Model Checking Nash Equilibria in MAD Distributed Systems

Federico Mari, Igor Melatti, Ivano Salvo, Enrico Tronci
Dep. of Computer Science
University of Rome “La Sapienza”
Via Salaria 113, 00198 Roma, Italy
Email: {mari,melatti,salvo,tronci}@di.uniroma1.it

Lorenzo Alvisi, Allen Clement, Harry Li
Dep. of Computer Science
University of Texas at Austin
1 University Station C0500, Austin, Texas, USA
Email: {lorenzo,aclement,harry}@cs.utexas.edu

Abstract—We present a symbolic model checking algorithm for verification of Nash equilibria in finite state mechanisms modeling Multiple Administrative Domains (MAD) distributed systems.

Given a finite state mechanism, a proposed protocol for each agent and an indifference threshold for rewards, our model checker returns PASS if the proposed protocol is a Nash equilibrium (up to the given indifference threshold) for the given mechanism, FAIL otherwise.

We implemented our model checking algorithm inside the NuSMV model checker and present experimental results showing its effectiveness for moderate size mechanisms.

I. INTRODUCTION

Cooperative services are increasingly popular distributed systems in which nodes (agents) belong to Multiple Administrative Domains (MAD). Thus in a MAD distributed system each node owns its resources and there is no central authority owning all system nodes. Examples of MAD distributed systems include Internet routing [25], [49], wireless mesh routing [40], file distribution [16], archival storage [41], cooperative backup [6], [17], [37].

In traditional distributed systems, nodes may deviate from their specifications (Byzantine nodes) because of bugs, hardware failures, faulty configurations, or even malicious attacks. In MAD systems, nodes may also deviate because their administrators are rational, i.e., selfishly intent on maximizing their own benefits from participating in the system (selfish nodes). For example, selfish nodes may change arbitrarily their protocol if that is at their advantage.

Cooperative file distribution (e.g. see [16]) is a typical example of the above scenario. Every peer will be happy to download file chunks from other peers. However, in order to save bandwidth, a selfish peer may modify its protocol parameters to disallow upload of its file chunks.

In this paper we present an automatic verification algorithm for MAD distributed systems. That is, given a protocol P for a MAD system and a property \( \varphi \) for P we want to automatically verify if \( \varphi \) holds for P.

Note that in a MAD system any node may behave selfishly. This rules out the classical approach (e.g. see [48]) of modeling nodes deviating from the given protocol as Byzantine [34]. In fact, doing so would leave us with a system in which all nodes are Byzantine. Very few interesting protocols (if any) work correctly under such conditions. Thus, in order to verify MAD systems, we first need a model for them in which protocol correctness can be formally stated and hopefully proved. This issue has been studied in [1], [15] where the BAR model has been introduced.

In BAR, a node is either Byzantine, Altruistic, or Rational. Byzantine nodes, as usual, can deviate from their specification in any way for any reason. Altruistic nodes follow their specification faithfully, without considering their self-interest. Rational nodes deviate selfishly from a given protocol if doing so improves their own utility. In the BAR framework correctness of a protocol with respect to a given property is stated as BAR tolerance. Namely, a protocol is BAR tolerant if it guarantees the desired property despite the presence of Byzantine and rational players.

Several BAR tolerant protocols have since been proposed [1], [36] to implement cooperative services for p2p backup and live data streaming. Taking into account how hard it is to formally prove correctness for classical distributed protocols it is not surprising that formally proving that a given protocol is BAR tolerant is indeed quite a challenge (e.g. see [15]).

This motivates investigating if the model checking techniques devised for classical distributed protocols can also be used in our framework. To this end we note that in order to show that a protocol is BAR tolerant, it is sufficient to show that it satisfies the given property when all rational nodes follow the protocol exactly and then to show that all rational nodes do, in fact, follow the protocol exactly.

If all rational nodes follow the given protocol exactly we are left with a system with only Byzantine and altruistic nodes. Well known model checking techniques (e.g. see [14] for a survey) are available to verify that such systems satisfy a given property despite the presence of a limited number of Byzantine nodes. It suffices, as usual, to model Byzantine nodes with nondeterministic automata.

Unfortunately, to the best of our knowledge, no model checking algorithm or tool is available to address the second BAR tolerance requirement, namely proving that all rational nodes do follow the given protocol exactly.

To fill this gap in this paper we present a symbolic model checking algorithm to automatically verify that it is in the best interest of each rational agent to follow exactly the given protocol. This is usually accomplished by proving that no
rational agent has an incentive in deviating from the proposed protocol. This, in turn, is done by proving that the proposed protocol is a Nash equilibrium (e.g. see [25], [111]).

A. Our contribution

First of all we need a formal definition of mechanism suitable for model checking purposes and yet general enough to allow modeling of interesting systems. Accordingly in Sect. III we present a definition of Finite State Mechanism suitable for modeling of finite state BAR systems as well as for developing effective verification algorithms for them. We model each agent with a Finite State Machine defining its admissible behavior, that is, using the BAR terminology, its Byzantine behavior. Each agent action yields a real valued reward which depends both on the system state and on agents’ actions. The proposed protocol constrains which actions should be taken in each state. This protocol defines the behavior of altruistic agents.

The second obstruction to overcome is the fact that we need to handle infinite games since nodes are running, as usual, nonterminating protocols. As a result, we need to rank infinite sequences of agents’ actions (strategies). This is done in Sects. V, VI by using a discount factor (as usual in game theory [26]) to decrease relevance of rewards too far in the future. In Prop. 1 we give a dynamic programming algorithm to effectively compute the value of a finite strategy in our setting.

To complete our framework we need a notion of equilibrium that can be effectively computed by only looking at finite strategies and that accommodates Byzantine agents. Accordingly, in Sect. VII we give a definition of mechanism Nash equilibrium accounting for the presence of up to $f$ Byzantine players (along the lines of [21]) and for agent tolerance to small ($\varepsilon > 0$) differences in rewards (along the classical lines of, e.g. [23], [26]). This leads us to the definition of $\varepsilon$-$f$-Nash equilibrium in Def. 4.

Sect. VIII gives our main theorem on which correctness of our verification algorithm rests. Theor. 2 shows that $\varepsilon$-$f$-Nash equilibria for finite state BAR systems can be automatically verified within any desired precision $\delta > 0$ by just looking at long enough finite sequences of actions.

Sect. IX presents our symbolic model checking algorithm for Nash equilibria. Our algorithm inputs are: a finite state mechanism $\mathcal{M}$, a proposed protocol for $\mathcal{M}$, the tolerance $\varepsilon > 0$ for agents to differences in rewards, the maximum number $f$ of allowed Byzantine agents, our desired precision $\delta > 0$. Our algorithm returns PASS if the proposed protocol is indeed a $(\varepsilon + \delta)$-$f$-Nash equilibrium for $\mathcal{M}$, FAIL otherwise.

We implemented (Sect. X) our algorithm on top of NuSMV [46] using ADDs (Arithmetic Decision Diagrams) [18] to manipulate real valued rewards.

Finally in Sect. XI we present experimental results showing effectiveness of our approach on moderate size mechanisms. For example, within 22 hours using 5 GB of RAM we can verify mechanisms whose global present state representation requires 32 bits. The corresponding normal form games for such mechanisms would have more than $10^{22}$ entries.

B. Related works

Design of mechanisms for rational agents has been widely studied (e.g. [49], [45], [13]). Design methods for BAR protocols have been investigated in [1], [36], [15], [21]. We differ from such works since our focus here is on automatic verification of Nash equilibria for finite state BAR systems rather than on design principles for them.

Algorithms to search for pure, mixed (exact or approximate) Nash equilibria in games have been widely studied (e.g. see [24], [19]). We differ from such works in two ways. First, all such line of research addresses explicitly presented games (normal form games) whereas we are studying implicitly presented games (namely, mechanisms defined using a programming language). Thus, (because of state explosion) the explicit representation of a mechanism has size exponential in the size of our input. This is much the same as the relationship between reachability algorithms for directed graphs and reachability algorithms for finite state concurrent programs. As a result the algorithms and tools (e.g. [27]) for explicit games cannot be used in our context. Second, we are addressing a verification problem, thus the candidate equilibrium is an input for us whereas it is an output for the above mentioned works.

The relationship between model checking and game theory has been widely studied in many settings.

Game theoretic approaches to model checking for the verification of concurrent systems have been investigated, for example, in [32], [35], [30], [39], [51]. An example of game based model checker capable of CTL, modal $\mu$-calculus and specification patterns is [29].

Model checking techniques have also been applied to the verification of knowledge and belief logics in game theoretic settings. Examples are in [7], [8], [12]. An example of a model checker for the logic of knowledge is MCK [28].

Applications of model checking techniques to game theory have also been investigated. For example, model checking techniques have been widely applied to the verification of games stemming from the modeling of multi-agent systems. See for example [31], [33], [38], [20]. An example of model checker for multi-agent programs is CASP [9]. An example of model checking based analysis of probabilistic games is in [5].

Note that the above papers focus on verification of temporal-like (e.g. temporal, belief, knowledge) properties of concurrent systems or of games whereas here we focus on checking Nash equilibria (of BAR protocols).

Synthesis of winning strategies for the verification game leads to automatic synthesis of correct-by-construction systems (typically controllers). This has been widely investigated in many settings. Examples are in [22], [47], [4], [2], [50], [52], [53], [3]. Note that the above papers focus on automatic synthesis (of systems or of strategies) whereas our focus here is on checking Nash equilibria (of BAR protocols).

Summing up, to the best of our knowledge, no model checking algorithm for the automatic verification of Nash equilibria of finite state mechanisms modeling BAR systems has been previously proposed.
II. Basic Notations

We denote an \( n \)-tuple of objects (of any kind) in boldface, e.g., \( \mathbf{x} \). Unless otherwise stated we denote with \( x_i \) the \( i \)-th element of the \( n \)-tuple \( \mathbf{x} \), \( x_{-i} \) the \((n-1)\)-tuple \( \langle x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n \rangle \), and with \( \langle x_{-i}, x \rangle \) the \( n \)-tuple \( \langle x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_n \rangle \).

We denote with \( \mathbb{B} \) the set \{0, 1\} of boolean values (0 for false and 1 for true). We denote with \( [n] \) the set \{1, \ldots, n\}. The set of subsets of \( X \) with cardinality at most \( k \) will be denoted by \( \mathcal{P}_k(X) \).

III. Finite State Mechanisms

In this section we give the definition of Finite State Mechanism by suitably extending the usual definition of the synchronous parallel of finite state transition systems. This guarantees that all mechanisms consisting of finite state protocols can be modeled in our framework.

Definition 1 (Mechanism Skeleton): An \( n \)-player (agent) mechanism skeleton \( \mathcal{U} \) is a tuple \( \langle \mathcal{S}, \mathcal{I}, \mathcal{A}, \mathcal{B}, \mathcal{h}, \beta \rangle \) whose elements are defined as follows.

- \( \mathcal{S} = \{S_1, \ldots, S_n\} \) is an \( n \)-tuple of nonempty sets (of local states). The state space of \( \mathcal{U} \) is the set (of global states) \( \mathcal{S} = \prod_{i=1}^{n} S_i \).
- \( \mathcal{I} = \{I_1, \ldots, I_n\} \) is an \( n \)-tuple of nonempty sets (of local initial states). The set of global initial states is \( I = \prod_{i=1}^{n} I_i \).
- \( \mathcal{A} = \langle A_1, \ldots, A_n \rangle \) is an \( n \)-tuple of nonempty sets (of local actions). The set of global actions (i.e. \( n \)-tuples of local actions) is \( A = \prod_{i=1}^{n} A_i \). The set of \( i \)-opponents actions is \( A_{-i} = \prod_{i=1,j \neq i}^{n} A_j \).
- \( \mathcal{B} = \langle B_1, \ldots, B_n \rangle \) is an \( n \)-tuple of functions s.t., for each \( i \in [n] \), \( B_i : S \times A_i \times S_i \rightarrow \mathbb{B} \). Function \( B_i \) models the transition relation of agent \( i \), i.e. \( B_i(s, a, s') \) is true iff agent \( i \) can move from (global) state \( s \) to (local) state \( s' \) via action \( a \). We require \( B_i \) to be serial (i.e. \( \forall s \in S \ \exists a \in A_i \ 3s' \in S_i \ s.t. \ B_i(s, a, s') \) holds) and deterministic (i.e. \( B_i(s, a, s') \land B_i(s, a, s'') \) implies \( s' = s'' \)). We write \( B_i(s, a) = 3s' B_i(s, a, s') \). That is, \( B_i(s, a) \) holds if action \( a \) is allowed in state \( s \) for agent \( i \). For each agent \( i \in [n] \), function \( B_i \) models the underlying behavior of agent \( i \). That is, the set of all possible choices of rational agent \( i \). As a result, \( B_i \) defines the transition relation for the Byzantine behavior for agent \( i \).
- \( \mathcal{h} = \langle h_1, \ldots, h_n \rangle \) is an \( n \)-tuple of functions s.t., for each player \( i \in [n] \), \( h_i : S \times A_i \rightarrow \mathbb{R} \). Function \( h_i \) models the payoff (reward) function of player \( i \). Note that \( h_i \) may be seen as a function \( h : S \times A \rightarrow \mathbb{R}^n \) s.t. \( h(s, a) = (h_1(s, a), \ldots, h_n(s, a)) \) for all global states \( s \in S \) and global actions \( a \in A \).
- \( \beta = \langle \beta_1, \ldots, \beta_n \rangle \) is an \( n \)-tuple of discounts, that is of real values such that for each \( i \in [n] \), \( \beta_i \in (0, 1) \).

Definition 2 (Mechanism): An \( n \)-player mechanism \( \mathcal{M} \) is a pair \( (\mathcal{U}, T) \) where: \( \mathcal{U} = \langle \mathcal{S}, \mathcal{I}, \mathcal{A}, \mathcal{B}, \mathcal{h}, \beta \rangle \) is a mechanism skeleton and \( T = \langle T_1, \ldots, T_n \rangle \) is an \( n \)-tuple of functions s.t., for each \( i \in [n] \), \( T_i : S \times A_i \rightarrow \mathbb{B} \). We require \( T_i \) to satisfy the following properties: 1) \( T_i(s, a) \) implies \( B_i(s, a); \) 2) (nonblocking) for each state \( s \in S \) there exists an action \( a \in A_i \) s.t. \( T_i(s, a) \) holds. Function \( T_i \) models the proposed protocol for agent \( i \), that is its obedient (or altruistic, following [1], [36]) behavior. More specifically, if agent \( i \) is altruistic then its transition relation is \( B_i(s, a, s') \land T_i(s, a) \).

Often we denote an \( n \)-player mechanisms \( \mathcal{M} \) with the tuple \( \langle \mathcal{S}, \mathcal{I}, \mathcal{A}, \mathcal{B}, \mathcal{h}, \beta \rangle \). Furthermore we may also call mechanism a mechanism skeleton. The context will always make clear the intended meaning.

Remark 1 (Finite State Agents): In order to develop our model checking algorithm we model each agent as a Finite State Machine (FSM). This limits agent knowledge about the past. In fact, the system state represents the system past history. Since our systems are nonterminating ones, histories (and thus agent knowledge) are in general unbounded. As for verification of security protocols (e.g. see [43], [44]) it is the modeler responsibility to develop a suitable finite state approximation of knowledge.

Definition 3: Let \( \mathcal{M} = \langle \mathcal{S}, \mathcal{I}, \mathcal{A}, \mathcal{B}, \mathcal{h}, \beta \rangle \) be an \( n \)-player mechanism and \( Z \subseteq [n] \). Let \( B_T : \mathcal{P}([n]) \times S \times A \times S \rightarrow \mathbb{B} \) be such that \( B_T(Z, s, a, s') = \bigwedge_{i=1}^{n} B_{T_i}(Z, s, a_i, s'_i) \), where

\[
B_{T_i}(Z, s, a_i, s'_i) = \begin{cases} B_i(s, a_i, s'_i) & \text{if } i \in Z \\ B_i(s, a_i, s'_i) \land T_i(s, a_i) & \text{otherwise.} \end{cases}
\]

\( B_T \) models the transition relation of mechanism \( \mathcal{M} \), when the set of Byzantine players is \( Z \) and all agents not in \( Z \) are altruistic.

IV. An Example of Mechanism

In order to clarify our definitions we give an example of a simple mechanism. Consider the situation in which a set of agents cooperate to accomplish a certain job. The job, in turn, consists of \( n \) tasks. Each task is assigned to at least one agent which may carry out the assigned task or may deviate by not doing any work. Carrying out the assigned task entails a cost (negative reward) for the agent. On the other hand, if all tasks forming the job are completed (and thus the job itself is completed) all agents that have worked to a task get a reward greater than the cost incurred to carry out the assigned task. If the job is not completed no agent gets anything. The mechanism skeleton (Def. 1) is defined as follows.

All agents have the same discount, say \( \beta = 0.5 \), and the same underlying (Byzantine) behavior, defined by the automaton \( B_i \) in Fig. 1.

![Fig. 1. Underlying behavior B_i for agent i.](image-url)
The mechanism (Def. 2) is $M = (S, I, A, T, B, h, \beta)$, where the proposed protocol $T_i$ for agent $i$ requires agent $i$ to cooperate, that is to carry out the assigned task. Formally: $T_i(s, a_i) = ((s_i = 0) \land (a_i = \text{work})) \lor ((s_i = 1) \land (a_i = \text{gain})) \lor ((s_i = 2) \land (a_i = \text{reset})).$

V. PATHS IN MECHANISMS

Let $M = (S, I, A, T, B, h, \beta)$ be an $n$ player mechanism and let $Z \subseteq [n]$ be a set of (Byzantine) agents.

A path in $(M, Z)$ (or simply a path when $(M, Z)$ is understood from the context) is a (finite or infinite) sequence $\pi = s(0)\alpha(0)s(1)\ldots s(t)\alpha(t)s(t+1)\ldots$ where, for each $t$, $s(t)$ is a global state, $\alpha(t)$ is a global action and $BT(Z, s(t), \alpha(t), s(t+1))$ holds.

The length of a path is the number of global actions in a path. We denote with $|\pi|$ the length of path $\pi$. If $\pi$ is infinite we write $|\pi| = \infty$. Note that if $\pi = s(0)$ then $|\pi| = 0$. Thus a path of length $0$ is not empty.

In order to extract the $t$-th global state and the $t$-th global action from a given path $\pi$, we define $\pi^{(t)}(t) = s(t)$ and $\pi^{(a)}(t) = \alpha(t)$. To extract local actions, we denote with $\pi^{(a)}(t)$ the action $a_i(t)$ at stage $t$ of agent $i$ and with $s_{<t}^{(a)}$ the actions $a_{<t}(t)$ at stage $t$ of all agents but $i$.

For each agent $i \in [n]$, the value of a path $\pi$ is $v_i(\pi) = \sum_{t=0}^{\infty} \beta_t^i h_i(\pi^{(s)}(t), \pi^{(a)}(t))$. Note that for any path $\pi$ and agent $i \in [n]$ the path value $v_i(\pi)$ is well defined also when $|\pi| = \infty$ since the series $\sum_{t=0}^{\infty} \beta_t^i h_i(\pi^{(s)}(t), \pi^{(a)}(t))$ converges for all $\beta_t^i \in (0, 1)$. The path value vector is defined as: $v(\pi) = (v_1(\pi), \ldots, v_n(\pi))$.

A strategy $\sigma$ in $(M, Z)$ is said to be $i$-altruistic if for all $t < |\pi|$, $T_i(\pi^{(s)}(t), \pi^{(a)}(t))$ holds.

Given a path $\pi$ and a nonnegative integer $k \leq |\pi|$ we denote with $\pi|_k$ the prefix of $\pi$ of length $k$, i.e. the finite path $\pi|_k = s(0)\alpha(0)s(1)\ldots s(k)$ and with $\pi|_k$ the tail of $\pi$, i.e. the path $\pi|_k = s(k)\alpha(s(k)+1)\ldots s(t)\alpha(t)s(t+1)\ldots$

We denote with $Path_h(s, Z)$ the set of all paths of length $k$ starting at $s$. Formally, $Path_h(s, Z) = \{\pi | \pi$ is a path in $(M, Z)$ and $|\pi| = k \land \pi^{(a)}(0) = s\}$.

We denote with $Path_h(s, f)$ the set of all paths of length $k$ feasible with respect to all sets of Byzantine agents of cardinality at most $f$. Formally, $Path_h(s, f) = \bigcup_{Z \subseteq f} Path_h(s, Z)$. We write $Path_h(s)$ for $Path_h(s, n)$.

Unless otherwise stated in the following, we omit the subscript or superscript horizon when it is $\infty$. For example, we write $Path(s, Z)$ for $Path_{\infty}(s, Z)$.

Note that if $i \notin Z$, all paths in $Path_h(s, Z)$ are $i$-altruistic, that is, agent $i$ behaves accordingly to the proposed protocol.

VI. STRATEGIES

Let $M = (S, I, A, T, B, h, \beta)$ be an $n$ player mechanism and let $Z \subseteq [n]$ be a set of (Byzantine) agents.

As usual in a game theoretic setting, we need to distinguish player actions (i.e. local actions) from those of its opponents. This leads to the notion of strategy.

A strategy $\sigma$ is a (finite or infinite) sequence of local actions for a given player. The length $|\sigma|$ of $\sigma$ is the number of actions in $\sigma$ (thus if $|\sigma| = 0$, the strategy is empty).

A strategy $\sigma$ for player $i$ agrees with a path $\pi$ (notation $\pi \succeq i \sigma$) if $|\sigma| = |\pi|$ and for all $t < |\sigma|$, $\sigma(t) = \pi^{(a)}_i(t)$.

Given a path $\pi$, the strategy (of length $|\pi|$) for player $i$ associated to $\pi$ is $\sigma(\pi, i) = \pi^{(a)}_i(0)\pi^{(a)}_i(1)\ldots \pi^{(a)}_i(|\pi|)$. .

The set of $Z$-feasible strategies of length $k$ for player $i$ in state $s$ is $Strat_k(s, Z, i) = \{\sigma(\pi, i) | \pi \in Path_h(s, Z)\}$.

The set of all $Z$-feasible strategies of length $k$ for player $i$ in state $s$ is $Strat_k(s, f, i) = \{\sigma(\pi, i) | \pi \in Path_h(s, f)\}$.

As for paths, a strategy $\sigma \in Strat_k(s, Z, i)$ is said to be $i$-altruistic if $i \notin Z$. We use the notations $\sigma^k_i$ and $\sigma^k$ to denote, respectively, the $k$-prefix and the tail after $k$ steps of a strategy.

The set of paths that agree with a set of strategies $\Sigma$ for an agent $i$ is defined as follows. $Path(s, Z, i, \Sigma) = \{\pi \in Path_h(s, Z) | \exists \sigma \in \Sigma, k = |\sigma| \land \pi \succeq i \sigma\}$. When $\Sigma$ is the singleton $\{\sigma\}$, we simply write $Path(s, Z, i, \sigma)$.

The guaranteed outcome (or the value) of a strategy $\sigma$ in state $s$ for player $i$ is the minimum value of paths that agree with $\sigma$. Formally: $v_i(Z, s, \sigma) = \min\{v_i(\pi) | \pi \in Path(s, Z, i, \sigma)\}$.

The value of a state $s$ at horizon $k$ for player $i$ is the guaranteed outcome of the best strategy of length $k$ starting at state $s$.

The worst case value of a state $s$ at horizon $k$ for player $i$ is the outcome of the worst strategy of length $k$ starting at state $s$.

We use as usual $v_i(Z, s)$ for $v_i^n(Z, s)$, and $u_i(Z, s)$ for $u_i^n(Z, s)$.

The finite horizon value of a state can be effectively computed by using a dynamic programming approach (Prop. 1). This is one of the main ingredients of our verification algorithm (Sect. IX). We omit proofs because of lack of space.

Proposition 1: Let $M = (S, I, A, T, B, h, \beta)$ be an $n$ player mechanism, $i \in [n]$, $Z \subseteq [n]$ and $s \in S$. The state values at horizon $k$ for player $i$ can be computed as follows:

$u_i^k(Z, s) = u_i^0(Z, s) = 0$;

$u_i^{k+1}(Z, s) = \max_{a_i \in A_i} \min_{a_{-i} \in A_{-i}} \{h_i(s, (a_i, a_{-i})) + \beta_i v_i^k(Z, s') | BT(Z, s, (a_i, a_{-i}), s')\}$;

$u_i^{n+1}(Z, s) = \min_{a_i \in A_i} \min_{a_{-i} \in A_{-i}} \{h_i(s, (a_i, a_{-i})) + \beta_i v_i^n(Z, s') | BT(Z, s, (a_i, a_{-i}), s')\}$

VII. NASH EQUILIBRIA IN MECHANISMS

Our notion of Nash equilibrium for a mechanism combines those in [23], [21]. Intuitively, a mechanism $M$ is $\ve$-f-Nash, if as long as the number of Byzantine agents is no more than $f$ (e.g. see [21]), no rational agent has an interest greater than $\ve$ (e.g. see [23], [26]) in deviating from the proposed protocol in $M$.

Definition 4 ($\ve$-f-Nash): Let $M = (S, I, A, T, B, h, \beta)$ be an $n$ player mechanism, $f \in \{0, \ldots, n\}$ and $\ve > 0$. $M$ is $\ve$-f-Nash for player $i \in [n]$ if $\forall Z \in \mathcal{P}_f([n] \setminus \{i\})$, $\forall s \in I$, $u_i(Z, s) + \ve \geq v_i(Z \cup \{i\}, s)$. $M$ is $\ve$-f-Nash if it is $\ve$-f-Nash for each player $i \in [n]$.

Note that $0$-f-Nash is the $\ve$-Nash equilibrium defined in [23], [26]. Furthermore, stretching Def. 4 by setting $\ve = 0$, we see that $0$-f-Nash is the $f$-Nash equilibrium defined in
of 0-Nash is the classical Nash equilibrium (e.g., see [26]). Observe that, for each agent $i$, we compare agent $i$ best reward when it considers deviating from the protocol $(v_i(Z \cup \{i\}, s))$, with agent $i$ worst reward when it obeys the protocol $(u_i(Z, s))$. The reason for tolerating a small ($\varepsilon$) tolerance on rewards when deviating from the proposed protocol in Def. 4 is that our aim is to verify Nash equilibria by looking only at finite strategies. It is well known (e.g. see Sect. 4.8 of [26]) that 0-Nash equilibria have been introduced to get within a finite horizon equilibria that are only available with an infinite horizon. This means that a finite horizon may not suffice to check that a mechanism is 0-Nash. The following example aims at clarifying the above well known game-theoretical issue framing it in our context.

**Example 1:** Let $M$ be a one agent (named 1) mechanism defined as follows. The underlying behavior $B_1$ for agent 1 is shown in Fig. 2 where on the automaton edges we show action names as well as payoff values since in this simple case the payoff function depends only on local states and local actions of the agent. The discount factor for agent 1 is $\beta_1 = \frac{1}{3}$. Let the proposed protocol $T$ of $M$ be defined as follows: $T_1(s, a) = ((s = 0) \land (a = a)) \lor ((s = 1) \land (a = d)) \lor ((s = 2) \land (a = e)) \lor (s \geq 3)$. We focus on the case $f = 0$, that is there are no Byzantine agents and hence $Z = \emptyset$. For all $k > 0$ we have: $u^k_1(\emptyset, 0) = -1 + \frac{1}{2} \sum_{i=0}^{k-2}(-1)^i = \frac{1}{2} - \frac{1}{2}k$ (the protocol $T$ prescribes to follow the strategy $a(do)A_{i=0}$). For all $k > 0$ s.t. $k$ is even and $\sigma_2 = c(gh)\sigma_2$ when $k$ is odd. Therefore, if $k$ is odd $u^k_1(\emptyset, 0) < u^k_1(\{1\}, 0)$, and if $k$ is even $u^k_1(\emptyset, 0) = u^k_1(\{1\}, 0)$. Thus there is no $k > 0$ s.t. for all $k \geq k$, $u^k_1(\emptyset, 0) \geq u^k_1(\{1\}, 0)$.

![Fig. 2. Agent Behavior](image)

Finding 0-Nash equilibria even for finite horizon games is not trivial (e.g. see [19]). As for infinite games, we note that game-theory results focus on showing that an $\varepsilon$ small enough can be found so that an infinite horizon 0-Nash equilibrium becomes a finite horizon 0-Nash one (e.g. see [26]). Our concern here is different. We are given $\varepsilon$ (fixed) and want to verify if the given mechanism is 0-Nash (actually, $\varepsilon$-f-Nash).

From Example 1 one may conjecture that $\varepsilon$-f-Nash ($\varepsilon > 0$) equilibria can be verified by just looking at long enough finite histories. Unfortunately this is not always the case as shown by the following example.

**Example 2:** Consider again the mechanism $M$ in Example 1. Let the proposed protocol $T$ of $M$ be defined as follows: $T_1(s, a) = ((s = 0) \land (a = a)) \lor (s \geq 1)$. Also in this case we focus on the case in which there are no Byzantine agents, that is $f = 0$ and $Z = \emptyset$. For all $k > 0$ we have: $u^k_1(\emptyset, 0) = \sum_{i=1}^{k} \frac{1}{2^i} = -1 + \frac{1}{2}k$ and $v^k_1(\{1\}, 0) = \frac{1}{2}k$. For all $k > 0$ we have: $\Delta(k) = v^k_1(\{1\}, 0) - u^k_1(\emptyset, 0) = (1 + \frac{1}{2^k})$. Now, $M$ is clearly 1-0-Nash however there is no $k > 0$ s.t. for all $k \geq k$, $\Delta(k) \leq 1$. That is, there is no finite horizon that allows us to conclude that $M$ is 1-0-Nash. Note, however, that for all $\delta > 0$ there exists a $k > 0$ s.t. for all $k \geq k$, $\Delta(k) \leq 1 + \delta$. Thus, for all $\delta > 0$, by just considering a suitable finite horizon $k > 0$, we can verify that $M$ is $(1 + \delta)$-0-Nash.

**VIII. VERIFYING $\varepsilon$-f-NASH EQUILIBRIA**

In this Section we give our main theorem (Theor. 2) on which correctness of our verification algorithm (Sect. IX) rests.

**Example 2** shows that, in general, using finite horizon approximations, we cannot verify $\varepsilon$-f-Nash equilibria. However the very same example suggests that we may get arbitrarily close to this result. This is indeed our main theorem.

**Theorem 2 (Main Theorem):** Let $M = \langle S, I, A, T, B, h, \beta \rangle$ be an $n$ player mechanism, $f \in \{0, 1, \ldots, n\}$, $\varepsilon > 0$ and $\delta > 0$. Furthermore, for each agent $i \in [n]$ let:

1. $M_i = \max\{[hi(s, a)] | s \in S$ and $a \in A\}$. 
2. $E_i(k) = 5 \beta_i^k M_i - M_i^{-1}$. 
3. $\Delta_i(k) = max\{v^k_i(Z \cup \{i\}, s) - u^k_i(Z, s) | s \in I, Z \in \mathcal{P}_i([n] \setminus \{i\})\}$. 
4. $\varepsilon_i(i, k) = \Delta_i(k) - 2E_i(k)$ 
5. $\varepsilon_2(i, k) = \Delta_i(k) + 2E_i(k)$

For each agent $i$, let $k_i$ be s.t. $4E_i(k_i) < \delta$. Then we have:

1. If for each $i \in [n]$, $\varepsilon \geq \varepsilon_2(i, k_i) > 0$ then $M$ is $\varepsilon$-f-Nash.
2. If there exists $i \in [n]$ s.t. $0 < \varepsilon \leq \varepsilon_1(i, k_i)$ then $M$ is not $\varepsilon$-f-Nash. Of course in such a case a fortiorem $M$ is not 0-f-Nash.
3. If for each $i \in [n]$, $\varepsilon_1(i, k_i) < \varepsilon$ and there exists $j \in [n]$ s.t. $\varepsilon < \varepsilon_2(j, k_j)$ then $M$ is $(\varepsilon + \delta)$-f-Nash.

**Proof:** Because of lack of space we omit the proof. See [42] for the complete proof.

**IX. $\varepsilon$-f-NASH VERIFICATION ALGORITHM**

Resting on Prop. 1 and on Theor. 2, Algorithm 1 verifies that a given $n$ agent mechanism $M$ is $\varepsilon$-f-Nash.

In Algorithm 1, $s$ and $a$ are vectors (of boolean variables) ranging respectively on (the boolean encoding of) states ($S$) and actions ($A$).

The set $Z$ of Byzantine agents in Algorithm 1 can also be represented with a vector of $n$ boolean variables $b = (b_1, \ldots, b_n)$ such that for each agent $i \in [n]$, agent $i$ is Byzantine iff $b_i = 1$. Accordingly, the constraint $Z \in \mathcal{P}_i([n] \setminus \{i\})$ in Def. 4 becomes a constraint on $b$, namely: $\sum_{i=1}^{n} b_i \leq f$ and $b_i = 0$. Along the same lines, the set $Z \cup \{i\}$ (used in Algorithm 1) can be represented with the boolean vector $b[b_i := 1]$ obtained from $b$ by replacing variable $b_i$ in $b$ with the boolean constant 1. For readability in Algorithm 1 we use $Z$ rather than its boolean representation $b$.

First of all, in line 3 of Algorithm 1, we compute the horizon $k$ needed for agent $i$ to achieve the required accuracy $\delta$.

Lines 4–11 use Prop. 1 to compute state values at horizon $k$ of state $s$ with the set of Byzantine players $Z$ when player $i$ obeys the protocol $(v^k_i(Z, s))$ as well as when player $i$ behaves arbitrarily within the underlying behavior $(v^k_i(Z \cup \{i\}, s))$. 

[5]
Line 12 computes the maximum difference between internal state values for a rational player and those for a player following the proposed protocol, by also maximizing over all \( Z \in \mathcal{P}_{f}([n] \setminus \{i\}) \). (hypothesis 3 of Theor. 2, see also Def. 4). Line 13 computes the values in hypotheses 4 and 5 of Theor. 2.

Line 14 returns \( \text{FAIL} \) as soon as the hypothesis of Theor. 2 is satisfied. In such a case from Theor. 2 we know that the given mechanism is not \( \varepsilon \)-game-Nash and thus it is not \( f \)-Nash.

Line 15 (16) returns, \( \text{PASS} \) with \( \varepsilon \) (\( \text{PASS} \) with \( \varepsilon + \delta \)) when the hypothesis of thesis 1 (3) of Theor. 2 is satisfied. In such a case from Theor. 2 we know that the given mechanism is \( \varepsilon \)-game-Nash (\( \varepsilon + \delta \)-game-Nash).

Algorithm 1 Checking if a mechanism is \( \varepsilon \)-game-Nash

1: CheckNash(mechanism \( \mathcal{M} \), int \( f \), double \( \varepsilon \), \( \delta \))
2: for all \( i \in [n] \) do
3: Let \( k \) such that \( 4 E_{i}(k) < \delta \)
4: Let \( s \in \mathcal{S} \) and \( Z \in \mathcal{P}_{f}([n] \setminus \{i\}) \)
5: \( u_{i}^{0}(Z, s) \leftarrow 0; u_{i}^{0}(Z, s) \leftarrow 0; \)
6: for \( t = 1 \) to \( k \) do
7: \( v_{i}^{t}(Z \cup \{i\}, s) \leftarrow \max_{a_{i} \in A, a_{-i} \in A_{-i}} \left[ h_{i}(s, \langle a_{i}, a_{-i} \rangle) + \right. \)
8: \( + \beta_{i} v_{i}^{t-1}(s', Z \cup \{i\}) \]
9: \( BT(Z \cup \{i\}, s, \langle a_{i}, a_{-i} \rangle, s') \)
10: \( u_{i}^{t}(Z, s) \leftarrow \min_{a_{i} \in A, a_{-i} \in A_{-i}} \left[ h_{i}(s, \langle a_{i}, a_{-i} \rangle) + \right. \)
11: \( + \beta_{i} u_{i}^{t-1}(s', Z, s') \]
12: \( \Delta_{i} \leftarrow \max\{v_{i}^{t}(Z \cup \{i\}, s) - u_{i}^{t}(Z, s) \mid s \in I, Z \in \mathcal{P}_{f}([n] \setminus \{i\}) \} \)
13: \( \varepsilon_{1}(i) \leftarrow \Delta_{i} - 2E_{i}(k) ; \varepsilon_{2}(i) \leftarrow \Delta_{i} + 2E_{i}(k) \)
14: if \( (\varepsilon < \varepsilon_{1}(i)) \) return (\( \text{FAIL} \))
15: if \( (\forall i \in [n] (\varepsilon_{2}(i) < \varepsilon)) \) return (\( \text{PASS} \) with \( \varepsilon + \delta \))
16: else return (\( \text{PASS} \) with \( \varepsilon \))

X. Implementation

We implemented Algorithm 1 within the NuSMV [46] model checker. Here we briefly describe the main ideas bridging the gap between Algorithm 1 and its NuSMV implementation.

First of all, we extended the SMV language so as to be able to define mechanisms. We should keep in mind that, once we have verified that the proposed protocol is a Nash equilibrium for the given mechanism, then we will undertake a standard CTL verification in order to check that the proposed protocol satisfies the desired safety and liveness properties. Accordingly, we confined most of our extensions to the SMV language inside SMV comments so that mechanism models can also be used for standard CTL verification again with NuSMV.

As a second step we implemented Algorithm 1 using Ordered Binary Decision Diagrams (OBDDs) [10] resting on the CUDD [18] OBDD package (which is also the one used in NuSMV). Note that all functions in Algorithm 1 depend only on boolean variables, namely those representing the actions of Byzantine agents.

From Algorithm 1 we see that we only have two kinds of functions: \( b_{2} \) functions, taking boolean and returning a boolean value, and \( b_{2} \) functions, taking boolean and returning a real value. As usual \( b_{2} \) functions can be effectively represented using OBDDs. As for \( b_{2} \) functions we used the Arithmetic Decision Diagrams (ADDs) available in the CUDD package. ADDs are designed to represent and efficiently manipulate \( b_{2} \) functions returning reals represented as \( c \) 64 bit double. All arithmetical operations on ADDs used in Algorithm 1 are available in the CUDD package. The only ones that we had to implement ourselves were the \( \max \) and \( \min \) functions on ADDs. We developed them with a suitable traversal of the ADD to be minimized (or maximized). See [42] for a full description of our symbolic implementation of Algorithm 1.

XI. Experimental Results

In order to assess effectiveness of our Nosh verifier we present experimental results on its usage on an \( n \) player mechanism \( \mathcal{M} \) generalizing the mechanism presented in Section IV. This is a meaningful and scalable case study that well serves our purposes.

A. Mechanism Description

We are given a set \( \mathcal{J} = \{0, \ldots m - 1\} \) of \( m \) jobs and a set \( \mathcal{T} = \{0, \ldots q - 1\} \) of \( q \) tasks. Function \( \eta : \mathcal{J} \rightarrow \mathcal{P}(\mathcal{T}) \) defines for each job \( j \) the set of tasks \( \eta(j) \) needed to complete \( j \).

Each agent \( i \in [n] \) is supposed to work (proposed protocol) on a given sequence of (not necessarily distinct) tasks \( T_{i} = (\tau(i, 0) \ldots \tau(i, \alpha(i) - 1)) \) starting from \( \tau(i, 0) \) and returning to \( \tau(i, 0) \) after task \( \tau(i, \alpha(i) - 1) \) has been completed. An agent may deviate from the proposed protocol by delaying execution of a task or by not executing the task at all. This models many typical scenarios in cooperative services.

An agent incurs a cost by working towards the completion of its currently assigned task. Once an agent has completed a task it waits for its reward (if any) before it considers working to the next task in its list. As soon as an agent receives its reward it considers working to the next task in its list.

A job is completed if for each task it needs, there exists at least one agent that has completed that task. In such a case, each of such agents receive a reward. Note that even if two (or more) agents have completed the same task, all of them get a reward.

Agent \( i \) can be modeled as follows. Its set of states is \( X_{i} = Y_{i} \times Z_{i} \), where: \( Y_{i} = \{0, \ldots (\alpha(i) - 1)\} \) and \( Z_{i} = \{0, 1, 2\} \). State variable \( y_{i} \) ranges on \( Y_{i} \), state variable \( z_{i} \) ranges on \( Z_{i} \).

State variable \( z_{i} \) models the working state of agent \( i \). Namely, \( z_{i} = 0 \) if agent \( i \) is not working; \( z_{i} = 1 \) if agent \( i \) has done its assigned work (that is it has completed its currently assigned task); \( z_{i} = 2 \) if agent \( i \) currently completed task has been used to complete a job. In the last case agent \( i \) gets its reward for having completed a task used by a job. State variable \( y_{i} \) keeps track of the task currently assigned to agent \( i \). That is, \( y_{i} = p \) iff agent \( i \) is supposed to complete task \( \tau(i, p) \) in its sequence of assigned tasks.
Agent $i$’s set of initial states is $I_i = \{(0,0)\}$ whereas agent $i$’s set of actions is $A_i = \{0,1\}$ with variable $a_i$ ranging on it. Variable $a_i$ models agent $i$’s choices. Namely, $a_i = 1$ if agent $i$ will work, and $a_i = 0$ otherwise.

The state space of $M$ is $S = \{Y_1, Z_1, \ldots, Y_n, Z_n\}$. The set of initial states of $M$ is $I = \{I_1, \ldots, I_n\}$ We denote with $s$ the vector (of present state variables) $(y_1, z_1, \ldots, y_n, z_n)$ and with $a$ the vector (of action variables) $(a_1, \ldots, a_n)$.

Let $\Phi(i)$ be the set of pairs $(j, p)$ such that the $p$-th task of agent $i$ task sequence is needed for job $j$. Formally, $\Phi(i) = \{(j, p) \mid (p \in \{0, \ldots, \alpha(i) - 1\} \land (j \in J) \land (\tau(i, p) \in \eta(j))\}$.

Let $\Gamma(t)$ be the set of pairs $(w, r)$ such that the $r$-th task of agent $w$ task sequence is $t$. Formally, $\Gamma(t) = \{(w, r) \mid (w \in [n]) \land (r \in \{0, \ldots, \alpha(w) - 1\}) \land (t = \tau(w, r))\}$.

Let $\varphi_j(s)$ be a boolean function which is true iff job $j$ is completed. That is, if for each task $t$ in job $j$ there exists an agent $w$ such that $w$ has completed task $t$. Formally, $\varphi_j(s) = \land_{t \in \psi(j)} \lor (w, r) \in \Gamma(t) \land (w = 1)$.

Let $\gamma_i(s)$ be a boolean function which is true iff agent $i$ is currently assigned to a task needed for a currently completed job. Formally, $\gamma_i(s) = (\lor (j, p) \in \Phi(i) \land (\varphi_j(s)))$.

Finally, the underlying behavior $\mathcal{B}_i$ of agent $i$ is defined as follows: $\mathcal{B}_i(s, a_i, y_i, z_i) =$

- $((z_i = 0) \land (a_i = 0) \land (y_i = y_i) \land (z_i = z_i)) \lor$
- $((z_i = 0) \land (a_i = 1) \land (y_i = y_i) \land (z_i = z_i)) \lor$
- $((z_i = 1) \land \gamma_i(s) \land (z_i = 2) \land (y_i = y_i)) \lor$
- $((z_i = 1) \land \gamma_i(s) \land (z_i = 1) \land (y_i = y_i)) \lor$
- $((z_i = 2) \land (y_i = y_i + 1) \mod \alpha(i) \land (z_i = 0))$.

The proposed protocol $T_i$ for agent $i$ is $T_i(s, a_i) = (a_i = 1)$, that is agent $i$ is supposed to carry out the assigned task as soon as it can. Reward $h_i$ for agent $i$ is defined as follows:

$$h_i(s, a_i) = \begin{cases} 
-1 & \text{if } (z_i = 0) \land (a_i = 1), \\
+4 & \text{if } (z_i = 2), \\
0 & \text{otherwise}.
\end{cases}$$

### B. Experimental Settings

In order to run our Nash verification experiments we instantiate the above class of mechanisms as follows. First of all, we take the number of agents ($n$) to be greater than or equal to that of tasks ($q$). Second, we take the number of jobs ($m$) to be equal to the number of tasks ($q$). Third, we define $\eta(j)$ (i.e. the set of tasks needed to complete job $j$) as follows: $\eta(j) = \{(j, (j + 1) \mod q)\}$. That is, each job requires two tasks and each task participates in two jobs. We take as task sequence for agent $i$ the sequence $T_i = ((i - 1) \mod q, \ldots, q - 1, 0, \ldots, (i - 1) \mod q - 1)$. In other words, all agents consider tasks with the same order (namely $(0, \ldots, q - 1)$). The only difference is that agent $i$ will start its task sequence from task $(i - 1) \mod q$. For each agent $i$ we set, $\beta_i = 0.5$ and $\beta = \{\beta_1, \ldots, \beta_n\}$. With the above settings we have only two parameters to be instantiated: $n$ (number of agents) and $m$ (number of jobs).

### C. Experimental Results

Table XI-C shows our experimental results on verification of the $\varepsilon$-$f$-Nash property for the mechanism described in Sects. XI-A, XI-B. Column Byzantines in Table XI-C gives the number of Byzantine agents ($f$). In all experiments we take $\varepsilon = 0.01$ and accuracy $\delta = 0.005$. With such settings the value of $k$ in line 3 of Algorithm 1 turns out to be 15 in all our experiments. Column Nash in Table XI-C shows the result returned by Algorithm 1, namely, $\text{PASS}$ if the mechanisms is $\varepsilon$-$f$-Nash or $(\varepsilon + \delta)$-$f$-Nash, $\text{FAIL}$ otherwise. The meaning of the other columns in Table XI-C should be self-explanatory.

From Table XI-C we see that we can effectively handle moderate size mechanisms. Such mechanisms correspond indeed to quite large games. In fact, given a finite horizon $k$, an $n$ player mechanism can be seen as a game whose outcomes are $n$-tuple $\langle \sigma_1, \ldots, \sigma_n \rangle$ of strategies of length $k$, where $\sigma_i$ is the strategy played by agent $i$. If the underlying behavior of agent $i$ allows it two actions for each state, then there are $2^k$ strategies available for agent $i$. This would yield a game whose normal form has $k^{nk}$ entries. In the mechanism used in Table XI-C, even without considering Byzantine players, each agent can choose at least among $fib(k)$ (the $k$-th Fibonacci number) strategies (many more if we consider Byzantine players). With horizon $k = 15$ and $n = 8$ players this leads to a normal form game with at least $fib(k)^n = 610^8 \approx 10^{22}$ entries.

### XII. Conclusions

We present a symbolic model checking algorithm for verification of Nash equilibria in finite state mechanisms modeling MAD distributed systems. Our experimental results show the effectiveness of the presented algorithm for moderate size mechanisms. For example, we can handle mechanisms which corresponding normal form games would have more than $10^{22}$ entries.

Future research work include: improvements to the presented algorithm in order to handle larger mechanisms, verification of Nash equilibria robust with respect to agent collusions.

### Acknowledgments

This work has been partially supported by MIUR grant TRAMP DM24283 and by NSF grant CRS-PDOS 0509338.

### References


<table>
<thead>
<tr>
<th>Agents</th>
<th>Jobs</th>
<th>Byzantines</th>
<th>Nash</th>
<th>CPU (sec)</th>
<th>Mem (MB)</th>
<th>Max BDD</th>
<th>Present</th>
<th>Action Bits</th>
<th>Model Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>PASS</td>
<td>1.49e+01</td>
<td>7.99e+01</td>
<td>5.88e+05</td>
<td>15</td>
<td>5</td>
<td>4.99e+01</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>FAIL</td>
<td>2.39e+01</td>
<td>7.41e+01</td>
<td>5.42e+05</td>
<td>15</td>
<td>5</td>
<td>4.99e+01</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>PASS</td>
<td>2.09e+02</td>
<td>8.18e+01</td>
<td>6.15e+05</td>
<td>18</td>
<td>6</td>
<td>6.90e+01</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>FAIL</td>
<td>2.99e+02</td>
<td>8.60e+01</td>
<td>4.35e+05</td>
<td>18</td>
<td>6</td>
<td>6.90e+01</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>4</td>
<td>PASS</td>
<td>1.86e+03</td>
<td>7.77e+02</td>
<td>5.02e+05</td>
<td>24</td>
<td>8</td>
<td>1.85e+03</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>4</td>
<td>FAIL</td>
<td>1.14e+04</td>
<td>2.17e+02</td>
<td>5.85e+05</td>
<td>24</td>
<td>8</td>
<td>1.85e+04</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>3</td>
<td>PASS</td>
<td>1.91e+04</td>
<td>1.85e+03</td>
<td>2.29e+07</td>
<td>28</td>
<td>/</td>
<td>2.46e+02</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>4</td>
<td>FAIL</td>
<td>2.26e+04</td>
<td>2.88e+03</td>
<td>5.69e+07</td>
<td>28</td>
<td>/</td>
<td>2.46e+02</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>2</td>
<td>PASS</td>
<td>8.05e+04</td>
<td>4.88e+03</td>
<td>3.53e+07</td>
<td>32</td>
<td>8</td>
<td>3.18e+02</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>3</td>
<td>N/A</td>
<td>&gt;1.2e+04</td>
<td>&gt;1.81e+03</td>
<td>&gt;3.13e+07</td>
<td>32</td>
<td>8</td>
<td>3.18e+02</td>
</tr>
</tbody>
</table>


