Distributed Computing Building Blocks for Rational Agents

Yehuda Afek, Yehonatan Ginzberg, Shir Landau Feibish, and Moshe Sulamy
Blavatnik School of Computer Science, Tel-Aviv University, Israel
afek@cs.tau.ac.il, jhonathe@mail.tau.ac.il, shirl11@post.tau.ac.il,
moshesulamy@mail.tau.ac.il

ABSTRACT

Following [4] we extend and generalize the game-theoretic model of distributed computing, identifying different utility functions that encompass different potential preferences of players in a distributed system. A good distributed algorithm in the game-theoretic context is one that prohibits the agents (processors with interests) from deviating from the protocol; any deviation would result in the agent losing, i.e., reducing its utility at the end of the algorithm. We distinguish between different utility functions in the context of distributed algorithms, e.g., utilities based on communication preference, solution preference, and output preference. Given these preferences we construct two basic building blocks for game theoretic distributed algorithms, a wake-up building block resilient to any preference and in particular to the communication preference (to which previous wake-up solutions were not resilient), and a knowledge sharing building block that is resilient to any and in particular to solution and output preferences. Using the building blocks we present several new algorithms for consensus, and renaming as well as a modular presentation of the leader election algorithm of [4].

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance Of Systems—*Fault-Tolerance*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms

Algorithms, Reliability, Theory

Keywords

distributed computing; game theory; message passing; consensus; renaming; leader election; knowledge sharing; rational agents

1. INTRODUCTION

A central issue in distributed computing, if not the main point, is multi-processor computing in the face of faulty processors. Many different faults are considered, from fail-stop to Byzantine faults. Recently, a new model, that of distributed game theory has emerged, in which processors, now called rational agents, are not faulty but may cheat in order to increase their profit according to some utility function [4, 1, 18, 2, 5].

For example, if an agent can cheat and cause an election algorithm to elect itself as a leader the agent profits one dollar, or if it can cheat causing a consensus algorithm to agree on 1 although its input was 0, then its profit is again some positive value. A basic property of the utility function considered in all cases, which is called *solution preference*, captures the notion that agents cannot gain if the algorithm fails (a zero profit is assigned to all the agents if the algorithm fails). As a result, though agents have preferences, they have no incentive to fail the algorithm.

The challenge for distributed algorithms in the context of game theory is to design algorithms that reach equilibrium, i.e., such algorithms in which the players, agents, have no incentive to cheat. That is, for any utility function materializing any combination of preferences, the algorithm should ensure that in any run the agents lose, or gain absolutely nothing, by deviating from the algorithm (trying to cheat). The algorithm, therefore, needs to govern the behavior of the rational agents and to punish agents (reducing their profit) for any deviation from it.

We present a comprehensive examination of the preferences and utilities of rational agents in a distributed network setting, building on the results presented in [4]. We take a closer look at factors that might affect the utilities of such agents in a distributed setting, and the actions that they may take as a result of these factors. Based on this examination, we provide a refined game-theoretic model which encompasses these preferences and gives insight to natural utilities that rational agents in a distributed setting may have. Specifically, we examine factors such as an agent's possible preference for a certain output or an agent's desire to send fewer messages.

Having identified the major factors which could influence the agents' utilities and modeled them, we go on to present building blocks for solving common problems in distributed computing which are resilient to this rational behavior. Such operations include performing wake-up and knowledge sharing in a variety of network topologies. In a network with n agents, our protocols achieve $(n-1)$-*strong equilibrium* for synchronous networks and $\frac{n}{2}$-*strong equilibrium* for asynchronous networks. Informally, a protocol which achieves k-*strong equilibrium* is a protocol in which no coalition of size at most k can improve its utility by deviating from the protocol. These protocols are in fact building blocks which can be used to construct many other distributed algorithms for rational agents.

This research was supported by the Ministry of Science and Technology, Israel, and by the Israel Science Foundation (grants 6693/11 and 1386/11)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

PODC'14, July 15–18, 2014, Paris, France.
Copyright 2014 ACM 978-1-4503-2944-6/14/07 ...$15.00.
http://dx.doi.org/10.1145/2611462.2611481 .

Finally, we demonstrate the usefulness of the two building blocks in the construction of new algorithms that achieve strong equilibrium for consensus, renaming, and reconstructing the leader election algorithms of [4].

1.1 Related Work

The work of Abraham, Dolev and Halpern [4] goes back and questions the fundamental problems in distributed computing, under the assumption that processors may behave as rational agents. In their work they have examined, among other things, protocols for various settings of the leader-election problem that not only deal with acquiescent or Byzantine processors but also with the *rational* processors.

It seems that the connection between distributed computing and algorithmic game-theory stemmed from the problem of secret sharing. One such example, where k-out-of-n secret sharing [28] comes in handy, is when participants do not fully trust each other with a secret, yet they would like to reconstruct it using their private parts of the secret, when there is an acceptance of more than k participants. It seems that known solution to the secret sharing problem do not handle the reconstruction phase, that can be modeled as a synchronous network of n processors, where each of them wants to learn the secret and also prefers that as few other processors as possible will learn the secret. This raises an interesting question: is there a protocol with a fixed running time that solves the problem? Halpern and Teague showed [18], among other things, that the problem cannot be solved with a bounded distributed protocol for the solution concept of iterated admissibility (i.e., iterated deletion of weakly dominated strategies). Further works continued the research on secret sharing and multiparty computation when both Byzantine and rational agents are present [2, 8, 13, 16, 23, 17]. At the same time, another line of research has shown applicative and theoretical results for cooperative services for what is known as the BAR model (Byzantine, acquiescent [33] and rational) [5, 26, 33] . Another related line of research asks whether a problem that can be solved with a mediator can be converted to a cheap talk based solution [2, 3, 6, 7, 9, 11, 21, 22, 24, 29, 30, 31]. This approach is very strong because there are many results that are based on a mediator, which other players cannot trust under the rationality assumption, if we can convert mediator based protocols to be based on cheap talk, many of the previous works that do not assume rationality may become relevant under this assumption.

2. MODEL

Our model consists of n processors in a distributed network. Each processor is a rational agent, and thus has *preferences* over the outcome of the protocol. Whenever a processor can increase its *expected utility*, it will deviate from any given protocol, in order to benefit itself. For protocols to be resilient to rational agents, we require them to reach *equilibrium*, such that agents have no incentive to deviate from the protocol. To reach this, we require agents to have a utility function satisfying the *solution preference*, such that agents do not prefer an outcome of the protocol where there is no solution to the problem, over an outcome in which there is a solution.

Following [4] we use the standard message-passing model, where all the processors are *rational agents*. The network is a simple, strongly-connected, finite graph. Each node represents an agent (i.e., a processor), and the edges represent communication links through which pairs of agents exchange messages. Agents may send messages through their outgoing links, and can identify the link an incoming message is received over. Denote n the total number of agents in the network. Each agent is assigned a unique *id* taken from a common name space, assumed to be the set of natural numbers. We assume that the topology of the network and n are common information, thus they are a-priori known to all. Each agent additionally knows its *id* and input, if it received any input as part of the protocol, but not the *id* or input of any other agent.

In this paper, both the synchronous and asynchronous network models are considered. In a synchronous network, agents execute the protocol in rounds. Each round r consists of agents receiving all messages sent on their incoming links in round $r-1$ (if $r>0$), performing any computations and updating internal variables, and sending messages on their outgoing links. In the asynchronous network we assume the standard asynchronous event driven communication model - in which message transmission delay is unbounded but finite [15].

Three fundamental distributed computing problems are considered here; *renaming*, *consensus* and *leader election*.

The *consensus* problem: each agent p start with an input i_p, and its output o_p is initialized to $\perp$. The processors must then agree on the output. That is, the processors communicate, and at the end of the protocol run (after all processors stop, assuming no processor fails) $\forall i \in \{1,..,n\}, o_i \neq \perp$ and: *(1)* **Agreement**: for each processor p, $o_p = v$ for some value v; (2) **Validity**: if $\exists j$ s.t. $o_j = v$, then there is some processor k for which $i_k = v$.

The *leader election* problem is the same as the consensus problem where the input of each agent is its *id*.

The *tight renaming* problem: each agent has a unique *id* and needs to choose a unique name from the tight range $1\ldots n$. Denote o_p to be the agent's output. Each agent p writes its new name in o_p, satisfying the following requirements: *(1)* **Agreement**: No two processors obtain the same new name. $\forall_{x,y}\, o_x \neq o_y$; (2) **Validity**: Each new name is an integer in the set $[1\ldots n]$.

2.1 Game Theoretic Model

Unlike processors in the standard fail-stop or Byzantine models, rational agents are not faulty. Instead, agents have preferences over the outcome of the protocol. If an agent can improve upon its preferences, it will deviate from the protocol (i.e., cheat).

For example, a rational agent p might prefer to send fewer messages in the protocol. In such a case, it might be lazy and not fully participate in the protocol, or otherwise deviate from it in any way in which its expected number of messages sent is as few as possible, so long as it does not cause the protocol to fail.

Formally, each agent p has a utility function u_p on the final states of the protocol. The final state of the protocol encompasses the execution leading to the state as visible by p and the output of the protocol in this final state. The utility function represents how good or profitable the result is, from p's point of view. The higher the utility, the better. An agent can have *any* utility function representing any preference; it can have a preference over the output of the algorithm, the number of messages it has sent, the amount of computation it does, or any other preference or combination of preferences, so long as the algorithm terminates in a legal global state, satisfying the solution preference (definition 2.1, below).

Let c be the state of p in the protocol execution, and let S_p be the set of all possible final states that may be reachable and are consistent with the current state of p, assuming all agents other than p follow the given protocol and the next step of p is op, which may be deviating from the protocol (i.e., cheating). For each final state $s \in S_p$, let x_s be the probability, estimated by p, that s is reached by taking the step op. The *expected* utility of p after taking step op in state c is then:

$$\mathbb{E}_{c,op}\left[u_p\right] = \sum_{\forall s \in S_p} x_s \cdot u_p(s)$$

If by deviating from the protocol by taking a step op, p can increase its expected utility, we then say that agent p has an *incentive* to deviate from the protocol (i.e., cheat).

For example, in the consensus problem, an agent p might have a preference over the agreement value, such that it prefers an agreement on 1 over an agreement on 0. Its utility function u_p might look as follows:

$$u_p = \begin{cases} 1 & \text{1 is decided} \\ 0 & \text{0 is decided or no agreement is reached} \end{cases}$$

At every point during the protocol execution, p estimates the probability for each possible outcome of the protocol, according to its knowledge while assuming all other agents follow the given protocol. Let x be the probability, estimated by p, that the protocol outcome results in an agreement on 1, then the expected utility of p is the following:

$$\mathbb{E}\left[u_p\right] = x * u_p(1) + (1-x) * u_p(0)$$

Since p is a rational agent, it will deviate from the protocol whenever it can increase its expected utility by any positive value.

To prevent rational agents from deliberately failing the algorithm (i.e., becoming *Byzantine* rational agents), we assume agents have utility functions that penalize the agents if the protocol errs. The *solution preference* guarantees that an agent never has an incentive for the protocol to fail.

DEFINITION 2.1 (SOLUTION PREFERENCE). *Let O be the set of all possible executions of the protocol. Let $o_L \in O$ be a legal execution of the protocol (e.g., producing a legal output), and let $o_E \in O$ be an erroneous execution of the protocol (e.g., producing an illegal output). To satisfy the solution preference, the utility function u_p of an agent p must satisfy the following:*

$$\forall o_L, o_E : u_p(o_L) \geq u_p(o_E)$$

Meaning, the utility function u_p of an agent p never assigns a higher utility to an outcome of the protocol in which there is either no solution or an erroneous solution than to an outcome in which there is a legal global solution.

We say that a protocol is resilient to rational behavior if it reaches *Nash equilibrium*. A protocol that reaches Nash equilibrium is guaranteed to execute correctly in the face of rational agents, with no agent being able to improve its utility by deviating from the protocol.

DEFINITION 2.2 (NASH EQUILIBRIUM PROTOCOL). *A protocol is said to reach Nash equilibrium if no agent can unilaterally increase its expected utility by deviating from it while assuming that all other agents follow the given protocol.*

While Nash equilibrium deals with a single agent deviating from the protocol, sometimes agents are in coalitions, working together to improve their utilities. We define a coalition of size t as a set of t rational agents.

We say that a protocol is resilient in the face of coalitions if it reaches *strong equilibrium*. A protocol that reaches t-strong equilibrium is guaranteed to execute correctly, with no coalition of k agents, where $k \leq t$, being able to improve its utility by deviating from the protocol, even in a coordinated way.

DEFINITION 2.3 (t-STRONG EQUILIBRIUM PROTOCOL). *A protocol reaches t-strong equilibrium if no coalition of size $k \leq t$ can increase the expected utility of one or more agents in the coalition by deviating from the protocol, assuming all agents not in the coalition follow the given protocol.*

In game theory, a game requires three things: players, the players' utilities, and the strategies available for each player. The distributed model with rational agents that we discuss in this paper, fulfills this paradigm in the following manner: 1) The set of players P is the set of all agents (i.e., processors) in the network. 2) Each player $p \in P$ has a utility function u_p, which can be any function that satisfies the solution preference (definition 2.1). 3) The strategy of each player p determines the messages that p sends. Player p can choose to send none, one, or several messages at any point in the protocol, and by that it executes its selected strategy.

2.2 Truthfulness

Consider a consensus protocol which reaches agreement by deciding on the maximum value of all agents. Consider an agent p with the utility function u_p from the consensus example in the previous subsection. Agent p has an incentive to lie about its input: provided it received an input of 0, it benefits by claiming an input of 1, since it raises the probability of 1 to be decided, thus raising its expected utility. In an execution where all agents start with an input of 0, agreement of 1 is reached, and thus the consensus fails since the validity requirement is violated. This shows that, even though each agent satisfies the solution preference, it is not enough in order to solve protocols in the game-theoretic model.

To overcome this, in game theory we design mechanisms that are *truthful*, such that agents do not have an incentive to lie about their input, regardless of any a-priori knowledge they may have of the input or strategies of other players.

However, in distributed protocols, though we assume no a-priori knowledge exists, such a requirement is not enough. Distributed protocols allow agents to deviate by lying about the values of other agents. Thus, even though we might design a truthful mechanism in which an agent does not have an incentive to lie about its input, in a distributed environment it might still have an incentive to lie about the values of other agents.

For example, a well-known mechanism in game theory is second price auction [32]. In it, the players are bidders submitting bids for an item. Each player has a private value v_i which is its value for the item, and submits a bid b_i to the auctioneer. The winner of the item is the player who sent the highest bid, and he pays a value equal to the second-highest bid. Let b_j be the second-highest bid, then the utility of the winner i is defined as $v_i - b_j$. The utility of all other players is 0.

This mechanism is known to be truthful, such that players have no incentive to bid any value other than $b_i = v_i$, even if they know the values and submitted bids of all other players.

However, in a distributed network, not all agents are necessarily directly linked to the auctioneer in the network. Thus, they may need to communicate through other agents in the network. These agents may lie regarding their input. An agent p, separating some bidders from the auctioneer, has a clear incentive to send $b_i = 0$ for each agent i sending its bid in the network through p, thus increasing p's chances of winning the item, and possibly also decreasing the amount paid for it. This is true even if p is truthful with regard to its own input, having no incentive to send a bid other than its true value for the item, i.e., $b_p = v_p$.

Following this example, we can clearly see that the existing definition of a truthful mechanism is not enough. For distributed protocol to be truthful, we require agents to have no incentive to lie regarding not only their input, but also the input of other agents.

DEFINITION 2.4 (TRUTHFUL PROTOCOL). *A protocol is said to be* truthful *if no agent participating in the protocol has incentive to lie about its input, or any other data it shares as part of the protocol. Meaning, an agent's utility for an outcome achieved by lying is no greater than its utility achieved by telling the truth.*

3. AGENTS WAKE-UP

In many distributed computing protocols, we assume that all agents start the protocol at the same time. However, this is usually not the case. Usually, processors are either woken up spontaneously at arbitrary points of time by the scheduler, or when receiving the first message over one of their input links.

The *agents wake-up* building block deals with waking up all agents. The building block can be executed as the first phase of any protocol, thus ensuring all agents are awake and start together.

A protocol for the building block starts by having agents wake-up at arbitrary points of time by the scheduler. The agents need to communicate so that, at the end of the protocol, all agents are awake. The protocol makes sure that, upon termination, agents also know the *id*s of all other agents. In addition, the protocol needs to succeed in the face of rational agents—the building block reaches t-strong equilibrium for $t < n$.

A wake-up procedure was included within each of the leader election protocols presented in [4]. However, it does not reach Nash equilibrium. The protocol fails in the presence of lazy rational agents, that prefer sending fewer messages, in the following way:

Consider a wake-up procedure that requires agents to send and forward messages around the ring, where only messages of the agent with the highest *id* are forwarded. Consider an agent p with a preference for sending fewer messages having, for example, the following utility function, where m_p denotes the number of messages p sends:

$$u_p = \begin{cases} 1 - \frac{m_p}{n} & \text{wake-up is successful} \\ 0 & \text{wake-up is unsuccessful} \end{cases}$$

In the procedure, each agent sends no more than n messages, thus u_p satisfies the solution preference (definition 2.1).

When p wakes up, it prefers not to send any wake-up message. It relies on another agent to wake up the others, doing the work for it—thus raising p's expected utility. In addition, if p is the only agent to wake-up spontaneously, no solution will be reached and the protocol will fail.

Even though it is possible that no solution will be reached, the utility u_p satisfies the solution preference (definition 2.1). This is because only the *expected* utility of p is raised, so p will "take the chance" of failing the protocol, given the positive probability that it will not fail and p will benefit.

The wake-up protocol must be resilient to the utility function discussed in the above example, and to any other utility function satisfying the solution preference (definition 2.1).

Another possible preference an agent might have is a preference over properties of its *id*. While the building block itself is oblivious to the agents' *id*s, it might be used by a protocol that is not.

For example, consider a leader election protocol that is run over the wake-up building block, where the agent with the highest *id* is elected leader. If we execute the protocol, an agent might have an incentive to lie about its *id*. Consider an agent p with the following utility function:

$$u_p = \begin{cases} 1 & p \text{ is elected leader} \\ 0 & p \text{ is } \textbf{not} \text{ elected leader, or no leader is elected} \end{cases}$$

Agent p will deviate from the building block by lying about its *id*. It prefers to send messages and pretend to have the highest *id*. While it is possible that another agent will have the same *id* that p lied about, according to u_p it still raises its expected utility by raising its chances to be elected leader.

Following this example, for the building block we assume that the protocols built on top of the wake-up building block are oblivious to the agents' *id*s. Indeed, in [4], a leader election protocol is suggested which elects the leader by choosing a random *id*. This ensures that an agent's specific *id* has no affect on the outcome of the protocol (the elected leader), and thus it can be preceded by the wake-up building block proposed here.

With these possible utilities in mind, we present protocols for the wake-up building block. In a synchronous network, all agents complete the protocol in the same round. The protocols reach t-strong equilibrium for $t < n$, for n rational agents with any utility function that satisfies the solution preference (definition 2.1).

The idea behind our procedure in order to achieve these properties, and prevent an agent from not sending wake-up messages when it should, is to require *all* agents to send wake-up messages. An agent does not finish the protocol before receiving wake-up messages from all other agents. Thus, an agent that does not send a wake-up message causes the protocol to fail, in contradiction to the solution preference (definition 2.1). A detailed description of the protocols and full proofs are provided in the full paper.

3.1 Ring Wake-Up Protocol

The protocol for wake-up in a ring is as follows: upon waking up, each agent sends a wake-up message that goes around the ring. An agents finishes the protocol when it has received a wake-up message from all other agents.

The wake-up message contains both the *id* of the originator of the message, and a hop counter k—initialized to $n-1$ by the originator, and decremented by each successor. This counter k represents the distance from the current recipient to the originator, in the direction of the message.

For a bidirectional un-oriented ring, each agent arbitrarily decides on a direction for its wake-up message. When agents receive messages on one link, they forward it on the other link. When finished, the ring orientation can be decided according to the direction chosen by the highest *id*. Thus, for a bidirectional ring, we further assume that the direction of the messages is irrelevant, and no agent has a preference on the direction (i.e., clockwise or counterclockwise).

3.2 Complete Network Wake-Up Protocol

The protocol in a complete network is as follows: upon waking up, each agent sends a wake-up message containing its *id* to all agents. An agent finishes the protocol when it has received a wake-up message from all other agents, and has sent wake-up messages to all other agents.

3.3 General Network Wake-Up Protocol

The protocol for a general network is as follows: upon waking up, each agent broadcasts a message containing its *id* through the network. An agent finishes the protocol when it has received a wake-up message from all other agents, i.e., n wake-up messages, as n is known a-priori.

In a synchronous network, this protocol is not enough. Since this is usually a preliminary building block, we require agents to finish the protocol in the same round, i.e., start together the next step of the protocol using this building block. To this end, a hop counter k is attached to each broadcast message, initialized to 0 and incremented in each hop. When a wake-up message is received by an agent, it can determine from the hop counter the round in which the originator of the wake-up started broadcasting. Once an agent receives all n wake-up messages, it can tell the earliest round at which an agent woke up. To end in the same round, an agent terminates $2n$ rounds after that first round.

Unlike the previous wake-up protocols, the protocol for a general network has an additional requirement.

CLAIM 1. *To reach t-strong equilibrium for $t < n$, the underlying topology must be at least $(t+1)$-connected.*

Sketch of proof: Assume by contradiction that the network is not $(t+1)$-connected, then a coalition of t agents might separate two parts of the network. Such a coalition can be lazy and decide not to wake up one of the separated parts, instead making up *id*s for that part and sending it to the rest of the network, essentially faking the participation of agents in that part of the network, thus saving itself from sending some messages. The protocol thus does not reach equilibrium. That is a contradiction, thus the network must be at least $(t+1)$-connected.

4. KNOWLEDGE SHARING

In many distributed algorithms, each process needs to know the private values of all other processes in order to perform the desired computation. The *knowledge sharing* building block deals with sharing the private values among all agents.

Facing rational agents poses a problem that an agent, once it knows all other values, might lie about its own value to increase its utility. Thus, the presented protocols for the building block make sure that each agent commits to its own value before learning the values of all other agents.

Each agent p starts with a value v_p, and needs to learn the values of all other agents $V = \{v_1, \dots, v_n\}$. For this building block, we assume agents know the *id*s of all other agents. If not, we can simply precede it with the agents wake-up building block.

A solution for knowledge sharing seems trivial: each agent broadcasts its value to all other agents. Once an agent receives the values V of all other agents, it terminates. However, this solution does not reach equilibrium. An agent can easily cheat regarding its value in order to increase its utility.

For example, consider a leader election protocol executed using this building block, where the leader is elected in the following way: Each agent's value is a random number, and the sum of all numbers *mod n* is the rank of the *id* elected leader. If we execute the protocol in the game-theoretic model, an agent might have an incentive to lie about its value.

Consider an agent p with the following utility function:

$$u_p = \begin{cases} 1 & p \text{ is elected leader} \\ 0 & p \text{ is } \textbf{not} \text{ elected leader, or no leader is elected} \end{cases}$$

Agent p might prefer to delay its participation in the protocol, so that it knows the values of all other agents. Once p knows all values V, it can then lie about its own value v_p—ensuring that it is elected as a leader.

Adapting the same techniques used in [4] to overcome this issue, we ensure that no agent learns the values of other agents before sending its own value. In addition, when facing coalitions, the protocols make sure that agents outside the coalition learn the values of all agents in the coalition before agents in the coalition learn the values of all other agents.

No matter how protocols for the building block are built, if it is used by a protocol that is not truthful (definition 2.4), an agent can have an incentive to lie about its value, regardless of the knowledge sharing protocol.

For example, consider a consensus protocol built on top of this building block, that decides on the maximum value of all agents. If we execute the protocol in the game-theoretic model, an agent might have an incentive to lie about its value. Consider an agent p with value $v_p \in \{0, 1\}$, with the following utility:

$$u_p = \begin{cases} 1 - d & \text{the consensus decision is } d \\ 0 & \text{no consensus is reached} \end{cases}$$

In an execution where $v_p = 1$, agent p prefers broadcasting $v_p = 0$. In case p is the only agent with an input of 1, the protocol decides 0 instead of 1 and p benefits. Otherwise, the decision remains the same and p does not benefit, but does not lose, either. Therefore, p raises its expected utility by deviating from the protocol, and thus it does not reach equilibrium.

In addition, consider an execution where all agents have the utility u_p, but all received 1 as input—the agents all broadcast the value 0, which is decided. It was not the input of any agent, and thus the validity requirement is violated.

This shows that, while the knowledge sharing building block does not violate truthfulness, it is dependent upon the *full knowledge property* of the protocols that use it. Therefore, according to 4.1, any protocol that uses the knowledge sharing building block must make sure that it satisfies the full knowledge property for the entire algorithm to be truthful.

DEFINITION 4.1 (FULL KNOWLEDGE PROPERTY). *For each agent that does not know the values of all other agents $V = \{v_1, \dots, v_n\}$, any output of the protocol is still equally possible.*

For example, even though the previous example shows that the consensus protocol fails with this building block, it is possible to achieve agreement with this building block. The consensus protocol shown in that example does not satisfy 4.1, since an agent can lie about its value in order to affect the outcome of the protocol, without knowing the values of other agents, thus raising its expected utility. In 5.3, however, we show a protocol that reaches agreement while satisfying 4.1, thus using the knowledge sharing building block properly.

With these utilities in mind, we present protocols for the knowledge sharing building block. The building block assumes the following: each agent p starts with a value v_p, agents know the *id*s of all other agents, and, in synchronous networks, all agents start the protocol together.

The protocols reach t-strong equilibrium, where $t < n$ for synchronous networks and $t < \frac{n}{2}$ for asynchronous networks, for n rational agents with any preference that satisfies the solution preference (definition 2.1), and any value that satisfies the full knowledge property (definition 4.1). In a synchronous network, all agents complete the protocol in the same round. A detailed description of the protocols and full proofs are provided in the full paper.

4.1 Knowledge Sharing in a Synchronous Ring Network

In round 1, each agent p sends a message with its value v_p. These messages are passed around the ring by all the agents, until each message returns to its originator. At this point, all the agents have received all of the data. If an agent p does not receive a single message from its predecessor each round, or receives a duplicate message it received before, it sets $o_p = \perp$, thus ensuring the correct participation of all due to the solution preference (definition 2.1). The message complexity of the protocol is n^2, and the time complexity is n rounds.

4.2 Knowledge Sharing in a Synchronous Complete Network

The protocol consists of a single round, in which each agent sends its value to all other agents, and receives the values of all other agents. If an agent p does not receive messages from all agents, it sets $o_p = \perp$, thus ensuring the correct participation of all due to the solution preference (definition 2.1). The message complexity of the protocol is n^2, and the time complexity is 1 round.

4.3 Knowledge Sharing in an Asynchronous Ring Network

The synchronous ring solution does not work in an asynchronous ring. Agents can not determine when to send and forward messages, and we can reach a state in which an agent might learn the values of all other agents before sending its own value to its successor. The agent might then have an incentive to lie about its value, as described in the leader election example, above.

To solve this issue, we propose a protocol similar to the one suggested in [4] for leader election in an asynchronous ring. We have the agent p with the lowest *id* be the *originator*, which starts the protocol by sending its value to its successor. Its successor then sends a message with its own value to its own successor, and so forth.

Messages are passed around the entire ring in the same manner. When the originator receives a message from its predecessor, this is an indication that the first cycle, in which each agent learned the value of its predecessor, has completed and the initiator begins the second round of messages by sending the value of its predecessor to its successor. Messages are passed around the entire ring again, in the same manner, where each agent sends the data of its predecessor. From cycles 2 to $n-1$, each agent sends the value it has received in the previous cycle, s.t., in round v agent u sends the input value of agent $u - v + 1 \mod n$. It is easy to see that the message and time complexity of the protocol is $n(n-1)$.

4.4 Knowledge Sharing in an Asynchronous Complete Network

For an asynchronous complete network, we can use the protocol presented in 4.3 for knowledge sharing in an asynchronous ring by embedding a unidirectional ring in the network. We embed the ring by sorting the *id*s of all agents from lowest to highest. Each agent's successor is the agent with the *id* following its own on the list. The successor of the agent with the highest *id* is the agent with the lowest *id*, thus forming a ring.

However, the added connectivity of a complete network hurts the coalition resiliency of the protocol in 4.3. While a coalition of size $t < \frac{n}{2}$ still must commit to its data before learning the data of all other agents, it can still break the protocol in 4.3.

Since the network is completely connected, any two agents in the network can communicate directly with each other, even if they are not neighbors in the embedded ring. Splitting the ring into two (or more) parts, agents in a coalition can share information and thus lie to the split parts of the ring regarding the input of the other part. This way, the coalition can make sure that the protocol succeeds (all agents have a valid output); however, the coalition has control over the outputs and can therefore alter the input messages sent in order to reach an output they prefer.

To prevent this, we use the added connectivity to our benefit. Upon finishing the protocol in 4.3, each agent sends its value to all other agents. This prevents agents from being able to lie regarding the values of other agents, thus making agents are truthful regarding the values that they send.

4.5 Knowledge Sharing in a Synchronous General Network

For a synchronous general network, we use a similar protocol to 3.3 for wake-up in a general network. Each agent p broadcasts its value v_p through the network. Once p receives the values of all other agents, it terminates.

If an agent p receives contradicting messages, or does not receive all values after n rounds, it sets $o_p = \perp$ and terminates, thus ensuring the correct participation of all due to the solution preference (definition 2.1).

As in the wake-up protocol in 3.3, the knowledge sharing protocol for a synchronous general network has an additional requirement:

CLAIM 2. *To reach t-strong equilibrium for $t < n$, the underlying topology must be at least $(t+1)$-connected.*

Sketch of proof: Assume by contradiction that the network is not $(t+1)$-connected, then there is a group of no more than t agents that split the graph. Assume that these agents form a coalition. Then, for any knowledge sharing protocol, the agents can cheat by lying on the input values of agents connected in the network only through the coalition.

The message complexity of the protocol is n^2, and the time complexity is n rounds.

5. USING THE BUILDING BLOCKS

Here we demonstrate the building blocks approach [14] by presenting Leader election, Renaming and Consensus algorithms that are truthful and reach Nash equilibrium building on the agents wake-up and knowledge sharing building blocks.

The protocol for leader election mostly borrows the ideas presented in [4]. However, it demonstrates how to adjust a problem to meet the requirements of the building blocks. In the protocols for renaming and consensus, we demonstrate how to use the building blocks further, for more complicated problems with less trivial solutions.

5.1 Leader Election

Here is a building blocks construction of the leader election algorithm of [4]:

PROTOCOL 5.1 (GAME-THEORETIC LEADER ELECTION). *For each agent p:*

(1) Initiate the wake-up building block.

(2) Let $v_p = random(1, \ldots, n)$.

(3) Knowledge sharing to learn $V = \{v_1, \dots, v_n\}$.

(4) For any k, if $(v_k < 1)$ or $(v_k > n)$, set $o_p = \perp$ and terminate.

(5) Calculate $(N = \sum_{k=1}^{n} v_k \mod n)$, and output: $o_p \leftarrow$ (the N-th rank id).

It is easy to see that protocol 5.1 follows the requirements of our building blocks, and correctly elects a leader while reaching equilibrium.

5.2 Renaming

Having a solution to the leader election problem that reaches equilibrium under the game-theoretic model, there seems to be a naive renaming protocol: execute leader election, then the leader randomizes the new names and sends them to all agents. However, this naive protocol does not reach equilibrium; the leader might have a preference for a certain assignment of names, and can assign them as it sees fit.

With that in mind, another possible protocol is to execute the leader election protocol n times. Each execution, the leader is given a name from $[1 \dots n]$, in order, and removed from the list of possible leaders for the next execution. While this protocol solves the problem, it has a very high message and time complexities. In addition, perhaps requirement 2.1 is too strong? We can think of a case where an agent cares only about certain names (or a single name), and not the entire solution.

For example, consider an agent p with the following utility:

$$u_p = \begin{cases} 1 - \frac{m_p}{n^2} & p \text{ knows its new name and sent } m_p \text{ messages} \\ 0 & p \text{ does not know its new name} \end{cases}$$

While u_p does not satisfy the solution preference (definition 2.1), it is still a reasonable utility to consider. Thus, once the agents have chosen a new name for p, it will stop sending messages and no longer participate in the protocol, thus raising its utility. Thus, the protocol does not reach equilibrium. In addition, the protocol fails and chooses no new names after the new name of p.

To solve this issue, we propose a protocol for one-shot renaming—where either all new names are chosen, or none is. With one-shot renaming, an agent can learn its new name only if all new names are assigned to all agents. Thus, the protocol reaches equilibrium, even when facing an agent with the preference presented in the above example.

The protocol is similar to the protocol presented in 5.1. However, instead of randomizing a single number, each agent randomizes n numbers, essentially running n leader elections simultaneously. The agents then share all n^2 numbers, and assign new names accordingly. The protocol works as follows:

PROTOCOL 5.2 (GAME-THEORETIC RENAMING). *For each agent p:*

(1) Initiate the wake-up building block.

(2) Choose a set of n random numbers as value:
$\forall k \in \{1, \dots, n\} : v_p[k] \leftarrow random(1, \dots, k)$.

(3) Knowledge sharing to learn $V = \{v_1, \dots, v_n\}$.

(4) For any k, j, if $(v_k[j] < 1)$ or $(v_k[j] > j)$, set $o_p = \perp$ and terminate.

(5) Calculate the set of sums:

$$\forall j \in \{1, \dots, n\} : N_j = \sum_{k=1}^{n} v_k[j] \pmod j.$$

The agent with the N_n^{th} highest id is assigned the new name n, and then removed from the list of ids, thus it is ignored in the following iterations. The agent with the N_{n-1}^{th} highest id is then assigned the new name $(n-1)$ and removed from the list of ids, and so on.

(6) Set o_p to p's new name as calculated in the previous step.

This randomization makes sure that each name chosen is random, thus satisfying the full knowledge property (definition 4.1). Again, it is easy to see that it satisfies all of the requirements for the building blocks, and assigns new names while reaching equilibrium.

5.3 Consensus

In [4], it is implied that the consensus problem can be solved in the presence of rational agents by electing a leader, and having the leader choose the consensus value as its input. However, this algorithm does not reach equilibrium, and may not uphold the validity requirement even if only one rational agent is present. The elected leader may have a preference over the agreement value, and thus has an incentive to lie about its input. It takes the chance of reaching no agreement, but its expected utility is raised overall, thus it benefits.

We can execute the building blocks, where the value for knowledge sharing is each agent's input to the consensus problem. However, how we calculate the decided value after the knowledge sharing is critical.

For example, agreeing on the maximum value of all agents does not work in the presence of ration agents, and does not reach equilibrium. An agent p with a preference for agreement on 1 has a clear incentive to share a value $v_p = 1$, even if its input to consensus was 0. Thus, we propose the following protocol for truthful consensus:

PROTOCOL 5.3 (GAME-THEORETIC CONSENSUS). *For each agent p:*

(1) Initiate the wake-up building block.

(2) Knowledge sharing to learn $V = \{v_1, \dots, v_n\}$.

(3) For any k, if $v_k \notin \{0, 1\}$, set $o_p = \perp$ and terminate.

(4) If the total number of 1's is odd or equals n, set $o_p = 1$, else set $o_p = 0$.

First, it is clear that without rational agents, the protocol solves consensus. In addition, since deciding according to the parity of all of the values, agents can not affect the outcome. An agent might learn the values of all agents but 1, and still has no incentive to lie about its value, as it can not determine what influence it will have on the decision, thus it satisfies the full knowledge property (definition 4.1).

When the number of agents n is odd, the protocol is truthful (definition 2.4), since no agent can benefit by lying about its input. However, when n is even, it is not satisfied. If, for example, a coalition of $(n-1)$ agents have a preference to agree on 1, each agent p in the coalition can lie such that $v_p = 1$. In such a case, the agreement will be 1, regardless of the remaining agent's input, and thus we do not reach equilibrium, and do not satisfy the validity requirement in the case that the input of all agents is 0. To handle it, for networks where n is even, each agent's input for the knowledge sharing also includes a random number, as in the leader election protocol 5.1, thereby electing a leader at the same time. The input of the leader is then considered *twice* in the computation, thus making the protocol truthful for even size networks as well.

The protocol satisfies all of the requirements for the building blocks, and reaches agreement while reaching equilibrium.

6. EXTENSIONS AND TRADEOFFS

6.1 Improving the Message Complexity of Knowledge Sharing

In most of the protocols presented in this work, our main aim was to obtain algorithms which would be most resilient to coalitions, even at the cost of increased message or time complexity. To illustrate the tradeoff between the two approaches, we present an alternative knowledge sharing protocol for a synchronous bi-directional ring network. The protocol significantly improves the message complexity, at the cost of reduced coalition resiliency.

The protocol *KSP* works as follows: We have the agent p with the lowest *id* (after the wake-up) be the *originator*, which starts the protocol by sending a message containing its value $\langle v_p \rangle$ both to its right and to its left neighbors, i.e., both clockwise and counter-clockwise accordingly. Each agent q that receives a message concatenates its own value to the message such that the message is now $\langle v_p; v_q \rangle$ and sends it to its neighbor in the direction in which the message was travelling, i.e., clockwise or counter-clockwise. Each message is sent one time completely around the ring until it gets back to the initiator. This ensures that all agents receive all data. However, messages are sent on both sides of the ring, thus agents send their value before receiving the values of all other agents and, due to the full knowledge property (definition 4.1), are guaranteed to be truthful.

If the number n of agents in the network is odd, the protocol works as-is. If n is even, the initiator sends the message to the right in round 1, and to the left in round 2. This is needed to prevent the possibility of some agent receiving both messages at the same round which could allow that agent to modify the values in the message in order to increase its utility.

The message complexity of the algorithm is $O(n)$ rather than $O(n^2)$ of the protocol presented in 4.1. The protocol is, however, a lot less resilient to coalitions, as it can not withstand even a coalition of 2 agents.

Furthermore, in some cases the bit-complexity of the above algorithm can be further reduced. One such example is a protocol for solving binary consensus. Our binary consensus protocol is based on parity and can therefore be computed in an *on-the-fly* manner using the following protocol: Two messages are initiated by the agent with the lowest *id*, as described in the *KSP* protocol, but now the messages contain 2 bits b_1 and b_2. Bit b_1 is used to indicate whether all inputs so far have only been 1 and is initialized to 1. Bit b_2 is initialized according to the value v_p of the initiator, meaning, if $v_p = 0$, b_2 is initialized to 0 otherwise b_2 is initialized to 1. Each agent q that receives a message does the following: if v_q equals 0, set $b_1 = 0$, otherwise, switch the value of b_2, that is, set $b_2 = !b_2$. Continue to send the message as described above. Once each agent has received both a message from the left and a message from the right, it can compute the output of the consensus based on those 2 messages alone.

6.2 Fail-Stop Model

We extend our model to a *fail-stop* model with rational agents, where an agent can fail by stopping (i.e., crashing) at any point in the protocol. From that point on it does not participate in the protocol—the agent can not send messages or perform computations, and any messages sent to it are discarded. A protocol is said to be *k-fault-tolerant* if it can withstand up to k faulty agents, and still produce a correct output for each of the non-faulty agents.

This model poses new challenges for solving problems in our game theoretic model. In previous protocols, when an agent does not participate, we "punish" it by failing the algorithm. Thus, we ensure its correct participation, due to the solution preference (definition 2.1). However, in the presence of faulty agents, such a solution will not work. An agent can fail-stop, stopping its participation and preventing the protocol from reaching a solution to the problem. Thus, our previous protocols are not even 1-fault-tolerant.

A possible solution might be to proceed with the protocol, regardless of which agents participate. However, this solution is clearly not in equilibrium. For example, an agent preferring to send fewer messages benefits by not participating. Thus, the agent pretends to fail-stop and we do not (and can not) "punish" it. The protocol thus does not reach equilibrium and, if all agents have that utility, none participate and no solution is reached.

For this reason, in the fail-stop model we require agents to have a *knowledge preference*. The knowledge preference makes sure that agents not only prefer the protocol not to fail, but additionally prefer to know at least part of the protocol's output.

DEFINITION 6.1 (KNOWLEDGE PREFERENCE). *Each agent p prefers knowing some output of the problem (o_q for some agent q) to not knowing any of the output, in the weak sense that it never assigns a higher utility to an outcome where it does not know o_q (for all q) to one in which it knows o_q (for some q).*

However, we are now faced with another problem. An agent can "bail-out" of the knowledge sharing building block. According to the protocol presented in 4.2, it will receive all data in the first round, and thus know the output while saving itself the messages it sends to other agents, thus raising its expected utility. Thus, we also require the input i_p of an agent p in the knowledge sharing building block to be determined randomly, such that we can choose to restart the protocol, and the input will be different. What this means is, we can only execute our protocol for the fail-stop model to solve problems in which the input for the knowledge sharing building block is random. This means that consensus, for example, can not be solved by this protocol, but leader election and renaming can.

The protocols for the agents wake-up and knowledge sharing building blocks are similar to those presented in 3.2 and 4.2, for a synchronous complete network. However, whenever an agent p stops sending messages when it is expected to, we "restart" the protocol without it—executing the agents wake-up or knowledge sharing building block again from the beginning, as if p is not part of the network. Whenever we restart the protocol, all inputs are randomized, as described above. Thus, an agent p not participating in the protocol does not know any output of the protocol, thus ensuring its correct participation due to the knowledge preference (definition 6.1). After at most k restarts, we are guaranteed to have an execution where no processor fail-stopped, and thus the protocol is k-fault-resilient for $k < n$.

While the fail-stop protocol reaches Nash equilibrium, it does not reach even 2-strong equilibrium. Any coalition can benefit by having some of its agents stop sending messages, while others continue to participate in the protocol. When the protocol is finished, the participating agents can send the results to the non-participating agents, thus satisfying the knowledge preference. Thus, it is not possible to reach k-strong equilibrium for $k \geq 2$.

It is worth noting that, while the protocol does not reach k-strong equilibrium for $k \geq 2$, coalitions will not harm the result of the protocol. Even when facing coalitions, the protocol still outputs a legal result.

6.3 Anonymous Networks

Sometimes, distributed computing problems need to be solved in anonymous networks—where agents have no *id*s.

While our protocols use the *id*s of the agents in the wake-up and output computation protocols, this is actually not always necessary. In synchronous networks, for problems which don't require *id*s or distinction among agents (such as the *consensus* problem), we can use the paradigm in an anonymous network.

The protocols are mostly the same for both ring and complete network. The only difference is that any messages with *id*s are sent without *id*s, and the *id* part of the message is ignored.

This allows both the wake-up and the knowledge sharing protocols to execute in much the same way, and then the output computation protocol is executed on the data of the agents $(d_1,\dots,d_n)$ alone, without the *id*s.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we presented building blocks that are useful for many common distributed problems. Additionally, we presented a refined game-theoretic model for a distributed network. Our building blocks are built with this model in mind, such that the protocols for these building blocks are all resilient in the face of rational agents, and coalitions of rational agents to some extent.

One natural question that arises is how this model can be applied to dealing with general graphs in an asynchronous network. General graphs pose some interesting challenges in the presence of rational agents, especially if some agent or group of agents can disconnect parts of the network. In 3.3 we proposed a protocol for solving *wake-up* in general graphs, and in 4.5 we proposed a protocol for solving *knowledge sharing* in general graphs in synchronous networks. We leave open the question of how the other building blocks and algorithms presented in this paper can be adapted to work in an asynchronous general graph network.

An additional network model which would be of interest is anonymous networks. In 6.3 we give some initial thoughts on solving the *wake-up* and *consensus* problems in these types of networks and it would be interesting to see how these can be improved.

We are currently exploring the problem of finding a *minimum spanning tree* in networks with rational agents, and what might be the natural preferences of rational agents in this problem setting. Other fundamental problems of distributed computing might be of interest, finding the natural preferences for each problem and developing new protocols, which are resilient to rational agents, for these known problems.

Additionally, we are looking into devising algorithms for the *fail-stop* model (left as an open question in [4]) which are resilient to both the presence of rational agents and faulty agents which can fail by stopping. An initial analysis of this model can be found in 6.2. Another model of interest is that of *Byzantine* rational agents (also left as an open question in [4]) in which an agent has a preference on the outcome, but does not satisfy the solution preference and prefers to fail the protocol if it does not benefit otherwise.

Acknowledgments

We are grateful to Michal Feldman for helpful discussions and for her enlightening course on Algorithmic Game Theory which has inspired this research, and to Eyal Itkin and the PODC reviewers of this paper for their helpful comments.

8. REFERENCES

[1] I. Abraham, L. Alvisi, and J. Y. Halpern. Distributed computing meets game theory: combining insights from two fields. *SIGACT News*, 42(2):69–76, 2011.

[2] I. Abraham, D. Dolev, R. Gonen, and J. Y. Halpern. Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In *PODC*, pages 53–62, 2006.

[3] I. Abraham, D. Dolev, and J. Y. Halpern. Lower bounds on implementing robust and resilient mediators. In *TCC*, pages 302–319, 2008.

[4] I. Abraham, D. Dolev, and J. Y. Halpern. Distributed protocols for leader election: A game-theoretic perspective. In *DISC*, pages 61–75, 2013.

[5] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth. Bar fault tolerance for cooperative services. In *SOSP*, pages 45–58, 2005.

[6] I. Bárány. Fair distribution protocols or how the players replace fortune. *Math. Oper. Res.*, 17(2):327–340, May 1992.

[7] E. Ben-Porath. Cheap talk in games with incomplete information. *J. Economic Theory*, 108(1):45–71, 2003.

[8] V. Dani, M. Movahedi, Y. Rodriguez, and J. Saia. Scalable rational secret sharing. In *PODC*, pages 187–196, 2011.

[9] Y. Dodis, S. Halevi, and T. Rabin. A cryptographic solution to a game theoretic problem. In *CRYPTO*, pages 112–130, 2000.

[10] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[11] F. Forges. Universal mechanisms. *Econometrica*, 58(6):1341–64, November 1990.

[12] G. N. Frederickson and N. A. Lynch. Electing a leader in a synchronous ring. *J. ACM*, 34(1):98–115, 1987.

[13] G. Fuchsbauer, J. Katz, and D. Naccache. Efficient rational secret sharing in standard communication networks. In *TCC*, pages 419–436, 2010.

[14] E. Gafni. Perspectives on distributed network protocols: A case for building blocks. 1:1.1.1–1.1.5, Oct 1986.

[15] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

[16] S. D. Gordon and J. Katz. Rational secret sharing, revisited. In *SCN*, pages 229–241, 2006.

[17] A. Groce, J. Katz, A. Thiruvengadam, and V. Zikas. Byzantine agreement with a rational adversary. In *ICALP (2)*, pages 561–572, 2012.

[18] J. Y. Halpern and V. Teague. Rational secret sharing and multiparty computation: extended abstract. In *STOC*, pages 623–632, 2004.

[19] Y. Heller. Minority-proof cheap-talk protocol. *Games and Economic Behavior*, 69(2):394–400, 2010.

[20] J. Hendler and D. Subramanian, editors. *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA.* AAAI Press / The MIT Press, 1999.

[21] S. Izmalkov, M. Lepinski, and S. Micali. Perfect implementation. *Games and Economic Behavior*, 71(1):121–140, 2011.

[22] M. Lepinski, S. Micali, C. Peikert, and A. Shelat. Completely fair sfe and coalition-safe cheap talk. In *PODC*, pages 1–10, 2004.

[23] A. Lysyanskaya and N. Triandopoulos. Rationality and adversarial behavior in multi-party computation. In *CRYPTO*, pages 180–197, 2006.

[24] R. McGrew, R. Porter, and Y. Shoham. Towards a general theory of non-cooperative computation. In *TARK*, pages 59–71, 2003.

[25] D. Monderer and M. Tennenholtz. Distributed games: From mechanisms to protocols. In *AAAI/IAAI*, pages 32–37, 1999.

[26] T. Moscibroda, S. Schmid, and R. Wattenhofer. When selfish meets evil: byzantine players in a virus inoculation game. In *PODC*, pages 35–44, 2006.

[27] S. A. Plotkin. Sticky bits and universality of consensus. In P. Rudnicki, editor, *PODC*, pages 159–175. ACM, 1989.

[28] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[29] Y. Shoham and M. Tennenholtz. Non-cooperative computation: Boolean functions with correctness and exclusivity. *Theoretical Computer Science*, 343(1âĂŞ2):97 – 113, 2005.

[30] A. Urbano and J. E. Vila. Computational complexity and communication: Coordination in two-player games. *Econometrica*, 70(5):1893–1927, September 2002.

[31] A. Urbano and J. E. Vila. Computationally restricted unmediated talk under incomplete information. *Economic theory*, 2004.

[32] W. Vickrey. Counterspeculation, auctions and competitve sealed tenders. pages 8–37, 1961.

[33] E. L. Wong, I. Levy, L. Alvisi, A. Clement, and M. Dahlin. Regret freedom isn't free. In *OPODIS*, pages 80–95, 2011.