**ABSTRACT**
Modern data centers can consist of hundreds of thousands of servers and millions of virtualized end hosts. Managing address assignment while simultaneously enabling scalable communication is a challenge in such an environment. We present ALIAS, an addressing and communication protocol that automates topology discovery and address assignment for the hierarchical topologies that underlie many data center network fabrics. Addresses assigned by ALIAS interoperate with a variety of scalable communication techniques. ALIAS is fully decentralized, scales to large network sizes, and dynamically recovers from arbitrary failures, without requiring modifications to hosts or commodity switch hardware. We demonstrate through simulation that ALIAS quickly and correctly configures networks that support up to hundreds of thousands of hosts, even in the face of failures and erroneous cabling, and we show that ALIAS is a practical solution for auto-configuration with our NetFPGA testbed implementation.

**Categories and Subject Descriptors**
C.2.1 [Network Architecture and Design]: Network Communications; C.2.2 [Computer-Communication Networks]: Network Protocols

**General Terms**
Algorithms, Design, Management, Performance, Reliability

**Keywords**
Data Center Address Assignment, Hierarchical Labeling

**1. ALIAS**
The goal of ALIAS is to automatically assign globally unique, topologically meaningful host *labels* that the network can internally employ for efficient forwarding. We aim to deliver one of the key benefits of IP addressing—hierarchical address assignments such that hosts with the same prefix share the same path through the network and a single forwarding table entry suffices to reach all such hosts—without requiring manual address assignment and subnet configuration. A requirement in achieving this goal is that ALIAS be entirely decentralized and broadcast-free. At a high level, ALIAS switches automatically locate clusters of good switch connectivity within network topologies and assign a shared, non-conflicting prefix to all hosts below such pockets. The resulting hierarchically aggregatable labels result in compact switch forwarding entries.\(^1\) Labels are simply a reflection of current topology; ALIAS updates and reassigns labels to affected hosts based on topology dynamics.

### 1.1 Environment
ALIAS overlays a logical hierarchy on its input topology. In this logical hierarchy, switches are partitioned into * levels and each switch belongs to exactly one level. Switches connect predominantly to switches in the levels directly above or below them, though pairs of switches at the same level (peers) may connect to each other via *peer links*.

One high-level dichotomy in multi-computer interconnects is that of direct versus indirect topologies [19]. In a direct topology, a host can connect to any switch in the network. With indirect topologies, only a subset of the switches connect directly to hosts; communication between hosts connected to different switches is facilitated by one or more intermediate switch levels. We focus on indirect topologies because such topologies appear more amenable to automatic configuration and because they make up the vast majority of topologies currently deployed in the data center [1, 11, 14, 3, 6]. Figure 1(a) gives an example of an indirect 3-level topology, on which ALIAS has overlaid a logical hierarchy. In the Figure, \(S_x\) and \(H_x\) refer to unique IDs of switches and hosts, respectively.

A host with multiple network interfaces may connect to multiple switches, and will have separate ALIAS labels for each interface. ALIAS also assumes that hosts do not play a switching role in the network and that switches are programmable (or run software such as OpenFlow [2]).

\(^1\)We use the terms “label” and “address” interchangeably.
1.2 Protocol Overview
ALIAS first assigns topologically meaningful labels to hosts, and then enables communication over these labels. As with IP subnetting, topologically nearby hosts share a common prefix in their labels. In general, longer shared prefixes correspond to closer hosts. ALIAS groups hosts into related clusters by automatically locating pockets of strong connectivity in the hierarchy—groups of switches separated by one level in the hierarchy with full bipartite connectivity between them. However, even assigning a common prefix to all hosts connected to the same leaf switch can reduce the number of required forwarding table entries by a large factor (e.g., the number of host facing switch ports multiplied by the typical number of virtual machines on each host).

1.2.1 Hierarchical Label Assignment
ALIAS labels are of the form \((c_{n-1} \ldots c_1.H.WM)\); the first \(n-1\) fields encode a host’s location within an \(n\)-level topology, the \(H\) field identifies the port to which each host connects on its local switch, and the \(WM\) field provides support for multiple VMs multiplexed onto a single physical machine. ALIAS assigns these hierarchically meaningful labels by locating clusters of high connectivity and assigning to each cluster (and its member switches) a coordinate. Coordinates then combine to form host labels; the concatenation of switches’ coordinates along a path from the core of the hierarchy to a host make up the \(c_i\) fields of a host’s label.

Prior to selecting coordinates, switches first discover their levels within the hierarchy, as well as those of their neighbors. Switches \(i\) hops from the nearest host are in level \(L_i\), as indicated by the \(L_1, L_2, L_3\) levels in Figure 1(a).

Once a switch establishes its level, it begins to participate in coordinate assignment. ALIAS first assigns unique \(H\)-coordinates to all hosts connected to the same \(L_1\) switch, creating multiple one-level trees with an \(L_1\) switch at the root and hosts as leaves. Next, ALIAS locates sets of \(L_2\) switches connected via full bipartite graphs to sets of \(L_1\) switches, and groups each such set of \(L_2\) switches into a hypernode (HN). The intuition behind HNs is that all \(L_2\) switches in an \(L_2\)HN can reach the same set of \(L_1\) switches, and therefore these \(L_2\) switches can all share the same prefix. This process continues up the hierarchy, grouping \(L_i\) switches into \(L_i\)HNs based on bipartite connections to \(L_{i-1}\)HNs.

Finally, ALIAS assigns unique coordinates to switches, where a coordinate is a number shared by all switches in an HN and unique across all other HNs at the same level. By sharing coordinates between HN members, ALIAS leverages the hierarchy present in the topology and reduces the number of coordinates used overall, thus collapsing forwarding table entries. Switches at the core of the hierarchy do not require coordinates and are not grouped into HNs (see Section 2.4 for more details). \(L_1\) switches select coordinates without being grouped into HNs. Further, we employ an optimization (Section 2.2) that assigns multiple coordinates to an \(L_1\) switch, one per neighboring \(L_i\)HN.

When the physical topology changes due to switch, host, or link failure, configuration changes, or any other circumstances, ALIAS adjusts all label assignments and forwarding entries as necessary (Sections 2.3 and 3.3).

Figure 1(b) shows a possible set of coordinate assignments and the resultant host label assignments for the topology of Figure 1(a); only topology-related prefixes are shown for host labels. For this 3-level topology, \(L_2\) switches are grouped in HNs (as shown with dotted lines), and \(L_1\) switches have multiple coordinates corresponding to multiple neighboring \(L_2\)HNs. Hosts have multiple labels corresponding to the \(L_2\)HNs connected to their ingress \(L_1\) switches.

1.2.2 Communication
ALIAS’s labels can be used in a variety of routing and forwarding contexts, such as tunneling, IP-encapsulation, or MAC address rewriting [13]. We have implemented one such communication technique (based on MAC address rewriting) and present an example of this communication below.

An ALIAS packet’s traversal through the topology is controlled by a combination of forwarding (Section 3.2) and addressing logic (Section 2.2) Consider the topology shown in Figure 1(b). A packet sent from \(H_1\) to \(H_2\) must flow up-
ward to one of $S_7$, $S_8$, or $S_9$, and then downward towards its destination. First, $H_4$ sends an ARP to its first-hop switch, $S_{13}$, for $H_2$’s label (Section 3.3). $S_{13}$ determines this label (with cooperation from nearby switches if necessary) and responds to $H_4$. $H_4$ can then forward its packet to $S_{13}$ with the appropriate label for $H_2$, for example $(1.2.1.0)$ if $H_2$ is connected to port 1 of $S_2$ and has VM coordinate 0. At this point, forwarding logic moves the packet to one of $S_7$, $S_8$, or $S_9$, all of which have a downward path to $H_2$; the routing protocol (Section 3.1) creates the proper forwarding entries at switches between $H_4$ and the core of the network, so that the packet can move towards an appropriate $L_3$ switch. Next, the packet is forwarded to one of $S_2$, $S_6$, or $S_0$, based on the $(1.x.x.x)$ prefix of $H_2$’s label. Finally, based on the second field of $H_2$’s label, the packet moves to $S_2$ where it can be delivered to its destination.

1.3 Multi-Path Support

Multi-rooted trees provide multiple paths between host pairs, and routing and forwarding protocols should discover and utilize these multiple paths for good performance and fault tolerance. ALIAS provides multi-path support for a given destination label via its forwarding component (Section 3.2). For example, in Figure 1(b), a packet sent from $H_4$ to $H_2$ with destination label $(1.2.1.0)$ may traverse one of five different paths.

An interesting aspect of ALIAS is that it enables a second class of multi-path support: hosts may have multiple labels, where each label corresponds to a set of paths to a host. Thus, choosing a label corresponds to selecting a set of paths to a host. For example, in Figure 1(b), $H_2$ has two labels. Label $(1.2.1.0)$ encodes 5 paths from $H_4$ to $H_2$, and label $(7.3.1.0)$ encodes a single $H_4$-to-$H_2$ path. These two classes of multi-path support help limit the effects of topology changes and failures. In practice, common data center fabric topologies will result in hosts with few labels, where each label encodes many paths. Policy for choosing a label for a given destination is a separable issue; we present some potential methods in Section 3.3.

2. PROTOCOL

ALIAS is comprised of two components, Level Assignment and Coordinate Assignment. These components operate continuously, acting whenever topology conditions change. For example, a change to a switch’s level may change changes to that switch’s and its neighbors’ coordinates. ALIAS also involves a Communication component (for routing, forwarding, and label resolution and invalidation); in Section 3 we present one of the many possible communication components that might use the labels assigned by ALIAS.

ALIAS operates based on the periodic exchange of Topology View Messages (TVMs) between switches. In an $n$-level topology, individual computations rely on information from no more than $n-1$ hops away.

Listing 1 gives an overview of the general state stored at each switch, as well as that related to level assignment. A switch knows its unique ID and the IDs and types (hosts or switches) of each of its neighbors (lines 1-3). Switches also know their own levels as well as those of their neighbors, and the types of links (regular or peer) connecting them to each neighbor (lines 4-6); these values are set by the level assignment protocol (Section 2.1).

2.1 Level Assignment

ALIAS level assignment enables each switch to determine its own level as well as those of its neighbors and to detect and mark peer links for special consideration by other components. ALIAS defines an $L_i$ switch to be a switch with a minimum of $i$ hops to the nearest host. For convenience, in an $n$-level topology, $L_i$ switches may be referred to as cores. Regular links connect $L_1$ switches to hosts, and $L_i$ switches to switches at $L_{i+1}$, while peer links connect switches of the same level.

Level assignment is bootstrapped by $L_1$ switch identification as follows: In addition to sending TVMs, each switch also periodically sends IP pings to all neighbors that it does not know to be switches. Hosts reply to pings but do not send TVMs, enabling switches to detect neighboring hosts. This allows $L_1$ identification to proceed without host modification. If hosts provided self-identification, then the protocol becomes much simpler. Recent trends toward virtualization in the data center with a trusted hypervisor may take on this functionality.

When a switch receives a ping reply from a host, it immediately knows that it is at $L_1$ and that the sending neighbor is a host, and updates its state accordingly (lines 3-4, Listing 1). If a ping reply causes the switch to change its current level, it may need to mark some of its links to neighbors as peer links (line 6). For instance, if the switch previously believed itself to be at $L_2$, it must have done so because of a neighboring $L_1$ switch and its connection to that neighbor is now a peer link.

Based on $L_1$ identification, level assignment operates via a wave of information from the lowest level of the hierarchy upwards; A switch that receives a TVM from an $L_1$ switch labels itself as $L_2$ if it has not already labeled itself as $L_1$, and this process continues up the hierarchy. More generally, each switch labels itself as $L_i$, where $i - 1$ is the minimum level of all of its neighbors.

On receipt of a TVM, a switch determines whether the source’s level is smaller than that recorded for any of its other neighbors, and if so, adjusts its own level assignment (line 4, Listing 1). It also updates its state for its neighbor’s level and type if necessary (lines 3,5). If its level or that of its neighbor has changed, it detects any changes to the link types for its neighbors and updates its state accordingly (line 6). For instance, if an $L_3$ switch moves to $L_2$, links to $L_2$ neighbors become peer links.

<table>
<thead>
<tr>
<th>Listing 1: ALIAS local state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. UID myId</td>
</tr>
<tr>
<td>2. UIDSet nbrs</td>
</tr>
<tr>
<td>3. Map(UID→NodeType) types</td>
</tr>
<tr>
<td>4. Level level</td>
</tr>
<tr>
<td>5. Map(UID→Level) levels</td>
</tr>
<tr>
<td>6. Map(UID→LinkType) link_types</td>
</tr>
</tbody>
</table>
The presence of unexpected or numerous peer links may indicate a miswiring, or erroneous cabling, with respect to the intended topology. If ALIAS suspects a miswiring, it raises an alert (e.g., by notifying the administrator) but continues to operate. In this way, miswirings do not bring the system to a halt, but are also not ignored.

ALIAS’s level assignment can assign levels to all switches as long as at least one host is present. Once a switch learns its level, it participates in coordinate assignment.

### 2.2 Label Assignment

An ALIAS switch’s label is the concatenation of \(n-1\) coordinates, \(c_{n-1}c_{n-2}...c_2c_1\), each corresponding to one switch along a path from a core switch to the labeled switch. A host’s label is then the concatenation of an ingress \(L_1\) switch’s label and its own \(H\) and \(W\) coordinates. As there may be multiple paths from the core switches of the topology to a switch (host), switches (hosts) may have multiple labels.

#### 2.2.1 Coordinate Aggregation

Since highly connected data center networks tend to have numerous paths to each host, per-path labeling can lead to overwhelming numbers of host labels. ALIAS creates compact forwarding tables by dynamically identifying sets of \(L_1\) switches that are strongly connected to sets of \(L_{n-1}\) switches below. It then assigns to these \(L_i\) hypernodes unique \(L_i\) coordinates. By sharing one coordinate among the members of an \(L_i\)HN, ALIAS allows hosts below this HN to share a common label prefix, thus reducing forwarding table entries.

An \(L_i\)HN is defined as a maximal set of \(L_i\) switches that all connect to an identical set of \(L_{i-1}\)HNs, via any constituent members of the \(L_{i-1}\)HNs. Each \(L_i\) switch is a member of exactly one \(L_i\)HN. \(L_2\)HN grouping are based on \(L_1\) switches rather than HNs. In Figure 2, \(L_2\) switches \(S_5\) and \(S_6\) connect to same set of \(L_1\) switches, namely \(\{S_1, S_2, S_3\}\), and are grouped together into an \(L_2\)HN, whereas \(S_4\) connects to \(\{S_1, S_2\}\), and therefore forms its own \(L_2\)HN. Similarly, \(S_7\) and \(S_8\) connect to both \(L_2\)HNS (though via different constituent members) and form one \(L_3\)HN while \(S_9\) forms a second \(L_3\)HN, as it connects only to one \(L_2\)HN below.

![Figure 2: ALIAS Hypernodes](image)

Since \(L_i\)HNS are defined based on connectivity to identical sets of \(L_{i-1}\)HNS, the members of an \(L_i\)HN are interchangeable with respect to downward forwarding. This is the key intuition that allows HN members to share a coordinate, ultimately leading to smaller forwarding tables.

ALIAS employs an optimization with respect to HN grouping for coordinate assignment. Consider switch \(S_1\) of Figure 2, and suppose the \(L_2\)HNS \(\{S_1\}\) and \(\{S_5, S_6\}\) have coordinates \(x\) and \(y\), respectively. Then \(S_1\) has labels of the form \(\ldots x_{c_1} \ldots y_{c_1}\), where \(c_1\) is \(S_1\)’s coordinate. Since \(S_1\) is connected to both \(L_2\)HNS, it needs to ensure that \(c_1\) is unique from the coordinates of all other \(L_1\) switches neighboring \(\{S_1\}\) and \(\{S_5, S_6\}\) (in this example, all other \(L_1\) switches).

It is helpful to limit the sets of switches competing for coordinates, to decrease the probability of collisions (two HNs selecting the same coordinate) and to allow for a smaller coordinate domain. We accomplish this as follows: \(S_1\) has two coordinates, one corresponding to each of its label prefixes, giving it labels of the form \(\ldots x_{c_1} \ldots y_{c_1}\). In this way \(S_1\) competes only with \(S_2\) for labels corresponding to HN \(\{S_1\}\). In fact, ALIAS assigns to each switch a coordinate per upper neighboring HN. This reduces coordinate contention without increasing the coordinate domain size.

#### 2.2.2 Decider/Chooser Abstraction

The goal of coordinate assignment in ALIAS is to select coordinates for each switch such that these coordinates can be combined into forwarding prefixes. By assigning per-HN rather than per-switch coordinates, ALIAS leverages a topology’s inherent hierarchy and allows nearby hosts to share forwarding prefixes. In order for an \(L_i\) switch to differentiate between two lower-level HNs, for forwarding purposes, these two HNs must have different coordinates. Thus, the problem of coordinate assignment in ALIAS is to enable \(L_i\)HNS to cooperatively select coordinates that do not conflict with those of other \(L_i\)HNS that have overlapping \(L_{i+1}\) neighbors. Since the members of an HN are typically not directly connected to one another, this task requires indirect coordination.

To explain ALIAS’s coordinate assignment protocol, we begin with a simplified Decider/Chooser Abstraction (DCA), and refine the abstraction to solve the more complicated problem of coordinate assignment. The basic DCA includes a set of choosers that select random values from a given space, and a set of deciders that ensure uniqueness among the choosers’ selections. A requirement of DCA is that any two choosers that connect to the same decider select distinct values. Choosers make choices and send these requests to all connected deciders. Upon receipt of a request from a chooser, a decider determines whether it has already stored the value for another chooser. If not, it stores the value for the requester and sends an acknowledgment. If it has already stored the requested value for another chooser, the decider compiles a list of hints of already selected values and sends this list with its rejection to the chooser. A chooser reselects its value if it receives a rejection from any decider, and considers its choice stable once it receives acknowledgments from all connected deciders.

We employ DCA within a single \(L_i\)HN and its \(L_{i-1}\) neighbors to assign coordinates to the \(L_{i-1}\) switches, as in Figure
3(a). The members of \( L_2 \)HN \( \{S_5, S_6\} \) act as deciders for \( L_1 \) choosers, \( S_1, S_2, \) and \( S_3 \), ensuring that the three choosers select unique \( L_1 \) coordinates.

Recall that as an optimization, ALIAS assigns to each switch multiple coordinates, one per neighboring higher level HN. We extend the basic DCA to have switches keep track of the HN membership of upward neighbors, and to store coordinates (and an indication of whether a choice is stable) on a per-HN basis. This is shown in Figure 3(b), where each \( L_1 \) switch stores information for all neighboring \( L_2 \) HNs. The figure includes two instances of DCA, that from Figure 3(a) and that in which \( S_5 \) is a decider for choosers \( S_1 \) and \( S_2 \).

Finally, we refine DCA to support coordinate sharing within an HN. Since each member of an HN may connect to a different set of higher level switches (deciders), it is necessary that all HN members cooperate to form a distributed chooser. HN members work with the help of a deterministically selected \emph{representative \( L_1 \) switch} (for example, the \( L_1 \) switch with the lowest MAC address of those connected to the HN). \( L_1 \) switches determine whether they represent a particular HN as a part of HN grouping calculations.

The members of an \( L_n \)HN, and the HN’s representative \( L_1 \) switch collaborate to select a shared coordinate for all HN members as follows: The representative \( L_1 \) switch performs all calculations and makes all decisions for the chooser, and uses the HN’s \( L_1 \) switches as \emph{virtual channels} to the deciders. \( L_n \)HN members gather and combine hints from \( L_1 \) deciders above, passing them down to the representative \( L_1 \) switch for calculations. The basic chooser protocol introduced above is extended to support reliable communication over the virtual channels between the representative \( L_1 \) switch and the HN’s \( L_1 \) switches. Additionally, for the distributed version of DCA, deciders maintain state about the HN membership of their \( L_2 \) neighbors in order to avoid falsely detecting conflicts; a decider may be connected to a single chooser via multiple virtual channels (\( L_2 \) switches) and should not perceive identical requests across such channels as conflicts.

Figure 3(c) shows two distributed choosers in our example topology. Choosers \( \{S_1, S_4\} \) and \( \{S_1, S_5, S_6\} \) are shaded in light and dark grey, respectively. Note that \( S_1 \) is a decider for both choosers while \( S_8 \) and \( S_9 \) are deciders only for the second chooser. \( S_2 \) and \( S_3 \) play no part in \( L_2 \) coordinate selection for this topology.

Our implementation does not separate each level’s coordinate assignment into its own instance of the extended DCA protocol; rather, all information pertaining to both level and coordinate assignment is contained in a single TVM. For instance, in a 5-level topology, a TVM from an \( L_3 \) switch to an \( L_2 \) switch might contain hints for \( L_2 \) coordinates, \( L_3 \)HN grouping information, and \( L_4 \) information on its way down to a representative \( L_1 \) switch. Full details of the Decider/Chooser Abstraction, a protocol derivation for its refinements, and a proof of correctness are available in a separate technical report [22].

Label assignment converges when all \( L_2 \) through \( L_{n-1} \) switches have grouped themselves into hypernodes, and all \( L_1 \) through \( L_{n-1} \) switches have selected coordinates.

### 2.2.3 Example Assignments

Figure 4 depicts the TVMs sent to assign coordinates to the \( L_2 \) switches in Figure 3’s topology. For clarity, we show TVMs only for a subset of the switches. In TVM 1, all core switches disallow the selection of \( L_2 \)-coordinate 3, due to its use in another HN (not shown). \( L_2 \) switches incorporate this restriction into their outgoing TVMs, including their sets of connected \( L_1 \) switches (TVMs 2a and 2b). \( S_1 \) is the representative \( L_1 \) switch for both HNs, as it has the lowest ID. \( S_1 \) selects coordinates for the HNs and informs neighboring \( L_2 \) switches of their HNs and coordinates (TVMs 3a and 3b).

### 2.3 Relabeling

Since ALIAS labels encode paths to hosts, topology changes may affect switch coordinates and hence host labels. For instance, when the set of \( L_1 \) switches reachable by a particular \( L_2 \) switch changes, the \( L_2 \) switch may have to select a new \( L_2 \)-coordinate. This process is coined \emph{Relabeling}.

Consider the example shown in Figure 5 where the highlighted link between \( S_3 \) and \( S_4 \) fails. At this point, affected switches must adjust their coordinates. With TVM 1, \( S_3 \) informs its \( L_1 \) neighbors of its new connection status. Since \( S_1 \) knows the \( L_1 \) neighbors of each of its neighbor-
ing $L_2$ switches, it knows that it remains the representative $L_1$ switch for both HNs. $S_1$ informs $S_4$ and $S_5$ of the HN membership changes in TVM 2a, and informs $S_6$ of $S_4$’s departure in TVMs 2b. Since $S_6$ simply left one HN and joined another, existing HN, host labels are not affected.

The effects of relabeling (whether caused by link addition or deletion) are determined solely by changes to the HN membership of the upper level switch incident on the affected link. Table 1 shows the effects of relabeling after a change to a link between an $L_2$ switch $\ell_2$ and an $L_1$ switch $\ell_1$, in a 3-level topology. Case 1 corresponds with the example of Figure 5; $\ell_2$ moves from one HN to another. In this case, no labels are created nor destroyed. In case 2, one HN splits into two and all $L_1$ switches neighboring $\ell_2$ add a new label to their sets of labels. In case 3, two HNs merge into a single HN, and with the exception of $\ell_2$, all $L_1$ switches neighboring $\ell_2$ lose one of their labels. Finally, case 4 represents a situation in which an HN simply changes its coordinate, causing all neighboring $L_1$ switches to replace the corresponding label.

Changes due to relabeling are completely encapsulated in the forwarding information propagated, as described in Section 3.2. Additionally, in Section 3.3 we present an optimization that limits the effects of relabeling on ongoing sessions between pairs of hosts.

### 2.4 M-graphs

There are some rare situations in which ALIAS provides connectivity between switches from the point of view of the communication component, but not from that of coordinate assignment. The presence of an M-graph in a topology can lead to this problem, as can the use of peer links. We consider M-graphs below and discuss peer links in Section 3.4.

ALIAS relies on shared core switch parents to enforce the restriction that pairs of $L_{n-1}$ HNs do not select identical coordinates. There are topologies, though, in which two $L_{n-1}$ HNs do not share a core and could therefore select identical coordinates. Such an M-graph is shown in Figure 6. In the example, there are 3 $L_2$ HNs, $\{S_4, S_6\}$, and $\{S_7\}$. It is possible that $S_4$ and $S_7$ select the same $L_2$-coordinate, e.g., 3, as they do not share a neighboring core. Since $\{S_5, S_6\}$ shares a parent with each of the other HNs, its coordinate is unique from those of $S_4$ and $S_7$. $L_1$ switches $S_1$ and $S_3$ are free to choose the same $L_1$-coordinates, 1 in this example. As a result, two hosts $H_1$ and $H_3$ are legally assigned identical ALIAS labels, (3.1.4.0), if both $H_1$ and $H_3$ are connected to their $L_1$ switches on the same numbered port (in this case, 4), and have IP coordinate 0.

$H_2$ can now see two non unique ALIAS labels, which introduces a routing ambiguity. If $H_2$ attempts to forward a packet to $H_1$, it will use the label (3.1.4.0). When $S_2$ receives the packet, $S_2$ can send this packet either to $S_5$ or $S_6$, since it thinks it is connected to an $L_2$ HN with coordinate 3 via both. The packet could be transmitted to the unintended destination $H_3$ via $S_5$ or $S_7$, $S_3$. When the packet reaches $S_3$, $S_3$ is in a position to verify whether the packet’s IP address matches $H_3$’s ALIAS label, by referencing a flow table entry that holds IP address-to-ALIAS label mappings. (Note that such flow table entries are already present for the communication component, as in Section 3.3.) A packet

**Figure 4:** Label Assignment: $L_2$-coordinates

**Figure 5:** Relabeling Example

<table>
<thead>
<tr>
<th>Time</th>
<th>$L_2$</th>
<th>TVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>dwn_nbrs($S_1, S_2$)</td>
<td>$L_2$-coordinate: 1</td>
</tr>
<tr>
<td>2a</td>
<td>dwn_nbrs($S_3, S_2$)</td>
<td>$L_2$-coordinate: 3</td>
</tr>
<tr>
<td>2b</td>
<td>HN ($S_3, S_6$)</td>
<td>$L_2$-coordinate: 7</td>
</tr>
</tbody>
</table>

**Table 1: Relabeling Cases**

<table>
<thead>
<tr>
<th>Case</th>
<th>Old HN</th>
<th>New HN</th>
<th>Single $L_1$ switch</th>
<th>Remaining $L_1$ switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Intact</td>
<td>Existing</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>Intact</td>
<td>New</td>
<td>+ 1 label</td>
<td>+ 1 label</td>
</tr>
<tr>
<td>3</td>
<td>Removed</td>
<td>Existing</td>
<td>None</td>
<td>- 1 label</td>
</tr>
<tr>
<td>4</td>
<td>Removed</td>
<td>New</td>
<td>Swap label</td>
<td>Swap label</td>
</tr>
</tbody>
</table>

**Figure 6:** Example M-graph
3. COMMUNICATION

3.1 Routing

ALIAS labels specify the ‘downward’ path from a core to the identified host. Each core switch is able to reach all hosts with a label that begins with the coordinate of any $L_{n-1}$-HN directly connected to it. Similarly, each switch in an $L_i$-HN can reach any host with a label that contains one of the HN’s coordinates in the $i^{th}$ position. Thus, routing packets downward is simply based on an $L_i$ switch matching the destination label’s $(i-1)^{th}$ coordinate to the that of one or more of its $L_{i-1}$ neighbors.

To leverage this simple downward routing, ingress switches must be able to move data packets to cores capable of reaching a destination. This reduces to a matter of sending a data packet towards a core that reaches the $L_{n-1}$-HN corresponding to the first coordinate in the destination label. $L_{n-1}$ switches learn which cores reach other $L_{n-1}$-HNNs directly from neighboring cores and pass this information downward via TVMs. Switches at level $L_i$ in turn learn about the set of $L_{n-1}$-HNNs reachable via each neighboring $L_{i+1}$ switch.

3.2 Forwarding

Switch forwarding entries map a packet’s input port and coordinates to the appropriate output port. The coordinate fields in a forwarding entry can hold a number, requiring an exact match, or a ‘don’t care’ (DC) which matches all values for that coordinate. An $L_i$ switch forwards a packet with a destination label matching any of its own label prefixes downward to the appropriate $L_{i-1}$-HN. If none of its prefixes match, it uses the label’s $L_{i-1}$ coordinate to send the packet towards a core that reaches the packet’s destination.

Figure 7 presents a subset of the forwarding tables entries of switches $S_7$, $S_4$, and $S_1$ of Figure 3(c), assuming the $L_1$-coordinate assignments of Figure 3(b) and that $S_1$ has a single host on port 3. Entries for exception cases are omitted.

All forwarding entries are directional, in that a packet can be headed ‘downwards’ to a lower level switch, or ‘upwards’ to a higher level switch. Directionality is determined by the packet’s input port. ALIAS restricts the direction of packet forwarding to ensure loop-free forwarding. The key restriction is that a packet coming into a switch from a higher level switch can only be forwarded downwards, and that a packet moving laterally cannot be forwarded upwards. We refer to this property as up*/across*/down* forwarding, an extension of the up*/down* forwarding introduced in [18].

3.3 End-to-End Communication

ALIAS labels can serve as a basis for a variety of communication techniques. Here we present an implementation based on MAC address rewriting.

When two hosts wish to communicate, the first step is generally ARP resolution to map a destination host’s IP address to a MAC address. In ALIAS, we instead resolve IP addresses to ALIAS labels. This ALIAS label is then written into the destination Ethernet address. All switch forwarding proceeds based on this destination label. Unlike standard Layer 2 forwarding, the destination MAC address is not rewritten hop-by-hop through the network.

Figure 8 depicts the flow of information used to establish end-to-end communication between two hosts. When an $L_1$...
switch discovers a connected host, it assigns to it a set of ALIAS labels. $L_1$ switches maintain a mapping between the IP address, MAC address and ALIAS labels of each connected host. Additionally, they send a mapping of IP address-to-ALIAS labels of connected hosts upwards to all reachable cores. This eliminates the need for a broadcast-based ARP mechanism. Arrows 1-3 in Figure 8 show this mapping as it moves from $L_1$ to the cores.

To support unmodified hosts, ALIAS $L_1$ switches intercept ARP queries (arrow 4), and reply if possible (arrow 9). If not, they send a proxy ARP query to all cores above them in the hierarchy via intermediate switches (arrows 5,6). Cores with the requested mappings reply (arrows 7,8). The querying $L_1$ switch then replies to the host with an ALIAS label (arrow 9) and incorporates the new information into its local map, taking care to ensure proper handling of responses from multiple cores. As a result, the host will use this label as the address in the packet’s Ethernet header. Prior to delivering a data packet, the egress switch rewrites the ALIAS destination MAC address with the actual MAC address of the destination host, using locally available state information encoded in the hardware forwarding table.

Hosts can have multiple ALIAS labels corresponding to multiple sets of paths from cores. However, during ARP resolution, a host expects only one MAC address to be associated with a particular IP address. To address this, the querying host’s neighboring $L_1$ switch chooses one of the ALIAS labels of the destination host. This choice could be made in a number of ways; switches could select randomly or could base their decisions on local views of dynamically changing congestion. In our implementation, we include a measure of each label’s value when passing labels from $L_1$ switches to cores. We base a host label’s value on connectivity between the host $h$ and the core of the network as well as on the number of other hosts that can reach $h$ using this label. To calculate the former, we keep track of the number of paths from the core level to $h$ that are represented by each label. For the latter, we count the number of hosts that each core reaches and weight paths according to these counts. An $L_1$ switch uses these combined values to select a label out of the set returned by a core.

Link additions and failures can result in relabeling. While the routing protocol adapts to changes, existing flows to previously valid ALIAS labels will be affected due to ARP caching in unmodified end hosts. Here, we describe our approach to minimize disruption to existing flows in the face of shifts in topology. We note however that any network environment is subject to some period of convergence following a failure. Our goal is to ensure that ALIAS convergence time at least matches the behavior of currently deployed networks.

Upon a link addition or failure, ALIAS performs appropriate relabeling of switches and hosts (Section 2.3) and propagates the new topology view to all switches as part of standard TVM exchanges. Recall that cores store a mapping of IP addresses-to-ALIAS labels for hosts. Cores compare received mappings to existing state to determine newly invalid mappings. Cores also maintain a cache of recently queried ARP mappings. Using this cache, core switches inform recent $L_1$ requesters that an ARP mapping has changed (arrows 10-11), and $L_1$ switches in turn send gratuitous ARP replies to hosts (arrow 12).

Additionally, ingress $L_1$ switches can preemptively rewrite stale ALIAS labels to maintain connectivity between pairs of hosts during the window of vulnerability when a gratuitous ARP has been sent but not yet received. In the worst case, failure of certain cores may necessitate an ARP cache timeout at hosts before communication can resume.

Recall that ALIAS enables two classes of multi-path support. The first class is tied to the selection of a particular label (and thus a corresponding set of paths) from a host’s label set, whereas the second represents a choice within this set of paths. For this second class of multi-path, ALIAS supports standard multi-path forwarding techniques such as ECMP [7]. Essentially, forwarding entries on the upward path can contain multiple next hops toward the potentially multiple core switches capable of reaching the appropriate top-level coordinate in the destination host label.

3.4 Peer Links
ALIAS considers peer links, links between switches at the same level of the hierarchy, as special cases for forwarding. There are two considerations to keep in mind when introducing peer links into ALIAS: maintaining loop-free forwarding guarantees and retaining ALIAS’s scalability properties. We consider each in turn below.

To motivate our method for accommodating peer links, we first consider the reasons for which a peer link might exist in a given network. A peer link might be added (1) to create direct connectivity between two otherwise disconnected HNs or cores, (2) to create a “shortcut” between two HNs (for instance, HNs with frequent interaction) or (3) unintentionally. ALIAS supports intentional peer links with up*/across*/down* forwarding. In other words, a packet may travel upwards and then may “jump” from one HN to another directly, or may traverse a set of cores, before moving downwards towards its destination.

Switches advertise hosts reachable via peer links as part of outgoing TVMs. While the up* and down* components
of the forwarding path are limited in length by the overall depth of the hierarchy, the across* component can be arbitrarily long. To avoid the introduction of forwarding loops, all peer link advertisements include a hop count.

The number of peer link traversals allowed during the across* component of forwarding represents a tradeoff between routing flexibility and ALIAS convergence. This is due to the fact that links used for communication must also be considered for coordinate assignment, as discussed in Section 2.4. Consider the example of Figure 9. In the figure, dotted lines indicate long chains of links, perhaps involving switches not shown. Since host $H_k$ can reach both other hosts, $H_i$ and $H_j$, switches $S_i$ and $S_j$ need to have unique coordinates. However, they do not share a common parent, and therefore, must cooperate across the long chain of peer links between them to ensure coordinate uniqueness. In fact, if a packet is allowed to cross $p$ peer links during the across* segment of its path, switches as far as $2p$ peer links apart must not share coordinates. This increases convergence time for large values of $p$. Because of this ALIAS allows a network designer to tune the number of peer links allowed per across* segment to limit convergence time while still providing the necessary routing flexibility. Since core switches do not have coordinates, this restriction on the length of the across* component is not necessary at the core level; cores use a standard hop count to avoid forwarding loops.

It is important that peer links are used judiciously, given the particular style of forwarding chosen. For instance, supporting shortest path forwarding may require disabling “shortcut” style peer links when they represent a small percentage of the connections between two HNs. This is to avoid a situation in which all traffic is directed across a peer link (as it provides the shortest path) and the link is overwhelmed.

3.5 Switch Modifications

We engineer ALIAS labels to be encoded into 48 bits to be compatible with existing destination MAC addresses in protocol headers. Our task of assigning globally unique hierarchical labels would be simplified if there were no possibility of collisions in coordinates, for instance if we allowed each coordinate to be 48-bits in length. If we adopted longer ALIAS labels, we would require modified switch hardware that would support an encapsulation header containing the forwarding address. Forwarding tables would need to support matching on pre-selected and variable numbers of bits in encapsulation headers. Many commercial switches already support such functionality in support of emerging Layer 2 protocols such as TRILL [20] and SEATTLE [10].

Our goal of operating with unmodified hosts does require some support from network switching elements. ALIAS $L_1$ switches intercept all ARP packets from hosts. This does not require any hardware modifications, since packets that do not match a flow table entry can always be sent to the switch software and ARP packets need not necessarily be processed at line rate. We further introduce IP address-to-ALIAS label mappings at cores, and IP address, actual MAC address, and ALIAS label mappings at $L_1$. We also maintain a cache of recent ARP queries at cores. All such functionality can be realized in switch software without hardware modifications.

4. IMPLEMENTATION

ALIAS switches maintain the state necessary for level and coordinate assignment as well as local forwarding tables. Switches react to two types of events: timer firings and message receipt. When a switch receives a TVM it updates the necessary local state and forwarding table entries. The next time its TVMsend timer fires, it compiles a TVM for each switch neighbor as well as a ping for all host neighbors. Neighbors of unknown types receive both. Outgoing TVMs include all information related to level and coordinate assignment, and forwarding state, and may include label mappings as they are passed upwards towards cores. The particular TVM created for any neighbor varies both with levels of the sender and the receiver as well as with the identity of the receiver. For instance, in a 3-layer topology, an $L_2$ switch sends the set of its neighboring $L_1$ switches downward for HN grouping by the representative $L_1$ switch. On the other hand, it need not send this information to cores.

Figure 10 shows the basic architecture of ALIAS. We have produced two different implementations of ALIAS, which we describe below.

4.1 Mace Implementation

We first implemented ALIAS in Mace [5, 9]. Mace is a language for distributed system development that we chose for two reasons; the Mace toolkit includes a model checker [8] that can be used to verify correctness, and Mace code compiles into standard C++ code for deployment of the exact code that was model checked.

We verified the correctness of ALIAS by model checking our Mace implementation. This included all protocols discussed in this paper: level assignment, coordinate and label assignment, routing and forwarding, and proxy ARP support with invalidations on relabeling. For a range of topologies with intermittent switch, host, and network failures, we verified (via liveness properties) the convergence of level and coordinate assignment and routing state as well as the correct
operation of label resolution and invalidation. Further, we verified that all pairs of hosts that are connected by the physical topology are eventually able to communicate.

4.2 NetFPGA Testbed Implementation
Using our Mace code as a specification, we implemented ALIAS into an OpenFlow [2] testbed, consisting of port NetFPGA PCI-card switches [12] hosted in 1U dual-core 3.2 GHz Intel Xeon machines with 3GB of RAM. 16 end hosts connect to the 20 4-port switches wired as a 3-level fat tree. All machines run Linux 2.6.18-92.1.18.el5 and switches run OpenFlow v0.8.9r2.

Although OpenFlow is based around a centralized controller model, we wished to remain completely decentralized. To accomplish this, we implemented ALIAS directly in the OpenFlow switch, relying only on OpenFlow’s ability to insert new forwarding rules into a switch’s tables. We also modified the OpenFlow configuration to use a separate controller per switch. These modifications to the OpenFlow software consist of approximately 1,200 lines of C code.

5. EVALUATION
We set out to answer the following questions with our experimental evaluation of ALIAS:

- How scalable is ALIAS in terms of storage requirements and control overhead?
- How effective are hypernodes in compacting forwarding tables?
- How quickly does ALIAS converge on startup and after faults? How many switches relabel after a topology change and how quickly does the new information propagate?

Our experiments run on our NetFPGA testbed, which we augment with miswirings and peer links as necessary. For measurements on topologies larger than our testbed, we rely on simulations.

5.1 Storage Requirements
We first consider the storage requirements of ALIAS. This includes all state used to compute switches’ levels, coordinates, and forwarding tables. For a given number of hosts, \( H \), we determined the number of \( L_1 \), \( L_2 \), and \( L_3 \) switches present in a 3-level, 128-port fat tree-based topology. We then calculated analytically the storage overhead required at each type of switch as a function of the input topology size, as shown in Figure 11. \( L_1 \) switches store the most state, as they may be representative switches for higher level HNs, and therefore must store state to calculate higher level HN coordinates.

We also empirically measured the storage requirements of ALIAS. \( L_1, L_2, \) and \( L_3 \) switches required 122, 52, and 22 bytes of storage, respectively for our 16-node testbed; these results would grow linearly with the number of hosts. Overall, the total required state is well within the range of what is available in commodity switches today. Note that this state need not be accessed on the data path; it can reside in DRAM accessed by the local embedded processor.

5.2 Control Overhead
We next consider the control message overhead of ALIAS. Table 2 shows the contents of TVMs, both for immediate neighbors and for communication with representative \( L_1 \) switches. The table gives the expected size of each field, (where \( S \) and \( C \) are the sizes of a switchID and coordinate), as well as the measured sizes for our testbed implementation (where \( S = 48 \), \( C = 8 \) bits). Since our testbed has 3 levels, TVMs from \( L_2 \) switches to their \( L_1 \) neighbors are combined with those to representative \( L_1 \) switches (and likewise for upward TVMs); our results reflect these combinations. Messages sent downwards to \( L_1 \) switches come from all members of an \( L_1 \) HN and contain per-parent information for each HN member; therefore, these messages are the largest.

![Figure 11: Storage Overhead for 3-level 128-port tree](image)

<table>
<thead>
<tr>
<th>Sender and Receiver</th>
<th>Field</th>
<th>Expected Size</th>
<th>Measured Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-to-All</td>
<td>level</td>
<td>( \log(n) )</td>
<td>2 bits</td>
</tr>
<tr>
<td>To Downward Neighbor</td>
<td>hints</td>
<td>( kC/2 )</td>
<td>( L_3 ) to ( L_2 ): 6B</td>
</tr>
<tr>
<td>To rep. ( L_1 ) switch</td>
<td>per-parent hints</td>
<td>( k^2C/4 )</td>
<td>( L_2 ) to ( L_1 ): 28B</td>
</tr>
<tr>
<td>To Upward Neighbor</td>
<td>coord</td>
<td>( C )</td>
<td>( L_2 ) to ( L_3 ): 5B</td>
</tr>
<tr>
<td>rep. ( L_1 )</td>
<td>HN</td>
<td>( kS )</td>
<td>( L_1 ) to ( L_2 ): 7B</td>
</tr>
<tr>
<td>From Rep. ( L_1 ) switch</td>
<td>per-parent coords</td>
<td>( kC/2 )</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Level and Coordinate Assignment TVM Fields

The TVM period must be at least as large as the time it takes a switch to process \( k \) incoming TVMs, one per port. On our NetFPGA testbed, the worst case processing time for a set of TVMs was 57\( \mu \)s plus an additional 29\( \mu \)s for updating forwarding table entries in OpenFlow in a small configuration. Given this, 100\( \mu \)s is a reasonable setting for TVM cycle at scale. \( L_1 \) switches send \( \frac{k}{2} \) TVMs per cycle while all other switches send \( k \) TVMs. The largest TVM is dominated by \( \frac{k^2S}{b} \) giving a control overhead of \( \frac{k^2S}{b} \frac{b}{m/C} \). For a network with 64-port switches, this is 31.5 kbps or 0.3% of a 10Gbps link, an acceptable cost for a routing/location protocol that scales to hundreds of thousands of ports with 4 levels and \( k = 64 \). This brings out a tradeoff between convergence time and control overhead; a smaller TVM cycle time is certainly possible, but would correspond to a larger amount of control data sent per second. It is also important...
to note that this control overhead is a function only of $k$ and TVM cycle time; it does not increase with link speed.

### 5.3 Compact Forwarding Tables

Next, we assess the effectiveness of hypernodes in compacting forwarding tables. We use our simulator to generate fully provisioned fat tree topologies made up of $k$-port switches. We then remove a percentage of the links at each level of the topology to model less than fully-connected networks. We use the smallest possible coordinate domain that can accommodate the worst-case number of pods for each topology, and allow data packets to cross as many peer links as needed, within the constraints of up*/across*/down* forwarding.

Once the input topology has been generated, we use the simulator to calculate all switches’ levels and HNs, and we select random coordinates for switches based on common upper-level neighbors. Finally, we populate forwarding tables based on the labels corresponding to the selected coordinates and analyze the forwarding table sizes of switches.

Table 3 gives the parameters used to create each input topology along with the total number of servers supported and the average number of number of forwarding table entries per switch. The table provides values for optimized forwarding tables (in which redundant entries are removed and entries for peer links appear only when providing otherwise unavailable connectivity) and unoptimized tables (which include redundant entries for use with techniques such as ECMP).

As the tables shows, even in graphs supporting millions of servers, the number of forwarding entries is dramatically reduced from the entry-per-host requirement of Layer 2 techniques.

As the provisioning of the tree reduces, the number of forwarding entries initially increases. This corresponds to cases in which the tree has become somewhat fragmented from its initial fat tree specification, leading to more HNs and thus more coordinates across the graph. However, as even more links are deleted, forwarding table sizes begin to decrease; for extremely fragmented trees, mutual connectivity between pairs of switches drops, and a switch need not store forwarding entries for unreachable destinations.

### 5.4 Convergence Time

We measured ALIAS’s convergence time on our testbed for both an initial startup period as well as across transient failures. We consider a switch to have converged when it has stabilized all applicable coordinates and HN membership information.

As shown in Figure 12(a), ALIAS takes a maximum of 10 TVM cycles to converge when all switches and hosts are initially booted, even though they are not booted simultaneously. $L_2$ switches converge most quickly since they simply facilitate $L_2$-coordinate uniqueness. $L_1$ switches converge more slowly; the last $L_1$ switch to converge might see the following chain of events: (1) $L_2$ switch $\ell_{2a}$ sends its coordinate to $L_3$ switch $\ell_3$, (2) $\ell_3$ passes a hint about this coordinate to $L_3$ switch $\ell_{3b}$, which (3) forwards the hint to its representative $L_1$ switch, which replies (4) with an assignment for $\ell_{2b}$’s coordinate.

These 4 TVM cycles combine with 5 cycles to propagate level information up and down the 3-level hierarchy, for a total of 9 cycles. The small variation in our results is due to our asynchronous deployment setting. In our implementation, a TVM cycle is 400 $\mu$s, leading to an initial convergence time of 4ms for our small topology. Our cycle time accounts for 57 $\mu$s for TVM processing and 291 $\mu$s for flow table updates in OpenFlow. In general, the TVM period may be set to anything larger than the time required for a switch to process one incoming TVM per port. In practice we would expect significantly longer cycle times in order to minimize control overhead.

We also considered the behavior of ALIAS in response to failures. As discussed in Section 2.3, relabeling is triggered by additions or deletions of links, and its effects depend on the HN membership of the upper level switch on the affected link. Figure 12(b) shows an example of each of the cases from Table 1 along with measured convergence time on our testbed. The examples in the figure are for link addition; we verified the parallel cases for link deletion by reversing the experiments. We measured the time for all HN membership and coordinate information to stabilize at each affected switch. Our results confirm the locality of relabeling effects; only immediate neighbors of the affected $L_2$ switch react, and few require more than the 2 TVM cycles used to recompute HN membership.

## 6. RELATED WORK

ALIAS provides automatic, decentralized, scalable assignment of hierarchical host labels. To the best of our knowledge, this is the first system to address all three of our goals simultaneously.
Our work can trace its lineage back to the original work on spanning trees [15] designed to bridge multiple physical Layer 2 networks. While clearly ground-breaking, spanning trees suffer from scalability challenges and do not support hierarchical labeling. SmartBridge [17] provides shortest path routing among Layer 2 hosts but is still broadcast based and does not support hierarchical host labels. More recently, Rbridges [16] and TRILL [20] suggest running a full-blown routing protocol among Layer 2 switches along with an additional Layer 2 header to protect against forwarding loops.

SEATTLE [10] improves upon aspects of Rbridge’s scalability by distributing the knowledge of host-to-egress switch mapping among a distributed directory service implemented as a one-hop DHT. In general, however, all of these earlier protocols target arbitrary topologies with broadcast-based routing and flat host labels. ALIAS benefits from the underlying assumption that we target hierarchical topologies.

VL2 [6] proposed scaling Layer 2 to mega data centers using end-host modification and addressed load balancing to improve agility in data centers. However, VL2 uses an underlying IP network fabric, which requires subnet and DHCP server configuration, and does not address the requirement for automation.

Most related to ALIAS are PortLand [13] and DAC [4]. PortLand employs a Location Discovery Protocol for host numbering but differs from ALIAS in that it relies on a central fabric manager, assumes a 3-level fat tree topology, and does not support arbitrary miswirings and failures. In addition, LDP makes decisions (e.g. edge switch labeling and pod groupings) based on particular interconnection patterns in fat trees. This limits the approach under heterogeneous conditions (e.g. a network fabric that is not yet fully deployed) and during transitory periods (e.g., when the system first boots). Contrastingly, ALIAS makes decisions solely based on current network conditions. DAC supports arbitrary topologies but is fully centralized. Additionally, DAC requires that an administrator manually input configuration information both initially and prior to any planned changes.

Landmark [21] also automatically configures hierarchy onto a physical topology and relabels as a result of topology changes for ad hoc wireless networks. However, Landmark’s hierarchy levels are defined such that even small topology changes (e.g. a router losing a single neighbor) trigger relabeling. Also, routers maintain forwarding state for distant nodes while ALIAS aggregates such state with hypernodes.

7. CONCLUSION

Current naming and communication protocols for data center networks rely on manual configuration or centralization to provide scalable communication between end hosts. Such manual configuration is costly, time-consuming, and error prone. Centralized approaches introduce the need for an out-of-band control network.

We take advantage of particular characteristics of data center topologies to design and implement ALIAS. We show how to automatically overlay appropriate hierarchy on top of a data center network interconnect such that end hosts can automatically be assigned hierarchical, topologically meaningful labels using only pair-wise communication and with no central components. Our evaluation indicates that ALIAS holds promise for simplifying data center management while simultaneously improving overall scalability.

8. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this Author’s Guide and the .cls and .tex files that it describes.

9. REFERENCES


