Enumerating Single Destination, Policy-Preferred Paths in AS-level Internet Topology Maps

Mehmet Engin Tozal
School of Computing and Informatics
University of Louisiana, Lafayette, LA 70504 USA
Email: metozal@louisiana.edu

***Abstract*—Using AS-level Internet topology maps to determine accurate AS-level paths is essential for network diagnosis, performance optimization, reliability improvement, resiliency enforcement and topology-aware application development. One significant drawback that we have observed in many studies is simplifying the AS-level topology map of the Internet to an undirected graph, and then using the hop distance as a means to find the shortest paths between ASes. A less significant drawback is restricting the shortest paths to only valley-free paths. Both approaches usually inflate the number of paths between ASes; introduce erroneous paths that do not conform to economic policies; and generate symmetric paths, which in reality is not a rule. As a result, the derived conclusions might be greatly misleading. In this study we introduce a single-destination, policy-preferred path enumeration algorithm which discovers policy consistent paths from all ASes to a destination AS in an AS-level Internet topology graph. Considering that our algorithm's run time complexity is the same as Dijkstra's shortest paths algorithm, we believe that the proposed algorithm will notably enhance the future works that leverage AS-level Internet topology paths.**

Keywords*—*Internet, Autonomous System, Graph, Path

I. Introduction

Internet's global infrastructure consists of tens of thousands of Autonomous Systems (ASes) connected to each other. ASes in the Internet adhere to the BGP [19] protocol to exchange and propagate inter-AS routing and reachability information. AS-level routing decisions largely depend on economic incentives and business relations between ASes rather than performance metrics. That is, inter-AS routes are typically congruent with the business relations among ASes [16].

Traditionally, business relations among ASes are categorized as customer-to-provider (c2p), peer-to-peer (p2p) and sibling-to-sibling (s2s) [8]. In a c2p relation, the provider AS provides global reachability to its customer AS. In return, the customer pays to the provider for the traffic exchanged between them. In a p2p relation, two peer ASes provide mutual reachability only to each other and their customer ASes. Peer ASes typically engage in settlement-free business agreements which means that neither party pays to the other for the traffic exchanged. In the less frequently observed s2s relation, two ASes provide full reachability to each other because they usually are operated by the same organization. More complex relations such as hybrid and partial relations are reported in the Internet as well [9]. However, c2p and p2p relations abstract the majority of the business agreements between ASes for practical purposes [16].

Using AS-level Internet topology maps to determine accurate AS-level paths is essential for network diagnosis, performance optimization, reliability improvement and topology-aware application development. To illustrate, inferring AS-level paths allows us (i) to detect congestion points which mostly occur on the links between ISPs [1]; (ii) to control effective neighbor selection in P2P networks [15]; (iii) to leverage the quality of VoIP services and reduce the traffic overhead [20]; (iv) to develop techniques for passive network delay estimation [14]; (v) to optimize server deployment in content delivery networks [10]; (vi) to analyze failures and determine reliability bottlenecks in the Internet [6]; and (vii) to generate synthetic network topologies [2].

One significant drawback in many studies is simplifying the AS-level topology map of the Internet to an undirected graph, and then using the hop distance as a means to find the shortest paths between ASes [2], [15], [20], [10]. Although such an approach facilitates the discovery of shortest paths between ASes for practical purposes, the derived conclusions might be greatly misleading. An AS in fact, may prefer a longer path to reach another AS, if the longer path is economically more advantageous.

A less significant drawback is restricting the shortest paths to only valley-free paths [6], [17], [12], [14]. A valley-free AS consists of zero or more c2p links followed by zero or one p2p link, and then zero or more p2c links. Although this approach is comparably better than the shortest paths, the valley-free property is not solely enough to reveal the policy consistent paths in an AS-level topology graph. An equivalently important property is the AS relation expressed at the first hop link of a path. Assuming that there are multiple valley-free paths from a source AS to a destination AS, the source AS prefers a path starting with an edge oriented to a customer AS compared to an edge oriented to a peer AS. Similarly, the source AS prefers a path starting with an edge oriented to a peer AS compared to an edge oriented to a provider AS. The incentive is again economics: often, a customer pays to a provider and peers do not pay to each other for the exchanged traffic.

As a result, both approaches usually inflate the number of paths between ASes; introduce erroneous paths that do not conform to economic policies; and/or generate symmetric paths, which in reality is not a rule.

In this study we introduce a single-destination, policy-preferred path enumeration algorithm which discovers policy consistent paths from all ASes to a given destination AS in an AS-level Internet topology graph. Our algorithm provides a holistic solution to the AS-level path enumeration problem by incorporating common practices and incentives in the inter-AS routing including shortest-distance preferred paths, valley-free preferred paths and first-hop-edge policy preferred paths. Given an AS-level Internet topology graph and a destination vertex, the algorithm starts from the destination vertex and incrementally builds AS paths in backwards from source

vertices toward the destination vertex. At each iteration, a new vertex is joined to the subgraph of the established, policy-preferred paths toward the destination vertex via one or more edges. At the end, the algorithm returns a rooted, directed, acyclic subgraph (r-DAG) of the input graph, which is formed by policy-preferred paths from the source vertices toward the destination vertex. Our algorithm runs in time $O((|V|+|E|)log|V|)$ where $|V|$ is the number of vertices and $|E|$ is the number of edges in the AS-level Internet topology graph. The time complexity of our algorithm is the same as Dijkstra's shortest paths algorithm with a priority queue implementation.

We believe that the proposed algorithm will notably affect the future endeavors that leverage AS-level Internet paths for network diagnosis, performance optimization, reliability improvement, resiliency enforcement and topology-aware application development.

The rest of the paper is organized as follows. Next section introduces the related work. Section III presents a graph model to represent AS-level Internet topology maps. We introduce the definition of the policy-preferred AS paths problem and a solution to the problem in Sections IV and V, respectively. In Section VI we present an empirical analysis of our solution on a real world dataset. Finally, Section VII concludes the paper.

II. Related Work

AS-level path inference, AS relation inference and AS-level Internet topology mapping have been active research fields in the last two decades [5], [16], [22], [18].

Many techniques in these fields can be categorized based on the source(s) that they employ in mapping and inference. Path trace based approaches use `traceroute`-like tools to collect path traces from multiple vantage points and employ IP address to AS number mapping techniques to build the links between ASes [4], [7]. Internet Routing Registry (IRR) databases and BGP looking glasses (LG) are usually used to augment existing AS-level Internet topologies [11], [13]. BGP routing table based approaches passively collect BGP updates and use the advertised paths to construct an AS-level topology map of the Internet [24], [23]. Most of these studies focus on not only mapping the Internet at the AS-level but also inferring the types of business relations between ASes [8], [9]. Different techniques have been introduced for AS-level path inference as well. Qiu and Gao proposed an algorithm [18] that incrementally builds new paths by extending a path set that is directly obtained from BGP routing tables. Mao et al.'s `RouteScope` algorithm [17] restricts the shortest paths between ASes to valley-free paths. In a more recent work, Tao et al. suggested a method [22] based on hyperbolicity metric.

In this study we propose a novel algorithm to enumerate policy consistent paths in an AS-level Internet topology graph. Our work is complementary to existing efforts in the sense that we leverage Internet topology maps collected, constructed and annotated by the existing projects.

III. AS-level Internet Graph Representation

Traditionally, an AS-level Internet topology map is modeled as a mixed, finite graph. In our work we found such a representation restrictive due to the complexity of handling mixed edges. In this study we model the AS-level Internet topology map as an edge labeled, finite, directed graph $G = (V, E, \Phi)$ comprising of a set of vertices $V = \{v_1, v_2, v_3, \ldots, v_{|V|}\}$ that represent the autonomous systems; a set of directed edges $E \subseteq \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$ denoting a logical connection from v_i to v_j; and an edge labeling function

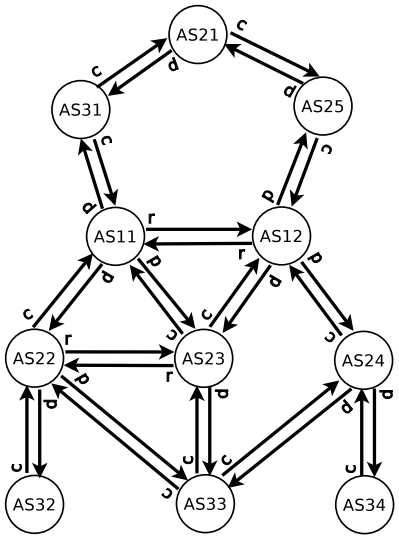

Fig. 1: AS-level Internet topology map representation

$\Phi : E \rightarrow \mathcal{L} = \{customer, provider, peer\}$ associating a business policy label to an edge (v_i, v_j) which reflects the business relation appointed by v_i between v_i and v_j. To illustrate, an edge labeled as *customer* connects a customer AS to a provider AS, whereas an edge labeled as *provider* connects a provider AS to a customer AS. Note that the function Φ is not necessarily symmetric. For instance, a customer-to-provider relation between vertices v_i and v_j is encoded asymmetrically as $\Phi((v_i, v_j)) = customer$ and $\Phi((v_j, v_i)) = provider$ whereas a peer-to-peer relation between v_i and v_j is encoded symmetrically as $\Phi((v_i, v_j)) = peer$ and $\Phi((v_j, v_i)) = peer$.

Figure 1 shows our graph representation of a simplified AS-level Intenet topology map. In the figure vertices correspond to ASes and the relations between ASes are represented only by directed edges. Any relation between two ASes is encoded by two directed edges. Furthermore, each directed edge has an associated policy label which is assigned by the predecessor (first) vertex of the edge. The policy labels simply reflect the role of the predecessor vertex in the relation. For example, the customer-to-provider relation between $AS31$ and $AS21$ is represented by a *customer* (c) edge from $AS31$ to $AS21$ and a *provider* (p) edge from $AS21$ to $AS31$ in Figure 1. The label of edge $(AS31, AS21)$, *customer* (c), denotes the role of $AS31$ in the relation and the label of edge $(AS21, AS31)$, *provider* (p), denotes the role of $AS21$ in the relation. Similarly, the peer-to-peer relation between $AS11$ and $AS12$ is represented by a *peer* (r) edge from $AS11$ to $AS12$ and another *peer* (r) edge from $AS12$ to $AS11$ in Figure 1.

IV. Policy-Preferred AS Paths Problem

A simple path, P, of hop length k from a source vertex v_s to a destination vertex v_d in G is defined as a sequence of connected, alternating edges which starts at vertex v_s and ends at vertex v_d, i.e., $P = \{(v_s, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_d)\}$ where $P[1][1] = v_s$, $P[k][2] = v_d$, $v_i \neq v_j$ and $P[i][2] = P[i+1][1]$ for $i < k$. We introduce the double square brackets "[][]" operator on a path. The first index of the operator refers to an edge at a particular hop and the second index refers to the predecessor/successor (first/second) vertex of the edge. To illustrate, $P[5][2]$ refers to the successor (second) vertex of the edge located at hop 5 on path P. Note that the edge index of the brackets operator can inclusively take an integer between 1 and $|P|$ and the vertex index can only take either 1 or 2.

A policy-preferred path, P, in an AS-level Internet topology graph is a simple path which conforms to the following four assumptions that exist in practice. Although deviations from these assumptions are reported [9], [21], they capture the most common cases in practice [16].

Explicit Business Relations: Two ASes connected to each other form a business agreement which is typically consistent with their routing policies [17], [8], [5], [16].

Valley-free AS Path: A simple path is called *valley-free* if the path consists of zero or more *customer* edges followed by zero or one *peer* edge, and then zero or more *provider* edges. A valley-free path is congruent with the business agreements among ASes. That is, a customer AS gets reachability service from its provider ASes and peer ASes provide mutual reachability only to each other and their customers. In Figure 1 the simple path $P = \{(AS32, AS22), (AS22, AS11), (AS11, AS12), (AS12, AS24)\}$ is a valley-free path from $AS32$ to $AS24$. However, the path $P = \{(AS32, AS22), (AS22, AS33), (AS33, AS24)\}$ is not valley-free because it makes a valley via the edges $(AS22, AS33)$ and $(AS33, AS24)$. Technically, $AS33$ does not re-announce the routes that it learned from one of its providers, $AS33$, to another provider, $AS22$, because it avoids to relay the traffic between its providers.

First-Hop-Edge Label Preferred AS Path: If there are multiple valley-free paths between a source and a destination AS, the source AS prefers one or more of these paths over the others according to the policy labels (*customer*, *provider*, *peer*) of the first hop edges. Specifically, a path starting with a *provider* edge is preferred over a path starting with a *customer* edge because a provider AS usually charges its customer AS for the traffic exchanged. A path starting with a *peer* edge is preferred over a path starting with a *customer* edge because peer ASes typically engage in settlement-free business agreements. On the other hand, a customer AS pays to its provider AS for the traffic exchanged. A path starting with a *provider* edge is preferred over a path starting with a *peer* edge because while peer relations are usually settlement-free, a provider AS charges its customer AS for the traffic exchanged. Briefly, one or more valley-free paths are preferred among multiple valley-free paths according to the policy labels of their first hop edges. Moreover, the set of edge policy labels $\mathcal{L} = \{customer, provider, peer\}$ is linearly ordered according to their preferabilities.

Definition 4.1. *"$\succ$" (succeeds or higher than). The set of edge labels $\mathcal{L} = \{customer, provider, peer\}$ is a linearly ordered set under strict total order relation "$\succ$" (succeeds or higher than) where $provider \succ peer \succ customer$.*

To illustrate, in Figure 1 there are two valley-free paths from $AS12$ to $AS31$ namely, $P_1 = \{(AS12, AS25), (AS25, AS21), (AS21, AS31)\}$ and $P_2 = \{(AS12, AS11), (AS11, AS31)\}$. P_1 starts with a *provider* edge, $\Phi(P_1[1]) = p$, and P_2 starts with a *peer* edge, $\Phi(P_2[1]) = r$. $AS12$ prefers P_1 over P_2 because $\Phi(P_1[1]) \succ \Phi(P_2[1])$. The incentive behind $AS12$ preferring P_1 over P_2 is that the customer AS, $AS25$, on P_1 pays for the traffic exchanged, whereas it has a settlement-free agreement with the peer AS, $AS11$, on P_2. Note that although P_1 is longer than P_2 in terms of hop distances, $AS12$ prefers P_1 over P_2 for economical reasons.

Shortest Hop Distance Preferred AS Path: If there are multiple valley-free paths starting with the same edge policy label from a source AS to a destination AS, the source AS prefers the path with the shortest length in terms of the hop distance. In Figure 1 there are two valley-free paths from $AS31$ to $AS12$ namely, $P_1 = \{(AS31, AS21), (AS21, AS25), (AS25, AS12)\}$ and $P_2 = \{(AS31, AS11), (AS11, AS12)\}$ where $\Phi(P_1[1]) = \Phi(P_2[1]) = c$. Since both paths start with the same edge policy label, c, $AS31$ prefers P_2 over P_1 because both providers $AS21 \in P_1$ and $AS11 \in P_2$ charge $AS31$ for the traffic exchanged, yet P_2 is shorter than P_1 in terms of hop distance, i.e., $|P_1| < |P_2|$. *Note that since the details of business agreements between ASes are confidential, it is common to assume that the cost of sending traffic through provider AS $AS21$ and provider AS $AS11$ are the same.* Another important observation is that the preferred paths between ASes in an AS-level Internet topology graph is not necessarily symmetric as $\{(AS31, AS11), (AS11, AS12)\}$ and $\{(AS12, AS25), (AS25, AS21), (AS21, AS31)\}$ are two asymmetric paths preferred by $AS31$ and $AS12$, respectively.

In summary, the "policy-preferred AS path" concept is an extension to the "valley-free AS path" concept [17], [8] with "first-hop-edge label preferred AS path" along with "shortest hop distance preferred AS path" specializations.

V. Solution to the Policy-Preferred AS Paths Problem

In the following, we first present an overview of the Policy-Preferred AS Path Enumeration Algorithm. Then, we provide the pseudocodes of the algorithm as well as the procedure to enforce the valley-free path property.

Single-destination policy-preferred AS path enumeration algorithm *finds all policy consistent paths to a single destination vertex from every source vertex in an AS-level Internet topology graph*. During the course of execution the algorithm distinguishes the vertices into three disjoint colors. Black vertices are fully explored vertices that already have one or more policy-preferred paths toward the destination vertex. Gray vertices are discovered but not fully explored vertices. White vertices are undiscovered vertices, yet. Initially the destination vertex is gray and the remaining vertices are white. Additionally, the algorithm tracks two variables for each vertex: `join label` and `join hop-distance`. The `join label` of a vertex reflects the policy label of the tentative edge(s) connecting the vertex to the subgraph of the established, policy-preferred paths. The `join hop-distance` of a vertex reflects the vertex's tentative hop distance to the destination.

The algorithm starts from a designated destination vertex and incrementally builds paths in backwards from source vertices toward the destination vertex. At each iteration, the most preferable gray vertex (highest join edge label and shortest join hop distance) is joined to the subgraph of the established, policy-preferred paths toward the destination vertex via one or more edges. When a gray vertex is selected, its incoming edges that preserve the valley-free property and induce on a gray or white vertex are relaxed. That is, the join edge labels and join hop distances of the white and gray neighbors of the selected vertex are updated if a higher edge policy label or a shorter distance is revealed via the selected vertex. At the end, the selected vertex is colored black.

The core of the algorithm lies in the step which selects the most preferable vertex among all gray vertices. At each iteration, the decision is made according to both join edge labels and join hop-distances of the gray vertices. The set of join labels, $\mathcal{L} = \{customer, provider, peer\}$, is an ordered set under strict total order relation "$\succ$" (succeeds or higher than) where $provider \succ peer \succ customer$ (Definition 4.1). The set of join hop-distances, $\mathbb{Z}_{>0}$, is also a linearly ordered set under the usual "$<$" (less than) relation. As a result, a new relation "$\succ^*$" (more preferable) which is defined on the Cartesian product of $\mathcal{L}$ and $\mathbb{Z}_{\geq 0}$ is a strict total order relation.

Definition 5.1. *"$\succ^*$" (more preferable). A vertex with join label l and join hop-distance h is more preferable compared to another vertex with join label l' and join hop-distance h' if and only if l succeeds l' or l is equal to l' and h is less than h'. That is,* $(l,h) \succ^* (l',h') \iff (l \succ l') \vee (l = l' \wedge h < h')$ *where* $l, l' \in \mathcal{L}$ *and* $h, h' \in \mathbb{Z}_{\geq 0}$.

Our implementation of the single-destination, policy-preferred AS path enumeration algorithm involves the following function definitions and data structures:

`in_neighbor(v_i)`: Provides an unordered list of vertices that have an incoming edge to vertex v_i.

$\Phi((v_i, v_j))$: Provides the v_i appointed edge policy label for the edge (v_i, v_j).

`color(v_i)`: Provides the track of progress in the algorithm for vertex v_i. Specifically, *white* means undiscovered vertex, *gray* means discovered but not fully explored vertex and *black* means fully explored vertex.

`join_edge_label(v_i)` and `join_hop_dist(v_i)`: Respectively provide the preferable join edge label and join hop-distance of vertex v_i during the course of the algorithm. Both join edge label and join hop-distance are subject to change after edge relaxations for white and gray vertices whereas, they are final for black vertices.

`Q`$(\succ^*)$: A priority queue holding the gray vertices according to the "$\succ^*$" (more preferable) relation given in Definition 5.1. The priority queue, Q, supports `push` and `pop` operations to insert a new element and retrieve the top element, respectively.

The policy-preferred AS path enumeration algorithm expects an AS level Internet topology graph, $G = (V, E, \Phi)$, and a destination vertex, $v_d \in V$, as input. The output of the algorithm is a list of successor vertices, π, toward the destination vertex v_d for each vertex in the topology graph.

Lines 2-6 in Algorithm 1 initializes the vertices by setting their colors to white, join labels to *nil* and join hop-distances to *infinity*. Lines 9 and 10 set up the destination vertex, v_d, by setting its color to gray and its join hop-distance to zero, respectively. Line 12 declares a priority queue of gray vertices ordered by the "$\succ^*$" (more preferable) relation and line 13 inserts the destination vertex v_d into the priority queue.

The algorithm processes the vertices of the topology graph until the loop at line 15 ends. At each iteration the most preferable, gray vertex in the priority queue is selected (line 16). If there is a tie, it is broken arbitrarily. Then, the incoming edges of the selected vertex are relaxed in a for-loop (line 17) with the condition that joining the edge to the selected vertex does not violate the valley-free property (line 18).

While relaxing the incoming edges of the selected vertex if the neighboring vertex is unexplored, i.e. it is white, then its join label is updated with the edge label of the relaxed edge; its join hop-distance toward the destination is updated by adding one to the join hop-distance of the selected vertex; its successor list is updated by the selected vertex; its color is set to gray; and finally it is added to the priority queue as shown at lines 20-24.

If the neighboring vertex is not fully explored yet, i.e. it is gray, and the relaxed edge's policy label succeeds the neighboring vertex's current join label, then a more preferable edge is found to connect the neighboring vertex to the subgraph of the established, policy-preferred paths toward the destination vertex. Hence, the join label of the neighboring vertex is replaced by the label of the relaxed edge; its join hop-distance is updated according to the join hop-distance of the selected vertex; its successor list is cleared and the selected vertex is added to the successor list as shown at lines 27-30.

Algorithm 1 Policy-Preferred AS Path Enumeration

Input: $G = (V, E, \Phi)$ ▷ AS-level Internet topology graph
Input: v_d ▷ destination vertex, $v_d \in V$
Output: π ▷ list of successor vertices towards v_d for each vertex

1: ▷ Initialize all vertices
2: **for all** $v_i \in V$ **do**
3: $color(v_i) \leftarrow white$
4: $join\_edge\_label(v_i) \leftarrow nil$
5: $join\_hop\_dist(v_i) \leftarrow \infty$
6: **end for**
7:
8: ▷ Initialize the destination vertex v_d
9: $color(v_d) \leftarrow gray$
10: $join\_hop\_dist(v_d) \leftarrow 0$
11:
12: $Q(\succ^*)$ ▷ A priority queue with "more preferable" relation $\succ^*$
13: $Q.push(v_d)$
14:
15: **while** Q is not empty **do**
16: $v_s \leftarrow Q.pop()$ ▷ selects the most preferable vertex in Q
17: **for all** $v_j \in in\_neighbor(v_s)$ **do**
18: **if** $is\_valley\_free(\Phi((v_j, v_s)), join\_edge\_label(v_s))$ **then**
19: **if** $color(v_j)$ is $white$ **then**
20: $join\_edge\_label(v_j) \leftarrow \Phi((v_j, v_s))$
21: $join\_hop\_dist(v_j) \leftarrow join\_hop\_dist(v_s) + 1$
22: $\pi(v_j).insert(v_s)$
23: $color(v_j) \leftarrow gray$
24: $Q.push(v_j)$
25: **else if** $color(v_j)$ is $gray$ **then**
26: **if** $\Phi((v_j, v_s)) \succ join\_edge\_label(v_j)$ **then**
27: $join\_edge\_label(v_j) \leftarrow \Phi((v_j, v_s))$
28: $join\_hop\_dist(v_j) \leftarrow join\_hop\_dist(v_s) + 1$
29: $\pi(v_j).clear()$
30: $\pi(v_j).insert(v_s)$
31: **else if** $\Phi((v_j, v_s)) = join\_edge\_label(v_j)$ **then**
32: **if** $join\_hop\_dist(v_j) > join\_hop\_dist(v_s) + 1$ **then**
33: $join\_hop\_dist(v_j) \leftarrow join\_hop\_dist(v_s) + 1$
34: $\pi(v_j).clear()$
35: $\pi(v_j).insert(v_s)$
36: **else if** $join\_hop\_dist(v_j) = join\_hop\_dist(v_s) + 1$ **then**
37: $\pi(v_j).insert(v_s)$
38: **end if**
39: **end if**
40: **end if**
41: **end if**
42: **end for**
43: $color(v_s) \leftarrow black$
44: **end while**
45:
46: **return** π

On the other hand, if the relaxed edge's policy label is equal to the neighboring vertex's current join label and joining the neighboring vertex through the relaxed edge has a shorter hop distance then, again a more preferable edge is found to connect the neighboring vertex to the subgraph of the established, policy-preferred paths toward the destination vertex. Hence, the neighboring vertex's join hop-distance is replaced according to the selected vertex; its successor list is cleared and the selected vertex is added to the successor list as shown at lines 33-35.

Lastly, if the relaxed edge's policy label is equal to the neighboring vertex's current join label and joining the neighboring vertex through the relaxed edge has the same join hop-distance, then an additional edge is found to connect the neighboring vertex to the subgraph of the established, policy-preferred paths toward the destination vertex. Hence, the selected vertex is simply added to the neighboring vertex's successor list as shown at line 37.

Finally, line 43 sets the color of the selected vertex to black to denote that it is fully explored and its policy consistent path(s) toward the destination vertex have been successfully computed. Note that if the destination vertex is not reachable from a source vertex via a valley-free path, its join label remains as *nil* and its join hop-distance remains as *infinity*.

Let $H = (V', E')$ be the output graph generated by Algorithm 1. $H = (V', E')$ is a subgraph of $G(V, E, \Phi)$ formed by the policy-preferred paths toward the destination

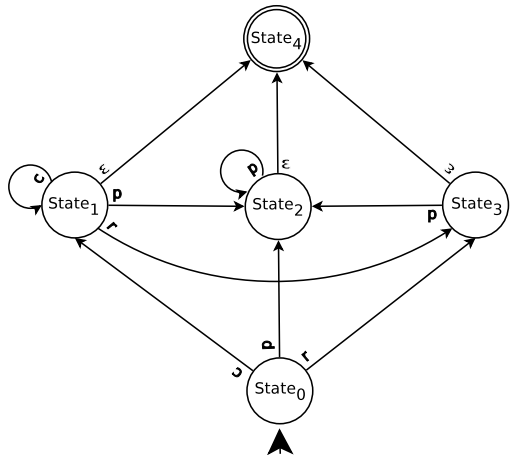

Fig. 2: NFA-ε recognizing valley-free AS paths

vertex such that $V' \subseteq V$ and $E' \subseteq E$.

Theorem 5.1. *$H = (V', E')$ is a directed, acyclic graph.*

Proof by Induction: Let $H_i = (V'_i, E'_i)$ be the subgraph of the established, policy-preferred paths after the i^{th} iteration of Algorithm 1. After the first iteration the subgraph of policy-preferred paths, H_1, consist of only the destination vertex, so the claim is true. Assume that the claim is true after $i-1$ iterations, i.e., $H_{i-1} = (V'_{i-1}, E'_{i-1})$ is a directed, acyclic graph. Let v_i be the next vertex to be joined to H_{i-1} via one or more edges. Since the paths are built in backwards, any edge, e_i, connecting v_i to H_{i-1} must have v_i as the predecessor vertex, i.e., $e_i = (v_i, u)$ such that $v_i \in V - V'_{i-1}$ and $u \in V'_{i-1}$. Put in other words, v_i is connected to H_{i-1} only via one or more of its outgoing edges. The minimum requirement for v_i to make a directed cycle in H_i is two edges such that the first edge is an outgoing edge from v_i and the second edge is an incoming edge to v_i from a vertex appearing on a path from v_i to the destination vertex. However, v_i is joined to H_{i-1} only via its outgoing edges therefore, joining v_i to H_{i-1} cannot form a directed cycle in H_i. ■

Corollary 5.1.1. *$H = (V', E')$ is a rooted, directed, acyclic graph (r-DAG) which is rooted by the destination vertex, $v_d \in V'$, because ideally v_d is the only vertex which does not have an outgoing edge in H.*

Algorithm 1 runs in time $O((|V| + |E|)log|V|)$ which is the same as Dijkstra's shortest paths algorithm with a priority queue implementation.

A. Valley-Free AS Path Detection Pseudocode

In this part we define the procedure used to enforce the valley-free path property at line 18 of Algorithm 1. By definition, a simple path is called *valley-free* if the path consists of zero or more *customer* edges followed by zero or one *peer* edge, and then zero or more *provider* edges.

We introduce a non-deterministic finite automaton with ε transitions, $(\Sigma, S, s_0, \delta, F)$, recognizing valley-free paths in an AS level graph, $G = (V, E, \Phi)$, in Figure 2. The automaton consists of an input alphabet $\Sigma = \{provider, peer, customer\}$, a set of states $S = \{State_0, State_1, State_2, State_3, State_4\}$, an initial state $s_0 = State_0$, a state transition function $\delta : S \times (\Sigma \cup \varepsilon) \rightarrow \mathcal{P}(S)$, and a set of final states $F = \{State_4\}$.

Our implementation of valley-free AS path detection procedure follows the NFA-ε given in Figure 2. Since Algorithm 1 incrementally explores alternative edges by relaxation and a subpath of a valley-free path is also valley-free, it is enough to check if the last edge label transition preserves the valley-free property before relaxing an edge.

Algorithm 2 Valley-Free AS Path Detection

```
Input: l_p ∈ L                                   ▷ Previous edge policy label
Input: l_n ∈ L                                   ▷ Next edge policy label
1: procedure is_valley_free(l_p, l_n)
2: return
3:     (l_p = customer ∧ (l_n = provider ∨ l_n = peer ∨ l_n = customer))
4:     ∨(l_p = peer ∧ l_n = provider)
5:     ∨(l_p = provider ∧ l_n = provider)
6:     ∨(l_n = nil)
7:
8: end procedure
```

Algorithm 2 provides a procedure which recognizes if joining an edge to the subgraph of the established, policy-preferred paths will violate the valley-free property or not. The algorithm expects a previous edge label and a next edge label as input. Lines 3-5 validates the transition from the previous edge label to the next edge label according to the NFA-ε given in Figure 2. Line 6 is introduced to validate the initial transitions from the destination vertex which has the void join label *nil* in Algorithm 1.

VI. Empirical Evaluations

In this part we present a comparison of policy-preferred paths and shortest paths on a real world dataset to provide the reader with more insight. Similar to many studies in the field, the Internet topology map used in our experiment is collected from CAIDA, in February, 2016 [3]. CAIDA provides highly accurate (around 99%) [16] Internet topology maps constructed from multiple sources including remote route collectors and traceroute outputs. The Internet topology map used in this study consists of 53,195 ASes connected to each other via 215,596 AS relations. We computed all-pairs, policy-preferred paths using the proposed algorithm (Algorithm 1) in this study. Due to memory overhead we computed all-pairs shortest paths using a textbook implementation of Dijkstra's shortest paths algorithm rather than Johnson's algorithm or FloydWarshall algorithm.

In the following we analyze the lengths (hop distances) of the paths appearing between more than 2.8 billions of AS pairs in our topology graph. The paths computed by the shortest paths algorithm are referred as **SP-paths** and the paths computed by the policy-preferred paths are referred as **PPP-paths**. The average path lengths for the PPP-paths and the SP-paths are 4.47 and 3.79, respectively. PPP-paths are longer than the SP-paths by 0.68 hops on the average. Since the majority of the paths in an AS level Internet graph are not very long, a difference of 0.68 between the averages of PPP-paths and SP-paths is significant.

Figure 3 shows the relative path length frequencies for both PPP-paths (dotted, blue) and SP-paths (dashed, red). The minimum and maximum path lengths in SP-paths are 1 and 12, respectively. Moreover, more than 99% of the path lengths are between 2 and 6. On the other hand, PPP algorithm generates longer paths since the paths have to adhere BGP policy consistency. The minimum and maximum path lengths in PPP-paths are 1 and 25, respectively. Besides, more than 99% of the path lengths are between 2 and 10.

Evidently, the SP algorithm generates shorter paths compared to the PPP algorithm by opting for "policy inconsistent" and "less preferable" paths. In order to gain a deeper insight we directly compared the path lengths between more than 2.8 billions of AS pairs.

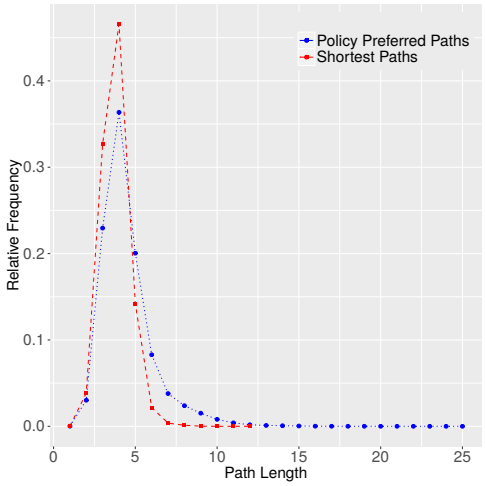

Fig. 3: Relative path length frequencies for PPP and SP

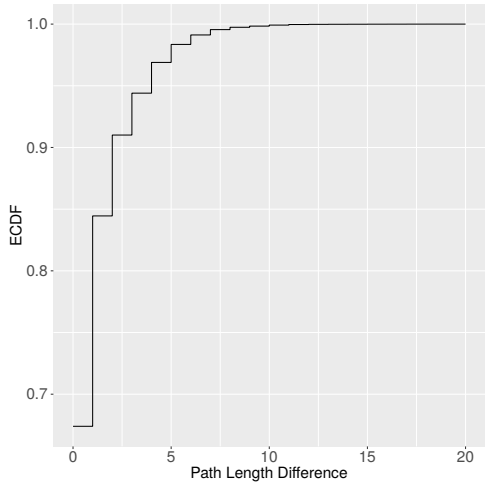

Fig. 4: Path length differences between PPP and SP

Figure 4 shows the path length differences calculated by subtracting the SP-path length from the corresponding PPP-path length for each AS pair. In the figure around 67% of the paths found by the SP and PPP algorithms have the same length, i.e., have difference zero. On the other hand, 33% of the paths differ by 1 to 20 hops, meaning that one third of the AS paths found by the SP algorithm violate policy consistency, for certain. Alternatively, *at least one third of the paths found by the SP algorithm are the paths that are not utilized by BGP, in reality*. Note that the PPP and SP algorithms may show discrepancy for the AS pair paths in the remaining two thirds as well. Specifically, in case there is a mixture of policy consistent (more preferable) and policy inconsistent (less preferable) shortest paths between a pair of ASes, the SP algorithm computes both types of the paths while the PPP algorithm computes only the policy consistent (more preferable) paths.

In summary, the experimental results reassures our case presented in the introduction section claiming that the analysis or adoption of AS level Internet topologies via a shortest paths algorithm can be greatly misleading.

VII. Conclusions

Using AS-level Internet topology maps to determine accurate AS-level paths is essential for detecting congestion points in the Internet, reducing the overall traffic in P2P networks, enhancing the quality of VoIP services, estimating network delays and optimizing service deployment in the Internet.

In this study we introduced a novel single-destination, policy-preferred path enumeration algorithm which discovers preferable, policy consistent paths from all ASes to a destination AS in an AS-level Internet topology graph. Our algorithm provides a holistic solution to the AS-level path enumeration problem by incorporating common practices and incentives in inter-AS routing. The proposed algorithm runs in time $O((|V|+|E|)log|V|)$ which is the same as Dijkstra's shortest paths algorithm with a priority queue implementation.

Our experimental results show that at least 33% of the AS paths found by a shortest paths algorithm are not utilized in reality because they violate policy consistency.

Finally, a C++ implementation of the presented algorithm is publicly available on our project website at http://nsrg.louisiana.edu/project/ntmaps/ .

References

[1] A. Akella, S. Seshan, and A. Shaikh. An empirical evaluation of wide-area internet bottlenecks. In *ACM IMC*, Miami, FL USA, Oct 2003.

[2] T. Bu and D. Towsley. On distinguishing between internet power law topology generators. In *INFOCOM, IEEE*, Jun 2002.

[3] CAIDA. Dataset. http://data.caida.org/datasets/as-relationships/serial-1/20160201.as-rel.txt.bz2.

[4] H. Chang, S. Jamin, and W. Willinger. Inferring as-level internet topology from router-level path traces. In *SPIE ITCom*, Denver, CO, USA, Aug 2001.

[5] X. Dimitropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, k. claffy, and G. Riley. AS Relationships: Inference and Validation. *ACM Computer Communication Review*, 37(1):29–40, Jan 2007.

[6] D. Dolev, S. Jamin, O. Mokryn, and Y. Shavitt. Internet resiliency to attacks and failures under {BGP} policy routing. *Computer Networks*, 50(16):3183 – 3196, 2006.

[7] A. Faggiani, E. Gregori, A. Improta, L. Lenzini, V. Luconi, and L. Sani. A study on traceroute potentiality in revealing the internet as-level topology. In *IFIP Networking*, Trondheim, Norway, Jun 2014.

[8] L. Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745, Dec 2001.

[9] V. Giotsas, M. Luckie, B. Huffaker, and K. Claffy. Inferring Complex AS Relationships. In *ACM IMC*, Nov 2014.

[10] S. Hasan, S. Gorinsky, C. Dovrolis, and R. Sitaraman. Trade-offs in optimizing the cache deployments of cdns. In *IEEE INFOCOM*, Toronto, Canada, Apr 2014.

[11] Y. He, G. Siganos, M. Faloutsos, and S. Krishnamurthy. Lord of the links: A framework for discovering missing links in the internet topology. *IEEE/ACM Trans. Netw.*, 17(2):391–404, Apr 2009.

[12] J. Juen, A. Johnson, A. Das, N. Borisov, and M. Caesar. Defending tor from network adversaries: A case study of network path prediction. *Proc. on Privacy Enhancing Technologies*, 2015(2):171–187, Jun 2015.

[13] A. Khan, T. Kwon, H. Kim, and Y. Choi. As-level topology collection through looking glass servers. In *ACM IMC*, Spain, Oct 2013.

[14] D. Lee, K. Jang, C. Lee, G. Iannaccone, and S. Moon. Scalable and systematic internet-wide path and delay estimation from existing measurements. *Computer Networks*, 55(3):838–855, Feb 2011.

[15] J. Li and K. Sollins. Exploiting autonomous system information in structured peer-to-peer networks. In *ICCCN*, Chicago, USA, Oct 2004.

[16] M. Luckie, B. Huffaker, K. Claffy, A. Dhamdhere, and V. Giotsas. AS Relationships, Customer Cones, and Validation. In *Internet Measurement Conference (IMC)*, pages 243–256, Oct 2013.

[17] Z. M. Mao, L. Qiu, J. Wang, and Y. Zhang. On as-level path inference. In *ACM SIGMETRICS*, Alberta, Canada, Jun 2005.

[18] J. Qiu and L. Gao. As path inference by exploiting known as paths. In *Global Telecommunications Conference, IEEE*, Nov 2006.

[19] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), Jan 2006.

[20] S. Ren, L. Guo, and X. Zhang. ASAP: an AS-Aware Peer-Relay Protocol for High Quality VoIP. page 70, 2006.

[21] M. Roughan, W. Willinger, O. Maennel, D. Perouli, and R. Bush. 10 lessons from 10 years of measuring and modeling the internet's autonomous systems. *IEEE JSAC*, 29(9):1810–1821, 2011.

[22] N. Tao, X. Chen, and X. Fu. As path inference: From complex network perspective. In *IFIP Networking Conference*, pages 1–9, May 2015.

[23] UCSD. Center for applied internet data analysis. www.caida.org.

[24] UO. Route views project. www.routeviews.org.