1-1-2010

Dealing With Misbehavior In Distributed Systems: A Game-Theoretic Approach

Nandan Garg
Wayne State University,

DEALING WITH MISBEHAVIOR IN DISTRIBUTED SYSTEMS: A GAME-THEORETIC APPROACH

by

NANDAN GARG

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2010

MAJOR: COMPUTER SCIENCE

Approved by:

Advisor Date

DEDICATION

To LORD KRISHNA.

ACKNOWLEDGMENTS

I take this opportunity to thank my advisor, Dr. Daniel Grosu, for his continued guidance. He is an exemplary advisor and his patience is most commendable. He helped me and motivated me in all the phases of life. I am grateful to him for recommending me to the department as a teaching assistant and supporting me as a research assistant.

I thank Dr. Vipin Chaudhary, for supporting me in various ways. I am grateful to him for being on my dissertation committee, providing valuable guidance and his generous financial support. I would like to thank Dr. Weisong Shi and Dr. Cheng-Zhong Xu for serving on my dissertation committee and providing invaluable help.

I thank the Chair of the Department of Computer Science Dr. Farshad Fotouhi and the Graduate Committee for providing me with assistantship positions and various other resources throughout my doctoral studies at Wayne State University.

I would like to thank my parents, my sisters and my wife for encouraging and supporting me all along in all circumstances of my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Problem Description

In the recent years, we are observing a paradigm shift toward large-scale distributed systems. The distributed algorithms deployed in distributed systems are executed by the resources/clients which are generally owned and operated by multiple entities (called agents) ranging from individual users to global organizations. Because of their differing sizes and structure, they have different goals and objectives. The distributed systems allow agents to participate interactively and the outcome of the interaction depends on the actions of the participating agents. For selfish motivations, these agents can manipulate the protocols. Such agents are known as *selfish-agents*. For example let us consider routing where the individual routers are in the same network or in autonomous domains. The routers are expected to forward packets to other routers and they do so voluntarily because of the low cost. However, as the cost of communication is increasing such as in video transmissions and the bandwidth is to be reserved for providing different levels of quality of service (QoS), the routers may choose not to forward some or all of the packets. Such behavior is selfish and the socially desired outcome is hampered (in this case some users may observe unreasonable delay in transmissions or even part of the network can become disconnected).

There are various goals an agent may try to achieve by deviating from the expected behavior. Some agents may act with malicious intent and deviate from the specification to throttle the expectations of system designers or other participants. Some agents may seem to deviate just because of some technical limitation or incompatibility. Some agents may just want to maximize their gains by deviating from the protocol. This last class of agents also called rational agents are extensively studied in game theory. Game theory studies the different strategies a rational agent may want to pursue to achieve their goals.

Generally the participating agents are required to inform the algorithm about some of its parameters (representing its preferences) which are used to calculate an output. The algorithm designers assume that the participating agents will report their preferences truthfully and, thus, the calculated output will be optimal. However, these parameters are generally private to the agent and an agent may strategize to disclose some value other than the actual value to influence the outcome in its favor. Misreporting the private values can lead to a suboptimal outcome. Considering the same example of routing (specifically BGP), the routers are required to report their per-packet cost of carrying traffic to compute the lowest-cost path between two nodes. If a router misreports its cost, the computed path will not be optimal and the network efficiency will be lowered. *Mechanism design* (also known as *implementation theory*), uses game-theoretic tools to study how the revelation of individual preferences can be constrained so as to aggregate the outcome to a social choice. This mechanisms (algorithms) use incentives to motivate the agents to reveal their values truthfully such that the socially desired outcome can be achieved.

In the last few years researchers have used game theory to model and solve a large number of open problems in networks and distributed systems. The applications of game theory cover a wide range of areas such as peer-to-peer systems, routing, wireless networks, sensor networks, network formation, grids, task scheduling, multicast cost-sharing, load balancing in heterogeneous systems, artificial intelligence, e-commerce, and online auctions. In this dissertation we focus on distributed systems, scheduling and information security.

1.2 Contributions

The goal of our research is to use game-theoretic tools and techniques to model and mitigate the misbehavior of participants in distributed systems. Specifically, we are considering three problems from distributed systems. First, we consider the problem of how the cost of a multicast transmission is shared among the users receiving the transmission. The users may try to manipulate the mechanisms to gain unfair advantages. We use game theory to develop distributed mechanisms which are not susceptible to manipulation by the users. Second, we consider the problem of antisocial behavior in auction-based scheduling. We use game-theoretic techniques to characterize and prevent misbehavior in such scheduling scenarios. Third, we address the problem of developing more secure systems by using game theory. When attackers attack a network, they interact with the victim system to achieve their goals. We analyze these interactions and use classical techniques such as deception to prevent attacks.

More specifically the contributions of this dissertation are:

- We consider a well known cost sharing mechanism which suffers from the possibility of being manipulated by strategic agents. The cost sharing mechanism works well when a tamper-proof model is assumed, but when the nodes are autonomous, we show that the nodes can cheat by sending manipulated values when executing the protocol. We propos a mechanism which uses digital signatures and auditing by a trusted party to catch and punish any cheating node. We also implement the existing and our proposed protocols to compare their performance and their economic properties. We deploy these mechanisms on Planetlab and conduct extensive experiements. To our knowledge this is the first work to experimentally compare these cost sharing mechanisms.
- In an auction-based scheduling mechanism, autonomous nodes send their bids to a central mechanism, which computes a schedule. When antisocial agents participate in such mechanisms, they can manipulate their bids to inflict losses on the other participating agents. We characteriz the behavior of such antisocial agents by developing a strategy that can be used by an agent to inflict losses on the other agents. The scheduling mechanism we consider is based on the second price sealed bid auction. Our results also apply to general repeated Vickrey auctions with related items.
- We develop a model of the interaction between an attacker and a honeynet system. The honeynet system has a few regular nodes and a few honeypots. The attacker has some means of identifying the honeypots using probes. The system administrator wants to hide the honeypots by manipulating the responses to the probes. We use extensive form games to model the interactions and involve deception to achieve the

goal. This work is the first work to model a security game using extensive form games and deception.

1.3 Dissertation Organization

The dissertation is structured as follows. In Chapter 2, we review game theory, mechanism design and some related work. This chapter presents the foundations of this work and introduces the terminology used. In Chapter 3, we present the proposed faithful distributed mechanism for sharing the cost of multicast transmission and discuss our results from the experiments comparing different mechanisms. Chapter 4 investigates the antisocial behavior in an auction-based scheduling mechanism. In Chapter 5 we model the interaction between an attacker and a honeynet system using extensive form games and suggest deception strategies for the honeypots. We conclude in Chapter 6 with a summary of our work and future directions.

1.4 Bibliographical Notes

Some parts of this dissertation were previously published in peer-reviewed journals and conferences/workshops. Parts of Chapter 3 were presented at the 12th IEEE Symposium on Computers and Communications (ISCC-07) [58] and at the IEEE 21st International Conference on Advanced Information Networking and Applications (AINA-07) [59]. The AINA-07 paper received *The IEEE Outstanding Student Paper Award*. An extended version combining these two papers was published in IEEE Transactions on Parallel and Distributed Systems [60]. The ideas and results presented in Chapter 4 were initially published in [61] and presented at the 7th Workshop on Advances in Parallel and Distributed Computational Models (APDCM'05). An extended version of this paper was published in IEEE Transactions on Systems, Man, and Cybernetics - Part A [62]. Portions of Chapter 5 were published in [57] and presented at the IEEE SMC Information Assurance Workshop (IAW'07) at the United States Military Academy, Westpoint, NY. The paper presented at IAW'07 was nominated for the *Best Paper award*.

CHAPTER 2: BACKGROUND

In this chapter we present the fundamental concepts we use in the rest of the dissertation. We review the concepts and the terminology related to incentives, game theory and mechanism design, and the related work on application of game theory to distributed systems. We discuss the concepts which are closely related to the topics of this research. For a more detailed review of concepts we refer the readers to the book "Microeconomic Theory" by Mas-Colell *et al.* [96] which, as the title suggests, discusses in detail the Microeconomic theory which begins by considering the behavior of individual agents and builds up to the theory of aggregate economic outcomes. We specifically bring Chapter 23 of the book, titled "Incentives and Mechanism Design", to reader's attention. We refer the readers to the book "A Course in Game Theory" by M. Osborne and A. Rubinstein [112] for a detailed presentation of game theory. A recent book in this area is "Multi-Agent Systems" by Y. Shoham and K. Leyton-Brown, which presents the fundamental concepts in distributed optimization, game theory and learning [139]. "Algorithmic Game Theory" edited by N. Nisan *et al.* [109], is an excellent reference for game theory applications such as cost sharing, security, networking, etc. The book contains some introductory material as well as many different computer science problems modeled using game theory in the form of chapters contributed by the top researchers in the field.

2.1 Games

Game theory aims to help us in understanding situations in which decision makers interact. It provides models to study such situations and analyze whether the assumptions of the interactions are correct or not.

2.1.1 Strategic Game

A *strategic game* is a model of interacting decision makers. Formally a strategic game consists of:

- a set of players;
- for each player, a set of actions;
- for each player, preferences over the set of action profiles, modeled by a utility function.

However the game does not specify the exact actions players take. A solution is a systematic description of the outcomes that may emerge in a family of games. Game theory suggests reasonable solutions for classes of games and examines their properties. There are different types of games to model different situations, like cooperative games, non-cooperative games, extensive games with and without perfect information, coalitional games, games with imperfect information, etc.

Cooperative vs. Non-Cooperative Games

A game in which the players can make binding commitments is a *cooperative game* as opposed to a *non-cooperative game*, in which they cannot. Stating it little differently, a cooperative game is a game where the interactions are between groups of players ("coalitions") instead of between individual players as in the non-cooperative games. Different solution concepts are employed to different types of games. In cooperative games the main solution concepts involved are: the core, the kernel, and the Shapley value. In non-cooperative games the solution concepts are based on the players maximizing their own utility functions subject to stated constraints. The main solution concepts are Nash equilibrium, subgame perfect Nash equilibrium and bayesian equilibrium.

Symmetric vs. Asymmetric Games

A *symmetric game* is a game where the payoffs for playing a particular strategy depend only on the other strategies employed, not on who is playing them. If one can change the identities of the players without changing the payoff to the strategies, then a game is symmetric. Most commonly studied asymmetric games are games where there are no identical strategy sets for both players.

Zero sum vs. Non-zero Sum Games

In *zero-sum games* the total benefit to all players in the game, for every combination of strategies, always adds to zero (or more informally, a player benefits only at the expense of others). Most games studied by game theorists (including the famous Prisoner's Dilemma [17]) are *non-zero-sum games*, because some outcomes have net results greater or less than zero.

Simultaneous vs. Sequential Games

Simultaneous games are games where both players move simultaneously, or if they don't move simultaneously, the later players are unaware of the earlier players' actions (making them effectively simultaneous). *Sequential games* (or dynamic games) are games where later players have some knowledge about earlier actions.

Games with Perfect Information vs. Imperfect Information

A game is a *game of perfect information* if all players know the moves made by all other players before making decisions. Thus, only sequential games can be games of perfect information, since in simultaneous games not every player knows the actions of the others. Most games studied in game theory are imperfect information games. Perfect information is different from *complete information* where every player knows the strategies and payoffs of the other players but not necessarily the actions.

2.1.2 Solution Concepts for Non-cooperative Games

Every *agent* or *player* is allowed to choose a particular *action* a_i from a set of actions A_i available to the agent. The action of the agent is based on its type. Sometimes an action is also referred as *strategy*. An *action profile* is an ordered set $\mathbf{a} = (\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_n})$, of one action for each of the n agents in the system.

An *equilibrium* of a game is an action profile consisting of a best strategy for each of the n players of the game.

An *equilibrium concept* or *solution concept* is a rule that defines an equilibrium or outcome of a game with self-interested agents based on the possible strategy profiles, payoff functions and information available to agents about each other. Some of the important solution concepts like Nash equilibrium, dominant strategy equilibrium, and Bayesian-Nash equilibrium are described here.

Nash Equilibrium

A *Nash equilibrium* is an action profile a such that no player i can do better by choosing an action different from a_i, given that every other player j adheres to a_j [104]. This equilibrium corresponds to a steady state where no player wants to deviate, if everyone else adheres to their actions. There are different algorithms suggested to compute the Nash equilibrium. The most famous for 2-player games is the Lemke-Howson method [90]. For n-player games ($n > 2$) the algorithm based on Simplicial Subdivision was used until recently. A new algorithm was recently proposed by Govindan and Wilson [66]. In [117], Porter *et al.* proposed search based methods for both 2-person and n-person games and showed that they perform better than the existing algorithms.

Although the Nash solution concept is fundamental to game theory, it makes very strong assumptions about the agents' information and the beliefs about other agents. To play a Nash equilibrium in a simultaneous game, every agent must have perfect information about the preferences of every other agent and agents must all select the same Nash equilibrium. The calculation of Nash equilibrium is computationally intractable (even for $n = 2$) [46, 125]. Papadimitrou *et al.* introduced another complexity class called PPAD (Polynomial Parity Argument, Directed version) [114] which differs from NP in that problems in PPAD always have at least one solution, whereas that in NP may not have a solution. Daskalakis *et al.* [41] showed that computation of Nash Equilibrium for general sum finite games for more than 2 players is PPAD-complete. Chen and Deng [30] extended the result to show that this is true even for two-player games. Conitzer and Sandholm [38] presented a number of hardness results for finding the existence of Nash equilibria in Normal-form, Bayesian and Stochastic games.

Dominant Strategy Equilibrium

In a *dominant strategy equilibrium* every agent has the same utility-maximizing strategy, independent of the strategies of the other agents. Dominant-strategy equilibrium is a very robust solution concept, because it makes no assumptions about the information available to the other agents. In the context of mechanism design, dominant strategy implementations of social choice functions are much more desirable than Nash implementations.

Bayesian-Nash Equilibrium

In the *Bayesian-Nash equilibrium* every agent selects a strategy to maximize the expected utility in equilibrium with expected-utility maximizing strategies of other agents. This solution assumes that the agents have some beliefs about the distribution of types of other agents. Agent i does not necessarily select the best strategy against the actual strategies of the other agents. The Bayesian-Nash equilibrium makes more reasonable assumptions about agent information than the Nash equilibrium, but it is a weaker solution concept than the dominant strategy equilibrium.

There are some software tools available which help in understanding and analyzing the games and their solutions. GAMUT [110] is a test-suite that is capable of generating games from a wide variety of games found in the literature specially for testing game theoretic algorithms. Gambit [97] is a library of game theory software and tools for the construction and analysis of finite extensive and normal form games. It is a open source software portable across platforms. As mentioned earlier, computation of Nash equilibrium is PPAD-complete and the existing methods to find one of possibly many Nash equilibria, are at best inefficient. One of the ways to quickly compute sample Nash equilibrium is to develop parallel programs that run on several machines in parallel. Widger and Grosu [151] developed a parallel algorithm which uses vertex enumeration to find all Nash equilibria in two-player general-sum normal form games. For the same category of games, they designed a parallel algorithm based on enumerating all supports of mixed strategies [150]. The parallel processing allows to solve large games efficiently. For n-player noncooperative games Widger and Grosu [152] designed a parallel algorithm which uses support enumeration method to find all Nash equilibria.

2.1.3 Terminology

An *agent* is a decision maker (any entity) interacting in the system under consideration. Agents are autonomous and may represent a human being, an organization, a wireless device, a computer or a resource such as a router. The assumption underlying many models in game theory is that the agents are rational, that is to say that an agent chooses the best action out of all the available actions, based on his preferences. There is a finite number of agents $n \geq 1$ in a system denoted by $c_i, (i = 1, \ldots, n)$. Agents who care only about the expected return of an investment, and not the risk (variance of outcomes or the potential gains or losses) are called *risk neutral* agents.

Each agent has some information which is private to him. This information can be a single value or a set of values depending on the problem. The private value of Agent i known as his *type* is denoted by $t_i \in T_i$. In our routing example, the cost of forwarding a packet is private to the router, which will be its type. The agents must make a collective choice o from some set O of possible alternatives. In the mechanism design context o represents the *output* of the mechanism, which in a broad term represents the solution of the problem (say an optimization) under given constraints. The output is based on the actions taken by the agents. The preferences of an agent are given by the function called *valuation* $v_i(t_i, o)$. This is the quantification of the value an agent associates with a particular outcome o when its type is t_i.

As mentioned earlier, generally, the mechanism gives incentives to the agents to motivate them to reveal their preferences truthfully. These incentives are called *payments* where p_i denotes the payment received by agent i. A negative value of the payment means that the agent pays money to the mechanism. For example in an auction scenario where the agent is not providing the service but consuming the resource, the payment is made by the agent to the mechanism. We use $\mathbf{p}$ to denote the vector of payments $(p_1, p_2, \ldots, p_n)$.

The *Utility function* u, also referred as the *payoff function*, represents the preference relation of the agent. It quantifies the preferences of an agent such that if the agent prefers the outcome x over y then $u(x) \geq u(y)$. The goal of a rational agent is to maximize its utility.

A very common assumption in auction theory and mechanism design is that the agents have *quasi-linear utility* functions. A quasi-linear utility function has the following form: $u_i = v_i + p_i$. This model is characterized by the fact that the utility is additively separable into money and everything else and that it is linear in money. This implies that the change in utility of an agent (because of payment p_i) does not influence his valuation of the consumed good or service. This assumption has another important consequences, so this model is assumed in most of the current algorithmic mechanism design literature [108]. This type of utility function can also be thought of as the profit of the agent, where valuation represents the cost of providing a service (or participating) and the payment is the remuneration received from the mechanism. For example in a task scheduling scenario, an agent is executing tasks and receives payments from the mechanism for that. The valuation in such case is expressed as a negative value. This utility function can also be thought of as the profit of the agent, where valuation represents the cost of providing a service (or participating) and the payment is the remuneration received from the mechanism. For example in a task scheduling scenario, an agent is executing tasks and receives payments from the mechanism for that. The valuation in such case is expressed as a negative value.

2.2 Mechanism Design

Mechanism design aims to study how privately known preferences of many agents can be aggregated toward a social choice. The main motivation of this field is micro-economic and the tools are game theoretic. Game theory studies the strategies of the agents when they interact, however mechanism design aims to design algorithms such that the strategies played by agents are such that the outcome is globally beneficial. In our context, the mechanism uses incentives to influence the strategies selected by the agents.

Before defining the mechanism we need to define the social choice function toward which the agents' preferences are aggregated. A social choice function selects an outcome given the types of the agents. Formally, Social choice function (SCF) $f : t \rightarrow O$ chooses an outcome $f(t) \in O$ given $t = (t_1, t_2, \ldots, t_n)$. The mechanism design problem is to implement the "rules of a game", i.e., defining possible strategies and the method used to select an outcome based on agent strategies and to implement the solution to the social choice function despite agent's self-interest. A mechanism solves a given problem by specifying the strategies A_i available to an agent i , an output function and a payment function to influence the strategy selection of an agent.

A mechanism $M = (\mathbf{A}, \mathbf{g}, \mathbf{p})$ specifies:

(i) The strategy space $\mathbf{A} = (A_1, \ldots, A_n)$ where A_i is the set of strategies available to agent i;

(ii) An output function $o = g(\mathbf{A})$, such that $g(a)$ is the outcome implemented by the mechanism for strategy profile $a = (a_1, \ldots, a_n)$; and

(iii) A payment function $p_i = p(\mathbf{A})$ where p_i is the payment given to agent i.

We denote the mechanism by $M(\mathbf{g}, \mathbf{p})$ and let the strategy space $\mathbf{A}$ be implicit. This is a general model of a mechanism. There may be different variations such as randomized mechanisms, iterative mechanisms, etc. A mechanism is said to *implement* a social choice function if the output computed with equilibrium agent strategies is a solution to the social choice function for all possible agent preferences.

2.2.1 Properties of mechanisms

The properties of mechanisms are often defined in terms of the properties of the social choice function they implement. In the following we describe some properties of social choice functions.

Pareto-optimality

A social choice function is *Pareto-optimal* if no agent can be better off, without making another agent worse off.

Efficiency

A social choice function is *allocatively efficient* if the allocation maximizes the total value over all agents i.e., the sum of valuations of all the agents is maximum for all preferences.

Budget Balance

A social choice function is *budget balanced* if the sum of payments to all agents is zero, which means there are no net transfers into the system or out of the system. In other words, the revenue generated from the agents exactly balances the cost incurred by the mechanism. Weakly budget balanced mechanisms can generate a surplus but not a deficit. One important observation is that in general if the mechanism has huge surplus, it means that the agents are overcharged, which reduces the individual utility of the agents.

Individual Rationality

This property places constraints on the level of expected utility that an agent receives from participation. In a mechanism, which is individually rational, an agent can always achieve as much expected utility from participation as without participation, given prior beliefs about the preferences of other agents. This is also known as the voluntary participation constraint.

Direct revelation

A *direct-revelation mechanism* is a mechanism in which the only actions available to agents are to make direct claims about their preferences to the mechanism i.e., the agents' strategies are to report their type.

Incentive Compatibility

In an incentive compatible mechanism the incentives of agents are aligned with those of the group, i.e., the behavior that optimizes the utility of an individual agent also optimizes the utility of the group. Hence agents report truthful information about their preferences in equilibrium.

In the game theory literature it is proven that it is not possible to build a mechanism with all the three properties i.e, budget balance, incentive compatibility and Pareto optimality. Faltings [48] studied the properties of mechanisms and proposed a mechanism which is incentive compatible and individually rational but sacrifices Pareto-optimality for achieving budget balance. This is based on the argument that the lack of budget balance causes huge losses to the agents because the surplus generated cannot be redistributed to the agents, which greatly reduces their utility.

Strategyproofness

A mechanism is *strategyproof* (or dominant-strategy incentive-compatible) if truth revelation is the dominant strategy for all the agents. Strategyproofness is a very useful property, both game-theoretically and computationally. The dominant-strategy implementation is very robust to assumptions about agents, such as the information and rationality of agents. Computationally, an agent can compute its optimal strategy without modeling the preferences and strategies of other agents.

In their seminal papers, Vickrey [147], Clarke [33] and Groves [71], proposed the Vickrey-Clarke-Groves family of mechanisms, often simply called the VCG mechanisms, for problems in which agents have quasi-linear preferences. The VCG mechanisms are allocatively-efficient and strategyproof direct-revelation mechanisms. However VCG mechanisms are not budget balanced - so a surplus or deficit exists. VCG mechanisms can be applied only to problems where the objective functions are simply the sum of agent's valuations. Such objective functions are known as *utilitarian functions*.

A general framework for designing truthful mechanisms for optimization problems where the agents' private data is one real valued parameter was proposed by Archer and Tardos [14]. Their framework can be applied to designing mechanisms for optimization problems with general objective functions and restricted form of valuations.

Group-strategyproofness

In the above notion of strategyproofness, it is assumed that the agents do not collude. However, it is known that even in strategyproof mechanisms like VCG, agents can collude to increase their utility. A stronger notion is *group-strategyproofness*, also known as *coalitional-strategyproofness*. A bidding mechanism is group-strategyproof if for any group of bidders, an arbitrary combination of bids might result in a net increase in the utility of the group, but in that case some bidder in the group suffers a decrease in utility.

Schummer [132] studied a similar problem where an agent can give bribe to another agent to manipulate the outcome. He proposed a decision rule to eliminate this possibility, which is known as bribe-proofness. Bribe-proofness is a weaker concept than group-strategyproofness since it assumes that the group size is equal to two.

Goldberg and Hartline [63] studied the single parameter agents and give several results for group strategyproofness. They also introduced a stronger notion than group-strategyproofness and called it t-truthfulness, where any coalition of size t or less cannot increase their total utility with non-truthtelling. They also design approximate profit maximizing auctions, relaxing the incentive property to only hold with high probability.

2.2.2 Complexity of Mechanism Design

As discussed previously, the economic literature stresses incentives in studying the multi-agent systems, whereas the theoretical computer science focused on computation complexity. The paper by Nisan and Ronen on *algorithmic mechanism design* (AMD) [108], was the first to address both. They put forth a centralized model that combined the incentive compatibility with the computational tractability. They mention that the objective function of a VCG mechanism is required to maximize the objective function which is computationally intractable in most settings and replacing the optimal algorithm with non-optimal algorithm results in untruthful mechanisms. They proposed an n-approximation mechanism for task scheduling where n is the number of agents and also designed a randomized mechanism which performs better.

The complexity of mechanism design is also studied in [37] where the authors showed that in a deterministic mechanism, the problem of deterministically choosing an outcome on the basis of reported preferences is NP-complete. They designed more efficient randomized mechanisms, where the mechanisms stochastically chose an outcome which is solvable by linear programming.

2.2.3 Distributed Mechanisms

The mechanisms discussed above are centralized mechanisms, where one central entity is responsible for collecting the preferences of the agents, calculating the outcome and then disbursing the payments. This model is simple, but it has inherent limitations. First, the agent implementing the centralized mechanism may not be reliable. Secondly, in a networked scenario, communicating the preferences of agents to the central mechanism and then distributing the payments will induce burden on the network. Moreover, as discussed earlier, computing the optimal outcome is computationally intractable, so it is judicious to distribute the computation over the participating agents. A central mechanism also faces problems of scalability, which a distributed mechanism can alleviate. A distributed mechanism is also favored in support of the argument that the networks, especially Internet, are so widely distributed that any mechanism deployed on such system should resonate with the architecture of the system.

Feigenbaum, Papadimitriou and Shenker [52] extended the AMD model to distributed settings, preserving the incentive compatibility and computational tractability properties of the mechanism. The design of such mechanisms falls under the category of Distributed Algorithmic Mechanism Design (DAMD). Feigenbaum and Shenker studied DAMD in [54] in the context of two problems they have analyzed earlier namely Multicast cost sharing and Interdomain routing and provided insight on the recent research done in these areas as well as specified the open problems and future research directions.

Carroll and Grosu [26] proposed a distributed mechanism for task scheduling based on secret sharing primitives. Instead of a central authority, a set of distributed auctions run by distributed agents determine the allocation and payment, while protecting the anonymity of the losing agents and the privacy of their bid.

2.3 Related Work

In this section we discuss the application of incentives and mechanism design in different areas related to networking and distributed systems.

2.3.1 Task Scheduling and Load Balancing

Nisan and Ronen [108] studied the problem of scheduling tasks on unrelated machines when the machines are distributed and owned by selfish agents. They proposed a formal model for studying optimization problems and designed truthful mechanisms for task scheduling and for finding the shortest paths in a directed graph (which may very well represent a network). The n-approximation mechanism for task scheduling is known as the Minwork mechanism. They gave a lower bound for deterministic mechanisms and also gave a randomized mechanism which performs better than the lower bound. They also noted that the mechanisms like VCG require to maximize the objective function, which makes the mechanism computationally intractable. If a sub-optimal function is used, the truthfulness of the mechanism cannot be preserved.

They also extended the basic model where the mechanism can verify the declarations of the agents and distribute the payments to agents based on the actual values as compared to declared ones. They studied the task scheduling problem in the context of mechanisms with verification and gave an n-approximation mechanism called compensation and bonus mechanism. They also noted that these mechanisms are intractable being based on exponential time optimal allocation algorithms. They suggested a polynomial time n-approximation mechanism which however looses its truthfulness property. They pointed out many open problems from their work like considering solution concepts other than dominant strategy equilibrium, considering other settings like repeated games and considering other assumptions like partial verification. They also noted that in a distributed computing scenario, it will be more efficient to have a distributed mechanism rather than a centralized one for obvious reasons.

Archer and Tardos [14] designed a 3-approximation randomized mechanism for the problem of scheduling jobs on related machines to minimize makespan. They also studied the *frugality* of payments in [15] where truthful mechanisms can keep the total payment low, as compared to the shortest path mechanism proposed in [108], where in some cases the mechanism may have to pay $\Omega(n)$ times the cost of the shortest path even when there is an alternate path of similar cost. They showed that this behavior is intrinsic to the problem. However the mechanism suggested in [14] never pays more than a logarithmic factor from the expected costs incurred by the machines, provided no single machine dominates the processing power.

The mechanism in [14] is a randomized mechanism which is truthful in expectation only. Andelman *et. al.* [10] designed the first deterministic truthful mechanism which is a 5-approximation mechanism. They also developed a deterministic FPTAS when the number of machines is constant. For scheduling jobs on unrelated machines the most efficient monotone mechanism for minimizing the makespan at present has an approximation ratio of 3 [86].

A Distributed MinWork (DMW) mechanism was proposed by Carroll and Grosu [26, 27] which extends the Nisan and Ronen [108] Minwork mechanism to a distributed setting where the agents themselves are responsible for running the mechanism, while protecting the anonymity of the losing agents and the privacy of their bids by using a secret-sharing scheme.

Computational grids are large-scale computing systems involving geographically distributed resources owned by self-interested agents. These agents may manipulate the resource allocation algorithm in their own benefit, thus severely degrading the performance of the system. Since the resources are heterogeneous, load balancing in such systems is an important measure to achieve optimal throughput. Grosu and Chronopoulos [70] developed a truthful mechanism for solving the static load balancing problem in heterogeneous distributed systems. They provided experimental results to show the effectiveness of their centralized mechanisms. They mentioned developing a distributed algorithm for the problem as future work.

Grosu and Chronopoulos [69] also designed a truthful mechanism for fair load balancing which gives a fair and Pareto-optimal solution for the problem while satisfying voluntary participation property. They also developed a fair load balancing protocol FAIR-LBM which implements their mechanism. They showed that the payment scheme used in their mechanism is frugal.

2.3.2 Peer-to-peer Systems

A *Peer-to-Peer* (or *P2P*) computer network relies on the resources (computing power, bandwidth, disk, etc.) of the participants in the network rather than concentrating them in a relatively few servers. In a pure peer-to-peer network, there are no explicit clients and servers, rather equal peer nodes simultaneously act as both “clients” and “servers” to the other nodes on the network. Recently, Peer-to-Peer networks have become fairly common because of the excellent scalability, resilience, fault tolerance, and other properties exhibited by them as compared to the traditional client/server model. Some P2P systems like Napster use the client-server model for some tasks and the peer-to-peer structure for other tasks. Different P2P systems have been developed to provide different services like distributed file sharing, distributed computing, distributed database, etc. Some examples of famous P2P systems are Napster, Gnutella [122] and Freenet [34]. For a general overview of P2P systems we refer the reader to [2].

The P2P systems are based on the resources provided altruistically by the participating nodes, however the nodes incur some cost for sharing their resources like disk, processing power or bandwidth. Since the nodes participating in a P2P network are owned by selfish agents, they have enough motivation not to show the altruistic behavior. Such agents may not want to share their resources but only use the resources provided by other agents. This behavior is known as *free-riding*. It is a very acute problem in P2P systems because there are more users who want to utilize the resources then those who contribute resources. In [3], Adar and Huberman analyzed the traffic on Gnutella to find that about 70% of the users do not share files, and nearly 50% of all responses are returned by the top 1% of sharing hosts. They also argued that free riding leads to degradation of the system performance and adds vulnerability to the system.

Some of the incentive schemes proposed recently for overcoming such problems are presented in the following subsections.

Inherent Generosity

Research in behavioral economics has shown that purely self-interested models do not usually explain the observed behavior of people. That is to say that there are some users who gain more by altruistic behavior rather than by being self-interested. Such approach was used by Feldman et. al. [56] to devise a modeling framework that studies free-riding taking the user generosity into account. Users decide to free-ride or contribute based on their generosity compared to contribution cost. If the social generosity is below a certain threshold the system collapses.

Monetary Payments

In such schemes, there is an exchange of money for consuming and providing services. The monetary schemes allow rich and flexible economic mechanisms but they are deemed impractical because of need of reliable infrastructure for accounting and micropayments.

Golle, Leyton-Brown and Mironov [64] studied the problem taking Napster as an example, where the individual users are provided with no incentive for sharing their own files and thereby adding value to the network and so many users decline to perform this altruistic act. They introduced a game-theoretic model to study the problem, proposed two classes of novel payment mechanisms, and analyzed the user strategies and the resulting equilibria. They also validated their analytical results using a multi-agent reinforcement learning model.

It is also difficult to devise mechanisms for distributing monetary payments. Some schemes are proposed by researchers based on either a central trusted third party or decentralized payment schemes based on virtual currency.

Reputation-based Schemes

In reputation based schemes, every user is assigned a rank depending on its behavior in the system and this rank is used to make decisions. There are many different ways developed to assign a rank to the agents. The agents may have to maintain ranks or histories of other agents to assign rank to them. In *direct-reputation based schemes* a user decides the reputation of another user based only on the service provided by that user to him. In *indirect reputation based schemes*, the reputation is assigned to an agent based on its behavior to other agents also. Some of the issues in such reputation based schemes are:

- How reputations are assigned to new agents joining the system?
- How the mechanism deals with collusive behavior such as ballot stuffing (false praise)

or bad mouthing (false accusation) [44, 45]?

- How the mechanism deals with agents leaving and joining the system again to gain new reputations (white-washing) [56]?

Buragohain, Agrawal and Suri [24] provided a game theoretic framework for using incentives in P2P systems to provide a differential service to users, which means that peers that contribute more get better quality of service. They first considered homogeneous peers which means that all peers derive equal benefit from everybody else and showed that there are two equilibria. They also studied the stability property of the equilibria. Then they considered heterogeneous peers with arbitrary benefit function for each pair of peers and derived the Nash equilibrium which is impervious to probability function used to implement the differential service, perturbations like users leaving or joining the system and non-strategic or non-rational players. They also suggested practical ways to implement a differential service incentive scheme in a P2P system.

Jurca and Faltings [79] developed a simple reputation mechanism where the mechanism does not exclude users having negative feedback but allows them to provide service at a constrained price. They showed that the mechanism based on averaging past feedback and reputation based service level agreements can push the market toward an efficient equilibrium point. Their mechanism requires low memory to maintain reputations, it is resistant to failures, and it can be deployed in distributed mechanisms. However the mechanism is not resistant to collusion and malicious users and there may be more than one undesired equilibrium points.

Shneidman and Parkes [136] studied the problem of rational agents in P2P systems and proposed mechanism design as a tool to design such networks where rational nodes are participating. They also described three open problems in AMD/DAMD work which are relevant in P2P settings containing rational agents.

Feldman *et al.* [55] studied the free riding problem in P2P systems and attributed it to large populations, high turnover, asymmetry of interest and low cost of new identities. They proposed a robust and scalable game-theoretic incentive technique based upon a reciprocative decision function. They addressed the above mentioned challenges (large populations, etc.) along with collusion, false reports and presence of traitors by using schemes such as discriminating server selection, shared history, maxflow based subjective reputation, adaptive stranger policies and short-term histories. In this context, traitors are the users who acquire high reputation by cooperating for a while and then defect before leaving the system.

In a P2P system nodes are required to forward packets of another nodes, however a rational agent may decide to drop packets to conserve local bandwidth. To reduce such free-riding, Blanc and Vahdat [22] proposed a scheme based on game-theoretic tools to provide incentives to users to forward packets. They modeled a P2P network as a random-matching game and showed that a simple reputation system can sustain cooperation as a robust and subgame-perfect equilibrium. They also showed that this equilibrium can tolerate malicious nodes and noise in the system. They also quantified some of the design trade-offs like effectiveness of reputation system even if monitoring a small fraction of nodes. They also discussed some open problems like nodes cheating by sending requests in batches when reputation is high, tampering with reputations, heterogeneous nodes with different number of requests, collusion among nodes, forwarding requests incorrectly rather than just dropping them and retrying dropped requests.

Ranganathan *et al.* [80] modeled incentive schemes using Multi-person Prisoner's Dilemma (MPD) to study the effectiveness of different schemes encouraging sharing in distributed file sharing systems. They considered three such schemes and studied their effectiveness by means of simulation:

(i) Token exchange - price based with soft-incentives i.e., non-monetary tokens;

(ii) Peer-approved - Reputation based;

(iii) Service quality - Reputation based.

2.3.3 Networks

Current networks consists of distributed nodes communicating with each other through routers and other devices. Besides Internet, different types of networks have emerged due to recent technological advances, such as wireless ad hoc networks and sensor networks. All the networks require interaction between distributed resources which are generally owned by different organizations and individuals. These interactions require the nodes to follow particular protocols to provide different services. However, as mentioned earlier, these nodes will act rationally and may strategize to deviate from the protocol to gain advantage. As mentioned in the routing example earlier, mechanism design and game theory can be used to give incentives to these nodes to cooperate and follow the protocol. There are different scenarios in which nodes may interact. We discuss the different categories of interactions and mechanisms proposed to solve the underlying problems.

Routing

Roughgarden and Tardos [126] studied the problem of routing traffic to optimize the performance of a congested network. They considered a network where the rate of traffic between each pair of nodes and a latency function for each edge specifying the time needed to traverse the edge is given. They assumed that each network user routes its traffic on the minimum latency path available to it. They noted that such a selfishly motivated assignment will not minimize the total latency. They quantified the degradation in network performance due to such unregulated traffic. They studied the flow at Nash equilibrium and compared it to that of the optimal flow.

Calculation of shortest path is an important part in routing. Nisan and Ronen [108] designed a truthful mechanism for calculating the shortest path. Hershberger and Suri [72] solved the problem by studying an edge's utility, i.e., how much it lowers the length of the shortest path by studying the differences in the path lengths with and without the edge. They computed these marginal values for all the edges of network efficiently and thus solved an open problem posed by Nisan and Ronen, specifically by computing Vickrey prices for all the edges in the same asymptotic time complexity as a single shortest path problem.

Feigenbaum *et al.* [51] studied the problem of routing traffic between Internet domains, which is currently handled by the Border Gateway Protocol (BGP). They designed a mechanism to solve the inter-domain routing problem by extending the work of Nisan and Ronen [108] and Hershberger and Suri [72]. Their work differs from the previous work in three aspects:

(i) They treated the nodes as strategic agents, rather than links.

(ii) Their mechanism calculates the lowest-cost route for all source-destination pairs and payments for all transit nodes on all of the routes (rather than computing route and payment for one pair at a time).

(iii) They developed a distributed algorithm (as compared to centralized algorithm in other works) which is a straightforward extension of BGP.

Cost Sharing for Multicast Transmissions

One of the most effective way to transmit data, specially audio/visual, over a network to large number of users is an overlay multicast transmission. In such systems the users who receive the transmission pay for the cost incurred by data flowing over the links [157]. One important problem is how the fair share of the users receiving the multicast is calculated? A one-pass mechanism to implement an axiomatic allocation scheme was proposed in [73]. Feigenbaum *et al.* [53] considered the setting where the users may misreport their values and designed strategyproof mechanisms for computing the cost shares of users. They proposed two distributed mechanisms namely Marginal Cost (MC) and Shapley value (SH), possessing different properties, to calculate the cost shares. Although the SH mechanism is budget balanced and economically efficient, the worst case complexity of the mechanism is high. Archer *et al.* [13] designed an approximately budget balanced mechanism with exponentially lower worst cast complexity. They also characterize the groups that can act strategically to collude and cheat against the MC mechanism. Adler and Rubenstein [4] considered the availability of multiple rates of transmission and studied their effect on MC mechanism. In [50], the authors extended their previous work and gave lower bounds on the network complexity for a particular class of mechanisms.

Wireless Networks

In wireless ad hoc networks, nodes communicate with far off destinations using intermediate nodes as relays. Relaying a request certainly requires energy and since the nodes have limited energy, it is in the interest of an agent not to participate in relaying the request of other nodes. However such selfish behavior will defeat the purpose of the whole network, if all nodes act in such a way. Assuming that the nodes are rational and will respond to incentives, mechanism design and game theoretic tools can be used to provide incentives to the nodes to relay requests. Srinivasan *et al.* [142] assumed that the nodes have a minimum lifetime constraint and determine the optimal throughput that each node should receive, which is Pareto-optimal. They proposed a distributed and scalable algorithm called Generous Tit-For-Tat (GTFT) to decide whether to accept or reject a relay request. They showed that GTFT results in a Nash equilibrium and proved that the system converges to the optimal operating point. However they assumed that all the nodes employ GTFT to converge to Nash equilibrium. They also assumed that there are no malicious nodes and that the relay cost is the same for all the nodes, which they have left as open problems.

One important component of radio resource management in wireless communication is power control. With the increasing demand for wireless data services, it is necessary to establish power control algorithms for information sources other than voice. Saraydar *et al.* [131] presented a power control solution for wireless data in the analytical setting of a game theoretic framework.

Network Formation

Yuen and Li [158] studied the same problem as [142] but rather than selectively relaying the requests, they developed a distributed incentive mechanism that motivates each node toward a more desirable network topology. Their mechanism is capable to dynamically adapt to the varying parameters of the network and constraints of nodes (such as discharging of battery) to construct new topology periodically. They also developed a distributed algorithm to implement their solution, which converges toward globally optimal strategy. However, they considered the network connectivity as their objective rather than minimizing the cost of any single path, which could be an interesting future direction.

Sensor Networks

Agah *et al.* [6] proposed a game theoretic framework for defending nodes in a sensor network. They applied three different schemes for defense. In the first scheme they formulated the attack-defense problem as a two-player, nonzero-sum, non-cooperative game between an attacker and a sensor network. They showed that this game achieves a Nash equilibrium and thus leads to a defense strategy for the network. In the second scheme they used Markov Decision Processes to predict the most vulnerable senor node. In the third scheme they used an intuitive metric (nodes traffic) and protected the node with the highest value of this metric. They evaluated the performance of each of these three schemes, and showed that the proposed game framework significantly increases the chance of success in defense strategy for sensor network.

2.3.4 Security

As our dependence on information technology is growing, the importance of information security is growing. At the same time challenges in securing the information are also increasing. There are two ways how game theory is useful in the field of information security. The first is by studying and analyzing the economics of information security, which investigates topics such as: what are the incentives to organizations/people to make information secure. In [11], R. Anderson argued that the information security issues not only arise due to technical incapabilities, but also because of lack of incentives, network externalities and other economic misalignments. H. Varian [146] suggested that many websites suffered distributed denial of attacks because the owners of the compromised systems did not have the similar incentive to protect their systems as the websites which were attacked using the compromised systems. He created a game theoretic model to analyze how security decisions of individual users impact the system as a whole. Kearns and Ortiz [83] investigated the problem of computing the equlibria of interdependent security games where the overall security of the individual depends on the action choices of all other agents in the system.

Another successful application of game theory is to model and solve problems in information security by formulating the multi-agent decision problem as a game. Invariably in a security game, agents with discordant motivations interact with each other. Such problems can be naturally modeled using non-cooperative game theory. Sallhammar*et al.* [129] used stochastic games to evaluate security and dependability by computing the probabilities of expected attacker behavior where attacks represents transitions between states. Lye and Wing [94] also modeled the interaction between attacker and system administrator as a two player general sum stochastic game and compute Nash equilibria using non-linear programming. Alpcan and Basar [9] developed a formal decision and control framework for intrusion detection and presented two game-theoretic tehcniques. Liu*et al.* [93] presented a game theoretic way to infer the intent, objectives and strategies of attacker along with an incentive based method to model the problem. The work in [6] used non-cooperative game theory to provide security in a sensor network. Agah*et al.* [5] used cooperative game theory to form clusters of sensor nodes in a wireless sensor network and suggested a strategy set which is guaranteed to converge to an equilibrium point.

2.3.5 Grids

Grid computing has emerged as a new paradigm for solving large scale problems in science, engineering and commerce. Grids comprise heterogeneous resources (PC, clusters, supercomputers) owned by various organizations or individuals, which provide processing power to a same/different set of users with heterogeneous requirements. The resource owners (called producers) provide resources to use by users (consumers). The resource reservation/allocation protocols may be quite complex because of the large scale and heterogeneity involved in the grid. Moreover they must be distributed, so as to avoid a single point of failure. Many resource allocation and scheduling algorithms have been proposed based on economic models in [25]. The authors considered two important models, bartering-based models and price-based models.

Auction models are suitable for grids because of their decentralized structure and use of incentives for resource owners to contribute resources. These mechanisms are based on brokering and trading policies between the producers and consumers. Das and Grosu [40] introduced the combinatorial auction model for resource allocation in grids, which uses an approximation algorithm for solving the combinatorial auction problem. Kant and Grosu [81] proposed a double auction allocation model for grids, and three double auction protocols for resource allocation. They analyzed these protocols in terms of economic efficiency and system performance and showed that Continuous Double Auction Protocol is better from both resources and users perspective providing high resource utilization.

2.4 Summary

In this chapter, we presented the basic concepts of game theory and mechanism design which lays the foundation of our work. We discussed the application of incentives in task scheduling, peer-to-peer systems, networking, information security and grids. In networking we discussed the applications related to routing, cost sharing of multicast transmissions, wireless networks, network formation, and sensor networks.

CHAPTER 3: FAITHFUL DISTRIBUTED SHAPLEY MECHANISMS FOR SHARING THE COST OF MULTICAST TRANSMISSIONS

In this chapter we develop faithful distributed protocols that implement the Shapley Value mechanism assuming the Autonomous Nodes Model. We use digital signatures to authenticate the messages sent by the nodes. We use auditing and verification to detect cheating by the nodes. We investigate experimentally the performance of MC and SH mechanisms. We also study the behavior of the mechanisms from an economic perspective. We implement and deploy the mechanisms in a distributed real-world setting (PlanetLab). We run experiments to analyze the execution time of the mechanisms and the convergence of SH mechanism. We also investigate the number of users that receive the transmission and how much payment the content provider receives in different mechanisms.

3.1 Introduction

Recently, the transmission of multimedia content has become one of the most widely used applications on the Internet. A very common model of multimedia content distribution is the one in which the content provider distributes the audio/video to the subscribers, (*e.g.*, on-demand video) using *multicast transmissions* [39], [91], [8], [143]. Multicast transmissions are efficient as they minimize the number of messages traversing a single link by creating a multicast tree [144], [145], [84], [47]. We consider overlay multicast where pure application level or hybrid techniques are used to build overlays on the existing network infrastructure such as Internet [18], [76], [32]. Since the multicast trees are overlays, they can be efficiently constructed using spatial information [121], [89]. Approaches such as Reliability-Oriented Switching Tree (ROST) [144] construct multicast trees which are not only reliable but the tree construction algorithm also prevents cheating and manipulation of bandwidth/time information.

In such multicast trees the content provider is the root of the tree and the receivers are the other nodes in the tree. A node in the multicast tree receives the transmission from the parent node and forwards the content to its children nodes, if any. We consider a model in which each node in the multicast tree has multiple users. The node receives the content from the content provider and delivers it to the users by using IP Multicast or other techniques which are efficient in local scopes. This model characterizes well the scenarios where smaller Content Distribution Networks (CDNs) or Internet Service Providers (ISPs) pay bigger CDNs/ISPs to receive the content which is then distributed to their subscribers. The subscribers are charged by the smaller content providers for receiving the content. In our model we refer to the smaller CDNs as *nodes* and to their subscribers as *users*. This two-tier architecture is highly scalable, efficient and economic. One example of such two-tier architecture is TOMA (Two-tier Overlay Multicast Architecture), presented in [88]. In TOMA the ISPs create a Multicast Service Overlay Network (MSON) as the backbone service domain. End-users subscribe to the proxies advertised by MSONs to receive the content. TOMA is shown to be profitable for ISPs implementing it. The authors show through simulations that such two-tiered approach is very beneficial to reduce the cost of the multicast using overlay provisioning methods. Another example of overlay architecture, presented in [19], consists of Multicast Service Nodes (MSNs) which are connected with the source of the content to form the multicast backbone. When a MSN receives the content, it delivers it to the users who have subscribed to it. The architecture is scalable and self-organizing. It adapts well to the dynamic nature of the content delivery where users join and leave regularly. Scattercast [29] is another architecture where ScatterCast proXies (SCXs) are strategically placed nodes which create source rooted distribution trees to deliver content originating at source (user connected to the source) to the other SCXs. This approach is useful in scenarios when the content provider is not a fixed node.

The multicast transmissions used to distribute content are mostly receiver initiated (*e.g.*, Video-on-Demand) [42]. In such systems the users who receive the transmission have the responsibility to pay for the cost incurred by data flowing over the links [157]. In general the content provider provides the content and charges the receiver for the content. This charge should cover the cost of the links used in the multicast transmission. If the content distributors do not charge the users, they may not be profitable [75]. Since in a multicast transmission the same content is received by many users, one important problem that needs to be solved is, how should the users share the cost of the multicast transmission in a fair way. To solve this problem we need to provide mechanisms for computing the cost shares for each user. An axiomatic approach to allocate the costs to the receiving users and the mechanism that implements it was proposed in [73]. Feigenbaum *et al.* [53] used mechanism design theory (a subfield of micro-economics) to design mechanisms for computing the cost shares. In our model the nodes are responsible to pay the content provider for the received content. This model is specially useful in scenarios where smaller CDNs subscribe to bigger CDNs and pay them to receive the content which they deliver to their subscribers. The smaller CDN's can be considered to be the MSONs in TOMA [88] or SCXs in Scattercast [29], who pay the content provider according to the number of users receiving the content. To track the number of copies of the multimedia content made by the smaller CDN's the content providers can use one of the techniques surveyed in [92] and [119]. Considering specifications such as Protocol Independent Multicast (Sparse Mode), PIM-SM [43], our model can be extended to include any Rendezvous Point (RP) as the root, as long as there is a valid tree. The multicast tree does not have to be a binary tree to provide efficient transmission. This model is also very useful when the nodes are organizations who pay for the content and deliver it to multiple users within their organization. In some cases the nodes may not even charge their users for the content. If they do, they may use our proposed mechanism for calculating the cost share of each user within their local scope.

The benefit for each user e is quantified by a private single valued parameter u_e known as the user's *utility*. User e will like to receive the transmission if her cost share x_e is less than her utility, *i.e.*, if her *welfare* $w_e = u_e - x_e$ is positive. Cost sharing mechanisms determine who receives the multicast and how much they have to pay for the service. However, the calculation of cost shares is not a trivial task, as the users may cheat by lying about their utility.

The users (called agents) are assumed to be rational (*i.e.*, they want to maximize their profit). They have strong motivation to lie about their private values in order to get extra benefit. The task of the system designer is to design mechanisms that achieve system-wide goals. These goals may be hampered if the agents lie about their private values. To prevent manipulation and motivate users to participate honestly, Mechanism Design (a subfield of Microeconomics) is used to model the behavior of the agents. The standard Algorithmic Mechanism Design (AMD) [108] and Distributed Algorithmic Mechanism Design (DAMD) [54] study the mechanisms where the outcome and the payment depend on the input provided by the participants. Of importance are the *strategyproof mechanisms* [96], in which the users obtain maximum profit when they declare their private values truthfully. In the standard AMD [108], a centralized trusted entity implements the mechanism, *i.e.*, collects the input from agents, calculates the outcome and distributes the payment. When the mechanism is implemented in a distributed fashion [53], the agents themselves execute the mechanism and collectively calculate the outcome and the payments. Distributed implementations of mechanisms have been proposed for various problems such as multicast cost sharing [53] and scheduling [27], [28]. Incentives have been employed in distributed systems such as BitTorrent [138] to motivate participants to follow the specified protocol. Game theory has also been used to analyze the interactions of Internet Service Providers and to design equilibrium strategies for network pricing [134]. Pricing the resources [95] provides the added advantage of efficient and just use of available resources, which otherwise have to face issues such as ‘free-riding’.

Feigenbaum *et al.* [53] proposed two distributed mechanisms to calculate the cost shares of the participants in a multicast transmission: the Marginal Cost (MC) mechanism and the Shapley Value (SH) mechanism. MC is a two phase mechanism whereas SH is an iterative mechanism. The convergence and scalability of these mechanisms have been studied theoretically, but no experimental evaluation was performed. These mechanisms assume that the agents may lie about their private values but they may not deviate from the specified distributed algorithm. This model of distributed implementation is known as the *Tamper-Proof Model (TPM)*. Thus we use MC-TPM and SH-TPM to denote the mechanisms proposed in [53]. However, the TPM assumption is weak because agents can easily manipulate the distributed mechanism in their favor, since they control it. The model in which the agents may deviate from the specified distributed mechanism, is called the *Autonomous Nodes Model (ANM)*. Mitchell and Teague [99] proposed an MC mechanism assuming the ANM. They augmented the original MC-TPM mechanism proposed in [53] with asymmetric key cryptographic primitives to prevent cheating. Specifically they used digital signatures to authenticate the sender of the messages and auditing to verify that the agents executed the mechanism correctly. The limitation of the MC mechanism for ANM, proposed in [99], is that it assumes that there is only one user per node. Cryptographic methods like group key distribution have also been used for designing protocols which are robust against inflated subscriptions when content distribution uses IP Multicast [65].

3.1.1 Contributions

In this chapter we propose a distributed Shapley Value mechanism for sharing the cost of multicast transmissions for the Autonomous Nodes Model, called SH-ANM. We use digital signatures to authenticate the messages sent by the nodes and perform auditing and verification to detect cheating by the nodes. Our mechanism has provisions that prevent deviations in any of the multiple iterations of the SH mechanism.

Shneidman *et al.* [137] proposed the concept of *faithful implementation* of a mechanism. When the distributed mechanisms are implemented by the agents themselves, they may deviate from the specified algorithm if it is beneficial to do so. These deviations are categorized in three groups, information revelation, message passing and computation. A *faithful implementation* is a specification of a mechanism where the agents cannot gain any benefit by deviating from it. We show that our proposed distributed mechanism is a faithful implementation of the SH mechanism. Different ways to prove faithfulness specification have been suggested in [137] such as redundancy, catch-and-punish, problem partitioning, etc. We have used the catch-and-punish method to enforce truthful behavior where there is a trusted node (may be root) who audits the nodes randomly and punishes the node when deviation is found.

There are other models such as BAR (Byzantine, Altruistic, Rational) model [7] to characterize the behavior of distributed systems. In our dissertation we focus on the rational behavior of the nodes, so we use the faithfulness specification as mentioned above.

The time overhead induced by our proposed faithful mechanism to calculate cost shares is negligible compared to the transmission time of the actual content. We show this by implementing the mechanism and deploying it in a distributed real-world setting provided by PlanetLab [115]. Our experiments also provide interesting insights into the behavior of SH mechanism from the economic as well as computational perspectives. To compare the performance of the proposed SH mechanism we also implement the MC mechanism (for both ANM and TPM) as well as SH-TPM and deploy them in the same setting (*i.e.*, PlanetLab). We compare the performance of our proposed mechanism with that of the other existing mechanisms (MC-TPM, MC-ANM and SH-TPM). We also study the convergence of the SH-ANM mechanism. In addition we investigate the effect of varying the number of users per node and varying the number of nodes in the multicast on the number of users that receive the transmission and the payment the content provider receives in SH mechanism.

3.1.2 Organization

The organization of the rest of the chapter is as follows. In section 3.2, we present the network model we use, the MC and SH mechanisms for TPM, and the MC mechanism for ANM. Section 3.3 begins with the description of how nodes can cheat in the original implementation of the SH mechanism for TPM. We then present our proposed mechanism implementing the SH mechanism for ANM. In section 3.4 we show that our mechanism is a faithful implementation of the SH mechanism. Section 3.5 describes the experimental setup and the results we obtained from the experiments. Section 3.6 concludes the chapter with a summary and future research directions.

3.2 Cost Sharing Mechanisms

In this section we first describe some properties of MC and SH mechanisms. We describe the network model used and present the MC and the SH mechanisms assuming the TPM [53]. We describe the MC mechanism for the ANM [99].

The MC mechanism is strategyproof and efficient (*i.e.*, maximizes the overall welfare), but not budget balanced. In fact it is known that it generally runs budget deficit and in many cases does not generate any revenue at all [103]. No strategyproof mechanism can be both efficient and budget balanced at the same time [68]. The MC mechanism is recommended when the multicast delivery may be subsidized, if the mechanism runs a budget deficit [54]. Since the MC mechanism does not generate sufficient revenue, it is not a suitable mechanism from the content provider's point of view. The content provider will quickly go out of business, especially when there exists competing content providers. In addition, the MC mechanism is susceptible to collusion [49].

The SH mechanism is a better choice from these considerations because it is budget balanced and group strategyproof. Mechanisms are characterized by standard properties like No-Positive Transfers (NPT), Voluntary Participation (VP) and Consumer Sovereignty (CS). No-Positive Transfers means that the cost-shares are non-negative ($x_e \geq 0$). A mechanism satisfies the Voluntary Participation property when it ensures that users are not charged, if they do not receive the transmission. Consumer Sovereignty property guarantees that if a user is willing to pay a high enough amount, she will definitely get the transmission (*i.e.*, users cannot be excluded arbitrarily). The SH mechanism satisfies Consumer Sovereignty property, in addition to No-Positive Transfers and Voluntary Participation. The MC mechanism satisfies No-Positive Transfers and Voluntary Participation properties, but does not satisfy Consumer Sovereignty [103]. Although the SH mechanism is not efficient, for large user populations it approaches perfect efficiency. From the class of group-strategyproof mechanisms which are budget-balanced, the SH mechanism minimizes the worst-case welfare loss [103]. The only drawback of the SH mechanism is that it has a higher network complexity [50]. However, we believe that the cost-share calculation will induce a relatively small overhead to the overall multicast transmission. Thus it is justifiable to prefer the SH mechanism, given its good properties.

3.2.1 Model

We assume the following network model. The user population Q resides at N nodes. Each user $e \in Q$ resides at some node $i \in N$. The nodes are connected by bidirectional links. $R \subseteq Q$ is the set of users who receive the multicast transmission. The transmission starts from a node $root \in N$ and flows through a static multicast tree $T(R) \subseteq T(Q)$, where $T(R)$ and $T(Q)$ denote the multicast tree connecting the nodes in R and Q respectively. The techniques used to create these trees are described in [53, 140, 120]. We denote the subtree rooted at node i as T_i. Each link connecting node i to its parent node p has a cost c_i associated with it. $\mathcal{C}_i$ denotes the set of all t children $k_1, \ldots, k_t$ of node i and r_i denotes the set of all the users at node i. The payment sent by node i to $root$ is denoted by $payment_i$.

In the description of the mechanism in Fig. 3.1 (and the rest of chapter) we use the primitive **send**(M, R) to denote that the node executing the mechanism sends message M to a node R, and **recv**(M, R) to denote that the node executing the mechanism receives message M from a node R.

```
Node $i$ executes
Phase 1 (Bottom-up)
    for each child $k \in \mathcal{C}_i$
        recv($W_k, k$);
    Calculate $W_i = U_i + \sum_{k \in \mathcal{C}_i} W_k - c_i$;
    if($W_i \geq 0$)
        set $\sigma_e = 1$ for all $e \in r_i$
        send($W_i, p$);
    else
        set $\sigma_e = 0$ for all $e \in r_i$
        send($0, p$);

Phase 2 (Top-down)
    if (node is $root$)
        for each child $k \in \mathcal{C}_{root}$
            send($W_{root}, k$);
    else
        recv($A_p, p$);
        if($\sigma_e = 0$ for all $e \in r_i$ OR $A_p < 0$)
                $x_e = 0, \sigma_e = 0$ for all $e \in r_i$
            for each child $k \in \mathcal{C}_i$
                send($-1, k$);
        else
            Calculate $A_i = min(A_p, W_i)$;
            for each child $e \in r_i$
                if($u_e \leq W_p$)
                    $x_e = 0$;
                else
                    $x_e = u_e - A_p$;
            for each child $k \in \mathcal{C}_i$
                send($A_i, k$);

Calculate $payment_i = \sum_{e \in r_i} x_e$;
send($payment_i, root$);
```

Figure 3.1: **MC-TPM**: Distributed MC mechanism for TPM

3.2.2 Marginal Cost Mechanism for the Tamper-Proof Model (MC-TPM)

Let us assume that U_i denotes the sum of utilities of all the users at node i ($U_i = \sum_{e \in r_i} u_e$). The vector of utilities of all users $e \in Q$ is denoted by u. If a user e receives the transmission then $\sigma_e = 1$, otherwise $\sigma_e = 0$. T_i^+ denotes the union of T_i and the link from i to p. $W(u)$

represents the net-worth of the system at utility vector u and $W_i(u)$ denotes the welfare (*i.e.*, utilities minus cost) of subtree T_i^+. The mechanism is executed in two phases, a bottom-up phase and a top-down phase as shown in Fig. 3.1. In the bottom-up phase, the welfare values $W_i(u)$ are calculated by:

$$W_i(u) = U_i + \left(\sum_{k \in \mathcal{C}_i | W_k(u) \geq 0} W_k(u) \right) - c_i. \tag{3.1}$$

Each node calculates the welfare value and sends it to its parent. Finally the root node receives the welfare values from its children. In the top-down phase, the minimum welfare value A_i of a node i is calculated and propagated down the tree. A_i is the smallest welfare value $W_{i'}(u)$ of any node i' in the path from i to *root*. A_i is used to decide which users receive the transmission and what will be their cost share. The details of how to calculate A_i and x_e are shown in Fig. 3.1 and described in [53].

3.2.3 Shapley Value Mechanism for the Tamper-Proof Model (SH-TPM)

The Shapley Value [135] mechanism is implemented using an iterative algorithm consisting of two phases (bottom-up and top-down). In the bottom-up traversal each node i determines the number of users α_i in T_i who choose to receive the transmission. β_i is the cost share of each of the resident users at node i who receive the transmission, *i.e.*, $x_e = \beta_i, \forall e \in r_i \cap R$. n_i represents the number of users at node i who choose to receive the transmission. The bottom-up traversal starts from the leaf nodes. A leaf node k reports its α_k value to its parent (for the leaf nodes, $\alpha_k = n_k$). Nodes other than leaf nodes calculate $\alpha_i = \sum_{k \in \mathcal{C}_i} \alpha_k + n_i$ and send it to their parent node. After *root* receives $\alpha_i, i \in \mathcal{C}_{root}$, it initiates the top-down traversal, where it sends $\beta_{root} = 0$ to each of its children. Each node

```
**Node $i$ executes**
j = 0;
**do**
{
    ***Phase 1*** (Bottom-up)
    $j = j + 1$;
    **for** each child $k \in \mathcal{C}_i$
        **recv**($\alpha_k^j, k$);
    Calculate $\alpha_i^j = \sum_{k \in \mathcal{C}_i} \alpha_k^j + n_i^j$;
    **send**($\alpha_i^j, p$);

    ***Phase 2*** (Top-down)
    **if** (node is $root$)
        **for** each child $k \in \mathcal{C}_{root}$
            **send**($0, k$);
    **else**
        **recv**($\beta_p^j, p$);
        Calculate $\beta_i^j = (c_i / \alpha_i^j) + \beta_p^j$;
        **for** each child $k \in \mathcal{C}_i$
            **send**($\beta_i^j, k$);
} **while** ($\beta_p^j \neq \beta_p^{j-1}$);
Calculate $payment_i = \beta_i^j * n_i^j$;
**send**($payment_i, root$);
```

Figure 3.2: **SH-TPM**: Distributed SH mechanism for TPM

receives β_p from its parent and computes β_i as follows:

$$\beta_i = \left(\frac{c_i}{\alpha_i} \right) + \beta_p \tag{3.2}$$

β_i is then sent to all the children of node i (*i.e.*, all nodes $k \in \mathcal{C}_i$). All users $e \in r_i$ are assigned the cost share $x_e = \beta_i$. If the cost share x_e of any user e is greater than its utility u_e then user e declines to receive the transmission. In that case α_i decreases and it needs to be updated in the next bottom-up traversal. This increases the cost shares of the other users sharing the links with e. Thus in each iteration of the bottom-up and the top-down traversal, users may be removed from the receiver set R and the cost shares are updated. These iterations are repeated until no more users are dropped and until the cost share of any user does not change in two subsequent iterations. Initially $R = Q$ and in the worst case one user is dropped in each iteration. If we assume that the algorithm converges in m iterations, the number of messages required in this case is $\Omega(n \times m)$. The detailed analysis of computational and communication complexity is presented in [53].

The mechanism is shown in Fig. 3.2, where β_i^j and α_i^j refer respectively to the β_i and α_i values in iteration j. An example of the execution of one iteration of the SH mechanism is shown in Fig. 3.3. In this example, it is assumed that each node has only one user. The α_i^j values propagate from child to parent in the bottom-up phase (shown by solid arrows) and β_i^j values are sent from parent to child in the top-down phase (shown by dashed arrows).

3.2.4 Marginal Cost Mechanism for the Autonomous Nodes Model (MC-ANM)

The strategyproof MC mechanism prevents users from lying about their utility but it does not prevent the deviation of nodes from the specified distributed mechanism. The authors in [99] assume that the nodes are autonomous and they can deviate from the mechanism to increase their welfare. They proposed the use of digital signatures to authenticate the messages sent by a node and the use of auditing by the content provider to detect cheating. The content provider (*root*) is assumed to be the administrator of the mechanism. However this does not make the mechanism centralized since the nodes themselves compute the outcome in a distributed fashion. The administrator audits nodes with certain probability and enforces payments in case of discrepancies.

The modified mechanism (called protocol A) [99] is as follows. Assume that a message M signed using the private key K_i of node i is denoted by $E_{K_i}[M]$. The bottom-up traversal is the same as in the MC mechanism for TPM, except that the values sent by the children are signed using their private keys and each node verifies the signature after receiving the message. In the top-down phase each node i sends message $E_{K_i}[A_i, W_k]$, where A_i is the value in the original MC mechanism and W_k is the message which i received from child k during the bottom-up phase. At the end of the mechanism execution the content provider

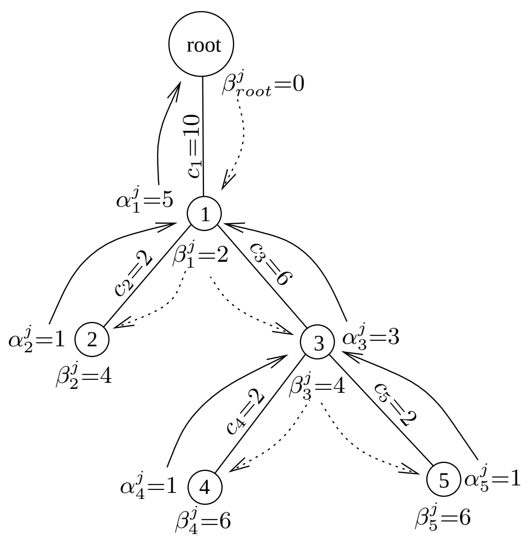


Figure 3.3: SH Mechanism with nodes executing the mechanism truthfully

audits each node i with probability P_a. It asks node i to send a proof of paying, $proof_i$ which consists of all the messages received by node i from its children and parent during the execution of the mechanism. To check the proof, the content provider decrypts all the messages contained in $proof_i$ (which are signed by the parent and the children of node i) and calculates the utility and the payment expected from node i. If the actual payment received from node i is different from the expected payment, node i has to pay a high penalty. Thus node i has no incentive to deviate from the mechanism (for proof, refer to [99]).

3.3 Shapley Value Mechanism for Autonomous Nodes Model (SH-ANM)

The distributed implementation of the SH mechanism in [53], is vulnerable to deviations by the nodes. In this section we present our proposed mechanism that prevents such deviations. We first present the notation used in describing the proposed mechanism. Then we describe the ways in which a node can cheat in the original mechanism. Finally we describe SH-ANM, our proposed mechanism.

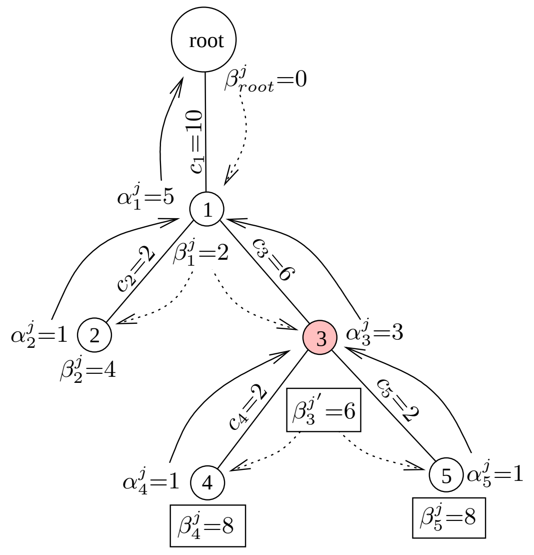


Figure 3.4: Node 3 cheats by sending modified values to its children

3.3.1 Notation

As described in section 3.2.3, SH is an iterative mechanism. It performs more than one iteration of the bottom-up and top-down traversals. The number of users at node i, in iteration j, who choose to receive the multicast transmission is denoted by n^j_i. For iteration j the number of users in the subtree rooted at i which receive the transmission is α^j_i. The cost share of users at node i, calculated in iteration j, is denoted by β^j_i. During the top-down phase of iteration j, node i receives cost share β^j_p from p. The cost share β^j_i is calculated using the formula $\beta^j_i = (c_i/\alpha^j_i) + \beta^j_p$ (from (3.2)). The message M signed by node i using its private key K_i is denoted by $E_{K_i}[M]$.

3.3.2 Cheating in the Tamper-Proof Model

In the following we show how a node can cheat by manipulating the values sent to other nodes. The scenario in which no cheating occurs is shown in Fig. 3.3. For simplicity we assume that every node has only one user. The values of α^j_i and β^j_i are shown for iteration j. In Fig. 3.3, the user at node 3 has to pay 4 and users at node 4 and 5, each pays 6.

By modifying β^j_i sent to its children, node i ensures that the users at node i receive the

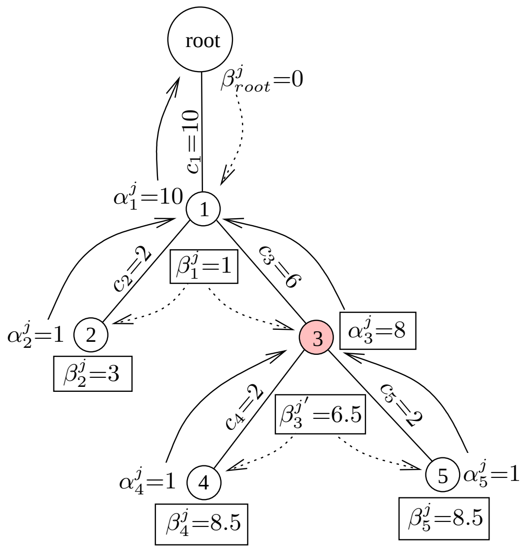


Figure 3.5: Node 3 cheats by sending modified values to its parent and children

transmission but pay nothing. To maintain the budget balance, node i makes its children and all the nodes in its subtree pay an extra amount, to compensate for the cost of transmission received by the users at node i. Node i sends $\beta^{j'}_i$ to its children instead of β^j_i. $\beta^{j'}_i$ is calculated using the formula $\beta^{j'}_i = (c_i + \beta^j_p * n^j_i)/(\alpha^j_i - n^j_i) + \beta^j_p$. Essentially node i divides the cost of link c_i among the users in its subtree, excluding users residing at i. The users at node i have also to share the costs of links from p to *root*, which is β^j_p. Node i distributes the share of its users to its descendants by adding $\beta^j_p * n^j_i$ to c_i and dividing only by the number of descendants $(\alpha^j_i - n^j_i)$. Thus, node i assigns zero as the cost share to its resident users. The cost share $\beta^{j'}_i$ is calculated in such a way that the budget remains balanced and the root cannot detect the cheating. Fig. 3.4 shows how node 3 cheats. The user residing at node 3 pays nothing and the users at node 4 and 5 each pays 8. Thus, each of them pays an extra amount of 2. The total amount overpaid collectively by the two users at node 4 and 5 is 4, compensating for the payment of the user at node 3.

Another way nodes can cheat is when a node i sends an inflated value $\alpha^{j'}_i$ instead of α^j_i to its parent in the bottom-up phase. Thus, the cost of the links connecting i to *root* will be shared among a greater number of (fake) users, reducing the per-user share at nodes in the tree $T(Q) - T_i$ (*i.e.*, nodes sharing the links connecting i to *root*). Node i then sends a manipulated value of β_i^j to its children and makes them pay an extra amount compensating for the reduced share of other users. In this case the benefit is received not only by node i but by all the nodes in subtree $T(Q) - T_i$. As shown in Fig. 3.5, node 3 sends $\alpha_i^{j'} = 8$ to its parents. As a result the payments by node 1 and 2 decrease to $\beta_1^j = 1$ and $\beta_2^j = 3$ respectively. Then node 3 calculates $\beta_3^{j'}$ using formula $\beta_i^{j'} = \beta_p^j + (c_i + \beta_p^j(\alpha_i^{j'} - \alpha_i^j))/(\alpha_i^j - n_i^j)$ and sends this $\beta_i^{j'}$ to its children. The cheating node assigns just the value β_p^j to the users residing at that node. In the example shown in Fig. 3.5, node 3 sends $\beta_3^{j'} = 6.5$ to its children thus making the payment by its children equal to 8.5 (instead of 6 originally). In addition it assigns $\beta_3^j = 1$ to its users and maintains the budget balance. This technique is specially useful when the cheating node wants to benefit the users in the subtree $T(Q) - T_i$ along with its users. In the cases discussed above, the extra amount to be paid by the children of the cheating node will be quite small if there are many users in the subtree T_i, thus, not many children nodes from T_i will drop out due to this increased extra amount.

There are other ways a node can manipulate the mechanism, for example by sending a very high value of β_i^j to its children. If β_i^j is sufficiently high, the utility of the users in subtree T_i will be less than their cost share and thus they will have to remove themselves from the set of receivers R of the transmission. Node i still receives the transmission, but it does not have to send the transmission further in the multicast tree. Thus, node i can save the upload bandwidth. Here the node is not acting maliciously, but rationally.

We want to detect such cheating and prevent it by penalizing the nodes who cheat. For this we require the nodes to sign the messages they send using digital signatures. We then use auditing/verification procedures to detect cheating. Since SH mechanism is essentially an iterative mechanism, a node can cheat in any of the iterations. To efficiently detect cheating, signing and auditing should be done for each iteration. We assume that the root node is a trusted node and it does not deviates from the protocol. The auditing and enforcement of payments can be done by a trusted party. It can be either a third party or the root node itself. Also it is assumed that the trusted party has sufficient means (lawful contracts, guarantees, advance payments, etc.) to force the node to pay the penalty due, in case that node is found cheating.

3.3.3 SH-ANM Mechanism

The proposed SH-ANM mechanism is executed in three phases. Phase 1 (bottom-up) and Phase 2 (top-down) are executed iteratively until the mechanism stabilizes. In each iteration j, in Phase 1, node i receives $E_{K_k}[\alpha_k^j]$ from each children $k \in \mathcal{C}_i$. Node i calculates

$$\alpha_i^j = \sum_{a=1}^{t} \alpha_{k_a}^j + n_i^j \tag{3.3}$$

and sends $E_{K_i}[\alpha_i^j]$ to its parent p.

Phase 2 (top-down) is initiated by *root* sending $\beta_{root}^j = 0$ to its children. Node i receives $E_{K_p}[\beta_p^j \,||\, \alpha_i^j]$ from its parent p. Here $||$ is the operator used to denote the concatenation (or grouping) of two values so that they can be treated as one message. Node i then calculates

$$\beta_i^j = \left(\frac{c_i}{\alpha_i^j}\right) + \beta_p^j \tag{3.4}$$

and sends $E_{K_i}[\beta_i^j \,||\, \alpha_k^j]$ to all its children. The cost share of the users at node i who receive the transmission is β_i^j.

Phase 3 starts after the mechanism stabilizes. In this phase payments are sent by the nodes to *root* and auditing is done by *root*. Assume that the mechanism stabilizes in m iterations. The payment sent by node i to *root* is calculated using $payment_i = \beta_i^m * n_i^m$. The root node audits node i with probability P_a by asking for a proof of payment. The proof of payment of node i is denoted by $proof_i$ and composed of: $proof_i = ||_{s=1}^{m}$ $(n_i^s \,||\, E_{K_p}[\beta_p^s \,||\, \alpha_i^s] \,||\, E_{K_{k_1}}[\alpha_{k_1}^s] \,||\, \ldots \,||\, E_{K_{k_t}}[\alpha_{k_t}^s])$. It is assumed that the root knows the public key of all the nodes who want to participate in the multicast transmission. After receiving $proof_i$ root calculates α_i^j using (3.3) and β_i^j using (3.4) for each iteration j. It then verifies that the payment received from users residing at node i is $\beta_i^m * n_i^m$. This computation is referred as verification of proof. Fig. 3.6 shows SH-ANM in algorithmic form. In the figure the process of verifying the proof is denoted by the primitive $\textbf{verify}(proof_i)$. If the payment received is not equal to $\beta_i^m * n_i^m$, node i has to pay a penalty $\mathcal{P}_i$ which is higher than any possible gain by cheating. To prevent the nodes from cheating, $P_a * \mathcal{P}_i$ should be strictly greater than any gain node i can obtain by deviating from the computational strategy, so that it has no incentive to cheat in Phase 1 or Phase 2. The maximum amount a node i can gain (or make the root to loose) by cheating denoted by G_i^{max} is equal to the sum of cost of all the links in subtree rooted at i and c_i. This can be written as $G_i^{max} = \sum_{j \in T_i} c_j$. If the mechanism is executed q times the penalty $\mathcal{P}_i$ for node i can be calculated as $\mathcal{P}_i = q * G_i^{max}$. Thus even if node i will be caught once out of q executions of mechanism, the penalty will be sufficiently high to discourage cheating. Alternatively, when the node is caught for the first time say after q executions of the mechanism, the penalty $\mathcal{P}_i$ can be calculated using the formula given above and then the root can audit that node in all the subsequent executions of the mechanism. If the node is caught again, it can be charged penalty $\mathcal{P}_i$ and then removed from the multicast transmission altogether. If a node a can present two different messages signed by node b then node b has to pay the penalty $\mathcal{P}_a$ and node a gets a reward (may be equal to $\mathcal{P}_a$).

When the mechanism stabilizes in m rounds, the number of messages required in SH-ANM is $\Omega(nm + n)$, which is a linear increase from that of SH. This increase is due to the extra messages required for auditing by the root node (*i.e.*, messages used for requesting and sending the proofs).

3.4 SH-ANM Mechanism's Properties

In this section we characterize the properties of the mechanism proposed in section 3.3 and show that users cannot increase their benefit by deviating in any way from the proposed mechanism. According to the classical mechanism design literature, rational users can be given incentives to truthfully reveal their private values and thus prevent users to exploit the

```
Node $i$ executes
        j = 1;
        do
        {

        Phase 1 (Bottom-up)
            for each child $k \in \mathcal{C}_i$
                recv($E_{K_k}[\alpha_k^j], k$);
            Calculate $\alpha_i^j = \sum_{a=1}^{t} \alpha_{k_a}^j + n_i^j$;
            send($E_{K_i}[\alpha_i^j], p$);

        Phase 2 (Top-down)
            if (node is $root$)
                for each child $k \in \mathcal{C}_{root}$
                    send($E_{K_{root}}[0||\alpha_k^j], k$);
            else
                recv($E_{K_p}[\beta_p^j \,||\, \alpha_i^j], p$);
                Calculate $\beta_i^j = (c_i/\alpha_i^j) + \beta_p^j$;
                for each child $k \in \mathcal{C}_i$
                    send($E_{K_i}[\beta_i^j \;||\; \alpha_k^j], k$);
            $j = j + 1$;
        }
        while ($\beta_p^j \neq \beta_p^{j-1}$);
        $m = j - 1$;

Phase 3 (Payment and Auditing)
        Calculate $payment_i = \beta_i^m * n_i^m$;
        send($payment_i, root$);
        if (node is $root$)
            request proof from node $i$ with probability $P_a$;
            recv($proof_i, i$);
            verify($proof_i$);
        else
            if proof is requested from $root$
                Prepare $proof_i = \|_{s=1}^{m} (n_i^s \,||\, E_{K_p}[\beta_p^s \,||\, \alpha_i^s] \,||$
                            $E_{K_{k_1}}[\alpha_{k_1}^s] \,||\, \ldots \,||\, E_{K_{k_t}}[\alpha_{k_t}^s]$);
                send($proof_i, root$);
```

Figure 3.6: **SH-ANM**: Distributed SH mechanism for ANM

system to get unjust benefit. However even such strategyproof mechanisms are susceptible to manipulation because the distributed algorithms implementing the mechanism are executed by the rational users themselves. Distributed mechanisms inevitably involve passing of messages between different nodes, which gives a chance to the rational nodes to manipulate the messages thereby increasing their profit unfairly. A stronger specification known as *faithful specification* proposed in [137], guarantees that the users cannot derive any unfair benefit, if the mechanism follows the specification. We first present the definitions of strong-communication compatibility, strong-algorithm compatibility and faithful implementation. Then we show that the proposed mechanism is a faithful implementation of the original SH mechanism. SH-ANM mechanism uses assymetric cryptography for signing the messages and decrypting the messages received from its children/parent. A cryptographic operation is encrypting a message by the node using its private key or decrypting a message using the public key of the node who sent the message. The number of cryptographic operations executed by a node i equal $2m * (1 + |\mathcal{C}_i|)$, where m is the number of rounds required by the mechanism to stabilize and $|\mathcal{C}_i|$ is the number of children of node i. In our experiments, a node has either zero or 2 children. Assuming $m = 4$, the number of crypto operations per node (except root node) varies from 8 to 24. The root node has to additionally verify the proofs which requires $2m * (1 + |\mathcal{C}_i|)$ operations for checking the proof from node i. In the bottom-up phase a node sends one value α_i to its parent and in the top down phase a node sends two values β_i^j and α_k^j to its children. A node has to just encrypt these values and thus the size of message to be encrypted/decrypted is very small. When constructing the proof, the node has to just combine all the messages received by that node and so there is no extra encryption/decryption performed at that time.

Definition 3.4.1 ***Strong-Communication Compatibility*** *[137]: A distributed mechanism is Strong-Communication Compatible (Strong-CC), if a participating node cannot obtain higher utility by deviating from the suggested message-passing strategy (independent of its information-revelation and computational actions), when the other nodes follow the suggested specification.*

Definition 3.4.2 ***Strong-Algorithm Compatibility*** *[137]: A distributed mechanism is*

Strong-Algorithm Compatible (Strong-AC), if a node cannot obtain higher utility by deviating from the suggested computational strategy (independent of its information-revelation and message-passing actions), when the other nodes follow the suggested specification.

Definition 3.4.3 ***Faithful implementation*** *[137]: A distributed mechanism specification is a faithful implementation when the corresponding centralized mechanism is strategyproof and when the specification is Strong-CC and Strong-AC.*

Theorem 3.4.1 *SH-ANM is a* faithful *distributed implementation of the SH mechanism.*

Proof. (sketch) In order to prove this we show that SH-ANM satisfies the three properties in Definition 3.4.3.

(i) The corresponding centralized mechanism is strategyproof: SH-ANM is a distributed implementation of the SH mechanism. Since the SH mechanism is group-strategyproof [103], which is a stronger property than strategyproofness, SH-ANM is also strategyproof.

(ii) Strong-CC: In all the three phases of SH-ANM no explicit message-passing takes place, *i.e.*, there is no message M which is received by node i and the same message M is sent by node i. We can assume that the links between nodes i and p are logical links and there may be physical nodes between two nodes who relay the message. The messages cannot be changed, because in all the phases the messages are digitally signed by the sending node i. So a node i cannot forge messages from other node a. This means that it is not possible for node i to change the message received from other node a and thus it has to follow the suggested message passing strategy. If a node i does not send the message it is supposed to send, the mechanism will not proceed and node i will not gain any benefit.

(iii) Strong-AC: In Phase 1, a node i may send a manipulated value of α_i^j to its parent p. Similarly in Phase 2, it may send a modified value of β_i^j to its children. However in Phase 3, *root* requests $proof_i$ from node i. $proof_i$ is composed of the messages received by node i in all iterations, *i.e.*, $proof_i = \|_{s=1}^{m} (n_i^s \,||\, E_{K_p}[\beta_p^s \,||\, \alpha_i^s] \,||\, E_{K_{k_1}}[\alpha_{k_1}^s] \,||\, \ldots \,||\, E_{K_{k_t}}[\alpha_{kt}^s])$. If node i cheated during Phase 1 or Phase 2, such cheating will be detected by the *root* in Phase 3 with probability P_a. This will cause node i to pay a penalty $\mathcal{P}_\rangle$. The penalty $\mathcal{P}_i$ for node i is chosen as mentioned in section 3.3.3, such that it is strictly greater than any gain node i can achieve by deviating from the protocol. Thus the node has no incentive to deviate and thus the mechanism's specifications are Strong-AC.

Since all three properties are satisfied, according to definition 3.4.3, SH-ANM is a faithful implementation of the SH mechanism. □

3.5 Experimental Results

3.5.1 Implementation

In order to investigate the performance of the proposed mechanism and the mechanisms presented in Section 3.2 and 3.3, we implemented them in a distributed environment. The program implementing the mechanism runs on all the nodes participating in the multicast (including the root node). Messages are sent and received by the nodes to calculate the cost shares, to demand/send proofs and to exchange control information. For encryption and decryption of messages using public key cryptography (specifically RSA) we used the Openssl library [1]. It is assumed that the public keys of the child and parent nodes are already available on each node, so no key exchange mechanism is employed. The implementation can be easily integrated within a multicast application where the selected cost-sharing mechanism is executed first and then the multicast data (*e.g.*, on-demand video) is transmitted.

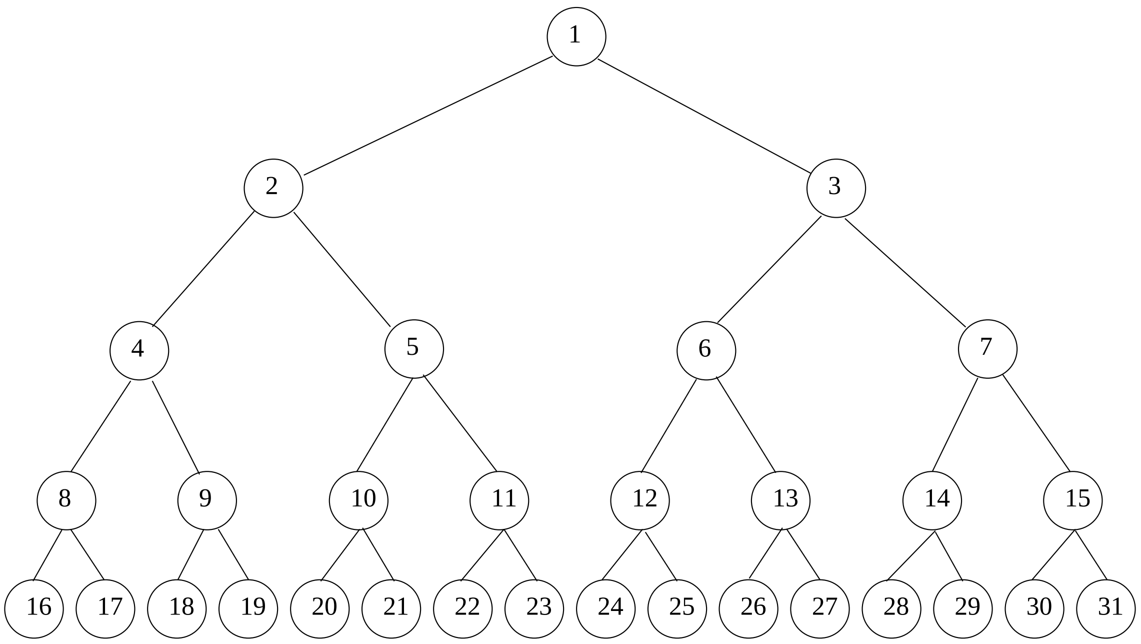
3.5.2 Experimental setup

We deployed the implementation on PlanetLab [115]. PlanetLab is a platform for developing, deploying and testing distributed services in a large scale distributed environment. We selected nodes randomly from all over the world to obtain realistic data. The nodes were chosen realistically, keeping in mind their geographical proximities. To conduct our experiments we used different number of nodes (7 to 127) and correspondingly generated the multicast trees. The multicast trees we used were complete binary trees as shown in

Fig. 3.7. Our proposed mechanism is not restricted to use only complete binary trees as multicast trees. It works with any type of multicast trees. The choice of a binary tree as a multicast tree in our experiments represents in a sense the worst case configuration where each node has only two children. This is because the time required to execute the mechanism depends on the depth of the multicast tree (shown in Fig. 3.12). For the same number of users the depth of a complete binary tree will be much greater as compared to that of other trees which have more than two children per node. Thus, in practice for the same number of nodes, the time required for execution will be much lower as there will be more than two children per node, making it more scalable.

In our experiments, the number of users per node was fixed in certain cases (mentioned below). In other cases the number of users at a node were randomly generated using the discrete uniform distribution over the interval [1,5]. The utilities of the users were also generated randomly using the uniform distribution over the interval [1,100]. A user drops out from the multicast when its cost share (calculated according to equation 3.4) is greater than its utility. A node drops out of the tree if no user on the node and its subtree are receiving the transmission (i.e. α_i =0). Unless stated otherwise the probability of cheating by nodes is $P_c = 0.5$ and the probability of checking the proof is $P_a = 0.5$.

In order to study the convergence and the scalability of the mechanisms we varied the number of users per node and also the number of nodes (to vary the depth of the multicast tree). We call a complete execution of a mechanism an experiment. Since PlanetLab is a very dynamic environment and one experiment would be insufficient to draw conclusions, we ran 60 experiments for each set of values and reported the average over those experiments. The main data collected for analysis are the time required for each node to execute the mechanism, the number of rounds required to stabilize (only for SH mechanisms), the number of users receiving the transmission and the payment received by the root.



1	planetlab2.cs.vu.nl	11	planetlab2.cs.wisc.edu	21	planet2.scs.stanford.edu
2	planetlab−1.tagus.ist.utl.pt	12	planetlab02.dis.unina.it	22	planetdev01.fm.intel.com
3	planetlab1.inf.ethz.ch	13	planetlab4.cs.st−andrews.ac.uk	23	planetlab1.cs.wisc.edu
4	planetlab1.cse.msu.edu	14	plab−1.sinp.msu.ru	24	planetlab−2.fokus.fraunhofer.de
5	planetlab1.eecs.umich.edu	15	planetlab1.iii.u−tokyo.ac.jp	25	planetlab−2.tagus.ist.utl.pt
6	planetlab2.inf.ethz.ch	16	planetlab3.cs.uiuc.edu	26	planetlab3.cs.st−andrews.ac.uk
7	plab−2.sinp.msu.ru	17	planetlab04.cs.washington.edu	27	planetlab2.cs.uoi.gr
8	planetlab−2.cmcl.cs.cmu.edu	18	planetlab−10.cs.princeton.edu	28	plnode01.cs.mu.oz.au
9	planetlab−01.bu.edu	19	planetlab−02.bu.edu	29	planetlab−01.kusa.ac.jp
10	planetlab2.eecs.umich.edu	20	planetlab6.cs.cornell.edu	30	planetlab2.iii.u−tokyo.ac.jp
				31	csplanetlab4.kaist.ac.kr

Figure 3.7: The multicast tree with 31 nodes used in the experiments

3.5.3 Results

We first analyze the effect of cheating on the existing SH-TPM mechanism. This provides the motivation for our work which proposes the SH-ANM mechanism that is able to prevent such cheating. The nodes can cheat in SH-TPM by sending manipulated values to their children, which causes the cheating nodes to pay less and the children of the cheating nodes to pay more. We did controlled experiments to compare the effect of cheating. We used SH-TPM with 31 nodes and 8 users per node. To see the influence of P_c, the probability of cheating on other nodes, we used 5 different values of P_c, *i*.e. 0.1,0.3,0.5,0.7 and 0.9. It is important to note here that only the nodes that have children have incentives to cheat.

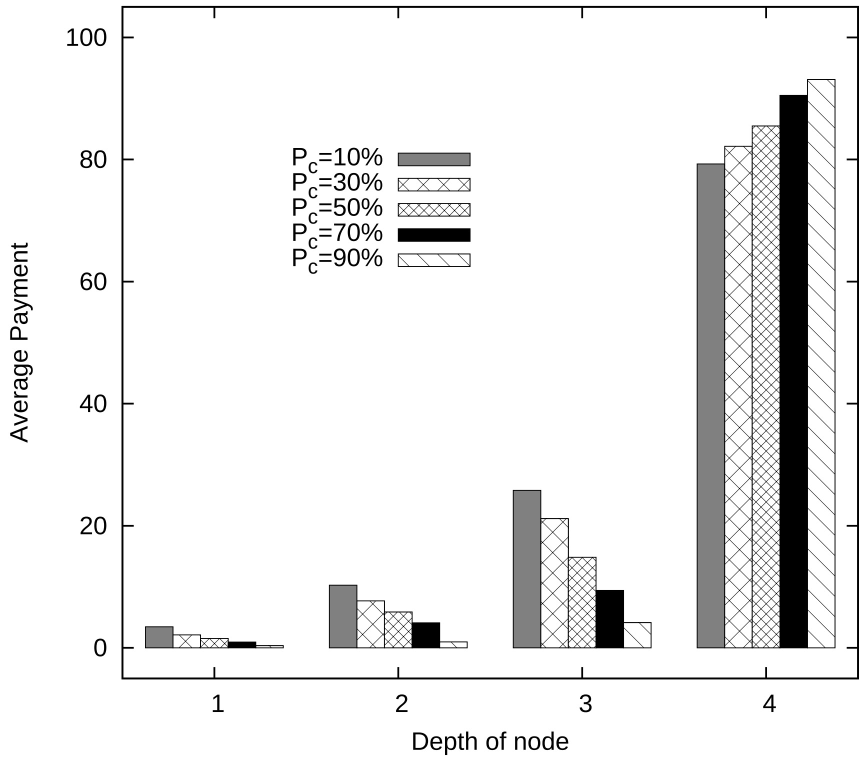


Figure 3.8: The effect of cheating on the payments in SH-TPM

Thus only nodes numbered from 2 to 15 cheated. Since the users at a node pay an amount proportional to the cost of transmission of multicast to that node, the average payment increases as the depth of the node increases. To present the results in Fig. 3.8 and Fig. 3.9 meaningfully, we banded together the values from nodes at the same depth in tree. That means for Fig. 3.8 we present average of the values of 'Average payment' made by nodes at depth d. We consider the root is at depth 0 and there are no users at root node (even if there are users at root, their payments will always be 0 according to the model). We observe in Fig. 3.8 that as P_c increases the payment made by nodes at depth 1 to 3 decreases and the payment made by nodes at depth 4 increases. This is because of the fact that when a node cheats, the payment by users of that nodes decreases but the payment by users at their children node increases to maintain the budget balance. The nodes at depth 2 and 3 may get an increased payment due to cheating by their parents but they transfer that payment to their children (because they are also cheating) and so the nodes that get affected and have

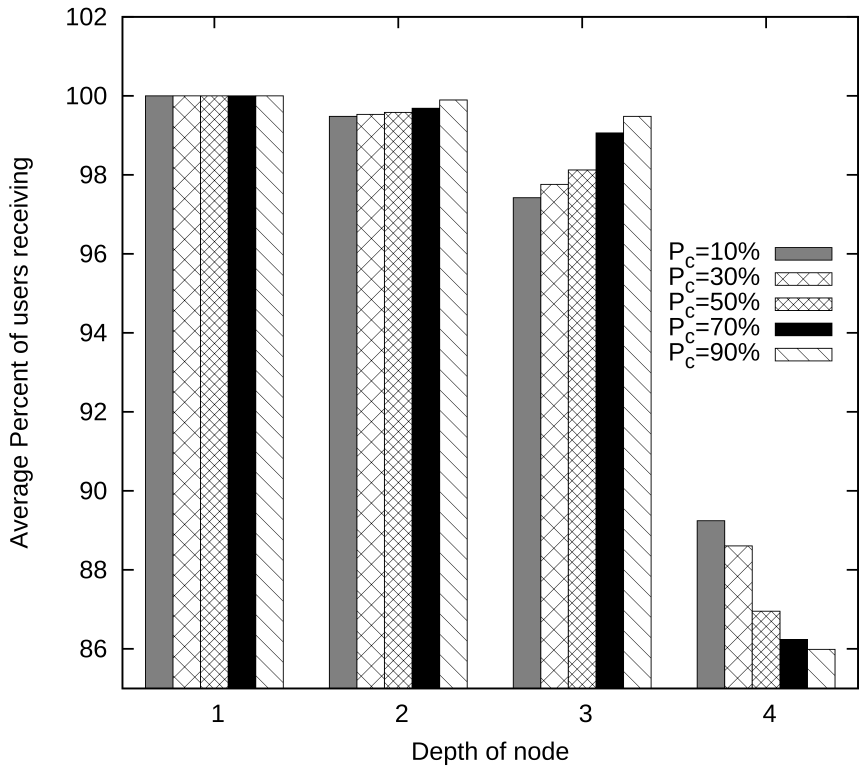


Figure 3.9: The effect of cheating on the number of users receiving the transmission in SH-TPM

to pay more are the leaf nodes (nodes at depth 4).

When the cost share of users at a node increases, there is more chance that users will drop off having their utility less than their cost share and vice-versa. Thus we see in Fig. 3.9 that as P_c increases, the average percent of users receiving the transmission increases at nodes at depth 2 and 3 and decreases for nodes at depth 4 (because of increase/decrease in their cost shares).

To detect such cheating and punish the nodes we developed the SH-ANM mechanism, which requires the nodes to sign the messages they send and then uses auditing at the end to catch cheating. To analyze the overhead induced by the message signing and auditing used in our proposed mechanism we compare the time required for execution of all four mechanisms (MC-TPM, MC-ANM, SH-TPM and SH-ANM). Fig. 3.10 shows the execution time of the four mechanisms at each node for a multicast tree of 31 nodes. Since we conducted 60

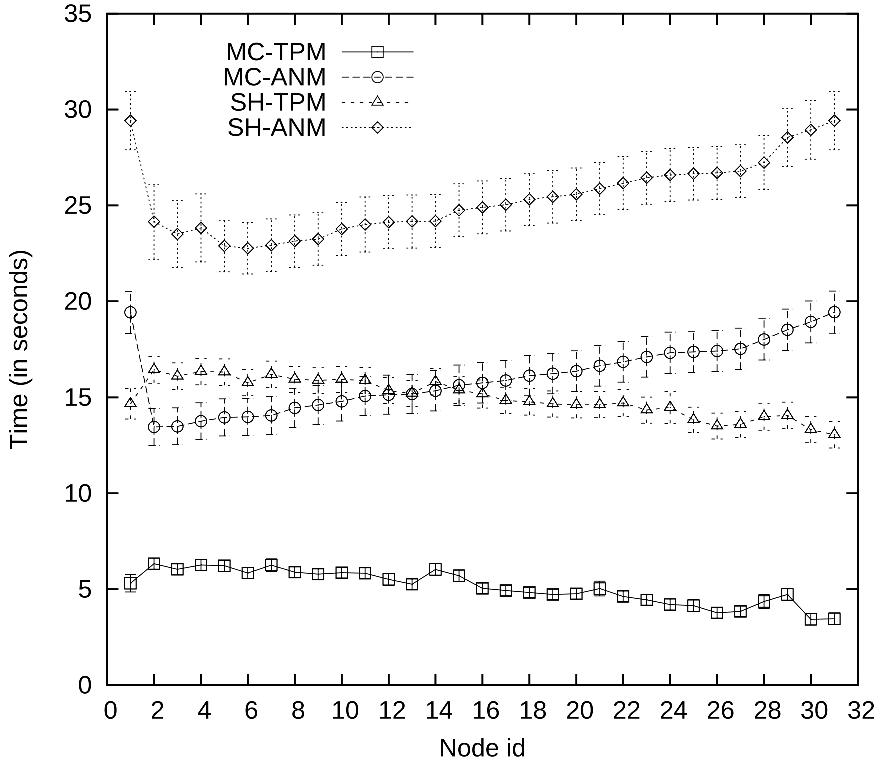


Figure 3.10: Execution time of the mechanisms

experiments for each mechanism to get the mean of the execution time, we also calculated the standard deviation and standard error of the execution time. The standard error is plotted as error bars in the figure. In the case of MC-TPM and MC-ANM mechanisms, the standard error is very small since they are not performing several iterations which usually induce more variance in the results.

The SH mechanisms require more time than the corresponding MC mechanisms. This is because they execute the bottom-up and top-down phases iteratively, whereas the MC mechanisms execute them only once. ANM mechanisms require more time compared to TPM mechanisms because of the additional messages used for sending and checking the proofs. We observe that the time required by SH-ANM mechanism is about 15 seconds more than that of SH-TPM mechanism. However this time is negligible since the overall multicast transmission, generally a video or audio, takes comparatively much longer time (in the order of minutes or hours). The total time required by cryptographic operations (message signing

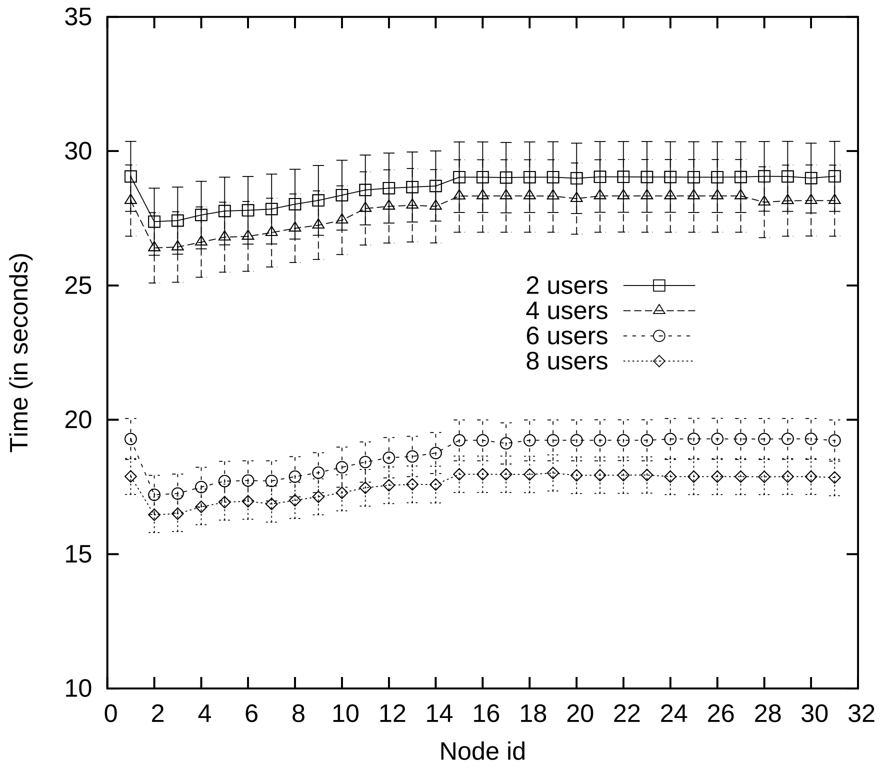


Figure 3.11: SH-ANM: Execution time at each node vs. number of users per node

and decrypting) was recorded separately and was found to be negligible (in the order of 0.1 to 0.4 seconds) due to short messages and very small number of cryptographic operations.

This execution time decreases as the number of users per node increases (as discussed below in reference to Fig. 3.11). The scenario we considered is when a smaller CDN acts as a node and receives the content from the source (bigger content provider) and delivers it to its users. In practice the smaller CDNs will have hundreds and thousands of subscribers thus reducing the overall time required for convergence of the mechanism as well as increasing the economic benefits (as discussed below in reference to Fig. 3.13).

To study the effect of the number of users per node on the convergence of the SH mechanisms as well as other economic properties, we executed the SH-ANM mechanism on 31 nodes fixing the number of users per node to 2, 4, 6 and 8 in different experiments. The utilities and link costs were generated randomly as explained above.

Counter-intuitively, the time required for the mechanism to stabilize decreases with the

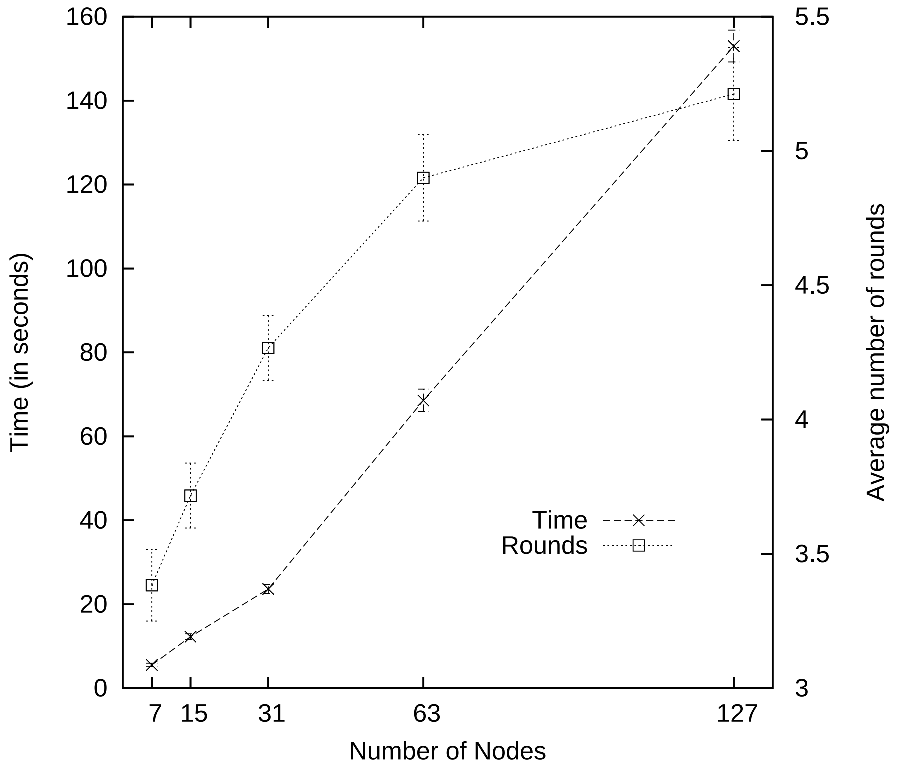


Figure 3.12: Execution time vs. number of nodes for SH-ANM

increase in the number of users, as shown in Fig. 3.11. This is because of the interesting economics behind the mechanism. The users share the cost of each link and as the number of users increases, the share per user decreases. Because of this, the mechanism stabilizes quickly and thus takes a smaller number of rounds and time to converge (as shown in Fig. 3.13). Fig. 3.11 shows the standard error as the error bars. We observe that the standard error also decreases as we increase the number of users per node.

In order to study the scalability of the SH-ANM mechanism we conducted another set of experiments by varying the number of nodes participating in the transmission. For these experiments we fixed the number of users per node to 8. As we see from Fig. 3.12, the time required to execute the mechanism increases as the number of nodes in the multicast tree (*i*.e. the the depth of the multicast tree) increase. This happens because the nodes have to send the messages to their parent in the bottom-up phase, starting from the leaf nodes, which goes up till the root and the message sent from root has to go to the leaf nodes in the

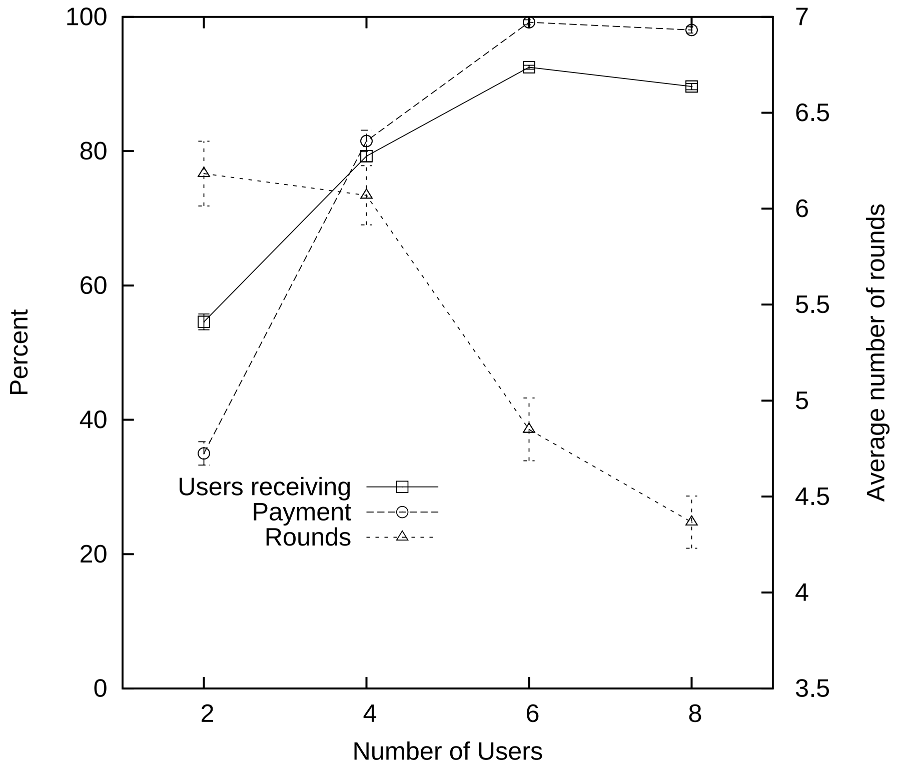


Figure 3.13: SH-ANM: Effect of number of users per node on various parameters

top-down phase. As the depth of the tree increases, this whole process takes more time and thus the time increases. The increase in the time is linear to the depth of the tree and thus the mechanism is quite scalable.

We also see from Fig. 3.12 that the average number of rounds required to stabilize the mechanism increases with the increase in the number of nodes. This is quite opposite to the observation that when the number of users was increased, the number of rounds decreased. The reason for this is that as the depth increases the cost share per user increases, especially for the leaf nodes, thus more users drop off, increasing the number of rounds necessary to converge. However we see in Fig. 3.12 that the increase in average number of rounds is linear to the increase in number of nodes. We also observed that the total number of users receiving the transmission decreased, although insignificantly, as the number of nodes is increased.

In Fig. 3.13 we show the average number of rounds required to complete the execution of the mechanism with varying number of users per node. The number of rounds is shown

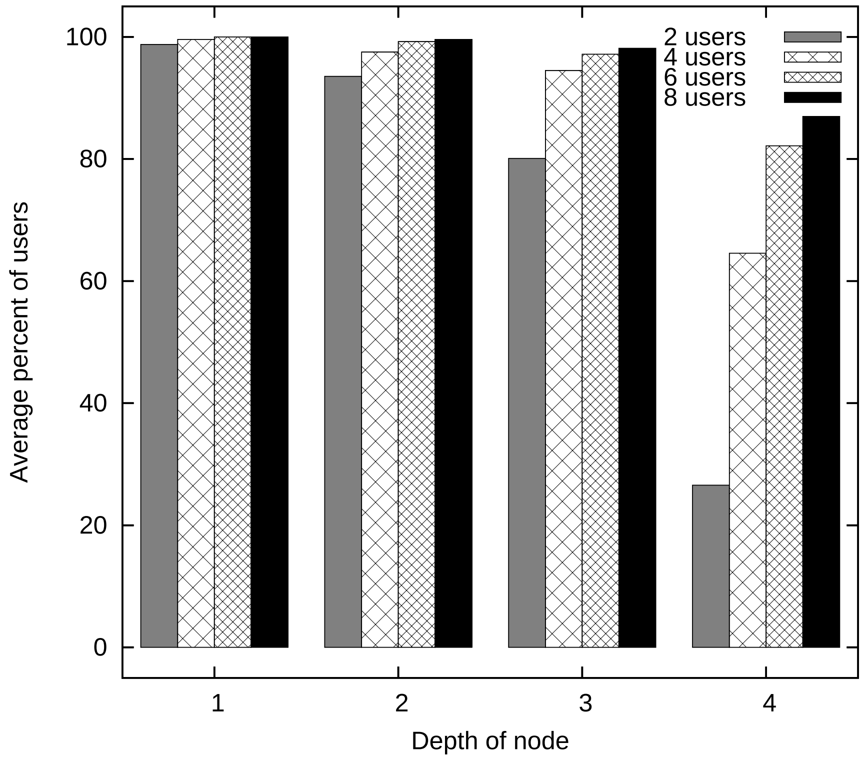


Figure 3.14: SH-ANM: Percentage of users per node receiving transmission

on the right vertical axis. We see that the number of rounds required to converge decreases as we increase the number of users per node. Another important consequence of increasing the number of users is that the total payment received by the mechanism increases as shown in Fig. 3.13. This is because as the cost share per user becomes smaller, a smaller number of users drop out from receiving the transmission. Although the SH mechanism is budget balanced, it will not receive the maximum possible payment if a subtree of the multicast tree does not receive the transmission. In other words, the root will receive the maximum payment if every subtree of the multicast tree has at least one user receiving the transmission. This happens when there is a small cost share per user, which is obtained when the number of users is increased, as evident from Fig. 3.13. In this figure we plot the ratio of the total payment received by the root to the sum of the costs of all links in the tree (not just those links which are involved in the transmission).

Another interesting property of the SH mechanism is observed in Fig. 3.14. In this figure also we present the average percent of users receiving the transmission grouped (and averaged) for all the nodes at a particular depth of the tree. We see that there is a decrease in the percentage of users receiving the transmission as the depth increases. This is because at the top of the tree, the cost share per user is small due to a small number of links connecting to the root. Since the cost share is small, more users will be able to receive the transmission, even if their utilities are low. As we go toward the leaf nodes the number of links necessary to reach the root increases and the cost share increases. This reduces the number of users on the nodes who receive the transmission. We also note from Fig. 3.14, that as the number of users per node is increased the percentage of users receiving the transmission increases. This is most significantly observable in the case of leaf nodes (at depth 4). As the number of users is increased from 2 to 8, the average percent of users who receive the transmission increases from 25% to 85%. This is because the cost share per user decreases as the number of users per node is increased and so less users drop off.

In the following, we focus on the economic aspects of our proposed mechanism and the MC and SH mechanisms in general. We have found several interesting results as well as verified many theoretical results from the experiments with distributed deployment of these mechanisms.

One significant observation is with regard to the payment received and the number of users receiving the multicast transmission in the MC and SH mechanisms. As pointed out earlier the MC mechanism is not budget-balanced and runs budget deficit in most of the cases, which is not a favorable property from the content providers perspective. In contrast, the SH mechanism is budget balanced. In Fig. 3.15 we show the payment received as percentage of the total cost of all links. We denote by 'MC' the MC-TPM mechanism when the number of users per node is generated randomly within the interval [1,5] and by 'MC-8' the MC-TPM mechanism with 8 users per node. We use similar notation for the SH mechanisms. We observe from Fig. 3.15 that the payment received in the case of MC mechanism is smaller than in the case of SH mechanism (for both [1..5] users and 8 users

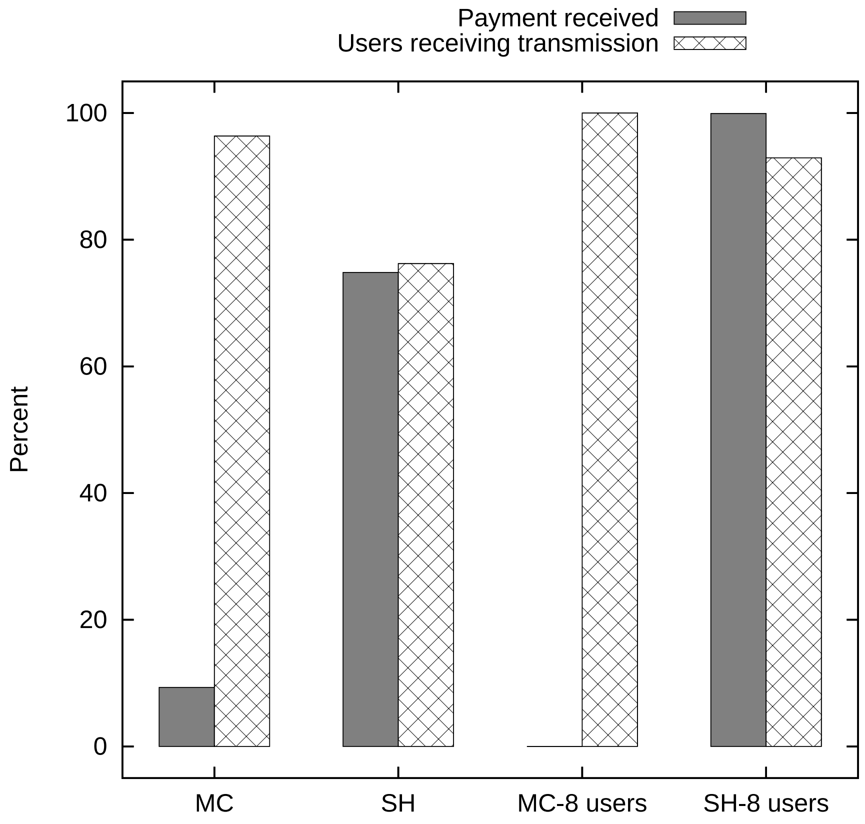


Figure 3.15: Payment received and number of users receiving the transmission in MC-TPM and SH-TPM mechanisms

per node). MC-8 generated no revenue at all (*i.e.*, all users pay 0), although all of the users received the transmission. In contrast SH-8 recovered 100% of the cost and about 95% users received the transmission. These results show that from the content provider's point of view, the SH mechanisms are more beneficial than the MC mechanisms.

3.6 Summary and Future Work

The Tamper-Proof Model (TPM) of distributed implementation assumes that the agents participating in the mechanism will not deviate from the distributed implementation. In the Autonomous Nodes Model (ANM), the agents may deviate and thus the existing TPM mechanisms are vulnerable to manipulations. We proposed a distributed cost sharing Shapley mechanism for ANM (SH-ANM) that is able to prevent such manipulations. To design the mechanism we used digital signatures for authentication of messages and auditing by the root node to penalize the cheating nodes. Thus, no node has incentives to cheat. We proved that the proposed mechanism is a faithful implementation of the original SH mechanism.

We implemented the proposed mechanism in a real-world environment provided by PlanetLab to analyze the overhead induced by the additional computation and communication resulting from the authentication and auditing procedures. From the experimental results we conclude that SH-ANM mechanism does not induce a significant overhead to the multicast transmission. We also deployed the existing MC-TPM, MC-ANM and SH-TPM mechanisms and compared their performance with that of SH-ANM.

Our mechanism relies on cryptography for the nodes to follow the specification. There are other methods suggested in [137], such as incentives, problem partitioning, redundancy to achieve faithfulness specification. In our future work we will try to find out ways to reduce the use of cryptography and use different methods (or combination of them) to enforce faithful behavior. In the future we also plan to study the convergence of the Shapley mechanism and develop faithful mechanisms for other distributed computing problems.

CHAPTER 4: ANTISOCIAL BEHAVIOR OF AGENTS IN SCHEDULING MECHANISMS

In this chapter we develop an antisocial strategy for the agents who intend to inflict losses on other participating agents in the context of online task scheduling on related machines [67]. The central mechanism allocates tasks of different sizes to different machines (agents) for execution and then issues payments as compensation for the use of their resources based on the Vickrey payment scheme. We study the effect of different agent's parameters on the losses inflicted to the other agents.

4.1 Introduction

Many current computer systems consist of geographically distributed resources coordinated by a central mechanism/protocol. The mechanism is responsible for allocating the load, controlling resources, disbursing remunerations, etc. The allocation and disbursement are done according to certain private values (parameters) of the participating agents. Generally resources are owned by self-interested organizations or agents with private goals. These agents may choose to reveal incomplete or untruthful information if that can deflect the outcome in their favor. An important problem in this setting is how to design protocols which obtain optimal outcomes even amidst this selfish behavior. *Mechanism design theory* [112] (also known as *Implementation theory*), a subfield of game theory, is an emerging approach to design such solutions for distributed multi-agent optimization problems. The mechanism design approach uses incentives to motivate the agents to report their true parameters and follow the protocol. The mechanisms having this property are called *truthful* (or *strategyproof*) mechanisms.

In a mechanism each participant has a privately known function called *valuation* which quantifies the agent's benefit or loss. *Payments* are designed and used to motivate the participants to report their true valuations. The goal of each participant is to maximize the difference of its valuation and payment. An agent makes profit if the payment it received is greater than its true valuation, and it suffers losses if the payment it received is less than its true valuation. However an agent can make profit and at the same time suffer *relative losses* if the payment to that agent is reduced because of some other agent misreports its valuation (as compared to the case in which all agents are reporting their true values).

Previous work in mechanism design assumed that agents' goal is to maximize their own profit and that an agent may deviate from the protocol if by doing so it can gain more profit [70, 108]. To tackle this problem incentive compatible mechanisms were developed. It is implicit in the assumption stated above that the agents do not care for others' profits. In the real world economy, sometimes a company may accept a lower profit or even sell goods at loss if it is able to reduce the profit of its competitors. For a short period of time or in particular cases such company may not give preference to maximizing its own profit. Such behavior is known as *antisocial behavior* and the agents having such motivations are called *antisocial agents*. A related issue in the economic literature is referred as "externalities" between bidders [77]. Externalities refer to the preferences of a bidder specifying besides himself, who else he/she wants to be the winner. Here we consider the antisocial agents who are interested in decreasing the profits of their rivals, whoever they may be. Such agents may exploit shortcomings in some mechanisms in order to inflict losses on the other agents. This is possible in mechanisms which employ payment schemes that depend on the reported valuations of the other agents (such as Vickrey payments). An antisocial agent needs to determine a bidding strategy that will allow it to inflict losses on the other participating agents while causing minimum losses to itself. Also given that the true value of an agent is a private value, the antisocial agent needs a way to infer the true values of the other agents such that it can modify its bid accordingly.

4.1.1 Related work

Recently many researchers have used mechanism design theory to solve problems in areas like resource allocation and task scheduling [107, 149, 153], congestion control and routing

[51, 82]. Nisan and Ronen [108] studied different mechanisms for shortest path and task scheduling. They proposed mechanisms to solve the shortest path problem and the problem of task scheduling on unrelated machines based on the popular VCG (Vickrey-Clarke-Groves) mechanism [33, 71, 147]. VCG mechanisms can be applied only to problems where the objective functions are simply the sum of agent's valuations and the set of outputs is finite. A general framework for designing truthful mechanisms for optimization problems where the agents' private data is one real valued parameter was proposed by Archer and Tardos [14]. Their framework can be applied to designing mechanisms for optimization problems with general objective functions and restricted form of valuations. They also studied the frugality of shortest path mechanisms in [15]. A truthful mechanism that gives the overall optimal solution for the static load balancing problem in distributed systems is proposed in [70]. Feigenbaum *et al.* [53] studied the computational aspects of mechanisms for cost sharing in multicast transmissions. A mechanism for low cost routing in networks is proposed in [51]. A scenario where agents solve scheduling problems in a distributed manner is presented and analyzed in [156]. The results and the challenges of designing distributed mechanisms are surveyed in [54]. In [141], the authors proposed a market-driven strategy for auctions where agents change their behavior according to the market. Goldberg and Hartline [63] studied the problem of designing mechanisms such that no coalition of agents can increase the combined utility of the coalition by engaging in a collusive strategy. Johari [78] considered three different market models for resource allocation in communication networks and power systems and quantified the efficiency loss in these environments when the market participants are price anticipating. They also showed that under reasonable assumptions their mechanisms minimize the efficiency loss, when considering price anticipating agents. The closest work to the present study is the work of Brandt and Weiss [23] in which the authors studied the behavior of antisocial agents in Vickrey auctions. They considered repeated Vickrey auctions in which the same item is auctioned in each round. They derived an antisocial strategy and introduced some notations to formalize the study of antisocial agents. This work considers a

different scenario where the tasks (items) are different but related in terms of their execution times.

4.1.2 Our contributions

We consider a mechanism for online task scheduling on related machines and develop an antisocial strategy for the participating agents. We characterize the antisociality of an agent and analyze the effect of different degrees of antisociality on the losses an agent can inflict to the other agents. We verify the correctness of the strategy by simulation. We also study the effect of changing different parameters on the amount of losses an antisocial agent can inflict on the other participating agents.

4.1.3 Organization

In Section 4.2 we present the formal model of the task scheduling problem and the scheduling mechanism. In Section 4.3 we present a formal characterization of the antisocial behavior and then present the antisocial strategy for the scheduling mechanism. In Section 4.4 we present and discuss experimental results. Section 4.5 concludes the chapter.

4.2 The Scheduling Problem and Mechanisms

4.2.1 The Scheduling Problem

We consider here the problem of scheduling $m \geq 1$ independent tasks $T_1, T_2, \ldots, T_m$ on $n \geq 1$ machines $M_1, M_2, \ldots, M_n$. Each task T_j is characterized by its processing requirement of r_j units of time. Each machine M_i is characterized by its processing speed $s_i > 0$ and thus task T_j would take $t^i_j = r_j/s_i$ time to be processed by M_i. A schedule $\mathbf{S}$ is a partition of the set of tasks indices into disjoint sets S^i, $i = 1, \ldots, n$. Partition S^i contains the indices corresponding to all the tasks allocated to M_i. The goal is to obtain a schedule $\mathbf{S}$ minimizing a given objective function such as makespan, sum of completion times, etc. In the scheduling literature, this problem is known as *scheduling on related machines* [116].

4.2.2 Scheduling Mechanisms

In this section we first describe the scheduling mechanism design problem and then present two scheduling mechanisms. The scheduling mechanisms are auction-based, which means that the mechanism announces the tasks (size of tasks) to be executed and invites bids (the estimated execution time for the tasks) from machines. Based on the received bids the mechanism allocates the tasks to the machines in the system. Generally the machines deploy a daemon/process to bid on behalf of the machine. Such process providing service to machine M_i is called an Agent, denoted by A_i. When a task is allocated to agent A_i representing machine M_i, the agent submits the task to the machine. While the machine M_i is busy in executing the task, agent A_i may take part in the bidding process to gather more tasks for execution.

Definition 4.2.1 (Mechanism design problem) The problem of designing a scheduling mechanism is characterized by:

(i) A finite set $\mathcal{S}$ of allowed outputs. The output is a schedule $\mathbf{S}(\mathbf{b}) = (S^1(\mathbf{b}), S^2(\mathbf{b}), \ldots, S^n(\mathbf{b}))$, $\mathbf{S}(\mathbf{b}) \in \mathcal{S}$, computed according to the agents' bids, $\mathbf{b} = (\mathbf{b}_1,\ \mathbf{b}_2,\ \ldots, \mathbf{b}_n)$. Here, $\mathbf{b}_i = (b^i_1, b^i_2, \ldots, b^i_m)$ is the vector of values (bids) reported by agent A_i to the mechanism.

(ii) Each agent A_i, $(i = 1, \ldots, n)$, has for each task T_j a privately known parameter t^i_j $(j = 1, \ldots, m)$ called the *true value* which represents the time required by agent A_i to execute task T_j. The preferences of agent A_i are given by a function called *valuation* $V_i(\mathbf{S}(\mathbf{b}), \mathbf{t}_i) = \sum_{j\in S^i(\mathbf{b})} t^i_j$ (i.e., the total time it takes to complete all tasks assigned to it). Here $\mathbf{t}_i = (t^i_1, t^i_2, \ldots, t^i_m)$.

(iii) Each agent goal is to maximize its *utility.* The utility of agent A_i is $U_i(\mathbf{b}, \mathbf{t}) = P_i(\mathbf{b}, \mathbf{t}) - V_i(\mathbf{S}(\mathbf{b}), \mathbf{t}_i)$, where P_i is the payment handed by the mechanism to agent A_i.

(iv) The goal of the mechanism is to select a schedule $\mathbf{S}$ that minimizes the make-span $C(S(\mathbf{b}), \mathbf{t})$, (i.e., $\min(C(S(\mathbf{b}), \mathbf{t}))$), where $C(S(\mathbf{b}), \mathbf{t}) = \max_i \sum_{j \in S^i(\mathbf{b})} t_j^i$.

An agent A_i may report a value (bid) b_j^i for a task T_j different from its true value t_j^i. The true value characterizes the actual processing capacity of agent A_i. The goal of a truthful scheduling mechanism is to give incentives to agents such that it is beneficial for them to report their true values. Now we give a formal description of a mechanism and define the concept of truthful mechanism.

Definition 4.2.2 (Mechanism) A *mechanism* is a pair of functions:

(i) The *allocation function*: $\mathbf{S}(\mathbf{b}) = (S^1(\mathbf{b}), S^2(\mathbf{b}), \ldots, S^n(\mathbf{b}))$. This function has as input the vector of agents' bids $\mathbf{b} = (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n)$ and returns an output $\mathbf{S} \in \mathcal{S}$.

(ii) The *payment function*: $P(\mathbf{b}, \mathbf{t}) = (P_1(\mathbf{b}, \mathbf{t}), P_2(\mathbf{b}, \mathbf{t}), \ldots, P_n(\mathbf{b}, \mathbf{t}))$, where $P_i(\mathbf{b}, \mathbf{t})$ is the payment handed by the mechanism to agent A_i.

Definition 4.2.3 (Truthful mechanism) A mechanism is called *truthful* or *strategyproof* if for every agent A_i with true value $\mathbf{t}_i$ and for every bids $\mathbf{b}_{-i} = (\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}, \mathbf{b}_{i+1}, \ldots, \mathbf{b}_n)$ of the other agents, the agent's utility is maximized when it declares its true value $\mathbf{t}_i$ (i.e., truth-telling is a dominant strategy).

Nisan and Ronen [108] designed a truthful mechanism for a more general problem called scheduling on unrelated machines. In this case the speed of machines depends on the task i.e., for a task T_j and agent A_i, the processing speed is s_{ij}. If $s_{ij} = s_i$ for all i and j then the problem reduces to the problem of scheduling on related machines [116]. The authors provided an approximation mechanism called MinWork, minimizing the total amount of work. MinWork is a truthful mechanism. In the following we present the MinWork mechanism.

Definition 4.2.4 (MinWork Mechanism) [108] The mechanism that gives an approximate solution to the scheduling problem is defined by the following two functions:

(i) The *allocation function*: each task is allocated to the agent who is able to execute the task in a minimum amount of time (Allocation is random when there are more than one agent with minimum type).

(ii) The *payment function* for agent A_i given by:

$$P_i(\mathbf{b}, \mathbf{t}) = \sum_{j \in S^i(\mathbf{b})} \min_{i' \neq i} t_j^{i'} \tag{4.1}$$

The MinWork mechanism can be viewed as running separate Vickrey auctions simultaneously for each task. Since an antisocial agent wants to infer the true value of another agent and bid in such a way that the other agent incurs losses, this is not possible if all tasks are auctioned simultaneously. In this work we consider a modification of MinWork mechanism that solves the problem of scheduling on related machines through repeated Vickrey auctions with heterogeneous commodities. This mechanism can be viewed as running a sequence of separate Vickrey auctions, one for each task (of different size). The main difference from MinWork is that auctions are run in sequence and not simultaneously and thus an agent can apply the strategy in subsequent runs and inflict losses on other agents.

Definition 4.2.5 (Modified MinWork Mechanism) The mechanism that gives an approximate solution to the problem of task scheduling on related machines is defined as follows:

For each task T_j $(j = 1, 2, \ldots, m)$:

(i) The *allocation function*: Task T_j is allocated to the agent who is able to execute it in a minimum amount of time.

(ii) The *payment function* for agent A_i given by:

$$P_i(\mathbf{b}, \mathbf{t}) = \min_{i' \neq i} t_j^{i'}, \qquad j \in S^i(\mathbf{b}) \tag{4.2}$$

The Modified Minwork mechanism (MMW), models the online scheduling problem [67, 133] where the jobs arrive to the scheduler over time and there is no ‘a priori’ knowledge about the arrival of the next job. Online scheduling has many applications in different areas such as real time systems [85], cluster computing [16], and wireless communications [12].

In the following we present the protocol that implements the Modified MinWork mechanism.

Protocol MMW:

For each task T_j, $j = 1, \ldots, m$:

1. Agent A_i, $i = 1, \ldots, n$ submits bid b^i_j to the mechanism.
2. After the mechanism collects all the bids it does the following:
   - 2.1. Computes the allocation using the allocation function.
   - 2.2. Computes the payments P_i for each agent A_i using the payment function.
   - 2.3. Sends P_i to each A_i.
3. Each agent receives its payment and evaluates its utility.

After receiving the payment each agent evaluates its utility and decides on the bid values for the next task. This mechanism preserves the truthfulness property in each round.

4.3 The Antisocial Strategy

In this section we design an antisocial strategy for the agents participating in the Modified MinWork mechanism. First we present a formal characterization of agent’s antisocial behavior and then present the proposed antisocial strategy.

4.3.1 Background

The mechanisms presented in Section 2 are auction-based mechanisms which provide with efficient system-wide solutions even when the participating agents intend to deviate from the standard behavior in order to gain more profit. Such mechanisms are still vulnerable to

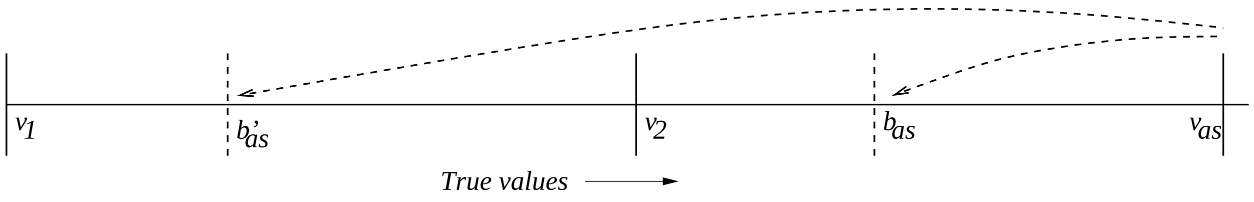


Figure 4.1: Antisocial Behavior

manipulation by agents who intend to inflict losses on the other agents, rather than trying to maximize their own profit. An agent can exploit the fact that certain payment schemes use the valuations of other agents to calculate the payment to the winner. One such popular scheme is the payment scheme used in Vickrey Auctions [123, 124, 130], also known as second-price sealed-bid auctions. In the Vickrey auction the payment to the winner is equal to the bid of the second best agent for the auctioned item. As mentioned before, the MMW mechanism can be viewed as running a Vickrey auction separately for each task. Thus an antisocial agent can reduce the profit of the winner (i.e., induce a loss) by bidding values close to the winner's bid (assuming that the antisocial agent knows the bids of the winner).

Figure 4.1 explains this antisocial behavior. In this figure, v_1, v_2 and v_{as} are the true valuations of three different agents A_1, A_2 and A_{as} respectively. When they all bid their true values agent A_1 wins the auction and receives v_2 as payment. However if agent A_{as} wants to act as an antisocial agent, it can bid lower than its true value, such that its bid is between the bids of the best agent and the second best agent (in this case A_1 and A_2). By carefully choosing this value agent A_{as} will be able to lower the profit of the best agent, as shown in Figure 4.1. If the agent A_{as} bids b'_{as} then agent A_1 still wins the auction but receives a payment equal to b'_{as} rather than v_2. Thus effectively the profit of A_1 is reduced by $v_2 - b'_{as}$.

The loss experienced here by the best agent is called *relative loss RL*. The relative loss can be defined as the loss experienced by an agent when the payment it receives is reduced as compared to the payment it would have received when all agents have reported their true valuations. This is different as compared to the absolute loss which is incurred when the payment is less than the agent's true valuation. An agent may gain profit (the payment received is greater than its true valuation), but still incurring relative loss, due to the antisocial behavior of another agent. The loss mentioned in the subsequent text is the relative loss (RL) unless mentioned otherwise. Since the actual amount of loss varies depending on the size of the task, it is more meaningful to represent the relative loss as percentage.

The relative loss (RL) is calculated as follows:

$$RL = \frac{P_i^T - P_i}{P_i^T} * 100 \tag{4.3}$$

where P_i^T is the payment received by the best agent A_i when all agents, including A_i, report their true values; and P_i is the payment received by the best agent A_i when one of the agents is antisocial.

To formalize the study of antisocial behavior in Vickrey auctions, Brandt and Weiss [23] introduced a parameter called the *derogation rate* of an agent. The derogation rate $d_i \in [0, 1]$ can be defined as the degree of antisocial behavior of an agent A_i. In other words an agent will try to maximize the weighted difference between its utility and the utility of the other agents, where the weight is characterized by d_i. Thus the derogation rate quantifies whether the agent gives preference to maximizing its profit or to inflicting losses to the other agents. The *payoff* of the agent depends on the derogation rate of the agent:

$$\mathit{payoff}_i = (1 - d_i)U_i - d_i \sum_{i' \neq i} U_{i'} \tag{4.4}$$

Every agent tries to maximize its payoff. A regular agent has $d_i = 0$ and a purely destructive agent has $d_i = 1$. A balanced agent has $d_i = 0.5$ i.e., it gives equal weight to its utility and others' losses.

The proposed strategy is for an agent participating in the Modified MinWork mechanism. As mentioned above, to be able to effectively inflict losses on the best agent, an antisocial agent needs to know the bids (true valuations) of the other agents. However as stated previously, the valuations are private values not known to the other agents. By using the proposed strategy an antisocial agent gains this information and use it to inflict losses to the best agent. Brandt and Weiss [23] studied a similar problem but in the context of repeated Vickrey auctions in which the same item is auctioned in each round. We base our strategy on those principles and develop a strategy for auctions involving tasks of different sizes. The strategy exploits the fact that the machines (agents) on which tasks are executed are related, that means that the true value of an agent (time required to execute a task) is proportional to the task size. We have also assumed that only the antisocial agent is manipulating its bids and that the other agents are reporting their true values.

In the proposed antisocial strategy, the antisocial agent bids according to its derogation rate and makes decisions based on whether it was allocated the last task or not. The agent starts by bidding its true valuation and if it loses, it reduces its bid step by step so that it can gradually bid less than the bid of the best agent. The amount by which the antisocial agent A_i reduces its bid in every step is characterized by the *step down percent*, q_i. If q_i is small the agent reduces the bid by a very small amount and vice versa. When the antisocial agent's bid is lower than the bid of the best agent, the antisocial agent wins and receives a payment equal to the bid of the best agent's which is assumed to be the true value of the best agent. After getting this information the antisocial agent calculates its subsequent bids such that it can inflict a relative loss to the best agent. This calculation depends on the derogation rate of the agent, the true value of the best agent and the task size. The antisocial agent keeps bidding accordingly for the subsequent tasks thereby inflicting losses to the best agent.

4.3.2 Antisocial Strategy

In describing the proposed antisocial strategy we use the following notations:

The proposed antisocial strategy is presented in Figure 4.2. Figure 4.3 shows the state diagram representing various stages of the strategy and the associated transition criteria.

$Priceknown$	boolean variable indicating whether A_i knows the price paid for a task in the last round (it also indicates if A_i won in the last round);
$PGreaterThanV$	boolean variable indicating if the payment received in the last round was greater than the valuation;
$DecreaseBid$	boolean variable to control execution, it indicates if the bid is to be decreased;
$t_j = t_j^i$	time required by antisocial agent A_i to execute the current task T_j (for simplicity we denote t_j^i by t_j since the strategy is followed by agent A_i);
$t_{j-1} = t_{j-1}^i$	true value of agent A_i for previous task T_{j-1} (i is implicit in t_{j-1});
P_i	payment received by agent A_i in the last round (if it won);
$b_{j-1} = b_{j-1}^i$	bid placed by agent A_i for the previous task, which might be different from the current bid (i is implicit in b_{j-1});
ϵ	an infinitesimal quantity;
d_i	derogation rate of the antisocial agent A_i;
q_i	step down percent of antisocial agent A_i's bid.

For a particular task, an antisocial agent following this strategy can be in one of six stages. Depending on the current stage, other parameters and the current allocation, the agent either stays in the same stage or moves to some other stage for the next task. Based on the current stage, the agent calculates the bid for the allocation of the current task. This strategy is followed only by antisocial agents.

The state diagram in Figure 4.3 can be divided in two areas, left and right. Left area (Stages 1 and 2) corresponds to a situation in which the antisocial agent is the best agent itself. In this case the best agent bids more than its true valuation so as to safeguard itself against possible attacks by other antisocial agents. If the presence of other antisocial agents is considered along with the best agent, the best agent will be able to safeguard only in some particular cases depending on its derogation rate and the derogation rate of the other antisocial agents. The right area (Stages 3, 4 and 5) corresponds to the case where an agent other than the best agent is antisocial, which might be generally the case.

An antisocial agent A_i starts in Stage 0 where it bids its true valuation. If the agent wins in Stage 0, the valuation of this agent is the lowest (and thus receives the payment equal to the true value of the second best agent). In this case the agent transitions to Stage 1. We do not focus on this case, because no loss can be inflicted to the best agent if the best agent

```
$Priceknown \leftarrow false$;
$PGreaterThanV \leftarrow false$;
$DecreaseBid \leftarrow false$;
$j \leftarrow 1$;
**Stage 0:** $b_j \leftarrow t_j$;
        $A_i$ bids $b_j$;
        **if** ($A_i$ wins)
            **then** $P_i \leftarrow price$;
                $Priceknown \leftarrow true$;
                $PGreaterThanV \leftarrow true$;
            **else** $DecreaseBid \leftarrow true$;
**do**
    $j \leftarrow j + 1$;
    **Stage 1:** **if** ($Priceknown$ **and** $PGreaterThanV$)
                **then** $b_j \leftarrow t_{j-1} + d_i \left(\frac{t_j}{t_{j-1}} * P_i - t_j\right)$;
                    $A_i$ bids $b_j$;
                    **if** ($A_i$ loses)
                        **then** $Priceknown \leftarrow false$;
    **Stage 2:** **if (not** $Priceknown$ **and** $PGreaterThanV$)
                **then** $b_j \leftarrow t_j + d_i \left(\frac{t_j}{t_{j-1}} * \left(\frac{b_{j-1} - t_{j-1}}{d_i} + t_{j-1}\right) - t_j\right)$;
                    $A_i$ bids $b_j$;
                    **if** ($A_i$ wins)
                        **then** $Priceknown \leftarrow true$;
                            $P_i \leftarrow price$;
                        **else** $DecreaseBid \leftarrow true$;
                            $Priceknown \leftarrow false$;
    **Stage 3:** **if** ($DecreaseBid$)
                **then** $b_j \leftarrow t_j * \left(\frac{b_{j-1}}{t_{j-1}} - q_i\right)$;
                    $A_i$ bids $b_j$;
                    **if** ($A_i$ wins)
                        **then** $Priceknown \leftarrow true$;
                            $P_i \leftarrow price$;
                            $PGreaterThanV \leftarrow false$;
                            $DecreaseBid \leftarrow false$;
    **Stage 4:** **if** ($Priceknown$ **and not** $PGreaterThanV$)
                **then** $b_j \leftarrow t_j - d_i \left(t_j - P_i * \frac{t_j}{t_{j-1}}\right) + \epsilon$;
                    $A_i$ bids $b_j$;
                    **if** ($A_i$ wins)
                        **then** $P_i \leftarrow price$;
                        **else** $Priceknown \leftarrow false$;
    **Stage 5:** **if (not** $Priceknown$ **and not** $PGreaterThanV$)
                **then** $b_j \leftarrow t_j - d_i \left(t_j - \frac{t_j}{t_{j-1}} * \left(\frac{b_{j-1} - t_{j-1} - \epsilon}{d_i} + t_{j-1}\right)\right) + \epsilon$;
                    $A_i$ bids $b_j$;
                    **if** ($A_i$ wins)
                        **then** $P_i \leftarrow price$;
                            $Priceknown \leftarrow true$;
                        **else** $Priceknown \leftarrow false$;
**while**($j < m$);
```

Figure 4.2: The antisocial strategy

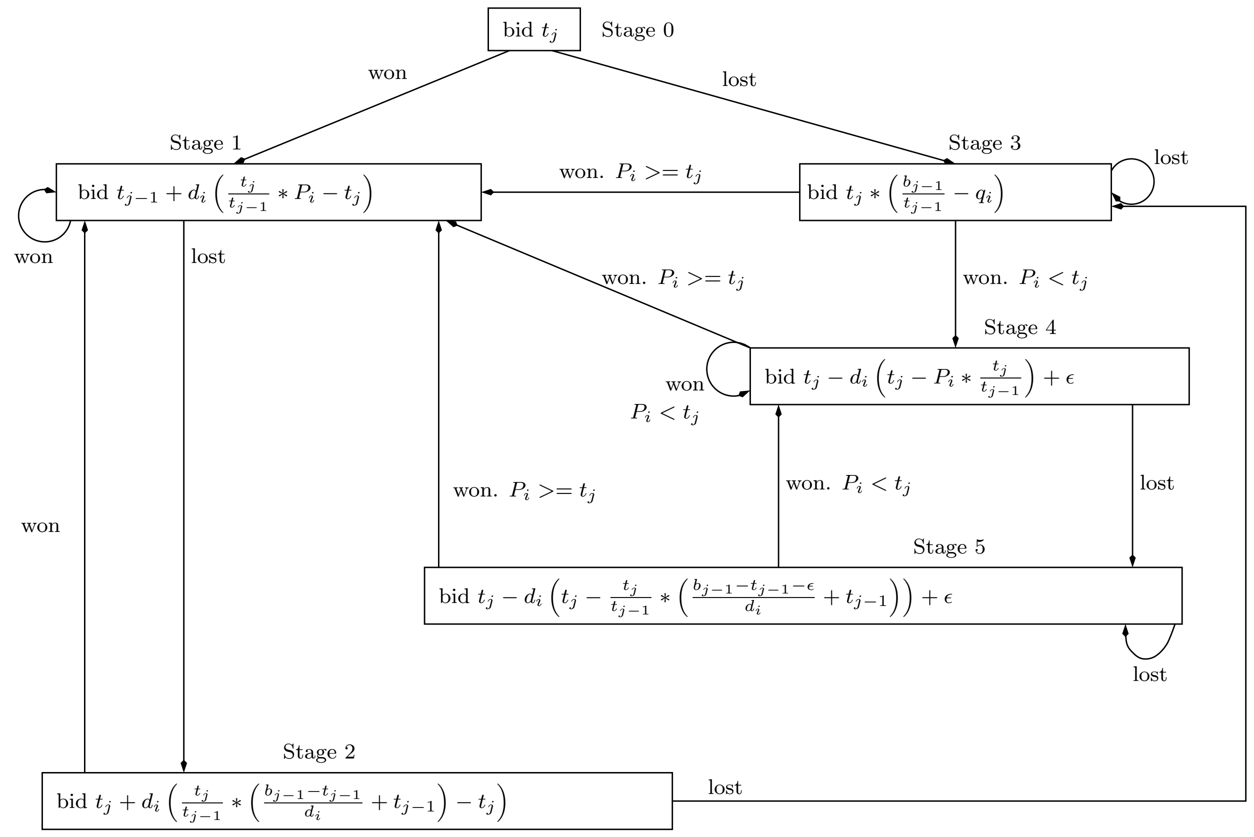

bid t_j
Stage 0
won
lost
Stage 1
bid $t_{j-1} + d_i\left(\frac{t_j}{t_{j-1}} * P_i - t_j\right)$
won. $P_i >= t_j$
Stage 3
bid $t_j * \left(\frac{b_{j-1}}{t_{j-1}} - q_i\right)$
lost
won
lost
won. $P_i >= t_j$
won. $P_i < t_j$
Stage 4
bid $t_j - d_i\left(t_j - P_i * \frac{t_j}{t_{j-1}}\right) + \epsilon$
won $P_i < t_j$
won. $P_i >= t_j$
won. $P_i < t_j$
lost
won
Stage 5
bid $t_j - d_i\left(t_j - \frac{t_j}{t_{j-1}} * \left(\frac{b_{j-1} - t_{j-1} - \epsilon}{d_i} + t_{j-1}\right)\right) + \epsilon$
lost
Stage 2
bid $t_j + d_i\left(\frac{t_j}{t_{j-1}} * \left(\frac{b_{j-1} - t_{j-1}}{d_i} + t_{j-1}\right) - t_j\right)$
lost

Figure 4.3: Antisocial strategy stages

itself is antisocial.

If the agent loses in Stage 0 then it transitions to Stage 3. In Stage 3, the antisocial agent reduces its bid by a small value anticipating to win. It keeps lowering the bid by a small percent q_i in each run until it wins. When the agent wins, it receives a payment P_i which is equal to the true value of the best agent. The transition is to Stage 4 or Stage 1 depending on whether the payment received, P_i, is less than or greater than its true valuation t_j. If the agent wins and receives a payment less than its true valuation (i.e., the agent is not the agent with the lowest true value) then it bids according to the strategies of the right area and vice versa.

In Stage 4 the agent bids a value which is between its own true valuation and the true value of the best agent, thus inflicting a loss to the best agent. If the agent wins in Stage 4 and the payment it receives is less than its true valuation, it remains in the same stage, otherwise goes to Stage 1. If the agent loses in Stage 4, it moves to Stage 5 because it does not receive any payment (so the calculation of the bid cannot be done according to Stage 4). In Stage 5, the agent calculates the bid of the best agent using the information from the previous bid (intrinsic in the equation for Stage 5) and then bids accordingly so as to inflict losses to the best agent. If the agent wins in Stage 5, it moves to Stage 4 or Stage 1 depending on whether the payment it receives is less than or greater than its own true valuation, respectively. If the agent looses in Stage 5, it remains in that stage where it can inflict losses on the best agent.

4.3.3 Effect on Makespan

As mentioned in the Section 4.3.2, the antisocial agent bids in such a way that in Stage 3, it wins the auction and the task is allocated to it. Since the actual time in which the machine corresponding to the antisocial agent can execute the task is greater than the time required by the best agent, it increases the makespan of the system (as compared to the makespan obtained when no antisocial agent is present). The increase is equal to the difference in the execution time of the antisocial agent and the best agent. Assuming that the task T_k is assigned to the antisocial agent A_{as} in Stage 3 and that the best agent is denoted by A_{best}, the increase in makespan due to the antisocial strategy will be $t_k^{as} - t_k^{best}$. However after this allocation, the antisocial agent moves to Stage 4 and then to Stage 5 where it will loose and it will not be allocated any further tasks. Thus the action of the antisocial agent will increase the makespan, but the increase will be negligible in the long run (considering a large number of tasks).

4.4 Experimental Results

To analyze the proposed antisocial strategy, we developed a discrete event simulator and simulated the mechanism and the antisocial strategy. The simulation was run with different values for some of the parameters in order to study their effect on the losses that can be inflicted by an antisocial agent on the other participating agents. The simulation

Table 4.1: Summary of parameter values used in the simulation

Parameter	Range/Value	Parameters kept constant
Number of agents (n)	16	-
ϵ	0.01% of task size	-
Step down percent (q_i)	5%	-
True values (t^i_j)	normal($0 \ldots 2$)	-
Antisocial position (π_i)	$1 \ldots 16$	t^i_j
Derogation rate (d_i)	$0 \ldots 1$	t^i_j, π_i
Task size (r_j)	uniform($0 \ldots 100$)	t^i_j, π_i, d_i
Number of tasks (m)	10000	t^i_j, π_i, d_i

environment and the results are presented in this section. In the following we use π_i to denote the position of the antisocial agent A_i in the sequence of agents sorted in increasing order of their valuations. For example $\pi_i = 1$ means the agent with the lowest true value, $\pi_i = 2$ means agent with the second lowest true value and so on.

4.4.1 Simulation Environment

The simulation was run with different combinations of parameters (presented in Table 4.1) to study their effect on the agents' losses. The parameters that were varied were the true values of the agents (t_i), the antisocial agent position (π_i), and the derogation rate (d_i). In the simulation, for a particular set of parameters, the mechanism allocates the tasks considering a normal scenario (no antisocial agent) and then considering the presence of one antisocial agent. The data related to the experiments (i.e., values of parameters, allocation, payments, etc.) in both cases is recorded.

We first simulated the strategy by considering a task scheduling scenario in which 16 agents are participating. To confirm the validity of the strategy when large number of agents are involved, we also simulated the strategy considering 128 agents. The results of the experiments involving 128 agents are presented in the second part of Section 4.2. The tasks are to be executed by the agents (resources) and the allocation is done according to the MMW mechanism presented in Section 4.2.5. The value of ϵ (a chosen infinitesimal quantity) was set to 0.01% of the task size (ϵ is taken relative to the task size to remove scaling errors). Also the step down percent q_i was set to 5%. For each simulation experiment one set of true values of the agents were generated according to the normal distribution. For a particular set of true values, the simulation was run for different antisocial positions ($\pi_i = [1 \ldots 16]$) (i.e., making the agent with the lowest true value as antisocial, then the second lowest as antisocial and so on). For a particular set of true values and antisocial position, the derogation rates for the antisocial agent were varied from 0 to 1 in steps of 0.1. For a particular set of true values, antisocial position and derogation rate the simulation was run for 10^5 tasks of different sizes. The task sizes were generated according to the uniform distribution. Since we are not proposing a new scheduling algorithm, the uniform distribution is sufficient to help us capture the economic aspect of the antisocial strategy. For this reason, we do not consider scheduling benchmarks. Moreover, as pointed out in [20], an analysis using randomly generated tasks is encouraged.

The recorded data is used to calculate the relative loss RL (as given in Equation 4.3), which the antisocial agent was able to inflict on the best agent. Since an antisocial agent at a particular antisocial agent position is able to inflict different amounts of relative loss to the winning agent at different derogation rates, we have recorded the maximum and the average relative loss percent that agent is able to inflict (average is taken over varying derogation rates). Similarly at a higher level, for a particular set of bids, when agents at different positions are antisocial they are able to inflict different amounts of losses to the winning agent so we have recorded the maximum and average relative loss percent an agent is able to inflict (averaged over different antisocial positions).

4.4.2 Results

The following results were achieved by running the simulation:

(i) The proposed antisocial strategy was experimentally validated.

(ii) The effect of derogation rate on the amount of inflicted loss was analyzed. The greater the derogation rate, the greater the loss inflicted by the antisocial agent.

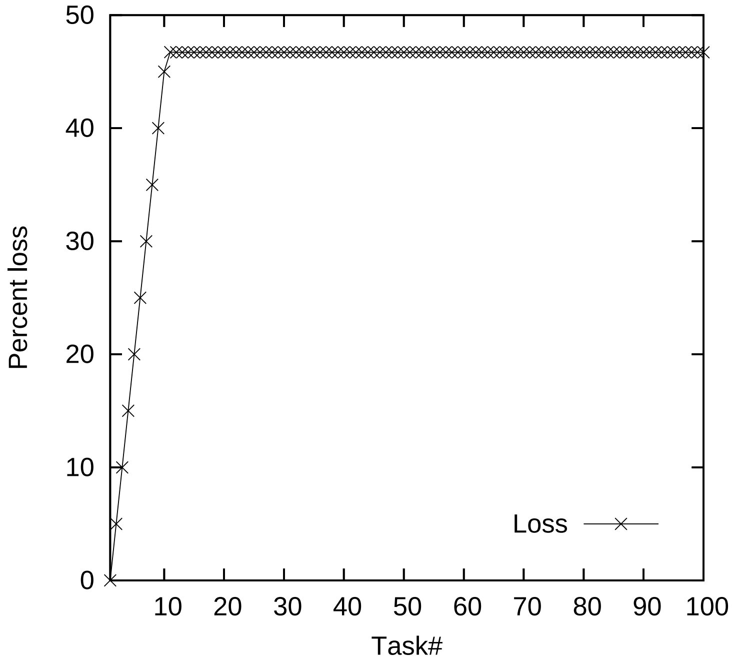


Figure 4.4: Relative loss RL (in percent) for each task

(iii) The effect of the antisocial agent position on the amount of inflicted loss was analyzed. The less π_i is ($\pi_i \neq 1$), the greater the average loss inflicted. The maximum loss is the same for all π_i (achieved at $d = 1$).

(iv) The effect of the true value on the amount of inflicted loss was analyzed. The bigger the difference in the true values of the best agent and the second best agent, the bigger the relative loss to the best agent.

Figure 4.4 shows the relative loss (as percent) inflicted by an antisocial agent. The figure shows the losses for the first one hundred tasks only, rather than the whole set of 10000 (since the relative loss percent becomes constant after first few tasks). The agent starts by bidding its true valuation and decreases its bid (thus increasing the relative loss to the best agent) until it is allocated the task and thus comes to know the valuation of the best agent (task 10). After this the antisocial agent keeps bidding according to Stage 5 and it steadily keeps inflicting losses on the best agent for the rest of the tasks. It is notable here that the agent learns the true valuation of the best agent only after 10 runs, which is quite negligible as compared to the number of tasks. The maximum loss inflicted is about 47%

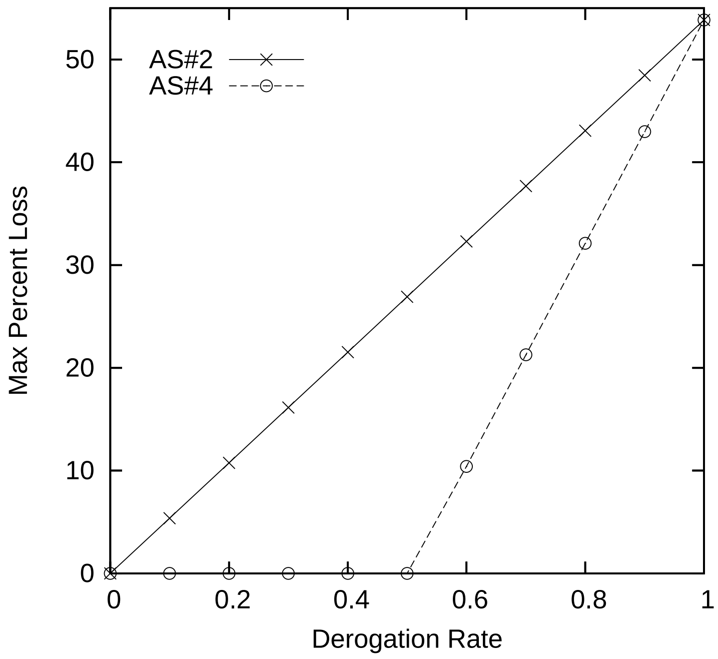


Figure 4.5: Max. relative loss inflicted vs. derogation rate (antisocial agents at position 2 and 4)

for the particular set of values of the parameters considered in Figure 4.4.

In Figure 4.5 we present the maximum relative loss inflicted to the best agent in two cases in which agent at position π_i= 2 and π_i=4 are antisocial. It can be seen that if we consider the second agent as antisocial, then by reducing its bid, it is always able to inflict losses. The loss percent grows with the derogation rate of the agent. If an agent other than the second best agent is antisocial, it is able to inflict losses only when its bid is less than the second best agent's bid, otherwise it is not (even if it knows the valuation of the best agent). As can be seen from the figure, the antisocial agent in position 4 ($\pi_i = 4$) is able to inflict losses only after its derogation rate is 0.6. However all the agents inflict maximum losses when their derogation rate is 1, i.e., when they are purely destructive.

As it can be seen in Figure 4.6, the maximum relative loss any agent can inflict on the best agent is equal to the percent difference between the valuations of the best agent and the second best agent (in this case 54%). Also the average loss an agent can inflict (with different derogation rates) decreases as the position of the antisocial agent increases (i.e., the

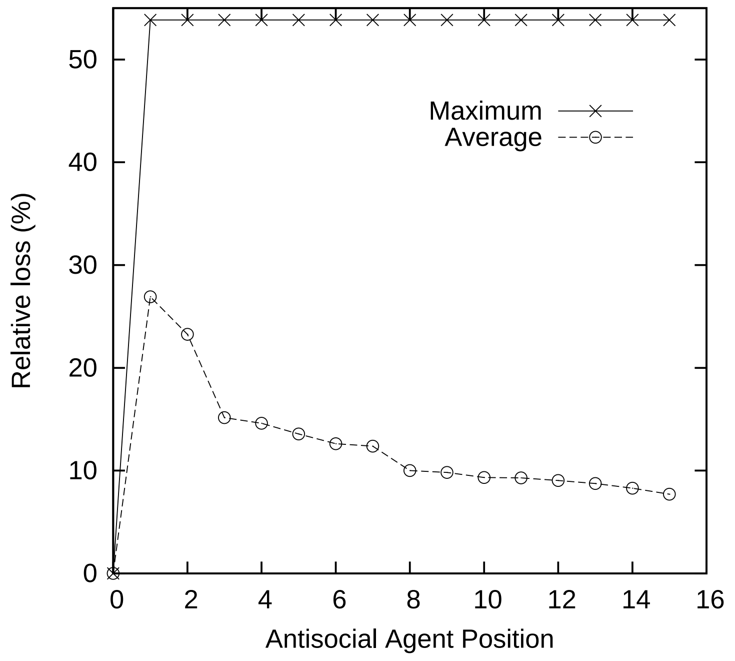


Figure 4.6: Maximum and average relative loss inflicted vs. antisocial agent position (n=16)

higher the antisocial agent position, the lower the average loss). This is due to the fact that the antisocial agent is able to inflict losses only when its derogation rate is sufficiently high thus making the bid of the antisocial agent less than the bid of the second best agent.

In the following we present results for a larger system, composed of 128 agents. Figure 4.7 shows that the pattern of losses inflicted on the best agent is the same even with large number of agents. This is attributed to following facts. We mentioned earlier in this section that there are three factors which determine the losses an antisocial agent is able to cause to the best agent. They are: a) the derogation rate; b) the true values of the agents; and c) the antisocial position π_i. At high derogation rates the antisocial agent bids extremely close (just a little higher) to the winning agent, which inflicts maximum loss to the winning agent. Thus at high derogation rates and independent of the position π_i, the antisocial agent will be able to inflict losses to the winning agent. Also it is expected that as the number of agents increases, there will be more agents that bid close to the bid of the winning agent. In this case the agents can still inflict losses which could be as much as the difference in the bids of the best agent and the second best agent. This amount can be significantly high in

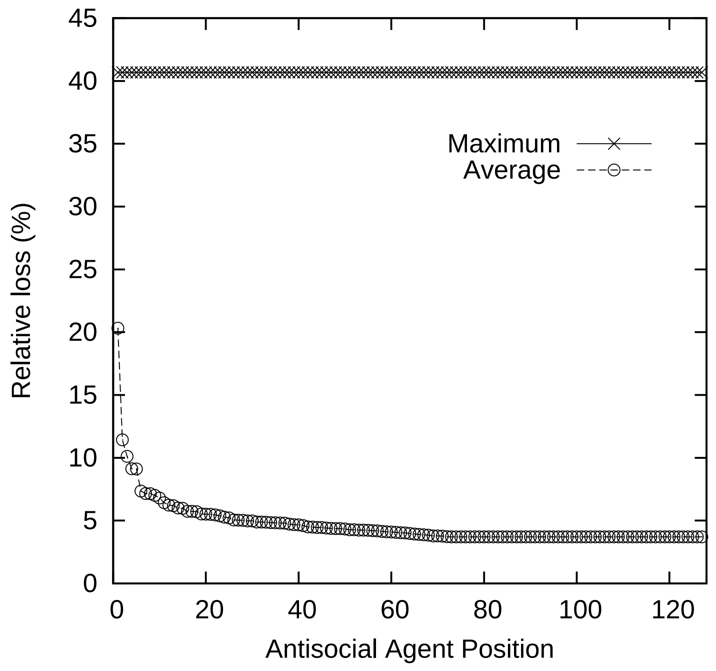


Figure 4.7: Maximum and Average relative loss inflicted vs. antisocial agent position (n=128)

highly heterogeneous environments. Moreover in our problem, a large number of tasks are allocated to the best agent over time, thus even if the loss inflicted by an agent seems to be less in one round, it will accumulate over time and it will become significantly larger, fulfilling the aim of the antisocial agent. In Figure 4.7, the average loss at higher antisocial positions is less because those agents are able to inflict losses only when their derogation rates are 1. However if they wish they can keep their d high to inflict significant losses up to the 'maximum' value.

The increase in makespan due to the antisocial behavior was negligible. In a typical experiment, the increase in makespan was 0.612 time units, which was only 0.00088% of the total time (makespan).

4.5 Summary and Future Work

As can be seen from the results presented in this chapter, the presence of an antisocial agent in the task scheduling scenario can inflict losses on the other agents. This is due to the second price payment policy followed by the mechanism we considered. The antisocial strategy presented in this chapter can be applied by an agent who wants to inflict losses on the other agents. We analyzed the effect of different parameters on the antisocial strategy. In particular, agents with high derogation rates inflict more losses on the other agents. Also the amount of loss depends on the difference in true values of the agent with the lowest valuation, the agent with the second lowest valuation, and the antisocial agent (if different from above two). The amount of loss that can be inflicted also depends on the number of agents having the true values lying in between the true values of the best agent and the antisocial agent.

We plan to consider the presence of two or more antisocial agents in the system and analyze the effect of their strategies on the profit of other agents. We also plan to develop strategies for non-antisocial agents which can spare them from losses inflicted by the antisocial agents.

CHAPTER 5: DECEPTION IN HONEYNETS: A GAME-THEORETIC ANALYSIS

In this chapter we present a game theoretic framework for modeling and analyzing deception in honeynets. We propose and analyze a *strategic game model* for the honeynet system. We determine the equilibrium solutions for this model and analyze the corresponding strategies of the attacker and the defender. We also propose the use of *extensive games of imperfect information* to analyze deception strategies in honeynets. We show how the proposed game can be used to model the attacker-honeynet interactions. We analyze the mixed strategy equilibrium solutions of the proposed game and characterize the deception moves of the defender and the corresponding attacker's actions. To our knowledge, this is the first work proposing a model of the attacker-honeynet interactions based on an extensive game of imperfect information.

5.1 Introduction

A honeypot as defined by the Honeynet Project [74] is "an information system resource whose value lies in unauthorized access or illicit use of that resource." In this dissertation we consider a honeypot to be a host that is left unprotected and available to the attackers. By extension we consider that a honeynet is composed of a set of interconnected honeypots. These systems are carefully monitored in order to learn new vulnerabilities, tools and techniques used to gain access. Honeynets are useful for analyzing large-scale attacks such as worms, viruses and botnets and have little value for analyzing specialized attacks targeted to a particular host.

Recently with the increase in the frequency of Internet attacks several research groups and organizations developed and deployed honeynet systems. The idea behind honeynets is the use of deception, a classical counter intelligence technique which was and it is still used extensively by the intelligence organizations. The computer security community began considering this idea only recently, [31] being the first known reference mentioning it.

Depending on the level of interactions between the attacker and the monitored system, honeynets can be classified into two categories: low-interaction honeynets and high-interaction honeynets. *Low-interaction honeynets* simulate a minimal set of system functionalities such as the network stack. The main advantage of low-interaction honeynets is their scalability, allowing the monitoring of hundreds of thousand of IP addresses. The major disadvantage is that they cannot monitor in detail the attacker's actions once the host is compromised. *High-interaction honeynets* simulate all the functionalities of a real system thus allowing the gathering of much richer information on the attacker's actions. The disadvantage of using high-interaction honeynets is their high cost and difficult management. When the honeypot is a real machine with its own IP address we call it a *physical honeypot*. If the machine is simulated by a another machine then we have a *virtual honeypot*. The physical honeypots are very expensive to install and maintain and are not scalable, while the virtual honeypots are scalable and inexpensive.

Attackers have certain ways to probe the network attempting to identify the honeypots [21]. Many current honeynet deployments do not use specific strategies to prevent the mapping of honeypots by the attackers. When a host is probed, it gives some response. This response is valuable to the attacker as she gains information about the honeynet. One way a defender can avoid the mapping of honeypots is to cleverly manipulate the response to the probe and deceive the attacker. Our goal in this chapter is to provide a game theoretical framework for modeling and characterizing deception in honeynets.

5.1.1 Related Work

Low-interaction honeypots usually monitor a large range of IP addresses and they have been very successful in identifying large scale worm outbreaks [100], understanding distributed denial-of-service attacks [101] and creating intrusion detection signatures [87]. "Network telescopes" [102, 113, 155] are examples of successful deployments that passively monitor inbound packets without completing the TCP handshake. iSink [155] is a measurement system deployed to monitor background radiation which employs active responders who emulate the real network behavior an attacker expects. Honeyd [118] is a low-interaction system that simulates the networking stack of different computer systems. In order to simulate real networks it creates virtual networks consisting of arbitrarily routing topologies.

The simplest way to implement high-interaction honeypots is to have individual servers for each monitored IP address. Since this approach is very expensive recently several researchers proposed the use of virtual machine environments to implement multiple honeypots on a single server [148, 154]. These systems are easy to manage and have a reduced deployment cost while they provide all functionalities of a real system. Vrable *et al.* [148] presented Potemkin, a prototype honeynet that uses the virtual machine approach supporting over 64,000 honeypots using only few physical servers.

Deception techniques for defending information systems have been investigated by several researchers. Cohen [35] provides a comprehensive discussion of deception issues for information protection. Among the conclusions of the above study is that deception is a valuable tool for intrusion detection and it has a positive effect on the defense and a negative effect on the attackers. The author stressed the need for rigorous mathematical analysis of deception in information protection. Cohen and Koike [36] shows how deception can be used to control the path of a computer system attack. Deception Toolkit (DTK) [35] is a collection of scripts that allows the emulation of different known vulnerabilities in order to deceive the attackers. Rowe *et al.* [128] suggested the use of fake honeypots to scare away possible attackers. They also discuss the escalation of honeypot anti-honeypot techniques. Rowe [127] presented several tools for evaluating honeypot deceptions.

5.1.2 Organization

The rest of the chapter is structured as follows. In Sect. 5.2 we describe a prototype strategic form game that models the attacker and the honeynet system. In Sect. 5.3 we propose the honeynet deception games in extensive form and discuss their properties.

5.2 A Prototype Honeynet Game in Strategic Form

We consider the attacker and the honeynet system as players in a strategic noncooperative game. We denote the *attacker* by A and the *defender*, which is the honeynet system, by D. The honeynet is assumed to be composed of k honeypot hosts out of n possible hosts within a block of IP addresses. The attacker must choose to attack one of these hosts. The goal of the attacker is to attack a host that is not a honeypot while the goal of the defender is to have a honeypot attacked by the attacker. In this initial model we assume that the attacker and the defender make their decisions independently and without any knowledge of the opponent player's moves. We call the game that models this situation a (n, k)-*honeynet game*, denoted as $HG(n, k)$.

Definition 5.2.1 A (n, k)-*honeynet game* consists of:

(i) the set of players $\mathcal{N} = \{A, D\}$.

(ii) the set of actions for each player:

  (a) $A_D = \{(x_1, x_2, \ldots, x_n) |\ k$ vector components are 1 and $n - k$ components are $0\}$. If $x_i = 1$ then D places a honeypot at position (address) i. If $x_i = 0$ then D places a regular host at position i.

  (b) $A_A = \{j | j = 1, \ldots, n\}$. A chooses to attack host j.

(iii) the payoff functions of each player:

$$u_A((x_1, x_2, \ldots, x_n), j) = \begin{cases} -c_1 & \text{if } x_j = 1, \\ c_2 & \text{if } x_j = 0 \end{cases} \tag{5.1}$$

$$u_D((x_1, x_2, \ldots, x_n), j) = \begin{cases} c_1 & \text{if } x_j = 1, \\ -c_2 & \text{if } x_j = 0 \end{cases} \tag{5.2}$$

where $c_i > 0$, for $i \in \{1, 2\}$.

The parameter c_1 represents the payoff gained by the defender and also the loss incurred by the attacker if the attacker attacks a honeypot. The parameter c_2 represents the payoff gained by the attacker and also the loss incurred by the defender if the attacker attacks a regular host.

As an example we consider a $(2, 1)$-honeynet game, $HG(2, 1)$. The payoff matrix for this game is given below. In this matrix the row player is A and the column player is D. The first component of the payoff pairs given in the matrix represents the payoff to A while the second component represents the payoff to D.

$$\begin{array}{c|c|c|}
 & (1,\,0) & (0,\,1) \\ \hline
1 & -c_1,\, c_1 & c_2,\, -c_2 \\ \hline
2 & c_2,\, -c_2 & -c_1,\, c_1 \\ \hline
\end{array}$$

One solution concept for strategic games is the Nash equilibrium [105], [106]. An *action profile* is a tuple with an entry containing the action for each player. An action profile is a *Nash equilibrium* when no player can do better by altering her action.

Definition 5.2.2 (Nash equilibrium) *[112] An action profile $a^* = (a_1^*, a_2^*, \ldots, a_n^*)$ is a* Nash equilibrium, *if*

$$u_i(a^*) \geq u_i(a_i, a_{-i}^*) \text{ for every action } a_i \text{ of player } i, \tag{5.3}$$

where a_{-i}^* is an action profile without player i, i.e., $a_{-i}^* = \left(a_1^*, \ldots, a_{i-1}^*, a_{i+1}^*, \ldots, a_n^*\right)$.

Any change of action by player i will reduce her payoff and consequently she will choose not to alter her action.

As can be seen from the payoff matrix above, $HG(2, 1)$ does not have a Nash equilibrium in pure strategies. Thus we need to consider mixed strategies since for a finite strategic game we will always be able to find a Nash equilibrium in mixed strategies [112]. A mixed strategy of a player is a probability distribution over the player's pure strategies. In the following we formally define the concept of Nash equilibrium in mixed strategies.

Definition 5.2.3 (Mixed strategy Nash equilibrium) *[112] The mixed strategy profile α^* is a* mixed strategy Nash equilibrium *if $u_i(\alpha^*) \geq u_i(\alpha_i, \alpha^*_{-i})$, for every mixed strategy α_i of player i, where α_{-i} is the action profile without player i, i.e., $\alpha^*_{-i} = \left(\alpha^*_1, \ldots, \alpha^*_{i-1}, \alpha^*_{i+1}, \ldots, \alpha^*_n\right)$.*

A pure strategy is a mixed strategy with an action that has probability 1.

In the case of $HG(2,1)$ we denote by a_i, $i \in \{1,2\}$ the pure strategy of A in which A attacks host i, and by d_i, $i \in \{1,2\}$ the pure strategy of D in which D places a honeypot at host i (*i.e.*, $d_1 = (1,0)$ and $d_2 = (0,1)$). We denote by $\alpha_A(a_i)$ the probability assigned by the mixed strategy α_A of the attacker to its strategy a_i; and by $\alpha_D(d_i)$ the probability assigned by the mixed strategy α_D of the attacker to its strategy d_i. It can be shown that the Nash equilibrium in mixed strategies is given by $(((\alpha_A(a_1), \alpha_A(a_2)), (\alpha_D(d_1), \alpha_D(d_2))) = ((\frac{1}{2}, \frac{1}{2}), (\frac{1}{2}, \frac{1}{2}))$. The Nash equilibrium in mixed strategies of $HG(2,1)$ corresponds to the intuitive solution in which the attacker is attacking host 1 and host 2 with equal probability and the defender is placing the honeypot at host 1 or host 2 with equal probability.

The equilibrium analysis of the general game $HG(n,k)$ is as follows. A has n pure strategies $\{a_1, \ldots, a_n\}$, where strategy a_i means A attacks host i. D has $m = \binom{n}{k}$ ways to place k honeypots at n hosts, where $\{x_1, x_2, \ldots, x_n\}$ is the vector denoting the placement of honeypots. We denote the pure strategies of D by $d_j, j = \{1, \ldots, m\}$. Using the payoff function in Definition 5.2.1 we see that there is no Nash equilibrium in pure strategies. Intuitively when A attacks node j, D would obtain a higher payoff if she chooses a strategy d_i, in which $x_j = 1$. However in that case A is better off attacking some other node $l \neq j$. Thus there is no pure strategy equilibrium for $HG(n,k)$. To calculate the mixed strategy equilibria of $HG(n,k)$, we consider the probabilities assigned to each pure strategy by the players (as mentioned above in $HG(2,1)$ example). Let c^D_{ij} denote the payoff of D when A selects strategy a_j and D selects strategy d_i. The expected payoff E^D_i received by D when playing d_i is $E^D_i = \sum_{j=1}^{n} \alpha_A(a_j) c^D_{ij}$

The total expected payoff of D will be:

$$E^D = \sum_{i=1}^{m} \alpha_D(d_i) E_i^D \tag{5.4}$$

It can be seen that to maximize E^D, D has to play all $\alpha_D(d_i)$ equally, because of the symmetrical payoffs. This means that at the equilibrium in mixed strategies the probability $\alpha_D(d_i)$ with which D selects strategy d_i is equal to $1/m$, *i.e.*, $\alpha_D(d_i) = 1/m, \forall i = 1, \ldots, m$. Intuitively, we can see that if D can increase its payoff by increasing the probability of playing strategy d_i and decreasing the probability of playing strategies d_i' then, it will be able to maximize its payoff by setting $d_i' = 0$ and $d_i = 1$, which will be a pure strategy. However, we have already seen that there is no pure strategy equilibrium for this game. Thus the mixed strategy Nash equilibrium for this game is when both players, the attacker and the defender, play all their strategies with equal probabilities.

Since this is a one-shot game the attacker and the defender choose their strategies independently and so the defender cannot influence the strategies of the attacker. The games proposed in this section do not consider deception strategies, but they are the prototype games that will be extended in the next section into games that take into account deception moves.

5.3 Honeynet Deception Games

We extend the games studied in the previous section to allow deception moves by the defender (honeynet). Examples of deception moves are: make it evident to the attacker that the system is a honeypot but in fact it is a regular host (we call these "fake honeypots"), or hide the honeypot such that the attacker believes it is a regular host. These can be achieved by different techniques some of them being described in [128, 127]. In this section we propose a game theoretic model considering two players, the Attacker and the Defender (honeynet). As in the previous section we assume that the honeynet is composed of k honeypots out of n possible hosts within a block of IP addresses.

As mentioned earlier, the attacker probes the hosts and analyzes their responses to identify honeypots. A clever defender will respond to the probes in a way that the attacker does not gain information on whether the host is a regular one or a honeypot. The response of the host depends on whether the host is a honeypot (different responses by low-interaction and high-interaction honeypots) or a regular host. The response of the honeypot may be generated such that to conceal the identity of the honeypot. However, it is safe to assume that after a few probes an attacker can successfully identify that a host is a honeypot, *i.e.,* the defender cannot infinitely deceive the attacker in believing a host to be regular host, when in fact it is a honeypot.

The goal of the attacker is to identify and attack a host that is not a honeypot while the goal of the defender is to conceal the honeypot (make it appear as a regular host) and have the honeypot attacked by the attacker. Even if the attacker does not attack the honeypot, the defender gathers crucial information if it can make the attacker probe the honeypot as much as possible.

In the proposed model the attacker and the defender *make their decisions sequentially* by considering the opponent player's moves. We model this scenario as a game with sequential moves. The first move is made by the defender, in which she decides how to place the k honeypots among the n possible hosts. There are $\binom{n}{k}$ different ways in which k honeypots can be placed into n positions. Next the attacker decides what host to probe out of the n possible hosts. The next move is by the defender, who responds to the probe of the attacker. In our model the defender has two types of responses, 'r' and 'h' where response 'r' means that the host being probed is a regular host and the response 'h' means the host being probed is a honeypot. It should be noted that these responses may not represent the actual state of the host, *i.e.*, the defender may lie to deceive the attacker. Depending on the response of the attacker, she may choose to probe the same host again or probe some other host. The game continues with alternative moves by the attacker and the defender in the form of probes and responses, until the attacker probes and the defender produces responses γ times. After γ

stages (probe-response) the attacker makes the final move in which she chooses to either attack a host or not attack any host. In the intermediate stages of the game the payoffs are computed based on whether the attacker probes a honeypot or a regular host. The payoff of the attacker and the defender in the final move (by the attacker) are based on whether the attacker attacks a honeypot or a regular host or chooses not to attack. The total payoff is the sum of the payoffs of intermediate moves and the final move.

In the above game the final move by the attacker is made after γ probes to the hosts, independent of the host being probed. Another scenario which can be considered is the one in which the attacker makes the final move after she has probed a host at least γ times. In that case the total number of probes made by the attacker will be greater than or equal to γ. In the following, we consider only the first scenario, in which the attacker makes the final move after γ probes, irrespective of the host probed. We call the game that models this situation a (n, k, γ)-*honeynet deception game*, denoted as $HDG(n, k, \gamma)$.

Since a strategic form game model does not capture the sequential nature of the decision making process, we will model the interactions between the attacker and the system as an extensive form game. An *extensive form game* is a dynamic model that expresses the sequential structure of decision making. In order to have a clear and manageable way to explain the concepts used in our game theoretic modeling we first present a particular version of the general game, $HDG(2, 1, 1)$, in which after one probe the attacker decides to attack one of the hosts or not. Later on we will describe the general game.

Fig. 5.1 depicts a $HDG(2, 1, 1)$ game in extensive form. In this figure the moves are indicated above the horizontal lines. The game has four stages:

(i) D1 - The defender (D) decides the position of the honeypot to be at host 1 or host 2 (denoted by actions 10 and 01, respectively). This information is not known to the attacker, thus $HDG(2, 1, 1)$ is a game of imperfect information.

(ii) A2 - The attacker (A) probes one of the hosts (1 or 2).

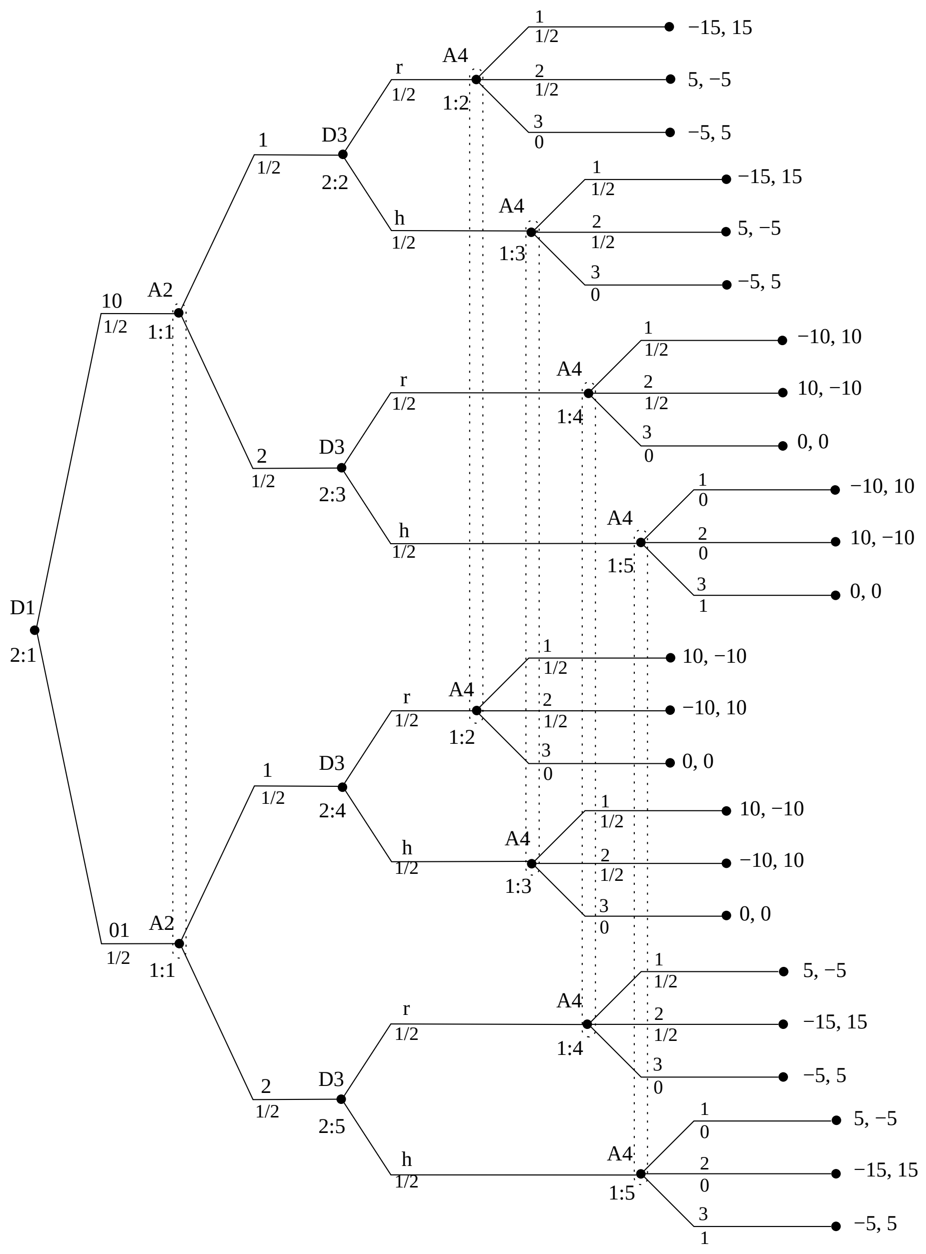


Figure 5.1: HDG(2,1,1) game

(iii) D3 - The defender produces a response to the probe of the attacker ('r'=regular host, or 'h'=honeypot).

(iv) A4 - The attacker either attacks one of the hosts (1 or 2) or does not attack at all (3).

The strategies of the defender are as follows:

(i) *Deception strategies*: When a host is probed, respond with 'r' when the host is an actual honeypot (*i.e.*, conceal the honeypot), or respond with 'h' when the host is an actual regular host (*i.e.*, show a fake honeypot).

An example of such strategy is when D places a honeypot at host 1 *i.e.*, chooses move (10), A probes host 1 and D responds with 'r'.

(ii) *Disclosure strategies*: When a host is probed, respond with the actual status of the host, *i.e.*, with 'r' when the host is an actual regular host and with 'h' when the host is an actual honeypot.

An example of such strategy is when D places a honeypot at host 1 *i.e.*, chooses move (10), A probes host 1 and D responds with 'h'.

A series of actions in an extensive game is called a *sequence*. Formally $B = (a_1 \dots a_n)$ is a *sequence* if action a_i was taken during stage i. The game in Fig. 5.1 has 38 such sequences: (10), (01), $(10, 1)$, $(10, 2)$, $(01, 1)$, $(01, 2)$, $(10, 1, r)$, $(10, 1, h)$, ..., $(01, 2, h, 2)$, $(01, 2, h, 3)$. The sequence $(10, 1, r, 3)$ indicates that D places a honeypot at host 1, A probes host 1, D replies that host 1 is a regular host and then A does not attack any host.

A sequence has *histories*. The sequence $(10, 1, r, 3)$ has five *subhistories*: $\emptyset$, (10), $(10, 1)$, $(10, 1, r)$, and $(10, 1, r, 3)$. The histories $\emptyset$, (10), $(10, 1)$ and $(10, 1, r)$ are *proper subhistories* of $(10, 1, r, 3)$; subhistory $(10, 1, r, 3)$ is a *terminal history*. The terminal histories are annotated with preferences. The *preference* of terminal history l is a tuple of payoffs $u_i(l)$, one for each player. The *preference* tuple indicates the payoff of A and D. The first component of the tuple in Fig. 5.1 is the payoff to A while the second component is the payoff to D. The *payoff function* $u_i(l)$, $i \in \{A, D\}$ gives the preference of the players given terminal history l. Player i, $i \in \{A, D\}$ prefers terminal history l' to l, if $u_i(l') > u_i(l)$ and is indifferent between histories l' and l if $u_i(l') = u_i(l)$. In the game represented in Fig. 5.1, D prefers history $(10, 1, r, 1)$ to $(10, 1, r, 2)$ as $u_D(10, 1, r, 1) > u_D(10, 1, r, 2)$.

$HDG(n, k, \gamma)$ is a game with imperfect information since all the players do not have precise information about the previous actions of all the other players at all stages of the game. Specifically, the attacker does not know how the defender has placed the honeypots in the first stage of the game. To represent this lack of information in the model, the notion of *information set* is used. Let us denote by H_i the set of histories after which player i moves. An information set is a partition of H_i. The collection of information sets is called the *information partition* of player i. When making a move, player i is informed about the information set that has occurred but not about the exact history within that set. In Fig. 5.1 the information set is given below the stage label in the format $x : y$ where x refers to the player number ($x = 1$ is the attacker and $x = 2$ is the defender) and y denotes the number of the information set. In $HDG(2, 1, 1)$ game, the attacker knows the information set $1 : 1 = \{(10), (01)\}$ after stage $D1$, but not which one of the two histories, (10) and (01), has occurred. Formally an extensive-form game can be defined as follows:

Definition 5.3.1 (Extensive-Form Game with Imperfect Information) *[111]*
An extensive-form game with imperfect information *consists of the following:*

(i) The set of players;

(ii) The set of terminal histories;

(iii) A function $P(l)$ that assigns a player to the move after subhistory l;

(iv) For each player, the information partition (a partition of the set of histories assigned to that player)

(v) Preferences over the set of lotteries over terminal histories, for each player.

We formally define the $HDG(2, 1, 1)$ game in extensive form (Fig. 5.1) as follows:

(i) $\{A, D\}$ is the set of players;

(ii) $\{(10, 1, r, 1)$, $(10, 1, r, 2)$, $(10, 1, r, 3)$, $(10, 1, h, 1)$, $(10, 1, h, 2)$, $(10, 1, h, 3)$, $(10, 2, r, 1)$, $(10, 2, r, 2)$, $(10, 2, r, 3)$, $(10, 2, h, 1)$, $(10, 2, h, 2)$, $(10, 2, h, 3)$, $(01, 1, r, 1)$, $(01, 1, r, 2)$,

$(01,1,r,3)$, $(01,1,h,1)$, $(01,1,h,2)$, $(01,1,h,3)$, $(01,2,r,1)$, $(01,2,r,2)$, $(01,2,r,3)$, $(01,2,h,1)$, $(01,2,h,2)$, $(01,2,h,3)\}$ is the set of terminal histories;

(iii) The function $P(h)$ such that $P(D1) = D, P(A2) = A$, $P(D3) = D$, and $P(A4) = A$, where D1, A2, D3 and A4 are the stages of the game as described earlier;

(iv) Information partitions: All histories after which the defender moves are contained in individual information sets, which means that the defender observes all actions of the attacker; the attacker's information partition after stage D1 consists of the histories {(01),(10)} and after stage D3 consists of {{(10,1,r),(01,1,r)}, {(10,1,h),(01,1,h)}, {(10,2,r),(01,2,r)} and {(10,2,h),(01,2,h)}}, which means that at any point in the game, the attacker cannot distinguish which host is a honeypot and which one is a regular host.

(v) The preferences over the terminal histories are calculated according to Table 5.1. The final preferences are the sums of the intermediate payoffs after stages A2 and A4. For example, consider terminal history (10,1,r,2). The defender places the honeypot at host 1 and the attacker probes host 1, thus the attacker obtains an intermediate payoff of -5 and the defender obtains an intermediate payoff of 5. However, in the final move the attacker attacks host 2 and thus obtains a payoff of 10 in that stage, making her total payoff of $-5 + 10 = 5$. The defender obtains -10 after stage A4 and her total payoff is $5 - 10 = -5$.

As we mentioned earlier, the objective of a player is to maximize her payoff. Players are not isolated; the actions of the defender influence and are influenced by the actions of the attacker. Let $A(I_i)$ denote the actions available to player i at information set I_i. A function which assigns an action $a_i \in A(I_i)$ to each of i's information sets I_i is called a *pure strategy* of player i in an extensive game. A *mixed strategy* of a player in an extensive game is a probability distribution over the player's pure strategies. We denote the mixed strategy (α_i) of player $i = \{A, D\}$ at information set I_i as a vector of probabilities, where each probability

Action	A's payoff	D's payoff
A probes a honeypot	-5	5
A probes a regular host	0	0
A attacks a honeypot	-10	10
A attacks a regular host	10	-10
A chooses not to attack	0	0

Table 5.1: Payoffs for $HDG(n, k, \gamma)$ game

corresponds to the available actions in the order in which the actions are given in the game definition. Thus when the defender has a mixed strategy of playing both the moves ('r' and 'h') at information set $2 : 2$ will be denoted by $\alpha_D(2 : 2)$.

The solution concept for extensive form games that we consider in this study is the mixed strategy Nash equilibrium. This equilibrium is a steady state in which no player can unilaterally increase her payoff. It is formally defined as follows:

Definition 5.3.2 (Nash equilibrium of extensive game) *[111] The mixed strategy profile α^* in an extensive game is called a* Nash equilibrium in mixed strategies *if, for each player i and every other mixed strategy α_i of player i, player i's expected payoff to α^* is at least as much as his expected payoff to $(\alpha_i, \alpha^*_{-i})$ where α_{-i} denotes the strategy profile of all players except player i.*

The Nash equilibria of an extensive game are often computed by constructing the strategic form of the game and analyzing it as a strategic game [111]. We used Gambit [98], a software tool for solving games, to compute the equilibria of $HDG(2, 1, 1)$. Gambit computed several Nash equilibria in mixed strategies. We will present and discuss only one of these equilibria of the $HDG(2, 1, 1)$ game which is shown in Fig. 5.1. In the figure the probability of an action being played in this particular mixed strategy equilibrium is shown below the horizontal lines representing the actions of the players. The mixed strategy equilibrium shown in the figure can be represented as $\alpha_D(2 : 1) = \alpha_D(2 : 2) = \alpha_D(2 : 3) = \alpha_D(2 : 4) = \alpha_D(2 : 5) = (1/2, 1/2)$, $\alpha_D(1 : 1) = (1/2, 1/2)$, $\alpha_D(1 : 2) = \alpha_D(1 : 3) = \alpha_D(1 : 4) = (1/2, 1/2, 0)$, $\alpha_D(1 : 5) = (0, 0, 1)$. This strategy implies that at stage D1, the defender chooses one of the configurations with equal probability (both 1/2), because she has the same expected payoff for both actions. The attacker probes one of the two hosts with equal probability 1/2, since she does not have any information about the placement of the honeypots. At stage D3, the defender plays the deception strategy with probability 1/2. This means that when a regular host is probed, the defender responds with 'h' half of the time to deceive the attacker. Similarly when a honeypot is probed, it responds with 'r' half of the time. This way the attacker can be deceived to attack a honeypot. When the attacker probes host 1, whatever response she gets, she attacks host 1 and host 2 with equal probability. This is safe for the attacker because she does not know which node is a honeypot. When the attacker probes host 2 and receives the response 'h', the attacker chooses not to attack, to avoid the risk of attacking the honeypot. However when she receives the response 'r', she attacks host 1 and host 2 with equal probability. This particular strategy is suitable for an attacker who wants to avoid the risk to a certain degree.

The general $HDG(n, k, \gamma)$ game can be formally defined as follows:

(i) $\{A, D\}$ is the set of players;

(ii) The set of terminal histories is denoted by set of vectors $\{(x_1, x_2, \ldots, x_m) | m = 2^{\gamma+1}$, where x_i is the action taken by the player at stage i of the game $\}$. The actions available to the players are as follows:

(a) $x_1 = \{(y_1 y_2 \ldots y_n) | k$ vector components are 1 and $n - k$ components are $0\}$. If $y_i = 1$ then D places a honeypot at position (address) i and if $y_i = 0$ then D places a regular host at position i. There are $q = \binom{n}{k}$ such combinations possible. We call this vector a *configuration* and denote configurations by $c_t | t = \{1, \ldots, q\}$

(b) $x_i = j, j \in \{1, \ldots, n\}$, are the actions available to A in intermediate stages (i is even, $i \neq m$) where A chooses to probe host j out of n hosts.

(c) $x_i = \{r, h\}$, are the actions available to D in different stages (i is odd, $i \neq 1$) (r denotes D's response to the probe that x_{i-1} is regular host, h denotes a honeypot (might be a deception move)).

(d) $x_m = j, j \in \{1, \ldots, n+1\}$ in which A chooses to attack host j ($j = n+1$ means no attack).

(iii) The function $P(l)$ such that $P(\emptyset) = D$, $P(l) = A$, if the last action in subhistory l was taken by D else $P(l) = D$. In other words, D and A play alternatively, starting with D and the last action is taken by A.

(iv) The information sets in the information partition of D consists of individual histories $\{\{(x_1, x_2)\}, \{(x_1, x_2, x_3, x_4)\}, \ldots, \{(x_1, x_2, \ldots, x_{m-2})\}\}$. The information partition of A has all histories with different values of x_1 but same values of $(x_2, \ldots, x_{m-1})$ grouped in the same information set. Formally, the information partition of A is $\{\{(c_t)\}$, $\{(c_t, 1, r)\}$, $\{(c_t, 1, h)\}$, $\{(c_t, 2, r)\}$, …, $(c_t, n, h, \ldots, h) | t = \{1, \ldots, q\}\}$

(v) The preferences over the terminal histories are calculated by summing the intermediate payoffs (given in Table 5.1) corresponding to the actions in the terminal history.

Since the information sets in which D moves contain individual histories, the defender observes all actions of the attacker. However each information set of the attacker contains q histories, each history corresponding to different configurations. For example, if $n = 3, k = 2, \gamma = 1$, there are 3 different configurations, $c_1 = 110, c_2 = 101$ and $c_3 = 011$. The information partition of the defender is as follows $\{$ $\{(\phi)\}$, $\{(110, 1)\}$, $\{(110, 2)\}$, $\{(110, 3)\}$, $\{(101, 1)\}$, $\{(101, 2)\}$, $\{(101, 3)\}$, $\{(011, 1)\}$, $\{(011, 2)\}$, $\{(011, 3)\}\}$

Each information set of the attacker will contain histories with different values of c_t but the same values for the other actions, *i.e.* $\{$ $\{(110),(101),(011)\}$, $\{(110, 1, r)$, $(101, 1, r)$, $(011, 1, r)\}$, $\{(110, 1, h)$, $(101, 1, h)$, $(011, 1, h)\}$, $\{(110, 2, r)$, $(101, 2, r)$, $(011, 2, r)\}$, $\{(110, 2, h)$, $(101, 2, h)$, $(011, 2, h)\}\}$. This means that at any point in time, the attacker cannot identify the placement of honeypots and regular hosts in the system.

This general game model allows us to analyze the interactions of the attacker and the defender. The different equilibrium solutions help the defender and the attacker choose the optimal strategies, based on their specific goals.

5.4 Summary and Future Work

In this chapter we proposed a game theoretic framework for modeling deception in honeynets. The framework is based on extensive form games of imperfect information. In an interesting scenario, attackers are trying to find and avoid honeypots whereas the system administrator is trying to hide them. We modeled this interaction using extensive form games. Since the action of the honeynet system is not observable by the attacker, we use imperfect information games, which is useful to model deception. We studied the equilibrium solutions of these games and showed how they are used to determine the strategies of the honeynet system. In the future we will enhance the model by considering beliefs of agents about other agents' actions and payoffs as well as use stochastic games to develop a more general and powerful model. We also plan to use techniques from machine learning to learn about the attackers strategies and make the defense very effective.

CHAPTER 6: CONCLUSION

In this dissertation, we presented our game theoretic approach to dealing with misbehavior in three specific areas of distributed systems namely, multicast cost sharing, auction-based scheduling, and information security.

In the area of multicast cost sharing, we developed a distributed implementation of Shapley value mechanism for sharing the cost of multicast transmissions. We showed that the proposed mechanism is a faithful implementation of the Shapley value mechanism. We also ran extensive experiments to find interesting results related to the marginal cost and the Shapley value mechanisms for both tamper-proof and autonomous node models. Our mechanism relies on cryptographic techniques and auditing to prevent nodes from deviating from the protocol. We plan to investigate alternative methods such as the use of incentives, partitioning the problem and using redundancy to achieve faithfulness.

For auction-based scheduling, we studied how agents with malicious intent participating in auction-based scheduling mechanisms negatively affect other participating agents. We developed an antisocial strategy that can be used by an agent to inflict losses on the other participating agents. We plan to consider the presence of more than one antisocial agent and study the effect of their strategies on the other agents' profits. Further work in this area would be to develop strategies which can prevent losses inflicted by the antisocial agents.

In the area of information security, we have developed a model to analyze the interactions between an attacker and a honeynet system. Our model allows the use of deception, which in our problem makes the honeynet system stronger by hiding the honeypots. We plan to consider more complex scenarios and study different solution concepts for the underlying game. We also plan to develop a general and powerful model using stochastic games to model interactions between players that can be used to derive effective deception strategies for the honeynet system.

REFERENCES

[1] Openssl. www.openssl.org.

[2] K. Aberer and M. Hauswirth. An overview of peer-to-peer information systems. In *Workshop on Distributed Data and Structures*, pages 171–188, 2002.

[3] E. Adar and B. A. Huberman. Free riding on gnutella. *First Monday*, 5(10), 2000.

[4] M. Adler and D. Rubenstein. Pricing multicasting in more practical network models. In *Proc. of the 13th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 981–990, 2002.

[5] A. Agah, S. K. Das, and K. Basu. A game theory based approach for security in wireless sensor networks. In *Proc. of the IEEE Performance, Computing and Communications Conference*, pages 259–263, 2004.

[6] A. Agah, S. K. Das, K. Basu, and M. Asadi. Intrusion detection in sensor networks: A non-cooperative game approach. In *Proc. of the 3rd IEEE International Symposium on Network Computing and Applications*, pages 343–346, 2004.

[7] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth. BAR fault tolerance for cooperative services. *ACM SIGOPS Operating Systems Review*, 39(5):45–58, 2005.

[8] K. C. Almeroth and M. H. Ammar. The use of multicast delivery to provide a scalable and interactive video-on-demand service. *IEEE Journal on Selected Areas in Communications*, 14(6):1110–1122, 1996.

[9] T. Alpcan and T. Basar. A game theoretic approach to decision and analysis in network intrusion detection. In *Proc. of the 42nd IEEE Conference on Decision and Control*, pages 2595–2600, Dec. 2003.

[10] N. Andelman, Y. Azar, and M. Sorani. Truthful approximation mechanisms for scheduling selfish related machines. *Theory of Computing Systems*, 40(4):423–436, 2007.

[11] R. Anderson. Why Information Security is Hard-An Economic Perspective. In *Proc. of the 17th Annual Computer Security Applications Conference*, 2001.

[12] M. Andrews, K. Kumaran, K. Ramanan, A. Stolyar, P. Whiting, and R. Vijayakumar. Providing quality of service over a shared wireless link. *IEEE Communications Magazine*, 39(2):150–154, 2001.

[13] A. Archer, J. Feigenbaum, A. Krishnamurthy, R. Sami, and S. Shenker. Approximation and Collusion in Multicast Cost Sharing. *Games and Economic Behavior*, 47(1):36–71, 2004.

[14] A. Archer and E. Tardos. Truthful mechanism for one-parameter agents. In *Proc. of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 482–491, October 2001.

[15] A. Archer and E. Tardos. Frugal path mechanisms. In *Proc. of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 991–999, January 2002.

[16] O. Arndt, B. Freisleben, T. Kielmann, and F. Thilo. A comparative study of online scheduling algorithms for networks of workstations. *Cluster Computing*, 3(2):95–112, 2000.

[17] R. Axelrod. *The Evolution of Cooperation.* Basic Books, New York, 1984.

[18] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proc. of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 205–217, August 2002.

[19] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Omni: An efficient overlay multicast infrastructure for real-time applications. *Computer Networks*, 50(6):826–841, 2006.

[20] J. C. Beck, A. J. Davenport, and M. S. Fox. Five pitfalls of empirical scheduling research. In *Principles and Practice of Constraint Programming*, volume 1330, pages 390–404. Lecture Notes in Computer Science, 1997.

[21] J. Bethencourt, J. Franklin, and M. Vernon. Mapping internet sensors with probe response attacks. In *Proc. of the USENIX Security Symposium*, pages 193–208, August 2005.

[22] A. Blanc, Y. Liu, and A. Vahdat. Designing incentives for peer-to-peer routing. In *Proc. of the IEEE Infocom Conference*, March 2005.

[23] F. Brandt and G. Weiß. Antisocial agents and Vickrey auctions. In *Proc. of the 8th Workshop on Agent Theories, Architectures and Languages*, pages 335–347, August 2001.

[24] C. Buragohain, D. Agrawal, and S. Suri. A game theoretic framework for incentives in p2p systems. In *Proc. of the 3rd International Conference on Peer-to-Peer Computing*, pages 48–56, 2003.

[25] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource allocation and scheduling in gridcomputing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1507–1542, 2002.

[26] T. E. Carroll and D. Grosu. Brief announcement: distributed algorithmic mechanism design for scheduling. In *Proc. of the 24th annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 128–128, 2005.

[27] T. E. Carroll and D. Grosu. Distributed algorithmic mechanism design for scheduling on unrelated machines. In *Proc. of the 8th International Symposium on Parallel Architectures, Algorithms, and Networks*, pages 194–199, 2005.

[28] T. E. Carroll and D. Grosu. A strategyproof mechanism for scheduling divisible loads in tree networks. In *Proc. of the 20th IEEE International Parallel and Distributed Processing Symposium*, April 2006.

[29] Y. Chawathe. Scattercast: an adaptable broadcast distribution framework. *Multimedia Systems*, 9(1):104–118, 2003.

[30] X. Chen and X. Deng. Settling the Complexity of Two-Player Nash Equilibrium. In *Proc. of the 47th Annual IEEE Symposium on Foundations of Computer Science*, page 272. IEEE Computer Society, 2006.

[31] B. Cheswick. An evening with berferd, in which a hacker is lured, endured, and studied. In *Proc. of the Winter Usenix Conference*, January 1992.

[32] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proc. of the ACM International Conference on Measurements and Modeling of Computer Systems*, pages 1–12, June 2000.

[33] E. Clarke. Multipart pricing of public goods. *Public Choice*, 8:17–33, 1971.

[34] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In H. Federrath, editor, *Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pages 46–66. Springer, 2000.

[35] F. Cohen. A note on the role of deception in information protection. *Computers and Security*, 17(6):483–506, 1998.

[36] F. Cohen. Leading attackers through attack graphs with deceptions. *Computers and Security*, 22(5):402–411, 2003.

[37] V. Conitzer and T. Sandholm. Complexity of Mechanism Design. *ArXiv Computer Science e-prints*, pages 5075–+, May 2002.

[38] V. Conitzer and T. Sandholm. New complexity results about Nash equilibria. *Games and Economic Behavior*, 63(2):621 – 641, 2008.

[39] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *Proc. of the second ACM International Conference on Multimedia*, pages 15–23. ACM Press, 1994.

[40] A. Das and D. Grosu. Combinatorial auction-based protocols for resource allocation in grids. In *Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium - Workshop 13*, 2005.

[41] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou. The complexity of computing a nash equilibrium. *Communications of the ACM*, 52(2):89–97, 2009.

[42] S. Deering and D. R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8(2):85–110, 1990.

[43] S. Deering, D. L. Estrin, D. Farinacci, V. Jacobson, L. Ching-Gung, and W. Liming. The PIM architecture for wide-area multicast routing. *IEEE/ACM Transactions on Networking*, 4(2):153–162, 1996.

[44] C. Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *Proc. of the 2nd ACM Conference on Electronic Commerce*, pages 150–157, 2000.

[45] C. Dellarocas. Mechanisms for coping with unfair ratings and discriminatory behavior in online reputation reporting systems. In *Proc. of the 21st International Conference on Information Systems*, pages 520–525, 2000.

[46] X. Deng and C. H. Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19(2):257–266, 1994.

[47] D. Dolev, O. Mokryn, and Y. Shavitt. On multicast trees: structure and size estimation. *IEEE/ACM Transactions on Networking*, 14(3):557–567, 2006.

[48] B. Faltings. A budget-balanced, incentive-compatible scheme for social choice. In *Agent-mediated E-commerce VI*, volume 3435. Springer Lecture Notes in Computer Science, 2005.

[49] J. Feigenbaum, A. Krishnamurthy, R. Sami, and S. Shenker. Approximation and collusion in multicast cost sharing. In *Proc. of the 4th ACM Conference on Electronic Commerce*, pages 280–280, June 2003.

[50] J. Feigenbaum, A. Krishnamurthy, R. Sami, and S. Shenker. Hardness results for multicast cost sharing. *Theoretical Computer Science*, 304(1-3):215–236, 2003.

[51] J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker. A bgp-based mechanism for lowest-cost routing. In *Proc. of the 21st ACM Symposium on Principles of Distributed Computing*, pages 173–182, July 2002.

[52] J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. In *Proc. of the 32nd Annual ACM Symposium on Theory of Computing*, pages 218–227, May 2000.

[53] J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63(1):21–41, August 2001.

[54] J. Feigenbaum and S. Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *Proc. of the 6th ACM Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 1–13, September 2002.

[55] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust incentive techniques for peer-to-peer networks. In *Proc. of the 5th ACM Conference on Electronic Commerce*, pages 102–111, 2004.

[56] M. Feldman, C. Papadimitriou, J. Chuang, and I. Stoica. Free-riding and whitewashing in peer-to-peer systems. In *Proc. of the ACM SIGCOMM workshop on Practice and Theory of Incentives in Networked Systems*, pages 228–236, 2004.

[57] N. Garg and D. Grosu. Deception in honeynets: A game-theoretic analysis. In *Proc. of the 8th IEEE SMC Information Assurance and Security Workshop*, pages 107–113, 2007.

[58] N. Garg and D. Grosu. Faithful distributed Shapley mechanisms for sharing the cost of multicast transmissions. In *Proc. of the 12th IEEE Symposium on Computers and Communications*, pages 741–747, July 2007.

[59] N. Garg and D. Grosu. Performance evaluation of multicast cost sharing mechanisms. In *Proc. of the 21st IEEE International Conference on Advanced Information Networking and Applications*, pages 901–908, May 2007.

[60] N. Garg and D. Grosu. A faithful distributed mechanism for sharing the cost of multicast transmissions. *IEEE Transactions on Parallel and Distributed Systems*, 20(8):1089–1101, Aug. 2009.

[61] N. Garg, D. Grosu, and V. Chaudhary. An antisocial strategy for scheduling mechanisms. In *Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium*, page 203a, April 2005.

[62] N. Garg, D. Grosu, and V. Chaudhary. Antisocial behavior of agents in scheduling mechanisms. *IEEE Transactions on Systems, Man and Cybernetics - Part A*, 37(6):946–954, Nov. 2007.

[63] A. V. Goldberg and J. D. Hartline. Collusion-resistant mechanisms for single-parameter agents. In *Proc. of the 16th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 620–629, 2005.

[64] P. Golle, K. Leyton-Brown, and I. Mironov. Incentives for sharing in peer-to-peer networks. In *Proc. of the ACM Conference on Electronic Commerce*, pages 264–267, 2001.

[65] S. Gorinsky, S. Jain, H. Vin, and Z. Yongguang. Design of multicast protocols robust against inflated subscription. *IEEE/ACM Transactions on Networking*, 14(2):249–262, 2006.

[66] S. Govindan and R. Wilson. A global newton method to compute Nash equilibria. *Journal of Economic Theory*, 110(1):65–86, 2003.

[67] R. L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, November 1966.

[68] J. Green, E. Kohlberg, and J. Laffont. Partial equilibrium approach to the free-rider problem. *Journal of Public Economics*, 6(4):375–394, 1976.

[69] D. Grosu and A. T. Chronopoulos. A truthful mechanism for fair load balancing in distributed systems. In *Proc. of the 2nd IEEE International Symposium on Network Computing*, pages 289–296, April 2003.

[70] D. Grosu and A. T. Chronopoulos. Algorithmic mechanism design for load balancing in distributed systems. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 34(1):77–84, February 2004.

[71] T. Groves. Incentive in teams. *Econometrica*, 41(4):617–631, 1973.

[72] J. Hershberger and S. Suri. Vickrey prices and shortest paths: What is an edge worth? In *Proc. of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 252–259, October 2001.

[73] S. Herzog, S. Shenker, and D. Estrin. Sharing the "cost" of multicast trees: an axiomatic analysis. *IEEE/ACM Transactions on Networking*, 5(6):847–860, 1997.

[74] Honeynet Project. *Know Your Enemy: Learning about Security Threats.* Pearson Education Inc., Boston, MA, second edition, 2004.

[75] C. Huang, J. Li, and K. W. Ross. Can internet video-on-demand be profitable? In *Proc. of the 26th IEEE International Conference on Computer Communications*, pages 2144–2152, 2007.

[76] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. O. Jr. Overcast: Reliable multicasting with an overlay network. In *Proc. of the 4th Symposium on Operating Systems Design and Implementation*, pages 197–212, October 2000.

[77] P. Jehiel, B. Moldovanu, and E. Stacchetti. How (not) to sell nuclear weapons. *American Economic Review*, 86(4):814–29, 1996.

[78] R. Johari. *Efficiency loss in market mechanisms for resource allocation.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, June 2004.

[79] R. Jurca and B. Faltings. Reputation-based pricing of p2p services. In *Proc. of the Wokshop on Economics of P2P Systems*, 2005.

[80] A. S. K. Ranganathan, M. Ripeanu and I. Foster. To share or not to share : An analysis of incentives to contribute in collaborative file-sharing environments. In *Workshop on Economics of Peer-to-Peer systems*, 2003.

[81] U. Kant and D. Grosu. Double auction protocols for resource allocation in grids. In *Proc. of the International Conference on Information Technology: Coding and Computing*, pages 366–371, 2005.

[82] R. Karp, E. Koutsoupias, C. H. Papadimitriou, and S. Shenker. Optimization problems in congestion control. In *Proc. of the 41st IEEE Symposium on Foundations of Computer Science*, pages 66–74, November 2000.

[83] M. Kearns and L. Ortiz. Algorithms for interdependent security games. *Advances in Neural Information Processing Systems*, 16, 2004.

[84] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos. Multicast routing for multimedia communication. *IEEE/ACM Transactions on Networking*, 1(3):286–292, 1993.

[85] G. Koren and D. Shasha. D^{over}: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM Journal on Computing*, 24(2):318–339, 1995.

[86] A. Kovacs. Fast monotone 3-approximation algorithm for scheduling related machines. In *13th Annual European Symposium on Algorithms*, volume 3669 of *Lecture Notes in Computer Science*, pages 616–627. Springer, 2005.

[87] C. Kreibich and J. Crowcroft. Honeycomb - creating intrusion detection sugnatures using honeypots. *ACM SIGCOMM Computer Communications Review*, 34(1):51–56, 2004.

[88] L. Lao, J. H. Cui, M. Gerla, and S. Chen. A scalable overlay multicast architecture for large-scale applications. *IEEE Transactions on Parallel and Distributed Systems*, 18(4):449–459, 2007.

[89] N. Laoutaris, G. Smaragdakis, K. Oikonomou, I. Stavrakakis, and A. Bestavros. Distributed placement of service facilities in large-scale networks. In *Proc. of the 26th IEEE International Conference on Computer Communications*, pages 2144–2152, 2007.

[90] C. E. Lemke and J. T. Howson, Jr. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, June 1964.

[91] V. O. K. Li and Z. Zaichen. Internet multicast routing and transport control protocols. *Proceedings of the IEEE*, 90(3):360–391, 2002.

[92] E. I. Lin, A. M. Eskicioglu, R. L. Lagendijk, and E. J. Delp. Advances in digital video content protection. *Proceedings of the IEEE*, 93(1):171–183, 2005.

[93] P. Liu, W. Zang, and M. Yu. Incentive-based modeling and inference of attacker intent, objectives, and strategies. *ACM Transactions on Information and System Security*, 8(1):78–118, 2005.

[94] K. Lye and M. Wing. Game strategies in network security. *International Journal of Information Security*, 4(1-2):71–86, 2005.

[95] J. MacKie-Mason and H. Varian. Pricing the internet. *Public Access to the Internet*, pages 269–314, 1995.

[96] A. Mas-Colell, M. D. Whinston, and J. R. Green. *Microeconomic Theory*. Oxford University Press, New York, 1995.

[97] R. McKelvey, A. McLennan, and T. Turocy. Gambit: Software tools for game theory, 2007.

[98] R. D. McKelvey, A. McLennan, and T. Turocy. *Gambit: Software Tools for Game Theory*. http://www.gambit-project.org/, Version 0.2007.01.30, 2007.

[99] J. Mitchell and V. Teague. Autonomous nodes and distributed mechanisms. In *Proc. of the Software Security - Theories and Systems. Mext-NSF-JSPS International Symposium*, pages 58–83, 2003.

[100] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *IEEE Security and Privacy*, 1(4):33–39, July 2003.

[101] D. Moore, C. Shannon, D. J. Brown, G. M. Voelker, and S. Savage. Inferring internet denial-of-service activity. *ACM Transactions on Computer Systems*, 24(2):115–139, 2006.

[102] D. Moore, C. Shannon, G. M. Voelker, and S. Savage. Network telescopes: Technical report. Technical Report CS2004-0795, UCSD, July 2004.

[103] H. Moulin and S. Shenker. Strategyproof sharing of submodular costs: budget balance versus efficiency. *Journal of Economic Theory*, 18(3):511–533, 2001.

[104] J. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36:48–49, 1950.

[105] J. F. Nash. Equilibrium point in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, Jan. 1950.

[106] J. F. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.

[107] N. Nisan, S. London, O. Regev, and N. Camiel. Globally distributed computation over Internet - ThePOPCORN project. In *Proc. of the 18th IEEE International Conference on Distributed Computing Systems*, pages 592–601, May 1998.

[108] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behaviour*, 35(1/2):166–196, April 2001.

[109] N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[110] E. Nudelman, J. Wortman, Y. Shoham, and K. Leyton-Brown. Run the gamut: A comprehensive approach to evaluating game-theoretic algorithms. In *Proc. of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 880–887, August 2004.

[111] M. Osborne. *An Introduction to Game Theory.* Oxford University Press, New York, NY, 2004.

[112] M. Osborne and A. Rubinstein. *A Course in Game Theory.* MIT Press, Cambridge, Mass., 1994.

[113] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of internet background radiation. In *Proc. of the Internet Measurement Conference*, pages 27–40, October 2004.

[114] C. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994.

[115] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. *ACM SIGCOMM Computer Communication Review*, 33(1):59–64, 2003.

[116] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, Upper Saddle River, NJ, 2002.

[117] R. Porter, E. Nudelman, and Y. Shoham. Simple search methods for finding a nash equilibrium. In *Proc. of the 19th National Conference on Artificial Intelligence*, pages 664–669, July 2004.

[118] N. Provos. A virtual honeypot framework. In *Proc. of the 13th USENIX Security Symposium*, pages 280–284, August 2004.

[119] S. Rafaeli and D. Hutchison. A survey of key management for secure group communication. *ACM Computing Surveys*, 35(3):309–329, 2003.

[120] S. Ramanathan. Multicast tree generation in networks with asymmetric links. *IEEE/ACM Transactions on Networking*, 4(4):558–568, 1996.

[121] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proc. of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1190–1199, June 2002.

[122] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Proc. of the 1st International Conference on Peer-to-Peer Computing*, pages 99–100. IEEE Computer Society, 2001.

[123] M. Rothkopf and R. Harstad. Two Models of Bid-Taker Cheating in Vickrey Auctions. *Journal of Business*, 68(2):257–67, 1995.

[124] M. Rothkopf, T. Teisberg, and E. Kahn. Why Are Vickrey Auctions Rare? *Journal of Political Economy*, 98(1):94–109, 1990.

[125] T. Roughgarden. *Selfish Routing and the Price of Anarchy.* The MIT Press, 2005.

[126] T. Roughgarden and E. Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2):236–259, 2002.

[127] N. C. Rowe. Measuring the effectiveness of honeypot counter-counterdeception. In *Proc. of the 39th Hawaii International Conference on System Sciences*, January 2006.

[128] N. C. Rowe, B. T. Duong, and E. J. Custy. Fake honeypots: A defensive tactic for cyberspace. In *Proc. of the IEEE Workshop on Information Assurance*, pages 223–230, June 2006.

[129] K. Sallhammar, B. Helvik, and S. Knapskog. On stochastic modeling for integrated security and dependability evaluation. *Journal of Networks*, 1(5):31, 2006.

[130] T. Sandholm. Limitations of the Vickrey auction in computational multiagent systems. In V. Lesser, editor, *Proc. of the First International Conference on Multi–Agent Systems.* MIT Press, 1995.

[131] C. Saraydar, N. Mandayam, and D. Goodman. Efficient power control via pricing in wireless data networks. *IEEE Transactions on Communications*, 50(2):291–303, 2002.

[132] J. Schummer. Manipulation through bribes. *Journal of Economic Theory*, 91(2):180–198, 2000.

[133] J. Sgall. On-line scheduling. In A. Fiat and G. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*, pages 196–231. Springer-Verlag, Berlin, 1998.

[134] S. Shakkottai and R. Srikant. Economics of network pricing with multiple isps. *IEEE/ACM Transactions on Networking*, 14(6):1233–1245, 2006.

[135] L. S. Shapley. A value for n-person games. *Contribution to the Theory of Games*, 2:31–40, 1953.

[136] J. Shneidman and D. C. Parkes. Rationality and self-interest in peer to peer networks. In *International Workshop on Peer-to-Peer Systems*, pages 139–148, 2003.

[137] J. Shneidman and D. C. Parkes. Specification faithfulness in networks with rational nodes. In *Proc. of the 23rd annual ACM Symposium on Principles of Distributed Computing*, pages 88–97, July 2004.

[138] J. Shneidman, D. C. Parkes, and L. Massoulie. Faithfulness in internet algorithms. In *Proc. of the ACM SIGCOMM Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems*, 2004.

[139] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press New York, NY, USA, 2008.

[140] S. Siachalou and L. Georgiadis. Algorithms for precomputing constrained widest paths and multicast trees. *IEEE/ACM Transactions on Networking*, 13(5):1174–1187, 2005.

[141] K. Sim and E. Wong. Toward market-driven agents for electronic auction. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 31(6):274–284, 2001.

[142] V. Srinivasan, P. Nuggehalli, C. Chiasserini, and R. Rao. Cooperation in wireless ad hoc networks. In *Proc. of the IEEE Conference on Computer Communications*, 2003.

[143] K. Sripanidkulchai, B. Maggs, and H. Zhang. An analysis of live streaming workloads on the internet. In *Proc. of the 4th ACM SIGCOMM Conference on Internet Measurement*, pages 41–54, October 2004.

[144] G. Tan and S. A. Jarvis. Improving the fault resilience of overlay multicast for media streaming. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):721–734, 2007.

[145] P. Van Mieghem, G. Hooghiemstra, and R. van der Hofstad. On the efficiency of multicast. *IEEE/ACM Transactions on Networking*, 9(6):719–732, 2001.

[146] H. Varian. System Reliability and Free Riding. *Economics of information security*, page 1, 2004.

[147] W. Vickrey. Counterspeculation, auctions, and competitivesealed tenders. *Journal of Finance*, 16(1):8–37, March 1961.

[148] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. In *Proc. of the 20th ACM Symposium on Operating Systems Principles*, pages 148–162, October 2005.

[149] W. E. Walsh, M. P. Wellman, P. R. Wurman, and J. K. MacKie-Mason. Some economics of market-based distributed scheduling. In *Proc. of the 18th IEEE International Conference on DistributedComputing Systems*, pages 612–621, May 1998.

[150] J. Widger and D. Grosu. Computing equilibria in bimatrix games by parallel support enumeration. In *Proc. of the 7th International Symposium on Parallel and Distributed Computing*, pages 250–256, 2008.

[151] J. Widger and D. Grosu. Computing equilibria in bimatrix games by parallel vertex enumeration. In *Proc. of International Conference on Parallel Processing*, pages 116–123, 2009.

[152] J. Widger and D. Grosu. Parallel computation of nash equilibria in n-player games. In *Proc. of the 12th IEEE International Conference on Computational Science and Engineering*, pages 209–215, 2009.

[153] R. Wolski, J. S. Plank, T. Bryan, and J. Brevik. G-commerce: market formulations controlling resource allocation on the computational grid. In *Proc. of the 15th IEEE International Parallel and Distributed Processing Symposium*, April 2001.

[154] L. K. Yan. Virtual honeynets revisited. In *Proc. of the IEEE Workshop on Information Assurance*, pages 232–239, June 2005.

[155] V. Yegneswaran, P. Barford, and D. Plonka. On the design and use of internet sinks for network abuse monitoring. In *Proc. of the Symposium on Recent Advances in Intrusion Detection*, pages 146–165, September 2004.

[156] B.-C. Yen and O. Wu. Internet scheduling environment with market-driven agents. *IEEE Transactions on Systems, Man and Cybernetics - Part A*, 34(2):281–289, March 2004.

[157] C. Yi, X. Yuan, and K. Nahrstedt. Optimal resource allocation in overlay multicast. *IEEE Transactions on Parallel and Distributed Systems*, 17(8):808–823, 2006.

[158] S. Yuen and B. Li. Strategyproof mechanisms towards evolutionary topology formation in autonomous networks. *ACM/Kluwer Mobile Networks and Applications (MONET),*

Special Issue on Non-Cooperative Wireless Networking and Computing, 10:961–970, 2005.

ABSTRACT

DEALING WITH MISBEHAVIOR IN DISTRIBUTED SYSTEMS: A GAME-THEORETIC APPROACH

by

NANDAN GARG

August 2010

Advisor: Dr. Daniel Grosu
Major: Computer Science
Degree: Doctor of Philosophy

Most distributed systems comprise autonomous entities interacting with each other to achieve their objectives. These entities behave selfishly when making decisions. This behavior may result in strategical manipulation of the protocols thus jeopardizing the system wide goals. Micro-economics and game theory provides suitable tools to model such interactions. We use game theory to model and study three specific problems in distributed systems. We study the problem of sharing the cost of multicast transmissions and develop mechanisms to prevent cheating in such settings. We study the problem of antisocial behavior in a scheduling mechanism based on the second price sealed bid auction. We also build models using extensive form games to analyze the interactions of the attackers and the defender in a security game involving honeypots.

Multicast cost sharing is an important problem and very few distributed strategyproof mechanisms exist to calculate the costs shares of the users. These mechanisms are susceptible to manipulation by rational nodes. We propose a faithful mechanism which uses digital signatures and auditing to catch and punish the cheating nodes. Such mechanism will incur some overhead. We experimentally analyze the overhead and other relevant economic properties of the proposed and existing mechanisms. We plan to perform more experiments to gain more insights about the scalability and the effect of other parameters on the effectiveness

of the mechanism. We plan to extend this research to develop new mechanisms which combine other tools like redundancy and incentives with cryptography to achieve faithfulness.

In a second price sealed bid auction, even though the bids are sealed, an agent can infer the private values of the winning bidders, if the auction is repeated for related items. We study this problem from the perspective of a scheduling mechanism and develop an antisocial strategy which can be used by an agent to inflict losses on the other agents. This study forms the basis of our plan to develop effective strategies to prevent agents from behaving in malicious/antisocial ways in scheduling and more general auction mechanisms.

In a security system attackers and defender(s) interact with each other. Examples of such systems are the honeynets which are used to map the activities of the attackers to gain valuable insight about their behavior. The attackers want to evade the honeypots while the defenders want them to attack the honeypots. These interesting interactions form the basis of our research where we develop a model used to analyze the interactions of an attacker and a honeynet system. Our goal is to enhance our preliminary model by considering more complex scenarios and develop tools for managing honeynets based on the strategies suggested by the game theoretic analysis. We also plan to develop a more general model of attacker-defender interaction by using stochastic games. This model can be easily used to develop deception strategies to make the honeynet systems more powerful.

AUTOBIOGRAPHICAL STATEMENT

Nandan Garg received his Bachelors of Computer Applications degree from Devi Ahilya University, Indore, India in year 2000 and Master of Computer Applications degree from Rajiv Gandhi Technical University, Bhopal, India in 2003. He received his Ph.D. degree in the Department of Computer Science, Wayne State University, Detroit, USA in 2010. His research interests include distributed systems, information security, networks and application of game theory to these domains (including incentive centered computing and mechanism design). He is a member of the IEEE since 2005. Nandan has published his research in top peer-reviewed journals like IEEE Transactions on Parallel and Distributed Systems and IEEE Transactions on Systems, Man, and Cybernetics - Part A. He has received several research and teaching awards such as the *IEEE Outstanding Student Paper Award* for his research paper at the IEEE 21st International Conference on Advanced Information Networking and Applications (AINA) in 2007, *The GM scholastic award* by General Motors Corporation in 2008 and *The Graduate Teaching Award* from the Department of Computer Science, Wayne State University in 2008.