Algorithmic Game Theory

Edited by

Noam Nisan

Hebrew University of Jerusalem

Tim Roughgarden

Stanford University

Éva Tardos

Cornell University

Vijay V. Vazirani

Georgia Institute of Technology

CAMBRIDGE
UNIVERSITY PRESS

CHAPTER 14

Distributed Algorithmic Mechanism Design

Joan Feigenbaum, Michael Schapira, and Scott Shenker

Abstract

Most discussions of algorithmic mechanism design (AMD) presume the existence of a trusted center that implements the required economic mechanisms. This chapter focuses on mechanism-design problems that are inherently distributed, i.e., those in which such a trusted center cannot be used. Such problems require that the AMD paradigm be generalized to *distributed algorithmic mechanism design* (DAMD).

We begin this chapter by exploring the reasons that DAMD is needed and why it requires different notions of economic equilibrium and computational complexity than centralized AMD. We then consider two DAMD problems, namely distributed VCG computation and multicast cost sharing, that illustrate the concepts of ex-post Nash equilibrium and network complexity, respectively.

The archetypal example of a DAMD challenge is interdomain routing, which we treat in detail. We show that, under certain realistic and general assumptions, one can achieve incentive compatibility in a collusion-proof ex-post Nash equilibrium without payments, simply by executing the Border Gateway Protocol (BGP), which is the standard for interdomain routing in today's Internet.

14.1 Introduction

To motivate the material in this chapter, we begin with a review of why game theory is relevant to computer science. As noted in the Preface to this book, computer science has traditionally assumed the existence of a central planner who dictates the algorithms used by computational nodes. While most nodes are assumed to be *obedient*, some nodes may malfunction or be subverted by attackers; such *byzantine* nodes may act arbitrarily.

This book's founding premise, in fact its *raison d'être*, is that there are many computational contexts in which there is no central (or cooperative) authority that controls the computational nodes. In particular, the Internet has changed computation from a largely local endeavor to one that frequently involves diverse collections of individuals (or machines acting on their behalf). For example, Web services, peer-to-peer systems, and even the interaction among packets on a wire are all cases in which

individuals with no ties to each other, except perhaps a common interest in a document or simultaneous use of a link, find themselves interacting over the Internet.

In such cases, it is often best to treat the computational entities as independent and selfish *agents*, interested only in optimizing their own outcome. As a category of behavior, selfishness lies between the extremes of automatic obedience and byzantine disruption; selfish agents are unwilling to follow a central planner's instructions, but they do not act arbitrarily. Instead, their actions are driven by incentives, i.e., the prospect of good or bad outcomes. The field of mechanism design, described in Chapter 9, has shown how, by carefully constructing economic mechanisms to provide the proper incentives, one can use selfish behavior to guide the system toward a socially desirable outcome.[1] This book is devoted to exploring the interaction of incentives and computing, a topic that has come to be known as Algorithmic Mechanism Design (AMD).

Substituting a decentralized set of incentives for a central planner is a radical departure from traditional algorithm design. However, most work in this new field of AMD assumes the presence of a central computational facility that performs the calculations required by the economic mechanism. In auctions, for example, the agents each have independent goals and desires, but the computation to determine winners and payments is done by the auctioneer, and the hardness of the computation is evaluated using traditional notions of complexity (see, e.g., Chapters 1, 9, 11, and 12). As such, AMD considers novel incentive-related algorithm *design* but uses a standard centralized model of algorithm *execution*.

This combination of decentralized incentives but centralized computation applies in a wide variety of settings, many of which have been described elsewhere in this book. This approach requires transmitting all the relevant information to a single, trusted entity (hereafter called the *trusted center*), which is feasible if (i) such a trusted center exists, and (ii) the communication required to transmit the information and the resulting computational burden on the trusted center are both manageable. However, if either of these two assumptions fails, then a more decentralized approach must be considered.

As we discuss in more detail in Section 14.3 of this chapter, the problem of interdomain routing is one in which a decentralized approach is valuable. The Internet is a collection of smaller networks, called Autonomous Systems (ASes), that are stitched together by the interdomain-routing system to form the fully connected Internet. The interdomain-routing system therefore plays a crucial role in the functioning, even the existence, of the Internet. However, any approach to interdomain routing must address the challenges of trust, scalability, and reliability. The ASes are competing economic entities who want to optimize the routing outcome achieved and minimize the private information revealed; accordingly, they not only act selfishly but are also unwilling to share private information with, or cede control to, any trusted center. Thus, the ASes must distribute the route computation among themselves.

Even if trust were not an issue, scalability would drive the system toward distributed route computation. Centralizing the route computation would involve transmitting the entire AS graph to a central location and updating it whenever the graph changed.

[1] This desired outcome is often defined as the optimum of some global objective function, but a wide variety of social standards can also be used.

Given the considerable size and volatility of the AS graph, such a centralized route computation would be infeasible.

Similarly, the need for reliability, so crucial in the Internet, tends to favor decentralized designs. In a centralized design, the trusted center becomes a single point of failure; the fate of the entire network rests on this single system that could fail or be subverted. As an example of how scalability and reliability can drive the need for decentralization, we note that current intradomain-routing algorithms, which do not span more than one AS and so are designed with the assumption of mutual trust among routers, are almost all distributed.

Thus, there is a need to decentralize not only incentives but also computation; this leads to Distributed Algorithmic Mechanism Design (DAMD), which is the central focus of this chapter. DAMD has the same dual concerns, incentive compatibility, and computational complexity, as AMD, but it differs in two important respects.

The first difference involves the nature of complexity. DAMD's measure of computational complexity is quite different from AMD's, because the computation is distributed. Any measure of the complexity of a distributed algorithm executed over an interconnection network T must consider at least five quantities: the total number of messages sent over T, the maximum number of messages sent over any one link in T, the maximum size of a message, the local computational burden at each node, and the storage required at each node. If a distributed algorithm requires an excessive expenditure of any one of these resources, then its complexity is unacceptable. We will use the term *network complexity* to refer to these, and other, metrics of the difficulty of distributed implementation.

If the interconnection network T is trusted by all the agents and can feasibly serve as the trusted center, then the measure of complexity is the main difference between AMD and DAMD. However, if the distributed computation is done by the agents, then a second difference arises: the strategic nature of the computation itself. In AMD, agents can manipulate a game only by their selection of actions among those described in the *definition* of the economic mechanism; they cannot affect the *computation* of the mechanism, because all outcomes are computed (by the trusted center) from the vector of strategies, according to the definition of the mechanism. If the agents themselves perform the computation using some distributed algorithm, then they have more opportunities to manipulate the outcome, e.g., by misrepresenting the results of a local computation to a neighboring agent or, more drastically, by simply not communicating with that neighboring agent at all, in an attempt to exclude him from the game. Our assumption of selfishness requires that we consider all forms of manipulative behavior when designing the economic mechanism; in particular, this means that we must provide incentives that ensure selfish agents find it in their best interest to perform the distributed computation correctly.

While this chapter discusses the use of incentives to prevent these other forms of manipulation, one can also use cryptographic protocols to replace trusted parties in mechanism computation. This active area of study is covered in Chapter 8 of this volume.

In the next section of this chapter, we briefly discuss two examples of DAMD. Our third section is devoted to an in-depth exploration of the incentive issues in interdomain routing. We conclude with open questions and exercises.

14.2 Two Examples of DAMD

As noted above, DAMD differs from AMD in two respects: the additional ways in which the agents can influence the outcome (referred to hereafter as "computational manipulation") and the measure of computational complexity (the aforementioned "network complexity").

Here, we briefly discuss two examples of DAMD that illustrate these issues. The first is a distributed implementation of a VCG mechanism (see Chapter 9); we will ignore network complexity in this example and focus on how to prevent manipulation of the computation. The second example is sharing the cost of a multicast transmission; it illustrates the notion of network complexity but, because we assume the presence of a trusted computational infrastructure, does not involve computational manipulation.

14.2.1 Distributed Implementation of VCG

We now discuss one way a set of agents can jointly implement a VCG mechanism without fear of manipulation. We start with a set of outcomes O and a collection of agents N, each with his own valuation v_i over those outcomes. In our notation, $\tilde{o}$ is an outcome that maximizes the *total social welfare* of the agents. That is, $\tilde{o} = \text{argmax}_{o \in O} \sum_{i \in N} v_i(o)$, W is the maximum total social welfare value, and W_{-i} denotes the maximum total social welfare of all agents except the i'th. For convenience, we focus on the particular mechanism in which $p_i = W_{-i} - W + v_i(\tilde{o})$, where p_i is the payment by agent i.

We assume that there is no trusted center; i.e., that the computation of the VCG mechanism must be done by the agents themselves. However, we do presume the existence of some central *enforcer* whose responsibility it is to implement the outcome $\tilde{o}$ decided upon by the agents and collect the payments; the enforcer can impose severe penalties if the agents do not agree on an outcome.

To see how a distributed computation can be manipulated, consider a network in which the nodes are connected in a ring, and there is exactly one agent at each node. Assume that the agents are computing a second-price auction of a single good by passing around a message containing the top two bids for that good. If an agent puts his bid on top and puts in a very low bid for the second bid, then he can get the good more cheaply (as long as these fields are not overwritten by some later agents that have higher bids).

More generally, consider any distributed algorithm A, capable of running over an arbitrary number of computational nodes, that takes as input a set of agent valuations and produces the maximizing outcome and the payments. As the preceding example suggests, if we run A over any subset of N to compute $\tilde{o}$, W, and each W_{-i}, then there is the possibility that an agent can manipulate the computation.

One way to avoid this is *replication*: Break the agents into two groups, have them exchange all their valuations, and then have each group compute its own version of $\tilde{o}$ and the p_i. If the two groups agree on the outcomes and payments, then those outcomes and payments are adopted; if not, all agents suffer a severe penalty. Here, an agent plays different roles in the two versions of the computation: In the first, his role is to help

compute the outcome and payments; in the other, his role is to provide his valuation so that others may perform this computation. For the first version, an agent i could engage in arbitrary computational manipulation in an attempt to obtain a more favorable p_i or choose an outcome he prefers to the socially optimal one; in the second version, all he could do is lie about v_i. Because the VCG mechanism is strategyproof, the agent will reveal truthfully to the other computational group and therefore, to avoid a severe penalty for inconsistency, will carry out the computation faithfully.

Notice that faithful computation is not a dominant strategy. If, for instance, all the other agents decide to choose a suboptimal outcome, then agent i is better off going along with that choice rather than causing a disagreement (and triggering the severe penalty). However, if all the other agents faithfully execute the prescribed algorithm A, then agent i is best off doing so as well. Thus, the most natural solution concept when considering computational manipulation is not dominant strategies but instead ex-post Nash equilibrium, which was defined in Chapter 9. We will expand on this point further when we discuss interdomain routing in Section 14.3 below.

In this example, we have focused on computational manipulation and ignored network complexity. In our next example, we do the opposite.

14.2.2 Sharing the Cost of a Multicast Transmission

Multicast is an Internet packet-transmission mode that delivers a single packet to multiple receivers. It is accomplished by setting up a shared delivery tree that spans 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. Because it is far more efficient than traditional unicast transmission (in which packets are sent only to a single destination), multicast is particularly appropriate for distributing popular real-time content, such as movies, to a large number of receivers.

Internet content distribution both provides benefits and incurs cost, which we can model as follows. We assume that there are agents, located at various places in the network, who would derive some utility from receiving the content and that a cost is incurred each time the content is transmitted over a network link. The policy question is how these costs and benefits should be distributed; more specifically, which agents should receive the content, and how much should each agent pay?

To define the problem more precisely, we consider a user population P residing at a set of network nodes N that are connected by bidirectional network links L. The multicast flow emanates from a source node $\alpha_o \in N$; given any set of receivers $S \subseteq P$, the transmission flows through a *multicast tree* $T(S) \subseteq L$ rooted at α_o that spans the nodes at which users in S reside. We make the natural assumption that routing is monotonic, i.e., that $S_1 \subseteq S_2 \Rightarrow T(S_1) \subseteq T(S_2)$.

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. The total cost $C(S)$ of reaching a set S of receivers is given by $C(S) = \sum_{l \in T(S)} c(l)$, and the net welfare $NW(S)$ of delivering content to this set of receivers is given by $NW(S) = \sum_{i \in S} u_i - C(S)$.

A *cost-sharing mechanism* determines which users receive the multicast transmission and how much each receiver is charged. We let $p_i \geq 0$ denote how much user i

is charged and σ_i denote whether user i receives the transmission; $\sigma_i = 1$ if the user receives the multicast transmission, and $\sigma_i = 0$ otherwise.

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

In our model, it is the *agents* who are selfish. The routers (represented by tree nodes), links, and other network-infrastructural components are obedient. The cost-sharing algorithm does not know the individual utilities, and so users could lie about them, but once they are reported to the network infrastructure (e.g., by sending them to the nearest router), the algorithms for computing $\sigma(u)$ and $p(u)$ can be reliably executed by the network. Thus, our interest here is in network complexity, not computational manipulation.

Given the selfish nature of agents, the mechanism should be strategyproof, i.e., revealing u_i truthfully should be a dominant strategy. There are two other desirable features one would want in a cost-sharing mechanism: budget balance (the sum of the charges p_i covers the total cost of transmitting the content) and efficiency (the total welfare is maximized). The classic result of Laffont and Green, as reviewed in Chapter 9, implies that no strategyproof mechanism with quasilinear utilities can achieve both budget balance and efficiency[2]; we therefore consider two separate mechanisms, one that achieves budget balance and one that achieves efficiency.

To achieve efficiency, we consider a VCG mechanism called *marginal cost* (MC). Let $\tilde{S}$ denote the largest set that maximizes $NW(S)$ (this is uniquely defined), and let $\widetilde{NW} = NW(\tilde{S})$; similarly, $\widetilde{NW}_{-i}$ is the maximum value over all S of $NW(S - i)$. Then the MC mechanism chooses the receiver set $\tilde{S}$ and sets payments $p_i = \sigma_i u_i - \widetilde{NW} + \widetilde{NW}_{-i}$.

For budget balance, we choose the Shapley Value (SH) mechanism. The mechanism shares the cost of each link equally among all the agents downstream of that link; an agent i is downstream of a link l if $l \in T(\{i\})$. To determine which agents receive the transmission, we first start with $S = P$ and compute the charges. We then eliminate any agent for which the charge exceeds the agent's utility (i.e., $p_i > u_i$) and recursively prune the receiver set until all agents within the set have utilities greater than or equal to their charge. The cross-monotonic nature of these charges (an agent is never charged less after another agent leaves the receiver set) guarantees that the resulting set is well defined, independent of the order in which agents are eliminated. To see why the ordering does not matter, consider the following. We say that an elimination (or pruning) is "legal" if the node to be removed is charged more than its utility; an elimination ordering is "legal" if each individual pruning is legal. We note that, if an agent i is charged more than his utility when the set S of agents remains, then this continues to hold when any subset of S remains (because cross-monotonicity requires

[2] More precisely, the Laffont–Green result reviewed in Chapter 9 shows that the only strategyproof, welfare-maximizing mechanisms with quasi-linear utilities are the VCG mechanisms, which are known not to be budget-balanced. Myerson and Satterthwaite have shown a more general result about the impossibility of achieving efficient and budget-balanced allocations with rational agents; see Chapter 9 for details.

that i's charges are at least as great). This means that the concatenation of any two legal elimination orderings is also a legal elimination ordering (where we ignore duplicate prunings). For example, if $(1, 5, 7, 3)$ and $(7, 2, 5, 8)$ are two legal orderings, then $(7, 2, 5, 8, 1, 3)$ is also legal, as is $(1, 5, 7, 3, 2, 8)$. Thus, if any two subsets S and S' can be arrived at by sequences of legal eliminations, then $S \cap S'$ can also be arrived at by a sequence of legal eliminations.

It is easy to see that both MC and SH are polynomial-time computable by centralized algorithms; so the issue is whether it is hard to implement them in a distributed fashion. Certainly any mechanism can be computed by sending all the valuations to a single node, doing the computation, and then returning the results to each agent. In the worst case, this would require sending $\Omega(|P|)$ bits over some number of links, which is clearly not desirable. It turns out that we cannot do substantially better than this for the SH mechanism.

Theorem 14.1 *Any distributed algorithm, deterministic or randomized, that computes the SH multicast cost-sharing mechanism must send $\Omega(|P|)$ bits over linearly many links in the worst case.*

By contrast, it is possible to compute MC using only two short messages per link and two simple calculations per node. This is done in two phases, the first a bottom-up traversal in which welfare values are computed for each subtree of $T(P)$ and the second a top-down traversal in which membership bits σ_i and cost shares p_i are computed for each $i \in P$. The algorithms are given in Figures 14.1 and 14.2. In these figures, $V(P)$ denotes the node set of tree $T(P)$, $Ch(\alpha)$ the set of children of node α, $res(\alpha)$ the set of users resident at node α, u^α the sum of utilities of users in $res(\alpha)$, c^α the cost of the link connecting α to its parent in $T(P)$, and $T^\alpha(P)$ the union of the subtree rooted at α and the link connecting α to its parent.

The reason that this simple two-phase algorithm suffices is that computing the MC cost share p_i does not require a from-scratch computation of NW_{-i}. Rather, it is enough to compute W^α for every node α in $V(P)$ during the computation of NW. Suppose that

At node $\alpha \in V(P)$
  After receiving a message A^β from each child $\beta \in Ch(\alpha)$
    $W^\alpha \leftarrow u^\alpha + (\sum_{\beta \in Ch(\alpha)} A^\beta) - c^\alpha$
    If $W^\alpha \geq 0$ then
    {
      $\sigma_i \leftarrow 1$ for all $i \in res(\alpha)$
      Send W^α to $parent(\alpha)$
    }
    Else
    {
      $\sigma_i \leftarrow 0$ for all $i \in res(\alpha)$
      Send 0 to $parent(\alpha)$
    }

Figure 14.1. Bottom-up traversal: Computing welfare values.

Initialize: Root α_o sends W^{α_o} to each of its children.
For each $\alpha \in V(P) - \{\alpha_o\}$
  After receiving message A from $parent(\alpha)$
    //Case 1: $T^\alpha(P) \cap T(\tilde{S}) = \emptyset$.
    //Set σ_i's properly at α and propagate non-membership downward.
    If $\sigma_i = 0$, for all $i \in res(\alpha)$, or $A < 0$, then
    {
      $p_i \leftarrow 0$ and $\sigma_i \leftarrow 0$ for all $i \in res(\alpha)$
      send -1 to β for all $\beta \in Ch(\alpha)$
    }
    //Case 2: $T^\alpha(P) \cap T(\tilde{S}) \neq \emptyset$.
    //Compute cost shares and propagate minimum welfare value downward.
    Else
    {
      $A \leftarrow \min(A, W^\alpha)$
      For each $i \in res(\alpha)$
        If $u_i \leq A$, then $p_i \leftarrow 0$, else $p_i \leftarrow u_i - A$
      For each $\beta \in Ch(\alpha)$
        Send A to β
    }

Figure 14.2. Top-down traversal: Computing membership bits and cost shares.

$i \in res(\beta)$ and that $y_i(u)$ is the smallest W^α of any node α on the path from β to the root of $T(P)$. If $u_i \leq y_i(u)$, then removing i from the set of potential receivers does not change the set of nodes to which the content is delivered. If $u_i > y_i(u)$, then removing i from the set of potential receivers *does* change the set of nodes, and the resulting difference $\widetilde{NW} - \widetilde{NW}_{-i}$ is $y_i(u)$. The proofs of these facts are left as an exercise for the reader.

Theorem 14.2 *MC cost sharing requires exactly two messages per link. There is an algorithm that computes the cost shares by performing one bottom-up traversal of $T(P)$, followed by one top-down traversal.*[3]

More information about AMD for cost sharing can be found in Chapter 15.

14.3 Interdomain Routing

We now turn to the problem of interdomain routing. To provide reachability between hosts, the various ASes that make up the Internet must be interconnected. However, as

[3] The algorithm is provably optimal with respect to the number of messages sent but is not known to be optimal with respect to the maximum size of a message. However, the maximum size of a message is polynomial in $\max_l \text{size}(c(l))$ and $\max_i \text{size}(u_i)$ and polylogarithmic in $|P|$ and $|N|$, and the two local computations required at each node are fast and space-efficient.

we noted earlier, the ASes are economically independent entities (indeed, frequently competitors), and there is no trusted center to which they are all accountable that could assign interdomain routes. Thus, the ASes themselves must compute the routes in a distributed fashion. The route computation scheme must handle three problematic aspects of interdomain routing: (i) there is a large number of ASes; (ii) different ASes have different criteria for choosing one route over another, and these criteria may conflict; and (iii) the collection of ASes and the links between them change frequently. All of these factors make DAMD a highly suitable approach to interdomain routing.

We can formally define the interdomain-routing problem as follows. The network topology is defined in terms of the *AS graph* $G = (N, L)$, where each node in $N = \{1, \ldots, n\}$ corresponds to an AS in the Internet, and each link in L corresponds to a direct connection between a pair of neighboring ASes. Because routing protocols typically compute routes for each destination independently, we can choose a particular destination AS d and let P^i be the set of all loop-free paths from i to d in G that are not removed from consideration.[4] An *interdomain-routing protocol* allocates to each source node $i \in N$ a route $R_i \in P^i$.

We now describe this problem in greater detail, first from the networking perspective and then from the mechanism-design perspective.

14.3.1 Networking Perspective

From a networking or protocol-design point of view, any wide-area routing protocol must fulfill, to some extent, the following requirements:

- For reasons of trust, scale, and robustness, the routing protocol must be *distributed*, carried out by the ASes themselves.
- In order to reduce routing state, the routing protocol must use *destination-based forwarding*; i.e., all routing decisions must be based solely on a packet's destination. Each AS has a single next hop for the destination d, and the resulting *route allocation* $T_d = \{R_1, \ldots, R_n\}$ forms a confluent tree to the destination d.
- The routing protocol should be *adaptive*, adjusting to the current network topology without relying on any *a priori* topology information.
- The routing protocol should be *time-efficient*, *communication-efficient* (in its use of communication between the ASes), and *space-efficient* (in its use of the storage space that each individual AS needs in order to participate in the protocol).

These requirements are satisfied by each of the common routing-protocol designs – namely *distance-vector*, *link-state*, and *path-vector* – although these designs differ in their space requirements. However, interdomain routing has one additional requirement:

- The routing protocol must produce loop-free routes even while individual ASes make *autonomous* decisions about which routes are preferable.

[4] A path from i to d could be "removed from consideration" because it is *filtered* by i or one of i's neighbors or because of link or node failures.

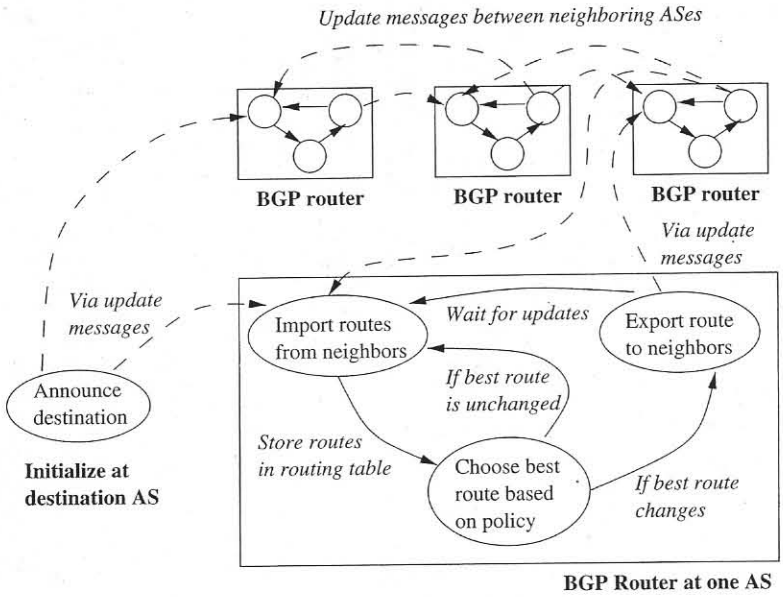

Figure 14.3. Route computation using a path-vector protocol.

Of the common routing-protocol designs, only path-vector satisfies this requirement. As a result, the current standard protocol for Internet interdomain routing, the Border Gateway Protocol (BGP), is a path-vector protocol. To see why path-vector is a suitable design choice, we describe BGP in more detail.

BGP allows adjacent nodes to exchange information through *update messages* that announce newly chosen routes (see illustration in Figure 14.3); a route announcement contains the entire path to the destination (the list of ASes in the path). A path-vector protocol (like most other routing protocols) computes routes to every destination AS independently; so we can focus on routes to a single destination d. The route-computation process is initialized when d announces itself to its neighbors by sending update messages. The rest of the routing tree to d is built recursively, as knowledge of how to reach d propagates through the network via subsequent update messages. We assume that the network is *asynchronous*, meaning that the arrival of update messages along selective links can be delayed.

The routing process at a particular node i has three stages that are iteratively applied:

(i) *Importing routes*: Routes to d are received via update messages from its neighbors. Node i has an *import policy* that specifies which of the routes it is willing to consider. All such importable routes are stored in an internal *routing table*. At any given time, i's internal routing table contains the latest importable routes.

(ii) *Route selection*: If there is more than one route to d in the routing table (i.e., more than one of i's neighbors has announced an importable route to d), node i must choose one (expressing a local preference over routes).

(iii) *Exporting routes*: Whenever there is a change to i's best route, it announces the newly selected route to some or all of its neighbors using update messages. Node i has

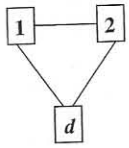

Figure 14.4. When AS 1 prefers route $12d$ to $1d$, and AS 2 prefers route $21d$ to $2d$, BGP (or any other path-vector protocol) can oscillate indefinitely.

an *export policy* that determines, for each neighbor j, which routes it is willing to announce to j at any given time.

AS autonomy is expressed through the freedom each AS has in choosing its routes, its import policy, and its export policy. These choices are based on local policy considerations and need not be coordinated with any other AS. The inclusion of the entire path in route announcements allows ASes to avoid routes with loops even while making otherwise arbitrary policy choices. Link-state or distance-vector routing protocols can avoid loops only if all ASes use the same criterion to choose routes and thus do not support autonomy.

One design requirement not explicitly listed here is *convergence*. Clearly the routing protocol should eventually enter a *stable* state in which every node prefers its currently chosen route to all others in its routing table, and all routing tables reflect the current route choices of its neighbors. Moreover, we would like the protocol to be *robust*, converging for every AS graph obtained by removing any set of nodes and links from the original instance.

Unfortunately, while the path-vector form of routing prevents loops, it does not ensure convergence; the routing announcements can enter a persistent oscillatory state. Consider the simple example depicted in Figure 14.4. Both nodes 1 and 2 would rather send traffic through the other source node than send traffic directly to the destination. Let us now simulate the execution of a path-vector protocol in the worst-case scenario: The computation is initialized when d announces itself to its two neighbors, nodes 1 and 2. At this point in time, these direct paths are the only routes available to d. Hence, 1 and 2 will choose the routes $1d$ and $2d$, respectively, and inform each other, via update messages, of their selected routes. Upon receipt of these update messages, nodes 1 and 2 will change their selected routes to, respectively, $12d$ and $21d$. However, now that none of the direct routes is being used, the indirect routes are no longer viable; so 1 and 2 are forced to return to their former routes $1d$ and $2d$, and the oscillation continues indefinitely. Note that, if the network had started with node 1's choosing and announcing $1d$ (having not yet seen an announcement of route $2d$), and then node 2 had chosen $21d$ (having seen route $1d$ announced before it chose and announced its own direct route $2d$), then no further changes would occur, and the network would be in a stable configuration; thus, convergence and oscillations can depend on timing.

A large body of networking research has addressed the problem of providing sufficient conditions on routing policies for the convergence of path-vector protocols. There is an inherent trade-off between the desired autonomy at the local level and robustness (in the sense defined above) at the global level. However, there is a known sufficient condition on policies, called *no dispute wheel*, that guarantees robust convergence

while allowing fairly expressive local routing policies. Any network instance on which a path-vector protocol might oscillate contains a dispute wheel and, more importantly, the absence of a dispute wheel means that the instance and every subinstance of it have unique stable route allocations to which the routing protocol converges, i.e., no dispute wheel implies robustness. The following definition provides an equivalent sufficient condition:

Definition 14.3 Define two relations on permitted routes:

(i) Let $R_1 \ominus_1 R_2$ iff R_1 is a subpath of R_2 that ends at d.

(ii) Let $R_1 \ominus_2 R_2$ iff $\exists i \in N : R_1, R_2 \in P^i$, and i prefers R_1 over R_2.

Let $\oslash = (\ominus_1 \cup \ominus_2)^*$ be the transitive closure of $\ominus_1, \ominus_2$. Note that $\oslash$ is inherently reflexive and transitive.

An interdomain-routing instance has *no dispute wheel* iff $R_1 \oslash R_2$ and $R_2 \oslash R_1$ together imply that R_1, R_2 start at the same node. (Informally, this is antisymmetry of $\oslash$ except that ties are allowed in valuations.)

Let us revisit the example in Figure 14.4. Recall that, on this instance, path-vector protocols may oscillate forever. This anomaly is manifested by the following dispute wheel:

$$1d \ominus_1 21d \ominus_2 2d \ominus_1 12d \ominus_2 1d.$$

So far, our discussion of interdomain routing has focused on traditional networking concerns. We now consider the problem from a mechanism-design perspective.

14.3.2 Mechanism-Design Perspective

The policy autonomy in BGP, which was previously allowed to be an arbitrary choice, can be seen as expressing a preference that an AS is selfishly trying to satisfy. To do so, we let each source node i have a private *valuation function* $v_i : S^i \to R_{\geq 0}$, where S^i is the set of all simple (noncyclic) routes from i to d in the complete graph we get by adding links to G.[5] The valuation function v_i specifies the "monetary value" of each route to source node i. We assume that $v_i(\emptyset) = 0$ and that, for all pairs of routes R_1 and R_2 through different neighboring nodes, $v_i(R_1) \neq v_i(R_2)$.[6] The routing policy of each node i is thus captured by v_i.

While each individual AS is trying to optimize its individual welfare, society as a whole has an interest in reaching a globally desirable outcome. While there are many goals one could choose, we shall focus here on *social-welfare maximization*. A route

[5] Because we do not assume that nodes know the network topology, we cannot assume that they can distinguish valid routes from invalid ones. Thus, the valuation functions are defined over the complete graph to model the possibility of nodes' announcing nonexistent routes.

[6] This assumption is consistent with current interdomain routing: Because at most one route to each destination can be installed in a router's forwarding table, nodes have some way to break ties, e.g., based on the next hop's IP address; so, valuations can be adjusted accordingly to match this. However, because only one route per neighbor is considered at a time, ties in valuation are permitted for routes through the same neighboring node.

allocation T_d maximizes the social welfare if

$$T_d = \text{argmax}_{T=\{R_1,\ldots,R_n\}} \sum_{i=1}^{n} v_i(R_i).$$

If we view a routing protocol from a mechanism-design perspective, it should satisfy the following two requirements:

- If implemented honestly, the protocol should maximize the social welfare.
- The protocol should be incentive-compatible, in that no AS is motivated to deviate from the actions it is asked to perform.

The precise definition of incentive compatibility needed in this setting depends on the nature of the solution concept (or economic equilibrium). We shall now discuss in detail the solution concept that we adopt for interdomain-routing mechanisms. Recall from Section 14.1 that DAMD poses inherently different strategic challenges from AMD, because, in the absence of a trusted center, the computation is performed by the strategic agents themselves. This allows the computational nodes to manipulate the mechanism strategically in ways other than "lying" about their private types. They can, for instance, alter the computation to their own benefit or refuse to pass messages if it suits their needs. In such a scenario, aiming for strategyproofness might be futile, because it is unlikely that there is a single computational behavior that is optimal no matter what the other agents do.

A more suitable solution concept is ex-post Nash equilibrium. The need to settle for ex-post Nash, rather than strategyproofness, can be viewed as the cost of distributing mechanism computation among the agents. We shall now formally define ex-post Nash in a distributed setting: Consider a computational network with n nodes and a set of possible outcomes O. Each node i has a private type $\theta_i \in \Theta_i$ and a utility function $u_i : O \times \Theta_i \to R$.

Definition 14.4 A *distributed mechanism* d^M is a 3-tuple $d^M = (\Sigma, g, s^M)$, where $\Sigma = (\Sigma_1, \ldots, \Sigma_n)$ is the *feasible strategy space* of the nodes, $g : \Sigma \to O$ is the *outcome function* computed by the mechanism, and $s^M = (s_1^M, \ldots, s_n^M) \in \Sigma$ is the *prescribed strategy*.

For every node i, $s_i^M \in \Sigma_i$ can be thought of as the algorithm that the mechanism designer intends i to execute. s_i^M is parameterized by the private type θ_i of the node i, with $s_i^M(\theta_i)$ specifying which actions node i should perform in every state of the mechanism and network, given that its type is θ_i.

Definition 14.5 A strategy profile $s^* \in \Sigma$ is an *ex-post Nash equilibrium* of a distributed mechanism $d^M = (\Sigma, g, s^M)$, if

$$u_i(g(s_1^*(\theta_1), \ldots, s_n^*(\theta_n)), \theta_i) \geq u_i(g(s_1^*(\theta_1), \ldots, s_i'(\theta_i), \ldots, s_n^*(\theta_n)), \theta_i)$$

for every node i, for every possible strategy $s_i' \in \Sigma_i$, for every possible θ_i, and for all possible private types $\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_n$ of the other nodes.

Although weaker than a dominant-strategy equilibrium, ex-post Nash equilibrium is a fairly strong solution concept; it does not require strategic agents to have any knowledge of or to make any assumptions about the private types of other agents. Contrast this with the standard Nash-equilibrium concept, in which agents are assumed to know the private types of other agents; in the interdomain-routing context, this would mean that ASes are assumed to know the local routing policies of other ASes, which is certainly unrealistic.

The ex-post Nash equilibrium solution concept is susceptible to collusion.[7] That is, while it is true that unilateral deviation by an AS from the prescribed strategy profile cannot benefit it, coordinated deviation by several ASes might prove to be beneficial to some. Therefore, if at all possible, we would like our mechanisms to ensure that no deviation by a group of ASes from the prescribed strategy profile is worthwhile. To achieve this, we introduce *collusion-proof ex-post Nash equilibria*. In a collusion-proof ex-post Nash equilibrium, no deviation by a group of agents can strictly improve the outcome of even a single agent in that group without strictly harming another.

14.3.3 A DAMD Approach: Combining the Two Perspectives

To achieve incentive-compatible interdomain routing, we must design a protocol that makes sense from both the networking and the mechanism-design perspectives. The networking requirements point to a path-vector framework combined with a class of routing preferences that guarantees convergence. Mechanism design requires that we incent agents to implement this routing protocol faithfully. Incentive compatibility is often achieved through payments; however, below we show that, under a reasonable set of assumptions about routing policies, one can achieve collusion-proof ex-post Nash equilibrium *without payments* simply by executing BGP.

14.3.3.1 Commercial Internet Routing and the Gao–Rexford Model

There are two types of business relationships that characterize most AS interconnections: *customer-provider* and *peering*. Customer ASes pay their provider ASes for connectivity, and peers are AS pairs that find it mutually advantageous to exchange traffic for free. One advantage of peering is that the two peers need not pay their respective providers to exchange traffic directly. An AS can be in many different relationships simultaneously: It can be a customer of one or more ASes, a provider to others, and a peer to yet others. These agreements are assumed to be relatively long-term contracts that are formed because of various external factors, e.g., traffic patterns and network sizes.

These business relationships naturally induce the following constraints on routing policies, known as the *Gao–Rexford constraints*:

No customer-provider cycles: Let G_{CP} be the digraph with the same set of nodes as G and with a directed edge from every customer to its provider. The Gao–Rexford constraints require that there be no directed cycles in this graph. This requirement is a natural economic assumption, because a cycle in G_{CP} implies that at least one AS is (indirectly) its own provider.

[7] The Nash equilibrium and dominant-strategy equilibrium concepts are also susceptible to collusion.

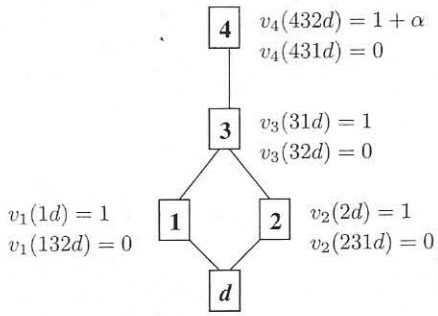

Figure 14.5. A routing instance that satisfies the Gao–Rexford constraints on which every path-vector protocol converges to a route allocation that is arbitrarily far from optimal.

Prefer customers over peers and peers over providers: A *customer route* is a route in which the next-hop AS is a customer. *Provider* and *peer routes* are defined similarly. Because, typically, customers pay providers for service, and peers exchange service for free, the Gao–Rexford constraints require that nodes always prefer (i.e., assign a higher value to) customer routes over peer routes, which are in turn preferred over provider routes.

Provide transit services only to customers: Transit service is carrying packets that originate and terminate at hosts outside the node. ASes are paid to carry customer packets but are not paid to carry peer or provider traffic. The Gao–Rexford constraints require that ASes not carry transit traffic between their providers and peers. Therefore, ASes should announce only customer routes to their providers and peers but should announce *all* of their routes to their customers.

These constraints ensure robustness without requiring coordination between ASes. In fact, if all ASes obey the Gao–Rexford constraints, then their valuations cannot induce a dispute wheel.

The Gao–Rexford constraints ensure robust convergence, but in general they do not guarantee that BGP converges to the social-welfare-maximizing route allocation. To see this, consider the example in Figure 14.5. Assume that d is a customer of 1 and 2, that 1 and 2 are customers of 3, that 3 is a customer of 4, and that $\alpha > 0$. Observe that this AS graph satisfies all the Gao–Rexford constraints. The unique stable route allocation (to $1, \ldots, 4$, respectively) is $\{1d, 2d, 31d, 431d\}$. However, the optimal route allocation is $\{1d, 2d, 32d, 432d\}$. This allocation will never be chosen by local decisions, because node 3 would much prefer routing through node 1, a route that is always available for it to choose. Therefore, because the value of α can be arbitrarily high, this implies that the route allocation computed by a path-vector protocol could be arbitrarily far from the welfare-maximizing route allocation.

This problem can be overcome by imposing the *policy-consistency* property.

Definition 14.6 Policy consistency holds iff, for every two adjacent nodes $i, j \in N$, and every two routes $\{Q, R\} \subseteq P^j$ such that $\{(i, j)Q, (i, j)R\} \subseteq P^i$[8] (in

[8] $(i, j)Q$ and $(i, j)R$ are the routes from i to d that have (i, j) as a first link and then follow Q and R, respectively.

particular, node i is not on Q or R),

$$\text{if } v_j(Q) \geq v_j(R), \text{ then } v_i((i,j)Q) \geq v_i((i,j)R).$$

Informally, policy consistency holds if, for every two neighboring nodes i, j, such that j is i's next-hop node on two routes, we have that, if j weakly prefers one route over another, then so must i. The policy-consistency property holds in the two most well studied special cases of interdomain routing. The first is the case in which the valuation of a route is solely a function of the route's next hop. (These are called "next-hop policies.") The second is the case in which there is some metric function that assigns a "length" to every link, and every valuation function prefers "shorter" routes (i.e., those with smaller total lengths in this metric). (These are called "metric-based policies.")

We are now ready to state, and prove, the following theorem.

Theorem 14.7 *If the Gao–Rexford constraints and policy consistency hold, then BGP converges to the social-welfare-maximizing route allocation and is incentive-compatible in collusion-proof ex-post Nash equilibrium (without any monetary transfer).*

PROOF We will actually prove a result that is stronger in two senses: First, we shall prove our result in the more general setting in which the valuation functions do not induce a dispute wheel, and policy consistency holds. Second, we shall prove that BGP actually converges to a solution (an allocation of routes) in which every AS gets its most desired route to the destination. That is, every AS will be assigned a route that maximizes its valuation function. We call this kind of route allocation a *locally optimal* solution. Observe that any locally optimal solution is also globally optimal in that it maximizes the total social welfare. Moreover, locally optimal solutions are *deviation-proof* in that there is no deviation by a group of agents that can strictly improve the outcome of even a single agent. This is far stronger than collusion-proof ex-post Nash equilibrium, which only requires that no deviation by a group of agents can strictly improve the outcome of a single agent in the group without strictly harming another agent in the group.

Because the Gao–Rexford constraints imply that there is no dispute wheel, we are assured (by the result mentioned in Section 14.3.1) that BGP will converge to a unique stable solution. We denote this solution by $T_d = \{S_1, \ldots, S_n\}$, where S_i is the route allocated to node i.

Lemma 14.8 *If the valuation functions do not induce a dispute wheel, and policy consistency holds, then BGP converges to a unique stable, locally optimal route allocation T_d.*

PROOF Consider a node $m \in N$. Let $R = u_k u_{k-1} \ldots u_i \ldots u_0$ be some loop-free route in P^{u_k}, such that $u_k = m$ and $u_0 = d$. By induction, we show for each $u_i \in R$ that S_i, the solution's route for node u_i in T_d, is at least as good as $R_i = u_i \ldots u_0$.

If $i = m$, then S_m is at least as good as R; because R and m were chosen arbitrarily, this establishes the local optimality of T_d.

Base case. $i = 0$. The induction hypothesis is trivially true, because the only route is the empty one.

Induction step. Assume that the induction hypothesis is true for u_{i-1}, i.e.,

$$v_{u_{i-1}}(S_{i-1}) \geq v_{u_{i-1}}(R_{i-1}). \tag{14.1}$$

Note that u_i does not lie on R_{i-1}, because R is loop-free.

Case I. Assume that $u_i \notin S_{i-1}$. Then extend S_{i-1} and R_{i-1} along the edge (u_i, u_{i-1}). $(u_i, u_{i-1})S_{i-1} \in P^{u_i}$; thus, from (14.1) and policy consistency, we have

$$v_{u_i}((u_i, u_{i-1})S_{i-1}) \geq v_{u_i}(R_i). \tag{14.2}$$

T_d is stable; so, S_i is at least as good as any other route at u_i; in particular,

$$v_{u_i}(S_i) \geq v_{u_i}((u_i, u_{i-1})S_{i-1}). \tag{14.3}$$

Combining (14.2) and (14.3) gives

$$v_{u_i}(S_i) \geq v_{u_i}(R_i),$$

which is the induction statement for u_i.

Case II. Assume that $u_i \in S_{i-1}$. We cannot use the policy-consistency argument as in Case I, because extending S_{i-1} to u_i creates a loop. This implies that $u_{i-1} \notin S_i$. Suppose that the induction statement is not true for i, i.e., that $v_{u_i}(R_i) > v_{u_i}(S_i)$. Then $R_i \ominus_2 S_i$. Because $u_{i-1} \notin S_i$ but $u_i \in S_{i-1}$, it must be that $S_i \ominus_1 S_{i-1}$. From the induction hypothesis, $S_{i-1} \ominus_2 R_{i-1}$, and, because $R_i = (u_i, u_{i-1})R_{i-1}$, $R_{i-1} \ominus_1 R_i$. Therefore, we have a cycle in the relation $\oslash$; in particular, we can say that $R_i \oslash R_{i-1}$ and $R_{i-1} \oslash R_i$, but these routes do not start at the same node. This violates the no-dispute-wheel property and shows that the assumption that $v_{u_i}(R_i) > v_{u_i}(S_i)$ leads to a contradiction. Therefore, $v_{u_i}(R_i) \leq v_{u_i}(S_i)$, which is the induction statement for u_i. (Recall that there are no ties in valuations.) □

Remark 14.9 Lemma 14.8 holds for every subinstance of the AS graph, because both the Gao–Rexford constraints and policy consistency hold for every subinstance.

Remark 14.10 No dispute wheel implies a unique collusion-proof ex-post Nash solution to which BGP converges. Hence, we are not concerned with the standard problem that arises when multiple equilibria exist, namely whether nodes select the same equilibrium.

14.4 Conclusion and Open Problems

In this chapter, we have reviewed the work that has been done on distributed algorithmic mechanism design, in which the presence of strategic computational agents introduces

new incentive and computational challenges for distributed computing. In particular, we have presented in detail some of the known results about DAMD for interdomain routing, which is the best motivated and most extensively studied problem in the area. There are at least two interesting directions for further research.

First, there is the general question of which other problems in networked computation are amenable to the approaches explored in this chapter. Several good candidates have been proposed, i.e., web caching, peer-to-peer file sharing, overlay-network construction, and distributed task allocation. Although both distributed algorithms and incentive compatibility have been considered in the literature about these problems, the results have not been pulled together into a coherent DAMD theory. The construction of such a theory remains a worthy goal.

Second, there are many questions about interdomain routing that have not been fully answered. There is still no complete characterization of the conditions under which BGP converges robustly. ("No dispute wheel" is sufficient but not known to be necessary.) Similarly, the conditions under which collusion-proof ex-post Nash equilibrium is reached simply by executing BGP have not been characterized completely. (Again, the Gao–Rexford and policy-consistency conditions presented in this chapter are sufficient but not known to be necessary.) In fact, necessary and sufficient conditions on AS graphs and routing policies have not yet been obtained for ex-post Nash equilibrium, even if we ignore collusion and allow payments. Both policy consistency and local optimality play an essential role in the main result presented in this chapter, and little is known about what can be obtained without them. In general, the network complexity of BGP is open, even in cases when convergence is assured.

14.5 Notes

Given the distributed and autonomous nature of Internet users, it is no surprise that the networking and distributed-systems literature provides some of the earliest applications of game theory and mechanism design to computer-science problems. These themes were first explored in an early series of papers from Columbia University, e.g., Ferguson (1989), Hsiao and Lazar (1988), Kurose et al. (1985), Kurose and Simha (1989), Mazumdar and Douligeris (1992), and Yemini (1981), which were followed by contributions from Miller and Drexler (1988a, 1988b), Sanders (1986, 1988a, 1988b), and others (Kelly, 1997; Kelly et al., 1998; La and Anantharam 1997; Murphy and Murphy, 1994; Mackie-Mason and Varian, 1995; Shenker, 1990, 1995). Because networking problems are inherently distributed, and network protocols must have reasonable network complexity, these papers were actually early forerunners of DAMD.

Nisan and Ronen were the first to combine algorithmic and economic concerns in a new area of study for which they coined the term "algorithmic mechanism design," and this book is largely an outgrowth of their seminal paper Nisan and Ronen (2001). The extension of AMD to DAMD was first explored in Feigenbaum et al. (2001), which considered the multicast cost-sharing problem described in Section 14.2 and articulated the notion of network complexity; the DAMD agenda was more broadly described soon thereafter in Feigenbaum and Shenker (2002). Subsequent work on DAMD for multicast cost sharing can be found in, e.g., Archer et al. (2004), Adler and

Rubenstein (2002), Fiat et al. (2002), and Feigenbaum et al. (2003). In particular, a generalization of Theorem 14.1 is proven in Feigenbaum et al. (2003).

Distributed VCG computation and the importance of ex-post Nash equilibria in DAMD were first presented by Parkes and Shneidman (2004) and developed further by Petcu et al. (2006).

The BGP specification can be found in Rekhter et al. (2006). The fact that BGP may not converge if there are no constraints on the AS graph or the domains' routing policies was first observed by Varadhan et al. (2000). The example of BGP divergence in Figure 14.4 and the proof that "no dispute wheel" guarantees robust convergence are presented in Griffin et al. (2002). Abstract properties of path-vector protocols are developed in, e.g., Griffin et al. (1999, 2003), Sobrinho (2005). The Gao–Rexford conditions and their implications were first studied in Gao and Rexford (2001) and further developed in, e.g., Gao et al. (2001). Partial results on the network complexity of BGP can be found in, e.g., Karloff (2004).

DAMD was first applied to interdomain routing by Feigenbaum et al. (2005b), who devised a BGP-based algorithm for lowest-cost routing. Computational manipulation by ASes and ex-post Nash equilibrium in BGP-based, lowest-cost routing was first studied by Shneidman and Parkes (2004). Hardness results for more general classes of routing policies can be found in Feigenbaum et al. (2005a, 2006b). A positive result about BGP-based, incentive-compatible routing under the Gao–Rexford and policy-consistency conditions is given in Feigenbaum et al. (2006a) and is the direct precursor of the result presented in Section 14.3. Sobrinho was the first to study policy constraints that guarantee optimality, both global and local; a result that is similar to (but weaker than) Lemma 14.8 is presented in Sobrinho (2005).

For basic background on Internet routing, see Kurose and Ross (2005), Peterson and Davie (2003), or other networking textbooks.

Acknowledgments

We thank Vijay Ramachandran and Rahul Sami for many helpful discussions of interdomain routing. The work of the first author was supported in part by ONR grants N00014-01-1-0795 and N00014-04-1-0725, NSF grant 0428422, HSARPA grant ARO-1756303, and US–Israeli BSF grant 2002065. The work of the second author was supported in part by US-Israeli BSF grant 2002065 and by Israeli Science Foundation grant 169/03. The work of the third author was supported in part by NSF grant 0428422.

Bibliography

A. Archer, J. Feigenbaum, A. Krishnamurthy, R. Sami, and S. Shenker. Approximation and collusion in multicast cost sharing. *Games Econ. Behav.*, 47(1):36–71, 2004.

M. Adler and D. Rubenstein. Pricing multicasting in more practical network models. In *13th Symp. on Discrete Algorithms*, pp. 981–990, ACM/SIAM, New York/Philadelphia, 2002.

J. Feigenbaum, D.R. Karger, V.S. Mirrokni, and R. Sami. Subjective-cost policy routing. In Xiaotie Deng and Yinyu Ye, editors, *First Workshop on Internet and Network Economics*, LNCS 3828:174–183, Springer, Berlin, 2005a.

J. Feigenbaum, A. Krishnamurthy, R. Sami, and S. Shenker. Hardness results for multicast cost sharing. *Theor. Comput. Sci.*, 304(1–3):215–236, 2003.

J. Feigenbaum, C.H. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *J. Comput. Syst. Sci.*, 63(1):21–41, 2001.

J. Feigenbaum, C.H. Papadimitriou, R. Sami, and S. Shenker. A BGP-based mechanism for lowest-cost routing. *Distr. Comput.*, 18(1):61–72, 2005b.

J. Feigenbaum, V. Ramachandran, and M. Schapira. Incentive-compatible interdomain routing. In *7th Conference on Electronic Commerce*, pp. 130–139, ACM, New York, 2006a.

J. Feigenbaum, R. Sami, and S. Shenker. Mechanism design for policy routing. *Distr. Comp.*, 18(4):293–305, 2006b.

J. Feigenbaum and S. Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *6th Intl. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pp. 1–13, ACM, New York, 2002.

A. Fiat, A.V. Goldberg, J.D. Hartline, and A.R. Karlin. Competitive generalized auctions. In *34th Symposium on Theory of Computing*, pp. 72–81, ACM, New York, 2002.

D.F. Ferguson. *The Application of Microeconomics to the Design of Resource Allocation and Control Algorithms*. Ph.D. Thesis, Columbia University, 1989.

L. Gao, T.G. Griffin, and J. Rexford. Inherently safe backup routing with BGP. In *20th INFOCOM*, pp. 547–556, IEEE, Pistacaway, 2001.

L. Gao and J. Rexford. Stable Internet routing without global coordination. *IEEE/ACM Trans. Networking*, 9(6):681–692, 2001.

T.G. Griffin, A.D. Jaggard, and V. Ramachandran. Design principles of policy languages for path vector protocols. In *SIGCOMM '03: Proc. 2003 Conf. Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 61–72, ACM, New York, 2003.

T.G. Griffin, F.B. Shepherd, and G. Wilfong. Policy disputes in path-vector protocols. In *7th Intl. Conf. on Network Protocols*, pp. 21–30, IEEE Computer Society, Los Alamitos, 1999.

T.G. Griffin, F.B. Shepherd, and G. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Trans. Networking*, 10(2):232–243, April 2002.

M.-T. Hsiao and A.A. Lazar. A game theoretic approach to decentralized flow control of markovian queueing networks. In Pierre-Jacques Courtois and Guy Latouche, editors, *Performance 87', Proc. 12th IFIP WG 7.3 Intl. Symp. Comp. Performance Modelling, Measurement, and Evaluation*, pp. 55–73, North-Holland, Amsterdam, 1988.

H. Karloff. On the convergence time of a path-vector protocol. In *Proc. 15th Symp. on Discrete Algorithms*, pp. 605–614, ACM/SIAM, New York/Philadelphia, 2004.

F.P. Kelly. Charging and rate control for elastic traffic. *Euro. Trans. Telecommuncations*, 8:33–37, 1997.

F.P. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: Shadow prices, proportional fairness, and stability. *J. Oper. Res. Soc.*, 49(3):237–252, 1998.

J.F. Kurose and K.W. Ross. *Computer Networking: A Top Down Approach Featuring the Internet*. Addison-Wesley, 2005.

J.F. Kurose and R. Simha. A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Trans. Comp.*, 38(5):705–717, 1989.

J.F. Kurose, M. Schwartz, and Y. Yemini. A microeconomic approach to decentralized optimization of channel access policies in multiaccess networks. In *5th Intl. Conf. on Distr. Comp. Sys.*, pp. 70–77, IEEE Computer Society, Los Alamitos, 1985.

R.J. La and V. Anantharam. Optimal routing control: Game theoretic approach. In *36th Conf. on Decision and Control*, pp. 2910–2915, IEEE, Piscataway, 1997.

J.K. Mackie-Mason and H. Varian. Pricing the Internet. In Brian Kahin and James Keller, editors, *Public Access to the Internet*, pp. 269–314, MIT Press, Cambridge, 1995.

R.R. Mazumdar and C. Douligeris. A game theoretic approach to flow control in an integrated environment. *J. Franklin Inst.*, 329(2):383–402, 1992.

M.S. Miller and K.E. Drexler. Incentive engineering: For computational resource management. In Bernardo A. Huberman, editor, *The Ecology of Computation*, pp. 231–266. North-Holland, Amsterdam, 1988a.

M.S. Miller and K.E. Drexler. Markets and computation: Agoric open systems. In Bernardo A. Huberman, editor, *The Ecology of Computation*, pp. 133–176. North-Holland, Amsterdam, 1988b.

J. Murphy and L. Murphy. Bandwidth allocation by pricing in ATM networks. In *Broadband Communications II: Proc. 2nd Intl. Conf.*, pp. 333–351, Elsevier, Amsterdam, 1994.

N. Nisan and A. Ronen. Algorithmic mechanism design. *Games Econ. Behav.*, 35(1):166–196, 2001.

D.C. Parkes and J. Shneidman. Distributed implementations of Vickrey-Clarke-Groves mechanism. In *3rd Intl. Joint Conf. on Autonomous Systems and Multiagent Systems*, pp. 261–268, IEEE Computer Society, Los Alamitos, 2004.

A. Petcu, B. Faltings, and D.C. Parkes. MDPOP: Faithful distributed implementation of efficient social choice problems. In *Proc. 5th Intl. Joint Conf. Autonomous Agents and Multiagent Systems*, ACM Press, New York, NY, 2006.

L.L. Peterson and B.S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann, 2003.

Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271, January 2006.

B.A. Sanders. An incentive compatible flow control algorithm for fair rate allocation in computer/communication networks. In *6th Intl. Conf. on Distr. Comp. Syst.*, pp. 314–320, IEEE Computer Society, Los Alamitos, 1986.

B.A. Sanders. An asynchronous, distributed flow control algorithm for rate allocation in computer networks. *IEEE Trans. Comp.*, 37(7):779–787, 1988.

B.A. Sanders. An incentive compatible flow control algorithm for rate allocation in computer networks. *IEEE Trans. Comp.*, 37(9):1067–1072, 1988.

S. Shenker. Efficient network allocations with selfish users. In Peter J. B. King, Isi Mitrani, and Rob Pooley, editors, *Performance '90, Proc. of the 14th IFIP WG 7.3 Intl. Symp. on Computer Performance Modelling, Measurement and Evaluation*, pp. 279–285, North-Holland, Amsterdam, 1990.

S. Shenker. Making greed work in networks: A game-theoretic analysis of switch service disciplines. *IEEE/ACM Trans. Networking*, 3(6):819–831, 1995.

J. Shneidman and D.C. Parkes. Specification faithfulness in networks with rational nodes. In *23rd Symp. on Princ. Distributed Computing*, pp. 88–97, ACM, New York, 2004.

J.L. Sobrinho. An algebraic theory of dynamic network routing. *IEEE/ACM Trans. Networking*, 13(5):1160–1173, 2005.

K. Varadhan, R. Govindan, and D. Estrin. Persistent route oscillations in inter-domain routing. *Comput. Networks*, 32(1):1–16, March 2000.

Y. Yemini. Selfish optimization in computer networks. In *20th Conf. Decision and Control*, pp. 281–285, IEEE, Pistacaway, 1981.

Exercises

14.1 Recall from Chapter 9 that, in a *second-price Vickrey auction* of a single item, the item is sold to the highest bidder, and the price that the winner pays is the second-highest bid. Consider a network in which there is one bidder at each node, and the nodes lie on a cycle. As in Section 14.2, we assume that there is no trusted center to implement an algorithm but that there is a central enforcer that

can implement the outcome decided upon by the agents and can impose severe penalties if the agents do not agree on an outcome. Give a distributed algorithm for computing the winner and the price in a second-price Vickrey auction on such a network that has the following properties: (i) it is incentive-compatible in ex-post Nash equilibrium; (ii) it requires no more than two messages to cross each link; and (iii) each message is at most $O(\log m + \log n)$ bits long, where m is the highest bid, and n is the number of bidders. Prove that your algorithm satisfies these three properties

14.2 Prove that, in the MC multicast cost-sharing mechanism, there is a single "largest" receiver set that maximizes NW.

14.3 Prove the correctness of the algorithm given in Section 14.2.2 for computation of MC cost shares.

14.4 A strategyproof mechanism is group strategyproof (GSP) if no coalition of deviating agents can achieve an outcome that is at least as good for all deviating agents and strictly better for at least one. For each of the MC and SH multicast cost-sharing mechanisms, either prove that it is GSP or provide a counterexample.

14.5 Consider a single-item, ascending-price auction with "jump bids." Type θ_i denotes agent i's value for the item. Bids are associated with a "bid price." In round t, the auctioneer announces an "ask price" p^t that is $\epsilon > 0$ above the highest bid received so far. Any agent can bid in round t, as long as the bid is at some price at or above p^t. The provisional winner is the agent with the current highest bid (breaking ties at random). The auction terminates when no agent bids at the current ask price, and the item is then sold to the provisional winner at its final bid price. The information state (p^t, x^t) defines the current ask price p^t and provisional winner $x^t \in \{1, \ldots, n\}$. The following is a straightforward bidding strategy that determines what agent i will do in state (p, x): If $p \leq \theta_i$ and $x \neq i$, then bid p; otherwise, do not bid. Prove that this strategy profile is an ex-post Nash equilibrium but not a dominant-strategy equilibrium.

14.6 Prove that policy consistency is satisfied if all ASes use next-hop policies, or if all use metric-based policies.

14.7 Give an interdomain-routing instance (i.e., an AS graph in which one AS is identified as the destination, each edge is identified as a peer edge or a customer-provider edge, and a valuation function is given for each source AS) that does not contain a dispute wheel but also does not satisfy the Gao–Rexford constraints. Explain why the Gao–Rexford constraints are not satisfied by this instance.

14.8 Prove that, in the interdomain-routing problem, it is NP-hard to find a route allocation that comes within a constant factor of the maximum social welfare if no restrictions are made on the valuation functions.