Interdomain Ingress Traffic Engineering through Optimized AS-Path Prepending

Ruomei Gao, Constantinos Dovrolis, Ellen W. Zegura
gaorm, dovrolis, ewz@cc.gatech.edu
Networking and Telecommunications Group, Georgia Tech, Atlanta 30332

Abstract— **In Interdomain Ingress Traffic Engineering (INITE), a "target" Autonomous System (AS) aims to control the ingress link through which the traffic of one or more upstream source networks flows to the target network or to its customers. Currently, there are few methodologies for systematic INITE. In practice, ISPs often attempt to manipulate, mostly in a trial-and-error manner, the AS-Path length attribute of upstream routes through a simple technique known as prepending (or padding). In this paper, we focus on prepending and propose a polynomial-time algorithm (referred to as OPV) that determines the optimal padding for an upstream route at each ingress link of the target network. Specifically, given a set of "elephant" source networks for a particular customer of the target network, and a set of maximum load constraints on the ingress links of the latter, OPV determines the minimum padding at each ingress link so that the load constraints are met, when it is feasible to do so. OPV requires as input an AS-Path length estimate from each source to each ingress link. We describe how to estimate this matrix, leveraging the BGP Looking Glass Servers that are abundant today for monitoring interdomain routing. To deal with unavoidable inaccuracies in the AS-Path length estimates, and also to compensate for the generally unknown BGP tie-breaking process in upstream networks, we develop a robust variation (RPV) of the OPV algorithm. We show that RPV manages to identify a padding vector that meets the given maximum load constraints, when it is feasible to do so, even in the presence of inaccurate AS-Path lengths and unknown BGP tie-breaking behavior.**

I. Introduction

Traffic Engineering (TE) refers to the design and implementation of controls that affect the flow of traffic in a network, or internetwork, to meet a performance objective. Typical objectives include load balancing across different links/paths, low delays, or meeting given capacity constraints. As opposed to packet scheduling and buffer management schemes, which operate in short time scales (less than a second), and infrastructure provisioning, which takes place in large time scales (days or longer), TE is a medium time scale operation (minutes to hours) and it often requires supervision from a human operator [1]. In relatively stable conditions, in terms of both routing changes and load variations, TE can be instrumental in improving network efficiency and robustness.

TE is broadly divided into two types: intradomain and interdomain. In intradomain TE, the operator of an Autonomous System (AS) controls the flow of traffic within that network by optimizing the link costs of the corresponding routing protocol (mostly OSPF or IS-IS), or through dynamic provisioning of virtual circuits (e.g., MPLS). For previous work in intradomain TE we refer the reader to [2], [3], [4], [5], [6] and to references therein. Intradomain TE assumes that the ingress and egress links of interdomain traffic flows are given as inputs, in the form of a traffic matrix, and they cannot be manipulated.

Interdomain TE, on the other hand, aims to control exactly those ingress and egress flows. Let us consider a "target" AS, referred to as $\mathcal{T}$. $\mathcal{T}$ has a number of ingress links, receiving traffic from upstream ASes. If $\mathcal{T}$ is not a stub network, traffic that is destined to one of that network's customers eventually leaves $\mathcal{T}$ through an egress link. Controlling the egress link that the traffic will flow through is referred to as *Interdomain Egress TE*. On the other hand, controlling the ingress link through which the traffic of a source network will enter $\mathcal{T}$ is referred to as *Interdomain Ingress TE (INITE)*.

Comparing the maturity, in terms of both operations and research, between INITE and the other types of TE, the former is much less deployed, understood, and trusted [7], [8]. The high-level reason is that INITE requires that the target network $\mathcal{T}$ has a way to affect BGP routing decisions in upstream ASes, without necessarily the active cooperation of those networks. In more detail, to perform INITE the operator of $\mathcal{T}$ would face the following major unknowns about the *upstream cloud*, i.e., the part of the Internet between a source network and the ingress links of $\mathcal{T}$:

- the *BGP policies*, especially those expressed through the Local Preference BGP attribute and the

ingress/egress routing filters deployed in the upstream cloud (note that the Local Preference attribute has the highest priority in the BGP route selection process),
- the actual AS-level topology of the upstream cloud, and in particular the *AS-Path lengths* from any upstream network to the ingress links,
- the *BGP tie-breaking behavior* of the upstream cloud, referring to the way a BGP speaker selects the best route among a set of candidate routes that have the same Local Preference and AS-Path length attributes in particular [9], [10], [11].

To the previous unknowns, one has to include the more common challenges of any TE problem, including variations in the traffic loads, unexpected infrastructure events (e.g., router crashes), etc.

Given the previous difficulties, it is not surprising that some ISPs avoid INITE. One form of INITE that is, however, used by some ISPs is that of *AS-Path prepending*, or simply "prepending" (also known as *AS-Path padding*). The idea is very simple: the target network $\mathcal{T}$ can make a route less attractive to its upstream ASes if it increases the AS-Path length of that route by adding several instances of its own AS-Number to that attribute. For instance, the AS-Path attribute $\{10\}$, where 10 is the AS-Number of $\mathcal{T}$, can be modified to $\{10,10,10\}$ before the corresponding route is advertised upstream. Effectively, this increases the route length from 1 to 3 hops, and so it becomes less likely that the route will be selected from an upstream network [9]. A recent measurement study showed that 32% of the routes in the AT&T network have some form of prepending, with about 90% of the corresponding paths extended by 1-5 hops [8]. Prepending is often performed in an ad-hoc manner, especially when an ISP simply wants to make a route unusable unless if there is a failure in other routes. In some cases, operators increase the degree of padding by a trial-and-error basis, until the AS-Path is long enough to reduce the load of that ingress link by a certain amount.

Our objective is to investigate the prepending technique more thoroughly, and understand both its potential and limitations. As a first step, in this paper, we show how to perform INITE in a systematic and algorithmic way using AS-Path prepending. We consider upstream clouds in which the route selection is not affected by the Local Preference attribute, meaning that the BGP routes are determined by the AS-Path length. The problem that we consider, in its basic form, is the following. Suppose that a target network $\mathcal{T}$ has N ingress links. A major customer network $\mathcal{D}$ of $\mathcal{T}$ receives most of its traffic from a set of M "elephant" sources, and an estimate of each source load s_i is given (i=1 ... M). $\mathcal{T}$ aims to impose a *maximum load* (maxload) constraint C_j at each ingress link j for the traffic that flows from the M sources to $\mathcal{D}$. To do so, $\mathcal{T}$ increases the AS-Path length of the route to $\mathcal{D}$ at ingress link j by "padding" its own AS-Number $a_j \geq 0$ times. The main questions then are: *what is the value of a_j for each link j that will meet the given maxload constraints, and when is it infeasible to meet these constraints through prepending?* In particular, we are interested in the *optimal prepending*, i.e., the padding vector A that minimizes the sum $\sum a_j$. We refer to the previous as the *Constrained Optimal Prepending (COP)* problem. COP, despite its simple statement, captures an important objective of ISPs, that of balancing the ingress load to a customer across a set of ingress links, and it attempts to leverage a currently used ad-hoc technique (prepending) in a more systematic methodology. Note that a trivial variation of COP is to consider all ingress traffic to $\mathcal{T}$, instead of the traffic to a customer $\mathcal{D}$; this would be the case, for instance, if $\mathcal{T}$ is a stub network.

The first contribution of this paper is to develop a polynomial-time algorithm, referred to as *OPV* (for Optimal Padding Vector), which solves the COP problem. The algorithm is very simple: at each iteration, an overloaded ingress link is chosen and its padding is incremented. Then, given the new padding vector algorithm, the mapping from sources to links is recomputed, and the algorithm moves to the next iteration.

A key input to the OPV algorithm is the *AS-Path length matrix P*. Each element $P_{i,j}$ of this matrix is an estimate of the shortest AS-Path length from a certain source network $\mathcal{S}_i$ to each ingress link $\mathcal{L}_j$. The second contribution of the paper is to describe four estimation techniques for P, leveraging the abundant routing Looking Glass Servers that are present in the Internet today, and to evaluate their accuracy. Overall, we show empirically that in 60-65% of the cases the estimation error is zero, in 85-90% it is less than ± 1 hop, and in about 95% it is less than ± 2 hops. Also, larger errors tend to happen mostly in longer paths.

The previous AS-Path length estimation errors are not insignificant, given that most AS-Paths that we estimate are 3-6 hops. To deal with errors in P, our third contribution is to develop a *Robust Padding Vector (RPV)* algorithm. RPV is a heuristic built on top of OPV. The objective in RPV is to determine a padding vector that will probably satisfy the given maxload constraints, even if the input matrix P has inaccurate elements and even if we do not know the BGP tie-breaking behavior in the upstream cloud. Simulation results show that, with the empirical error distribution in P that we observe, and with completely unknown tie-breaking behavior, RPV can still find a padding vector that satisfies the maxload constraints in more than 90% of the cases, as long as such a padding vector exists.

The paper is structured as follows. In § II, we describe COP more formally, and state the key underlying assumptions. In § III, we propose and study the OPV algorithm, proving that it finds the optimal padding vector in polynomial time. In § IV, we describe how to estimate the AS-Path length matrix and evaluate the accuracy of the proposed techniques. In § V, we present the RPV algorithm and evaluate its robustness through simulations. The related work in the context of interdomain TE is discussed in § VI. We conclude in § VII.

II. Problem Statement and Formulation

In this section, we first describe the COP problem more formally, state the formulation's key assumptions, and make some remarks regarding related operational issues.

A. COP Problem Statement

Consider a target network $\mathcal{T}$ that aims to do INITE through prepending (see Figure 1). INITE is performed separately for each major customer $\mathcal{D}$ of $\mathcal{T}$, (but $\mathcal{D}$ and $\mathcal{T}$ can be the same if the latter is a stub network, or if $\mathcal{T}$ performs INITE to all ingress traffic). In the following, $\mathcal{D}$ is an implied destination network, so it is not included in the notation. Suppose that $\mathcal{T}$ has N ingress links that can receive traffic for $\mathcal{D}$. Each ingress link $\mathcal{L}_j$ originates a route for $\mathcal{D}$, and advertises that route to its upstream BGP peer.

The traffic that is destined to $\mathcal{D}$ may be produced by a potentially large number of upstream source networks. Instead of considering individual source networks, which may be impractical, we focus on *super-source ASes*. A super-source AS $\mathcal{S}$ receives traffic destined to $\mathcal{D}$ from potentially several source networks. In the following, we refer to super-source ASes simply as "source networks", with the understanding that they may not be where the traffic actually originates from. Suppose that M is the number of source networks for the traffic to $\mathcal{D}$, and let s_i be the average traffic load forwarded by source $\mathcal{S}_i$. S=$\{s_i, i = 1 \dots M\}$ is the *source load vector*. The identification of source networks and the estimation of the load vector S are discussed in § IV.

The effectiveness of any AS-Path prepending technique, including ours, is limited by the use of local routing policies expressed through the Local Preference attribute. The reason is that the Local Preference attribute has a higher priority in the BGP path selection process than the AS-Path length (see Table II). Consider a source

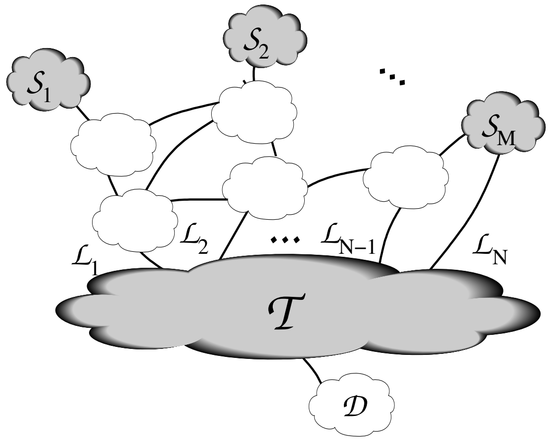


Fig. 1. High-level architecture of ingress interdomain traffic engineering.

network i, and suppose that i maintains a distinct route to each ingress link $\mathcal{L}_j$ of $\mathcal{T}$. This can be achieved if $\mathcal{T}$ advertises a unique *wayfinding prefix* at each link $\mathcal{L}_j$ (we return to this point in § IV). The selected routes from source i to the N ingress links of $\mathcal{T}$ form an AS path tree denoted by $\mathcal{U}_i$, as illustrated in Figure 2. In the following, we assume that the branching nodes of this tree do not select their best routes to the N ingress links based on Local Preference. In other words, the candidate routes for $\mathcal{T}$ at each branching node of $\mathcal{U}_i$ are assigned the same Local Preference value. The design of automated ways for examining the previous assumption is an important task for future work.

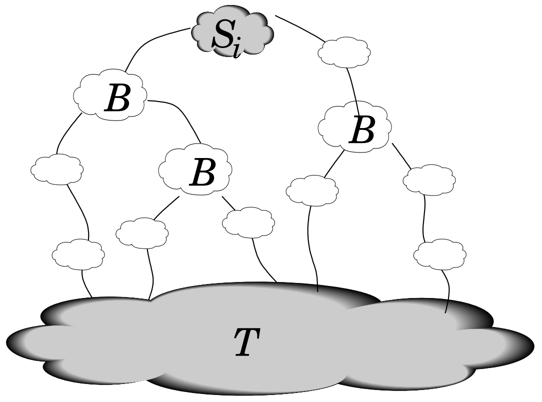


Fig. 2. Branching networks, labeled B in the diagram, in the upstream cloud $\mathcal{U}_i$

One of the key parameter is the BGP tie-breaking behavior in the upstream cloud. As shown in Table II, when two routes have the same Local Preference and AS-Path length, an ordered-list of five other criteria is used to choose the best route. We do not attempt to estimate or infer all the parameters that affect those tie-breaking cri-

$\mathcal{T}$	target network	$\mathcal{D}$	destination network
$\mathcal{S}$	source network	M	number of sources
$\mathcal{L}$	ingress link	N	number of ingress links
I	instance of COP problem	$\mathcal{U}$	upstream cloud
S	source load vector	C	link maxload vector
P	AS-Path length matrix	Q	tie-breaking matrix
A	padding vector	$P(A)$	padded length matrix
$L(A)$	link assignment vector	$R(A)$	link load vector

TABLE I
MAIN NOTATIONS.

teria; such a task would be extremely difficult and error-prone. We assume however that the tie-breaking behavior is determined by a matrix Q. Specifically, suppose that the two routes to $\mathcal{D}$ that originated from links $\mathcal{L}_j$ and $\mathcal{L}_k$ are received with the same AS-Path length by an AS $\mathcal{R}$ in the upstream cloud $\mathcal{U}_i$. Then, $\mathcal{R}$ will choose $\mathcal{L}_j$ if $Q_{i,j} < Q_{i,k}$, and $\mathcal{L}_k$ otherwise. Different columns of the same row of Q must be different, while their absolute values do not matter. Notice that Q is just a model of the BGP tie-breaking behavior; it is not related to a real BGP attribute. Furthermore, Q represents a *globally consistent tie-breaking behavior*, i.e., we assume that a tie between two routes to $\mathcal{D}$ with the same AS-Path length is broken in the same way in any AS in $\mathcal{U}_i$. In the next section, when we study the OPV algorithm, we assume that Q is known. Then, in § V, we relax that assumption and attempt to find a robust padding vector even when Q is completely unknown.

1. Higher local preference
2. Shorter AS path
3. Lower origin type
4. Lower MED value
5. E-BGP routes preferred over I-BGP routes
6. Lower IGP metric to next-hop
7. Lower BGP router ID

TABLE II
CRITERIA FOR BGP BEST ROUTE SELECTION.

Another key parameter is the *AS-Path length matrix* P. The element $P_{i,j}$ is the length of the shortest AS-Path from the source network $\mathcal{S}_i$ to the ingress link $\mathcal{L}_j$, where $P_{i,j}$ is a positive integer. Any ties in the length of the shortest AS-Path are broken based on the Q matrix. Note that $P_{i,j}$ is the "unpadded" AS-Path length, i.e., it should be measured before the target network applies any prepending. Estimation techniques for the matrix P are given in § IV. For now we assume that P is completely and accurately known.

The ingress link $\mathcal{L}_j$ can increase the length of the AS-Path attribute by a_j, where a_j is a non-negative integer, through prepending. The vector A=$\{a_j, j = 1 \ldots N\}$ is the *padding vector*. Given a padding vector A, the "padded" AS-Path length of the route to link $\mathcal{L}_j$ is $P_{i,j}(A) = P_{i,j} + a_j$.

Based on the previous model, we can now prove the following fact.

Lemma II.1: Given a padded AS-Path length matrix $P(A)$ and a tie-breaking matrix Q, the traffic of a source network $\mathcal{S}_i$ will enter the target network $\mathcal{T}$ through the ingress link $\mathcal{L}_j$, where

$$j = \arg \min_{1 \leqslant l \leqslant N} P_{i,l}(A) \tag{1}$$

If there is a tie in (1) between links $\mathcal{L}_j$ and $\mathcal{L}_k$, then $Q_{i,j} < Q_{i,k}$.

Proof: In the simplest case, $\mathcal{S}_i$ receives directly from $\mathcal{T}$ the route to link $\mathcal{L}_j$. Since that route has the minimum AS-Path length (or, in case of a tie, it is preferred), $\mathcal{S}_i$ will select that route for reaching $\mathcal{T}$ and the Lemma is proven.

Suppose now that $\mathcal{S}_i$ does not directly receive the route to $\mathcal{L}_j$. Let $\mathcal{R}$ be an AS in $\mathcal{U}_i$, residing in the shortest path from $\mathcal{S}_i$ to $\mathcal{L}_j$. Then, $P_{i,j}(A)$=$H_{i,r} + H_{r,j}(A)$, where $H_{i,r}$ is the length of the shortest AS-Path from $\mathcal{S}_i$ to $\mathcal{R}$, and $H_{r,j}(A)$ is the length of the shortest AS-Path from $\mathcal{R}$ to $\mathcal{L}_j$. $\mathcal{S}_i$ can only receive the $\mathcal{L}_j$ route if $\mathcal{R}$ selects that route to reach $\mathcal{T}$. To prove the Lemma, we need to show that $\mathcal{R}$ cannot select another route to reach $\mathcal{T}$.

Suppose that $\mathcal{R}$ also receives a route to $\mathcal{T}$ through link $\mathcal{L}_k$, and selects that route. This means that $H_{r,k}(A) < H_{r,j}(A)$, where $H_{r,k}(A)$ is the length of the shortest AS-Path from $\mathcal{R}$ to $\mathcal{L}_k$. Thus, $H_{i,r} + H_{r,k}(A) < H_{i,r} + H_{r,j}(A) = P_{i,j}(A)$.

Case-1: $\mathcal{S}_i$ receives the route to $\mathcal{L}_k$ also through $\mathcal{R}$. Then $P_{i,k}(A) = H_{i,r} + H_{r,k}(A)$, and so $P_{i,k}(A) < P_{i,j}(A)$. This contradicts the definition of $\mathcal{L}_j$ in (1).

Case-2: $\mathcal{S}_i$ receives the route to $\mathcal{L}_k$ through another path.

Then $P_{i,k}(A) \leq H_{i,r}+H_{r,k}(A)$, where the equality is broken in favor of that other path. Then, $P_{i,k}(A) \leq P_{i,j}(A)$, which again contradicts (1). ■

Based on the previous Lemma, we can now define the *link assignment vector* $L(A)=\{l_i(A), i = 1 \dots M\}$, where $l_i(A)$ is the link $\mathcal{L}_j$ that source $\mathcal{S}_i$ selects based on (1). From the link assignment vector $L(A)$, the expected load at an ingress link j is

$$r_j(A) = \sum_{k=1 \dots M : l_k(A)=j} s_k \qquad (2)$$

The vector $R(A)=\{r_j(A), j = 1 \dots N\}$ is the *link load vector*.

The type of INITE that we consider is based on a set of maximum load constraints for each ingress link and each customer $\mathcal{D}$. Specifically, let c_j be the maximum traffic load (maxload) allowed at link j, with $C=\{c_j, j = 1 \dots N\}$ being the corresponding *link maxload vector*. A link j is *overloaded* if $r_j(A) > c_j$. When none of the N ingress links is overloaded, we say that A is *acceptable*.

The COP problem can be now stated as follows. *Given an instance $I=(S, C, P, Q)$, determined by the source load vector S, the link maxload vector C, the AS-Path length matrix P, and the tie-breaking matrix Q, is there an acceptable padding vector A? When this is the case, determine the* optimal padding vector A^*, *such that*

$$\sum_{j=1}^{N} a_j^* \leq \sum_{j=1}^{N} a_j \qquad (3)$$

across all acceptable padding vectors A. When there is an acceptable padding vector for a given instance I, we say that I is *feasible*; otherwise I is *infeasible*. The reasoning behind the previous optimality objective is to avoid unnecessary padding, given that excessive padding in practice sometimes triggers upstream route filtering.

Example 1 illustrates the COP problem.

Instance I:

$$S = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}, C = [4,4], P = \begin{bmatrix} 1 & 2 \\ 3 & 2 \\ 2 & 5 \end{bmatrix}, Q = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 1 & 2 \end{bmatrix}$$

Without prepending: A=[0, 0]

$$L(A) = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}, R(A) = [5, 2].$$

Link $\mathcal{L}_1$ is overloaded.

After prepending: A=[2, 0]

$$P(A) = \begin{bmatrix} 3 & 2 \\ 5 & 2 \\ 4 & 5 \end{bmatrix}, L(A) = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} \quad R = [4, 3].$$

Acceptable padding vector.

Example 1

B. Remarks

Together with the previously mentioned assumptions, we also make the following assumptions regarding operational issues that may be relevant in practice:

- The destination network $\mathcal{D}$ is a customer of $\mathcal{T}$ only, so there is no way that prepending in $\mathcal{T}$ can shift the traffic away from $\mathcal{T}$ to a different provider. For multihomed destinations, our techniques would still apply as long as $\mathcal{T}$ is the primary provider, and any secondary providers of $\mathcal{D}$ are used only as a backup. Moreover, with a small modification, our algorithm can also solve the case that $\mathcal{D}$ is multihomed, which will be explained later in this paper.
- The time scales in which $\mathcal{T}$ adjusts its padding vectors are relatively short compared to the time scales in which the routes from the source networks to $\mathcal{D}$ change. Previous work has shown that BGP routes of major traffic sources tend to be stable for days or weeks [12].
- Some ISPs have an agreement with their peers that they will announce the same AS-Path to a certain destination through all their peering links. If that is the case, $\mathcal{T}$ can group together all ingress links to each peer, and then apply the proposed algorithms at the level of link groups.

III. Optimal Padding Vector Algorithm

In this section, we first present the OPV algorithm, and then prove that it can determine the optimal padding vector A^*, when the given instance I is feasible, in polynomial time.

A. Optimal Padding Vector (OPV) algorithm

Algorithm 1 OPV (I=(S, C, P, Q))

```
1:  Compute $\Phi$ from (4) and (5)
2:  $A^{(0)} = [0, \dots, 0]$;
3:  for $m = 0$ to $\Phi$-1 do
4:      Compute $L(A^{(m)})$ from (1)
5:      Compute $R(A^{(m)})$ from (2)
6:      if $A^{(m)}$: acceptable (i.e., $\forall j, r_j(A^{(m)}) \leqslant c_j$) then
7:          return $A^{(m)}$
8:      end if
9:      $A^{(m+1)} = A^{(m)}$
10:     Identify an overloaded link $j$, i.e., $r_j(A^m) > c_j$
11:     $a_j^{(m+1)} = a_j^{(m)} + 1$
12: end for
13: return $I$: Infeasible
```

The OPV algorithm takes as input an instance I=(S, C, P, Q) of the COP problem. OPV examines the feasibility of a single padding vector $A^{(m)}$ in every iteration m, starting from the zero padding vector. With each padding vector that is not acceptable, the algorithm identifies an overloaded link, and then increments the corresponding padding element. The exact selection of the overloaded link does not matter. The algorithm exits either when it has found an acceptable padding vector, or when it has completed Φ iterations.

The bound Φ is computed as follows:

$$\phi_j = \max_{1\leqslant s\leqslant M} \max_{1\leqslant i\leqslant N} (P_{s,i} - P_{s,j}) \tag{4}$$

and

$$\Phi = N + \sum_{1\leqslant j\leqslant N} \phi_j - \min_{1\leqslant j\leqslant N} \phi_j \tag{5}$$

Note that Φ can be computed in polynomial time, as shown in Lemma III.3.

As mentioned earlier in the paper, a minor change can make the algorithm applicable on multihomed network $\mathcal{D}$ of $\mathcal{T}$. When $\mathcal{D}$ is multihomed to other ISPs with the shortest path $\tilde{p_i}$ to source S_i, we change the bound ϕ to be the minimum of Equation 4 and $\tilde{p_i}$. Thus, $\mathcal{T}$ will not "over-prepend" the path so that it pushes the traffic to other providers of $\mathcal{D}$.

We prove later in this section two important properties of OPV. First, when I is feasible, OPV reports the optimal padding vector A^*, defined in (3). Second, when I is infeasible, OPV exits after Φ iterations reporting that indeed, there is no acceptable padding vector. Example 2 shows the iterations of OPV in the case of a feasible instance.

Instance I:
$S = \begin{bmatrix} 4 \\ 3 \end{bmatrix}, C = [2, 8, 2]\,, P = \begin{bmatrix} 1 & 2 & 2 \\ 1 & 2 & 2 \end{bmatrix}, Q = \begin{bmatrix} 2 & 3 & 1 \\ 1 & 3 & 2 \end{bmatrix}$

Bound Φ:
N+ $\sum \phi_j - \min \phi_j = 3 + (1 + 0 + 0) - 0 = 4$

Iteration 0:
$A^{(0)} = [0, 0, 0]$, then $L(A^{(0)}) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, R(A^{(0)}) = [7, 0, 0]$.
Link 1 is overloaded; increment a_1.

Iteration 1:
$A^{(1)} = [1, 0, 0]$, then $L(A^{(1)}) = \begin{bmatrix} 3 \\ 1 \end{bmatrix}, R(A^{(1)}) = [3, 0, 4]$.
Link 1 is overloaded; increment a_1.

Iteration 2:
$A^{(2)} = [2, 0, 0]$, then $L(A^{(2)}) = \begin{bmatrix} 3 \\ 3 \end{bmatrix}, R(A^{(2)}) = [0, 0, 7]$.
Link 3 is overloaded; increment a_3.

Iteration 3:
$A^{(3)} = [2, 0, 1]$, then $L(A^{(3)}) = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, R(A^{(3)}) = [0, 7, 0]$.
No overloaded links.
Reported vector: $A = [2, 0, 1]$.

Example 2

B. Properties of the OPV algorithm

The following lemma proves that, with a feasible instance, the optimal padding vector A^* has at least one zero element.

Lemma III.1: Suppose that the instance I is feasible, with optimal padding vector A^*. There exists a link j with $a^*_j = 0$.

Proof: Suppose $a^*_i > 0$ for all links i. Let $a^*_k = \min_{1\leqslant i\leqslant N}(a^*_i)$. Construct a padding vector A' such that $a'_i = a^*_i - a^*_k$. From (2), $R(A') = R(A^*)$. Since A^* is acceptable, A' is also acceptable. But $\sum a'_i < \sum a^*_i$, which contradicts that A^* is the optimal padding vector. So, there must exist a link with $a^*_j = 0$. ■

A key property of OPV, proven next, is that, in a feasible instance, if an element k of the padding vector reaches in some iteration m the value that the corresponding optimal padding vector element has, i.e., if $a^{(m)}_k = a^*_k$, then the link k will not be overloaded in any subsequent iteration, and so its padding element will remain at a^*_k.

Lemma III.2: Suppose that the instance I is feasible, with optimal padding vector A^*. If the OPV padding vector in iteration m is such that $a^{(m)}_j \leqslant a^*_j$ for all links j and there exists a link k such that $a^{(m)}_k = a^*_k$, then in all subsequent iterations n ($n > m$), $a^{(n)}_k = a^{(m)}_k = a^*_k$.

Proof: We prove the lemma by contradiction. Suppose that the padding of link k is the first to exceed the corresponding value a^*_k of A^* after the n-th iteration. In other words, in iteration n we have that $a^{(n)}_k = a^*_k$ and $a^{(n)}_j \leqslant a^*_j$ for all j, while in the next iteration $a^{(n+1)}_k > a^*_k$.

According to OPV, the padding of link k can only be increased if that link is overloaded. So, in iteration n there was at least one source s such that $l_s(A^{(n)}) = k$, but $l_s(A^*) = h \neq k$. The fact that source s preferred link k over link h in iteration n means that

$$P_{s,k}(A^{(n)}) = P_{s,k} + a^{(n)}_k \leq P_{s,h} + a^{(n)}_h$$

where the case of equality is broken in favor of link k through the Q matrix, and so

$$P_{s,k} - P_{s,h} \leq a^{(n)}_h - a^{(n)}_k \tag{6}$$

In the optimal vector A^*,

$$P_{s,k}(A^*) = P_{s,k} + a^*_k > P_{s,h} + a^*_h$$

and so

$$P_{s,k} - P_{s,h} > a^*_h - a^*_k \tag{7}$$

From (6) and (7) we get that $a_h^* - a_k^* < a_h^{(n)} - a_k^{(n)}$. But $a_k^* = a_k^{(n)}$, and so $a_h^* < a_h^{(n)}$, which contradicts our earlier assumption. ■

The following lemma shows that, for a feasible instance, each optimal padding element is upper bounded by a certain function of the AS-Path length matrix that can be computed in polynomial time.

Lemma III.3: Suppose that the instance I is feasible, with optimal padding vector A^*. Then, for any link j the corresponding optimal padding is bounded by

$$a_j^* \leqslant \phi_j + 1 \tag{8}$$

where ϕ_j is given by (4).

Proof: We prove the lemma by contradiction. Suppose that there exists a link i with $a_i^* > \phi_i + 1$. Let us construct a new padding vector A' such that $a'_j = a_j^*$ for all links $j \neq i$, and $a'_i = \phi_i + 1$. We prove next that A' is also acceptable.

First, suppose that $r_i(A') > r_i(A^*)$. This means that there exists a source s with $l_s(A') = i$ and $l_s(A^*) \neq i$. So, for all links $j \neq i$

$$P_{s,i} + a'_i = P_{s,i} + \phi_i + 1 \leqslant P_{s,j} + a_j^*$$

where all ties (equalities) are broken in favor of link i, and so

$$\phi_i + 1 - a_j^* \leqslant P_{s,j} - P_{s,i} \leqslant \phi_i$$

which means that $a_j^* \geqslant 1$ for all links $j \neq i$. We also assumed that $a_i^* > \phi_i + 1 > 1$, and so $a_j^* \geqslant 1$ for all links j (including i). This contradicts Lemma III.1.

From the previous contradiction, we have that $r_i(A') = r_i(A^*) < c_i$, i.e., link i is not overloaded with A'. Any other link $j \neq i$ cannot be overloaded with A' either, because those links are not overloaded with A^*, $a'_j = a_j^*$ for all $j \neq i$, and $a'_i < a_i^*$. So, the padding vector A' is acceptable, which contradicts (3) because $\sum a'_j < \sum a_j^*$. Consequently, $a_j^* \leqslant \phi_j + 1$ for all links j. ■

The following theorem gives the main result for the OPV algorithm, for the case of feasible instances.

Theorem III.4: Suppose that the instance I is feasible, with optimal padding vector A^*. The OPV algorithm returns the vector A^* as its final outcome in polynomial time.

Proof: Note that the OPV algorithm starts from the zero vector and it increments only one padding element in every iteration. Furthermore, Lemma III.2 shows that if an element of the OPV padding vector has reached its corresponding value in A^*, then it will not increase any further in subsequent iterations. Consequently, it is certain that after $1 + \sum a_j^*$ iterations the OPV algorithm will terminate, reporting the optimal padding vector A^* as its final outcome.

From Lemma III.3, we get that the number of required iterations is $1 + \sum_{j=1}^{N} a_j^* \leq (N+1) + \sum_{j=1}^{N} \phi_j$. We also know from Lemma III.1, however, that at least one element of A^* is zero. In the worst-case (maximum number of iterations), that element can be the link with the minimum ϕ_j term. Thus, the required number of iterations becomes

$$1+\sum_{j=1}^{N} a_j^* \leq (N+1)+\sum_{j=1}^{N} \phi_j-(\min_{1\leqslant j\leqslant N} \phi_j+1) = \Phi \tag{9}$$

■

The following theorem gives the main result for the OPV algorithm in the case of infeasible instances.

Theorem III.5: If instance I is infeasible, then the OPV algorithm detects that there is no acceptable padding vector in polynomial time.

Proof: Note that the OPV algorithm fails to find an acceptable padding vector after Φ iterations, it exits reporting that the instance is infeasible.

If I was feasible, then, from Theorem III.4, OPV would have reported A^* in at most Φ iterations. Since an acceptable padding vector has not been found after Φ iterations, then it does not exist. ■

Example 2 demonstrates that the bound Φ of (9) is actually tight. The value of Φ in that example is 4.

IV. Estimation of Input Parameters

In this section, we discuss the two key unknown inputs of the OPV algorithm, namely the set of super-source networks and the corresponding source load vector S, and the length estimation matrix P. As mentioned in § II, we do not attempt to estimate the tie-breaking matrix Q; the algorithm of the following section determines a robust padding vector independent of Q. Also, the maxload vector C is supposed to be known given that it is chosen by the target network operator.

A. Selection of super-sources

We assume that the ingress routers of $\mathcal{T}$ collect statistics (with Cisco's NetFlow for instance) of the arriving traffic, aggregated by source network; this is a common requirement for any type of traffic engineering. Because there may be too many source networks for a given customer $\mathcal{D}$, or because those networks may not be in upstream clouds, $\mathcal{T}$ can map one or more large sources of traffic to a single super-source $\mathcal{S}$. The requirement is that

$\mathcal{S}$ has to be in the AS-Path from the actual source networks to any ingress link of $\mathcal{T}$.

To examine the former requirement, $\mathcal{T}$ can use AS-level topology maps, constructed with multiple vantage points as described in [13]. For example, the NetFlow data may show that several major sources of traffic for $\mathcal{D}$ belong to three stub ASes that are all customers of a regional or tier-2 provider AS-X, and that those three stub networks are not multi-homed (i.e., they can only reach $\mathcal{T}$ through AS-X). In that case, AS-X can be considered as a super-source $\mathcal{S}_i$. The corresponding source load element s_i would be estimated as the sum of the load estimates of the actual sources, measured with NetFlow. The previous procedure can be applied to detect M super-sources, capturing a large fraction of the traffic to $\mathcal{D}$.

B. Estimation of AS-Path length matrix

Recall that $P_{i,j}$ is the AS-Path length from source $\mathcal{S}_i$ to the ingress link $\mathcal{L}_j$. To estimate $P_{i,j}$, $\mathcal{T}$ has to originate a route for a *wayfinding prefix* at each ingress link $\mathcal{L}_j$. A wayfinding prefix can be just the IP address of $\mathcal{L}_j$; if that is an unacceptably long prefix, the wayfinding prefix can cover instead a small part of the target network's address space that includes $\mathcal{L}_j$ but no other ingress links. The same set of wayfinding prefixes can be used for estimating the matrix P for all customers of $\mathcal{T}$.

To estimate $P_{i,j}$, the operator of $\mathcal{T}$ can use one of the four following techniques, in the given order. The first three techniques rely on *Looking Glass Servers*. LGS's are abundant in the Internet today, providing a "peek" inside a network's BGP routing tables. They are mostly used by ISPs for problem diagnosis and monitoring of interdomain routing. The publicly available LGS's that are listed in *www.traceroute.org* include 273 servers that in some cases provide access to multiple routers within an AS [14]. It is likely however that major ISPs have private access to even more LGS's in remote networks, to accommodate synergistic problem diagnosis.

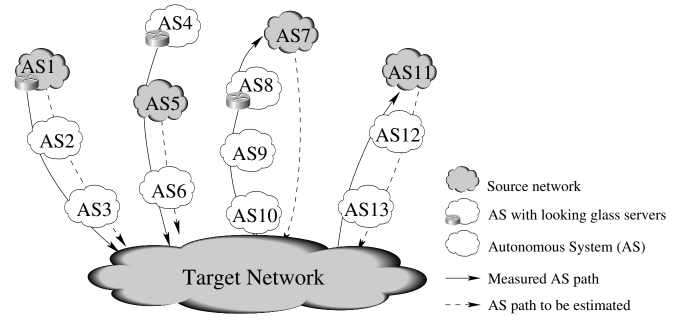


Fig. 3. Four estimation techniques for the AS-Path length.

1) LGS in $\mathcal{S}$: The source network $\mathcal{S}$ may include an LGS. This is the ideal case, and it leads to the most accurate estimation. In the example of Figure 3, AS1 is a source network that deploys an LGS. $\mathcal{T}$ can just query that LGS for the AS-Path to each wayfinding prefix.

2) LGS in a customer of $\mathcal{S}$: Here, a customer of $\mathcal{S}$ deploys an LGS. In that case, $\mathcal{T}$ can query that LGS for each wayfinding prefix, and then remove from the returned AS-Paths the part that includes that customer AS. In Figure 3, AS4 is a customer of the source network AS5. AS4 deploys an LGS.

3) LGS in the path from $\mathcal{T}$ to $\mathcal{S}$: Another possibility is that $\mathcal{T}$ can locate an LGS in a network in the reverse path, from $\mathcal{T}$ to $\mathcal{S}$. In Figure 3, AS7 is a source network and AS8 is a network deploying an LGS in the reverse path. In that case, $\mathcal{T}$ can estimate the AS-Path length $P_{i,j}$ as the sum of the AS-Path lengths from the LGS to $\mathcal{L}_j$ and from the LGS to $\mathcal{S}$. This technique assumes that the AS-Paths from the LGS to $\mathcal{S}$ and from $\mathcal{S}$ to the LGS have the same length.

4) Reverse path estimation: When none of the previous techniques is applicable, $\mathcal{T}$ can just estimate the length of the AS-Path from $\mathcal{S}$ to $\mathcal{L}_j$ based on the length of the reverse path from $\mathcal{L}_j$ to $\mathcal{S}$. That reverse path is of course directly available from the border router at $\mathcal{L}_j$. The problem with this technique is that it relies on the symmetry of the forward and reverse AS-Paths, which is often not the case in the Internet. On the positive side, the technique still produces a correct estimate if the two paths have the same length, even if those paths are different. This case is represented in the path between $\mathcal{T}$ and AS11 in Figure 3.

AS-Path length distribution and estimation errors*:* The first two of the four previous techniques would give no estimation errors, because the corresponding LGS's report directly the AS-Path from the source to the ingress links of the target network.

The third and fourth techniques, on the other hand, depend on the symmetry assumption, and they will probably suffer from estimation errors. To quantify these errors, we used 79 of the publicly available LGS's listed at [14]. We estimated the AS-Path length from every LGS to each of the 78 other LGS's using the previous two techniques: "LGS in the reverse path" (when applicable), and "reverse path estimation" (always applicable). Then, we compared each AS-Path length estimate with the length of the actual AS-Path, as that was reported in the remote LGS.

Figure 4 shows the empirical probability density function for the estimated AS-Path lengths. Note that about 90-95% of the paths are up to 6 AS hops, including any prepending, while 60% of the paths are either 4 or 5 hops.

The unconditional estimation error for the previous two estimation techniques is as follows: 60-65% of the esti-

	e=-6	e=-5	e=-4	e=-3	e=-2	e=-1	e=0	e=1	e=2	e=3	e=4	e=5	e=6
$\hat{p}$=3	0.2	0.8	2.5	2.3	4.4	10.6	78.0	1.3	0.0	0.0	0.0	0.0	0.0
$\hat{p}$=4	0.0	1.0	3.8	2.0	3.2	12.4	70.0	7.4	0.3	0.0	0.0	0.0	0.0
$\hat{p}$=5	0.0	1.0	3.1	0.4	3.0	12.4	64.1	11.8	3.8	0.1	0.3	0.0	0.0
$\hat{p}$=6	0.0	0.2	1.9	3.1	2.7	9.0	57.8	19.1	5.5	0.6	0.0	0.0	0.0
$\hat{p}$=7	0.0	0.0	0.5	1.0	2.0	12.1	45.5	21.2	11.1	6.6	0.0	0.0	0.0
$\hat{p}$=8	0.0	0.0	0.0	0.0	1.5	1.5	4.5	34.8	22.7	13.6	16.7	4.5	0.0
$\hat{p}$=9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.5	7.5	25.0	37.5	25.0	2.5
$\hat{p}$=10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	13.3	6.7	40.0	40.0	0.0
$\hat{p}$=11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	25.0	50.0	25.0
$\hat{p}$=12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0

TABLE III

PROBABILITY (%) OF AS-PATH LENGTH ESTIMATION ERROR e, CONDITIONED ON THE ESTIMATED AS-PATH LENGTH $\hat{p}$ (BOTH e AND $\hat{p}$ MEASURED IN AS HOPS).

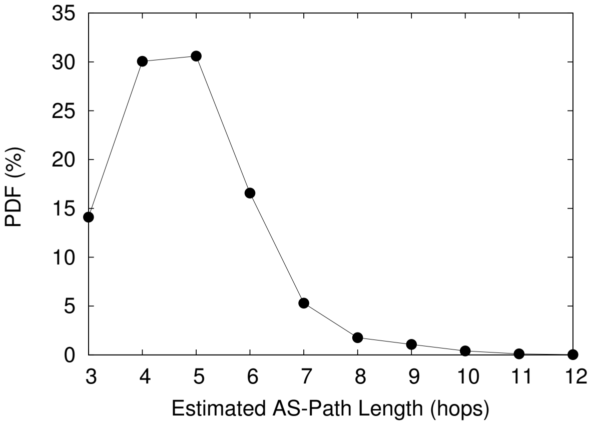


Fig. 4. Empirical probability distribution function of estimated AS-Path length.

mates have zero error, 85-90% have an error of less than ± 1 hop, and 95% have an error of less than ± 2 hops. More importantly, Table III shows the conditional probability of an estimation error e, given the estimated AS-Path length $\hat{p}$. The estimation error e is measured as $\hat{p} - p$, where p is the actual AS-Path length. The conditional probability that an estimate is error-free if $\hat{p}$ is 3 or 4 hops is 78% and 70%, respectively, which is higher than the corresponding unconditional accuracy. Similarly, the conditional probability that an estimate has an error of less than ± 1 hop if $\hat{p}$ is 5 or 6 hops is 88% and 86%, respectively. We use this empirical error distribution, as well as the AS-Path length distribution of Figure 4, in the robustness study of the next section.

V. ROBUST PADDING VECTOR ALGORITHM

The previous section showed that the estimate of the AS-Path length matrix P may include errors, and it gave us an empirical distribution for the estimation error probability. In this section, we first describe an algorithm that aims to determine a robust padding vector in the presence of these errors, and then evaluate the effectiveness of that algorithm with simulations.

A. Robust Padding Vector Algorithm

Recall that an instance I of the COP problem is defined by the set (S, C, P, Q). The OPT algorithm of § III was developed to find an optimal padding vector A^* for I. In practice, we do not have a way to estimate Q, and P may include estimation errors. To deal with these two issues, we develop the *RPV (Robust Padding Vector)* algorithm.

The basic idea in RPV is the following. Suppose that our original estimate of P is P_0. P_0, together with an arbitrary tie-breaking matrix Q and the given S and C vectors, form the instance I_0 that we start from. Another important instance is I_a; this is the unknown *actual instance* that is based on the correct AS-Path length matrix and the real tie-breaking behavior[1]. Even though I_a is unknown, it belongs to the space $\mathcal{I}$ of all possible instances that can be generated by applying a given error model to P_0, and by considering an arbitrary tie-breaking behaviors.

First, RPV generates a subset $\mathcal{I}_X$ of $\mathcal{I}$ that consists of X feasible instances. For each instance I_i in $\mathcal{I}_X$, the corresponding optimal padding vector (computed with OPV) is A_i. The set of padding vectors $\mathcal{A}$=$\{A_i, i = 1 \ldots X\}$ is our candidates for the desired robust solution. To generate padding vectors that differ significantly, we create the subset $\mathcal{I}_X$ by applying the AS-Path length estimation error model only to the largest sources of I_0. Errors that correspond to small sources (relative to the rest of the sources) may not have an impact on the resulting padding vector.

Second, RPV generates a large subset $\mathcal{I}_Y$ of $\mathcal{I}$ that includes Y instances. The fraction of feasible instances in $\mathcal{I}_Y$ is the *feasibility index* of $\mathcal{I}_Y$. If Y is sufficiently large, the feasibility index estimates the probability that the actual instance I_a is feasible.

[1]We still assume that the real tie-breaking behavior can be captured by a matrix such as Q.

Third, for each candidate padding vector A_i in $\mathcal{A}$, RPV measures the fraction of *feasible* instances in $\mathcal{I}_Y$ for which A_i is acceptable. That fraction is the *robustness index* of A_i. If Y is large, the robustness index estimates the probability that the corresponding padding vector A_i is acceptable for the actual instance I_a, conditioned on the fact that the latter is feasible.

Eventually, RPV reports the padding vector $\tilde{A}$ with the maximum robustness index. The reason is that that vector maximizes the likelihood that it will be acceptable for I_a. The higher Y is, the more samples we collect from the space of possible instances, and so the more reliable our robustness index will be. The robustness, on the other hand, can be increased if we increase X.

B. Robustness evaluation

We evaluate the robustness of the padding vector that RPV reports using simulations. In the following, we consider a target network with N=5 ingress links that have the same maxload constraint (i.e., C_j=C for all j). The number of source networks is M=100. The source load distribution is the same for all sources, and it follows one of the following models: Uniform, Exponential, and Pareto (shape parameter: 1.7). The mean source load $\bar{s}$ is the same in all distributions. The AS-Path length matrix P_0 is generated based on the empirical distribution shown in Figure 4. The error model that is applied to P_0 is based on the conditional estimation errors of Table III. The RPV algorithm uses X=100 and Y=10000 instances. $\mathcal{I}_X$ is formed by applying errors to the set of sources that generate, in total, at least 80% of the aggregate load.

The feasibility index depends on the relation between the aggregate offered load $M\bar{s}$ and the aggregate maxload constraint NC. Given the source load vector S, the following fraction ρ is our *load metric*,

$$\rho = \frac{\sum_{i=1}^{M} s_i}{NC} \tag{10}$$

ρ represents well the tightness of the given resource allocation problem for the homogeneous maxload constraints that we consider, and of course it should be less than 100% for any instance to be feasible. To achieve a certain load ρ given an instance with a source load vector S, we calculate the required C from (10).

Figure 5 shows CDFs for the feasibility index and the robustness index of $\tilde{A}$ for each of the previous three load distribution models, and for three load conditions (ρ=0.5, 0.6, and 0.7). Each CDF resulted from 1000 executions of the RPV algorithm with a different original instance I_0. The CDF curves that are not visible in the graphs are equal to 100% for all instances.

Note that even with the Uniform distribution for S, which is the most likely among the three to produce feasible instances, the feasibility index drops below 90% when ρ is 0.7 (Fig 5(a)). For the heavy-tailed Pareto sources, the feasibility is often below 90% even when ρ=0.6 (Fig 5(c)). The feasibility could be even lower if we had simulated non-homogeneous maxload constraints across different links.

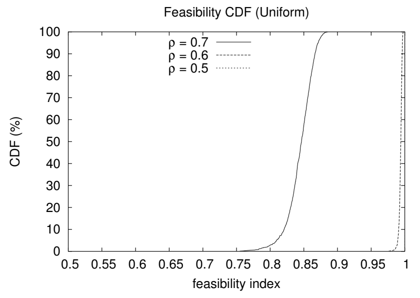
Figure 5 shows that the robustness index of the padding vector that RPV reports is larger than 90% for all three load conditions, with both the Uniform and Exponential distributions. With the Pareto distribution, on the other hand, the robustness can be lower than 90% in 10-20% of the cases, but rarely lower than 80% (Fig 5(f)). Note that a higher load ρ does not necessarily mean a lower robustness index. The reason is that the latter is conditioned on feasible instances only. It can happen that even though the feasibility index is low, a large number of padding vectors are acceptable for the few feasible instances.

The overall conclusion from these results is that RPV can produce a robust padding vector $\tilde{A}$, in the sense that that vector will be acceptable for the actual instance I_a with a probability of more than 80-90%, if the latter is feasible. For I_a to be feasible with a high probability however, say more than 90%, the load metric ρ should be below 50-70%, depending on the distribution of S, at least for the homogeneous maxload constraints that we simulated here.

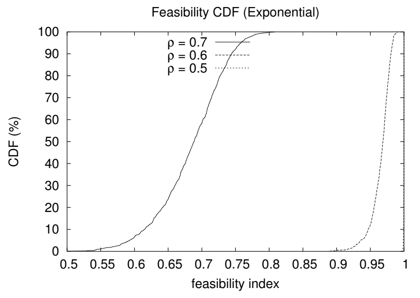
VI. Related Work

An excellent survey for interdomain TE appears in [7]. The authors explain why INITE is harder than other types of TE. They also summarize the main methods to perform INITE, namely selective/different advertisements (or advertisement of more specific prefixes) on different peering links, AS-Path prepending, use of the MED attribute in ASes that have multiple peering links, and use of various BGP communities to signal TE decisions between different ASes.

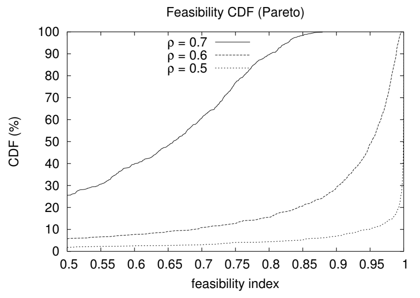
The feasibility of interdomain TE by a stub AS was examined in [15] (without considering a particular TE mechanism). That work concluded that, despite the important variability of interdomain flows, it would be useful for a stub AS to engineer its ingress traffic, aggregated at an appropriate level. In a follow-up work, based on more recent measurements, the authors made more pessimistic conclusions regarding the feasibility of interdomain traffic engineering [16]. At least for their UCL and PSC traces (both stub networks), they showed that there was significant temporal variability in the traffic carried by each major AS path. Furthermore, due to limited aggregation at



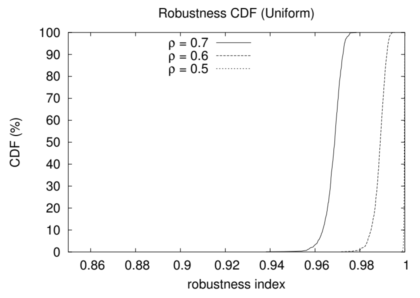
(a) Feasibility, Uniform



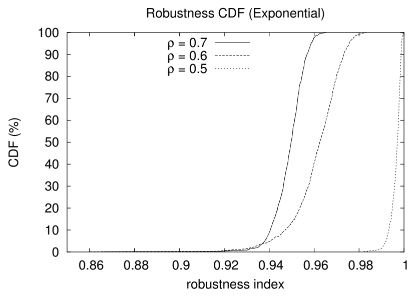
(b) Feasibility, Exponential



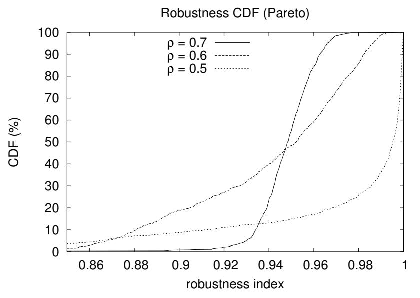
(c) Feasibility, Pareto



(d) Robustness, Uniform



(e) Robustness, Exponential



(f) Robustness, Pareto

Fig. 5. Feasibility and robustness for three source load distributions and load conditions. Note that the CDF curves that are not visible are equal to 100% for all instances.

the AS level, they argued that to effectively control ingress traffic one would need to affect the route selection in a large number of ASes.

The use of the BGP community attribute for INITE has been studied in depth in [17]. The authors presented various drawbacks of using that attribute: the requirement for a manually encoded filter for each supported community, the fact that each AS must advertise the semantics of its own community values to peers, and the transitivity of that attribute. They proposed the use of a new standardized form of extended community, referred to as "redistribution community", which supports the following actions (among others): the attached route should not be announced, or it should be announced only to specific BGP speakers, and that the attached route should be prepended a number of times when announced to specific peers.

The Internet Draft [18] proposes a community, referred to as BGP PCC, that carries another community to be attached to a route, when the latter is sent to a distant AS. The PCC value, viewed as a tuple (AS-X, AS-Y, c), represents a request directed from the originating AS to AS-X to send community c to AS-Y when the associated route is sent from AS-X to AS-Y. The PCC community can be used for INITE, by allowing an originating AS to affect the route selection process in a remote AS.

Instead of adding extensions to BGP, Agarwal et al. proposed an Overlay Policy Control Architecture (OPCA) [19]. A major motivation behind OPCA is to support INITE. OPCA is basically an overlay network running on top of BGP to facilitate policy exchanges between ASes. It consists of a set of policy agents deployed in the participating ASes that communicate through an overlay protocol to process external policy announcements and negotiate with remote ASes the selection of interdomain paths for ingress traffic.

Feamster et al. focused on egress interdomain TE [8]. They showed how to move traffic in a predictable fashion by tuning certain BGP policies, and also how to limit the influence of neighboring domains on the local path selection process through BGP policies. That work includes measurements (from the AT&T network) for the frequency and extent of AS-Path prepending: 32% of the routes have some form or prepending, with about 90% of the corresponding paths extended by 1-5 hops, while the maximum prepending was 16 hops.

Uhlig et al. also focused on egress interdomain TE in [20]. They modeled the egress TE problem as follows: given m destination networks, a cost function for each of p downstream providers, and the constraint that only n BGP filters can be configured, the objective is to find the best n filters for the m destinations and the p providers, starting from the default BGP configuration.

Bressoud and Rastogi examined the intradomain problem of choosing the optimal set of border routers for the advertisement of a set of routes from a transit provider. The objective is to minimize the cost of traffic across that provider while meeting certain capacity constraints at the border routers [21]. The mathematical formulation of that work has some similarities with our formulation. Note however that the problem considered in [21], which is a variation of the Generalized Assignment Problem (GAP), is NP-Hard, while COP can be solved in polynomial time. The reason is that the mapping of sources to links is more constrained in COP than in GAP.

VII. Conclusions

INITE is challenging mostly because it requires that an AS is able to affect BGP routing decisions in remote ASes. Previous research proposals in this area have focused on the use of special BGP communities, which require active cooperation from the upstream ASes. In practice, ISPs have been using AS-Path prepending to control the flow of ingress traffic. Prepending is widely viewed, however, as an ad-hoc technique and it has not received much attention in the related literature. In this work, we made a first step to explore the use of prepending in a more algorithmic framework, and to explore its potential and limitations.

The main contribution of this paper at the more theoretical level is to present a polynomial-time algorithm (OPV) that can determine the optimal padding vector given constraints on the maximum load of each ingress link. Even though OPV relies on accurate information of several parameters that can only be roughly estimated in practice, it is still important because it provides the best-case scenario for the effectiveness of prepending if all the required information was available. At the more practical level, the contribution of the paper is to describe how to apply prepending in a robust manner, considering that the AS-Path length information may be subject to estimation errors and the tie-breaking behavior is unknown. Interestingly, our simulations show that it is possible to determine an acceptable padding vector in that case as well, as long as the maximum load constraints are not too tight.

The greatest limitation of AS-Path prepending, and consequently of our work, is that prepending can affect the route selection process only if the upstream ASes do not use the Local Preference attribute to enforce policy decisions. It remains an open issue whether ISPs can identify branching nodes with restrained usage of Local Preference values in their upstream clouds in a simple and accurate manner. When that is the case, however, we believe that the results in this paper can be used to guide the prepending process in a systematic and more well-understood manner.

References

[1] D. Awduche, A. Chui, A. Elwalid, I. Wiljaja, and X. Xiao, “Internet RFC 3272: Overview and principles of Internet traffic engineering,” May 2002.

[2] B. Fortz, J. Rexford, and M. Thorup, “Traffic engineering with traditional IP routing protocols,” *IEEE Communications Magazine*, October 2002.

[3] A. Feldmann and J. Rexford, “IP network configuration for intradomain traffic engineering,” *IEEE Communications Magazine*, October 2001.

[4] B. Fortz and M. Thorup, “Internet traffic engineering by optimizing OSPF weight,” in *IEEE INFOCOM*, 2000.

[5] M. Kodialam and T. V. Lakshman, “Minimum interference routing with applications to MPLS traffic engineering,” in *IEEE INFOCOM*, 2000.

[6] A. Elwalid, C. Jin, S. Low, and I. Widjaja, “MATE: MPLS adaptive traffic engineering,” in *IEEE INFOCOM*, 2001.

[7] B. Quoitin, S. Uhlig, C. Pelsser, L. Swinnen, and O. Bonaventure, “Interdomain traffic engineering with BGP,” *IEEE Communications Magazine*, May 2003.

[8] N. Feamster, J. Borkenhagen, and J. Rexford, “Guidelines for interdomain traffic engineering,” in *ACM SIGCOMM Computer Communications Review*, 2003.

[9] J. W. Stewart, *BGP4: Inter-Domain Routing in the Internet*, Addison Wesley Longman, Inc., 1999.

[10] Cisco, “BGP best path selection algorithm http://www.cisco.com/warp/public/459/25.shtml,” .

[11] Juniper, “Route preferences http://www.juniper.net/techpubs/software/junos/junos63/swconfig63-routing/html/protocols-overview4.html,” .

[12] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, “BGP routing stability of popular destinations,” in *ACM SIGCOMM Internet Measurement Workshop*, 2002.

[13] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz, “Characterizing the Internet hierarchy from multiple vantage points,” in *IEEE INFOCOM*, 2002.

[14] T. Kernen, “www.traceroute.org,” .

[15] S. Uhlig and O. Bonaventure, “Implications of interdomain traffic characteristics on traffic engineering,” *Special issue on traffic engineering of European Transactions on Telecommunications*, 2002.

[16] S. Uhlig, V. Magnin, O. Bonaventure, C. Rapier, and L. Deri, “Implications of the topological properties of Internet traffic on traffic engineering,” in *19th ACM Symposium on Applied Computing, Special Track on Computer Networks*, 2004.

[17] B. Quoitin, S. Tandel, S. Uhlig, and O. Bonaventure, “Using redistribution communities for interdomain traffic engineering,” *Computer Communications Journal*, pp. 355–363, mar 2004.

[18] S. Agarwal and T. G. Griffin, “BGP proxy community community,” http://www.ietf.org/internet-drafts/draft-agarwal-bgp-proxy-community-00.txt, Janurary 2004.

[19] S. Agarwal, C.-Nee Chuah, and R. H. Katz, “OPCA: Robust interdomain policy routing and traffic control,” in *The 6th IEEE Conference on Open Architectures and Network Programming*, 2003.

[20] S. Uhlig, O. Bonaventure, and B. Quoitin, "Interdomain traffic engineering with minimal BGP configurations," in *18th International Teletraffic Congress*, 2003.

[21] T. Bressoud, R. Rastogi, and M. Smith, "Optimal configuration for BGP route selection," in *IEEE INFOCOM*, 2003.