Distributed Protocols for Leader Election: a Game-Theoretic Perspective*

Ittai Abraham[1], Danny Dolev[2]**, and Joseph Y. Halpern[3]***

[1] Microsoft Research, Silicon Valley
ittaia@microsoft.com
[2] The Hebrew University of Jerusalem, Jerusalem, Israel
dolev@cs.huji.ac.il
[3] Computer Science Department, Cornell University, Ithaca, NY 14853, USA
halpern@cs.cornell.edu

Abstract. We do a game-theoretic analysis of leader election, under the assumption that each agent prefers to have some leader than to have no leader at all. We show that it is possible to obtain a *fair* Nash equilibrium, where each agent has an equal probability of being elected leader, in a completely connected network, in a bidirectional ring, and a unidirectional ring, in the synchronous setting. In the asynchronous setting, Nash equilibrium is not quite the right solution concept. Rather, we must consider *ex post* Nash equilibrium; this means that we have a Nash equilibrium no matter what a scheduling adversary does. We show that ex post Nash equilibrium is attainable in the asynchronous setting in all the networks we consider, using a protocol with bounded running time. However, in the asynchronous setting, we require that $n > 2$. We can get a fair *ϵ-Nash* equilibrium if $n = 2$ in the asynchronous setting, under some cryptographic assumptions (specifically, the existence of a pseudo-random number generator and polynomially-bounded agents), using ideas from bit-commitment protocols. We then generalize these results to a setting where we can have deviations by a coalition of size k. In this case, we can get what we call a fair k-resilient equilibrium if $n > 2k$; under the same cryptographic assumptions, we can a get a k-resilient equilibrium if $n = 2k$. Finally, we show that, under minimal assumptions, not only do our protocols give a Nash equilibrium, they also give a *sequential* equilibrium [**?**], so players even play optimally off the equilibrium path.

1 Introduction

As has been often observed, although distributed computing and game theory are interested in much the same problems—dealing with systems where there are many agents,

* This is the authors copy of the paper that will appear in DISC 2013.
** Incumbent of the Berthold Badler Chair in Computer Science. Part of the work was done Supported in part by The Israeli Centers of Research Excellence (I-CORE) program, (Center No. 4/11), by NSF, AFOSR grant FA9550-09-1-0266, and by the Google Inter-University Center for Electronic Markets and Auctions.
*** Supported in part by NSF grants IIS-0534064, IIS-0812045, and IIS-0911036, IIS-0911036, and CCF-1214844, AFOSR grants FA9550-08-1-0438, FA9550-09-1-0266, and FA9550-12-1-0040, and ARO grant W911NF-09-1-0281.

facing uncertainty, and having possibly different goals—in practice, there has been a significant difference in the models used in the two areas. In game theory, the focus has been on rational agents: each agent is assumed to have a utility on outcomes, and be acting so as to maximize expected utility. In distributed computing, the focus has been on the "good guys/bad guys" model. The implicit assumption here is that there is a system designer who writes code for all the processes in the system, but some of the processes may get taken over by an adversary, or some computers may fail. The processes that have not been corrupted (either by the adversary or because of a faulty computer) follow the designer's protocol. The goal has typically been to prove that the system designer's goals are achieved, no matter what the corrupted processes do.

More recently, there has been an interest in examining standard distributed computed problems under the assumption that the agents are *rational*, and will deviate from the designer's protocol if it is in their best interest to do so. Halpern and Teague [**?**] were perhaps the first to do this; they showed (among other things) that secret sharing and multiparty communication could not be accomplished by protocols with bounded running time, if agents were using the solution concept of iterated admissibility (i.e., iterated deletion of weakly dominated strategies). Since then, there has been a wide variety of work done at the border of distributed computing and game theory. For one thing, work has continued on secret sharing and multiparty computation, taking faulty and rational behavior into account (e.g., [**?**,**?**,**?**,**?**,**?**]). There has also been work on when and whether a problem that can be solved with a trusted third party can be converted to one that can be solved using *cheap talk*, without a third party, a problem that has also attracted the attention of game theorists (e.g., [**?**,**?**,**?**,**?**,**?**,**?**,**?**,**?**,**?**,**?**,**?**,**?**,**?**,**?**]). This is relevant because there are a number of well-known distributed computing problems that can be solved easily by means of a "trusted" mediator. For example, if fewer than half the agents are corrupted, then we can easily do Byzantine agreement with a mediator: all the agents simply tell the mediator their preference, and the mediator chooses the majority. Another line of research was initiated by work on the *BAR* model [**?**]; see, for example, [**?**,**?**]. Like the work in [**?**,**?**], the BAR model allows Byzantine (or faulty) players and rational players; in addition, it allows for *acquiescent* players, who follow the recommended protocols.[4] Traditional game theory can be viewed as allowing only rational players, while traditional distribution computing considers only acquiescent and Byzantine players.

In this paper, we try to further understand the impact of game-theoretic thinking on standard problems in distributed computing. We consider the classic distributed computing problem of electing a leader in an anonymous network (a network where, initially, each process knows its own name, but does not know the name of any other process). Leader election is a fundamental problem in distributed computing. Not surprisingly, there are numerous protocols for this problem (see, e.g., [**?**,**?**,**?**,**?**,**?**]) if we assume that no agents have been corrupted; there have also been extensions that deal with corrupted agents [**?**,**?**]. Much of this work focuses on leader election in a ring (e.g., [**?**,**?**,**?**,**?**,**?**]).

[4] Originally, the "A" in "BAR" stood for *altruistic*, but it was changed to stand for "acquiescent" [**?**].

In this paper we study what happens if we assume that agents are rational. It is easy to show that if all agents (a) prefer to have a leader to not having a leader and (b) are indifferent as to who is the leader, then all the standard distributed computing protocols work without change. This can be viewed as formalizing the intuition that in the standard setting in distributed computing, we are implicitly assuming that all the agents share the system designer's preferences. But what happens if the agents have different preferences regarding who becomes the leader? For example, an agent may prefer that he himself becomes the leader, since this may make the cost of routing to other agents smaller. In this case, the standard protocols (which typically assume that each agent has a distinct id, and end up electing the agent with the lowest id, or the agent with the highest id, as the leader) do not work; agents have an incentive to lie about their id. Nevertheless, there is always a trivial Nash equilibrium for leader election: no one does anything. Clearly no agent has any incentive to do anything if no one else does. We are thus interested in obtaining a *fair* Nash equilibrium, one in which each agent has an equal probability of being elected leader. Moreover, we want the probability that someone will be elected to be 1.[5] In the language of the BAR model, we allow acquiescent and rational players, but not Byzantine players.

It is easy to solve leader election with a mediator: the agents simply send the mediator their ids, and the mediator picks an id at random as the leader and announces it to the group. We cannot immediately apply the ideas in the work on solving the problem with a mediator and then replacing the mediator with cheap talk to this problem because all these results assume (a) that agents have commonly-known names, (b) that the network is completely connected, and (c) the network is synchronous. Nevertheless, we show that thinking in terms of mediators can be helpful in deriving a simple protocol in the case of a completely connected network that is a fair Nash equilibrium in which a leader is elected with probability 1. We can then modify the protocol so that it works when the network is a ring. We also show that our protocol is actually k-*resilient* [**?**,**?**]: it tolerates coalitions of size k, as long as $n > k$. This forms an interesting contrast to work on Byzantine agreement, where it is known that the network must be $2k + 1$ connected to tolerate k Byzantine failures [**?**]. But we can tolerate coalitions of k rational players even in a unidirectional ring.

These protocols work if the network is synchronous. What happens in an asynchronous setting? Before answering this question, we need to deal with a subtlety: what exactly a Nash equilibrium is in an asynchronous setting? To make sense of Nash equilibrium, we have to talk about an agent's best response. An action for an agent i is a best response if it maximizes i's expected utility, given the other agents' strategies. But to compute expected utility, we need a probability on outcomes. In general, in an asyn-

[5] Without the last requirement, the existence of a fair Nash equilibrium follows from well-known results, at least in the case of a completely connected network. We can model our story as a symmetric game, one where all agents have the same choice of actions, and an agent's payoff depends only on what actions are performed by others, not who performs them. In addition to showing that every game has a Nash equilibrium, Nash also showed that a symmetric game has a symmetric Nash equilibrium, and a symmetric equilibrium is clearly fair. However, in a symmetric equilibrium, it may well be the case that there is no leader chosen. For example, a trivial symmetric equilibrium for our game is one where everyone chooses a candidate leader at random. However, in most cases, agents choose different candidates, so there is no leader.

chronous setting, the outcome may depend on the order that agents are scheduled and on message-delivery times. But we do not have a probability on these. We deal with these problems in this setting by using the standard approach in distributed computing. We assume that an adversary chooses the scheduling and chooses message-delivery times, and try to obtain a strategy that is a Nash equilibrium no matter what the adversary does. This intuition gives rise to what has been called in the literature an *ex post Nash equilibrium*. We provide a simple protocol that gives a fair ex post Nash equilibrium provided that $n > 2$. More generally, we provide a fair ex post k-resilient equilibrium as long as $n > 2k$. We then show that these results are optimal: there is no fair k-resilient ex post Nash equilibrium if $n \leq 2k$.

The lower bounds assume that agents are not computationally bounded. If we assume that agents are polynomially-bounded (and make a standard assumption from the cryptography literature, namely, that a pseudorandom number generator exists), then we can show, using ideas of Naor [**?**], that there is a fair ex post ϵ-*Nash* equilibrium in this case (one where agents can gain at most ϵ by deviating) for an arbitrarily small ϵ; indeed, we can show that there is a fair ex post ϵ–k-resilient equilibrium as long as $n > k$.

Finally, we show that, under minimal assumptions, not only do our protocols give a Nash equilibrium, they also give a *sequential* equilibrium [**?**], so players even play optimally off the equilibrium path.

2 The Model

We model a network as a directed, simple (so that there is at most one edge between each pair of nodes), strongly connected, and finite graph. The nodes represent agents, and the edges represent communication links. We assume that the topology of the network is common knowledge, so that if we consider a completely connected network, all agents know that the network is completely connected, and know that they know, and so on; this is similarly the case when we consider unidirectional or bidirectional rings. Deviating agents can communicate only using the network topology; there is no "out of band" communication. We assume that, with each agent, there is associated a unique id, taken from some commonly-known name space, which we can take to be a set of natural numbers. Initially agents know their ids, but may not know the id of any other agent. For convenience, if there are n agents, we name them $1, \ldots, n$. These are names used for our convenience when discussing protocols (so that we can talk about agent i); these names are not known by the agents. Message delivery is handled by the channel (and is not under the control of the agents). Agents can identify on which of their incoming links a message comes in, and can distinguish outgoing links.

When we consider synchronous systems, we assume that agents proceed in lockstep. In round m, (if $m > 0$) after all messages sent in round $m - 1$ are received by all agents, agents do whatever internal computation they need to do (including setting the values of variables); then messages are sent (which will be received at the beginning of round $m + 1$). [6] In the asynchronous setting, agents are scheduled to move at arbitrary times by a (possibly adversarial) scheduler. When they are scheduled, they perform the same kinds of actions as in the synchronous case: receive some messages that were sent

[6] Thus, the synchronous model assumes no "rushing".

to them earlier and not yet received, do some computation, and send some messages. For ease of exposition, we assume that the message space is finite. While we assume that all messages sent are eventually received (uncorrupted), there is no bound on message delivery time. Nor do we make any assumption on the number of times one agent can be scheduled relative to another, although we do assume that agents are scheduled infinitely often (so that, for all agents i and times t, there will be a time after t when i is scheduled).

For leader election, we assume that each agent i has a variable $leader_i$ which can be set to some agent's id. If, at the end of the protocol, there is an id v such that $leader_i = v$ for all agents i, then we say that the agent with id v has been elected leader. Otherwise, we say that there is no leader. (Note that we are implicitly requiring that, when there is a leader, all the players know who that leader is.) We assume that each agent i has a utility on outcomes of protocols. For the purposes of this paper, we assume that agents prefer having a leader to not having one, in the weak sense that each agent i never assigns a higher utility to an outcome where there is no leader than to one in which there is a leader (although we allow the agent to be indifferent between an outcome where there is no leader and an outcome where there is a leader). We make no further assumptions on the utility function. It could well be that i prefers that he himself is the leader rather than anyone else; i could in addition prefer a protocol where he sends fewer messages, or does less computation, to one where he sends more messages or does more computation. Nevertheless, our eassumptions require that player i never prefers an outcome where there is no leader to one where there is, even if the latter outcome involves sending many messages and a great deal of computation (although in fact our protocols are quite message-efficient and do not require much computation). Note that our assumptions imply that agent i can "punish" other agents by simply setting $leader_i$ to $\perp$; this ensures that there will be no leader. In the language of [**?**], this means that each agent has a *punishment strategy*.

A strategy profile (i.e., a strategy or protocol for each agent) is a *Nash equilibrium* if no agent can unilaterally increase his expected utility by switching to a different protocol (assuming that all the other agents continue to use their protocols). It is easy to see that if all the agents are indifferent regarding who is the leader (i.e., if, for each agent i, i's utility of the outcome where j is the leader is the same for all j, including $j = i$), then any protocol that solves leader election is a Nash equilibrium. Note that it is possible that one Nash equilibrium *Pareto dominates* another: all agents are better off in the first equilibrium. For example, if agents are indifferent about who the leader is, so that any protocol that solves leader election is a Nash equilibrium, all agents might prefer an equilibrium where fewer messages are sent; nevertheless, a protocol for leader election where all agents send many messages could still be a Nash equilibrium.

For the remainder of this paper, we assume that each agent has a *preference for leadership*: agent i's utility function is such that i does not give higher utility to an outcome where there is no leader than to one where there is a leader. (Agent i may also prefer to be the leader himself, or have preferences about which agent j is the leader if he is not the leader; these preferences do not play a role in this paper.)

3 The protocols

We consider protocols in three settings: a completely connected network, a unidirectional ring, and a bidirectional ring. We also consider both the synchronous case and the asynchronous case.

3.1 Completely connected network, synchronous case

Consider leader election in a completely connected network. First suppose that we have a mediator, that is, a trusted third party. Then there seems to be a naive protocol that can be used: each agent tells the mediator his id, then the mediator picks the highest id, and announces it to all the agents. The agent with this id is the leader. This naive protocol has two obvious problems. First, since we assume that the name space is commonly known, and all agents prefer to be the leader, agents will be tempted to lie about their ids, and to claim that the highest id is their id. Second, even if all agents agree that an agent with a particular id v is the leader, they don't know which agent has that id.

We solve the first problem by having the mediator choose an id at random; we solve the second problem by having agents share their ids. In more detail, we assume that in round 1, agents tell each other their ids. In round 2, each agent tells the mediator all the set of ids he has heard about (including his own). In round 3, the mediator compares all the sets of ids. If they are all the same, the mediator chooses an id v at random from the set; otherwise, the mediator announces "no leader". If the mediator announces that v is the leader, each agent i sets $leader_i = v$ (and marks the incoming link on which the id v was originally received); otherwise, $leader_i$ is undefined (and there is no leader).

It is easy to see that everyone using this protocol gives a Nash equilibrium. If some agent does not send everyone the same id, then the mediator will get different lists from different agents, and there will be no leader. And since a leader is chosen at random, no one has any incentive not to give his actual id. Note that this protocol is, in the language of [**?**,**?**], *k-resilient* for all $k < n$, where n is the number of agents. That is, not only is it the case that no single agent has any incentive to deviate, neither does any coalition of size k. Moreover, the resulting Nash equilibrium is *fair*: each agent is equally likely to be the chosen leader.

Now we want to implement this protocol using cheap talk. Again, this is straightforward. At round 1, each agent i sends everyone his id; at round 2, i sends each other agent j the set of ids that he (i) has received (including his own). If the sets received by agent i are not all identical or if i does not receive an id from some agent, then i sets $leader_i$ to $\perp$, and leader election fails. Otherwise, let n be the cardinality of the set of ids. Agent i chooses a random number N_i in $\{0, \ldots, n-1\}$ and sends it to all the other agents. Each agent i then computes $N = \sum_{i=1}^{n} N_i \pmod n$, and then takes the agent with the Nth highest id in the set to be the leader. (If some agent j does not send i a random number, then i sets $leader_i = \perp$.) Call this protocol for agent i $LEAD_i^{cc}$. The formal pseudocode of the protocol appears in the full paper, available at http://www.cs.cornell.edu/home/halpern/papers/leader.pdf. Let $\mathbf{LEAD}^{cc}$ denote the profile $(LEAD_1^{cc}, \ldots, LEAD_1^{cc})$ (we use boldface for profiles throughout the paper). Clearly, with the profile $\mathbf{LEAD}^{cc}$, all the agents will choose the same leader. It is also easy to see that no agent (and, indeed, no group of size $k < n$) has any incentive to deviate from this strategy profile.

Theorem 1. $\mathbf{LEAD}^{cc}$ *is a fair,* k*-resilient equilibrium in a completely connected network of* n *agents, for all* $k < n$*.* [7]

Up to now we have implicitly assumed that each agent somehow gets a signal regarding when to start the protocol. This assumption is unnecessary. Even if only some agents want to start the protocol, they send a special round 0 message to everyone asking them to start a leader election protocol. The protocol then proceeds as above.

3.2 Unidirectional ring, synchronous case

We give a Nash equilibrium for leader election in a unidirectional ring, under the assumption that the ring size n is common knowledge. This assumption is necessary, for otherwise an agent can create k sybils, for an arbitrary k, and pretend that the sybils are his neighbors. That is, i can run the protocol as if the ring size is $n + k$ rather than n, simulating what each of his sybils would do. No other agent can distinguish the situation where there are n agents and one agent has created k sybils from a situation where there are actually $n+k$ agents. Of course, if any of i's sybils are elected, then it is as if i is elected. Thus, creating sybils can greatly increase i's chances of being elected leader, giving i an incentive to deviate. (However, the overhead of doing may be sufficient to deter an agent from doing so. See the discussion in Section 4.) Note that in the case of a completely connected network, given that the topology is common knowledge, the number of agents is automatically common knowledge (since each agent can tell how many agents he is connected to).

The protocol is based on the same ideas as in the completely connected case. It is easy to ensure that there is agreement among the agents on what the set of agents is; implementing a random selection is a little harder. We assume that the signal to start leader election may come to one or more agents. Each of these agents then sends a "signed" message (i.e., a message with his id) to his neighbor. Messages are then passed around the ring, with each agent, appending his id before passing it on. If an agent receives a second message that originated with a different agent, the message is ignored if the originating agent has a lower id; otherwise it is passed on. Eventually the originator of the message with the highest id gets back the message. At this point, he knows the ids of all the agents. The message is then sent around the ring a second time. Note that when an agent gets a message for the second time, he will know when the message should make it back to the originator (since the system is synchronous and he knows the size of the ring).

At the round when the originator gets back the message for the second time, each agent i chooses a random number $N_i < n$ and sends it around the ring. After n rounds, all agents will know all the numbers $N_1, \ldots, N_n$, if each agent indeed sent a message. They can then compute $N = \sum_{i=1}^{n} N_i \pmod n$, and take the agent with the Nth highest id in the set to be the leader. If agent i does not receive a message when he expects to, then he aborts, and no leader is elected. For example, if an agent who originated a message does not get his message back n rounds and $2n$ rounds after he sent it, or gets a message from an originator with a lower id, then he aborts. Similarly, if an agent who forwarded an originator's message does not get another message from that

[7] All proofs can be found in the full paper.

originator n rounds later or get a message from another originator with a lower id, then he aborts. Finally, for each of the n rounds after the originator with the highest id gets back his message for the second time, each agent i should get a random number from the appropriate agent (i.e., k rounds after the originator with the highest id gets back his message for the second time, agent i should get agent j's random number, j is k steps before i on the ring). If any of these checks is not passed, then i aborts, and no leader is chosen. Call this protocol for agent i $LEAD_i^{uni}$. The formal pseudocode of this and all other protocols mentioned in this paper appear int the full paper.

We would now like to show that $\mathbf{LEAD}^{uni}$ gives a k-resilient fair Nash equilibrium. But there is a subtlety, which we already hinted at in the introduction. In a Nash equilibrium, we want to claim that what an agent does is a best response to what the other agents are doing. But this implicitly assumes that the outcome depends only on the strategies chosen by the agents. But in this case, the outcome may in principle also depend on the (nondeterministic) choices made by nature regarding which agents get an initial signal. Thus, we are interested in what has been called an *ex post* Nash equilibrium. We must show that, no matter which agents get an initial signal, no agent has any incentive to deviate (even if the deviating agent knows which agents get the initial signal, and knows the remaining agents are playing their part of the Nash equilibrium). In fact, we show that no coalition of $k < n$ agents has any incentive to deviate, independent of nature's choices.

Theorem 2. $\mathbf{LEAD}^{uni}$ *is a fair,* k*-resilient (ex post) equilibrium in a unidirectional ring with* n *agents, for all* $k < n$*.*

3.3 Bidirectional ring, synchronous case

It is easy to see that the same protocol will work for the case of the bidirectional ring. More precisely, if there is agreement on the ring orientation, each agent implements the protocol above by just sending left, ignoring the fact that he can send right. If there is no agreement on orientation, then each originating agent can just arbitrarily choose a direction to send; each agent will then continue forwarding in the same direction (by forwarding the message with his id appended to the neighbor from which he did not receive the message). The originator with the highest id will still be the only one to receive his original message back. At that point the protocol continues with round 2 of the protocol for the unidirectional case, and all further messages will be sent in the direction of the original message of the originator with the highest id. Since it is only in the second round that agents append their random numbers to messages, what happened in the first round has no effect on the correctness of the algorithm; we still get a Nash equilibrium as before.

3.4 Asynchronous Ring

We now consider an asynchronous setting. It turns out to be convenient to start with a unidirectional ring, then apply the ideas to a bidirectional ring. For the unidirectional ring, we can find a protocol that gives an ex post Nash equilibrium provided that there are at least 3 agents in the ring.

Consider the following protocol. It starts just as the protocol for the unidirectional case in the synchronous setting. Again, we assume that the signal to start a leader election may come to one or more agents. Each of these agents then sends a message with his id to his neighbor. Messages are then passed around the ring, with each agent appending his id before passing it on. If an agent receives a second message that originated with a different agent, the message is ignored if the originating agent has a lower id; otherwise it is passed on. Eventually the originator of the message with the highest id gets back the message. The originator checks to make sure that the message has n (different) ids, to ensure that no "bogus" ids were added. The message is then sent around the ring a second time. When an agent i gets the message the second time, he chooses a random number N_i mod n and sends it to his neighbor (as well as passing on the list of names). Agent i's neighbor does not pass on N_i; he just keeps it. Roughly speaking, by sending N_i to his neighbor, agent i is committing to the choice. Crucially, this commitment must be made before i knows any of the random choices other than that of the agent j of whom i is the neighbor (if i is not the originator). When the originator gets the message list for the second time (which means that it has gone around the ring twice), he sends it around the ring the third time. This time each agent i adds his random choice N_i to the list; agent i's neighbor j checks that the random number that i adds to the list is the same as the number that i sent j the previous time. When the originator gets back the list for the third time, it now includes each agent i's random number. The originator then sends the list around the ring for a fourth time. After the fourth time around the ring, all agents know all the random choices. Each agent then computes $N = \sum_{i=1}^{n} N_i \pmod{n}$, and then takes the agent with the Nth highest id in the set to be the leader. Each time an agent i gets a message, he checks that it is compatible with earlier messages that he has seen; that is, the second time he gets the message, all the ids between the originator and his id must be the same; the third time he gets the message, all the ids on the list must be the same as they were the second time he saw the message; and the fourth time he gets the message, not only must the list of ids be the same, but all the random choices that he has seen before can not have been changed. If the message does not pass all the checks, then agent i sets $leader_i$ to $\perp$.

Clearly this approach will not work with two agents: The originator's neighbor will get the originator's random choice before sending his own, and can then choose his number so as to ensure that he becomes leader. (We discuss how this problem can be dealt with in Section 3.7.) As we now show, this approach gives a fair ex post Nash equilibrium provided that there are at least three agents. In an asynchronous setting, nature has much more freedom than in the synchronous setting. Now the outcome may depend not only on which agents get an initial signal, but also on the order in which agents are scheduled and on message delivery times. Ex post equilibrium implicitly views all these choices as being under the control of the adversary; our protocol has the property that, if all agents follow it, the distribution of outcomes is independent of the adversary's choices. However, for the particular protocol we have given, it is easy to see that, no matter what choices are made by the adversary, we have a Nash equilibrium. While considering ex post Nash equilibrium seems like a reasonable thing to do in asynchronous systems (or, more generally, in settings where we can view an adversary

as making choices, in addition to the agents making choices), it is certainly not the only solution concept that can be considered. (See Section 4.)

What about coalitions? Observe that, for the protocol we have given, a coalition of size two does have an incentive to deviate. Suppose that i_1 is the originator of the message, and i_1 is i_2's neighbor (so that i_2 will be the last agent on the list originated by i_1). If i_1 and i_2 form a coalition, then i_2 does not have to bother sending i_1 a random choice on the second time around the ring. After receiving everyone's random choices, i_2 can choose N_{i_2} so that he (or i_1) becomes the leader. This may be better for both i_1 and i_2 than having a random choice of leader.

We can get a protocol that gives a k-resilient (ex post) Nash equilibrium if $n > 2k$. We modify the protocol above by having each agent i send his random choice k steps around the ring, rather than just one step (i.e., to his neighbor). This means that i is committing N_i to k other agents. In more detail, we start just as with the protocol presented earlier. Each agent who gets a signal to start the protocol sends a message with his id to his neighbor. The messages are then passed around the ring, with each agent appending his id. If an agent receives a second message that originated with a different agent, the message is ignored if the originating agent has a lower id; otherwise it is passed on. Eventually the originator of the message with the highest id gets back the message. The originator checks to make sure that the message has n ids, to ensure that no "bogus" ids were added. The message is then sent around the ring a second time; along with the message, each agent i (including the sender) sends a random number N_i. Agent i's neighbor does not pass on N_i; he just keeps it, while forwarding the list of ids. When the originator gets the message the third time, he forwards to his neighbor the random number he received in the previous round (which is the random number generated by his predecessor on the ring). Again, his neighbor does not forward the message; instead he sends to his successor the random number he received (from the originator) on the previous round. At the end of this phase, each agent knows his random id and that of his predecessor. We continue this process for k phases altogether. That is, to when the originator gets a message for the third time, he sends this message (which is the random number chosen by his predecessor's predecessor) to his successor. Whenever an agent gets a message, he forwards the message he received in the previous phases. At the end of the jth phase for $j \leq k$, each agent knows the random numbers of his j closest predecessors. After these k phases complete, the sender sends his random number to his neighbor; each agent then appends his id to the list, and it goes around the ring twice. Each agent checks that the random numbers of his k predecessors agree with what they earlier told him. At the end of this process, each agent knows all the random numbers. As usual, each agent then computes $N = \sum_{i=1}^{n} N_i \pmod n$ and chooses as leader the agent with the Nth highest id.

Each time an agent i gets a message, he checks that that it is compatible with earlier messages that he has seen; that is, the second time he gets the message, all the ids between the originator and his id must be the same; the third time he gets the message, all the ids on the list must be the same as they were the second time he saw the message; and the fourth time he gets the message, not only must the list of ids be the same, but all the random choices that he has seen before can not have been changed. He also checks that he has gotten the random choices of his k predecessors on the ring. If the message

does not pass all the checks, then agent i sets $leader_i$ to $\perp$. Call this protocol for agent i $A\text{-}LEAD_i^{uni}$.

Theorem 3. *If $n > 2k$, then $\mathbf{A\text{-}LEAD}^{uni}$ is a fair, k-resilient ex post equilibrium in an asynchronous unidirectional ring.*

We can also use this approach to get a fair Nash equilibrium in a bidirectional network. If agents know the network orientation, they send all their messages in only one direction, implementing the protocol in the unidirectional case. If they do not know the orientation, they first proceed as in the synchronous, exchanging ids to determine who has the highest id. That agent then chooses a direction for further messages, and again they can proceed as in the unidirectional case.

3.5 Asynchronous completely connected network

We can use the ideas above to get a protocol for a completely connected network, embedding a unidirectional ring into the network, but now the added connectivity hurts us, rather than helping. When we embed a ring into the network, each coalition member may be able to find out about up to k other random choices. Since now coalition members can talk to each other no matter where they are on the ring, we must have $n > k(k+1)$ to ensure that a coalition does not learn all the random choices before the last member announces his random choice. We can do better by using ideas from secure multi-party computation and secret sharing [**?**].

To do secret sharing, we must work in a finite field; so, for ease of exposition, assume that n is a power of a prime. As in the synchronous case, agents start by sending their ids to all other agents, and then exchanging the set of ids received, so that they all agree on the set of ids in the system. (Of course, if an agent i does not get the same set of ids from all agents, then i sets $leader_i = \perp$.) We denote by agent i the agent with the ith largest id. Each agent i chooses a random value $N_i \in \{0, \ldots, n-1\}$ and a random degree-$(k+1)$ polynomial f_i over the field $F_n = \{0, \ldots, n-1\}$ such that $f_i(0) = N_i$. Then i sends each agent j the message $f_i(j)$. Once i receives $f_j(i)$ from all agents j, then i sends *DONE* to all agents. Once i receives *DONE* messages from all agents, i sends $s_i = \sum_{j=1}^{n} f_j(i)$ to all agents. After receiving these messages, i will have n points on the degree-$(k+1)$ polynomial $\sum_{j=1}^{n} f_j$ (if no agents have lied about their values). After i has received the messages s_j for all agents j, i checks if there is a unique polynomial f of degree $k+1$ such that $f(j) = s_j$ for $j = 1, \ldots, n$. If such a polynomial f exists, and $f(0) = N$, then i takes the agent with the Nth highest id as leader; otherwise, i sets $leader_i$ to $\perp$. Call this protocol $A\text{-}LEAD_i^{cc}$.

Theorem 4. *If $n > 2k$ then $\mathbf{A\text{-}LEAD}^{cc}$ is a fair, ex post k-resilient equilibrium in an asynchronous completely connected network.*

3.6 A matching lower bound

We now show that Theorems 3 and 4 are the best we can hope for; we cannot find a fair ex post k-resilient strategy if $n \leq 2k$.

Theorem 5. *If $n \leq 2k$, then there is no fair, ex post k-resilient equilibrium for an asynchronous unidirectional ring (resp., bidirectional ring, completely connected network).*

Observe that all the protocols above are *bounded*; although they involve randomization, there are only boundedly many rounds of communication. This is also the case for the protocol presented in the next section. If we restrict to bounded protocols, using ideas of [**?**,**?**], we can get a stronger result: we cannot even achieve an ϵ–k-resilient equilibrium (where agents do not deviate if they can get within ϵ of the utility they can get by deviating) for sufficiently small ϵ.

Theorem 6. *If $n \leq 2k$, then there exists an $\epsilon > 0$ such that for all ϵ' with $0 < \epsilon' < \epsilon$, there is no fair, ex post ϵ'–k resilient equilibrium for an asynchronous unidirectional ring (resp., bidirectional ring, completely connected network).*

3.7 Doing better with cryptography

In the impossibility result of Section 3.6, we implicitly assumed that the agents were computationally unbounded. For example, even though our proof shows that, in the 2-agent case, one agent can always do better by deviating, it may be difficult for that agent to recognize when it has a history where it could do better by deviating. As we now show, if agents are polynomially-bounded and we make an assumption that is standard in cryptography, then we can get a fair ϵ–k-resilient equilibrium in all these topologies, even in the asynchronous settings, as long as $n > k$. Our solution is based on the bit-commitment protocol of Naor [**?**]. Bit commitment ideas can be traced back to the coin-flipping protocol of Blum [**?**].

The key idea of the earlier protocol is that i essentially commits N_i to his neighbor, so that he cannot later change it once he discovers the other agents' random choices. We can achieve essentially the same effect by using ideas from *commitment* protocols [**?**]. In a commitment protocol, an agent Alice commits to a number m in such a way that another agent Bob has no idea what m is. Then at a later stage, Alice can reveal m to Bob. Metaphorically, when Alice commits to m, she is putting it in a tamper-proof envelope; when she reveals it, she unseals the envelope.

It should be clear how commitment can solve the problem above. Each agent i commits to a random number N_i. After every agent has received every other agents' commitment, they all reveal the random numbers to each other. This approach will basically work in our setting, but there are a few subtleties. Naor's commitment protocol requires agents to have access to a *pseudorandom* number generator, and to be polynomially bounded. We can get an ϵ–k-resilient protocol for ϵ as small as we like (provided that Bob is polynomially bounded) by choosing a sufficiently large security parameter for the pseudorandom number generator, but we cannot make it 0. Thus, we actually do not get a fair ex post Nash equilibrium, but a fair ex post ϵ-*Nash* equilibrium.

In the full paper, we show how the protocol in the synchronous setting for the unidirectional ring can be modified by using Naor's commitment scheme to get a protocol $A\text{-}LEAD_i^{ps,uni}$ that works in the asynchronous setting. There is another subtlety here. It is not enough for the commitment scheme to be secure; it must also be *non-malleable* [**?**]. Intuitively, this means that each choice made by each agent j must be independent of the choices made by all other agents. To understand the issue, suppose that the agent i just before the originator on the ring knows every other agent j's random choice N_j before committing to his own random choice; metaphorically, i has an envelope containing N_j for each agent $j \neq i$. (This is actually the case in our protocol.) Even if i cannot

compute N_j, if he could choose N_i in such a way that $\sum_{i=1}^{n} N_i \pmod n$ is 3, he could then choose his id to be 3. If the scheme were malleable, it would be possible for j's choice to depend on the other agents' choices even if j did not know the other agents' choices. Indeed, we want not just non-malleability, but *concurrent* non-malleability. In the protocol, agents engage in a number of concurrent commitment protocols; we do not want information from one commitment protocol to be used in another one. We assume for ease of exposition that Naor's scheme is *concurrently pseudo-non-malleable*; not only can no agent guess other agents' bit with probability significantly greater than $1/2$, they also cannot make a choice dependent on other agents' choices with probability significantly greater than $1/2$, even running many instances of the protocol concurrently. (Note that concurrent non-malleable commitment schemes are known; see [**?**] for the current state of the art.)

Theorem 7. *For all ϵ, if agents are polynomially bounded and pseudorandom number generators exists, then $A\text{-}LEAD^{ps,uni}$ (with appropriately chosen security parameters) is a fair, ϵ–k-resilient ex post equilibrium in an asynchronous unidirectional ring, for all $k < n$.*

The same result holds in the case of a bidirectional ring and completely connected network; we can simply embed a unidirectional ring into the network, and run $A\text{-}LEAD^{ps,uni}$.

4 Discussion and Open Questions

The paper illustrates some issues that might arise when trying to apply game-theoretic approaches to distributed computing problems. Perhaps what comes out most clearly in the case study is the role of *ex post* Nash equilibrium, both in the upper bounds and lower bounds. To us, the most important question is to consider, when applying game-theoretic ideas to distributed computing, whether this is the most appropriate solution concept. While it is the one perhaps closest to standard assumptions made in the distributed computing literature, it is a very strong requirement, since it essentially means that players have no incentive to deviate even if they know nature's protocol. Are there reasonable distributions we can place on adversary strategies? Do we have to consider them all?

Besides this more conceptual question, there are a number of interesting technical open problems that remain. We list a few here:

- We have focused on the case that agents are rational. In [**?**,**?**], we also considered agents who were faulty. Our protocols break down in the presence of even one faulty agent. It is well known that Byzantine agreement is not achievable in a graph of connectivity $\leq 2f$, where f is the number of failures. This suggests that we will not be able to deal with one faulty agent in a ring. But it may be possible to handle some faulty agents in a completely connected network.
- We have focused on leader election. It would be interesting to consider a game-theoretic version of other canonical distributed computing problems. We believe that the techniques that we have developed here should apply broadly, since many problems can be reduced to leader election.

– In [**?**], it is shown that, in general, if we can attain an equilibrium with a mediator, then we can attain the same equilibrium using cheap talk only if $n > 3k$. Here we can use cheap talk to do leader election in the completely connected asynchronous case (which is implicitly what was assumed in [**?**]) as long as $n > k$. Thus, we beat the lower bound of [**?**]. There is no contradiction here. The lower bound of [**?**] shows only that there exists a game for which there is an equilibrium with a mediator that cannot be implemented using cheap talk if $n \leq 3k$. It would be interesting to understand what it is about leader election that makes it easier to implement. More generally, can we refine the results of [**?**,**?**] to get tighter bounds on different classes of problems?
– We have focused on "one-shot" leader election here. If we consider a situation where leader election is done repeatedly, an agent may be willing to disrupt an election repeatedly until he becomes leader. It would be of interest to consider appropriate protocols in a repeated setting.
– We made one important technical assumption to get these results in rings: we assumed that the ring size is known. As we argued earlier, this assumption is critical, since otherwise an agent can create sybils and increase his chances of becoming leader. However, this deviation comes at a cost. The agent must keep simulating the sybils for all future interactions. This may not be worth it. Moreover, ids must also be created for these sybils. If the name space is not large, there may be an id clash with the id of some other agent in the ring. This will cause problems in the protocols, so if the probability of a name clash is sufficiently high, then sybils will not be created. It would be interesting to do a more formal game-theoretic analysis of the role of sybils.