Addressing the Scalability of Ethernet with MOOSE

Malcolm Scott, Daniel Wagner-Hall, Andrew Moore and Jon Crowcroft
University of Cambridge Computer Laboratory
{Malcolm.Scott, daw63, Andrew.Moore, Jon.Crowcroft}@cl.cam.ac.uk

Abstract

Ethernet does not scale well to large networks. The flat MAC address space, whilst having obvious benefits for the user and administrator, is the primary cause of this poor scalability; other recent efforts to improve upon Ethernet's scalability have addressed symptoms, rather than this underlying cause. In this paper we present MOOSE, Multi-level Origin-Organised Scalable Ethernet, an Ethernet switch architecture that performs in-place rewriting of MAC addresses in order to impose a hierarchy upon the address space without reconfiguration or modification of connected devices. This removes the need for switches to maintain large forwarding databases, is of direct use in implementing improved routing, and allows for a variety of other scalability and security innovations. We also present a globally-scalable, distributed and resilient protocol for the automatic assignment of addresses to switches, and for detecting and cheaply resolving addressing conflicts.

1 Introduction

Ethernet has lasted well since its inception in the '70s [1] with Ethernet frame-structure and addressing remaining ubiquitous in the data centre environment as in many others. Alongside IP and IP-transported services such as iSCSI, it is now commonplace to see converged network services such as physical disk interfaces and cluster interconnects layered directly over Ethernet (e.g. ATA-over-Ethernet and variants of Infiniband). However, Ethernet exhibits scalability issues on networks of more than a few thousand devices, such as costly and energy-dense address table logic and storms of broadcast traffic.

Aside from more physical devices, virtualised infrastructure further increases the density of Ethernet addresses in data centres. Widely-used layer-2 virtualisation mandates a unique Ethernet address per virtual machine [2]. This means that each physical machine in a data centre may represent many tens of Ethernet devices.

The traditional method of avoiding such problems is the artificial subdivision of a network, but this introduces an administrative burden, requires significant routing equipment and also precludes seamless migration—a necessity for virtualised infrastructure [3]. While IP Mobility [4] addresses the problem of maintaining higher-layer connections when roaming between subnets, it requires client support that is neither ubiquitous or reliable. Common practice sees the provision of one physical Ethernet network covering an entire data centre, or even an entire WAN of data centres.

Our approach, Multi-level Origin-Organised Scalable Ethernet (MOOSE), provides all the advantages of an Ethernet network without the capital and running costs and administrative overhead of a IP router-based approach. MOOSE does this by providing a hierarchical addressing scheme without requiring host reconfiguration or modification.

Ethernet's scalability is limited firstly by the forwarding database that every switch in an Ethernet network must maintain [5, §7.8–7.9]. A switch's forwarding database contains one entry per source address seen in any frame passing through that switch, and stores that MAC address together with the learnt location of that address—the port on which packets from that address were last seen. This is later used to determine on which port to transmit frames destined for that address. Devices frequently broadcast frames throughout the network (e.g. ARP queries) so active devices on the network are listed in most switches' forwarding databases most of the time.

In modern switches the capacity of this database is generally of the order of 16,000 entries [6]. (Higher-capacity forwarding databases exist but are currently constrained to very high-end switches.) On a moderately large network, full databases are a serious risk. If the database becomes full, entries will be discarded; frames for unknown addresses are flooded to all ports and the resulting traffic storm could cause major problems, especially in the presence of low-capacity edge links.

Traditionally the forwarding database has been stored in a content-addressable memory (CAM) as lookups must be very fast, particularly as 10 Gbit/s Ethernet becomes ubiquitous. As networks grow, the number of entries in a switch's forwarding database must naturally increase; however, increasing the capacity of CAMs without sacrificing speed whilst constraining energy con-

sumption is proving to be challenging [7, 8]. Cheaper switches use DRAM in place of a CAM, but this is likely to remain slower especially for large tables.

Secondly, Ethernet's inability to handle networks containing loops also presents a scalability problem. The Rapid Spanning Tree Protocol, RSTP [5, §17], must remove loops by disabling any redundant links. On a dense mesh network, RSTP will disable a large proportion of links; this constrains frames to suboptimal routes and may introduce bottlenecks in the network, particularly around the root of the spanning tree. In a data centre environment, this potentially amounts to a very large proportion of capacity being wasted wherever redundant fibres are installed, e.g. between cabinet switches and between data centres.

Thirdly, not only does Ethernet flood frames destined for unknown hosts, but it also uses—and encourages higher-layer protocols to use—broadcast for control messages. For example, ARP [9] performs address resolution via broadcast queries, and DHCP [10] uses broadcast messages for automatic configuration. It is impractical to replace these protocols entirely as this would require software upgrades to every device, but it would be desirable for the network to minimise the amount of broadcast traffic required to be forwarded.

In this paper we identify the relevant underlying problems in the design of Ethernet (section 2), review previous work (section 3) and present the MOOSE switch architecture, which addresses inadequacies in the fundamental operation of Ethernet in a novel yet backwards-compatible way (section 4). By revisiting the addressing scheme itself, rather than simply addressing symptoms of the problem as many previous proposed solutions have done, we can go about solving all of the above scalability problems and more.

A working high-level software implementation of MOOSE is described and evaluated in section 6.

This work expands on our previous paper presented at the First Workshop on Data Center – Converged and Virtual Ethernet Switching (DC CAVES) [11]. We have added a crucial piece of the architecture—an automatic addressing scheme with cheap conflict resolution—and have better addressed the key issue of compatibility with existing protocols.

2 Ethernet's Underlying Problem

The original Ethernet was a shared-medium network, where every frame was broadcast and no switching took place. Modern-day wired Ethernet-based networks instead consist almost entirely of point-to-point links; as a result of this, the distinction between unicast, broadcast and multicast has become more important. 802.11 wireless LANs are the one remaining vestige of Ethernet operating over shared media, where one switch (access point) serves many hosts on the same radio channel.

Ethernet's poor scalability arises in various guises, as outlined in section 1. It would seem at first glance that these are entirely distinct and unrelated. However, there is a common underlying cause: that ***MAC addresses provide no location information***.

Globally-unique MAC addresses are structured such that the first three bytes of a device's address contain an organisationally unique identifier (OUI) allocated to the device's manufacturer by the IEEE, with the remaining three bytes allocated by the manufacturer. This hierarchy exists solely for the purpose of allocating unique addresses in a decentralised fashion, and is of no use to Ethernet switches, which must treat the unicast address space as flat.

A flat address space has the advantage that no configuration of devices is required; a device can use its unique, manufacturer-assigned MAC address anywhere on any network. However, this leaves each switch with the task of discovering and storing the location of every addressable device.

If the MAC address space were not flat, but instead contained enough information to locate the device possessing the address, several advantages would be gained. Firstly, large forwarding databases would no longer have to be maintained on every switch. This location information could instead be distributed across the network so that frames are directed towards their destinations according to successive stages of a hierarchy.

Secondly, a hierarchical MAC address space would also make the addition of shortest-path routing considerably easier. Shortest-path routing is clearly a desirable property for a network, yet it is one that Ethernet does not provide. Flat addressing does not lend itself to easy routing: any address can be located anywhere on the network, which means either advertising every host's MAC address via the routing protocol—which scales very poorly—or providing some other location lookup service. The use of hierarchical addresses, with each switch handling a block of sequential addresses akin to an IP subnet, would reduce the routing problem to the one that routing protocols were designed to solve.

Thirdly, this would allow for reduction of broadcast traffic in a variety of different ways. Hierarchical MAC addresses could, for example, be mapped directly and deterministically onto the IP address space, if appropriate for the specific deployment. This would allow switches to respond directly and simply to DHCP and ARP queries, avoiding the need to forward the most common sources of broadcast frames. Alternatively, a

distributed directory service can be used, which is less limiting and is thus our preferred approach as detailed in section 4.5.

The facility for network administrators to assign locally administered addresses (LAAs) to devices has existed for as long as Ethernet. However, configuring and maintaining the LAA on every device based upon where they are connected would be a considerable and unwelcome administrative overhead. In this paper we therefore present MOOSE, a system for applying hierarchical addressing to an Ethernet transparently and without any configuration to edge devices.

3 Related Work

It is well-known that traditional Ethernet scales poorly, and there have been various attempts in recent years to rectify this. The most widely-used of these in real-world networks is MPLS-VPLS (Multiprotocol Label Switching—Virtual Private LAN Service) [12]. This connects Ethernet islands together through tunnels across a MPLS cloud. MPLS works by adding one or more labels to the start of every frame, i.e. encapsulating the frame inside its own protocol.

In MPLS-VPLS, the label edge routers (LERs) must determine the frame's initial label(s) based upon the destination address via a lookup table. Frames follow prenegotiated label-switched paths (LSPs) that, unlike Ethernet, are not constrained to follow a spanning tree; LSPs are precomputed at connection setup time and the relevant next hop is stored in a lookup table on each intermediate switch. Each switch must hence use each frame's label to index into this lookup table to determine how to switch the frame.

The effect, once the connection has been negotiated, is to provide what appears to be one or more large Ethernet networks, transparently overlaid on the MPLS cloud. Whilst this solves effectively the problem of shortest-path routing across the MPLS cloud, the overlay Ethernets are still susceptible to the usual scalability problems—and in fact VPLS adds further large lookup tables on every switch that can in some configurations scale even worse than Ethernet's forwarding databases. LERs must map every MAC address to a LSP; label switch routers (LSRs) must store the next hop for every LSP in which they participate, which in the core of the network could scale as $O(\text{hosts}^2)$.

A similar scheme is proposed by Hadžić [13], with the difference that Ethernet-inside-Ethernet encapsulation is used rather than a new protocol. This has the advantage that less processing is required on intermediate switches in the backbone network. However, routes across the backbone are constrained to a spanning tree, and encapsulating switches must obtain a new destination address for every frame using a lookup table that—like Ethernet's forwarding database—must contain every transmitting MAC address. Due to its heavy basis on Ethernet, this shares many of Ethernet's scalability problems.

SmartBridge [14] and Rbridges [15] both encapsulate Ethernet frames in a new inter-switch protocol, and run a link-state routing protocol between switches. The link state graph includes the location of every MAC address—necessary because the address space remains flat and any address could appear anywhere—i.e. it again contains every host. Furthermore, switches must perform expensive computation to update routing tables whenever a MAC address joins or leaves the network.

Myers *et al.* [16] suggest that Ethernet's main failing is its broadcast service, and propose a new architecture in which hosts make explicit use of directory services operated by switches rather than broadcasting queries. It is clear that switches' participation is necessary in order to deal with the broadcast problem; however the modifications to Ethernet suggested are not backwards-compatible and would require at least software modifications to all connected devices. Ethernet is, perhaps unfortunately, too widespread for this to be practical; transparent interception of broadcast frames and subsequent local handling or redirection via multicast or unicast remains the only practical solution. The use of hierarchical addressing is a useful stepping-stone to such a system, and our architecture includes a transparent directory service (ELK, section 4.5) for this purpose.

SEATTLE [17] takes a more scalable approach. A routing protocol is operated between switches, but in contrast to the approaches described above and in common with MOOSE, the routing protocol only propagates switch location information, rather than every MAC address on the network. Flat MAC addresses are still used, and hence a mechanism is required to look up the switch to which a given address is connected. This is achieved by using a distributed hash table (DHT) operating on participating switches with local caching to alleviate load. This is certainly a step in the right direction but introduces considerable complexity to switches, since they now must maintain and update the DHT continually, and it is clear that a SEATTLE switch would have a significant software component in the data path. MOOSE alleviates some of the complexity of SEATTLE by a combination of hierarchical addresses and delegation to a separate directory service.

4 MOOSE Architecture

The basic operation of MOOSE is to assign a new hierarchical MAC address to each host on the network, as-

signed dynamically and automatically from the unicast LAA space. This dynamically-assigned address is referred to as a *MOOSE address* to avoid confusion with hosts' static, manufacturer-assigned MAC addresses.

Every frame entering the network has its source address rewritten in-place to the sending host's MOOSE address by the first MOOSE-aware switch it traverses; the new source address becomes the sending host's MOOSE address. The switch that performs address rewriting for a host—i.e. the closest MOOSE switch to that host—is the host's *home switch* and is responsible for assigning a MOOSE address to that host. (If non-MOOSE switches or hubs are in use, a host may have more than one "closest" MOOSE switch, in which case an RSTP-like protocol must be used to elect a switch to handle each edge segment.)

The destination address is left intact in the expectation that it already is a MOOSE address. Hosts' ARP caches will already contain the MOOSE addresses of any hosts being communicated with as any packet received will already have had its source address rewritten; a host's manufacturer-assigned MAC address is never seen outside of the segment containing that host. This is a crucial point since encapsulation-based technologies such as MPLS do not reveal to the destination host the address used for routing; as a result, switches must also convert destination as well as source addresses of frames entering the network. In other words, once again switches must maintain large tables of remote hosts on the network. The only destination rewriting that MOOSE switches perform, however, is of the destination addresses of frames destined for local hosts back to their manufacturer-assigned MAC addresses; this is simple as the required information is already known, and necessary because otherwise that host's network interface card would discard the frame as misaddressed.

A MOOSE address consists of a *switch identifier* followed by a *host identifier*. For our examples, we simply use a fixed three-byte switch identifier followed by a fixed three-byte host identifier, as illustrated in figure 1. Since these two identifiers when concatenated must form a unicast LAA, the settings of two bits in the first byte of the switch identifier are fixed: the least significant bit must be 0 to indicate a unicast address, and the second-least significant bit must be 1 to indicate a LAA. To cater for variable length switch identifiers, some means of introducing separation between the switch and host identifiers is required. Two possible implementations would be for:

- the first three bits of the address indicate how many of the following 5-bit blocks make up the switch prefix;

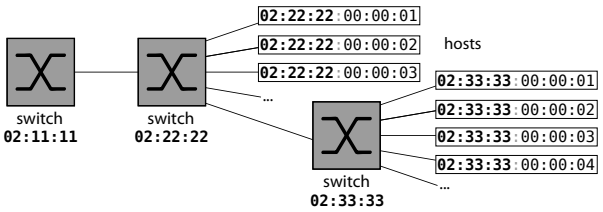

Figure 1: Assignment of MOOSE addresses by switches

- some constant delimiter to appear between the switch identifier and host identifier, with switch identifiers not allowed to contain the delimiter.

The former is simple and gives 8 classes of switch identifier. Because the size of a MOOSE network is limited by the placement of IP routers, these classes should be sufficient. Additionally, because switches are free to change their identifier, they may trivially switch to a larger class if they have too many attached hosts, or a smaller class becomes full.

The latter removes the fixed classes, allowing for more flexibility with the sizes of switch identifiers, at the cost of complexity, and a reduction in the available address space.

Each switch can select for itself a unique switch identifier, as identifier conflict resolution is cheap (section 4.2). When first joining the routing protocol (section 4.1), conflict should be very unlikely, as the switch will in the process gain an up-to-date list of in-use identifiers. Depending on requirements, the switch identifier may itself be a hierarchical address—e.g. six bits to identify a network area followed by two bytes to identify a switch within that area—which could then be used to aid routing decisions.

Each host is assigned a host identifier by its home switch from the pool of identifiers available to that switch. Only a host's home switch ever bases a switching decision on the host identifier, so the detail of how these are allocated can vary from switch to switch. Suitable schemes include:

- sequential assignment;
- the port number followed by a sequential portion (to allow for multiple hosts connected to one port);
- a hash of the host's real MAC address.

The latter two approaches are preferable to a simple sequential assignment, as they better isolate certain kinds of denial-of-service attack in which a malicious host attempts to use up all available host identifiers on the switch. They also require less state to be shared between ports. The third option has the further advantage that it is deterministic and hence can be recovered easily in the event of a crash.

It is hence possible to route frames through the net-

work to remote hosts by simply inspecting the switch identifier in the destination address, and ignoring the host identifier until the frame reaches the destination host's home switch. Switches no longer need to keep a table of all MAC addresses seen recently; they only need store the locations of other switches and of any directly-connected hosts.

As well as reducing the amount of data that must be consulted in order to make switching decisions, this provides extra resilience by making this data much more predictable. The number of MAC addresses in a network can increase unexpectedly in the event of an address flooding attack [18] or even under normal operation if the network contains open wireless access points; relying on the MAC address list for forwarding leads to some of the vulnerabilities of Ethernet. The set of switch identifiers participating in MOOSE switching, on the other hand, is kept predictable and manageable by ensuring that neighbouring switches (discovered using LLDP [19]) are authenticated before they can participate in the routing protocol. This authentication can be achieved at layer 3 using the security features found in most popular routing protocols and/or at layer 2 using 802.1X [20]. As the switch identifier is the only address consulted for forwarding decisions, a MOOSE switch is likely to remain reliable in the face of attacks that could have brought down a traditional Ethernet. Furthermore, any attacks based upon MAC address spoofing cannot function on a MOOSE network as the user-provided MAC address is translated immediately.

4.1 Shortest Path Routing

As described so far, MOOSE switches must still forward frames along a spanning tree. As discussed in section 1, this is an undesirable property of Ethernet as it can cause frames to take a highly suboptimal path through the network. The foundations are in place to do much better than this using shortest-path routing.

For the purpose of frame forwarding, a MOOSE switch can be considered akin to a layer 3 router; it has one locally-connected subnet—containing all addresses starting with its switch identifier—and delivers frames to other subnets by passing them to an appropriate neighbouring switch. Bearing this in mind, the switch can run a routing protocol of the kind normally used for IP, such as a variant of OSPF [21]. This allows frames to be routed along the shortest available path, rather than being constrained to a spanning tree. OSPF-OMP [22] may be particularly desirable due to its ability to make use of multiple equal-cost routing paths in order to improve performance [23].

4.2 Address Selection and Conflict Resolution

For reasons akin to those of the flaws of Ethernet, it is undesirable to guarantee universally unique predetermined MOOSE switch identifiers. Due to the reduced size of the switch ID space compared to the MAC address space, this would also be infeasible. We therefore propose that each switch selects an initial address for itself during startup. This could result in more than one switch claiming an address, which would be undesirable, so to mitigate the potential for MOOSE addresses to find themselves in conflict we additionally propose a simple and inexpensive conflict resolution protocol.

Suppose two switches each have the same identifer. We note that if these switches are on separate MOOSE networks (on disconnected networks, or separated by an IP router), this situation brings no issue. Should they be on the same MOOSE network, however, a conflict exists and must be resolved. Any routing protocol would require a switch to know which port other switches are connected to, for instance by OSPF neighbour lists, or simply by receiving frames and noting the switch port and source MOOSE address. When a switch receives a MOOSE frame, it looks up the source switch in its forwarding database, which is likely in fast Content Addressable Memory. If it finds that source switch to be on a port other than that which it recognises from its table, one of three situations may be possible:

- the source switch may be the same as the known switch, and have physically moved, or a topology change has occurred;
- the source switch may be a different one to the known switch, and they are in conflict;
- the source switch may be the same as the known switch, but is sending frames down a different route to the last used route.

To avoid disruption to the network in the first case, and to give scope for switches to migrate within the network, the switch which detected the possible conflict should ascertain whether the known switch is still alive and present. The conflict-resolving switch thus attempts to send a unicast frame to the known switch, via the port stored in the forwarding database, asking whether it is there at a regular interval until a timeout. This will reach the known switch rather than the new switch if it is still present as other switches beyond that port must not have detected the conflict yet. The nature of the timeout we leave unspecified, and can be implementation specific. It may, for instance, be a pre-defined constant, or it may vary based on QoS information gathered if such capabilities are supported. When a MOOSE switch receives

such a frame, it should promptly respond with an acknowledgement frame, showing that it is alive.

If, within the timeout period, the conflict resolver finds the known host not to be alive, no conflict exists, so the switch updates its view of the network by removing the old entry from its forwarding database and triggering a routing protocol refresh.

If, on the other hand, the host is found to be alive, a conflict exists. The conflict resolver then sends a frame to the more recently found switch indicating that it is in conflict and should change its address. That switch, upon receiving this frame, changes its address and sends a gratuitous ARP for each of its connected hosts, so that the rest of the network is aware of the change. To mitigate the risks of a denial of service attack, or faulty equipment sending out conflict frames, an exponential backoff algorithm should be used when receiving conflict notification frames.

A switch should have a timer, and counter influencing the maximum value of the timer, both initialised to 0. When a conflict notification frame is received, the counter is incremented (subject to a saturation value to avoid excessive timeouts). After a conflict has been resolved—i.e. the switch has changed its address—a timer starts counting down from some time exponential in that counter; subsequently the switch will only change its address if the timer has returned to 0 by the time the conflict frame is received. The counter should be reset to 0 when the timer reaches 0. Using this scheme the event of true conflict is handled quickly, even in the unlikely case that the newly acquired address is also in conflict. Any node emitting malicious or erroneous conflict notifications, however, is rate-limited enough that their damage potential is much restricted, subject to a sufficient timer being chosen.

Algorithm 1 Conflict resolution backoff

if $timer > 0$ **then**
  if $counter < counter\_max$ **then**
    $counter = counter + 1$;
  end
  // Discard conflict notification frame
else
  $timer = k^{counter}$;
  change_address();
end

This could be further enhanced by detecting repeated conflicts involving the same switch or switches, in a manner similar to BGP Route Flap Damping [24], and performing more aggressive steps to avoid further conflicts—for example using a significantly increased timeout, and/or having *both* switches in conflict select new addresses.

Algorithm 2 Conflict resolution timer

foreach *clock tick* **do**
  if $timer > 0$ **then**
    $timer = timer - 1$;
  else
    $counter = 0$;
  end
end

The conflict resolution algorithm brings a marked improvement on the equivilent vulnerability of Ethernet, that MAC addresses can be spoofed. We build in a flexible, well-defined system of recovery. The decentralised nature of the system makes it much less open to denial of service attack than any centralised directory may be. Having every MOOSE switch acting as a barrier to the propagation of packets from addresses in conflict provides a strong separation between recently bridged networks with conflicting addresses, so that communication within the individual networks may continue without modification, until bridge-crossing traffic appears, at which point resolution quickly happens. We also remove the possibility for forwarding databases to frequenty have to switch their entry for a conflicted address, which can happen with MAC conflicts in traditional Ethernet. Additionally, in the case of a switch identifier spoofing attack, the conflict resolver acts as a hard boundary for the effects of such an attack.

It is possible that the switch performing conflict resolution could send a suggested replacement switch address to the switch in conflict, known by the conflict resolver to have a low probability of being present on the network (because it is not present in its forwarding database). This would reduce the chance of repeated collisions, and potentially allow for longer backoff periods, but may be premature optimisation.

Because multi-path routing is often desirable, we could introduce an extra datum during the source address rewriting performed by MOOSE switches. When an ingress MOOSE switch rewrites the source address of an Ethernet frame to a MOOSE address, it could also prepend some hash of its manufacturer-assigned MAC address to the data field, and increment the length field as necessary. The egress switch, when rewriting the MOOSE destination address to a host's MAC address, then strips out this added datum. This allows the conflict resolver to check whether conflicts actually exist by local lookup, rather than probing other switches, at the cost of added memory requirements in every switch. This

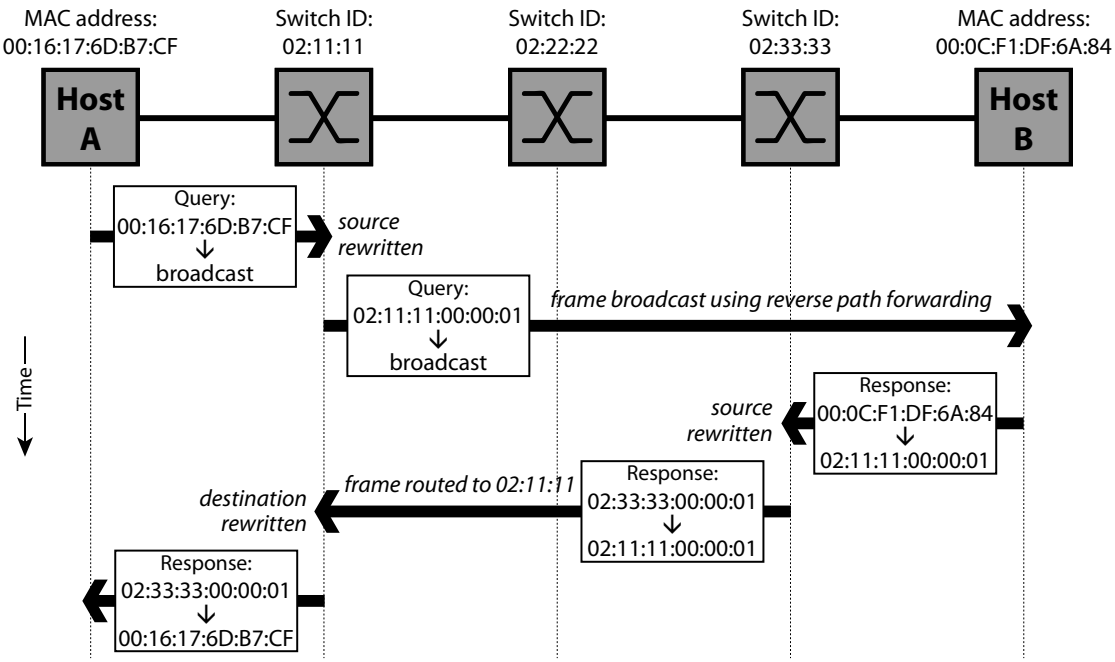

Figure 2: Sequence diagram of a broadcast query and subsequent unicast response

may push the frame to be larger than Ethernet's maximum, so may require fragmenting the packet into two, at small added cost. Alternatively, assuming jumbo frames are permitted by the hardware, the maximum frame size could be marginally reduced to allow for this in the same manner as for 802.1Q VLAN tags.

From the cheapness of conflict resolution, certain other address management tasks become simple. A switch is free to choose its address when it joins the network however it wishes—attempting to re-use its last-used address, from a list of preferred addresses, or by generating an address entirely at random. More intricate addressing schemes may be used on managed networks if desired, perhaps encapsulating deeper layers of hierarchy.

4.3 Broadcast and Multicast

Since Ethernet does still need to support arbitrary broadcast frames, these must still be forwarded along a spanning tree in order that they reach each host exactly once. An explicit spanning tree protocol is not required however, as the tree can be deduced from the routing table via reverse path forwarding in a similar manner to Protocol-Independent Multicast (PIM) [25]. In other words, broadcast packets are routed as if they had been sent to the all-hosts multicast group.

More general multicast groups can be implemented using a combination of IGMP snooping [26] as used by modern Ethernet switches, and participation of the MOOSE switches in PIM routing.

4.4 Example

To illustrate the basic behaviour of MOOSE switches, before we go on to describe further features, we will offer a simple example. We will describe the steps involved in forwarding a broadcast frame containing a query in some higher-layer IPv4-based protocol, and subsequent unicast frame containing the response, between two hosts A and B via three MOOSE switches 02:11:11, 02:22:22 and 02:33:33; see figure 2.

4.4.1 Query

i) Host A transmits the broadcast query frame as it would on any Ethernet network, with its own manufacturer-assigned MAC address in the Ethernet header's source field and the broadcast address (FF:FF:FF:FF:FF:FF) in the destination field.

ii) The frame is received by switch 02:11:11, which observes the non-MOOSE address in the frame's source field, and rewrites the source field into a MOOSE address containing the switch identifier and the appropriate host identifier. As this is Host A's first frame, the switch must allocate a host identifier (in this case 00:00:01, making Host A's complete MOOSE address 02:11:11:00:00:01).

iii) The three switches broadcast the frame using reverse path forwarding away from Host A.

iv) The frame is received by Host B (and any other hosts on the network) in its current form; no further rewriting is performed.

4.4.2 Response

i) Host B looks up Host A's IP address in its ARP cache to determine a suitable destination address for the response frame. Since the rewritten query frame arrived at Host B with the source field containing the MOOSE address 02:11:11:00:00:01, this is the address returned by the cache lookup.

ii) As above, switch 02:33:33 assigns a MOOSE address to Host B (02:33:33:00:00:01) and rewrites the source address of the frame.

iii) The frame is now routed through the network based solely on the destination switch identifier—the host identifier is ignored for now. The routing table is consulted for the location of switch 02:11:11 and the frame is forwarded accordingly.

iv) On receiving the frame, switch 02:11:11 observes that it is destined for a host directly connected to itself (02:11:11:00:00:01). It prepares the frame for transmission along its final hop by rewriting the destination address to Host A's manufacturer-assigned MAC address. The source field of the frame is again left as the MOOSE address of Host B in order that this address is used for any further communication with Host B.

4.5 Directory Service

A directory service, Enhanced Lookup (ELK), runs in conjunction with the basic MOOSE switch described so far. ELK exists to handle ARP and DHCP queries in a broadcast-free manner by learning mappings from IP addresses to MOOSE addresses. The master ELK directory is served by one or multiple systems for resilience and is reached using an anycast MOOSE address; the layer-2 anycast feature is a convenient side-effect of running a routing protocol. Slave copies of the directory can be held nearer the edge of the network in order to take load away from the masters; slaves can be reached for lookups via a separate anycast address, and the entire herd of ELK can be kept synchronised via the masters using a combination of multicast and unicast.

MOOSE switches intercept ARP and DHCP packets broadcast by hosts and convert them into anycast ELK queries to the nearest slave (for ARP) or master (for DHCP). (DHCP handling could make use of the protocol's existing DHCP relay mechanism.) The ELK slave answers ARP queries directly using information in the directory; as it does so, if the query is from a host not in the directory, it learns the sender's IP address to MOOSE address mapping. The ELK master can also act as a DHCP server, populating the ELK directory as it grants IP address leases to clients.

The one case in which the ELK directory will not contain the answer to a query is when answering an ARP request for a host that is not configured to use DHCP and that has not yet itself sent an ARP packet (i.e. has not yet communicated via IP). This must be dealt with by flooding the query to every active switch port, in a manner akin to current Ethernet switches, and caching the result in the ELK directory. Although this is not ideal, it is necessary in order to deal with this scenario in a compatible manner, and is unlikely to happen frequently.

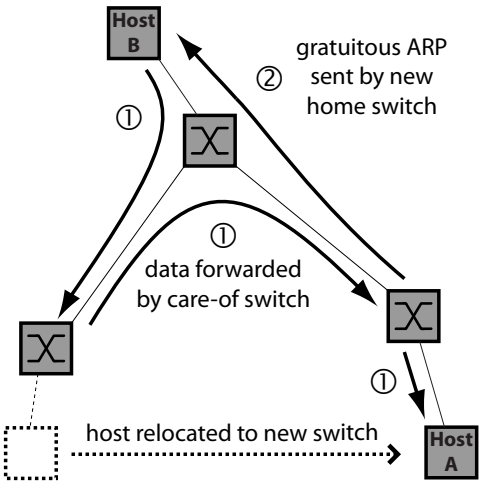

Figure 3: Two ways to handle a host A roaming onto another switch whilst maintaining communication with another host B

4.6 Mobility

A consequence of introducing location-based hierarchy into MAC addresses is the need to explicitly handle host mobility. In a traditional Ethernet, hosts can migrate between switches as the switches will learn the host's new location as soon as it sends a frame. With MOOSE, if a host relocates to a new switch its address changes and any ARP cache entries on other hosts pertaining to the migrated host become incorrect; frames will continue to be sent to the host's old location for a while. There are two strategies for dealing with this, as illustrated in figure 3, which can be used separately or in conjunction:

i) The previous home switch of the migrated host can forward frames sent to the host's old address until outdated ARP cache entries expire. This is similar to IP Mobility [4]: the previous home switch essentially becomes a care-of agent for the host. However, unlike IP Mobility, it requires no host support. A handover protocol is necessary for the old and new home switches to set up such forwarding: on the arrival of a new host at a switch, that switch would ask all other switches (via multicast) whether any had seen this host before, identifying it using its manufacturer-assigned MAC address, and would instruct such switches to redirect frames.

ii) A broadcast ARP announcement (or "gratuitous ARP") can be sent by the new home switch to immediately update remote ARP caches and the ELK directory with the new MOOSE address. This is the technique used by Xen when migrating live virtual machines [3]. Unlike the previous approach, this works even if the previous switch is no longer reachable, for example if this host migration was as a result of a switch failure. This is a simpler approach as a handover protocol is not required, but results in additional broadcast traffic.

Unless the frequency of host migrations is very high, the additional load introduced by either mobility approach is expected to be negligible.

5 Interoperability Considerations

5.1 Layer-violating Protocols

In an ideal world, free from layering violations, all layer 3 protocols would operate correctly on top of MOOSE in exactly the same way that they currently operate on top of Ethernet, with no protocol-specific handling necessary in the switch. In reality, however, protocols abound which use hosts' MAC addresses for purposes other than layer 2 addressing or which place MAC addresses in the frame payload. DHCP and ARP have already been mentioned as such protocols which must be specifically handled by edge switches in order to operate; luckily, the rewriting required for these important protocols is simple.

Of particular concern are recent standards for layering on top of Ethernet protocols which were previously used solely on dedicated hardware interconnects, such as Fibre Channel over Ethernet (FCoE) [27]. In order to support FCoE and similar protocols on a MOOSE network, each edge switch will need to be able to interpret and rewrite individual protocols that are in use. A production MOOSE switch would, therefore, need to be implemented such that it is possible to add rewriting support for additional protocols after manufacture, for example by loading an additional software or FPGA configuration module.

Ultimately, in the general case, this problem could be addressed more satisfactorily by extending the Ethernet standard to provide a protocol-agnostic method for a layer 2 network to inform hosts of their own addresses; LLDP [19] would make a good basis for this extension. This would allow the use of network-assigned MAC addresses for any protocol, with some rewriting performed either partially (within the frame payload) or fully by the host itself, and furthermore would allow higher-layer protocols to respond to changes of the host's network-assigned address (e.g. due to mobility). Such a mechanism could be deployed incrementally as needed, with switches able to perform address rewriting for hosts which are not able to do this themselves. This is, however, a very long-term solution, and protocol-specific rewriting on the switch is likely to be required for the foreseeable future.

FCoE in particular is unusual, however, as it already does its own dynamic allocation of MAC address to devices. It is conceivable that an extension to FCoE could be developed which allows a network-wide dynamic address assignment scheme such as MOOSE to be exploited to provide addresses directly to fibre channel devices.

5.2 Edge Virtual Bridging

The rise of virtualisation has caused an unanticipated proliferation of software switches, usually in the host operating system or hypervisor which provides network connectivity to multiple virtual machines. Since software switches are almost always neither fast nor centrally manageable in the same way as hardware switches, there is ongoing work to standardise—by Cisco as Port Extension [28] and by the IEEE as P802.1Qbg [29]—a means of making these software switches act merely as additional ports which are logically part of a more central hardware switch. This reduces the work required by a virtual edge switch: frames from local virtual edge ports can be forwarded straight out via the uplink to a physical switch without consideration, and frames from the uplink will arrive simply tagged with a virtual edge port identifier.

(The scope of Port Extension in particular is greater than this, and allows for physical port extenders to exist in place of switches where a large number of ports but a small amount of processing is required, but virtualisation is likely to be the most significant use case.)

Edge Virtual Bridging and Port Extension require very little adaptation to be implemented on a MOOSE switch. It is unlikely, although too early in the standardisation process to say for certain, that the virtual bridge will need to be MOOSE-aware. A virtual-bridging-aware physical MOOSE switch will thus simply need to take into account the possibility that one physical port may hide a large number of virtual ports when allocating host identifiers, as it would if it had an Ethernet switch connected on that port. If, however, the virtual bridge is made MOOSE-aware, the hierarchical addressing of MOOSE could be exploited to allow the virtual bridge to allocate host identifiers itself, given that it is likely to be aware of the exact number and nature of virtual edge ports. The parent MOOSE switch would accordingly allocate an address prefix to each child virtual bridge, and hosts' full MOOSE addresses could be formed as:

switch ID	:	**child ID**	:	**host ID**
(parent)		*allocated by parent*		*allocated by child*

6 Implementation

We have implemented a MOOSE switch in threaded, object-oriented Python as a proof-of-concept. The architecture is intended for clarity and modularity rather than raw performance, but this implementation is still capable of switching data at 100 Mbps on a modern desktop PC.

Data forwarding and control functions are kept separate for clarity and to mimic a hardware implementation.

6.1 Data plane

The software MOOSE switch is intended to be run on a Linux PC with several network interface cards, and uses raw sockets to send and receive frames directly in promiscuous mode, so that all frames are received whether or not they are addressed to that PC.

Each network interface is managed by two independent threads: a FrameReceiver and a FrameTransmitter. A Port object is maintained in shared memory, containing shared data structures such as a per-port forwarding database (implemented as two Python dictionaries: one for locally-connected hosts and one for remote switches). The relation between modules is outlined in figure 4.

The FrameReceiver thread does most of the work; the main steps performed are:

i) A received frame is immediately packaged in a Frame object, which provides methods for accessing individual fields within the frame's headers. Some checks are run so that unwanted frames are dropped: for example, frames whose source addresses indicate that they have already passed through this switch, which could be a sign of a routing protocol malfunction or a misconfigured switch elsewhere in the network with the same switch identifier.

ii) If the frame is a DHCP or ARP query, it is transferred to the control plane for conversion into an ELK query.

iii) Once the frame has been received and checked to be valid, the source address is rewritten if it is not already a MOOSE address. This process requires a host identifier, which is reused or allocated as appropriate; allocated identifiers are stored in a Python dictionary (hash table). In this implementation, in order to allow each port to issue host identifiers independently, each identifier starts with a byte identifying to which port this host is connected; for example, the host identifier of the first host seen on port 3 will be 03:00:01.

iv) The locally-connected-host forwarding database is updated where necessary based upon the frame's source address.

v) The frame is passed to the relevant forwarding database, which obtains the correct destination Port object from its internal dictionary. The frame is enqueued with the FrameTransmitter of this port.

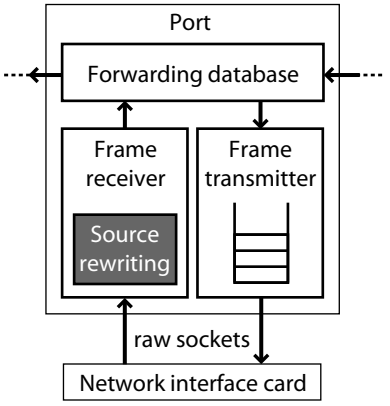

Figure 4: Prototype data plane architecture

The only processing the FrameTransmitter performs before sending the frames over the relevant raw socket onto the network is to rewrite the destination address to be the target host's manufacturer-assigned MAC address if the destination switch ID is this switch's, i.e. if that host is directly connected to this switch.

The use of threads for parallel transmission to and reception from each network interface makes this software design analogous to a basic hardware design, with several independently-operating ports interconnected by a switch fabric. It is our eventual goal to produce a hardware prototype as described in section 7.

6.2 Control plane

The control plane operates largely independently of the data plane, in a separate thread, as it is much less timing-sensitive than the data plane. In a production implementation, the control plane would likely run in software on a microprocessor.

The main function of the control plane is to run the routing protocol in order to determine the location of and best route to every other switch. This implementation uses PWOSPF [30], a simple link state routing protocol based on OSPF version 2 [21], as a proof of concept—the authentication features of OSPF are not required for a prototype implementation. A production MOOSE switch would likely require a full-featured OSPF implementation (or another routing protocol) to ensure security and resilience, and in particular to pre-

```
Address       HWtype HWaddress           Iface
10.100.11.1   ether  02:00:0c:01:00:01   eth1
10.100.11.3   ether  02:00:0a:01:00:01   eth1
10.100.11.4   ether  02:00:0a:03:00:01   eth1
10.100.11.8   ether  02:00:0b:02:00:01   eth1
```

Figure 5: ARP cache of an unmodified Linux PC connected to a network of MOOSE switches (02:00:0a, 02:00:0b, 02:00:0c) showing the addresses automatically assigned to other hosts

vent the unauthorised spoofing of switches by end users.

As PWOSPF is designed to operate on 4-byte IP addresses rather than the 3- or 6-byte identifiers used in MOOSE, the fields intended for IP addresses are retrofitted to contain a switch identifier followed by one null byte of padding. Each switch handles all addresses starting with its switch identifier, which is equivalent to each switch routing a subnet of length 24 bits.

The routing protocol calculates the shortest path to every other switch using Dijkstra's algorithm [31]. The results are used to update the remote-switch forwarding database maintained in each Port object of the local data plane with the best Port on which to output frames destined for each switch.

The control plane would also be responsible for operating the mobility handover protocol; however this protocol was unimplemented in the prototype.

6.3 Evaluation

The prototype MOOSE switch was found to behave transparently to a variety of unmodified devices communicating via IP with each other and with other hosts on the Internet, both with physical and wireless connections to a network of three MOOSE switches. The only visible effect was, as intended, that hosts' individual ARP caches show MOOSE addresses, as shown in figure 5.

A second test was run on a virtual network comprising six virtual switches each connected to ten Xen virtual machines acting as client hosts. This allows for comparison with a traditional Ethernet switch (as implemented by the Linux bridging driver). The forwarding database of each switch was inspected during a period when all clients were actively transmitting broadcast packets. In the case of Ethernet, the switches' forwarding databases each contained sixty entries. In the case of MOOSE, each switch's two forwarding database dictionaries contained five entries for the other switches and ten entries for the locally-connected hosts respectively. The storage requirement of the forwarding has been reduced from $O(\text{hosts})$ to $O(\text{switches})$, assuming that the number of locally-connected hosts is a small constant; this is a significant improvement.

7 Conclusions and Future Work

Ethernet remains popular due to its simplicity and ubiquity, but is showing its age and exhibits serious scalability issues in large deployments. Previously-proposed improvements address either a few of the problems in a simple way, or most of the problems in a highly complex or backwards-incompatible way. We have demonstrated a simple, novel and easily-implementable approach for significantly boosting the scalability of Ethernet, along with a working software implementation.

Our next step will be to produce a true hardware prototype. This will be built using the NetFPGA platform [32]. The NetFPGA card comprises four gigabit Ethernet interfaces connected directly to an FPGA with full control over everything from the Ethernet PHY and MAC inwards, plus a PCI interface to allow frames to be passed to a PC for processing in software (e.g. for running the control plane).

We also intend to implement a more extensive set of additional Ethernet features, including in particular 802.1Q VLANs and Quality-of-Service provision.

8 Acknowledgements

We acknowledge the support of the UK EPSRC which funded this project through grant EP/D076803/1. We are also grateful for David Simner's invaluable security insight, and for the countless comments and suggestions made by him, Ian Abel, Dave Eyers, Malte Schwarzkopf, Laurence Aitchison and Derek Murray.

Useful feedback and suggestions were provided by several attendees of the ITC 21 First Workshop on Data Center – Converged and Virtual Ethernet Switching.

References

[1] R. M. Metcalfe and D. R. Boggs, "Ethernet: distributed packet switching for local computer networks," *Commun. ACM*, vol. 19, no. 7, pp. 395–404, 1976.

[2] P. Barham, *et al.*, "Xen and the art of virtualization," in *SOSP*, 2003, pp. 164–177.

[3] C. Clark, *et al.*, "Live migration of virtual machines," in *USENIX NSDI*, 2005.

[4] C. Perkins, "IP Mobility Support for IPv4," RFC 3344 (Proposed Standard), Aug. 2002, updated by RFC 4721. [Online]. Available: http://www.ietf.org/rfc/rfc3344.txt

[5] IEEE, "802.1D: Standard for local and metropolitan area networks: Media access control (MAC) bridges," 2004.

[6] 3Com Corporation, "Switch 5500G 10/100/1000 family data sheet." [Online]. Available: http://www.3com.com/other/pdfs/products/en_US/400908.pdf

[7] F. Yu, *et al.*, "Efficient multimatch packet classification and lookup with tcam," *IEEE Micro*, vol. 25, no. 1, Jan. 2005.

[8] K. Pagiamtzis and A. Sheikholeslami, "Content-Addressable Memory (CAM) circuits and architectures: a tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, pp. 712–727, 2006.

[9] D. C. Plummer, "Ethernet Address Resolution Protocol," RFC 826 (Standard), Nov. 1982. [Online]. Available: http://www.ietf.org/rfc/rfc826.txt

[10] R. Droms, "Dynamic Host Configuration Protocol," RFC 2131 (Draft Standard), Mar. 1997, updated by RFCs 3396, 4361. [Online]. Available: http://www.ietf.org/rfc/rfc2131.txt

[11] M. Scott, A. Moore, and J. Crowcroft, "Addressing the scalability of Ethernet with MOOSE," in *ITC 21 First Workshop on Data Center – Converged and Virtual Ethernet Switching (DC CAVES)*, Sep. 2009.

[12] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," RFC 3031 (Proposed Standard), Jan. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3031.txt

[13] I. Hadžić, "Hierarchical MAC address space in public Ethernet networks," in *IEEE GLOBECOM*, vol. 3, 2001.

[14] T. Rodeheffer, C. Thekkath, and D. Anderson, "SmartBridge: a scalable bridge architecture," in *SIGCOMM*, 2000.

[15] R. Perlman, "Rbridges: transparent routing," in *INFOCOM*, vol. 2, 2004.

[16] A. Myers, E. Ng, and H. Zhang, "Rethinking the service model: Scaling Ethernet to a million nodes," in *ACM SIGCOMM Workshop on Hot Topics in Networking*, Nov. 2004.

[17] C. Kim, M. Caesar, and J. Rexford, "Floodless in SEATTLE: a scalable Ethernet architecture for large enterprises," in *SIGCOMM*, 2008, pp. 3–14.

[18] S. Sipes, "Why your switched network isn't secure," in *Intrusion Detection FAQ*. The SANS Institute, Sep. 2000. [Online]. Available: http://www.sans.org/resources/idfaq/switched_network.php

[19] IEEE, "802.1AB: Station and media access control connectivity discovery," 2009.

[20] ——, "802.1X: Port based network access control," 2004.

[21] J. Moy, "OSPF Version 2," RFC 2328 (Standard), Apr. 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2328.txt

[22] C. Villamizar, "OSPF optimized multipath (OSPF-OMP)," IETF Internet Draft, Feb. 1999. [Online]. Available: http://tools.ietf.org/html/draft-ietf-ospf-omp-02

[23] G. M. Schneider and T. Nemeth, "A simulation study of the OSPF-OMP routing algorithm," *Computer Networks*, vol. 39, no. 4, pp. 457–468, 2002.

[24] C. Villamizar, R. Chandra, and R. Govindan, "BGP Route Flap Damping," RFC 2439 (Proposed Standard), Nov. 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2439.txt

[25] A. Adams, J. Nicholas, and W. Siadak, "Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised)," RFC 3973 (Experimental), Jan. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc3973.txt

[26] M. Christensen, *et al.*, "Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches," RFC 4541 (Informational), May 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4541.txt

[27] T11 FC-BB-5 working group, "Fibre channel backbone – 5," Jun. 2009.

[28] J. Pelissier, "Introduction to port extension," in *ITC 21 First Workshop on Data Center – Converged and Virtual Ethernet Switching (DC CAVES)*, Sep. 2009.

[29] A. Jeffree, P. Congdon, and J. Pelissier, "P802.1Qbg: Edge virtual bridging," Unapproved PAR, Sep. 2009. [Online]. Available: http://ieee802.org/1/files/public/docs2009/new-qbg-draft-par-0909-V2.pdf

[30] Stanford University High-Performance Networking Group, "Pee-Wee OSPF protocol details." [Online]. Available: http://web.archive.org/web/20070708180017/http://yuba.stanford.edu/cs344_public/docs/pwospf_ref.txt

[31] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[32] J. W. Lockwood, *et al.*, "NetFPGA—an open platform for gigabit-rate network switching and routing," in *IEEE MSE*, 2007, pp. 160–161.