available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cosrev

Survey

Robust and scalable middleware for selfish-computer systems

Shlomi Dolev[a], Elad M. Schiller[b,*], Paul G. Spirakis[c], Philippas Tsigas[b]

[a] Department of Computer Science, Ben-Gurion University of the Negev, 84105, Israel
[b] Department of Computing Science, Chalmers University of Technology and Göteborg University, Rännvägen 6B Göteborg, S-412 96, Sweden
[c] Research Academic Computer Technology Institute, N. Kazantzakis str., University Campus, 265 00 Rio, Patras, Greece

ARTICLE INFO

Article history:
Received 22 June 2010
Received in revised form
6 September 2010
Accepted 28 September 2010

Keywords:
Game theory
Distributed computing
Self-stabilization
Folk-theorem

ABSTRACT

Distributed algorithm designers often assume that system processes execute the same predefined software. Alternatively, when they do not assume that, designers turn to non-cooperative games and seek an outcome that corresponds to a rough consensus when no coordination is allowed. We argue that both assumptions are inapplicable in many real distributed systems, e.g., the Internet, and propose designing self-stabilizing and Byzantine fault-tolerant distributed game authorities. Once established, the game authority can secure the execution of any complete information game. As a result, we reduce costs that are due to the processes' freedom of choice. Namely, we reduce the price of malice.

"Capitalism is the best economic system in the world because it demands and rewards hard work. It challenges us to be excellent.

But like everything else in life, capitalism can be perverted and exploited. Bad people can find ways to cheat. That's why the federal government oversees the American economy to make sure there is some justice and honesty in pursuit of profit." [Bill O'Reilly, September 18, 2008 in *FOX NEWS*].

1. Introduction

Game theory analyzes social structures of agents that have freedom of choice within a moral code. Society allows freedom and selfishness within this moral code, which is enforced by existing social structures, i.e., legislative, executive, and judicial. Social rules encourage individual profit from which the entire society gains. Distributed computer systems can improve their scalability and robustness by using explicit social structures. We propose using a game authority middleware to enforce the rules of the game, which the honest majority decides upon.

The power of game theory is in predicting the game outcome for specific assumptions. The prediction holds as long as the players cannot tamper with the social structure or change the rules of the game, e.g., the prisoner cannot escape from prison in the classical prisoner dilemma. Therefore, we cannot predict the game outcome without suitable assumptions on failures and honest selfishness.

There are attempts to define various aspects of selfish-computer systems: the selfish MAC Layer that does not back off in [1], the Byzantine Nash equilibrium of a replicated state-machine in [2], and the selfish mechanism for virus inoculation in the presence of malicious agents in [3], to name a

* *Corresponding author.* Fax: +46 31 772 3663.
E-mail addresses: dolev@cs.bgu.ac.il (S. Dolev), elad@chalmers.se (E.M. Schiller), spirakis@cti.gr (P.G. Spirakis), tsigas@chalmers.se (P. Tsigas).

1574-0137/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.cosrev.2010.09.008

few. In fact, [3] discovered that the performance ratio between selfish mechanisms that do and do not have malicious agents (named the *price of malice*, PoM), is as important as the performance ratio between selfish mechanisms and centralistic mechanisms (named, respectively, the *price of anarchy* (PoA) [4,5] and the *price of stability* (PoS) [6]). We study these performance ratios in the presence of game authority implementation, and discover significant improvements.

We argue that when designing distributed selfish-computer systems it is unsuitable to assume that all software layers and components act selfishly. Under this strong assumption, the designer has to consider a complex game among all selfish agents, which has many possible software actions and imprecise cost (utility) in the presence of failures. Moreover, not all games have a predictable outcome; many games have very long stabilization periods, and incomplete information games deteriorate the system efficiency.[1] Consequently, designers cannot predict the outcome without a suitable perspective on the various system aspects.

This paper explains how to implement a game authority so that the middleware can (1) deter deviation (from the moral code), and (2) recover after the occurrence of transient failures.

1.1. *The middleware services*

The game authority facilitates interaction among agents of the (higher) application layer, where users control programs. The middleware implements a social structure that relies on the common moral code of the majority of entities, and overcomes the non-moral Byzantine behavior of minorities.[2] Our social structure follows the principle of power separation. The key middleware services are as follows.

- *The legislative service*, which allows agents to set up the rules of the game in a democratic manner, e.g., robust voting [7], which can facilitate a democratic decision about a preferable outcome for the majority.
- *The judicial service*, which audits the agents' actions and orders the executive service to punish agents following their foul play.
- *The executive service*, which executes actions and manages their associated information: publishing utilities, collecting choice of actions, and announcing the play outcome. Moreover, by order of the judicial service, this service restricts the action of dishonest agents.

Our proposal for a general game authority focuses on the implementation of the judicial service and its potential benefits in the context of distributed systems of selfish computers. This general design of the game authority is presented in order to demonstrate the proof of existence as a general middleware, rather than the most efficient implementation.

[1] Game theory uses this term to describe a game in which individual agents may not be able to predict (precisely) the effect their actions will have on the other agents.

[2] Assuming standard requirements for Byzantine agreement, i.e., more than two-thirds of the processes are (selfish but) honest, and authentication utilizes a Byzantine agreement that needs only a majority. Moreover, the communication graph is not partitioned; e.g., there are $2f+1$ vertex disjoint paths between any two processes, in the presence of at most f Byzantine processes.

2. Case study: game authorities that deter subsystem takeovers

We consider scenarios of joint deviations for which there are simple yet efficient implementations of the game authority. The implementation is based on an equilibrium of strategies that autonomous agents have devised, and all possible joint deviations by a group of at most D deviators. The autonomous agents deter the deviation; if any one of all possible joint deviations happens, then the deviating group will lose payoff, compared to what they would get by obeying the equilibrium strategy.

We consider noncooperative games in which every joint deviation is coordinated by an arbitrary agent—the *coordinator*. The coordinator selects the actions of its *subordinates*, and has no control over the *autonomous* agents. The coordinator and autonomous agents maximize their individual payoffs by a deliberate and optimized selection of actions. The autonomous agents cannot enforce coordinated behavior, and have no a priori knowledge about the identities of the coordinator and its subordinates.

We present a design of autonomous systems that are required to deter subsystem takeovers by implementing the judicial service. Subsystem takeovers can model scenarios in which users abuse their access privileges to remote machines.[3] (Such privileges might be gained by hostile malware.) We show that a simple strategy can deter subsystem takeovers. Moreover, we show that this simple strategy guarantees system recovery after the occurrence of an unexpected combination of deviations.

The judicial service is implemented by deterministic self-stabilizing finite automata. We analyze the complexity of that simple strategy and demonstrate a lower bound that asymptotically matches the costs of our implementation.

2.1. *Self-stabilization*

One of the design enhancements that we consider is the recovery from transient faults. Self-stabilizing systems [10,11] can recover after the occurrence of transient faults. These systems are designed to automatically regain their consistency from any starting state of the automaton. The arbitrary state may be the result of violating the assumptions of the game model or the system settings. The correctness of a self-stabilizing system is demonstrated by considering every sequence of actions that follows the last transient fault and is, therefore, proved assuming an arbitrary starting state of the automaton. We explain how to implement a general game authority that can recover from the occurrence of transient failures. Moreover, in the context of joint deviations, we demonstrate that the proposed strategy for deterring subsystem takeovers is self-stabilizing.

[3] The abuser (i.e., the coordinator) deprives the individual benefit of an arbitrary subset of agents (i.e., the remote machines). We assume that the coordinator does not compensate its subordinates for their losses. Therefore, the notion of subordinates' deviation should be modeled by games that have no side payments or transferable utilities (similar to the definitions in [8]). Hence, neither the sum nor the maximum of the deviators' payoffs should be considered (our approach is different from that in [9]).

3. Document outline

Basic concepts are explained in Section 4. The problem background in explained in Section 5, together with examples. A general game authority middleware is presented in Section 6. A study of a deterministic game authority that deters subsystem takeovers is presented in Section 7, together with a proof sketch of its optimal strategic complexity. Tolerating transient faults is considered in Section 8. The audition of mixed strategies is considered in Section 9. Some of the cost benefits and applications of the game authority are exemplified in Section 10. Related results, open problems, and conclusions appear in Section 11.

4. Preliminaries

Throughout, we follow the definitions and notations of [12]. Throughout, we use N to denote the set of agents, A_i the set of actions, and $\succsim_i$ the preference relation (where $i \in N$ is an agent). We represent single-stage games, G, in their strategic form as $\langle N, A = (A_i), \succsim = (\succsim_i)\rangle$, and in their extensive form as $\langle N, H, \succsim = (\succsim_i)\rangle$. We refer to solution concepts such as the Nash equilibrium, and feasible and enforceable payoff profiles.

4.1. Profiles

We refer to a collection of values of some variable, one for each agent, as a *profile*. Similar to the single element profile notation (i.e., $x = (x_i)_{i\in N}$, (x_{-i}, x_i), and X_{-i} of [12]), we consider profile notation for subsets of elements $s \subseteq N$. We define profiles x_s, x_{-s} to be the list of elements $(x_i)_{i\in s}$, respectively $(x_i)_{i\in N\setminus s}$ for all $s \subseteq N$. Given a list of elements $x_{-s} = (x_i)_{i\in N\setminus s}$ and a profile $x_s = (x_i)_{i\in s}$, we denote by (x_{-s}, x_s) the profile $(x_i)_{i\in N}$. We denote by X_s, X_{-s} the sets $\times_{j\in s} X_j$, $\times_{j\in N\setminus s} X_j$, respectively, where the set of elements is defined as X_i for each $i \in N$.

4.2. Repeated games

Throughout, we consider the game $\Gamma = \langle N, H, \succsim = (\succsim_i)\rangle$, in which the constituent game $G = \langle N, A = (A_i), \succsim = (\succsim_i)\rangle$ is repeated an infinite number of times. We assume that all periods (plays) are synchronous; i.e., all agents make simultaneous moves. Moreover, by the end of each round, all agents have observed the actions taken by all other agents.

4.3. Preference relations for repeated games

A preference relation expresses the desire of the individual for one particular outcome over another. For the constituent game, G, the relation $\succsim_i$ refers to agent i's preferences. Suppose that $\succsim_i$ can be represented by a *payoff/utility function* $u_i : A \to \mathbb{R}$, for which $u_i(a) \geq u_i(b)$ whenever $a \succsim_i b$. We assume that, in Γ, agent i's preference relation $\succsim_i^\star$ is based on a payoff function u_i. Namely, $(a^t) \succsim_i^\star (b^t)$ depends only on the relation between the corresponding sequences $(u_i(a^t))$ and $(u_i(b^t))$.

4.4. The limit of means criterion

The limit of the means criterion [13,14] treats the future as no more important than the present. The sequence v_i^t of real numbers is preferred to the sequence w_i^t if and only if $\lim_{T\to\infty} \Sigma_{t=1}^{T}(v_i^t - w_i^t)/T > 0$.

4.5. Games in extensive form

The *extensive form* of a game describes the game as a decision tree, which is directed from the root downwards. Each node of the tree represents every reachable stage of the game. Starting from the initial node, agents take synchronous (simultaneous) choices of actions. Given any internal node in the tree, each possible action profile leads from that node to another node. A node is said to be terminal if it has no outgoing action profiles.

A history is a sequence of action profiles that corresponds to a directed path from the root of the decision tree. The set of all histories is denoted by H. We note that history $(a^k)_{k=[1,K]} \in H$ is terminal if it is infinite or if there is no $(a^k)^{K+1}$ such that $(a^k)_{k=[1,K+1]} \in H$. The set of terminal histories is denoted Z. Moreover, for each agent $i \in N$, a preference relation $\succsim_i$ is defined on Z. Let h be a history of length k; we denote by (h, a) the history of length $k+1$ consisting of h followed by a. We denote an extensive game with perfect information and synchronous (simultaneous) moves as $\Gamma = \langle N, H, \succsim = (\succsim_i)\rangle$.

4.6. Subgames

In large (or infinite) decision trees, it is useful to isolate parts of the tree in order to establish simpler games. When the initial node of a subgame is reached in a larger game, agents can concentrate on only that subgame; they can ignore the history of the rest of the game.

Let $\Gamma = \langle N, H, \succsim\rangle$ be an extensive game with perfect information and synchronous (simultaneous) moves. Let $H|_h$ be the set of sequences h' of actions for which $(h, h') \in H$. We define $\succsim_i|_h$ as $h' \succsim_i|_h h''$ if, and only if, $(h, h') \succsim_i (h, h'')$. The *subgame* $\Gamma(h)$ of game Γ that follows the history h is the extensive game $\langle N, H|_h, \succsim\rangle$. By defining a new decision tree, $H|_h$ in the subgame, agents can concentrate on only the subgame $\Gamma(h)$; they can ignore the history of the rest of the game.

4.7. Strategies for individuals

Agents protect their benefits by following a long-term plan (or program) of action selection. We call this plan the *(individual) strategy* st_i of agent $i \in N$. We define st_i as a function that assigns an action in A_i to every finite sequence of outcomes in the game $\Gamma = \langle N, H, \succsim = (\succsim_i)\rangle$.

5. Background of the problem

We illustrate basic notions in game theory and elements of the problem at hand using an example (a more detailed tutorial appears in [15]).

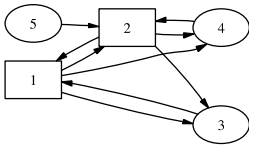
We describe an example of a system with n selfish-computers. We denote by $N = \{1, \ldots, n\}$ the set of agents; each represents a selfish-computer. Every agent decides whether to be a Server or a Client. An agent receives a payoff of -1 for every period in which it decides to be a Server (independently of the other agents' choices). The payoff of a Client depends on the decisions of other agents.

In every period, every Server i, reveals its access list s_i of agents that can access its services. Moreover, every Client j, writes a single agent, say i, in s_j (possibly not matching the access list of agent i). Namely, for Client j, $s_j = \{i\}$ means that j would like to access i as a Server. In case that i indeed chooses to be a Server, then Client j can access i if $j \in s_i$.

Let $G = (V, E)$ be a directed graph that is induced by the access lists. The set V is the set of agents, N. Let $i \in N$ be a Server and $j \in N$ be an agent (that is either a Client or a Server), then $(i, j) \in E$ if, and only if, $i \in s_j \wedge j \in s_i$. See the example of the right.

The Client j receives the payoff of $+1$ (or 0) if the strongly connected component that contains j in the induced graph G includes (respectively, does not include) the majority of agents in N (i.e., more than $\frac{|N|}{2}$).

An example of the induced graph



Above is an example of the induced directed graph. Servers 1 and 2 (boxes) receive the payoff of -1 each. Clients 3 and 4 (ellipses) receive the payoff of $+1$ each. Client 5 is not connected to a strongly connected component that includes a majority of agents. Therefore, 0 is the payoff of Client 5.

Fig. 1 – The shared responsibility n-agent game.

Our example of a system of selfish computers considers the *shared responsibility* service, which is presented in Fig. 1. We model the service as an uncooperative game among n agents. An agent decides whether it would participate as a Server or as a Client. Servers specify their access list; the list restricts the access of other agents (clients or servers). Clients benefit the most whenever they can access a majority of selfish computers via a server that relays the communications.

5.1. *Single-stage games*

The payoff matrix that considers a three-agent instance of the game is presented in Fig. 2. The matrix describes the payoff that agent 1 gets for every possible combination of actions that the agents may take. We note that the payoff of any Server is less than the payoff of any Client (in the single-stage game). Therefore, all agents decide to be clients and thus receive the payoff of 0.[4] This is a *Nash equilibrium*.

5.2. *Infinitely repeated games*

If the single-stage game is repeated infinitely, the agents can benefit from a *sequence of cooperation* steps in which the agents take turns for responsibility. A possible cooperation sequence is presented in Fig. 3. In this sequence of cooperation, every agent is supposed to eventually receive the average payoff of $(|N| - 2)/|N|$.[5] In that sense, if all agents "play along" then $(|N|-2)/|N|$ is a *feasible* payoff. Unfortunately, the selfish agent j might deviate from the sequence of cooperation. Suppose that j knows that all other agents would always allow j to access their services. Then agent j can decide to deviate and be a Client whenever it is the turn of j to be a Server.

5.3. *Punishment*

In uncooperative games, all agents must monitor the actions of agent j, and deter j from deviation by punishment. The punishment should hold j to the minimal payoff value that the punishing agent can enforce. In the shared responsibility game, j receives a minimal enforceable payoff when the punishing agents take a sequence of steps in which (1) they exclude j from their access list, and (2) they "play along" among themselves. A punishing sequence in which the payoff of 0 is enforced on agent 3 is shown in Fig. 4. In that sense, 0 is an *enforceable* payoff.

5.4. *Grim trigger strategies*

One can consider the following strategy. Initially, agent i follows the sequence of cooperation. However, as soon as agent j defects, agent i follows the punishment scheme that *forever* holds j down to its minimal enforceable payoff. In the shared responsibility game, agent j cannot benefit from defecting, because the punishment eventually reduces j's average payoff from $(|N| - 2)/|N|$ to 0. Thus, agent j would prefer to cooperate. Thus, the grim trigger strategy is the Nash equilibrium for infinitely repeated games.

There is a clear disadvantage to the grim trigger strategy: while the agents hold down j to its minimal payoff, their payoff might be reduced as well. The equilibrium will continue to be played forever, even if j defects only once. Thus, the threatened response may seem unreasonable, especially when it is too costly to the punishing agents. In other words, the knowledge that the punishing agents will respond to j's defection by an unrelenting punishment is what keeps j from defecting. However, if j does in fact defect, it may

[4] Starting from any entry of the matrix, consider a sequence of (unilateral) changes to the agents' choice of action. Once every agent was able to change its choice, all agents choose to be clients.

[5] In every $|N|$ periods of the cooperative sequence there are $N-1$ periods in which agent $i \in N$ is a Client (that is served by others) and a single period in which the payoff of agent i is -1.

Agent 1 \ Agent 2	⟨Server, {1}⟩	⟨Server, {3}⟩	⟨Server, {1, 3}⟩	⟨Client, {1}⟩	⟨Client, {3}⟩
⟨Server, {2}⟩	−1	−1	−1	−1	−1
⟨Server, {3}⟩	−1	−1	−1	−1	−1
⟨Server, {2, 3}⟩	−1	−1	−1	−1	−1
⟨Client, {2}⟩	+1	0	+1	0	0
⟨Client, {3}⟩	+1	+1	+1	+1	+1

(a) Agent 3: ⟨Server, {1}⟩ **or** ⟨Server, {1, 2}⟩**.**

Agent 1 \ Agent 2	⟨Server, {1}⟩	⟨Server, {3}⟩	⟨Server, {1, 3}⟩	⟨Client, {1}⟩	⟨Client, {3}⟩
⟨Server, {2}⟩	−1	−1	−1	−1	−1
⟨Server, {3}⟩	−1	−1	−1	−1	−1
⟨Server, {2, 3}⟩	−1	−1	−1	−1	−1
⟨Client, {2}⟩	+1	0	+1	0	0
⟨Client, {3}⟩	0	0	0	0	0

(b) Agent 3: ⟨Server, {2}⟩, ⟨Client, {1}⟩**, or** ⟨Client, {2}⟩**.**

Fig. 2 – The payoff matrices of the shared responsibility game with three agents. The headings of tables, columns and rows are in the form of ⟨*a*, *s*⟩, where *a* is the action, and *s* is the access list of agent 1. The matrix is symmetrical, and thus the payoffs of agents 2 and 3 are, in fact, described as well.

Period	Agent 1	Agent 2	Agent 3
1	⟨S, {2, 3}⟩	⟨C, {1}⟩	⟨C, {1}⟩
2	⟨C, {2}⟩	⟨S, {1, 3}⟩	⟨C, {2}⟩
3	⟨C, {3}⟩	⟨C, {3}⟩	⟨S, {1, 2}⟩
⋮	⋮	⋮	⋮

Fig. 3 – A sequence of cooperation for three agents. The entry format follows that of Fig. 2.

Period	Agent 1	Agent 2
1	⟨S, {2}⟩	⟨C, {1}⟩
2	⟨C, {2}⟩	⟨S, {1}⟩
⋮	⋮	⋮

Fig. 4 – A scheme for punishing agent 3. The entry format follows that of Fig. 2.

no longer be beneficial for the punishers to punish. That is what makes the grim trigger strategy unbelievable.

5.5. *The perfect folk theorem*

In the context of repeated games, the Nash equilibrium can be refined to exclude strategies such as the grim trigger strategy (for example, see the solution concept of subgame perfect equilibrium [16,12]). Roughly speaking, the "refined equilibrium" represents the Nash equilibrium in every "stage" of the repeated game. And thus, if agent *j* defects only once, then the punishing agents would punish *j* merely for a finite period. At the end of the punishment period, all agents return to cooperate.

The perfect Nash folk theorem provides strategies that can deter an agent from deviating unilaterally (see [12], and references therein). A sketch of a strategy is presented in Fig. 5. The strategy is a deterministic and finite automaton that plays the sequence of cooperation as long as there are no deviations. The automaton retaliates to the deviation of an agent with a punishment for a sufficiently long (but finite) period.

6. The middleware

The task of the game authority is to verify that all agents play the game by its rules. This can be achieved by verifying that all agents follow the rules of the game in each play. Namely, before the start of every play, the agents make sure that there is a majority of (honest but selfish) agents that agree on their cost functions and on the result of the previous play (if there was such a play). The agents then play the game and the system publishes the outcome of the play to all agents. Once the outcome is published, the game authority can audit the actions of the agents and punish the agents that did not play by the rules. We organize the middleware of game authority by using the three services (legislative, judicial, and executive).

6.1. *The legislative service*

A key decision that the legislative service makes is about the rules of the game. In more detail, the service is required to guarantee *coherent game settings*; i.e., all honest agents agree on the game $\Gamma = \langle N, (\Pi_i)_{i\in N}, (u_i)_{i\in N}\rangle$. In particular, the service defines the cost (utility) functions $(u_i)_{i\in N}$.

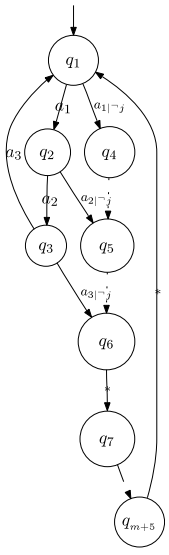
We note that existing manipulation resilience voting algorithms can facilitate these decisions (see [7]). For the sake of simplicity, we assume that the agents have fixed preferences throughout the system run and consider a predefined game Γ, which the society elects before starting the system. We note that a possible design extension can follow the agents' changing preferences and repeatedly reelect the system's game.

6.2. *The judicial service*

The task of the judicial service is to audit the actions that the agents take in every play. Moreover, the judicial service orders the executive service to punish agents that do not play honestly.

In more detail, the service is required to guarantee the following. (1) *Legitimate action choice*. Every honest agent $i \in N$ chooses actions π_i only from its set of applicable actions Π_i. (2) *Private and simultaneous action choice*. The choice of actions of all honest agents is taken simultaneously;, i.e., the system does not reveal the choice of agent $i \in N$ before all have committed to their actions. (3) *Foul plays*. Action $\pi_i \in \Pi_i$ of

On the right side of this figure, we sketch an automaton for agent $i \in \{1,2,3\}$. For brevity, the sketch merely considers a specific deviating agent $j \in \{1,2,3\}$, such that $i \neq j$. †



• $Norm$ **(normal) states** The set $Norm$ includes the states q_1, q_2, and q_3. The $Norm$ states emulate the cooperation sequence of Figure 3. The automaton starts from state q_1 and chooses its action according to the first row of the table in Figure 3. As long as agent j does not deviate, the automaton stays in the $Norm$ states. Namely, if all agents choose their actions according to the sequence of cooperation, then from $Norm$ state q_k the automaton moves to state $q_{k+1 \mod 3}$. ‡

• **The** $Norm$**-**d **(deviated) states** Suppose that agent j deviates while the automaton is in a $Norm$ state, q_k, where $k \in \{1,2,3\}$. In this case, the automaton moves to state q_{k+3}. The set $Norm$-d includes the states q_4 and q_5. The $Norm$-d states emulate the cooperation sequence of Figure 3. However, all of the $Norm$-d states lead to the punishment periods. We use $*$ to denote an arbitrary output of the automaton. Namely, regardless of the choice of the other agents, the automaton moves from state q_4 to q_5 and then to q_6, which is the starting state of the punishment.

• **The** P **(punishing) states** The state that emulates the punishment scheme q_6, q_7, q_{m+5}. A scheme for punishing agent 3 appears in Figure 4. (It is easy to describe similar schemes that punish any other agent.) The automaton starts punishing agent j in state q_6 and chooses its action according to the first row of the table in Figure 4. Regardless of the choice of the other agents, the automaton moves from state q_k to the state $q_{k'}$, where $k \in [6, m+5]$ and $k' = k+1 \mod m+5$.

• **Estimating the constant** m^* We note that the maximum benefit of agent j from a single deviation is 1. Moreover, a complete cycle of the cooperation sequence provides the payoff of $frac13$ (see Figure 3), whereas in every period of punishment j's payoff is 0 (see Figure 4). Therefore, agent j receives a benefit of $1 - fracx3$ whenever j is being punished for x periods. Thus, $m \geq 3$ is any integer that is an integral multiple of 3.

† We note that this sketch can be completed easily by considering any deviating agent. Moreover, we consider all the transitions that the figure on the right does not describe. The state q_6 (double circled) is the state to which all non-described transitions go. ‡ We define $m(\mod g)$ to be the integer q with $1 \leq q \leq \gamma$ satisfying $m = \ell\gamma + q$ for some integer ℓ (e.g., $\gamma(\mod \gamma) = \gamma$).

Fig. 5 – A sketch of a strategy for the shared responsibility game with three agents.

agent $i \in N$ is *foul* if π_i is not i's best response to π_{-i}, where (π'_i, π_{-i}) is the strategy profile of the previous play. The judicial service should instruct the executive service to punish the agents that make foul plays.

6.3. *The implementation*

We start by considering the cases in which agents follow merely pure strategies before we turn to considering mixed strategies (in Section 9).

The algorithm relies on a Byzantine agreement protocol (BAP) and cryptographic primitives such as commitment schemes (see [17]). Moreover, we use a Byzantine common pulse generator (similar to the one of [18]) to synchronize the different services. Moreover, the Byzantine common pulse generator allows the system to repeat a sequence of activating the different instantiations of the Byzantine agreement protocol.

Requirements (1)–(3) can be guaranteed. Upon a pulse, all agents start a new play of the game that is carried out by a sequence of several activations of the Byzantine agreement protocol. The play starts by announcing the outcome, $\pi \in \Pi$, of the previous play (if there was such a play). Here the Byzantine agreement protocol is used to assure that all agents agree on π. Next, every agent chooses its best response, $\pi'_i \in \Pi_i$, to π_{-i}, and uses a commitment scheme in order to announce its commitment on π'_i without revealing π'_i. Again, we use the Byzantine agreement protocol in order to ensure that all agents agree on the set of commitments. Once all commitments are agreed upon, the agents reveal their strategy profile of the play, π'. Moreover, all agents audit the strategy profile of the play and use the Byzantine agreement protocol to agree on the set, $N' \subset N$, of agents that have made a foul play. Lastly, the judicial service orders the executive service to punish N' and to play according to π'.

6.4. *The executive service*

The task of the executive service is to carry out the agents' actions. The service manages the associated information of the actions: announcing the play outcome, publishing the utilities, and collecting the choice of actions. Moreover, by order of the judicial service, this service restricts the action of dishonest agents according to the punishment scheme.

Punishment schemes. Effective punishment is an essential mechanism for reducing the price of malice (PoM [3]) when considering detectable manipulation. However, punishment is useful when there is a price that the dishonest agent is not willing to pay. In other words, a complete Byzantine agent bears any punishment while aiming at maximizing the social

cost. Therefore, it seems that the only effective option is to disconnect Byzantine agents from the network (see [19]). We note that there are punishment schemes based on agent reputation or real money deposits. Other approaches consider more elaborate punishment schemes in which dishonest agents can be deterred (see [20,21]).

We assume that the executive service is trustworthy. This assumption is common in the game theory literature under the name of a *trusted third party*, e.g., the taxation services in mechanism design (see [22]). We note that [23] facilitates such assumptions. In Section 7, we present a miniature game authority and explain how to construct a punishment scheme using deterministic autotmata.

7. Case study: game authorities that deter subsystem takeovers

We consider scenarios of joint deviations for which there are simple yet efficient implementations of the game authority. The implementation is based on an equilibrium of strategies that autonomous agents have devised, and all possible joint deviations by a group of at most D deviators. We show how to deter these joint deviations using a simple strategy that can be implemented by finite automata. We analyze the complexity of that simple strategy and demonstrate a lower bound that asymptotically matches the costs of our implementation.

7.1. Subsystem takeovers

The perfection property is a key feature of the notion of subgames. This property specifies that a strategy profile is a Nash equilibrium in every subgame. Given a game $\Gamma = \langle N, H, \succsim \rangle$, we define the strategy profile $st = (st_i)_{i\in N}$ as a *subgame perfect equilibrium that is t-defendable* from joint deviations of any subordinate group $s \in S(t)$, where $t \in [1, |N|)$ is a constant and $S(t) = \{s : s \in 2^N \setminus \{\emptyset, N\} \wedge |s| \leq t\}$ is the set of all possible subordinate groups.[6]

Joint strategies Let $s \subseteq N$ be any group of agents, $st_s = (st_i)_{i\in s}$ their joint strategies, and $h \in H$ a history of the extensive game $\Gamma = \langle N, H, \succsim \rangle$. We denote by $\Gamma(h)$ the subgame that follows the history h. Moreover, denote by $st_s|_h$ the joint strategies that st_s induces in the subgame $\Gamma(h)$ (i.e., $st_s|_h(h') = st_s(h, h')$ for each $h' \in H|_h$). We denote by O_h the outcome function of $\Gamma(h)$. Namely, $O_h(st_{-s}|_h, st_s|_h)$ is the outcome of the subgame $\Gamma(h)$ when the agents take the strategy profile $(st_{-s}|_h, st_s|_h)$.

Perfect and t-defendable subgame equilibria. Given a number $t \in [1, |N|)$, we say that the subgame $st = (st_i)_{i\in N}$ cannot recover from a joint deviation of a subordinate group $s \in S(t)$ if there is a joint deviation st'_s of the agents in s such that, for any $h \in H\backslash Z$, it holds that, for an arbitrary agent $i_{coord} \in s$ (the coordinator), we have $O_h(st_{-s}|_h, st'_s|_h) \succ_{i_{coord}}|_h O_h(st_{-s}|_h, st_s|_h)$. When the subgame st can recover from any joint deviation of the subordinate groups, $s \in S(t)$, we say that st is a t-defendable equilibrium.

We note that, while the joint deviation st'_s is required to guarantee the benefit of coordinator, there are no guarantees for the benefits of the subordinates. In more detail, there could possibly exist a subordinate agent $j_{subor} \in s \setminus \{i_{coord}\}$ such that $O_h(st_{-s}|_h, st'_s|_h) \prec_{j_{subor}}|_h O_h(st_{-s}|_h, st_s|_h)$. We mention that joint deviations in which agent j_{subor} may exist are not considered by [24–29] (see the discussion in Section 11).

s-enforceable payoff profiles To support a feasible outcome, each subordinate group and its coordinator must be deterred from deviating by being "punished". The concept of enforceable payoff profiles considers a single agent that may deviate (see [12]). We extend that concept to consider the deviation of subordinate groups.

Define the minmax payoff in game Γ of a subordinate group $s \in S$, denoted $v_i|_s$, to be the lowest payoff that the autonomous agents $N' = N \setminus s$ can force upon the coordinator $i \in s$:

$$v_i|_s = \min_{a_{-s}\in A_{-s}} \max_{a_s\in A_s} u_i(a_{-s}, a_s). \quad (1)$$

Given a minmax payoff profile $v_i|_s$, the payoff profile $w|_s$ is called strictly *s-enforceable* if $w_i|_s > v_i|_s$ for all $i \in s$. Denote by $p_{-s} \in A_{-s}$ one of the solutions of the minimization problem on the right-hand side of Eq. (1).

7.2. Folk theorem for games with subsystem takeovers

The folk theorem is a class of proofs which show that every feasible and enforceable profile of payoffs can be achieved by a subgame perfect equilibrium (see [12], and references therein). In this section, we present Lemma 1, which is a folk theorem for games with subsystem takeovers.

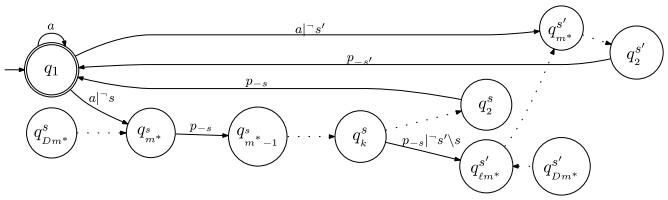
Joint deviations are more complex than unilateral ones. The coordinator of a subordinate group can synchronize its subordinates' deviations and divide them in groups: a group of provoking agents, and a group of "fifth column" agents.[7]

For example, suppose that in the shared responsibility game the subordinate is $s = \{j_1, j_2\}$. The coordinator can synchronize the following deviation: agent j_1 provokes the autonomous agents by not following its duty to be a Server. The autonomous agents retaliate by punishing agent j_1. We note that the deviation of the provoking agents does not reveal the fact that the "fifth column" agent, j_2, is the coordinators' subordinate. Therefore, the autonomous agents expect j_2 to participate in the punishment of its fellow member j_1. Alas, agent j_2 betrays the autonomous agents; while the autonomous group is punishing, agent j_2 deviates from punishing and enters j_1 in its access list. Hence, the synchronized deviation can protect j_1's profit.

Lemma 1 considers the payoff profiles that the autonomous group can guarantee in the presence of subsystem takeovers. A payoff profile w that is strictly s-enforceable $\forall s \in S(D)$ is called strictly *D-defendable*. If $a \in A$ is an outcome of Γ for which $u(a)$ is strictly D-defendable in Γ, then we refer to a as a *strictly D-defendable outcome of* Γ.

[6] We do not consider the case when $s = N$, because it refers to a system that is controlled by a single agent.

[7] Fifth column [30]: Clandestine group of subversive agents who attempt to undermine a nation's solidarity by any means.



On the right, an automaton for agent $i \in N$ is sketched. For brevity, the automaton considers: a single subordinate group, $s' \in S(D)$, † and a sequence of cooperation that consists of the repetition of a single outcome, i.e., $\gamma = 1$. ‡

• $Norm$ **(normal) states** The automaton starts from $Norm$ state, q_1, and chooses its action according to the profile of actions, $a = (a_i)_{i\in N}$, that guarantees the strictly D-defendable outcome, w. As long as no subordinate group deviates, the automaton stays in the $Norm$ state q_1. In the case where the subordinate group s deviates from a, then the automaton moves to state $q^s_{m^*}$ from which the punishment of s begins.

• P **(punishing) states** For each subordinate group, $d \in S(D)$, the set P includes the punishing states $\{q^d_k\}_{d\in S(D),k\in[2,Dm^*]}$. The punishment of the subordinate group $s \in S(D)$ uses a payoff profile, p_{-s} that is s-enforceable. § As long as the agents in $N' = N \setminus \{s\}$ do not deviate from the punishment scheme p_{-s}, the automaton moves from q^s_k to $q^d_{k'}$, where $k \in [2, Dm^*]$ and $k' = k + 1 \mod m^*$. ‡ Suppose that while the automaton is in state q^s_k, the agents $s' \setminus s$ betray the autonomous group N' and deviate from the punishment scheme p_{-s}. In this case, the automaton moves from state q^s_k to state $q^{s'}_{\ell m^*}$, where s' is the set of exposed deviators, and $\ell = |s'|$.

We note that: † This sketch can be easily completed by considering any subordinate group. Moreover, we consider all the transitions that the figure above does not describe. The state q_1 (double circled) is the state to which all such non-described transitions go. ‡ The more general case appears in Figure 5. § Use $m(\mod g)$ as defined in Figure 5.

Fig. 6 – A strategy sketch for a repeated game with n agents and the limit of means criterion.

Lemma 1. *Let Γ be an infinitely repeated game of $G = \langle N, (A_i), (u_i)\rangle$ with the limit of means criterion. Every feasible and strictly D-defendable payoff profile of Γ has a subgame perfect equilibrium payoff profile that is D-defendable.*

Proof outline. The strategy profile of the automata, $(atm_i)_{i\in N}$, is illustrated in Fig. 6. We use the constant m^* that we now turn to estimate. After the first deviation, the sequence of punishment starts, during which the coordinator might increase its benefit for ϱ periods of betrayal, where $0 \le \varrho \le |s'\setminus s|$. However, a suffix of the punishment sequence is guaranteed to include $|s'|m^*$ periods in which there are no further betrayals and the automaton plays $v_i|_{s'}$. Thus, the coordinator's potential benefit is $(\gamma + \varrho)g^*$, where g^* is the maximal amount that any coordinator $j \in N$ can gain when any subordinate group $s \in S(D)$ deviates from any action profile in G. (Namely, g^* is the maximum of $u_j(a_{-s}, a'_s) - u_j(a)$ over all $j \in N, s \in S(D), a'_s \in A_s$ and $a \in A$. Moreover, we assume that g^* is given.)

The coordinator cannot increase its benefit during the punishment suffix, which has no further betrayals. We explain how to choose a large enough m^* so that the punishment is effective. The alternative payoff of the coordinator is at least the sum of $w_j - v_j|_{s'}$ taken over all $|s'|m^*$ periods of the punishment suffix. Since w is strictly D-defendable, and $s \in S(D)$, $w|_s$ is s-enforceable, and $w_j|_s > v_j|_s$ (recall $v_j|_s$ from Eq. (1)). Therefore, there exists an integer $m^* \ge 1$ that is an integral multiple of γ such that, for all $j \in N$ and $s' \in S(D)$,

$$g^*(\gamma + D - 1) < m^*(w_j - v_j|_{s'}). \tag{2}$$

The proof specifies the strategy described above. Moreover, the proof verifies that, in the case where there are no deviations, the automaton follows the sequence of cooperation. In addition, for any non-empty subordinate group that deviates, the automaton follows a finite and effective sequence of punishment. □

We note that existing work on joint deviation in repeated games, such as [25,27], does not bound the costs that are related to the strategy complexity. The finite automaton that is considered by Lemma 1 allows us to present such bounds (see Section 11).

7.3. *Complexity issues of games with subsystem takeovers*

Computational game theory has several ways of measuring complexity (see [31]). The two most related to games with subordinates' deviations are as follows.

Costs of finding strategies. The computational complexity of a game model describes the asymptotic difficulty of finding a solution as the game grows arbitrarily. We give the *shared responsibility game* as an example for which it is possible to efficiently find strategies that deter subordinates' deviations. Unfortunately, this is not the general case; finding strategies that deter joint deviations is at least as hard as finding a Nash equilibrium, which is known to also be computationally intractable for infinitely repeated games; see [32,33].

Costs of implementing strategies. What is the minimal amount of memory required for implementing a given strategy? Kalai and Stanford [34] answer this question in the context of finite-state machines, and show that it is the size of the smallest automaton that can implement the strategy. The difficulty that Lemma 2 raises is that selfish computers that try to deter subordinates' deviations may exhaust their resources.

Lemma 2. *The complexity of a strategy that deters subordinates' deviations is in* $\Theta(D\binom{n}{D})$*, where* n *is the number of agents, and* D *is an upper bound on the size of the subordinate group.*[8]

Proof outline. A strategy that deters subordinates' deviations is presented in Section 7.2. The automaton that implements these strategies requires $O(D\binom{n}{D})$ states. The lower bound part of this lemma is demonstrated by considering every subordinate group, $s \in S(D)$, and all the possible sequences of deviations. There are at least $\binom{n}{D} - 1$ subordinate groups. The proof verifies the existence of at least D different periods in which the deviators may deviate before all of them deviate. Only after the last deviation can the strategy complete the punishment of the subordinate. Therefore, there are at least $D(\binom{n}{D} - 1)$ different subgames in which a subordinate group deviates. Thus, by [34], the strategy complexity is in $\Theta(D\binom{n}{D})$. □

8. Self(ish)-stabilizing

We now shift focus to show an implementation of the game authority that can recover from transient failures and periods during which the agents act upon short-lived myopic logic. We name the combination of these two properties *self(ish)-stabilization*. We start by considering a general game authority before turning to the example presented in Section 7 and demonstrate the ability to tolerate transient faults.

8.1. *Self-stabilizing judicial services*

The game authority can be coded in the form of a do forever loop that is supported by a self-stabilizing Byzantine pulse synchronization algorithm (that is similar to [18]). Thus, by showing that every service is self-stabilizing, we show that the entire middleware is self(ish)-stabilizing.

It is easy to see that the legislative service is stateless and therefore self-stabilizing. We note that not every judicial service is self-stabilizing, because the Byzantine agreement protocol (BAP) has an internal state (e.g., epoch numbers). We demonstrate that the judicial service can be self-stabilizing by showing the existence of a self-stabilizing Byzantine agreement algorithm. We remark that the executive service is application dependent, and therefore should be made self-stabilizing on a case basis.

The self-stabilizing Byzantine agreement algorithm is a composition of two distributed algorithms. We use the self-stabilizing Byzantine clock synchronization algorithm of [18]. Whenever the clock value reaches the value 1, the self-stabilizing Byzantine agreement algorithm invokes the Byzantine agreement protocol (BAP) (e.g., [35,36]). We take the clock size, $\log M$, to be large enough to allow exactly one Byzantine agreement, where M is the number of clock values.

Theorem 1 ([37,38,20])**.** *The algorithm described above is a self-stabilizing Byzantine agreement protocol.*

[8] The lower bound holds when considering t-strong [24] or t-resilient [27] equilibria, because the t-defendable property implies the t-strong and t-resilient properties (see Section 11).

We call the self-stabilizing Byzantine agreement described above SSBA.

The settings of a distributed system We formally describe self-stabilizing distributed systems that stabilize in the presence of Byzantine faults before we turn to demonstrating the proof.

The system consists of a set of computing and communicating entities, which we call *processors*. We denote the set of processors by $\mathcal{P}$, where $|\mathcal{P}|$ is finite. The graph $G(\mathcal{V}, \mathcal{E})$ denotes the communication graph of the system, where $\mathcal{V}$ is the set of processors, and where there is an edge in $\mathcal{E}$ between every pair of processors p_i and p_j that can directly communicate. All such p_i and p_j are called *neighbors*. We assume that every processor has a unique identifier.

In the proposed system, every processor emulates an agent, which is a program that encodes an agent in a strategic game. The program of a processor p_i consists of a sequence of *steps*. For ease of description, we assume a synchronous model in which all processors execute steps automatically and simultaneously. The *state* s_i of a processor p_i consists of the values of all processor variables (including its control variables such as the value of its program counter).

A *common pulse* triggers each step of p_i. The step starts sending messages to neighboring processors, receiving all messages sent by the neighbors, and changing its state accordingly.

As mentioned before, a processor is *Byzantine* if it does not follow its program, i.e., does not execute the *agent* or does not participate correctly in implementing the game authority middleware. We assume standard requirements for Byzantine protocols, i.e., more than two-thirds of the processes are (selfish but) honest, and authentication utilizes a Byzantine agreement that needs only a majority. Moreover, the communication graph is not partitioned; i.e., there are $2f + 1$ vertex disjoint paths between any two processes, in the presence of at most f Byzantine processes.

We describe the global state of the system, the system *configuration*, by the vector of the state of the processors $(s_1, s_2, \ldots, s_n)$, where each s_i is the state of processor p_i. We describe the system configuration in the instance in which the pulse is triggered, when there are no messages in transit. We define an *execution* $E = c_0, c_1, \ldots$ as a sequence of system configurations c_i, such that each configuration c_{i+1} (except the initial configuration c_0) is reached from the preceding configuration c_i by an execution of steps by all the processors.

The *task* τ of a distributed system is defined by a set of executions LE called *legal* executions. For example, task τ may be defined by the correct game behavior of agents which is carried out according to the rules of the game, and in which the set of *enabled agents* present rational behavior. By enabled agents, we mean that rational agents may decide to punish mischievous agents and disable their foul plays.

Self-stabilizing distributed systems [10,11] can recover after the occurrence of transient faults. The system is designed to automatically regain its consistency from any starting configuration. The arbitrary configuration may be the result of unexpected faults caused by the environment and/or mischievous agents' behavior.

The correctness of a self-stabilizing system is demonstrated by considering every execution that follows the last

transient fault and is, therefore, proved by assuming an arbitrary starting configuration. The system should exhibit the desired behavior (of task τ) in an infinitely long period after a finite convergence period.

A configuration c is *safe* with regard to a task τ and to the distributed system if every nice execution that starts from c belongs to LE. We say that the algorithm satisfies its task τ when its execution is in LE (the property of closure). An algorithm is self-stabilizing if, starting from an arbitrary configuration, it reaches a safe configuration within a finite convergence period (the property of convergence).

The proof of Theorem 1. We define the set of legal executions, with respect to the task of the self-stabilizing Byzantine agreement protocol (BAP), as the set of executions, LE, in which the Byzantine agreement protocol (BAP) properties hold (i.e., termination, validity, and agreement). Let E be an execution, and $c \in E$ be a configuration, such that (1) all the clock values are 1, and (2) the Byzantine agreement protocol (BAP) is at its starting configuration (e.g., identical epoch and round numbers).

Lemma 3 (*Convergence*)**.** *Starting from an arbitrary configuration, we reach a safe configuration within $O(n^{(n-f)})$ clock pulses.*

Proof. Starting from an arbitrary configuration, within an expected $O(n^{(n-f)})$ clock pulses, a configuration c is reached in which all clock values are 1 (see [18]). In the atomic step that immediately follows c, all processes invoke the Byzantine agreement protocol (BAP) before changing their clock value. Hence, c is safe. □

Lemma 4 (*Closure*)**.** *Let E be an execution that starts in a safe configuration. Then $E \in LE$.*

Proof. Starting from a safe configuration, there is a period of M pulses in which no process assigns 1 to its clock. During this period, the Byzantine agreement protocol (BAP) is executed for a long enough time that allows exactly one Byzantine agreement. Moreover, at the end of this period, all processes assign 1 to their clocks. Thus, there is an infinite sequence of such periods in which the Byzantine agreement protocol (BAP) reaches Byzantine agreements. □

Lemmas 3 and 4 imply Theorem 1.

8.2. *Case study: tolerating transient faults in the presence of subsystem takeovers*

When designing a distributed system of selfish computers, it is unsuitable to assume that failures never occur. Most of the existing literature on repeated games considers agents that have identical views on the history of actions. In practice, it is unlikely that all selfish computers never fail to observe an action in an infinite system run. Once a single selfish computer misinterprets an action, the outcome of the game cannot be predicted by the current theory.

Transient faults are regarded as faults that temporarily violate the assumptions made by the system designer about the game model and the system settings. For example, the system designer may assume the existence of a constant upper bound, D, on the size of the subordinate group. In this case, a transient fault could be a joint deviation of more than D agents. (Recall that Lemma 2 implies a possible failure in allocating sufficient memory as D grows.)

We show that the automata that are considered in Section 7 can automatically regain their consistency from any starting state of the automata. The arbitrary state may be the result of violating the assumptions about the game model or the system settings, i.e., transient faults. Lemma 5 extends Lemma 1 by showing that the automata can be made to recover from transient faults. Thus, there are self(ish)-stabilizing systems of that deter subordinates' deviations.

Lemma 5. *Let Γ be an infinitely repeated game of G under the limit of means criterion. Then, there are self(ish)-stabilizing automata that implement subgame perfect equilibria that are D-defendable in Γ.*

We consider two proofs of Lemma 5: a simple one that uses known techniques of clock (state) synchronization and a more general one. Dolev [11] presents clock synchronization algorithms that within a finite number of steps after the last transient fault synchronize the clock values. One can view the clock values as the states of the automata. Namely, requiring identical state values of the autonomous automata is the same as requiring the same time values in all clocks. This simple proof assumes that all states are distinguishable, e.g., all automata send their state number in every period. This assumption implies that during the punishment period all automata have to communicate and that there is no "punishment by silence". Sometime this assumption is too restrictive. For example, in some applications it might be required that the autonomous agents do not communicate or interact during the sequence of punishment. In this case, no communication implies that the states are indistinguishable. Therefore, we also consider a more general proof in which only the first state of any punishment sequence is distinguishable. For the sake of brevity, we present here the proof outline; the complete and general proof of Lemma 5 can be found in [20,21].

Proof outline. Let us construct an additional sequence of punishment using the states $P(\emptyset, k)$, where $k \in [1, Dm^*]$. In these states, the automaton plays according to a D-defendable Nash equilibrium. Without loss of generality, suppose that the states $P(d, Dm^*) : d \in S(D) \cup \{\emptyset\}$ are distinguishable from all the other states.[9]

Self-stabilization requires the properties of closure and convergence that can be verified by a variant function (see [11]). Every step of the automata monotonically decreases the value of the variant function until it reaches zero, which implies the end of the stabilization period. In order to define the potential function, we represent the automata as directed graphs.

• *The automaton as a graph.* The graph $\Phi = (V, E)$ has the set of states as the set of vertices, V. Moreover, the transition function, $\tau()$, induces directed edges of E, i.e., $(v, u) \in E \leftrightarrow \exists a \in A : \tau(v, a) = u$.

[9] This assumption can be implemented, for example, by letting all selfish computers broadcast the indices of their current states at the end of every period. We note that any additional costs that the broadcast induces can be compensated by selecting a larger m^*.

• *The variant function.* We define LongDistance(q_j) to be the length of the maximal simple and directed path in graph Φ, from state q_j to state $P(\emptyset, Dm^*)$. (A simple and directed path is one without any directed cycles.) We define the variant function $\Phi()$ to be 0 if all automata $(atm_i)_{i\in N'}$ are in the same state, where $s \in S(D)$ is the subordinate group and $N' = N \setminus s$. For all other cases, we define $\Phi(c) = \max_{j\in N'}$ LongDistance(q_j). It can be observed in Fig. 6 that $0 \leq \Phi(c) \leq (\gamma + m^*D^2)$.

• *Closure.* Suppose that $\Phi() = 0$, which means that automata $(atm_i)_{i\in N'}$ are in the same state. Since the automata are deterministic they all move to the same state, and $\Phi() = 0$ holds.

• *Convergence.* The proof verifies this property by showing that all steps decrease the value of $\Phi()$. Let us construct the automaton such that all undefined transitions move to state $P(\emptyset, Dm^*)$. In particular, the automaton moves to state $P(\emptyset, Dm^*)$ when more than D deviators are observed. The proof verifies that, if automaton $atm_i : i \in N'$ is in any punishing state, then all automata $(atm_i)_{i\in N'}$ move to state $P(\emptyset, Dm^*)$, and stay in $P(\emptyset, Dm^*)$, until all automata $(atm_i)_{i\in N'}$ move to state $P(\emptyset, Dm^*)$. □

We follow the spirit of Kalai and Stanford [34] and define the strategy complexity of a self-stabilizing strategy, st_i, as the number of distinct strategies induced by st_i in all possible subgames that start *after* stabilization. In that sense, Lemma 5 shows a self-stabilizing strategy that has asymptotically the same complexity as the non-self-stabilizing one (presented in Lemma 1).

9. Auditing mixed strategies

So far, we have explained that the game authority can audit agents that use only pure strategies. In this section, we consider mixed strategies. We start by exploring scenarios in which mixed strategies raise troubling questions and then we propose a solution. We explore these scenarios by looking into the well-known game of *matching pennies*.

Matching pennies is a game for two agents, A and B, in which each agent has two strategies: heads and tails. The agents choose their actions secretly and then reveal their choices simultaneously. If the pennies match (both heads or both tails), agent A receives 1 from agent B. If the pennies do not match (one heads and one tails), agent B receives 1 from agent A. This game has no pure (deterministic) nash equilibrium, but there is a unique Nash equilibrium in mixed strategies: each agent chooses heads or tails with equal probability. This way each agent makes the other indifferent between choosing heads or tails, so neither agent has an incentive to try a different strategy.

9.1. Hidden manipulative strategies

Suppose that agent B has a hidden manipulation for the heads strategy; the manipulation has no effect on the game whenever the pennies match or when B plays tails. However, whenever the pennies do not match and B chooses the heads strategy with manipulation, then A pays 9 to B. The new game is presented in Fig. 7. Clearly, since agent B knows that agent

A/B	Heads	Tails	Manipulate
Heads	$(+1,-1)$	$(-1,+1)$	$(+1,-1)$
Tails	$(-1,+1)$	$(+1,-1)$	$(-9,+9)$

Fig. 7 – Matching pennies with a hidden manipulation strategy.

A plays each of the two strategies with probability 1/2, then B plays the manipulated heads strategy with probability 1. The manipulation by B is successful, because B is able to increase its expected profit from 0 to 4, while A has decreased its expected profit from 0 to −4.

9.2. Validating random choices

The above hidden manipulative strategies can be extended to a more general form. Namely, we consider dishonest agents that deviate from an equilibrium by selecting actions that, according to the game model, should decrease their benefit, i.e., are not the best response. The challenge is in verifying that a sequence of random choices follows a distribution of a credible mixed strategy.

9.3. The solution

In every round of the game, the agents use a private pseudo-random generator for privately selecting actions according to the strategy profile of the round. We ensure that an action is indeed random by taking Blum's approach [17]. Namely, the agents commit to their strategy profile using a cryptographic commitment scheme. Before the play and after all agents have received all commitments, the agents publicly reveal their private action selection. Therefore, just before the next play starts, the honest majority can detect any foul play using a Byzantine agreement protocol (BAP).

We note that, in our implementation of the judicial service, we take the simplest auditing approach; the agents audit each other's actions in every round of the game. A possible extension can consider any bounded number of rounds. Here, for the sake of efficiency, the agents commit to the private seed that they use for their pseudo-random generator; they reveal their seed at the end of the sequence of rounds and then audit each other's actions. In practice, one may consider several auditing techniques (see [39]) and decide to verify the honest selfishness of agents that raise suspicion among the honest majority.

9.4. Benefit: reduced price of malice

By auditing the choices of the agents the game authority clearly reduces the ability of dishonest agents to manipulate.

10. Application: repeated resource allocation

The agent society is composed of individuals with different goals and wishes regarding the preferable outcome. The opportunity to select a game that the honest majority prefers shifts the perspective of the distributed system designer. Transitionally, the designer should aim at modeling the

system precisely, and should consider all possible failures. Using the proposed middleware of game authority, the distributed system designer can virtually set the rules of the game, because the game authority can guarantee that these rules are followed.

In this new situation, there is a need to estimate the eventual performance criteria of repeated games, rather than the cost in a particular play, e.g., the price of anarchy (PoA) [4,5] that considers the worst Nash equilibrium and the price of stability (PoS [6]) that considers the best Nash equilibrium.

We consider an example of a *repeated resource allocation* (RRA) scenario in which a consortium of Internet companies shares licenses for advertisement clips on video Web sites. We note that the unpredictable loads on the hosts cause service availability issues. There are many complex ways to model this scenario. One simple way is as follows. In every play, each agent places a (single-unit) demand for a resource. We assume that at the end of every play all agents know the load that exists on the resources. The load of a resource determines the time it takes to service the demands for this resource. Every agent wishes to minimize the time it takes to service its demands for the resources that it chooses. We assume that the number of plays is *unknown*, i.e., every play could be the last one. Thus, selfish agents choose resources in an ad hoc manner. In other words, the choices are according to a repeated Nash equilibrium: independent in every round.

Corollary 1 claims that the simplest game of RRA is optimal. Therefore, it could be that the consortium majority prefers backlog size as the host's only selection criterion (and rejects criteria such as video content and attempts of synchronized advertisement). In this case, the game authority can support the agent society's preferences, whereas in the case of more complex selection criteria, the game outcome may be hard to predict, or the multi-round anarchy cost might be higher. The multi-round anarchy cost is defined as the (eventually) expected ratio between the cost of the worst-case equilibrium and that of the optimal (centralistic) solution.

Corollary 1 (*Supervised RRA*). *A game authority that supervises the RRA game can guarantee an $O(1)$ multi-round anarchy cost.*

We now turn to demonstrate Corollary 1 and show that the repeated resource allocation (RRA) game has an asymptotically optimal cost whenever the game authority assures that all agents are honestly selfish. Let B (bins) be a set of resources ($|B| = b > 1$), $\ell_a(k)$ the load of $a \in B$ (after k rounds), $M(k) = \max\{\ell_a(k)\}_{a\in B}$, $m(k) = \min\{\ell_a(k)\}_{a\in B}$, and $EM(k)$ the expectation of $M(k)$ after a sequence $S = \pi(0), \pi(1), \ldots$, where $\pi(k) \in \Pi$ is a result of a Nash equilibrium way to select resources on the round k. The k-round cost of anarchy is the ratio $R(k) = SC(k)/OPT(k)$, where $SC(k)$ is the worst-case $EM(k)$ over all possible sequences S, and $OPT(k)$ is the optimal solution. As for the repeated resource allocation, $\Sigma_{a\in B}\ell_a(k) = nk$, $OPT(k) = \lfloor nk \rfloor/b + 1$, and $R(k) \leq SC(k)b/nk$. Lastly, let $R = \lim_{k\to\infty} R(k)$ be the *asymptotic cost of anarchy* (if it exists, $R = \limsup_{k\to\infty} R(k)$).

The initial zero demand for all resources is assumed (when considering the asymptotic behavior of the repeated resource allocation service). Therefore, by information completeness, the loads on every resource are known after k plays and a repeated Nash equilibrium is being formed throughout the play.

Theorem 2 (*Replaces Corollary 1*). *When the game authority supervises the repeated resource allocation service, it holds that $\forall k : R(k) \leq 1 + 2b/k$, and $R = 1$.*

Proof. For a particular play k, define x_i^a to be the probability that agent i places its demand on resource $a \in B$ ($\Sigma_{a\in B} x_i^a = 1$). Suppose that agent i places its demand on resource a. The expected load on resource $a \in B$ is $\lambda_i^a = 1 + \Sigma_{i\neq j} x_j^a + \ell_a(k)$. (Since agents make independent choices, we use the subscript notation to represent agent i's perspective.)

Lemma 6. $\Delta(k) = M(k) - \ell_a(k) \leq 2n - 1$ $(\forall a \in B)$.

Proof. Suppose, in contradiction, that the assertion of the lemma does not hold in round k. Let $k' \leq k$ be the first round at which $\Delta(k) > 2n - 1$, and, without the loss of generality, assume that $\Delta(k) > 2n - 1$ in any round between k' and k. At round $k + 1$, we denote $a' \in B$ to be a resource with maximal load, and $i_0 \in N$ to be an agent with $x_{i_0}^a, x_{i_0}^{a'} > 0$ $(a, a' \in B)$. The Nash equilibrium selection requires that $\lambda_{i_0}^a = \lambda_{i_0}^{a'}$, which implies that $1 + \Sigma_{i\neq i_0} x_i^a + \ell_a(k) = 1 + \Sigma_{i\neq i_0} x_i^{a'} + \ell_{a'}(k)$; $1 + \Sigma_{i\neq i_0} x_i^a = 1 + \Sigma_{i\neq i_0} x_i^{a'} + \Delta(k)$, and $\Delta(k) = \Sigma_{i\neq i_0}(x_i^a - x_i^{a'})$. The lemma is established because $\Sigma_{i\neq i_0}(x_i^a - x_i^{a'}) \leq n-1$ contradicts $n < \Delta(k)$. □

Thus, no agent supports both a and a' in her/his play, i.e., if a has any support, then all agents place their demand solely on a. By Lemma 6, $(b-1)\Delta(k) \leq (b-1)(2n-1)$, and by the definition of $\Delta(k)$, we get $(b-1)M(k) - \Sigma_{a\neq a'}\ell_a(k) \leq (b-1)(2n-1)$ (denoted as Eq_1). We also know that $\Sigma_{a\in B}\ell_a(k) = M(k) + \Sigma_{a\neq a'}\ell_a(k)$ and $M(k) + \Sigma_{a\neq a'}\ell_a(k) = nk$ (denoted as Eq_2). By adding equations Eq_1 and Eq_2, we get $bM(k) \leq nk + (b-1)(2n-1)$, which implies that $M(k) \leq (nk + (b-1)(2n-1))/b$. Since $OPT(k) \leq nk/b$, $R(k) = EM(K)/OPT(k) \leq (nk + (b-1)(2n-1))/nk = 1 + (b-1)(2n-1)/nk \leq 1 + 2b/k$. □

11. Discussion

The solution concept of strong Nash equilibrium [24–26] aims at deterring a coalition of deviators that may all benefit from their joint deviations. Moreover, the solution concept of a resilient Nash equilibrium [27] aims at deterring a coalition of deviators that may increase the payoff of at least one deviator, but is committed to keeping the benefits of all the other deviators. We mention that coalition-proof strategies consider agents that can communicate prior to play, but cannot reach binding agreements (see [28,29]). In the context of repeated games, the collective dynamic consistency (of coalition-proof strategies) considers equilibria for which agents do not wish to jointly renegotiate throughout the course of the game (see [40]). This work considers harder deviations, in which the coordinator benefits and the subordinates may lose payoff. Therefore, our strategy can deter the deviations that are mentioned above.

Self(ish)-stabilization [41–46] was earlier considered for single stage games. The game authority [41,42,45] verifies that no agent violates the game model of the stage game. Spanning trees among selfish parties are studied by [43,44].

Reactive systems that are inspired by game theory appear in [46].

The research path of BAR fault tolerance systems [19,47] studies cooperative services that span multiple administrative domains, such as a backup service [48], a peer-to-peer data streaming application [49], and Synchronous Terminating Reliable Broadcast [50]. BAR fault tolerance systems consider a minority of Byzantine computers that deviate arbitrarily and a single selfish deviator (out of the set of all selfish computers). Between every pair of selfish computers, the grim trigger strategy is used, which suffers primarily from the inability to recover from transient faults (see [51]). In other words, an agent that (involuntarily) deviates once is punished forever. We consider the more realistic model of infinitely repeated games, in which any group of D agents can always deviate. We offer a more sensible solution: the system punishes the deviators for a bounded period after the last betrayal. This type of punishment better fits the cases of non-deliberate misbehavior of selfish computers and transient faults.

A k-resilient Nash equilibrium [27] is a joint strategy for which no member of a coalition C of size up to k can do better. It is shown that such a k-resilient Nash equilibrium of a single-stage game exists for secret sharing and multiparty computation. We consider a different problem in a more realistic game model.

11.1. *Critical review of the relevant literature*

• *Why is the model of repeated games considered?* In distributed systems, single-stage games reflect tasks that are less common compared to settings of infinitely repeated games. Repeated games are best known for their ability to model cooperation among selfish agents. For example, the perfect folk theorem (see [13,14]) presents a strategy in which its payoff in infinitely repeated games is better than the payoff of the single-stage Nash equilibrium. The theorem can explain periods of war and peace among selfish agents that can deviate unilaterally. For this reason, the model of repeated games is regarded as more realistic than the model of single stage games.

• *Why do we use DFA and not Turing machines?* Deterministic and finite automata (DFA) can implement the strategies of the folk theorem (see [12], and references therein). The literature considers strategies that can be implemented by deterministic and finite automata as a separate and "simpler" class of strategies (see [52,53]). In fact, there is evidence that this class is strictly weaker than the class of strategies that Turing machines can implement (see the survey [54], and references therein).

We note that some of the existing results (such as [27,23]) consider poly-time (or probabilistic) Turing machine, which can be emulated by finite (or probabilistic) automata. The reduction increases the number of states that the automaton uses by a non-polynomial factor. We present simpler implementations.

• *Why do we not consider coalitions in which all agents are faulty?*[10] Eliaz [55] and later [50,48,49,19,47] considered coalitions in which all of the deviators may possibly be faulty. The inherent difficulty is that no punishment deters a coalition in which all agents are Byzantine. In this case, the literature proposes using either strategies for single-stage games or grim trigger strategies.

In distributed systems, single-stage games reflect tasks that are less common compared to settings of infinitely repeated games. Lack of credibility is the Achilles heel of grim trigger strategies; deviating agents are forever punished due to mistakes that are made at the moment of weakness. Furthermore, the system cannot recover from transient faults in these settings.

We assume that a single rational agent controls a set of deviators and propose a perfect strategy that deters the deviators with a finite period of punishment. Thus, in the context of self-stabilization it is essential to require that not all deviators are faulty.

• *Why do we not consider coalitions in which all agents are rational?* A coalition in which all deviators are rational is required to promote (or at least protect) the benefit of its members (see [24–29], and references therein). This is not the case with subsystem takeovers; here the coordinator dictates the actions of its subordinates and ignores their benefits. Therefore, by assuming that not all deviators are rational, it is "harder" for the autonomous (non-deviating yet selfish) agents to protect their benefits, because the requirements regarding joint deviations are explicitly less restrictive.

We do not claim to be the first to consider strategies for protecting the benefit of the autonomous (non-deviating yet selfish) agents (see [27,24]). However, we present strategies for protecting the benefit of autonomous agents in the presence of deviating coalitions that do not protect the social benefit of all deviators. It is important to see that previous work [27,24] considered strategies for protecting the benefit of autonomous agents in the presence of deviating coalitions that indeed protect the social benefit of all deviators.

In more detail, footnote 1 of [27] states: "Of course, in a more refined model of the game ... everyone in the coalition would do better". Namely, Abraham et al. [27] implicitly do not consider deviations in which the social benefit of the coalition is worsening. However, the social benefit of the coalition can be worsening in the case of subsystem takeover, because only the coordinator is required to benefit, whereas the subordinates can be made to take their least preferable actions.

• *Are there strategies for coping with more than one rational deviator within the subordinate group?* Our definition of subsystem takeovers has a straightforward extension that considers collations of k rational agents that collectively and totally control t subordinate agents. For example, rational agent 1 controls subordinating agents 1_1, 2_1 and 3_1, and rational agent 2 controls subordinating agents 1_2, 2_2 and 3_2. Another example is when agents 1 and 2 reach an agreement about the behavior of their subordinates. Our strategies can deter such deviations because we consider an arbitrary coordinator and punish the entire subordinate group.

[10] One may think about a subordinate agent as a faulty one. The reason is that a subordinate agent does not selfishly promote its own benefit, because it is controlled by another selfish (non-faulty) agent, i.e., the coordinator. However, subordinate agents do not present an arbitrary behavior (as in [55,50,48,49,19,47]).

Generally speaking, given an integer $t \in [1, |N|]$, we have that a *t-defendable* Nash equilibrium is a *t-resilient* Nash equilibrium, and a *t-resilient* Nash equilibrium is a *t-strong* Nash equilibrium. Also, let $\mathcal{X}$ be any of the properties *defendable*, *resilient*, and *strong*. Then, for any $t \in [2, |N|]$, we have that a $(t+1)$-$\mathcal{X}$ Nash equilibrium is also a t-$\mathcal{X}$ Nash equilibrium. Therefore, a 1-$\mathcal{X}$ Nash equilibrium [56] is the conventional Nash equilibrium, and the n-$\mathcal{X}$ Nash equilibrium is the conventional strong Nash equilibrium [26,25].

We investigate another aspect of selfish computers in autonomous systems. Prior work, such as [26,27], assumed that any coalition of agents deviates only when their benefits are improved (see [26]), or at least protected (see [27]). This assumption implies that there are no (faulty) computers that do not optimize their benefits. Once the coalition includes at least one such faulty computer, there are no explicit guarantees for the system's behavior; the correct members of any coalition have a larger set of joint deviations from which they might benefit. This aspect is not considered by earlier work; subsystem takeovers model a more severe case in which almost all of the deviating agents are possibly faulty (e.g., all subordinates are faulty, and the coordinator optimizes its benefit.) Interestingly, autonomous recovery can be guaranteed in the presence of subsystem takeovers.

- *Do the assumptions of synchrony and observable actions hold in distributed system?* These are well-known settings that can be realized; every period can be defined to be sufficiently long to allow the stabilization of the underlying protocols (i.e., the actions' implementation). This behavior can be facilitated by the game authority [41,42,45] in which a self-stabilizing Byzantine clock synchronization protocol periodically restarts a Byzantine agreement protocol. The agreement explicitly facilitates observable actions, and the synchronization protocol overcomes timing failures.

11.2. Future directions

Several fundamental questions remain to be studied. For example, can we ensure the players to act in a smart way? Namely, the smartest possible way, as players may act irrationally due to lack of analytical skills, or just because it is too computationally demanding to find the best strategy profile.

There are several important aspects in this type of problem. One aspect is the ability to find the best strategy profile and another aspect is to be able to prove that one would act in a rational way when following a given strategy.

The following anecdote can depict the problem. Two people had been walking for many hours on a deserted country road. At sundown, they decided to sleep on the road because it was a moonless night. The rational person decided to sleep on the muddy roadside, to avoid being harmed by a car, while the other person decided to sleep more conveniently in the middle of the road. Just after midnight, a car approached, and at the last moment the driver noticed the sleeping person in the middle of the road and turned to the roadside.

11.3. Conclusions

Distributed algorithm designers often assume that processes execute identical software. Alternatively, when they do not assume this, designers turn to non-cooperative games. The game authority middleware places itself between these extremes, enabling the majority of the system processes to vote for and enforce the rules of the game.

Interestingly, the experience gained in structuring human society proves that scalability and advancement are gained by promoting honest selfishness and freedom of choice for individuals. The individual participates in forming the infrastructures that establish social rules and in their enforcement. The chosen rules promote competitiveness and individual gain from individual creativity and effort. Creativity and effort are imperative for the success of both individuals and society. Therefore, an honest majority with a beneficial attitude designs the rules in a way that individual success is driven by the individual actions whose outcome advances society.

Our game authority design is a step towards forming computer system structures that are inspired by a successful democratic society. We believe that such a middleware infrastructure is essential for the advancement and scalability of Internet-wide societies.

Acknowledgements

This work has been partially supported by the ICT Programme of the European Union under contract number ICT-2008-215270 (FRONTS). The third author acknowledges the support of the Propondis Foundation. We thank Elias Koutsoupias for helpful discussions.

REFERENCES

[1] M. Cagalj, S. Ganeriwal, I. Aad, J.-P. Hubaux, On selfish behavior in CSMA/CA networks, in: INFOCOM, IEEE, 2005, pp. 2513–2524.
[2] A.S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, C. Porth, BAR fault tolerance for cooperative services, SIGOPS Operating Systems Review (ISSN: 0163-5980) 39 (5) (2005) 45–58.
[3] T. Moscibroda, S. Schmid, R. Wattenhofer, When selfish meets evil: Byzantine players in a virus inoculation game, in: [57], 2006, pp. 35–44.
[4] E. Koutsoupias, C.H. Papadimitriou, Worst-case equilibria, Computer Science Review 3 (2) (2009) 65–69.
[5] E. Koutsoupias, C.H. Papadimitriou, Worst-case equilibria, in: C. Meinel, S. Tison (Eds.), STACS, in: Lecture Notes in Computer Science, vol. 1563, Springer, 1999, pp. 404–413.
[6] E. Anshelevich, A. Dasgupta, J.M. Kleinberg, É. Tardos, T. Wexler, T. Roughgarden, The price of stability for network design with fair cost allocation, in: FOCS, IEEE Computer Society, ISBN: 0-7695-2228-9, 2004, pp. 295–304.
[7] E. Elkind, H. Lipmaa, Hybrid voting protocols and hardness of manipulation, in: X. Deng, D.-Z. Du (Eds.), ISAAC, in: LNCS, vol. 3827, Springer, ISBN: 3-540-30935-7, 2005, pp. 206–215.
[8] R.J. Aumann, The core of a cooperative game without side payments, Transactions of the American Mathematical Society 98 (3) (1961) 539–552.
[9] A. Hayrapetyan, É. Tardos, T. Wexler, The effect of collusion in congestion games, in: [58], 2006, pp. 89–98.
[10] E.W. Dijkstra, Self-stabilizing systems in spite of distributed control, Communications of the ACM (ISSN: 0001-0782) 17 (11) (1974) 643–644.
[11] S. Dolev, Self-Stabilization, MIT Press, Cambridge, MA, USA, ISBN: 0-262-04178-2, 2000.

[12] M. Osborne, A. Rubinstein, A Course in Game Theory, MIT Press, 1994.
[13] R.J. Aumann, L.S. Shapley, Long-term competition: a game-theoretic analysis, Dept. of Economics, Los Angeles University of California, 1992.
[14] A. Rubinstein, Equilibrium in supergames, in: N. Meggido (Ed.), Essays in Game Theory in Honor of Michael Maschler, Springer-Verlag, Berlin, 1994.
[15] S. Hart, Robert Aumann's game and economic theory, Scandinavian Journal of Economics 108 (2) (2006) 185–211.
[16] R. Selten, Spieltheoretische Behandlung eines Oligopolmodells mit Nachfragetragheit, Zeitschrift fur die Gesamte Staatswissenschaft 12 (1965) 301–324.
[17] M. Blum, Coin flipping by telephone a protocol for solving impossible problems, SIGACT News (ISSN: 0163-5700) 15 (1) (1983) 23–27.
[18] S. Dolev, J.L. Welch, Self-stabilizing clock synchronization in the presence of Byzantine faults, Journal of the ACM (ISSN: 0004-5411) 51 (5) (2004) 780–799.
[19] A. Clement, J. Napper, H.C. Li, J.-P. Martin, L. Alvisi, M. Dahlin, Theory of BAR games, in: [59], 2007, pp. 358–359.
[20] S. Dolev, E.M. Schiller, P.G. Spirakis, P. Tsigas, Strategies for repeated games with subsystem takeovers implantable by deterministic and self-stabilizing automata, Tech. Rep. 2008:11, Department of Computer Science and Engineering, Chalmers University of Technology and Göteborg University, 2008.
[21] S. Dolev, E.M. Schiller, P.G. Spirakis, P. Tsigas, Game authority for robust and scalable distributed selfish-computer systems, Theoretical Computer Science (ISSN: 0304-3975) 411 (26–28) (2010) 2459–2466.
[22] J. Feigenbaum, S. Shenker, Distributed algorithmic mechanism design: recent results and future directions, in: DIALM'02: Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, ACM Press, NYC, NY, USA, ISBN: 1-58113-587-4, 2002, pp. 1–13.
[23] I. Abraham, D. Dolev, J.Y. Halpern, Lower bounds on implementing robust and resilient mediators, in: R. Canetti (Ed.), TCC, in: Lecture Notes in Computer Science, vol. 4948, Springer, ISBN: 978-3-540-78523-1, 2008, pp. 302–319.
[24] N. Andelman, M. Feldman, Y. Mansour, Strong price of anarchy, in: N. Bansal, K. Pruhs, C. Stein (Eds.), SODA, SIAM, ISBN: 978-0-898716-24-5, 2007, pp. 189–198.
[25] A. Rubinstein, Strong perfect equilibrium in supergames, International Journal of Game Theory 9 (1) (1980) 1–12.
[26] R.J. Aumann, Acceptable points in general cooperative *n*-person games, in: Contributions to the Theory of Games, vol. 4, 1959, pp. 287–324.
[27] I. Abraham, D. Dolev, R. Gonen, J.Y. Halpern, Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation, in: [57], 2006, pp. 53–62.
[28] B. Bernheim, M. Whinston, Coalition-proof Nash equilibria. II. Applications, Journal of Economic Theory 42 (1) (1987) 13–29.
[29] D. Moreno, J. Wooders, Coalition-proof equilibrium, Games and Economic Behavior 17 (1) (1996) 80–112.
[30] E. Britannica, Encyclopaedia Britannica Online, Encyclopaedia Britannica, 2004.
[31] N. Nisan, T. Roughgarden, É. Tardos, V.V. Vazirani, Algorithmic Game Theory, Cambridge University Press, New York, NY, USA, 2007.
[32] C. Daskalakis, P.W. Goldberg, C.H. Papadimitriou, The complexity of computing a Nash equilibrium, in: [58], 2006, pp. 71–78.
[33] C. Borgs, J. Chayes, N. Immorlica, A. Kalai, V. Mirrokni, C. Papadimitriou, The Myth of the Folk theorem, Report 07-082, Electronic Colloquium on Computational Complexity, ECCC (ISSN: 1433-8092), 14th Year, 82nd Report, 2007.
[34] E. Kalai, W. Stanford, Finite rationality and interpersonal complexity in repeated games, Econometrica 56 (2) (1988) 397–410.
[35] J.A. Garay, Y. Moses, Fully polynomial Byzantine agreement for processors in rounds, SIAM Journal on Computing (ISSN: 0097-5397) 27 (1) (1998) 247–290.
[36] L. Lamport, R. Shostak, M. Pease, The Byzantine generals problem, ACM Transactions on Programming Languages and Systems (ISSN: 0164-0925) 4 (3) (1982) 382–401.
[37] S. Dolev, E.M. Schiller, P.G. Spirakis, P. Tsigas, Strategies for repeated games with subsystem takeovers: implementable by deterministic and self-stabilizing automata (extended abstract), in: A. Manzalini (Ed.), Autonomics, vol. 37, ISBN: 978-963-9799-34-9, 2008.
[38] S. Dolev, E.M. Schiller, P. Spirakis, P. Tsigas, Strategies for repeated games with subsystem takeovers implementable by deterministic and self-stabilizing automata, International Journal on Autonomous and Adaptive Communications Systems (IJAACS). Special Issue with selected papers from Autonomics 2008, accepted February 2009.
[39] T.F. Lunt, Automated audit trail analysis and intrusion detection: a survey, in: Proceedings of the 11th National Computer Security Conference, 1988, pp. 65–73. http://www.csl.sri.com/papers/survey88/.
[40] B. Bernheim, D. Ray, Collective dynamic consistency in repeated games, Games and Economic Behavior 1 (4) (1989) 295–326.
[41] D. Shlomi, E.M. Schiller, S.G. Paul, Game Authority: for Robust Distributed Selfish-Computer Systems, Tech. Rep., Department of Computer Science and Engineering, Chalmers University of Technology and Goteborg University, 2006.
[42] S. Dolev, E.M. Schiller, P. Spirakis, Game authority: for robust distributed selfish-computer systems, Tech. Rep., Dynamically Evolving, Large-Scale Information Systems, DELIS, 2006. Accessible via: http://delis.upb.de/docs/.
[43] A. Dasgupta, S. Ghosh, S. Tixeuil, Selfish stabilization, in: A.K. Datta, M. Gradinariu (Eds.), SSS, in: Lecture Notes in Computer Science, vol. 4280, Springer, ISBN: 978-3-540-49018-0, 2006, pp. 231–243.
[44] J. Cohen, A. Dasgupta, S. Ghosh, S. Tixeuil, An exercise in selfish stabilization, ACM Transactions on Autonomous and Adaptive Systems 3 (4) (2008).
[45] S. Dolev, E.M. Schiller, P.G. Spirakis, P. Tsigas, Game authority for robust and scalable distributed selfish-computer systems, in: [59], 2007, pp. 356–357.
[46] H. Cao, A. Arora, Stabilization in dynamic systems with varying equilibrium, in: T. Masuzawa, S. Tixeuil (Eds.), Stabilization, Safety, and Security of Distributed Systems, 9th International Symposium, SSS 2007, Paris, France, November 14–16, 2007, in: Lecture Notes in Computer Science, vol. 4838, Springer, ISBN: 978-3-540-76626-1, 2007, pp. 67–81.
[47] L. Alvisi, BAR—where distributed computing meets game theory, in: A. Bondavalli, F.V. Brasileiro, S. Rajsbaum (Eds.), Dependable Computing, Third Latin-American Symposium, LADC 2007, Morella, Mexico, September 26–28, 2007, Proceedings, in: Lecture Notes in Computer Science, vol. 4746, Springer, ISBN: 978-3-540-75293-6, 2007, pp. 235–236.
[48] A.S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, C. Porth, BAR fault tolerance for cooperative services, in: A. Herbert, K.P. Birman (Eds.), SOSP, ACM, ISBN: 1-59593-079-5, 2005, pp. 45–58.
[49] H.C. Li, A. Clement, E.L. Wong, J. Napper, I. Roy, L. Alvisi, M. Dahlin, BAR gossip, in: 7th Symposium on Operating Systems Design and Implementation, OSDI'06, November 6–8, Seattle, WA, USA, USENIX Association, 2006, pp. 191–204.
[50] A. Clement, J. Napper, H. Li, J.-P. Martin, L. Alvisi, M. Dahlin, Theory of BAR games architecture, Technical Report TR-06-63, The University of Texas at Austin, Department of Computer Sciences, 2006.

[51] R. Axelrod, On six advances in cooperation theory, Analyse und Kritik 22 (1) (2000) 130–151.
[52] R.J. Aumann, et al. Survey of repeated games, in: Essays in Game Theory and Mathematical Economics in Honor of Oskar Morgenstern, vol. 4.
[53] D. Pearce, Repeated games: cooperation and rationality, in: Advances in Economic Theory: Sixth World Congress.
[54] J.Y. Halpern, Computer Science and Game Theory: A Brief Survey, CoRR abs/cs/0703148.
[55] K. Eliaz, Fault tolerant implementation, Review of Economic Studies 69 (3) (2002) 589–610.
[56] J. Nash, Equilibrium points in *n*-person games, Proceedings of the National Academy of Sciences of the United States of America 36 (1950) 48–49.
[57] E. Ruppert, D. Malkhi (Eds.), Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23–26, 2006, ACM, ISBN: 1-59593-384-0, 2006.
[58] J.M. Kleinberg (Ed.), Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21–23, 2006, ACM, ISBN: 1-59593-134-1, 2006.
[59] I. Gupta, R. Wattenhofer (Eds.), Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007, Portland, Oregon, USA, August 12–15, 2007, ACM, ISBN: 978-1-59593-616-5, 2007.

Shlomi Dolev received his B.Sc. in Engineering and B.A. in Computer Science in 1984 and 1985, and his M.Sc. and D.Sc. in Computer Science in 1990 and 1992 from the Technion Israel Institute of Technology. From 1992 to 1995 he was at Texas A&M University, as a post-doctoral assistant of Jennifer Welch. In 1995 he joined the Department of Mathematics and Computer Science at Ben-Gurion University, where he is now an full professor. He was a visiting researcher/professor at MIT, DIMACS, and LRI, for several periods during summers. Shlomi is the author of the book "Self-stabilization", published by the MIT Press. He has published more than 160 journal and conference scientific articles, and has served on the program committee of more than 60 conferences, including the ACM Symposium on Principles of Distributed Computing, and the International Symposium on DIStributed Computing. He is an associate editor of the IEEE Transactions on Computers, the AIAA Journal of Aerospace Computing, Information and Communication, and a guest editor of the Distributed Computing Journal and Theoretical Computer Science. His research grants include IBM faculty awards, Intel academic grants, and grants from the NSF. Shlomi is the founding chair of the Computer Science Department at Ben-Gurion University, where he now holds the Rita Altura Trust Chair in Computer Science. His current research interests include distributed computing, distributed systems, security and cryptography, and communication networks: in particular, the self-stabilization property of such systems. Recently, he has been involved in optical computing research.

Elad M. Schiller (born 1974) received his Ph.D., M.Sc., and B.Sc. in Computer Science in 2004, 2000, and 1998, from the Department of Mathematics and Computer Science at Ben-Gurion University of the Negev. His research excellence has been acknowledged by two highly competitive research fellowships from the Israeli government and the Swedish government. He is now an associate professor in the Department of Computer Science and Engineering at Chalmers University of Technology and Gothenburg University. His research interests include distributed computing, with special emphasis on self-stabilizing algorithms, wireless communications, and game theory.

Paul G. Spirakis, born in 1955, obtained his Ph.D. from Harvard University, USA, in 1982. Has served as a postdoctoral researcher at Harvard University and as an assistant professor at New York University, (the Courant Institute). He was appointed as a Full Professor in the Department of Computer Science and Engineering of Patras University (Greece) in 1990.

He has been honoured several times with international prizes and grants (e.g. NSF), and also with the top prize of the Greek Mathematics Society. He was acknowledge as one of the top 50 scientists worldwide in Computer Science with respect to "The best Nurturers in Computer Science research", published by B. Kumar and Y.N. Srikant, ACM Data Mining, 2005.

He was appointed a Distinguished Visiting Scientist of Max Planck Informatik.

He is the Director of the Research Academic Computer Technology Institute (RA.CTI). His research interests include algorithms and complexity and interaction of complexity and game Theory.

He has extensively published in most of the important Computer Science journals and most of the significant refereed conferences. He has edited various conference proceedings, and is currently an Editor of several prestigious journals. He has published two books through Cambridge University Press, and eight books in Greek.

He was the Greek National Representative in the Information Society Research Programmes (IST) from January 1999 till June 2002. He was elected unanimously as one of the two Vice-Presidents of the Council of the European Association for Theoretical Computer Science (EATCS). He has been a member of ISTAG (Information Society Technologies Advisory Group), a prestigious body of about 40 individuals advising the EU on research policy, from January 2003 to January 2005.

Paul Spirakis has been member of the ACM Europe Task Force since Fall 2008. The ACM Europe Task force is now the ACM Europe Council. Also, he is a member of the ERC/IDEAS Panel of evaluations for Computer Science, 2009–2010. He consults for the Greek State, the European Union, and several major Greek computing industries.

Philippas Tsigas' research interests include concurrent data structures for multiprocessor systems, communication and coordination in parallel systems, fault-tolerant computing, mobile computing, and information visualization. He received his B.Sc. in Mathematics from the University of Patras, Greece, and his Ph.D. in Computer Engineering and Informatics from the same university. He was at the National Research Institute for Mathematics and Computer Science, Amsterdam, the Netherlands (CWI), and at the Max-Planck Institute for Computer Science, Saarbrücken, Germany, before joining Chalmers University of Technology. At present he is a professor at the Department of Computing Science at Chalmers University of Technology, Sweden (www.cse.chalmers.se/~tsigas).