An Asynchronous Control Plane for CRUX

Project Report

Pasindu Nivanthaka Tennage

306888

CS-699

Computer and Communication Sciences- EDIC
DEDIS
Supervisor: Bryan Alexander Ford

Introduction

CRUX is a locality preserving replication system for key value stores that provides low latency stretch, availability and strong consistency guarantees of a key value store in the face of network partitions and node failures [1]. Providing availability and consistency guarantees in the face of network partitions is a problem faced by systems distributed across multiple geographical regions. The strawman approach for solving this problem is running a separate instance of the system in every region, thus in the face of the network partitions, the regions which are not affected by the partition can operate without interruption. However, replicating the system in all the regions results in high resource overhead, and CRUX introduces algorithms to replicate the system which reduces the additional overhead to a minimum, while the partition resistance property stays efficient. CRUX is generic in the sense, it can be applied to any distributed key value store.

Crux leverages an inter-node latency map for its locality preserving constructions, however, it does not handle topology dynamics present in distributed systems. Thus an algorithm to handle node churn and node movements (mobile nodes) is required for CRUX. The control plane for CRUX is the algorithm which serves this purpose. The CRUX is replicated in regions, whereas the control plane is global. The current control plane for CRUX addresses these requirements by employing a system which evolves with node movement [2]. However, the current control plane of CRUX is implemented using synchronous protocols. Synchronous protocols make the underlying assumption that the set of nodes have synchronized clocks, which does not hold in practice due to clock skew and drift. Hence the current control plane algorithm fails to operate correctly in the absence of synchronized clocks.

To address this problem with synchronized CRUX control plane [2], it is required to make the control plane fully asynchronous, and in this project, we propose a novel control plane algorithm for CRUX which is fully asynchronous. Our asynchronous CRUX control plane algorithm provides support for initiating a new CRUX instance, joining new nodes to an existing CRUX system and handling node leavings (can be both planned and unplanned), without relying on any synchrony assumptions.

In the following sections, we first describe the design overview and then the control plane protocol. We then present our simulation results and discussion of the control plane algorithm.

Design Overview

Our asynchronous control plane continuously updates the set of nodes in CRUX and their latency matrix, without relying on any timeouts. It consists of four building blocks; 1. membership component which determines the set of nodes which are participating in the CRUX, 2. ping distance component which periodically collects the ping distances to each other node which are used when assigning nodes to CRUX regions, 3. region component which handles the regions in CRUX based on the membership and ping distances, and 4. epoch component which divides the time into epochs. Using these four components, we explain the high level workflow of the control plane in figure 1.

As shown in figure 1, the control plane operates in non overlapping epochs. An epoch consists of six main tasks, five of which should be done in sequential lock steps, and a task which runs in parallel with the aforementioned five steps. First, consensus is made about the set of nodes participating in the epoch. Second, ping distances are calculated for the set of nodes that each node agreed in step 1. Third, the ping distances are multicast to each node in the agreed set of nodes in step 1. Fourth, a ping distance matrix is generated,

which contains the ping distances observed by each node and nodes participate in the consensus on the ping distance matrix. Finally a new CRUX instance is created using the agreed set of nodes and the ping distance matrix. When these five steps are completed (an epoch completes) and there are new nodes waiting to join the next epoch, a new epoch of the control plane starts. The registration for the next epoch task listens to the new nodes that are willing to join for the next epoch of the control plane.

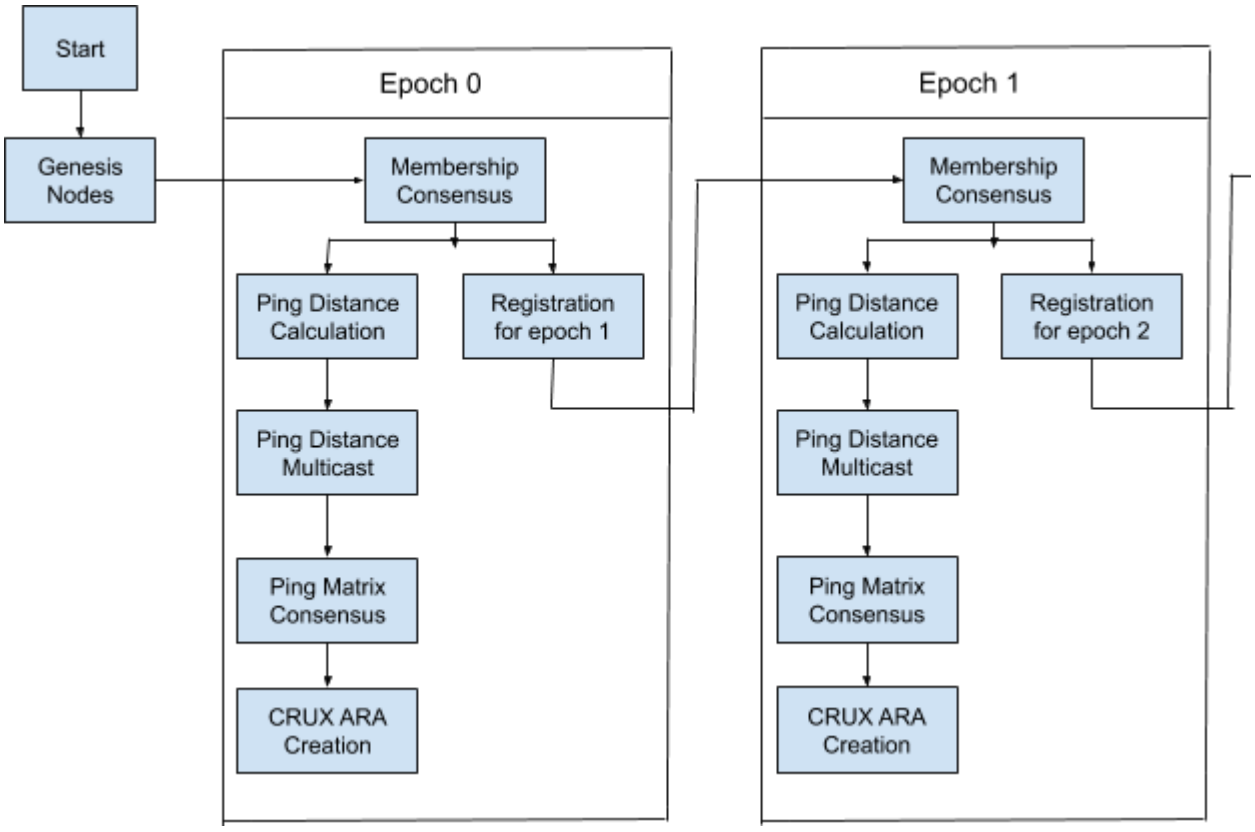

Figure 1: Control plane workflow: Control plane works in non overlapping epochs

Making the control plane fully asynchronous has its own challenges. First, each of the above five steps should be done in the exact sequential order in each of the nodes, thus we need a distributed barrier to impose the ordering and synchronization. Second, having consensus in a totally asynchronous environment is considered an impossible problem when there is at least one failing node. To address these two issues, we use TLC [3] and QSC [3] protocols respectively. TLC provides an abstraction of a synchronized barrier on top of a totally asynchronous message passing system which guarantees the progression of a majority of nodes, thus the first problem is addressed. QSC provides a randomized asynchronous consensus algorithm, which guarantees that a majority of nodes will progress making the consensus, given that there are enough consensus rounds.

We make several assumptions in our design. There is a set of well defined nodes, called the *genesis nodes* as shown in the figure 1. Each point to point communication is done using reliable unicast (TCP). We only assume non byzantine behavior of nodes in this protocol and the nodes have fail stop behavior.

Protocol

In this section we describe the details of the control plane components we described in the above section, in detail. Due to time considerations, we do not implement the ping distance calculation and CRUX ARA creation steps, and they will be covered in our future work.

First we discuss how a new node joins the admission committee for the next epoch. A new node should first contact the admission committee of current epoch, inform them that the node is interested in joining for the next epoch, prove himself as an eligible node for the next epoch, and then ask the admission committee to inform when the admission committee starts the construction of the next epoch.

A new node first sends a “join message” with a name, a public key and asks for a threshold-signature from the admission committee. The new node needs to get a threshold signature from the admission committee, because threshold signature resembles that a majority of the admission committee is in favor of having this new node in the next epoch. If the new node manages to get back a threshold-signature from the admission committee, it broadcasts it again to the admission committee (because there is no other way for each member of the admission committee to know whether the new node got threshold signatures).

Each node in the admission committee starts a new epoch of the control plane when there is more than zero number of new nodes waiting to join the admission committee or, a predefined time has elapsed since the last epoch of the control plane. We use QSC protocol as described below to make the membership consensus. Consensus of membership list makes sure that a majority of the members of the new admission committee agree upon some membership list that is proposed by a previous admission committee member. Without loss of generality, let's assume that each node is in TLC step 0 at the beginning. Upon reaching s = 0, members of the admission committee start the membership consensus. N refers to the total number of nodes, and f refers to the maximum number of faulty nodes ($N>2f+1$).

1. At s = 0, each node in the admission committee, generates a random number. We assume that no two nodes obtain the same random number, and ties are broken using a node identifier as the secondary number.

2. Then each node Pi, multicasts a message with {view of the members (other nodes in the admission committee, and the new nodes), the random number}, to the admission committee.

3. Then the TLC witnessing process [3] happens

4. Node Pi, upon receiving more than $N-f$ number of acks, multicasts a witnessed message to the admission committee.

5. Node Pj, after collecting $N-f$ number of witnessed messages to move to s = 1, makes a list of nodes from which it received the $N-f$ number of messages and multicasts it with s = 1.

6. Then the TLC witnessing process happens.

7. Each node Pk, at s = 2, knows about a majority of witnessed proposals in S = 0, because of the 2 step broadcast property of TLC [3].

8. Then each node uses the default TLC messages to move to s = 3.

9. By s=3, each node Pi individually chooses node Pj's s = 0 membership list, as the best possible set, by evaluating the following condition.
   If (Node Pi used some node Pk's s = 1 message with node list to move to s = 2, and Pj was in that list && Pj's proposal at s = 0, has the highest random number, out of all the (s = 0)'s message's random numbers, which Pi has heard by s = 2) or (out of all nodes in s = 1 node lists messages node Pi heard at step s = 3, Pj is a member in some message && Pj's proposal at s = 0, has the highest random number, out of all the (s = 0)'s message's random numbers, which Pi has heard by s = 2)

Steps 1-9 above correspond to one round of consensus protocol in QSC. There are two possible outcomes at the end of a single consensus round. A node has reached the consensus / or not. If the node has reached the consensus, it can continue to the ping distance calculation step, but has to make sure that the majority of the nodes have reached the consensus. If the node has not seen the consensus, it should participate in another consensus round.

To address both these cases, we propose the following approach. If the consensus is seen with respect to node Pi, it will set a flag in the next unwitnessed message indicating that it has reached the consensus. Irrespective of the fact whether the consensus is made or not, each node will start a new consensus round.

Then, in step s = 5, each node receives a majority of s =3 messages and checks whether a majority of nodes have reached the consensus. If yes, then all the nodes aborts the new consensus round and moves to the ping calculation step. However if a majority of the nodes has not reached the consensus, the new consensus round is continued.

This approach guarantees that a majority of nodes will reach the consensus in the same TLC step, thus providing the distributed lock step (barrier) property.

The admission committee then notifies the newly joined nodes that they have been added to the node list and sends the current step. After this, the admission committee retires and the selected members of the new epoch are treated as the admission committee.

The admission committee calculates the ping distances to each other member in the admission committee (in the current version of the project, we do not implement this and we synthetically generate ping distances). Upon completing the distance calculation each node moves to the next TLC step and multicasts its ping distance vector to the admission committee. Multicasting of the ping distances guarantees that at the end of the multicast process, a majority of nodes have seen a majority of ping distance vectors from the other nodes.

Then all the nodes in the admission committee participate in a consensus process identical to what we explained above for the consensus of the membership. Consensus of the ping matrix makes sure that a majority of the new admission committee agree upon a ping distance matrix that is proposed by any one admission committee member. Upon reaching consensus on the ping distances, each node moves to the next TLC step, and calculates the levels, bunches, clusters and ARAs.

Experiments and Analysis

In this section we first present our prototype implementation of the asynchronous control plane algorithm. Then we present our simulation results. We implemented the asynchronous control plane algorithm in the Cothority framework [4] which runs on top of the ONET [5] framework. Both ONET and Cothority framework are written in GoLang [7]. Our prototype implementation is available at [6]. We run our simulations in the ONET.

We first evaluate the asynchronous control plane's ability to handle node churn. A correctly executing control plane should be able to start a new epoch of the control plane with a given number of nodes, handle new node joins and handle node leavings. To experiment on the correct execution of the control plane, we perform two experiments. Experiment 1 simulates the behavior of the control plane when new nodes are added in random time intervals. We start running the control plane initially with five genesis nodes and then add nodes with random arrival times. Experiment 2 simulates the behavior of the control plane when the nodes leave. We start running the control plane initially with 15 genesis nodes and remove nodes in random time intervals. We set network wide link latency to 20ms. Figure 2 and 3 depict the experiment 1 and 2 results respectively.

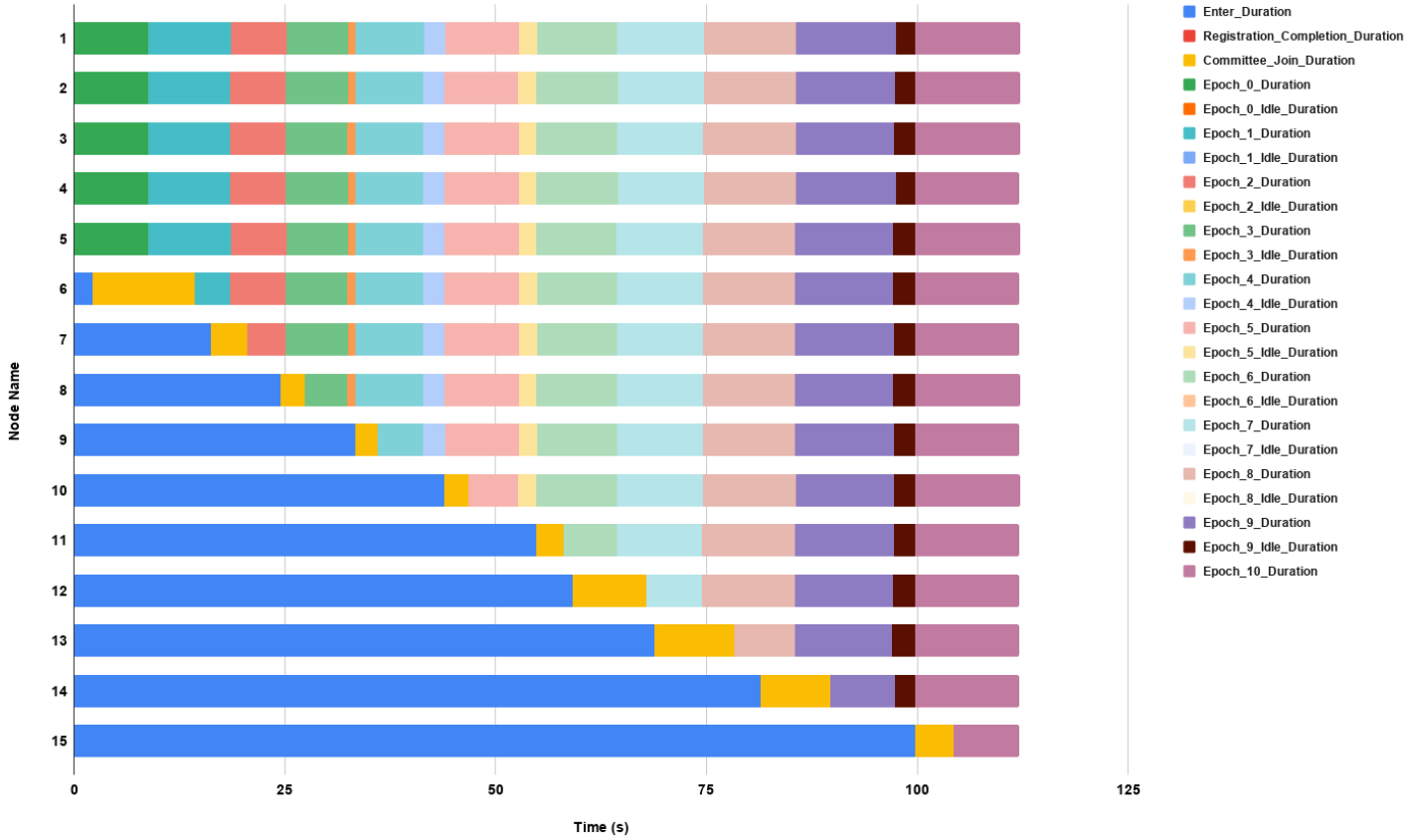

Figure 2: Control plane evolution with adding new nodes (adding 10 new nodes periodically results in having 10 epochs of the control plane, and the new nodes are included in the immediate next epoch)

We observe that the control plane dynamically handles the admission committee membership when new nodes are added. New nodes have to wait only until the completion of the currently running epoch (this waiting time is shown in dark yellow colour in figure 1), and are added to the next epoch. We also observe that all the nodes progress in a lock step fashion in each epoch, though the wall clock time is different for each node.

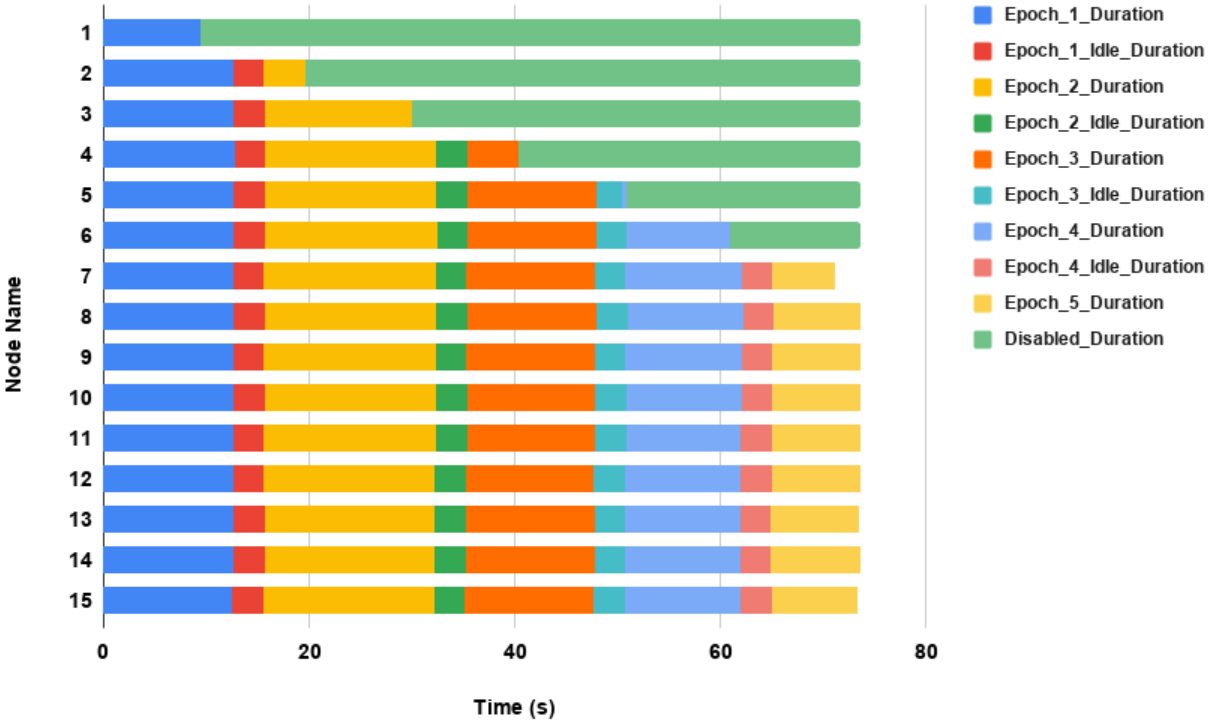

Figure 3: Control plane evolution with leaving nodes (the control plane continuous to operate despite some nodes leave)

As seen in figure 3, the control plane continues to operate correctly despite node leavings at arbitrary times. Nodes 1-5 leave the system at arbitrary times, and the control plane detects this behavior and reflects that by removing the node which left from the immediate next epoch.

A key requirement of an asynchronous control plane is that it should continue to progress with a majority of nodes despite arbitrary network delays. We experiment on the network resilience of the control plane, by adding high delay components to links that are connected to N number of nodes; first we add high delays to all the links connected to one node, second to all the links connected to two selected nodes and so on (N can be also considered as the number of relatively slow nodes). We measure the time for the completion of a single epoch while varying N. For all values of N, we set the network wide delay to 20ms and per link high delay to 100ms. Figure 4 depicts this behavior.

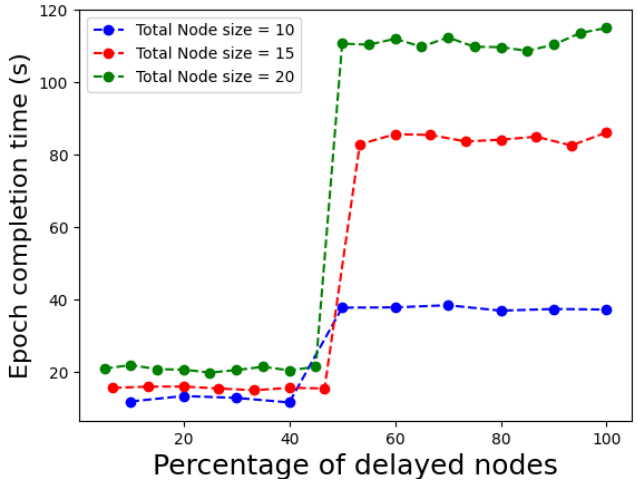

Figure 4: Resilience to network delays: the epoch completion time remains lower when a majority of nodes are fast, and increases when the majority of nodes are slow

We observe that when a majority of nodes are fast, they progress with a lower epoch time, for example when the percentage of delayed nodes is less than 50%, the epoch takes close to 10 seconds, whereas when the percentage is greater than 50%, the epoch takes close to 40 seconds, when the total node size is 10. This observation confirms that the asynchronous control plane makes progress with a majority of nodes irrespective of the high network delays. When the percentage of delayed nodes exceeds 50%, the speed of the majority becomes dependent on the slow link speed; hence the epoch time increases.

Bandwidth usage is an important aspect of the control plane algorithm. Theoretically the number of messages per epoch can be derived as N+3N*T = N*(1+3T), where N is the number of nodes and T is the number of TLC steps for an epoch. T depends on the number of rounds it takes for the node consensus and the ping matrix consensus to finish. If we define the average message size to be m, the total traffic for a single epoch is N*(1+3T)*m; a linear function of N. We simulate the bandwidth usage of the control plane w.r.t the number of nodes by measuring the bandwidth for different number of nodes. Figure 5 below depicts this behavior.

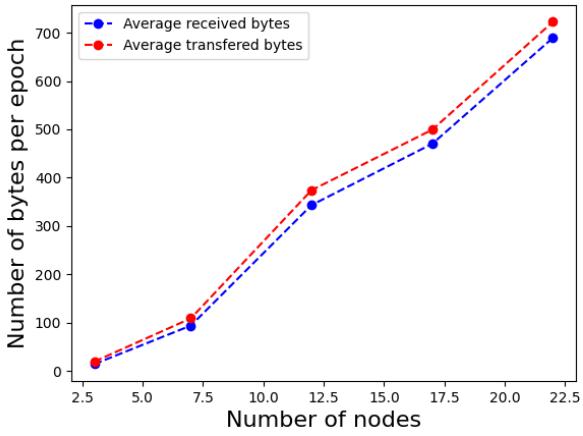

Figure 5: Bandwidth usage with increasing number of nodes

We can observe that the bandwidth usage increases near linearly when the number of nodes are increased. This validates our theoretical results. However it should also be noted that linear increase in bandwidth is not an optimum characteristic of distributed systems, and we will explore on how to reduce the bandwidth usage in our future work. Moreover, the bandwidth usage of each node is roughly equal, because the TLC algorithm treats each node equally, and each node receives and sends roughly an equal number of messages.

Summary and Future Work

In this project we aimed at building a fully asynchronous control plane for CRUX which guarantees a majority progression of nodes. We implemented our prototype in GoLang. We also simulated the evolution of the control plane with respect to churn, resilience to network delays, and bandwidth characteristics of the control plane.

The current implementation uses point to point TCP connections between each two nodes; in a N node system there are N^2 + N numbers of connections. While this approach scales for small < 20 number of nodes, it does not scale to higher > 20 number of node systems. Hence, scalable broadcast methods will be explored as the future work.

We use the QSC protocol for the consensus. Evaluating the performance (bandwidth, CPU time) of QSC against the classical consensus protocols such as RAFT, PAXOS is a plausible future work. This will give insights into which consensus algorithm performs better in an asynchronous control plane algorithm.

References

[1] C. Basescu, M. Nowlan, K. Nikitin, J. Faleiro and B. Ford, "Crux: Locality-Preserving Distributed Services", arXiv.org, 2018. [Online]. Available: https://arxiv.org/abs/1405.0637. [Accessed: 19- Jun- 2020].

[2] Pannatier. A, “A Control Plane in Time and Space for Locality-Preserving Blockchains”, Master Thesis, École Polytechnique Fédérale de Lausanne, 2020

[3] Ford. B, Jovanovic. P, Syta. E, “Que Sera Consensus: Simple Asynchronous Agreement with Private Coins and Threshold Logical Clocks”, arXiv.org, 2020. [Online]. Available: https://arxiv.org/pdf/2003.02291.pdf. [Accessed: 19- Jun- 2020].

[4] "dedis/cothority", GitHub, 2018. [Online]. Available: https://github.com/dedis/cothority. [Accessed: 19- Jun- 2020].

[5] "dedis/onet", GitHub, 2018. [Online]. Available: https://github.com/dedis/onet. [Accessed: 19- Jun- 2020].

[6] "student_20_tlcCtrlPlane/onet", GitHub, 2018. [Online]. Available: https://github.com/dedis/student_20_tlcCtrlPlane/ . [Accessed: 19- Jun- 2020].

[7] "The Go Programming Language", Golang.org, 2017. [Online]. Available: https://golang.org/. [Accessed: 19- Jun- 2020].