ROME: Routing On Metropolitan-scale Ethernet

Chen Qian and Simon S. Lam
Department of Computer Science, The University of Texas at Austin
{cqian, lam}@cs.utexas.edu

***Abstract*—We present the architecture and protocols of ROME, a layer-2 network designed to be backwards compatible with Ethernet and scalable to tens of thousands of switches and millions of end hosts. ROME is based upon a recently developed geographic routing protocol, greedy distance vector (GDV). Switches in ROME do not need any location information. Protocol design innovations in ROME include a stateless multicast protocol, a Delaunay DHT, as well as routing and host discovery protocols for a hierarchical network. ROME protocols do not use broadcast. Extensive experimental results from a packet-level event-driven simulator, in which ROME protocols are implemented in detail, show that ROME protocols are efficient and scalable to metropolitan size. Furthermore, ROME protocols are highly resilient to network dynamics. The routing latency of ROME is only slightly higher than shortest-path latency. To demonstrate scalability, we provide simulation performance results for ROME networks with up to 25,000 switches and 1.25 million hosts.**

I. Introduction

Layer-2 networks, *each* scalable to tens of thousands of switches and connecting millions of end hosts, are needed for important future and current applications and services [18] including: metro Ethernet [1], [3], [9], [10], wide area Ethernet [4], data center networks [8], as well as enterprise and provider networks. We refer to such large-scale layer-2 networks collectively as *metropolitan-scale Ethernet*.

Ethernet offers plug-and-play functionality and a flat MAC address space. Ethernet MAC addresses, being permanent and location independent, support host mobility and facilitate management functions, such as trouble shooting and access control. For these reasons, Ethernet is easy to manage. However Ethernet is not scalable to a large network because it uses a spanning tree routing protocol that is highly inefficient and not resilient to failures. Also, it relies on network-wide flooding for host discovery.

Today's metropolitan and wide area Ethernet services provided by network operators are based upon a network of IP (layer-3) and MPLS routers which interconnect relatively small Ethernet LANs [9]. Adding the IP layer to perform end-to-end routing in these networks nullifies Ethernet's desirable properties. IP routing protocols (such as, RIP, OSPF, and IS-IS) are also not scalable, even though they provide shortest paths and are much more efficient than Ethernet's spanning tree protocol. More importantly, large IP networks require massive efforts by human operators to configure and manage.

Therefore, it is desirable to have a *metropolitan-scale layer-2 network that is backwards compatible with Ethernet*, i.e., its switches interact with hosts by Ethernet frames using conventional Ethernet format and semantics.

978-1-4673-2447-2/12/$31.00

Towards this goal, Myers et al. [18] proposed replacing Ethernet broadcast for host discovery by a layer-2 distributed directory service. In 2007, replacing Ethernet broadcast by a distributed hash table (DHT) was proposed independently by Kim and Rexford [13] and Ray et al. [21]. In 2008, Kim et al. presented SEATTLE [12] which uses link-state routing, a one-hop DHT (based on link-state routing) for host discovery, and multicast trees for broadcasting to VLANs. Scalability of SEATTLE to metropolitan scale is limited by link-state broadcast. In 2009, AIR [23] was proposed to replace link state routing in SEATTLE. However, its latency was found to be larger than the latency of SEATTLE by 1.5 orders of magnitude. In 2011, VIRO [11] was proposed to replace link-state routing. To construct a rooted virtual binary tree for routing, it uses a centralized algorithm for large networks and a distributed algorithm for small networks. In all three papers [12], [23], [11], simulation experiments were performed for networks of several hundred switches.

To achieve the goal of metropolitan-scale Ethernet, we present the ROME architecture and protocols. ROME protocols utilize some recent advances in geographic routing, namely, GDV on VPoD [20] and MDT [14]. Geographic routing is highly scalable because the routing state needed at each node is independent of network size. Switches in ROME *do not need any location information*. For routing in ROME, a virtual space is first specified, such as, a rectangular area in 2D.[1] At network initialization, each switch assigns itself a random location in the virtual space and discovers its neighbors (switches). Each pair of directly-connected switches exchange their unique identifiers (e.g., MAC addresses) and self-assigned locations. Then, the switches have enough information to construct and maintain a multi-hop Delaunay triangulation using MDT protocols [14].

A. Services provided by MDT, VPoD, and GDV

A Delaunay triangulation (DT) is a graph that can be computed from a set of node locations in a Euclidean space [7]. In a DT, two nodes sharing an edge are said to be DT neighbors. For 2D, Bose and Morin [5] proved that greedy forwarding in a DT provides guaranteed delivery to a destination node. For 3D and higher dimensional Euclidean spaces, Lee and Lam [15] generalized their result and proved that greedy forwarding in a DT provides guaranteed delivery to the node closest to a destination *location*. Since two neighbors in a DT graph may not be directly-connected, nodes maintain forwarding tables

[1] 2D, 3D, or a higher dimension can be used [14].

for communication between DT neighbors multiple hops apart (hence the name, multi-hop DT [14]).

Switches, having constructed a multi-hop DT, then repeatedly exchange messages with their neighbors, including multi-hop DT neighbors, and change their positions. Using the VPoD protocol [20], each switch moves its location in the virtual space by comparing, for each neighbor, the (Euclidean) distance with the routing cost between them. (Routing cost can be in any additive metric.) A switch stops when the location change has converged to less than a threshold value. When all switches finish, the distance between two switches in the virtual space approximates the routing cost between them. Then switches use their updated locations to construct a new multi-hop DT to be used by the GDV routing protocol [20].

GDV performs greedy routing in the multi-hop DT. When a packet is stuck at a switch that is a local minimum (i.e., a switch closer to the destination *location* than any of its directly-connected and multi-hop DT neighbors), GDV uses MDT forwarding for recovery. Therefore, GDV also provides guaranteed delivery to the switch closest to a destination location [14], [20]. It has been shown that the VPoD protocol is very effective such that GDV's routing cost is not much higher than that of shortest-path routing. Lastly, MDT and VPOD protocols do not use broadcast. In particular, MDT has a very efficient and effective search method for each switch to find its multi-hop DT neighbors. As a result, a multi-hop DT has been shown to be highly resilient to rapid topology changes [14], [20].

B. Contributions and paper outline

Protocol design innovations of this paper include the following: (i) a stateless multicast protocol to support VLAN and other multicast applications; (ii) protocols for host and service discovery using a new method, called Delaunay DHT (D^2HT); (iii) new routing and host discovery protocols for a hierarchical network.

ROME and SEATTLE were evaluated and compared using a packet-level event-driven simulator in which ROME protocols (including GDV, MDT, and VPoD) and SEATTLE protocols are implemented in detail. Every protocol message is routed and processed by switches hop by hop from source to destination. Experimental results show that ROME protocols are efficient and scalable. The routing latency of ROME is only slightly higher than the shortest-path latency. ROME protocols are highly resilient to network dynamics and switches quickly recover after a period of churn. To demonstrate scalability, we provide simulation performance results for ROME networks with up to 25,000 switches and 1.25 million hosts.

The balance of this paper is organized as follows. In Section II, we present *location hashing* in a virtual space and stateless multicast. In Section III, we present Delaunay DHT and its application to host discovery, i.e., address and location resolution. In Section IV, we present ROME's architecture and routing protocols for hierarchical networks. In Section V, we present performance evaluation and comparison of ROME versus SEATTLE. We conclude in Section VI.

II. Routing in ROME

Consider a network of switches with an arbitrary topology (any connected graph). Each switch selects one of its MAC addresses to be its identifier. End hosts are connected to switches which provide frame delivery between hosts. Ethernet frames for delivery are encapsulated in ROME packets. Switches interact with hosts by Ethernet frames using conventional Ethernet format and semantics. ROME protocols run only in switches. Link-level delivery is assumed to be reliable.

A. Virtual space for switches

A Euclidean space (2D, 3D, or a higher dimension) is chosen as the virtual space. The number of dimensions and the minimum and maximum coordinate values of each dimension are known to all switches. Each switch determines for itself a location in the space represented by a set of coordinates.

Location hashing. To start ROME protocols, each switch boots up and assigns itself an initial location randomly by hashing its identifier, ID_S, using a globally-known hash function H. The hash value is a binary number which is converted to a set of coordinates. Our protocol implementation uses the hash function MD5 [22], which outputs a 16-byte binary value. 4 bytes are used for each dimension. Thus locations can be in 2D, 3D, or 4D.[2]

Consider, for example, a network that uses a 2D virtual space. For 2D, the last 8 bytes of $H(ID_S)$ are converted to two 4-byte binary numbers, x and y. Let MAX be the maximum 4-byte binary value, that is, $2^{32}-1$. Also let min_k and max_k be the minimum and maximum coordinate values for the kth dimension. Then the location in 2D obtained from the hash value is $(min_1 + \frac{x}{MAX}(max_1 - min_1),\ min_2 + \frac{y}{MAX}(max_2 - min_2))$, where each coordinate is a real number. The location can be stored in decimal format, using 4 bytes per dimension. Hereafter, for any identifier, ID, we will use $H(ID)$ to represent its location in the virtual space and refer to $H(ID)$ as the identifier's *location hash* or, simply, *location*.

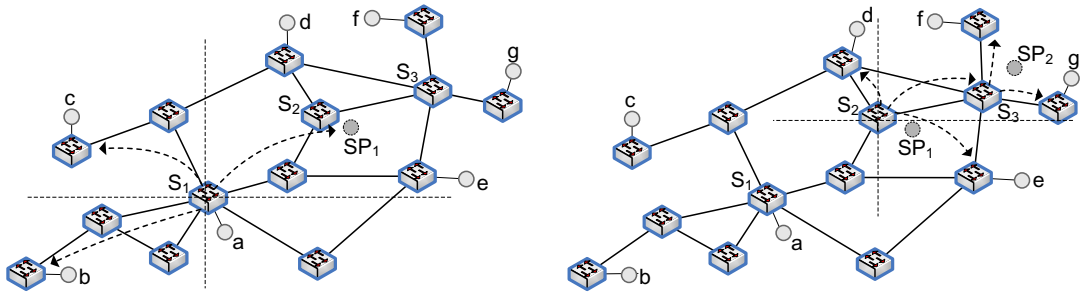
Switches discover their directly-connected neighbors and, using their initial locations, proceed to construct a multi-hop DT [14]. Switches then update their locations using VPoD and construct a new multi-hop DT as described in subsection I-A.

Unicast routing. Unicast packet delivery in ROME is provided by GDV routing in the multi-hop DT maintained by switches. In a correct multi-hop DT, *GDV routing provides guaranteed delivery of any packet to the switch that is closest to the packet's destination location* [14], [20].

B. Hosts

Hosts have IP and MAC addresses. Each host is directly connected to a switch called its *access switch*. An access switch knows the IP and MAC addresses of every host connected to it. The routable address of each host is the location of its access switch in the virtual space, also called the *host's location*. Hosts are not aware of ROME protocols and

[2]Conceptually, a higher dimensional space gives VPoD more flexibility but requires more storage space and control overhead. Our experimental results show that VPoD's performance in 2D is already very good.



(a) Split at S_1 (b) Splits at S_2 and S_3

Fig. 1. Example of stateless multicast

run ARP [19], DHCP [6], and Ethernet protocols in the same way as when they are connected to a conventional Ethernet.

C. Stateless multicast and its applications

To provide the same services as conventional Ethernet, ROME needs to support group-wide broadcast or multicast, for applications, such as, VLAN, teleconferencing, television, replicated storage/update in data centers, etc.

A straightforward way to deliver messages to a group is by using a multicast tree similar to IP multicast [12]. When there are many groups with many hosts in each group, the amount of multicast state stored in switches can become a scalability problem.

We present a *stateless multicast* protocol for group-wide broadcast in ROME. A group message is delivered using the locations of its receivers without construction of any multicast tree. Switches do not store any state for delivering group messages.

The membership information of stateless multicast is maintained at a *rendezvous point* (RP) for each group. The RP of a group is determined by the location hash $H(ID_G)$, where ID_G is the group's ID. The switch whose location is closest to $H(ID_G)$ serves as the group's RP. The access switch of the sender of a group message sends the message to the RP by unicast. GDV routing guarantees message delivery to the switch closest to $H(ID_G)$.

The RP then forwards the message to other group members (receivers) as follows: The RP partitions the entire virtual space into multiple regions. To each region with one or more receivers, the RP sends a copy of the group message with the region's receivers (their locations) in the message header (actually the ROME packet header). The destination of the group message for each region is a location, called *split position* (SP), which is either (i) the closest receiver location in that region, or (ii) the mid-point of the two closest receiver locations in the region. By GDV routing, the group message will be routed to a switch closest to the SP. This switch will in turn partition its region into multiple sub-regions and send a copy of the group message to the SP of each sub-region. Thus a multicast tree rooted at the RP grows recursively until it reaches all receivers. The tree structure is not stored anywhere. At each step of the tree growth, a switch computes SPs for the next step based on receiver locations in the group message it is to forward.

We present an example of stateless multicast in Figure 1(a). The group consists of 7 hosts a,b,c,d,e,f,g, connected to different switches with locations in a 2D virtual space as shown. Switch S_1 serves as the RP. Host a sends a message to the group by first sending it to S_1. Upon receiving the message, S_1 realizes that it is the RP. S_1 partitions the entire virtual space into four quadrants and sends a copy of the message by unicast to each of the 3 quadrants with at least one receiver. The message to the northeast quadrant with four receivers (d,e,f, and g) is sent to a split position, SP_1, which is the midpoint between the locations of d and e, the two receivers closest to S_1. The message will then be routed by GDV to S_2, the switch closest to SP_1.

Subsequently, S_2 partitions the space into four quadrants and sends a copy of the message to each of the three quadrants with one or more receivers (see Figure 1(b)). For the northeast quadrant that has two receivers, the message is sent to the split position, SP_2, which is the midpoint between the locations of f and g. The message to SP_2 will be routed by GDV to S_3, the switch closest to SP_2, which will unicast copies of the message to f and g.

At any time during the multicast, when a switch realizes that a receiver is a directly-connected host, it can transmit the message directly to the host and removes the host from the set of receivers in the message to be forwarded.

In ROME, for each group, its group membership information is stored in only one switch, the group's RP. For this group, no multicast state is stored in any other switch. This is a major step towards scalability. The tradeoff for this gain is an increase in communication overhead from storing a set of receivers in the ROME header of each group message. Experimental results in subsection V-F show that this communication overhead is small. This is because when the group message is forwarded by the RP and other switches, the receiver set is partitioned into smaller and smaller subsets.

The implementation of stateless multicast, as described, is not limited to the use of a 2D space. Also, partitioning of a 2D space at the RP, or at a switch closest to a SP, is not limited to four quadrants. The virtual space can be partitioned into any number of regions evenly or unevenly. A study of other virtual spaces and partitioning methods for implementing

stateless multicast will be future work.

Stateless multicast for VLAN. Members of a VLAN are in a logical broadcast domain; their locations may be widely distributed in a large-scale Ethernet. ROME's stateless multicast protocol is used to support VLAN broadcast. When a switch detects that one of its hosts belongs to a VLAN, it sends a Join message to location $H(ID_V)$, where ID_V is the VLAN ID. By GDV, The Join message is routed to the switch closest to $H(ID_V)$, which is the RP of the VLAN. The RP then adds the host to the VLAN membership. The protocol for a host to leave a VLAN is similar. VLAN protocols in ROME are much more efficient than the current VLAN Trunking Protocol used in conventional Ethernet [2]. The number of global VLANs is restricted to 4094 in conventional Ethernet [9]. There is no such restriction in ROME because stateless multicast does not require switches to store VLAN information to perform forwarding.

III. Host and Service Discovery in ROME

Suppose a host knows the IP address of a destination host from some upper-layer service. To route a packet from its source host to its destination host, switches need to know the MAC address of the destination host as well as its location, i.e., location of its access switch. Such address and location resolution are together referred to as *host discovery*.

A. Delaunay distributed hash table

The benefits of using a DHT for host discovery include the following: (i) uniformly distributing the storage cost of host information over all network switches, and (ii) enabling information retrieval by unicast rather than flooding. The one-hop DHT in SEATTLE [12] uses *consistent hashing* of identifiers into a circular location space and requires that every switch knows *all* other switches. Such global knowledge is made possible by link-state *broadcast*.

In ROME, the Delaunay DHT (or D^{2}HT) uses *location hashing* of identifiers into a Euclidean space (2D, 3D, or a higher dimension) as described in subsection II-A. D^{2}HT uses greedy routing (GDV) in a multi-hop DT where every switch only needs to know its directly-connected neighbors and its neighbors in the DT graph. Furthermore, each switch uses a very efficient search method to find its multi-hop DT neighbors without broadcast [14].

In D^{2}HT, information about host i is stored as a key-value tuple, $t_i =< k_i, v_i >$, where the key k_i may be the IP (or MAC) address of i, and v_i is host information, such as its MAC address, location, etc. The access switch of host i is the *publisher* of i's tuples. A switch that stores $< k_i, v_i >$ is called a *resolver* of key k_i. The tuples are stored as *soft state*.

To publish a tuple, $t_i =< k_i, v_i >$, the publisher computes its location $H(k_i)$ and sends a publish message of t_i to $H(k_i)$. Location hashes are randomly distributed over the entire virtual space. It is possible but unlikely that a switch exists at the exact location $H(k_i)$. The publish message is routed by GDV to the switch whose location is closest to $H(k_i)$, which then becomes a resolver of k_i. When some other switch needs host i's information, it sends a lookup request message to location $H(k_i)$. The lookup message is routed by GDV to the resolver of k_i, which sends the tuple $< k_i, v_i >$ to the requester. A publish-lookup example is illustrated in Figure 2.

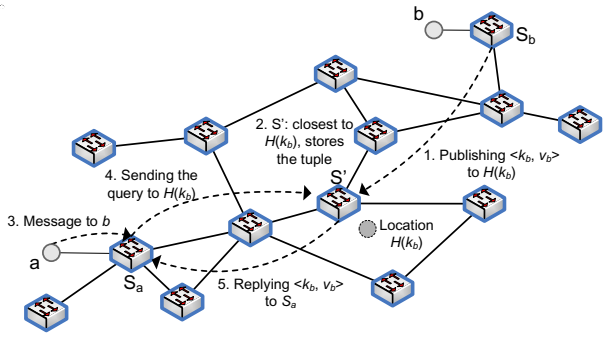


Fig. 2. S_b publishes a tuple of b. S_a performs a lookup of b

B. Host discovery using D^{2}HT

In ROME, the routable address of host i is i's location c_i, which is the location of its access switch. There are two key-value tuples for each host, for its IP-to-MAC and MAC-to-location mappings.

In a tuple for host i, the key k_i may be its IP or MAC address. If k_i is the MAC address, value v_i includes location c_i and the unique ID, S_i, of i's access switch. If k_i is the IP address, the value v_i includes the MAC address, MAC_i, as well as c_i and S_i. Note that the host location is included in both tuples for each host.

After a host i is plugged into its access switch S_i with location c_i, the switch learns the host's IP and MAC addresses, IP_i and MAC_i, respectively [12]. S_i then constructs two tuples: $< MAC_i, c_i, S_i >$ and $< IP_i, MAC_i, c_i, S_i >$, and stores them in local memory. S_i then sends publish messages of the two tuples to $H(IP_i)$ and $H(MAC_i)$.

Note that each switch stores two kinds of tuples. For a tuple with key k_i stored by switch S, if S is i's access switch, the tuple is a *local tuple* of S. Otherwise, the tuple is published by another switch and is an *external tuple* of S. Switches store key-value tuples as *soft state*.

Each switch interacts with directly-connected hosts using frames with conventional Ethernet format and semantics. When a host j sends its access switch S_j an ARP query frame with destination IP address IP_i and the broadcast MAC address, S_j sends a lookup request to location $H(IP_i)$, which is routed by GDV to a resolver of IP_i. The resolver sends back to S_j the tuple $< IP_i, MAC_i, c_i, S_i >$. After receiving the tuple, the access switch S_j caches the tuple and transmits a conventional ARP reply frame to host j. When j sends an Ethernet frame with destination MAC_i, the access switch S_j retrieves location c_i from its local memory and sends the Ethernet frame to c_i. If S_j cannot find the location of MAC_i in its local memory because, for instance, the cached tuple has been overwritten, it sends a lookup request which is routed by GDV to $H(MAC_i)$ to get the MAC-to-location mapping of host i.

All publish and lookup messages are unicast messages. Host discovery in ROME is accomplished *on demand* and is

flooding-free.

Reducing lookup latency. Reducing the lookup latency for host discovery is a significant concern. We designed and evaluated three techniques to speed up key-value lookup, including using multiple independent hash functions to publish each key-value tuple at multiple locations, hashing to a smaller region in the virtual space, and caching key-value tuples for popular hosts in every switch. A discussion of these techniques is omitted due to page limitation.

C. Maintaining consistent key-value tuples

A key-value tuple $< k_i, v_i >$ stored as an external tuple in a switch is *consistent* iff (i) the switch is closest to the location $H(k_i)$ among all switches in the virtual space, and (ii) c_i is the correct location of i's access switch. At any time, some key-value tuples may become inconsistent as a result of host or network dynamics.

Host dynamics. A host may change its IP or MAC address, or both. A host may change its access switch, such as, when a mobile node moves to a new physical location or a virtual machine migrates to a new system.

Network dynamics. These include the addition of new switches or links to the network as well as deletion/failure of existing switches and links. MDT and VPoD protocols have been shown to be highly resilient to network dynamics (churn) [14], [20]. Switch states of the multi-hop DT as well as switch locations in the virtual space recover quickly to correct values after churn. The following discussion is limited to how host and network dynamics are handled by switches in the role of publisher and in the role of resolver in D^2HT.

As a publisher, each switch ensures that local tuples of its hosts are correct when there are host dynamics. For example, if a host has changed its IP or MAC address, the host's tuples are updated accordingly. If a new host is plugged into the switch, it creates tuples for the new host. New as well as updated tuples are published to the network. In addition to these reactions to host dynamics, switches also periodically refresh tuples they previously published. For every local tuple $< k_i, v_i >$, S sends a refresh message every T_r second to its location $H(k_i)$. The purpose of a refresh message are twofold: (i) If the switch closest to location $H(k_i)$ is the current resolver, timer of the soft-state tuple in the resolver is refreshed. (ii) If the switch closest to $H(k_i)$ is different from the current resolver, the refresh message notifies the switch to become a resolver.

As a resolver, each switch sets a timer for every external tuple stored in local memory. The timer is refreshed by a request or refresh message for the tuple. If a timer has not been refreshed for T_e time, timeout occurs and the tuple will be deleted by the resolver. T_e is set to a value several times that of T_r.

For faster recovery from network dynamics, we designed and implemented a technique, called *external tuple handoff*. When a switch detects topology or location changes in the multi-hop DT, it checks the location $H(k_i)$ of every external tuple $< k_i, v_i >$. If the switch finds a physical or DT neighbor closer to $H(k_i)$ than itself, it sends a *handoff message* including the tuple to the closer neighbor. The handoff message will be forwarded by GDV until it reaches the switch closest to $H(k_i)$, which then becomes the tuple's new resolver.

D. DHCP server discovery using D^2HT

In a conventional Ethernet, a new host broadcasts a Dynamic Host Configuration Protocol (DHCP) discover message to find a DHCP server. Each DHCP server that has received the discover message allocates an IP address and broadcasts a DHCP offer message to the host. The host broadcasts a DHCP request to accept an offer. The selected server broadcasts a DHCP ACK message. Other DHCP servers, if any, withdraw their offers.

In ROME, the access switch of each DHCP server publishes the server's information to a location using a key known by all switches, such as,"DHCPSERVER1". When the access switch of a host receives a DHCP discover message, the message is routed by GDV to the location of a DHCP server, without use of flooding. There is no duplicate DHCP offer. To be compatible with a conventional Ethernet, the access switch replies to the host with a DHCP offer and later transmits a DHCP ACK in response to the host's DHCP request.

IV. ROME for Hierarchical Ethernet

A metropolitan or wide area Ethernet spanning across a large geographic area typically has a hierarchical structure comprising many *access networks* interconnected by a *core network* [10]. Each access network has one or more *border switches*. The border switches of all access networks form the core network. Consider a hierarchical network consisting of 500 access networks each of which has 2000 switches. The total number of switches is 1 million. At 100 hosts per switch, the total number of hosts is 100 millions. We believe that a 2-level hierarchy is adequate for metropolitan scale in the foreseeable future.

A. Routing in a hierarchical network

For hierarchical routing in ROME, separate virtual spaces are specified for the core network and each of the access networks, called *regions*. Every switch knows the virtual space of its region (i.e., dimensionality as well as maximum and minimum coordinate values of each dimension). Every border switch knows two virtual spaces, the virtual space of its region and the virtual space of the core network, called *backbone*.

The switches in a region first discover their directly-connected neighbors. They then use MDT and VPoD protocols to determine their locations in the region's virtual space (*regional locations*) and construct a multi-hop DT for the access network. Similarly, the border switches use MDT and VPoD protocols to determine their locations in the virtual space of the backbone (*backbone locations*) and construct a multi-hop DT for the core network. Each border switch sends its information (unique ID, regional and backbone locations) to all switches in its region.

The Delaunay DHT requires the following extension for hierarchical routing: Each key-value tuple $< k_i, v_i >$ of host i stored at a resolver includes additional information, B_i, which specifies the IDs and backbone locations of the border switches in host i's region.

When a host sends an Ethernet frame to another host, its access switch obtains, from its cache or using host discovery, the destination host's key-value tuple, which includes border switch information of the destination region. This information allows the access switch to determine whether to route the frame to its destination using *intra-region* routing or *inter-region* routing.

Intra-region routing. The sender's access switch indicates in the ROME packet header that this is an intra-region packet. The routable address is the regional location of the access switch of the receiver. The packet will be routed by GDV to the access switch of the receiver as previously described. In the example of Figure 3, an intra-region packet is routed by GDV from access switch S_1 to destination host's access switch S_2 in the same regional virtual space.

Inter-region routing. For a destination host in a different region, an access switch learns, from the host's key-value tuple, information about the host's border switches and their backbone locations. This information is included in the ROME header encapsulating every Ethernet frame destined for that host. We describe inter-region routing of a ROME packet as illustrated in Figure 3. The origin switch S_1 computes its distances in the regional virtual space to the region's border switches, S_3 and S_4. S_1 chooses S_3 which is closer to S_1 than S_4. The packet is routed by GDV to S_3 in the regional virtual space. S_3 learns from the ROME packet header, S_5 and S_8, border switches in the destination's region. S_3 computes their distances to destination S_7 in the destination region's virtual space. S_3 chooses S_5 because it is closer to the destination location. The packet is then routed by GDV in the backbone virtual space to S_5. Lastly, the packet is routed, in the destination region's virtual space, by GDV from S_5 to S_7, which extracts the Ethernet frame from the ROME packet and transmits the frame to the destination host.

Note that at the border switch S_3, it has a *choice of minimizing the distance traveled by the ROME packet in the backbone virtual space or in the destination region's virtual space*. In our current ROME implementation, the distance in the destination region's virtual space is minimized. This is based upon our current assumption that the number of switches in an access network is larger the number of switches in the core network. This choice at a border switch is programmable and can be easily reversed. Lastly, it is not advisable to use the sum of distances in two different virtual spaces (specified independently) to determine routing because they are not comparable. This restriction may be relaxed but it is beyond the scope of this paper.

B. Host discovery in a hierarchical network

As illustrated in Figure 4, the key-value tuple $< k_i, v_i >$ of host i is published to two resolvers in the entire network,

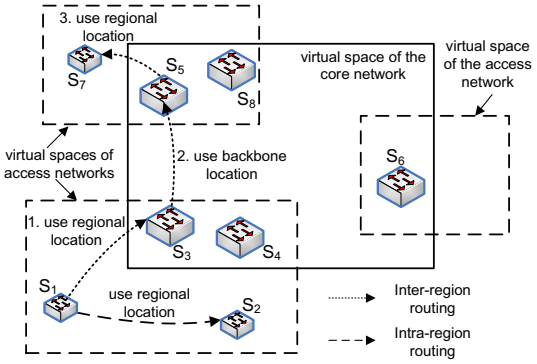


Fig. 3. Routing in a hierarchical Ethernet

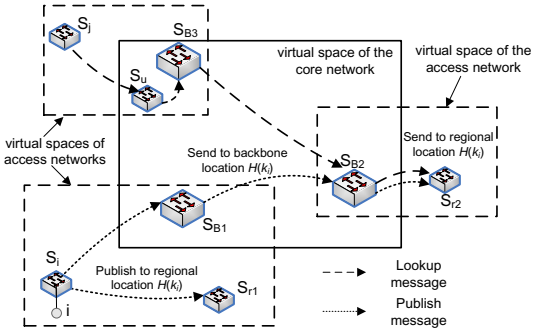


Fig. 4. Tuple publishing and lookup in a hierarchical Ethernet

namely: a *regional resolver* and a *global resolver*. The regional resolver is the switch closest to location $H(k_i)$ in the same region as host i; it is labeled by S_{r1} in the figure. The publish and lookup protocols are the same as the ones presented in subsection III-B. To find a tuple with key k_i, a switch sends a lookup message to position $H(k_i)$ in its own region. A regional resolver provides fast responses to queries needed for intra-region communications.

However, switches outside of the host's region cannot find its regional resolver. Therefore, the key-value tuple $< k_i, v_i >$ of host i is also stored in a global resolver to respond to host discovery for inter-region communications. The global resolver can be found by any switch in the entire network. As shown in Figure 4, to publish a tuple $< k_i, v_i >$ to its global resolver, the publish message is first routed by GDV to the regional location of one of the border switches in the region, labeled by S_{B1} in the figure. S_{B1} computes location $H(k_i)$ in the backbone virtual space and includes it with the publish message which is routed by GDV to the border switch closest to backbone location $H(k_i)$ in the core network, labeled by S_{B2} in the figure. Switch S_{B2} serves as the global resolver of host i if it has enough memory space. Switch S_{B2} can optionally send the tuple to a switch in its region such that all switches in the region share the storage cost of the global resolver function (called *two-level location hashing*). In two-level location hashing, the publish message of tuple $< k_i, v_i >$ sent by S_{B2} is routed by GDV to a switch closest to the regional location $H(k_i)$ (labeled by S_{r2} in the figure) inside S_{B2}'s access network. S_{r2} then becomes a global resolver of host i.

To discover the key-value tuple $< k_i, v_i >$ of host i, a switch S_j first sends a lookup message to location $H(k_i)$ in its region.

The lookup message arrives at a switch S_u closest to $H(k_i)$, as illustrated in Figure 4 (upper left). If S_j and host i were in the same region, S_u would be the regional resolver of i and it would reply to S_j with the key-value tuple of host i. Given that S_j and host i are in different regions, it is very unlikely that S_u happens to be a global resolver of host i (however the probability is nonzero). If S_u cannot find host i's tuple in its local memory, it forwards the lookup message to one of the border switches in its region, S_{B3} in Figure 4. Then S_{B3} computes location $H(k_i)$ in the backbone virtual space and includes it with the lookup message, which is routed by GDV to the border switch S_{B2} closest to $H(k_i)$. In the scenario illustrated in Figure 4, S_{B2} is not host i's global receiver and it forwards the lookup message to switch S_{r2} closest to the regional location $H(k_i)$, which is the global resolver of host i.

In the above examples, the core and access networks use different virtual spaces but they all use the same hash function H. We note that different hash functions can be used in different networks. It is sufficient that all switches in the same network (access or core) agree on the same hash function, just like they must agree on the same virtual space.

V. Performance Evaluation

A. Methodology

The ROME architecture and protocols have been designed with the objectives of *scalability*, *efficiency*, and *reliability*. ROME was evaluated using a packet-level event-driven simulator in which ROME protocols as well as the protocols, GDV, VPoD, and MDT [14], [20] used by ROME are implemented in detail. Every protocol message is routed and processed by switches hop by hop from source to destination. Since our focus is on routing protocol design, queueing delays at switches were not simulated. Packet delays from one switch to another on an Ethernet link are sampled from a uniform distribution in the interval $[50\,\mu s, 150\,\mu s]$ with an average value of $100\,\mu s$. This abstraction, aside from speeding up simulation runs, allows performance evaluation and comparison of routing protocols unaffected by congestion issues. The same abstraction was used in the packet-level simulator of SEATTLE [12].

For comparison with ROME, we implemented SEATTLE protocols in detail in our simulator. We conducted extensive simulations to evaluate ROME and SEATTLE in large networks and dynamic networks with reproducible topologies. For the link-state protocol used by SEATTLE, we use OSPF [17] in our simulator. The default OSPF link state broadcast frequency is once every 30 seconds. Therefore, in ROME, each switch runs the MDT maintenance protocol once every 30 seconds.

In ROME, a host's key-value tuple may be published using one location hash or two location hashes. In the case of publishing two location hashes for each tuple, the area of the second hash region is 1/4 of the entire virtual space.

Performance criteria. *Storage cost* is measured by the *average number of entries stored per switch*. These entries include forwarding table entries and host information entries (key-value tuples).

Control overhead is communication cost measured by the *average number of control message transmissions*, for three cases: (i) network initialization, (ii) network in steady state, and (iii) network under churn. Control overhead of ROME for initialization includes those used by switches to determine virtual locations using VPoD, construct a multi-hop DT using MDT protocols, and populate the D^2HT with host information for all hosts. Control overhead of SEATTLE for initialization includes those used by switches for link-state broadcast and to populate the one-hop DHT with host information for all hosts. During steady state (also during churn), switches in SEATTLE and ROME use control messages to detect inconsistencies in forwarding tables as well as key-value tuples stored locally and externally. Extra control messages are used to repair inconsistencies in forwarding tables and key-value tuples to recover from churn.

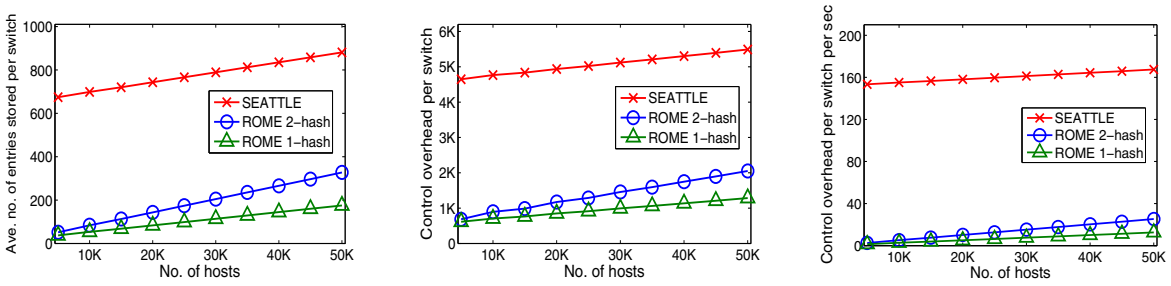
We measure two kinds of *latencies to deliver ROME packets*: (i) latency of the first packet to an unknown host, which includes the latency for host discovery, and (ii) latency of a packet to a discovered host.

To evaluate ROME's (also SEATTLE's) resilience under churn, we show the *routing failure rates* of first packets to unknown hosts and packets to discovered hosts. Successful routing of the first packet to an unknown host requires successful host discovery as well as successful packet delivery by switches from source to destination.

Network topologies used. The first set of experiments used the AS-6461 topology with 654 routers from Rocketfuel data [24] where each router is modeled as a switch. To evaluate the performance of ROME as the number of switches increases, synthetic topologies generated by BRITE with the Waxman model [16] at the router level were used. Every data point plotted in Figures 6, 7, and 9 is the average of 20 runs from different topologies generated by BRITE. *Upper and lower bars in the figure show maximum and minimum values of each data point* (these bars are omitted in Figure 7(c) for clarity). Most of the differences between maximum and minimum values in these figures are very small (many not noticeable) with the exception of latency values in Figures 7(a) and (b).

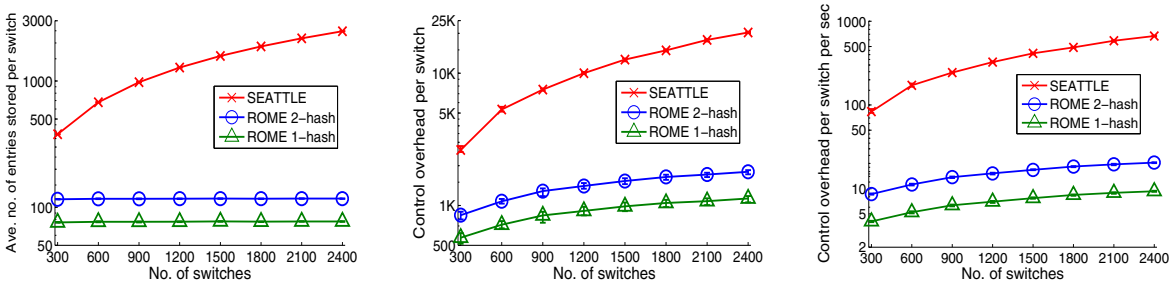
B. Varying the number of hosts

For a network with n switches and m hosts, a conventional Ethernet requires $O(nm)$ storage per switch while SEATTLE requires $O(m)$ storage per switch. We found that ROME also requires $O(m)$ storage per switch with a smaller absolute value than that of SEATTLE. We performed simulation experiments for a fixed topology (AS-6461) with 654 switches. The number of hosts at each switch varies. The total number of hosts of the entire network varies from 5,000 to 50,000. We found that the storage costs of ROME and SEATTLE for forwarding tables are constant, while their storage costs for host information increase linearly as the number of hosts increases. In Figure 5(a), the difference between the storage costs of ROME and SEATTLE is the difference in their forwarding table storage costs per switch. The host information storage cost of ROME using two (location) hashes is close to, but not larger than,



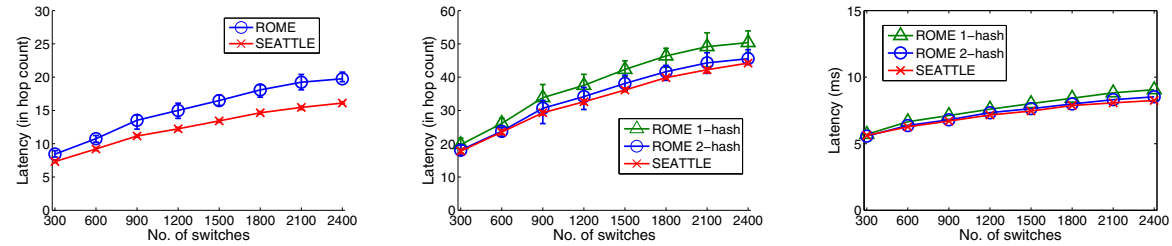
(a) Storage cost (b) Control overhead for initialization (c) Control overhead in steady state

Fig. 5. Performance comparison by varying the number of hosts



(a) Storage cost (b) Control overhead for initialization (c) Control overhead in steady state

Fig. 6. Performance comparison by varying the number of switches



(a) Packet to a discovered host (b) 1st packet to an unknown host (c) 1st packet to unknown host in hybrid nets

Fig. 7. Latency vs. number of switches

twice the storage cost of ROME using one hash.

Figures 5(b) and 5(c) show the control overheads of ROME and SEATTLE, for initialization and in steady state. We found that the control overheads for constructing and updating SEATTLE's one-hop DHT and ROME's D^{2}HT both increase linearly with m and they are about the same. However, the figures show that ROME's overall control overhead is much smaller than that of SEATTLE. This is because ROME's forwarding table construction and maintenance are flooding-free and thus much more efficient.

C. Varying the number of switches

In this set of experiments the number n of switches increases from 300 to 2,400 while the average number of hosts per switch is fixed at 20. Thus the total number of hosts of the network also increases linearly from 6,000 to 48,000. The results are shown in Figure 6. Note that each y-axis is in logarithmic scale.

Figure 6(a) shows storage cost versus n. Note that while the storage cost of SEATTLE increases with n, ROME's storage cost is almost flat versus n. At $n = 2400$, ROME's storage cost is less than 1/20 of the storage of SEATTLE.

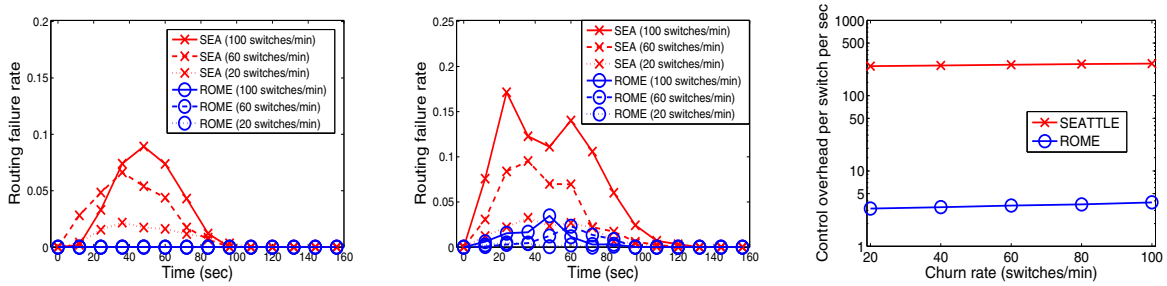
Figures 6(b) and (c) show that the control overheads of ROME for initialization and in steady state are both substantially lower than those of SEATTLE. These control overheads of ROME increase slightly with n. This is because the paths from publishers to resolvers in a larger network are longer.

D. Routing latencies

These experiments were performed using the same network topologies (with 20 hosts per switch on average) as in subsection V-C. Figure 7(a) shows the latency (in average number of hops) of packets to discovered hosts. Note that ROME's latency is not much higher than the shortest-path latency of SEATTLE.

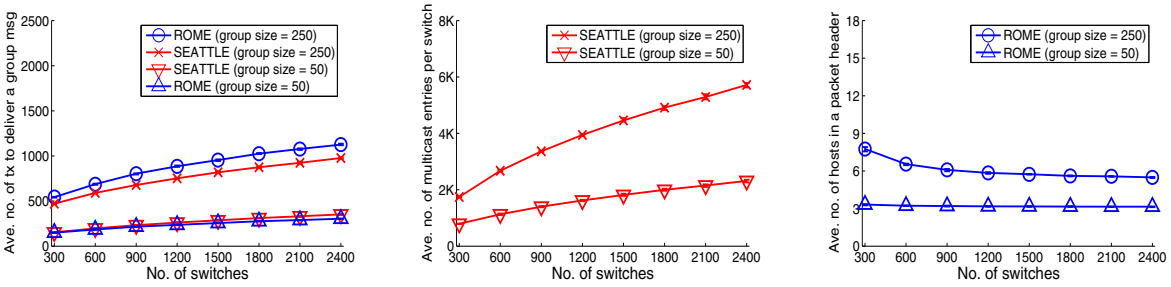
Figure 7(b) shows the latency of first packets to unknown hosts for SEATTLE and for ROME using one and two hashes. This latency includes the round-trip delay between sender and resolver, and the subsequent latency from sender to destination. By using two hashes instead of one, the latency of ROME improves and becomes very close to the latency of SEATTLE. At $n = 300$, the latency of ROME (2-hash) is actually smaller than the latency of SEATTLE.

We also performed experiments to evaluate ROME and SEATTLE latencies in hybrid networks, where 20% of the switches are replaced by wireless switches. The packet delay of a wireless hop is sampled uniformly from $[5\ ms, 15\ ms]$



(a) Routing failure rate to a discovered host (b) Routing failure rate to an unknown host (c) Control overhead

Fig. 8. Performance under network dynamics



(a) Ave. no. of transmissions to deliver a group message (b) Multicast storage cost per switch of SEATTLE (c) Ave. no. of destinations in the ROME header of a group message

Fig. 9. Performance of multicast

with an average value of 10 *ms*, much higher than 100 μs for a wired connection. Figure 7(c) shows that SEATTLE still has the lowest latency, but the difference between SEATTLE and ROME is negligible.

E. Resilience to network dynamics

We performed experiments to evaluate the resilience of ROME using two hashes and SEATTLE under network dynamics for networks with 1,000 switches and 20,000 hosts. Before starting each experiment, consistent forwarding tables and DHTs were first constructed. During the period of 0-60 seconds, new switches joined the network and existing switches failed. The rate at which switches join, equal to the rate at which switches fail, is called the churn rate. Figure 8(a) shows the routing failure rates to discovered hosts as a function of time for ROME and SEATTLE. Different curves correspond to churn rates of 20, 60, and 100 switches per minute. At these very high churn rates, the routing failure rate of ROME is close to zero. The routing failure rate of SEATTLE is relatively high but it converged to zero after 100 seconds (40 seconds after churn stopped).

Figure 8(b) shows routing failure rates to unknown hosts versus time. Both SEATTLE and ROME experienced many more routing failures which include host discovery failures. The routing failure rate of ROME at the churn rate of 100 switches/minute is still less than that of SEATTLE at the churn rate of 20 switches/minute.

Figure 8(c) shows the control overhead (per switch per second) during a period of churn and recovery versus churn rate. The control overhead of SEATTLE is very high due to link-state broadcast. The control overhead of ROME is about two orders of magnitude smaller than that of SEATTLE.

ROME has much smaller routing failure rates and control overhead because each switch (using the MDT maintenance protocol) can find all its neighbors in the multi-hop DT of switches very efficiently without broadcast.

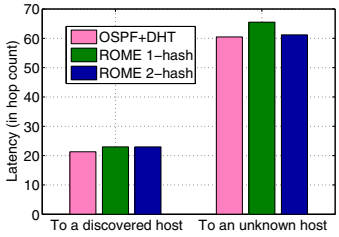


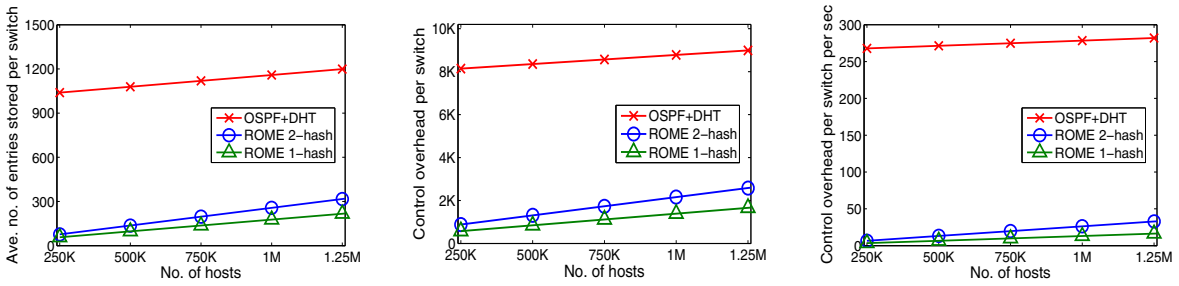
Fig. 10. Latency comparison for a hierarchical network

F. Performance of multicast

Both SEATTLE and ROME provide multicast support for services like VLAN. SEATTLE uses a multicast tree for each group which requires switches in the tree to store some multicast state. ROME uses the stateless multicast protocol described in subsection II-C. We performed experiments using the same network topologies (with 20 hosts per switch on average) as in subsection V-C. The average multicast group size is 50 or 250 in an experiment. The number of groups is 1/10 of the number of hosts.

Figure 9(a) shows the average number of transmissions used to deliver a group message versus the number n of switches. For multicast using a tree, this is equal to the number of links in the tree. SEATTLE used few transmissions than ROME in experiments for average group size 250. ROME used fewer transmissions in experiments for average group size 50.

Figure 9(b) shows the amount of multicast state (average number of groups) per switch in SEATTLE versus n. (ROME's multicast is stateless.) Each switch in SEATTLE stores multicast state for a large number of groups, i.e., thousands in



(a) Storage cost (b) Control overhead for initialization (c) Control overhead in steady state

Fig. 11. Performance comparison for a hierarchical network

these experiments. (Group membership information stored at rendezvous points is not included because it is needed by both ROME and SEATTLE.) On the other hand, ROME requires the packet header of each group message to store a subset of hosts in the group. (SEATTLE does not have this overhead.) Figure 9(c) shows the average number of hosts in a ROME packet header. For experiments in which average group size is 50, the number is around 3. For experiments in which average group size is 250, the number is about 6.

G. Performance of a very large hierarchical network

We use a hierarchical network consisting of 25 access networks of 1000 switches each (generated by BRITE at router level). Two switches in each access network serve as border switches in a backbone network of 50 switches with topology generated by Brite at AS level. Kim et al. [12] discussed ideas for a multi-level one-hop DHT. Based upon the discussion, we implemented in our packet-level event-driven simulator an extension to SEATTLE for routing in a hierarchical network, which we refer to as "OSPF+DHT".

We performed experiments for this network of 25,000 switches for 250K to 1.25 million hosts. Figure 10 shows the routing latencies for ROME and OSPF+DHT. ROME's latency to a discovered host is very close to the shortest-path latency of OSPF+DHT, much closer than the latencies in single-region experiments shown in Figure 7(a). ROME's latency to an unknown host is also very close to the shortest-path latency of OSPF+DHT. Figure 11 shows the storage cost per switch, control overheads for initialization and in steady state. The performance of ROME is about an order of magnitude better than the OSPF+DHT approach.

VI. Conclusions

We present the architecture and protocols of ROME, a layer-2 network that is backwards compatible with Ethernet. Protocol design innovations include a stateless multicast protocol, a Delaunay DHT (D^2HT), as well as routing and host discovery protocols for a hierarchical network. Experimental results show that ROME protocols are efficient and scalable. ROME protocols are highly resilient to network dynamics and its switches quickly recover after a period of churn. The routing latency of ROME is only slightly higher than the shortest-path latency. To demonstrate scalability, we provide simulation performance results for ROME networks with up to 25,000 switches and 1.25 million hosts.

Acknowlegment. This work was sponsored by National Science Foundation grants CNS-0830939 and CNS-1214239.

References

[1] Metro ethernet. http://metro-ethernet.org/.
[2] Understanding VLAN Trunk Protocol (VTP). Cisco Technical Support & Documentation, 2007.
[3] AT&T. Metro ethernet service. http://www.business.att.com/enterprise/Service/network-services/etherne%t/metro-gigabit/.
[4] AT&T. Wide area ethernet. http://www.business.att.com/enterprise/Service/network-services/etherne%t/wide-area-vpls/.
[5] P. Bose and P. Morin. Online routing in triangulations. *SIAM journal on computing*, 33(4):937–951, 2004.
[6] R. Droms. Dynamic Host Configuration Protocol. RFC 2131, 1997.
[7] S. Fortune. Voronoi diagrams and Delaunay triangulations. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, second edition, 2004.
[8] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *Proceedings of ACM SIGCOMM*, 2009.
[9] S. Halabi. *Metro Ethernet*. Cisco Press, 2003.
[10] M. Huynh and P. Mohapatra. Metropolitan Ethernet Network: A Move from LAN to MAN. *Computer Networks*, 51, 2007.
[11] S. Jain, Y. Chen, S. Jain, and Z.-L. Zhang. VIRO: A Scalable, Robust and Name-space Independent Virtual Id ROuting for Future Networks. In *Proc. of IEEE INFOCOM*, 2011.
[12] C. Kim, M. Caesar, and J. Rexford. Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises. In *Proc. of Sigcomm*, 2008.
[13] C. Kim and J. Rexford. Revisiting Ethernet: Plug-and-play made scalable and efficient. In *Proceedings of IEEE LAN/MAN Workshop*, May 2007.
[14] S. S. Lam and C. Qian. Geographic Routing in *d*-dimensional Spaces with Guaranteed Delivery and Low Stretch. In *Proceedings of ACM SIGMETRICS*, June 2011.
[15] D.-Y. Lee and S. S. Lam. Protocol design for dynamic Delaunay triangulation. Technical Report TR-06-48, The Univ. of Texas at Austin, Dept. of Computer Science, December 2006; an abbreviated version in *Proceedings IEEE ICDCS*, June 2007.
[16] A. Medina, A. Lakhina, I. Matta, , and J. Byers. BRITE: An Approach to Universal Topology Generation. In *Int. Workshop on Modeling, Analysis and Simulation of Computer and Telecom. Systems*, 2001.
[17] J. Moy. OSPF Version 2. RFC 2328, 1998.
[18] A. Myers, T. E. Ng, and H. Zhang. Rethinking the service model: Scaling ethernet to a million nodes. In *Proceedings of HotNets*, 2004.
[19] D. Plummer. An Ethernet Address Resolution Protocol. RFC 826, 1982.
[20] C. Qian and S. S. Lam. Greedy Distance Vector Routing. In *Proceedings of IEEE ICDCS*, June 2011.
[21] S. Ray, R. Guerin, and R. Sofia. A distributed hash table based address resolution scheme for large-scale Ethernet networks. In *Proceedings of Int. Conf. on Communications*, June 2007.
[22] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, 1992.
[23] D. Sampath, S. Agarwal, and J. Garcia-Luna-Aceves. Ethernet on AIR: Scalable Routing in Very Large Ethernet-based Networks. In *Proceedings of IEEE ICDCS*, 2010.
[24] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *Proceedings of ACM SIGCOMM*, 2002.