Brief Announcement: Leader Election for Arbitrarily Connected Networks in the Presence of Process Crashes and Weak Channel Reliability

Sergio Rajsbaum[†], Michel Raynal[⋆,‡], Karla Vargas[†]

[†]Instituto de Matematicas, UNAM, Mexico
[⋆]Univ Rennes IRISA, 35042 Rennes, France
[‡]Department of Computing, Polytechnic University, Hong Kong

1 Introduction

In the leader election problem each process p_i has a local variable $leader_i$, and it is required that all the local variables $leader_i$ forever contain the same identity, which is the identity of one of the processes.If processes may crash, the system is fully asynchronous, and the elected leader must be a process that does not crash, leader election cannot be solved [4]. Not only the system must no longer be fully asynchronous, but the leader election problem must be weakened to the *eventual leader election problem*. This problem is denoted Ω in the failure detector parlance [1,2]. Notice that the algorithm must elect a new leader each time the previously elected leader crashes.

ADD channels were introduced in [5], as a realistic partially synchronous model of channels that can lose and reorder messages. Each channel guarantees that some subset of the messages sent on it will be delivered in a timely manner and such messages are not too sparsely distributed in time. More precisely, for each channel there exist two constants K and D, not known to the processes (and not necessarily the same for all channels), such that for every K consecutive messages sent in one direction, at least one is delivered within D time units after it has been sent.

Even though ADD channels seem so weak, it is possible to implement an *eventually perfect failure detector* in an arbitrarily connected network of ADD channels [3]. A implementation of $\Diamond P$ using messages of size $O(n \log n)$ in the same model was presented [6]. The goal of this paper is move from $\Diamond P$ to Ω using messages of $O(log\ n)$.

This paper shows that it is possible to implement Ω in an arbitrarily connected network of eventual ADD channels where asynchronous processes may fail by crashing using messages of $O(log\ n)$. Then, we propose an implementation of Ω in networks with unknown membership whose messages are eventually of size $O(log\ n)$ too.

Designing leader election ADD-based algorithms using messages whose size is bounded, is a difficult challenge since while the constants K and D do exist. We found it even more challenging to work under the assumption that some edges might not satisfy any property at all; our algorithm works under the assumption that only edges on an (unknown to the processes) spanning tree are guaranteed to comply with the ADD property.

2 Model of Computation

The system consists of a finite set of processes $\Pi = \{p_1, p_2, ..., p_n\}$. Any number of processes may fail by crashing. A process is *correct* if it does not crash, otherwise, it is *faulty*. The communication network is represented by a directed graph $G = (\Pi, E)$, where an edge $(p_i, p_j) \in E$ means that there is a unidirectional channel that allows the process p_i to send messages to p_j. Is required the existence of a spanning tree containing all correct processes and the root being the leader, i.e. the correct process with the smallest identity.

A directed channel (p_i, p_j) satisfies the *ADD property* if there are two constants K and D (unknown to the processes such that for every K consecutive messages sent by p_i to p_j, at least one is delivered to p_j within D time units after it has been sent. The other messages from p_i to p_j can be lost or experience arbitrary delays.

```
initialization                                          ----Code for p_i----
(1)  leader_i ← i; hopbound_i[i] ← n; set timer_i[i, n] to +∞;
(2)  for each j ∈ {1, ··· , n} \ {i} and each x ∈ {1, ··· , n}
(3)    do timeout_i[j, x] ← any positive integer; set timer_i[j, x] to timeout_i[j, hb];
(4)    set penalty_i[j, x] to −1; hopbound_i[j] ← 0
(5)  end for.

(6)  every T time units of clock_i() do
(7)    if (hopbound_i[leader_i] > 1)
(8)      then for each j ∈ out_neighbors_i
                  do send ALIVE(leader_i, hopbound_i[leader_i] − 1) to p_j end for
(9)    end if.

(10) when ALIVE(ℓ, hb ← n − k) such that ℓ ≠ i is received   % from a process in in_neighbors_i
(11)   if (ℓ ≤ leader_i)
(12)     then leader_i ← ℓ;
(13)          if ([timer_i[leader_i, hb] expired)
                 then increase the value of timeout_i[leader_i, hb] end if;
(14)          set timer_i[leader_i, hb] to timeout_i[leader_i, hb];
(15)          not_expired_i ← {x | timer_i[leader_i, x] not expired };
(16)          hopbound_i[leader_i] ←
                 max{x ∈ not_expired with smallest non-negative penalty_i[leader_i, x]}
(17)   end if.

(18) when timer_i[leader_i, hb] expires and (leader_i ≠ i) do
(19)     penalty_i[leader_i, hb] ← penalty_i[leader_i, hb] + 1;
(20)     if ( ∧_{1≤x≤n} ([timer_i[leader_i, x] expired))
(21)          then leader_i ← i
(22)          else  same as lines 15-16
(23)     end if.
```

Algorithm 1: Eventual leader election in the ◇ADD model with known membership

3 An Algorithm for Eventual Leader Election in the $\Diamond$ADD Model with Unknown Membership

The second algorithm (only described here due space limitations) solves eventual leader election in the $\Diamond$ADD model with unknown membership, which means that, initially, a process knows nothing about the network, it knows only its input/output channels.

Initially p_i communicate its identity to its neighbors. Once its neighbors know about it, p_i no longer send its identity. And the same with other names that p_i learns. For that, p_i keeps a *pending set* for every channel connected to it that helps it to keep track of the information that it needs to send to its neighbors. So initially, p_i adds the pair $(\texttt{new}, i)$ to every pending set.

When process p_i receives an $\text{ALIVE}()$ message from p_j, this message can contain information about the leader and the corresponding pending set that p_j saves for p_i. First, p_i processes the information contained in the pending set and then processes the information about the leader. If p_i finds a pair with a name labeled as new and does not know it, it stores the new name in the set $knonw_i$, increases its hopbound value, and adds to every pending set (except to the one belonging to p_j) this information labeled as new. In any case, p_i needs to communicate p_j that it already knows that information, so p_i adds this information to the pending set of p_j but labeled as an acknowledgment.

When p_j receives $name$ labeled as an acknowledgment from p_i, i.e. $(\texttt{ack}, name)$, it stops sending the pair $(\texttt{new}, name)$ to it, so it deletes that pair from p_i's pending set. Eventually, p_i receives a pending set from p_j not including $(\texttt{new}, name)$, so p_i deletes $(\texttt{ack}, name)$ from p_j's pending set.

As in Algorithm 1, every process keeps as leader a process with minimum id. This part is similar to Algorithm 1, only ignoring the penalties since we are assuming that all channels are $\Diamond$ ADD.

4 Conclusion

The $\Diamond$ADD model is a particularly weak partially synchronous communication model. Assuming first that the correct processes are connected by a spanning tree made up of $\Diamond$ ADD channels, this article has presented an algorithm that elects an eventual leader, using messages of only size $O(\log n)$.

References

1. Chandra T.D., Hadzilacos V., and Toueg S., The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685-722 (1996)
2. Chandra T.D. and Toueg S., Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225-267 (1996)
3. Kumar S. and Welch J.L., Implementing $\Diamond P$ with bounded messages on a network of ADD channels. *Parallel Processing Letters*, Vol. 29(1) 1950002, 12 pages (2019)
4. Raynal M., *Fault-tolerant message-passing distributed systems: an algorithmic approach. Springer*, 492 pages, ISBN 978-3-319-94140-0 (2018)
5. Sastry S. and Pike S.M., Eventually perfect failure detectors using ADD channels. *5th Int'l Symposium on Parallel and Distributed Processing and Applications (ISPA'07)*, Springer LNCS 4742, pp. 483-496 (2007)
6. Vargas K. and Rajsbaum S., An eventually perfect failure detector for networks of arbitrary topology connected with ADD channels using time-to-live values. *49th IEEE/IFIP Int'l Conference on Dependable Systems and Networks (DSN'19)*, IEEE Press, pp. 264-275 (2019)