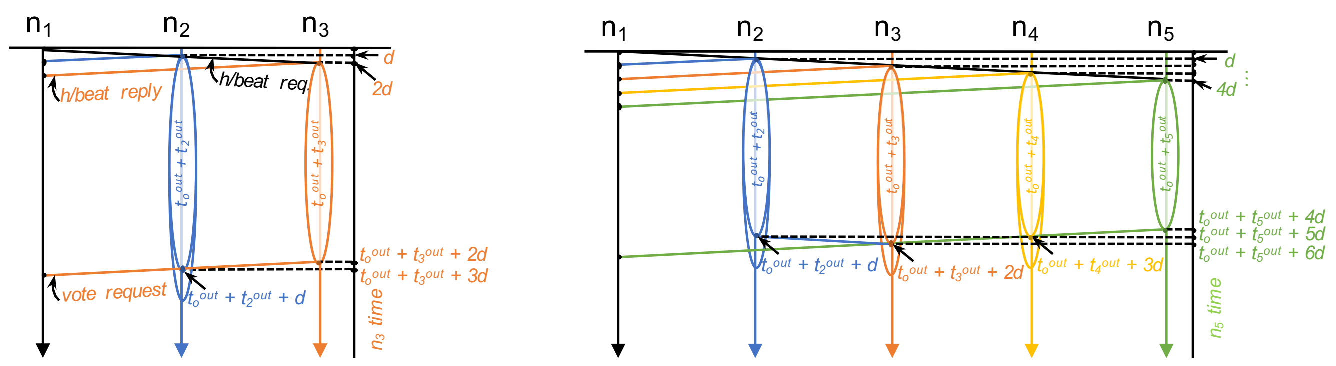
Once a leader has been elected, it begins servicing the incoming commands. Each command sent to the leader is appended to its log as a new entry. If it is sent to a follower, it is redirected to the leader and the same process is followed. The leader issues *append requests* in parallel to all other nodes to replicate this entry. When the leader gets the replies from a *majority* of the nodes, the log entry is applied to the leader's state machine and considered as *commited*. Subsequently, the leader resends append requests to the followers for applying the entry to their state machines.

IV. Raft over a Network

Raft designers assume that the cluster nodes communicate over a mesh network with links featuring equal delays. However, this is not the case for most real networks, where different node pairs exhibit different communication delays. As follows, some nodes collect faster the required votes from a quorum. Although the election of such nodes facilitates a faster communication between the leader and the majority of the followers, it also imposes several considerations if they are the best choice for leaders, in terms of the average response time. Let's see this challenge in the following toy example.

A. Bus network with 3 nodes

Let's assume the simplest cluster $\mathcal{N}$ with $N = 3$ interconnected nodes n_1, n_2 and n_3 shaping a bus network, where n_2 is located in the middle with equal distance to n_1 and n_3, as it is depicted in Figure 1(a). The delay of the link connecting two nodes is proportional to the distance between them. Each node n_i has its election timeout $t_o^{\text{out}} + t_i^{\text{out}}$, where t_o^{out} is fixed and t_i^{out} is chosen from the standard uniform distribution, $\forall i \in \{1, 2, 3\}$, with Probability Density Function (PDF) $f(\tau) = 1$ for $\tau \in [0, 1]$ and 0 otherwise. When n_1 is the leader, the delays to the followers n_2 and n_3 are d and $2d$ respectively. Assuming that heartbeat requests are successfully sent until a n_1 failure, the last heartbeat arrives later at n_3 compared to n_2, thus n_3 starts counting down to trigger a new election a time interval d after n_2 does. The winner of the next election is n_3 if and


n1
n2
n3
h/beat reply
h/beat req.
d
2d
$t_o^{out} + t_2^{out}$
$t_o^{out} + t_3^{out}$
$t_o^{out} + t_3^{out} + 2d$
$t_o^{out} + t_3^{out} + 3d$
vote request
$t_o^{out} + t_2^{out} + d$
n3 time
n1
n2
n3
n4
n5
d
4d
$t_o^{out} + t_4^{out}$
$t_o^{out} + t_5^{out}$
$t_o^{out} + t_5^{out} + 4d$
$t_o^{out} + t_5^{out} + 5d$
$t_o^{out} + t_5^{out} + 6d$
$t_o^{out} + t_2^{out} + d$
$t_o^{out} + t_3^{out} + 2d$
$t_o^{out} + t_4^{out} + 3d$
n5 time

(a) Bus network with 3 nodes.

(b) Bus network with 5 nodes.

Fig. 1. Leadership transition over bus networks with 3 or 5 nodes. **(a)** n_3 becomes leader after n_1, if $t_o^{\text{out}}+t_3^{\text{out}}+3d < t_o^{\text{out}}+t_2^{\text{out}}+d \Rightarrow t_3^{\text{out}}+2d < t_2^{\text{out}}$. **(b)** n_5 becomes leader after n_1, if $t_5^{\text{out}}+5d < t_2^{\text{out}}+d \Rightarrow t_5^{\text{out}}+4d < t_2^{\text{out}}$, $t_5^{\text{out}}+6d < t_3^{\text{out}}+2d \Rightarrow t_5^{\text{out}}+4d < t_3^{\text{out}}$ and $t_5^{\text{out}}+5d < t_4^{\text{out}}+3d \Rightarrow t_5^{\text{out}}+2d < t_4^{\text{out}}$.

only if its vote request is received by n_2, before n_2 finishes its countdown. Otherwise, n_2 (that has come back from its instant failure) votes for itself and gets the vote of n_1, which is closer to n_2. This happens if $t_3^{\text{out}}+2d < t_2^{\text{out}}$, since n_3 starts its countdown after d and the vote request is received by n_2 a time interval d after n_3 sends it.

Figure 1(a) illustrates this information. If $2d \leq 1$, the probability of this event is

$$p = \Pr[t_3^{\text{out}}+2d < t_2^{\text{out}}] = \int_{2d}^{1}\int_{0}^{\tau_2-2d} f(\tau_3)f(\tau_2)d\tau_3 d\tau_2$$
$$= \int_{2d}^{1}\int_{0}^{\tau_2-2d} 1\ d\tau_3 d\tau_2 = \frac{1}{2}(1-2d)^2. \qquad (1)$$

Modeling the transition from one leader to a new one using a Markov chain, we get the transition probability matrix $\mathbf{P} = [p_{li} : \forall n_l, n_i \in \mathcal{N}]$, where $p_{11} = p_{22} = p_{33} = 0$, $p_{13} = p_{31} = p$ and $p_{21} = p_{23} = 1/2$ (p_{21} and p_{23} are equal, since n_1 and n_3 are symmetrical in relation to n_2). Finally, the steady-state probabilities are $\pi = [\pi_1, \pi_2, \pi_3]$, where $\pi_1 = \pi_3 = (1+p)/(4+2p(1-p))$ and $\pi_2 = (1-p^2)/(2+p(1-p))$. After multiple *instant failures* of each leader (meaning that leader is down for a period almost equal to the average election timeout), n_i is the new leader with probability π_i, $\forall i \in \{1,2,3\}$. In case that $d = 0$, then obviously $p = 1/2$ and $\pi_1 = \pi_2 = \pi_3 = 1/3$.

Commands sent to the nodes are replicated and commited at different times, depending on the receiver node and its state. Let's compute the time t_{il}^{rep} required for a command to be sent to n_i and n_i inform back that it is commited, when n_l is the leader. If n_1 is both the command receiver and the leader, it needs time $t_{11}^{rep} = 2d$ to send the append requests and receive the reply from n_2. The reply from n_3 is useless, since n_1 and n_2 consist a cluster majority. The same time $t_{22}^{rep} = t_{33}^{rep} = 2d$ is needed for n_2 and n_3 respectively. On the other hand, if the command goes to n_1 but it is not leader, the corresponding time is either $t_{12}^{rep} = t_{22}^{rep} + 2d = 4d$ or $t_{13}^{rep} = t_{33}^{rep} + 4d = 6d$, when n_2 or n_3 is leader respectively. This time is actually the sum of the time required by the leader to handle the command plus the time needed for n_1 to redirect the command to the leader and get the commitment confirmation back. Similarly, $t_{32}^{rep} = 4d$ and $t_{31}^{rep} = 6d$, while $t_{21}^{rep} = t_{23}^{rep} = t_{22}^{rep} + 2d = 4d$. After multiple failures, the average response time for each n_i is $t_i^{rep} = \sum_{n_l \in \mathcal{N}} \pi_l t_{il}^{rep}$, and over all nodes is $t^{rep} = \sum_{n_i \in \mathcal{N}} \lambda_i t_i^{rep}$, where λ_i is the rate of commands to n_i ($\sum_{n_i \in \mathcal{N}} \lambda_i = 1$).

At this point, we show how the election timeouts can be configured for changing the leadership probabilities π and the average response time t^{rep}. One option in to select t_1^{out} and t_3^{out} from a uniform distribution in the range $[0, \alpha]$, with PDF $f(\tau/\alpha)$, while $\alpha < 1$ for increased probabilities of nodes n_1 and n_3 to be leaders, since they will most probably start earlier their campaign. If $\alpha \leq 1-2d$, then probability p of Eq. (1) changes to

$$p = \int_{2d}^{1}\int_{0}^{\tau_2-2d} f(\tau_3/\alpha)f(\tau_2)d\tau_3 d\tau_2$$
$$= \int_{2d}^{1}\int_{0}^{\min\{\tau_2-2d,\alpha\}} \frac{1}{\alpha}\ d\tau_3 d\tau_2 = 1-2d-\frac{\alpha}{2}$$

and $\pi_1 = \pi_3 = 1/(2+4d+\alpha)$. In case that $4d < 1$ and $\alpha = 1-4d$, then the nodes have *equal leadership probabilities*. Another option is to change the PDF of n_2 to $f_2(\tau/\alpha_2)$. Assuming that $2d \leq \alpha_2 < 1$, then Eq. (1) changes to

$$p = \int_{2d}^{\alpha_2}\int_{0}^{\tau_2-2d} f(\tau_3)f(\tau_2/\alpha_2)d\tau_3 d\tau_2$$
$$= \int_{2d}^{\alpha_2}\int_{0}^{\tau_2-2d} \frac{1}{\alpha_2}\ d\tau_3 d\tau_2 = \frac{1}{2\alpha_2}(\alpha_2-2d)^2,$$

which is an increasing function of α_2. Moreover, π_2 is a decreasing function of p, thus π_2 decreases with α_2. In this way, *the leadership probability of the central node π_2 can be increased*, by decreasing α_2.

Depending on the λ_i rates, the optimal choice for the ranges of the election timeouts for minimum response time t^{rep}, can be one of the two presented options or something completely different. Now, let's proceed with a generalized network.

B. Generalized Network

Now, the cluster has N nodes $n_i \in \mathcal{N}$, $\forall i \in \{1,2,\ldots,N\}$, with election timeouts $t_o^{\text{out}} + t_i^{\text{out}}$ chosen randomly as before with PDF $f(\tau/\alpha_i)$. We also denote as $F(\tau)$ the Cumulative

Distribution Function (CDF) of the standard uniform distribution, which is $F(\tau)=\tau$ for $\tau\in[0,1]$, 0 for $\tau<0$ and 1 otherwise. The delay between two nodes $n_i,n_j\in\mathcal{N}$ is d_{ij}, with $d_{ii}=0$. We assume instant failures again, which means that previous leader votes in the election for the new leader. In the event that $n_l\in\mathcal{N}$ is the leader and fails, both $n_i\in\mathcal{N}-\{n_l\}$ and every other follower $n_z\in\mathcal{N}-\{n_i,n_l\}$ start their timeouts after receiving the last heartbeat from n_l, which is after delay d_{li} and d_{lz} respectively. Thus, n_i becomes candidate and votes for itself if its timeout plus d_{li} is less than the timeout of every other follower n_z plus d_{lz} plus the time needed for the vote request of n_z to be received by n_i, which is valid if $t_i^{\text{out}}+d_{li}<t_z^{\text{out}}+d_{lz}+d_{zi},\forall n_z\in\mathcal{N}-\{n_i,n_l\}$. Otherwise, n_i remains follower voting for one of the n_z sent the vote request earlier. Moreover, n_i is voted by $n_j\in\mathcal{N}-\{n_i\}$ if the n_i's vote request is received earlier by n_j than any other vote request (from all other followers n_z and n_j itself), which requires that $t_i^{\text{out}}+d_{li}+d_{ij}<t_z^{\text{out}}+d_{lz}+d_{zj},\forall n_z\in\mathcal{N}-\{n_i,n_l\}$.

Heartbeat timeout is slightly higher than $2\max_{n_i,n_j\in\mathcal{N}}\{d_{ij}\}$ and should be an order of magnitude less than the minimum of the average election timeouts of all nodes, which is $t_o^{\text{out}}+\min_{n_i\in\mathcal{N}}\alpha_i/2$. Assuming this, the probability of a node n_i to be candidate after the fall of leader n_l and be voted by at least all nodes $n_j\in\mathcal{N}'\subseteq\mathcal{N}-\{n_i\}$ and itself is equal to

$$\begin{aligned}p_{li}^{\mathcal{N}'}&=\Pr[\{n_i\text{ is at least voted by }n_i\text{ and }\mathcal{N}'\}]\\&=\Pr\Big[\bigcap_{n_z\in\mathcal{N}-\{n_i,n_l\}}\{t_z^{\text{out}}>t_i^{\text{out}}+d_{liz}^{\mathcal{N}'}\}\Big]\\&=\int_0^{\alpha_i}f\Big(\frac{\tau_i}{\alpha_i}\Big)\Big(\prod_{n_z\in\mathcal{N}-\{n_i,n_l\}}\int_{\tau_i+d_{liz}^{\mathcal{N}'}}^{\infty}f\Big(\frac{\tau_z}{\alpha_z}\Big)d\tau_z\Big)d\tau_i\\&=\frac{1}{\alpha_i}\int_0^{\alpha_i}\prod_{n_z\in\mathcal{N}-\{n_i,n_l\}}\Big(1-F\Big(\frac{\tau_i+d_{liz}^{\mathcal{N}'}}{\alpha_z}\Big)\Big)d\tau_i,\end{aligned}\quad(2)$$

where $d_{liz}^{\mathcal{N}'}=d_{li}-d_{lz}+\max_{n_j\in\mathcal{N}'\cup\{n_i\}}\{d_{ij}-d_{zj}\}$. This is enough for n_i to be the new leader, if $\mathcal{N}'\in\mathcal{N}^{-n_i}$, assuming that $\mathcal{N}^{-n_i}$ is the family of sets over $\mathcal{N}-\{n_i\}$ that have cardinality equal to $\lceil(N-1)/2\rceil$. This means that n_i has to be voted by a cluster majority including itself. As follows, the probability of n_i to be the next leader after n_l is

$$p_{li}=\Pr\Big[\bigcup_{\mathcal{N}'\in\mathcal{N}^{-n_i}}\{n_i\text{ is at least voted by }n_i\text{ and }\mathcal{N}'\}\Big].\quad(3)$$

Finally, there is a low probability that election ends with a split vote, in which case the new leader is the same with before, thus $p_{ll}=1-\sum_{n_i\in\mathcal{N}-\{n_l\}}p_{li}$. Then, the transition probability matrix is $\mathbf{P}=[p_{li}:\forall n_l,n_i\in\mathcal{N}]$ and the steady-state probabilities π is an eigenvector of this matrix with eigenvalue equal to one.

The response time of the command send to n_i, when n_l is the leader, is given by $t_{il}^{rep}=2(d_{il}+d_{l*}),\forall n_i,n_l\in\mathcal{N}$. Furthermore, $d_{l*}=d_{lz}$, where n_z is the one with rank equal to $\lceil(N-1)/2\rceil$, when all nodes are increasingly ordered based on their delay to n_l. Actually, n_z is the farthest from n_l that has to reply to its append request in order n_l to proceed with the commitment. The average response times $t_i^{rep}=\sum_{n_l\in\mathcal{N}}\pi_l t_{il}^{rep}$, $\forall n_i\in\mathcal{N}$ and $t^{rep}=\sum_{n_i\in\mathcal{N}}\lambda_i t_i^{rep}$ are given as before.

Now, lets see how these equations are specified in case of a bus network with 5 nodes.

C. Bus network with 5 nodes

In this case, $\mathcal{N}=\{n_1,n_2,\dots,n_5\}$ and $N=5$, where n_i is before n_{i+1} and the delay between them is d, $\forall i\in\{1,2,\dots,4\}$, as depicted in Figure 1(b). We assume that the fixed part of election timeout is equal to the maximum value that random part can take (as it happens usually in the Raft implementations), thus $t_o^{\text{out}}=1$. Heartbeat timeout is $2\max_{n_i,n_j\in\mathcal{N}}\{d_{ij}\}=8d$, which should be an order of magnitude less than the minimum of the average election timeouts of all nodes, which is no less than 1. Based on these, we safely choose $d\in[0,0.03]$.

Let's estimate the probability that n_5 is the next leader after n_1. This happens only if n_5 is at least voted by its two closest nodes, $\mathcal{N}'=\{n_3,n_4\}$. There is no way n_5 to be voted by n_2 or n_1 and not by $\mathcal{N}'$. From Eq. (3) and Eq. (2), we get

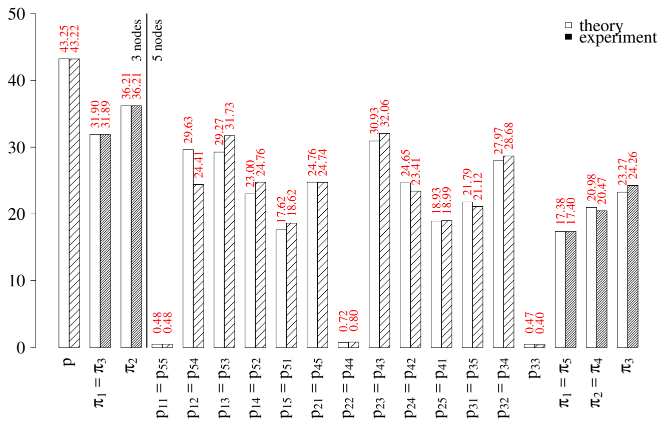
$$\begin{aligned}p_{15}&=\Pr[\{n_5\text{ is at least voted by }n_5\text{ and }\mathcal{N}'\}]=\\&=\frac{1}{\alpha_5}\int_0^{\alpha_5}\Big(1-F\Big(\frac{\tau_5+4d}{\alpha_2}\Big)\Big)\Big(1-F\Big(\frac{\tau_5+4d}{\alpha_3}\Big)\Big)\\&\qquad\Big(1-F\Big(\frac{\tau_5+2d}{\alpha_4}\Big)\Big)d\tau_5,\end{aligned}$$

since $d_{152}^{\mathcal{N}'}=4d$, $d_{153}^{\mathcal{N}'}=4d$ and $d_{154}^{\mathcal{N}'}=2d$ (which are also confirmed in Figure 1(b)). All transition probabilities can be estimated using the same formula, while the leadership probabilities are given as an eigenvector of the transition probability matrix $\mathbf{P}$. According to our model, for $d\in[0,0.03]$ and $\alpha_3=1$, leadership is equally shared if $\alpha_2=\alpha_4\simeq1-3.06d$ and $\alpha_1=\alpha_5\simeq1-10.89d$. In Section V, we confirm experimentally that leadership is shared equally under these ranges.

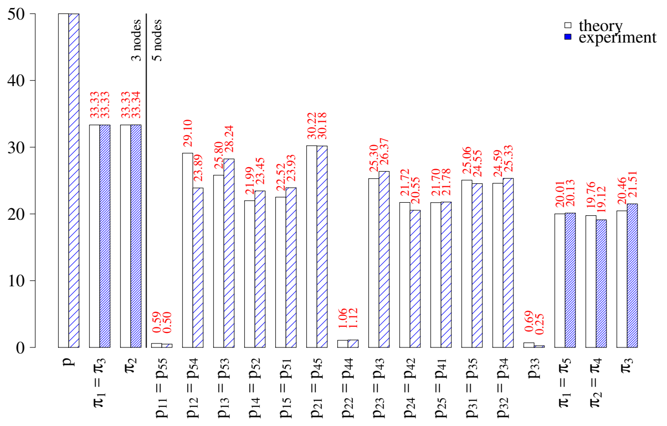
V. Testbed Experimentation

For the purpose of validating our theoretical results, we proceed with extended experimentation using 3 or 5 nodes of our NITOS testbed [13]. We use *etcd* [14] for building a distributed key-value data store over the NITOS nodes, exploiting the Raft consensus algorithm. More specifically, we use an example usage of etcd's Raft library, called *raftexample*, provided as part of the same project.

For etcd's Raft implementation, the time is slotted and the timeouts are measured as integer number of slots. By default, the slot is 100 milliseconds. The election timeout is the sum of a fixed period equal to 10 slots and a random interval between 0 and 9 slots. At the start of each slot, if node has received a heartbeat request, its timeout is zeroed. Otherwise, if timeout has expired, node transits to candidate. Due to the non precisely synchronized clocks of the nodes, at each follower, the duration between the moment it received the last heartbeat from leader and the moment that its next slot starts is random and less or equal to 1 slot. Thus, the election timeout at each node will be between 10 and 20 slots or between 1 and 2 seconds. Mapping 1 second to the upper limit 1 of t_i^{out}, then $t_o^{\text{out}}=1$ too.



(a) Equal ranges of election timeouts.



(b) Equal leadership probabilities.

Fig. 2. Probabilities **P** and π for the bus network with 3 and 5 nodes, when $d = 0.035$ and $d = 0.025$ respectively. The empty and filled bars correspond to the theoretically and experimentally estimated values respectively.

In order to emulate the instant failures, we disable each leader to send more than one heartbeat request at each term. In this way, followers assume that leader is fallen after the first heartbeat and leader is back again to vote for the triggered election. We repeat 20000 instant failures and elections in order to estimate the transition and leadership probabilities, by tracking the log files and measuring how many times each node has been leader and which node was the predecessor.

The plots of Figure 2 show the probabilities for the bus networks of Figure 1. For the network with 3 nodes, we assume that the delay between two adjacent nodes is 35 milliseconds ($d = 0.035$). We highlight that heartbeat timeout is almost equal to $2d_{13} = 4d = 0.14$, which is a magnitude of order less than the average election timeout 1.5. The experimentally estimated p is the average between the measured p_{13} and p_{31}. The same holds for $\pi_1 = \pi_3$, where the estimated value is the average between the two measured ones. We also mention that $p_{ii} = 0$ and $p_{21}, p_{23} \simeq 0.5$, as we expect from our theoretical analysis. Both plots confirm that our theoretical analysis (empty bars) gives precisely the experimentally estimated probabilities (filled bars). The left bars of Figure 2(a) show the unequal leadership probabilities, as result of the equal ranges of the election timeouts ($\alpha_2 = \alpha$), while the corresponding bars of Figure 2(b) show the opposite.

The right bars of Figure 2(a) show the same probabilities for equal ranges of the election timeouts at the bus network with 5 nodes, when $d = 0.025$. The heartbeat timeout is almost equal to $2d_{15} = 8d = 0.2$ and a magnitude of order less than the average election timeout. We group again our experimentation results by presenting the average values between e.g. p_{15} and p_{51} or π_1 and π_5, due to the network symmetry. Figure 2(b) shows the corresponding probabilities for election timeouts limited by $\alpha_3 = 1$, $\alpha_2 = \alpha_4 = 1 - 3.06d = 0.92$ and $\alpha_1 = \alpha_5 = 1 - 10.89d = 0.73$ (given at the end of Section IV-C). The experimentation results are very close to the theoretical predictions (maximum difference is less than 5% for the theoretically and experimentally estimated values of p_{12}).

VI. Conclusions & Future Work

In this paper, we model the leadership probabilities in a Raft based cluster of SDN controllers, which is operated over a network. This model can be used for adjusting the ranges of the election timeouts of the controllers, in order to change appropriately the leadership probabilities. This can be done for minimizing the average response time for a command sent to each controller. Our future work includes the study of the same model with use of the Raft implementation of ONOS or ODL. We will also model scenarios with non instant failures, where failed leaders do not vote for the new leader.

Acknowledgment

The research leading to these results has received funding by the European Horizon 2020 Programme for research, technological development and demonstration under Grant Agreement Number 857201 (H2020 5G-VICTORI).

References

[1] D. Ongaro and J. Ousterhout. In Search of an Understandable Consensus Algorithm. In *Proc. USENIX ATC*, 2014.
[2] H. Howard et al. Raft Refloated: Do We Have Consensus? *ACM SIGOPS*, 49(1):12–21, January 2015.
[3] OpenDaylight (ODL). https://www.opendaylight.org/.
[4] Open Network Oper/ing Sys. (ONOS). https://wiki.onosproject.org/.
[5] B. Heller, R. Sherwood, and N. McKeown. The Controller Placement Problem. In *Proc. HotSDN*, 2012.
[6] K. Choumas et al. SDN Controller Placement and Switch Assignment for Low Power IoT. *Electronics*, 9(2), 2020.
[7] M. Karatisoglou et al. Controller Placement for Minimum Control Traffic in OpenDaylight Clustering. In *Proc. 5GWF*, 2019.
[8] E. Sakic, F. Sardis, J. W. Guck, and W. Kellerer. Towards adaptive state consistency in distributed SDN control plane. In *Proc. ICC*, 2017.
[9] T. Zhang, A. Bianco, and P. Giaccone. The Role of Inter-Controller Traffic in SDN Controllers Placement. In *Proc. IEEE NFV-SDN*, 2016.
[10] Y. Zhang et al. When Raft Meets SDN: How to Elect a Leader and Reach Consensus in an Unruly Network. In *Proc. APNet*, 2017.
[11] E. Sakic and W. Kellerer. Response Time and Availability Study of RAFT Consensus in Distributed SDN Control Plane. *IEEE Trans. on Network and Service Management*, 15(1):304–318, 2018.
[12] R. Hanmer, L. Jagadeesan, V. Mendiratta, and H. Zhang. Friend or Foe: Strong Consistency vs. Overload in High-Availability Distributed Systems and SDN. In *Proc. ISSREW*, 2018.
[13] NITOS testbed. https://nitlab.inf.uth.gr/NITlab/nitos.
[14] etcd: A distributed key-value store. https://etcd.io/.