Collaborative Verification of Forward and Reverse Reachability in the Internet Data Plane

Hongkun Yang and Simon S. Lam
Department of Computer Science, The University of Texas at Austin
{yanghk, lam}@cs.utexas.edu

Abstract—To debug reachability problems, a network operator often asks operators of other networks for help by telephone or email. We present a new protocol, COVE, for automating the exchange of data plane reachability information between networks in a business relationship. A network deploys COVE in a host (its local verifier) which can construct both forward and reverse reachability trees in the Internet data plane for the network’s provider/customer cone. Each edge in a tree is annotated by a set of packets that can traverse the edge. COVE was designed with partial deployment in mind. Reachable networks that do not deploy COVE are leaf nodes in reachability trees. Partial trees are useful.

We constructed an Internet dataset of 2,649 ASes and performed experiments in which up to 170 workstations ran COVE as local verifiers to construct forward and reverse provider (also customer) trees for ASes. The results of these experiments demonstrate scalability of COVE to very large ASes in the Internet. We illustrate applications of COVE to solve the following network management problems: evaluating inbound load balancing policies, what-if analysis before adding a new provider, finding additional paths, configuring default routes as backup, black hole detection, and persistent forwarding loop detection.

I. INTRODUCTION

Networks connect to the Internet to reach other networks. Network operators need reachability information to perform a variety of network management tasks, such as, debugging reachability when packets vanish into black holes [13], finding redundant paths, what-if analysis prior to changing routing policy or adding a new provider, etc.

Two different views of reachability are of interest to researchers and operators, namely: “how do I see the world?” and “how does the world see me?” [7]. We refer to these two views as, respectively, forward reachability and reverse reachability. Reachability information can be obtained from two sources: (i) the control plane, namely, Border Gateway Protocol (BGP) messages exchanged between autonomous systems (ASes), import and export policies as well as the BGP decision process of each AS; (ii) the data plane, namely, forwarding tables in routers used to send data packets to other routers. Within a network, data plane reachability also depends on packet filters (e.g., ACLs and firewalls) that guard input and output ports of routers and switches [15], [28].

Each AS receives BGP messages containing routes to destination prefixes from its neighbor ASes. Wider views of forward reachability are available from BGP router collectors placed in hundreds of ASes in the Internet [26], [23]. Such control plane reachability, however, is not a good predictor of data plane reachability. For example, a destination may be reachable in the control plane but unreachable in the data plane because some network connectivity change has not been observed in the monitored portion of the control plane [13]. Also, a destination may be unreachable in the control plane but is in fact reachable in the data plane by default routes (not advertised in the control plane) [7]. For the same reasons, even if a destination is reachable in the data plane as well as the control plane, the two paths may be different.

Active probes (pings and traceroutes) are widely used to measure data plane reachability [27], [13], [7], [12], [14], [11] but they have limitations and biases. For example, a probe may be filtered along the way to its destination or its response is filtered on its way back; also, the destination may choose not to reply [25]. Thus the absence of a response is not a reliable indication of unreachability [12].

Currently, when network operators debug reachability problems, they often ask operators of other networks for help by telephone, email, or posting on mailing lists [20], [18]. We searched the NANOG Mailing List from April 2008 to July 2014 and found that 14.5% of emails in the list are about reachability problems. Network operators are generally willing to help each other debug reachability problems for the common good. However, operators of today’s very high speed networks use very slow means of communication to ask each other for reachability information.

Towards automating the exchange of reachability information between networks, we have designed and implemented a protocol, named COVE, for networks (namely ASes) in business relationships. COVE is designed to improve a network’s views of forward and reverse reachability in the Internet data plane even if the protocol is partially deployed by as few as two networks connected by a link.

A network deploys COVE by running the protocol in a host, called its local verifier. Local verifiers of two networks exchange (encrypted) messages with each other only if the networks have a business relationship. However, the local verifier of a network, say B, which has independent relationships with networks, A and C, can query the local verifier of A on behalf of C, and vice versa. Once there is a set of interconnected networks that deploy COVE, there is strong incentive for another network to join and for the set to accept a new member to increase every network’s view of reachability.

Input and output ports of links that connect different net-

1COVE is acronym for COllaborative VErification. For ease of exposition, we use “network” and “AS” interchangeably unless otherwise stated.
works are, by definition, public ports. Only reachability information from the public input ports of a network to its public output ports is provided to another network. Reachability information to ports inside a network is kept private.

COVE is the first protocol designed and implemented to enable discovery of forward and reverse inter-network reachability in the data plane. The reachability information obtained using COVE is ground truth assuming that local verifiers of networks in a business relationship provide ground truth information to each other. A summary of our contributions in this paper follows:

**Internet dataset for experiments.** To show that COVE is efficient and scalable for deployment by ASes in the Internet, we need a large and realistic Internet dataset. We began with data from different open sources, collected during October 2013 [3], [1], [4], [26], [23], [21]. There were 41,244 ASes in the Internet topology and 582,947 prefixes. We used C-BGP [2] to compute routing tables of all ASes. Many of the ASes have numerous missing links [19]. However, we found a subgraph in the topology, consisting of 2,649 “monitored” ASes (each of which has all of its provider, peer, and customer links in the Internet topology) and 57,429 links. All 15 tier-1 providers are included in the subgraph.

**COVE design and implementation.** COVE is an application-layer protocol designed for the local verifier of a network, say net\(_1\), to construct forward and reverse provider trees and, if net\(_1\) has at least one customer, forward and reverse customer trees. These trees are rooted at net\(_1\). Each tree node is a network. Reachable networks that do not run COVE are leaf nodes in trees. Each edge in a reachability tree, say from net\(_x\) to net\(_y\), is annotated by a set of packets that can travel in the link from net\(_x\) to net\(_y\). We use binary decision diagram (BDD) as the data structure for representing packet sets. It encodes longest prefix match semantics and has been shown to be both time and space efficient [28].

**COVE performance and scalability.** We constructed forward and reverse provider trees for the provider cones of 21 monitored tier-4 ASes. We also constructed forward and reverse customer trees for the customer cones of 50 ASes randomly selected from the monitored tier-2 and tier-3 ASes. We measured communication overheads of COVE as well as protocol run time for tree construction. The results of these experiments demonstrate scalability of COVE to very large ASes in the Internet. For example, the largest customer cone size among the 50 selected ASes is 5,863.

**Network management applications.** The operator of an AS can use all four reachability trees of the AS to perform a variety of network management tasks. The reverse trees, in particular, are especially useful since there is currently no other method to get them. We performed experiments to demonstrate several useful network management applications, namely: evaluation of inbound load balancing polices, what-if analysis prior to adding a new provider link, finding additional forward and reverse paths, configuring default routes as backup, black hole detection, and persistent forwarding loop detection.

The balance of this paper is organized as follows. In Section II, we describe how the Internet dataset for our experiments was constructed. In Section III, we present the basics of COVE. In Section IV, we present protocols for constructing forward and reverse trees for a provider/customer cone of an AS. In Section V, we describe our methodology for performing experiments and present statistics of forward and reverse trees for 50 customer cones and 21 provider cones. In Section VI, we present experimental results that demonstrate how network operators can use COVE to perform very useful network management tasks which cannot be done effectively using existing methods and tools. We discuss related work in Section VII and conclude in Section VIII.

**II. Internet Dataset**

We constructed a large set of ASes with BGP routing tables that we used to evaluate the performance and scalability of COVE. The routing tables were obtained using control plane data available from many open sources. As such, they are not ground truth of reachability in the Internet data plane. However, since every AS in our dataset has all of its peer, provider, and customer links, we can obtain routing tables that are representative of the scale and characteristics of routing tables in the Internet.

**A. Data collection and computation of routing tables**

We first downloaded the Internet topology from CAIDA created with data collected during October 2013 [1]. Using new methods for inferring AS relationships, the customer-provider and peer-peer relationships in the topology are, respectively, 99.6% and 98.7% accurate when compared to validation data [17].

**BGP policies.** Each entry (i.e., route) in the BGP routing table of an AS has several fields including a destination prefix, an AS path to the prefix, and a local preference value assigned by the AS. A prefix may have multiple table entries corresponding to all valid routes with the best route marked. The BGP decision process of an AS is a sequence of steps to select the best route from valid routes learned from advertisements of neighbor ASes. A recent survey of interdomain routing policies [10] shows that the majority of surveyed networks follow the Gao-Rexford (GR) model [8]. For route export, the GR model specifies that (i) a customer route may be advertised to all neighbor ASes, and (ii) a peer or provider route may be advertised only to customers. For route selection, the GR model specifies that routes are assigned local preference values such that customer routes are preferred over peer routes which are preferred over provider routes.

In the BGP decision process of an AS, a route with the highest local preference is selected. Among routes with the highest local preference value, the route with the shortest AS path length is selected. These are the most important criteria.

AS path prepending is a widely used technique to increase the AS path length in a prefix advertisement. This technique impacts the BGP decisions of ASes that receive the advertisement and change the reverse reachability tree of the prefix. Therefore, we can greatly improve the accuracy of AS routing tables in our Internet dataset by making use of path prepending information of BGP routes in route selection.
We downloaded BGP routes obtained by the route collectors\(^2\) of RouteView [26], RIPE [23], Packet Clearing House [21], and Internet2 [4] during October 2013. The collectors peered with 357 different ASes. We found that over 50% of prefixes have AS path prepending indicating that it plays a very important role in route selection.

We also downloaded prefix origin data from the UCLA Internet topology archive [3] and mapped each prefix to an AS that originates the prefix (for prefixes with length $<30$). Our dataset has 41,244 ASes from the Internet topology [1] and 582,947 prefixes from the prefix origin data [3].

**Computation of routing tables.** Computing BGP routing tables of all ASes in the Internet topology is time consuming. An $O(V^2)$ algorithm was proposed [9] where $V$ is the number of ASes under the assumption that prefixes originating from the same AS have the same path from each source, i.e., shortest paths are computed between ASes rather than between prefixes and ASes. However, due to the widespread use of AS path prepending, two prefixes originating from the same AS may have different paths from the same source. Computing routing tables of all ASes by considering prefixes one by one has a time complexity of $O(nV^2)$, where $N$ is the number of prefixes. In our dataset, $N = 582,947$ and $V = 41,244$.

We found an approach to partition the set of all prefixes into equivalence classes, namely: Two prefixes belong to the same equivalence class if and only if they originate from the same AS and in every AS, they have identical import and export policies. Thus prefixes in the same equivalence class have the same route in every AS. Therefore, we only need to compute BGP routes to each equivalence class of prefixes, from which we get BGP routing tables of ASes. Using this approach, the time complexity is reduced to $O(nV^2)$, where $n$ is the number of equivalence classes. We found a total of $n = 107,663$ equivalence classes in our dataset.

We applied C-BGP, a centralized tool that solves BGP decisions for an interconnected set of ASes to compute BGP routes exchanged between ASes and the BGP routing tables of all ASes. We ran C-BGP using 3-way parallelization to compute BGP routing tables for all ASes in the Internet topology. The computation time was less than 24 hours.

**B. Monitored ASes and statistics**

It is well-known that the Internet topology that can be discovered from data obtained by route collectors is incomplete due to missing links [19]. Most of the missing links are peer links due to the widespread use of the Gao-Rexford export policy. For our purposes, however, we only need a large subset of ASes that have all of their links in the discovered topology.

**Definition 1 (Monitored AS).** An AS is monitored if and only if one of the following conditions holds:

1. the AS peers with a BGP route collector;
2. the AS is a provider of a monitored AS.

Our definition of a monitored AS is the same as the definition of a “covered” AS in prior work [19]. A monitored AS has all its customer, peer, and provider links in the discovered Internet topology with probability close to 1 [7], [19].

We found a subgraph of 2,649 monitored ASes and 57,429 links in the discovered topology of our dataset. The monitored ASes include all tier-1 networks. The subgraph is connected and is much larger than what we need to evaluate the computation and communication overheads, as well as scalability of COVE.

We use the list of tier-1 networks from CAIDA [1]. For $k > 1$, a tier-$k$ network is a neighbor of a tier $k-1$ network. That is, the shortest path from a tier-$k$ network to a tier-$1$ network is $k$ AS hops. Table I shows the number of ASes of each tier in the monitored set. Most of the monitored ASes are in tier 2 and tier 3.

<table>
<thead>
<tr>
<th>Tiers</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>1094</td>
</tr>
<tr>
<td>3</td>
<td>879</td>
</tr>
<tr>
<td>4</td>
<td>23</td>
</tr>
</tbody>
</table>

**TABLE I: Tier distribution of monitored ASes.**

Figure 1 shows the link distribution of monitored ASes. The maximum number of providers for an AS is less than 70. A few networks in tier 2 and tier 3 have over 1000 peers each. A few tier-2 networks have over 1000 customers each. Nearly half of tier-1 networks have over 1000 customers each.

Monitored ASes have more links than unmonitored ASes in the dataset. On average, a monitored tier-2 AS has 4 provider links, 55 peer links, and 20 customer links; an unmonitored tier-2 AS has 2.4 provider links, 0.4 peer link, and 1.1 customer links. There are two reasons: (i) unmonitored ASes miss most of their peer links, and (ii) BGP route collectors usually peer with large ISPs. By demonstrating that COVE is scalable to monitored ASes, we show that it is scalable to practically all ASes in the Internet.

**III. COVE BASICS**

We present the basics of COVE in this section. More details of the COVE protocol are presented in Section IV.

A network deploys COVE by running the COVE protocol in a host, called its local verifier. Initially, the local verifiers
of two networks in a business relationship establish a persistent TCP connection and authenticate each other (e.g., using SSL). Afterwards, they share a symmetric key for encrypting query and reply messages between them. Two local verifiers exchange messages with each other only if their networks have a business relationship.

Consider two networks, net_a and net_b, in a business relationship and directly connected by at least one link. Each link is bidirectional: An output port of net_a transmits packets in the link to an input port of net_b; also, an output port of net_b transmits packets in the link to an input port of net_a. Input and output ports of links that connect different networks are, by definition, public ports. Only reachability information from the public input ports of a network to its public output ports is provided to another network.\footnote{A network in our model can be used to represent interconnected ASes that belong to the same owner. For such a network, reachability to ports of internal links connecting sibling ASes is private.}

A. Reachability sets in the data plane

Before constructing reachability trees, the local verifier of each network precomputes reachability sets in the data plane from each of its public input ports to each of its public output ports. The forwarding tables in routers and switches of a network, as well as the access control lists (ACLs) that guard input and output ports are packet filters, which allow some packets through a port while dropping others. The set of packets that can travel from one port to another port along a path can be obtained by intersecting the packet filters in the path. When there are multiple paths in a network from one port to another port, the reachability set between the ports is the union of the reachability sets along the paths.

There exist tools that use centralized algorithms to compute port-to-port reachability sets, namely: Header Space \cite{15} and AP Verifier \cite{28}. For use in COVE, each reachability set from a public input port to a public output port computed by one of these algorithms must also be filtered to remove packets that do not satisfy the network’s BGP export policy for the public input port.

A set of packets can be represented by a predicate whose variables are bits in the packet header. (Packets with identical headers are considered to be the same by a packet filter.) Thus a predicate \( P \) specifies the set of packets for which \( P \) evaluates to true. It has been shown that binary decision diagram (BDD) is a very efficient data structure for representing predicates that specify packet sets \cite{28}. We use BDDs in COVE to represent reachability sets for two more reasons. First, BDD can be used to encode longest prefix match semantics of IP forwarding \cite{28}. Second, intersection and union of packet sets correspond to conjunction and disjunction of predicates. These logical operations can be performed on BDDs efficiently using graph-based algorithms \cite{6}.

B. Customer and provider trees

Figure 2 illustrates the customer and provider cones of an AS (named Initiator). Note that each AS is included in its customer and provider cones. Also, the provider cone does not include tier-1 ASes.

Consider an AS, net_a. We use \( LV_a \) to denote its local verifier. \( LV_a \) constructs both forward and reverse reachability trees. A forward tree is rooted at net_a as the source. A reverse tree is rooted at net_a as the destination. Each tree node stores the identifier of an AS. Multiple nodes in a tree can store the same AS. For example, when there are multiple forward paths from net_a to net_g, there are multiple net_g nodes in the forward tree. Edges in a tree are directed. Each edge in a tree, say from net_x to net_y, is annotated by a set of packets that can travel in the link from net_x to net_y. \( LV_a \) constructs two forward trees (also two reverse trees) for the customer and provider cones of net_a.

Suppose all ASes, except tier-1 ASes, deploy COVE. The leaf nodes in customer trees are stub ASes. The leaf nodes in provider trees include: (i) tier-1 providers; (ii) peer ASes of providers in the provider cone; (iii) any end provider, net_d, such that in a forward tree, there is a path from the root AS to net_d only for packets with destinations in net_d’s customer cone and, in a reverse tree, there is a path from net_d to the root AS only for packets from sources in net_d’s customer cone.

If COVE is partially deployed, reachable ASes that do not deploy COVE are also leaf nodes in reachability trees.

C. Forward and reverse path construction examples

To illustrate tree construction, we describe two path construction examples for the network in Figure 3. \( LV_a \) initiates tree construction by sending queries to local verifiers of its providers, net_b and net_c.

We first informally describe how a particular path in the forward tree is constructed (message details in Section IV). \( LV_b \) receives the query containing \( P_1 = \text{true} \) (set of all packets) from \( LV_a \) and finds that it has output ports to net_c and net_f. Consider the output port to net_c. \( LV_b \) retrieves the precomputed reachability set, \( F_1 \), between the input port coming from net_a and the output port going to net_c. \( LV_b \) computes \( P_2 = P_1 \land F_1 \) and includes \( P_2 \) and its path in a reply to \( LV_a \). Then \( LV_b \) sends a query (on behalf of \( LV_a \)) containing \( P_2 \) to \( LV_c \).

Upon receiving the query, \( LV_c \) retrieves the precomputed reachability set, \( F_2 \), between the input port coming from net_b and the output port going to net_g. \( LV_c \) computes \( P_3 = P_2 \land F_2 \) and includes \( P_3 \) and its path in a reply to \( LV_a \). The reply from \( LV_c \) is sent back to \( LV_b \) which forwards the reply to \( LV_a \). In this example, \( LV_c \) does not send a query to \( LV_g \) because \( LV_g \) is a tier-1 network.\footnote{4 or because it does not deploy COVE}
When \( LV_x \) receives the reply containing \( P_2 \), it stores \( P_2 \) on the edge connecting \( net_b \) to \( net_c \) in \( P_2 \)’s path in the forward tree. When it receives the reply containing \( P_3 \), it stores \( P_3 \) on the edge connecting \( net_c \) to \( net_g \) in \( P_3 \)’s path in the forward tree. \( P_3 \) is the set of packets that can travel from the \( a \)-to-\( b \) link to the \( c \)-to-\( g \) link in the forward path.

We next informally describe how a particular path in the reverse tree is constructed (message details Section IV). \( LV_b \) receives the query containing \( Q_1 = true \) (set of all packets) from \( LV_a \) and finds that it has input ports from \( net_e \) and \( net_f \). Consider the input port from \( net_f \). \( LV_b \) retrieves the precomputed reachability set, \( R_1 \), between the input port coming from \( net_f \) and the output port going to \( net_a \). \( LV_b \) computes \( Q_2 = Q_1 \land R_1 \) and includes \( Q_2 \) and its path in a reply to \( LV_a \). Then \( LV_b \) sends a query (on behalf of \( LV_a \)) containing \( Q_2 \) to \( LV_f \).

Upon receiving the query, \( LV_f \) retrieves the precomputed reachability set, \( R_2 \), between the input port coming from \( net_g \) and the output port going to \( net_b \). \( LV_f \) computes \( Q_3 = Q_2 \land R_2 \) and includes \( Q_3 \) and its path in a reply to \( LV_a \). The reply from \( LV_f \) is sent back to \( LV_b \) which forwards the reply to \( LV_a \). In this example, \( LV_b \) does not send a query to \( LV_g \) because \( LV_g \) is a tier-1 network.

When \( LV_f \) receives the reply containing \( Q_2 \), it stores \( Q_2 \) on the edge connecting \( net_f \) to \( net_b \) in \( Q_2 \)’s path in the reverse tree. When it receives the reply containing \( Q_3 \), it stores \( Q_3 \) on the edge connecting \( net_g \) to \( net_f \) in \( Q_3 \)’s path in the reverse tree. \( Q_3 \) is the set of packets that can travel from the \( g \)-to-\( f \) link to the \( b \)-to-\( a \) link in the reverse path.

**Observation.** A reply message destined to the initiator’s local verifier is generated after a query is processed. Different queries and replies generated by different local verifiers are sent as separate messages (no merging). This approach results in more messages, but smaller messages. The advantage is that the initiator’s local verifier can construct a partial tree even if some reply messages are missing. Such a partial tree is still useful for checking reachability.

**IV. COVE Protocol**

**Tree notation.** Each node of a tree stores a tuple \((net, flag, tag)\), where \( net \) is a network identifier; \( flag \) has 3 possible values: \( continue \), \( finish \), and \( loop \); if \( flag \) is assigned value \( finish \), \( tag \) has 5 possible values: \( tier-1 \), \( peer \), \( non-COVE \), \( stub \), and \( end provider \).

In a reachability tree, an edge between a node storing \( net_x \) and a node storing \( net_y \) is annotated by a 5-tuple \( (net_x, port_x, net_y, port_y, P) \) where \( port_x \) is a public port of \( net_x \), \( port_y \) is a public port of \( net_y \), and \( P \) is a non-empty set of packets. In a forward tree, \( port_x \) sends to \( port_y \) and \( P \) is at the exit of \( port_x \) which is an output port. In a reverse tree, \( port_y \) sends to \( port_x \) and \( P \) is at the entrance of \( port_x \) which is an input port.

For a tree node, \( n \), elements in its tuple are denoted by \( n.net, n.flag, \) and \( n.tag \). For a tree edge, \( e \), elements in its tuple are denoted by \( e.net_x, e.port_x \), and \( e.P \), etc.

**Path notation.** A path, denoted by \( path(1,k), k \geq 1 \) is specified by

\[
net_1, port_1; net_2, port_2; \ldots; net_k, port_k; net_{k+1}
\]

where, for \( i = 1, 2, \ldots, k + 1 \), \( net_i \) is a network identifier, \( port_i \) is a public port of \( net_i \); for \( i = 1, 2, \ldots, k, net_i \) and \( net_{i+1} \) are neighbors. In a forward tree, each port is an output port and \( path(1, k) \) denotes a path from \( net_1 \) to \( net_{k+1} \). In a reverse tree, each port is an input port and \( path(1, k) \) denotes a path from \( net_{k+1} \) to \( net_1 \).

**Protocol messages.** COVE has two types of messages for tree construction: (i) a query message and (ii) a reply message in response to a query.

The protocol message header has one bit to indicate direction (forward or reverse) and a type field that indicates the specific protocol message and message content. In COVE, each packet set is represented by a BDD. The BDD of a packet set is converted into a byte sequence by the source local verifier for inclusion in a message; it is later converted back to a BDD by the destination local verifier.

**Neighbor networks.** For a provider tree, a network’s neighbor networks are its direct providers. For a customer tree, a network’s neighbor networks are its direct customers.

**A. Forward tree construction**

**Tree initialization.** The local verifier of \( net_1 \) creates a new forward tree with its root node storing \( net_1 \). For each neighbor network, \( LV_i \) creates a node storing its identifier and an edge from node \( net_1 \) to the node. For each node in the initial tree, \( n.flag = continue \); for each edge \( e \) in the initial tree, \( e.P = true \).

For every public output port connecting \( net_1 \) to a neighbor network \( net_2 \), \( LV_1 \) sends to \( LV_2 \) a query containing \( path(1,1); P_1 \), where \( P_1 = true \) and \( path(1,1) = net_1; port_1; net_2 \).

**Query processing.** The local verifier of network \( net_k \) has received a query containing:

\[
path(1, k-1); P_{k-1}
\]

where \( k \geq 2 \), \( path(1, k-1) \) denotes a path from \( port_1 \) in \( net_1 \) to \( net_{k-1} \), \( P_{k-1} \) is the set of packets that can travel from the exit of \( port_1 \) to the exit of \( port_{k-1} \) which connects to a public input port, \( port_{in} \), of \( net_k \).

To process the query message, \( LV_k \) finds all public output ports of \( net_k \) connected to neighbor networks and, for a provider tree, peer networks also. For each such public output port, \( port_k \), \( LV_k \) retrieves the precomputed reachability set, \( F \), from \( port_{in} \) to \( port_k \). \( LV_k \) then computes \( P_k = P_{k-1} \land F \), and, if \( P_k \neq false \), creates a 5-tuple. The first two elements of the 5-tuple are the identifiers of \( port_k \) and \( net_{k+1} \). The

**Fig. 3: Network example.**
third element is $P_k$. The fourth element is a flag value. The fifth element is a tag value.

The flag value is loop if a loop is detected in path$(1,k)$. The flag value is finish if net$_{k+1}$ is a tier-1, a peer, or a network that does not deploy COVE. Otherwise, the flag value is continue and LV$_k$ sends a query containing path$(1,k); P_k$, on behalf of net$_{k+1}$, through port$_k$ to net$_{k+1}$.

In a 5-tuple, if the flag value is finish, the tag value may be tier-1, peer, or non-COVE providing information about net$_{k+1}$.

Each 5-tuple created is stored in a set output set. After all public output ports of net$_k$ connecting to neighbor networks have been processed, LV$_k$ sends a reply message to net$_1$ containing

$$\text{path}(1,k-1); \text{output set}$$

The reply message is relayed by local verifiers along the reverse path of path$(1,k-1)$.

A reply message from net$_k$ with an empty output set indicates that net$_k$ is a leaf in the tree.

**Reply processing.** LV$_1$ which initiated tree construction receives a reply message containing path$(1,k-1); \text{output set}$. LV$_1$ already has a tree RT. It first checks if there exists $m$, $m > 1$ such that path$(m,k-1)$, within path$(1,k-1)$, does not exist in RT. If so, it inserts new nodes and new edges of the missing part to RT. In each newly inserted node, flag is initialized to continue; in each newly inserted edge, $P$ is initialized to null.

Then LV$_1$ finds the tree node $n$ that stores net$_k$ as the endpoint of path$(1,k-1)$. If output set is empty, then $n$.flag is assigned value finish and $n$.tag is assigned value stub (for a customer tree) or value end provider (for a provider tree). Otherwise, LV$_1$ processes the 5-tuples in output set, one by one, and updates node and edge information in the tree.

**B. Reverse tree construction**

The local verifier of net$_1$ creates a new reverse tree with its root node storing net$_1$. For each neighbor network, LV$_1$ creates a node storing its identifier and an edge from the node to the root node. For every neighbor network net$_2$ connected to an input port of net$_1$, LV$_1$ sends a query including path$(1,1); P_1$ to LV$_2$, where $P_1 = \text{true}$.

When LV$_k$ receives a query containing path$(1,k-1); P_{k-1}$ where $k \geq 2$ and $P_{k-1}$ is the set of packets that can travel from the entrance of port$_{k-1}$ to the entrance of port$_1$. Let port$_\text{out}$ be the public output port of net$_k$ connected to port$_{k-1}$. To process the query message, LV$_k$ finds all public input ports of net$_k$ connected to neighbor networks and, for a provider tree, peer networks also. For each such public input port, port$_{i}$, LV$_k$ retrieves the reachability set, $R$, from port$_k$ to port$_\text{out}$.

Additional details of reverse tree construction are omitted herein due to space limitation. Both initialization as well as query and reply processing for a reverse tree are similar to those for a forward tree presented above.

**C. Extended provider trees**

Consider a local verifier that has constructed forward and reverse provider trees for its network net$_1$. Nodes of each tree are peer or provider networks. If a network, net$_k$, is stored in a node and it deploys COVE, LV$_1$ can send a special query message, relayed by local verifiers along the path to it, to request for net$_k$’s customer trees as well as its peer networks.

When LV$_1$ receives a customer tree in a reply from net$_k$, it can extend net$_1$’s provider tree by attaching the customer tree to every node that stores net$_k$. Useful applications of extended forward provider trees include detection of black holes and persistent forwarding loops (see examples in Section VI).

**D. Protocol termination condition**

Consider a local verifier that initiated the construction of a reachability tree RT. It terminates tree construction after processing a reply message if the following two conditions for a complete and correct tree are satisfied:

1) for each node in RT, flag = continue if and only if it is a non-leaf node,

2) for each edge in RT, $P \neq \text{null}$.

The first condition ensures that RT includes all paths between the root AS and all reachable ASes. The second condition ensures that RT has all reachability sets for its edges. The termination condition applies to any customer tree, any provider tree, and any extended provider tree.

![Fig. 4: Forward provider tree of AS 58421.](image)

**E. Examples of provider trees of an AS in the Internet**

Figures 4 and 5 show examples of forward and reverse provider trees, respectively, of a real AS in our dataset. Each edge is associated with a non-empty packet set (not shown in the figures). Only non-leaf nodes are shown in the figures with each tier-4 network shown as an oval, each tier-3 network shown as an octagon, and each tier-2 network shown as a rectangle. Each node in the tree is annotated with an AS name and the number and type of leaf nodes connected to it. For example, the rightmost node at the bottom in Figure 4 is annotated with

AS 4648
24 (1) peers
3 (3) providers

where 4648 is its AS number, it connects to 24 peer leaf nodes (including 1 tier-1 network) and 3 provider leaf nodes (including 3 tier-1 networks).
In the forward provider tree in Figure 4, packets are actually routed in the data plane along each forward path from the root AS to every leaf network. Figure 5 shows the the reverse provider tree which is much larger than the forward tree and has many more leaf nodes. Numerous peer links appear in the reverse provider tree. In the reverse provider tree, each reverse path from a leaf node indicates an available path in the data plane from the leaf network to the root AS.

We provide statistics of reachability trees in the next section.

V. EXPERIMENTS TO CONSTRUCT REACHABILITY TREES

A. Methodology

We implemented COVE as an application-layer protocol. We ran experiments by using up to 170 workstations connected by Ethernet in our department. (These are workstations in public labs shared by all users.) For experiments with less than 170 local verifiers of ASes, each local verifier ran on a workstation (using one core). For networks consisting of more than 170 ASes, we used a discrete-event packet-level simulator of COVE in which message generation and processing use COVE implementation code. In particular, query and reply messages were encrypted. The message sizes and communication overheads obtained by the simulator are the same as those from COVE running on workstations. Reachability trees constructed by the simulator are also the same.

We manually verified our high-level specification of the COVE protocol. To make sure that there is no implementation error in our code, we ran COVE on workstations to construct forward reachability trees for the Stanford and Internet2 datasets and found them to be the same as reachability trees computed by two different centralized algorithms [15], [28]. (For running COVE, the input and output ports of each router in the Stanford and Internet2 datasets were considered to be public input and output ports of an AS.)

There are 23 tier-4 ASes in the set of monitored ASes. Two of them do not have any provider. For the other 21 tier-4 ASes, we ran COVE on workstations and constructed 21 forward provider trees and 21 reverse provider trees.

For experiments to construct customer trees, we randomly selected 50 non-stub ASes among monitored tier-2 and tier-3 ASes (there are 2,573 in total). Figure 6 shows customer cone size distributions. Note that customer cone sizes of monitored ASes are much larger than those of unmonitored ASes. This is because BGP route collectors usually peer with large networks that have large customer cone sizes.

The customer cone sizes of ASes selected for our experiments are, on average, much larger than those of monitored ASes. The median customer cone sizes of monitored tier-2 and tier-3 ASes are 5 and 1, respectively. The median customer cone size of our 50 selected ASes is 91 (with the largest being 5,863).

In experiments to be presented, all ASes in each customer or provider cone deploy COVE.

B. Tree Statistics

Customer trees. Figures 7(a)-(b) show the numbers of tree nodes (leaf and non-leaf) and height versus cone size for forward and reverse trees of the 50 selected ASes. The height of a reachability tree is defined to be the length of the longest path between the root and a leaf node.

In Figure 8(a), the bottom two curves are distributions of the numbers of paths between the root AS and another AS in the forward and reverse trees. The vast majority of the ASes have one path from, or to, the root AS. Because we obtained the dataset from control plane information, there are no default
routes in the dataset. This explains why the average number of paths from an AS to the root AS in a reverse customer tree was found to be 0.87, showing that some ASes need default routes to reach the tier-2 or tier-3 provider at the root.

Multiple paths between two ASes in a reachability tree often share links. To quantify such path dependency, we also computed the expected number of paths after one link is down (with each link in the paths between the two ASes chosen equally likely to be down). The top two curves in Figure 8(a) are distributions obtained when one link was down.

C. Protocol construction overheads

Provider trees. Figures 7(c)-(d) show the number of tree nodes (leaf and non-leaf) and height versus cone size for forward and reverse provider trees of the 21 monitored tier-4 ASes.

Figure 8(b) shows distributions of the numbers of paths between the root AS and another AS in the forward and reverse trees. Distributions for the expected number of paths when one link is down are also shown in the figure. Reverse provider trees have an order of magnitude more paths than forward provider trees. On average, there are 11.26 paths from a provider to the root AS in a reverse provider tree; there are 1.49 paths from the root AS to a provider in a forward provider tree.

Message sizes. Figure 9(a) shows the distributions of sizes of query messages for constructing customer trees and provider trees. Figure 9(b) shows the distributions of sizes of reply messages for constructing customer trees and provider trees. The average size of one query message is 69.5 KB and the average size of one reply message is 35.7 KB, which were calculated over all messages used for constructing customer and provider trees.

Number of message transmissions to construct a tree. Figures 10(a) and 10(b) show the average number of message transmissions per local verifier versus cone size to construct a forward and a reverse customer tree, respectively, for the 50 selected ASes. Figures 10(c) and 10(d) show the average number of message transmissions per local verifier versus cone size to construct a forward and a reverse provider tree, respectively, for the 21 monitored tier-4 ASes.

Each message transmission is a TCP transmission from a local verifier to another (one hop). Every query message is transmitted in one hop. A reply message may be transmitted over multiple hops to reach the local verifier that initiated tree construction. Therefore, the number of reply message transmissions is larger than the number of query message transmissions.

Protocol run time to construct a tree. The run time to construct a reachability tree is measured from the time a local verifier, \( LV_a \), initiates tree construction by sending queries to its neighbors until \( LV_a \) terminates tree construction after receiving replies. Figures 11(a) and 11(b) show protocol run times to construct a forward customer tree and a reverse customer tree, respectively, for customer cones with less than 170 ASes. Constructing a reverse customer tree takes an order of magnitude more time than constructing a forward customer tree for the same customer cone. On average, local verifiers running on workstations connected by Ethernet took 0.43 second to construct a forward customer tree and 2.56 seconds to construct a reverse customer tree.

Figures 11(c) and 11(d) show the protocol run times to
construct a forward provider tree and a reverse provider tree for all monitored tier-4 ASes, respectively. On average, a forward provider tree was constructed in 6.7 seconds, and a reverse provider tree was constructed in about 10 seconds after its inbound traffic shift can be detected. Using COVE, the AS can quickly estimate the resultant inbound traffic shift.

With the reverse provider tree, the AS can use historical traffic data to quickly estimate the resultant inbound traffic shift. For example, in Figure 5, AS 4768 has two providers: AS 9901 and AS 4648. We carried out three experiments running COVE in which a specific prefix, 202.0.58.0/24, is advertised by AS 4768. In each experiment, AS 4768 changed the number of times its AS number is prepended in route advertisements to AS 58421 on average. After adding a provider link to AS 9304, a provider has 13.7 reverse paths on average (Figure 13(b)).

**Minimum height of reverse provider tree.** The operator finds that adding a provider link to AS 4766 minimizes the height of the reverse provider tree to 8 (Figure 13(a)). The height of the original reverse provider tree of AS 58421 is 11.

**Maximum number of paths in reverse provider tree.** The operator finds that adding a provider link to AS 9304 maximizes the average number of reverse paths to AS 58421. In the original reverse tree, a provider has 12.7 reverse paths to AS 58421 on average. After adding a provider link to AS 9304, a provider has 13.7 reverse paths on average (Figure 13(b)).

**Maximum expected number of paths after one link down in reverse provider tree.** The operator finds that adding a provider link to AS 4589 maximizes the average of expected numbers of reverse paths after one link down. In the original reverse tree, an AS has 10.3 paths to AS 58421 on average after one link down. After adding a provider link to AS 4589, a provider has 11.6 paths to AS 58421 on average after one link down (Figure 13(c)).

**C. Default Routes**

Default routes are widely used by ASes in the Internet. Experimental results [7] showed that only 19% of stub ASes, 42% of small ISPs, and 61% of large ISPs were found to be default-free. Since our dataset was derived from control plane information of the Internet, it does not include additional paths in the Internet made possible by default routes. In what
follows, we present results that quantify such increase as well as the effectiveness of default routes configured as backup.

Additional paths. In each AS in the 50 customer and 21 provider cones in our dataset, we configured a default route to its provider if the AS has one provider or to a randomly selected provider if the AS has multiple providers. Default routes configured from customer to provider can increase the number of paths in reverse customer trees and forward provider trees only.

We found that after adding default routes, the average number of paths from an AS in a reverse customer tree to the root AS increased to 1.40 from 0.87 before. After adding default routes, the average number of paths from the root AS to an AS in a forward provider tree increased to 1.51 from 1.49 before. Thus, default routes are much more effective for stub ASes than for ISPs, as expected.

Default routes as backup. For each provider link of an AS (in the 50 customer cones and 21 provider cones) with multiple providers, we provision for its failure by configuring backup default routes to another provider (randomly selected if more than one remaining providers after link failure). We constructed reverse customer trees for the 50 selected ASes and forward provider trees for the 21 monitored tier-4 ASes and, for each AS affected by a link failure, computed the number of paths after a link failure. This number is compared to the number of paths after a link failure for the same set of ASes when no backup default route is configured.

Figure 14 shows distributions of numbers of paths after a link failure for two cases: (i) backup default routes are configured, and (ii) no backup default route. For reverse customer trees, an AS affected by a link failure has 1.36 paths after the failure to the root AS, on average, when there are backup default routes; this number decreases to 0.33 when there is no backup default route. For forward provider trees, an AS affected by a link failure has 2.74 paths after the failure, on average, from the root AS if there are backup default routes; this number decreases to 1.66 if no backup default route. We conclude that default routes, configured as backup, are very effective.

D. Detecting Black Holes

Black hole detection is useful for debugging some reachability problems. The local verifier of an AS, net1, can detect black holes in ASes stored in nodes of its reachability trees. We define a black hole in an AS as a prefix of packets that can enter the AS but are not forwarded out of the AS, except for packets whose destinations are inside the AS. ($LV_1$ can find, in the BGP routing tables of net1, prefixes originated from each AS stored in a tree.)
In a forward tree of net₁, packets in the reachability set associated with each edge, say from netₓ to netᵧ, are actually forwarded by netₓ. In a reverse tree, the reachability set associated with each edge, say from netᵧ to netₓ, is a superset of the set of packets that are actually forwarded by netᵧ. Furthermore, in a forward tree, LV₁ finds black holes that are prefixes originated from other ASes. In a reverse tree, LV₁ finds black holes that are prefixes originated from net₁ itself. For these reasons, the algorithms for computing black holes of ASes in forward and reverse trees are slightly different (these algorithms are omitted due to space limitation).

Of the black holes in an AS, netₖ, found by our algorithms in reachability trees rooted at net₁, some may be due to black holes in forwarding tables within netₖ or packets intentionally dropped by routers in netₖ. The remaining black holes should be investigated.

Consider the provider cone of AS 58421 in Figure 5. Suppose AS 4637 advertises an aggregated route to 134.159.0.0/16 to replace multiple more specific prefixes. There are unused addresses in 134.159.0.0/16 and route aggregation creates black holes [16] in forwarding tables of AS 4637. Furthermore, suppose AS 4637 has default routes to AS 4766 and AS 4766 has default routes to AS 2516. As a result, packets destined to black holes in 134.159.0.0/16 will travel in a persistent forwarding loop from AS 4367 to AS 4766, to AS 2516, and then back to AS 4637, as shown in Figure 16. It has been shown that default routes are used in many small and large ISPs [7] and route aggregation may cause black holes and forwarding loops [16].

The persistent forwarding loop described above can be detected as follows: The local verifier of AS 58421 first computes its forward provider tree, and then extends the tree with AS 2516’s forward customer tree. AS 2516 has an unusually large customer cone (8,781 ASes) and its forward customer tree has 22,512 nodes and occupies 82.2 megabytes. Given the customer tree of AS 2516, the time to compute the extended provider tree and detect the loop took only 4.2 seconds of computation time running on one workstation core.

We illustrate this example with AS 2516 that has an unusually large customer cone (8,781 ASes) to demonstrate scalability of COVE and our approach. In comparison, of the 1,694 monitored tier-2 ASes in our Internet dataset, the 90-percentile value of customer cone size is 202 ASes; of the 879 monitored tier-3 ASes, the 90-percentile value of customer cone size is 16 ASes.

E. Partial deployment

In the provider or customer cone of an AS, when only some ASes in the cone deploy COVE, the AS’s reachability trees constructed do not include all ASes in the cone. Nevertheless, these partial trees still provide useful data plane reachability information. In particular, the AS can find multiple forward and reverse paths in partial trees. The AS can find inbound traffic changes in a partial reverse provider tree after tuning its route advertisements. When the AS’s operator plans a link change, a partial reverse provider tree still provides useful information. The AS can still check black holes in ASes in a partial forward tree.
However, the AS may not be able to check black holes in ASes in a partial reverse tree that does not have some ASes in the cone. Detecting persistent forwarding loops may not be possible when the forward provider tree has to be extended with the forward customer tree of a non-COVE AS.

VII. RELATED WORK

Active probes (pings and traceroutes) have been widely used to measure data plane reachability [27], [13], [7], [12], [14], [11] but they have limitations and biases, some of which are discussed in Introduction. Prior to COVE, there was no method/protocol for constructing reverse reachability trees in the Internet data plane. Reverse provider trees constructed by COVE show that there are numerous reverse paths in the data plane to reach an AS. But currently, an AS does not have any reverse path information. An existing approach to find some available reverse paths is by performing AS-path poisoning in the control plane to trigger BGP rerouting [7], [14].

Another limitation of active probes not mentioned in Introduction is that the operator of a network needs help at other Internet locations to send probes back to her own network to discover reverse reachability paths. An approach to piece together an approximate reverse path to an AS using known paths from many vantage points in the Internet was proposed [12]. This approach provides an estimated reverse path which is better than assuming a symmetric path but does not provide enough accuracy for many network management decisions.

Using active probes, persistent forwarding loops from configuration errors were found to be prevalent in the Internet [27]. Numerous papers have been published on detecting prefix hijacking using active probes in the data plane as well as control plane data. The reader is referred to two recent papers on this topic [5], [24] and references therein.

VIII. CONCLUSION

We designed and implemented an application-layer protocol (COVE) and an efficient data structure (based on BDD) for automating the exchange of data plane reachability information between networks in a business relationship. The inter-network reachability information in the Internet data plane obtained using COVE is ground truth assuming that local verifiers of networks in a business relationship provide ground truth information to each other.

We constructed an Internet dataset, consisting of 2,649 monitored ASes each of which has all of its provider, peer, and customer links, for evaluating the performance of COVE.5 We found COVE to be scalable to very large ASes in the Internet. COVE was designed to improve a network’s views of forward and reverse reachability in the Internet data plane even if COVE is partially deployed by as few as two networks connected by a link. Once there is a set of interconnected networks that deploy COVE, there is strong incentive for another network to join and for the set to accept a new member to increase every network’s views of data plane reachability. We also illustrated how network operators can use COVE to perform very useful network management tasks which cannot be done effectively using existing methods and tools.

Acknowledgment. This work was sponsored by National Science Foundation grant CNS-1214239. We thank the anonymous reviewers of ICNP for their constructive comments.

REFERENCES