Brief Announcement: Low-Communication Self-Stabilizing Leader Election in Large Networks

Thamer Alsulaiman[1], Andrew Berns[2], and Sukumar Ghosh[1]

[1] Department of Computer Science, The University of Iowa
[thamer-alsulaiman, sukumar-ghosh]@uiowa.edu
[2] Department of Computer Science, The University of Wisconsin – La Crosse
aberns@uwlax.edu

Abstract. This paper makes two contributions: (1) On a completely connected network of n anonymous processes, it presents a synchronous randomized algorithm to solve the *weak leader election* problem in expected $O(1)$ time using $O(\sqrt{n}\log n)$ messages with high probability, (2) It presents a self-stabilizing algorithm for the *strong leader election* problem on a completely connected network of processes with unique ids – it stabilizes in expected $O(1)$ number of rounds (or in $O(\log n)$ rounds with high probability) using $O(n)$ messages. In doing so, the algorithm also solves the stabilizing unison problem.

1 Introduction

Given a network of processes, the goal of leader election is to elect a unique process as the leader, and notify every non-leader process about the identity of the leader. This version of the problem is called the *strong* or *explicit* leader election. There is a weaker version of this problem, known as the *weak* or *implicit* leader election that waives the requirement of notification. On a completely connected network of n processes communicating via message passing, Kutten et al. [2] presented a synchronous randomized algorithm that elects a leader in $O(1)$ rounds with a message complexity of $O(\sqrt{n}\log^{\frac{3}{2}} n)$ with high probability. In this paper, we present a variation of [2] that elects a leader in expected $O(1)$ rounds using $O(\sqrt{n}\log n)$ messages with high probability. Subsequently, we present a self-stabilizing algorithm for the strong leader election problem – it stabilizes in expected $O(1)$ number of rounds (or in $O(\log n)$ rounds with high probability) using $O(n)$ messages, which is optimal.

2 Weak Leader Election

We use the synchronous message-passing model $\mathcal{CONGEST}$ on a completely-connected topology. We assume all nodes know an upper bound on n, the number of processes in the system. Algorithm 1 sketches our weak leader election algorithm.

Algorithm 1 Weak leader election

Variables: $leader(i)$ (initially $\perp$), $voted(i)$ (initially $false$)

Round 1a

Each node becomes an *observer* with probability $\frac{2\log n}{n}$.

Round 1b

Each observer becomes a *candidate* with probability $\frac{1}{\log n}$. Each candidate randomly chooses an *id* from $\{0, 1, \ldots, n^2\}$. Call it *rid*.

Round 2

1. Each candidate sends an election message of (*rid, election*) to a randomly chosen set of $2\sqrt{n}\log n$ neighbors. Call them *voters*. Upon receipt, each voter sets $voted(i)$ to $true$.
2. Each observer sends an *observe* message to a randomly chosen set of $\sqrt{n}$ processes. Call them *agents*.

Round 3

1. Each agent i sends to its observer the value of $voted(i)$.
2. If $leader(i) = \perp$, voter i sends an acknowledgment to the candidate with the largest value of *rid*.

Round 4

1. If a candidate receives $2\sqrt{n}\log n$ acknowledgments, it becomes a leader.
2. If an observer node u does not receive at least one message from an agent i with $voted(i) = true$, then u repeats Algorithm 1 beginning at Round 1b.

In Algorithm 1, a process becomes a *candidate* for election in two steps: first it becomes an *observer* and subsequently zero or more of the observers are chosen as candidates. At least one node becomes an observer with high probability. The observers check if a candidate was elected, and if no candidate is detected, eventually one or more of them choose themselves as candidates for election in a future round.

Theorem 1. *Algorithm 1 solves the implicit leader election problem with high probability. The expected running time of Algorithm 1 is $O(1)$ rounds, while the message complexity is $O(\sqrt{n}\log n)$ in expectation.*

The proof of this theorem can be sketched by noting that, given there is at least one candidate, with high probability, each observer detects its existence, and no observer from Round 1*b* becomes a leader thereafter. The expected number of candidates after Round 1b is constant. Furthermore, with high probability at least one node has become a candidate in $O(\log^3 n)$ rounds, and with high probability, only one candidate receives all acknowledgements.

3 Self-Stabilizing Leader Election

We use a modified version of the weak leader election algorithm to elect a unique *monitor*. The monitor collects the system's state and, if illegal, declares itself as the new leader. These steps will be indefinitely repeated, as in Katz and Perry's

paper [1], although a different monitor may be picked in each iteration. Algorithm 2 contains our full program. We assume in this algorithm that all processes have a unique identifier. Note that, since strong leader election has a lower bound of $\Omega(n)$, we use a simpler version of the monitor election algorithm.

Algorithm 2 Self-Stabilizing Leader Election

Execute the following actions continuously based upon variable $round(i)$. Increment $round(i)$ after each round.

Round 0

With probability $\frac{1}{n}$, each process i sends an election message containing $(id_i, election, round(i))$ to every other process. If there is a recipient j such that $round(j) \neq 0$, then upon receipt of the election message, it resets its round number to 0, and clears all incoming channels.

Round 1

Every recipient of an election message returns an *ack* to the election message sender with the largest identifier.

Round 2

The process that receives $(n-1)$ *ack* messages takes up the role of election monitor, and sends out a *query* to every other process, asking them to send the identifier of their leader.

Round 3

A process receiving one or more *query* messages replies with its leader identifier to the query message sender with the largest identifier.

Round 4

A process that receives $(n-1)$ query response messages checks the legality of the configuration. If the configuration is legal then it takes no action. If the configuration is illegal, it sets itself as the new leader and notifies every other process. A process resets its leader only when it receives exactly one such notification.

Theorem 2. *Algorithm 2 is a self-stabilizing leader election algorithm. A legal configuration is reached in expected $O(1)$ rounds, and $O(\log n)$ rounds with high probability. The algorithm exchanges $O(n)$ messages if channels are initially clear, or $O(n^2)$ messages if the channels initially contain arbitrary messages, before reaching a legal configuration.*

The proof of this theorem begins by noting that an election message is sent out in an expected constant number of rounds, at which point the phase clocks are synchronized. The remainder of the proof parallels that of Theorem 1.

References

1. Katz, S., Perry, K.J.: Self-stabilizing extensions for message-passing systems. Distrib. Comput. 7(1), 17–26 (Nov 1993), http://dx.doi.org/10.1007/BF02278852
2. Kutten, S., Pandurangan, G., Peleg, D., Robinson, P., Trehan, A.: Sublinear bounds for randomized leader election. In: Frey, D., Raynal, M., Sarkar, S., Shyamasundar, R.K., Sinha, P. (eds.) ICDCN. Lecture Notes in Computer Science, vol. 7730, pp. 348–362. Springer (2013)