6-24-2007

A Distributed Hash Table based Address Resolution Scheme for Large-scale Ethernet Networks

Saikat Ray
University of Bridgeport, saikatr@bridgeport.edu

Roch A. Guérin
University of Pennsylvania, roch@guerinfamily.org

Rute Sofia
Siemens AG Corporate Technology, rute.sofia@siemens.com

Copyright 2007. Forthcoming from *International Conference on Communications 2007 (ICC 200a7)*, June 2007, pages 6446-6453. http://dx.doi.org/10.1109/ICC.2007.1066

A Distributed Hash Table based Address Resolution Scheme for Large-scale Ethernet Networks

Saikat Ray, Roch Guérin and Rute Sofia

Abstract— **Ethernet's plug-&-play feature is built on its use of flat (location independent) addresses and use of broadcasts to resolve unknown MAC addresses. While plug-&-play is one of Ethernet's most attractive features, it also affects its scalability. As the number of active MAC addresses in the network grows beyond the capacity of forwarding caches in bridges, the odds of "cache-misses," each triggering a broadcast, grow as well. The resulting increase in broadcast bandwidth consumption affects scalability. To address this problem, we propose a simple address resolution scheme based on an adaptation of distributed hash tables where a single query suffices in the steady state. The new scheme is implemented on *advanced* bridges maintaining backward compatibility with legacy bridges and eliminating reliance on broadcasts for address discovery. Comparisons with a legacy, broadcast-based scheme are carried out along several metrics that demonstrate the new scheme's robustness and ability to improve scalability.**

I. Introduction

Ethernet's popularity can to a large extent be attributed to its low cost and ease-of-use. In particular, its "plug-&-play" feature allows an Ethernet device to plug into a network and become operational without the need for any configuration. This simplicity does, however, come at the cost of some loss in efficiency and scalability. Specifically, it relies on two design choices: (i) use of flat (location independent) addresses and (ii) broadcast-based resolution of the location of an unknown address. To understand their scalability implications, let us examine their role in Ethernet's operation (as per 802.1D [1]).

Consider an Ethernet network comprised of several segments, each with a number of nodes (end-hosts). The Network Interface Card (NIC) of each end-host is identified through a "burnt-in" and globally unique 48-bit MAC address. Each segment is a broadcast medium where NICs see all packets transmitted on the segment. Segments are inter-connected through *bridges* that ensure delivery of (a single copy of) packets to end-hosts residing on a segment, and prevent unnecessary packet transmissions to other segments to the extent possible. This requires that upon receiving a packet, an Ethernet bridge be able to determine at most *one* port on which to forward the packet. For this purpose, each bridge builds a forwarding table where entries consist of triplets of the form $<$ MAC Address, PORT, AGE $>$. Populating the forwarding table is carried out based on the *source* addresses of packets that a bridge receives. Specifically, upon receiving a packet on port P1 with a previously unknown source address SA, the bridge creates an entry of the form $<$SA,P1,T$>$. This entry is then used to make forwarding decisions for packets *destined* for address SA—they are sent out on port P1 (such packets received on P1 are dropped).

The main issue Ethernet faces when forwarding packets is what to do packets when there are no matching entries in the forwarding table. This is not only an *initialization* problem, but very much also a *steady-state* one. Specifically, since the lack of hierarchical structure in Ethernet MAC addresses precludes address aggregation, without additional mechanisms the number of entries in the forwarding table of a bridge can grow linearly with the number of addresses in the network. Size constraints of forwarding tables [2] rapidly make this impractical in large networks. As a result, forwarding tables usually keep only *active* addresses. Entries are deleted (according to some policy) either when the forwarding table is full and a new entry needs to be stored, or when their AGE exceeds a limit so as to remove stale entries.[1] The removal of entries creates the potential for *cache misses*, whenever a bridge receives a packet for a destination not present in its forwarding table.

When a cache miss occurs, a bridge broadcasts the packet on a logical spanning tree that bridges maintain by running the Spanning Tree Protocol [1]. Broadcast packets are transmitted on all ports associated with the links of the logical spanning tree, except for the port on which the packet was received. This ensures that the destination end-host DA, if present in the network, receives the packet. A reply packet from the destination results in the creation of a new $< \mathrm{DA}, \mathrm{PORT}, \mathrm{AGE} >$ entry in the forwarding tables of bridges on its path. This entry is then available to forward subsequent packets destined for DA.

For a given forwarding table size, as the number of MAC addresses in the network begins to exceed the number of entries in the forwarding table, a bridge will have to delete increasingly recent entries to make room for new active addresses. In general, as the number of addresses increases, so do the number of cache-misses. Ultimately, a bridge may enter a "thrashing" mode where entries are deleted before they can be used to forward the next packet to that address. As a result, the network experiences a growing steady-state fraction of broadcast traffic, which consumes network bandwidth and affects scalability.[2] This is hardly ever an issue in Ethernet's original deployment context—Local Area Networks (LANs) with a relatively small number of nodes and typically abundant bandwidth—and its convenience far outweighed this potential scaling deficiency.

Recently, there has been renewed interest in larger Ethernet deployments, hence stressing scalability, and spanning wider ar-

Saikat Ray is with the Department of Electrical Engineering, University of Bridgeport; Roch Guérin is with the Department of Electrical and Systems Engineering, University of Pennsylvania and Rute Sofia is with Siemens AG Corporation Technology, Information and Communications, Munich, Germany.

Research supported in part by a gift to the University of Pennsylvania by the Siemens AG Corporation Technology.

[1] Removal of stale entries is needed to avoid sending packets to the old location of a relocated MAC address.

[2] If there are n bridges, then the fractional increase in traffic is $O(n)$.

eas e.g., Metropolitan Area Networks (MAN), involving many more nodes, so that bandwidth is more expensive. Addressing the potential rise in broadcast traffic to mitigate its impact on scalability is therefore of interest. There have been proposals aimed at enhancing Ethernet and improving its performance in a MAN environment. We review some of them in Section II, but none fully address the scalability problem caused by more frequent broadcasts as the number of nodes (MAC addresses) increases as may be the case in an Ethernet MAN.

In this work, we develop a distributed mechanism which altogether avoids the use of broadcasts in cases of cache-misses. The mechanism allows bridges to rapidly "learn" the location of any MAC address, and hence the appropriate forwarding decisions, without resorting to broadcasts. This is achieved by storing the location of MAC addresses in a distributed hash table (DHT), with the ability to retrieve the location of an unknown address from the DHT using a single query. The scheme is scalable in the sense that the expected storage requirement at a bridge is "constant," independent of the network size. Further, the bulk of the storage requirement of the DHT scheme is on the "control path", which typically relies on cheaper and slower memory than data path forwarding tables. Unlike the proposal in [3], we do not eliminate the broadcast service of Ethernet; it is the unintended broadcasts of unicast packets that we reduce; a native broadcast/multicast packet is still forwarded throughout the network. Thus our scheme is backward-compatible. Note that the address location mechanism we use for MAC address resolution can easily be applied to eliminate reliance on broadcast in other address resolution schemes, e.g., ARP queries.

II. Related Work

Mitigating the impact of Ethernet's flat addresses on forwarding table size has until now been dealt with primarily by controlling the size the of the network itself. One such approach is through *Virtual LANs* (VLANs [4]), where an additional field in the packet header, the VLAN tag [5], is used to partition a single large Ethernet network into multiple networks; one for each VLAN. One disadvantage is that nodes in different VLANs cannot directly talk to each other. Another common technique is to restrict the network topology to *edge* bridges connected by the *core* bridges, where end-hosts connect only to edge bridges. Using so-called MAC-in-MAC (MiM) [6] encapsulation at edge-bridges, core bridges then only need to maintain forwarding entries to edge-bridges—a much smaller set; i.e., the burden of large forwarding tables is shifted to the edge-bridges. A more detailed discussion can be found in [5].

The above schemes, while important in practice, still rely on broadcast whenever a cache-miss occurs, and do not fundamentally improve Ethernet's scalability: they simply either limit the network size (VLANs), or shift the burden of having large tables to a subset of nodes (MiM)[3]. Moreover, they require extensive configuration, and hence move away from the plug-&-play design of Ethernet. Our solution, on the other hand, introduces no topological restriction, and by avoiding broadcasts in cases of cache-misses, fundamentally improves Ethernet's scalability while retaining its plug-&-play nature.

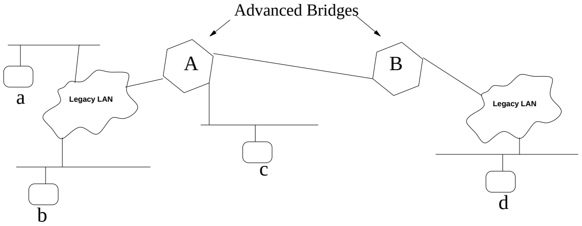


Fig. 1. A typical network composed of advanced and legacy bridges, shared medium segments and nodes.

III. System Overview

We start this section with a brief description of terminology. An *advanced bridge* is a bridge that implements the proposed distributed address resolution mechanism; a *legacy* bridge is a bridge that performs traditional broadcast based address resolution. A legacy *segment* is a part of the network (a connected subgraph) that does not contain any advanced bridge. Legacy segments operate according to the standard Ethernet protocol. Although advanced and legacy bridges work differently, it is critical that the former be *backward compatible* with the latter in the sense of all data packets being delivered correctly regardless of how advanced and legacy bridges are placed in the network. Thus we consider a network consisting of both advanced and legacy bridges connected in an arbitrary manner; e.g., the hybrid network shown in Fig. 1.

Next, we review packet forwarding operation in a hybrid network using as a reference the network of Fig. 1, which consists of three legacy segments connected by advanced bridges. Section IV discusses the exact mechanisms used in realizing this behavior. In steady-state, packets in such a hybrid network are forwarded as follows: If both the source and the destination nodes reside on the same legacy network segment, e.g., nodes a and b in Fig. 1, then packets follow the standard Ethernet forwarding procedures, i.e., along the spanning tree in the segment; advanced bridges do not get involved. If the source and destination nodes reside in different segments, e.g., nodes b and d, the packet is first delivered to the advanced bridge A that forwards it to the advanced bridge B, from where it finally reaches node d. To facilitate consistent forwarding among the legacy bridges in a legacy segment, when two or more advanced bridges attach to a legacy segment only one is allowed to send/receive packets to/from other segments; this advanced bridge is called the "designated advanced bridge" of the segment. (It is possible to allow multiple designated advanced bridges per legacy segment, but this introduces substantial added complexity to properly deal with all combinations of packet forwarding scenarios.) Note that such a selection can be readily accomplished following anyone from a number of standard "election" schemes, e.g., [7]. Upon receiving a packet originating on the legacy segment to which it is attached, the "ingress" designated advanced bridge must then determine the destination or "egress" advanced bridge connected to the (legacy) segment where the destination node resides. This

[3] Fundamentally, the scalability of MiM schemes are equally poor.

determination is a key step in the operation of advanced bridges, and as we shall see is performed without resorting to broadcast in most cases.

IV. Detailed Design

A. Backward Compatibility

Backward compatibility requires correct forwarding of data packets without changes to legacy bridges. There are many options for ensuring backward compatibility. We choose a simple mechanism, which relies on advanced bridges dropping all legacy control packets, i.e., the Bridge Protocol Data Units (BPDUs). In other words, advanced bridges behave like end-systems and do not participate in the spanning tree protocol. Thus, in a legacy segment a spanning tree is constructed that connects the legacy bridges, but does not extend across advanced bridges. This effectively allows advanced bridges to partition legacy segments. In fact, a legacy segment can now be defined as follows: if two legacy bridges A and B do not remain connected after all the advanced bridges are removed from the system, then A and B reside on two different legacy segments; else, they belong to the same legacy segment. By default, an end-host directly connected to an advanced bridge forms a legacy segment of its own. Thus the network is composed of legacy segments connected by advanced bridges. Section IV-C describes how this ensures backward compatibility.

B. Start-up

A boot-up or restart process in the network begins with legacy bridges within each affected segment forming connectivity among them by running the standard spanning tree protocol. Because advanced bridges do not forward BPDUs, each legacy segment forms a separate spanning tree "in parallel" and reasonably fast because of the relatively small size of individual segments. The bootstrapping of advanced bridges proceeds in parallel. However, notice that in order for advanced bridges to begin exchanging information, there must be connectivity between them, i.e., either through direct links or through legacy segments. In the latter case, since legacy bridges do not forward packets while computing the spanning tree (i.e., data packets are dropped during this period), intermediate legacy segments must have converged before data can be exchanged. Once connectivity is available, we assume for brevity that all advanced bridges become aware of each other after a short time, e.g., by means of announcements to a well-known multicast group ALL_ADVANCED_BRIDGES. This ensures that they are in a position to forward data to each other, e.g., by running a spanning tree protocol among them or through some other means that are beyond the scope of this paper.

C. Encapsulation

In order to ensure backward compatibility without imposing topological constraints, legacy bridges must be able to forward packets sent by advanced bridges. This applies to both packets destined to and originating from a legacy segment, as well as packets that need to transit (from one advanced bridge to another) through a legacy segment.

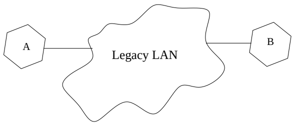
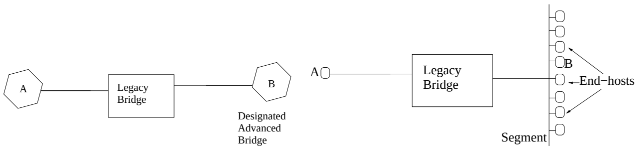


Fig. 2. View from an advanced bridge for encapsulated packets. Each legacy segment acts like a shared medium link.



(a) Actual topology. (b) View from the legacy bridge.

Fig. 3. View from the legacy bridge. It cannot distinguish between case 3(a) where a designated advanced bridge is connected to the port and case 3(b) where a shared media is connected to the port.

The latter is achieved by having advanced bridges rely on MAC-in-MAC (MiM) encapsulation [6], as illustrated in Fig. 2. Suppose that an advanced bridge A with MAC address α and an advanced bridge B with MAC address β are connected to the same legacy segment. In order to forward a (transit) packet to advanced bridge B, the advanced bridge A uses an encapsulation header with the source address and the destination addresses set to α and β, respectively. The legacy segment then forwards the packet from A to B using its standard forwarding mechanisms, and both advanced bridges view the legacy segment simply as a shared medium segment (cf. Fig. 2).

Packet exchanges between legacy segments occur through the segments' designated advanced bridges that encapsulate and decapsulate packets originating from and destined to the segments. In other words, the designated advanced bridge encapsulates packets originated from its legacy segment(s) and headed for "external" destinations, before forwarding them to the appropriate advanced bridge (see Section IV-D and Section IV-E for details on how this advanced bridge is identified). Conversely, the designated advance bridge of a legacy segment receives (encapsulated) packets from nodes outside the segment destined to nodes in the segment, and performs the necessary decapsulation before forwarding the packets on the segment for local forwarding and delivery.

To see how the proposed encapsulation/decapsulation method works seamlessly with legacy bridges, consider the scenario shown in Fig. 3. The legacy bridge is connected to two advanced bridges, A and B, and B is the designated advanced bridge for this legacy segment. Thus, the legacy bridge receives packets from all nodes outside this segment on this port since those are first sent to B, the designated bridge for the legacy segment, which decapsulates them before forwarding them onto the legacy segment. Thus, the legacy bridge views this port as connected to a shared medium segment on which all those nodes are attached. In other words, the legacy bridge cannot distinguish between packets originating from nodes directly attached to this shared medium segment, and (external) packets that transit through B.

The equivalence of these two views from the standpoint of a legacy bridge shows that the system remains backward compatible in the sense of delivering data packets correctly without changes to legacy bridges. Similar schemes are also used in RBridges [8].

D. MAC Address Learning and Registration

The essence of the proposed mechanism is that designated advanced bridges learn which MAC addresses they are responsible for, and *register* them with other advanced bridges to facilitate their subsequent retrieval. In particular, the mechanism relies on two different roles that a bridge can assume; namely, the *owner* and the *registrar* of MAC addresses. The owner of a MAC address is simply the advanced bridge that is the designated bridge for the legacy segment where the address resides. The registrar, on the other hand, is an advanced bridge where MAC addresses are registered together with the identity (address) of their owner bridge. Not all advanced bridges need to assume the role of a registrar; registration is done only on advanced bridges that agree to act as a registrar.

Each advanced bridge maintains 4 tables: a forwarding table in its data path, and a *peer* table, an *owning* table and a *registrar* table in its control path. A registrar bridge maintains another table in the control path, the *registration* table.

The forwarding table is similar to that of a legacy bridge; and populated either by observing transiting encapsulated as well as unencapsulated packets, or by making queries as described in Section IV-E. The peer table holds forwarding information for advanced bridges present in the network—i.e., the address and port to be used to send a packet to that advanced bridge. The owning table is populated by observing unencapsulated packets from the segment for which the advanced bridge is the designated bridge The registrar table holds the set of registrar bridges. Entries in all tables are subjected to an aging process. In general, the aging process used for the forwarding table may differ from that used for other tables.

An advanced bridge A registers each MAC address x in its owning table by sending a message to a specific registrar $R(x)$ identified through a universally known function $R(\cdot)$. Only one message needs to be sent for all MAC addresses with a common registrar. Upon receiving such a message, registrar $R(x)$ adds the association $< x, A, \text{AGE} >$ to its registration table.

$R(\cdot)$ depends on the IDs of the registrar bridges stored in the registrar table. For example, let $\mathscr{R}$ denotes the set of advanced bridges that have agreed to act as registrars and $\mathscr{X}$ the set of MAC addresses. Suppose M_i is the ID of registrar $AB_i(\in \mathscr{R})$, and $d : \mathscr{R} \times \mathscr{X} \to \mathbb{R}^+$ denotes a proximity function between an advanced bridge ID and a MAC address, x. (The proximity is *not* topological; it is in the space of ID's.) Finally, let $h : \mathscr{X} \to \mathscr{X}$ be a "scrambling" function. ($h(\cdot)$ is also a hash function, but we use the name "scrambling" function for $h(\cdot)$ to distinguish it from $R(\cdot)$—the universally known hash function.) Then an advanced bridge that owns a MAC address x registers it with the registrar whose ID is

$$R(x) = \operatorname*{arg\,min}_{AB_i \in \mathscr{R}} d(M_i, h(x)), \tag{1}$$

i.e., to the registrar whose ID is *closest* to the scrambled version of the MAC address to be registered. If the minimum is not unique, one of the minimizers is chosen in a deterministic fashion. Thus, the registrar of given a MAC address can be unambiguously identified. This particular hash function is *consistent* [9], which helps in fault localization. In particular, if a registrars fails, then only two other registrars with adjacent IDs are affected.

We note that in practice $\mathscr{R}$ could be configured based on some policy. Note also that we do not specify the exact form of the function $d(.,.)$; an optimal selection of this function is outside the scope of this paper. The scrambling function $h(\cdot)$ is included in the computation to ensure that MAC addresses are "assigned" to registrars as uniformly as possible. With a good scrambling function, MAC addresses are assigned to different registrars with uniform probability. In such a situation, each registrar stores, on average, N/M MAC addresses where N is the total number of MAC addresses present in the network and M is the number of registrars. This is comparable, again under the assumption of a uniform distribution of MAC addresses, to the size of a bridge's own owning table. In other words, when all advanced bridges are registrars, a registrar stores about twice the amount of storage it would need if it was not a registrar. More generally, when a fraction c of advanced bridges are registrars, the increase in storage at each registrar is about $2/c$.

The registrar maintains an age for each entry. Under normal operations, the owner advanced bridge regularly refreshes the registration of its active MAC addresses by sending refresh registration messages to the corresponding registrar bridges.

E. MAC Address Lookup

When a designated advanced bridge A receives a packet whose destination MAC address x is not in its forwarding table, instead of broadcasting the packet, it uses Eq. (1) to compute the ID of the registrar for that MAC address, say, B, and simply sends the packet (encapsulated) to B. Upon receiving this *query* packet and decapsulating it, the registrar performs a MAC address destination lookup in its registration table. If the address is present, it both forwards the packet (encapsulated) to the owning advanced bridge C of the destination MAC address and immediately sends a message to A notifying it that C is the owner advanced bridge of MAC address x. Designated bridge A then adds this new association to its forwarding table, and from this point onwards directly sends to advanced bridge C packets (encapsulated) destined for x. Note that this changes the forwarding decision from $A \to B \to C$ to $A \to C$. So there is a slight chance of packet reordering, which we assume to be small enough to be acceptable under normal scenarios.

Note that, unlike the schemes proposed in [10, 11] where a packet goes through potentially several nodes until it reaches one node with the knowledge of the destination, in our scheme the query is sent directly to the bridge that knows the location of the unknown MAC address. This is because we assume that each advanced bridge knows the *entire* set of registrars, $\mathscr{R}$ (stored in the registrar table). This is a reasonable assumption in our environment, but not when considering an Internet-scale network as is the case in [10, 11].

If the registrar does not find an entry for x in its registration table, e.g., the owner bridge C has not yet registered it or the entry was aged out, then it multicasts the packet to all advanced bridges, including the advanced bridge A that asked for the MAC address association in the first place. However, upon receiving this broadcast "reply", advanced bridge A will *not* redistribute it back on the local segment to which the source MAC address resides, hence avoiding packet duplicates on this segment. All other designated bridges will, however, decapsulate the broadcast packet and further broadcast it on their own legacy segments. This broadcast provides the necessary boot-up process to discover new destination nodes.

F. Correctness of Packet Forwarding

We review different possible packet forwarding scenarios and show that in each case the destination node receives one and only one copy of a packet. We consider three different cases: (i) both the source and the destination nodes reside on the same legacy segment, (ii) the source and the destination nodes are on different segments, but a single advanced bridge is the designated advanced bridge for both of them and (iii) the source and the destination nodes are on different segments with different designated advanced bridges.

1) Source and Destination on the Same Legacy Segment: Consider nodes a and b in Fig. 1. If the legacy bridges in the segment have an entry for MAC address b, the packet goes directly to node b. If one or more of the legacy bridges do not have an entry for b in their forwarding tables, then the packet might be broadcast on the legacy segment. In both cases node b receives the packet. Now note that whether or not the packet is broadcast, the designated advanced bridge A may see this packet as well. If A knows (according to its owning table) that it *owns* b, then it drops that packet since it then concludes that b and a both are residing on the same legacy segment. However, if A does not know about b, then it queries the appropriate registrar as mentioned in the previous section. Since A owns b, it is likely that b wont be registered at the identified registrar, and as a result the packet may be broadcast to the entire network and come back to A. However, since A knows at this point that a belongs to this legacy segment and a is the source of this packet, it does not deliver the packet back to this segment. Thus b does not get a duplicate packet.

2) Source and Destination on Different Segments Connected by a Single Advanced Bridge: Consider nodes b and c in Fig. 1. If A knows about c (from its owning or forwarding table), it simply forwards the packet to the appropriate port. However, if it does not know about c, then it again queries the corresponding registrar. This may either result in receiving a reply from the registrar that A itself is the owner of c, or as in the previous case the packet may come back to A as a broadcast packet. In both cases, A broadcasts the packet to every legacy segment where it is the designated bridge except the segment where the source, b, resides. Thus, node c receives one and only one copy of this packet.

3) Source and Destination on Different Segments Connected by Different Advanced Bridges: Consider nodes b and d in Fig. 1. If A knows (from its lookup table) that d resides on a legacy segment for which B is the designated advanced bridge, it sends the packet (encapsulated) to B, which in turn delivers the packet to d. The case where B does not know that it owns d is discussed with other similar "error" scenarios in Section IV-G. If A does not know about d, it again queries the corresponding registrar. As before, this results in either receiving a reply from the registrar that B is the owner of d, or getting a broadcast packet back from the registrar. In the first case, A updates its database and start directly forwarding to B packets destined to d. In the second case, A sends the packet to every legacy segment for which it is the designated bridge except the segment on which the source, b, resides. Since d is not in any of those segments, this does not result in the packet reaching its intended destination. However, B also gets the broadcast packet from the registrar and forwards it onto the legacy segments for which it is the designated advanced bridge. Hence, the target destination d ultimately receives one and only one copy of the packet.

G. Pathological Scenarios

In this section, we review a number of error scenarios that can arise because of failures or state mismatches between advanced bridges, e.g., differences in timers or aging policies used across the various tables maintained by advanced bridges.

1) Synchronization Mismatch: Since the owner and the registrar age table entries independently, it is possible that an entry is still stored at a registrar, but has been removed from the owner. In such cases, a registrar might redirect packets to a node that does not know how to deal with them. A similar scenario arises when an advanced bridge forwards a packet to another advanced bridge based on entries still cached in its forwarding table, while the owning bridge has aged out the corresponding entries from its owning (and forwarding) table.

We follow a simple rule to deal with such situations. An advanced bridge that receives an encapsulated packet carrying an ultimate destination address it does not know, forwards the packet onto the legacy segments for which it is the designated advanced bridge and sends out an error message to the originating advanced bridge. Upon receiving the error message, the originating advanced bridge deletes the entry from all its cache (if the entry still existed).

2) Unavailable Registrar: If a registrar node A becomes unavailable (i.e., the node is either down or disconnected from part or all of the network), then it is removed from the registrar tables of all the nodes that cannot reach A. The detection can happen in a number of different ways, e.g., when an advanced bridge does not receive a reply to a query, and triggers an update of the registrar table. This might in turn result in a broadcast message to all other bridges. Similarly, an owner can detect a down registrar after sending a registration request and take similar actions. A liveness mechanism between advanced bridges in the form of hello messages will typically also be used and provide another detection capability: if hellos are not received from an advanced bridge for some period of time, it is considered dead. Note that none of the active flows are affected by the demise of a registrar since their entries are already in the active cache of the corresponding advanced bridges.

After a registrar node is determined to be unavailable, the universally known function $R(\cdot)$ is modified by purging the dead registrar node from the set $\mathscr{R}$. During the time the function $R(\cdot)$ is being modified, or when the advanced bridges do not yet know that one of the registrars is down, packets may be sent to it, which will be dropped. Such occasional packet drops, while unavoidable, can be minimized with rapid detection and proactive notification of a down registrar node.

3) Unavailable Owner: If an owner advanced bridge dies or loses connectivity to a set of other advanced bridges, then all the end-nodes it owned become unreachable to those advanced bridges. Eventually all the corresponding entries in other bridges (advanced and legacy) will be deleted through the aging process. Alternately, any node that detects any (owner, registrar, or designated) advanced bridge as dead can proactively send a broadcast message that removes the entries from all the advanced bridges containing the dead node. In addition, advanced bridges connected to legacy segments for which the dead bridge was the designated advanced bridge trigger the election of a new one.

V. Performance Evaluation

This section compares the performances of the proposed and the legacy schemes. We show that the proposed scheme indeed significantly reduces broadcast load, and more importantly that its performance is relatively insensitive to various system parameters such as cache sizes and traffic patterns.

A. Simulation Environment

The simulations are conducted using an in-house packet level simulator. Each data point is obtained by running the simulator for 10 sec of simulation (virtual) time. Unless stated otherwise, the network consists of $M = 30$ bridges. All of them are advanced bridges in the DHT scenario and all are legacy bridges in the legacy scenario. In the DHT scenario, advanced bridges register the MAC addresses they own once every 100 ms. To reduce control traffic, only differential registration updates are sent. In both scenarios, the network boots up with empty tables (caches), which are populated as the simulation progresses. The *Least Recently Used* policy is used as the cache replacement strategy with each cache entry aged with 10 ms granularity. Cache entries are refreshed each time they are the target of an address lookup. The maximum age of the entries is more than 10 sec, the duration of each simulation, so that the results reflect cache misses due to cache-overwrites. Shortest path forwarding between the bridges is used in both cases for a fair comparison. Traffic is generated in the form of *sessions* between source and destination nodes. The session arrival process is taken to be Poisson with rate $\lambda = 5000$ (for the whole network), with source and destination MAC addresses selected in most cases with uniform probability among all addresses present in the network. In each session, the source generates n consecutive packets that are $\tau = 10$ ms apart and are acknowledged by the destination after 1 ms. Therefore, $2\lambda n$ packets are generated per second. Both the number of addresses, N, and the session size, n, are varied across simulations. These scenarios are not necessarily meant to emulate actual traffic patterns, but instead to allow us to systematically explore performance, and in particular sensitivity to broadcast traffic. The main metric of interest is the ratio of measured network load to generated network load that are in units of *link-usage*; i.e., the cumulative number of links traversed by packets. The measured network load indicates how many links have actually been traversed by a packet, whereas the generated network load indicates how many links *would have* been traversed if no address resolution was needed. For the DHT scenario, the network load also includes the overhead of sending periodic registration messages (only incremental changes are sent). The default size of the forwarding table is 100 and that of the registration and owning tables (in the DHT scenario) are 1000. A small forwarding table produces a realistic number of cache-misses with a reasonably low packet arrival rate and number of MAC addresses, allowing us to evaluate the relative merits of the schemes without the need for very long simulations. The effect of varying the forwarding table size is explored in Section V-D. Note that larger registration and owning table sizes are not unreasonable, since they reside on the control path where memory is slower and, thus, cheaper.

B. Reduction in Network Load

We first study the scalability of both schemes by varying N, the number of MAC addresses present in the system. The session size is set to $n = 1$, i.e., a random traffic pattern that is expected to exacerbate cache misses and stress the impact of address resolution broadcasts. This is demonstrated in Fig. 4: as N increases the impact of broadcast traffic grows rapidly in legacy systems, even if it eventually levels off (when every packet results in a cache miss). In contrast, the DHT scheme exhibits a much more progressive traffic increase because of its lower reliance on broadcasts.

C. Effect of Session Length

This section investigates the impact of the session length, n. We repeat the previous scenario, but with session length $n = 5$. i.e., now 5 consecutive packets spaced 10 ms apart constitute a session. The results are plotted in Figure 5(a), again as a function of the number of addresses, N. As expected, the longer session size helps reduce the broadcast load, but nevertheless we see that those benefits apply equally to both schemes (a reduction in load by a factor of approximately 2), with the DHT scheme still significantly superior.

To further explore the impact of session length, we simulate next a system with a fixed number of addresses, $N = 50,000$, but varying the session size, n, as shown in Fig. 5(b). The resulting behavior is interesting if not completely surprising. As n initially increases, the fraction of the network load contributed by cache miss related broadcasts decreases as expected, since

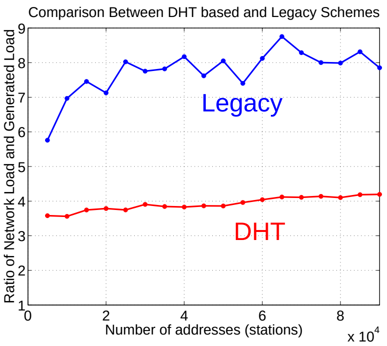
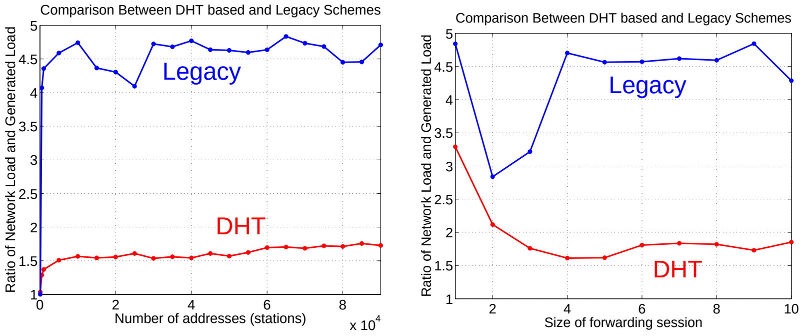


Fig. 4. Reduction in network load.



(a) (b)

Fig. 5. Effect of different session sizes.

the cache entry created by the first reply packet is now leveraged to avoid broadcasting subsequent packets in the session. This behavior, however, reverses as n increases further. For larger values of n, the fraction of traffic attributable to broadcast traffic starts rising again. The (conjectured) explanation for this behavior is that as n grows, so does the overall traffic intensity in the network, and in particular the average number of *active* sessions. This is because, the session arrival rate λ is kept fixed, while the session duration, as measured through n, increases. As a result, the number of active sessions increases with n, and eventually exceeds the capacity of the cache size so that entries created for new sessions actually push-out entries associated with still active sessions. This in turn triggers an increase in broadcast traffic, as each session now triggers more than one broadcast instance. Both DHT and legacy schemes exhibited this behavior, but again because of its lesser reliance on broadcast, the DHT scheme was much less sensitive to this phenomenon.

To confirm our findings, we repeated the experiments with a forwarding table of 200, and observed the same behavior except for a shift in the location of the inflection point that occurred at $n = 9$ instead of $n = 3$. We omit the plot due to space constraint.

D. Effect of Cache Size

We explore next the effect of different forwarding cache sizes, which we vary from 20 to 1200 with $N = 50,000$ and $n = 5$. The results are reported in Fig. 6 and show that performance improves for both schemes as the cache size increases, with a threshold beyond which further increases offer only limited benefits. The presence of a threshold in the legacy scenario is not unexpected as the following qualitative explanation shows. Let ρ be the (average) rate with which a new entry is pushed downwards in the cache of a bridge by the new entries. I.e., ρ is the rate of "new" (not already in the cache) packet arrivals at the bridge. Also let T be the time between when a cache entry is created by an ACK packet and when the next packet in the session arrives at the bridge. Clearly, the cache-entry made by the ACK packet will go downwards about ρT positions within this time. If the cache size is more than ρT, then the cache entry is retained when the next packet arrives and no cache-miss occurs. But when the cache size is reduced (just) below ρT, cache misses occur, each resulting into a broadcast, and thus increasing the network load. However, each broadcast itself also increases ρ since the packets that did not go to a bridge before now goes there, increasing the rate of new cache entry creation, and thus increasing ρ for that bridge. However, this increase in ρ in turn increases the rate of cache miss due to the increased difference between the cache size and ρT. Thus a positive feedback effect takes place due to which the fractional network load increases very rapidly. However, this increase in ρ is bounded above since a packet goes to at most all the bridges in the network. Past this point, as the cache size decreases further, the network load still increases due to the difference in ρT and the cache size, but it ultimately reaches a maximum (when almost every packet is broadcast) as the cache size is close to 0. Finally, note that the difference in fractional network load between the legacy and DHT scenarios in the large cache regime occurs due to the (possible) broadcast of the first packet of a session in the legacy scheme.

We confirmed our findings by repeating the simulation with $\lambda = 10000$. We observed the very same behavior, but the threshold shifted to 280, instead of 160. We omit the plot due to space constraint.

E. Effect of Number of Destinations

In the traffic model used so far, all end-hosts are potential destinations of the packets. We now explore the effect of having only a few end-hosts as destinations or traffic sinks. The simulation results are reported in Fig. 7. Here, the number of destinations, ν, is varied from 5 to 2500; $n = 5$, $N = 50,000$; and the size of the forwarding cache is set to 100. For very small ν—in particular for $\nu = 5$ and 25—the legacy scheme performs better. In this regime, the destination addresses are always present in the forwarding cache, and furthermore the low cache churning rate also ensures that the source address of a session's first packet can be entered in the cache and retained so that it is available for the returning ACK to use. Thus there is hardly any broadcast in this regime. Since the legacy scheme has no overhead other than broadcasts, it behaves better than the DHT based scheme. However, the legacy scheme quickly loses this advantage with increasing ν; $\nu = 50$ already behaves as when all end-hosts are potential destinations. In this regime, the cache already experiences some thrashing and a large number

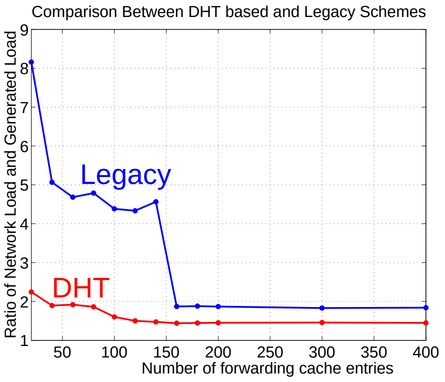


Fig. 6. Effect of Cache size.

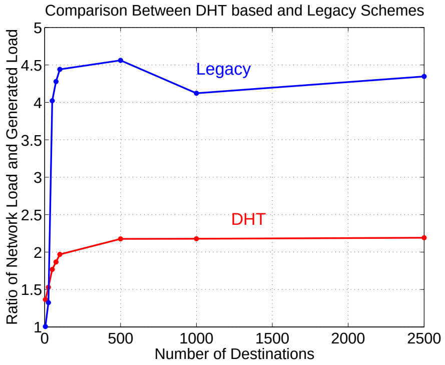


Fig. 7. Effect of number of destinations.

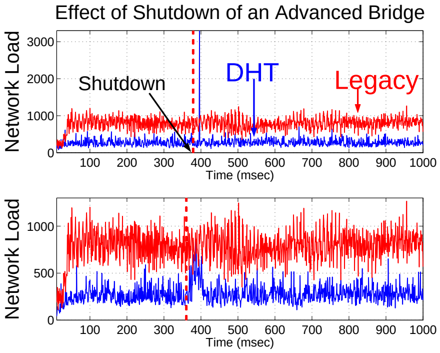


Fig. 8. Effect of an advanced bridge going down.

of packets are broadcast. The DHT based scheme, is again less sensitive to ν.

F. Effect of an Unreachable Advanced Bridge

As discussed in Section IV-G, there are several error conditions for the proposed mechanism. Due to lack of space, we only show the effect of an advanced bridge going down.

We run an advanced mode simulation with $N = 50000$, $n = 5$. After the simulation time has reached about 400 ms, we shutdown one of the bridges, A. As a result, all the packets sent to A are lost. In particular, query packets sent to A are not replied back. When a node B who sent such a query does not receive a reply before a timeout, it decides that the registrar is down. For simplicity, we use a “one strike out” rule; in practice usually more than one lost query will be used to arrive at such a decision. After deciding that the registrar bridge is down, bridge B sends a multicast *flush* message to all other advanced bridges. Bridge B and every bridge that receives this flush message remove A from their registrar tables and recompute the new registrar for the addresses that were registered to A. After that, new registration messages are sent out for those affected addresses. Since all these events, including the sending of new registration messages, occur within a very short time period, the network load surges when this happens. The corresponding simulation results are reported in the upper subplot of Fig. 8. In this figure, the surge occurs about 17 ms after A is shutdown and lasts for less than 1 ms. The initial delay depends on when a cache miss occurs for a destination address for which A was the registrar node.

The surge in the network load can be avoided by randomly delaying the flush message to other nodes or registration messages. However, the cost of such a random delay is that during this time some nodes still may send query packets (which contain data) to A that are lost. This trade-off is shown in Fig. 8, lower subplot. Here we used a random delay uniformly distributed between 0 ms to 20 ms. Clearly, the surge has reduced almost to the normal level. The period of high load now lasts for about 20 ms, as we expect. In this case, 6 query packets were lost compared to the earlier case where only one query packet was lost.

VI. Conclusion

This paper proposed and evaluated a simple address resolution mechanism aimed at reducing Ethernet’s reliance on broadcasts to resolve unknown addresses. The scheme is based on a simple adaptation of distributed hash tables that enables address resolutions using a single query. Advanced bridges are used to implement the scheme in a manner that remains backward compatible with legacy Ethernet. The mechanisms and protocol implemented at advanced bridges were outlined, and the performance of the scheme was evaluated along several metrics and compared to that of legacy bridges. The results showed that the approach was successful in substantially reducing broadcast induced network load in networks involving a large number of addresses. More importantly, it was robust across a broad range of configurations in terms of cache sizes and traffic patterns. There are clearly many missing details before a full specification is in place, and additional evaluations are in order, but the approach appears to offer a promising direction for extending Ethernet’s scalability while preserving its plug-&-play nature.

References

[1] *Media Access Control Bridges*, IEEE Std. 802.1D, 2004.

[2] K. Pagiamtzis and A. Sheikholeslami, “Content-addressable memory (CAM) circuits and architectures: A tutorial and survey,” *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, March 2006.

[3] A. Myers, T. E. Ng, and H. Zhang, “Rethinking the service model: Scaling ethernet to a million nodes,” in *ACM SIGCOMM Workshop on Hot Topics in Networking*, November 2004.

[4] *Virtual Bridged Local Area Networks*, IEEE Std. 802.1Q, 2003.

[5] M. Ali, G. Chiruvolu, and A. Ge, “Traffic engineering in metro ethernet,” *IEEE Network*, pp. 10–17, March/April 2005.

[6] *Provider Backbone Bridges*, IEEE Std. 802.1 ah, 2006, draft.

[7] J. Moy, “OSPF version 2,” Internet Engineering Task Force, RFC 2328, Apr. 1998.

[8] “Transparent interconnection of lots of links (trill).” [Online]. Available: http://www.postel.org/rbridge/

[9] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, “Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web,” in *Proc. STOC*, 1997.

[10] M. Caesar, M. Castro, E. B. Nitingale, G. O’Shea, and A. Rowstron, “Virtual ring routing: Network routing inspired by DHTs,” in *Proc. SIGCOMM*, September 2006.

[11] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker, “ROFL: Routing on flat labels,” in *Proc. SIGCOMM*, September 2006.