This paper was presented as part of the main technical program at IEEE INFOCOM 2011

VIRO: A Scalable, Robust and Namespace Independent Virtual Id ROuting for Future Networks

Sourabh Jain, Yingying Chen, Zhi-Li Zhang, Saurabh Jain
{sourj, yingying, zhzhang, saurabh}@cs.umn.edu
University of Minnesota-Twin Cities

***Abstract*—In this paper we propose VIRO — a novel, *virtual identifier (Id)* routing paradigm for future networks. The objective is three-fold. First, VIRO directly addresses the challenges faced by the traditional layer-2 technologies such as Ethernet, while retaining its simplicity feature. Second, it provides a uniform *convergence* layer that integrates and unifies routing and forwarding performed by the traditional layer-2 and layer-3, as prescribed by the traditional local-area/wide-area network dichotomy. Third and perhaps more importantly, VIRO *decouples routing from addressing*, and thus is *namespace-independent*. The key idea in our design is to introduce a *topology-aware, structured* virtual id (vid) space onto which both physical identifiers as well as higher layer addresses/names are mapped. VIRO completely eliminates *network-wide flooding* in both the *data* and *control* planes, and thus is highly scalable and robust. Furthermore, VIRO effectively localizes failures, and possesses built-in mechanisms for fast rerouting and load-balancing.**

***Index Terms*—Routing, Future Networks, Scalability, Addressing & Namespace management**

I. Introduction

Today's Internet is increasingly strained to meet the demands and requirements of these Internet services and their users, such as scalability to accommodate the increasing number of network components and host devices, high availability, robustness, mobility and security. As the universal "glue" that pieces together various heterogeneous physical networks, the Internet Protocol (IP) suffers certain well-known shortcomings, e.g., the need for careful and extensive network configurations – in particular, the need for *address management and configuration*, inherently *reactive* approaches for handling network failures, relatively poor support for mobility, and so forth. In addition, despite the potential benefits offered by a larger address space, transition from IPv4 to IPv6 has been difficult and slow; among a variety of other factors, the tight coupling of addressing, routing and other network layer functions clearly make such transition not an easy task. While Ethernet is largely *plug-&-play*, as it was originally developed for small, local area networks, this traditional layer-2 technology can hardly meet the scale as well as the demanding efficiency and robustness requirements imposed on today's large, dynamic networks. A flurry of "fixes" [1]–[9] have been proposed to address some of these limitations.

In this paper we propose *VIRO* — a novel and *paradigm-shifting* approach to network routing and forwarding that is not only highly scalable and robust, but also is *namespace-independent*: i) VIRO directly and simultaneously addresses the challenges faced by the traditional layer-2 technologies such as Ethernet–while retaining its "plug-&-play" feature–as well as those associated with the IP networks. ii) VIRO provides a uniform *convergence* layer that integrates and unifies routing and forwarding performed by the traditional layer-2 (data link layer) and layer-3 (network layer), as prescribed by the traditional local-area/wide-area network dichotomy. iii) Perhaps more importantly, VIRO *decouples routing from addressing*, and thus is *namespace-independent*. The key idea behind VIRO is the introduction of a *topology-aware, structured virtual identifier* (*vid* in short) space onto which both physical identifiers (e.g., Ethernet MAC addresses) as well as higher layer addresses/names (e.g., IPv4/IPv6 addresses or flat-id names) are mapped. Built on top of this topology-aware, structured *vid* space, DHT (*Distributed Hash Table*)-style routing and look-up mechanisms are employed for resolving name/address and *vid* mappings as well as for routing and forwarding data purely based on *vid*'s. VIRO is highly *scalable and robust*, with built-in mechanisms for load-balancing, fast rerouting and other key features needed to support future networking and application needs. Moreover, VIRO allows new (global or local) addressing and naming schemes (e.g., a flat-id namespace [10]) to be introduced into networks without the need to modify core router/switch functions, and can easily and flexibly support inter-operability between existing and new address schemes/namespaces.

In a nutshell, VIRO is designed with two broad sets of goals: i) to support – *with minimal manual configuration* – (future) *large, dynamic* networks which connect tens or hundreds of thousands of diverse devices with *rich physical topologies*; and ii) to meet the *high availability, robustness, mobility, manageability and security requirements* of these networks and the services running on top of them. These goals are motivated partly by the rise of huge data centers, emergence of cloud-computing and services, as well as the continued trends in large campus, enterprise and ISP (wired, wireless and cellular data) networks to use 1/10/100 Gigabit Ethernet as the core (layer-2) networking technology. Toward these goals, in this paper we outline an initial basic design. The remainder of the paper is organized as follows. In Section II we provide an overview of VIRO and its three key components, and briefly discuss the related work. These three components and their basic operations are presented in more details in Section III, Section IV and Section V, respectively. Section VI provides the detailed simulation based evaluation of VIRO. The paper is concluded in Section VII.

978-1-4244-9921-2/11/$26.00 ©2011 IEEE

II. Overview and Related Work

In this section we provide an overview of VIRO – in particular, its three key components, *vid space construction and vid assignment*, *VIRO routing*, and *vid lookup and forwarding* – and briefly discuss the related work. A summary of the terminologies and notations used in the paper is included at the end of the section.

A. Design of VIRO: An Overview

The key idea behind VIRO is the introduction of a *topology-aware, structured virtual id (vid) space* onto which physical/application identifiers are mapped, see Fig.1(a) for an illustration. By *topology-aware*, we mean that the physical network topology, as formed by the connections among "routing-nodes" or VIRO switches, is *embedded* into a *structured* space, e.g., a Kademlia-like virtual tree [11], a hypercube, a d-dimensional Euclidean space, in such a manner that *physical proximity among VIRO switches are approximately preserved.* For the physical network topology shown in Fig.1(a), Fig. 1(b) shows such an embedding using a *virtual binary tree*, where only the *leaf* nodes correspond to physical (switching/routing) devices, i.e., routing nodes, whereas all intermediate nodes in the virtual binary tree are logical (thus the term *virtual* binary tree!), representing all the VIRO switches residing within its subtree. Furthermore, since *vid* space is topology aware, the *vid* of a node reflects the *relative location* of the node in the underlying physical topology.

The topology-aware, structured *vid* space is constructed at the network *bootstrapping* phase, i.e., when the network is being set up. Let $\mathcal{L}$ denote the number of bits used to represent the vid space. The *vid* of a node is the $\mathcal{L}$-bit long binary string along the path from the root to the corresponding leaf. The *logical distance* between a pair of vids in this vid space is defined as $\mathcal{L}$ *minus the length of the longest common prefix for the pair*, see Eq.(1). In Sec. III we will describe how the vid space construction and *vid* assignment can be performed in either a centralized or distributed fashion during the bootstrapping phase. Once the *vid* space has been constructed, when a new routing node joins the network, its *vid* will be assigned based on the subtree (and its neighbors in the subtree) that it is attached to. For end-hosts, their *vid*'s are dynamically assigned at the time when they join the network: when an end-host is attached, either via wired or wireless link, to a node in the network, it is assigned an *extended* vid consisting of the $\mathcal{L}$-bit vid of the node plus a randomly assigned l-bit local id. The routing node that an end-host is attached to will be referred to as its *host-node*. Hence the *vid*'s for all end-hosts attached to the same host-node share an $\mathcal{L}$-bit prefix, and thus they are at 0 logical distance from each other.

Taking advantage of the topology-aware, structured vid space, *VIRO* employs a DHT-style routing algorithm to build routing tables at each node so as to maintain network-wide connectivity and perform end-to-end data delivery. In VIRO, routing tables are constructed *piece-meal-wise* using the vid logical distance instead of physical distance (e.g., hop counts). VIRO uses a *publish-&-query* mechanism at each node to publish and query relevant routing information to build routing entries at each level using a round-by-round, bottom-up procedure (see Section IV). As a result, VIRO completely eliminates *network-wide flooding* in both the data plane (unlike Ethernet switching algorithm) and *control plane* (unlike OSPF and other shortest path routing algorithms). Furthermore, because of the natural *hierarchical* structure of the vid space, routing information regarding far-away part of the network is automatically aggregated using the vid prefixes. Hence the routing table size is $O(\log N)$, where N is the number of routing-nodes, as opposed to $O(N)$ (as in the case of OSPF). Unlike OSPF, in VIRO no *network-wide* full topology needs to be maintained by any switch, thanks to the structured vid space, and hence changes in network topology do not need to be flooded globally. Due to the aggregate routing information maintained by switches, failure of a link or switch node can be *localized*, without affecting nodes in far-away parts of the network. Furthermore, topology diversity can be easily exploited in VIRO by using multiple routing entries; hence failure of one routing entry (a nexthop node or link) does not affect network-wide reachability.

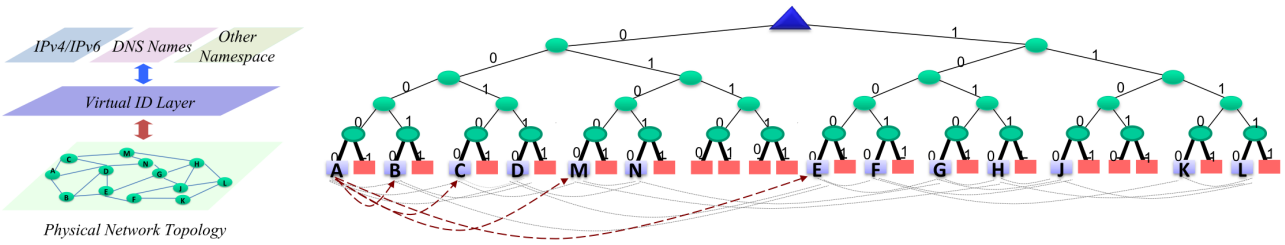
The third major component of VIRO is *vid lookup and forwarding*. VIRO maps both physical addresses and higher-layer addresses/names onto the *vid* space. Routing and forwarding *between VIRO nodes/switches* is performed using *vid*'s; only at the network *edge* are the physical addresses/logical addresses/names needed (between a VIRO switch and an end-host) to locate individual end-hosts, or data/services to be delivered. VIRO performs address/name resolution and *vid* look-up by building the (standard) DHT look-up mechanisms on top of the same *vid* space.

In summary, 1) VIRO is highly scalable, robust; 2) it decouples routing from addressing, and thus is namespace-independent; 3) it provides seamless and efficient support for multi-homing, mobility and access control; 4) VIRO localizes failures, and provides *built-in* mechanisms for fast rerouting and load-balancing; and 5) VIRO can be readily extended to enable multiple (logical) topologies or multiple virtualized networks on top of the same physical network substrate to further enhance network robustness or service isolation.

B. Related Work

Closely related to our work, SEATTLE [2] focuses primarily on addressing the scalability issues of Ethernet. While SEATTLE eliminates data plane flooding, it employs OSPF-like shortest path routing, which requires *network-wide flooding* of link state advertisements (LSAs) in maintaining network topology and tracking its changes. SEATTLE thus suffers the same limitations plaguing OSPF-based IP routing: for example, it is limited to the use of shortest paths; load-balancing and fast rerouting can be messy to implement. In contrast, VIRO avoids these inherent problems in shortest-path routing. It is far more scalable and robust (e.g., with $O(\log N)$ routing table sizes instead of $O(N)$ in OSPF).

Our work is also substantially different from the "flat-id" based routing schemes such as VRR [12], UIP [13] and



(a) VIRO and *vid* layer

(b) *vid* space as a *virtual binary tree*: the grey dotted lines denote physical connectivity, the red dashed lines indicate the logical neighbors at *different* levels of node A; the red boxes represent the *unused vid*'s.

Fig. 1. Overview of VIRO.

ROFL [14], which advocate a *flat* universal id space to replace the current global IP address space. These schemes employ a DHT-style randomly and consistently hashed id assignment–which produces an id-space completely independent of the underlying network topology–and perform routing based on logical distance to the id of the destination, incurring a stretch penalty (which is unbounded in the worst case). In addition, link/node failures and node dynamics (node joining, leaving or moving around in the network) often induce a network-wide effect, as two logically close nodes may be far away in the underlying physical network. By introducing a *topology-aware*, structured vid space, VIRO circumvents these problems: it incurs fairly small routing stretches, and effectively localizes the effect of failures. More importantly, VIRO is *namespace-independent*, allowing any namespace to be used, be it hierarchical or flat, and supporting inter-operability across namespaces.

Furthermore, several works such as PathDCS [15], GHT [16] and NoGeo [17] explored the coordinate based routing schemes, where routing nodes are assigned an identifier based on their physical location. Although using the coordinate-based schemes (eg., NoGeo) for id assignment may be a good idea when the network is georgaphic-dispersed (e.g., a wide-area network) with each node/subnet in one location; however, when we have a densely connected network, e.g., data centers, coordinate-based schemes for id assignment would not be a good idea, as they cannot take physical topology into account.

C. Notations & Definitions

In the following we define and list the key notations and terminologies that will be used to describe VIRO in the remaining sections. (See Table I for a quick summary.)

$vid(x)$	Virtual id (vid) of a node x
$pid(x)$	Physical/persistent address/name or other lower/higher layer identifier of a node x
$d(x,y)$	Shortest physical hop distance between nodes x and y
$\delta(x,y)$	Logical distance between nodes x and y
$vid_k(v_x)$	First k bits(from left) of $vid(x)$
$lcp(v_x,v_y)$	Length of the longest common prefix for v_x and v_y
$S_k(x)$	$\{y : \delta(x,y) \leq k\}$ kth sub-tree of node x
$B_k(x)$	$\{y : \delta(x,y) = k\}$ kth bucket of node x
$rdv_k(x)$	kth level Rendezvous points for node x
$R_k(x)$	Reachability information for node x's bucket $B_k(x)$
$hash_k(val)$	Hash function to get k bit hash value for val

TABLE I
SUMMARY OF NOTATIONS

Logical Distance $\delta(x,y)$: The logical distance between any two nodes say x and y in an $\mathcal{L}$-bit vid space is defined as:

$$\delta(x,y) = \mathcal{L} - lcp(vid(x), vid(y)). \quad (1)$$

Here, $vid(x)$ and $vid(y)$ are the virtual ids for the nodes x and y. $lcp(vid(x), vid(y))$ is the length of the longest common prefix for binary strings $vid(x)$ and $vid(y)$. e.g. if $vid(x) = 0011$, $vid(y) = 0101$, and $\mathcal{L} = 4$ then $\delta(x,y) = 4 - lcp(vid(x), vid(y)) = 4 - 1 = 3$

Bucket $B_k(x)$: It is the set of nodes which are at logical distance of k from node x.

Sub-tree $S_k(x)$: It is the set of nodes which are at no more than logical distance of k from node x.

Rendezvous Point ($rdv_k(x)$): For a node x, a *rendezvous point* at a level k, $rdv_k(x)$, is a node in the sub-tree $S_{k-1}(x)$, which stores the connectivity information to reach its k-th bucket $B_k(x)$. It is the node which is closest (based on the xor distance) to the vid given by $vid_{\mathcal{L}-r}(x)hash_r(vid_{\mathcal{L}-r}(x))$ for $r = k-1$ in the vid space.

Gateway: The *gateway* for a node x to reach Bucket $B_k(x)$ is a node $y \in S_{k-1}(x)$ such that it has a (physical) edge to a node $z \in B_k(x)$.

pid: We use pid to denote either the physical address (e.g., MAC address), IPv4/IPv6 addresses, persistent name (e.g., a flat-id name) or other addresses/names that are used by either lower layer or higher layer to address, name or identify a given entity (an end-host, information or service of interest, etc).

Host-Node: A *host-node* for an end-host is the node in the network that it is directly connected to.

Access-Node: An *access-node* for an end-host is the node which stores the mapping $pid \Rightarrow vid$. An access-node for a given pid is determined using the $vid = hash_L(pid)$, it is the node closest (based on the xor distance) to the vid given by the $hash$ value of the pid.

Reachability Information ($R_k(x)$): It is a 4-tuple set, which contains following information about the reachability to a given bucket $B_k(x)$ for node x. It consists of following values: a) Bucket level k, b) vid prefix in $B_k(x)$ that is reachable using this entry. c) *Nexthop* to reach any node in this bucket. d) Logically closest gateway to reach this $B_k(x)$ prefix.

III. Virtual Id Assignment

The initial *vid* space construction and *vid* assignment is performed at the network bootstrapping process when the network is being set up. As mentioned earlier, the *vid* space is structured in a virtual binary tree (with a depth of $\mathcal{L}$), where

each node (VIRO switch) resides at a leaf node of the tree, and is assigned an $\mathcal{L}$-bit vid corresponding to the bit-string from the root to this leaf node. After the network is set up, the *vid* assignment for a new node that subsequently joins the network is done based on its location and the *vid*'s of its physical neighbors, and the (extended) *vid* for an end-host that joins the network is assigned by its host-node (to which it is attached). We describe these operations in more detail below.

A. vid Space Construction at Bootstrap

During the vid construction (and subsequent addition/deletion of nodes), two key *invariant* properties are always maintained: i) The *closeness* property: if two nodes are close in the vid space, then they are also close in the physical topology; in particular, if $\delta(x, y) = 1$ i.e., nodes x and y are *logical* neighbors, then they must be physically directly connected. ii) The *connectivity* property: any two *logically adjacent* (i.e., sharing a common prefix) and *non-empty* sub-trees must be physically connected, i.e., there must be at least one physical (wired/wireless) link connecting one node in a sub-tree to another node in the other sub-tree. We have designed two modes of vid space construction and vid assignment for bootstrapping a VIRO network: a *centralized* algorithm and a *distributed* algorithm. Both algorithms guarantee that the constructed *vid* space satisfies these two properties.

The centralized mode is designed for networking environments (e.g., ISP, large campus/enterprise or data center networks) where the (at least the initial) topology is *pre-planned* and thus known *a priori*. Given the topology, the centralized vid assignment algorithm employs a *top-down* approach to assign vid's by starting from the root of the virtual binary tree: it recursively partitions the network topology into two subgraphs, and appends 1-bit to the (already assigned) vid prefixes of the nodes in each subgraph. Due to space limitation, we omit the detailed description of the algorithm here. (see [18]).

The distributed mode is more suitable for networking environments (e.g., home or small office networks, wireless ad hoc networks) where networks are set up in a *piecemeal*, *unplanned* or *ad hoc* fashion. The distributed *vid* assignment algorithm employs a *bottom-up* approach to assign *vid*'s by starting from the leaf nodes (namely, the lower-level *vid* bits (a suffix) are determined first, and higher-level bits are recursively assigned). As illustrated in Fig. 2, VIRO nodes first discover their physical neighbors (e.g., via an OSPF-like adjacency discover protocol, or *local* broadcast), and collaboratively run a recursive clustering algorithm (similar to those used in wireless ad hoc networks, see, e.g., [19]) to construct a virtual binary tree.

B. vid Assignment upon New Node Join

After the network is set up (and the initial *vid* assignment completed), when a new VIRO node (switch) joins the network, its *vid* is assigned based on its location and the *vid*'s of its physical neighbors. More specifically, it picks one of the *unused vid*'s that are closest to one of its physical neighbors,

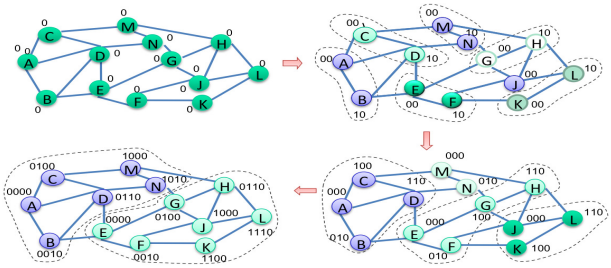


Fig. 2. Virtual ID assignment process using bottom-up clustering method. In this example, nodes are initially assigned a *vid* 0, before the bootstrap process. In the subsequent steps, nodes are clustered with their neighboring nodes and '0' or '1' bit is prepended to their *vid*s

thus joining a subtree (of an appropriate level) of this neighbor. Using Fig. 2 as an example, suppose a new node joins the network and is connected to both node A and node B. It can pick one of the two unused vid's: $\{00001, 00011\}$, joining either the level-1 subtree of node A or node B (see Fig. 1(b)). We assume that in general $\mathcal{L}$ is configured in such a manner that there will be sufficient empty slots under each subtree to accommodate new node joins. In the (extremely rare) case where $\mathcal{L}$ is ill-chosen initially, one could easily expand $\mathcal{L}$ by appending additional bits. In our design, we reserve a 128-bit header space for $\mathcal{L}$, while allowing $\mathcal{L}$ to be configured with a smaller value (i.e., only the first $\mathcal{L}$ bits are used). This provides the advantage of using a smaller $\mathcal{L}$ (thus fewer routing table entries per node) as well as the flexibility to expand $\mathcal{L}$ if needed.

C. Host vid Assignment

In VIRO, the *vid* for end-hosts comprises two parts (see Fig. 3): the $\mathcal{L}$-bit *host-node vid* part identifies the host-node (VIRO switch) that an end-host is directly connected to, i.e., the *vid* of its host-node; the second l-bit *host vid* part identifies the end-host, and distinguishes it from other end-hosts attached to the same host-node. This l-bit host *vid* part is selected randomly by the host-node, e.g., via a (l-bit) random hash function, $hash_l(pid(h))$, using a $pid(h)$ (e.g., MAC or IP address) of the end-host, to ensure local uniqueness. The $(\mathcal{L} + l)$-bit (expanded) *vid* thus uniquely identifies each end-host in the network. When an end-host moves from one part of the network to another (thus attaches itself to a different VIRO node), its *vid* is also re-assigned. The host-node periodically publishes and maintains pid-vid mappings of end-hosts attached to it, see Section V for details.

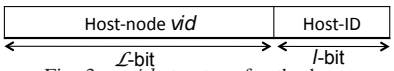


Fig. 3. vid structure for the hosts.

IV. VIRO Routing

Routing in VIRO is inspired by Kademlia-like [11] DHTs. The key difference is that DHTs assume end-to-end connectivity, and utilize the underlying IP-layer for routing and look-up operations, whereas VIRO must build end-to-end connectivity by itself, using the (local) physical connectivity among VIRO nodes. In the following we describe how this is done in VIRO, and once the routing tables are constructed, how forwarding is performed. We will also briefly discuss how failures are handled by VIRO.

A. Overview and the Routing Invariant Property

Similar to Kademlia, each node in VIRO maintains a routing table with (at most) $\mathcal{L}$ entries, one per level of the (virtual binary) tree. Given an arbitrary (VIRO) node x in the network, all other nodes in the network fall within one of $\mathcal{L}$ buckets, $B_k(x)$, $1 \leq k \leq \mathcal{L}$. For some k, $B_k(x)$ may be empty, i.e., there are no VIRO switches residing at the corresponding leaf nodes. The VIRO routing tables are constructed in such a manner that for any non-empty $B_k(x)$, $1 \leq k \leq \mathcal{L}$, as long as node x knows how to reach a node within $B_k(x)$, say, $y \in B_k(x)$, then x can reach (e.g., via y) *any* node within $B_k(x)$. Hence each node x in VIRO only needs to maintain $\mathcal{L}$ routing entries.

Take the network in Fig. 1(b) as an example. Representing each level of routing entries by the *vid* prefix (of nodes within the corresponding bucket), the routing table at node A (with $vid_A = 00000$) contains 5 routing entries: the level-1 ($\{00001\}$) routing entry is empty (no node in $B_1(A)$); the level-2 ($\{0001*\}$) routing entry would contain B, the only node in $B_2(A)$; the level-3 ($\{001**\}$) routing entry would contain either C or D or both ; the level-4 ($\{01***\}$) routing entry would contain either M or N, or both; and the level-5 ($\{1****\}$) routing entry would contain (at least) one of the 7 nodes in $B_5(A)$ (i.e. E, F, G, H, J, K, L).

Unlike standard DHTs where end-to-end connectivity is assumed, VIRO must build up end-to-end connectivity by itself. This is done through a *bottom-up* procedure, where for $k = 2, \ldots, \mathcal{L}$, the first $k-1$ routing entries are constructed before the level-k routing entry is built. If there is a node, say, y, resides within $B_1(x)$, then by the *closeness* property of the *vid* space, node x and node y must be directly connected (as $\delta(x, y) = 1$). Hence the level-1 can be trivially built. More generally, if $B_k(x)$ is not empty, then by the *connectivity* property there must exist another node, say y, within $S_{k-1}(x)$ that is physically connected to a node, say z, in $B_k(x)$. Hence as long as x can reach y, then via y it can reach $z \in B_k(x)$, thus any node in $B_k(x)$. Node y is thus referred to as a (level-k) *gateway* to $B_k(x)$. Since $y \in S_{k-1}(x)$ (thus $\delta(x, y) \leq k-1$), y is contained in one of $k-1$ buckets of x, i.e., $y \in B_{k'}(x), k' < k$. In other words, once the first $k-1$ routing entries have been constructed, the gateway node y to $B_k(x)$ can be reached using these first $k-1$ routing entries. Using y, we can build the level-k routing entry to reach any node in $B_k(x)$. This leads us to the following *invariant* property, referred as the *routing Invariant Property* (which is essentially a re-statement of the *connectivity* property of the *vid* space), that must be satisfied by any VIRO routing table construction algorithm. In the next subsection we present one such algorithm for constructing VIRO routing tables.

Routing Invariant Property. Let V be the set of all VIRO nodes, and E the set of all *physical* links between VIRO nodes. Suppose the following *connectivity* condition holds for any $x \in V$ and non-empty $B_k(x)$, where $1 \leq k \leq \mathcal{L}$,

$$\exists z \in B_k(x) \wedge \exists y \in S_{k-1}(x) \text{ such that } (y, z) \in E. \quad (2)$$

Then using y as a level-k gateway, it would guarantee the connectivity for node x to reach any node in $B_k(x)$.

B. Routing Table Construction Algorithm

VIRO employs a *bottom-up, round-by-round* procedure, starting with round $k = 1$ to $\mathcal{L}$, to build routing tables that satisfy the above invariant property. At each round k (for building level-k routing entry), each node x needs to discover a level-k gateway that satisfies Eq.(2) so as to reach nodes in $B_k(x)$. We employ a *publish-&-query* based method for this discovery process, thereby completely eliminating *network-wide (control plane) flooding*. Before this round-by-round procedure is invoked, each node x first runs a *local* physical neighbor discovery protocol (e.g., via HELO messages, or local broadcast) to discover its (physically) directly connected neighbors. (Note that these (physical) neighbors may reside in any $B_k(x)$, $1 \leq k \leq \mathcal{L}$.) At the first round ($k = 1$), if $B_1(x)$ is not empty, then any node $y \in B_1(x)$ must be a direct (physical) neighbor of node x. Hence the level-1 routing entry is constructed trivially based on node x's locally discovered physical neighbors.

More generally, suppose the first k routing entries have already been constructed at node x. In round $k+1$, if node x is directly connected to node $z \in B_{k+1}(x)$ (as discovered during the local physical neighbor discovery process), node x is then a level-$(k+1)$ gateway (for nodes within $S_k(x)$). Besides installing itself as a gateway in its own level-$(k+1)$ routing entry, it declares (to other nodes within $S_k(x)$) that it is a level-$(k+1)$ gateway by *publishing* this information (i.e., $\langle$ level-$(k+1)$ gateway, $vid(x)\rangle$) to the level-$(k+1)$ *rendezvous point(s)* within $S_k(x)$. Depending on the level k and robustness requirement, one or multiple rendezvous points (say, k rendezvous points per level k) may be used, and they are selected based on certain *consistent* rules. We use $rdv_{k+1}(x)$ to denote a level-$(k+1)$ rendezvous point which is responsible for maintaining the level-$(k+1)$ gateway information. If node x is *not* directly connected to a node in $B_{k+1}(x)$, it discovers and learns about a level-$(k+1)$ gateway by sending a query to one of the rendezvous points, $rdv_{k+1}(x)$. The rendezvous point then replies with one of the (published) level-$(k+1)$ gateways. Node x thus constructs its level-$(k+1)$ routing entry, and moves on to the next round until all $\mathcal{L}$ routing entries have been built. Using a level-k gateway, each node x can recursively look up its routing table (using first $(k-1)$ routing entries only) and discover the *next-hop* (a directly connected neighbor) to reach the gateway. The basic algorithm is described in pseudo-code in Algorithm 1. We note that the local physical neighbor discovery process as well as the publish-query-based routing process are performed *periodically* by each node. In other words, each node periodically updates its routing entries by periodically publishing and querying gateway information.

Again using the network in Fig. 1(b) as an example, in the following we illustrate how routing tables at node A can be constructed using Alg. 1. First, during the local physical neighbor discovery process, node A discovers its three phys-

Algorithm 1 Constructing routing tables for node i

```
1:  S_0(i) := i, B_0(i) := i;
2:  Input: N_1(i) ← Physical neighbors for node i
3:  Output: Routing table for the node i
4:  for k = 1 to ℒ do
5:    if x in N_1(i) and δ(i,x) = k then
6:      R_k(i) := (BucketDistance = k, Prefix = pfx_{ℒ−k}(i),
        NextHop = x, Gateway = i)
7:      Publish (rdv_k(i), edge(x ↔ i))
8:    else
9:      gw_k := Query (rdv_k(i), k,i)
10:     if gw_k is not Nil then
11:       d := δ(gw_k, i)
12:       nexthop_k := R_d(i).nexthop
13:       R_k(i) := (BucketDistance = k, Prefix = pfx_{ℒ−k}(i),
          NextHop = nexthop_k, Gateway = gw_k)
14:     end if
15:   end if
16: end for
```

ical neighbors, $B \in B_2(A)$, $C \in B_3(A)$ and $D \in B_3(A)$. Using the information about its local physical neighbors, in round 1, A constructs a *null* level-1 routing entry. In round 2 and round 3, A constructs its level-2 and level-3 routing entries by entering itself as the gateway, and also publishes itself as the level-2 and level-3 gateways (to reach $B_2(A)$ and $B_3(A)$ respectively). To build its level-4 routing entry, A queries a level-4 rendezvous point and discovers a level-4 gateway, say, node C (which is connected to node $M \in B_4(A)$). It installs C as its level-4 gateway (which is also the next-hop to reach C itself). Similarly in round 5, A queries and discovers a level-5 gateway, node B (which is connected to node $E \in B_5(A)$), and installs it in its level-5 routing entry. We show the final routing-table for node A in Table II.

Bucket	Prefix	NextHop	Gateway
1	00001	-	-
2	0001*	B	A
3	001**	C,D	A
4	01***	C	C
5	1****	B	B

TABLE II
ROUTING TABLE FOR NODE A SHOWN IN FIG. 2

Last but not the least, we note that in order to ensure no routing loop is formed during the construction of routing tables, the *gateway selection* process must be *consistent*. In other words, when a node x queries a level-k rendezvous point $rdv_k(x)$ to discover a level-k gateway, the rendezvous point $rdv_k(x)$ – which may have learned multiple gateways, say, four level-5 gateways, B, D, M, and N, to reach $B_5(A)$ – cannot select an *arbitrary* gateway and return it to node x (see [18] for an example where such a strategy may cause a routing loop). Assuming that only one (default) gateway is installed in the routing table of each node, a simple consistent strategy is to always select one of those gateways whose vid is closest to the vid of the querying node. Under this selection rule, in the case of node A, node B will be selected as the level-5 gateway for node A to reach $B_5(A)$. More generally, when multiple gateways are installed in the routing tables, and used, say, for load-balancing or fast rerouting, a generalized consistent gateway selection rule is devised by associating a special *forwarding directive*[1] with each level-k gateway. When this (level-k) gateway is selected to reach $B_k(x)$, the associated forwarding directive is also included in the packet header to direct subsequent packet forwarding toward this gateway. Due to space limitation, we do not provide the details here. In [18] we formally prove that *when either the simple or the generalized consistent gateway selection rule is used, loop-free routing/forwarding is guaranteed.*

C. Forwarding Algorithm

Given the routing tables constructed above, forwarding using vid is fairly straightforward. Consider a node x which wants to send a (*normal data*) packet to a node with $vid = dest$. Node x forwards the packet directly to $dest$ if it is one of its physical neighbors. Otherwise, it computes the logical distance $k = \delta(x, dest)$, and sends the packet to the next-hop as indicated in its level-k routing entry. (If its level-k routing entry is null, it implies that $B_k(x)$ is empty, hence node x simply drops the packet). Upon receiving this packet, the next-hop performs the same operation to determine its next-hop for forwarding the packet. When the packet is a control packet, e.g., *publish*, *query* packets, or pid-vid mapping registration/lookup packets (see the next section), a similar process is used: the key difference here is that the destination *vid* does not correspond to a physical node, instead it is a *key* that is meant to identify the VIRO node whose vid is *closest* to this key. In this case, for node x if its level-k routing entry is empty, where $k = \delta(x, dest)$, it does *not* drop the packet. Instead it flips the $(\mathcal{L}-k)$th bit (counting from the left) in the destination vid, and uses this *updated vid* to look up the routing table, to find a valid nexthop to reach the node closest to vid. If the level-$(k-1)$ routing entry is also empty, it flips the $(\mathcal{L}-(k-1))$th bit in the (updated) destination vid, and looks up its level-$(k-2)$ routing entry. This process stops either when a nexthop, or node x discovers that it is the closest node to this destination vid.

D. Handling Node/Link Failures

VIRO handles node/link failures, without resorting to flooding of failure notifications (as used in OSPF). Instead, it utilizes a *withdraw & update* mechanism: Upon discovering the failure, a node adjacent to a failed node (say, a gateway node) or a failed link (to a gateway node) notifies the appropriate rendezvous point(s) by withdrawing its previously published connectivity information. When a rendezvous point receives this withdraw notification, it notifies all (or a subset of) nodes in an affected sub-tree, namely, those that are currently using the failed or no longer reachable gateway in their routing tables. More specifically, the rendezvous point sends an *update* message containing the *withdrawal* of the current gateway, replacing it with a new gateway. Hence only these nodes that are affected by the failures need to update their routing entries. (When multiple gateways are used, fast

[1]The forwarding directive is a $\mathcal{L}$-bit key, whose first $\mathcal{L}-k$ bits are the same as those in the vid of the querying node, is associated with the level-k gateway whose *vid* is closest to this key

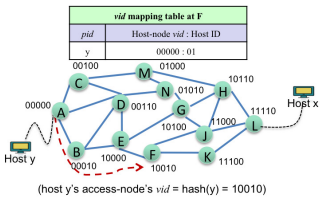


Fig. 4. vid publish process.

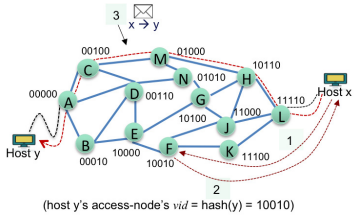


Fig. 5. Steps in packet forwarding.

rerouting can be invoked at the node detecting the failure, see [18]) Hence under VIRO, failures are *localized*, as only those nodes within the same subtree of the failed node/link are likely to be affected. Nodes outside this subtree are in general not affected, thus no routing table updates are needed at all. When a rendezvous node fails, a neighboring node would then take over and serve as the new rendezvous node. The gateway information would be either (partially) recovered from the current gateway(s) stored in its routing table, or from another rendezvous point (when multiple rendezvous points are used), or in the worst case, learned later through periodic publications by gateways. In practice, we assume that for large networks, multiple rendezvous points will be used for enhanced robustness.

V. Virtual ID Lookup and Forwarding

In VIRO, address/name resolution (i.e., pid-vid mappings) is implemented on top of the *vid* space using Kademlia-style DHTs. Once the vid of a host is looked up using its pid, packet forwarding between VIRO nodes is done solely based on vid. For more flexible and better support for mobility, *geographically-scoped hash* functions [20] may also be used for *vid* lookup and address/name resolution. Due to space limitation, in the following we describe only the *basic vid* lookup and forwarding operations in VIRO.

When an end-host h is attached to a VIRO switch (which becomes the host's *host-node*), in addition to assigning a *vid* to the end-host, the host-node also *publishes* pid-vid mappings (*key-value* pairs), e.g., MAC address, IP address, or a flat-id name to *vid* mappings. Let $\langle pid(h), vid(h)\rangle$ denote such a mapping. Using $pid(h)$ as the key, the mapping is stored at the node – referred to as the *access-node* of pid for the host h – whose vid is closest to $hash_{\mathcal{L}}(pid(h))$ based on the XOR distance, as in [11]. The host-node also stores these pid-vid mappings in its local cache. As long as the end-host is attached to its host-node, the host-node will periodically publish its pid-vid mappings. Fig. 4 illustrates the vid publish mechanism: when a host (y) joins the network by connecting to node A, the host-node A for y publishes the mapping at the access node, which is F in the example.

When another node wants to look up the $vid(h)$ for a host h identified by its $pid(h)$ in some namespace (e.g., the MAC address, IPv4 address or flat-id name of host h), it uses $pid(h)$ as the key, or more precisely, the hashed key, $vid = hash_{\mathcal{L}}(pid(h))$, and queries the network. The corresponding access-node (whose *vid* is closest to $hash_{\mathcal{L}}(pid(h))$ then responds with the stored $vid(h)$, if host h exists in the network; otherwise, an error message is returned.

We illustrate the packet forwarding mechanism in Fig. 5, which consists of two steps: a) *Host vid lookup:* when a host x wants to send a packet to a destination host y, it first performs the vid resolution for host y, as described earlier. Namely, host x sends a pid-vid mapping request to its host-node (node L), which forwards it to the corresponding access-node for host y, namely, node F whose vid is closest to $hash(pid(y))$ (step 1 in Fig. 5); upon receiving this request, node F looks up its mapping table, and returns y's vid (00000:01) to x (step 2). b) Packet forwarding using vid: host x includes y's vid as the destination vid in the packet, sends it to its host-node L. Using y's vid, the packet is then forwarded from node L to node A (y's host-node), using the VIRO routing algorithm. Node A then delivers it to host y (step 3).

VI. Evaluation

Through extensive simulations, in this section we evaluate VIRO using various real and synthetic network topologies. We also compare VIRO with several existing routing protocols such as OSPF [21], SEATTLE [2], and VL2 [22]. Since, OSPF, SEATTLE and VL2 use link-state routing protocol [23] to construct the forwarding tables for the nodes. Therefore, in the following when comparing the control overheads of VIRO with these protocols, the link-state routing algorithm (OSPF) is used as the representative example.(More in [18])

We have developed our customized in-house simulator for VIRO so as to extensively simulate VIRO on large network topologies using the available computing resources. The following topologies are used in our evaluation, which is also summarized in Table III.

Router Level AS topologies: These are the router level AS topologies collected by RocketFuel [24]. We provide simulations results for following three AS's topologies: i) (AS1) AS 1755, ii) (AS3) AS 3967 and iii) (AS6) AS 6461.

Data Center Topologies: We generated a large number of data-center topologies using the Fat-Tree [25] based designs to evaluate VIRO. Here we provide results for following three topologies: i) DC1, ii) DC3, iii) DC5. Each of these topologies were created using 125, 320 and 500 commodity switches respectively, which were arranged to form 3 layers: i) ToR (top of rack switches), ii) Aggregation switches and iii) Core switches. Hence, maximum shortest distance between any two switches(nodes) in these topologies is 4 hops.

Synthetic Router Level AS topologies: We used Brite [26] topology generator to generate the router level AS topologies containing different number of nodes. See Table III.

A. vid-assignment evaluation

We evaluate the efficacy of the vid assignment in generating a topology-aware vid space using both the distributed and centralized vid assignment methods. For this evaluation we considered all the topologies mentioned earlier. However, due to space limitations we present results for only one topology in each category i.e. BT600, DC500 and AS3967. We plot average physical (minhop) distance for each pair of nodes

Router Level AS Topologies	AS1(295 nodes, 543 edges)	AS3(353 nodes, 820 edges)	AS6(654 nodes, 1332 edges)
Data Center Topologies	DC1(125 nodes, 500 edges)	DC3(320 nodes, 2048 edges)	DC5(500 nodes, 4000 edges)
Synthetic topologies using BRITE	BT2(200 nodes, 790 edges)	BT4(400 nodes, 1590 edges)	BT6(600 nodes, 2390 edges)

TABLE III
SUMMARY OF THE TOPOLOGIES USED IN EVALUATION.

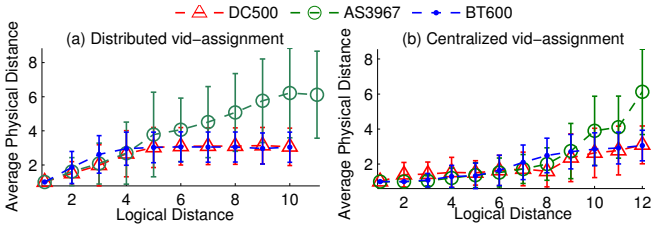


Fig. 6. Distribution of *physical*(MinHop) distance with the *logical* distance. (Vertical bars at each data point represent the 95% confidence interval).

at different logical distances in Fig. 6. In this figure, y-axis shows the average physical distance for all the pairs of nodes, at a given logical distance shown on the x-axis. As seen in this figure, the average physical distance for the pairs of nodes increases with the logical distance for both distributed and centralized *vid* assignments. Therefore, the pairs of nodes which are logically closer to each other are likely to be physically close too. These results show that both the centralized and distributed *vid* assignments embed physical topology and hence provide *topology-aware vid*-layer.

B. Routing Overheads

Routing Stretch: VIRO does not use shortest path routing, therefore it incurs a marginal overhead in terms of the routing optimality. We measure this overhead using *routing stretch*. We define routing stretch as the ratio of the length of the path taken using VIRO and the shortest path length between a source and destination pair. Fig. 7 shows that average routing stretch remains close to 1 for most of the topologies. Furthermore, centralized *vid* assignment incurs much smaller stretch than distributed *vid* assignment, it is because an optimal *vid* assignment is achieved using graph-partitioning algorithms.

Routing Table Size: A key metric for evaluating the scalability of routing protocol is the size of routing table at each node. As seen in Fig. 8, VIRO creates much smaller routing tables than link-state routing protocol. This is because VIRO stores only one routing entry for each logical distance, on the other hand link-state routing treats every node equally and has to keep routing entry for each node in the network.

Since OSPF and SEATTLE use link-state routing protocol to create similar routing tables, these protocols also keep similar routing tables, which grows linearly with the number of nodes in the network.

Control Overhead: Another metric that we use to evaluate the scalability of the routing protocol is the control overhead. We estimate the control-overhead for a node by counting the number of control-messages processed by that node to build the complete routing table. In case of VIRO this control-overhead is created by the control-messages corresponding to *Rendezvous publish* and *query* packets. In this evaluation we also consider four different variants of VIRO by allowing more than one rendezvous node at different level. Here VIRO-1, VIRO-2, VIRO-4 are different variants of VIRO with maximum of 1, 2 and 4 rendezvous nodes at each level respectively. We compare the overhead due to control-messages used by VIRO and link-state in Fig. 9. In this figure, x-axis represents the different topologies and y-axis(plotted on log-scale) shows the average number of control-messages processed by each node in the network. As seen in this figure, control-overhead is much larger for the OSPF than VIRO. It is because of the "flooding" based mechanism used by the link-state routing protocol. In case of VIRO it is much lesser due to the *publish-&-query* mechanism used by it. These results show that VIRO has better scalability than the link-state based protocols in terms of routing- table size and control overhead.

C. Failure Dynamics

In this section we compare the effect of failure dynamics for VIRO and SEATTLE by simulating random node failures. A recent study [22] has shown that 50% of the network device failures in a data-center are caused by the failures of less than 4 network equipments and 95% device failures are due to the failure of less than 20 network equipments. This shows that most of the network failures are caused by the failure of a very small number of devices. Therefore, we simulate a large number of failure scenarios by randomly removing small number of nodes from the topology.

Failure Control-Overhead: As seen in Fig. 10(a), our no flooding based mechanism used in VIRO helps in reducing the number of failure notification messages drastically, while overhead for OSPF style routing protocols is much larger.

Next we evaluate the effect of rendezvous node failures for VIRO. Here, we consider the scenario with only one rendezvous node per level. It is because the failure of a rendezvous node in case of multiple rendezvous nodes does not create any overhead during failures, as nodes can easily switch to other replicas of the rendezvous nodes in the event of failures. Fig. 10(b) plots the overhead due to the failure of the rendezvous nodes at different levels. In this figure y-axis shows the control overhead on each node in the same sub-tree as failed rendezvous node level, which is shown on the x-axis. This figure shows that control-overhead to spread the failure notifications increases with the level of rendezvous node. However it stays very small for even higher levels, e.g. it is only 6 control messages per node for the failure of the rendezvous node at level 14.

We measure the effectiveness of VIRO to localize the effect of failures by comparing the control overhead on the nodes with the logical distance from the failed node (see Fig. 10(c)). In this figure y-axis shows the control overhead on a node

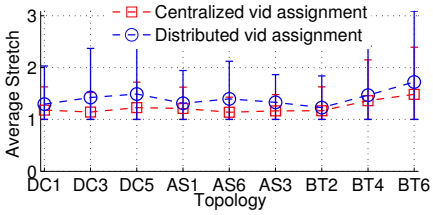


Fig. 7. Routing stretch distribution for VIRO.

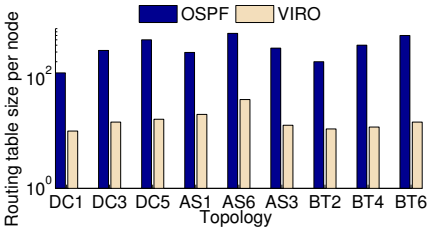


Fig. 8. Routing Table size comparison.

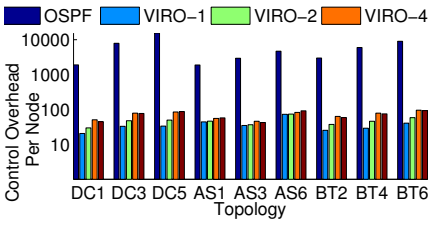
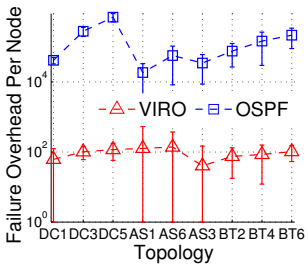
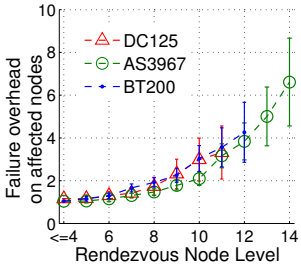


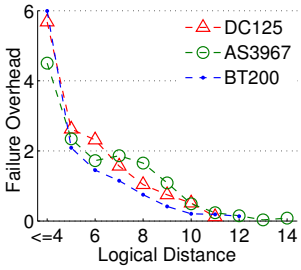
Fig. 9. Control-overhead comparison.



(a) Control overhead



(b) RDV Failure



(c) Failure Localization

Fig. 10. Comparison of VIRO and Link-state for failures. (vertical bars show the 95% confidence interval for the mean values)

with respect to logical distance from the failed node, which is shown on the x-axis. It shows that nodes which are logically far from the failure are less affected by the failures. On the other hand failures are more likely to affect the nodes which are close to it. Therefore, VIRO is very effective in localizing the effect of failures.

VII. Conclusion & Future Work

In this paper we have presented *VIRO*—a novel routing architecture for large-scale networks. The key idea in our design is to introduce a *topology-aware, structured* virtual id (*vid*) space onto which both physical identifiers as well as higher layer addresses/names are mapped. VIRO completely eliminates *network-wide flooding* in both the *data* and *control* planes, and thus is highly scalable and robust. Furthermore, because of the structured vid space, VIRO effectively localizes the effect of failures, performs fast rerouting and support multiple (logical) topologies on top of the same physical network substrate to further enhance network robustness. VIRO also facilitates the support for virtualized networks and network services, as well as enables access control and isolation of services for security and performance. Our evaluation of VIRO using many synthetic and real topologies shows the immense scalability and robustness of VIRO, while keeping the overheads very low. As part of the on-going work, we are developing a prototype of VIRO using OpenFlow and Click modular router.

Acknowledgements

We gratefully acknowledge the support of our sponsors. The work is supported in part by the NSF grants CNS-0905037, CNS-1017647 and CNS-1017092 and the DTRA Grant HDTRA1-09-1-0050.

References

[1] "Ietf trill working group," http://www.ietf.org/html.charters/trill-charter.html.
[2] C. Kim, M. Caesar, and J. Rexford, "Floodless in seattle: a scalable ethernet architecture for large enterprises," *SIGCOMM*, 2008.
[3] R. Perlman *et al.*, "Rbridges: transparent routing," in *INFOCOM*, 2004.
[4] A. Myers, E. Ng, and H. Zhang, "Rethinking the service model: Scaling Ethernet to a million nodes," in *HotNets*, 2004.
[5] S. Sharma *et al.*, "Viking: A multi-spanning-tree Ethernet architecture for metropolitan area and cluster networks," in *INFOCOM*, 2004.
[6] T. L. Rodeheffer, C. A. Thekkath, and D. C. Anderson, "Smartbridge: a scalable bridge architecture," in *SIGCOMM*, 2000.
[7] C. Kim and J. Rexford, "Revisiting Ethernet: Plug-and-play made scalable and efficient," in *15th IEEE Workshop on Local & Metropolitan Area Networks*, 2007.
[8] S. Ray *et al.*, "A Distributed Hash Table based Address Resolution Scheme for Large-Scale Ethernet Networks," in *ICC*, 2007.
[9] C. Alaettinoglu and A. Shankar, "Viewserver Hierarchy: A New Inter-Domain Routing Protocol and its Evaluation," *INFOCOM*, 1993.
[10] H. Balakrishnan *et al.*, "A layered naming architecture for the internet," in *SIGCOMM*, 2004.
[11] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Proceedings of IPTPS02*, 2002.
[12] M. Caesar *et al.*, "Virtual ring routing: network routing inspired by DHTs," in *SIGCOMM*, 2006.
[13] B. Ford, "Unmanaged internet protocol: taming the edge network management crisis," *SIGCOMM*, 2004.
[14] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, and I. Stoica, "Rofl: routing on flat labels," in *SIGCOMM*, 2006.
[15] C. Ee, S. Ratnasamy, and S. Shenker, "Practical data-centric storage," in *NSDI 2006*.
[16] S. Ratnasamy *et al.*, "Ght: a geographic hash table for data-centric storage," in *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*.
[17] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, "Geographic routing without location information," in *MobiCom '03*.
[18] S. Jain *et al.*, "VIRO: A Plug & Play, Scalable, Robust and Namespace Independent Virtual Id Routing for Future Networks," in *Tech report, at http://networking.cs.umn.edu/veil/viro.pdf*.
[19] L. Ramachandran, M. Kapoor, A. Sarkar, and A. Aggarwal, "Clustering algorithms for wireless ad hoc networks," in *DIALM*, 2000.
[20] Y. Yu, G. Lu, and Z. Zhang, "Enhancing location service scalability with HIGH-GRADE," in *2004 IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, 2004.
[21] J. Moy *et al.*, "OSPF Version 2," 1994.
[22] A. Greenberg *et al.*, "VL2: A scalable and flexible data center network," *SIGCOMM*, 2009.
[23] J. McQuillan, I. Richer, E. Rosen, B. Beranek, and N. Inc, "The new routing algorithm for the ARPANET," *Communications, IEEE Transactions on [legacy, pre-1988]*, vol. 28, no. 5, pp. 711–719, 1980.
[24] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring isp topologies with rocketfuel," *IEEE/ACM Trans. Netw.*, 2004.
[25] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *SIGCOMM*, 2008.
[26] A. Medina *et al.*, "BRITE: a flexible generator of Internet topologies," 2000.