# Distributed Algorithmic Mechanism Design

A Dissertation

Presented to the Faculty of the Graduate School

of

Yale University

in Candidacy for the Degree of

Doctor of Philosophy

By

Rahul Sami

Dissertation Director: Joan Feigenbaum

December 2003

## Abstract

Distributed Algorithmic Mechanism Design

Rahul Sami

2003

Distributed algorithmic mechanism design (DAMD) is an approach to designing distributed systems that takes into account both the distributed-computational environment and the incentives of autonomous agents. In this dissertation, we study two problems, multicast cost sharing and interdomain routing. We also touch upon several issues important to DAMD in general, including *approximation*, *compatibility with existing protocols*, and *hardness* that results from the *interplay* of incentives and distributed computation.

The *multicast cost-sharing* problem involves choosing a set of receivers for a multicast transmission and determining payments for them to offset the bandwidth costs of the multicast. We focus on cost-sharing mechanisms that are group-strategyproof and budget-balanced. We prove fundamental lower bounds on the network complexity of group-strategyproof mechanisms that are exactly or approximately budget-balanced. The Shapley-value mechanism (SH) is perhaps the most economically compelling mechanism in this class. We give a group-strategyproof mechanism that exhibits a tradeoff between the other properties of SH: It can be computed by an algorithm that is more communication-efficient than SH, but it might fail to achieve exact budget balance or exact minimum welfare loss (albeit by a bounded amount). We also show that no strategyproof mechanism for multicast cost sharing can be both approximately efficient and approximately budget-balanced.

*Interdomain routing* is the routing of traffic between Internet domains or *Autonomous Systems*, a task currently performed by the Border Gateway Protocol (BGP). We first show that there is a unique strategyproof mechanism for *lowest-cost routing.* Moreover, the prices required by this mechanism can be computed with a straightforward change to BGP that causes only modest increases in routing-table size and convergence time.

We also formulate the *policy routing* mechanism-design problem. We show that, with arbitrary route valuations, it is NP-hard to find a welfare-maximizing (or even approximately welfare-maximizing) set of routes. For an important class of restricted valuations, *next-hop preferences*, a welfare-maximizing set of routes can be computed with a strategyproof mechanism in polynomial time (in a centralized computational model). However, we show that this mechanism appears to be incompatible with BGP, and hence is hard to compute in the context of the current Internet.

## Acknowledgments

This dissertation would not have been possible without the help of many people. Above all, I would like to thank my advisor, Joan Feigenbaum. None of my dissertation research would have been possible without her perceptive suggestions, constant encouragement, and particularly, her boundless energy. From the time I started working with her, I learnt that Joan's concern for her students goes far beyond the dissertation; she has been a mentor to me in every imaginable way, from introducing me to other researchers to making me aware of all the situations that arise in academic life.

Scott Shenker has been like a second advisor to me, ready with encouraging words, brilliant ideas, and honest advice whenever I needed them. While visiting Scott at Berkeley for a month, I became convinced that I really wanted to switch to working in this area. Arvind Krishnamurthy has also been an invaluable help to me– I have had many enjoyable and productive discussions about research (and cricket) with him.

This dissertation contains joint work with many colleagues I have been fortunate to work with: Joan, Scott, Arvind, Tim Griffin, Christos Papadimitriou, and Vijay Ramachandran.

I have been influenced by many other professors during my stay at Yale. I am very grateful to René Peralta for the fascinating courses, crazy soccer games, and for a sense of humour that helped me keep things in perspective. Ravi Kannan kindly agreed to serve on my thesis committee. I learnt much from Bradley Kuszmaul and Dana Henry, my advisors during my earlier stint as a computer architecture student.

I would never have made it through graduate school without all my friends in the department. I am thankful to Gabriel Loh, my friend, colleague, and officemate for four years, for the many long hours we spent together brainstorming, hacking, and making frequent trips to *Koffee?*; Gauri Shah, for always offering to help, and for laughing with me through my final year; and also Karhan Akcoglu, Patrick Huggins, Vijay Ramachandran, and Vinod Viswanath, for making my stay in the department very pleasant.

Finally, I would like to thank my family and friends, without whose support none of this would have been possible, and Neha Menon, for keeping me smiling even through the most stressful periods of work.

# Contents

# List of Figures

# Chapter 1

# Introduction

With the advent of the Internet, we are seeing the design and deployment of large-scale distributed systems. The deployed systems include large-scale parallel computing projects such as SETI@home [ACK$^+$02] and factoring [LM90], Internet services such as file-sharing and caching, and systems to support the basic network functionality, such as routing and congestion control. Usually, the primary focus of research on these distributed systems is the development of good distributed algorithms, *i.e.*, algorithms with low computational complexity and communication requirements. This research often tacitly assumes that the different components of the system perform their tasks as specified by the system designer. Sometimes, particularly in the cryptography literature, some of the participants are explicitly modeled as *adversaries*, who can deviate arbitrarily from the specification in order to defeat the intentions of the system designer or the other participants; the focus is then on designing systems that function well in spite of these adversaries. Often, distributed-systems research is also concerned with issues of *fault tolerance*, *i.e.*, developing algorithms and protocols that are robust in the presence of failures.

Computational and communication complexity, security against adversaries, and fault tolerance are certainly important features of Internet-based computation. However, one aspect of distributed systems on the Internet that has not been addressed until recently is the involvement of *self-interested* parties. Most of these distributed programs run on computers at many different locations on the Internet, owned and operated by a wide assortment of entities ranging from individual users to governmental and trans-national organizations. Each of these parties has its own goals and objectives; furthermore, these parties often operate with partial or complete autonomy and can exercise this autonomy in order to achieve their own objectives. Rather than follow the "rules" of the system as laid down by the designer, they can *strategize* to manipulate the system to their advantage. There are many situations in which such strategic deviation cannot be prevented. For example, the party may be required to report information that is intrinsically private, such as its own preferences; the party's behavior may not be fully observable or verifiable by any other component of the system; there may be legal difficulties that preclude enforcing any contract. These strategizing parties are best modeled as the "selfish agents" studied in game theory. An agent need not be a single individual but can also be a group or organization; the precise character of an agent depends heavily on the particular context. For instance, in the multicast scenario described in Chapter 3, a single user is the most natural "agent," whereas, in the interdomain-routing scenario in Chapter 4, an entire corporation is more naturally treated as a single agent.

The selfish agents involved in the system cannot be relied on to blindly follow any prescribed algorithm – whenever possible, they might strategize for their own gain. However, it is frequently possible for the system designer to model the objectives of an individual agent reasonably well; for example, if the agent is a corporation, we may assume that it seeks to maximize its profit to create value for its shareholders. Then, the system designer can exploit the fact that a selfish agent is primarily interested in maximizing its own personal gain: Although the agent will not obediently follow any prescribed protocol, we can assume that it will respond to *incentives*. Thus, we need not design algorithms that achieve correct results in the face of adversarial behavior on the part of some agents, merely algorithms that work correctly in the presence of predictably selfish behavior by all agents.

This approach – structuring incentives so as to induce the desired behavior of selfish agents – lies at the heart of *mechanism design*, a large area of research in economics. Informally, a mechanism is a system with the following form: Each agent can select a strategy from an allowed range of strategies. The system then processes all the agents' selected strategies and outputs an *outcome* as well as monetary *payments* to (or receipts from) each agent. Our aim is for the outcome to achieve some desirable global goal (for example, a congestion-control mechanism may want to select flow rates to maximize overall throughput). The mechanism is called *incentive-compatible* with this goal if the monetary transfers (payments or receipts) are such that selfish behavior on the part of each agent will lead to this desirable outcome's being achieved.

Economic mechanism design focuses on issues of incentive and strategy and largely ignores computational considerations. The *algorithmic mechanism design* approach of Nisan and Ronen [NR01] combines the two considerations. For simplicity, Nisan and Ronen considered a centralized computational model; however, they suggested that a distributed computational model would be more appropriate for the Internet, where agents are scattered across a network, and communication may be expensive or slow.

Feigenbaum, Papadimitriou, and Shenker [FPS01] extended the Nisan-Ronen framework to include distributed computation, in order to encompass systems in which the agents are distributed across a network, and it is impractical to collect all the input data at a single location and compute the mechanism in a centralized fashion. They initiated the *distributed algorithmic mechanism design (DAMD)* approach to designing systems for the Internet, which combines the incentive-compatibility considerations of mechanism design with the distributed-computing objective of designing systems with modest computation and communication requirements. A more detailed overview of mechanism design, algorithmic mechanism design, and distributed algorithmic mechanism design can be found in Chapter 2.

In this dissertation, I attempt to further our understanding on distributed algorithmic mechanism design. The thesis of my research is:

> **The distributed-computing context can have a major impact on the feasibility of a mechanism.**

DAMD is a very general framework that can potentially be applied to a broad class of problems. However, the exact interplay of incentives and distributed-computation constraints that arises in a system depends heavily on the context. Thus, my dissertation research is focused on two specific problems: *multicast cost sharing* and *interdomain routing.*

The multicast cost-sharing problem is a mechanism-design problem that was first introduced by Feigenbaum *et al.* [FPS01] and has since been studied by many researchers, including Jain and Vazirani [JV01], Adler and Rubenstein [AR02], Fiat *et al.* [FGHK02], and Mitchell and Teague [MT02]. The problem is described in detail in Chapter 3; here, we only outline it in brief. The problem is as follows: There is some digital content (say, a movie) at one node of the network and a set of users who are potentially interested in receiving that content. The content can be delivered to any subset of users, using a multicast tree for efficiency. However, bandwidth is costly, and each link has a certain associated cost that must be met if the link is used. Each user is willing to pay up to a certain amount for the content, but the maximum amount she will pay (the 'utility' of the content to her) is private information known only to the user herself. A mechanism for this multicast cost-sharing problem takes the users' reported utilities as input and decides which users receive the content, as well as how much each receiver pays.

There are two features that make this a useful problem for exploring issues in DAMD: It involves dividing costs among many users of an Internet service, and the costs involved exhibit *economies of scale*, *i.e.*, the marginal cost of serving an additional user goes down as the user base grows. These features are common to many network applications, and we believe that the analytical ideas we use for the multicast cost-sharing problem can be applied to other problem domains as well.

In joint work with Joan Feigenbaum, Arvind Krishnamurthy, and Scott Shenker [FKSS03, AFK$^+$03], I studied the communication requirements of multicast cost-sharing mechanisms. We focused on mechanisms that are *budget-balanced*, *i.e.*, the total payment exactly equals the incurred cost, and *group-strategyproof*, *i.e.*, no group of users can collude to manipulate the mechanism to their advantage. (In addition, we assume that the mechanism satisfies certain other natural properties, which are detailed in Chapter 3.) Feigenbaum *et al.* [FPS01] studied one particularly attractive mechanism of this class, the *Shapley-value* mechanism (SH). They proved a lower bound on the communication required to compute this mechanism for a restricted class of algorithms called "linear distributed algorithms": Any such algorithm must use $\Omega(np)$ bits of communication in the worst case, where $n$ is the number of nodes in the multicast tree and $p$ is the number of users.

In this dissertation, I present the following contributions towards our understanding of the multicast cost-sharing mechanism design problem:

- Any distributed algorithm, deterministic or randomized, that computes a budget-balanced, group-strategyproof multicast cost-sharing mechanism must send $\Omega(p)$ bits over $\Omega(n)$ links in the worst case. This lower bound applies, in particular, to the SH mechanism.

- Any distributed algorithm, deterministic or randomized, that computes an approximately budget-balanced, group-strategyproof multicast cost-sharing mechanism must send $\Omega(\log p)$ bits over $\Omega(n)$ links in the worst case.

- There is no strategyproof multicast cost-sharing mechanism satisfying certain natural properties that is both approximately efficient and approximately budget-balanced.

- There is a group-strategyproof mechanism that exhibits a trade-off between the properties of SH: It can be computed by an algorithm that is more communication-efficient than the SH mechanism (exponentially more so in the worst case), but it might fail to achieve exact budget balance or match the overall welfare of the SH mechanism (albeit by a bounded amount).

Apart from their immediate relevance to multicast cost sharing, these results shed new light on the general issue of approximation in algorithmic mechanism design. It is known that approximating a mechanism's output can destroy its strategic properties (see, *e.g.*, Nisan and Ronen [NR00]). Our results show that, in some circumstances, it may be possible to preserve the strategic properties of a mechanism while trading off accuracy and communication complexity.

The multicast cost-sharing problem is centered on an application-level Internet service. In contrast, the interdomain-routing problem described in Chapter 4 attempts to capture the incentive issues involved in providing basic network functionality: The discovery and distribution of good Internet routes must be supported by many autonomous organizations with independent goals. This touches upon additional complications that do not arise in proposals for new user-level services such as multicast, *e.g.*, issues of *backward compatibility.* It is unlikely that a proposal for an interdomain-routing protocol that is radically different from, and perhaps incompatible with, the current *de facto* standard protocol (*BGP*, the Border Gateway Protocol), will be adopted in the foreseeable future. Instead, it is important that any new mechanism co-exist with BGP to facilitate a gradual phase-in. Thus, BGP is a vital part of the distributed-computation context within which any interdomain routing mechanism must operate.

In joint work with Joan Feigenbaum, Christos Papadimitriou, and Scott Shenker [FPSS02], I studied the problem of *lowest-cost* interdomain routing. Our model is as follows: The natural strategic agents in the interdomain-routing context are the domains or *Autonomous Systems (ASes)*. We assume a simple cost model that is similar to earlier models of lowest-cost routing in the algorithmic mechanism design literature [NR01, HS01], in which each AS incurs a privately known, per-packet cost for every packet it carries. The goal of the mechanism is to minimize the total cost of routing any given traffic, which requires us to find the lowest-cost route for every source-destination pair. We show:

- There is a unique strategyproof mechanism for lowest-cost routing that pays nothing to ASes that carry no transit traffic.
- There is a "BGP-based" algorithm for computing the prices required by this mechanism, *i.e.*, an algorithm that requires only modest changes to BGP. The algorithm has reasonable computational complexity: It requires a small constant-factor growth in the size of the BGP routing tables and a modest increase in BGP convergence time.

Our idea of including the current standard protocol in the computational model is in itself a novel addition to the mechanism-design approach. The existing protocol can steer the design towards mechanisms that are easier to adopt and also serve as a yardstick for "acceptable" complexity.

The lowest-cost routing model may be too simplistic – in practice, ASes have more complex costs and preferences, which they express through their *routing policies*. In joint work with Joan Feigenbaum, Tim Griffin, Vijay Ramachandran, and Scott Shenker, I studied extensions of the algorithmic mechanism design approach to more general policies than lowest-cost routing. We formulated a model in which sources of network traffic have a *valuation* for potential routes to a destination, and this value need not be based on per-packet transit costs. The goal of the mechanism is to maximize the sum of all the ASes' valuations. We prove the following results, indicating that policy-routing mechanisms may be difficult to implement in this "BGP-based" computational model:

- In the most general case (in which ASes have arbitrary valuation functions over all paths to a given destination), finding an optimal outcome is NP-hard; it is even NP-hard to find an outcome that is approximately optimal, up to any reasonable factor.

- An important restricted class of valuation functions is the one in which an AS's valuation of a route is only based on its next hop on that route. Valuations based on the next hop alone can capture preferences due to customer/peer/provider relationships that pairs of ASes may have, and so this is an interesting class of valuations. In this case, finding the optimal routes reduces to finding a maximum-weight directed spanning tree and can be computed in polynomial time. The payment computation can also be performed in polynomial time, and thus the mechanism appears to be feasible in a centralized setting.

- The communication required to dynamically recompute the payments when valuations change may be unacceptably high in the context of BGP: There is a family of networks with $n$ ASes, which are Internet-like in that they are sparse and have low diameter, for which any update in valuations must result in $\Omega(n)$ messages, over a contiguous open subset of valuations. With this generically high communication, routing could be done in a *link-state* fashion, in which all relevant information is broadcast to all agents. BGP uses the alternative *path-vector* approach in order to reduce communication requirements, but adding the payment computation for this mechanism could defeat this purpose.

In both the multicast cost-sharing and interdomain-routing problems, we find that some (but not all) mechanisms that appear to be feasible in a centralized computational model turn out to be impractical in the distributed-computing context. Taken together, the results on multicast cost sharing and interdomain routing support my thesis that the distributed-computation context must be taken into account when designing Internet mechanisms.

The case of budget-balanced, group-strategyproof multicast cost-sharing is particularly interesting: If either the incentive-compatibility requirement (group-strategyproofness) or the distributed-computation requirement is dropped, the problem becomes easy to solve; thus, the hardness arises from the interplay of incentives and distributed computation. We call such problems *canonically hard*; they may form part of a "complexity theory" of distributed algorithmic mechanism design. Further, by showing the existence of canonically hard problems, we have demonstrated that distributed computation and incentive compatibility are not orthogonal issues that can be tackled independently; an approach that combines both is essential.

The rest of this dissertation is structured as follows: In Chapter 2, I review some concepts from Mechanism Design and Algorithmic Mechanism Design. In Chapter 3, I describe the multicast cost-sharing problem. I review the earlier and related work on this problem and then present our new results. In Chapter 4, I focus on the interdomain-routing problem. I describe earlier work on formulating routing as a mechanism-design problem and then explain where our model differs from the earlier ones. I then outline the "BGP-based" computational model and present our results on lowest-cost routing and policy routing. Finally, in Chapter 5, I summarize and mention some interesting open problems for future work.

# Chapter 2

# Background: Distributed Algorithmic Mechanism Design

In this chapter, we review concepts and terminology from classical mechanism design, as well as algorithmic mechanism design and distributed algorithmic mechanism design. Our aim is to present formal definitions of the concepts we use, to make clear the assumptions implicit in our analysis, and to outline the relationship between our work and other research in this field. We are not attempting a comprehensive survey of the literature in this vast field; we refer the readers to the chapter on mechanism design theory in the book by Mas-Colell, Whinston, and Green [MWG95, Ch. 13], and to the survey article by Jackson [Jac01]. A recent survey of distributed algorithmic mechanism design can be found in [FS02].

## 2.1 Mechanism Design Framework

Figure 2.1 depicts a simple mechanism setting: There are $n$ agents; each agent $i$ has some private information, called her private *type* $t_i$. This type is drawn from a set $T$ of possible types; the set $T$ is known to all, but $t_i$ is known only to agent $i$. For example, the agents could be bidders at an auction; in this case, the private type of an agent is the amount she is willing to pay for the item being auctioned. We use $\mathbf{t}$ to denote the vector $(t_1, t_2, \ldots, t_n)$.

The function of a mechanism in this setting is to solve a decision or allocation problem that affects all the agents. There is a set $\mathcal{O}$ of possible decisions (or allocations), and the

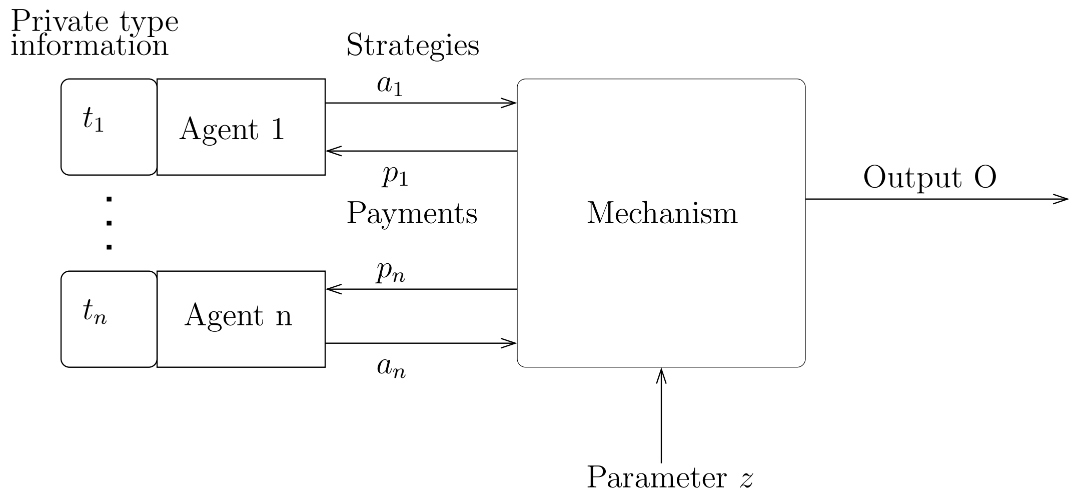


Figure 2.1: The general mechanism-design setting

mechanism must pick some decision $o \in \mathcal{O}$. For example, an auction decides which agent the item should belong to; in this case, the set of possible decisions is $\mathcal{O} = \{1, 2, ...n\}$. Typically, we want the decision to follow certain principles or exhibit some ethically appealing property; in the auction example, we might want the item to be allocated to the agent who values it the most. This is not trivial when the desired decision must depend on the private information of the agents (as in the auction case, where the value each agent places on the item is private). These constraints on the decision are expressed by requiring that the decision conform to a certain function of the agent types, the *social choice function*. We discuss social choice functions in more detail in Section 2.1.4.

In order to implement a social choice function, the mechanism must get some input from the agents. The design of the mechanism therefore includes a *message space* or *strategy space* $A$; each agent $i$ sends the mechanism some message $a_i \in A$. (The mechanism could invite each agent to declare her type, in which case we would have $A = T$, but the mechanism does not have to take this form.) The mechanism thus receives as input a vector of strategies $\mathbf{a} = (a_1, \ldots, a_n)$. It uses this input to compute a decision $o$, by following some output function $o = O(\mathbf{a})$.

The mechanism also decides on the money transfers that accompany this decision; this gives the mechanism designer the tools to incentivize the agents. Formally, the mechanism also computes payments $p_1, p_2, \ldots, p_n$. We use the convention that $p_i$ is the money paid by the mechanism *to* agent $i$; there is no restriction on the sign, however, and so a negative value can be used to indicate money received from agent $i$. Returning to our auction example, $p_i$ would typically be non-positive: The winner would have to pay the mechanism, but there is no money transfer with any other agent. We use $\mathbf{p}$ to denote the vector of payments $(p_1, p_2, \ldots, p_n)$. The mechanism computes the payment vector by following some payment function $\mathbf{p}(\mathbf{a}) = (p_1(\mathbf{a}), p_2(\mathbf{a}), \ldots, p_n(\mathbf{a}))$.[1]

The goal of the mechanism designer is to design the output and payment functions such that selfish behavior by the agents leads to a predictable strategy profile $\mathbf{a}(\mathbf{t})$ (which depends on the true type $\mathbf{t}$), and the output function for this predicted strategy exactly corresponds to the social choice function. We will develop the game-theoretic framework to describe "predictable strategies" in Section 2.1.1 and Section 2.1.3, and then mention some overall objectives for the mechanism in Section 2.1.4.

One final addition to this model is that we allow the mechanism design framework to include a parameter $z$ from a space $Z$. We do this because we are often interested in solving a *class* of related problems; for instance, in Chapter 3, we are interested in solving a class of cost-sharing problems, each corresponding to a different instance of multicast tree topology and cost. In a sense, this parameter represents the *public* information in the system; in the multicast problem, the public information consists of the tree topology and the link costs. This parameter $z$ can influence the desired social choice function, the mechanism output function, and the payments.

We are now ready to define a mechanism formally:

**Definition 2.1** *We are given a set of $n$ agents, type space $T$, decision space $\mathcal{O}$, and parameter space $Z$. A* **mechanism** *$\mathcal{M}$ is a tuple $(A, O, p_1, p_2, \ldots p_n)$, where $A$ is the strategy space,*
*$O : Z \times A^n \to \mathcal{O}$ is the output function, and $p_i : Z \times A^n \to \mathbb{R}$ is the payment function for agent $i$. A* **game form** *$\mathcal{M}_z$ is the mechanism $\mathcal{M}$ with the parameter fixed to $z \in Z$.*

The relevance of the type space $T$ will become apparent when we discuss the SCF that the mechanism is implementing.

[1]We use $\mathbf{p}$ and $p_i$ to denote the payment function as well as the actual payments computed. It will be clear from the context which of these we are referring to.

For conciseness, we will often let the strategy space $A$ be implicit and write the mechanism simply as a pair $\mathcal{M} = (O, \mathbf{p})$, where $\mathbf{p} = (p_1, p_2, \ldots, p_n)$.

We remark that this is not the most general model of a mechanism design problem. Many variations are possible: For example, we could have different type spaces and strategy spaces for different agents; there may also be probabilities assigned to the type profiles; or the mechanism itself may be randomized. Further, we have depicted the running of the mechanism as a one-round game (*i.e.*, a game in strategic form). It is also possible to consider *iterative* mechanisms, with many rounds of communication between the agents and the mechanism; this would result in an *extensive-form* game.

### 2.1.1 Quasilinear utility model

An agent $i$'s chosen strategy $a_i$ thus affects both the final decision and her payment $p_i$. What is the combined impact of the decision and the payment on the agent? The answer requires us to specify a *utility model.*

Throughout this dissertation, we use the *quasilinear* utility model. In this model, agent $i$'s overall *welfare*[2] $w_i$ is given by:

$$w_i = u_i(t_i, o) + p_i \ ,$$

where $u_i(t_i, o)$ is the *utility* she derives from the decision $o$ of the mechanism; this may depend on the agent's type. It is this welfare $w_i$ that agent $i$ seeks to maximize.

The quasilinear model is characterized by the fact that the welfare is *additively separable* into money and everything else and that it is *linear* in money. There are several consequences to this assumption. Among other things, it implies that agents are risk-neutral and that there is no "income effect," *i.e.*, the change in wealth of agent $i$ (due to the payment $p_i$) does not influence her utility $u_i$ for the consumed good or service. In general, neither of these assumptions is likely to be satisfied exactly; however, it is often a reasonable approximation for a particular mechanism, such as a movie multicast, in which the money transfers induced

[2]The terms *welfare* and *utility* are used interchangeably in the algorithmic mechanism design literature; what we call welfare is often referred to as utility. Throughout this dissertation, we stick to the convention that the utility is that portion of the welfare that is derived from the decision of the mechanism.

by the mechanism do not significantly alter the wealth of the participating agents. The quasilinear model has the advantage of analytical simplicity. For this reason, most of the algorithmic mechanism design literature to date assumes quasilinear utilities.

### 2.1.2 Induced Game

Suppose that we are given a mechanism $\mathcal{M}$ and a particular instance of the problem specified by parameter $z \in Z$. For every type profile $\mathbf{t}$, the game form $\mathcal{M}_z$ induces a strategic-form game among the agents as follows: Each agent $i$ chooses and plays some strategy $a_i$. The "payoff" for agent $i$ when the strategy profile is $\mathbf{a}$, is her resultant welfare:

$$w_i(\mathbf{a}) = u_i(t_i, O(z, \mathbf{a})) + p_i(z, \mathbf{a})$$

Note that agent $i$'s type $t_i$ is an implicit argument in her welfare $w_i(\mathbf{a})$.

### 2.1.3 Solution Concepts

We assume that all the agents are rational, selfish maximizers, *i.e.*, that they will only attempt to maximize their own welfare. Further, we assume that they know the exact form of the mechanism $\mathcal{M}$. Recall that mechanism designers want to exploit the predictably selfish behavior of the agents. This leads to the following question: Given a type profile $\mathbf{t}$, when can we predict the strategy profile $\mathbf{a}$ that results from selfish behavior by the agents?

Game theory provides us with many different ways to answer this question, depending on the assumptions we make about the agents' information about other agents' types and strategies, the agents' ability to coordinate strategies, *etc.* These assumptions are embodied in the *solution concept* used.

**Nash equilibrium** One solution concept that is very widely used is that of a *Nash equilibrium.* A strategy profile $\mathbf{a}$ is said to be a Nash equilibrium if, given the equilibrium strategy for all other agents, $a_i$ is an optimal strategy for agent $i$. Formally, we can express this as follows: We use the notation $\mathbf{a}^{-i}$ to denote the strategies of all agents except $i$, *i.e.*, $\mathbf{a}^{-i} = (a_1, \ldots, a_{i-1}, a_{i+1}, \ldots a_n)$.

**Definition 2.2** *A strategy profile* **a** *is a* **Nash equilibrium** *if*

$$\forall i \quad \forall a_i' \in A \quad w_i(\mathbf{a}^{-i}, a_i) \geq w_i(\mathbf{a}^{-i}, a_i')$$

A Nash equilibrium can be viewed as a "self-enforcing" contract: if agent $i$ is confident that all other agents are going to play their equilibrium strategies, then she cannot improve her welfare by playing any strategy other than $a_i$. Thus, if **a** is a Nash equilibrium strategy in the game induced by a mechanism, and all agents believe that every other agent is going to play according to this strategy, then selfish behavior will lead all agents to choose strategy profile **a**.

However, one major drawback of the Nash-equilibrium solution concept is that it requires agents to have complete knowledge (or belief) about every other agent's strategy. This problem arises because, in any game, there may be many different Nash equilibria. For the equilibrium strategy to be self-enforcing, all agents have to play according to the same one of these multiple equilibria. This requires some *coordination* among agents; in some domains, such coordination may be achieved (for instance, through repeated games and learning), but it appears to be too stringent a requirement for many Internet mechanisms.

Attempts to get around this problem of multiple equilibria have been made in the literature on *implementation theory*; Jackson [Jac01] surveys the literature in this field. There has also been a lot of research on *refinements* of Nash equilibria: If we make additional assumptions about which strategies are "reasonable" for an agent to play, most of the Nash equilibria may be eliminated, leading to a simpler coordination problem.

**Dominant strategy equilibrium** A stronger solution concept is that of *dominant strategies.*

**Definition 2.3** *A strategy profile* **a** *is said to be a* **dominant-strategy equilibrium** *if, in the game under consideration, agent i's optimal strategy is always to play $a_i$,* regardless of the strategy played by any other agent*:*

$$\forall i \quad \forall a_i' \in A \quad \forall \tilde{\mathbf{a}}^{-i} \in A^{n-1} \quad w_i(\tilde{\mathbf{a}}^{-i}, a_i) \geq w_i(\tilde{\mathbf{a}}^{-i}, a_i')$$

A dominant-strategy equilibrium is also a Nash equilibrium.

Thus, dominant strategies do not suffer from the coordination problem of Nash equilibria: An agent $i$ can choose her optimal strategy $a_i$ without worrying about other agents' types or strategies at all. This is particularly attractive in an Internet setting, where an agent may not even know which other agents are participating in the mechanism. Thus, if a mechanism always induces games with dominant strategy equilibria, an agent's dominant strategy only depends on her own private type $t_i$ (and perhaps the parameter $z$). In this case, there is a very strong reason to believe that selfish behavior will lead the agents to collectively play the dominant strategy profile $\mathbf{a}$.[3]

**Strategyproof Mechanisms** While dealing with the dominant strategy solution concept, we often focus our attention on *strategyproof mechanisms*. A strategyproof mechanism is one in which the strategy space $A$ is identical to the type space $T$, and, for every instance $z$ and every type profile $\mathbf{t}$, the induced game has a dominant-strategy equilibrium in which each agent $i$ plays the *truthful* strategy $a_i = t_i$. The *revelation principle* says that any mechanism with a dominant strategy solution can be transformed into a strategyproof mechanism. It is easy to see why this is true: We can wrap the mechanism with a procedure that asks for the type of each agent and computes their dominant strategies for them.

**Definition 2.4** *A mechanism is* **strategyproof** *if $A = T$ and, for any parameter $z$, the game form has the property that*

$$\begin{aligned} \forall z, \quad \forall \mathbf{t} \in T^n \quad [ \quad \forall i, \quad \forall a_i \in T, \quad \forall \mathbf{a}^{-i} \in T^{n-1}, \\ w_i(\mathbf{a}^{-i}, t_i) \quad = \quad u_i(t_i, O(z, \mathbf{a}^{-i}, t_i)) + p_i(z, \mathbf{a}^{-i}, t_i) \\ \geq w_i(\mathbf{a}^{-i}, a_i) \quad = \quad u_i(t_i, O(z, \mathbf{a}^{-i}, a_i)) + p_i(z, \mathbf{a}^{-i}, a_i) \quad ] \end{aligned}$$

Thus, a strategyproof mechanism induces each agent $i$ to reveal her private type information $t_i$ to the mechanism, in all circumstances. The mechanism will ideally use this

[3]It is possible for the induced game to have multiple dominant-strategy equilibria, but this is rarer than in the case of Nash equilibria. Further, we are often dealing with strategyproof mechanisms, in which case it is reasonable to expect an agent to play the simplest dominant strategy, which is to truthfully reveal her type.

information to output the desired social choice function $SCF(\mathbf{t})$.

**Group-strategyproof Mechanisms** Strategyproofness implies that, in all circumstances, no single agent can gain from lying about her type. However, it may still be possible for a *group* of agents to collude to improve all their welfares. This leads us to the definition of a stronger property: *group-strategyproofness.*

Let $S \subseteq [1, 2, \ldots, n]$ denote a subset of the agents, and let $\mathbf{t}^S = \{t_i | i \in S\}$. We say the group $S$ has a successful strategy if there is some strategy $\tilde{\mathbf{t}}^S$ that strictly increases at least one member of the group without reducing the welfare of any other member. For any strategy profile $\mathbf{a}$, we use the notation $\mathbf{a}^{-S}$ to denote the vector of strategies of all agents not in $S$: $\mathbf{a}^{-S} = \{a_j | j \notin S\}$.

**Definition 2.5** *A mechanism is called* **group-strategyproof** *if* $A = T$ *and no group can ever have a successful strategy, i.e.,*

$$\begin{aligned}
\forall z, \quad \forall \mathbf{t} \in T^n \quad \{ \quad \forall S, \quad \forall \tilde{\mathbf{t}}^S \in T^{|S|}, \quad \forall \mathbf{a}^{-S} \in T^{n-|S|}, \\
\big[\exists i \in S \quad s.t. \;\; w_i(\mathbf{a}^{-S}, \tilde{\mathbf{t}}^S) \;\; > \;\; w_i(\mathbf{a}^{-S}, \mathbf{t}^S) \\
\implies \exists j \in S \quad s.t. \quad w_j(\mathbf{a}^{-S}, \tilde{\mathbf{t}}^S) \;\; < w_j(\mathbf{a}^{-S}, \mathbf{t}^S)\big] \}
\end{aligned}$$

(Recall that $i$'s true type $t_i$ is an implicit argument of the function $w_i(\cdot)$.)

**Which is the "right" solution concept?** We have described some of the solution concepts used in the mechanism-design literature; there are many more solution concepts that we have not described. The question of which solution concept is most appropriate for Internet mechanisms has been debated by many researchers (see, *e.g.*, [NR01]), and was studied at length by Friedman and Shenker [FS97]. Strategyproofness seems to be a safe choice, provided collusion between agents is impractical, because the incentive compatibility of the mechanism then holds regardless of the extent of any agent's knowledge of other agents' preferences and strategies. Group-strategyproofness will give security against collusion as well.

However, there is a trade-off between the strength of the solution concept and the range of social choice functions that can be implemented. Classic impossibility results show that certain social choice functions cannot be implemented by strategyproof mechanisms [Arr63, Gib71, Sat75, GL79]. If the context of the particular mechanism justifies a weaker solution concept (such as Nash equilibrium), these SCFs may be implemented.

In this dissertation, we follow most of the algorithmic mechanism design literature in choosing strategyproofness and group-strategyproofness as our solution concepts.

### 2.1.4 Overall Goals

We now turn to defining overall goals for a mechanism. Recall that the mechanism selects both a decision $o \in \mathcal{O}$ and a vector of payments $\mathbf{p}$. However, the mechanism may not have complete flexibility to choose them independently; requirements such as budget balance (discussed below) may restrict the choice. In general, we assume that the mechanism must choose an *outcome* $(o, \mathbf{p}) \in \mathcal{F} \subseteq \mathcal{O} \times \mathbb{R}^n$, where $\mathcal{F}$ is the space of feasible outcomes. A mechanism is designed to implement a given social choice function, $\text{SCF} : T^n \to \mathcal{F}$, which maps a type profile $\mathbf{t}$ to a "desirable" outcome $(o, \mathbf{p})$. (More generally, we may have a social choice correspondence that maps each type vector to a set of acceptable outcomes.) Here, "desirability" is decided by the mechanism designer (or social planner), who must set objectives for his mechanism. There is a large branch of economics, called *social choice theory*, that deals with formalizing "desirable" goals for society.

How is the function SCF picked? Usually, it is built up from a combination of axioms, each representing a constraint on the mechanism, or an ethical principle that we would like to follow. For example, "If, at type profile $\mathbf{t}$, all agents like decision $o$ best, then, $\text{SCF}(\mathbf{t}) = o$." In this section, we describe two objectives, *efficiency* and *budget-balance*, which we target for the mechanisms in this thesis.

**Efficiency (Pareto-optimality)** The principle of Pareto-optimality says that, if all the participants of the game induced by a mechanism prefer outcome $(o', \mathbf{p}')$ to outcome $(o, \mathbf{p})$, then $(o, \mathbf{p})$ should not be the chosen outcome. In other words, if $(o, \mathbf{p})$ is the chosen decision, it should be "Pareto-optimal": There should be no decision $(o', \mathbf{p}') \in \mathcal{F}$ that all the participants prefer to $(o, \mathbf{p})$. Moulin [Mou91, pg. 14] calls the Pareto-optimality principle "the single most important concept in welfare economics." A Pareto-optimal outcome is also called an *efficient* outcome, and a mechanism is said to be *efficient* if it always gives a Pareto-optimal outcome.

When we are dealing with quasilinear utilities (defined in Section 2.1.1), this concept of efficiency takes on a convenient analytical form. The important feature of quasilinear utilities here is that all agents derive the same welfare from money; thus, *money transfers among agents are irrelevant to the efficiency* of the mechanism. Not all money need be transferred between agents; the mechanism may run a deficit or generate a surplus. This can be dealt with by assuming that there is a non-strategic party associated with the mechanism, such as society at large or the mechanism operator, that bears the deficit or surplus. In this case too, we can extend the quasilinear utilities to the party bearing the deficit or the surplus, and hence the Pareto-optimality of the mechanism is independent of the money transfers.[4] This leads us to define the *efficiency* or *overall welfare* $W(o)$ of decision $o$:

$$W(o) = \sum_{i=1}^{n} u_i(t_i, o)$$

With quasilinear utilities, an outcome $(o, p)$ is efficient if and only if $o$ is a decision in $\mathcal{O}$ that maximizes the overall welfare $W(o)$.[5] Maximizing $W(o)$, the sum of all agent utilities, is the classical *utilitarian* goal.

When the decisions may have different intrinsic costs to society, these have to be taken into account. For example, in the multicast cost-sharing problem described in Chapter 3, each decision $o$ corresponds to a different selected multicast tree, resulting in different costs $C(o)$.[6] In such situations, the efficiency or overall welfare (with costs) $W(o)$ is defined as:

$$W(o) = \sum_{i=1}^{n} u_i(t_i, o) \; - \; C(o)$$

A mechanism is then efficient if and only if it maximizes the overall welfare with costs.

[4]For this reason, games with quasilinear utility functions are also known as *transferable utility* games.

[5]Here, we assume that all balanced money transfers are feasible; this is true of all the mechanism design problems we consider.

[6]The cost may also depend on the problem instance, fixed by parameter $z$; for simplicity, we omit $z$ from the notation.

**Budget Balance** The other overall mechanism goal we consider in this dissertation is *budget balance*. As mentioned earlier, this can be viewed as a constraint on the mechanism's outcome: If there are no costs, then the payment vector $\mathbf{p}$ must satisfy $\sum_i p_i = 0$. If there is an intrinsic cost $C(o)$ associated with each decision $o$, then the outcome $(o, \mathbf{p})$ of the mechanism must satisfy

$$C(o) + \sum_{i=1}^{n} p_i = 0$$

In other words, the revenue generated from the agents exactly balances the cost the mechanism incurs.

It is also possible to consider a weaker budgetary constraint, in which the mechanism is allowed to run a surplus, but not a deficit. If the mechanism is a commercial entity, it is clear that this represents a real feasibility constraint: the mechanism can make a profit and survive, but it cannot make a loss.

However, there are situations in which a large surplus could also be detrimental to the company running the mechanism. In particular, a large surplus means that the agents are being charged more than what is required to cover the cost of the service the mechanism is providing, and so it is open to being undercut by competition. In such cases, we may want a mechanism that is as close to achieving exact budget balance as possible. Another situation in which exact budget balance may be required is when the mechanism is set up by the agents to cooperatively solve an allocation problem.

### 2.1.5 Vickrey-Clarke-Groves Mechanisms

One of the most striking results in mechanism design is a general technique for constructing strategyproof, efficient mechanisms in the quasilinear setting. This construction can be traced back to Vickrey's second-price auction [Vic61] and was further developed by Clarke [Cla71] and Groves [Gro73]. In the algorithmic mechanism design literature, these are called Vickrey-Clarke-Groves (VCG) mechanisms.

The VCG construction seems to give us all that we need– strategyproofness and optimal efficiency. However, one drawback of VCG mechanisms is that they are usually not budget-balanced; the construction relies on having the freedom to run a surplus or a deficit. This is one ramification of a result due to Green and Laffont [GL79], which shows that there is in general no strategyproof mechanism that is both budget-balanced and efficient. VCG mechanisms are also not group-strategyproof; typically, a colluding group of agents can easily improve all their individual welfares.

## 2.2 Algorithmic Mechanism Design

In Section 2.1, we described the economic aspect of mechanism design. We now turn to the computational aspect, which was first studied by Nisan and Ronen [NR01] in their seminal paper on *algorithmic mechanism design*.

Nisan and Ronen observed that, for a mechanism to be feasible in practice, the functions $O(\mathbf{a})$ and $\mathbf{p}(\mathbf{a})$ have to be tractable. Specifically, for a mechanism to be tractable, the output and payment functions must be polynomial-time computable; perhaps surprisingly, even this simple requirement ruled out many mechanisms that would appear ideal from an economic standpoint. This arises because, in many problem settings, finding an optimal (*e.g.*, efficient) decision is an NP-hard combinatorial optimization problem.

This naturally leads us to ask whether an approximation algorithm can be used to yield a mechanism that is approximately optimal. However, Nisan and Ronen [NR00] showed that this is not as straightforward as it may appear: Replacing an exact solution by an approximate solution may destroy the game-theoretic properties (*e.g.*, strategyproofness) of the mechanism. Thus, approximation in the context of mechanism design has to be done carefully; general approximation techniques have been developed for some classes of mechanism design problems [AT01].

The original paper of Nisan and Ronen [NR01] sparked a large body of research on algorithmic aspects of mechanism design. In particular, there is growing interest in incentive compatibility in both distributed and centralized computation in the theoretical computer science community (see, *e.g.,* [AT02, FPS01, FGHK02, HS01, NR00, RT02]) and in the "distributed agents" part of the AI community (see, *e.g.,* [MT99, Par99, PU00, San99, Wel93, WWWM01]). One problem that has been intensively studied is that of *combinatorial auctions*, which are auctions in which bidders may bid for subsets of a set of goods rather than for individual goods. The computational challenge here arises from the combinatorial explosion of the number of subsets that must be considered. Combinatorial auctions have an immediate application in the FCC Spectrum Auctions.

## 2.3 Distributed Algorithmic Mechanism Design

Nisan and Ronen's model of tractable computation is based on polynomial-time *centralized* computation. One of the main motivations for algorithmic mechanism design is the study of Internet mechanisms, and so they suggested that a distributed, Internet-like computational model might be more suitable in some cases.

In the Internet, the agents are often dispersed across the network; thus, the input and output of the mechanism must occur at dispersed locations. One way to compute the mechanism is to send all the input strategies to a single location, compute the output and payments in a centralized manner, and then send the required information back to the agent locations. However, this approach may require prohibitively high communication and it may cause congestion near the centralized server. This led Feigenbaum, Papadimitriou, and Shenker to consider distributed computational models in their paper on multicast cost-sharing mechanisms [FPS01]; this started the study of *distributed algorithmic mechanism design*. Feigenbaum *et al.* pointed out that for a mechanism to be feasible in an Internet setting, it must be computable by a distributed algorithm with low computational complexity *and* modest communication requirements. More specifically, the distributed algorithm should ideally have the following properties:

- The local computations require polynomial-time.
- Low communication complexity; the total number of messages sent is ideally $\mathbf{O}(s)$.
- Each message is reasonably small, *e.g.*, polylog$(s)$.
- No single link is congested, *i.e.*, the maximum number of messages on a link is $\mathbf{O}(1)$.

(Here, $s$ denotes the input size of a given instance of the problem.) They introduced the term *network complexity* to cover these four aspects of a distributed algorithm. A mechanism is said to have "good network complexity" if it satisfies all these properties.

We remark that the communication complexity of mechanisms is related to the concept of *message space dimensionality* that has been developed in the economics literature [MR74, Wal77]. The parallels between the two approaches are demonstrated by Nisan and Segal, in their analysis of combinatorial auctions [NS03]. However, the concepts are different in that the message-space dimensionality only deals with the informational requirements at equilibrium, whereas the communication complexity of computing the mechanism includes the information transfers required to reach equilibrium as well.

Finally, we note that our framework of distributed algorithmic mechanism design includes both distributed information (inputs and outputs) and distributed computation; there has also been work on studying the impact of distributed information alone on algorithmic mechanism design [MT99, NS03].

# Chapter 3

# Multicast Cost Sharing[†]

## 3.1 Introduction

In the standard *unicast* model of Internet transmission, each packet is sent to a single destination. Although unicast service has great utility and widespread applicability, it cannot efficiently transmit popular content, such as movies or concerts, to a large number of receivers; the source would have to transmit a separate copy of the content to each receiver independently. The *multicast* model of Internet transmission relieves this problem by setting up a shared delivery tree spanning all the receivers; packets sent down this tree are replicated at branch points so that no more than one copy of each packet traverses each link. Multicast thus greatly reduces the transmission costs involved in reaching large user populations.

The large-scale, high-bandwidth multicast transmissions required for movies and other potential sources of revenue are likely to incur substantial transmission costs. The costs when using the unicast transmission model are separable in that the total cost of the transmission is merely the sum of the costs of transmission to each receiver. Multicast's use of a shared delivery tree greatly reduces the overall transmission costs, but, because the total cost is now a submodular and nonlinear function of the set of receivers, it is not clear how to share the costs among the receivers. A recent series of papers has addressed the

[†]This chapter describes joint work with Joan Feigenbaum, Arvind Krishanmurthy, and Scott Shenker; parts of it were reported in [FKSS03, AFK$^+$03]. Aaron Archer provided valuable comments.

problem of cost sharing for Internet multicast transmissions. In the first paper on the topic, Herzog *et al.* [HSE97] considered axiomatic and implementation aspects of the problem. Subsequently, Moulin and Shenker [MS01] studied the problem from a purely economic point of view. Several more recent papers [FPS01, AR02, FGHK02, MT02] adopt the *distributed algorithmic mechanism design* approach, which augments a game-theoretic perspective with distributed computational concerns. In this chapter, we extend the results of [FPS01] by considering a more general computational model and approximate solutions. We also extend a classic impossibility [GL79] result by showing that no strategyproof mechanism can be both approximately efficient and approximately budget-balanced.

## 3.2 Multicast Cost Sharing Model

We use the multicast-transmission model of [FPS01]: There is a user population $P$ residing at a set of network nodes $N$, which are connected by bidirectional network links $L$. The multicast flow emanates from a source node $\alpha_s \in N$; given any set of receivers $R \subseteq P$, the transmission flows through a *multicast tree* $T(R) \subseteq L$ rooted at $\alpha_s$ and spans the nodes at which users in $R$ reside. It is assumed that there is a *universal* tree $T(P)$ and that, for each subset $R \subseteq P$, the multicast tree $T(R)$ is merely the minimal subtree of $T(P)$ required to reach the elements in $R$. This approach is consistent with the design philosophy embedded in essentially all multicast-routing proposals (see, *e.g.,* [BFC93, DEF$^+$96, HC93, PLB$^+$99]).

Each link $l \in L$ has an associated cost $c(l) \geq 0$ that is known by the nodes on each end, and each user $i$ assigns a utility value $u_i$ to receiving the transmission. Note that $u_i$ is known only to user $i$ *a priori.*

A *cost-sharing mechanism* determines which users receive the multicast transmission and how much each receiver is charged. Using the mechanism-design terminology introduced in Chapter 2, we can describe the mechanism-design problem: The users are the strategic agents. The private type information that user $i$ has is $u_i$, her utility for the transmission. We are interested in strategyproof mechanisms, and hence we can assume that the strategy space $A$ is the space of possible utility values, which is the set of all non-negative real numbers. The mechanism asks each user $i$ to report her utility value; user $i$ can strategize by reporting any value $\mu_i \geq 0$ in place of $u_i$. The decision space $\mathcal{O}$ is the set of all subsets of receivers. We use $\sigma_i$ to denote whether user $i$ receives the transmission: $\sigma_i = 1$ if the user receives the multicast transmission, and $\sigma_i = 0$ otherwise. The decision output by a mechanism is thus a vector $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_|P|)$.

This is a cost-sharing mechanism, and thus, it will receive payments from the users; thus, the payments $p_i$ as defined in Chapter 2 will be negative. We let $x_i = -p_i$ denote how much user $i$ is charged and $\sigma_i$ denote whether user $i$ receives the transmission; $\sigma_i = 1$ if the user receives the multicast transmission, and $\sigma_i = 0$ otherwise. We use $\mathbf{u}$ to denote the input vector $(u_1, u_2, \ldots, u_{|P|})$.

We want mechanisms to solve the cost-sharing problem for all possible multicast trees, rather than a simple problem. Therefore, the mechanism is parametrized by a parameter $z = (N, T(P), \{c_l\}, \alpha_s)$ that describes the non-private information in a particular problem instance: the set of nodes, the universal multicast tree, the link costs, and the source of the multicast. In order to keep the notation simple, we do not explicitly include this parameter in our formulae; when we are dealing with a single instance of the problem, it will be clear from the context.

The mechanism $M$ is then a pair of functions $M(\mathbf{u}) = (\sigma(\mathbf{u}), \mathbf{x}(\mathbf{u}))$. The practical feasibility of deploying the mechanism on the Internet depends on the network complexity of computing the functions $\mathbf{x}(\mathbf{u})$ and $\sigma(\mathbf{u})$. It is important to note that both the inputs and outputs of these functions are distributed throughout the network; that is, each user inputs his $u_i$ from his network location, and the outputs $x_i(\mathbf{u})$ and $\sigma_i(\mathbf{u})$ must be delivered to him at that location.

The *receiver set* for a given input vector is $R(\mathbf{u}) = \{i \mid \sigma_i = 1\}$. A user's individual *welfare* is given by $w_i = \sigma_i u_i - x_i$. The cost of the tree $T(R)$ reaching a set of receivers $R$ is $c(T(R))$, and the overall welfare, or *net worth*, is $NW(R) = u_R - c(T(R))$, where $u_R = \sum_{i \in R} u_i$ and $c(T(R)) = \sum_{l \in T(R)} c(l)$. The overall welfare measures the total benefit of providing the multicast transmission (the sum of the utilities minus the total cost).

Figure 1 depicts an instance of the multicast cost-sharing problem. There are five potential receivers, each located at a particular node of the multicast tree and each having a certain utility value for receiving the multicast transmission. For example, the notation $u_1 =$

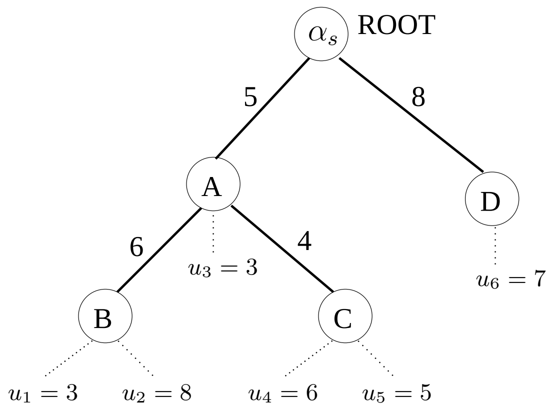


Figure 3.1: A multicast cost-sharing problem.

3 beside the leftmost node on the second level from the top means that potential receiver number 1 is located at this node and is willing to pay at most 3 to receive the transmission. The numerical values on the links represent the costs of sending the transmission over those links. The source of the transmission is the root node at the top level of the tree. If $R \subseteq \{1, \ldots, 6\}$ is the set of actual receivers, then the transmission will be sent only to the nodes of the tree at which members of $R$ are located. The total cost of this transmission will be the sum of the costs of the links in the smallest subtree that contains these nodes and the root. For example, if $R = \{2, 3, 4\}$, then the total cost of the transmission would be 15. The role of the cost-sharing mechanism is to determine, for each instance, what the receiver-set $R$ should be and how much each member of $R$ should be charged.

Our goal is to explore the relationship between incentives and computational complexity, but, before we do so, we first comment on several aspects of the model. The cost model we employ is a poor reflection of reality, in that transmission costs are not per-link; current network-pricing schemes typically only involve usage-based or flat-rate access fees, and the true underlying costs of network usage, though hard to determine, involve small incremental costs (*i.e.*, sending additional packets is essentially free) and large fixed costs (*i.e.*, installing a link is expensive). However, we are not aware of a well-validated alternative cost model, and the per-link cost structure is intuitively appealing, relatively tractable, and widely used (*e.g.*, in [AR02, FGHK02, FPS01, JV01, MT02]).

We assume that the total transmission costs are shared among the receivers. There are certainly cases in which the costs would more naturally be borne by the source (*e.g.*, broadcasting an infomercial) or the sharing of costs is not relevant (*e.g.*, a teleconference among participants from the same organization); in such cases, our model would not apply. However, we think that there will be many cases, particularly those involving the widespread dissemination of popular content, in which the costs would be borne by the receivers.

There are certainly cases, such as the high-bandwidth broadcast of a long-lived event such as a concert or movie, in which the bandwidth required by the transmission is much greater than that required by a centralized cost-sharing mechanism (*i.e.*, sending all the link costs and utility values to a central site at which the receiver set and cost shares could be computed). For these cases, our feasibility concerns would be moot. However, Internet protocols are designed to be general-purpose; what we address here is the design of a protocol that would share multicast costs for a wide variety of uses, not just long-lived and high-bandwidth events. Thus, the fact that there are scenarios (*e.g.*, the transmission of a shuttle mission, as explained below) in which our feasibility concerns are relevant is sufficient motivation; they need not be relevant in all scenarios.

In comparing the bandwidth required for transmission to the bandwidth required for the cost-sharing mechanism, one must consider several factors. First, and most obvious, is the transmission rate $b$ of the application. For large multicast groups, it will be quite likely that there will be at least one user connected to the Internet by a slow modem. Because the multicast rate must be chosen to accommodate the slowest user, one can't assume that $b$ will be large. Second, the bandwidth consumed on any particular link by centralized cost sharing mechanisms scales linearly with the number of users $n = |P|$, but the multicast's usage of the link is independent of the number of users. Third, one must consider the time increment $\Delta$ over which the cost accounting is done. For some events, such as a movie, it would be appropriate to calculate the cost shares once (at the beginning of the transmission) and not allow users to join after the transmission has started. For other events, such as the transmission of a shuttle mission, users would come and go during the course of the transmission. To share costs accurately in such cases, the time increment $\Delta$ must be fairly short. The accounting bandwidth on a single link scales roughly as $n$, which must be compared to the bandwidth $\Delta b$ used over a single accounting interval. Although small multicast groups with large $\Delta$ and $b$ could easily use a centralized mechanism, large multicast groups with small $\Delta$ and $b$ could not.

We have assumed that budget-balanced cost sharing, where the sum of the charges exactly covers the total incurred cost, is the goal of the charging mechanism. If the charging mechanism were being designed by a monopoly network operator, then one might expect the goal to be maximizing revenue. There have been some recent investigations of revenue-maximizing charging schemes for multicast (see, *e.g.*, [FGHK02]), but here we assume, as in [HSE97, MS01, FPS01, AR02], that the charging mechanism is decided by society at large (*e.g.*, through standards bodies) or through competition. Competing network providers could not charge more than their real costs (or otherwise their prices would be undercut) nor less than their real costs (or else they would lose money), and so budget balance is a reasonable goal in such a case. For some applications, such as big-budget movies, the bandwidth costs will be insignificant compared to the cost of the content, and then different charging schemes will be needed, but for low-budget or free content (*e.g.*, teleconferences) budget-balanced cost-sharing is appropriate.

Lastly, in our model it is the *users* who are selfish. The routers (represented by tree nodes), links, and other network-infrastructure components are obedient. Thus, the cost-sharing algorithm does not know the individual utilities $u_i$, and so users could lie about them, but once they report them to the network infrastructure (*e.g.*, by sending them to the nearest router or accounting node), the algorithms for computing $\mathbf{x}(\mathbf{u})$ and $\sigma(\mathbf{u})$ can be reliably executed by the network. Ours is the simplest possible strategic model for the distributed algorithmic mechanism-design problem of multicast cost sharing, but, even in this simplest case, determining the inherent network complexity of the problem is non-trivial. Alternative strategic models (*e.g.*, ones in which the routers are selfish, and their strategic goals may be aligned or at odds with those of their resident users) may also present interesting distributed algorithmic mechanism-design challenges. Preliminary work along these lines is reported in [MT02].

## 3.3 Notation and Terminology

In order to state our results more precisely, we need additional notation and terminology.

In general, we only consider mechanisms that satisfy four natural requirements[1]:

**No Positive Transfers (NPT)**: $x_i(\mathbf{u}) \geq 0$; in other words, the mechanism cannot *pay* receivers to receive the transmission.

**Voluntary Participation (VP)**: $w_i(\mathbf{u}) \geq 0$; this implies that users are not charged if they do not receive the transmission and that users who do receive the transmission are not charged more than their reported utilities.

**Consumer Sovereignty (CS)**: For any given problem instance $z$ there exists some $\mu_{cs}$ such that $\sigma_i(\mathbf{u}) = 1$ if $u_i \geq \mu_{cs}$; this condition ensures that the network cannot exclude any agent who is willing to pay a sufficiently large amount, regardless of other agents' utilities.

**Symmetry[2] (SYM)**: If $i$ and $j$ are at the same node or are at different nodes separated by a zero-cost path, and $u_i = u_j$, then $\sigma_i = \sigma_j$ and $x_i = x_j$.

We are concerned with mechanism that are *strategyproof*, or *group-strategyproof* (GSP); these properties were defined in Chapter 2.

## 3.4 Overall Objectives

In addition to these basic requirements, there are certain other desirable properties that one could expect a cost-sharing mechanism to possess. A cost-sharing mechanism is said to be *efficient* if it maximizes the overall welfare, and it is said to be *budget-balanced* if the revenue raised from the receivers covers the cost of the transmission exactly. It is a classical result in game theory [GL79] that a strategyproof cost-sharing mechanism that satisfies NPT, VP, and CS cannot be both budget-balanced and efficient.

Moulin and Shenker [MS01] have shown that there is only one strategyproof, efficient mechanism, called *marginal cost* (MC), defined in Section 3.9 below, that satisfies NPT, VP, and CS. They have also shown that, while there are many GSP, budget-balanced

[1]The one exception is Section 3.9, in which we do not assume SYM; that section contains an impossibility result, and so not making this assumption only makes the result stronger.

[2]This straightforward definition is less restrictive than the one given by Moulin and Shenker [MS01]. The SH, JV, and EG mechanisms that we use as examples satisfy the more stringent definition of symmetry in [MS01] as well.

mechanisms that satisfy NPT, VP, and CS, the most natural one to consider is the *Shapley value* (SH), defined in Section 3.7 below, because it minimizes the worst-case efficiency loss.

Both MC and SH also satisfy the SYM property. The *egalitarian* (EG) mechanism of Dutta and Ray [DR89] is another well studied GSP, budget-balanced mechanism that satisfies the four basic requirements. Jain and Vazirani [JV01] present a novel family of GSP, approximately budget-balanced mechanisms[3] that satisfy NPT, VP, and CS. Each mechanism in the family is defined by its underlying cost-sharing function, and the resulting mechanism satisfies the SYM property whenever the underlying function satisfies it. We use the notation JV to refer to the members of the Jain-Vazirani family that satisfy SYM.

It is noted in [FPS01] that, for multicast cost sharing, both MC and SH are polynomial-time computable by centralized algorithms. Furthermore, there is a distributed algorithm given in [FPS01] that computes MC using only two short messages per link and two simple calculations per node. By contrast, [FPS01] notes that the obvious algorithm that computes SH requires $\Omega(|P| \cdot |N|)$ messages in the worst case and shows that, for a restricted class of algorithms (called "linear distributed algorithms"), there is an infinite set of instances with $|P| = O(|N|)$ that require $\Omega(|N|^2)$ messages. Jain and Vazirani [JV01] give centralized, polynomial-time algorithms to compute the approximately budget-balanced mechanisms in the class JV.

## 3.5 Our Results

In this chapter, we show that:

- Any distributed algorithm, deterministic or randomized, that computes a budget-balanced, GSP multicast cost-sharing mechanism must send $\Omega(|P|)$ bits over linearly many links in the worst case. This lower bound applies, in particular, to the SH and EG mechanisms.

- Any distributed algorithm, deterministic or randomized, that computes an approximately budget-balanced, GSP multicast cost-sharing mechanism must send $\Omega(\log(|P|))$

[3]The mechanisms in [JV01] actually satisfy a more stringent definition of approximate budget balance than we use; thus, our network-complexity lower bounds apply to them *a fortiori*.

bits over linearly many links in the worst case. This lower bound applies, in particular, to the SH, EG, and JV mechanisms.

(In both these results, the "worst case" is worst with respect to all possible network topologies, link costs, and user utilities.)

In order to prove the first of these lower bounds (*i.e.*, the one for exact budget balance), we first prove a lower bound that holds for all mechanisms that correspond to *strictly cross-monotonic* cost-sharing functions. Cross-monotonicity, a technical property defined precisely in Section 3.6, means roughly that the cost share attributed to any particular receiver cannot increase as the receiver set grows; the SH and EG cost-sharing functions for a broad class of multicast trees are examples of strictly cross-monotonic functions but not the only examples. Our lower bound on the network complexity of strictly cross-monotonic mechanisms may be applicable to problems other than multicast.

It is well known that there is no strategyproof mechanism that is both (exactly) efficient and budget-balanced on all problem instances [GL79]. This in itself does not rule out the existence of a strategyproof mechanism that is approximately efficient and approximately budget-balanced. However, we prove that this is also impossible:

- There is no strategyproof multicast cost-sharing mechanism satisfying NPT, VP, and CS that is both approximately efficient and approximately budget-balanced.

Finally, we attack the question of finding an approximation to the SH mechanism:

- We present a group-strategyproof mechanism that exhibits a trade-off between the properties of SH: It can be computed by an algorithm that is more communication-efficient than the natural SH algorithm (exponentially more so in the worst case), but it might fail to achieve exact budget balance or exact minimum welfare loss (albeit by a bounded amount).

## 3.6 Submodular Cost-sharing Problems

We briefly digress from the multicast problem to study a more general class of cost-sharing problems. Consider the general situation in which we want a mechanism to allow the users to share the cost of a common service. We restrict our attention to the case of binary preferences: User $i$ is either "included," by which he attains utility $u_i$, or he is "excluded" from the service, giving him 0 utility. A mechanism can use the utility vector $\mathbf{u}$ as input to compute a set $R(\mathbf{u})$ of users who receive the service and a payment vector $\mathbf{x}(\mathbf{u})$. Further, suppose that the cost of serving a set $S \subseteq P$ of the users is given by $C(S)$.

**Definition 3.1** *A cost function $\mathcal{C}$ is called* **submodular** *if, for all $S, T \subseteq P$, it satisfies*

$$C(S \cup T) + C(S \cap T) \leq C(S) + C(T).$$

Submodularity is often used to model economies of scale, in which the marginal costs decrease as the serviced set grows. One example of a submodular cost function is the multicast presented in Section 3.1, where the cost of delivering a multicast to a set $R$ of receivers is the sum of the link costs in the smallest subtree of the universal tree that includes all locations of users in $R$.

Moulin [Mou99] has shown that any mechanism for submodular cost sharing that satisfies budget-balance, GSP, VP, and NPT must belong to the class of *cross-monotonic cost-sharing mechanisms*. A mechanism in this class is completely characterized by its set of cost-sharing functions $g = \{g_i : 2^P \rightarrow \mathbb{R}_{\geq 0}\}$. Here $g_i(S)$ is the cost that $g$ attributes to user $i$ if the receiver set is $S$. For brevity, we will refer to $g = \{g_i\}$ as a "cost-sharing function," rather than a set of cost-sharing functions.

**Definition 3.2** *We say that $g$ is* **cross-monotonic** *if, $\forall i \in S, \forall T \subseteq P$, $g_i(S \cup T) \leq g_i(S)$. In addition, we require that $g_i(S) \geq 0$ and, $\forall j \notin S$, $g_j(S) = 0$.*

Then, the corresponding cross-monotonic mechanism $M_g = (\sigma(\mathbf{u}), \mathbf{x}(\mathbf{u}))$ is defined as follows: The receiver set $R(\mathbf{u})$ is the unique largest set $S$ for which $g_i(S) \leq u_i$, for all $i$. This is well defined, because, if sets $S$ and $T$ each satisfy this property, then cross-monotonicity implies that $S \cup T$ satisfies it. The cost shares are then set at $x_i(\mathbf{u}) = g_i(R(\mathbf{u}))$.

There is a natural iterative algorithm to compute a cross-monotonic cost-sharing mechanism [MS01, FPS01]: Start by assuming the receiver set $R^0 = P$, and compute the resulting cost shares $x_i^0 = g_i(R^0)$. Then drop out any user $j$ such that $u_j < x_j^0$; call the set of remaining users $R^1$. The cost shares of other users may have increased, and so we need to compute the new cost shares $x_i^1 = g_i(R^1)$ and iterate. This process ultimately converges, terminating with the receiver set $R(\mathbf{u})$ and the final cost shares $x_i(\mathbf{u})$.

Jain and Vazirani [JV01] give a geometric characterization of the space of cross-monotonic mechanisms for a given submodular cost-sharing problem. A mechanism $M_g$ can be represented as a point in $\mathbb{R}^{n2^{n-1}}$ as follows: For each $S \subseteq P$ and each $i \in S$, we have a coordinate variable $\mathbf{x}(S, i)$. Then, we represent $M_g$ by the point $x_g$ defined by $x_g(S, i) = g_i(S)$. Given any submodular cost function $\mathcal{C} = \{C(S)\}$, the cross-monotonic mechanisms $M_g$ for sharing the cost function $\mathcal{C}$ satisfy these conditions:

$$\text{budget balance:} \quad \sum_{i \in S} g_i(S) = C(S) \quad \forall S \subseteq P \tag{3.1}$$

$$\text{cross-monotonicity:} \quad g_i(S) \leq g_i(S \cup j) \quad \forall S \subset P \;\; \forall i \in S \;\; \forall j \notin S \tag{3.2}$$

Moreover, any point $x_g$ for which conditions (3.1) and (3.2) hold corresponds to a cross-monotonic mechanism for $\mathcal{C}$. Thus, the set of points corresponding to feasible cross-monotonic mechanisms for $\mathcal{C}$ forms a polytope $\mathcal{P}(\mathcal{C})$, defined by the linear equalities and inequalities in conditions (3.1) and (3.2) and the implicit constraints $g_i(S) \geq 0$.

## 3.7 Lower bound for exact submodular cost sharing

In this section, we prove a basic communication-complexity lower bound that applies to the distributed computation of many submodular cost-sharing mechanisms. We first prove this lower bound for all mechanisms that satisfy "strict cross-monotonicity" as well as the four basic properties discussed in Section 3.1. We then show that, whenever the underlying cost function is strictly subadditive, the resulting Shapley-value mechanism is strictly cross-monotonic and hence has poor network complexity. Finally, we discuss the special case of multicast cost sharing and describe very general conditions under which the multicast cost will be strictly subadditive. In particular, we present an infinite family of instances that have strictly subadditive costs and show that, on these instances, any cost-sharing mechanism that satisfies the four basic requirements is equivalent to SH and must have poor network complexity.

In section Section 3.6, we noted that any mechanism for a submodular cost-sharing problem that is group-strategyproof and satisfies the properties NPT, VP, and CS must be a cross-monotonic cost-sharing mechanism. Now, we consider a subclass of the cross-monotonic mechanisms:

**Definition 3.3** *A cross-monotonic cost-sharing function $g = \{g_i : 2^P \to \mathbb{R}_{\geq 0}\}$ is called* **strictly cross-monotonic** *if, for all $S \subset P, i \in S$, and $j \notin S$, $g_i(S \cup \{j\}) < g_i(S)$. The corresponding mechanism $M_g$ is called a strictly cross-monotonic mechanism.*

We now prove a lower bound on the communication complexity of strictly cross-monotonic cost-sharing mechanisms. Our proof is a reduction from the *set disjointness* problem: Consider a network consisting of two nodes $A$ and $B$, separated by a link $l$ (see Figure 3.2). Node $A$ has a set $S_1 \subseteq \{1, 2, \ldots, r\}$, node $B$ has another set $S_2 \subseteq \{1, 2, \ldots, r\}$, and one must determine whether the sets $S_1$ and $S_2$ are disjoint. It is known that any deterministic or randomized algorithm to solve this problem must send $\Omega(r)$ bits between $A$ and $B$. (Proofs of this and other basic results in communication complexity can be found in [KN97].)

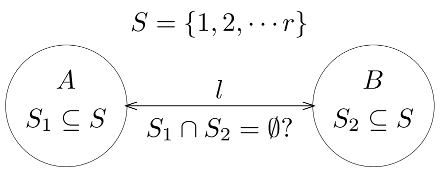


Figure 3.2: The set disjointness problem

**Theorem 3.1** *Suppose $M_g$ is a strictly cross-monotonic mechanism corresponding to a cost-sharing function $g$ and satisfying VP, CS, and NPT. Further, suppose that the mechanism must be computed in a network in which a link (or set of links) $l$ is a cut and there are $\Omega(|P|)$ users on each side of $l$. Then, any deterministic or randomized algorithm to compute $M_g$ must send $\Omega(|P|)$ bits across $l$ in the worst case.*

**Proof:** For simplicity, assume that the network consists of two nodes $A$ and $B$ connected by one link $l$ and that there are $r = |P|/2$ users at each of the two nodes. (For the more general case, we consider equal-sized subsets of the users at $A$ and $B$, and the remainder of the proof is identical.) Arbitrarily order the users at each node. We can now call the users $a_1, a_2, \ldots, a_r$ and $b_1, b_2, \ldots, b_r$. Because the mechanism $M_g$ is strictly cross-monotonic, we can find a real value $d > 0$ such that, for all $S \subset P, i \in S, j \notin S$,

$$g_i(S \cup \{j\}) < g_i(S) - d.$$

For each user $i \in P$, we will define two possible utility values $t_i^L$ and $t_i^H$ as follows:

- First, the values for $a_1$ and $b_1$ are

$$t_{a_1}^H = g_{a_1}(\{a_1, b_1\}), \qquad t_{a_1}^L = t_{a_1}^H - d$$

$$t_{b_1}^H = g_{b_1}(\{a_1, b_1\}), \qquad t_{b_1}^L = t_{b_1}^H - d$$

- Similarly, the values for $a_k$ and $b_k$ are

$$t_{a_k}^H = g_{a_k}(\{a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_k\}), \qquad t_{a_k}^L = t_{a_k}^H - d$$

$$t_{b_k}^H = g_{b_k}(\{a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_k\}), \qquad t_{b_k}^L = t_{b_k}^H - d$$

Now, we show how to reduce from the set disjointness problem to the mechanism $M_g$. Node $A$ gets a subset $S_1 \subseteq \{1, \ldots, r\}$ and constructs a utility vector $\mathbf{u}$ for the users at $A$, defined by

$$\forall i \in S_1 \;\; u_{a_i} = t_{a_i}^H$$

$$\forall i \notin S_1 \;\; u_{a_i} = t_{a_i}^L$$

Similarly, node $B$ is given set $S_2$ and constructs a utility vector $\mathbf{v}$ for the users at $B$, defined by

$$\forall i \in S_2 \;\; v_{b_i} = t_{b_i}^H$$

$$\forall i \notin S_2 \;\; v_{b_i} = t_{b_i}^L$$

They now run mechanism $M_g$ on input $(\mathbf{u}, \mathbf{v})$ and check whether the receiver set $R_g(\mathbf{u}, \mathbf{v})$

is empty.

**Claim**: $R_g(\mathbf{u},\mathbf{v})$ is empty iff $S_1$ and $S_2$ are disjoint.

**Proof of claim**: To show the "if" direction, we can simulate the iterative algorithm to compute the receiver set. We start with $R = P$. Then, because $S_1$ and $S_2$ are disjoint, either $r \notin S_1$ or $r \notin S_2$. Assume, without loss of generality, that $r \notin S_1$. Now, $u_{a_r} = t^L_{a_r} < g_{a_r}(R)$, and hence $a_r$ must drop out of the receiver set $R$. But now, because of strict cross-monotonicity, it follows that $g_{b_r}(P - \{a_r\}) > g_{b_r}(P) = t^H_{b_r}$, and so $b_r$ must also drop out of the receiver set. Repeating this argument for $r-1, r-2, \ldots, 1$, we can show that the receiver set must be empty.

To show the "only if" direction, assume that $i \in S_1 \cap S_2$. Then, let $T = \{a_1, \ldots, a_i, b_1, \ldots, b_i\}$. $u_{a_i} = t^H_{a_i} = g_{a_i}(T)$, and $v_{a_i} = t^H_{b_i} = g_{b_i}(T)$. Further, for all $j < i$, it follows from strict cross-monotonicity that $g_{a_j}(T) < t^L_{a_j} \leq u_{a_j}$, and $g_{b_j}(T) < t^L_{b_j} \leq v_{b_j}$. Thus, the receiver set $R_g(\mathbf{u},\mathbf{v}) \supseteq T$, and hence it is nonempty. $\square$

Theorem 3.1 follows from this claim and the communication complexity of set disjointness. $\square$

### 3.7.1 Strictly Subadditive Cost Functions

In this section, we show that, for a class of submodular cost functions, the Shapley-value mechanism (which is perhaps the most compelling mechanism from an economic point of view) is strictly cross-monotonic and hence has poor network complexity.

Theorem 3.1 provides a sufficient condition, strict cross-monotonicity, for a mechanism to have poor network complexity. However, for some submodular cost functions, it is possible that no mechanism satisfies this condition: If the costs are additive, *i.e.*, if the cost of serving a set $S$ is exactly the sum of the costs of serving each of its members individually, then there is a unique mechanism satisfying the basic properties. This mechanism is defined by:

$$R(\mathbf{u}) = \{i | u_i \geq C(\{i\})\}$$

$$x_i(\mathbf{u}) = C(\{i\}) \text{ if } i \in R(\mathbf{u}), \text{ and } x_i(\mathbf{u}) = 0 \text{ otherwise.}$$

This mechanism is very easy to compute, either centrally or in a distributed manner, because there is no interaction among the users' utilities; in essence, we have $|P|$ independent local computations to perform.

We need to exclude these trivial cost functions in order to prove general lower bounds for a class of submodular functions. This leads us to consider submodular cost functions that are *strictly subadditive*:

$$\forall S \subseteq P, S \neq \emptyset, \;\; \forall i \in P, \;\; C(S \cup \{i\}) < C(S) + C(\{i\})$$

For a given cost function $\mathcal{C}$, there may be many $g = \{g_i\}$ for which the corresponding mechanism $M_g$ satisfies the basic properties NPT, VP, CS, and SYM. However, Moulin and Shenker [Mou99, MS01] have shown that, for any given submodular cost function, the cross-monotonic mechanism that minimizes the worst-case efficiency loss is the Shapley-value mechanism (SH). This is a cross-monotonic cost-sharing mechanism corresponding to a function $g^{SH}$, defined by:

$$\forall S \subseteq P \quad \forall i \in S, \qquad g_i^{SH}(S) = \sum_{R \subseteq S - \{i\}} \frac{|R|!(|S| - |R| - 1)!}{|S|!}[C(R \cup \{i\}) - C(R)] \qquad (3.3)$$

The SH mechanism is therefore a natural mechanism to choose for submodular cost sharing. The following lemma shows that this mechanism has poor network complexity.

**Lemma 3.1** *The SH mechanism for a strictly subadditive cost function is strictly cross-monotonic.*

**Proof:** We need to show that, for all sets $S$, for all $i \in S, j \notin S$, $g_i^{SH}(S \cup \{j\}) < g_i^{SH}(S)$. The proof follows directly from the definition of $g_i^{SH}(S)$ in Equation 3.3. We use $\mathrm{MC}_i(R)$ to denote $[C(R \cup \{i\}) - C(R)]$, the *marginal cost* of serving $i$ in set $R$. Consider a set

$S \subseteq P - \{j\}$ and a user $i \in S$. Let $r = |R|, s = |S|$.

$$\begin{aligned}
g_i^{SH}(S \cup \{j\}) &= \sum_{R \subseteq S \cup \{j\} - \{i\}} \frac{r!(s-r)!}{(s+1)!} \mathrm{MC}_i(R) \\
&= \sum_{R \subseteq S - \{i\}} \left[ \frac{r!(s-r)!}{(s+1)!} \mathrm{MC}_i(R) + \frac{(r+1)!(s-r-1)!}{(s+1)!} \mathrm{MC}_i(R \cup \{j\}) \right] \\
&= \sum_{R \subseteq S - \{i\}} \frac{r!(s-r-1)!}{s!} \left[ \frac{s-r}{s+1} \mathrm{MC}_i(R) + \frac{r+1}{s+1} \mathrm{MC}_i(R \cup \{j\}) \right] \qquad (3.4)
\end{aligned}$$

It follows from submodularity of costs that, for all $R$, $\mathrm{MC}_i(R \cup \{j\}) \leq \mathrm{MC}_i(R)$. Further, strict subadditivity implies that, for $R = \emptyset$, $\mathrm{MC}_i(R \cup \{j\}) < \mathrm{MC}_i(R)$. Thus, Equation 3.4 yields

$$\begin{aligned}
g_i^{SH}(S \cup \{j\}) &< \sum_{R \subseteq S - \{i\}} \frac{r!(s-r-1)!}{s!} \mathrm{MC}_i(R) \\
g_i^{SH}(S \cup \{j\}) &< g_i^{SH}(S)
\end{aligned}$$

□

**Corollary 3.1** *For a strictly subadditive cost function, any algorithm (deterministic or randomized) that computes the SH mechanism in a network must communicate $\Omega(|P|)$ bits across any cut that has $\Theta(|P|)$ users on each side of the cut.* □

Note that the network may consist of a root node $\alpha_s$ with no resident users, a node $A$ with $\frac{|P|}{2}$ resident users, another node $B$ with $\frac{|P|}{2}$ resident users, a link from $\alpha_s$ to $A$, and a path from $A$ to $B$ consisting of $|N| - 3$ nodes, each with no resident users. Each link in the path from $A$ to $B$ is a cut with $\Theta(|P|)$ users on each side, and thus $\Omega(|P|)$ bits must be sent across linearly many links. In what follows, we call these the *path instances.*

### 3.7.2 A class of strictly cross-monotonic mechanisms

Feigenbaum *et al.* [FPS01] conjectured (in the context of multicast cost sharing) that the high communication requirements are not specific to the Shapley-value mechanism but arise for any cross-monotonic mechanism that satisfies a property they called *nonseparability.*

Here, we present a result that captures the essence of their conjecture – that "most" cross-monotonic mechanisms have poor network complexity.

Recall the polytope $\mathcal{P}(\mathcal{C})$ of cross-monotonic mechanisms, first introduced by Jain and Vazirani [JV01], that was described in Section 3.6. We observe that, for strictly subadditive cost functions, $x_g \in \mathcal{P}(\mathcal{C})$ is an *interior* point of $\mathcal{P}(\mathcal{C})$ *if and only if* none of the inequalities in condition (3.2) is satisfied with equality. In other words, for any interior point $x_g$, the mechanism $M_g$ is strictly cross-monotonic. This leads to the following result:

**Theorem 3.2** *For a strictly subadditive submodular cost function $\mathcal{C}$, let $M_g$ be a cross-monotonic cost sharing mechanism corresponding to an interior point of the polytope $\mathcal{P}(\mathcal{C})$. Then, any deterministic or randomized algorithm that computes $M_g$ in a network must communicate $\Omega(|P|)$ bits across any cut that has $\Theta(|P|)$ users on each side of the cut.* $\square$

### 3.7.3 Multicast cost sharing

We now return to the special case of multicast cost sharing. Recall that the cost function associated with an instance of the multicast cost-sharing problem is determined by the structure of the universal multicast tree $T$, the link costs, and the locations of the users in the tree; so the cost $C(S)$ of serving user set $S \subseteq P$ is $\sum_{l \in T(S)} c(l)$, where $T(S)$ is the smallest subtree of $T$ that includes all nodes at which users in $S$ reside. It is not hard to show that there are many instances that give rise to strictly subadditive functions $\mathcal{C}$. In fact, we have the following lemma:

**Lemma 3.2** *Consider any instance of multicast cost sharing in which, for every two potential receivers $i$ and $j$, there exists a link $l \in T(\{i\}) \cap T(\{j\})$ such that $c(l) > 0$. The cost function associated with this instance is strictly subadditive.*

**Proof:** Given $S \subseteq P - i$, pick any $j \in S$. Let $l$ be any link in $T(\{j\}) \cap T(\{i\})$ with $c(l) > 0$. Then, $C(S \cup \{i\}) \leq C(S) + C(\{i\}) - c(l) < C(S) + C(\{i\})$. $\square$

For example, whenever the source of the multicast has only one link from it, and this link has non-zero cost, the associated cost function is strictly subadditive. One such family of instances (parametrized by $n = |P|$) is shown in Figure 3.3. There are three nodes, $\alpha_s$, $A$, and $B$, and $n$ users. There are $(n/2)$ users at each of $B$ and $A$, with utilities $u_1, u_2, \ldots, u_{\frac{n}{2}}$

and $v_1, v_2, \ldots, v_{\frac{n}{2}}$, respectively. The link $l_1$ between $\alpha_s$ and $A$ costs $C$, while the link $l_2$ between $A$ and $B$ is free (has 0 cost).

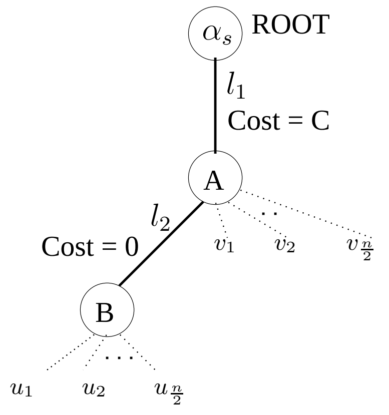


Figure 3.3: Multicast tree with strictly subadditive costs

It follows immediately from Corollary 3.1 that the Shapley-value mechanism for this family of trees requires $\Omega(|P|)$ bits of communication across linearly many links. In addition, we now show that any mechanism that satisfies the basic properties, such as EG, must be identical to the SH mechanism on instances of the type shown in Figure 3.3; thus, the lower bound extends to all such mechanisms.

**Lemma 3.3** *Consider multicast cost-sharing instances of the type shown in Figure 3.3. Let $M_g$ be a cross-monotonic cost-sharing mechanism that satisfies SYM, corresponding to a cost-sharing function $g = \{g_i\}$. Then, $g$ (and $M_g$) are completely determined on these instances by*

$$\forall S \subseteq P, \forall i \in S \quad g_i(S) = \frac{C}{|S|} \ .$$

**Proof:** For any receiver set $S$, if the utility values of all users in $S$ are increased to some sufficiently large value, the receiver set will still be $S$. Because the mechanism satisfies SYM, this implies that $g_i(S) = g_j(S)$ for any pair $i, j$ in this set. The budget-balance requirement then forces $g_i(S) = \frac{C}{|S|}$ for all $i$ in set $S$. □

It follows from Lemma 3.3 that any such mechanism must be strictly cross-monotonic on this family of instances. Thus, Theorem 3.1 and Lemma 3.3 imply the following lower bound for multicast cost sharing. The worst-case instances include the path instances defined in Section 3.7.1, with cost $C$ on the link from $\alpha_s$ to $A$ and cost 0 on all the other links; these instances are identical to the ones in Figure 3.3, except that they contain a zero-cost path of length $|N| - 2$ from $A$ to $B$ instead of a single zero-cost link.

**Theorem 3.3** *Any distributed algorithm, deterministic or randomized, that exactly computes a budget-balanced, GSP multicast cost-sharing mechanism that satisfies the four basic properties must send $\Omega(|P|)$ bits over linearly many links in the worst case.* □

Note that this lower bound applies to the EG mechanism for multicast cost-sharing referred in Section 3.1.

## 3.8 Network complexity of approximately budget-balanced mechanisms

In view of the lower bounds presented in Section 3.7, it is natural to ask whether one can *approximate* a budget-balanced, GSP mechanism in a communication-efficient manner. In this case, we do not have a clean analogue of Corollary 3.1, because cross-monotonic cost functions no longer characterize the class of feasible mechanisms. However, for the special case of multicast cost sharing, we can still prove a result similar to Theorem 3.3 that provides a lower bound on the network complexity of approximately budget-balanced, GSP mechanisms.

First, we recall the definition of an "approximately budget-balanced" mechanism. As explained at length in, *e.g.*, [NR00], one cannot define an approximation of a cost-sharing mechanism $(\sigma, \mathbf{x})$ simply as a pair $(\sigma', \mathbf{x}')$ such that $\sigma'$ and $\mathbf{x}'$ approximate $\sigma$ and $\mathbf{x}$, respectively, *as functions*. Such an approach may destroy the game-theoretic properties of $(\sigma, \mathbf{x})$, *e.g.*, the resulting "mechanism" $(\sigma', \mathbf{x}')$ may not be strategyproof! For our purposes in this section, a *$\kappa$-approximately budget-balanced mechanism*, where $\kappa > 1$ is a constant, is a mechanism $(\sigma, \mathbf{x})$ with the following properties: VP, NPT, CS, SYM, and

$$\forall z = (N, T(P), c(\cdot), \alpha_s), \text{ and } \mathbf{u} : (1/\kappa) \cdot c(T(R(\mathbf{u}))) \leq \sum_{i \in R(\mathbf{u})} x_i(\mathbf{u}) \leq \kappa \cdot c(T(R(\mathbf{u}))).$$

An approximation to a specific budget-balanced mechanism such as SH or EG would have to satisfy at least one additional (non-strategic) condition. For example, because SH is the GSP, budget-balanced mechanism that minimizes worst-case efficiency loss, an approximation to SH would have to come within a constant factor of SH's efficiency loss in the worst case.

We extend the lower-bound technique of the previous section so that it applies to $\kappa$-approximately budget-balanced mechanisms, when $\kappa$ is upper-bounded away from $\sqrt{2}$, *i.e.*, when $\kappa \leq \sqrt{2} - \epsilon$, for some fixed $\epsilon > 0$. As before, we want to reduce from the set-disjointness problem where node $A$ has a set $S_1 \subseteq \{1, 2, \ldots, r\}$, node $B$ has another set $S_2 \subseteq \{1, 2, \ldots, r\}$, and one must determine whether the sets $S_1$ and $S_2$ are disjoint. We again construct the multicast tree shown in Figure 3.3 with $(n/2)$ users at each of $B$ and $A$, with utilities $u_1, u_2, \ldots, u_{\frac{n}{2}}$ and $v_1, v_2, \ldots, v_{\frac{n}{2}}$, respectively.

We first prove some basic lemmas about group-strategyproof mechanisms for this multicast cost sharing problem.

**Lemma 3.4** *Let $M$ be a $\kappa$-approximately budget-balanced mechanism for the multicast cost sharing problem in Figure 3.3 that satisfies GSP. Then, if $\mu$ is a utility profile of the $n$ users such that*

$$\exists h \geq 1 \text{ such that } \forall i \in \{1, 2, \ldots, h\} \quad \mu_i > \kappa C/h$$

*then the receiver set $R(\mu)$ specified by this mechanism is nonempty.*

**Proof:** Let $\mu$ be such a utility profile, and consider any value of $h$ for which the given condition holds. Let $\mu_{cs}$ be the bound for which the CS condition holds, *i.e.*, if $\mu_i \geq \mu_{cs} \implies i \in R(\mu)$. Let $S = \{1, 2, \ldots, h\}$. Define a utility profile $\mu^S$ by

$$\begin{aligned} \mu_i^S &= \mu_{cs} \quad \forall i \in S \\ \mu_i^S &= \mu_i \quad \forall i \notin S \end{aligned}$$

By the CS condition, $S \subseteq R(\mu^S)$. Further, by the SYM condition, we must have $\forall i, j \in S,\ \ x_i(\mu^S) = x_j(\mu^S)$. Further, because the NPT condition implies that $x_i(\mu^S) \geq 0$, for all $i \notin S$, and the approximate budget-balance condition requires that the revenue be bounded by $\kappa C$, we must have $x_i(\mu^S) \leq \kappa C/h$, for all $i \in S$.

It follows that $\forall i \in S$, $x_i(\mu^S) < \mu_i$. Now suppose that $R(\mu)$ is empty. Then, at utility profile $\mu$, the coalition $S$ could strategize to report a utility profile $\mu^S$; then, for each $i \in S$, $i$ would receive the transmission and pay less than $\mu_i$ for it. This would constitute a successful group strategy, which contradicts the assumption that $M$ is group-strategyproof. $\square$

**Lemma 3.5** *Let $M$ be a $\kappa$-approximately budget-balanced mechanism for the multicast cost sharing problem in Figure 3.3 that satisfies GSP. Then, if $\mu$ is a utility profile such that*

$$\mu_1 \geq \mu_2 \geq \ldots \geq \mu_n \qquad \textit{and}$$

$$\nexists h \geq 1 \textit{ such that } \quad \mu_h \geq C/(\kappa h)$$

*then the receiver set $R(\mu)$ specified by this mechanism is empty.*

**Proof:** Let $\mu$ be such a utility profile, and let $S = R(\mu)$. Suppose that $S \neq \emptyset$. Let $h = \max\{i | i \in S\}$, which implies that $\forall i \in S,\ \mu_i \geq \mu_h$. By the conditions of the lemma, $\mu_h < C/(\kappa h)$; thus, the approximate budget-balance condition combined with VP implies that $\exists j \in S$ such that $x_j(\mu) > x_h(\mu)$. It then follows that $\mu_j > \mu_h$. (If $\mu_j = \mu_h$, then by SYM we would have $x_j(\mu) = x_h(\mu)$.)

Now, define the utility profile $\mu'$ by

$$\begin{aligned} \mu'_h &= \mu_j \\ \mu'_i &= \mu_i \quad \forall i \neq h \end{aligned}$$

If $h \notin R(\mu')$, then at utility profile $\mu'$, $h$ could strategize to report $\mu_h$ and get transmission with payment $x_h(\mu)$; this would be a successful strategy because $x_h(\mu) < x_j(\mu) \leq \mu_j = \mu'_h$. Mechanism M is strategyproof, so we must have $h \in R(\mu')$. Further, we must also have $x_h(\mu') = x_h(\mu)$ for the same reason: If $x_h(\mu') > x_h(\mu)$, then $h$ could strategize at $\mu'$, and, if $x_h(\mu) > x_h(\mu')$, then $h$ could strategize at $\mu$.

Now, by applying the SYM condition at $\mu'$, we must have $j \in R(\mu')$ and $x_j(\mu') = x_h(\mu')$. This implies that $x_j(\mu') = x_h(\mu) < x_j(\mu)$. But now, $h$ and $j$ could collude and strategize at $\mu$ to report $\mu'$; this strictly increases $j$'s welfare (as her payment is strictly reduced), and leaves $h$'s welfare unchanged, and hence it would be a successful group strategy. This contradicts the fact that M satisfies GSP. $\square$

The ordering condition $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_n$ in Lemma 3.5 is only included for simplicity of the exposition; we can always relabel the agents such that it holds.

**Theorem 3.4** *Any distributed algorithm, deterministic or randomized, that computes a $\kappa$-approximately budget-balanced, GSP multicast cost-sharing mechanism, where $\kappa \leq \sqrt{2} - \epsilon$ for some fixed $\epsilon > 0$, must send $\Omega(\frac{\log |P|}{\log \kappa})$ bits of communication over linearly many links in the worst case.*

**Proof:** It is more convenient to work with an alternative representation of the input utility vectors. We use only a restricted set of the possible utility vectors $\mathbf{u}$ and $\mathbf{v}$ for the users located at nodes $B$ and $A$ respectively, where $(\mathbf{u}, \mathbf{v})$ satisfies the following conditions:

- Let $\beta = 3(\kappa + \delta)^2/(2 - (\kappa + \delta)^2)$, where $\delta > 0$ is an arbitrarily small constant only required to make the inequalities strict. Restrict the set of allowable utilities to $T = \{0, t_{\beta,1}, \ldots, t_{\beta,r}\}$, where
$$t_{\beta,i} = \frac{(\kappa + \delta)C}{2\beta^{i+1}} .$$
- Each of $\mathbf{u}$ and $\mathbf{v}$ is (internally) sorted, *i.e.*, $i < j \implies u_i \geq u_j$ and $v_i \geq v_j$. There is no restriction on the relationship between $u_i$ and $v_j$.

Consider node $B$. Define $n_B(q)$ to be the number of users at node $B$ who have utility at least $q$. Let $\vec{y} = (y_1, y_2, \ldots, y_r)$, where $y_i = n_B(t_{\beta,i})$. Similarly, let $n_A(q)$ be the number of users at $A$ with utility at least $q$; let $\vec{z} = (z_1, z_2, \ldots, z_r)$, where $z_i = n_A(t_{\beta,i})$.

For this class of utility profiles, there is a one-to-one mapping between values of $\mathbf{u}$ and $\vec{y}$. Because $\mathbf{u}$ is sorted, the monotonically decreasing function $n_B(\cdot)$ completely defines the utility vector; $u_1$ must be the largest $q$ for which $n_B(q) > 0$, and so on. Furthermore, by definition, there is a unique $\vec{y}$ for any $\mathbf{u}$. A similar correspondence holds for $\mathbf{v}$ and $\vec{z}$.

We first prove a useful lemma about approximately budget-balanced mechanisms on this class of instances.

**Lemma 3.6** *Let $M$ be a $\kappa$-approximately budget-balanced mechanism for the multicast cost sharing problem in Figure 3.3 that satisfies GSP. Let vectors $\vec{y}$, $\vec{z}$ be defined corresponding to the utility profiles $\mathbf{u}$ and $\mathbf{v}$, as described above. Then, $M$ satisfies the following two properties:*

*(i). If there is an $i$ such that $(y_i + z_i) \geq 2\beta^{i+1}$, then mechanism $M$ will compute a non-empty receiver set on this instance.*

*(ii). If for all $i$ we have $(y_i + z_i) \leq (3\beta^i + \beta^{i+1})$, then mechanism $M$ will compute an empty receiver set on this instance.*

**Proof:** (i): Observe that with a suitable ordering of the players, the conditions of Lemma 3.4 are satisfied if $(y_i + z_i) \geq 2\beta^{i+1}$.

(ii): Assume that the receiver set is non-empty and that the conditions of Lemma 3.5 do not apply due to the presence of some $h$ such that $h \cdot \mu_h \geq C/\kappa$, where $\mu$ is the utility profile of $P$ sorted in decreasing order.

Let $\mu_h = t_{\beta,k}$ for some $k$. Since $h \leq y_k + z_k$, we note that $h \cdot \mu_h \leq (y_k + z_k)t_{\beta,k} \leq (3\beta^k + \beta^{k+1})t_{\beta,k} = C/(\kappa + \delta)$. This violates the assumption that $h \cdot \mu_h \geq C/\kappa$. □

We now use Lemma 3.6 to provide a reduction from the set disjointness problem as follows. Recall that node $A$ has a set $S_1 \subseteq \{1, 2, \ldots, r\}$ and node $B$ has another set $S_2 \subseteq \{1, 2, \ldots, r\}$. We must make sure that, if $S_1 \cap S_2 \neq \emptyset$, there is a set of receivers who can share $\kappa C$, and, if $S_1 \cap S_2 = \emptyset$, there is no set of receivers who can share even $C/\kappa$. For this, we construct the vectors $\vec{y}$ and $\vec{z}$ using the rules:

$$\begin{aligned}
y_i &= \lceil \beta^{i+1} \rceil \quad \text{if } i \in S_2 \\
y_i &= \lceil \beta^{i} \rceil \quad \text{otherwise} \\
z_i &= \lceil \beta^{i+1} \rceil \quad \text{if } i \in S_1 \\
z_i &= \lceil \beta^{i} \rceil \quad \text{otherwise}
\end{aligned}$$

These are valid input vectors, because $y_i \leq y_{i+1}$ and similarly for $\vec{z}$. If $i \in S_1 \cap S_2$, then

$y_i + z_i \geq 2\beta^{i+1}$, and so there is transmission. If $S_1$ and $S_2$ are disjoint, then, for all $i$,

$$y_i + z_i < \beta^i + \beta^{i+1} + 2 < 3\beta^i + \beta^{i+1} \ ,$$

where the 2 arises because of the ceiling terms.

This means that, in the target instance,

$$(y_i + z_i)t_{\beta,i} < \frac{C}{\kappa} \ ,$$

and consequently there is no transmission.

Thus, we can use this $\kappa$-approximate mechanism to solve the set-disjointness problem, and this implies that the mechanism must use $\Omega(r)$ bits of communication, where

$$r = \Theta\left(\frac{\log n}{\log \beta}\right) = \Theta\left(\frac{\log n}{2\log(\kappa+\delta) - \log(2-(\kappa+\delta)^2)}\right).$$

If we require $\kappa$ to be upper-bounded away from $\sqrt{2}$, then the right-hand side is $\Theta(\frac{\log n}{\log \kappa})$. Thus, the statement of the theorem follows. □

We note that this lower bound applies to the approximate mechanisms described in [JV01], as well as to the approximations to SH that we will present in Section 3.11. The mechanisms SF and SSF described in this chapter provide the best known corresponding upper bound: They require $\Omega(h \cdot \frac{\log n}{\log \kappa})$ utility values to be communicated on each link to achieve $\kappa$-approximate budget balance, where $h$ is the height of the multicast tree $T(P)$.

## 3.9 An impossibility result for approximate budget-balance and approximate efficiency

As stated in Section 3.1, it is a classical result in game theory that no strategyproof cost-sharing mechanism can be both budget-balanced and efficient [GL79]. We now consider whether this fundamental impossibility result holds when the budget-balance and efficiency considerations are replaced by their approximate counterparts. In this section, we do not assume that the cost-sharing mechanisms have the SYM property; the impossibility result that we present here does not require this assumption. Furthermore, this result only requires the mechanism to be strategyproof, not GSP as in Section 3.8.

We first review the definition of the MC mechanism, which was shown by Moulin and Shenker [MS01] to be the only efficient mechanism that satisfies VP, NPT, and CS. Given an input utility profile $\mathbf{u}$, the MC receiver set is the unique *largest efficient set* of users. To compute it, as shown in [FPS01], one recursively computes the *welfare* (also known as *net worth* or *efficiency*) of each node $\beta \in N$:

$$W(\beta) = \left( \sum_{\substack{\gamma \in Ch(\beta) \\ W(\gamma) \geq 0}} W(\gamma) \right) - c(l) + \sum_{i \in Res(\beta)} u_i \ ,$$

where $Ch(\beta)$ is the set of children of $\beta$ in the tree, $Res(\beta)$ is the set of users resident at $\beta$, and $c(l)$ is the cost of the link connecting $\beta$ to its parent node. Then, the largest efficient set $R(\mathbf{u})$ is the set of all users $i$ such that every node on the path from $i$ to the root $\alpha_s$ has nonnegative welfare. The *total efficiency* is $\mathrm{N}W(R(\mathbf{u})) = W(\alpha_s)$.

Another way to view this is as follows: The algorithm partitions the universal tree $T(P)$ into a forest $F(\mathbf{u}) = \{T_1(\mathbf{u}), T_2(\mathbf{u}), \ldots, T_k(\mathbf{u})\}$. A link from $T(P)$ is included in the forest if and only if the child node has nonnegative welfare. $R(\mathbf{u})$ is then the set of users at nodes in the subtree $T_1(\mathbf{u})$ containing the root.

Once $F(\mathbf{u})$ has been computed, for each user $i$, define $X(i, \mathbf{u})$ to be the node with minimum welfare value in the path from $i$ to its root in its partition. Then, the cost share $x_i(\mathbf{u})$ of user $i$ is defined as

$$\begin{aligned} x_i(\mathbf{u}) = &\quad \max(0, u_i - W(X(i, \mathbf{u}))) && \forall i \in R(\mathbf{u}) \\ x_i(\mathbf{u}) = &\quad 0 && \forall i \notin R(\mathbf{u}) \end{aligned}$$

If multiple nodes on the path have the same welfare value, we let $X(i, \mathbf{u})$ be the one nearest to $i$.

By a $\gamma$-*approximately efficient mechanism*, where $0 < \gamma < 1$, we mean one that always achieves total efficiency that is at least $\gamma$ times the total efficiency achieved by MC.

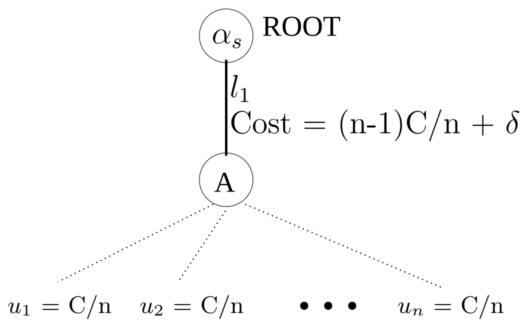


Figure 3.4: An example of a multicast tree that fails to achieve approximate efficiency and approximate budget-balance.

We can ask whether there is any strategyproof mechanism that satisfies the basic requirements of NPT, VP, and CS and is both approximately efficient and approximately budget-balanced. We now show that this is impossible, using the following approach: We construct a family of multicast trees and utility profiles for which any approximately efficient mechanism must transmit to all users. We show that the strategyproofness condition and the VP condition together place an upper bound on the revenue collected in these instances. This upper bound is less than that required for even approximate budget balance, and hence we have our negative result.

Consider the tree in Figure 3.4. There are $n$ users, each with utility $C/n$ resident at a node A that is separated from the root node by a link of cost $((n-1)C/n) + \delta$. It is easy to see that this instance of multicast cost-sharing displays the following properties.

**Property 1** *Any $\gamma$-approximately efficient mechanism must transmit to all $n$ users if* $0 < \delta < C/n$.

**Property 2** *Any $\gamma$-approximately efficient mechanism must transmit to all $n$ users even if one user, say $i$, lowers his utility to $\delta + \epsilon$, for any $\delta, \epsilon > 0$.*

**Property 3** *Any $\gamma$-approximately efficient, strategyproof mechanism that satisfies VP assigns to each user a cost share of at most $\delta$.*

If the cost share $x_i(\mathbf{u})$ were greater than $\delta$, the user $i$ could strategize by claiming that his utility was $v_i = \delta + \epsilon < x_i(\mathbf{u})$. By VP and the requirement of $\gamma$-approximate efficiency, the mechanism would need to include user $i$ and assign him a cost share $x_i(\mathbf{u}|^i v_i) \le v_i < x_i(\mathbf{u})$, which would imply a violation of strategyproofness.

Therefore, the revenue collected by a strategyproof mechanism that achieves $\gamma$-approximate efficiency is bounded from above by $n\delta$.

**Property 4** *A $\gamma$-approximately efficient, strategyproof mechanism cannot be $\kappa$-approximately budget-balanced if $\delta < (C(n-1))/(n(\kappa n - 1))$.*

In summary, we have:

**Theorem 3.5** *A strategyproof mechanism for multicast cost sharing that satisfies the basic requirements of NPT, VP, and CS cannot achieve both $\gamma$-approximate efficiency and $\kappa$-approximate budget-balance for any pair of constants $\kappa$ and $\gamma$.* $\square$

## 3.10 Strategically Faithful Approximate Mechanisms

In view of the proof given in Section 3.7 that exact computation of the SH mechanism has unacceptably high communication cost, it is natural to ask the following question: Can one compute an *approximation* to the SH mechanism using an algorithm that is significantly more communication-efficient? To approach this question, we must first say what it means to "approximate the SH mechanism."

A multicast cost-sharing mechanism is a pair of functions $(\sigma, \mathbf{x})$. Thus, one may be tempted to define an approximation of the mechanism as a pair of functions $(\sigma', \mathbf{x}')$ such that $\sigma'$ approximates $\sigma$ well (for each $\mathbf{u}$, these are characteristic vectors of subsets of $P$; so, we may call $\sigma'$ a good approximation to $\sigma$ if, for each $\mathbf{u}$, the Hamming distance between the vectors is small), and $\mathbf{x}'$ approximates $\mathbf{x}$ well (in the sense, say, that, for some $p$, the $L^p$-difference of $\mathbf{x}(\mathbf{u})$ and $\mathbf{x}'(\mathbf{u})$ is small, for each $\mathbf{u}$). The mechanism $(\sigma', \mathbf{x}')$, however, would not be interesting if its game-theoretic properties were completely different from those of $(\sigma, \mathbf{x})$. In particular, if $(\sigma', \mathbf{x}')$ were not strategyproof, then agents might misreport their utilities; thus, even if $(\sigma, \mathbf{x})$ and $(\sigma', \mathbf{x}')$ were, for each $\mathbf{u}$, approximately equal as pairs of functions, the resulting equilibria might be very different, *i.e.*, $(\sigma'(\mu), \mathbf{x}'(\mu))$ might be very far from $(\sigma(\mathbf{u}), \mathbf{x}(\mathbf{u}))$, where $\mu$ is the reported utility vector when using the approximate mechanism $(\sigma', \mathbf{x}')$. Thus, we require that our approximate mechanisms retain the strategic properties – strategyproof or group-strategyproof – of the mechanism that they are approximating. In addition, if the original mechanism has some property, such as budget balance or efficiency, that does not relate to the underlying strategic behavior of agents but is an important design goal of the mechanism, then we would want the approximate mechanism to approximate that property closely.

The SH mechanism is GSP, budget-balanced, and, among all mechanisms with these two properties, the unique one that minimizes the worst-case welfare loss. We should therefore strive for a GSP mechanism that has low network complexity and is approximately budget-balanced and approximately welfare-loss minimizing in the worst case. "Approximately budget-balanced"[4] can be taken to mean that there is a constant $\beta > 1$ such that, for all problem instances $z = (N, T(P), c(\cdot), \alpha_s$, and utility profiles $\mathbf{u}$:

$$(1/\beta) \cdot c(T(R(\mathbf{u}))) \leq \sum_{i \in R(\mathbf{u})} x_i(\mathbf{u}) \leq \beta \cdot c(T(R(\mathbf{u})))$$

The *efficiency loss* of a mechanism $M$ on a particular run $I = (z, \mathbf{u})$ is the difference between the optimal net worth of $I$ (*i.e.*, that realized by the MC mechanism) and the net worth realized by $M$. (Recall that the parameter $z = (N, T(P), \{c(l)\}, \alpha_s)$ describes the non-private information for a particular instance of the problem, *i.e.*, it describes the cost structure associated with the multicast.) The SH mechanism minimizes the worst-case loss in the following sense: For any given cost structure $z$ , the worst-case efficiency loss $L(z)$ of a mechanism $M$ on this cost structure is the maximum, over all possible utility profiles $\mathbf{u}$, of the efficiency loss of $M$ on the instance $(z, \mathbf{u})$. Among all GSP, budget-balanced mechanisms, the SH mechanism achieves the minimum $L(z)$, for any parameter $z$; further, SH is the only mechanism to achieve this minimum for *all* cost structures $z$. A mechanism $M$ is "approximately efficiency-loss minimizing in the worst case" if there is a constant $\gamma > 1$ such that, for all cost structures $z$, the worst-case efficiency loss of $M$ on this cost structure is at most $\gamma$ times the worst-case efficiency loss of SH on the same cost structure.

[4]An alternative definition of approximate budget balance could allow for only a one-sided error, *e.g.*, a surplus but not a deficit, as in [JV01].

We do not obtain an approximate SH mechanism here, but we do make some progress toward the goal; our mechanism is GSP and fails to achieve exact budget balance and exact minimum-welfare loss by bounded amounts, but the bounds are not constant factors. Furthermore, there is a distributed algorithm that computes this mechanism using far less communication over the links of $T(P)$ than appears to be needed for SH computation.

This notion of approximating a mechanism $M$ that we use in this chapter – roughly, "retain the strategic properties of $M$ but approximate the other mechanism design goals" – is called *strategically faithful approximation*. Approximation is an increasingly active area of algorithmic mechanism design, and several other interesting notions of approximation have been put forth – see Section 5 of [FS02] for an overview. Here we mention only the work that is most closely related to the results in this chapter.

[NR00] were the first to address the question of approximate computation in algorithmic mechanism design. They considered VCG mechanisms in which optimal outcomes are NP-hard to compute (as they are in combinatorial auctions). They pointed out that, if an optimal outcome is replaced by a computationally tractable approximate outcome, the resulting mechanism may no longer be strategyproof. The above discussion of how we should define "approximating the SH mechanism" and why approximating the pair of functions $(\sigma, \mathbf{x})$ is not sufficient is based on the analogous observation in our context. [NR00] approach this problem by developing a notion of "feasible" strategyproofness and describing a broad class of situations in which NP-hard VCG mechanisms have feasibly strategyproof approximations. This approach is not applicable to SH-mechanism approximation for several reasons: SH is not a VCG mechanism; we are not seeking an approximation to an NP-hard optimization problem but rather a communication-efficient approximation to an apparently communication-inefficient, but polynomial-time computable, function; we are interested in network complexity in a distributed computational model, and [NR00] were interested in time complexity in a centralized computational model. Approximate multicast cost sharing was first addressed by [JV01]. They exhibited a GSP, approximately budget-balanced,[5] polynomial-time mechanism based on a 2-approximation algorithm for

[5]The [JV01] definition of approximate budget balance is more stringent than the one we suggest in this section; it does not allow a budget deficit (and also requires, as ours does, a constant-factor bound on the budget surplus).

the minimum-Steiner-tree problem. Their approach is also not applicable to SH-mechanism approximation, because they are concerned with time complexity in a centralized computational model, their network is a general directed graph (rather than a multicast tree, as it is in our case), and they are not attempting to approximate minimum worst-case welfare loss. Finally, "competitive-ratio" analysis (a form of approximation) has been studied for a variety of strategyproof auctions (see, *e.g.*, [FGHK02], [GHW01], and [LN00]).

## 3.11 Towards approximating the SH mechanism

In this section, we develop a GSP mechanism that exhibits a trade-off between the other properties of the Shapley value: It can be computed by an algorithm that is more communication-efficient than the natural SH algorithm (exponentially more so in the worst case), but it might fail to achieve exact budget balance or exact minimum welfare loss (albeit by a bounded amount).

First, in Section 3.1, we review the natural SH algorithm given in [FPS01]. In Section 3.2, we give an alternative SH algorithm that also has unacceptable network complexity but that leads naturally to our approach to approximation. In Sections 3.3, 3.4, and 3.5, we define a new mechanism that has low network complexity, prove that it is GSP, and obtain bounds on the budget deficit and the welfare loss.

### 3.11.1 The natural multi-pass SH algorithm

The Shapley-value mechanism divides the cost of a link $l$ equally among all receivers downstream of $l$. The mechanism can be characterized by its cost-sharing function $g : 2^P \mapsto \mathbb{R}^P_{\geq 0}$ [MS01, Mou99]. For a receiver set $R \subseteq P$, player $i$'s cost share is $g_i(R)$. [FPS01] present a natural, iterative algorithm that computes SH. We restate it here:

The simplest case of the SH cost-share problem is the one in which all $u_i$ are sufficiently large to guarantee that all of $P$ receives the transmission. (For example, $u_i > C(T(P))$, for all $i$, would suffice.) For this case, the SH cost shares can be computed as follows.[6] Do a bottom-up traversal of the tree that determines, for each node $\alpha$, the number $n_\alpha$ of

[6]This simple case is essentially a distributed version of the linear-time algorithm given in [Meg78].

users in the subtree rooted at $\alpha$. Then, do a top-down traversal, which the root initiates by sending the number $md = 0$ to its children. After receiving message $md$, node $\alpha$ computes $md' \equiv \left(\frac{c(l)}{n_\alpha}\right) + md$, where $l$ is the network link between $\alpha$ and its parent, assigns the cost share $md'$ to each of its resident users, and sends $md'$ to each child. Thus, each user ends up paying a fraction of the cost of each link in its path from the source, where the fraction is determined by the number of users sharing this link.

In the general case, we initially start, as before, with $R = P$ and compute the cost shares as above. However, we cannot assume that $u_i \geq md'$ for all $i$, and so some users may prefer not to receive the transmission. After each pass up and down the tree, we update $R$ by omitting all users $i$ such that $u_i < md'$ and repeat. The algorithm terminates when no more users need to be omitted.

Unfortunately, this algorithm could make as many as $|P|$ passes up and down the tree and send a total of $\Omega(|N| \cdot |P|)$ messages in the worst case. Moreover, [FPS01] contains a corresponding lower bound for a broad family of algorithms: There is an infinite class of inputs, with $|P| = \mathbf{O}(|N|)$, for which any "linear distributed algorithm" that computes SH sends $\Omega(|N|^2)$ messages in the worst case.

### 3.11.2 A one-pass SH algorithm

Our first step toward a more communication-efficient mechanism that has some of the desirable properties of SH is to present a distributed algorithm for SH that makes just one pass up and down the tree. We do this by communicating, in a single message, a digest of the utility profile of all the players in a subtree. This algorithm still sends more than $|N| \cdot |P|$ communication bits in the worst case, and thus it is not directly usable. However, we show in Section 2.3 how approximating the functions communicated in this one-pass SH algorithm leads to a new mechanism that can be computed in a significantly more communication-efficient manner and has other desirable properties.

Let $\mu$ be the (reported) utility profile. Then, for every link $l$ in $T(P)$, the digest we compute is:
$n_l(p, \mu) \overset{def}{=}$ *the number of players in the subtree beneath $l$ who are each willing to pay $p$ for the links above $l$* (*i.e.*, the number of players in this subtree who will not drop out of the

receiver set when their cost share for the links from the root down to but excluding $l$ is $p$).

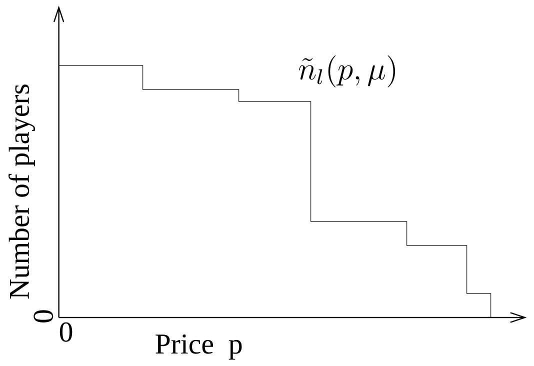


Figure 3.5: The function $n_l(p, \mu)$ computed for each link $l$

(We put the utility profile $\mu$ in explicitly as an argument so that it can be used below in the proof of group strategyproofness; however, in any one run of the algorithm, $\mu$ is fixed.)

Note that this definition requires that the cost from the leaves through $l$ has already been adjusted for. The information conveyed through the function $n_l(p, \mu)$ is a sufficient digest of the costs and utilities in the subtree beneath $l$, because the SH mechanism does not distinguish between receivers downstream of $l$ when sharing the cost of $l$ or its ancestors; all such receivers pay the same amount for these links. For each link, we compute this function at all prices $p$. The function $n_l(p, \mu)$ is monotonically decreasing with $p$, and, for any given utility profile $\mu$, can be represented with at most $|P|$ points with coordinates $(p_i, n_i)$ corresponding to the "corners" in the graph of $n_l(p, \mu)$ in Figure 3.5. We use this *list-of-points* representation of $n_l(p, \mu)$ in our algorithm.

The [FPS01] statement of the multicast cost-sharing problem allows for players at intermediate (non-leaf) nodes; however, to simplify the discussion, we can treat each of these players as if it were a child node with one player and parent link-cost zero. Thus, we assume, without loss of generality, that all players are at leaf nodes only.

The function $n_l(p, \mu)$ is computed at the node $\alpha_l$ below $l$ in the tree. The computation is easy if $\alpha_l$ is a leaf node. Let $p_{\alpha_l}$ be the number of agents at $\alpha_l$, and assume that $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_{p_{\alpha_l}}$. Let $c(l)$ be the cost of link $l$. For a given price $p$, compute $n_l(p, \mu)$ as follows. Let $k = 0$. If $p + \frac{c(l)}{p_{\alpha_l} - k} \leq \mu_{(p_{\alpha_l} - k)}$, then stop with $n_l(p, \mu) = p_{\alpha_l} - k$. Otherwise, increment $k$ by 1 and repeat the test. If $k$ reaches $p_{\alpha_l} - 1$, and the test fails (*i.e.*, if

$p + c(l) > \mu_1$), then stop with $n_l(p, \mu) = 0$.

If $\alpha_l$ is not a leaf node, we have to include the functions reported by its children in this calculation. Suppose we are at node $\alpha_l$ and have received the functions $n_{l_i}(p, \mu)$ from all the child links $\{l_1, l_2, \ldots l_r\}$ of $l$. We can compute $n_l(p, \mu)$ in two steps:

- **Step 1**: First, we compute a function

$$m_l(p, \mu) = \sum_{i=1}^{r} n_{l_i}(p, \mu)$$

  Intuitively, $m_l(p, \mu)$ is the number of players beneath $l$ who are willing to pay $p$ each towards the cost from the root down to (and *including*) $l$. This is apparent from the definition of $n_{l_i}(p, \mu)$. If each $n_{l_i}(\cdot)$ is specified as a sorted list of points, we can compute $m_l(\cdot)$ by merging the lists and adding up the numbers of players.

- **Step 2**: Now, we have to account for the cost $c(l)$ of the link $l$ to compute the function $n_l(p, \mu)$. For any $p$ such that $p \cdot m_l(p, \mu) \geq c(l)$, we have

$$n_l(p - \frac{c(l)}{m_l(p, \mu)}, \mu) \;\geq\; m_l(p, \mu) \;, \tag{3.5}$$

  because the $m_l(p, \mu)$ players who were willing to pay $p$ for the path including $l$ can share the cost of $l$. Equation 3.5 need not be a strict equality because it is possible that, for a price $q < p$, the larger set of size $m_l(q, \mu)$ has

$$q - \frac{c(l)}{m_l(q, \mu)} \geq p - \frac{c(l)}{m_l(p, \mu)}$$

  and so could also support the price $p' = p - (c(l)/m_l(p, \mu))$ each for the links above $l$. However, the value of $n_l(p, \mu)$ must correspond to $m_l(p', \mu)$ for some $p' \geq p$, because every player beneath $l$ who receives the transmission pays an equal amount for the link $l$. It follows that

$$n_l(p, \mu) = \max_{\left\{p' - \frac{c(l)}{m_l(p', \mu)} \geq p\right\}} m_l(p', \mu) \tag{3.6}$$

  When the right hand side of Equation 3.6 is undefined (because there is no $p'$ satisfying the condition), we set $n_l(p, \mu) = 0$. Given a list of points $(p^{(i)}, m^{(i)})$ corresponding to $m_l(\cdot)$, we can compute $n_l(\cdot)$ through the following procedure: For each point $(p^{(i)}, m^{(i)})$, we get the transformed point $(p^{(i)} - (c(l)/m^{(i)}), m^{(i)})$. We then sort the list of these transformed points and throw away any point that is dominated by a higher $m_i$ at the same or higher price.

In this manner, we can inductively compute $n_l(\cdot)$ for all links, until we reach the root. At the root, we can combine the functions received from the root's children to get $m_{root}(\cdot)$. Because there are no further costs to be shared, it follows that there are $m = m_{root}(0, \mu)$ players that are willing to share the costs up to the root. Also, there is no set of more than $m$ players that can support the cost up to the root, and so $m$ is the size of the unique largest fixed-point set computed by the Shapley-value mechanism.

Now, we have to compute the prices charged to each player. Assuming that the nodes have stored the functions $n_l(\cdot)$ on the way up the tree, we compute the prices on the way down as follows: For each link $l$, we let $x_l$ be the cost share of any receiver below $l$ for the path down to (but not including) $l$. If $l$ is the link from node $\beta$ to $\beta$'s parent, then we use $x_l$ and $x_\beta$ interchangeably. Then, $x_{root} = 0$ and, if $l$ has child links $l_1, l_2, \ldots l_k$,

$$x_{l_j} = x_l + \frac{c(l)}{n_l(x_l, \mu)} \tag{3.7}$$

We descend the tree in this manner until we get a price $x_i$ for every player $i \in P$: If $i$ is at node $\beta$, and $l$ is the link from $\beta$ to its parent, then $x_i = x_l + \frac{c(l)}{n_l(x_l,\mu)}$. Then, we include $i$ in $R(\mu)$ iff $x_i \leq \mu_i$, and if included $i$ pays $x_i$.

The following two lemmas show that this one-pass algorithm computes the SH mechanism.

**Lemma 3.7** *The outcome computed by this algorithm is budget-balanced.*

**Proof:** By definition, there are exactly $n_l(x_l, \mu)$ players beneath $l$ who can pay $x_l$ for the path down to but excluding $l$. It follows that

$$\forall j \; n_l(x_l, \mu) = m_l(x_{l_j}, \mu) = \textstyle\sum_i n_{l_i}(x_{l_i}, \mu) \; .$$

Using this inductively until we reach the leaves, we can show that there are $n_l(x_l, \mu)$ players downstream of $l$ in the receiver set chosen by the algorithm, *i.e.*, with $x_i \leq \mu_i$. Equation 3.7 then shows that the cost of each link is exactly balanced, and hence the overall mechanism is budget-balanced. □

**Lemma 3.8** *The receiver set computed by this algorithm is the same as the receiver set computed by the Shapley-value algorithm given in Section 2.1.*

**Proof:** By Lemma 3.7, we know that the set $R(\mu)$ constructed can bear the cost of transmitting to $R(\mu)$. Let $\overline{R}(\mu)$ be the receiver set chosen by the iterative Shapley-value algorithm (*i.e.*, the one in Section 2.1). Because $\overline{R}(\mu)$ is the greatest fixed point, $\overline{R}(\mu) \supseteq R(\mu)$.

We show that $\overline{R}(\mu) = R(\mu)$ as follows. Let $\overline{x}_l(\mu)$ be the cost shares of individual receivers for the path down to but excluding $l$ corresponding to the receiver set $\overline{R}(\mu)$. Let $\overline{n}_l(\mu)$ be the number of receivers below $l$ in this outcome. By induction, we can show that Steps 1 and 2 of the algorithm described in this section maintain the property

$$n_l(\overline{x}_l(\mu), \mu) \geq \overline{n}_l(\mu)$$

Because this is true at the root, it follows that $|R(\mu)| \geq |\overline{R}(\mu)|$. Hence $R(\mu) = \overline{R}(\mu)$. □

The two algorithms (one-pass and iterative) are both budget-balanced, with the same receiver set and the same cost-sharing function; thus they both compute the SH mechanism.

### 3.11.3 A communication-efficient approximation of $n_l(\cdot)$

The algorithm for the Shapley-value mechanism described in the previous section makes only one pass up and down the tree. However, in the worst case, the function $n_l(\cdot)$ passed up link $l$ requires $|P|$ points $(p_i, n_i)$ to represent it, which is undesirable.

Our approach to approximating the SH mechanism is as follows: We replace the function $n_l(\cdot)$ in the one-pass SH mechanism by a small **approximate representation of** $n_l(\cdot)$; only this approximate representation is communicated up the tree, resulting in an exponential saving in the worst-case number of communication bits. What should this approximation look like? To begin with, we would like to **underestimate** $n_l(\cdot)$ at every point, effectively underestimating the players' utilities, so that we can still compute a feasible receiver set in one pass.

For each link $l$, instead of $n_l(p,\mu)$, the mechanism uses an under-approximation $\tilde{n}_l(p,\mu)$. The approximation we choose is simple and is illustrated in Figure 3.6. For some parameter $\kappa > 1$, we round down all values of $n_l(p,\mu)$ to the closest power of $\kappa$. The resulting function $\tilde{n}_l(p,\mu)$ has at most $(\log|P|/\log\kappa)$ "corners," and so it can be represented by a list of $\mathbf{O}(\log|P|)$ points.

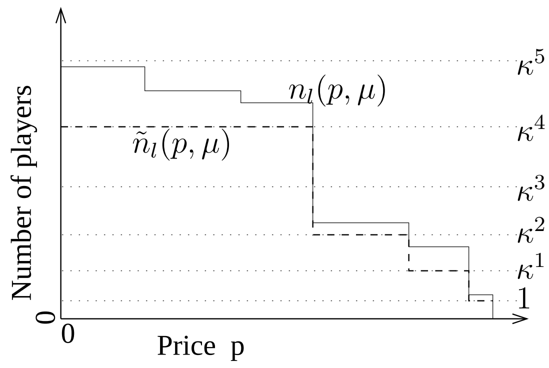


Figure 3.6: Approximation to $n_l(p,\mu)$.

At the leaf nodes, we first compute the exact function $n_l(p,\mu)$ as before, and from this we compute the approximation $\tilde{n}_l(p,\mu)$ as illustrated in Figure 3.6. At non-leaf nodes, we compute $\tilde{n}_l(p,\mu)$ by using the following modified versions of Steps 1 and 2 of the one-pass algorithm:

- **Step 1'**: Compute

$$\hat{m}_l(p,\mu) = \sum_{l_i} \tilde{n}_{l_i}(p,\mu)$$

  (This step is unchanged; we do an exact summation, but the input functions are approximate.)

- **Step 2'**: First, adjust for cost $c(l)$ as before

$$\hat{n}_l(p,\mu) = \max_{\left\{p' - \frac{c(l)}{\hat{m}_l(p',\mu)} \geq p\right\}} \hat{m}_l(p',\mu)$$

Then, approximate the function $\hat{n}_l(\cdot)$ by $\tilde{n}_l(\cdot)$:

$$\tilde{n}_l(p, \mu) = \kappa^{\lfloor \log_\kappa \hat{n}_l(p,\mu) \rfloor}$$

Because $\tilde{n}(\cdot)$ is given in the list-of-points representation, this is easily done by dropping elements of the list that do not change $\tilde{n}_l(p, \mu)$.

The function $\hat{n}_l(\cdot)$ computed on the way up is stored at the node beneath $l$.[7] On the way down, we compute

$$x_{l_j} = x_l + \frac{c(l)}{\tilde{n}_l(x_l, \mu)}$$

Note that Step 2' guarantees that there are at least $\hat{n}_l(x_{l_j}, \mu)$ players beneath $l$ who can afford to pay $p' = x_{l_j}$ for the links from the root through $l$.

We can now define a mechanism (called Mechanism SF, for "step function") by computing $x_i$ for $i \in P$ as in the one-pass algorithm for SH in Section 2.2, including $i$ in the receiver set if $x_i \geq \mu_i$, and assigning cost share $x_i$ to $i$ if $i$ is included. However, we now have a situation in which the number of receivers downstream of link $l$ is potentially greater than $\tilde{n}_l(x_l, \mu)$, because $\tilde{n}_l(\cdot)$ is an under-approximation. Thus, SF does not achieve exact budget balance; there may be a budget surplus.

For example, consider running mechanism SF on the instance shown in Figure 3.7 with $\kappa = 2$. Node $B$ computes $\hat{n}_{l_1}(\cdot)$ as follows: If only player 4 is included, he would have to pay the entire cost of link $l_1$ and hence have only 12 left to pay for link $l_3$; this gives us a corner at point $(12, 1)$. Further computations show that the other corner points are $(11, 2)$, $(10, 3)$, and $(3, 4)$. Figure 3.7 also shows the approximate function $\tilde{n}_{l_1}(\cdot)$: the only difference is that the corner at $(10, 3)$ is dropped. Similarly, node $C$ computes $\tilde{n}_{l_2}(\cdot)$; in this case, there is a single corner at $(7, 2)$.

Now, node $A$ receives the approximate functions $\tilde{n}_{l_1}(\cdot)$ and $\tilde{n}_{l_2}(\cdot)$. It then combines them to compute $\hat{n}_{l_3}(\cdot)$. It turns out that the only way to share the cost of $l_3$, based on the received $\tilde{n}_{l_1}(\cdot)$ and $\tilde{n}_{l_2}(\cdot)$, is to admit two players from each of $A$ and $B$; each of these players is willing to pay at least 7 for links $l_3$ and above, and so can share the cost of link $l_3$

[7]If there are space constraints, it is easy to modify the mechanism to store $\tilde{n}_l(\cdot)$ instead, by rounding $\hat{m}_l$ to a compact approximation $\tilde{m}_l$ and using this function to compute $\tilde{n}_l(\cdot)$ in Step 2.

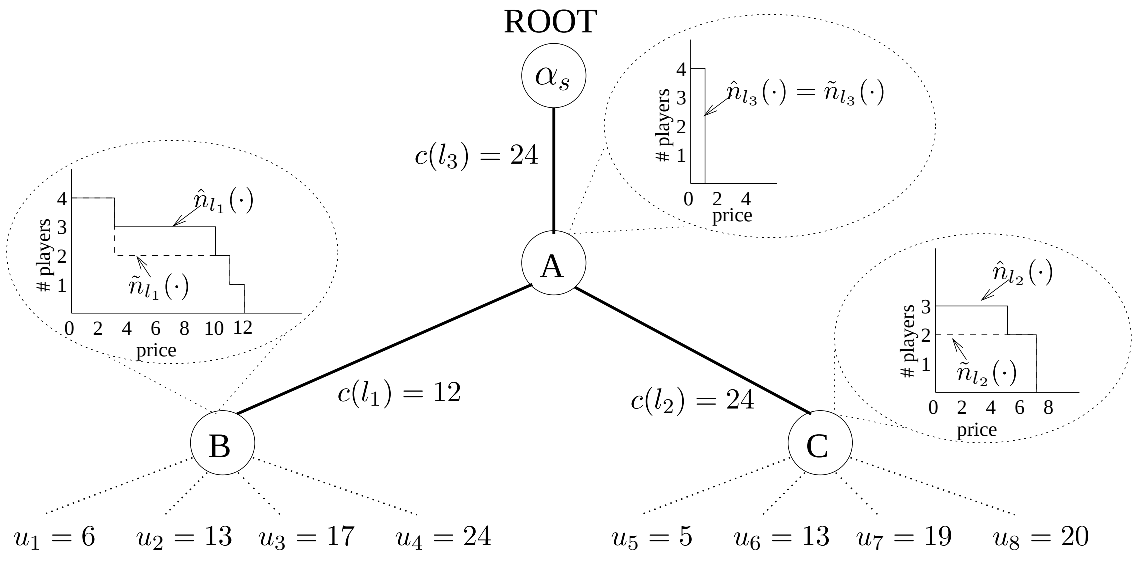


Figure 3.7: Example illustrating budget surplus of Mechanism SF with $\kappa = 2$.

and be willing to pay up to 1 more. Thus, the function $\hat{n}_{l_3}(\cdot)$ has a single corner at $(1, 4)$. Our approximation procedure makes no difference in this case, and so the function $\tilde{n}_{l_3}$ is identical. Finally, the root receives $\tilde{n}_{l_3}$, and as there are no additional costs to share, it computes that transmission is feasible.

On the way down, the payments are computed as follows: At node $A$, the cost of $l_3$ is divided by $\hat{n}_{l_3}(0, \mathbf{u}) = 4$, and thus $x_{l_1} = x_{l_2} = 6$. Node $B$ then adds on the additional cost of $l_1$, and divides it among $\hat{n}_{l_1}(6, \mathbf{u}) = 3$ players. Thus, the ask price for players at node $B$ is $6 + 4 = 10$; players $2, 3$, and 4 are included in the receiver set and pay 10 each. Similarly, the ask price at node $C$ is computed to be $6 + 12 = 18$; players 7 and 8 are included and pay 18 each. The total amount collected is 66, but the cost of transmission is only 60, resulting in a surplus of 6. This surplus arises because node $A$ counted on having only 4 receivers sharing the cost of $l_3$, whereas there were actually 5 receivers.

### 3.11.4 Group strategyproofness of mechanism SF

#### Notation

Throughout this section, we use $\mathbf{u} = (u_1, u_2, \ldots, u_n)$ to indicate the true utility profile of the players. Recall that $\mu|^i r_i$ denotes the utility profile $(\mu_1, \mu_2 \ldots, \mu_{i-1}, r_i, \mu_{i+1}, \ldots, \mu_n)$, *i.e.*, the utility vector $\mu$ perturbed by replacing $\mu_i$ by $r_i$.

Now, let $\mu$ be the reported utility profile. Then $S = \{i \mid u_i \neq \mu_i\}$ is the strategizing group. This strategy is **successful** if no member of $S$ has a lower welfare as a result of the strategy, and at least one member has a higher welfare as a result of the strategy:

$$\forall i \in S \;\; w_i(\mu) \geq w_i(\mathbf{u})$$

$$\exists j \in S \text{ such that } w_j(\mu) > w_j(\mathbf{u})$$

We prove that mechanism SF is GSP in three steps: First, we prove that, if there is a successful (individual or group) strategy, there is a successful strategy $\mu$ in which all colluding players *raise* their utility, *i.e.*, $\mu_i \geq u_i$. This is intuitive, because, if a player receives the transmission, she is not hurt by raising her utility further. Next, we show that a receiver has no strategic value in raising her utility: If $x_i \leq u_i < \mu_i$, then the outcome of the mechanism (both receiver set and cost shares) is unchanged in moving from strategy $\mu$ to $\mu|^i u_i$. Finally, we combine these two results to show that a successful strategy against mechanism SF cannot exist.

For the first part, we formalize our argument that it is sufficient to consider strategies in which all members raise their utilities. The key to this is showing that the following monotonicity property holds:

**Lemma 3.9** *Monotonicity: Let* $\mathbf{u}$ *be a utility profile and* $\mu$ *be the perturbed profile obtained by increasing one element of* $\mathbf{u}$ *(*$\mu = \mathbf{u}|^i\mu_i$*, where* $\mu_i > u_i$*). Then, the following properties hold:*

*(i).* $\forall l, x \;\; \tilde{n}_l(x, \mu) \geq \tilde{n}_l(x, \mathbf{u})$

*(ii).* $\forall j \in P \;\; x_j(\mu) \leq x_j(\mathbf{u})$

*(iii).* $\tilde{R}(\mu) \supseteq \tilde{R}(\mathbf{u})$

*(Here* $x_j(\mu)$ *is the* ask price *computed for player* $j$ *in the downward pass.)*

**Proof:** Note that our approximation technique has the property that, if $\hat{n}_l(x, \mu) \geq \hat{n}_l(x, \mathbf{u})$, then $\tilde{n}_l(x, \mu) \geq \tilde{n}_l(x, \mathbf{u})$. Statement *(i)* is then immediately true at the leaves and follows by induction at non-leaf nodes. Because the cost of any link $l$ is divided among $\hat{n}_l(x_l, \mu)$ players, statement *(ii)* follows from statement *(i)*. Finally, because the utilities are the same (or higher in the case of player $j$), statement *(ii)* implies statement *(iii)*. $\square$

Lemma 3.9 suggests that, for any successful strategy $\mu$, we can get a successful strategy $\mu'$ by raising $\mu_i$ to $u_i$ whenever $\mu_i < u_i$. However, we first have the technical detail of eliminating non-receivers from the strategizing group:

**Lemma 3.10** *Let $\mu$ be a strategy for group $S$. Suppose $i \in S$ and $i \notin \tilde{R}(\mu)$. Let $\mu'$ be the strategy $\mu|^i u_i$. Then, $x_j(\mu') \leq x_j(\mu)$, for all $j \in P$.*

**Proof:** Because $i \notin \tilde{R}(\mu)$, $x_i(\mu) > \mu_i$. When $\mu_i \leq u_i$, the statement follows directly from Lemma 3.9. When $\mu_i > u_i$, we can show that $\tilde{n}_l(x_l(\mu), \mu') = \tilde{n}_l(x_l(\mu), \mu)$ by induction on the height of $l$ (where $l$ is the link from the location of $i$ to its parent), and the statement follows. $\square$

Combining the last two results, we get:

**Lemma 3.11** *Suppose a group $S$ has a successful strategy. Then, $S$ has a successful strategy $\mu'$ where $\mu'_i \geq u_i$.*

**Proof:** By lemma 3.10, we can assume, without loss of generality, that $S$ has a successful strategy $\mu$ such that $S \subseteq R(\mu)$. Define a sequence of strategies

$$\mu = \mu^{(0)}, \mu^{(1)}, \dots \mu^{(n-1)}, \mu^{(n)} = \mu'$$

where $\mu^{(k)} = \mu^{(k-1)}|^k u_k$ if $u_k > \mu_k$, $\mu^{(k)} = \mu^{(k-1)}$ otherwise. The monotonicity property implies that, if $\mu^{(k-1)}$ is a successful strategy, so is $\mu^{(k)}$. $\square$

Now, we prove that, if a receiver $i$ raises his utility, the solution is not altered:

**Lemma 3.12** *Let $\mathbf{u}$ be a utility profile, and let $\mu$ be the perturbed profile obtained by increasing one element of $\mathbf{u}$ ($\mu = \mathbf{u}|^i \mu_i$, where $\mu_i > u_i$). If $u_i \geq x_i(\mathbf{u})$, then*

$$\forall l, \forall x < x_l(\mathbf{u}) \;\; \tilde{n}_l(x, \mu) = \tilde{n}_l(x, \mathbf{u})$$

**Proof:** It is obviously true if $i$ is at a leaf and $l$ is the link from the leaf to its parent, because the utility $\mu_i$ only affects the value of $\tilde{n}_l(\cdot)$ at prices above $u_i \geq x_i$. (This is a result of our *pointwise* approximation scheme; not all approximations would have this property.) Also, because of the monotonically decreasing nature of $\tilde{n}_l(\cdot)$, this property is maintained by Steps 1' and 2' as we move up the tree. □

A corollary of lemma 3.12 is that, when the conditions of the lemma hold, the output of the mechanism is identical for inputs $\mathbf{u}$ and $\mu$. This follows from the fact that $\tilde{n}_l(\cdot)$ is not evaluated at prices above $x_l(\mathbf{u})$ on the way down, and so inductively $x_l(\mu) = x_l(\mathbf{u})$ for all links $l$. Hence, each player gets the same ask price $x_i(\mu) = x_i(\mathbf{u})$.

We can now prove the main result:

**Theorem 3.6** *Mechanism SF is GSP.*

**Proof:** Assume the opposite, *i.e.*, that there is a successful group strategy against mechanism SF. Then, by lemma 3.11, there is a group strategy $\mu$ for some set $S$, where every member of $S$ receives the transmission after the strategy. Define the sequence of strategies:

$$\mu = \mu^{(0)}, \mu^{(1)}, \ldots \mu^{(n-1)}, \mu^{(n)} = \mathbf{u}$$

where $\mu^{(k)} = \mu^{(k-1)}|^k u_k$. It follows from lemma 3.12 that, if $\mu^{(k-1)}$ is a successful strategy for $S$, so is $\mu^{(k)}$. This implies that $\mathbf{u}$ is a successful strategy, which is a contradiction. □

### 3.11.5 Alternative proof that mechanism SF is GSP

We now present an alternative proof of Theorem 3.6 that builds on previous results in the mechanism-design literature Moulin and Shenker [MS01, Mou99] discussed a family of budget balanced mechanisms based on *cross-monotonic cost-sharing* functions and proved them to be group strategyproof. In this section, we develop a new characterization of the family of cross-monotonic cost-sharing mechanisms and show that mechanism SF is a member of this family.

Let $g$ be a cross-monotonic cost-sharing function, and let $M_g = (\sigma(\mu), \mathbf{x}(\mu))$ be the corresponding cross-monotonic mechanism.

**Lemma 3.13** *$M_g$ is group strategyproof.*

This was proved by [Mou99] in the context of budget-balanced mechanisms, but his proof extends directly to all cross-monotonic mechanisms.

Let $\mathcal{M} = \{M_g \mid \text{g is cross-monotonic}\}$ be the set of cross-monotonic mechanisms. We give an alternate characterization of the mechanisms in this set that does not explicitly use the cost-sharing function in the construction of the receiver set.

**Theorem 3.7** *Fix the tree and the costs $c(l)$, and let $U = \mathbb{R}^P_{\geq 0}$ be the space of possible utility profiles. A mechanism $M = (\sigma(\mu), \mathbf{x}(\mu))$ is in $\mathcal{M}$ iff it satisfies the following properties:*

1. *Consumer sovereignty: $\exists B$ such that, for all $i$, for all $\mathbf{u} \in U$ such that $u_i \geq B$, $i \in R(\mathbf{u})$.*
2. *Monotonicity of receiver set: if $\mathbf{u}$,$\mathbf{u}'$ are utility profiles such that, for all $i$, $u_i \leq u'_i$, then $R(\mathbf{u}) \subseteq R(\mathbf{u}')$.*
3. *Let $\mathbf{u}, \mathbf{u}' \in U$ be utility profiles such that $R(\mathbf{u}) = R(\mathbf{u}')$. Then, $x_i(\mathbf{u}) = x_i(\mathbf{u}')$, for all $i$. In other words, the cost shares are a function of the receiver set alone, and we can use the notation $x_i(S)$ to indicate the payment of player $i$ when the receiver set is $S$.*
4. *$x_i(.)$ is cross-monotonic on the space of receiver sets, i.e., if $S \subseteq S'$, then $x_i(S') \leq x_i(S)$, for all $i \in S$.*
5. *For any $S \subseteq P$, let $U(S) = \{\mathbf{u} \in U \mid R(\mathbf{u}) = S\}$. Then, $U(S)$ is closed under the pointwise minimum operation: If $\mathbf{u}, \mathbf{u}' \in U(S)$, and $\mathbf{u}''$ is defined by $u''_i = \min(u_i, u'_i)$, then $\mathbf{u}'' \in U(S)$.*
6. *$x_i(S) = \min_{\mathbf{u} \in U(S)} u_i$*

**Proof:**

*If direction:* Consider a mechanism M $= (\sigma(\mu), \mathbf{x}(\mu))$, and let $R(\mu)$ be the receiver set corresponding to $\sigma(\mu)$. Assume M satisfies properties 1-6.

Properties 5 and 6 say that this mechanism partitions the utility space into "regions" $U(S)$ corresponding to every receiver set $S$. Every region has a unique minimum point $\overline{u}(S)$ defined by

$$\begin{aligned} \overline{u}_i(S) &= x_i(S) \quad \text{if } i \in S \\ \overline{u}_i(S) &= 0 \quad \text{if } i \notin S \end{aligned}$$

Consider the utility profile $\mathbf{u}^S$ given by

$$\begin{aligned} \mathbf{u}_i^S &= B \quad \text{if } i \in S \\ \mathbf{u}_i^S &= 0 \quad \text{if } i \notin S \end{aligned}$$

Then, by property 1, $R(\mathbf{u}^S) \supseteq S$.

Now, consider the cost-sharing function $g$ defined by

$$g_i(S) \overset{def}{=} \overline{u}_i(R(\mathbf{u}^S)) \tag{3.8}$$

For any $S' \supseteq S$, we know by property 2 that $R(\mathbf{u}^{S'}) \supseteq R(\mathbf{u}^S)$. Then, by property 4, it follows that, for all $i \in S$, $g_i(S') \leq g_i(S)$, and hence $g$ is a cross-monotonic cost-sharing function. It only remains to be shown that, for all $\mu$, $R(\mu)$ is the unique largest set $S$ for which

$$\forall i \in S \; g_i(S) \leq \mu_i \tag{3.9}$$

$R(\mu)$ satisfies equation (3.9), because $\mu_i \geq \overline{u}_i(R(\mu)) = x_i(R(\mu))$. Consider any set $T$ that also satisfies this condition. Then, by assumption,

$$\begin{aligned} \forall i \;\; g_i(T) &\leq \mu_i \\ \Longrightarrow \overline{u}_i(R(\mathbf{u}^T)) &\leq \mu_i \\ \Longrightarrow R(\overline{u}(R(\mathbf{u}^T))) &\subseteq R(\mu) \quad \text{by property 2} \end{aligned}$$

We note that $R(\overline{u}(R(\mathbf{u}^T))) = R(\mathbf{u}^T) \supseteq T$, and so it follows that $T \subseteq R(\mu)$. Because this is true for all such $T$, $R(\mu)$ must be the largest set for which equation (3.9) is satisfied.

*Only If direction:* Consider any cross-monotonic mechanism $M_g$. Let $B = \max_i g_i(\{i\})$. Then, it is easy to verify that each of the properties above is satisfied. $\square$

Figure 3.8 illustrates one possible partition for two players. The mechanisms in $\mathcal{M}$ are

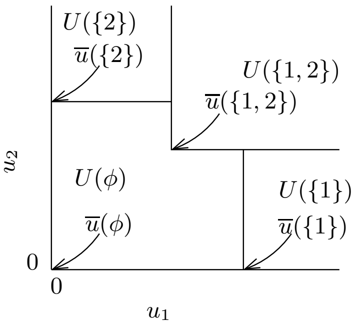


Figure 3.8: Partition of utility space for a two player mechanism

completely characterized by the points $\overline{u}(S)$, over all $S \subseteq P$.

**Theorem 3.8** *Mechanism SF $\in \mathcal{M}$.*

**Proof:** We show that mechanism SF has all the properties listed in Theorem 3.7.

*Property 1:* Let $B$ be the maximum cost of a path from any player to the root. Then, if $u_i \geq B$, $i \in \tilde{R}(\mathbf{u})$.

*Property 2:* The monotonicity of mechanism SF was proved in Lemma 3.9.

*Property 3:* Suppose $R(\mathbf{u}) = R(\mathbf{u}') = S$. Then, using Lemma 3.12 repeatedly, we can show that $x_i(\mathbf{u}) = x_i(\mathbf{u}^S)$, where $\mathbf{u}^S$ is defined as in the proof of theorem 3.7. Similarly, it also follows that $x_i(\mathbf{u}') = x_i(\mathbf{u}^S)$, and so $x_i(\mathbf{u}) = x_i(\mathbf{u}')$. Hence this property is valid, and we can refer to the payment function as $x_i(S)$.

*Property 4:* For receiver sets $S$ and $S'$ such that $S \subseteq S'$, consider the utility profiles $\mathbf{u}^S$ and $\mathbf{u}^{S'}$. The conditions of Lemma 3.9 apply, and so $x_i(S) = x_i(\mathbf{u}^S) < x_i(\mathbf{u}^{S'}) = x_i(S')$.

*Properties 5 and 6:* For any utility profile $\mathbf{u}$, with receiver set $R(\mathbf{u}) = S$, consider the utility profile $\overline{u}$ defined by

$$\begin{aligned} \overline{u}_i &= x_i(S) \quad \text{if } i \in S \\ \overline{u}_i &= 0 \quad \text{if } i \notin S \end{aligned}$$

Note that it is sufficient to show that $R(\overline{u}) = S$ to prove both properties 5 and 6.

We can prove that $R(\overline{u}) = R(\mathbf{u}) = S$ by increasing the elements of $\overline{u}_i$ to $u_i$ one at time and showing that the receiver set remains the same at each step. For $i \notin S$, we can show this by induction on $\tilde{n}_l(.)$, as in Lemma 3.10. For $i \in S$, this follows directly from Lemma 3.12. □

### 3.11.6 Mechanism SSF: bounded budget deficit and welfare loss

While mechanism SF is group strategyproof and has a bounded budget deficit, it has a potentially fatal flaw: it may output an empty receiver set in situations in which the SH mechanism would give a large receiver set. As a result, it may incur a very large welfare loss with respect to the SH mechanism. In this section, we present a simple modification of mechanism SF, called SSF (for "scaled SF"), and prove bounds on its budget deficit and loss of net worth with respect to the SH mechanism. The goal of the modification is to ensure that, for every utility profile, the mechanism has a receiver set at least as large as the SH receiver set. We do this by discounting the cost of each link by a bounded fraction; this converts the budget surplus of mechanism SF to a budget deficit, but improves the worst-case welfare loss.

**Mechanism SSF:**

Let $h_l$ be the height of link $l$ in the tree. (If one of the endpoints of link $l$ is a leaf, then $h_l = 1$.) Then, define the *scaled cost* $c^\kappa(l)$ of the link $l$ to be $c(l)/(\kappa^{h_l})$. Run mechanism SF assuming link costs $c^\kappa(l)$ instead of $c(l)$, to compute a receiver set $R^\kappa(\mathbf{u})$ and cost shares $x_i^\kappa(\mathbf{u})$.

**Lemma 3.14** *Mechanism SSF is GSP.*

**Proof:** The player's utility does not affect the scaled costs, and mechanism SF is GSP for any tree costs. □

Let $R(\mathbf{u})$ be the receiver set in the (exact) Shapley-value mechanism. We now show that $R^\kappa(\mathbf{u}) \supseteq R(\mathbf{u})$.

**Lemma 3.15** *For any link $l$, let $\tilde{n}_l^\kappa(x, \mathbf{u})$ be the approximation computed by mechanism SSF. Let $n_l(x, \mathbf{u})$ and $x_l$ be defined as in the one-pass exact Shapley-value algorithm given in Section 2.2. Then,*

$$\forall l \quad \tilde{n}_l^\kappa(x_l, \mathbf{u}) \geq \frac{n_l(x_l, \mathbf{u})}{\kappa^{h_l}}$$

**Proof:** We prove the statement by induction on $h_l$. For $h_l = 1$, it is true because of our approximation method. Suppose the statement is true for all links of height no more than $\alpha_s$, and $h_l = r + 1$. Let $\{l_1, l_2, \dots, l_k\}$ be the child links of $l$. By the inductive assumption, $\tilde{n}_{l_i}^\kappa(x, \mathbf{u}) \geq (n_{l_i}(x_{l_i}, \mathbf{u}))/\kappa^r$. It follows that

$$\begin{aligned} \hat{m}_l^\kappa(x_{l_i}, \mathbf{u}) &= \sum_{i=1}^{k} \tilde{n}_{l_i}^\kappa(x_{l_i}, \mathbf{u}) && (3.10) \\ &\geq \frac{n_l(x_l, \mathbf{u})}{\kappa^r} && (3.11) \end{aligned}$$

From the computation of the ask prices $x_l$ and $x_{l_i}$ in the exact Shapley value mechanism, we know

$$x_l = x_{l_i} - \frac{c(l)}{n_l(x_l, \mathbf{u})} \ .$$

Let

$$x' = x_{l_i} - \frac{c^\kappa(l)}{\hat{m}_l^\kappa(x_{l_i}, \mathbf{u})} \ . \qquad (3.12)$$

Then, $x' \geq x_l$ follows from Equation (3.11).

Now, in *Step 2'* of mechanism SSF, the function $\hat{m}_l^\kappa(\cdot)$ is adjusted for the scaled cost $c^\kappa(l)$ to compute the function $\hat{n}_l^\kappa(\cdot)$. Equation (3.12) guarantees that $\hat{m}_l^\kappa(x_{l_i}, \mathbf{u})$ players in the subtree below $l$ can share the additional cost $c^\kappa(l)$ and still be willing to pay $x'$ each for links above $l$. Thus, we have

$$\hat{n}_l^\kappa(x', \mathbf{u}) \quad \geq \quad \hat{m}_l^\kappa(x_{l_i}, \mathbf{u}) \ ,$$

and, because $x' \geq x_l$, $\hat{n}_l^\kappa(x_l, \mathbf{u}) \geq \hat{n}_l^\kappa(x', \mathbf{u})$. Finally, in passing from $\hat{n}^\kappa(\cdot)$ to $\tilde{n}^\kappa(\cdot)$, we get

$$\begin{aligned} \tilde{n}_l^\kappa(x_l, \mathbf{u}) &\geq \frac{\hat{n}_l^\kappa(x_l, \mathbf{u})}{\kappa} \\ \tilde{n}_l^\kappa(x_l, \mathbf{u}) &\geq \frac{n_l(x_l, \mathbf{u})}{\kappa^{r+1}} \end{aligned}$$

And thus the statement is proved by induction. □

**Lemma 3.16** $R^\kappa(\mathbf{u}) \supseteq R(\mathbf{u})$.

**Proof:** Using Lemma 3.15,

$$\frac{c^\kappa(l)}{\tilde{n}_l^\kappa(x_l, \mathbf{u})} \leq \frac{c(l)}{n_l(x_l, \mathbf{u})} \ ,$$

and we can show inductively that $x_l^\kappa \leq x_l$ for all links $l$. Because this is true at the leaves, it follows that $R^\kappa(\mathbf{u}) \supseteq R(\mathbf{u})$. □

**Bounding the budget deficit:** Unlike mechanism SF, which is balanced or runs a surplus, mechanism SSF may generate a budget deficit (but never a surplus). However, the deficit (as a fraction of the cost) can be bounded in terms of $\kappa$ and the height $h$ of the tree:

**Theorem 3.9**

$$\frac{c(T(R^\kappa(\mathbf{u})))}{\kappa^h} \leq \sum_{i \in R^\kappa(\mathbf{u})} x_i^\kappa(\mathbf{u}) \leq c(T(R^\kappa(\mathbf{u})))$$

**Proof:** Let $X = \sum_{i \in R^\kappa(\mathbf{u})} x_i^\kappa(\mathbf{u})$. Because mechanism SF never runs a deficit,

$$X \geq c^\kappa(T(R^\kappa(\mathbf{u}))) \geq \frac{c(T(R^\kappa(\mathbf{u})))}{\kappa^h}.$$

We now show that mechanism SSF never runs a budget surplus. For each link $l$, let $x_l$ denote the offer price computed by mechanism SSF. Consider a link $l$, and let $l_1, l_2, \cdots, l_k$ be its child links. Note that the cost of link $l$ is factored into $x_{l_i}$ by assuming that there are $\hat{n}_l^\kappa(x_l, \mathbf{u})$ receivers downstream of $l$. It is sufficient to prove that, for any link $l$, the number of receivers downstream of $l$ (in $R^k$) is at most $\kappa^{h_l} \cdot \hat{n}_l^{\kappa}(x_l, \mathbf{u})$; as the cost of link $l$ has been scaled down by $\kappa^{h_l}$, it follows that we never collect a surplus with respect to the true cost.

We prove this by induction on the height $h_l$ of $l$. When $h_l = 1$, this is clearly true: there are exactly $\hat{n}_l^{\kappa}(x_l, \mathbf{u})$ receivers downstream of $l$. Assume it is true for all links of height at most $r$, and consider a link $l$ of height $r+1$. By the inductive assumption, for each child link $l_i$, we have

$$\hat{n}_{l_i}^{\kappa}(x_{l_i}, \mathbf{u}) \geq \frac{1}{\kappa^r} \times \text{ number of receivers downstream of } l_i \text{ in } R^k \ .$$

Thus, we have

$$\tilde{n}_{l_i}^{\kappa}(x_{l_i}, \mathbf{u}) \geq \frac{1}{\kappa^{r+1}} \times \text{ number of receivers downstream of } l_i \text{ in } R^k \ ,$$

and so

$$\hat{m}_l^{\kappa}(x_{l_i}, \mathbf{u}) \geq \frac{1}{\kappa^{r+1}} \times \text{ number of receivers downstream of } l \text{ in } R^k \ .$$

Finally, the computation of the price $x_{l_i}$ from $x_l$ satisfies $\hat{n}_l^{\kappa}(x_l, \mathbf{u}) = \hat{m}_l^{\kappa}(x_{l_i}, \mathbf{u})$, which gives us

$$\hat{n}_l^{\kappa}(x_{l_i}, \mathbf{u}) \geq \frac{1}{\kappa^{r+1}} \times \text{ number of receivers downstream of } l \text{ in } R^k \ .$$

Thus, by induction this is true for every link $l$. The total payment collected for any link $l$ is at most $\kappa^{h_l} c^{\kappa}(l) \leq c(l)$, and so mechanism SSF never runs a budget surplus. □

**Bounding the worst-case welfare loss:** Let $T^{\kappa}$ and $T$ be the multicast trees corresponding to the receiver sets $R^{\kappa}(\mathbf{u})$ and $R(\mathbf{u})$ respectively. Then, $T^{\kappa}$ can be written as a disjoint union of trees, $T^{\kappa} = T \cup T_1 \cup T_2 \cup \ldots \cup T_r$. The corresponding relation for the receiver set is $R^{\kappa}(\mathbf{u}) = R(\mathbf{u}) \cup R_1 \cup R_2 \cup \ldots \cup R_r$, where $R_i$ is the subset of players in $R^{\kappa}(\mathbf{u})$ who are attached to some node in $T_i$. Some of these subtrees may have negative welfare, and so the overall welfare of the SSF mechanism may be less than the welfare of the Shapley value. However, we can bound the worst-case welfare loss (with respect to the exact Shapley value) in terms of the total utility $U = \sum_{i \in P} u_i$:

**Theorem 3.10**

$$NW(R^\kappa(\mathbf{u})) \geq NW(R(\mathbf{u})) - (\kappa^h - 1)U$$

**Proof:** The welfare of the receiver set $R^\kappa(\mathbf{u})$ is

$$\begin{aligned} NW(R^\kappa(\mathbf{u})) &= \sum_{i \in R^\kappa(\mathbf{u})} u_i - c(T(R^\kappa(\mathbf{u}))) \\ &= NW(R(\mathbf{u})) + \sum_{j=1}^{r} NW(R_j) \end{aligned}$$

Now, for any subtree $T_j$ of $T^\kappa$,

$$U(T_j) = \sum_{i \in T_j} u_i \geq c^\kappa(T_j) \geq \frac{c(T_j)}{\kappa^h} \implies NW(T_j) \geq -(\kappa^h - 1)U(T_j)$$

and hence

$$\begin{aligned} NW(R^\kappa(\mathbf{u})) &\geq NW(R(\mathbf{u})) - (\kappa^h - 1)\sum_{j=1}^{r} U(T_j) \\ &\geq NW(R(\mathbf{u})) - (\kappa^h - 1)\sum_{i \in R^\kappa(\mathbf{u})} u_i \\ &\geq NW(R(\mathbf{u})) - (\kappa^h - 1)U \end{aligned}$$

□

To summarize, Mechanism SSF sends $O(\log_\kappa n)$ points $(p_i, n_i)$ over each link, incurs a cost of at most $\kappa^h$ times the revenue collected, and has an welfare loss of at most $(\kappa^h - 1)U$ with respect to the SH mechanism.

For example, when $|P| = 100,000$ and $h = 5$, the natural algorithm for the SH mechanism given in [FPS01] would require about $100,000$ messages to be sent across a link in the worst case. Our algorithm for SSF requires one bottom-up pass and one top-down pass, *i.e.*, exactly two messages over each link. The maximum size of each point $(p_i, n_i)$ in a message in the bottom-up pass is always bounded by $O(\log |P| + \max_{i \in P} \log u_i)$ bits, and the maximum size of a message sent in the top-down pass is always bounded by $O(\log(\sum_{l \in L} c(l)))$ bits. For $|P| = 100,000$, $h = 5$, and $\kappa = 1.03$, SSF has a budget deficit of at most 14% of the tree cost and a worst-case welfare loss with respect to SH of at most 16% of the total utility, and the largest message sent in the bottom-up pass contains at most 400 points $(p_i, n_i)$. As another example, when $|P| = 10^6$ and $h = 10$, we can use $\kappa = 1.02$ to achieve a worst-case deficit of 18% and worst-case welfare loss of 22% of the total utility, with maximum bottom-up message size of 700 points, or use $\kappa = 1.04$ to achieve corresponding bounds 33%, 48%, and 350 points.

# Chapter 4

# Interdomain Routing[†]

## 4.1 Introduction

The Internet is comprised of many separate administrative domains or *Autonomous Systems* (ASes). Routing occurs on two levels, intradomain and interdomain, implemented by two different sets of protocols. Intradomain-routing protocols, such as OSPF, route packets within a single AS. Interdomain routing, currently handled by the Border Gateway Protocol (BGP), routes packets between ASes. Although routing is a very well-studied problem, it has been approached by computer scientists primarily from an engineering or "protocol-design" perspective. In this chapter, we continue the study of routing from a mechanism-design perspective, concentrating specifically on interdomain routing, for reasons explained below. We study two different formulations of the routing problem. In Section 4.2 we introduce our first formulation, the *lowest-cost routing* problem. In Section 4.3, we provide a formal statement of the problem and in Section 4.4 we derive a strategyproof pricing scheme. In Section 4.5, we describe the BGP-based computational model that we use for the distributed price-calculation algorithm given in Section 4.6. In Section 4.8, we turn to our second formulation, the *policy routing* mechanism-design problem, in which

[†]Sections 4.2 to 4.7 describe joint work with Joan Feigenbaum, Christos Papadimitriou, and Scott Shenker, and was reported in [FPSS02]. Ramesh Govindan provided us with a recent AS graph, and taught us about some of the intricacies of BGP. We also thank Kunal Talwar for helpful discussions on the role of incentives in AS-graph formation, and Gauri Shah for suggesting improvements to the convergence time analysis. Sections 4.8 to 4.10 describe joint work with Joan Feigenbaum, Tim Griffin, Vijay Ramachandran, and Scott Shenker.

source ASes have valuations over alternative routes to a destination. Here, the mechanism design problem appears to be more difficult: In Section 4.9 we show that in the most general case (in which ASes have arbitrary valuations) it is NP-hard to find the routing tree that maximizes overall welfare; it is NP-hard even to find a tree that is approximately optimal, for any reasonable approximation factor. In Section 4.10, we consider an interesting class of restricted valuations, *next-hop* preferences, in which an AS's valuation for a route depends only on which of its neighbors the route passes through. In this case, we find the problem reduces to finding a maximum-weight directed spanning tree, and is thus solvable in polynomial time. We derive a strategyproof mechanism, and show that the payments are also polynomial-time computable, and hence the mechanism appears to be tractable for a centralized computation. However, in Section 4.10.2, we argue that this mechanism is incompatible with BGP, and hence the mechanism appears to be hard in a BGP-based computational model.

## 4.2 Lowest-Cost Routing

In our first formulation of the routing-mechanism design problem, each AS incurs a per-packet *cost* for carrying traffic, where the cost represents the additional load imposed on the internal AS network by this traffic. To compensate for these incurred costs, each AS is paid a *price* for carrying *transit* traffic, which is traffic neither originating from nor destined for that AS. It is through these costs and prices that consideration of "incentive compatibility" is introduced to the interdomain-routing framework, which, as currently implemented, does *not* consider incentives. We are following previous work on mechanism design for routing [NR01, HS01] by introducing incentives in this way. Our goal is to maximize network efficiency by routing packets along the lowest-cost paths (LCPs). Standard routing protocols (such as BGP) can compute LCPs given a set of AS costs. However, under many pricing schemes, an AS could be better off lying about its costs;[1] such lying would cause traffic to take non-optimal routes and thereby interfere with overall network efficiency.

[1]There are two ways in which lying might increase the AS's total welfare: Announcing a lower-than-truthful cost might attract more than enough additional traffic to offset the lower price, or announcing a higher-than-truthful cost might produce an increase in the price that is more than sufficient to offset any resulting decrease in traffic.

To prevent this, we first ask how one can set the prices so that ASes have no incentive to lie about their costs; as we discuss in Chapter 2, such pricing schemes are called "strategyproof." We also require that ASes that carry no transit traffic receive no payment. We prove that there is only one strategyproof pricing scheme with this property; it is a member of the Vickrey-Clarke-Groves (VCG) class of mechanisms [Vic61, Cla71, Gro73]. We next ask how the VCG prices should be computed, and we provide a "BGP-based" distributed algorithm that accomplishes this.

Our results contribute in several ways to the understanding of how incentives and computation affect each other in routing-protocol design. Nisan and Ronen [NR01] and Hershberger and Suri [HS01] considered the LCP mechanism-design problem, motivated in part by the desire to include incentive issues in Internet-route selection. The LCP mechanism studied in [NR01, HS01] takes as input a biconnected graph, a single source, a single destination, and a (claimed) transmission cost for each link; the strategic agents are the links, and the mechanism computes, in a strategyproof manner, both an LCP for this single routing instance and a set of payments to the links on the LCP. This mechanism is a member of the VCG family and forms the point of departure for our work. However, our formulation of the problem differs in three respects, each of which makes the problem more representative of real-world routing:

- First, in our formulation, it is the nodes that are the strategic agents, not the links as in [NR01, HS01]. We make this choice, because we are trying to model *interdomain* routing. ASes actually *are* independent economic actors who could strategize for financial advantage in interdomain-routing decisions; in the BGP computational model into which we seek to incorporate incentive issues, it is the nodes that represent ASes and that are called upon to "advertise" their inputs to the protocol. Formulations in which the links are the strategic agents might be more appropriate for intradomain routing, but it is not clear that incentive issues are relevant in that context; because all links and routers within a domain are owned and managed by a single entity, they are unlikely to display strategic behavior.

- Second, instead of taking as input a single source-destination pair and giving as output a single LCP, our mechanism takes in $n$ AS numbers and constructs LCPs for all source-destination pairs. Once again, we make this choice in order to model more accurately what BGP actually does. This complicates the problem, because there are now $n^2$ LCP instances to solve.

- Third, we compute the routes and the payments not with a centralized algorithm, as is done in [NR01, HS01], but with a distributed protocol based on BGP. This is necessary if the motivation for the mechanism-design problem is Internet routing, because interdomain-route computation is in fact done in a distributed fashion, with the input data (AS-graph topology) and the outputs (interdomain routes) stored in a distributed fashion as well. The various domains are administratively separate and in some cases competitors, and there is no obvious candidate for a centralized, trusted party that could maintain an authoritative AS graph and tell each of the ASes which routes to use. Real-world BGP implementations could be extended easily to include our pricing mechanism, and we prove that such an extension would cause only modest increases in routing-table size and convergence time.

Our approach of using an existing network protocol as a substrate for realistic distributed computations may prove useful generally in Internet-algorithm design, not only in routing or pricing problems. Algorithm design for the Internet has the extra subtlety that adoption is not a decision by a systems manager, concerned only with performance and efficiency, but rather a careful compromise by a web of autonomous entities, each with its own interests and legacies. Backward compatibility with an established protocol is a constraint and criterion that is likely to become increasingly important and prevalent.

Despite these efforts to formulate the problem realistically, there are several aspects of reality that we deliberately ignore. First, per-packet costs are undoubtedly not the best cost model, *e.g.*, in some cases transit costs are more administrative than traffic-induced. Second, BGP allows an AS to choose routes according to any one of a wide variety of local policies; LCP routing is just one example of a valid policy, and, in practice, many ASes do not use it [TGS01]. Furthermore, most ASes do not allow non-customer transit traffic on their network.[2] In the lowest-cost routing formulation, we ignore general policy routing and transit restrictions; we only use LCPs. Lastly, BGP does not currently consider general path costs; in the cases in which AS policy seeks LCPs, the current BGP simply computes *shortest* AS paths in terms of number of AS hops. This last aspect is minor, because it would be trivial to modify BGP so that it computes LCPs; in what follows, we assume that this modification has been made.

Because of these limitations, our results on lowest-cost routing clearly do not constitute a definitive solution to the incentive problem in interdomain routing. Nonetheless, they represent measurable progress on two fronts. First, although it does not capture all of the important features of interdomain routing, our problem formulation is an improvement over the previous ones in the algorithmic mechanism-design literature [NR01, HS01], as explained above. Second, we have expanded the scope of distributed algorithmic mechanism design, which has heretofore been focused mainly on multicast cost sharing [FPS01, AFK$^+$03, FKSS03].

## 4.3 Statement of Lowest-Cost Routing Problem

The network has a set of nodes $N$, $n = |N|$, where each node is an AS. There is a set $L$ of (bidirectional) links between nodes in $N$. We assume that this network, called the *AS graph*, is biconnected; this is not a severe restriction, because the route-selection problem only arises when a node has multiple potential routes to a destination. For any two nodes $i, j \in N$, $T_{ij}$ is the intensity of traffic (number of packets) originating from $i$ destined for $j$.

We assume that a node $k$ incurs a transit cost $c_k$ for each transit packet it carries. In the terminology of Chapter 2, $c_k$ is the private type of agent $k$.

For simplicity, we assume that this cost is independent of which neighbor $k$ received the packet from and which neighbor $k$ sends the packet to, but our approach could be extended to handle a more general case. We write $\mathbf{c}$ for the vector $(c_1, \ldots, c_n)$ of all transit costs and $\mathbf{c}^{-k}$ for the vector $(c_1, \ldots, c_{k-1}, c_{k+1}, \ldots c_n)$ of all costs except $c_k$.

[2] We say that two ASes are "interconnected" if there is a traffic-carrying link between them. Interconnected ASes can be *peers*, or one can be a customer of the other. Most ASes do not accept transit traffic from peers, only from customers.

We also assume that each node $k$ is given a payment $p_k$ to compensate it for carrying transit traffic. In general, this payment can depend on the costs $\mathbf{c}$, the traffic matrix $[T_{ij}]$, and the network topology. Our only assumption, which we invoke in Section 4.4, is that nodes that carry no transit traffic whatsoever receive no payment.

Our goal is to send each packet along the LCP, according to the true cost vector $\mathbf{c}$. We assume the presence of a routing protocol like BGP that, given a set of node costs $\mathbf{c}$, routes packets along LCPs. Furthermore, we assume that, if there are two LCPs between a particular source and destination, the routing protocol has an appropriate way to break ties. Let $I_k(\mathbf{c}; i, j)$ be the indicator function for the LCP from $i$ to $j$; *i.e.*, $I_k(\mathbf{c}; i, j) = 1$, if node $k$ is an intermediate node on the LCP from $i$ to $j$, and $I_k(\mathbf{c}; i, j) = 0$ otherwise. Note that $I_i(\mathbf{c}; i, j) = I_j(\mathbf{c}; i, j) = 0$; only the *transit* node costs are counted. The objective function we want to minimize is the total cost $V(\mathbf{c})$ of routing all packets:

$$V(\mathbf{c}) = \sum_{i,j \in N} T_{ij} \sum_{k \in N} I_k(\mathbf{c}; i, j) c_k$$

Minimizing $V$ is equivalent to minimizing, for every $i, j \in N$, the cost of the path between $i$ and $j$.

We treat the routing problem as a mechanism-design problem in which the ASes are the strategic agents. Each node plays the game by reporting a transit cost. A node's transit cost is private information not known to any other node, and thus no other agent can assess the correctness of an agent's claimed transit cost. Moreover, $V(\cdot)$ is defined in terms of the true costs, whereas the routing algorithm operates on the declared costs; the only way we can be assured of minimizing $V(\cdot)$ is for agents to input their true costs. Therefore, we must rely on the pricing scheme to incentivize agents to do so.

To do so, we design an algorithmic mechanism as described in Chapter 2. The mechanism takes as input the AS graph and the vector $\mathbf{c}$ of declared costs[3] and produces as output the set of LCPs and prices.[4] The pricing mechanism must be strategyproof so that

[3]We will often use $\mathbf{c}$ to denote the declared costs and the true costs; usually, the context will make clear which we mean.

[4]BGP can take the AS graph and $\mathbf{c}$ as input and produce the set of LCPs. We use this output of BGP in our mechanism and do not alter this aspect of BGP in our algorithm.

agents have no incentive to lie about their costs.

The *utility* each AS derives from the set of routes chosen by the mechanism is the negative of the total cost it incurs in transiting traffic along those routes, *i.e.*, "cost" is a synonym for "disutility" here. For a given cost vector $\mathbf{c}$, the payment $p_k$ minus the total costs incurred by a node $k$ is $w_k(\mathbf{c}) = p_k - \sum_{i,j} T_{i,j} I_k(\mathbf{c}; i,j) c_k$. In the terminology of Chapter 2, $w_k(\cdot)$ is the *welfare* of agent $k$. In this context, the mechanism is strategyproof if for all $x$, $w_k(\mathbf{c}) \geq w_k(\mathbf{c}|^k x)$, where the expression $\mathbf{c}|^k x$ means that $(\mathbf{c}|^k x)_i = c_i$, for all $i \neq k$, and $(\mathbf{c}|^k x)_k = x$.

## 4.4 The Pricing Mechanism

Recall that we assume we have a biconnected graph with a routing algorithm that, when given a vector of declared costs $\mathbf{c}$, will produce a set of LCPs, breaking ties in an appropriate manner; these paths are represented by the indicator functions $\{I_k(\mathbf{c}; i,j)\}_{k \in N}$. Furthermore, both the inputs and the outputs are distributed, *i.e.*, neither ever resides at a single node in the network. In this section, we derive the pricing scheme, and, in Sections 4.5 and 4.6, we describe the distributed computation.

We require that the pricing mechanism be strategyproof and that nodes that carry no transit traffic receive no payment. We now show that these two conditions uniquely determine the mechanism we must use. Moreover, we show that they require that the payments take the form of a per-packet price that depends on the source and destination; that is, the payments $p_k$ must be expressible as

$$p_k = \sum_{i,j \in N} T_{ij} p_k^{ij},$$

where $p_k^{ij}$ is the per-packet price paid to node $k$ for each transit packet it carries that is sent from node $i$ destined for node $j$.

**Theorem 4.1** *When routing picks lowest-cost paths, and the network is biconnected, there is a unique strategyproof pricing mechanism that gives no payment to nodes that carry no* *transit traffic. The payments to transit nodes are of the form* $p_k = \sum_{i,j \in N} T_{ij} p_k^{ij}$*, where*

$$p_k^{ij} = c_k I_k(\mathbf{c}; i, j) + \left[ \sum_{r \in N} I_r(\mathbf{c}|^k \infty; i, j) c_r - \sum_{r \in N} I_r(\mathbf{c}; i, j) c_r \right].$$

**Proof:** Consider a vector of costs $\mathbf{c}$. Let $u_k(\mathbf{c})$ denote the total utility derived by a node for this cost vector:

$$u_k(\mathbf{c}) = -c_k \sum_{i,j \in N} T_{ij} I_k(\mathbf{c}; i, j).$$

We can rewrite our objective function as

$$V(\mathbf{c}) = \sum_{i,j \in N} T_{ij} \sum_{k \in N} I_k(\mathbf{c}; i, j) c_k = - \sum_{k \in N} u_k(\mathbf{c}).$$

Thus, the objective function $V(\cdot)$ is simply the negative of the overall welfare (efficiency) defined in Chapter 2. Note that the routing function $\{I_k(\mathbf{c}; i, j)\}_{k \in N}$ minimizes this quantity, *i.e.*, the mechanism is efficient. A classic result due to Green and Laffont [GL79] states that any strategyproof, efficient pricing mechanism must be a VCG mechanism, with payments expressible as

$$p_k = -u_k(\mathbf{c}) - V(\mathbf{c}) + h_k(\mathbf{c}^{-k}),$$

where $h_k(\cdot)$ is an arbitrary function of $\mathbf{c}^{-k}$. When $c_k = \infty$, we have $I_k(\mathbf{c}|^k \infty; i, j) = 0$, for all $i, j$ (because the graph is biconnected, and all other costs are finite); so (1) $p_k = 0$, because we require that payments be 0, and (2) $u_k(\mathbf{c}) = 0$. Thus,

$$h_k(\mathbf{c}^{-k}) = V(\mathbf{c}|^k \infty).$$

This, in turn, implies that

$$\begin{aligned}
p_k &= V(\mathbf{c}|^k\infty) + u_k(\mathbf{c}) - V(\mathbf{c}) \\
&= \sum_{i,j\in N} T_{ij}\Big\{c_k I_k(\mathbf{c};i,j) + \sum_{r\in N} I_r(\mathbf{c}|^k\infty;i,j)c_r - \sum_{r\in N} I_r(\mathbf{c};i,j)c_r\Big\} \\
&= \sum_{i,j\in N} T_{ij}p_k^{ij},
\end{aligned}$$

where

$$p_k^{ij} = c_k I_k(\mathbf{c};i,j) + \left[\sum_{r\in N} I_r(\mathbf{c}|^k\infty;i,j)c_r - \sum_{r\in N} I_r(\mathbf{c};i,j)c_r\right].$$

□

This mechanism belongs to the Vickrey-Clarke-Groves (VCG) family [Vic61, Cla71, Gro73]. It is in essence a node-centric, all-pairs extension of the LCP mechanism studied by Nisan and Ronen [NR01] and Hershberger and Suri [HS01]. There are several aspects of this result that are worth noting. First, although the payments could have taken any form and could have depended arbitrarily on the traffic matrix, it turns out the payments are a sum of per-packet payments that do not depend on the traffic matrix. Second, the price $p_k^{ij}$ is zero if the LCP between $i$ and $j$ does not traverse $k$. Thus, these payments can be computed, once one knows the prices, merely by counting the packets as they enter the node. Third, although the costs did not depend on the source and destination of the packet, the prices do. Lastly, the payment to a node $k$ for a packet from $i$ to $j$ is determined by the cost of the LCP and the cost of the lowest-cost path that does not path through $k$. We use the term *$k$-avoiding path* to refer to a path that does not pass through node $k$.

For example, consider the AS graph in Figure 4.1, and suppose the traffic consists of a single packet from $X$ to $Z$. The LCP is $XBDZ$, which has transit cost 3. How much should AS $D$ be paid? The lowest-cost $D$-avoiding path from $X$ to $Z$ is $XAZ$, which has transit cost 5. Hence, Theorem 4.1 says that $D$ should be paid $c_D + [5-3] = 3$. Similarly, AS $B$ is paid $c_B + [5-3] = 4$. Note that the total payments to nodes on the path is greater than the actual cost of the path. A more extreme example of *overcharging* occurs in sending a packet from $Y$ to $Z$. The LCP is $YDZ$, which has transit cost 1. However, the next best

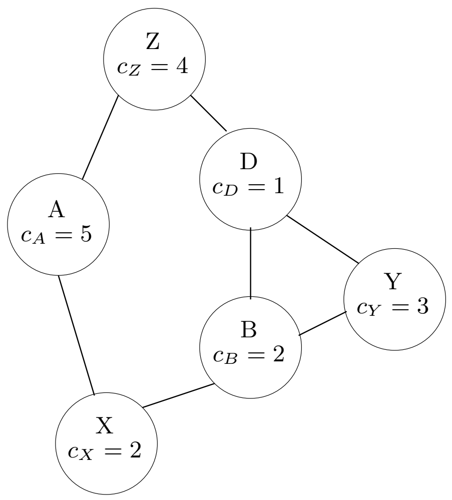


Figure 4.1: Example AS graph from Section 4.4

path is $YBXAZ$ which has cost 9, and hence $D$'s payment for this packet is $1 + [9 - 1] = 9$, even though $D$'s cost is still 1. We return to this issue of overcharging in Section 4.7. These examples also show why the network must be biconnected; if it weren't, the payment would be undefined.

## 4.5 BGP-based Computational Model

We now seek to compute these prices $p_k^{ij}$, using the current BGP algorithm, which is the repository of interdomain routing information, as the computational substrate. We adopt the abstract model of the BGP protocol described by Griffin and Wilfong [GW99], which involves several simplifying assumptions. Specifically, we assume that there is at most one link between any two ASes, that the links are bidirectional, and that each AS can be treated as an atomic entity without regard to intradomain routing issues. The network can then be modeled as a graph in which every node represents an AS, and every edge represents a bidirectional interconnection between the corresponding ASes.

BGP is a *path-vector* protocol in which every node $i$ stores, for each AS $j$, the lowest-cost *AS Path* (the sequence of ASes traversed) from $i$ to $j$; in this vector, ASes are identified by their AS numbers. In addition, in our treatment, the LCP is also described by its total cost (the sum of the declared AS costs). If $d$ is the diameter of the network (the maximum number of ASes in an LCP), a router stores $\mathbf{O}(nd)$ AS numbers and $\mathbf{O}(n)$ path costs. BGP's route computation is similar to all path-vector routing protocols. Each router sends its routing table and, in our treatment, its declared cost, to its neighbors, and each node can then, based on this information, compute its own LCPs. When there is more than one LCP, our model of BGP selects one of them in a loop-free manner (to be defined more precisely below). As mentioned earlier, we are making the oversimplifying assumption that every node is using lowest cost as its routing policy.

These routing-table exchanges only occur when a change is detected; that is, a router only sends its routing table to its neighbors when that table is different from what was sent previously. Routing tables can change either because a link was inserted or deleted (which would be detected by the nodes on either end) or when updated routing-table information is received from some other router that changes the paths and/or costs in the current table.[5]

The computation of a single router can be viewed as consisting of an infinite sequence of *stages*, where each stage consists of receiving routing tables from its neighbors, followed by local computation, followed (perhaps) by sending its own routing table to its neighbors (if its own routing table changed). The communication frequency is limited by the need to keep network traffic low, and hence the local computation is unlikely to be a bottleneck. Thus, we adopt as our measures of complexity the number of stages required for convergence and the total communication (in terms of the number of routing tables exchanged and the size of those tables).

If we assume that all the nodes run synchronously (exchange routing tables at the same time), BGP converges, *i.e.*, computes all LCPs, within $d$ stages of computation (where, again, $d$ is the maximum number of AS hops in an LCP). Each stage involves $\mathbf{O}(nd)$ communication on any link.[6] The computation time required by node $i$ in a single stage is $\mathbf{O}(nd \times \text{degree}(i))$.

Because this level of complexity is already deemed feasible in the current Internet, we

[5]In practice, BGP only sends the portion of the routing table that has changed. Nodes keep the routing tables received from each of their neighbors so that they can reconstruct the new routing table from the incremental update. Because the worst-case behavior is to send the entire routing table, and we care about worst-case complexity, we ignore this incremental aspect of BGP in the statements of our bounds.

[6]Because of the incremental nature of updates, where nodes need only process and forward routing entries that have changed, the communication and computational load is likely to be much lower in practice.

seek to compute the prices with a similar (or better) complexity and state requirements. We describe such an algorithm in the next section.

## 4.6 Distributed Price Computation

We want to compute the $p_k^{ij}$ using the BGP computational model described in Section 4.5. The input to the calculation is the cost vector $\mathbf{c}$, with each $c_i$ known only to node $i$. The output is the set of prices, with node $i$ knowing all the $p_k^{ij}$ values.[7] In describing our algorithm we assume a static environment (no route changes). The effect of removing this assumption is that the process of "converging" begins again each time a route is changed.

Our algorithm introduces additional state to the nodes and to the message exchanges between nodes, but it does not introduce any new messages to the protocol. In particular, all messages are between neighbors in the AS graph. The added state at each node consists of the reported cost of each transit node and the set of prices. This is $\mathbf{O}(nd)$ additional state, resulting in a small constant-factor increase in the state requirements of BGP. The costs and prices will be included in the routing message exchanges, and so there will be a corresponding constant-factor increase in the communication requirements of BGP.

We first investigate how the prices $p_k^{ij}$ at node $i$ are related to the prices at $i$'s neighbors.

Let $P(\mathbf{c}; i, j)$ denote the LCP from $i$ to $j$ for the vector of declared costs $\mathbf{c}$, and let $c(i, j)$ denote the cost of this path. Define $P^{-k}(\mathbf{c}; i, j)$ to be the lowest-cost $k$-avoiding path from $i$ to $j$. Recall that, if there are multiple LCPs between two nodes, the routing mechanism selects one of them in a loop-free manner. *Loop-free* means that the routes are chosen so that the overall set of LCPs from every other node to $j$ forms a tree. In other words, for each destination $j$, we assume that the LCPs selected form a tree rooted at $j$; call this tree $T(j)$. For example, the tree $T(Z)$ corresponding to the graph in Figure 4.1 is shown in Figure 4.2. We say that $D$ is the *parent* of $B$ in $T(Z)$ or, equivalently, that $B$ is a *child* of $D$ in $T(Z)$.

We treat each destination $j$ separately. Consider the computation of $p_k^{ij}$ at some node $i$ for another node $k$ on the path from $i$ to $j$. Let $a$ be a neighbor of $i$. There are four cases:

[7]More precisely, these are the parts of the input and output that we introduce; BGP, with its standard distributed input (AS graph and costs) and distributed output (LCPs) is used as a substrate.

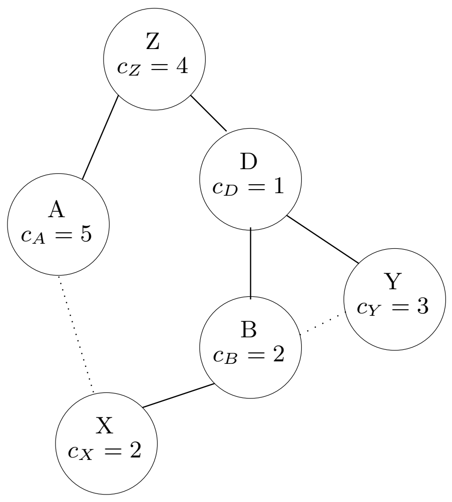


Figure 4.2: Tree T(Z) for the example in Figure 4.1

- **Case (i)**: $a$ is $i$'s parent in $T(j)$

  In this case, provided that $a$ is not $k$, we can extend any $k$-avoiding path from $a$ to $j$ to a $k$-avoiding path from $i$ to $j$, and so the following inequality holds:

$$p_k^{ij} \le p_k^{aj} \tag{4.1}$$

- **Case (ii)**: $a$ is $i$'s child in $T(j)$

  Here, note that $k$ must be on the LCP from $a$ to $j$. Further, given any $k$-avoiding path from $a$ to $j$, we can add or remove the link $ia$ to get a $k$-avoiding path from $i$ to $j$, and so we have:

$$p_k^{ij} \le p_k^{aj} + c_i + c_a \tag{4.2}$$

- **Case (iii)**: $a$ is not adjacent to $i$ in $T(j)$, and $k$ is on $P(\mathbf{c}; a, j)$.

$$p_k^{ij} \le p_k^{aj} + c_a + c(a, j) - c(i, j) \tag{4.3}$$

  Consider $P^{-k}(\mathbf{c}; a, j)$, the lowest-cost $k$-avoiding path from $a$ to $j$. We can always add the edge $ia$ to this path to get a $k$-avoiding path from $i$ to $j$. The inequality is then apparent by substituting the costs of the paths.

- **Case (iv)**: $a$ is not adjacent to $i$ in $T(j)$, and $k$ is not on $P(\mathbf{c}; a, j)$. In this case, we can add the edge $ia$ to $P(\mathbf{c}; a, j)$ to construct a $k$-avoiding path from $i$ to $j$. It is easy to see that
$$p_k^{ij} \leq c_k + c_a + c(a, j) - c(i, j) \tag{4.4}$$

Note that these four cases are not exhaustive. In particular, the case in which $a = k$ is the parent of $i$ are excluded. In this case, the link $ia$ will not be used in $P^{-k}(\mathbf{c}; i, j)$; thus, we can ignore neighbors in this category.

Let $b$ be the neighbor of $i$ on $P^{-k}(\mathbf{c}; i, j)$; *i.e.*, the link $ib$ is the first link on this path. We claim that, for this neighbor, the upper bounds in the previous inequalities are tight:

**Lemma 4.1** *Let $ib$ be the first link on $P^{-k}(\mathbf{c}; i, j)$. Then, the corresponding inequality (4.1)-(4.4) attains equality for $b$.*

**Proof:** We can consider each of the four cases separately.

- **Case (i)**: Given that $P^{-k}(\mathbf{c}; i, j)$ goes through its parent, it follows that $b$ is not $k$, and so $p_k^{ij} = p_k^{bj}$.
- **Case(ii)**: If $P^{-k}(\mathbf{c}; i, j)$ passes through a child $b$, it is easy to see that $p_k^{ij} = p_k^{bj} + c_i + c_b$.
- **Case(iii)**: In this case, if $P^{-k}(\mathbf{c}; i, j)$ passes through $b$, it must contain $P^{-k}(\mathbf{c}; b, j)$, and so Inequality 4.3 is an exact equality.
- **Case(iv)**: In this case, the lowest-cost $k$-avoiding path through $b$ must contain $P(\mathbf{c}; b, j)$, and so Inequality 4.4 is exact.

□

Inequalities (4.1)-(4.4) and Lemma 4.1 together mean that $p_k^{ij}$ is exactly equal to the minimum, over all neighbors $a$ of $i$, of the right-hand side of the corresponding inequality.

Thus, we have the following distributed algorithm to compute the payment values:

### 4.6.1 The Algorithm

Consider each destination $j$ separately. The BGP table at $i$ contains the LCP to $j$:

$$P(\mathbf{c}; i, j) \equiv v_s, v_{s-1}, \cdots, v_0 = j,$$

and the cost of this path, $c(i,j)$, where $v_s, v_{s-1}, \cdots, v_0$ are the nodes on the LCP to $j$ and $c(i,j) = \sum_{r=1}^{s} c_{v_r}$.

Note that each node can infer from the routing tables it receives from its neighbors whether $a$ is its parent, child, or neither in the tree $T(j)$, for each neighbor $a$.

At the beginning of the computation, all the entries of $p^{ij}_{v_r}$ are set to $\infty$. Whenever any entry of this price array changes, the array and the path $P(\mathbf{c}; i, j)$ are sent to all neighbors of $i$. As long as the network is static, the entries decrease monotonically as the computation progresses. If the network is dynamic, price computation (and, as explained above, convergence) must start over whenever there is a route change.

When node $i$ receives an updated price from a neighbor $a$, it performs the following updates to its internal state.

- If $a$ is $i$'s parent in $T(j)$, then $i$ scans the incoming array and updates its own values if necessary:
$$p^{ij}_{v_r} = \min(p^{ij}_{v_r}, p^{aj}_{v_r}) \quad \forall r \leq s-1$$
- If $a$ is a child of $i$ in $T(j)$, $i$ updates its payment values using
$$p^{ij}_{v_r} = \min(p^{ij}_{v_r}, p^{aj}_{v_r} + c_i + c_a) \quad \forall r \leq s$$
- If $a$ is neither a parent nor a child, $i$ first scans $a$'s updated path to find the nearest common ancestor $v_t$. Then $i$ performs the following updates:
$$\begin{aligned} \forall r \leq t \;\; p^{ij}_{v_r} &= \min(p^{ij}_{v_r}, p^{aj}_{v_r} + c_a + c(a,j) - c(i,j)) \\ \forall r > t \;\; p^{ij}_{v_r} &= \min(p^{ij}_{v_r}, c_k + c_a + c(a,j) - c(i,j)) \end{aligned}$$

The algorithm is summarized in Figure 4.3.

## 4.6.2 Correctness of the algorithm

Inequalities (4.1)-(4.4) can be used to show that the algorithm never computes a value $p^{ij}_k$ that is too low. In order to show that the $p^{ij}_k$ values will ultimately converge to their true

```
Initialize ()
{
   /* Compute routes, initialize payments */
   for each destination $j$
      Compute $P(\mathbf{c};i,j)$ and $c(i,j)$
      $[v_s, v_{s-1}, \cdots, v_1] = P(\mathbf{c};i,j)$
      for each node $k$ on $P(\mathbf{c};i,j)$
         $p_k^{ij} := \infty$
}

Update $\left(a,\ j,\ c(a,j),\ P(\mathbf{c};a,j),\ [p_{u_1}^{aj}, p_{u_2}^{aj}, \cdots, p_{u_l}^{aj}]\right)$
{
   /* Called when an UPDATE message                                      */
   /* for destination $j$ is received                                    */
   /* from neighbor $a$.                                                 */
   /* $u_1, u_2, \cdots u_l$ are the transit nodes                       */
   /* on the route $P(\mathbf{c};a,j)$ from $a$ to $j$                   */
   modified := FALSE

   if $a$ is on $P(\mathbf{c};i,j)$                                   /* parent */
      /* $u_r = v_r$, for $r = 1, 2, \cdots l$ */
      for each $k$ in $\{v_1, v_2, \cdots v_l\}$
         if $p_k^{ij}$ >   $p_k^{aj}$
            $p_k^{ij}$ :=  $p_k^{aj}$
            modified := TRUE

   else if $i$ on $P(\mathbf{c};a,j)$                                  /* child */
      /* $u_r = v_r$, for $r = 1, 2, \cdots (l-1)$ */
      for each $k$ in $\{v_1, v_2, \cdots v_{l-1}\}$
         if $p_k^{ij}$ >   $p_k^{aj} + c_a + c_i$
            $p_k^{ij}$ :=  $p_k^{aj} + c_a + c_i$
            modified := TRUE

   else                                                /*neither parent nor child*/
      $t :=$ largest index such that $u_t = v_t$
      for each $k$ in $\{v_1, v_2, \cdots v_t\}$
         if $p_k^{ij}$ >   $p_k^{aj} + c_a + c(a,j) - c(i,j)$
            $p_k^{ij}$ :=  $p_k^{aj} + c_a + c(a,j) - c(i,j)$
            modified := TRUE
      for each $k$ in $\{v_{t+1} \cdots v_s\}$
         if $p_k^{ij}$ >   $c_k + c_a + c(a,j) - c(i,j)$
            $p_k^{ij}$ :=  $c_k + c_a + c(a,j) - c(i,j)$
            modified := TRUE

   if modified = TRUE
      /* Send UPDATE message to neighbors*/
      for each neighbor $b$ of $i$
         send UPDATE $\left(i,\ j,\ c(i,j),\ P(\mathbf{c};i,j),\right.$
                        $\left.[p_{v_1}^{ij}, p_{v_2}^{ij}, \cdots, p_{v_s}^{ij}]\right)$ to $b$
}
```

Figure 4.3: Price-computation algorithm run by AS $i$

values, we observe that, for every node $s$ on $P^{-k}(\mathbf{c};i,j)$, the suffix of $P^{-k}(\mathbf{c};i,j)$ from $s$ to $j$ is either $P(\mathbf{c};s,j)$ or $P^{-k}(\mathbf{c};s,j)$. It follows that, in general, the path $P^{-k}(\mathbf{c};i,j)$ consists of a sequence of nodes $[v_l, v_{l-1}, \cdot\cdot, v_1, u_m, u_{m-1}, \cdot\cdot, u_1]$ such that, for each $u_x$, $P(\mathbf{c};u_x,j)$ is the suffix $[u_x, u_{x-1}, \cdot\cdot, u_1]$, and, for each $v_y$, $P^{-k}(\mathbf{c};v_y,j)$ is the suffix $[v_y, v_{y-1}, \cdot\cdot, v_1, u_m, u_{m-1}, \cdot\cdot, u_1]$. Note that, once the LCPs are computed, $u_m$ will know the correct $P(\mathbf{c};u_m,j)$ and cost $c(a,j)$. This information will be sent to $v_1$ in the next update message from $u_m$ to $v_1$; thus, $v_1$ will then be able to compute the correct $P^{-k}(\mathbf{c};v_1,j)$ and $p_k^{v_1 j}$. Proceeding by induction on $y$, we can show that $i$ will ultimately have all the correct $p_k^{ij}$ values.

In fact, the preceding inductive argument shows that all prices will be stable after $d'$ stages, where $d'$ is the maximum over all $i, j, k$, of the number of nodes on $P^{-k}(\mathbf{c};i,j)$. In general, $d'$ can be much higher than the lowest-cost diameter $d$ of a graph. However, we don't find that to be the case for the current AS graph, as we explain in Section 4.7.

### 4.6.3 Convergence time

Up to this point, we have assumed for simplicity that the prices computation begins only after the LCPs have been found. In reality, however, the algorithm can start to compute prices even before the routes have stabilized. This leads to the following bound:

**Lemma 4.2** *Let $d_i = \max\{|P(\mathbf{c};i,j)|, |P^{-k}(\mathbf{c};i,j)|\}$, where $|P|$ denotes the number of hops in path $P$. Then, after the first $d_i$ stages, $i$ knows the correct path $P(\mathbf{c};i,j)$, and the correct price $p_k^{ij}$.*

**Proof:** The intuition behind this proof is as follows: The critical information that $i$ needs to compute the correct price $p_k^{ij}$ is the cost of $P(\mathbf{c};i,j)$ and the cost of $P^{-k}(\mathbf{c};i,j)$. (The cost $c_k$ can be distributed with LCPs to $k$, and so it will be known to $i$ before or at the same time as the cost of $P(\mathbf{c};i,j)$.) After these costs have been discovered, the price $p_k^{ij}$ will not change. The key observation is that, for both $P(\mathbf{c};i,j)$ and $P^{-k}(\mathbf{c};i,j)$, all suffixes of the path are also LCPs or minimum-cost $k$-avoiding paths; moreover, this is true even at intermediate stages of the computation. Using this, we can show that the costs along these paths will be propagated further in each stage and hence will reach $i$ in $d_i$ stages.

We now formalize this argument into an inductive proof. First, observe that both the LCP costs and the prices never increase as the computation proceeds. Now, if there is an $r$-hop path from $i$ to $j$ with cost $\alpha$, then, after $r$ stages, we must have $c(i,j) <= \alpha$. It follows that $i$ discovers the LCP in $|P(\mathbf{c};i,j)| \leq d_i$ stages from the beginning.

Now, suppose $|P^{-k}(\mathbf{c};i,j)| = r$. We can write $P^{-k}(\mathbf{c};i,j)$ as $v_r, v_{r-1}, \cdots, v_1, j$, where $v_r = i$. Let $c'(v_m,j)$ denote the total cost *along this path* from $v_m$ to $j$. We show by induction on $m$ that, after $m$ stages, node $v_m$ satisfies one of the following two conditions:

1. $c(v_m,j) = c'(v_m,j)$, and the current LCP from $v_m$ to $j$ does not pass through $k$.

2. The current LCP from $v_m$ to $j$ passes through $k$, and the current $p_k^{v_m j} = c'(v_m,j) - c(v_m,j)$.

The base case for $v_1$ is easy: After 1 stage of computation, condition 1 is clearly satisfied. Suppose it is true for all $r \leq (m-1)$. After $(m-1)$ stages, $v_{m-1}$ satisfies one of the two conditions; we consider the two cases separately:

- **Case (i)**: $v_{m-1}$ satisfies condition 1

  In this case, in the $m$th stage $v_m$ will receive an advertised path from $v_{m-1}$ that has cost $c'(v_m,j)$ and does not pass through $k$. If this is the lowest-cost path to $j$ that $v_m$ has seen, then $v_m$ will satisfy condition 1 at the end of this stage. If not, then the current LCP from $v_m$ to $j$ must pass through $k$, or else $P^{-k}(\mathbf{c};i,j)$ would not pass through $v_{m-1}$. In this scenario, the path advertised by $v_{m-1}$ must be the lowest-cost $k$-avoiding path from $v_m$ to $j$; hence after this stage condition 2 will be satisfied.

- **Case (ii)**: $v_{m-1}$ satisfies condition 2

  In this case, $v_{m-1}$ advertises a path of cost less than $c'(v_m,j)$ to $v_m$. Thus, the LCP from $v_m$ to $j$ after the $m$th stage must pass through $k$, or else it would be part of the lowest-cost $k$-avoiding path from $i$ to $j$. Now, the advertisement from $v_{m-1}$ (which includes the price $p_k^{v_{m-1} j}$) allows $v_m$ to infer that there is a candidate $k$-avoiding path of cost $c'(v_m,j)$; this is the lowest-cost $k$-avoiding path, and so condition 2 will be satisfied.

Extending this inductive argument to $v_r = i$, we see that, after $r$ stages, $i$ will know of the existence of a $k$-avoiding path of cost $c'(v_r, j)$. Thus, after $d_i$ stages, $i$ will know the correct LCP to $j$ and its cost, as well as the cost of the lowest-cost $k$-avoiding path to $j$, and so it will compute the correct price $p_k^{ij}$. □

**Corollary 4.1** *After* $\max(d, d')$ *stages, every node has the correct LCPs and prices.*

□

### 4.6.4 Using the Prices

At the end of the above price computation, each node $i$ has a full set of prices $p_k^{ij}$. The next question is how we can use these prices actually to compute the revenue due each node.

The simplest approach is to have each node $i$ keep running tallies of owed charges; that is, every time a packet is sent from source $i$ to a destination $j$, the counter for each node $k \neq i, j$ that lies on the LCP is incremented by $p_k^{ij}$. This would require $\mathbf{O}(n)$ additional storage at each node. At various intervals, nodes can send these quantities in to whatever accounting and charging mechanisms are used to enforce the pricing scheme. We assume that the submission of these running totals is done infrequently enough that the communication overhead can be easily absorbed.

In summary, we have:

**Theorem 4.2** *Our algorithm computes the VCG prices correctly, uses routing tables of size $O(nd)$ (i.e., imposes only a constant-factor penalty on the BGP routing-table size), and converges in at most* $\max(d, d')$ *stages.*

## 4.7 Analysis of Overcharging

We now turn to the issue of *overcharging*. VCG mechanisms have been criticized in the literature because there are graphs in which the total price along a path, *i.e.*, the sum of the per-packet payments along the path, is much more than the true cost of the path. Examples of this phenomenon were given in Section 4.4. In the worst case, this total path price can be arbitrarily higher than the total path cost [AT02]. Although this is undesirable, it may be unavoidable, because VCG mechanisms are the only strategyproof pricing mechanisms for protocols that always route along LCPs. In addition, our distributed algorithm has a convergence time (measured in number of stages) of $d'$, whereas BGP's convergence time is $d$; in the worst case, $\frac{d'}{d}$ could be $\Omega(n)$. These are serious problems that could undermine the viability of the pricing scheme we present here. Thus, we ask whether these problems occur in practice.

To provide a partial answer to this question, we looked at the prices that would be charged on the current AS graph if we assumed that all transit costs were the same. Out of a 9107-node AS graph, reflecting a recent snapshot of the current Internet[8], we selected a 5773-node biconnected subset. We then computed $d$, $d'$, and the payments that would result from our pricing scheme, assuming a transit cost of 1 for each node. We find that $d = 8$ and $d' = 11$, and so the convergence time of the pricing algorithm is not substantially worse than that of BGP. The highest transit node price was 9, and, with uniform traffic between all pairs, the mean node payment is 1.44. In fact, 64% of the node prices were 1, and 28% of them were 2. Thus, overcharging appears not to be a problem in this case, reflecting the high connectivity of the current Internet. Of course, the values of $d$ and $d'$ and the overcharging margin would be different with non-uniform transit costs; however, we expect them to exhibit similar trends towards low $d$, $d'$, and overcharging margin.

It would be interesting to ask whether this is because of the incentive issues in AS-graph *formation*. In this chapter, we merely looked at the routing aspects of a given AS graph. However, if one considers the incentives present when an AS decides whether or not to connect to another AS, the resulting transit prices would be a serious consideration. In particular, we conjecture that high node prices will not be sustainable in the Internet precisely because, if present, they would give an incentive for another AS to establish a link to capture part of that revenue, thereby driving down the transit prices.

[8]These data were taken from Route Views [Rou], which collates BGP tables from many sites across the Internet.

## 4.8 The Policy Routing Problem

The results in the preceding sections are based on a simple model in which ASes attempt to minimize per-packet transit costs. In practice, ASes have more complex costs and route preferences that are embodied in their *routing policies*. In this section, we introduce a formulation of the policy-routing algorithmic mechanism design problem.

Our model of policy routing retains several features of the lowest-cost routing problem described in Section 4.2: The network consists of $n$ Autonomous Systems. For simplicity, we treat each AS as an atomic entity; thus, the network can be modeled as a graph with nodes corresponding to the autonomous systems. The edges in this graph correspond to BGP peering relationships between ASes; we have a directed edge from node $a$ to node $b$ if $b$ advertises its routes to $a$. In practice, the edges in this graph may vary with the destination; however, we assume that these edges are identical for routes to any destination.

A **route** from a node $i$ to a node $j$ is simply a directed path, with no cycles, from $i$ to $j$ in this graph. The **routing problem** in this network is as follows: For each pair of nodes $i$ and $j$, we need to select a single route from $i$ to $j$. Further, we insist that the sets of all routes to destination $j$ form a tree rooted at $j$. This is a natural restriction when packets are routed one hop at a time (as opposed to being routed in an end-to-end manner, *e.g.*, source-routed). A candidate solution to the routing problem is thus a set of directed trees, one for each destination. The trees for different destinations are independent of each other, and hence it is possible to analyze the model for a single destination.

The root difference between the lowest-cost routing problem and the policy-routing problem lies in the source of preferences. In the former, the costs incurred by transit carriers result in them preferring routes that do not pass through them; in the latter, ASes have differing preferences over alternative routes, and the constraint that routes form a tree leads to conflicts of interest. There are many reasons why ASes may have real economic preferences for different routes: Two different routes from $i$ to $j$ may lead to differing transit costs, customer satisfaction, or service payments. In this section, we assume that AS $i$'s preferences among the candidate solutions are dictated entirely by the route from $i$ to $j$ in each solution, independent of the routes from other nodes to $j$. In a sense, this is complementary to the lowest-cost routing model, in which AS $i$'s utility for a tree depends only on the routes on which it was a *transit* node.

Further, we suppose that AS $i$'s preferences for paths can be expressed as a **utility function** $u_i : \mathcal{P}_{ij} \to \mathbb{R}$, where $\mathcal{P}_{ij}$ is the union of all possible paths from $i$ to $j$ and the *empty path* $\perp$ (which corresponds to solutions in which there is no route from $i$ to $j$). Only the relative utilities are important, and so we can normalize this function by requiring that $u_i(\perp) = 0$. Further, we assume that for any route $P_{ij}$ from $i$ to $j$, $u_i(P_{ij}) \geq 0$; in other words, having any route to $j$ cannot be worse for $i$ than having no route at all.

Abstractly, a mechanism for the routing problem for destination $j$ uses the user utility profile $\mathbf{u}$ and outputs a routing tree $T$ and a vector of payments $\mathbf{p} = (p_1, \ldots, p_n)$, where $p_i$ is the amount of money paid to $i$. We use the notation $u_i(T)$ to denote $i$'s utility for its path to $j$ in the tree $T$.

The overall economic goal of this routing mechanism is to maximize the *overall welfare*, *i.e.*, the sum of the users utilities. In other words, we want to maximize a function $W(T)$ where

$$W(T) = \sum_{i \in P} u_i(T)$$

We call this the *welfare-maximizing routing problem*.

## 4.9 NP-hardness of the general problem

In this section, we show that it is not tractable to maximize the overall welfare when the route valuations can be arbitrary. We will focus on computing routes to a single destination only. BGP essentially computes routes to different destinations in parallel, so it is sufficient to consider the single-destination case. Moreover, our results show that a policy routing mechanism is hard to compute, even for a single destination. It follows *a fortiori* that it is hard to compute the mechanism when all destinations are considered (although the complexity may not grow by a factor of $n$).

An instance of the routing problem we are considering is as follows: We are given a directed graph $G$, with a distinguished destination node $j$. Each node $i$ is associated with a set $S_i$ of paths from $i$ to $j$ in $G$, representing the set of *allowed paths*, and a valuation function $u_i : S_i \to \mathbb{R}_{\geq 0}$.[9]

We now show that if the valuation functions are arbitrary, it is NP-hard to compute a tree that maximizes the overall welfare. We prove this result by a reduction from the *Independent Set* problem: Given a graph $G$ with vertices $N$, find a largest subset $S$ of $N$ such that no two vertices in $S$ have an edge between them. This problem is known to be NP-hard [Kar72]; in fact, it is even NP-hard to approximate the size of the largest independent set to within a factor of $n^{\frac{1}{2}-\epsilon}$ [Hås99].

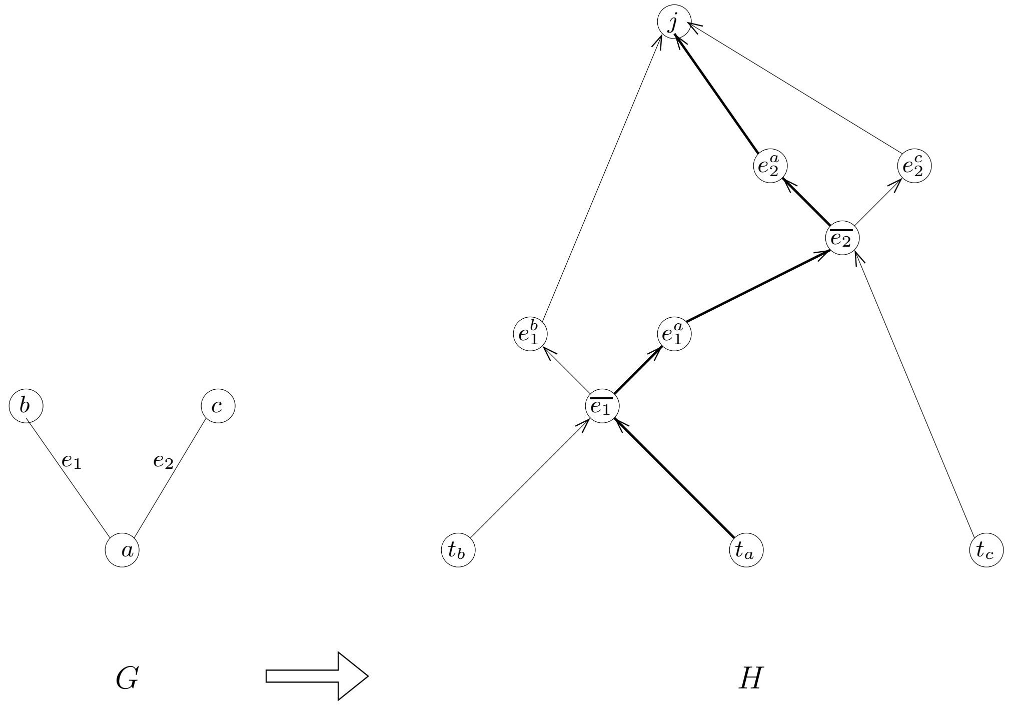


Figure 4.4: Reduction from Independent Set. The path $P_a$ is shown in bold.

Given an instance $G = (N, E)$ of the Independent Set problem, we construct an instance $H$ of the welfare-maximizing routing problem. The construction is illustrated in Figure 4.4. For each vertex $v$ in $N$, we have a *terminal vertex* $t_v$ in $H$. In addition, for each edge

[9]There may be an exponentially high number of paths from $i$ to $j$ in the graph (and, indeed, in the Internet). Thus, it might seem that even describing the AS valuation functions completely is a hopeless task. However, it is possible that an ASes valuation function can be described with polynomial amounts of space. We include a set of allowed paths in the problem description simply to provide one such representation: A path $P_{ij}$ implicitly has valuation 0 if it is not in the allowed set. The NP-hardness reduction in this section shows that, even when all ASes have valuation functions that can be expressed concisely using this representation, it is NP-hard to find a welfare-maximizing routing tree. Any other concise representation of valuation functions with small support would suffice for the reduction described here.

$e = (v_1, v_2)$ in $E$, we add three vertices $e^{v_1}, e^{v_2}$, and $\overline{e}$ to $H$. We also add directed edges from $\overline{e}$ to $e^{v_1}$ and $e^{v_2}$. Finally, we add a special *destination vertex* $j$ to $H$. We then choose an arbitrary order for the edges in $E$. For a vertex $v$ in $N$, let $e_1, e_2, \ldots, e_l$ be the edges incident on $v$ in $G$, in that order. We add the directed edges $(t_v, \overline{e}_1), (e_1^v, \overline{e}_2), \ldots (e_{l-1}^v, \overline{e}_l), (e_l^v, j)$ to $H$.

In this manner, we construct a directed path

$$P_v = (t_v, \overline{e}_1), (\overline{e}_1, e_1^v), (e_1^v, \overline{e}_2), \ldots, (\overline{e}_l, e_l^v), (e_l^v, j)$$

for each terminal vertex $t_v$. Now, we let $S_{t_v} = \{P_v\}$, and $u_{t_v}(P_v) = 1$, for each such vertex. For a nonterminal vertex $\overline{e}$ corresponding to an edge $e = (v_1, v_2)$ in $G$, we let $S_{\overline{e}} = \{\overline{P}_{v_1}, \overline{P}_{v_2}\}$, where $\overline{P}_{v_1}$ is the suffix of $P_{v_1}$ from $\overline{e}$ to $j$, and $\overline{P}_{v_2}$ is the suffix of $P_{v_2}$ from $\overline{e}$ to $j$. We let $u_{\overline{e}}(\overline{P}_{v_1}) = u_{\overline{e}}(\overline{P}_{v_2}) = 0$. Similarly, for a vertex of the form $e^v$, we let $S_{e^v}$ contain only the suffix of $P_v$ from $e^v$ to $j$, and let $e^v$'s valuation for this path be 0.

**Lemma 4.3** *Given an instance $G = (N, E)$ of the Independent Set problem, let $(H, \{S_i\}, \{u_i(\cdot)\})$ be an instance of the welfare-maximizing routing problem constructed as described above. Let $T^*$ be an optimal routing tree for this problem. Then, the following conditions hold:*

*(i). For any vertices $v_1, v_2 \in N$ such that $(v_1, v_2)$ is an edge in $G$, at most one of $t_{v_1}$ and $t_{v_2}$ has a path to $j$ in $T^*$.*

*(ii). If $S \subseteq N$ is an independent set, then $W(T^*) \geq |S|$.*

**Proof:** (i) Let $e$ be the edge $(v_1, v_2)$. If $t_{v_1}$ has a path to $j$, it must be the path $P_{v_1}$. The vertex $\overline{e}$ lies on this path, and hence the unique path from $\overline{e}$ to $j$ in $T^*$ must pass through $e^{v_1}$, not $e^{v_2}$. It then follows that the path $P_{v_2}$ is not contained in $T^*$, and hence there is no path from $t_{v_2}$ to $j$ in $T^*$.
(ii) No two vertices in $S$ have any edge in common; hence, if $v_1, v_2 \in S$, the paths $P_{v_1}$ and $P_{v_2}$ are disjoint. Thus, the union of paths $P_v$ for all $v \in S$ forms a tree $T(S)$. Further, we note that $W(T(S)) = |S|$. $T^*$ is optimal, and hence $W(T^*) \geq |S|$. $\Box$

**Corollary 4.2** *If $S$ is a maximum independent set in $G$, then $T(S)$ is an optimal routing tree. Conversely, if $T^*$ is an optimal routing tree, then $S = \{v|t_v$ has a path to $j$ in $T^*\}$ is a maximum independent set in $G$.*

□

Finally, we observe that this reduction implies that even an approximately optimal routing tree is hard to find: If $\tilde{T}$ is an approximately optimal routing tree, then $\tilde{S} = \{v|t_v$ has a path to $j$ in $\tilde{T}\}$ is an approximately maximum independent set in $G$, with the same approximation factor. Note that we reduce a graph with $n$ vertices to a network with $\mathbf{O}(n^2)$ nodes and $\mathbf{O}(n^2)$ allowed paths. Thus, an $(n^2)^{\frac{1}{4}-\epsilon} = n^{\frac{1}{2}-\frac{\epsilon}{4}}$ approximation to the welfare-maximizing routing problem would give us an $n^{\frac{1}{2}-\frac{\epsilon}{4}}$ approximation to the independent set problem, and an $(n^2)^{\frac{1}{2}-\epsilon} = n^{1-\frac{\epsilon}{2}}$ approximation to the welfare-maximizing routing problem would give us an $n^{1-\frac{\epsilon}{2}}$ approximation to the independent set problem. Combining this with known results on the hardness of computing exactly maximum independent sets and approximately maximum independent sets [Kar72, Hås99], we have the following theorem:

**Theorem 4.3** *Given a general network on $n$ nodes, with a total of $\mathbf{O}(n)$ allowed paths and arbitrary AS-path valuations,*

- *Unless $NP = P$, there is no polynomial-time algorithm to compute a welfare-maximizing routing tree.*
- *For any $\epsilon > 0$, unless $NP = P$, there is no polynomial-time algorithm to compute a tree the total welfare of which approximates that of a welfare-maximizing routing tree to within a factor of $n^{\frac{1}{4}-\epsilon}$.*
- *For any $\epsilon > 0$, unless $NP = ZPP$, there is no polynomial-time algorithm to compute a tree the total welfare of which approximates that of a welfare-maximizing routing tree to within a factor of $n^{\frac{1}{2}-\epsilon}$.*

□

Theorem 4.3 probably rules out the possibility of exactly or approximately solving this problem for the most general case. There are two possible approaches to restricting the scope of the problem in order to make it more tractable. The first is to restrict the class of networks, while still covering Internet-like situations. The second approach is to restrict the class of allowable route valuations; we pursue the second approach in Section 4.10.

## 4.10 Next-hop preferences

In this section, we consider solutions to the welfare-maximizing routing problem with a restricted class of AS preferences. Specifically, we assume that AS $i$'s valuation $u_i(P_{ij})$ for route $P_{ij}$ depends only on the *next hop* from $i$ on this route (*i.e.*, the valuation depends only on which of $i$'s neighbors this route passes through). The motivation for this is that an AS is likely to have different economic relationships with different neighbors (customers, providers, and peers), leading to different valuations for routes depending on which neighbor is used for transit; however, it is reasonable to assume that two routes to $j$ through the same neighbor have a similar economic impact on $i$. Further, we assume that the set of allowed routes from $i$ is likewise determined solely by which neighbors of $i$ may be used to transit packets destined to $j$.

With this assumption, $i$'s valuation function can be written as a function $u_i(a)$ of the neighboring AS $a$. Similarly, the set of $i$'s allowed routes can be expressed as a set $S_i$ of $i$'s neighbors that can be used to carry transit traffic to $j$. (The set $S_i$ reflects agreements between $i$ and its neighbors: If $a \in S_i$, it means that, in principle, $i$ is willing to send packets through $a$, *and* $a$ is willing to accept packets from $i$ for destination $j$.)

This leads to a convenient combinatorial form of the welfare-maximizing routing problem. We construct a graph $G_j$, with a vertex corresponding to each AS, and an identified destination vertex $j$. If $a \in S_i$, we include a directed edge $e$ from $i$ to $a$; we assign this edge a *weight* $u_e = u_i(a)$. A routing tree is then simply a directed tree (*arborescence*) $T$ with all edges directed towards the root $j$. Further, an AS $i$'s valuation for its route in $T$ is the weight $u_e$ of the edge outgoing from $i$ in $T$ if such an edge exists or 0 otherwise. Thus, the overall welfare with routing tree $T$ is

$$W(T) = \sum_{e \in T} u_e$$

It follows that the welfare-maximizing routing tree $T^*$ is a *maximum-weight directed tree* with root $j$ in $G_j$.

We first show that we can restrict our attention to directed *spanning trees.*

**Lemma 4.4** *Suppose we are given a weighted graph $G_j$, with vertex set $N$. Define $R \subseteq N$ by*

$$R \overset{def}{=} \{i \in N \mid \textit{There is a path from } i \textit{ to } j \textit{ in } G_j\} \cup \{j\}$$

*Then, there is a maximum-weight directed tree with root $j$ that spans $R$.*

**Proof:** Let $T^*$ be a maximum-weight directed tree with root $j$. Suppose there is some vertex $v \in R$ such that $v \notin T^*$. There is a path from $v$ to $j$ in $G_j$; we can add edges from this path to $T^*$ without decreasing its weight, because the valuations are always non-negative. By adding edges along this path in order, we can eventually grow the tree to include $v$, without reducing its weight. □

Note that the ASes that cannot even reach $j$ can be completely ignored for the purpose of finding routes to $j$. Also, it is easy to compute, for each AS $i$, whether $j$ is reachable from $i$. This, combined with Lemma 4.4, means that without loss of generality, we can assume that $T^*$ spans the vertex set $N$.

Thus, we want to compute a maximum-weight directed spanning tree, with edges directed towards $j$ (a *maximum-weight $j$-arborescence*).[10] This is a well-studied problem; one distributed algorithm for this problem was given by Humblet [Hum83].

### 4.10.1 A VCG Mechanism

We now describe an efficient, strategyproof mechanism for the welfare-maximizing routing problem with next-hop valuations. This is a direct application of the theory of Vickrey-

[10]This is essentially equivalent to the problem of computing a *minimum*-weight $j$-arborescence, with weights adjusted appropriately.

Clarke-Groves (VCG) mechanisms. It follows from the characterization of efficient, strategyproof mechanisms [GL79] that the payment to AS $i$ must have the form:

$$p_i = \sum_{a \neq i} u_a(T^*) \; + \; h_i(\mathbf{u}^{-i}) \tag{4.5}$$

Here, $h_i(\cdot)$ is an arbitrary function of $\mathbf{u}^{-i}$, the vector of valuations of all agents other than $i$. We normalize the payment by requiring that nodes that do not carry transit traffic (leaf nodes in $T^*$) are not paid. The rationale for this requirement here is that leaf nodes are not contributing to other agents' value.

Let $T^{-i}$ be the maximum weight $j$-arborescence in $N \backslash \{i\}$.[11] Then, $W(T^{-i})$ is a function of $\mathbf{u}^{-i}$ alone. Recall that an AS can refuse to accept transit traffic, *i.e.*, effectively cut off all *incoming* edges. If AS $i$ did this, it would force the optimal tree to have it as a leaf node. We would then have $T^* = T^{-i} \cup (i, a)$, where $(i, a)$, an edge from AS $i$ to some other AS $a$ in the network, is the heaviest outgoing edge from $i$. As $i$ would be a leaf, the payment $p_i$ must be 0 in this case; for this to occur, we must have $h_i(\mathbf{u}^{-i}) = -W(T^{-i})$. Substituting back into Equation 4.5, we get the following formula for the payment $p_i$:

$$\begin{aligned} p_i &= \sum_{a \neq i} u_a(T^*) \; - \; W(T^{-i}) && (4.6) \\ &= W(T^*) - u_i(T^*) \; - \; W(T^{-i}) && (4.7) \end{aligned}$$

We call this the *MDST mechanism*. In order to compute this mechanism, we will have to compute the MDST, as well as the payment $p_i$ to be given to each AS $i$. The payments can be computed by solving $(n-1)$ minimum-weight $j$-arborescence instances (one for each node except $j$).

### 4.10.2 Proving hardness in a BGP-based model

Up to this point, we have formulated the problem of finding the welfare-maximizing routing tree with next-hop preferences as a maximum-weight directed-spanning-tree problem and derived the natural strategyproof, efficient mechanism for this problem. This mechanism

[11]We assume the network is 2-connected, and hence such a tree exists.

is polynomial-time computable in a centralized computational model; this leads us to hope that, as in the case of lowest-cost routing, we can find a BGP-based distributed algorithm for it. Unfortunately, this appears not to be the case. In this section, we argue that this mechanism is incompatible with BGP.

"BGP-based computation" is not yet a precisely defined term. It is relatively easy to argue that an algorithm, for example the LCP-mechanism price-computation algorithm described in Section 4.6 above, does not cause large changes in the structure or performance of BGP. In order to prove impossibility results, however, we need to identify specific properties that we expect a BGP-based computation to have. We identify three such properties that suffice for the negative result sought in this section. We do not claim that these properties provide us with a full fleshed out "BGP computational model"; that is a goal for future work.

Consider routing to some destination $j$. The properties we require of any BGP-based computation of the routes to $j$ are:

**P1** The routing table should have roughly the same form and size of BGP routing tables, *i.e.*, there should be $\mathbf{O}(l)$ space used for a route of length $l$.

**P2** Routes should be computable in time proportional to the *diameter* of the network rather than the total size of the network.

**P3** When a node fails, or there is a change in the information (such as costs or preferences) associated with the node, the change should not always have to propagate to the whole network; instead, it should usually be propagated only to a small subset of nodes. In a *link-state* routing protocol, any change has to be broadcast to all the nodes in the network. BGP is a *path-vector* protocol, partly to avoid this dynamic communication burden; thus, a BGP-based algorithm should preserve this property.

As the set of routes to $j$ forms a tree, we cannot prevent changes in a few nodes near the root from affecting many other nodes. Similarly, it seems acceptable that a large change in the cost or preference of node $i$ can put it near the root and hence affect many nodes. However, we don't want *every* change to result in this much communication. Formally, we require that there are $\mathbf{o}(n)$ nodes that trigger $\Omega(n)$ `UPDATE` messages when one of them fails and comes back up or changes its cost or valuation by an infinitesimal amount.

It may be argued that requirements P2 and P3 capture desirable properties of distributed algorithms generally and not BGP-based algorithms in particular. This is not an obstacle for our purposes in this section. Because we are trying to show that the MDST mechanism is *not* BGP-compatible, it suffices to show that it does not have properties required for a larger class of algorithms that contain those that are BGP-based.

Another important point is that we do not necessarily require these conditions to hold for all possible networks and all possible cost or preference values. The only networks that we care about are "Internet-like" networks, those that can plausibly represent an AS graph, or some subgraph of an AS graph. For this reason, we restrict ourselves to networks that satisfy three properties: They must be sparse, with average node degree $\mathbf{O}(1)$; they must have small diameter $\mathbf{O}(\log n)$; and, when any one node is removed from the network, the diameter must remain $\mathbf{O}(\log n)$.

It is more difficult to identify what "reasonable" cost or preference values might be. We definitely want them to be polynomial in $n$ and preferably polylogarithmic in $n$. Further, we are not as concerned with hardness that may arise because of some strange coincidence of specific numerical values that happen to produce a very unstable state. At the same time, there is no single natural distribution with respect to which we can analyze the average-case complexity of an algorithm. Instead, we insist that any hardness result hold over an open set of cost or preference values; this means that the hardness holds over a region of preference space with non-zero volume, instead of at isolated points. This is similar in spirit to the *smoothed analysis* of Spielman and Teng [ST01].

First, we observe that the LCP-mechanism satisfies these properties, provided the costs are similar to each other, not very skewed. We follow the notation in Section 4.6.3, using $d$ to denote the length (in hops) of the longest chosen route and $d'$ to denote the length of the longest relevant minimum-cost $k$-avoiding path, for some $k$.

(P1) We have shown in Theorem 4.2 that the routing table is only a constant factor larger than the BGP routing table.

(P2) By Theorem 4.2, the mechanism converges in $\max(d, d')$ stages. For *unweighted* Internet-like graphs, both $d$ and $d'$ are $\mathbf{O}(\log n)$. If the weights are very skewed, the convergence may take $\Omega(n)$ stages; however, if all the weights are in the range $[1, r]$, for small $r$, then $d$ and $d'$ are at most a factor of $r$ greater than their respective values in the underlying graph. (Any path with more hops would have a cost higher than that of the corresponding LCP or minimum-cost $k$-avoiding path in the underlying graph.) In this case, the LCP mechanism converges in $\mathbf{O}(r \log n)$ stages.

(P3) The failure of a node $i$ only affects the nodes for which $i$ lies on the LCP or on the minimum-cost $k$-avoiding path (for some $k$). Thus, each node is affected by at most $dd'$ other node failures; this argument also holds for cost increases. Similarly, when the node comes back up, only those nodes that end up having it on their $LCP$ or minimum-cost $k$-avoiding path are affected. Finally, we note that a small change in the cost of one node does not change the routing tree (except in the rare case that multiple paths have the same length). Thus, a node near the root of the tree may impact $\Omega(n)$ nodes, but as most nodes are near the leaf of the tree, most changes only affect $O(dd')$ nodes. In Internet-like graphs with weights in a small range, we expect $d$ and $d'$ to be polylog$(n)$, and so most changes trigger `UPDATE` messages among only a small subset of the $n$ ASes.

By contrast, we show that the welfare-maximizing routing problem does not satisfy these properties, even for networks and preference values that fit our definition of "reasonable."

### 4.10.3 Long convergence time

Figure 4.5 shows an example of a network with $2n-1$ nodes for which a BGP-based algorithm for the welfare-maximizing routing mechanism takes $\Omega(n)$ stages to converge. The network consists of a balanced $j$-arborescence. The leaf nodes are $a_1, a_2, \ldots, a_n$. The network can be extended to have diameter $2 \log n$ by adding reverse edges with lower preference values; these reverse edges do not affect our argument, and so we omitted them from Figure 4.5. Similarly, by adding one more low-preference edge from each internal node, we can arrange for the diameter to remain small even when any one node is removed. Each node is adjacent to at most 4 other nodes, and so the network satisfies the sparseness requirement as well.

The preference values are shown as numbers (weights) on the edges in Figure 4.5. Each

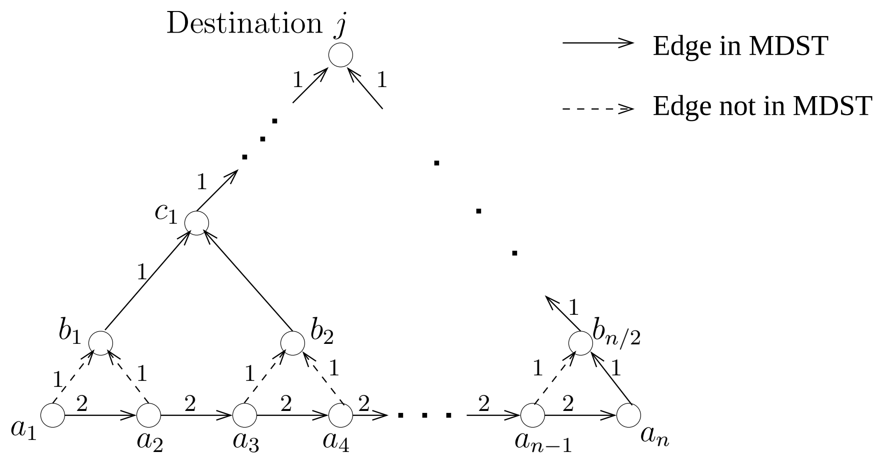


Figure 4.5: Network with low diameter and a long path in MDST.

$a_i$ in $\{a_1, a_2, \ldots, a_{n-1}\}$ prefers to route through its neighbor $a_{i+1}$ (value 2) rather than take the path up the tree (value 1). Thus, the welfare-maximizing routing solution, given by the maximum-weight directed spanning tree in this network, consists of the path $a_1 a_2 \cdots a_n$, attached to the remainder of the tree at $a_n$. Note that the values are in a small range $[1, 2]$. We also remark that this remains the optimal solution even if any subset of the next-hop values are perturbed by a small amount (less than 0.5 each).

Thus, the optimal solution has a route of length $\Omega(n)$, for any preference values in an open set around the specified values. BGP builds routes on a hop-by-hop basis. An AS can use a route only when its next hop on the route has advertised it, and it can itself extend and advertise the route only in the next stage. Thus, we have proved that any such algorithm does not satisfy property P2:

**Theorem 4.4** *Any BGP-based algorithm for computing the next-hop welfare-maximizing mechanism in the network of Figure* 4.5*, over an open set of preference values in a small range, takes* $\Omega(n)$ *stages to converge.*

□

Given the hop-by-hop route construction in BGP, it may seem that a more reasonable requirement than P2 is that the number of stages required for convergence is proportional to the length of the longest route. However, the length of the longest selected route is also a function of the mechanism under consideration (in this case, the MDST mechanism);

for this reason, we prefer the more stringent requirement P2, which is independent of the mechanism. One of the reasons that the MDST mechanism is incompatible with BGP is precisely that it may select very long routes even in networks with small diameter and hence will cause BGP (or any hop-by-hop protocol substrate) to converge very slowly.

### 4.10.4 Extensive dynamic communication

It may be argued that the long route in Figure 4.5 is unlikely to arise, because long routes are inherently undesirable, and hence ASes will lower their preference values for neighbors with long routes to the destination. In other words, even though next-hop valuations may adequately capture an AS's preferences at any given time, these valuations will themselves evolve (over a longer time period, perhaps) to rule out value profiles that lead to long routes. In this section, we show that, even if there are no long routes, any algorithm to compute the next-hop welfare-maximizing mechanism will not satisfy condition P3: There are situations in which every change in a single node's valuation will trigger update messages to at least half of the other nodes.

We show this by constructing a network as depicted in Figure 4.6. The network has $n = 2^m + 1$ nodes. We construct it with by recursively constructing clusters of nodes.

At the bottom, we construct a 1-cluster that consists of two nodes, $B$ and $R$. The 1-cluster has two edges, a "blue" edge from $R$ to $B$ and a "red" edge from $B$ to $R$. Here, "blue" and "red" are simply labels that we attach to the edges to clarify the analysis; they have no particular semantics. Each of these two edges has weight $L - 1$, where $L = 2m + 4$.

In each cluster in our construction, we identify two special nodes: One is the "blue port" and one is the "red port." For a 1-cluster, $B$ is the blue port and $R$ is the red port. We recursively construct $(k + 1)$-clusters from two $k$-clusters, for $k = 1, 2, \ldots, m - 1$, using the construction in Figure 4.6(b): We add a blue edge from the blue port of the right $k$-cluster to the blue port of the left $k$-cluster; the latter then serves as the blue port of the $(k + 1)$-cluster. Similarly, we add a red edge from the red port of the left $k$-cluster to the red port of the right $k$-cluster, which serves as the red port of the $(k + 1)$-cluster. These edges both have weight $L - 2k - 1$.

Once we have built up the $m$-cluster in this manner, we complete the network construction as shown in Figure 4.6(c): We add one more node, the destination $j$. We also add a blue edge from the blue port of the $m$-cluster to $j$, with weight $L - 2m - 1 = 3$, and a red edge from the red port of the $m$-cluster to $j$, with weight $L - 2m - 2 = 2$. The complete network, for $m = 3$, is shown in Figure 4.6(d).

This network is sparse (each node has only two outgoing edges) and has low diameter, as required. All the valuations are in the range $[1, L]$, where $L = \mathbf{O}(\log n)$. The network we have just built has two distinguished directed spanning trees to destination $j$: one consisting of all the blue edges and one consisting of all the red edges. In each of these trees, the longest path (route) has $m+1 = \mathbf{O}(\log n)$ hops. We will now show that these two directed spanning trees have greater weight than any other directed spanning tree with destination $j$.

**Lemma 4.5** *If $T$ is a $j$-arborescence in a network of the form shown in Figure 4.6, and $T$ has both blue and red edges, then there is another $j$-arborescence $\tilde{T}$ such that $W(\tilde{T}) \geq W(T) + 2$.*

**Proof:** Consider a minimum-sized cluster that has both red and blue outgoing edges in $T$. Suppose this is a $(k+1)-cluster$, as shown in Figure 4.7(a). Consider the two $k$-clusters it is composed of, and label the ports $B_1, R_1, B_2, R_2$ as shown.

Now, the $(k+1)$-cluster has a blue outgoing edge; it must be from the blue port $B_1$. All smaller clusters have only one color of outgoing edge in $T$. It follows that the left $k$-cluster must have only blue edges. Similarly, the red outgoing edge must be from the port $R_2$, and so the right $k$-cluster must have all red edges. Thus, the spanning tree $T$ must include the blue spanning tree of the left $k$-cluster, the red spanning tree of the right $k$-cluster, and the two outgoing edges with weight $L - 2k - 3$ (or less if $k = m - 1$).

We now construct the tree $\tilde{T}$ as shown in Figure 4.7(b): we replace the red spanning tree by a blue spanning tree, and replace the red outgoing edge by the blue edge within the $(k+1)$-cluster, with weight $L - 2k - 1$. Because of the symmetric construction of the $k$-clusters, the red and blue spanning trees have the same weight. Thus, the overall weight of $\tilde{T}$ is at least 2 higher than the weight of $T$. □

**Lemma 4.6** *For the network and weights $\mathbf{u}$ as constructed in Figure 4.6, the maximum-weight $j$-arborescence $T^*(\mathbf{u})$ is the blue spanning tree. Further, for any node $B_x$ that is the* *blue node of its* 1*-cluster,* $T^{-B_x}(\mathbf{u})$ *(the maximum-weight* $j$*-arborescence on* $N\backslash\{B_x\}$*) is the red spanning tree restricted to* $N\backslash\{B_x\}$*.*

**Proof:** From Lemma 4.5, we know that the maximum weight $j$-arborescence must be either entirely blue or entirely red. At the top level, the blue edge has a higher weight than the red edge; at all other levels of the construction, the weights are the same. Thus, the blue spanning tree must be the maximum-weight $j$-arborescence $T^*(\mathbf{u})$.

The red spanning tree has $B_x$ as a leaf and has weight only 1 less than optimal. Any other $j$-arborescence with $B_x$ as a leaf must have both red and blue edges and hence have weight at least 2 less than optimal, by Lemma 4.5. Finally, we observe that any $j$-arborescence on $N\backslash\{B_x\}$ can be extended to a $j$-arborescence that has $B_x$ as a leaf, by adding the red edge $(B_x, R_x)$ with weight $L-1$. Thus, the restriction of the red subtree to $N\backslash\{B_x\}$ must be optimal. $\square$

Now, consider perturbing the weights $\mathbf{u}$ by adding an amount $\delta_e$ to the weight of each edge $e$, for any $\delta_e$ with absolute value less than $\frac{1}{n}$. Then, the weight of any spanning tree cannot change by 1 or more, and so Lemma 4.6 still holds.

This leads us to the hardness result for this section:

**Theorem 4.5** *For the network constructed in Figure 4.6 and an open set of valuations in a small range, any infinitesimal change in valuation must cause* `UPDATE` *messages to be sent to at least* $(n-3)/2$ *nodes.*

**Proof:** We start with the weight vector $\mathbf{u}$. A perturbed weight vector $\tilde{\mathbf{u}}$ can be constructed from $\mathbf{u}$ as follows: For each node $i$, we add $\delta_i^{\text{blue}}$ to the weight of the blue outgoing edge from $i$ and $\delta_i^{\text{red}}$ to the weight of the red outgoing edge from $i$, where $|\delta_i^{\text{blue}}|, |\delta_i^{\text{red}}| < \frac{1}{n}$. This corresponds to picking a weight vector from an open set around $\mathbf{u}$.

Consider the payment $p_{B_x}$ due to some node $B_x$. Let $k$ be such that $B_x$ is the blue port of a $k$-cluster, but not the blue port of a $(k+1)$-cluster. Then, the blue outgoing edge from $B_x$ has weight $(L-2k-1)$. The red outgoing edge from $B_x$ must have weight $(L-1)$, and so using Lemma 4.6 and Equation 4.7, we get

$$\begin{aligned}
p_{B_x} &= W(T^*) - u_{B_x}(T^*) - W(T^{-B_x}) \\
&= W(\text{blue spanning tree}) - (L - 2k - 1) - [W(\text{red spanning tree}) - (L-1)] \\
&= \Big[W(\text{blue spanning tree}) - W(\text{red spanning tree})\Big] + 2k \\
&= \Big[1 + \sum_{i \in N} (\delta_i^{\text{blue}} - \delta_i^{\text{red}})\Big] + 2k \qquad (4.8)
\end{aligned}$$

Note that $p_{B_x}$ satisfies Equation (4.8) for any perturbed weight vector $\tilde{\mathbf{u}}$ in the given range. Now, suppose we start from some weight vector $\tilde{\mathbf{u}}$, and then there is an infinitesimal change in $\delta_a^{\text{blue}}$ (or $\delta_a^{\text{red}}$) for some node $a$. It follows from Equation (4.8) that $p_{B_x}$ changes when this happens, and hence node $B_x$ must receive an update message (or else, it cannot update its value of $p_{B_x}$). This is true for every blue node, and thus an infinitesimal change in any node's preference must cause price updates at every blue node (a total of $\frac{n-1}{2}$ nodes). Apart from the node $a$ that originated the change (which may be a blue node), every other blue node must receive an update message, thus proving the theorem statement.[12] $\square$

Theorem 4.5 shows the essence of why the MDST mechanism appears difficult for a BGP-based computational model: A small change at any one node can cause changes that are global, not confined to the routes the node lies on. This appears to be an inherent problem of the maximum-weight directed spanning tree structure: Even if we neglected the payment computation, the failure of any blue node would force the red spanning tree to be used, effectively changing the routes of all other nodes.

[12]We assume here that the payment $p_{B_x}$ must be stored at $B_x$. Even if this is not true, we could get a result that is nearly as strong, as follows: $p_{B_x}$ must be stored at *some* node. By property P1, each node can store $\mathbf{O}(m)$ values only; thus, the payments for all the blue nodes must be distributed across $\Omega(n/m) = \Omega(\frac{n}{\log n})$ nodes, which must all receive `UPDATE`s every time the preferences change.

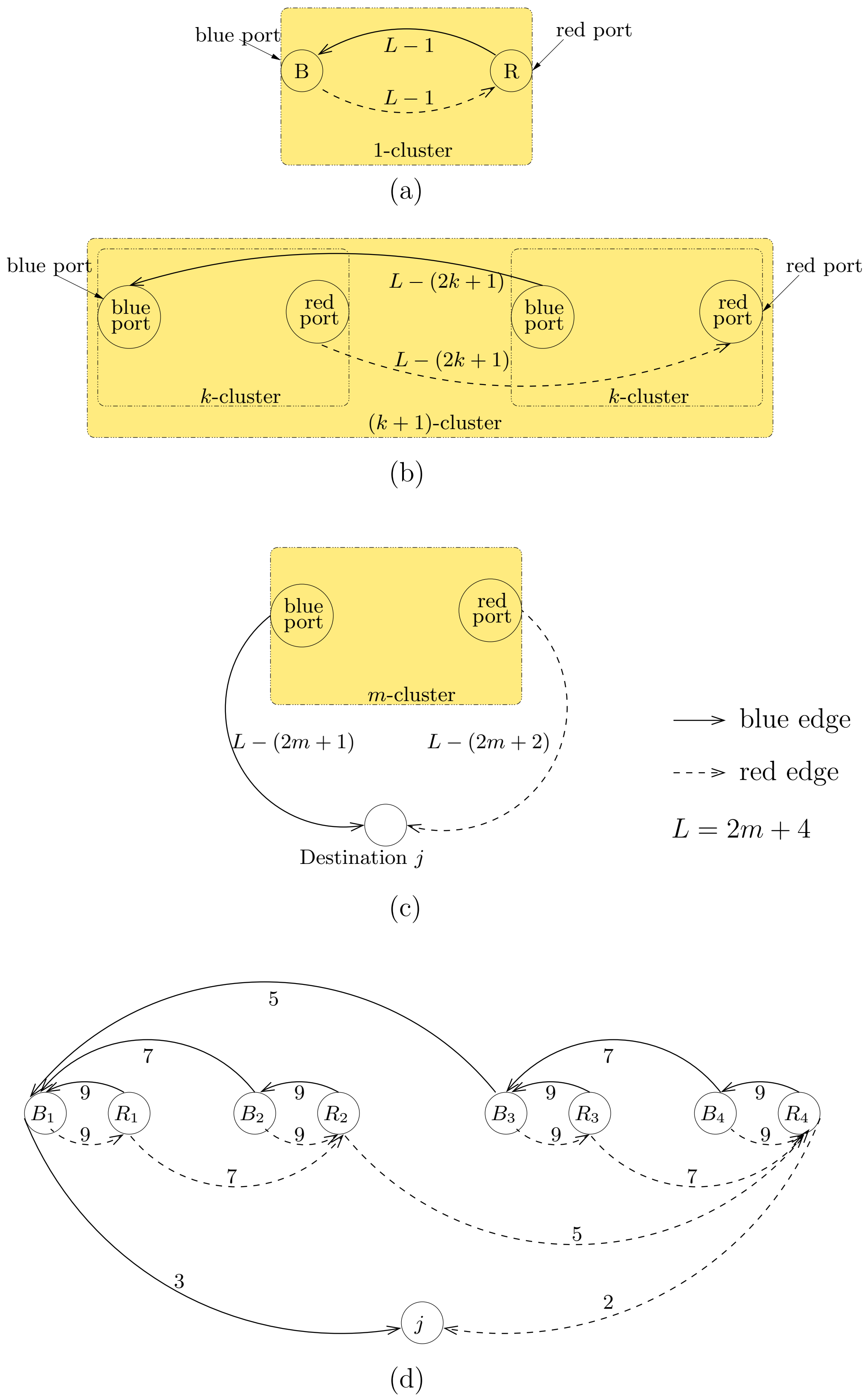
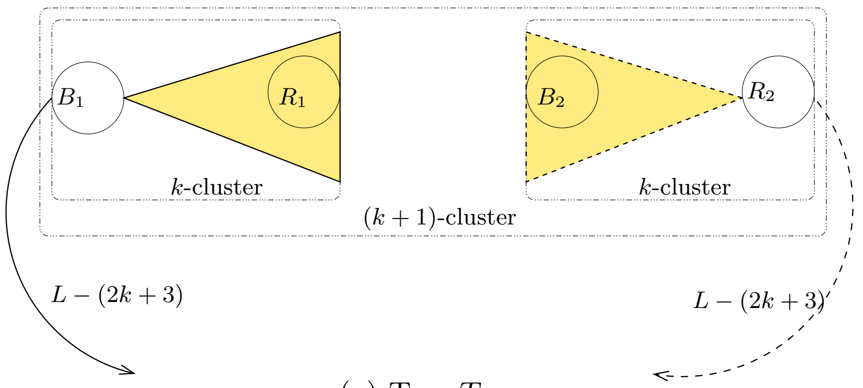
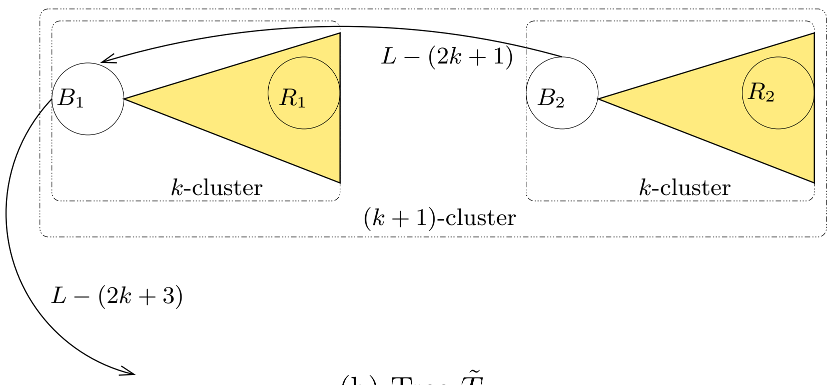


Figure 4.6: Construction of network for Section 4.10.4. (a) Basic 2 node cluster (b) Recursive construction (c) Top level of network (d) Complete network for $m = 3$



(a) Tree $T$



(b) Tree $\tilde{T}$

Figure 4.7: Construction to increase the weight of a tree $T$ with both red and blue edges.

# Chapter 5

# Conclusion

In this dissertation, we have explored aspects of distributed algorithmic mechanism design through the study of two problems, multicast cost sharing and interdomain routing. The thesis of this dissertation is that the distributed-computational environment of a mechanism can significantly influence its feasibility. Our results support this thesis: In both problem domains, we find mechanisms that appear to be feasible in a centralized computational model, yet are provably hard in a distributed computational model. We also present mechanisms, for both multicast cost-sharing and routing problems, that do appear to be practical for distributed computation.

Our research also touches upon several important aspects of distributed algorithmic mechanism design in general: We consider *approximate* mechanisms and present upper and lower bounds for the multicast cost-sharing problem. We incorporate the current standard protocol (BGP) into our computational model for interdomain-routing mechanisms; this is potentially important for other network services. Finally, we introduce the concept of *canonically hard* problems, *i.e.*, mechanism-design problems in which hardness arises from a combination of the strategic requirements and the distributed-computational model. Finding and analyzing such problems may lead to a clearer picture of tractability and intractability in Internet computation.

## 5.1 Open Questions

Distributed algorithmic mechanism design is a new field, and much work remains to be done before we fully understand the interaction between incentives and distributed computation on the Internet.

We first mention some specific open questions that are suggested by the results in this thesis.

The results in Chapter 3 lead naturally to the following question about the SH mechanism: Is there an approximation to the SH mechanism with the same worst-case network complexity as mechanism SSF? That is, is there a mechanism with the same worst-case network complexity that also achieves constant-factor bounds on the budget deficit (or surplus) and on the worst-case welfare loss?

In Chapter 4, we considered two formulations of incentive issues that arise in interdomain routing. It would be interesting to study a formulation that includes both transit costs and source preferences. Another interesting direction is to augment the network model with link or node capacities in order to tackle the problem of routing in congested networks. This is particularly natural, because it seems plausible that transit traffic imposes costs only in the presence of congestion.

One important issue that is not yet completely resolved is the possibility of nodes strategizing in the *computation* of the mechanism. This issue does not arise in [NR01, HS01], where the mechanism is a centralized computational device that is distinct from the strategic agents who supply the inputs or in the distributed multicast cost-sharing mechanisms, where the mechanism is a distributed computational device (*i.e.*, a multicast tree) that is distinct from the strategic agents (who are users resident at various nodes of the tree but not in control of those nodes). However, it is of great relevance in the interdomain-routing problem. On the one hand, we acknowledge that ASes may have incentives to lie about costs in order to gain financial advantage, and we provide a strategyproof mechanism that removes these incentives. On the other hand, it is these very ASes that implement the distributed algorithm we have designed to compute this mechanism; even if the ASes input their true costs, what is to stop them from running a different algorithm that computes prices more favorable to them? If ASes are required to sign all of the messages that they send and to verify all of the messages that they receive from their neighbors, then the protocol we gave in Section 4.6 can be modified so that all forms of cheating are detectable [MSTT01]. Achieving this goal without having to add public-key infrastructure (or any other substantial new infrastructure or computational capability) to the BGP-based computational model is the subject of ongoing further work.

We now turn to some exciting directions for future work on distributed algorithmic mechanism design, apart from the two problems we have studied in this dissertation. Perhaps the most important direction is simply to study more problem domains. There are many systems that appear to involve incentives as well as distributed computation, including peer-to-peer file sharing, wireless ad-hoc networks, and grid computing. We believe that understanding more problems in detail will enable the development of a richer theory of such systems. In particular, it would be interesting to find other problems that are "canonically hard" as defined in Chapter 3, *i.e.*, where the difficulty of the problem arises from the interplay of incentives and distributed computation; hardness results of this form may ultimately be part of a "complexity theory of Internet computation."

Another important direction is to broaden the scope of distributed algorithmic mechanism design by introducing other solution concepts. Strategyproofness is a very strong (and hence restrictive) solution concept. Many real-world economic mechanisms are not strategyproof, but they appear to perform adequately in practice for well-understood reasons (*e.g.*, sufficient competition in markets). It would be interesting to identify problem settings in which a weaker solution concept is justified and then to study the impact of weakening the solution concept on the network complexity of the mechanism.

# Bibliography

[ACK$^+$02] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: An experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, November 2002.

[AFK$^+$03] Aaron Archer, Joan Feigenbaum, Arvind Krishnamurthy, Rahul Sami, and Scott Shenker. Approximation and collusion in multicast cost sharing. To appear in *Games and Economic Behavior*, 2003.

[AR02] Micah Adler and Dan Rubenstein. Pricing multicasting in more practical network models. In *Proceedings of the 13th Annual ACM-SIAM Symposium On Discrete Mathematics (SODA '02)*, pages 981–990. ACM Press/SIAM, New York, January 2002.

[Arr63] Kenneth J. Arrow. *Social Choice and Individual Values.* Wiley, New York, 1963.

[AT01] Aaron Archer and Éva Tardos. Truthful mechanisms for one-parameter agents. In *42nd IEEE Symposium on Foundations of Computer Science (FOCS '02)*, pages 482–491. IEEE Computer Society Press, October 2001.

[AT02] Aaron Archer and Éva Tardos. Frugal path mechanisms. In *Proceedings of 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '02)*, pages 991–999. ACM Press/SIAM, New York, January 2002.

[BFC93] Anthony Ballardie, Paul Francis, and Jon Crowcroft. Core based trees. In *ACM SIGCOMM '93*, pages 85–95. ACM Press, New York, September 1993.

[Cla71] E. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

[DEF$^+$96] Stephen Deering, Deborah L. Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, and Liming Wei. The PIM architecture for wide-area multicast routing. *IEEE/ACM Transactions on Networking*, 4(2):153–162, April 1996.

[DR89] Bhaskar Dutta and Debraj Ray. A concept of egalitarianism under participation constraints. *Econometrica*, 57:615–635, 1989.

[FGHK02] Amos Fiat, Andrew V. Goldberg, Jason D. Hartline, and Anna R. Karlin. Competitive generalized auctions. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC '02)*, pages 72–81. ACM Press, New York, May 2002.

[FKSS03] Joan Feigenbaum, Arvind Krishnamurthy, Rahul Sami, and Scott Shenker. Hardness results for multicast cost sharing. To appear in *Theoretical Computer Science*, 2003. Extended abstract appeared in *Proceedings of the 22nd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '02)*, LNCS volume 2556, pages 133–144. Springer-Verlag, Berlin,2002.

[FPS01] Joan Feigenbaum, Christos Papadimitriou, and Scott Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63:21–41, 2001.

[FPSS02] Joan Feigenbaum, Christos Papadimitriou, Rahul Sami, and Scott Shenker. A BGP-based mechanism for lowest-cost routing. In *21st ACM Symposium on Principles of Distributed Computing (PODC '02)*, pages 173–182, July 2002.

[FS97] Eric Friedman and Scott Shenker. Learning and implementation in the internet. Preprint. Available at `http://www.icir.org/shenker/decent.ps`, 1997.

[FS02] Joan Feigenbaum and Scott Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communication (DIALM '02)*, pages 1–13. ACM Press, New York, September 2002.

[GHW01] Andrew V. Goldberg, Jason D. Hartline, and Andrew Wright. Competitive auctions and digital goods. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '01)*, pages 735–744. ACM Press/SIAM, New York, January 2001.

[Gib71] A. Gibbard. Manipulation of voting schemes: A general result. *Econometrica*, 48:587–601, 1971.

[GL79] J. Green and J. Laffont. Incentives in public decision making. In *Studies in Public Economics*, volume 1, pages 65–78. North Holland, Amsterdam, 1979.

[Gro73] Theodore Groves. Incentives in teams. *Econometrica*, 41:617–663, 1973.

[GW99] Timothy G. Griffin and Gordon Wilfong. An analysis of BGP convergence properties. In *Proceedings of ACM SIGCOMM '99*, pages 277–288. ACM Press, New York, September 1999.

[Hås99] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[HC93] H. Holbrook and D. Cheriton. Ip multicast channels: Express support for large-scale single-source applications. In *ACM SIGCOMM '99*, pages 65–78. ACM Press, New York, September 1993.

[HS01] John Hershberger and Subhash Suri. Vickrey prices and shortest paths: What is an edge worth? In *Proceedings of the 42nd annual ACM Symposium on the Foundations of Computer Science (FOCS '01)*, pages 129–140. ACM Press, New York, 2001.

[HSE97] Shai Herzog, Scott Shenker, and Deborah Estrin. Sharing the "cost" of multicast trees: an axiomatic analysis. *IEEE/ACM Transactions on Networking*, 5(6):847–860, December 1997.

[Hum83] Pierre A. Humblet. A distributed algorithm for minimum weight directed spanning trees. *IEEE Transactions on Communications*, COM-31(6):756–762, June 1983.

[Jac01] Matthew O. Jackson. A crash course in implementation theory. *Social Choice and Welfare*, 18:655–708, 2001.

[JV01] Kamal Jain and Vijay Vazirani. Applications of approximation algorithms to cooperative games. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC '01)*, pages 364–372. ACM Press, New York, July 2001.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations (Proceedings of a Symposium on the Complexity of Computer Computations)*, pages 85–103. Plenum Press, New York, 1972.

[KN97] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, Cambridge, UK, 1997.

[LM90] Arjen Lenstra and Mark S. Manasse. Factoring by electronic mail. In *Advances in Cryptology, Eurocrypt '89*, LNCS volume 434, pages 355–371. Springer-Verlag, Berlin, 1990.

[LN00] Ron Lavi and Noam Nisan. Competitive analysis of incentive compatible on-line auctions. In *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC '00)*, pages 233–241. ACM Press, New York, October 2000.

[Meg78] N. Megiddo. Computational complexity of the game theory approach to cost allocation for a tree. *Mathematics of Operations Research*, 3:189–196, 1978.

[Mou91] Hervé Moulin. *Axioms of cooperative decision making*. Cambridge University Press, Cambridge, UK, 1991.

[Mou99] Hervé Moulin. Incremental cost sharing: Characterization by group strategyproofness. *Social Choice and Welfare*, 16(2):279–320, 1999.

[MR74] Kenneth Mount and Stanley Reiter. The informational size of message spaces. *Journal of Economic Theory*, 8:161–192, 1974.

[MS01] Hervé Moulin and Scott Shenker. Strategyproof sharing of submodular costs: Budget balance versus efficiency. *Economic Theory*, 18:511–533, 2001.

[MSTT01] John Mitchell, Rahul Sami, Kunal Talwar, and Vanessa Teague, December 2001. Private communication.

[MT99] Dov Monderer and Moshe Tennenholtz. Distributed games: From mechanisms to protocols. In *Proceedings of the 6th National Conference on Artificial Intelligence and of the 11th Conference on Innovative Applications of Artificial*

*Intelligence (AAAI/IAAI '00)*, pages 32–37. AAAI/MIT Press, Menlo Park, CA, July 1999.

[MT02] John Mitchell and Vanessa Teague, 2002. Private communication.

[MWG95] Andreu Mas-Colell, Michael Whinston, and Jerry Green. *Microeconomic Theory.* Oxford University Press, New York, 1995.

[NR00] Noam Nisan and Amir Ronen. Computationally feasible VCG mechanisms. In *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC '00)*, pages 242–252. ACM Press, New York, 2000.

[NR01] Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behavior*, 35:166–196, 2001.

[NS03] Noam Nisan and Ilya Segal. The communication requirements of efficient allocations and supporting Lindahl prices. (Working paper), 2003.

[Par99] David C. Parkes. *iBundle*: An efficient ascending price bundle auction. In *Proceedings of the ACM Conference on Electronic Commerce (EC '99)*, pages 148–157. ACM Press, New York, November 1999.

[PLB$^+$99] R. Perlman, C.-Y. Lee, A. Ballardie, J. Crowcroft, Z. Wang, T. Maufer, C. Diot, and M. Green. Simple multicast: A design for simple low-overhead multicast. IETF Internet Draft (Work in Progress), 1999.

[PU00] David C. Parkes and Lyle H. Ungar. Iterative combinatorial auctions: Theory and practice. In *Proceedings of the 7th Conference on Artificial Intelligence and of the 12th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI '00)*, pages 74–81. AAAI/MIT Press, Menlo Park, CA, July 2000.

[Rou] University of Oregon Route Views Project. `http://www.routeviews.org`.

[RT02] Tim Roughgarden and Éva Tardos. How bad is selfish routing? *Journal of the ACM*, 49:236–259, 2002.

[San99] Tuomas Sandholm. Distributed rational decision making. In G. Weiss, editor, *Multiagent systems: A Modern Introduction to Distributed Artificial Intelligence*, pages 201–258. MIT Press, Cambridge, MA, 1999.

[Sat75] Mark Sattertwaite. Strategy-proofness and arrow's conditions: existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10:198–217, 1975.

[ST01] Daniel Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC '01)*, pages 296–305. ACM Press, New York, July 2001.

[TGS01] H. Tangmunarunkit, R. Govindan, and S. Shenker. Internet path inflation due to policy routing. In *SPIE ITCom 2001*, pages 188–195. SPIE Press, Bellingham, August 2001.

[Vic61] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[Wal77] Mark Walker. On the informational size of message spaces. *Journal of Economic Theory*, 15:366–375, 1977.

[Wel93] Michael Wellman. A market-oriented programming environment and its applications to distributed multicommodity flow problems. *Journal of AI Research*, 1:1–23, 1993.

[WWWM01] M. Wellman, W. Walsh, P. Wurman, and J. Mackie-Mason. Auctions for decentralized scheduling. *Games and Economic Behavior*, 35:271–303, 2001.