Compact Location Encodings for Scalable Internet Routing

Feng Wang[†], Lixin Gao[*], Xiaozhe Shao[*], Hiroaki Harai[§], Kenji Fujikawa[§]
[†] School of Engineering and Computational Science, Liberty University, Lynchburg, VA 24515, USA
[*] Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003, USA
[§] National Institute of Information and Communications Technology, Tokyo 184-8795, Japan

***Abstract*—The Internet is facing the double-challenge of accelerating growth of routing table size and ever higher reliability requirements. Considerable progress has been made toward the scalability and reliability of the Internet. However, most of the proposals are only partial solutions that address some of the challenges. In this paper, we present a new addressing encoding scheme and a corresponding forwarding mechanism for Internet routing to solve the aforementioned problems. Underlying our design is a succinct data structure that allows us to compactly embed a set of addresses into packet headers. At the same time, the structure allows the data plane to efficiently extract multiple address information for the same destination without decompression. We provide time and space complexity analysis, and present experimental results evaluating the performance of our encoding method. It shows that the proposed encoding method can achieve a good compression factor without degrading packet-forwarding performance.**

I. Introduction

Today, the world is becoming more interconnected. On the one hand, many new heterogeneous network technologies and applications of very large scale are emerging, such as data centers and social network applications. On the other hand, according to the predictions from Cisco Systems, by 2020, 50 billion "smart" embedded devices will be connected to the Internet [1]. As the address space of IPv4 is nearly exhausted, the increasing users, applications, services, and devices pose a great demand on the Internet. At the same time, Internet users expect to have any information/service available at their fingertips. This exacerbates the demand for a highly available and scalable Internet.

Considerable progress has been made toward the scalability and reliability of the Internet. The well-known schemes include the locator/ID separation based solutions and hierarchical routing. However, most of the proposals are only partial solutions that address some of the challenges, and have difficulty meeting the high availability and scalability requirements. The locator/ID separation based solutions, such as LISP and ILNP [2], [3], [4], can significantly reduce routing table size and enhance the support of multi-homing. However, these approaches rely on an identifier-to-locator mapping to recover a failure in the Internet. The identifier-to-locator mapping might not be available at all intermediate nodes, and/or might need to resort to active probing to discover the best mapping. As a result, when there is a temporary link or node failure, it might take much longer than typical end-to-end latency to reach a service or host [5], [6]. Hierarchical routing has long been proposed to reduce the overhead of routing table [7], [8]. Kleinrock and Kamouns [7] showed that hierarchical routing can reduce the length of the routing table under certain assumptions. However, previous work has shown that implementing resilient routing in hierarchical routing is challenging [9], [10], [6], [11].

We believe that the challenges of achieving resilient Internet routing are mainly rooted in the weakness of the current data plane. In the existing proposals, topologically dispersed locations are intended to be used as a means to increase resilience of the Internet. The problem is that even though a host may have multiple addresses, both the control plane and the data plane are not aware of this information. Specifically, the data plane has no way to associate the addresses with the same destination so that it treats them separately and individually.

After recognizing that the key issue for addressing the aforementioned problems is to embed multiple addresses into packets, we propose to exploit multiple addressing – packets carrying multiple destination addresses instead of one single address to meet the scalability and availability requirements. Nonetheless, an efficient implementation of this method is not trivial. There are several challenges in supporting multiple addressing. First, simply embedding multiple addresses into packets will substantially increase the space overhead of packet headers. Previous work has proposed some encoding algorithms, such as XBW-transform [12], to compress a labeled tree. However, it is still not clear how to employ those compression methods in multiple addressing, and the impact that those methods will have on the performance of packet forwarding. Second, multiple addressing requires a fast and efficient forwarding mechanism. Specifically, when a compression method is employed, the time of decompression becomes another critical issue.

In this paper, we develop a succinct data structure, Compact Prefix Directed Acyclic Graph (CP-DAG), which allows us to encode multiple addresses in the packet header in a space efficient way. And, most importantly, the structure allows the data plane to extract multiple addresses without decompression. Subsequently, we propose a corresponding packet forwarding plane. Finally, we call to the reader's attention to the fact that the idea to encode multiple addresses into the packet header is different from source routing. In source routing, such as recent work SlickPackets [13], the sender specifies alternative paths

in the packet header that a packet could traverse the network. In order to decide the paths, a map of the available links from the sender to the source is required, which imposes much higher overhead on source routing. Unlike source routing, in our solution a sender simply specifies multiple destination addresses instead of one single address in the packet header. Each router in the network determines the path based on the packet's destination addresses. Thus, the overhead of our solution is relatively smaller than that of source routing because our solution does not need to determine the end-to-end paths.

We extensively evaluated the scalability and efficiency of our design using practical data generated from today's Internet AS-level topology. Our time and space complexity analysis and experimental results suggest that the proposed CP-DAG encoding and the corresponding forwarding mechanism can achieve fast and efficient packet forwarding, and impose low packet space and access overhead. Hence, the proposed encoding method and the forwarding mechanism can scale to the future Internet routing.

The rest of the paper is organized as follows. In Section II, we introduce an addressing scheme allowing large numbers of nodes to co-exist and communicate in the Internet and the forwarding plane. In Section III, we introduce the concept of a CP-DAG. In Section IV, we describe how to perform packet forwarding based on CP-DAGs. In Section V, we present our measurement results. In Section VI, we present the related work. We conclude the paper in Section VII with a summary.

II. Approach Overview

In this section, we provide an overview of our solution. The purpose of this overview is to highlight the improved scalability and reliability features of our solution. We first briefly introduce the addressing scheme that is based on our previous work, HANA and HIMALIS [8], [14]. And then, we present the high level idea of the corresponding forwarding plane. Finally, we underline the fundamental challenges in implementing the forwarding mechanism.

A. Provider-rooted Addressing Scheme

The proposed addressing scheme follows hierarchical provider-rooted addressing and the identifier/locator separation design principles. The address of each host is split into two components: a *host name* and a *locator*. The name of a host distinguishes the host among all other hosts. Each host is assigned a globally unique host name by a naming scheme. As proposed in HIMALIS [14], a host name has two parts: *local host name* and *global domain name*. The host name is used for identifying a host, and would not change even if the host moved to another location.

A locator is used to describe the network attachment point(s) to which a host connects. Each host can have multiple locators representing different routes toward it. A host name will map to one locator if the host has only a single topological location. Otherwise, multiple topological locations will map to the same host name. We can use the existing hierarchical mapping system to implement the mapping. For example, a global directory service and an intermediate network mapping service proposed in our previous work HIMALIS [14] are used to maintain the mapping.

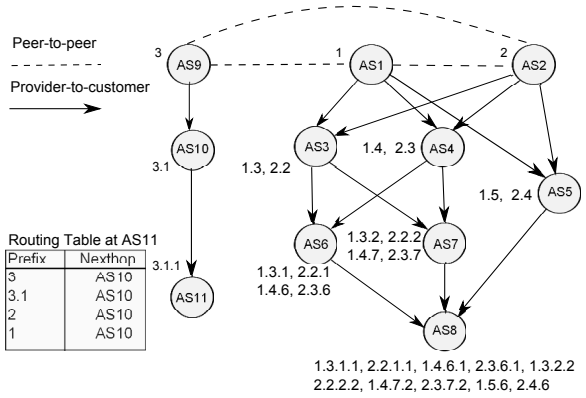


Fig. 1. An example of provider-rooted prefix labelling. The label beside a node corresponds to its prefix.

Each locator is automatically allocated according to the provider-customer hierarchies in the Internet topology. Specifically, the top level or tier-1 providers obtain addresses from an address assignment authority, such as IANA or ICANN. The tier-1 providers can then allocate its address space to its subscribers. In consequence, every subscriber inherits one sub-address from each of its providers.

In this paper, each locator is a variable-length string, which consists of three parts: *prefix*, *midfix*, and *suffix*. All three elements are also variable length. The prefix part consists of a set of integer labels. The first label is allocated by a tier-1 provider, and other labels are allocated by non-tier-1 providers. The midfix and suffix are allocated by the network in which a host resides. Each AS is divided into different areas, and each area is uniquely identified by a midfix. A suffix is a unique host ID within the area. Consequently, a prefix and the following midfix at a certain level of hierarchy can be combined into a new prefix for the level underneath.

Fig.1 is used to demonstrate the prefix labeling scheme. We use delimiter '.' to join elements of a prefix. In this example, AS1 and AS2 are tier-1 providers, and they have addresses `1` and `2`, respectively. As a result, AS3 obtains two prefixes `1.3` and `2.2` from AS1 and AS2. Subsequently, for a host inside AS8, there are ten prefixes associated with the host. Note that in this paper, we focus on the prefix labeling because the midfix and suffix allocation are needed only for intra-domain routing, and different midfix and suffix allocation schemes can be employed inside an AS.

Furthermore, an inter-domain location routing protocol, including HANA proposed in our previous work [8], can be used to propagate routing information about each network location. More specifically, to each provider, an AS only announces the prefixes that are inherited from the provider. The prefixes assigned by other providers cannot be advertised to the provider. In addition, the provider does not have to inform customers of all its locators. It can announce aggregated prefixes to its non-provider neighbors, such as customers and peers. Providers

shall aggregate their customers' addresses and announce the whole address block rather than the specific address block for each customer. For example, in Fig.1, the routing table at AS11 contains two aggregated entries, `1` and `2`, and the two entries, `3` and `3.1`, from its providers.

B. Forwarding Plane

Before we present the forwarding mechanism based on the proposed addressing scheme, we employ a naive method, a pointer-based graph representation to encode a set of prefixes associated with a host. The purpose of this simple encoding is to illustrate the basic idea of the forwarding plane. We will present a more efficient encoding algorithm in the next section. Note that when we encode a set of locators into a packet header, we focus on encoding a set of prefixes. In Section V, we present a way to encode the midfix and suffix numbers.

Upon receiving a packet with a set of prefixes, a router infers every prefix by performing a depth-first search. The router reads the first node. Then, it follows the pointers of the node to retrieve other nodes. Finally, the router concatenates the labels of those visited nodes into a prefix. After that, the router looks up its routing table to determine if the prefix is reachable. If that is the case, it obtains the next hop towards the destination and forwards the packet accordingly. For instance, in Fig.1, suppose that a router at AS3 receives a packet destined to AS8. The first prefix `1.3.1.1` is obtained by employing the depth first search.

If the first derived prefix is unreachable, a router attempts to derive the second prefix and forwards the packet accordingly. If no alternative prefix can be found, the packet is discarded. Each router along the path from the source to the destination does the same. From the above, we observe that the delivery of packets to a destination does not stop as long as a router has an alternative path to the locators of the destination.

C. Challenges

We outline two fundamental challenges in designing an efficient forwarding mechanism for our proposed addressing scheme.

- The space overhead of packet header. We augment the packet header with a list of prefixes, which potentially incurs high packet header overhead. The challenge is to design an efficient compression encoding algorithm to encode the prefixes. In the paper, we propose two compression methods to address this issue.
- The time efficiency of forwarding. The proposed forwarding plane incurs extra packet processing overhead because it traverses the graph to derive the prefixes. Moreover, if we compress the graph, the time of decompression becomes another critical issue. In Section IV, we improve the forwarding plane through a two phase lookup to address this challenge.

In the rest of the paper, we describe our design and implementation to address the two challenges in detail.

III. Compact Location Encoding

In this section, we present two compact location encoding methods. The key idea of our first compression method, Compact Prefix DAG (Directed Acyclic Graph), or CP-DAG, is to compress a pointer-based DAG, where recurrent substructures exist only once. The DAG is then encoded into a bit string. Unlike the first method, *XBW-transform* is a pointerless encoding method based on previous work [12]. Both methods result in a smaller packet header size, but the former is more scalable and efficient than the latter in terms of space and time overhead, according to our analysis and measurements. Thus, we focus on the introduction of the CP-DAG encoding method.

A. CP-DAG Code

First, we present the following steps to construct a CP-DAG:

- The first step is to perform a tree compression. We first represent a set of prefixes in a tree format. To encode a set of prefixes into a tree, we first add a "virtual" root node, whose children are the labels representing the prefixes of tier-1 providers. The fake root node is labeled with 0. This step is to take advantage of common subtrees so that the prefix tree can be transformed into a DAG by eliminating all recurrent subtrees. We use the technique that was introduced by previous work [15], [16] to identify the repeated subtrees. Removing such repetitions finally results in a DAG.
- The second step is to assign a *level ordering* to the result DAG. A level ordering of a DAG is the ordering of the nodes that are visited by a breadth-first search traversal starting at the root and traversing in left-to-right order at each level. The left-to-right ordering is maintained from one level to the next level order. Thus, the level of a node is the number of edges in the path from the root to the node. We refer to the level of the root as level 0. The level below that is level 1, and so on.
- The third step is to label each node. A triplet $s = (chd, label, ptr)$ is associated with each node in the DAG, where chd represents a child type, $label$ represents a label, and ptr represent a pointer. A *child type* is used for storing the children belonging to each node. We use one bit to indicate if the child is the last child of its parent or not. If a node is the last child of its parent, the bit is 1, otherwise it is 0. A *label* is a value allocated by the node's parent or a tier-1 provider' prefix label. A *pointer* is used to tell the order of the first child, and each leaf node has an empty pointer.
- The last step is to generate the codeword. According to the level-order of a CP-DAG, the DAG can be packed consecutively into a bit string, called a *CP-DAG codeword*. The order of each node is the index value of the node in the string.

Thus, in a CP-DAG, each prefix can be represented by a sequence of labels describing the path from the root to a leaf node. In addition, the children of the root are defined

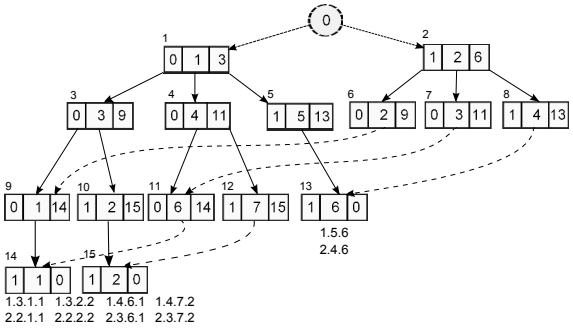


Fig. 2. The structure of a CP-DAG. Each of the nodes of the DAG contains a child type (1 bit), a label, and a pointer.

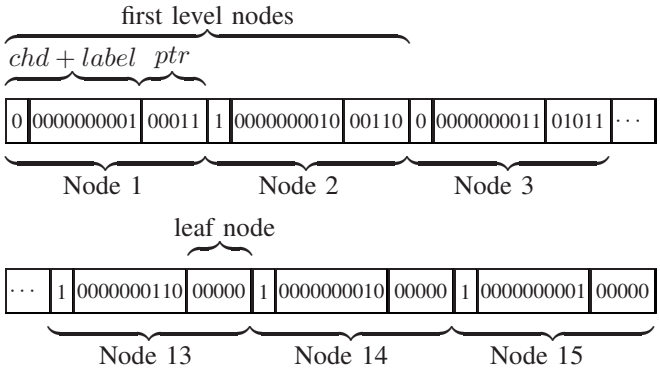


Fig. 3. An example of a CP-DAG codeword associated with the CP-DAG shown in Fig.2.

as the *first level nodes*. Fig. 2 shows the construction of the corresponding CP-DAG of the previous example. The numbers inside each node are a child type, a label and a pointer. The corresponding CP-DAG codeword is shown in Fig.3.

From the figures, we find that all the children of a parent are numbered contiguously. As a result, the order of any child node can be computed efficiently by using a pointer, which points to the beginning of the child node block, and the child type bit, which will identify the last child. This feature is desirable when a router attempts to extract multiple address from a CP-DAG so that it does not need to restart at the top of the DAG to retrieve a new prefix.

Each node in a CP-DAG represents a single label value assigned by a parent. However, we can extend a CP-DAG such that a node can accommodate more than one label. This extension is motivated by the increasingly prevalent peer-to-peer connection scenarios. Due to the page limit, we refer to [17] for more detailed discussion of supporting peer-to-peer connections in a CP-DAG.

B. CP-DAG Operations

For a given CP-DAG codeword, S, we define the following operations:

- $Select(S, i) = s_i$ returns the *i-th* node from S.
- $GetTop(S)$ returns all the first level nodes from S.
- $GetChildren(S, i)$ returns all children of the *i-th* node.

Due to the page limitation, we refer the readers to the full version of this paper for the implementation [17].

C. XBW Code

Our second method is a pointerless tree representation, and is inspired by the XBW transform [12]. The XBW is a generalization of the Burrows-Wheeler Transform that applies to ordered labeled trees. We use the transform to achieve compact storage of a set of prefixes in a tree format.

The basic idea is to represent a set of prefixes in three strings: a string S_{last} that indicates if a node is the last child of its parent, a string S_I that indicates if a node is an interior node or leaf node, and a string S_α containing all the labels of the nodes. We define a triplet $xbt(T) = \{S_{last}, S_I, S_\alpha\}$ as a XBW codeword of a set of prefixes T. For more details about the strings construction, we refer the reader to [12].

Two primitives are defined to navigate the XBW structure: $rank_s(S, q)$ that returns the number of times symbol s occurs in the prefix $S[1, q]$, and $select_s(S, q)$ that returns the position of the q-th occurrence of symbol s in S. For a given XBW codeword $xbw(T)$, we define three operations based on the two primitives: $Select(xbt, i) = s_i$, $GetTop(xbt)$, and $GetChildren(xbt, i)$. $GetTop(xbt)$ returns all the first level nodes from $xbt(T)$, and $GetChildren(xbt, i)$ returns all children of the i-th node. Due to the page limitation, we refer the readers to the full version of this paper for the implementation [17].

D. Space Complexity

In this subsection, we analyze the space complexity of the two representations. First, the information-theoretic lower bound for storing a set of prefixes in a tree T with t nodes is $2t - (\log t) + t\lceil \log |\delta(T)| \rceil$, where $\delta(T)$ is the degree of the tree T [12]. The degree of a tree is the maximum number of outgoing edges of any of its nodes.

Second, for a CP-DAG, suppose that there are $n \leq t$ nodes in the graph. The upper bound of the length of the codeword is equal to the number of nodes times the size of each node. The size of each node is determined by the size of a label and the number of nodes in the DAG, $1 + \lceil \log |\delta(T)| \rceil + \log n$. The total length is equal to $n \times (1 + \lceil \log |\delta(T)| \rceil + \log n)$. Thus, the space complexity is $O(n \log n)$. Third, the XBW representation takes at most $2t + t\lceil \log |\delta(T)| \rceil$ bits [12]. From this analysis, we conclude that the CP-DAG encoding has the smallest space overhead. In Section V, we present our experimental measurements on the two encoding methods, which are consistent with our analysis result.

IV. Codeword-based Packet Forwarding

Before we present an improved codeword-based packet forwarding algorithm, we use an example to demonstrate the idea. In Fig.1, suppose that a host at AS11 sends a packet to a server inside AS8. Based on the path that the packet traverses, we define an *uphill-path* as a path from a sender to the providers of the destination. An uphill-path consists of one or more customer-to-provider links, and one or none

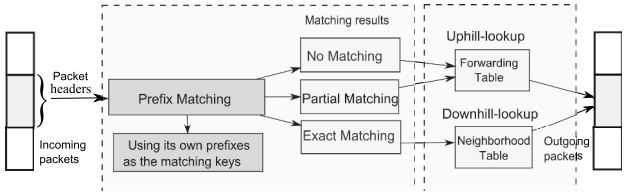


Fig. 4. Overview of codewords based packet forwarding, which consists of two main phases: prefix matching and next hop lookup.

peering link. A *downhill-path* is a path from the providers to its customers, and finally to the destination. A downhill-path consists of one or more provider-to-customer links. In this example, path $AS11 \rightarrow AS10 \rightarrow AS9$ is an uphill-path, and path $AS1 \rightarrow AS3 \rightarrow AS6 \rightarrow AS8$ is a downhill-path.

If we examine the routing tables at the ASes along the uphill-path, for example, AS11, AS10 and AS9 in Fig.1, we find that they have two routes to the destination. Most importantly, the two routes are already aggregated so that they has one single label. Thus, for the ASes along an uphill-path, they do not need to know the whole prefix. This implies that they can forward the packets just based on the first label of the prefixes. Similarly, for the ASes along a downhill path, we find that the prefixes of the destination and the ASes' locators contain a common part. In other words, the locators of those ASes are the prefixes of the destination's locators. Consequently, those ASes can derive the next label directly from the compressed representation and use it to make forwarding decisions. For example, suppose that a router at AS6 receives a packet with locator `1.3.1.1`. One of the router's locators is `1.3.1`, which is a prefix of `1.3.1.1`. In this case, the last label of `1.3.1.1` represents the location of the customer so that the router can use it to make forwarding decisions.

Hence, we propose a new forwarding algorithm, which consists of two main operations: *prefix matching* and *next hop lookup*, as shown in Fig.4. Upon receiving a packet with a CP-DAG or XBW codeword, each router first uses its own prefixes as the matching keys to search the code. Then, the incoming packet is marked by a type based on the matching result. Finally, a forwarding lookup table associated with the packet type is selected to search the outgoing interface. Each router has two different ways to lookup next hop: *uphill-lookup* and *downhill-lookup*. Uphill-lookup is to forward packets along an uphill-path to reach the providers of the destination, while downhill-lookup is to forward packets along a downhill-path from the providers to the destination.

The forwarding algorithm presented in Algorithm 1 is used to forward packets carrying CP-DAG or XBW codewords. To simplify our description, for a node $S[i]$ in a CP-DAG or XBW codeword, we use $last(S[i])$ to denote the child type of the label. If the node is the rightmost or last child of its parent, $last(S[i]) = 1$. Otherwise, $last(S[i]) = 0$. We use $\alpha(S[i])$ to denote the label of $S[i]$. We use $\rho(S[i])$ to indicate if the node is a leaf ($\rho(s) = 0$) or an internal node ($\rho(s) = 1$). Finally, $\pi(S[i])$ is the string obtained by concatenating the labels on the upward path from the root to node s. We use $\oplus$ as the string appending operator, $\sim$ as the prefix matching operator, and $|S|$ to represent the length of S. Finally, we use a set P to denote a router's own prefixes. In the following, we present the details about the algorithm.

A. Prefix Matching

Upon receiving a packet with a CP-DAG or XBW codeword, a router first uses its own prefixes as the matching keys to search the codeword. The prefix matching is performed from the top level to the bottom level:

1) First Level Matching: First, the router reads the first level nodes, and compares the labels of those nodes with its own prefixes. If there is more than one prefix matching the first level labels, the router will select the prefix with the longest prefix matching. If after that there is still more than one prefix, the router will select one of them, according to its routing policy. Note that the first level matching can reduce the set of possible prefixes to one prefix, which is used for further matching.

2) Traversing Other Levels: The router continues to match the selected prefix if the prefix has any unmatched labels. The router follows the pointer of the matched node s_i at the first level (CP-DAG), or the index value (XBW) to retrieve the next level node. The node is the first child of s_i. Its label is compared with the prefix again. If the label does not match the prefix, other children will be retrieved and examined. The matching process will continue until either the whole prefix is matched or it is determined that none of them match the prefix. Thus, the incoming packet is classified as either an "Exact matching" packet if the selected prefix can be matched entirely, or a "Partial matching" packet when they cannot.

B. Next Hop Lookup

Each router maintains two FIBs: a forwarding table and a neighborhood table. A forwarding table only keeps the prefixes advertised by providers or peers, while a neighborhood table contains the labels allocated to customers. Based on the type of an incoming packet, the two tables are used to lookup the next hop:

- **No match packet**. This case implies that the router and the destination do not have the same top provider(s). The router will select all the first level nodes and use the uphill-lookup to search those labels in its forwarding table.
- **Exact match packet**. In this case, the packet already arrives at a provider of the destination so the router uses the downhill-lookup to search the next hop. All the children of the last encountered node are used as the searching keys to search its neighborhood table. Furthermore, if the last encountered node is a leaf, the packet has already arrived at the destination network. The midfix and suffix values of the destination will be used to reach the destination.
- **Partial match packet**. This case means that the router and the destination host have the same provider(s). The

Algorithm 1: *Packet Forwarding Algorithm*

```
input  : a packet with either a CP-DAG or XBW codeword S, and P
output : a set of next hops for the packet, Nexthops
Select the first level nodes S_1 = GetTop(S) ;
foreach S[i] ∈ S_1, P[i] ∈ P do /* first level matching */
    if α(S[i]) ~ P[i] then M ← (S[i], P[i]);
end
if M = ∅ then                                   /* No match */
    foreach S[i] ∈ S_1 do Nexthops = SearchFIB(α(S[i])) ;
    return Nexthops ;
else
    π(S[i]) = π(S[i]) ⊕ α(S[i]) ;
    while |P[i]| > |π(S[i])| do    /* matching other levels */
        child = GetChildren(S[i] ;
        foreach S[j] ∈ child do
            if α(S[j]) ~ P[i] then
                π(S[j]) = π(S[i]) ⊕ α(S[j]) ;
            end
        end
        if No child matching P[i] then break ;
    end
    if P[i] = π(S[j]) then                      /* Exact match */
        C = GetChildren(S[j]) ;
        if C = ∅ then                           /* Leaf node */
            de-encapsulate the packet to get midfix and suffix: hid ;
            Nexthops ← SearchNeg(hid) ;
        else
            foreach c_i in C do Nexthops ← SearchNeg(α(c_i)) ;
            return Nexthop ;
        end
    else                                        /* Partial match */
        Nexthops ← SearchFIB(π(S[i])) ;
        return Nexthop ;
    end
end
```

matched part of the prefix represents the prefix of the common provider. The router will use the uphill-lookup to search the common provider's prefix in its forwarding table.

C. Time Complexity

We first analyze prefix match time and next hop lookup time along an uphill-path. Let t be the number of labels, w be the length of one label, k be the average number of first level nodes in a CP-DAG or XBW codeword, n be the average number of children for a node, and s be the length (in terms of nodes) of the longest prefix. Let m be the average number of a router's prefixes. Prefix match starts at the first level. Then, identifying an unsuccessful match takes $O(mk)$ time. For a "No match" packet, the next hop lookup step takes $O(kw)$ time to search all the k nodes in the forwarding table. Furthermore, it takes $O(mk+s)$ steps to identify a partial match in the worst case. The common parts of the matched prefix are used to perform next hop lookup so that the lookup complexity is $O(sw)$. Thus, for the two codewords, the prefix match time is the same along an uphill-path.

Second, we analyze the match time and the next hop lookup time along a downhill-path. In this case, the packet type should be "Exact match". For a CP-DAG codeword, it takes $O(mk+s)$ steps, and the next hop lookup complexity is $O(nw)$. On the contrary, for an XBW codeword, it takes more time to extract the children because of the pointerless representation. By scanning the two string S_I and S_{last}, it takes $O(mk+3t)$ steps. The next hop lookup complexity is the same as the CP-DAG.

Next, we compare the time complexity of a CP-DAG and XBW codeword with a pointerless list format proposed in [13]. Let N be the total number of prefixes. As shown in Table I, in the worst case, the time to extract all prefixes from the list is $O(N)$, and the lookup time for all the prefixes is $O(swN)$. Thus, the time complexity in a list format is much larger than using a CP-DAG codeword.

Scheme	uphill-path	downhill-path
CP-DAG	$O(mk+s)+O(sw)$	$O(mk+s)+O(nw)$
XBW	$O(mk+s)+O(sw)$	$O(mk+3t)+O(nw)$
List	$O(N)$+ $O(swN)$	$O(N)$+$O(swN)$

TABLE I
TIME COMPLEXITY FOR TWO DIFFERENT ENCODING SCHEMES.

The above analysis shows that at each router, the prefix match time and the next hop lookup time depend on the length of the router's prefixes, not the length of the destination prefixes. This also implies that the time complexity is proportional to the level at which a router forwards a packet. Top level backbone routers only need to access the first level part so that the match and lookup times are greatly reduced compared to IP forwarding. Although low level routers need to explore the codeword to match its prefixes, the next hop lookup time could not be worse than IP forwarding. In order to reduce the lookup time further, we can explicitly store the first level nodes in a packet header when it traverses an uphill-path. After the packet arrives at one of the providers of the destination, the first level nodes are replaced by the provider's prefix. As a result, the following routers can directly use the prefix instead of inferring it from the codeword.

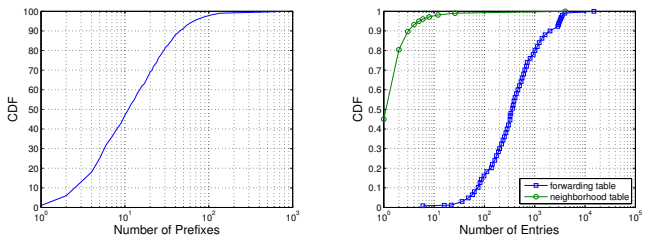
V. PERFORMANCE EVALUATION

In this section, we investigate the scalability of the proposed addressing scheme and the efficiency of the proposed forwarding method.

A. Scalability

1) Scalability of the proposed addressing scheme: We first investigate the scalability in terms of the number of prefixes allocated by the scheme, and the number of entries in the forwarding table and neighborhood table. To measure the scalability on a realistic network topology, we download daily dumps of BGP tables from all monitors deployed by Route-Views [18] and RIPE RIS [19] to generate the current Internet AS-level topology. We use the routing tables in 05/01/2013, which contains a total of 44,863 ASes. For each AS, we build its AS-level graph. The AS relationship inference algorithm proposed in [20] is adopted in this paper because this algorithm gives the most accurate inference compared to the existing algorithms [21], [22], [23].

Fig.5 (a) shows the cumulative distribution of the number of prefixes. It can be seen that 90% of the ASes will have less than 45 prefixes. Fig.5 (b) shows the cumulative distribution



(a) Number of prefixes (b) Size of forwarding tables

Fig. 5. Scalability of the proposed addressing scheme. (a) CDF of the number of prefixes, and (b) CDF of the size of forwarding table and neighborhood table.

of the number of forwarding entries. More than 90% of the ASes have less than 2,000 entries in their forwarding tables, and 4 entries in their neighborhood tables.

2) Scalability of the proposed encoding methods: To investigate the scalability of the encoding methods, we define a compression factor as the ratio between the the proposed prefix encoding formats (CP-DAG and XBW) and a list representation:

$$\text{Compression factor} = \frac{\text{Size of a list format}}{\text{Size of a CP-DAG/XBW codeword}}.$$

The compression performance results are shown in Fig.6. When we encode each AS's prefixes, we use three different label sizes: 8 bits, 10 bits, and 16 bits. In Fig.6(a)-(c), we show the size of the selected prefixes in a CP-DAG codeword, XBW codeword, and list format according to the label sizes. In Fig.6(a), when each label is encoded in 8 bits, the majority of ASes (about 80%) in the Internet have CP-DAG codewords with less than 89 bytes, and about 50% of the ASes have CP-DAG codes with less than 46 bytes. If those prefixes are encoded in XBW codewords, the majority of the ASes have XBW codes with less than 120 bytes, and about half of them have less than 50-byte XBW codewords. On the contrary, if those prefixes are encoded in a list format, the size will be 160 bytes for the majority of the ASes, and half of them are less than 70 bytes. We observe that our encoding methods generate smaller packet header size than the list format, and the size of CP-DAG codes is the smallest among the three formats. From Fig.6(b) and (c), we make the same observations. Note that this observation is consistent with our space complexity analysis in Section III-D.

In Fig.7(a)-(c), we show the compression factor for the CP-DAG and XBW codewords with the three label sizes. From Fig.7(a), we find that 40% of the ASes can achieve a compression factor with more than 1.5 (saving 33% of packet space) when the prefixes are encoded in CP-DAG codes with 8 bits label; only 14% of the ASes can have the same compression factor using XBW encoding. However, when we use 10 bits and 16 bits to represent each label in CP-DAG codes, we find that more than 50% (10-bit) and 60% (16-bit) of the ASes have a compression factor of more than 1.8 (saving 44% of packet space), respectively. Furthermore, for more than 20% (10-bit) and 40% (16-bit) of the ASes, they can save more than 50% of the packet space (compression factor $= 2$) when their prefixes are encoded in CP-DAG codes. Compared to CP-DAG codes, only 20% of the ASes can achieve a compression factor of more than 1.5 by using XBW encoding.

Hence, our experiments show that CP-DAG codes can produce a higher compression rate than XBW codes.

B. Efficiency of Packet Forwarding

The goal of this measurement is to investigate efficiency of the proposed forwarding method in terms of locator lookup delay. The lookup delay includes the delay of decoding individual locators and next-hop lookup latency. To achieve this goal, we first design a common forwarding plane, including sofware/hardware implementations. We evaluate and compare the forwarding overhead of the proposed encoding methods under the same forwarding plane.

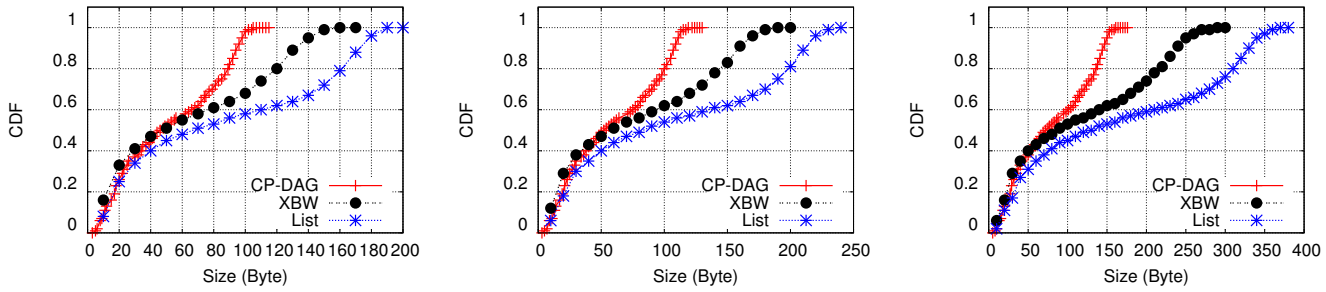
1) Testbed: The testbed consists of a Dell PowerEdge server with Intel Xeon CPUs running at 2.493 GHz clock speed and two desktop computers. One of two computers generates traffic destined to the other one via the sever. The sever is designed as a router to forward the traffic. We use the Click modular router [24] to implement the data forwarding plane on the server.

2) Embedding CP-DAG or XBW in IPv6 Packet Headers: In this paper, a CP-DAG or XBW code is embedded into an IPv6 header. The IPv6 header consists of two parts: the IPv6 base header, and optional extension headers [25]. First, we utilize two IPv6 extension headers to encode a node's host ID. We use a Hop-to-Hop Options header to encode a codeword, and a Destination Options header to encode the midfix and suffix. Second, to reduce the lookup delay further, we explicitly store the first level nodes or a locator in the original IPv6 source and destination addresses. Due to the page limitation, we direct the readers to the full version of this paper for the implementation [17].

3) Measurement Results: In our implementation, we use 10 bits to represent a label because our scalability measurements in Section V-A have shown that 10-bit label size can produce a better compression rate. We present the measurement results in Fig.8. We measure the lookup delay at two different phases: uphill lookup and downhill lookup. Since tier-1 ASes need to modify the destination address format if a packet is destined to their customers, we measure the lookup delay at tier-1 ASes as well.

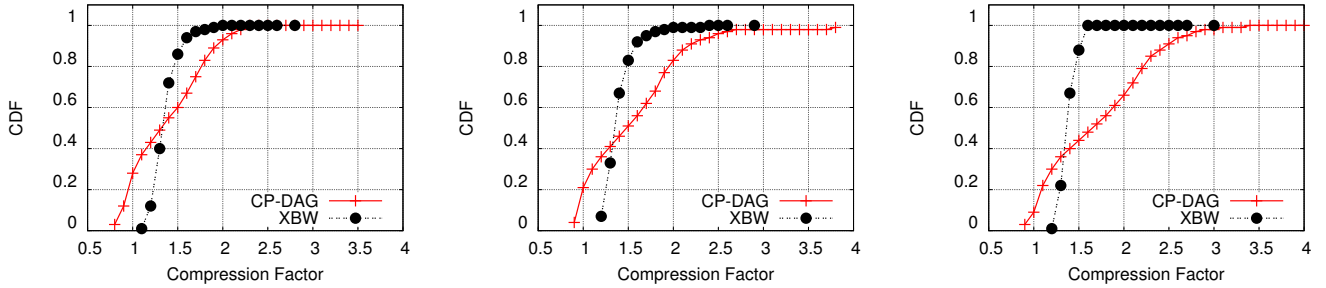
In order to understand the forwarding performance during a failure, we evaluate the lookup delay when the first locator fails. In Fig.8, the labels of X-axle are used to represent the locator failures. "First" means that the first locator is used to forward the packet, while "Second" means that the second locator is used to forward the packet because the first one fails.

We show the uphill lookup delay in Fig.8(a). We see that the uphill lookup delays for CP-DAG and XBW are almost the same. On the contrary, the lookup delay of a list format is the longest because a router needs to search the whole locator to make the forwarding decision. The downhill lookup delay



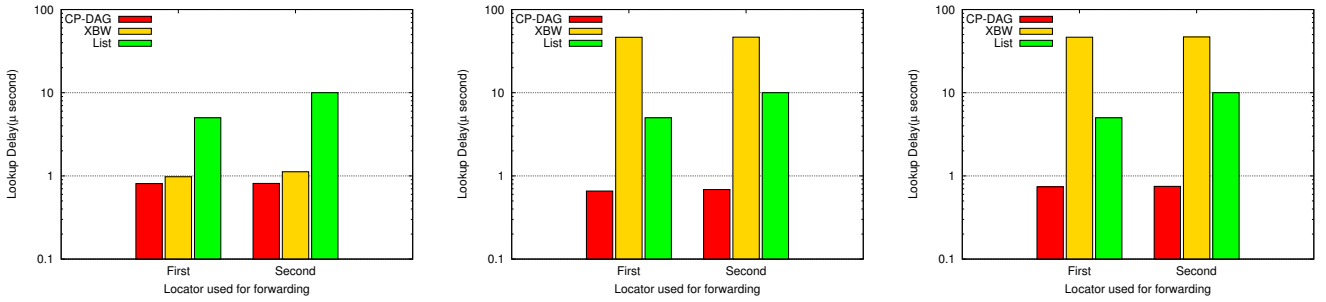
(a) Label with 8 bits (b) Label with 10 bits (c) Label with 16 bits

Fig. 6. CDF of the size of prefixes encoded in a list, CP-DAG, and XBW code format.



(a) Label with 8 bits (b) Label with 10 bits (c) Label with 16 bits

Fig. 7. CDF of compression factors for the prefixes encoded in a list, CP-DAG, and XBW code format.



(a) Uphill Lookup Delay (b) Downhill Lookup Delay (c) Lookup Delay at a tier-1 AS

Fig. 8. Locator lookup delay for different encoding methods.

for XBW is the longest time as shown in Fig.8(b). The reason is because that a downhill router needs to calculate the index of the children, which incurs the packet processing overhead. On the contrary, the children can be inferred by following a pointer in a CP-DAG codeword so that forwarding a packet with a CP-DAG codeword is the fastest. In Fig.8(c), we show the packet processing and lookup delay when a tier-1 router modifies the address. We observe that the delay for a CP-DAG code is the shortest time. Moreover, from Fig.8, we find that the forwarding delay for CP-DAG codewords does not increase significantly during a failure.

Hence, our measurement results show that our forwarding method for CP-DAG codewords is the fastest forwarding mechanism. It does not increase packet header processing cost significantly, and the locator processing and lookup delay is almost the same at each router. Furthermore, it can reduce the failover reaction time.

VI. Related Work

There are several works closely related to ours. NIP [26] is proposed to support multi-homing and mobility capabilities by introducing a three-tuple addressing scheme. However, NIP is based on a flat node ID space, which is different than our hierarchical addressing scheme. In addition, NIP simply represents a set of node IDs in a list format, which causes an increase in data packet size. XIA [27] is a network architecture

to support multiple communication types and allow networks to accommodate new styles of communication. However, addresses in XIA are specified as DAGs, which could result in a large packet header. NIRA [28] is an inter-domain routing system to enable end users to choose their own routes. In NIRA, a route consists of a source address and a destination address. Up-graphs from the source and destination are used to derive the addresses. However, NIRA packets only contain one single source and destination address, which is different from ours. SlickPackets [13] is based on source routing to achieve fast re-routing during a failure. To meet the goal, SlickPackets first generates a forwarding graph, which consists of a set of alternate paths from the source to the destination. And then, the graph is embedded into packet headers. The main differences between our work and SlickPackets are: 1) Our work focuses on both the scalability and reliability issues of Internet while SlickPackets focuses only on the reliability issue, 2) Our method is based on Locator/ID separation, not a source routing based solution so that it does not need to assign link labels that are required in SlickPackets, 3) a CP-DAG is used to compactly represent a set of locators, so it can be easily built based on a host's locators, without obtaining a map of the available links as required in SlickPackets, and 4) a CP-DAG code can achieve a better compression rate than the two list based encoding methods proposed in SlickPackets.

VII. Conclusion and Future Direction

We have presented a CP-DAG based packet forwarding method to improve the scalability and reliability of Internet routing. The proposed CP-DAG can encode multiple prefixes in an efficient way. As our performance analysis and evaluation show, using the proposed addressing scheme and the CP-DAG compression can reduce packet space overhead without degrading packet-forwarding performance.

The future research remains to address issues such as implementing a path-vector based inter-domain and intra-domain routing protocols to propagate routing information about locators and routing inside each area; designing a mapping system to map host names to locators and CP-DAG, and designing protocols for CP-DAG construction and advertisement.

VIII. Acknowledgment

This work was supported by National Science Foundation grant CNS-1402857 and CNS-1402594 and under NSF-NICT Collaborative Research JUNO (Japan-U.S. Network Opportunity) Program.

References

[1] D. Evans, "The Internet of Things: How the Next Evolution of the Internet Is Changing Everything." http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf, 2011.

[2] L. Iannone, D. Saucez, and O. Bonaventure, "Locator/ID Separation Protocol (LISP) Map-Versioning," *RFC 6834*, January 2013.

[3] L. Iannone, D. Saucez, and O. Bonaventure, "Implementing the Locator/ID Separation Protocol: Design and experience," *Computer Networks.*, vol. 55, pp. 948–958, March 2011.

[4] R. Atkinson, S. Bhatti, and U. S. Andrews, "Identifier-Locator Network Protocol (ILNP) Architectural Description," *RFC 6740.*, 2012.

[5] D. Saucez and B. Donnet, "On the Dynamics of Locators in LISP," in *IFIP'12*, pp. 385–396, 2012.

[6] D. Saucez, J. Kim, L. Iannone, O. Bonaventure, and C. Filsfils, "A Local Approach to Fast Failure Recovery of LISP Ingress Tunnel Routers," in *IFIP'12*, pp. 397–408, 2012.

[7] L. Kleinrock and F. Kamoun, "Hierarchical Routing for Large Networks, Performance Evaluation and Optimization," *Computer Networks*, vol. 1, pp. 155–174, January 1977.

[8] K. Fujikawa, H. Tazaki, and H. Harai, "Inter-AS Locator Allocation of Hierarchical Automatic Number Allocation in a 10,000-AS Network," in *SAINT 2012*, 2012.

[9] H.-J. Shin, D. Pei, M. Lad, Y. Choi, and L. Zhang, "The Impact of Multi-homing on Network Reliability and Stability: A Case Study," in *ICCCN 2005*, pp. 543–548, 2005.

[10] R. Gummadi and R. Govindan, "Practical Routing-Layer Support for Scalable Multihoming," in *Infocom 2005*, 2005.

[11] Y. Song, L. Gao, and K. Fujikawa, "Resilient Routing under Hierarchical Automatic Addressing," in *GLOBECOM'11*, pp. 1–5, 2011.

[12] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan, "Compressing and Indexing Labeled Trees, with Applications," *J. ACM*, vol. 57, no. 1, 2009.

[13] G. T. Nguyen, R. Agarwal, J. Liu, M. Caesar, P. B. Godfrey, and S. Shenker, "Slick Packets," *SIGMETRICS Perform. Eval. Rev.*, vol. 39, pp. 205–216, June 2011.

[14] V. P. Kafle, R. Li, D. Inoue, and H. Harai, "Design and Implementation of Security for HIMALIS Architecture of Future Networks," *IEICE Transactions on Information and System*, vol. E96-D, pp. 226–237, February 2013.

[15] P. J. Downey, R. Sethi, and R. E. Tarjan, "Variations on the Common Subexpression Problem," *J. ACM*, vol. 27, pp. 758–771, October 1980.

[16] P. Flajolet, P. Sipala, and J.-M. Steyaert, "Analytic Variations on the Common Subexpression Problem," in *ICALP* (M. Paterson, ed.), vol. 443 of *Lecture Notes in Computer Science*, pp. 220–234, Springer, 1990.

[17] F. Wang, L. Gao, S. Xiaozhe, H. Harai, and K. Fujikawa, "Compact Location Encodings for Scalable Internet Routing." http://rio.ecs.umass.edu/html/publication/.

[18] D. Meyer, "University of Oregon Route Views Project." http://www.routeviews.org/, 2004.

[19] "RIPE RIS, Ripe routing information service." http://www.ripe.net/ris.

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

[21] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz, "Characterizing the Internet Hierarchy from Multiple Vantage Points," in *INFOCOM*, 2002.

[22] L. Gao, "On Inferring Autonomous System Relationships in the Internet," in *Proc. IEEE GLOBAL INTERNET*, November 2000.

[23] G. Di Battista, T. Erlebach, A. Hall, M. Patrignani, M. Pizzonia, and T. Schank, "Computing the Types of the Relationships Between Autonomous Systems," *IEEE/ACM Trans. Netw.*, vol. 15, pp. 267–280, April 2007.

[24] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router," *ACM Trans. Comput. Syst.*, vol. 18, pp. 263–297, August 2000.

[25] S. Deering and R. Hinden, *RFC 2460 Internet Protocol, Version 6 (IPv6) Specification*. Internet Engineering Task Force, December 1998.

[26] A. Gladisch, R. Daher, and D. Tavangarian, "Node-oriented Internet Protocol: A Novel Concept for Enhancement of Mobility and Multihoming in Future Internet," in *LCN Workshops*, pp. 1070–1077, IEEE, 2012.

[27] R. Grandl, D. Han, S.-B. Lee, H. Lim, M. Machado, M. Mukerjee, and D. Naylor, "Supporting Network Evolution and Incremental Deployment with XIA," in *ACM SIGCOMM '12*, (New York, NY, USA), pp. 281–282, ACM, 2012.

[28] X. Yang, D. Clark, and A. W. Berger, "NIRA: A New Inter-domain Routing Architecture," *IEEE/ACM Trans. Netw.*, vol. 15, pp. 775–788, August 2007.