Yale University
Department of Computer Science

Incentive-Compatible Interdomain Routing

Joan Feigenbaum Vijay Ramachandran
Michael Schapira

YALEU/DCS/TR-1342
May 2006

This work is supported in part by the U.S. Department of Defense (DoD) University Research Initiative (URI) program administered by the Office of Naval Research (ONR) under grants N00014–01–1–0795 and N00014–04–1–0725. An extended abstract was previously published as [4].

Incentive-Compatible Interdomain Routing

Joan Feigenbaum*
Department of Computer Science
Yale University
New Haven, CT, USA
jf@cs.yale.edu

Vijay Ramachandran†
Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ, USA
vijayr@cs.stevens.edu

Michael Schapira‡
Department of Computer Science
The Hebrew University
Jerusalem, Israel
mikesch@cs.huji.ac.il

Abstract

The routing of traffic between Internet domains, or *Autonomous Systems* (ASes), a task known as *interdomain routing*, is currently handled by the Border Gateway Protocol (BGP) [17]. Using BGP, autonomous systems can apply semantically rich routing policies to choose interdomain routes in a distributed fashion. This expressiveness in routing-policy choice supports domains' autonomy in network operations and in business decisions, but it comes at a price: The interaction of locally defined routing policies can lead to unexpected global anomalies, including route oscillations or overall protocol divergence (see, *e.g.*, [20]). Networking researchers have addressed this problem by devising constraints on policies that guarantee BGP convergence without unduly limiting expressiveness and autonomy (see, *e.g.*, [7, 8]).

In addition to taking this engineering or "protocol-design" approach, researchers have approached interdomain routing from an economic or "mechanism-design" point of view. It is known that lowest-cost-path (LCP) routing can be implemented in a truthful, BGP-compatible manner [3] but that several other natural classes of routing policies cannot [2, 5]. In this paper, we present a natural class of interdomain-routing policies that is more realistic than LCP routing and admits incentive-compatible, BGP-compatible implementation. We also present several positive steps toward a general theory of incentive-compatible interdomain routing.

This work is supported in part by the U.S. Department of Defense (DoD) University Research Initiative (URI) program administered by the Office of Naval Research (ONR) under grants N00014–01–1–0795 and N00014–04–1–0725. An extended abstract was previously published as [4].

*Supported in part by ONR grants N00014–01–1–0795 and N00014–04–1–0725, NSF grants 0208972, 0219018 and 0428422, and HSARPA grant ARO–1756303.

†Work done while at the International Computer Science Institute (ICSI), Berkeley, CA, USA. Supported by ONR grant N00014–01–1–0795, by NSF grant CNS-0524139, and by the Stevens Technogenesis program.

‡Supported in part by ONR grant N00014–01–1–0795. Work done in part while visiting Yale University.

1 Introduction

The Internet is comprised of many separate administrative domains known as *Autonomous Systems* (ASes). Routing occurs on two levels, intradomain and interdomain, implemented by two different sets of protocols. Intradomain-routing protocols, such as OSPF [15], route packets within a single AS. Interdomain routing, currently handled by the Border Gateway Protocol (BGP) [17], routes packets between ASes. It has been studied by computer scientists for many years from an engineering or "protocol-design" perspective and recently from an economic or "mechanism-design" perspective as well. Combining algorithmic and economic considerations in the study of interdomain routing is very natural, because the many separate domains that make up the Internet really are independent economic agents that must jointly execute a distributed algorithm in order to choose routes.

In their seminal paper [16], Nisan and Ronen gave the following formulation of interdomain routing as a 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 given a *payment* for carrying *transit* traffic, which is traffic neither originating from nor destined for that AS. It is through these costs and payments that consideration of "incentive compatibility" was introduced to the interdomain-routing framework, which, as currently realized by BGP, does not explicitly consider incentives. The goal in [16] was to optimize the use of network bandwidth by routing packets along *lowest-cost paths* (LCPs) and to do so with a *truthful mechanism* that can be computed in *polynomial time*. Nisan and Ronen observed that the Vickrey-Clarke-Groves (VCG) mechanism, well known to be truthful, solves the LCP mechanism-design problem and can be computed in polynomial time. Many researchers have followed up on Nisan and Ronen's original work, including Feigenbaum, Papadimitriou, Sami, and Shenker [3], who showed that lowest-cost paths and VCG payments could be computed in a "BGP-compatible" fashion, *i.e.*, computed by a distributed algorithm that requires only small modifications to the (already universally deployed) Border Gateway Protocol. In this paper, we continue the study of BGP-compatible, truthful computation of interdomain routes and payments that was begun in [3].

Although it was viewed as a step forward in our attempt to understand the interplay of engineering, algorithmics, and economics in interdomain routing, the work in [3] was by no means a fully satisfactory solution. In particular, one of the valuable features of BGP is that it allows ASes to choose interdomain routes according to semantically rich policies that meet their operational and business requirements; LCP routing is just one example of a valid policy, and, in practice, many ASes do not use it [1]. Thus, it is natural to ask whether more realistic, expressive interdomain-routing policies admit truthful, BGP-compatible computation of routes and payments. Previous work on this question has been discouraging: Negative results have been obtained for general policy routing [5], for "subjective-cost" policy routing [2], for "forbidden-set" policy routing [2], and for "next-hop" policy routing [5]. The next-hop case (defined below) admits a satisfactory centralized-algorithmic solution, but the stringent requirements put forth in [5] for a satisfactory distributed-algorithmic solution cannot be met.

In this paper, we provide the first example of a class of policies that is more realistic than LCP and that admits incentive-compatible, BGP-compatible computation of routes and payments, to wit: next-hop policies that obey the Gao-Rexford conditions for global stability. We now proceed to describe these policies and then outline other contributions of this paper; the latter contribute to a general theory of incentive-compatible interdomain routing.

The *next hop* of a route is the source AS's immediate neighbor along that route. An AS has a *next-hop policy* if it decides among available routes to a destination based solely on the routes' next hops. Because ASes do not control packet forwarding beyond the neighboring AS to which traffic is initially sent, it is realistic to express route preferences based on next hops alone. However, uncoordinated and unconstrained local configuration of next-hop policies can produce routing instability [10, 20].

Gao and Rexford [7] proposed constraints on policies that guarantee route stability without global coordination. They assume that two types of business relationships exist between neighboring pairs of ASes: *customer-provider*, in which one AS purchases connectivity from another, and *peering*, in which two ASes agree to carry transit traffic to and from each other's customers, *e.g.*, to shortcut routes through providers. (These relationships accurately represent today's commercial Internet; see [13].) These relationships naturally induce route preferences. Gao and Rexford formalized these preferences (we review the formalization in Sec. 3.1) and proved that they induce stable routing if there are *no customer-provider cycles* (*i.e.*, no AS is an indirect customer of itself). This requirement is realistic, because it is unlikely that a large Internet provider would purchase connectivity from a smaller ISP in its own customer hierarchy.

We show that this realistic class of policies admits incentive-compatible, BGP-compatible computation of routes and payments. Furthermore, we are able to give positive results for more general classes of policies. We identify three conditions that together form a sufficient constraint on policies to permit the computation of welfare-maximizing routes by any path-vector protocol (including BGP). We show that, if any of these conditions is violated, the *price of anarchy* [14]—a measure of how far from optimal the computed routing tree is, with respect to welfare maximization—for path-vector routing is unbounded. We also exhibit an incentive-compatible algorithm that, while not space-efficient, computes payments and routes for any class of routing policies that obeys the first two of these three conditions and, through its payments, enforces that nodes obey the third condition. This general-case algorithm is not subject to any of the methods of rational manipulation formulated by Shneidman and Parkes [18].

Our space-efficient implementation for the realistic class of policies discussed above is a special case of the general-case algorithm; we also discuss another space-efficient special case, that of *metric-based valuations*, that is a generalization of lowest-cost routing.

The remainder of the paper is organized as follows. In Sec. 2, we formally define the interdomain-routing problem and review some necessary notation. We then, in Sec. 3, give an incentive-compatible, BGP-compatible algorithm to compute routes and payments for next-hop policies that obey the Gao-Rexford conditions. Following that, we discuss the three conditions on policies that permit welfare-maximizing route computation in Sec. 4 and give an algorithm for the general case in Sec. 5. We present open questions and conclude in Sec. 6.

2 Preliminaries

We begin this section by formally defining the interdomain-routing problem and providing some useful notation. We then review the Border Gateway Protocol (BGP), the standard protocol used for interdomain routing today.

2.1 Problem Statement

In the interdomain-routing problem, we are given an AS graph $G = (N, L)$ that describes the network topology. The set of nodes N corresponds to the ASes in the graph. Because routes are computed independently for each destination, without loss of generality, we assume that N consists of n source nodes $\{1, \ldots, n\}$ and a destination node d. The set of links L corresponds to connections between ASes. Let $L^i \subset 2^L$ be the set of all *simple* routes (*i.e.*, routes with no loops) from i to d in G.

An instance $I = (G, \mathcal{P}, \mathcal{V})$ of the *interdomain-routing problem* is defined by an AS graph G, a set of *permitted routes* $\mathcal{P}(i) = P^i \subset L^i$ for each node $i \in [n]$, and the *valuation function* $\mathcal{V}(i) = v_i : P^i \to \mathbb{R}_{\geq 0}$ of each node. Every set P^i contains the paths in L^i that are not removed from consideration by either i itself or i's neighbors. Every valuation function v_i specifies the "monetary value" of each route $R \in P^i$ from node i. We assume that $v_i(\emptyset) = 0$, *i.e.*, no route is worth nothing, and that, for all pairs of routes R_1 and R_2 through different neighboring nodes, $v_i(R_1) \neq v_i(R_2)$, *i.e.*, there are no ties in valuations.[1] The *routing policy* of each node i is thus captured by v_i and P^i: The only routes considered for i are those in P^i, and preference among these routes is given by the valuation function v_i.

The goal is to allocate to each source node $i \in [n]$ a route $R_i \in P^i$. The resulting *route allocation* $T_d = \{R_1, \ldots, R_n\}$ should form a confluent tree to the destination d. Furthermore, we are interested in route allocations that maximize the "total social welfare" of the nodes, *i.e.*, we want to find an allocation satisfying

$$T_d = \mathrm{argmax}_{T=\{S_1,\ldots,S_n\}} \sum_{i=1}^{n} v_i(S_i).$$

Incentive compatibility is introduced into this problem by paying nodes for their contribution to the routing tree in the hope of incentivizing truthful behavior. Therefore, in our version of the problem, we assume, as in [18], that N contains one more node, called *the bank*, that is in charge of distributing a payment $s_i(T_d)$ to each source node i based on the path allocation T_d.

We define the *utility function* of each node i, $u_i : \prod_i P^i \to \mathbb{R}$, to be $u_i(T_d) = v_i(R_i) + s_i(T_d)$. Although the global goal is to maximize the total social welfare, every rational node i would only be interested in maximizing its own utility, even if this comes at the expense of not achieving the global goal. An algorithm (protocol) is *truthful* if it is in the best interest of each node to reveal its true valuation function to the algorithm. An algorithm is *incentive-compatible* (with respect to some notion of equilibrium) if it is in the best interest of each node to comply with all the algorithm's instructions (with respect to the same notion of equilibrium); compliance includes, but is not limited to, providing truthful input of valuation functions.

A distributed setting such as ours poses an inherently different challenge for the design of incentive-compatible mechanisms (see [3, 18]) than a centralized one. This is because the computation is performed by the strategic agents themselves and not by a reliable third party. In this paper, we focus on achieving incentive compatibility in *ex-post Nash equilibrium*, which has been argued to be most appropriate for distributed-mechanism computation [18]; using this equilibrium concept enables the consideration of several forms of rational manipulation other than lying about inputs (see Sec. 5.2 for a detailed discussion).

We are interested in efficient, distributed, and incentive-compatible welfare-maximizing algorithms for the interdomain-routing problem. We require our algorithms to assume no prior knowledge of the nodes of the topology of the network.

2.2 Notation

First, we present some notation for the representation of routes. A *simple* route is a finite sequence of consecutive links from a source node to the destination node that contains no loops (cycles). *All routes in this paper are simple unless stated otherwise.* We say that node i is in route R (or write $i \in R$) if i participates in one of the links in R.

[1] This assumption is consistent with BGP and the model of interdomain routing in [10]: Because at most one route can be installed in a router's forwarding table to each destination, nodes have some deterministic 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.

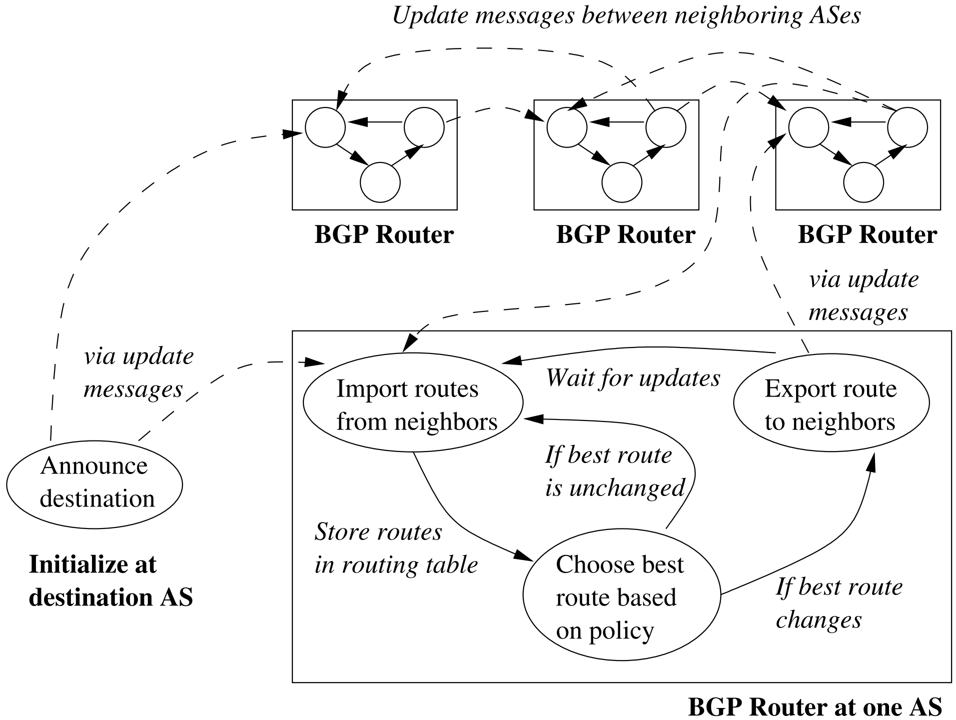

Figure 1: Route computation using BGP.

If R is a route from j (its *source*) to the destination d, and i is a node that is not in R and is adjacent to j in G, we denote by $(i, j)R$ the route that has (i, j) as a first link and then follows R to the destination. If j and k are intermediate nodes on a route R, we denote by $R_{[j,k]}$ the subpath of R from j to k.

Throughout this paper, we will consider sub-instances of the interdomain-routing problem obtained by removing one node from the AS graph G. For every node i, we denote by G^{-i} the subgraph of G that contains all nodes in N except i and all links in L except those i participates in. We can now define $I^{-i} = (G^{-i}, \mathcal{V}', \mathcal{P}')$ to be a sub-instance of the original interdomain-routing instance I, in which the AS graph is G^{-i} and, for each node $j \neq i$, $\mathcal{P}'(j) = \{R \in \mathcal{P}(j) \mid i \notin R\}$, *i.e.*, any route containing i is removed from the permitted-route set of j, and $\mathcal{V}'(j)$ is $\mathcal{V}(j)$ restricted to the sub-domain $\mathcal{P}'(j)$, *i.e.*, the valuation of a permitted route in I^{-i} is identical to the valuation of that route in I. We denote by T_d^{-i} a welfare-maximizing route allocation for I^{-i}.

2.3 Routing with BGP

The Border Gateway Protocol (BGP) [17] belongs to the family of *path-vector protocols*, the abstract properties of which were studied in [9]. A sketch of how BGP computes routes is shown in Fig. 1. The basic idea is that a routing tree to a given destination is built, hop-by-hop, as knowledge of how to reach that destination propagates through the network. Communication between nodes takes place through *update messages* that announce chosen routes.

The process is initialized when some destination AS d announces itself to its neighbors by sending update messages. Then, each node i iteratively establishes routes to d by:

1. importing, via update messages, routes to d chosen by neighbors[2] and storing the routes in a *routing table*;

2. choosing the best route from i to d (through a neighbor of i) among those available in the routing table based on local routing policy; and

3. if there is a change to i's best route, exporting the newly selected route to all of i's neighbors using update messages.[3]

At any given time, each node's (internally stored) routing table contains the route updates received from its neighbors, and each node is assigned at most one best route based on its policy. (A node may not have a best route if it has not yet received any updates or if its neighbors have *withdrawn* their routes, *e.g.*, because of network failures). We assume that the network is asynchronous; so, it is possible that the network delays the arrival of update messages along selective links.

Path-vector routing has several advantages. First, because the only routes considered are those announced by neighbors, the protocol enforces the requirement that route choices form a confluent tree. Second, each node is able to maintain its autonomy by making its route choice based on local, expressive routing policies. Third, changes in the network due to the addition or subtraction of nodes or links can be announced through update messages, and routers can use alternate routes stored in the routing table to adapt quickly. Fourth, because entire paths are announced, nodes can check for loops and exclude them from routing tables.

Because BGP is currently the standard protocol for Internet interdomain routing, we desire algorithms that are *BGP-compatible*, *i.e.*, that can be implemented with only small modifications to BGP.

3 A Realistic, Incentive-Compatible Routing Model

In this section, we present an incentive-compatible, BGP-compatible algorithm for the interdomain-routing problem (defined in the previous section) when valuation functions belong to a restricted but realistic class of policies: next-hop routing that obeys the Gao-Rexford conditions for global stability. We first define this class of policies. We then present the algorithm and discuss its properties.

3.1 Policies for the Commercial Internet

Packets are forwarded based on destination alone; therefore, it is sensible for ASes to use *next-hop policies*—those that only consider the immediate neighbor along a route—because an AS actually has no control over packets once they are forwarded to a neighboring AS. We formally define these policies as follows.

Definition 3.1. If $i \in [n]$, define $\mathrm{neighbors}(i) = \{j \in N \mid (i,j) \in L\}$, *i.e.*, the set of nodes adjacent to i.

Definition 3.2. If $R' \in L^j$ and $R = (i,j)R'$, then define the *next hop* on R to be $\mathrm{next}(R) = j$. Node $i \in [n]$ has a *next-hop valuation function* v_i iff there exists a function $f_i : \mathrm{neighbors}(i) \to \mathbb{R}_{\geq 0}$ such that, for every route $R \in P^i$, $v_i(R) = f(\mathrm{next}(R))$.

If all nodes have next-hop valuation functions, we say that "the instance uses next-hop policies." Next-hop policies are semantically rich enough to permit global routing instability (see Sec. 4.1); therefore, we require additional constraints on policies. One realistic and well-studied set of constraints, which we discuss

[2]Some neighbors may refuse to send particular routes.

[3]Again, nodes may not send certain routes to certain neighbors.

in this section, assumes that some business hierarchy underlies the AS graph and that policies are based on the economic nature of this hierarchy.

Huston's study of the commercial Internet [13] suggests two types of business relationships that characterize AS inter-connections: Pairs of neighboring nodes have either a *customer-provider* or a *peering* relationship. Customer nodes pay their provider nodes for connectivity—access to Internet destinations through the provider's links and advertisement of customer destinations to the rest of the Internet. Peers are nodes that find it mutually advantageous to exchange traffic for free among their respective customers, *e.g.*, to shortcut routes through providers. A node can be in many different relationships simultaneously: It can be a customer of one or more nodes, a provider to others, and a peer to yet other nodes. These agreements are assumed to be longer-term contracts that are formed because of various external factors, *e.g.*, the traffic pattern between two nodes.

Intuitively, these business relationships naturally induce routing policies. Gao and Rexford [7] formally modeled these relationships and policies with the following three conditions.

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. We demand that there be no directed cycles in this graph. If this requirement is met, we say that "the AS graph contains no customer-provider cycles." This demand is a natural economic assumption, because, if there is a cycle in G_{CP}, then a node is indirectly its own provider.

Prefer customers to peers and peers to providers: A *customer route* is a route in which the next-hop AS is a customer. *Provider* and *peer routes* are defined similarly. We require that nodes always prefer (*i.e.*, assign a higher value to) customer routes over peer routes, which are in turn preferred to provider routes. This also has an obvious economic justification given the financial agreement for each relationship.

Provide transit services only to customers: Nodes do not always carry *transit traffic*—traffic that originates and terminates at hosts outside the node. Nodes are obligated (by financial agreements) to carry transit traffic to and from their customers, but nodes do not carry transit traffic among only providers and peers. Therefore, nodes should share only customer routes with their providers and peers but should share *all* of their routes with their customers.

It was proven in [6, 7] that, if all nodes obey these conditions, enforced naturally by Internet economics, BGP predictably converges to a stable routing tree, even after node and link failures. Later work [8, 19] showed that the Gao-Rexford conditions are only one class of policies that prevent routing anomalies; we will discuss the more general characterization in Sec. 4.2 below.

Using the terminology and notation of Sec. 2, we formally define the Gao-Rexford conditions as follows:

Definition 3.3. The *Gao-Rexford conditions* hold iff the AS graph contains no customer-provider cycles, and, for all nodes $i \in [n]$, the following hold for all pairs of nodes $\{j,k\} \subset \mathrm{neighbors}(i)$ and for all pairs of routes $\{R_j, R_k\} \subset P^i$ such that $\mathrm{next}(R_j) = j$ and $\mathrm{next}(R_k) = k$:

1. If j is a customer and k is not, then $v_i(R_j) > v_i(R_k)$. If j is a peer and k is a provider, then $v_i(R_j) > v_i(R_k)$. (The remaining cases are implied by symmetry.)

2. If neither j nor k is a customer, then $(j,i)R_k \notin P^j$ and $(k,i)R_j \notin P^k$, because i does not share R_k with j or R_j with k. If j is a customer, then, whatever i's relationship to k, R_j is shared with k, and R_k is shared with j. Thus, $(k,i)R_j \in P^k$ if permitted by k, and $(j,i)R_k \in P^j$ if permitted by j.

3.2 A BGP-Compatible Algorithm

The following algorithm is a straightforward extension to BGP that computes routes and payments for incentive-compatible, welfare-maximizing routing when policies are next-hop based and obey the Gao-Rexford conditions described above in Sec. 3.1.

The algorithm essentially computes best routes using BGP but adds extra information to update messages so that nodes can compute the mechanism's payments. This information is also stored in nodes' routing tables, requiring one extra bit of storage for every transit AS on an imported route. These bits are used to determine the next hop of the best *k-avoiding route*—the best route in I^{-k}—for every transit node k on the best route for each node in I. The next hops are used directly in computing payments and can be stored using one extra row in the routing table, denoted L_i below.

The extra bit per transit node in each row of the routing table and the extra row used to store the next hops require a constant-factor increase in the space complexity of the original BGP; a similar amount of extra storage was used by the algorithm described in [3] for lowest-cost-path routing. We use the term *BGP-compatible* to mean that the algorithm has the same basic structure as BGP and that it is "space-efficient," in that it requires only a modest increase to the storage requirement of the original BGP. This is consistent with use of the term in [3].

Computation of best routes and k-avoiding next hops is triggered when nodes receive update messages, just as in BGP (see Sec. 2.3). Update-message processing is divided into two cases: (I) the message is from the most valued neighbor that has yet sent a message, in which case the route contained in the message is chosen as the best route; and (II) the message is not from the most valued neighbor that has yet sent a message, in which case the extra bits in the message are used to update the choices of the best k-avoiding next hops. Unlike BGP, if node x chooses node y as its next hop, an update message is still sent from x back to y; this extra message is used to send availability to y of k-avoiding routes through x and is processed using case (II).

Setting: An instance of the interdomain-routing problem with next-hop policies obeying the Gao-Rexford conditions. As in Def. 3.2, we assume that there exists at each node $i \in [n]$ a function $f_i : \mathrm{neighbors}(i) \to \mathbb{R}_{\geq 0}$, such that $v_i(R) = f_i(\mathrm{next}(R))$.

Outcome: A route allocation $T_d = \{R_1, \ldots, R_n\}$ that forms a confluent tree to d, such that

$$T_d = \mathrm{argmax}_{T=\{S_1,\ldots,S_n\}} \sum_{i=1}^{n} v_i(S_i).$$

Structure of Update Messages: An update message m sent by node i contains a route $R_m \in P^i$ and, for every $k \in R_m$ ($k \notin \{i, d\}$), a bit $B_m(k)$. $B_m(k) = 1$ if i has, in its routing table, a k-avoiding route to d, *i.e.*, some route $R \in P^i$ such that $k \notin R$. These bits are used to correctly populate the list L_i, defined below, that is used to compute the mechanism's payments.

Storage at Each Node: Each node i has a routing table Y_i indexed by neighbors of i. If $j \in \mathrm{neighbors}(i)$, then $Y_i(j)$ is the update message sent by node j, so that at most one advertised route is stored per neighbor. Initially, $Y_i(j) = \emptyset$ for all j. Each node i also has a list L_i: Assume the current best route at i is R_i; if $k \in R_i$ is a transit node ($k \notin \{i, d\}$), then $L_i(k) = \mathrm{next}(R')$, where R' is the best k-avoiding route in i's routing table. $L_i(k)$ will be used, at the end of the algorithm, to compute the component of the payment to node k that is attributable to node i, denoted s^i_k. Fig. 2 shows an example of the storage at each node.

Start: AS d sends update message $m = (d, \emptyset)$ to all neighbors.

Dest.	$L_z(2)=1$	$L_z(4)=2$	$L_z(5)=1$	$\rightarrow L_z$: best k-avoiding next-hop ASes for transit k on z's best route
d	AS 2	AS 4 $B_2(4)=1$	AS 5 $B_2(5)=0$	$\rightarrow R_2$, the route chosen by neighbor AS 2; *z's current best route* $\rightarrow B_2$, the bit vector sent with update from neighbor 2
d	AS 1	AS 3 $B_1(3)=0$	AS 5 $B_1(5)=1$	$\rightarrow R_1$, the route chosen by neighbor AS 1 $\rightarrow B_1$, the bit vector sent with update from neighbor 1

Figure 2: An example routing table for source node z using the algorithm from Sec. 3.2.

Update-Message Processing: Let $m = (R_m, B_m)$ be the update message received at node i from $j \in \text{neighbors}(i)$. If $(i,j)R_m \notin P^i$ and $\text{next}(R_m) \neq i$, then discard the message. Otherwise, $(i,j)R_m \in P^i$ or $\text{next}(R_m) = i$, and the update message should be stored in the routing table so that $Y_i(j) = (R_m, B_m)$.

(Case I) If $\text{next}(R_m) \neq i$ and

$$f_i(j) = \max_{\{j' \in \text{neighbors}(i) | Y_i(j) \neq \emptyset\}} f_i(j'),$$

i.e., j is the most valued neighbor that has sent an update message, then either R_m is a new best route to d (*i.e.*, R_m is the new R_i) or the neighbor exporting R_m has an updated bit vector B_m. Reset L_i to empty and, for each $k \in R_m$ such that $k \neq d$, do the following to repopulate L_i: If $B_m(k) = 1$, then set $L_i(k) = j$; if $B_m(k) = 0$ or $k = j$, then:

1. Let $A = \text{neighbors}(i) - \{j\}$ and let

   $$a = \text{argmax}_{\{a' \in A | Y_i(a') \neq \emptyset\}} f_i(a')$$

   be the most valued node in A. Let $(R_a, B_a) = Y_i(a)$ be the routing-table entry for a.

2. If $k \notin R_a$, then set $L_i(k) = a$.

3. If not, $k \in R_a$. If $B_a(k) = 1$, then set $L_i(k) = a$.

4. If $L_i(k)$ has still not been set, then repeat at (1) with $A = A - \{a\}$. Discontinue repeat if $A = \{a\}$, *i.e.*, there would be no nodes left in A.

Finally, set $R_i = (i,j)R_m$.

(Case II) If $\text{next}(R_m) = i$ or

$$f_i(j) \neq \max_{\{j' \in \text{neighbors}(i) | Y_i(j) \neq \emptyset\}} f_i(j'),$$

i.e., j is not the most valued neighbor that has sent an update message, then, for each current transit node $k \in R_i$ ($k \notin \{i, d\}$), set $L_i(k) = j$ if j has a k-avoiding route and j is more valued than $L_i(k)$, the current best k-avoiding next hop; *i.e.*:

1. $f_i(j) > f_i(L_i(k))$; and either

2a. $k \in R_m$ and $B_m(k) = 1$; or

2b. $k \notin R_m$.

If any changes were made to L_i in either of the cases above (including any time Case I was triggered), then send update messages $m' = (R_i, B'_m)$ to all neighbors of i, where $B'_m(k) = 1$ if $L_i(k) \neq \emptyset$ and $B'_m(k) = 0$ if $L_i(k) = \emptyset$. (If R_i is a non-customer route and neighbor n is also a non-customer, then the update message $(\emptyset, \emptyset)$ should be sent to comply with the Gao-Rexford conditions, implying a withdrawal of the previous route. Note that, in Lem. 3.7 below, we prove that a withdrawal will never happen.)

Payment Computation: Once the algorithm converges, the bank obtains from each node i the *payment component* $s^i_k = f_i(\mathrm{next}(R_i)) - f_i(L_i(k))$ for every $k \in R_i$ ($k \notin \{i, d\}$), which is the component of the total payment to k that is attributable to i. The bank then disburses to each node k a payment $s_k = \sum_{i \neq k} s^i_k$.

We next investigate the truthfulness and correctness of the algorithm. We show that the algorithm converges, at which time each node i has a valid, utility-maximizing route R_i to d and, for each $k \in R_i$ ($k \notin \{i, d\}$), the next hop of the best route in G^{-k}, $L_i(k)$, that is used in the computation of payment components s^i_k.

3.3 Truthfulness and Correctness

We define the payment to each node to be

$$s_k = \sum_{i \neq k} v_i(R_i) - \sum_{i \neq k} v_i(R_i^{-k}), \tag{1}$$

where R_i is the route allocated to i in T_d, and R_i^{-k} is the route allocated to i in T_d^{-k}.

Our mechanism then belongs to the family of *Vickrey-Clarke-Groves* (VCG) mechanisms. A classic result of Green and Laffont [11] states that a truthful pricing mechanism maximizing a social-welfare function of the form $V(T_d) = \sum_{i=1}^{n} v_i(R_i)$ must be a VCG mechanism, with payments expressible as

$$p_k = \sum_{i \neq k} v_i(R_i) - h_k(T_d^{-k}), \tag{2}$$

in which $h_k(\cdot)$ is an arbitrary function of T_d^{-k}. In particular, this means that every strategic agent's payment must depend solely on the other agents. Note that, if

$$h_k(T_d^{-k}) = \sum_{i \neq k} v_i(R_i^{-k})$$

in (2), then $p_k = s_k$.

Intuitively, the payment to each node i is the increase in the social welfare of the other nodes caused by i's participation in the algorithm. The key observation is that these payments can be "broken down" into components computed by the different nodes (in a distributed fashion). Loosely speaking, node i's component in the payment to node j corresponds to j's contribution to i's welfare—the difference in the values i assigns to the paths he gets with and without j. These components are computed during the algorithm, and the final payment is the sum of payment components computed once the algorithm converges.

Definition 3.4. The *payment component* for j attributable to i is

$$s^i_j = v_i(R_i) - v_i(R_i^{-j}),$$

and the *payment* to each node k is

$$s_k = \sum_{i \neq j} s^i_k.$$

It is easy to verify that the payment s_k in Def. 3.4 is the same as that in (1). At the end of the algorithm, each node i has enough information to compute s^i_j for all transit nodes j: Because preferences are next-hop based, $s^i_j = v_i(R_i) - f_i\left(L_i(j)\right)$, where f_i is the next-hop valuation as in Def. 3.2, and $L_i(j)$ is the next hop of the best j-avoiding route computed by the algorithm. Payment components must only be computed for transit nodes: If j is not a transit node on i's best route, *i.e.*, $j \notin R_i$, then $R_i = R_i^{-j}$, and $s^i_j = 0$.

VCG payments guarantee the truthfulness of the algorithm. In Sec. 5.2, we show that (with minor modifications) our algorithm is immune to all types of rational manipulation as formulated by Shneidman and Parkes [18]; this means our algorithm is incentive-compatible with respect to ex-post Nash equilibrium. The algorithm is BGP-compatible because it has the same structure as BGP and requires only a constant-factor increase in space complexity.

Theorem 3.5. *The algorithm in Sec. 3.2 is truthful and BGP-compatible.*

Proof. As discussed above, payments to nodes have the form of VCG payments; VCG payments guarantee truthfulness [11]. We must now show that the algorithm is BGP-compatible. In addition to the routing-table storage required by the original BGP, this algorithm requires, at node i, storage of:

1. the bit $B_m(j)$ for every $j \in R_m$ sent in an update message m stored at i; and
2. the next hops on the currently best known k-avoiding routes for every $k \in R_i$, where R_i is the current best route to d.

This requires one additional bit per transit AS, per row (update message) in the routing table and one additional row to store the next hops. This amounts to a constant-factor increase in space complexity and fulfills our requirements for BGP compatibility. □

The following theorem implies the correctness of the algorithm.

Theorem 3.6. *Regarding the algorithm in Sec. 3.2 on instances with next-hop valuations obeying the Gao-Rexford conditions:*

(C1) *the algorithm converges;*

(C2) *the output T_d is optimal (welfare-maximizing); and*

(C3) *the nodes $L_i(k)$ are indeed the next hops of the optimal routes for i in G^{-k}.*

Proof. We will show that the Gao-Rexford conditions imply (C1) and that adding next-hop valuations implies (C2). These are special cases of more general results, which are discussed in Sec. 4.2–4.4. The welfare-maximizing routing tree output by the algorithm with this class of policies has the additional property that the routes allocated to the nodes are not only globally optimal, but also locally optimal (best with respect to each node's valuation function). Therefore, if nodes comply with the algorithm's instructions, they should receive their highest valued routes. This result is also true for more general classes of policies; see Sec. 4.5 below. The proof of (C3) is particular to this algorithm and this class of policies.

To prove (C1), we must show that our algorithm will stop sending update messages along every edge in the network. The Gao-Rexford conditions imply convergence of simple path-vector protocols (SPVPs) like BGP—discussed in Sec. 2.3—on instances and sub-instances [6, 7]; however, our algorithm differs slightly from SPVPs. In both algorithms, an update message is sent from i to j when a new best route is chosen at i. In SPVPs, this message either (1) contains the new route (if i can export its choice to j), or (2) contains a withdrawal (if i cannot export its choice to j). In our algorithm, (1) still occurs, but (2) does not; in particular, our algorithm does not send or process withdrawal messages. However, the following lemma shows that this is irrelevant: For valuations obeying the Gao-Rexford conditions, withdrawal messages are never sent.

Lemma 3.7. *If, at some time, node a sends node i an update message (R_m, B_m) such that $R_m \neq \emptyset$,* i.e.*, node a exports a route to node i, and we assume there are no failures, then at any future time, there will exist a route R_a in i's routing table, such that* $\text{next}(R_a) = a$.

Informally, this lemma means that once a node exports a usable route to a neighbor (where "usable" means allowed by the Gao-Rexford conditions), any route chosen by the node will be a usable route for that neighbor. Therefore, route withdrawals are unnecessary; routes are only replaced with new (usable) routes.

Proof. Changes to the routing table are update-driven. A change, due to a new update or withdrawal, will only be sent if a switches from R_m to some other route R_a. We must show that, in this case, an update message with R_a is sent to i, and a withdrawal is not sent.

If a is a provider of i, then a will export R_a to i. Therefore, we can assume, without loss of generality, that a is a peer or customer of i; then R_m must be a customer route of a, or it would not have been sent to i. If a switches to R_a because $v_a(R_a) > v_a(R_m)$, then R_a must also be a customer route, and it will be exported to i. If not, then R_m must have been withdrawn. (If it was replaced, next-hop policies dictate that $v_a(R_a) = v_a(R_m)$, and that route will be exported to i.) In this case, its customer $c = \text{next}(R_m)$ switched to a route that was filtered; but, this new route must be a non-customer route at c. Because it is less valued than the customer route $R_{m[c,d]}$, that switch must have also happened because of a withdrawal, and these same arguments apply. This could continue downstream to d, but the last link must be a customer route that is always available; this leads to a contradiction. ☐

Given Lem. 3.7, the convergence implied by the Gao-Rexford conditions for SPVPs also applies to our algorithm, because the dynamics of update messages (for route choices in the original instance I) are the same as that of SPVPs. However, our algorithm also finds next hops in sub-instances I^{-k}, where $k \in [n]$; to do so, it sends update messages whenever the availability of k-avoiding routes changes (*i.e.*, a change in the list L_i). These messages are not used in SPVPs, so we must show that they eventually stop as well.

First, note that the Gao-Rexford conditions hold for sub-instances if they hold for the original instance; therefore, a unique, stable routing tree exists for each sub-instance, and route withdrawals are unnecessary. Second, because valuations are next-hop based, only the availability of a k-avoiding route through a given neighbor needs to be known, not the route itself. (This is why the algorithm only sends a bit vector of availability.) But, because routes are never withdrawn, once a neighbor indicates k-avoiding-route availability, a k-avoiding route through that neighbor will always be available in the future. Because there are a finite number of neighbors, k-avoiding-route availability can only improve a finite number of times. Thus, at some point along every edge, update messages will no longer be sent for this reason.

Therefore, we have shown that at some point, update messages will no longer be sent (either for route choices in I or next hops in I^{-k}); thus, the algorithm converges, proving (C1).

To prove (C2), we show that the welfare-maximizing routing tree for the instance, $O = (O_1, \ldots, O_n)$, is stable—*i.e.*, for every node i and every $j \in \text{neighbors}(i)$, $v_i(O_i) \geq v_i((i,j)O_j)$ (see Def. 4.1). It is clear (and implied by [6, 7]) that our algorithm converges to a stable routing tree. Because the Gao-Rexford conditions imply that there is a unique stable routing tree [6, 7], the tree computed by the algorithm must be welfare-maximizing.

We show this by contradiction: Assume that the welfare-maximizing tree O is not stable; then there is some node i with a neighbor j such that

$$v_i(O_i) < v_i((i,j)O_j). \tag{3}$$

Construct the routing tree $O' = (O_1{}', \ldots, O_n{}')$ as follows. Let $O'_i = (i,j)O_j$ and $O'_j = O_j$. (Note that $i \notin O_j$, otherwise $v_i((i,j)O_j) \not> v_i(O_i)$ because the route would not be simple.) For $k \notin \{i,j,d\}$, if $i \notin O_k$, let $O'_k = O_k$. If $i \in O_k$, then let $O'_k = O_{k[k,i]}(i,j)O_j$.

The latter is possible because $O'_k \in P^k$. Let $m \in O'_k$ be the neighbor of i such that $\mathrm{next}(O'_m) = i$. If m is a customer of i, then O'_i will be exported to m, and O'_m will be exported and extended to k because the Gao-Rexford conditions imply these links are customer links. If m is not a customer of i, then j must be a customer of i; if (3) is true, then the Gao-Rexford conditions imply this. Therefore, the route O'_i must be exported to m; because no relationships have changed, if O_i was exported by m, so will O'_i.

In the tree O', (3) implies that node i has higher welfare. Because valuations are next-hop based and the nodes routing through i continue to route through i, the valuations of no other nodes have changed. Therefore, O' has higher welfare than O, contradicting the welfare maximization of O. This must mean that O is stable, which completes the proof of (C2).

Note that the above argument can also be used to prove that the unique stable routing tree, which is welfare-maximizing, assigns each node its most valued route. Assume this is not true: Some node does not receive its most valued path R; in (3), replace $(i,j)O_j$ with R, and the contradiction follows. Therefore, the routing tree found by the algorithm is essentially "optimal" in the global (welfare-maximizing) and local sense. This fact will be used below, and is a special case of Thm. 4.13. For the remainder of this proof, we will use "optimal" to mean both globally and locally optimal because of this equivalence; for a full discussion of this for more general cases of policies, see Sec. 4.5.

To prove (C3), we shall require the following four lemmas.

Lemma 3.8. *If j is the optimal next hop for i, and, for some $k \in [n]$, j has a k-avoiding route, then the next hop of the optimal k-avoiding route at i is also j.*

This lemma justifies the step in the algorithm that immediately sets k-avoiding next hops whenever an update message containing a new best route is received.

Proof. Assume that j is not the optimal k-avoiding next hop; then, because j has a k-avoiding route, there must be some other node a with a better k-avoiding route R_a^{-k}. Because of next-hop valuations, this implies $f_i(a) > f_i(j)$. But, because a is not optimal for i with k present, this implies that a must not have a usable route to d when k is present. Thus, when k is present, a chooses a route R_a through k but does not export it to i. (Otherwise, removing k would make no difference.) This means that $v_a(R_a) > v_a(R_a^{-k})$, and $\mathrm{next}(R_a)$ and i both must be non-customers of a (only non-customer routes can be filtered by Gao-Rexford policies). But this means that R_a^{-k} must also be a non-customer route at a because it is less valued than R_a. In this case, R_a^{-k} would not be exported to i either, contradicting the possibility that a has a usable k-avoiding route for i. □

Lemma 3.9. *If node i has not received an update message from neighbor a, then either node a's route in I^{-k} (for any $k \in [n]$) cannot be exported to i, or node a has no route in I^{-k}.*

This lemma means that neighbors with k-avoiding routes permitted at i will send update messages to i; information from neighbors that do not send update messages to i is irrelevant in computing payment components.

Proof. If a is routing through i, then a will send an update message if it has any k-avoiding routes available. Thus, without loss of generality, we can assume that a is not routing through i.

If a has not sent an update message to i because it has not learned any paths to d, then a also has no k-avoiding routes to d.

The remaining case is that a has not sent an update message to i because it cannot share its route R_a with i. In this case, i must not be a customer of a, and $\text{next}(R_a)$ is also not a customer of a. If $k \notin R_a$, then R_a is a k-avoiding route, but a cannot export it to i because $\text{next}(R_a)$ and i are both non-customers.

If $k \in R_a$, then a may choose a different route R_a^{-k} in I^{-k}. If R_a^{-k} is a non-customer route, then it is still unusable by i, which accounts for an update not being sent. If R_a^{-k} is a customer route, then it must not be available to a when k is present, otherwise a would choose it over the non-customer route R_a. But this is not possible, because every link $(u, w) \in R_a^{-k}$ is a customer link, including the last link to d. This means the route must be exported up the chain of providers to a at all times, which leads to a contradiction; therefore, R_a^{-k} cannot be a customer route at a, which makes it unusable to i. □

Lemma 3.10. *If $k \notin R_a$, the route allocated to a by the algorithm for the original instance I, and $(i, a)R_a \in P^i$, then there exists a route $R_a^{-k} \in P^a$ such that $(i, a)R_a^{-k} \in P^i$ for the sub-instance I^{-k}.*

This lemma has to do with k-avoiding-route availability. Even though a node may choose a k-avoiding route as its best route for I, it may be that downstream changes prevent it from choosing that route in the sub-instance I^{-k} in which k is removed; in fact, it is possible that no k-avoiding route is available. This lemma excludes this possibility. The algorithm uses this fact to populate the lists L_i.

Proof. If no node $j \in R_a$ chooses a different path (other than R_j) when k is not present, then R_a itself is a k-avoiding path usable by i. If some downstream node j switches to a different path R'_j when k is removed, then the path $R_{a[a,j]}R'_j$ should be usable at i, unless it is filtered somewhere between j and i.

Assume this happens. The relationships of nodes between j and i have not changed: Because these nodes originally propagated R_j, they would also propagate R'_j; therefore, j itself must filter R'_j. This means that R'_j must be a non-customer route, and the node upstream of j towards a must also be a non-customer. But because R_j was not filtered, it must be a customer route. Because $v_j(R_j) > v_j(R'_j)$ in this case, j would never have switched to R'_j upon removal of k unless R_j was filtered downstream of j. However, this same argument applies to all downstream nodes (which must all be customers); because the last link adjacent to d must be a customer link and the direct route is always exported, this leads to a contradiction. □

Lemma 3.11. *Given some fixed k, it is not possible for $L_i(k) = j$ and $L_j(k) = i$ at the same time.*

In the algorithm, nodes send their next hops k-avoiding-route availability. This lemma precludes the possibility that two nodes choose each other as their k-avoiding next hop.

Proof. If i is a customer of j, then the only routes exported to j are customer routes. Therefore, if i exports a k-avoiding route R to j such that j considers $(j, i)R$ its best k-avoiding route, R is a customer route at i. This implies $f_i(\text{next}(R)) > f_i(j)$; so, $L_i(k) \neq j$. The same argument works, by symmetry, if j is a customer of i.

If i and j are peers, then the only routes they can share are customer routes. Assume that each node chooses the other as a best k-avoiding next hop; then each must have a customer route exported to the other. But those customers would be better choices for k-avoiding next hops, contradicting the assumption. □

We are now ready to prove (C3). By (C1), the algorithm converges, and when it does, by (C2), the route choice is optimal; thus every node i receives a route through its most highly valued neighbor j. From

Lemma 3.7, we know that, once i learns a route through j, it always has a current update message from j; update messages are sent whenever a change to the best route or the best k-avoiding next hop (for any k) occurs.

Once the algorithm converges, consider the entry $L_i(k)$ for each k. The algorithm populates these entries in the following way. $L_i(k) = j$ if $B_j(k) = 1$ or $k \notin (i,j)R_j$; *i.e.*, $L_i(k) = j$ if j has a k-avoiding route. By Lemma 3.8, if j has a k-avoiding route for some k, then this setting is optimal.

If $B_j(k) = 0$ and $k \in (i,j)R_j$, then j does not have a k-avoiding route. In this case, the algorithm sets $L_i(k)$ to be the most valued neighbor m that has sent an update message (R_m, B_m) in which either $k \notin R_m$ or $B_m(k) = 1$. First, we show that the algorithm chooses the most valued neighbor; then we show that the neighbor has a k-avoiding route.

By Lemma 3.9, we must only consider neighbors that send update messages as candidates for the optimal k-avoiding next hop; thus the algorithm is not excluding viable choices by examining update messages alone. The entry for $L_i(k)$ is set in either Case I or Case II of the algorithm. If set in Case I, the entry is the most valued neighbor because the latest update messages are scanned in decreasing order of valuation; the scan is accurate because Case I resets L_i and then examines the most recent update messages. If set in Case II, the entry is the most valued because $L_i(k)$ is only set when an update message is received from a neighbor more valued than the previous $L_i(k)$, which was either set by a Case I or Case II message; thus, at convergence, the entry will represent the most valued neighbor with a k-avoiding route.

By Lemma 3.10, if $k \notin R_m$, then m must have a k-avoiding route usable by i, and the algorithm does not need to scan B_m. If $B_m(k) = 1$, the update message from m itself states that m has a k-avoiding route. Therefore, the neighbor chosen for $L_i(k)$ certainly has a k-avoiding route.

Finally, Lemma 3.11 and the Gao-Rexford conditions assure us that the next hops chosen at different nodes do not create routing loops; thus they are consistent with a tree. ☐

4 Towards a General Theory of Incentive-Compatible Interdomain Routing

In Sec. 3, we presented a realistic class of policies that admits incentive-compatible, BGP-compatible computation of routes and payments. However, many of our techniques apply to other classes of policies. In this section and the next, we present several positive steps toward a general theory of incentive-compatible interdomain routing.

The algorithm in Sec. 3.2 is able to find a welfare-maximizing, or globally optimal, route allocation, even though routes are chosen through local decisions. Local decision making cannot always achieve a globally optimal solution; the class of policies described in Sec. 3 satisfy specific constraints that allow this. In this section, we describe three constraints on routing policies. For each, we give an example in which removing the constraint results in an unbounded *price of anarchy*, meaning that the result of nodes' acting rationally but selfishly is arbitrarily worse than the result of a centralized, optimal computation. In other words, local decisions using a BGP-compatible protocol may not find a welfare-maximizing route allocation if one or more of the constraints are not satisfied. We then show, however, that these three constraints together form a sufficient condition for policies to admit distributed, incentive-compatible computation of welfare-maximizing routes. Later, in Sec. 5, we present an algorithm that is not space-efficient but computes welfare-maximizing routes and VCG payments for any class of policies that obeys these three constraints.

4.1 Stability, Robustness, and the Price of Anarchy

Path-vector protocols like BGP function much like an iterative game, because, at each step of the protocol, ASes examine the routes chosen by their neighbors and make local decisions as to which routes are best. Convergence to some equilibrium is thus an implicit goal of the protocol. Informally, a route allocation is *stable* if no node prefers changing his allocated route to a route that follows one of its neighbors' allocated routes. A stable route allocation can be regarded as a Nash equilibrium.

Definition 4.1. A route allocation $T_d = \{R_1, ..., R_n\}$ is stable iff, for every node i,

$$v_i(R_i) = \text{argmax}_{\{(i,j)R_j \in P^i | (i,j) \in L \wedge i \notin R_j\}} v_i((i,j)R_j).$$

However, a stable route allocation that is reached by local, selfish decision making may not be welfare maximizing. The *price of anarchy* [14] measures how bad selfish computation can be.

Definition 4.2. In an instance I, let

$$W_{\text{selfish}} = \min_{\text{stable } T_d = \{R_1, \ldots, R_n\}} \sum_{i=1}^{n} v_i(R_i)$$

be the minimum total social welfare obtained by a stable routing tree, and let

$$W_{\text{opt}} = \max_{T_d = \{R_1, \ldots, R_n\}} \sum_{i=1}^{n} v_i(R_i)$$

be the maximum total social welfare (over all routing trees). The *price of anarchy* of path-vector routing on I is

$$\frac{W_{\text{opt}}}{W_{\text{selfish}}}.$$

To design a welfare-maximizing path-vector protocol—a distributed protocol in which decisions are made locally and selfishly—we must find conditions under which the price of anarchy is 1. We develop such a condition in the remainder of this section.

In addition to stability, network operators want routing to respond to topology changes due to failures. Stability even in the presence of failures is formally defined as follows.

Definition 4.3. An instance of the interdomain-routing problem is *robust* iff, for every sub-instance obtained by removing any set of nodes and links from the original graph, there exists a unique stable route allocation to which a path-vector protocol converges from any initial route allocation.

4.2 Dispute Wheels

There is an inherent trade-off in achieving the desired autonomy and policy expressiveness at a local level and robustness at the global level [8]. Early work conjectured that only shorest-paths routing might be provably stable [20]. However, Griffin, Shepherd, and Wilfong [10] presented a sufficient condition on policies that guarantees robust convergence while allowing policies broader than shortest-path routing.

This condition is called *no dispute wheel*. A dispute wheel is essentially a representation of a set of nodes and their routing policies (*i.e.*, ordinal preferences on paths) that induce a routing anomaly. A network instance on which BGP might oscillate contains a dispute wheel; thus, the absence of a dispute wheel in an

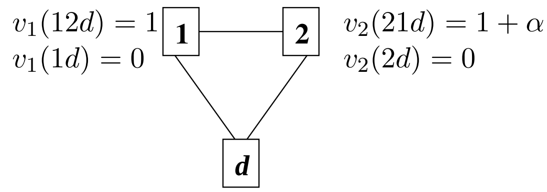

Figure 3: A routing instance with a dispute wheel.

instance guarantees that it will never oscillate. More strongly, the absence of a dispute wheel means that the instance and every sub-instance has a unique stable route allocation.

The absence of a dispute wheel is, in fact, the broadest-known sufficient condition for stability and robustness. In the design of an incentive-compatible routing mechanism, we want to ensure that our routing algorithm does reach a stable tree in some equilibrium. We therefore require that nodes' valuations, which, in our model, express routing policies, do not induce a dispute wheel.

The following defines an equivalent sufficient condition using the language of valuation functions. The equivalence between this definition and the original definition of a dispute wheel in [10] can be found in [8, 9].

Definition 4.4. Define two relations on permitted routes in an instance I:

1. Let $R_1 \ominus_1 R_2$ iff R_1 is a suffix of R_2, *i.e.*, there is some j such that $R_1 = R_{2[j,d]}$ and $R_1 \in P^j$.
2. Let $R_1 \ominus_2 R_2$ iff $\exists\, i \in N : R_1, R_2 \in P^i$ and $v_i(R_1) > v_i(R_2)$.

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

Instance I has *no dispute wheel* iff $R_1 \oslash R_2$ and $R_2 \oslash R_1$ implies that either $R_1 = R_2$ or R_1, R_2 start at the same node. (Informally, this is antisymmetry of $\oslash$ except that ties are allowed in valuations.)

Fig. 3 shows a routing instance (DISAGREE, from [10]) with policies that induce a dispute wheel. This instance has two stable route allocations: $\{1d, 21d\}$ and $\{12d, 2d\}$. Because the network is asynchronous, the timing of update messages may cause BGP to converge to either of these solutions or oscillate between them [10]. This anomaly is manifested by the following dispute wheel:

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

The price of anarchy in this example is $(1 + \alpha)$, which can be arbitrarily bad given the choice of $\alpha > 0$.

4.3 Policy Consistency

Our interdomain-routing problem is an optimization problem in which each node assigns *cardinal* values to the different routes. Even without dispute wheels, finding a stable route allocation based on ordinal preferences does not suffice, because that allocation's value can be much lower than that of the optimal route allocation.

Fig. 4 shows an instance without a dispute wheel; assume $\alpha > 0$. The unique stable route allocation 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, the price of anarchy in this example, $1 + \frac{1}{399}\alpha$, is also unbounded.

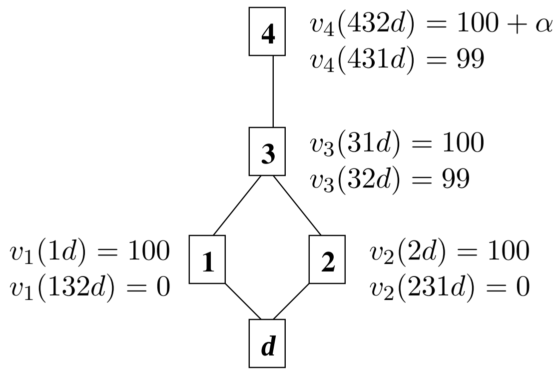

Figure 4: A routing instance without policy consistency.

To overcome this problem, we formally introduce the *policy-consistency* property. This property helps to ensure that the optimal route allocation is stable and, when combined with dispute-wheel freeness and *consistent filtering* (defined in the next subsection), means that any path-vector protocol converges to an optimal route allocation. (We explore the interesting connections between the three conditions in Thm. 4.10, first studied in a modified form by Sobrinho in [19].)

Informally, a node i is policy-consistent with an adjacent node j if there are no two routes from i to d with next hop j, such that j prefers one to the other, but i disagrees.

Definition 4.5. Let i and j be two adjacent nodes in G. We say that i is *policy-consistent* with j iff for every two routes $\{Q, R\} \subset P^j$ such that $i \notin Q$, $i \notin R$, and $\{(i,j)Q, (i,j)R\} \subset P^i$,

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

Definition 4.6. An instance is policy-consistent (or "policy consistency holds") iff, for every two adjacent nodes i and j, i is policy-consistent with j.

Assuming policy consistency in a network is natural for the same reason that next-hop preferences are: Nodes have little control over forwarding paths beyond the next hop. Note that next-hop valuations are, in fact, policy-consistent.

Other examples in which policy consistency holds are *metric-based valuations* (defined in [9]):

Definition 4.7. Let $\delta : L \to \mathbb{R}_{>0}$ be a positive real-valued function that specifies the "length" of each link (a "metric" function). A valuation function v that is based on δ is one in which $v(Q) > v(R)$ iff $\sum_{l \in Q} \delta(l) < \sum_{l \in R} \delta(l)$.

It is easy to see that, if all nodes' valuations are based on the same underlying metric function δ, then the network is policy-consistent. In particular, if $\delta(l) = 1$ for every link l, then this is precisely the well known shortest-path-routing problem.

4.4 Consistent Filtering

In traditional formulations of interdomain routing, nodes are allowed to *filter* routes arbitrarily when exporting updates to or importing updates from neighbors, *i.e.*, nodes can arbitrarily remove paths from consideration (restricting P^i).

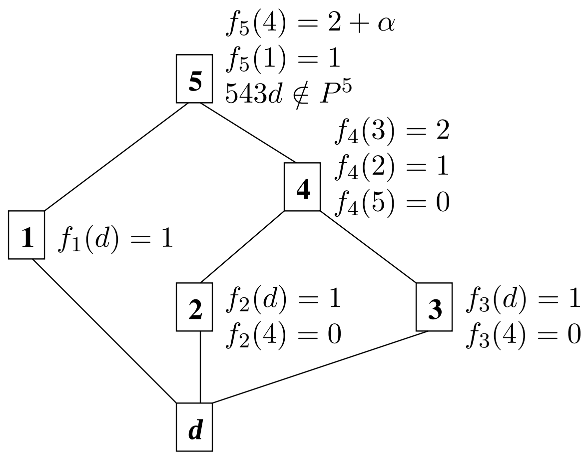

Figure 5: Next-hop policies without consistent filtering.

In the welfare-maximizing formulation of interdomain routing, arbitrary filtering is often disallowed. Arbitrary filtering, like the lack of policy consistency, can make the price of anarchy unbounded, because a node may value a route that is filtered by a neighbor much more than any other route available. This is the case in Fig. 5, an instance with next-hop policies (which are policy-consistent) and no dispute wheel. (Again, assume $\alpha > 0$.) Although node 5 generally prefers routing through node 4, the path $543d$ is filtered. If node 4 chooses to route through node 2, node 5 can route through node 4, and this leads to the optimal route allocation, $\{1d, 2d, 3d, 42d, 542d\}$. However, this allocation is not stable, because node 4 prefers routing through node 3, which prevents node 5 from routing through node 4, leading to the unique stable route allocation $\{1d, 2d, 3d, 43d, 541d\}$. The price of anarchy in this example is $1 + \frac{1}{6}\alpha$, which can grow arbitarily large as $\alpha \to \infty$.

In order to achieve our objective of welfare maximization, we require that nodes not filter routes arbitrarily. If a node filters a route, it must value that route less than any route that is not filtered—this is called *consistent filtering*.

Definition 4.8. Node i *filters consistently* with respect to (adjacent) node j iff, for any route $R \in P^i$ such that $(j, i)R \notin P^j$ and $(j, i)R$ is simple, $v_i(R) < v_i(Q)$ for all routes $Q \in P^i$ such that $(j, i)Q \in P^j$.

We say that an instance "filters consistently" if every node filters consistently with respect to every other adjacent node.

Remark 4.9. The *isotonicity* property studied by Sobrinho in [19] for its relationship to optimal routing essentially combines policy consistency and consistent filtering.

4.5 Local and Global Optimality

We now turn to the interesting relationship among the three properties presented in this section: no dispute wheel, policy consistency, and consistent filtering. Recall that, if an instance has no dispute wheel, then it has a unique stable route allocation. The following theorem states that, if all three properties hold, then this unique route allocation is globally optimal (*i.e.*, it maximizes the total social welfare).

Theorem 4.10. *If the valuation functions do not induce a dispute wheel, and nodes filter consistently and are policy-consistent, then there exists a unique stable route allocation T_d, and*

$$T_d = \text{argmax}_{T=\{S_1,\ldots,S_n\}} \sum_{i=1}^{n} v_i(S_i).$$

Proof. We will use the following lemma in the proof of the theorem.

Lemma 4.11. *If $T = \{R_1, \ldots, R_n\}$ is a globally optimal allocation for an instance with policy consistency and consistent filtering, then T is stable.*

Proof. Assume by contradiction that T is not stable; then, by Def. 4.1, there are two adjacent nodes i and j such that

$$v_i(R_i) < v_i((i,j)R_j). \tag{4}$$

Let k be a node such that $\text{next}(R_k) = i$. Because k is policy consistent with i, and because i and k filter consistently, (4) implies that

$$v_k(R_k) < v_k((k,i)(i,j)R_j);$$

by induction, this is also true for every node k' with next hop k in T, *etc.*, so that every node u routing through i in T prefers the route $R_{u[u,i]}(i,j)R_j$ to R_u. Note that we have identified a route allocation in which i and all nodes routing through i are strictly better off, and all nodes not routing through i are unaffected. This new allocation has higher total social welfare than T; however, this contradicts the optimality of T. Thus, our assumption must be incorrect, and T must therefore be stable. □

Now, let T be some optimal route allocation. By Lemma 4.11, because of policy consistency and consistent filtering, T is stable. However, because there is no dispute wheel, there is only one stable allocation T_d [10]. Therefore, $T = T_d$, and the unique stable allocation is also optimal. □

A locally optimal route allocation is one in which nodes are assigned their most valued routes.

Definition 4.12. A route allocation $T_d = \{R_1, \ldots, R_n\}$ is *locally optimal* iff, for every node i, $R_i = \text{argmax}_{R \in P^i}\, v_i(R)$, *i.e.*, every node i is allocated its highest-valued route.

The following theorem shows that the combination of no dispute wheel, policy consistency, and consistent filtering ensures not only global optimality but local optimality as well.

Theorem 4.13. *If an instance has no dispute wheel, consistently filters, and is policy consistent, then any globally optimal, stable route allocation is also locally optimal.*

Proof. Consider a node $m \in N$. Let $R = u_k u_{k-1} \ldots u_i \ldots u_0$ be some simple 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 $R_i = R_{[u_i,d]}$. When $i = m$ we get that S_m is at least as good as R; because R and m were chosen arbitrarily, we prove 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.*,

$$v_{u_{i-1}}(S_{i-1}) > v_{u_{i-1}}(R_{i-1} = u_{i-1}u_{i-2}\ldots d). \tag{5}$$

Note that u_i does not lie on R_{i-1}, or R would not be simple.

Case I. Assume $u_i \notin S_{i-1}$. Then extend S_{i-1} and R_{i-1} along the edge (u_i, u_{i-1}). Consistent filtering ensures that $(u_i, u_{i-1})S_{i-1} \in P^{u_i}$; thus, from (5) and policy consistency, we have

$$v_{u_i}((u_i, u_{i-1})S_{i-1}) > v_{u_i}(R_i = u_i u_{i-1} u_{i-2} \dots d). \tag{6}$$

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

$$v_{u_i}(S_i) > v_{u_i}((u_i, u_{i-1})S_{i-1}). \tag{7}$$

Combining (6) and (7) gives

$$v_{u_i}(S_i) > v_{u_i}(R_i),$$

which is the induction statement for u_i.

Case II. Assume $u_i \in S_{i-1}$. In this case we cannot use the policy consistency argument as in Case I, because extending S_{i-1} to u_i creates a loop. But then $u_{i-1} \notin S_i$.

Suppose the induction statement for i is not true: $v_{u_i}(R_i) > v_{u_i}(S_i)$. 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; thus the assumption that $v_{u_i}(R_i) > v_{u_i}(S_i)$ leads to a contradiction. Therefore, $v_{u_i}(R_i) < v_{u_i}(S_i)$, which is the induction statement for u_i. (Recall there are no ties in valuations.) $\square$

Remark 4.14. (C2) in Thm. 3.6 is a special case of this result, because the Gao-Rexford conditions imply no dispute and consistent filtering, and next-hop valuations imply policy consistency.

Remark 4.15. Global and local optimality also hold for sub-instances. If any of the three properties (no dispute wheel, policy consistency, consistent filtering) hold in an instance, they also hold in all sub-instances. Thus, all sub-instances of an instance satisfying the requirements of Thm. 4.10 and Thm. 4.13 also satisfy the requirements of these theorems.

5 An Algorithm for General Classes of Policies

The no-dispute-wheel property guarantees that any path-vector protocol converges to the unique stable route allocation. When combined with policy consistency and consistent filtering, this route allocation is globally optimal. Therefore, if these three properties hold, we can use a path-vector protocol to compute welfare-maximizing routes. However, there is still the matter of ensuring that the ASes have no motivation to rationally manipulate the protocol in order to better their outcome.

We now present an incentive-compatible, distributed algorithm for interdomain routing on instances that are dispute-wheel-free and policy-consistent. We investigate its incentive-compatiblity properties in detail; its payment structure naturally enforces consistent filtering and truthful participation, and the algorithm is also not subject to other forms of rational manipulation as formulated by Shneidman and Parkes in [18].

The BGP-compatible algorithm in Sec. 3 is a specific case of this algorithm; we conclude this section by presenting another BGP-compatible special case, that of metric-based valuations. Note that the general-case algorithm is not BGP-compatible, because its implementation requires more than a modest increase to the storage space at each node.

5.1 Algorithm Specification

This algorithm can be thought of as a "meta-algorithm" in the sense that it ignores implementation aspects (including those related to internal memory considerations and message passing). We prove the correctness of our algorithm for the case in which policies do not induce a dispute wheel and nodes are policy-consistent.

Setting: An instance $I = (G, \mathcal{P}, \mathcal{V})$ of the interdomain-routing problem that is dispute-wheel-free and policy-consistent.

Outcome: A route allocation $T_d = \{R_1, \dots, R_n\}$ that forms a confluent tree to d, such that

$$T_d = \text{argmax}_{T=\{S_1,\dots,S_n\}} \sum_{i=1}^{n} v_i(S_i).$$

The Algorithm: The algorithm runs $n+1$ copies of a path-vector protocol (see Sec. 2.3) to find the locally optimal route allocation T_d^{-i} for each I^{-i}, $1 \le i \le n$, and the locally optimal route allocation T_d for I. It can therefore be regarded as composed of many similar "sub-algorithms" that are executed simultaneously.

Once all sub-algorithms reach a stable route allocation, every node j is assigned its route in T_d. Its payment is computed as follows: Every node i computes a *payment component* for j, $s_j^i = v_i(R_i) - v_i(R_i^{-j})$, in which R_i and R_i^{-j} are the routes allocated to i in T_d and T_d^{-j}, respectively. The total payment to node j is defined to be the sum of these payment components, $s_j = \sum_{i \neq j} s_j^i$.

Theorem 5.1. *If policies do not induce a dispute wheel and are policy-consistent, this algorithm converges to a route allocation that maximizes total social welfare.*

Proof. The payments computed by the algorithm naturally enforce consistent filtering; we defer this discussion to Sec. 5.2. Therefore, we can assume that instances (and, by Rem. 4.15, all sub-instances) have no dispute wheel, are policy consistent, and consistently filter. By Thm. 4.10, there exists a unique stable route allocation for the instance and each sub-instance; by Thm 4.13, this route allocation is both globally and locally optimal. Because each sub-algorithm converges to a locally optimal route allocation, the final route allocation for the original instance maximizes total social welfare, and the allocations for each sub-instance can be used to compute the payments (that enforce consistent filtering). □

In Sec. 4.5, we showed that, if an instance is dispute-free, is policy-consistent, and filters consistently, then every path-vector protocol converges to a route allocation that is both globally and locally optimal. However, Thm. 5.1 only requires no dispute wheel and policy consistency. In Sec. 5.2, we show that these two properties suffice—if both properties hold, then nodes have no incentive, given the payments computed, to filter any routes.

The local optimality of the route allocation reached by the algorithm leads to two important observations regarding the computation of payments: First, all payment components calculated by the nodes are nonnegative; so, the payment to each node is nonnegative. Hence, we are guaranteed that nodes will not have to pay the bank for their participation in the algorithm. Second, node i's payment component s_j^i for every node $j \notin R_i$ (R_i is i's optimal route) is always 0, because $R_i = R_i^{-j}$. Therefore, every node i only needs to store in its memory alternate routes and payment-component values for the transit nodes on its best route.

5.2 Incentive Compatibility

To prove that our mechanism is incentive-compatible, we first consider the restricted case in which the only form of rational manipulation available to the nodes is not revealing their true preferences. In particular, nodes can lie about what routes are available by filtering routes arbitrarily.

Theorem 5.2. *The payments $s_i = \sum_{j \neq i} s_j$ have the form of VCG payments.*

Proof. The proof is identical to that of Thm. 3.5, except that $f_i(L_i(k))$ (next-hop policies) is replaced with $v_i(R_i^{-k})$ (valuation using general policies). $\square$

VCG payments guarantee the strongest possible result for the restricted case: truthful behavior of all nodes leads to a *dominant-strategy equilibrium*. That is, a rational node's best strategy is conveying its true preferences no matter what the preferences of the other nodes are. Hence, a node need not make any kind of assumptions on the other nodes' behavior or have any *a priori* knowledge about their preferences. Thus, incentives naturally enforce the consistent-filtering condition, because nodes have no motivation to filter routes beyond what is necessary to enforce no dispute wheel (*e.g.*, the third Gao-Rexford condition; see Sec. 3.1).

As pointed out by Shneidman and Parkes [18], in a distributed setting, there are many other forms of rational manipulation available to the strategic agents. This is because the computation is executed by the strategic agents themselves (and not by some reliable third party, as is the case in a centralized setting). In our model, for example, nodes may refuse to pass messages or choose to alter the contents of messages that go through them.

Let us consider the more general case in which nodes have many ways of rationally manipulating the algorithm. We prove incentive compatibility by showing that a node cannot benefit by deviating from the information revelation, communication, and computational actions it is instructed to perform by the protocol.[4] We make use of the techniques in [18] to show that, with a minor adjustment, our algorithm obtains incentive compatibility in *ex-post Nash equilibrium*. The only modification needed is requiring, as in [18], that all communication between the bank and the nodes be signed and receive signed acknowledgments. (The bank has the power to investigate when receipts are not received.)

An ex-post Nash equilibrium is a robust solution concept: In such an equilibrium, no single node would deviate from the algorithm even if it knew the other nodes' private valuations. If we aim at an ex-post Nash equilibrium, we must assume only that all nodes are rational and wish to maximize their utilities.[5] Shneidman and Parkes view the need to settle for an ex-post Nash equilibrium in the general case (instead of an equilibrium in dominant strategies in the restricted case) as "the cost of distributing mechanism computation across a network" [18].

Theorem 5.3. *The modified algorithm is incentive compatible in ex-post Nash equilibrium.*

Proof. We prove the theorem by addressing the various ways in which a node might attempt to rationally manipulate the algorithm. We show that such possible attempts can only harm the node. The proof relies on the fact that the bank is a trusted party, that the bank-nodes communication uses cryptographic signing, and that the bank also has the power to "restart" the algorithm if it notices any rational manipulation attempts.

Let us look at a single node i and assume that all other nodes are obeying the algorithm's instructions. Node i could choose to misreport its true preferences or what routes are available when asked by other nodes. However, i's payment depends solely on other nodes; i is paid for its contribution to social welfare. It can be shown using VCG argumentation that this payment technique means that i has nothing to gain by lying; in particular, i gains nothing from lying about the availability of routes (see Cor. 5.4 below).

[4] These three properties are what Shneidman and Parkes [18] refer to as IC-, CC-, and AC-compatibility.

[5] The ex-post Nash equilibrium concept is strictly stronger than the well known Nash-equilibrium concept. A Nash-equilibrium-oriented implementation of our algorithm would have to assume that every node is familiar with the preferences of all other nodes. This assumption is unrealistic in interdomain routing.

Another possible way in which i might try to rationally manipulate the algorithm is by refusing to pass messages from other nodes addressed to the bank or by passing messages after altering their contents. The cryptographic signing of bank-nodes communication excludes the possibility of i making any such attempts. This is true no matter what the computational and information-revelation actions of i are.

Finally, one more possible form of rational manipulation is performing incorrect calculations of i's components of other nodes' payments. However, it is easy to see that i has nothing to gain by doing so, since these payments have no effect on i's route or payment. This is true no matter what the communication and information-revelation actions of i are.

One can also show that any combination of rational manipulation attempts will not lead to any improvement in i's condition compared to what he would get by abiding by the algorithm's rules. This is achieved using the general proof technique of Shneidman and Parkes [18].[6]

Hence, we have shown that this slightly modified version of the algorithm in Sec. 5.1 is not subject to any of the possible forms of rational manipulation available to the nodes. Hence the modified algorithm is incentive-compatible in ex-post Nash equilibrium. □

Corollary 5.4. *The incentive structure of our mechanisms ensures consistent filtering.*

This is because in ex-post Nash equilibrium no node has an incentive to filter any of the routes. Because this fact is especially important to prove the correctness of our algorithm, we explicitly include the argument used for its proof below.

Proof. Assume that node i filters some route $R \in P^i$ on export to node j. If R is not i's optimal route, then filtering R has no effect on the algorithm's route allocation to i or j, because, given Thm. 4.10, R is unstable as a route choice at i; so, i will export some other route to j.

Therefore, without loss of generality, assume that R is i's optimal route. There are two cases:

1. $(j,i)R$ is not j's optimal route. In this case, $s_i^j = 0$ regardless of whether or not i filters R; so, there is no change in i's utility.

2. $(j,i)R$ is j's optimal route. In this case, filtering R will force j to choose another route R'. Because i filters its best route, $i \notin R'$, and so $s_i^j = 0$. However, if i had not filtered R, then i would be a transit node on j's best path, and $s_i^j \geq 0$. Therefore, i's utility can only decrease by filtering.

Nodes thus have no incentive to filter routes arbitrarily. □

Remark 5.5. Because dispute-free policies imply robustness, the problem has a unique stable solution; this solution is also optimal. Since every such stable solution is an ex-post Nash equilibrium, we have only one ex-post Nash equilibrium. Therefore, we avoid the problem that arises when multiple equilibria exist, *i.e.*, making sure that the nodes select the same equilibrium.

Remark 5.6. As in [18], we too assume that nodes are *benevolent* in the sense that they will implement the algorithm's instructions as long as they do not strictly prefer choosing another strategy. Therefore, we only require a weak ex-post Nash equilibrium.

[6] Using the terminology of [18], what we have shown is that the corresponding centralized algorithm is truthful and that the specification is strong-CC and strong-AC.

5.3 Metric-Based Valuations

The algorithm in Sec. 3.2 is a special case of the general algorithm in Sec. 5.1; the class of policies used in the former allows the algorithm to be more space-efficient than running $n+1$ copies of a path-vector protocol. We now briefly present another special case, that of *metric-based valuations*, defined in Sec. 4.3.

Metric-based valuations are inherently dispute-wheel free [8, 9, 19]; they are also policy-consistent. Thus, if nodes do not filter routes arbitrarily, metric-based valuations permit incentive-compatible, distributed computation of welfare-maximizing routes.

The important observation regarding metric-based valuations is that, just as with next-hop policies, when running a path-vector protocol on an instance with metric-based valuations, an AS need not store in its memory and communicate in each time step entire paths. This is because the value an AS assigns a route depends solely on the route's *length*, and so merely storing and communicating routes' lengths is sufficient. Thus, to compute $s_k^i = v_i(R_i) - v_i(R_i^{-k})$, node i only needs the lengths of R_i and R_i^{-k}, as these determine the valuation; furthermore, because of local optimality, node i need only do this for transit nodes on its best (and chosen) route to the destination ($s_k^i = 0$ for non-transit nodes).

A straightforward extension of BGP can be used to propagate this information. Update messages from j will include, in addition to j's best route R_j, the length of R_j and, for every $k \in R_j$ ($k \notin \{j, d\}$), the length of the best known route at j that avoids k. Update messages are sent whenever this information changes at j. The BGP routing table is extended to store this extra information, requiring $O(1)$ extra space per node, per route, stored in the table.

When a node receives an update message, it checks the provided lengths to determine whether a shorter k-avoiding route is known (for each transit node k on the current best route). At the end of the algorithm, nodes have enough information to compute the payment components s_k^i.

Remark 5.7. We note that, because routes with shorter lengths are chosen as best, routes are forced to be simple. If a node knows of a k-avoiding route with a loop, it must also know of the route without the loop. If all lengths are positive, then the simple path will be strictly shorter.

We present the details of the algorithm for this class of policies below.

Setting: An instance of the interdomain-routing problem with metric-based valuations, *i.e.*, an instance in which: (1) there exists a positive function $\delta : L \to \mathbb{R}_{>0}$ specifying the length of each link; and (2) for every node i and for all pairs of permitted routes $\{Q, R\} \subset P^i$, $v_i(Q) > v_i(P)$ if and only if $\sum_{l \in Q} \delta(l) < \sum_{l \in R} \delta(l)$.

Outcome: A route allocation $T_d = \{R_1, \ldots, R_n\}$ that forms a confluent tree to d, such that

$$T_d = \text{argmax}_{T=\{S_1,\ldots,S_n\}} \sum_{i=1}^{n} v_i(S_i).$$

Structure of Update Messages: An update message from a node m contains: (1) R_m, the current choice of best route at m; (2) $\Delta_m = \sum_{l \in R_m} \delta(l)$, that route's length; and (3) for each transit node $k \in R_m$, the length of the best route known to m that avoids k, denoted $A_m(k)$.

Storage at Each Node: Each node i has a routing table Y_i, indexed by neighbors of i. At most one (the most current) update message is stored from each neighbor. Initially, $Y_i(j) = \emptyset$ for all $j \in \text{neighbors}(i)$. Each node also indicates its choice of current best route, denoted R_i; that route's length, Δ_i; and, for each transit node $k \in R_i$ ($k \notin \{i, d\}$), the length of the best known k-avoiding path is stored, denoted $L_i(k)$.

Start: The destination node d sends the message $(R_d = \emptyset, \Delta_d = 0, A_d = \emptyset)$ to all of its neighbors.

Update-Message Processing: When node i receives an update message (R_m, Δ_m, A_m) from node m, the route and length is first extended to node i, and the message is then stored in the routing-table entry for m, denoted $Y_i(m)$. The length of $R'_m = (i,m)R_m$ is $\Delta'_m = \Delta_m + \delta(i,m)$. The length $A'_m(k)$ of any k-avoiding route known to m, if used at i, would be $A_m(k) + \delta(i,m)$. Thus, the entry stored is $Y_i(m) = (R'_m, \Delta'_m, A'_m(k))$.

(Case I.) If $v_i(R'_m) > v_i(R_i)$, then the update message contains a better route. The current best route is changed and the list of lengths of transit-node-avoiding routes is repopulated:

1. Set R_i to be R'_m and Δ_i to be Δ'_m.
2. Clear the list L_i.
3. For each $k \in R_i$ ($k \notin \{i,d\}$):
   (a) Set $L_i(k) = \infty$.
   (b) For each $j \in \text{neighbors}(i)$ such that $Y_i(j) \neq \emptyset$:
      i. If $k \notin R'_j$ and $\Delta'_j < L_i(k)$, then set $L_i(k) = \Delta'_j$ and continue to check the next neighbor j.
      ii. If $k \in R'_j$ and $A'_m(k) < L_i(k)$, then set $L_i(k) = A'_m(k)$ and continue to check the next neighbor j.

(Case II.) If $v_i(R'_m) < v_i(R_i)$, then the update message does not contain a better route. However, it may contain better k-avoiding routes for transit nodes k on the current best route to d. For each $k \in R_i$ ($k \notin \{i,d\}$), if $A'_m(k) < L_i(k)$, then set $L_i(k) = A'_m(k)$.

If any changes were made to R_i or L_i, then send the update message $(R_i, \Delta_i, A_i(k))$, in which $A_i(k) = L_i(k)$ for each transit node $k \notin \{i,d\}$ on R_i.

Payment Computation: When the algorithm converges, each node i has a route R_i, which is its route in the routing tree, and enough information to compute its payment component for transit nodes k on R_i, $s^i_k = v_i(R_i) - v_i(R_i^{-k})$, because $\sum_{l \in R_i} \delta(l) = \Delta_i$ and $\sum_{l \in R_i^{-k}} \delta(l) = L_i(k)$. The total payment made to node j by the bank is the sum of the payment components for j, $s_j = \sum_{a \neq j} s^a_j$.

Theorem 5.8. *The algorithm converges to an optimal tree and computes the correct VCG payments.*

Proof. Metric-based valuations satisfy the requirements of theorems discussed earlier that imply convergence of the algorithm and optimality of the output tree. We must still prove that each payment components are computed correctly. Because R_i is the optimal route for the original instance, its length is known, and valuations are metric-based, it is enough to show that nodes have enough correct information to determine $\sum_{l \in R_i^{-k}} \delta(l)$. We will show that this is simply $L_i(k)$.

The algorithm converges once the routes R_i and lengths $L_i(k)$ stop changing everywhere (at all nodes i and for all transit nodes k). Because nodes have no incentive to arbitrarily filter, we know that the best route information sent via update messages is accurate and complete.

Consider $T_d \setminus \{k\}$ for some $k \in T_d$, and let T^* be the component still containing d. For all nodes $i \in T^*$, $R_i^{-k} = R_i$, because the routes R_i are the shortest (and thus most valued) routes to d, and they are unaffected by removing k. Let $H_0 = \left\{ i \notin T^* \mid \exists j \in T^* : \{i,j\} \in T_d^{-k} \right\}$. Although the nodes in H_0 normally route through k, when k is removed, these nodes' best routes have a next hop in T^*. Thus, for each node $i \in H_0$, R_i^{-k} is always made available because $j = \mathrm{next}\left(R_i^{-k}\right)$ exports R_j as its best choice. If $\delta(i,j) + \Delta_j$ is indeed shortest, the algorithm correctly sets (through Case II of update-message processing) $L_i(k) = \delta(i,j) + \Delta_j$, corresponding to the length of $(i,j)R_j = R_i^{-k}$.

Now consider nodes $H_1 = \left\{i \notin T^* \mid \exists j \in H_0 : \{i,j\} \in T_d^{-k}\right\}$. Every node $i \in H_1$ will receive an update message from a neighbor $j \in H_0$ with $A_j(k) = \sum_{l \in R_j^{-k}} \delta(l)$. Because $(i,j)R_j^{-k}$ is shortest, the algorithm will correctly set (through Case I, if $\mathrm{next}(R_i) = j$, or Case II, if $\mathrm{next}(R_i) \neq j$) $L_i(k) = \delta(i,j) + A_j(k) = \sum_{l \in R_i^{-k}} \delta(l)$. We can continue this argument for $H_x = \left\{i \notin T^* \mid \exists j \in H_{x-1} : \{i,j\} \in T_d^{-k}\right\}$ until we have shown that $L_i(k)$ is set correctly for every node. □

6 Conclusions and Open Questions

In this paper, we addressed the problem of incentive-compatible interdomain routing. Our main result is a BGP-compatible, incentive-compatible mechanism for a realistic class of routing policies, thus answering an open question posed in [3]. Additionally, we stated general conditions that are sufficient for designing incentive-compatible, welfare-maximizing protocols for more general classes of routing policies. Using this general characterization, we presented a BGP-compatible mechanism for yet another class of valuations, namely metric-based valuations. It would be interesting to find other natural classes of valuations for which BGP-compatible mechanisms exist.

There are many other issues that remain unresolved and call for further research. One such issue is that of designing distributed (preferably BGP-compatible) mechanisms that obtain *good approximations* to the total social welfare. Very little is known about the approximability of the interdomain-routing problem. Feigenbaum, Sami, and Shenker [5] show that, if we impose no restrictions on the routing policies, then no good approximation ratio is attainable. A first step towards the design of BGP-compatible approximation mechanisms is finding a nontrivial characterization of routing policies for which the price of anarchy is low.

Introducing incentive compatibility into the interdomain-routing setting involves paying ASes for their participation in the algorithm. The way these payments are computed leads to many interesting questions: How can we make sure that the ASes are not overpaid for the transit services they provide? (VCG mechanisms are often criticized in the literature for overpaying the strategic agents.) In our formulation, the ASes do not pay each other but are paid by *the bank* (as in [18]). Is it possible to get rid of the bank and have ASes pay other ASes directly for transit services rendered?

A distributed setting such as ours poses an inherently different challenge for the design of incentive-compatible mechanisms (see [3, 18]) than a centralized one. This is because the computation is performed by the strategic agents themselves and not by a reliable third party. We reconcile the strategic model and the distributed computational model by using techniques similar to those in [18]. In particular, we use cryptographic signing. Is it possible to reconcile the two models without having to resort to this technique?

Finally, the question of optimal communication complexity for the computation of routes and payments remains open. We have stressed space complexity in this paper, but there may be an increase over BGP in the number of update messages sent by our algorithms. This is because our algorithms have an additional condition that triggers sending an update message, namely, any change to the best known k-avoiding route (or next hop), for any transit node k on the current best path. Update messages are not sent for this reason in the original BGP. Although the message complexity of our algorithms is not unreasonable with respect to BGP's worst-case performance, the optimal number of messages needed to compute payments in addition to routes is currently unknown.

7 Acknowledgements

The authors would like to thank Tim Griffin, Aaron Jaggard, Jennifer Rexford, Rahul Sami, and Scott Shenker for many helpful discussions about interdomain routing.

References

[1] M. Caesar and J. Rexford. BGP Policies in ISP Networks. *IEEE Network Magazine* **19**(6):5–11, Nov. 2005.

[2] J. Feigenbaum, D. Karger, V. Mirrokni, and R. Sami. Subjective-Cost Policy Routing. In *Proc. Wshp. Internet and Network Economics (WINE)*, pp. 174–183, LNCS vol. 3828. Springer-Verlag, Dec. 2005.

[3] J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker. A BGP-based Mechanism for Lowest-Cost Routing. *Distributed Computing* **18**(1):61–72, Jul. 2005.

[4] J. Feigenbaum, V. Ramachandran, and M. Schapira. Incentive-Compatible Interdomain Routing (Extended Abstract). In *Proc. ACM Electronic Commerce (EC'06)*. ACM Press, Jun. 2006.

[5] J. Feigenbaum, R. Sami, and S. Shenker. Mechanism Design for Policy Routing. In *Proc. 23rd Symp. Principles of Distributed Computing (PODC'04)*, pp. 11–20, ACM Press, Jul. 2004.

[6] L. Gao, T. G. Griffin, and J. Rexford. Inherently Safe Backup Routing with BGP. In *Proc. IEEE INFOCOM'01*, pp. 547–556. IEEE Computer Society, Apr. 2001.

[7] L. Gao and J. Rexford. Stable Internet Routing without Global Coordination. *IEEE/ACM Trans. Networking* **9**(6):681–692, Dec. 2001.

[8] T. G. Griffin, A. D. Jaggard, and V. Ramachandran. Design Principles of Policy Languages for Path Vector Protocols. In *Proc. ACM SIGCOMM'03*, pp. 61–72. ACM Press, Aug. 2003.

[9] T. G. Griffin, F. B. Shepherd, and G. Wilfong. Policy Disputes in Path Vector Protocols. In *Proc. 7th Int'l Conf. Network Protocols (ICNP)*, pp. 21–30. IEEE Computer Society, Nov. 1999.

[10] T. G. Griffin, F. B. Shepherd, and G. Wilfong. The Stable Paths Problem and Interdomain Routing. *IEEE/ACM Trans. Networking* **10**(2):232–243, Apr. 2002.

[11] J. Green and J. Laffont. Incentives in Public Decision Making. In *Studies in Public Economics*, vol. 1, pp. 65–78. North Holland, Amsterdam, 1979.

[12] J. Hershberger and S. Suri. Vickrey Prices and Shortest Paths: What is an edge worth? In *Proc. 42nd Symp. Foundations of Computer Science (FOCS)*, pp. 129–140. IEEE Computer Society, 2001.

[13] G. Huston. Interconnection, Peering, and Settlements. In *Proc. Internet Global Summit (INET)*. The Internet Society, Jun. 1999.

[14] E. Koutsoupias and C. H. Papadimitriou. Worst-case Equilibria. In *Proc. 16th Symp. Theoretical Aspects of Computer Science (STACS)*, pp. 387–396, LNCS vol. 1563 (G. Meinel and S. Tison, eds.). Springer-Verlag, Mar. 1999.

[15] J. Moy. Open Shortest Pouting First (OSPF) version 2. RFC 2328. Internet Engineering Task Force (IETF), Apr. 1998.

[16] N. Nisan and A. Ronen. Algorithmic Mechanism Design. *Games and Economic Behavior* **35**(1,2):166–196, 2001.

[17] Y. Rekhter and T. Li. A Border Gateway Protocol (BGP-4). RFC 1771. Internet Engineering Task Force (IETF), Mar. 1995.

[18] J. Shneidman and D. C. Parkes. Specification Faithfulness in Networks with Rational Nodes. In *Proc. 23rd ACM Symp. Principles of Distributed Computing (PODC'04)*, pp. 88–97, ACM Press, Jul. 2004.

[19] J. L. Sobrinho. An Algebraic Theory of Dynamic Network Routing. *IEEE/ACM Trans. Networking* **13**(5):1160–1173, Oct. 2005.

[20] K. Varadhan, R. Govindan, and D. Estrin. Persistent Route Oscillations in Interdomain Routing. *Computer Networks* **32**(1):1–16, Jan. 2000.