ID2203 – Distributed Systems, Advanced course

Ayoub Bargach and Vasileios Charalampidis

March 11, 2018

Abstract

Distributed computing is now a central paradigm in a lot of web applications. It offers solutions to make systems more resilient and scalable. From this concept, new paradigms of distributed databases like blockchain have been made possible. In this project, we will focus on decentralized distributed key-value store.

1 Introduction

One of the requirement to have a highly scalable and available data is to design a distributed system which can handles it. In this project, we aim to build a basic decentralized key-value store solution which answers to this need.

The project will be built using a java library for distributed computing, Kompics. We need to make some assumptions before starting our project. Therefore, in the first part we will start by describing the goals of our problem, and making some assumptions in the underlying transmission channels.

> A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.
> Leslie Lamport

Failure aspects are central and implies the above stacks designs such as consensus. This implementation aims to handle GET, PUT and CAS operations using the following abstractions : Crash-stop model, Abortable Sequential Consensus, Replicated State Machine, Monarchical Eventual Leader Detection. In order to use these abstractions, we need at least to assume BEB (Best-Effort Broadcast) and Eventual Perfect Failure Detection. Thus, we assume that our underlying system is assimilable to a Perfect links transmission system.

2 A distributed key-value store

2.1 Description

Nowadays, the need of storing large amount of data in databases has led to the common trend of distributed key-value stores. The main concept of such systems is to have a unique key which is associated with a specific set of data. For example, in the case of a bank, the key can be the account number of a specific customer and the value can be all data associated with this specific account. What makes these systems exceptional is that they are distributed.

This means that we can have multiple server/nodes that communicate with each other in a way that they distribute the keys, and as a result they divide the need of storing large amount of data in only one database. The big motivation behind those systems is that a single point of failure is avoided. If a database fails e.g. due to power outages, cooling failures, and natural disasters, then the data are not lost. This is achieved by having several replicas in different nodes.

2.2 Functional specifications

Before starting, we need to define the functional specifications of our problem. The purpose is to implement a distributed key-value store with linearizable operations and specific replication degree.

- The system is distributed through a set of nodes (named servers in the project).
- The system defines the partitions within the nodes. The partitions are replicated with respect to a specific replication degree δ.
- Nodes are able to detect failure of other nodes.
- The system should manage 3 operations GET, PUT and CAS and ensure their linearizable consistency.

3 Specificities of the distributed key-value store

This section aims to give more information about the key-value store solution.

3.1 Assumptions on the underlying system

Before start developing, let us make some assumptions on the system. We aim to implement a Fail-noisy abstraction :

Partially synchronous We are not in the case of a synchronous system like in embedded designs. Here, we are potentially using the internet to define our system. So, assuming that the system is partially synchronous means that the system is asynchronous and eventually, it becomes after a while synchronous to run and terminate algorithms.

Crash-stop model This property handles node failures. We assume that nodes do not have access to a persistent storage. Thus, they cannot save their state and consequently, they can not recover after a failure. If a node crashes, it restarts with the initial state. The reconfiguration will not be handled. So, even if the node crashes for a long time, it will always be taken into account by the system.

Perfect links Messages are handled between nodes using the protocol TCP. It a session based protocol which guarantees the sending and delivering of messages betweens nodes. Perfect links guarantees that transactions between correct processes are always completed. We will see further what this assumption implies.

Eventually perfect failure detection This abstraction takes into account the fact that the system is partially synchronous. This failure detection is eventually perfect and we are not sure for a finite time if the suspected node have effectively crashed.

3.2 Replication groups

Replication groups refer to the partitions of nodes that communicate with each other in order to keep a consistent and updated state. They can be realized as a single unit that mainly aim to provide safety in the process of storing data. This means that if a node inside a replication group crashes, all the requests will be handled by the other nodes of the partition providing in such way safety and availability.

3.3 Linearizable consistency

Linearizability is a property that guarantees that once an operation is completed within a register in one process, the operation is replicated on all other distributed processes that belongs to the same replication group. To be more precise, after a write operation in a given register, all read operations in all nodes must return the value that has been written. That means that you cannot have two different read values between a write operation.

3.4 Paxos algorithm

Paxos is used to reach consensus between nodes. A very simple way to think of Paxos is the following: a) a leader node (i.e. Proposer) is elected, b) the leader proposes a value to other nodes (i.e. Acceptors), c) the other nodes can either accept or reject the proposed value, d) if a majority of nodes have accepted, then consensus has been reached and the leader sends a commit message to all nodes.

4 Implementation and simulations

In this part, we will discuss the implementation aspects of our system. Also, for each component, a simulation scenario has been written to test if component's behavior respects the functional specifications.

4.1 The Kompics library

To simulate the distributed systems, we use a specific library: Kompics. This library defines 3 concepts: components, events and channels. In the model, the nodes are a set of components driven by events and connected by channels. For each component, we need to define unidirectional ports and the events that can be sent through.

Each component is scheduled on one thread which increase the isolation for each component and simulate the parallelism of each node.

Components choose events they handle by subscribing to event-handler of each port.

Channels are FIFO (First-In-First-Out) ordered. The property of no creation ensure that each message is delivered once.

Kompics provides classes to handle time, network and simulations.

4.2 Development tools and methodology

For this project, we used specific tools and IDEs :

- IDE : IntelliJ IDEA, by JetBrains.
- Source code : github for code management.
- Java JRE 1.8 environment.
- Tests : Kompics.
- Simulations : Kompics.

All specifications and timetable (Gantt diagram) have been described in the preliminary report.

During the code development, we followed the following process :

- Describe the assumption and the abstraction.
- Analyse the underlying algorithm.
- Transpose the algorithm to Kompics concepts and code.
- Test the code.
- Run simulation scenario to validate the behavior of the component.

4.3 The partitioning method

For our implementation we considered a partitioning method in which every partition group (3 nodes per partition, i.e. the replication degree equals 3) has a specific key (same for each participating node inside the partition) and can handle 10 client keys. This means that if for example the first partition is identified with the key 0, then a request to PUT a key-value with key equal to 5, will be handled by itself. But if the request has a key 12, then it will be handled by the second partition, and so on.

4.4 Infrastructure

Before analyzing how we implement each component, this section give an overview of the whole system and interactions between components.

The list with all the implemented components is the following:

- Best-Effort Broadcast
- Eventual Perfect Failure Detector
- Monarchical Eventual Leader Detector
- Key-Value Store

- Abortable Sequence Consensus (Variant of Multi-Paxos)

Figure 1 illustrates the infrastructure of our system with all the interconnections between components. The next section provides details for each component respectively.

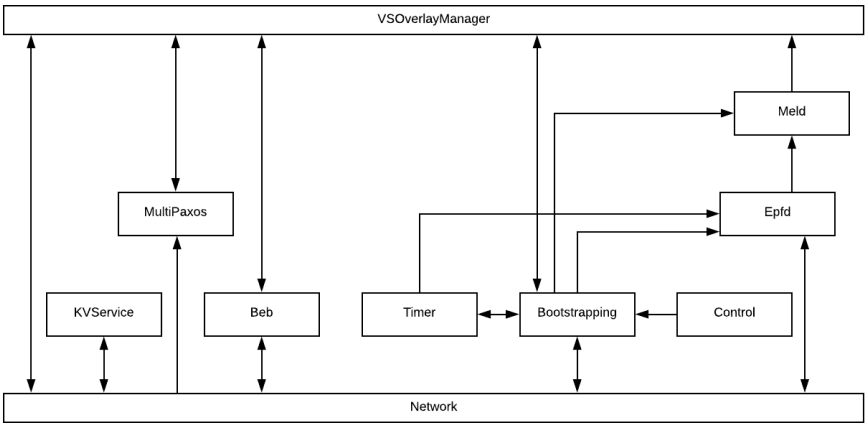

Figure 1: Overview

4.5 Components ans Simulations

4.5.1 Best-Effort Broadcast

The first component we aim to implement is Best-Effort Broadcast. This abstraction guarantees that a broadcast message is delivered to all correct nodes if the sender does not crash. The algorithm that we implemented can be found in Appendix A. The corresponding component that we created has the following Kompics representation:

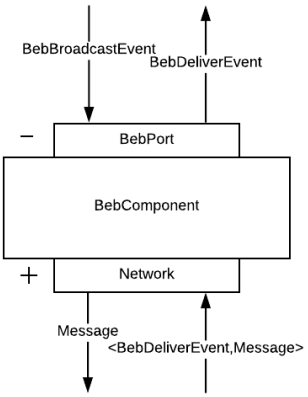

Figure 2: Component Best-Effort Broadcast

With Best-Effort Broadcast, we have to ensure the following properties:

Validity If a process sends a message m, then every correct process should delivers m. (Liveness)

No duplication No message is delivered more than once. (Safety)

No creation If a process delivers m, then m was previously broadcast by a process. (Safety)

All the above properties are a consequence of Perfect Point-to-Point Links. In order to validate this behavior in our system, we decided to verify both abstractions with the same simulation. Here is our proposed scenario:

1. We create our system with different replication groups for a total of 6 nodes.
2. Before sending a Get message, we kill a node from the aimed replication group.
3. We send a Get message to a node which belongs to the wrong replication group : This way, we firstly test the Point-to-Point Perfect Link.
4. Once the "wrong" node delivers the Get request, it broadcasts it to reach the correct replication group.
5. If all correct nodes of the replication group delivered the Get request, it means that the Best-Effort Broadcast is working properly.

The results are shown in the Appendix B.

Also, we did not test that our channels are actually FIFO Perfect Point-To-Point Links. This is ensured by the TCP/IP stack. Indeed, each packet have an identifier. When the packets are delivered, they are in the good order (So you can see your video from the beginning !).

4.5.2 Monarchical Eventual Leader Detector

Before testing the Monarchical Eventual Leader Detector, we need to analyze the Eventual Perfect Failure Detector. This abstraction uses the PPPL as can be seen from the corresponding algorithm in Appendix A. The corresponding component that we created has the following Kompics representation:

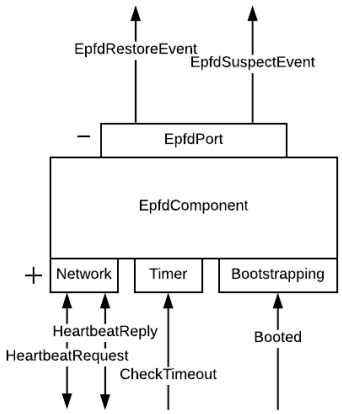

Figure 3: Component Eventually Perfect Failure Detector

We have to ensure the following properties:

Strong completeness Eventually, every process that crashes must be suspected by all correct processes.

Eventual Strong Accuracy Eventually, no correct process is suspected by any correct process.

The simulation stochastic process is the following :

1. We launch 6 normal nodes.
2. We kill one normal node and wait a moment.
3. All correct processes must suspect the crashed node.
4. We launch again this crashed node with the same IP.
5. This new node is no more in the suspected set of crashed nodes.

The results are shown in the Appendix C.

We are sure now that the EPFD is working. We can use it to implement the Monarchical Eventual Leader Detector as we are in the crash-stop model abstraction. We aim at finding the leader in each partition so that to ensure liveness of consensus process. The algorithm of Eventual Leader Detector can be found in Appendix A. The corresponding component that we created has the following Kompics representation:

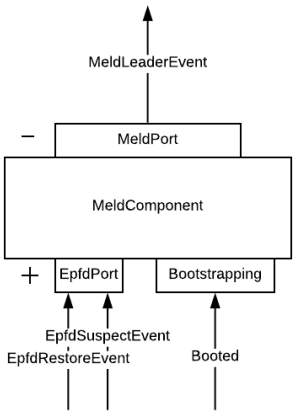

Figure 4: Component Monarchical Eventual Leader Detector

The process that has the smallest IP and the smallest TCP port is the highest ranked. This is our definition of the function $maxrank()$. This algorithm should ensure the following properties :

Eventual Accuracy There is a time after which every correct process trust a correct process.

Eventual Agreement There is a time after which no two processes trust different leaders.

So the way we define our ranking function impacts directly the proper behavior of our component. Here the simulation process that we followed:

1. We initiate our system with 6 nodes.
2. Kill the node with the highest rank.
3. Analyze the printouts to identify the new leaders. There must be one leader elected per replication group.

The results are shown in the Appendix D.

4.5.3 MultiPaxos

The final part that we implemented is a multipaxos variant : Abortable Sequence Consensus. We need a consensus abstraction to be able to implement the CAS operation. Indeed, all the nodes need to agree in a new sequence of keys (or commands). Other solutions that can be used: Consensus based on ballot leader election or uniform consensus based on epoch leader election for instance. The algorithm followed for Abortable Sequence Consensus can be found in Appendix A. The corresponding component that we created has the following Kompics representation:

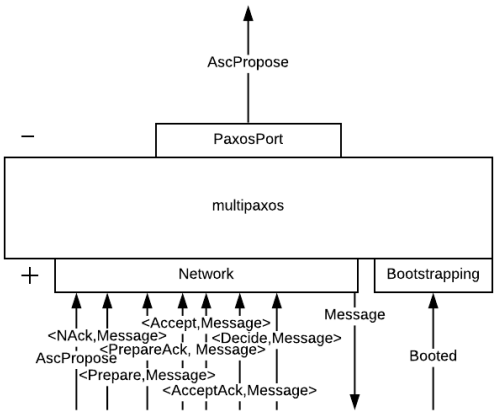

Figure 5: Component Abortable Sequence Consensus

We implement this version to test independently the different components. The ASC only need a FIFO Perfect Link.

The ASC support the following properties :

Validity If process p decides v, then v is a sequence of proposed commands without duplicates.

Uniform Agreement If process p decides u and process q decides v then one is a prefix of the other.

Integrity If a process p decides u and after decides v then u is a prefix of v.

Termination If a command C is proposed, then eventually, it is contained in a decided sequence of each correct process.

The aim is to show that all nodes in the same replication group agree eventually in a same sequence of commands. Here is the simulation process that we followed:

1. We initiate our system with 6 nodes.
2. Client1 sends a PUT operation.
3. Client2 sends a PUT operation.
4. Client3 sends a CAS operation.
5. All client must agree in the sequence of commands. This way, a replicated state machine is ensured.

The results are shown in the Appendix E.

5 Discussion

The main goals of this project were to implement and test a simple portioned, distributed in-memory key-value store with linearizable operation semantics.

The requirements can be summarized in supporting a partitioned key-space of some kind e.g. range-partitioned integers, in distributing partitions over the available nodes, in replicating key-value entries with specific replication degree, in supporting GET, PUT, CAS operations from an external system e.g. a client, and finally in detecting failures of nodes in the cluster.

All the above were successfully been deployed and tested by constructing a Java project upon the already given code in github. All the implemented components were tested for their proper operation by providing simple scenarios. The GET, PUT, CAS operations were also tested in order to also prove the linearizable property.

The BEB component works well. Each correct message delivers the messages. This functionality has been tested by sending a put request to the wrong replication group. The request have been broadcast to reach the correct one. (Appendix B)

The Eventually Perfect Failure Detector also works properly. We temporary change the port of one of the nodes. We can see that the node is suspected and added to the set of suspected nodes. After the restart of the node, it is removed from the suspected nodes. (Appendix C)

The third simulation was about the Monarchical Leader Election. We kill the leader (with the highest rank that is given by the lowest IP, .1 for instance). We can see that a new leader is elected (.2 in our example). (Appendix D)

Finally, the proper behavior of AscDecide which performs the CAS operation have been successfully handled in Appendix E.

The advantage of the algorithm is the linearizability provided by consensus and the resilience that can handle N/2 crashed processes because we are in the fail-noisy model.

However, there is limitation because we did not implement reconfiguration of our network. Also, the crashed nodes can not recover from a crash. They can even know if they had crashed. Also, the ASC algorithm on a partially asynchronous model does not guarantee a maximum time of execution. We are just sure that the execution will eventually occur.

6 Conclusion

The work for this project was equally divided among the writers. Ayoub took some time in the beginning to learn Java and worked on the testing/simulation part, as well as the Paxos algorithm. Vasileios worked with the integration of client and common packages, as well as the remaining components of the system and the partitioning system.

The project helped us the understand different aspects of distributed computing. From Perfect failure implementation to consensus, we had an overview of the global behavior of a distributed systems. Some implementations need to be improved using for example a consensus based on Ballot Leader Election. We did not had enough time experiment different solutions so we can understand better the relative performance between algorithms.

Appendices

Appendix A

Algorithm 3.1: Basic Broadcast

Implements:
BestEffortBroadcast, **instance** *beb*.

Uses:
PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle$ *beb*, *Broadcast* | m $\rangle$ **do**
forall $q \in \Pi$ **do**
trigger $\langle$ *pl*, *Send* | q, m $\rangle$;

upon event $\langle$ *pl*, *Deliver* | p, m $\rangle$ **do**
trigger $\langle$ *beb*, *Deliver* | p, m $\rangle$;

Figure 6: Best-Effort Broadcast

Algorithm 2.7: Increasing Timeout

Implements:
EventuallyPerfectFailureDetector, **instance** $\diamond\mathcal{P}$.

Uses:
PerfectPointToPointLinks, **instance** *pl*.

upon event $\langle\ \diamond\mathcal{P}, \mathit{Init}\ \rangle$ **do**
$\mathit{alive} := \Pi$;
$\mathit{suspected} := \emptyset$;
$\mathit{delay} := \Delta$;
starttimer(*delay*);

upon event $\langle\ \mathit{Timeout}\ \rangle$ **do**
if $\mathit{alive} \cap \mathit{suspected} \neq \emptyset$ **then**
$\mathit{delay} := \mathit{delay} + \Delta$;
forall $p \in \Pi$ **do**
if $(p \notin \mathit{alive}) \wedge (p \notin \mathit{suspected})$ **then**
$\mathit{suspected} := \mathit{suspected} \cup \{p\}$;
trigger $\langle\ \diamond\mathcal{P}, \mathit{Suspect} \mid p\ \rangle$;
else if $(p \in \mathit{alive}) \wedge (p \in \mathit{suspected})$ **then**
$\mathit{suspected} := \mathit{suspected} \setminus \{p\}$;
trigger $\langle\ \diamond\mathcal{P}, \mathit{Restore} \mid p\ \rangle$;
trigger $\langle\ \mathit{pl}, \mathit{Send} \mid p, [\text{HEARTBEATREQUEST}]\ \rangle$;
$\mathit{alive} := \emptyset$;
starttimer(*delay*);

upon event $\langle\ \mathit{pl}, \mathit{Deliver} \mid q, [\text{HEARTBEATREQUEST}]\ \rangle$ **do**
trigger $\langle\ \mathit{pl}, \mathit{Send} \mid q, [\text{HEARTBEATREPLY}]\ \rangle$;

upon event $\langle\ \mathit{pl}, \mathit{Deliver} \mid p, [\text{HEARTBEATREPLY}]\ \rangle$ **do**
$\mathit{alive} := \mathit{alive} \cup \{p\}$;

Figure 7: Eventually Perfect Failure Detector

Algorithm 2.8: Monarchical Eventual Leader Detection

Implements:
EventualLeaderDetector, **instance** Ω.

Uses:
EventuallyPerfectFailureDetector, **instance** $\diamond\mathcal{P}$.

upon event $\langle\ \Omega, \mathit{Init}\ \rangle$ **do**
$\mathit{suspected} := \emptyset$;
$\mathit{leader} := \perp$;

upon event $\langle\ \diamond\mathcal{P}, \mathit{Suspect} \mid p\ \rangle$ **do**
$\mathit{suspected} := \mathit{suspected} \cup \{p\}$;

upon event $\langle\ \diamond\mathcal{P}, \mathit{Restore} \mid p\ \rangle$ **do**
$\mathit{suspected} := \mathit{suspected} \setminus \{p\}$;

upon $\mathit{leader} \neq \mathit{maxrank}(\Pi \setminus \mathit{suspected})$ **do**
$\mathit{leader} := \mathit{maxrank}(\Pi \setminus \mathit{suspected})$;
trigger $\langle\ \Omega, \mathit{Trust} \mid \mathit{leader}\ \rangle$;

Figure 8: Monarchical Eventual Leader Detector

Algorithm 1 Multi-Paxos: Prepare Phase

Implements:

AbortableSequenceConsensus, **instance** *asc*.

Uses:

FIFOPerfectPointToPointLinks, **instance** *fpl*.

1: **upon event** $\langle\ asc, Init\ \rangle$ **do**
2: $t := 0$; ▷ logical clock
3: $prepts := 0$; ▷ acceptor: prepared timestamp
4: $(ats, av, al) := (0, \langle\rangle, 0)$; ▷ acceptor: timestamp, accepted seq, length of decided seq
5: $(pts, pv, pl) := (0, \langle\rangle, 0)$; ▷ proposer: timestamp, proposed seq, length of learned seq
6: $proposedValues := \langle\rangle$; ▷ proposer: values proposed while preparing
7: $readlist := [\bot]^N$;
8: $accepted := [0]^N$; ▷ proposer's knowledge about length of acceptor's longest accepted seq
9: $decided := [0]^N$; ▷ proposer's knowledge about length of acceptor's longest decided seq

10: **upon event** $\langle\ asc, Propose \mid v\ \rangle$ **do**
11: $t := t + 1$;
12: **if** $pts = 0$ **then**
13: $pts := t \times N + rank(self)$;
14: $pv := prefix(av, al)$;
15: $pl := 0$;
16: $proposedValues := \langle v \rangle$;
17: $readlist := [\bot]^N$;
18: $accepted := [0]^N$;
19: $decided := [0]^N$;
20: **for all** $p \in \Pi$ **do**
21: **trigger** $\langle\ fpl, Send \mid p, [\text{PREPARE}, pts, al, t]\ \rangle$;
22: **else if** $\#(readlist) \leq \lfloor N/2 \rfloor$ **then**
23: $proposedValues := proposedValues + \langle v \rangle$; ▷ append to sequence
24: **else if** $v \notin pv$ **then**
25: $pv := pv + \langle v \rangle$;
26: **for all** $p \in \Pi$ **such that** $readlist[p] \neq \bot$ **do**
27: **trigger** $\langle\ fpl, Send \mid p, [\text{ACCEPT}, pts, \langle v \rangle, \#(pv) - 1, t]\ \rangle$;

28: **upon event** $\langle\ fpl, Deliver \mid q, [\text{PREPARE}, ts, l, t']\ \rangle$ **do**
29: $t := max(t, t') + 1$;
30: **if** $ts < prepts$ **then**
31: **trigger** $\langle\ fpl, Send \mid q, [\text{NACK}, ts, t]\ \rangle$;
32: **else**
33: $prepts := ts$;
34: **trigger** $\langle\ fpl, Send \mid q, [\text{PREPAREACK}, ts, ats, suffix(av, l), al, t]\ \rangle$;

35: **upon event** $\langle\ fpl, Deliver \mid q, [\text{NACK}, pts', t']\ \rangle$ **do**
36: $t := max(t, t') + 1$;
37: **if** $pts' = pts$ **then**
38: $pts := 0$;
39: **trigger** $\langle\ asc, Abort\ \rangle$

Figure 9: Abortable Sequence Consensus

Algorithm 2 Multi-Paxos: Accept Phase

40: **upon event** $\langle\ fpl, Deliver \mid q, [\text{PrepareAck}, pts', ts, vsuf, l, t']\ \rangle$ **do**
41: $\quad t := max(t, t') + 1;$
42: $\quad$ **if** $pts' = pts$ **then**
43: $\quad\quad readlist[q] := (ts, vsuf);$
44: $\quad\quad decided[q] := l;$
45: $\quad\quad$ **if** $\#(readlist) = \lfloor N/2 \rfloor + 1$ **then**
46: $\quad\quad\quad (ts', vsuf') := (0, \langle\rangle);$
47: $\quad\quad\quad$ **for all** $(ts'', vsuf'') \in readlist$ **do**
48: $\quad\quad\quad\quad$ **if** $ts' < ts'' \vee (ts' = ts'' \wedge \#(vsuf') < \#(vsuf''))$ **then**
49: $\quad\quad\quad\quad\quad (ts', vsuf') := (ts'', vsuf'');$
50: $\quad\quad\quad pv := pv + vsuf';$
51: $\quad\quad\quad$ **for all** $v \in proposedValues$ **such that** $v \notin pv$ **do**
52: $\quad\quad\quad\quad pv := pv + \langle v \rangle;$
53: $\quad\quad\quad$ **for all** $p \in \Pi$ **such that** $readlist[p] \neq \bot$ **do**
54: $\quad\quad\quad\quad l' := decided[p];$
55: $\quad\quad\quad\quad$ **trigger** $\langle\ fpl, Send \mid p, [\text{Accept}, pts, suffix(pv, l'), l', t]\ \rangle;$
56: $\quad\quad$ **else if** $\#(readlist) > \lfloor N/2 \rfloor + 1$ **then**
57: $\quad\quad\quad$ **trigger** $\langle\ fpl, Send \mid q, [\text{Accept}, pts, suffix(pv, l), l, t]\ \rangle;$
58: $\quad\quad\quad$ **if** $pl \neq 0$ **then**
59: $\quad\quad\quad\quad$ **trigger** $\langle\ fpl, Send \mid q, [\text{Decide}, pts, pl, t]\ \rangle;$

60: **upon event** $\langle\ fpl, Deliver \mid q, [\text{Accept}, ts, vsuf, offs, t']\ \rangle$ **do**
61: $\quad t := max(t, t') + 1;$
62: $\quad$ **if** $ts \neq prepts$ **then**
63: $\quad\quad$ **trigger** $\langle\ fpl, Send \mid q, [\text{Nack}, ts, t]\ \rangle;$
64: $\quad$ **else**
65: $\quad\quad ats := ts;$
66: $\quad\quad$ **if** $offs < \#(av)$ **then**
67: $\quad\quad\quad av := prefix(av, offs);$ ▷ truncate sequence
68: $\quad\quad av := av + vsuf;$
69: $\quad\quad$ **trigger** $\langle\ fpl, Send \mid q, [\text{AcceptAck}, ts, \#(av), t]\ \rangle;$

70: **upon event** $\langle\ fpl, Deliver \mid q, [\text{AcceptAck}, pts', l, t']\ \rangle$ **do**
71: $\quad t := max(t, t') + 1;$
72: $\quad$ **if** $pts' = pts$ **then**
73: $\quad\quad accepted[q] := l;$
74: $\quad\quad$ **if** $pl < l \wedge \#(\{p \in \Pi \mid accepted[p] \geq l\}) > \lfloor N/2 \rfloor$ **then**
75: $\quad\quad\quad pl := l;$
76: $\quad\quad\quad$ **for all** $p \in \Pi$ **such that** $readlist[p] \neq \bot$ **do**
77: $\quad\quad\quad\quad$ **trigger** $\langle\ fpl, Send \mid p, [\text{Decide}, pts, pl, t]\ \rangle;$

78: **upon event** $\langle\ fpl, Deliver \mid q, [\text{Decide}, ts, l, t']\ \rangle$ **do**
79: $\quad t := max(t, t') + 1;$
80: $\quad$ **if** $ts = prepts$ **then**
81: $\quad\quad$ **while** $al < l$ **do**
82: $\quad\quad\quad$ **trigger** $\langle\ asc, Decide \mid av[al]\ \rangle;$ ▷ zero-based indexing
83: $\quad\quad\quad al := al + 1;$

Figure 10: Abortable Sequence Consensus

Appendix B

```
01/01 00:00:12 INFO [main] Kompics - NodesSize 9
```

```
01/01 00:00:12 DEBUG[main] s.k.i.o.VSOverlayManager - Generated assignments:
LookupTable(
0 -> [/192.168.0.1:45678, /192.168.0.2:45678, /192.168.0.3:45678]
10 -> [/192.168.0.4:45678, /192.168.0.5:45678, /192.168.0.6:45678]
20 -> [/192.168.0.7:45678, /192.168.0.8:45678, /192.168.0.9:45678]
)
[...]
01/01 00:00:18 DEBUG[main] s.k.i.ParentComponent - Slowly dying...
01/01 00:00:18 DEBUG[main] s.k.i.ParentComponent - Sending Kill to child: Component(b8ec65
01/01 00:00:18 DEBUG[main] s.k.i.ParentComponent - Sending Kill to child: Component(24aa17
01/01 00:00:18 DEBUG[main] s.k.i.ParentComponent - Sending Kill to child: Component(0125f7
01/01 00:00:18 DEBUG[main] s.k.i.ParentComponent - Sending Kill to child: Component(593211
01/01 00:00:18 DEBUG[main] s.k.i.ParentComponent - Sending Kill to child: Component(dbfeca
01/01 00:00:18 DEBUG[main] s.k.i.ParentComponent - Sending Kill to child: Component(dda185
01/01 00:00:18 DEBUG[main] s.k.i.ParentComponent - Sending Kill to child: Component(2dd228
01/01 00:00:18 DEBUG[main] s.k.i.o.VSOverlayManager - dying...
01/01 00:00:18 DEBUG[main] s.k.i.k.KVService - dying...
01/01 00:00:18 DEBUG[main] s.k.i.b.c.BebComponent - dying...
01/01 00:00:18 DEBUG[main] s.k.i.e.c.EpfdComponent - dying...
01/01 00:00:18 DEBUG[main] s.k.i.m.c.MeldComponent - dying...
01/01 00:00:18 DEBUG[main] s.k.i.m.c.multipaxos - dying...
01/01 00:00:18 DEBUG[main] s.k.i.b.BootstrapClient - dying...
01/01 00:00:18 DEBUG[main] s.k.i.ParentComponent - Got Killed event from Component(b8ec657
01/01 00:00:18 DEBUG[main] s.k.i.ParentComponent - Active set has 6 members
01/01 00:00:18 DEBUG[main] s.k.i.ParentComponent - Got Killed event from Component(24aa177
01/01 00:00:18 DEBUG[main] s.k.i.ParentComponent - Active set has 5 members
01/01 00:00:18 DEBUG[main] s.k.i.ParentComponent - Got Killed event from Component(0125f7c
01/01 00:00:18 DEBUG[main] s.k.i.ParentComponent - Active set has 4 members
01/01 00:00:18 DEBUG[main] s.k.i.ParentComponent - Got Killed event from Component(5932114
01/01 00:00:20 DEBUG[main] s.k.i.b.BootstrapServer - dying...
01/01 00:00:20 DEBUG[main] s.k.i.ParentComponent - Got Killed event from Component(fb1f38d
01/01 00:00:28 DEBUG[main] s.k.i.s.ScenarioClient - Started!
01/01 00:00:28 INFO [main] s.k.i.s.ScenarioClient - Sending GetOperation{id=88956319-d177-
01/01 00:00:28 INFO [main] s.k.i.o.VSOverlayManager - <RouteMsg, Message> Self not contain
01/01 00:00:28 INFO [main] s.k.i.b.c.BebComponent - BroadcastEvent -> From: /192.168.0.1:4
01/01 00:00:28 INFO [main] s.k.i.b.c.BebComponent - BroadcastEvent -> From: /192.168.0.1:4
01/01 00:00:28 INFO [main] s.k.i.b.c.BebComponent - BroadcastEvent -> From: /192.168.0.1:4
01/01 00:00:28 INFO [main] s.k.i.b.c.BebComponent - PL_DeliverEvent -> To: /192.168.0.4:45
01/01 00:00:28 INFO [main] s.k.i.b.c.BebComponent - PL_DeliverEvent -> To: /192.168.0.5:45
01/01 00:00:28 INFO [main] s.k.i.b.c.BebComponent - PL_DeliverEvent -> To: /192.168.0.6:45
01/01 00:00:28 INFO [main] s.k.i.o.VSOverlayManager - <BebDeliverEvent> Received by leader
01/01 00:00:28 INFO [main] s.k.i.o.VSOverlayManager - <BebDeliverEvent> Forwarding message
01/01 00:00:28 INFO [main] s.k.i.o.VSOverlayManager - <BebDeliverEvent> Forwarding message
01/01 00:00:28 INFO [main] s.k.i.o.VSOverlayManager - <RouteMsg, Message> Message delivere
01/01 00:00:28 INFO [main] s.k.i.o.VSOverlayManager - <RouteMsg, Message> Message delivere
01/01 00:00:28 INFO [main] s.k.i.k.KVService - GET operation event arrived for address /19
01/01 00:00:28 INFO [main] s.k.i.k.KVService - GET operation event arrived for address /19
01/01 00:00:28 INFO [main] s.k.i.k.KVService - GET operation event arrived for address /19
01/01 00:00:28 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=88956319
01/01 00:00:28 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=88956319
```

```
01/01 00:00:28 WARN [main] s.k.i.s.ScenarioClient - ID 88956319-d177-8fbb-b0fb-911e967c651
01/01 00:00:28 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=88956319
01/01 00:00:28 WARN [main] s.k.i.s.ScenarioClient - ID 88956319-d177-8fbb-b0fb-911e967c651
```

Appendix C

```
01/01 00:00:12 INFO [main] Kompics - NodesSize 9
01/01 00:00:12 DEBUG[main] s.k.i.o.VSOverlayManager - Generated assignments:
LookupTable(
0 -> [/192.168.0.1:45678, /192.168.0.2:45678, /192.168.0.3:45678]
10 -> [/192.168.0.4:45678, /192.168.0.5:45678, /192.168.0.6:45678]
20 -> [/192.168.0.7:45678, /192.168.0.8:45678, /192.168.0.9:45678]
)

01/01 00:00:28 DEBUG[main] s.k.i.ParentComponent - Slowly dying...
01/01 00:00:28 DEBUG[main] s.k.i.ParentComponent - Sending Kill to child: Component(b8ec65
01/01 00:00:28 DEBUG[main] s.k.i.ParentComponent - Sending Kill to child: Component(24aa17
01/01 00:00:28 DEBUG[main] s.k.i.ParentComponent - Sending Kill to child: Component(0125f7
01/01 00:00:28 DEBUG[main] s.k.i.ParentComponent - Sending Kill to child: Component(593211
01/01 00:00:28 DEBUG[main] s.k.i.ParentComponent - Sending Kill to child: Component(dbfeca
01/01 00:00:28 DEBUG[main] s.k.i.ParentComponent - Sending Kill to child: Component(dda185
01/01 00:00:28 DEBUG[main] s.k.i.ParentComponent - Sending Kill to child: Component(2dd228
01/01 00:00:28 DEBUG[main] s.k.i.o.VSOverlayManager - dying...
01/01 00:00:28 DEBUG[main] s.k.i.k.KVService - dying...
01/01 00:00:28 DEBUG[main] s.k.i.b.c.BebComponent - dying...
01/01 00:00:28 DEBUG[main] s.k.i.e.c.EpfdComponent - dying...
01/01 00:00:28 DEBUG[main] s.k.i.m.c.MeldComponent - dying...
01/01 00:00:28 DEBUG[main] s.k.i.m.c.multipaxos - dying...
01/01 00:00:28 DEBUG[main] s.k.i.b.BootstrapClient - dying...
KILL

01/01 00:00:28 DEBUG[main] s.k.i.ParentComponent - Got Killed event from Component(b8ec657

START AGAIN
01/01 00:00:49 DEBUG[main] s.k.i.ParentComponent - Active set has 7 members
01/01 00:00:49 DEBUG[main] s.k.i.ParentComponent - Started!
01/01 00:00:50 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
```

Appendix D

```
Generating LookupTable...
01/01 00:00:12 INFO [main] Kompics - NodesSize 9
01/01 00:00:12 DEBUG[main] s.k.i.o.VSOverlayManager - Generated assignments:
LookupTable(
0 -> [/192.168.0.1:45678, /192.168.0.2:45678, /192.168.0.3:45678]
10 -> [/192.168.0.4:45678, /192.168.0.5:45678, /192.168.0.6:45678]
20 -> [/192.168.0.7:45678, /192.168.0.8:45678, /192.168.0.9:45678]
)
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapServer - Seeding assignments...
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapClient - /192.168.0.2:45678 Booting up.
```

```
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapClient - /192.168.0.3:45678 Booting up.
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapClient - /192.168.0.4:45678 Booting up.
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapClient - /192.168.0.5:45678 Booting up.
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapClient - /192.168.0.6:45678 Booting up.
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapClient - /192.168.0.7:45678 Booting up.
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapClient - /192.168.0.8:45678 Booting up.
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapClient - /192.168.0.9:45678 Booting up.
01/01 00:00:12 INFO [main] s.k.i.o.VSOverlayManager - Got NodeAssignment, overlay ready.
01/01 00:00:12 INFO [main] s.k.i.o.VSOverlayManager - Got NodeAssignment, overlay ready.
01/01 00:00:12 INFO [main] s.k.i.o.VSOverlayManager - Got NodeAssignment, overlay ready.
01/01 00:00:12 INFO [main] s.k.i.o.VSOverlayManager - Got NodeAssignment, overlay ready.
01/01 00:00:12 INFO [main] s.k.i.o.VSOverlayManager - Got NodeAssignment, overlay ready.
01/01 00:00:12 INFO [main] s.k.i.o.VSOverlayManager - Got NodeAssignment, overlay ready.
01/01 00:00:12 INFO [main] s.k.i.o.VSOverlayManager - Got NodeAssignment, overlay ready.
01/01 00:00:12 INFO [main] s.k.i.o.VSOverlayManager - Got NodeAssignment, overlay ready.
01/01 00:00:12 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:12 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:12 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:12 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:12 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:12 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:12 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:12 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:15 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:15 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:16 INFO [main] s.k.i.b.BootstrapServer - 9 hosts in ready set.
01/01 00:00:16 INFO [main] s.k.i.b.BootstrapServer - Finished seeding. Bootstrapping compl
01/01 00:00:16 INFO [main] s.k.i.o.VSOverlayManager - Got NodeAssignment, overlay ready.
01/01 00:00:16 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:16 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.1
01/01 00:00:16 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.1
01/01 00:00:16 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:16 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:18 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.2
01/01 00:00:18 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.3
01/01 00:00:19 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.1
01/01 00:00:19 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.2
01/01 00:00:19 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
01/01 00:00:19 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:19 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:20 DEBUG[main] s.k.i.ParentComponent - Active set has 6 members
01/01 00:00:20 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.3
01/01 00:00:21 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
01/01 00:00:21 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:22 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
01/01 00:00:22 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.1
01/01 00:00:22 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:23 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.3
01/01 00:00:24 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
01/01 00:00:24 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
```

```
01/01 00:00:25 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
01/01 00:00:25 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.1
01/01 00:00:25 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:26 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.3
01/01 00:00:27 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
01/01 00:00:27 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:28 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
01/01 00:00:28 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.1
01/01 00:00:28 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:29 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.3
01/01 00:00:30 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
01/01 00:00:30 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:31 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
01/01 00:00:32 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.1
01/01 00:00:32 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:32 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.3
01/01 00:00:34 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
01/01 00:00:34 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:34 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
01/01 00:00:36 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.1
01/01 00:00:36 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:36 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.3
01/01 00:00:38 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
```

Appendix E

```
01/01 00:00:12 INFO [main] Kompics - NodesSize 9
01/01 00:00:12 DEBUG[main] s.k.i.o.VSOverlayManager - Generated assignments:
LookupTable(
0 -> [/192.168.0.1:45678, /192.168.0.2:45678, /192.168.0.3:45678]
10 -> [/192.168.0.4:45678, /192.168.0.5:45678, /192.168.0.6:45678]
20 -> [/192.168.0.7:45678, /192.168.0.8:45678, /192.168.0.9:45678]
)
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapServer - Seeding assignments...
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapClient - /192.168.0.2:45678 Booting up.
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapClient - /192.168.0.3:45678 Booting up.
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapClient - /192.168.0.4:45678 Booting up.
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapClient - /192.168.0.5:45678 Booting up.
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapClient - /192.168.0.6:45678 Booting up.
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapClient - /192.168.0.7:45678 Booting up.
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapClient - /192.168.0.8:45678 Booting up.
01/01 00:00:12 INFO [main] s.k.i.b.BootstrapClient - /192.168.0.9:45678 Booting up.
01/01 00:00:18 DEBUG[main] s.k.i.s.ScenarioClient - Started!
01/01 00:00:18 INFO [main] s.k.i.s.ScenarioClient - Sending PutOperation{id=d7776a15-ba53-
01/01 00:00:18 INFO [main] s.k.i.o.VSOverlayManager - <RouteMsg, Message> Self not contain
01/01 00:00:18 INFO [main] s.k.i.b.c.BebComponent - BroadcastEvent -> From: /192.168.0.1:4
01/01 00:00:18 INFO [main] s.k.i.b.c.BebComponent - BroadcastEvent -> From: /192.168.0.1:4
01/01 00:00:18 INFO [main] s.k.i.b.c.BebComponent - BroadcastEvent -> From: /192.168.0.1:4
01/01 00:00:18 INFO [main] s.k.i.b.c.BebComponent - PL_DeliverEvent -> To: /192.168.0.4:45
01/01 00:00:18 INFO [main] s.k.i.b.c.BebComponent - PL_DeliverEvent -> To: /192.168.0.5:45
01/01 00:00:18 INFO [main] s.k.i.b.c.BebComponent - PL_DeliverEvent -> To: /192.168.0.6:45
```

```
01/01 00:00:18 INFO [main] s.k.i.o.VSOverlayManager - <BebDeliverEvent> Received by leader
01/01 00:00:18 INFO [main] s.k.i.o.VSOverlayManager - <BebDeliverEvent> Forwarding message
01/01 00:00:18 INFO [main] s.k.i.o.VSOverlayManager - <BebDeliverEvent> Forwarding message
01/01 00:00:18 INFO [main] s.k.i.o.VSOverlayManager - <RouteMsg, Message> Message delivere
01/01 00:00:18 INFO [main] s.k.i.o.VSOverlayManager - <RouteMsg, Message> Message delivere
01/01 00:00:18 INFO [main] s.k.i.o.VSOverlayManager - <AscDecide> A decide event was recei
01/01 00:00:18 INFO [main] s.k.i.o.VSOverlayManager - <AscDecide> A decide event was recei
01/01 00:00:18 INFO [main] s.k.i.o.VSOverlayManager - <AscDecide> A decide event was recei
01/01 00:00:18 INFO [main] s.k.i.k.KVService - PUT operation event arrived for address /19
01/01 00:00:18 INFO [main] s.k.i.k.KVService - PUT operation event arrived for address /19
01/01 00:00:18 INFO [main] s.k.i.k.KVService - PUT operation event arrived for address /19
01/01 00:00:18 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=d7776a15
01/01 00:00:18 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=d7776a15
01/01 00:00:18 WARN [main] s.k.i.s.ScenarioClient - ID d7776a15-ba53-9967-1088-b50f710d313
01/01 00:00:18 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=d7776a15
01/01 00:00:18 WARN [main] s.k.i.s.ScenarioClient - ID d7776a15-ba53-9967-1088-b50f710d313
01/01 00:00:38 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
01/01 00:00:38 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient - Started!
01/01 00:00:38 INFO [main] s.k.i.s.ScenarioClient - Sending PutOperation{id=f63b1654-8d48-
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient2 - Started!
01/01 00:00:38 INFO [main] s.k.i.s.ScenarioClient2 - Sending CasOperation{id=a907d222-3390
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <MeldLeaderEvent> New leader elected
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <RouteMsg, Message> Self not contain
01/01 00:00:38 INFO [main] s.k.i.b.c.BebComponent - BroadcastEvent -> From: /192.168.0.1:4
01/01 00:00:38 INFO [main] s.k.i.b.c.BebComponent - BroadcastEvent -> From: /192.168.0.1:4
01/01 00:00:38 INFO [main] s.k.i.b.c.BebComponent - BroadcastEvent -> From: /192.168.0.1:4
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <RouteMsg, Message> Self not contain
01/01 00:00:38 INFO [main] s.k.i.b.c.BebComponent - BroadcastEvent -> From: /192.168.0.1:4
01/01 00:00:38 INFO [main] s.k.i.b.c.BebComponent - BroadcastEvent -> From: /192.168.0.1:4
01/01 00:00:38 INFO [main] s.k.i.b.c.BebComponent - BroadcastEvent -> From: /192.168.0.1:4
01/01 00:00:38 INFO [main] s.k.i.b.c.BebComponent - PL_DeliverEvent -> To: /192.168.0.4:45
01/01 00:00:38 INFO [main] s.k.i.b.c.BebComponent - PL_DeliverEvent -> To: /192.168.0.5:45
01/01 00:00:38 INFO [main] s.k.i.b.c.BebComponent - PL_DeliverEvent -> To: /192.168.0.6:45
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <BebDeliverEvent> Received by leader
01/01 00:00:38 INFO [main] s.k.i.b.c.BebComponent - PL_DeliverEvent -> To: /192.168.0.4:45
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <BebDeliverEvent> Forwarding message
01/01 00:00:38 INFO [main] s.k.i.b.c.BebComponent - PL_DeliverEvent -> To: /192.168.0.5:45
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <BebDeliverEvent> Forwarding message
01/01 00:00:38 INFO [main] s.k.i.b.c.BebComponent - PL_DeliverEvent -> To: /192.168.0.6:45
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <BebDeliverEvent> Received by leader
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <BebDeliverEvent> Forwarding message
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <BebDeliverEvent> Forwarding message
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <RouteMsg, Message> Message delivere
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <RouteMsg, Message> Message delivere
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <RouteMsg, Message> Message delivere
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <RouteMsg, Message> Message delivere
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <AscDecide> A decide event was recei
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <AscDecide> A decide event was recei
```

```
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <AscDecide> A decide event was recei
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <AscDecide> A decide event was recei
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <AscDecide> A decide event was recei
01/01 00:00:38 INFO [main] s.k.i.o.VSOverlayManager - <AscDecide> A decide event was recei
01/01 00:00:38 INFO [main] s.k.i.k.KVService - PUT operation event arrived for address /19
01/01 00:00:38 INFO [main] s.k.i.k.KVService - PUT operation event arrived for address /19
01/01 00:00:38 INFO [main] s.k.i.k.KVService - PUT operation event arrived for address /19
01/01 00:00:38 INFO [main] s.k.i.k.KVService - CAS operation event arrived for address /19
01/01 00:00:38 INFO [main] s.k.i.k.KVService - CAS operation event arrived for address /19
01/01 00:00:38 INFO [main] s.k.i.k.KVService - CAS operation event arrived for address /19
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=f63b1654
01/01 00:00:38 WARN [main] s.k.i.s.ScenarioClient - ID f63b1654-8d48-d626-0fba-802aea080f0
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=f63b1654
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient2 - Got OpResponse: OpResponse{id=f63b165
01/01 00:00:38 WARN [main] s.k.i.s.ScenarioClient2 - ID f63b1654-8d48-d626-0fba-802aea080f
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=f63b1654
01/01 00:00:38 WARN [main] s.k.i.s.ScenarioClient - ID f63b1654-8d48-d626-0fba-802aea080f0
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=f63b1654
01/01 00:00:38 WARN [main] s.k.i.s.ScenarioClient - ID f63b1654-8d48-d626-0fba-802aea080f0
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient2 - Got OpResponse: OpResponse{id=f63b165
01/01 00:00:38 WARN [main] s.k.i.s.ScenarioClient2 - ID f63b1654-8d48-d626-0fba-802aea080f
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=f63b1654
01/01 00:00:38 WARN [main] s.k.i.s.ScenarioClient - ID f63b1654-8d48-d626-0fba-802aea080f0
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=f63b1654
01/01 00:00:38 WARN [main] s.k.i.s.ScenarioClient - ID f63b1654-8d48-d626-0fba-802aea080f0
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient2 - Got OpResponse: OpResponse{id=f63b165
01/01 00:00:38 WARN [main] s.k.i.s.ScenarioClient2 - ID f63b1654-8d48-d626-0fba-802aea080f
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=a907d222
01/01 00:00:38 WARN [main] s.k.i.s.ScenarioClient - ID a907d222-3390-9509-38a6-42071644a4e
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=a907d222
01/01 00:00:38 WARN [main] s.k.i.s.ScenarioClient - ID a907d222-3390-9509-38a6-42071644a4e
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient2 - Got OpResponse: OpResponse{id=a907d22
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=a907d222
01/01 00:00:38 WARN [main] s.k.i.s.ScenarioClient - ID a907d222-3390-9509-38a6-42071644a4e
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=a907d222
01/01 00:00:38 WARN [main] s.k.i.s.ScenarioClient - ID a907d222-3390-9509-38a6-42071644a4e
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient2 - Got OpResponse: OpResponse{id=a907d22
01/01 00:00:38 WARN [main] s.k.i.s.ScenarioClient2 - ID a907d222-3390-9509-38a6-42071644a4
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=a907d222
01/01 00:00:38 WARN [main] s.k.i.s.ScenarioClient - ID a907d222-3390-9509-38a6-42071644a4e
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient - Got OpResponse: OpResponse{id=a907d222
01/01 00:00:38 WARN [main] s.k.i.s.ScenarioClient - ID a907d222-3390-9509-38a6-42071644a4e
01/01 00:00:38 DEBUG[main] s.k.i.s.ScenarioClient2 - Got OpResponse: OpResponse{id=a907d22
01/01 00:00:38 WARN [main] s.k.i.s.ScenarioClient2 - ID a907d222-3390-9509-38a6-42071644a4
01/01 00:00:38 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
01/01 00:00:38 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address removed: /192.168.0
01/01 00:00:40 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.1
01/01 00:00:40 INFO [main] s.k.i.e.c.EpfdComponent - Suspected address added: /192.168.0.1
```