Minimizing the Amount of Shared Memory for an Election Protocol

Grégory Mermoud, Marc A. Schaub, and Grégory Théoduloz

School of Computer and Communication Sciences,
Ecole Polytechnique Fédérale de Lausanne (EPFL),
1015 Lausanne, Switzerland

Abstract. We consider a particular application of asynchronous leader election protocols using shared memory to a hypothetical pope election process. We propose an algorithm using 2 bits of shared memory which runs in quadratic time. We introduce the notion of cheating and propose a variation of our algorithm which is resistant to cheating while still running in polynomial time. We establish a lower bound for the running time in the case where shared memory is unlimited and introduce an algorithm which runs in linear time. Finally we compare the cost of the three algorithms and discuss their respective tradeoffs.

1 Introduction

Herein we describe different protocols solving the problem of electing a leader within an asynchronous distributed system. Our problem statement is represented by a parabol in which processes become cardinals, shared memory bits become switches, the success bit becomes some white smoke in the sky and the election winner becomes Pope.

The n cardinals are gathered in order to elect a new Pope among themselves. Each cardinal c_i has a particular vote intention F_i that was somehow inspired by the Holy Spirit. All cardinals are completely silent and their only mean of communication is a set of s switches located in the Sixtin Chapel. They have a private memory. The election process is divided into atomic steps that go as follows: a cardinal c_i enters the chapel and can do the following actions and exits.

- Modify the switches as desired.
- Put his ballot V_i in the box. Each cardinal can do that only once.
- Launch the white smoke to announce to the world that a new Pope is elected by setting a shared write-only bit T to 1.

There is no assumption on the order of the cardinals. Let a scheduler S choose the next cardinal to enter the chapel. A *fair* scheduler does not neglect any

cardinal forever; a *b-bounded fair* scheduler does not neglect any cardinal for more than b steps, for some integer bound $b \geq n$.

We want to find an election protocol respecting the following requirements:

1. The election must be unanimous, that is, when the white smoke appears, then all ballet papers must designate the same cardinal as Pope. Formally, when $T = 1$, then $V_i = V_j$ for all $1 \leq i, j \leq n$.
2. The number of intentions of the winner is maximal. Formally, when $T = 1$, then $| \{j \mid F_j = V_1\} | \geq | \{j \mid F_j = i\} |$ for all $1 \leq i \leq n$.
3. The election must terminate within a finite number of steps, that is, if the scheduler is fair, then $T = 1$ within a finite number of steps.

We do not know how the switches are set up at the beginning. The cardinals know the number n of cardinals, but they are oblivious to the choices made by the scheduler.

First, we are looking for the smallest number s of switches necessary to solve the problem with a fair scheduler. Moreover, assuming a b-bounded scheduler, we are interested in the *cost* $s \cdot t_b$ of a given solution, where t_b is the maximal number of steps till termination. Section 2 presents an algorithm using only 2 switches. Section 3 provides a formal definition of cheating and discusses a variant of our 2-switches algorithm that is cheat-resistant. Section 4 discusses the problem of finding a lower bound for the duration of the election process and compares the performances of the algorithms presented in this contribution.

2 2-switch Solution

2.1 Algorithm

This algorithm uses 2 switches to encode four states called Ⓐ, Ⓑ, ⓪ and ①. One special cardinal is called *leader cardinal*; all other cardinals are called *normal cardinals*. The leader cardinal plays a central role in the algorithm. During the *vote intention collection phase*, he collects the number of votes every cardinal gets by asking all normal cardinals to tell him if they intend to vote for this cardinal or not. This information allows him to determine which cardinal has obtained the most vote intentions. During the *voting phase*, he tells all other cardinals who has obtained the most vote intentions. Since the inital state of the shared memory is unknown, an *initialization phase* has to make sure that the state is set to Ⓐ at the begining of the vote intention collection phase and that every cardinal knows when the vote intention collection phase starts.

Initialization The general idea of this phase is that every cardinal informs the leader cardinal exactly once that he has done the initialization step and

considers himself as initialized once the leader cardinal has acknowledged this information. The leader cardinal counts the number of acknowledgements he sends, which allows him to know when all other cardinals are initialized.

When a cardinal which has not yet done his initialization sees that the state is ⓪ or Ⓐ, he sets it to ① and waits for the leader cardinal's acknowledgement. If this cardinal then sees Ⓑ, he knows that the leader cardinal has acknowledged his initialization and waits for a change to state Ⓐ indicating that the second phase has started. If he sees ⓪, he knows that the initialization has not been acknowledged. He therefore starts the same process again by setting the state to ①.

When the leader cardinal enters the chapel for the first time, he doesn't know if he is the first cardinal to enter the chapel or not. He always sets the state to ⓪, which means that if another cardinal is waiting on an acknowledgement, this acknowledgement fails. Afterwards, when the leader cardinal sees ①, he acknowledges the initialization by setting the state to Ⓑ and increases the *initialized cardinals counter* in his memory. Once the initialized cardinals counter is equal to the number of normal cardinals and the state is ⓪, every cardinal knows that he is initialized and the special cardinal sets the state to Ⓐ, which starts the vote intention collection phase.

Vote Intention Collection We separate the vote intention collection phase into successive *rounds*. During a round, the special cardinal collects binary answers to a specific question from every normal cardinal. We use one round for every cardinal and the question therefore is: *are you voting for the current cardinal?* The leader collects the number of votes obtained by the current cardinal. Each cardinal keeps track of who is the current cardinal for whom vote intentions are currently being collected in his private memory. The states ① and ⓪ are used by a cardinal to tell the leader cardinal if the current cardinal is his favorite or not. The other two states are alternatively used as *waiting states*: during the first round, Ⓐ is the waiting state, then for the second round, Ⓑ is the waiting state and so on. If a cardinal who has not yet voted in the current round sees that the system is in the current waiting state, he sets the state to either ① or ⓪ depending on his read-only preference, increments the counter telling him which cardinal he is currently voting for and starts waiting on the next waiting state. When the leader cardinal sees either state ⓪ or ①, he increases the counter of cardinal who have given their intention for the current cardinal and, if appropriate, the number of vote intentions the current cardinal gets. He then switches the system back to the current waiting state unless all other cardinals have given their intention. In this case he increments the counter telling him who he is currently collecting vote intentions for and switches to the next waiting state, which starts the next round. Once he has collected vote intentions for all cardinals, he knows who has won the election and it is possible to move to the voting phase of the algorithm. Similarly, when a normal cardinal notices that he

has given his intention for every cardinal, he switches to the voting phase and waits for what would have been the next waiting state.

Voting In this phase, the leader cardinal sends the number of the winning cardinal to every other cardinal one at a time. When a cardinal has received this information, he casts his ballot. Once the leader cardinal has sent the information to all cardinals, he votes himself and then launches the white smoke.

In order to send information in serially, the states Ⓐ and Ⓑ are used as *channel available state* and *acknowledgement state*. The acknowledgement state is equal to the last waiting state in the vote intention collection phase. When a cardinal who has not voted yet sees the channel available state, he sets it to the acknowledgement state to indicate that he has reserved the channel. When the leader cardinal sees an acknowledgement state, he starts transmiting the number of the winner one bit at a time using states ⓪ and ① and waiting for an acknowledgement from the receiving cardinal for each bit. Since the length of the message is know beforehands, both cardinal know when they have sent respectively received the whole information. The receiving cardinal then votes and doesn't do anything in the case he enters the chapel again afterwards. The leader cardinal increases the *having voted cardinal counter* in his memory and sets the shared memory back to the *channel available state*. If the having voted counter is equal to the number of normal cardinals, the special cardinal casts his ballot and launches the smoke, which succesfully concludes the algorithm.

The complete pseudo-code of this algorithm is given in Appendix A.

2.2 Proof of Correctness

We must prove that the proposed algorithm meets the safety and liveness requirements given in Section 1. We decompose the proof into three parts corresponding to the three phases of the algorithm. We prove that each phase eventually terminates assuming a fair scheduler and that the safety properties hold when the smoke is released. For the vote intention collection phase, we give a detailed proof sketch; for the initialization and voting phases, the proof schema is similar and therefore we give only the underlying intuitions.

Beforehand we assume the following definitions. The leading cardinal is denoted by c_L. At a given point in time, a cardinal *is waiting on a state* s if he does not modify anything if the state is different from s. Let W_s be the set of cardinals that are waiting on the state s. We define the *opposite of a waiting state* w, denoted by $\bar{w}$ to be such that if w is Ⓐ, $\bar{w}$ is Ⓑ, and reciprocally.

Vote Intention Collection We show that the votes are properly transmitted to the leading cardinal and that this phase eventually terminates.

From the definition of the algorithm, we know that the leading cardinal must maintain the following counters: the counter *current* whose value is the cardinal for which intentions are currently being collected, the counter *seen* whose value is the number of cardinals whose intentions have been recorded and the counter *int* whose value is the number of recorded intentions in favor of cardinal *current* – the cardinal for which we are currently collecting intentions. Moreover, normal cardinals maintain their own, private counter *current* whose value is the number of the cardinal for which he will give its intention when the state is the appropriate waiting state. We show that the counters have an appropriate value along the round and that normal cardinals are all giving their vote before moving to the next round.

First of all, we can do the following observations. Every round in this phase has an associated waiting state ω. The leader ends the round as finished when he sets the state to $\bar{\omega}$ and internally increments his *current* counter. A normal cardinal increments his *current* counter when he starts waiting on $\bar{\omega}$. Additionally, at any time during a round, the set W_ω contains all cardinals that have not given their information yet and the set $W_{\bar{\omega}}$, all normal cardinals that have already given their information.

The invariant that we prove states that the counters maintained by the leading cardinal and the normal cardinals have a value that reflects the current state of every cardinal. Assume that the value of the leading cardinal's *current* counter is i. Formally, if the current state is ω (i.e., the leading cardinal has taken into account the last intention), the counters are as follows: $seen = |W_{\bar{\omega}}|$ and $int = \sum_{j \in W_{\bar{\omega}} \cup \{c_L\}} \mathbf{1}_{F_j = i}$. If the current state is a data state (⓪ or ①), and let P be the last cardinal to have given his intention, then P is in $W_{\bar{\omega}}$ and the counters are as follows: $seen = |W_{\bar{\omega}}| - 1$ and $int = \sum_{j \in W_{\bar{\omega}} \cup \{c_L\}, j \neq P} \mathbf{1}_{F_j = i}$. The state cannot be $\bar{\omega}$ because when the leading cardinal sets the state to $\bar{\omega}$, we move to the next round. Moreover, normal cardinals waiting for ω have a value for the *current* counter identical to the one of the leading cardinal's *current* counter whilst normal cardinals waiting for $\bar{\omega}$ have already the next value for their *current* counter.

At the beginning of this phase, assuming the initialization phase succeeded, every cardinal is waiting on the state Ⓐ, which is the waiting state of the first round (ω = Ⓐ). Therefore, W_ω initially contains all normal cardinals whilst $W_{\bar{\omega}}$ is empty. From the specification of the algorithm, we then notice that this invariant holds during the course of the round[1]. Moreover, the leading cardinal ends the round by setting the state to $\bar{\omega}$ when *seen* has the value $n-1$. At this point, W_ω is empty whilst $W_{\bar{\omega}}$ contains all normal cardinals. Consequently, every cardinal is waiting on the appropriate state and has a correct value for his *current* counter. Since the waiting state of the next round is $\bar{\omega}$, the invariant still holds at the beginning of the next round.

[1] A formal proof of this fact would consider every possible transition at a given time and show that if they are taken and the invariant holds, the invariant still holds after.

Knowing that this invariant holds, we can notice that the algorithm eventually terminates given that the scheduler is fair: no cardinals is waiting on a state that will never appear again. Moreover, the cardinals that have already given their intention will not interfere with the others since the next waiting state $\bar{\omega}$ will only be set once every normal cardinal has given its intention,. Finally, upon termination of the phase, due to the successive invariants holding, the intention counters for every candidate cardinals has the expected value.

Initialization We must show that at the end of the initialization phase, every normal cardinal is waiting on an Ⓐ state and that the leading cardinal has just set the state to Ⓐ. Due to the special care taken for being sure that every cardinal successfully notifies the fact that he is initialized to the leader exactly once, the initialization phase result in the expected result.

Voting Let the free-channel state f be the opposite of the waiting state of the last round. From the invariant that is valid for the last round of the vote intention collection phase, W_f contains all normal cardinals. Considering the counter *transd* of cardinals to which the identity of the pope has been communicated, we can write the same kind of invariant as before. We do not give any further details for this part. The crucial point is that the message sent to the cardinals are of a fixed length and therefore, we do not need a termination state. The smoke is ensured to be sent at a moment at which every cardinal has voted because the leading cardinal waits for the last acknowledgment from the last cardinal before sending the smoke. Since the last cardinals puts his vote in the ballot box at the moment at which he acknowledges the last bit, the number of ballots inside the box is truly n. Unanimity is guaranteed due to the leading cardinal sending the very same name to every normal cardinal. Finally, the winner has compulsorily a maximum number of initial intentions because we know that the votes are properly collected during the collection phase. □

2.3 Evaluation of the Cost

Initialization During the initialization phase, every of the $n-1$ normal cardinals has to get initialized and one cardinal might have to go through the initialization procedure twice. The initialization procedure therefore has to be repeated n times. The initialization procedure can be subdivided into three phases. First, a cardinal which has not been initialized yet has to enter the chapel. This will take at most $b-n$ steps for the first cardinal and at most b step for the last cardinal. Secondly the leader cardinal has to enter the chapel. This takes at most b steps. Finally, the same normal cardinal has to enter the chapel again, which also takes at most b steps. The worst case duration of the initialization phase therefore is $(3b - n/2)n$, which is in $O(b \cdot n)$.

Vote Intention Collection For each round all normal cardinal need to send his answer to the leader cardinal. Each communication starts with a cardinal who has not answered during the current round entering the chapel. This will take $b-n$ steps for the first cardinal and b step for the last cardinal to answer. Then the leader cardinal has to acknowledge the information, which happens after at most b steps. The worst case duration of one round therefore is $(2b-n/2)(n-1)$.

The vote intention collection phase uses one round per cardinal. The worst case duration of the second phase therefore is $(2b-n/2)(n-1)n$, which is in $O(b\cdot n^2)$.

Voting In the voting phase every normal cardinal needs to receive the data, which means that $n-1$ messages will be sent. A cardinal which has not received the information yet needs to reserve the chanel. This will take $b-n$ steps for the first cardinal and b step for the last cardinal. The leader cardinal can then start sending the information when he enters the chapel again after at most b steps. Each message has a size of $\lceil log_2(n) \rceil$ bits. Sending a bit first means waiting on a normal cardinal to receive it and then for the leader cardinal to receive the acknowledgement. Both of these step have a cost of b. The worst case duration of the voting phase therefore is $(n-1)(\lceil log_2(n) \rceil 2b + 2b - n/2)$ which is in $O(b \cdot n \cdot log(n))$.

The overall duration for this algorithm is in $O(b\cdot n^2)$. The cost is asymptotically equal to the duration since the size of the shared memory is a constant.

Private Memory Requirement The maximum amount of private memory is needed by the leader cardinal during the vote intention collection phase. He needs to keep track of:

- the index of the current cardinal
- the number of votes obtained by the current cardinal
- the number of cardinals having given their intention for the current cardinal
- the number of intentions obtained by the cardinal with the best score so far
- the number of the best cardinal so far.

The leader cardinal can re-use one of these counters (for example the one used to count the number of cardinals having given their intention for the current cardinal) to count the number of initialized cardinal in the initialization phase and the number of cardinal having voted in the voting phase. In addition, the special cardinal needs to keep track of the state he is currently in, which requires 2 bits. A normal cardinal needs one counter to keep track of the current cardinal and 3 bits to keep track of his state. The total private memory requirement per cardinal is therefore $5\lceil log_2(n) \rceil + 2$ which is in $O(log(n))$.

3 Robustness to Cheating

3.1 Definition of Cheat-resistance

One of the criteria for evaluating a distributed algorithm is its resistance to cheat. We restrict cheating to a subset of all the possible cheats and clearly distinguish cheats from failures. Informally, we are interested in cheats that cannot be detected and do not lead to incoherent outcome: a pope must be unanimously elected in the end. Therefore, cheating consists in electing a different pope than the one with the highest number of intentions. Moreover, we focus on the case in which there is exactly one cheating cardinal.

Some conditions are added to specify what the cheating cardinal can do In particular, we require that the cheater has no advantages compared to the other cardinals: the cheater has the same amount of private memory and he cannot cooperate with the scheduler. The requirement about a non-cooperative scheduler is needed because if the cheater and the scheduler cooperated (and in particular, if the schedule is known by the cheater), the cheat would be undetectable whichever protocol is used[2].

Finally, we consider that the cheating cardinal is opportunistic: he can decide not to cheat in case of an unpropitious situation. Nevertheless, the cheater must not be detectable by any other cardinal under any circumstances.

Invisibly Cheating Protocol To capture the kind of cheat we want to forbid, we first define what an invisibly cheating protocol is. A protocol P' is an *invisibly cheating protocol against a protocol* P if there exists a cardinal c such that if the cardinal c uses the protocol P' whilst the others use the protocol P, the following conditions are true.

1. The leader election is unanimous and terminates (requirements 1 and 3 from the specification of the problem).
2. The elected pope is either the one with the highest number of intentions or cardinal c's favorite.
3. There exists a schedule and set of intentions such that (1) cardinal c's favorite has not the maximal number of intentions and (2) the elected pope is cardinal c's favorite.
4. The protocol P' uses the same information and the same amount of private memory than what is used in the protocol P.

[2] Consider the case in which the cheating cardinal is scheduled every second round by the scheduler and he knows who the next cardinal is going to be. In that case, he can systematically change the state of the switches to some well-chosen state so as not to be discovered and fool all other cardinals.

Cheat-resistant Protocol A protocol P is *cheat-resistant* if no invisibly cheating protocols against P exist.

Note that a cheat-resistant protocol is robust only to certain kind of cheating. In particular, it is important to note that the choice of the cardinal that can be illegitimately elected pope is fixed before the protocol is run. It rules out cheating protocol that decide in favor of whom the cheat will be depending on some information only available whilst the algorithm is running.

3.2 Cheat-resistant Variant of the Algorithm

The algorithm presented in Section 2 is not cheat-resistant with respect to the definition given in the previous section. Due to the presence of a leading cardinal whose decision (i.e., the cardinals he claims to have the highest number of intentions) cannot be verified by the other cardinals, an invisibly cheating protocol exists.

We propose thereafter a variant of the original 2-switch algorithm that is cheat-resistant. To prevent invisible cheating, we need to transmit more information between the leading cardinal and the other ones so as to make possible for a cardinal to check whether his vote has been altered. Therefore, we need to non-anonymously collect vote intentions during the first phase and transmit the whole ordered set of vote intentions during the voting phase. By using this strategy, every cardinal will be able to check that his vote has not be modified by the leading cardinal. Moreover, since during the voting phase the leading cardinal has no way of knowing to whom he is sending the information, he cannot safely cheat.

Consequently, we modify the different phases of the original algorithm in the following way.

Initialization The initialization phase is left unchanged.

Vote Intention Collection We keep the same way of communication as before but we modify the information collected in one round. A round consists in discovering the value of the j-th bit of the number of cardinal i's favorite. Cardinal i uses the state corresponding to the j-th bit of the number; the other normal cardinals always use ⓪. For every normal cardinal (the leader does not have to communicate his intention), we need $\lceil \log_2 n \rceil$ rounds, resulting in a total number of rounds of $(n-1)\lceil \log_2 n \rceil$.

Voting Instead of transmitting the name of the winner to the others, the leading cardinal sends the whole ordered set of intentions including his own to the others.

They can then vote for the cardinal with the highest number of vote and the minimal index (to guarantee unanimity). This phase requires to send to the $n-1$ normal cardinals a message of length $n\lceil \log_2 n \rceil$.

This modified algorithm respects the requirements given in Section 1. The proof rely essentially on the same arguments than the one given for the original algorithm in 2.2. The only difference is in the information transmitted. Since we know that the information is adequately transmitted, the algorithm terminates (since the number of round is finite) and the outcome will be the expected one.

The running cost of this algorithm is higher than the original one but is still polynomial. The intention collection phase has $(n-1)\lceil \log_2 n \rceil$ rounds (for each candidate, the intentions of all cardinals but the leading one must be collected). In the voting phase, the leading cardinal transmits $n\lceil \log_2 n \rceil$ bits to every cardinal. Therefore, the number of steps assuming a b-bounded scheduler is asymptotically in $O(b \cdot n^2 \cdot \log n)$.

We claim that this algorithm is cheat-resistant. To verify that claim, we will consider two different cases based on the assumption that there is exactly one cheater: either the leading cardinal cheats or a normal cardinal cheats.

- If the leading cardinal wanted to cheat, he would have to send to cardinals a different ordered set of intentions. Suppose he needs to modify one of the intention to makes his favorite have a maximal number of vote. In that case, the cardinal whose intention has been modified will notice it and therefore the cheat has been detected. The leading cardinal cannot avoid that detection because he cannot know to which cardinal he is transmitting the information. If he modifies his own intention, the modification cannot be detected by any other cardinal but it does not give more intentions to his favorite and therefore the elected pope will not be his favorite unless his favorite has initially a maximal number of intentions.
- If a normal cardinal wanted to cheat, he would have to systematically modify the information sent by the leader. Since he does not know the schedule, he cannot systematically succeed in changing the communicated information.

Consequently, no cardinals can invisibly cheat.

4 Global Cost Minimization

4.1 Lower Bound on the Running Time

In addition to minimizing the size of the shared memory, we want to minimize the duration of the algorithm in the case where the amount of shared memory is unlimited.

Claim Every election protocol for a bounded scheduler with bound b needs at least $3b - n + 3$ steps.

Proof For all protocols, there exists a scenario in which the intention of every single cardinal is required to determine the winner. By definition of a bounded scheduler with bound b, it is possible that one cardinal C_{last} does not enter the chapel during the first b steps of the algorithm. If every other cardinal enters the chapel exactly once between step $b - n + 2$ and step b, then C_{last} can enter the chapel $b - n + 1$ times in a row (between steps $b + 1$ and $2b - n + 2$). Since C_{last} never entered the chapel before the serie of visits and no other cardinal enters the chapel during the serie, C_{last} cannot exclude that the initial state of the shared memory he sees is is actually the random state at the begining of the algorithm. He cannot conclude anything from the shared memory he sees, and therefore cannot cast his ballot at the end of the serie. He will thus have to enter the chapel again to vote, which in the worst case would happen after b steps, or at step $3b - n + 3$. Given that this cardinal's ballot is essential for succesfully concluding the algorithm, and that no assumptions at all were made on the content and the usage of the shared memory, we conclude that there is no algorithm that has a worst case runing time below $3b - n + 3$.

4.2 Algorithm with a Running Time of $3b + 3$

Herein we present an algorithm whose running time is $3\,b+3$. On the other hand, it uses $n \cdot (\lceil log_2 n \rceil + 2)$ switches: for each cardinal c_i, we need one intention ($\lceil log_2 n \rceil$ bits), plus two bits f_i, v_i to signal that he has given his intention, respectively he has voted.

Algorithm The initialization phase goes as follows: a cardinal entering the chapel for the first time sets all switches to 0. Everytime he enters again, he modifies the switches in order to signify his intention and sets f_i to 1 until each cardinal has entered at least once, that is, $f_j = 1$ for every cardinal c_j. It happens after $2\,b+2$ steps in the worst case and at this moment, the cardinal has enough information to vote. Thus, he sets v_i to 1 and drops its ballot paper in the box. The last cardinal to vote, say c_j, knows that he is the last because all $v_i = 1$ for every other cardinal c_i. Thus he sets T to 1. It happens after $3\,b+3$ in the worst case.

Evaluation Each cardinal needs only one bit of private memory in order to remember that he has already entered the chapel. The asymptotic cost of this algorithm is in $O(b \cdot n \cdot log(n))$.

4.3 Lower Bound on the Shared Memory Space

We informally argue that an algorithm with one switch is not possible if the scheduler is only known to be fair. Since one switch can only represent two states, it is impossible to exchange information; any encoding based on two states cannot work. The problem is that we can only encode the data using an unary code (one of the state must be used as acknowledgment).

Considering that fact, the receiver cannot determine at which moment the transmission is finished. There are no states that we could use for that purpose. The number of bits transmitted as far is of no relevance since the only way in which we could notice that this value is different from the value transmitted as far is to wait for a next bit being sent.

Even though the previous reasoning is completely informal, it gives sufficient evidence that a lower bound for the number of switches is 2 if the only property of the scheduler we assume is fairness.

4.4 Comparison of the Algorithms

Table 1 compares the asymptotic and exact costs of the different algorithms we introduce. In particular, we notice that the straightforward algorithm not only has a smaller asymptotic cost than the 2-switch algorithm, but also a significantly cost lower exact cost for the concrete case with 115 cardinals. This raises the question of the usefulness of an algorithm which reduces the amount of shared memory but increases the overall costs.

Further extra cost arises when using the cheat resistant variant, and it is again important to balance this extra cost with respect to the need of security and robustness. Moreover, a solution that fits well for a given pair of values n and b may not be the best for other values, and it is therefore important to consider the specific requirement of a given setting in order to choose the most appropriate algorithm.

Algorithm	Switches	Steps	Memory	Asymptotic cost	Exact cost
2-sw	2	$O(b\,n^2)$	$O(log(n))$	$O(b\,n^2)$	$4,591,016$
Cr 2-sw	2	$O(b\,n^2 log(n))$	$O(log(n))$	$O(b\,n^2 log(n))$	$31,766,788$
Fast	$n(log(n)+2)$	$3b$	1	$O(b\,n\,log(n))$	$357,075$

Table 1. Summary of the performance of the algorithms in term of number of switches, duration of the algorithm, amount of private memory per cardinal and the asymptotic cost. The last column is the exact cost for $b = 115$ and $n = 115$. The abbreviations *2-sw*, *Cr 2-sw* and *Fast* respectively stand for the basic 2-switch algorithm, the cheat-resistant 2-switch algorithm and the fast algorithm running in $3b$ steps.

5 Conclusion

In this paper we propose a protocol using the minimal number of 2 switches. We also provide a formal definition of cheating and a variant of our 2-switches algorithm that is cheat-resistant under this definition. Moreover, we discuss the problem of minimizing the duration of the algorithm with an unlimited amount of shared memory and we prove that a lower bound for the cardinal voting problem is $3b - n + 3$ steps. We propose a simple algorithm whose duration is close to this lower bound.

Originally, this project aims to build a protocol solving the election problem within an asynchronous distributed system by using a minimal amount of shared memory. Our solutions show that by doing so, we are actually increasing both the running time and the overall cost of the protocol.

Since one can imagine situations in which the amount of shared memory needs to be extremely limited, the answer to this question, finding a 2-switch algorithm is not only an intellectual challenge but might actually be useful in practice. It is however important to balance the advantages of reducing the number of shared memory with respect to the running time requirements; in most situations, an algorithm which lowers both the running time and the overall cost might be preferable.

Furthermore, we considered the serious issues arising from the possibility for individual participants to cheat. While we introduced a well-defined notion of what a cheat is and where succesfull in building an algorithm which is resistant to this kind of cheats, one could imagine further and more elaborate attacks, as well as system failures, which we do not cover at all. Cheat resistance has a price and it is important to compare the extra overhead to the risk of a particular setting when choosing if – and how – to integrate cheat resistance in such a protocol.

Our work proposed several solutions to a simple problem. None of them can be considered as optimal in every situation, but we provide an range of possibilites for tailoring a protocol that fits particular constraints on shared memory, private memory and runing time.

A Detailed 2-switch Protocol

Below is a listing of the complete protocol for the 2-switch algorithm.

- Private variables
  - `N`: number of cardinals
  - `COUNTERS_SIZE`: Size of the counters in bit, equal to $\lceil log(n) \rceil$
- Shared variables
  - `state`: 2-bits memory, takes values A,B,0,1. Not initalized.
  - `smoke`: set to true to launch the white smoke (write-only)

– Private variables
  - `my_intention`: vote intention of the cardinal (read-only)
  - `my_vote`: vote of the cardinal (write-only)
  - `leader_cardinal`: true if this is the leader cardinal (read-only)
  - `phase`: internal state of the cardinal. Initialized to `FIRST_VISIT`.
  - `counter`: multi-use counter, initialized to 0 (size: `COUNTERS_SIZE` bits)
  - `current_vote_counter`: number of votes of the current cardinal. Used by the leader cardinal, intialized to 0 (size: `COUNTERS_SIZE` bits)
  - `candidate_index`: index of the cardinal currently receiving votes. Used by both the leader and the normal cardinals. Initialized to 0. (size: `COUNTERS_SIZE` bits)
  - `best_cardinal`: number of the current best cardinal. Passed from the leader to the normal cardinals in the voting phaseInitialized to 0.
  - `max_votes`: number of vote intentions of the current best. Initialized to 0. (size: `COUNTERS_SIZE` bits)

The notation `variable[index]` denotes bit `index` of `variable`.

```
1  if leader_cardinal
2    if phase = FIRST_VISIT
3      state := 0
4      phase := INIT
5    elsif phase = INIT
6      if counter = N - 1
7        if state = 0
8          phase := VOTE_COLLECTION
9          state := A
10         counter := 0
11         if candidate_index = my_favorite
12                 current_vote_counter := 1
13         else
14                 current_vote_counter := 0
15         fi
16       fi
17     elsif
18       if state := 1
19         state := B
20         counter := counter + 1
21       fi
22     fi
23   elsif phase = VOTE_COLLECTION
24     if state = 1
25       current_vote_counter := current_vote_counter + 1
```

```
26      fi
27      if state = 0 or state = 1
28        counter := counter + 1
29        if counter := N - 1
30          counter := 0
31          if current_vote_counter > max_votes
32            max_votes := current_vote_counter
33            best_cardinal := candidate_index
34          fi
35          candidate_index := candidate_index + 1
36          if candidate_index = N
37            phase := VOTE
38          fi
39          if candidate_index = my_favorite
40                  current_vote_counter := 1
41          else
42                  current_vote_counter := 0
43          fi
44        fi
45        if (candidate_index mod 2) = 0
46          state := A
47        else
48          state := B
49        fi
50      fi
51    elsif phase = VOTE and candidate_index > 0
52      if ((N mod 2) = 0 and state = B) _
53         or ((N mod 2) = 1 and state = A)
54        if counter < COUNTERS_SIZE
55          state := best_cardinal[counter]
56          counter := counter + 1
57        else
58          candidate_index := candidate_index - 1
59          counter := 0
60          if candidate_index = 1
61            phase := DONE
62            my_vote := best_cardinal
63            smoke := 1
64            candidate_index := 0
65          else
66            if (N mod 2) == 0
67              state := A
68            else
69              state := B
70            fi
```

```
71          fi      fi
72      fi
73    fi
74  else
75    if phase = FIRST_VISIT
76      if state = 0 or state = A
77        state := 1
78        phase := INIT
79      fi
80    elsif phase = INIT
81      if state = B
82        state := 0
83        phase := VOTE_COLLECTION
84      elsif state = 0
85        state := 1
86      fi
87    elsif phase = VOTE_COLLECTION
88      if ((counter mod 2 = 0) and state = A) _
89         or ((counter mod 2 = 1) and state = B)
90        if my_intention = counter
91          state := 1
92        else
93          state := 0
94        fi
95        counter := counter + 1
96        if counter = N
97          phase := WAIT_FOR_CHANNEL
98          counter := 0
99        fi
100     fi
101   elsif phase = WAIT_FOR_CHANNEL
102     if ((N mod 2) = 0) and state = A
103       phase := GET_WINNER
104       state := B
105     elsif ((N mod 2) = 1) and state = B
106       phase := GET_WINNER
107       state := A
108     fi
109   elsif phase = GET_WINNER
110     if state = 0 or state = 1
111       best_cardinal[counter] := state
112       counter := counter + 1
113       if counter = COUNTERS_SIZE
114         my_vote := best_cardinal
115         phase := DONE
```

```
116         fi
117         if ((N mod 2) = 0)
118           state := B
119         else
120           state := A
121         fi
122       fi
123     fi
124   fi
```