Specification Faithfulness in Networks with Rational Nodes

Jeffrey Shneidman, David C. Parkes
Division of Engineering and Applied Sciences,
Harvard University,
33 Oxford Street, Cambridge MA 02138
{jeffsh, parkes}@eecs.harvard.edu

ABSTRACT

It is useful to prove that an implementation correctly follows a specification. But even with a provably correct implementation, given a choice, would a node choose to follow it? This paper explores how to create distributed system specifications that will be faithfully implemented in networks with *rational* nodes, so that no node will choose to deviate. Given a strategyproof centralized mechanism, and given a network of nodes modeled as having *rational-manipulation* faults, we provide a proof technique to establish the *incentive-, communication-, and algorithm-compatibility* properties that guarantee that participating nodes are faithful to a suggested specification. As a case study, we apply our methods to extend the strategyproof interdomain routing mechanism proposed by Feigenbaum, Papadimitriou, Sami, and Shenker (FPSS) [7], defining a faithful implementation.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity; J.4 [**Computer Applications**]: Social and Behavioral Sciences—*Economics.*; I.2.11 [**Distri buted Artificial Intelligence**]:

General Terms

Algorithms, Design, Economics.

Keywords

Distributed Algorithmic Mechanism Design, Computational Mechanism Design, Rational Manipulation, Algorithm Compatibility, Communication Compatibility, Incentive Compatibility, Failure Models, Rational Failure

1. OVERVIEW

This paper considers how to create provably faithful specifications that are implemented on networks of *rational* nodes. In these networks, a node acts in *self-interested* fashion to better its outcome in some distributed mechanism.

It is not hard to find evidence of rational behavior in existing distributed systems. Internet users can game their TCP settings to obtain better service at the expense of others [27]. Users cheat in distributed computations in order to drive up their "contributed computation" credit [15]. There is interesting work documenting the "free rider" problem [1] and the "tragedy of the commons" [12] in a data centric peer to peer setting.

This behavior can be classified as a type of failure, which should stand independently from other types of failures in distributed systems and is indicative of an underlying incentive problem in a system's design when run on a network with rational nodes. Whereas traditional failure models are overcome by relying on redundancy, rational manipulation can also be overcome with design techniques such as *problem partitioning*, *catch-and-punish*, and *incentives.* In such a network one can state a strong claim about the *faithfulness* of each node's implementation. This claim of faithfulness, like traditional distributed systems claims (e.g: *safety* and *liveness* [35]), is made with particular assumptions about the knowledge available to network participants. Typical distributed systems knowledge assumptions include node failure characteristics: for instance, a given specification might state safety properties on the assumption that no *link failures* will occur in the network. The knowledge assumptions particularly relevant to our scenario are drawn from economics and known as *equilibrium concepts.* This paper focuses on (and justifies) the use of *ex post* Nash (without collusion) as a reasonable knowledge assumption. The *ex post* Nash solution concept does not require nodes to have any knowledge of the private information of other nodes, but *does* assume that nodes are rational utility-maximizers and model other participants as such.

When one can prove that a specification will be faithfully followed by rational nodes in a distributed network, one can certify the system to be *incentive-*, *communication-*, and *algorithm-compatible* (IC, CC and AC). Such a system is provably robust against rational manipulation.

We introduce a general decomposition proof technique, that splits a distributed algorithm into disjoint phases, each of which can be proven IC, CC, and AC by showing that a node cannot benefit from any combination of deviation from the specification relevant to that phase. To ensure that one phase is disjoint from the next, a phase is *certified* before the subsequent phase begins.[1]

To demonstrate this decomposition technique, we modify a well-defined lowest cost interdomain routing problem proposed by Feigenbaum, Papadimitriou, Sami, and Shenker

PODC'04, July 25–28, 2004, St. Johns, Newfoundland, Canada.
Copyright 2004 ACM 1-58113-802-4/04/0007 ...$5.00.

[1] This decomposition technique is similar in spirit to offline compositional reasoning proof techniques in distributed systems [18], where automata are examined and proved correct in order to extrapolate correctness of the overall system.

(FPSS) [7], to create a specification that is *ex post* Nash incentive-, communication-, and algorithm-compatible. Unlike the original FPSS, we do not assume that nodes will be faithful in their computation or message passing.

2. RELATED WORK

The ideal prerequisites and reference reading for this paper are found in the distributed systems and algorithmic mechanism design literature. A "short list" in distributed systems would include a paper characterizing failure models [29], as well as an introduction to specifications and their proof techniques [35]. The economics subfield of *mechanism design* (MD) studies how to build systems that exhibit good behavior *in equilibrium*, when self-interested nodes pursue self-interested strategies. Readers unfamiliar with the fundamentals of mechanism design may wish to seek out a concise introduction [24, 13].

Algorithmic mechanism design (AMD) [21, 22] assumes centralized decision making, with nodes reporting complete private information to a center, but seeks to make the central computation tractable. *Indirect mechanisms*, on the other hand, provide an increased computational role to nodes, but still remain largely centralized in that nodes are directly connected to a center and are limited to what we term "information-revelation" actions [24].

Distributed algorithmic mechanism design (DAMD) [7, 8] considers MD in a network setting with no center, and distributes computation across the self-interested nodes. DAMD is full of new challenges since one can no longer assume an obedient networking and mechanism infrastructure where rational players control the message paths and mechanism computation. In concluding their seminal work on AMD, Nisan and Ronen noted the "set of problems" that come in implementing a mechanism in a network [23], suggesting that cryptography and distributed payment handling be considered. Feigenbaum et al. [7] identify the problem in DAMD as "the need to reconcile the strategic model with the computational model."

Our work attempts a comprehensive treatment of rational manipulation in distributed systems, and provides a framework and a way of reasoning about faithful behavior in mechanisms. Specifically, we are concerned with bringing all aspects of the distributed algorithm itself into an equilibrium. In a companion paper focused on problems in distributed AI [25], we consider a variant on the model in this paper in which there is still a center and trusted communication with nodes, but in which the goal is to off-load as much of the computation as possible onto nodes. Some of the general principles (the *partition-*, *information-revelation*, and *redundancy* principles) in that work also prove useful when dealing with fully-distributed implementations on networks. The specific contribution here is to provide general proof techniques and to extend the ideas to apply to *networks* without a center.

A number of research projects can be viewed as foreshadowing aspects of achieving general algorithm faithfulness. Some TCP research has focused on modifying the specification and introducing *obedient participants* to bring faithful computation into line [27]. Other work has assumed trusted communication (or a totally connected communication graph), but no trusted entity to perform computation and made heavy use of cryptography [3]. In mobile networks, some work has looked at achieving faithful message passing in resource-constrained environments [36, 14], through the use of *payments* and penalties. Message passing has also been studied in the context of an auction over a peer to peer network, with a centralized obedient auctioneer but nodes that may wish to drop or change bids from their neighbors [20].

3. RATIONAL MANIPULATION

Imagine that a designer specifies a *leader-election algorithm* to select a computation server in a network whose nodes are distributed across many administrative domains. The winner of this leader election is responsible for running some CPU-intensive task. The designer wants the most powerful node to be selected and specifies an algorithm where each node is to submit its true computation power and then come to a distributed consensus as to which node should be leader. The designer provides a correct implementation, but is dismayed to find that in practice, the protocol fails to elect the most powerful node. What has gone wrong?

In this toy election problem, it is possible that nodes (representing users) do not want to participate faithfully in the distributed algorithm. By truthfully revealing a node's computational power and following the distributed election protocol, a node is in danger of being tasked with a cpu-intensive chore that would take resources away from local jobs. The selfish administrator of that node might like to change the election-protocol code to execute something other than the code provided by the system designer.

Researchers have previously characterized the nuances of node failure according to observed behavior and failure remedy [28, 17, 5, 9]. Into the traditional taxonomy that ranges from Failstop to Byzantine, it is appropriate to introduce *rational manipulation* as a class of system failure. Extending the typical distributed systems failure taxonomy to include rational manipulation is justified for several reasons:

- The Internet is already showing evidence of rational manipulation in algorithms that were not designed to handle this type of failure. An anecdotal list appears in previous work [30].
- The behavior of a node that is deviating from a specification for selfish reasons would currently be classified in distributed systems failure taxonomies as a subset of Byzantine behavior. However, rational failures are *predictable* and *motivated* because a node will only manipulate in order to increase its own utility in the mechanism. This provides new opportunities for designing against failure, through tools such as incentives and careful problem partitioning.
- It is either suboptimal, or impossible, to use Byzantine Fault Tolerance (BFT) techniques to build systems robust to rational-manipulation failure [30]. BFT requires minimum levels of obedient connectivity and computation to work [18], whereas we might want to design systems in which *every* participant is capable of rational manipulation. BFT algorithms can be suboptimal in the sense that they require a large processing overhead.

3.1 Modeling Node Behavior

Before defining a distributed mechanism specification we need a language for specifications. This language will also make clear the range of behaviors available to a node. We find it useful to consider a mechanism specification expressed in terms of behaviors generated by *state machines* [35]. A state machine SM consists of the following components:

1. A set L of *states*, a subset of which are *initial states.*

2. A set $A = \{IA, EA\}$ of actions, of which set IA are *internal actions* and set EA of *external actions.*

3. A set T of state transitions of the form (s, a, s') where s and s' are states in L and a is an action in A.

Given this state machine SM, a specification $s : L \rightarrow A$, defines an action $s(l) \in A$ for each state $l \in L$. A node's *state* captures all relevant information about its role in a mechanism. For instance, the state will include received messages, partial computations, private knowledge about itself, and derived or estimated knowledge about other nodes and the world. *External* actions in a distributed computational system represent actions with some external effect; these actions generate a message to one or more neighbors. These messages can represent the results of calculations, messages forwarded from other nodes, or simply contain information about this node. *Internal* actions are those that do not generate a message. Internal actions can eventually cause an external action to occur.

3.2 Traditional Mechanism Design

State machines are a good way to describe a mechanism specification. Traditional mechanism design (MD), however, often assumes that the only actions available to a node are information revelation actions. Here, a node is allowed to provide input into a *center*, which is often described as a function from everybody's information revelation to some mechanism outcome.

In mechanism design language, consider a system with nodes, $i \in \mathcal{I}$. There are N nodes altogether. Traditional MD considers an implementation problem, in which nodes have private information $\theta_i \in \Theta_i$ (often referred to as the *type* of a node) and the goal is to implement an outcome $f(\theta) \in \mathcal{O}$ with useful properties (as defined by the designer), from a set of feasible outcomes $\mathcal{O}$. A node's type defines all relevant information that pertains to the outcome decision, as well as capturing information about its preferences for different outcomes. Notation $\theta_{-i} = (\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_N)$ denotes the type vector without node i.

A *centralized* (often called a *direct-revelation*) mechanism, $M = (f, \Theta)$ asks nodes to report types $\hat{\theta} \in \Theta = \Theta_1 \times \ldots \times \Theta_N$ to a trusted obedient center that then selects the outcome $f(\hat{\theta})$. Nodes need not be truthful, but are instead modeled as game-theoretic utility-maximizers (rational), with a *utility function*, $u_i(o; \theta_i) \in \mathbb{R}$, that induces a preference ordering s.t. $u_i(o_1; \theta_i) < u_i(o_2; \theta_i)$ implies that node i prefers o_2 to o_1. Incentives are provided so that node i chooses to report $\hat{\theta}_i = \theta_i$ in equilibrium.

In this model, the center can collect type information from nodes without interference from other nodes, and then compute the outcome, report the outcome to the nodes, and finally enforce the outcome. But, in problems such as leader election there is no trusted center and we need to involve the self-interested nodes themselves in computing, communicating, and enforcing the outcome of the mechanism.

3.3 Distributed Mechanism Specification

Distributed mechanism design should encompass actions beyond private information revelation. Rather than discussing a node's reported type $\hat{\theta}_i$, it makes sense to talk of a node's strategy $s(\theta)$, which captures how it behaves in all states of the world. Rather than an outcome rule $f(\theta) \in \mathcal{O}$, that depends on the reported type information, we now must speak of an outcome rule $g(s(\theta)) \in \mathcal{O}$ that depends on the sequence of actions taken by a node. We now present the distributed state machine description in an alternate form, in which the actions, states, and transitions are subsumed by strategy and outcome function.

Definition 1. ***Distributed mechanism specification****,* $d_M = (g, \Sigma, s^m)$ *defines an* outcome rule*, g, a feasible* strategy space $\Sigma = (\Sigma_1 \times \ldots \times \Sigma_N)$*, and a* suggested strategy*,* $s^m = (s_1^m, \ldots, s_N^m)$.

It is helpful to think of the suggested strategy $s_i^m \in \Sigma_i$ as the algorithm that the designer would like node i to follow. Strategy s_i^m is conditioned on the type θ_i of a node, with $s_i^m(\theta_i)$ defining the specification of the action that node i with type θ_i should take in each state (within the state-machine model of node behavior). The feasible strategy space, Σ_i, places no constraints on internal actions but can constrain external actions. Outcome rule, $g(s(\theta)) \in \mathcal{O}$, describes the outcome when nodes follow strategy s and have types θ.

Suggested strategy s_i^m decomposes into three strategies, $s_i^m = (r_i^m, p_i^m, c_i^m)$, with *information-revelation* strategy r_i^m, *message-passing* strategy p_i^m, and *computational* strategy c_i^m. Each sub-strategy is responsible for generating one of three kinds of external actions (those corresponding with the sub-strategy), which we formally define in Section 3.4. Formally, we can model this as each strategy simulating the entire specification, $s_i^m(\theta_i)$, but only performing its corresponding external actions. Notice that because only one action is taken in each state, no pair of sub-strategies will engage in multiple external actions simultaneously.

Given a distributed mechanism specification we are interested in understanding whether nodes have any incentive to deviate from the suggested actions.

3.4 Action Classification

The leader election example shows the three components of a strategy: *information revelation*, in providing an input to the election algorithm; *message passing* between nodes; and *computation*, in following the consensus algorithm. We now provide a formal classification of the external actions in a mechanism specification.

Information Revelation. A node may be asked to reveal information about its private type, such as its computational power in the leader-election setting. We also extend the notion of type to include the concept of *semi-private* type information. The type θ_i in traditional MD is most usually viewed as private information to a node. Alternatively, some information can be *common knowledge* to all nodes or some subset of nodes. In distributed systems it is useful to define the notion of *semi-private* type information. Semi-private type information exists when some subset of the type of a node is known to at least one other node, but not to all nodes. A good example is network topology: we can imagine that the link between node A and node B is common knowledge to both nodes, while other type information (such as node transit cost) remains private, and other nodes need not know about the existence of this link.

Definition 2. *External actions $r_i \in EA$ are **information-revelation** actions when the only effect is to reveal consistent (perhaps partial and untruthful) information about a node's type to some other node(s).*

Here, we use *consistent* to mean that there is a single type $\hat{\theta}_i$ that would have sent these messages in the suggested specification.

We can provide a careful definition of the information-revelation actions in the suggested specification.[2] Suppose for exposition that all nodes except node i follow the suggested specification. Then, the information-revelation actions for node i in $s_i^m(\theta_i)$ are those for which any deviation on any subset of the actions will do no more than implement an outcome that would be selected by following the suggested specification for some (perhaps untruthful) type $\hat{\theta}_i$. Formally, $g(s', s_{-i}^m(\theta_{-i})) = g(s_i^m(\hat{\theta}_i), s_{-i}^m(\theta_{-i}))$, for all s' that deviate from $s_i^m(\theta_i)$ only in information-revelation actions, and for all θ_i and all θ_{-i}. Thus, information- revelation actions provide a node with no more power to manipulate than that available to a node in a centralized mechanism.[3]

Message passing. A node may be asked to pass messages as part of the mechanism computation. For instance, if two nodes are communicating over a logical direct connection, there may still be rational nodes acting as part of the physical underlay.

Definition 3. *External actions $p_i \in EA$ are **message-passing** actions when the only effect is to send a message, received from another node, to one (or more) neighbors.*

Computation. A node may be asked to take part in a mechanism calculation. For instance, a node in the lowest-cost interdomain-routing setting can be asked to perform part of a distributed computation to determine the lowest-cost path, or to determine payments.

Definition 4. *External actions $c_i \in EA$ are **computational actions** when the action can affect the outcome rule used in the distributed mechanism specification (and when the action is more than simple message-passing).*

Computational actions have a wider effect than simply forwarding a message or revealing type information. Unlike information-revelation actions, these computational actions introduce opportunities for a node to affect the outcome of a mechanism that do not exist in centralized mechanisms! By definition, a computational action is one for which there is at least one deviation from the suggested specification that will implement an outcome that would **not** be selected by $g(s^m(\hat{\theta}_i, \theta_{-i}))$ for *any* report $\hat{\theta}_i$ by the node.

3.5 Knowledge Assumptions

A formal definition of rational manipulation requires that we are clear about the knowledge assumptions that we, as designers, make of nodes in a system. In an economic context, these knowledge assumptions must support the *equilibrium solution concept* that is adopted to model the behavior of rational nodes.

In traditional MD, it is common to design for a *dominant-strategy* equilibrium. Recall that a traditional mechanism $M = (f, \Theta)$ implements outcome $f(\hat{\theta}) \in \mathcal{O}$ based on reports $\hat{\theta}$, perhaps untruthful. A mechanism is *strategyproof* when truth-revelation is a *dominant-strategy* equilibrium.

Definition 5. *Centralized mechanism $M = (\Theta, f)$ is **strategyproof** if $u_i(f(\theta_i, \theta_{-i}); \theta_i) \geq u_i(f(\hat{\theta}_i, \theta_{-i}); \theta_i)$ for all θ_i, all $\hat{\theta}_i \neq \theta_i$, and all θ_{-i}.*

Strategyproofness is particularly useful because it makes an extremely weak knowledge assumption. Nodes need not know the types of other nodes, and nodes need not even believe that other nodes will be rational. In this work we adopt *ex post* Nash equilibrium as our solution concept, which requires correspondingly stronger knowledge assumptions.

Definition 6. *A strategy profile s^* is an **ex post Nash equilibrium** in distributed mechanism specification $d_M = (g, \Sigma, s^m)$ if s_i^* satisfies $u_i(g(s^*(\theta)); \theta_i) \geq u_i(g(s_i'(\theta_i), s_{-i}^*(\theta_{-i})); \theta_i)$, for all nodes, for all $s_i' \neq s_i^*$, for all types θ_i, and for all types θ_{-i} of other nodes.*

In an *ex post* equilibrium no node would like to deviate from its strategy even if it knows the private type information of the other nodes. Thus, as designers we can be agnostic as to whether or not nodes have any knowledge about the private type of other nodes. The main assumption when adopting *ex post* Nash is that the rationality of nodes is common knowledge amongst nodes. Although a stronger assumption than required for a strategyproof mechanism, we view this as a necessary cost in moving away from centralized computation on a trusted node. Nodes are now involved in implementing the rules of a mechanism, and it seems unlikely that arbitrary deviations by other nodes will still sustain the appropriate incentives for a node to behave faithfully.

The knowledge assumption in *ex post* Nash is still much weaker than that required in the more standard Nash equilibrium solution concept, which has received greater attention in recent literature on network games [26]. Adopting this notion of Nash equilibrium in our setting would require a node to have knowledge of other nodes' private information, which is usually unrealistic.

Remark 1. *We assume that nodes, although self-interested, are also benevolent in the sense that a node will implement the suggested strategy as long as it does not strictly prefer some other strategy. Thus, a weak ex post Nash equilibrium (in which a node can have other equally good best-responses) is considered sufficient for a faithful implementation.*

Remark 2. *A distributed mechanism may have multiple equilibria, but we are content to achieve implementation in but one of these equilibria. We agree with Brafman and Tennenholtz [2], that the fact that we are considering computational systems makes this assumption more palatable. The typical problem that arises with multiple equilibria is that of*

[2] Notice that we need to rule out the possibility that an agent can provide *inconsistent* information about its type, for example different information to different neighbors, because a deviation from any number of information-revelation actions must have no more effect than that of misreporting the agent's type and following the suggested specification.

[3] The definition excludes actions in which useful computation is also "smuggled" within the message, for example "solve problem P_1 (and report the solution) if your value is v_1 and solve problem P_2 (and report the solution) if your value is v_2." We classify these kinds of actions as *computational actions* when the solutions to P_1 or P_2 can change the outcome rule g, and not just the information that a node reveals during an implementation.

selection: how can nodes select the same equilibrium. By distributing an implementation of a suggested specification, it is reasonable to expect that some nodes will be obedient and follow the suggested specification. This acts as a correlating device, preventing other nodes from playing another equilibrium.

Remark 3. *Although truth-revelation may remain a dominant-strategy for nodes given that all nodes follow the suggested computational and message-passing actions, in equilibrium a rational node must also reason about whether or not other nodes will follow these suggested computational and message-passing actions. Thus, the equilibrium solution concept must adopt the "lowest-common denominator," which is ex post Nash in our model.*

3.6 Rational Manipulation

We use the term *rational node* to describe a node that attempts rational manipulation.

Definition 7. *A node exhibits **rational manipulation** if it fails to implement the suggested specification in an attempt to selfishly better its outcome in a distributed mechanism, given a particular knowledge assumption about other nodes.*

Formally, if $s_i^m \in \Sigma_i$ is the suggested strategy, and if s_{-i} is the set of strategies that node i *believes* all other nodes will follow, then a node exhibits *rational manipulation* if it follows some alternate strategy $s_i \neq s_i^m$ for which $u_i(g(s_i, s_{-i}); \theta_i) > u_i(g(s_i^m, s_{-i}); \theta_i)$.

Nodes are actively working to change the actions, transitions, and states in both their internal state machine and in the state machines of other nodes (through the effect of messages sent to these nodes) for selfish reasons.

We are seeking specifications with the following property:

Definition 8. *Distributed mechanism specification $d_M = (g, \Sigma, s^m)$ is an (ex post) **faithful implementation** of outcome $g(s^m(\theta)) \in \mathcal{O}$ when suggested strategy s^m is an ex post Nash equilibrium.*

3.7 Useful Properties: IC, CC and AC

We introduce *communication- and algorithm compatibility* as ways of describing mechanisms tolerant to rational manipulation. We also extend the idea of incentive compatibility, found in the mechanism design literature, to allow for incremental information-revelation.

Each statement can be defined for a particular equilibrium concept, that itself must be justified by a knowledge assumption. To keep these definitions concrete we adopt *ex post* Nash, which seems to be useful when considering distributed implementations of strategyproof mechanisms.

Definition 9. *A distributed mechanism specification $d_M = (g, \Sigma, s^m)$ is **incentive compatible** (IC) when there exists an ex post Nash equilibrium in which node i cannot receive higher utility by deviating from the suggested information-revelation strategy $r_i^m(\theta_i)$, for all nodes i and all types θ.*

Most commonly, the suggested information-revelation strategy for a node will expect the node to reveal *truthful* information about its private type through communication with other nodes. IC means that a rational node will choose to follow these actions.

Definition 10. *A distributed mechanism specification $d_M = (g, \Sigma, s^m)$ is **communication compatible** (CC) when there exists an ex post Nash equilibrium in which node i cannot receive higher utility by deviating from the suggested message-passing strategy $p_i^m(\theta_i)$, for all nodes i and all types θ.*

CC means that a rational node will choose to participate in the suggested message-passing actions within the distributed-mechanism specification.

Definition 11. *A distributed mechanism specification $d_M = (g, \Sigma, s^m)$ is **algorithm compatible** (AC) when there exists an ex post Nash equilibrium in which node i cannot receive higher utility by deviating from the suggested computational strategy $c_i^m(\theta_i)$, for all nodes i and all types θ.*

AC means that a rational node will choose to participate in the suggested computational actions within the distributed-mechanism specification. Properties IC, CC and AC are required for a faithful distributed implementation. Moreover, IC, CC and AC are *sufficient* for a faithful implementation:

Proposition 1. *A distributed mechanism specification $d_M = (g, \Sigma, s^m)$ in which suggested strategy $s^m = (r^m, p^m, c^m)$ is IC, CC and AC in the **same** ex post Nash equilibrium is a faithful implementation of outcome $g(s^m(\theta)) \in \mathcal{O}$.*

Proof. IC, CC and AC provide for the existence of an equilibrium in which nodes will follow suggested information-revelation r_i^m, and similarly for message-passing p_i^m and computation c_i^m. To achieve faithfulness we simply need that there is an equilibrium that achieves each one of these simultaneously. □

3.8 Strong AC and Strong CC

This section provides a general proof method to demonstrate specification faithfulness in networks with rational nodes. We define *strong*-AC and *strong*-CC, and show that together with the strategyproofness of the corresponding centralized mechanism, *strong*-AC and *strong*-CC provide IC, and in turn a faithful implementation.

We reduce the problem of proving (*ex post* Nash) faithfulness to that of:

1. Demonstrating that the corresponding centralized mechanism is strategyproof.

2. Strong-CC: a rational node should *always* follow its suggested message-passing strategy (whatever its information-revelation and computational actions).

3. Strong-AC: a rational node should *always* follow its suggested computational strategy (whatever its information-revelation and message-passing actions).

In fact, we will further break-up the proof, by advocating a further decomposition into disjoint mechanism phases.

Definition 12. *A distributed mechanism specification $d_M = (g, \Sigma, s^m)$ is **strong-CC** if a node cannot receive higher utility by deviating from the suggested message-passing actions $\hat{c}_i$, **whatever** its computational and information-revelation actions, when other nodes follow the suggested specification.*

Definition 13. *A distributed mechanism specification $d_M = (g, \Sigma, s^m)$ is **strong-AC** if a node cannot receive higher utility by deviating from the suggested computational actions $\hat{p}_i$, **whatever** its message-passing and information-revelation actions, when other nodes follow the suggested specification.*

Together, strong-CC and strong-AC rule out any useful joint deviations in which a node changes its communication, computational, and its information-revelation actions to gain an advantage.

Proposition 2. *A distributed mechanism specification $d_M = (g, \Sigma, s^m)$ is a **faithful** implementation of outcome $g(s^m(\theta))$ when the corresponding centralized mechanism is strategyproof and when the specification is strong-CC and strong-AC.*

Proof. To prove that the specification is an *ex post* Nash equilibrium we first assume that every node except i is following suggested specification, s^m_{-i}. By strong-CC and strong-AC, node i will follow the suggested message-passing and computation actions. (Notice that we can rule out joint deviations of both message-passing and computation actions). To prove IC, we can now assume that all nodes follow suggested communication- and message-passing actions. Let $f(\theta) = g(s^m(\theta))$ denote the outcome rule in the corresponding centralized mechanism. By definition of information-revelation actions, the space of possible outcomes becomes $g(s^m_i(\hat{\theta}_i), s^m_{-i}(\hat{\theta}_{-i}))$, but $g(s^m_i(\hat{\theta}_i), s^m_{-i}(\hat{\theta}_{-i})) = f(\hat{\theta}_i, \hat{\theta}_{-i})$, and $u_i(f(\theta_i, \hat{\theta}_{-i}); \theta_i) \geq u_i(f(\hat{\theta}_i, \hat{\theta}_{-i}); \theta_i)$ for all $\hat{\theta}_{-i}$, all θ_i, and all $\hat{\theta}_i \neq \theta_i$ by strategyproofness of $g(s^m(\theta)) = f(\theta)$. □

Remark 4. *In applying Proposition 2 one must be careful to ensure that actions labeled as "information-revelation" within the suggested specification satisfy the technical requirement of consistent information-revelation which can require consistency checking.*

Remark 5. *Distributed implementations of mechanisms introduces a new issue not encountered in traditional centralized MD, which is that the outcome computed by nodes must be enforced (we call this the "execution phase" in the interdomain routing example). Typically, the execution actions that correspond with the outcome must themselves be shown to be strong-AC and strong-CC. In interdomain routing, this means that nodes choose to follow lowest-cost paths and choose to respect payments.*

3.9 A General Proof Technique

For faithful adherence to a specification, we need to demonstrate strong-CC and strong-AC as well as the consistency of information-revelation actions. The following approaches are useful to this end:

Break Into Phases. A distributed mechanism can be decomposed into *disjoint phases*, each of which is proven strong-CC and strong-AC without worrying about joint deviations involving actions in other phases. Phases are separated during runtime with *checkpoints* where some node(s) certify a phase outcome and start a subsequent phase. One must be sensitive to the added computational and communication complexity in using checkpoints. This decomposition technique is powerful because it can allow an exponential reduction in the number of joint manipulation actions that must be checked in a faithfulness proof.

Tools for Strong-AC, Strong-CC and Consistency. Strong-AC, strong-CC, and information-revelation consistency can be achieved within a phase through the use of various techniques. **Payments** can be used to avoid untruthful information revelation, and also to provide incentives for nodes to perform faithful computation and message passing. **Redundancy** can be powerful, with computational and message-passing actions that deviate from a specification detected and penalized (via **catch and punish**). Catch and Punish can also be used without redundancy, when a subset of "checker" nodes can combine forces to completely monitor the behavior of a third node. Another technique is **problem partitioning**. At one extreme, partitioning could mean mean running the mechanism on nodes that cannot benefit from the mechanism outcome, with nodes split and one half computing the mechanism outcome for the other. More interestingly, we can also structure the computation so that a node is not involved in a calculation where it has a vested interest in the outcome [25]. Finally, **cryptography** can be used to make deviations from a specified algorithm detectable [3], and to make it impossible to rationally change a message, which can be useful for communication compatibility [20].[4]

4. EXAMPLE: INTERDOMAIN ROUTING

The remainder of the paper will focus on building a distributed mechanism specification that is faithful. The specification extends an interdomain routing distributed mechanism created by Feigenbaum, Papadimitriou, Sami, and Shenker (FPSS) [7]. FPSS is the first research to combine mechanism design ideas with a common Internet algorithm (the Border Gateway Patrol (BGP) interdomain routing protocol). The importance of this section is to show how various techniques can be used to prove strong-CC and strong-AC in a real system.

4.1 FPSS Interdomain Routing

The Internet is composed of many separate domains known as *autonomous systems* (ASs) such as Harvard, Berkeley, Micrososft, etc. Each AS can be modeled as a rational node. The goal in FPSS is to maximize network efficiency by routing packets along lowest cost paths (LCPs) for various traffic source-destination pairs.

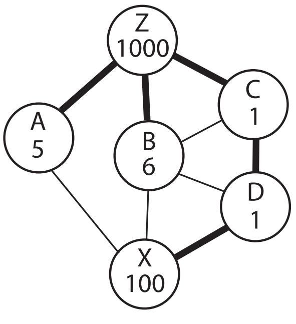

Figure 1: LCPs from Z.

Each node incurs a per-packet *transit cost* for transiting traffic on behalf of other nodes. The cost represents the additional load imposed by external traffic on the internals of an individual node. It costs nothing for a node to transit a packet originating or terminating at that node.

For instance, Figure 1 shows a network with the LCPs from Z to all other nodes drawn with bold lines. Numbers are the per-packet transit node costs incurred by each node. Assuming that the numbers in this figure represent true transit costs, the total LCP cost of sending a packet from X to Z is 2; the cost of sending a packet from Z to D is 1. The cost of sending a packet from B to D is 0 since there are no transit nodes between B and D.

To compensate a transit node for its routing services, each

[4] The problem with cryptography is the cost: if a system relies heavily on this technique, computation and communication complexity can become prohibitive.

transit node is given a payment for carrying traffic. FPSS observes, however, that *"under many pricing schemes, a node could be better off lying about its costs; such lying would cause traffic to take non-optimal routes and thereby interfere with overall network efficiency."*

Example 1: In Figure 1, path X-D-C-Z is the lowest cost path between X and Z; if C declared a cost of 5, X-A-Z would become the X to Z LCP. C can benefit from this manipulation, even if it loses the X to Z traffic, if it can make up the financial loss with higher payments received by transiting D to Z traffic. This has damaged overall efficiency - packets from X to Z are now being routed over a path whose true cost is higher.

FPSS seeks a pricing scheme that is dominant strategy incentive compatible (*strategyproof*), meaning that nodes can do no better than to declare their true transit costs. They achieve this by using a Vickrey-Clarke-Groves (VCG) mechanism [34, 11, 6] where transit nodes are paid based on the utility that they bring to the routing system plus their declared cost. The FPSS algorithm is distributed; lowest cost paths (LCPs) and pricing tables are computed by each node using information from neighbors in an iterative calculation. Following the abstract model of the Border Gateway Protocol (BGP) proposed by Griffin & Wilfong (GW) [10], FPSS assumes a static environment. FPSS also assumes a biconnected graph to ensure that the VCG payments well-defined. The abstract model of GW is extended to add additional information to the local state stored at nodes and to messages sent.

We find it useful to describe FPSS in terms of **construction** and **execution** phases. A construction phase deals with mechanism set-up, while execution deals with actual usage. Each node maintains three types of data for the mechanism construction phases:

- **[DATA1] Transit cost list.** Contains this node's knowledge about declared transit costs of other nodes in the network.
- **[DATA2] Routing table.** Each entry in this table contains the shortest path from this node to each destination, along with a ordinal representing the aggregate path cost.
- **[DATA3] Pricing table.** This sparse table contains the per-packet payment to be made by this node to each transit node on the shortest path, for each destination.

In the *first construction phase* the transit-cost information [DATA1] is constructed. In the *second construction phase*, routing and pricing tables are computed and stored as [DATA2] and [DATA3]. Nodes relay any changes of local data to their neighbors. Neighbors, in turn, update their local data with this new information and propagate changes to their neighbors. This continues until the information converges. Nodes can then use the mechanism to route traffic, and FPSS enters the *execution phase.* FPSS uses one additional type of data for mechanism execution:

- **[DATA4] Payment list.** Contains the amount of money that this node owes others for having originated traffic that traversed those transit nodes.

Each node is expected to use the pricing table correctly to calculate and store the payments that it owes transit nodes. This payment is compensation for requiring those nodes to transit packets originated locally. FPSS suggests that this list of total payments ([DATA4]) can be reported to an accounting and charging mechanism, that we call a *bank.*

This specification could be formalized with a state machine. The external actions already in the original FPSS would be included in this state machine, as follows: declaring the transit cost and providing connectivity information are *information-revelation* actions; relaying other nodes' transit-cost announcement are *message-passing* actions; and updating and forwarding routing and pricing tables are *computation* actions. An additional computation action comes in a node's reporting payments to the bank, and message passing actions exist for those nodes on the path to the bank. Message-passing is also used during execution for routing along LCPs.

4.2 Extending the FPSS Specification

In FPSS there is nothing to prevent nodes from rationally manipulating routing and pricing tables, or lying about connectivity information in the construction phase, or reporting inaccurate payments during execution (indeed, this was not their research goal). In addition, other nodes are in a position to intercept and selfishly modify payment tallies sent to the bank. In building a faithful specification based on FPSS, the key challenge is to prove strong-CC and strong-AC. (The corresponding centralized mechanism is already strategyproof for transit-cost declarations, and manipulations of the semi-private connectivity information will become apparent in the routing table calculations.)

In unpublished work, Mitchell *et al.* [19] have explored the use of cryptographic signing between routing nodes to ensure truthful connectivity declarations, and have suggested that this technique could extend to mechanism computation as well. In our presentation, we will instead favor redundancy, catch and punish, and problem partitioning, and bring the *entire* specification— message-passing and computation and execution —into equilibrium. We, too, are forced to rely on a small amount of cryptographic signing (in our case, to ensure communication compatibility) but the use of the three other techniques makes this signing requirement small. Given certain network topologies, it may be possible to eliminate signing altogether.[5]

We introduce a new role for nodes, that of *checker nodes.* These checker nodes perform redundant computation, which creates the opportunity for a catch-and-punish scheme that provides incentives for rational nodes to be faithful. The assignment of the checker nodes is very important: every neighbor of a node is assigned as a checker for that node. The node that is being checked is known as the *principal*, to refer to its role in the core distributed algorithm. Every node in the biconnected network plays the role of both a principal node and a checker node for all of its neighbors.

[5] Two examples when this may happen: The first example is when the network of rational nodes is an overlay that runs on top of an obedient underlay, similar to how many peer to peer applications work today. Here, certain messages to nodes outside of the mechanism (such as a bank) can follow an obedient overlay. The second example is when in a network of rational nodes a node is able to establish a path to some endpoint where all intermediate nodes are guaranteed to be partitioned from any information contained in the message. A special case of this scenario assumed by some previous work [3], is when the communication graph is fully connected, and therefore there are no intermediate nodes that could have an interest in the message.

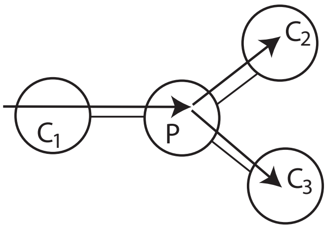

Figure 2: A computation or message passing deviation by Principal node (P) can be checked by checker nodes.

The checker nodes execute a redundant computation that mirrors what the principal is computing, and must receive a complete set of the messages received by the principal. Even though some checkers rely on the principal to forward these messages, there is always at least one checker that will catch any attempted deviation from the suggested specification. Ultimately, we rely on the bank to compare state-information reported by the principal and checker and penalize nodes for any deviation.

Our bank goes beyond (in FPSS language) "whatever accounting and charging mechanisms [that] are used to enforce the pricing scheme." In our specification, the bank is a trusted and obedient entity that can also perform simple comparisons, and enforce penalties when it detects a problem. The way that problems are detected is phase specific. In the construction phases, this penalization takes the form of not allowing the mechanism to progress to the next phase. In the execution phase, this penalty is a well-defined monetary unit that is epsilon-above the attempted deviation.[6] All communication between the bank and a node is signed with acknowledgments to ensure communication compatibility of these messages.

As an example, Figure 2 illustrates the role of checker nodes C_1, C_2 and C_3 in monitoring principal P. Each checker node is asked to perform the internal computation of P based on the copies it receives of P's messages. First, one should establish that the checker will follow this algorithm in equilibrium with a faithful P. This can be argued in our FPSS setting through partitioning— a checker cannot individually benefit from allowing a deviation by P. Second, suppose P is supposed to forward m to nodes C_2 and C_3 to allow them to replicate its calculations, but deviates and forwards the message as m'. Although this might change the view that C_2 and C_3 have of P's input, checker C_1 was on the incoming path of m and still has the correct view. Also, P cannot deviate in its calculation without deviating in its message-passing because the checkers will perform the correct calculations and when a check is made of internal state there will be a discrepancy.[7]

4.3 Faithfulness Proofs

In this section, we show how a faithfulness proof can be constructed for our extension to FPSS. The entire proof cannot be presented due to space constraints. We opt instead to show strong-AC, strong CC and consistent information revelation for the second construction phase and leave the rest to the extended version of this paper [33].

For each phase, we must show strong-AC, strong-CC, and consistent information revelation irrespective of a node's behavior in other phases of the mechanism. Once this is shown for each phase, one can use Proposition 2 to show that the entire mechanism is faithful. We assume that every node wishes to make progress in the mechanism, and indeed has a strong negative value when a construction phase does not progress. We further assume that the checkpointing node responsible for halting one phase and "green-lighting" the next phase is the bank node. Keeping with the FPSS specification, we assume a static network in these proofs.

Given strong-AC and strong-CC, the first construction phase terminates, and terminates with common transit cost tables [DATA1] across all nodes. The goal in the second-construction phase is to establish correct routing tables ([DATA2]) and pricing tables ([DATA3*]). Finally, the goal in the *execution phase* is to ensure that packets are sent along the LCPs and that correct network usage logs are recorded and correct payments collected. Our extensions to FPSS add more information to [DATA3]:

- **[DATA3*] Pricing table (extended).** This sparse matrix of per-node prices is the same as in [DATA3], but in addition, it is important to store an *identity tag.* This tag identifies the node that triggered the most recent FPSS pricing table update. (In the case of a pricing tie, this tag field actually contains the union of the nodes that suggested the same pricing entry.)

This second-construction phase uses problem partitioning, node redundancy, and catch-and-punish for strong-AC and strong-CC. It needs *no* cryptography in intra-node messages, but for CC between a node and the bank, we do assume that messages to and from the bank are signed.

Checker nodes completely surround a principal in the network, and act as a clone of the principal in all computation respects. The difference between the principal and the checker is that the checker does not send outputs of computations to neighbors. It is critical to understand the role and limitation of these checkers: the checkers perform the "heavy lifting" of checking a computation, but do not actually catch manipulation problems; this task is left to the checkpointing bank. In this phase, a deviation in calculating routing or pricing tables results in the bank not proceeding to the execution phase.

We can divide the tasks in this phase into actions taken by a [PRINC]ipal, actions taken by the principal's [CHECK]er nodes, and the actions taken by the [BANK].

- **[PRINC1] On receiving routing table update from neighbor:** *Message Passing:* Forward message to all checkers. *Computation:* Recompute LCPs based on new information; send recomputed LCPs as a routing table update to all neighbors.
- **[PRINC2] On receiving pricing table update from neighbor:** *Message Passing:* Forward message to all checkers. *Computation:* Recompute pricing tables based on new information; update tag information for every

[6] The bank is *not* as powerful as the traditional mechanism center. It does not actually perform the distributed mechanism computation, and instead checks results performed by others. The complexity of bank operation is described in the longer technical report [33]. It is an open problem to design a distributed bank that runs on the same network of rational nodes.

[7] Furthermore, the principal cannot report its true routing table but follow another routing table later because checks are made during the execution phase against the (correct) routing tables in checker nodes.

changed pricing entry to reflect source of change; send new pricing tables to all neighbors.

- **[CHECK1] When the principal forwards a routing update:** *Computation:* Verify that declared LCP is correct with local cost information. Re-run the LCP routing update. Check for consistency between the checker node's own LCP (acting as a principal) and the LCP stored on behalf on the foreign principal.
- **[CHECK2] When the principal forwards pricing tables:** *Computation:* Ignore messages with identity tags that are not checker nodes of the principal; Re-run the pricing table computation.
- **[BANK1]** At a network quiescence point, ask all principals and checkers for routing table information [DATA2] (a hash of the entire table is sufficient) and check for a deviation (difference). If there is a deviation, then signal all nodes to restart this phase. Otherwise, run [BANK2].
- **[BANK2]** Ask all principals and checkers for pricing table information [DATA3*] (again, a hash is sufficient) and check for a deviation. If there is a deviation, then signal all nodes to restart this phase. Otherwise, "greenlight" the execution phase.

In fact, the original FPSS formulation already exhibits limited problem partitioning. The price-update rules are specified in a way that prevents a node from increasing its incoming payment through changing the pricing messages.[8] However, problem partitioning alone cannot ensure strong faithfulness properties. There remain the following possible manipulations, which must be considered jointly:

1. A node can (in [PRINC1]) drop, change, or spoof (create) forwarded routing table update messages.
2. LCPs can be miscomputed (in [PRINC1]) and new LCP updates can be dropped, changed, or spoofed.
3. A node can (in [PRINC2]) drop, change, or spoof forwarded pricing table update messages.
4. Pricing tables can be miscomputed (in [PRINC2]) and new pricing table updates can be dropped, changed, or spoofed.

The goal of these manipulations is either to increase incoming payment from other nodes, or decrease the outgoing payment due to other nodes. It is important that at the end of this phase, the correct LCPs are reflected in the routing tables, and the pricing tables are accurate and refect the correct per-message prices as defined by the FPSS algorithm. We show that this phase is strong-AC and strong-CC. There is no revelation of private (transit cost) information, and deviations in revelation of *semi-private* connectivity information revelation are protected against though strong-AC and strong-CC (because such a deviation would require not forwarding messages along some link or not performing appropriate updates to internal LCP tables).

First, observe that the suggested specification for checkers in this phase is strong-AC and strong-CC. No checker wants to deviate if the other nodes (in their role as principals and checkers) are faithful, because deviation would cause the phase to be restarted.

For any principal-destination pair, one checker must be on the shortest path from principal to destination. That checker has a correct view of the cost of that path (in its other role as a principal in the network), because each node has a local transit cost table by the end of the first phase. Assume the behavior specified in [CHECK1]. Now, all checkers ignore LCP information that is not judged correct through their local transit cost table (known as a principal). The result is that a principal has no way to successfully change the LCP information stored by every checker. Any routing table deviation shows up by comparing a hash of [DATA2] between the principal and its multiple checkers. This means that manipulations (1-2 above) are caught by [BANK1].

Manipulations in the pricing table information (3-4 above) are more subtle, since here a checker node does not have an innate ability to verify messages based on internal information that it already has as a principal. Also notice that while problem partitioning ensures a node has no reason to modify its newly created outgoing pricing update messages (since its utility is not affected by changes caused by these messages), a node might like to change the pricing table that it must use for its own originated traffic. To show how checking nodes make these manipulations detectable, assume the behavior specified in [CHECK2]. Now, consider a principal A and a pair of checker nodes B and C. First, dropping a pricing-table message received from B (*ie.* not forwarding to C) will result in an inconsistency between B and C and therefore a restart by the bank. The same argument holds for changing an incoming message. More difficult is the third case, where a principal can spoof a new fake message. However, this spoof will create an inconsistency in the identity tag information in [DATA3*]. This inconsistency will be caught by [BANK2].

To see strong-AC and strong-CC, and to also see why joint deviations are not possible, notice that the routing and pricing information are two disjoint sets that must be kept in agreement with all checker nodes. A deviation is potentially useful *only* if the principal can cause every checker to deviate in a way that generates the same routing and/or pricing tables. By the arguments above, this is not possible since each manipulation has a disjoint side effect with every other manipulation. Any such deviation would cause a restart.

Theorem 1. *The extended FPSS specification is a faithful implementation of the VCG-based shortest-path interdomain routing mechanism.*

Proof. By Proposition 2, and since all phases of this specification are strong-AC, strong-CC, and have consistent information-revelation, and since the corresponding centralized mechanism is strategyproof. □

5. DISCUSSION

While this paper concerns rational manipulation, it has taken a fairly narrow view of rationality. For example, certain nodes may make *worsening the outcome of other nodes* the main goal besides maximizing their own utility. In the real world, companies are willing to take a short-term loss to drive competitors out of business. Such *anti-social* behavior has been previously characterized [4]. Other nodes might act maliciously, where their rational behavior is described as bringing down a system. imply introducing other failures, such as general omissions or even failstop, may cause

[8] In Section 6 of FPSS, notice that while a pricing update message has the potential to trigger a series of pricing table updates on various nodes, each of these nodes *ignores* (by the pricing update rules) the node that caused the update.

the system to falsely detect and punish manipulation. Further work needs to explore how other failure models affect faithfulness in systems with the rational-manipulation failure model.

6. ACKNOWLEDGMENTS

Thanks to Joan Feigenbaum, Steven Gortler, Rahul Sami, Margo Seltzer, Danni Tang, and anonymous reviewers for their useful comments. Thanks to Jim Waldo for casting rational manipulation in the role of a canonical system failure. Invaluable feedback on early versions of this work was received from poster session participants at EC-03 [32] and SOSP-03 [31], and by seminar participants at Stanford and MIT. This work is supported in part by NSF grants IIS-0238147 and ACI-0330244.

7. REFERENCES

[1] Eytan Adar and Bernardo Huberman. Free Riding on Gnutella. *First Monday*, 5(10), October 2000.

[2] Ronen Brafman and Moshe Tennenholtz. Efficient Learning Equilibrium. To appear, *Artificial Intelligence Journal*, 2004.

[3] Felix Brandt. A verifiable, bidder-resolved auction protocol. In *Proc. 5th Int. W. Deception, Fraud and Trust in Agent Societies*, pages 18–25, 2002.

[4] Felix Brandt and Gerhard Weiß. Antisocial Agents and Vickrey Auctions. In *Eighth Int. W. on Agent Theories, Architectures, and Languages (ATAL-2001)*, pages 120–132, 2001.

[5] Navin Budhiraja, Keith Marzullo, Fred B. Schneider, and Sam Toueg. Primary-backup protocols: Lower bounds and optimal implementations. Technical Report TR92-1265, 1992.

[6] E H Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

[7] Joan Feigenbaum, Christos Papadimitriou, Rahul Sami, and Scott Shenker. A BGP-based mechanism for lowest-cost routing. In *Proc. of the 2002 ACM Symp. on Princ. of Distr. Comput.*, pages 173–182, 2002.

[8] Joan Feigenbaum and Scott Shenker. Distributed Algorithmic Mechanism Design: Recent Results and Future Directions. In *Proc. 6th Int. W. on Discrete Alg. and Methods for Mobile Computing and Commun.*, pages 1–13, 2002.

[9] Juan A. Garay and Kenneth J. Perry. A continuum of failure models for distributed computing. In *W. on Distributed Algorithms*, pages 153–165, 1992.

[10] Timothy Griffin and Gordon T. Wilfong. An analysis of BGP convergence properties. In *SIGCOMM*, pages 277–288, 1999.

[11] Theodore Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

[12] Garrett Hardin. The Tragedy of the Commons. *Science*, 162:1243–1248, 1968.

[13] Matthew O. Jackson. Mechanism theory. In *The Encyclopedia of Life Support Systems*, EOLSS Publishers, 2000

[14] M. Jakobsson, J. Hubaux, and L. Buttyan. A micropayment scheme encouraging collaboration in multi-hop cellular networks. In *Proc. of Financial Crypto 2003.*, 2003.

[15] Leander Kahney. Cheaters Bow to Peer Pressure, 2001. `http://www.wired.com/news/technology/0,1282,41838,00.html`.

[16] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The Eigentrust algorithm for reputation management in P2P networks. In *Proc. 12th Int. World Wide Web Conf. (WWW)*, 2003.

[17] Leslie Lamport and Michael J. Fischer. Byzantine generals and transactions commit protocols. Technical Report Opus 62, Menlo Park, California, 1982.

[18] Nancy Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, 1996.

[19] J.Mitchell, R.Sami, K.Talwar and V.Teague. Private communication, December 2001.

[20] Dov Monderer and Moshe Tennenholtz. Distributed games: From mechanisms to protocols. In *Proc. AAAI-99*, pages 32–37, July 1999.

[21] Noam Nisan and Amir Ronen. Algorithmic Mechanism Design. In *Proc. of the 31st ACM Symp. on Theory of Computing*, pages 129–140, 1999.

[22] Noam Nisan and Amir Ronen. Computationally feasible VCG mechanisms. In *Proc. 2nd ACM Conf. on Elec. Commerce (EC-00)*, pages 242–252, 2000.

[23] Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Econ. Behavior*, 35:166–196, 2001.

[24] David C. Parkes. *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency (Chapter 2).* PhD thesis, Univ. of Penn., May 2001.

[25] David C. Parkes and Jeffrey Shneidman. Distributed Implementations of Vickrey-Clarke-Groves Mechanisms. To appear, *Proc. 3rd Int. Conf. on Autonomous Agents and Multiagent Systems*, 2004.

[26] Tim Roughgarden. Designing networks for selfish users is hard. In *Proc. 42nd Ann. Symp. on Foundations of Computer Science*, pages 472–481, 2001.

[27] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson. TCP congestion control with a misbehaving receiver. *Computer Communication Review*, 29(5), 1999.

[28] F.B. Schneider. Byzantine generals in action: Implementing fail-stop processors. *ACM Transactions on Computer Systems 2(2)*, 1984.

[29] Fred B. Schneider. What good are models and what models are good. In Sape Mullender, editor, *Distributed Systems*, chapter 2. Addison Wesley, 1993.

[30] Jeff Shneidman and David C. Parkes. Rationality and self-interest in Peer to Peer networks. In *2nd Int. W. on Peer-to-Peer Systems (IPTPS'03)*, 2003.

[31] Jeffrey Shneidman. Systems Must Address Rational Failure. Poster, *19th ACM Symp. on Operating Systems Princ. (SOSP '03)*, October 2003.

[32] Jeffrey Shneidman and David C. Parkes. Using redundancy to improve robustness of distributed mechanism implementations. In *Proc. Fourth ACM Conf. on Electronic Commerce (EC'03)*, 2003.

[33] Jeffrey Shneidman, David C. Parkes and Danni Tang. Specification Faithfulness in Networks with Rational Nodes. Technical report, Harvard University, 2004.

[34] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[35] William E. Weihl. *Specifications of Concurrent and Distributed Systems*, chapter 3. Addison Wesley, 1992.

[36] S. Zhong, Y. Yang, and J. Chen. Sprite: A simple, cheat-proof, credit-based system for mobile ad hoc networks. In Proc. *INFOCOM*, 2003.