

All-Path Bridging: Path Exploration Protocols for Data Center and Campus Networks

Elisa Rojas^a, Guillermo Ibañez^a, Jose Manuel Gimenez-Guzman^{a,*}, Juan A. Carral^a, Alberto Garcia-Martinez^b,
Isaias Martinez-Yelmo^a, Jose Manuel Arco^a

^a*Dept. Automática, Universidad de Alcalá, Alcalá de Henares (Madrid), Spain.*

^b*Dept. Ingeniería Telemática, Universidad Carlos III Madrid, Spain.*

Abstract

Today, link-state routing protocols that compute multiple shortest paths predominate in data center and campus networks, where routing is performed either in layer three or in layer two using link-state routing protocols. But current proposals based on link-state routing do not adapt well to real time traffic variations and become very complex when attempting to balance the traffic load. We propose All-Path bridging, an evolution of the classical transparent bridging that forwards frames over shortest paths using the complete network topology, which overcomes the limitations of the spanning tree protocol. All-Path is a new frame routing paradigm based on the simultaneous exploration of all paths of the real network by a broadcast probe frame, instead of computing routes on the network graph. This paper presents All-Path switches and their differences with standard switches and describes ARP-Path protocol in detail, its path recovery mechanisms and compatibility with IEEE 802.1 standard bridges. ARP-Path is the first protocol variant of the All-Path protocol family. ARP-Path reuses the standard ARP Request and Reply packets to explore reactively the network and find the fastest path between two hosts. We compare its performance in terms of latency and load distribution with link-state shortest-path routing bridges, showing that ARP-Path distributes the load more evenly and provides lower latencies. Implementations on different platforms prove the robustness of the protocol. The conclusion is that All-Path bridging offer a simple, resilient and scalable alternative to path computation protocols.

Keywords: Path Exploration, Path computation, Routing Switches, Shortest Path Bridges, All-Path

1. Introduction

Today Ethernet networks pose stringent functional and performance requirements like low latency, effective network utilization (load distribution [1]), self-configuration, scalability, transparency to existing hosts and routers, compatibility with existing protocol layering and topology independence. The challenge to fulfill these requirements is aggravated by the high variability of the traffic transported by the network [2]. In this scenario, being able to efficiently use the available communication capacity, especially in the case of data centers, requires traffic engineering techniques different from those applied to corporate networks or ISPs, since there are no applicable (stable) traffic matrices [3]. Standard solutions combine physical topologies, which naturally provide multiple equal-cost paths among source/destination pairs of edge switches

(the so-called Top-of-Rack (ToR) switches), such as folded Clos/Fat Tree topologies [3], with routing technologies able to exploit this multiplicity of paths. In general, paths are assigned to flows according to a hash computation based on the parameters identifying the flow. The blind nature of this assignment regarding path utilization and flow size makes this assignment strategy suboptimal in terms of network utilization.

The need for a new type of switches providing shortest paths to overcome the limitation of the Spanning Tree Protocol (STP) led to the creation, by 2004/2005, of two standardization groups: Shortest Path Bridges (SPB) [4, 5] and Transparent Routing Bridges (TRILL RBridges) [6]. These two proposals aimed, among other objectives, to build switched networks of big size, organized as a single IP subnet while allowing full utilization of infrastructure links to obtain shortest paths. The basic routing paradigm in both proposals is to hybridize bridges into routing bridges by using a layer-two variant of the proven link-state routing protocol (IS-IS) to compute shortest path routes between switches. This dominance has inhibited the research on new bridging paradigms. In the absence of novel proposals, path computation is by far the dominant approach for shortest path bridging in large Ethernet networks [3, 7, 8] and, surprisingly, path exploration as a form

*Corresponding author

Email addresses: elisa.rojas@uah.es (Elisa Rojas),
guillermo.ibanez@uah.es (Guillermo Ibañez),
josem.gimenez@uah.es (Jose Manuel Gimenez-Guzman),
juanantonio.carral@uah.es (Juan A. Carral),
alberto@it.uc3m.es (Alberto Garcia-Martinez),
isaias.martinezy@uah.es (Isaias Martinez-Yelmo),
josem.arco@uah.es (Jose Manuel Arco)

of finding routes has been scarcely used in networks, only for on-demand routing in Ad-Hoc networks [9]. Path computation protocols involve, due to the huge size of current networks, significant complexity both in terms of computation and control message exchange, and additional loop control mechanisms because link-state database may temporarily be not consistent (synchronized) between nodes. Moreover, additional complexity is added to obtain path diversity between switches by computing multiple symmetric equal-cost paths for load balancing [10].

Looking for alternatives to the current link-state routing paradigm we propose and implement All-Path [11, 12], an evolution of the new family of classic backward learning transparent bridging paradigm that is based on path exploration and it is suitable for data center, campus and enterprise networks. The key advantages of All-Path are simplicity, low latency, self-configuration and native distribution of load. The first All-Path protocol, named ARP-Path, relies on the standard ARP Request packet (Neighbor Solicitation for IPv6) to explore multiple paths simultaneously and to select the one with the lowest latency. In this way, it selects a path according to an instantaneous indication of the load in all the possible paths between the sender of the ARP Request and the destination. A path is set up for every pair of nodes for which an ARP Request/Reply exchange is performed, and it is refreshed each time the ARP exchange proceeds. Similar to swarm routing, used paths are also self-maintained via forward refreshing of learnt MAC addresses. Unused paths (MAC addresses) expire as in standard switches.

The main contributions of this paper are as follows:

- We present All-Path switches and their differences with standard switches and describe ARP-Path protocol in detail, its path recovery mechanisms and compatibility with IEEE 802.1 standard bridges.
- We compare ARP-Path with ECMP routing regarding scalability and fault tolerance.
- We perform a comparison of load distribution and latencies through simulation in a scenario especially suited for SPB due to the evenly used traffic distribution.

The remainder of the paper is structured as follows. Section 2 presents a description of how path exploration mechanisms work for later defining the conditions of loop freeness in section 3. We compare the proposed protocols with the path computation paradigm in sections 4 and 5. We briefly describe an experimental testbed and the protocol family evolution in sections 6 and 7, respectively. Finally, in section 8 conclusions are provided.

2. All-Path Bridging

With the intention of going beyond link-state routing for shortest path bridging, we have explored an alterna-

tive, based solely on the mechanisms of transparent bridging: frame flooding, filtering and address learning. The predecessor of our loop free path exploration mechanism is Reverse Path Forwarding (RPF) [13]. In a sense, we have moved this mechanism from layer three (source IP address and routing tables at routers) to layer two (source MAC address and port-MAC forwarding tables of switches via learning).

2.1. All-Path Switches vs. Standard Switches

All-Path switches do not differ very much from classical standard switches, but we can point out three differences between them: first, source addresses are only learnt from specific frames; second, but essential, the learning of source address at the port of first arrival (of broadcast frames) blocks further learning (for a short time) of the same source address at other switch ports in order to prevent loops; and third, frames destined to unknown MAC unicast addresses, i.e. addresses for which the switch has no port associated to the destination MAC addresses, are not flooded. A path recovery mechanism is used in case of link or switch failure. All-Path switches can implement link aggregation, 802.1Q VLAN tags, and in general all other IEEE standard features because the forwarding mechanism is fully independent of them.

2.2. Basic ARP-Path protocol description

ARP-Path is the first protocol of a new family of transparent switches that we identify as All-Path switches. The basic idea behind All-Path switches is to explore simultaneously all possible network paths between a given pair of nodes with a broadcast frame while at the same time preventing frame loops with the locked association on first-arrival-port to the MAC source address of the frame (for a detailed study of loop prevention, see section 3). The path is built by processing in a special way the IP to MAC address resolution packets (either IPv4 or IPv6) generated by the source host. We are focusing on the IPv4 to MAC address resolution protocol: ARP. Every switch forwards the ARP Request broadcast message through all its output links, except the one that received the packet. A deeper insight through an example is given in the next subsections. In subsections 2.2.1 and 2.2.2 we describe how the path to destination is created by the ARP Request message and how it is created towards the source by the ARP Reply message, respectively. In subsection 2.2.3 we summarize the operation of ARP-Path, including its pseudocode. Finally, in subsection 2.2.4 we include the procedure used for path recovery after link failures.

2.2.1. Path exploration (ARP Request)

The process, described in Fig. 1 (left), works as follows: whenever a source host S wants to communicate with a destination host D and there is not a valid entry for D in its ARP cache table, the host sends an ARP Request broadcast packet to resolve the IP address of D.

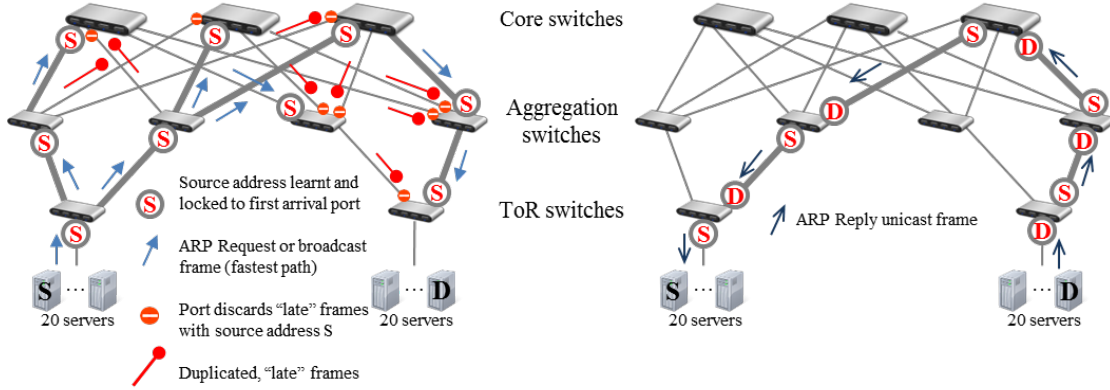


Figure 1: Path discovery: S to D (left), D to S (right).

The ingress switch of S receives the frame from this host and associates the MAC address of S to the port through which it has (first) received the message, temporarily locking the learning (the association) of S address to this port, and preventing all other ports of the ingress switch from learning and also from forwarding further received broadcast frames from source address S during the lock timer interval. Thus, frames with source address S arriving to other ports of the switch will be discarded as late frames. Then, the ingress switch in the group of ToR switches forwards the ARP Request frame to all ports except the one where it was received. Aggregation switches behave as ToR switches, associating address S to the port that first receives the frame. Afterwards, aggregation switches also broadcast the frame through all other ports except the port where it was first received. The same happens at the core switches, which will broadcast the frame to the other aggregation switches and will receive late copies that will be discarded as well. Therefore, the ARP Request is broadcasted through all the switches in the network and reaches all the servers with no loops.

Hence, the temporary association (locking) of address S to a port at every switch is propagated across the network as a tree rooted at host S, until the network edge switches and their hosts are reached, including the host D, destination of the ARP Request. A chain of switches with an input port locked to S is now in place between S and D.

2.2.2. Path completion (ARP Reply)

The mechanism for path set up in the opposite direction is shown in Fig. 1 (right). The ARP Request from S is followed by the corresponding ARP Reply from D. The reply is transported in a unicast frame, sourced at D and addressed to S, and it will follow back the branch of the sink tree that was selected in the previous phase, that is, the fastest branch reaching from source to destination. Thus, ARP-Path switches take advantage of the ARP Reply processing to learn the port to reach D (i.e. the receiving port of the ARP Reply). As the ARP Re-

ply is transported in a unicast frame, only the switches located in the branch connecting S to D will learn about D location. The ARP Reply frame also refreshes the path from S to D.

2.2.3. ARP-Path basic forwarding pseudocode

Figure 2 presents the pseudocode associated to the protocol forwarding operation. It introduces two implementation tables: the Learning Table (LT) and the Blocking/Broadcast Table (BT), designed to maintain the forwarding state and to avoid loops in the network, respectively. LT keeps forwarding entries with the mapping of unicast addresses to their output ports. LT entries have a long expiration timeout (over 10s) and may be refreshed in two ways: in the forward direction by unicast frames and in the backward direction by ARP Request and Reply messages. Note that refresh in the forward direction implies to update the entry associated to the output port of the frame while refresh in the backward direction is its equivalent for the input port. BT is linked to the loop-free forwarding mechanism and every table entry represents a temporal lock of a given unicast address (the source of the broadcast addressed frame) to an input port in the switch. BT entries have a short expiration timeout (see section 3) and they are refreshed by new broadcast frames received from the same unicast source address at the same input port.

2.2.4. Path recovery

Given that every link in a network connects two ports (one at each end of that link), when a link fails, both ports become no longer valid for forwarding packets, so their associated entries in LT must be updated accordingly for ARP-Path. When a node fails, the same happens for all the links connected to that node.

In ARP-Path, when a switch detects that a port is down the recovery procedure is triggered, whose objective is to update LT in every switch belonging to the broken path. With this purpose, ARP-Path switches create and broadcast a LinkFail packet with its own MAC address as

ARP-Path Forwarding Operation

```
1: if dst_mac is BROADCAST or MULTICAST then
2:   if (src_mac is not in BT) || (src_mac is in BT && input_port == BT_port) then
3:     if (ARP Request && src_mac is not in LT) then
4:       Update LT (new src_mac and input_port entry)
5:     Update BT (new src_mac and input_port entry, or refresh timer)
6:     Broadcast frame (through all ports but the incoming one)
7:   else
8:     Discard frame
9:   else if dst_mac is UNICAST then
10:    if (ARP Reply && src_mac is not in LT) then
11:      Update LT (new src_mac and input_port entry)
12:    Update LT (refresh timer of dst_mac)
13:    Forward frame (through port associated to LT entry of dst_mac)
```

Figure 2: Pseudocode of the ARP-Path protocol.

source address and with the multicast *ARP-Path_Multicast* MAC address as destination address. The content of this LinkFail packet is a list of all the MAC addresses affected by the link or node failure in its LT. Afterwards, the switch deletes all those entries from its LT. The ARP-Path recovery operation is sketched in Fig. 3a.

Every switch receiving the LinkFail message checks if it is the edge switch of any of the flushed host addresses encapsulated in that LinkFail packet (i.e. if the host is directly connected to the switch). In this case the edge switch responds with a unicast LinkReply, whose source address is the MAC address of the directly connected host, that is, the MAC address being repaired, and whose destination address is the source MAC address of the LinkFail packet. In this way, the path is recreated as usual. An advantage of this procedure is that this unicast message only repairs the section of the path needed and not the whole path, which requires less messages and less time. The number of broadcast messages needed to repair all paths in the network will be usually two (single link failure). The forwarding operation including the recovery messages is shown in Fig. 3b, where bold text highlights the differences respect to the normal operation shown in Fig. 2.

2.3. Coexistence with standard switches IEEE 802.1 and 802.1Q

ARP-Path switches may coexist with standard switches in core-island mode. A core of ARP-Path switches may interconnect islands of standard switches running the Rapid Spanning Tree Protocol (RSTP), as described in [14].

Regarding the coexistence of VLANs and our proposal, it must be noted that ARP-Path has the advantage of being fully architecturally independent of VLANs, as ARP-Path does not need to assign a VLAN ID to every edge switch to create separated forwarding domains to prevent broadcast loops. Therefore, VLAN IDs can be used for

traffic separation and security and do not interfere with the basic ARP-Path forwarding mechanisms.

3. Loop-free broadcasting

The proposed ARP-Path protocol is a novel yet simple mechanism that provides path exploration capabilities and obtains the fastest path in a network in conjunction with load balancing. However, it is necessary to assure that this mechanism is loop-free to avoid packet storms and undesired misbehaviours. This characteristic is achieved in regular Ethernet switches using STP. In order to prevent loops, STP is used to block all redundant links by creating a single tree but destroying the possibility of a pure layer-two shortest path bridging protocol.

The ARP-Path protocol is based on the simultaneous exploration of all paths in a network with a broadcast frame, which is a simple and efficient mechanism to find shortest paths without blocking links. Unlike STP, ARP-Path is not limited to just one tree, using a locking mechanism for avoiding loops instead. In this section we show the conditions for loop freeness in ARP-Path.

Constraint 1. *All connections among switches are dedicated point-to-point links.*

Constraint 2. *The timer for lock removal must be greater than the slowest loop path in the network.*

Property 1. *When constraints 1 and 2 are fulfilled, loop-free transmission is assured in any network topology.*

Proof. There are two kind of possible loops than can occur in any switch. The first kind is the case when the frame arrives on the same port that triggered the locking mechanism. The second kind is the case when the frame arrives on a different port from the port that triggered the locking mechanism. Constraint 1 inherently removes the

ARP-Path Recovery Operation

- 1: **if** link fails and switch detects that port down **then**
- 2: Create LinkFail packet with:
- 3: src_mac = switch_id or switch_mac (current switch)
- 4: dst_mac= arppath_mcast
- 5: encapsulated data = mac addresses in LT associated to down port
- 6: Broadcast frame through all ports
- 7: Erase all entries in LT associated to down port

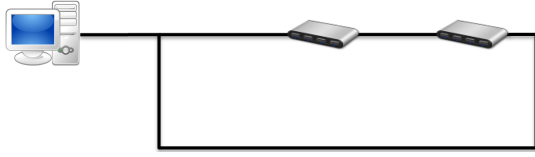
(a)

ARP-Path Forwarding Operation including Path Recovery

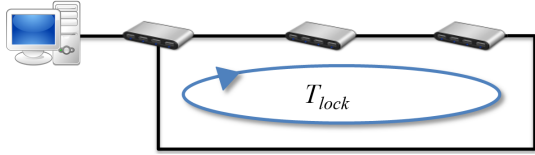
- 1: **if** dst_mac is BROADCAST or MULTICAST **then**
- 2: **if** (src_mac is not in BT) || (src_mac is in BT && input_port == BT_port) **then**
- 3: **if** ((ARP Request && src_mac is not in LT) || **LinkFail**) **then**
- 4: Update LT (new src_mac and input_port entry)
- 5: Update BT (new src_mac and input_port entry, or refresh timer)
- 6: **if** **LinkFail** && encapsulated mac directly connected **then**
- 7: **Create LinkReply packet with:**
- 8: src_mac= encapsulated mac directly connected (mac being repaired)
- 9: dst_mac=src_mac of **LinkFail**
- 10: **Forward frame (through input_port)**
- 11: **else**
- 12: Broadcast frame (through all ports but the incoming one)
- 13: **else**
- 14: Discard frame
- 15: **else if** dst_mac is UNICAST **then**
- 16: **if** ((ARP Reply && src_mac is not in LT) || **LinkReply**) **then**
- 17: Update LT (new src_mac and input_port entry)
- 18: Update LT (refresh timer of dst_mac)
- 19: Forward frame (through port associated to LT entry of dst_mac)

(b)

Figure 3: Pseudocode of the ARP-Path protocol including Path Recovery.



(a) Constraint 1: Only point-to-point links are permitted between switches (not compliant scheme).



(b) Constraint 2: $T_{lock} > LoopDelay$.


: ARP-Path switch.

Figure 4: Constraints for loop-free broadcasting

possibility of the first kind of loops. Constraint 2 removes the possibility of the second kind of loops since the locking mechanism will remain active until the slowest frame arrives. \square

Property 2. *The upper bound for the minimum locking time in a network with N switches is as follows:*

$$T_{lock} = \sum_{n=1}^N (d_{proc,n} + d_{queue,n} + d_{trans,n} + d_{prop,n}), \quad (1)$$

where d_{proc} , d_{queue} , d_{trans} and d_{prop} stand for the processing, queueing, transmission and propagation delays, respectively.

Proof. The worst case of a loop in any network occurs when all switches participate in the loop. In any switch a packet must go through the processing time, the queueing time, the transmission time and the propagation time. \square

Property 3. *The upper bound for the minimum locking time in a network with N switches where all of them share the same processing, queueing, transmission and propagation times is:*

$$T_{lock} = N \cdot (d_{proc} + d_{queue} + d_{trans} + d_{prop}). \quad (2)$$

Proof. If all terms in the summary are the same, the summary can be replaced by a product of the number of switches. \square

Figure 4a illustrates how switches must not be connected since only point-to-point connections must be allowed between switches. Figure 4b illustrates a valid topology fulfilling loop-free conditions; switches are only connected via point-to-point links and T_{lock} is greater than

the loop delay, which shows that replicated packets will not be forwarded, avoiding packet storms. Considering the case of Fast Ethernet switches with transmission buffer of 4MB (hence, $d_{trans} = 4 \cdot 2^{20} \cdot 8/10^8$ s), a propagation delay in copper of $5 \mu s$ (d_{prop}), neglecting the processing and queueing delays (d_{proc} and d_{queue}) and applying corollary 3, we have $T_{lock} = N \cdot 0.336s$, which implies a short timer in the order of a few seconds.

4. Comparison with link-state protocols

In this section we qualitatively compare the ARP-Path protocol with a link-state protocol like SPB (next section contains a numerical evaluation by simulation). Notice that the comparison with link-state routing is difficult because the two paradigms differ drastically: proactive computation of paths between switches versus reactive (on the fly) path set up by the standard ARP dialog between hosts. Intuitively, opposite to path computation protocols, where increasing the number of network nodes exponentially increases the complexity of route computation, the scalability of path exploration protocols like ARP-Path protocol is only limited by the size of forwarding tables and the amount of broadcast traffic injected to the hosts.

We consider a network of b switches, E links between switches (edges) and H final physical hosts. We assume that $H \gg b$ (H between one and two orders of magnitude bigger than b). For the case in which hosts have one or more virtual machines installed, we consider h as the total number of virtual machines in the network (being $\chi = h/H$ the average number of virtual machines per final physical host). Note also that not every virtual machine will be active at every moment, so we define h_a as the mean number of active virtual machines, which is a fraction of h , i.e. $h_a = \alpha h = \alpha \chi H$. For example, $\alpha = 0.5$ represents the case when half of the total virtual machines are active.

In the next subsections we compare different aspects of the ARP-Path protocol versus a link-state protocol like SPB. Moreover, we also include Table 1, which shows the summary of this comparison.

4.1. Forwarding state

At every switch, a link-state protocol needs a routing table entry per switch or virtual machine of the entire network, i.e. $b + h = b + \chi H$. ARP-Path creates an entry per active virtual machine in the network (h_a) but only at switches located in active paths. Active paths depend on two parameters, s and s_e , being the former the average number of switches that form a path for a flow or couple of virtual machines and the latter the average number of switches that also share the path to the same destination from different sources (note that s_e switches are not included in s). ARP-Path depends on the addition of s and s_e because paths are shared, i.e. when a flow creates a path (defined by s switches on average), different sources

Table 1: ARP-Path vs. SPB comparison.

	SPB	ARP-Path
Forwarding state per switch	$b + \chi H$	$\alpha \chi H \cdot (s + s_e)/b$
Number of messages for routing	$b \cdot E$	-
Number of broadcast replicas	$b - 1 + H$	$2 \cdot E - (b - 1) + H$
Computational complexity	Proportional to b^2	-
Computational complexity for path diversity	Proportional to b^3	-
Messages after link failure	$[2 \cdot E, b \cdot E]$	$[2 \cdot (2 \cdot E - (b - 1)) + \phi, 2 \cdot (2 \cdot E - (b - 1)) + e_b \cdot \phi]$

can join the already existing paths just by adding branches (defined by s_e switches on average), thus defining a tree in the end ($s + s_e$). Finally, we have to divide the total amount of entries in the network by the total number of switches (b), as we are comparing the number of table entries per switch. Concluding, for the ARP-Path protocol the forwarding states that must be stored at each switch are $h_a \cdot (s + s_e)/b = \alpha \chi H \cdot (s + s_e)/b$.

It is important to note that the forwarding state in a link-state protocol depends on the number of virtual machines (h), unlike ARP-Path, which only depends on the current active virtual machines (h_a) that are a fraction of h . For that reason, let us consider the worst case for ARP-Path, which is an extreme situation that requires: i) all virtual machines are active ($h = h_a$ and $\alpha = 1$), ii) each host has an active communication with every other host of the network at the same time and iii) when there are several possible paths, all of them are used. In that very unlikely scenario and for ARP-Path, all the network is in a tree ($s + s_e = b$) and therefore each switch must store $h_a \cdot (s + s_e)/b = h_a \cdot b/b = h$ forwarding states, which is still lower, but very similar, than the required forwarding states in a link-state protocol ($b + h \approx h$, as $b \ll h$). However, as usually $\alpha < 1$ and especially, $s + s_e \ll b$, the number of table entries per switch in ARP-Path is typically much lower in ARP-Path than in SPB (as it will be numerically studied in Section 4.5).

4.2. Number of messages

To compute routes, SPB broadcasts link-state packets and the list of connected hosts to every switch, i.e. the number of messages is $b \cdot E$. Obviously, the route computation cost is null for ARP-Path since it does not compute paths, leveraging the ARP Request and Reply messages instead, previously emitted in every communication. As the ARP procedure appears in both link-state and ARP-Path protocols, we have not included these ARP messages in the comparison.

The number of broadcast replicas is slightly higher in ARP-Path than in SPB. SPB broadcasts messages only through trees (from RSTP, $b - 1 + H$). ARP-Path not only uses tree-links but also cross-links, there is one copy in each link ($E + H$), plus an additional copy in each cross-link ($E - (b - 1)$) thus resulting in a total of $(2 \cdot E - (b - 1) + H)$. However, the number of ARP Request messages forwarded

to hosts can be greatly reduced using ARP proxies at edge switches if needed [15]. Moreover, an interesting variant for the ARP-Path protocol that prevents the ARP Request messages reaching the hosts at low processing effort in ToR consists of ToR intercepting all ARP Request messages to their directly connected hosts. Other methods to reduce broadcast messages are centralized or distributed directory systems where the host is registered by its edge switch. For example, TRILL edge switches periodically exchange their lists of attached hosts by using multicast communication. This last method could be applicable to ARP-Path as well.

4.3. Computational complexity

SPB uses the link-state protocol IS-IS to acquire the network topology and then applies the Dijkstra shortest path algorithm to compute routes [5]. The time needed to execute the Dijkstra algorithm, for a network of b switches, is proportional to b^2 [16]. However, there are multiple implementations of the Dijkstra algorithm and some of them are able to reduce the execution time by using Fibonacci or binary heaps based implementations [17]. As ARP-Path does not compute routes, its computational complexity is null.

If path diversity is desired, IS-IS for SPB implements hash-based ECMP multipath routing between switches to distribute load [10, 18, 19], but at the cost of requiring that paths must be congruent, i.e. paths must coincide in the two unicast directions. To achieve this, SPB computes all shortest paths of all nodes at every node, so the computational complexity of the Dijkstra algorithm is then multiplied by b . On the other side, the ARP-Path protocol sets up on-demand paths between hosts that diversify naturally according to instantaneous path latency at set-up time, so load distribution is natively achieved without any computational complexity.

Although computational complexity does not apply strictly to ARP-Path because there is no computation of routes, ARP-Path will take some time to work properly due to the need of exchanging messages. However, it is important to note that SPB will also require time to exchange the needed messages, in addition to the time required to compute routes. The study of the time required by ARP-Path to exchange messages is conducted by considering both path set up and path repair times.

In ARP-Path, path set up time is near zero, because the standard ARP dialog is used to simultaneously establish the path, which is a hardware learning and locking process only slightly different (the locking function and the absence of links blocked by the spanning tree protocol) than the process at standard switches. The ARP messages are standard, so no additional effort is spent on messages exchange for normal path set up. This is confirmed by the measured path set up time in hardware implementations (NetFPGA) which was very similar to the standard ARP Request/ARP Reply process (10 to 60 microseconds depending on frame size, similar to a commercial switch D-Link DS-1008-A). It must be also emphasized that there is not any convergence procedure in path set up, as it is just a simultaneous race of ARP Request replicas that set up a tree rooted at source host, and the fastest branch is selected for the MAC learning and final path definition.

Regarding the time needed to repair a path, it is the sum of two contributions: failure detection and failure recovery. As failure detection does not depend on the protocol, we focus on failure recovery. The time required by this procedure will be the RTT of the fastest available path from source to destination plus the time needed by switches to process the required and abovementioned messages.

4.4. Fault tolerance

ARP-Path shows high resiliency because the path for every destination host is created just at the time when it is needed, not computed in advance. This means that even if there is only a single path available in the network, it will be found.

With link-state protocols like IS-IS, used by SPB, if a link fails, the adjacent nodes will redistribute the new link state to all switches over all E links since all paths might be recomputed. Thus, the number of messages to recover from a link failure is $b \cdot E$. However, the number of broadcast messages could be sent just by the edge nodes of the failed link, being in the best case $2 \cdot E$. Hence, the number of messages after a link failure is in the range of $[2 \cdot E, b \cdot E]$.

In ARP-Path, with the repair procedure, the two switches connected to the failed link broadcast a LinkFail packet. These packets are forwarded until they reach edge bridges. That will be a total of $2 \cdot (2 \cdot E - (b - 1))$ messages, considering the formula given in section 4.2 minus the links to the hosts, since LinkFail packets are not sent to them. Then, affected edge bridges will reply with a unicast LinkReply packet directed to the switch originating the LinkFail packet. There will be at least two edge bridges replying, thus at least ϕ unicast messages are needed, being ϕ the network diameter. Notice that the path reaching from one end to that switch plus the path from the other end traverse ϕ nodes, independently of where the failure occurred. Also, and being e_b the number of edge bridges, a maximum of $e_b \cdot \phi$ messages are needed when every edge

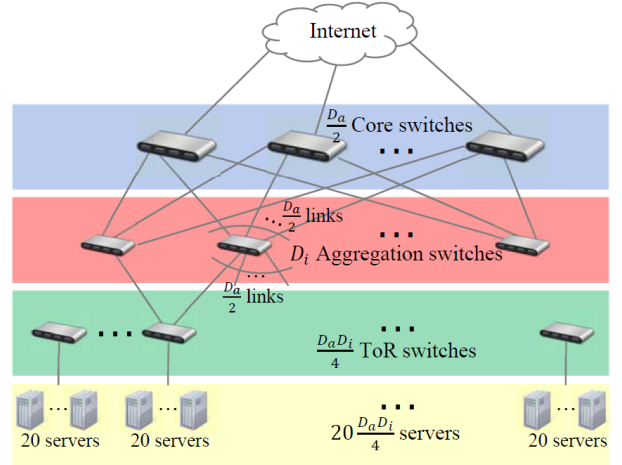


Figure 5: Clos network structure used in VL2 [3].

bridge is affected by the link failure and all unicast messages sent in reply need to go across the whole diameter of the network to reach the switch that emitted the Link-Fail. Therefore, the total number of messages produced in ARP-Path to repair the paths affected by a link failure is in the range: $[2 \cdot (2 \cdot E - (b - 1)) + \phi, 2 \cdot (2 \cdot E - (b - 1)) + e_b \cdot \phi]$.

4.5. Evaluation in VL2 data center scenario

We evaluate the previous subsections for a representative current data center network topology. More precisely we use the VL2 Clos network topology (Fig. 5) proposed in [3], having 10Gbit/s links, $D_a/2$ core switches, D_i aggregation switches, $(D_a \cdot D_i)/4$ ToR switches and 20 servers per ToR switch. That results in a bisection bandwidth of $5 \cdot D_a \cdot D_i$ Gbit/s. In order to accommodate $H = 25000$ servers (medium-sized data center), we have selected $D_a = 50$ and $D_i = 100$, while the number of ToR switches is $b_e = 1250$, the number of aggregation switches is $D_i = 100$ and the number of core switches is $D_a/2 = 25$. Thus, $b = D_a D_i / 4 + D_i + D_a / 2 = 1375$ switches and the number of links between switches is the number of aggregation switches multiplied by the number of links that every aggregation switch has, i.e. $E = D_i(D_a/2 + D_a/2) = D_i \cdot D_a = 5000$ links. The table sizes of ARP-Path edge switches depend on the number of active hosts per ToR. We consider (from the measurements in [3]) a ratio of $\alpha \cdot \chi = 5$ simultaneous active virtual machines per physical host as a representative average. This may vary widely depending on the traffic matrix characteristics, including its locality properties, sometimes promoted by network administrators to optimize network performance. Unlike SPB, ARP-Path only depends on the number of active virtual machines, which is usually a small portion of the total number of virtual machines. However, for the comparison we are considering the worst scenario for ARP-Path (see section 4.1), where $\alpha = 1$ and $s + s_e = b$. This is a very unreal scenario, as it is not usual that all virtual machines are active (usually $\alpha < 1$) and communicating with

Table 2: ARP-Path vs. SPB comparison in a VL2 Clos network assuming $b = 1375$ and $e_b = 1250$ switches, $\chi = 5$ virtual machines per final physical host, $H = 25000$ servers, $\alpha = 1$, $\phi = 4$, $s + s_e = b$ and $E = 5000$ links.

	SPB	ARP-Path
Forwarding state per switch	126375	< 125000
Number of messages for routing	6875000	-
Number of broadcast replicas	26374	33626
Computational complexity	Proportional to 1375^2	-
Computational complexity for path diversity	Proportional to 1375^3	-
Messages after link failure	[10000, 6875000]	[17256, 22252]

all the rest of virtual machines at the same time (usually $s + s_e \ll b$). With all these considerations, we include a comparison of a link-state protocol like SPB versus ARP-Path for the above-mentioned features in Table 2. As the comparison of the number of forwarding states per switch is very limited in the table (only the very unlikely worst case of ARP-Path is included), in Fig. 6 we include a comparison in a wider range of scenarios, considering $\chi = 5$ and for several values of α (fraction of active virtual machines respect to the total number of virtual machines). In the y-axis we represent $s + s_e$, ranging from $s_e = 0$ (best case for ARP-Path since there are no extra branches added to the path) to $s + s_e = b$ (worst case for ARP-Path, since every switch belongs to the tree, as defined in Table 2). It must be noted that we have chosen $s = 5$, i.e. a pessimistic scenario that considers that every path traverses the core switches (see Fig. 5). From Fig. 6 and considering that usually $s + s_e \ll b$, we can conclude that the number of forwarding states per switch in ARP-Path is usually several orders of magnitude lower than those required in SPB.

A key aspect for scalability is the type of memory hardware used for forwarding tables: Ternary CAM (TCAM) or L2/Eth (SRAM). Whilst TCAM are typically limited to 2000 – 4000 entries, L2/Eth tables can reach up to 10^5 entries [20]. TCAM memories are used when wildcard selection (longest prefix match) is needed, but SRAM suffices to forward frames based only on exact matching of the tuple $\langle \text{MAC address, VLAN ID} \rangle$, as is the case for All-Path protocols. As we can use state-of-the-art L2/Eth (SRAM) switches like BCM56648 with 10^5 entries, Fig. 6 also includes that value, showing that, for the scenario under consideration, SPB cannot be implemented using BCM56648 switches because they require more than 10^5 forwarding states per switch, although there exist alternatives like bloom filters focused on reducing the size of forwarding tables [21]. However, except in a very limited number of extreme scenarios, ARP-Path can be implemented in switches like BCM56648, as the number of forwarding states per switch is lower than 10^5 .

To evaluate ARP-Path scalability, let us assume a data center with 800000 servers and 3 active virtual machines per server. Such a data center may be implemented with the Clos network shown in Fig. 5 with different combina-

tions of D_i and D_a . Let us choose $D_a = 640$ and $D_i = 250$. This means 320 ($D_a/2$) core switches and 250 aggregation switches plus 40000 ToR switches. Total number of bridges is then 40570. Let us compute a more realistic value for S_e , the number of extra bridges participating in a path that create the tree rooted at a server. Assuming $S_e = 0.03 \cdot b$ means that three per cent of all bridges (1217) will also participate in the paths to the server (this offers a wide margin because the ToR of this server is carrying only $1/40000$ of the data center traffic on average). The forwarding state obtained by the formula for this value is 72290 entries, still an acceptable value for tables in SRAM in switch chips (note that SRAM based storage is much less limited in size, power and cost than TCAM based table storage).

Regarding the time needed to establish a path in such a large datacenter network, it will be the same that it takes for the ARP Request/Reply dialog to complete (i.e. Round Trip Time, RTT). In these topologies and in normal load conditions (where queueing delay does not vary a lot with load) maximum RTT is independent of the network size in terms of number of servers, e.g. in VL2, the number of links to be traversed is six and the number of traversed switches is five (two ToRs, two aggregation and one core switch). As stated in section 4.3, the processing time per switch is quite similar to standard bridges, performed also in hardware.

5. Simulation Results

We have implemented both SPB and ARP-Path in OMNeT++, focusing the comparison on load distribution capabilities and latencies for both protocols. Some preliminary results can be found in [11, 14]. The topology under study is shown in Fig. 7. We have chosen this network scenario as it is expected that SPB ECMP performs very well due to the even traffic distribution used. It comprises a core section of four meshed switches and ten ToR or access switches (five at each side of the core) connected to the two core switches facing each access side (there are four 3-hop paths reaching from every access switch in one side to every switch at the opposite access side). We have considered 250 servers, having 25 servers connected to each access switch. All links are 100 Mbit/s and switches have

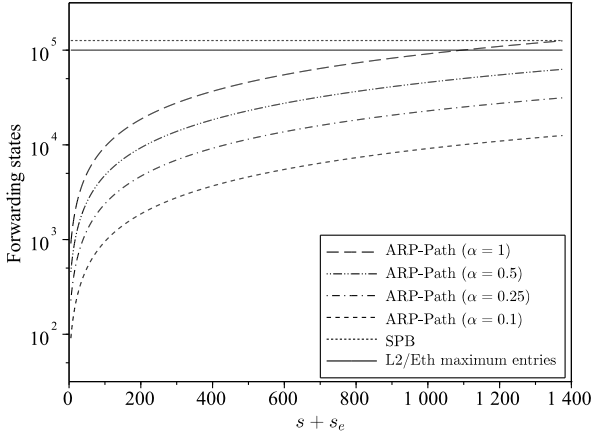


Figure 6: Comparison in the number of table entries ($\chi = 5$).

$2\mu s$ processing time. A single flow generator (with exponentially distributed flow interarrival times with rate λ) installs new flows in the network by randomly selecting a pair of source and destination servers. Flow sizes are Pareto distributed with mean 35Mbytes [22]. Each simulation runs for 10000 s. SPB routes were pre-computed by evenly assigning each source/destination server pair to one of the four possible shortest paths. It must be noted that, given the random distribution of traffic source and destination, the scenario depicted takes full advantage of SPB-ECMP multipath capabilities, i.e. in the chosen scenario SPB is expected to behave very well.

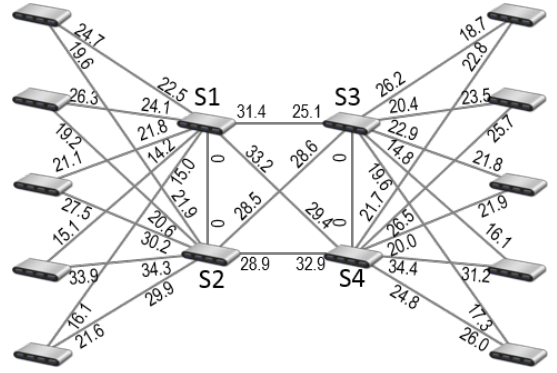
5.1. Load distribution

The first analysis is about the load distribution capabilities of both SPB and ARP-Path. In Fig. 7 we show the loads at the different links for both protocols for the specific case of $\lambda^{-1} = 0.4s$, although other values of λ studied produced similar results. As the links are 100Mbit/s each, that number can be interpreted as either the load of the link in Mbit/s or in percentage. The values show the incoming load at each port. Analyzing those figures, we can conclude that load is more evenly distributed in ARP-Path than in SPB. It is also important to note that SPB does not use some links (links S1-S2 and S3-S4 are empty).

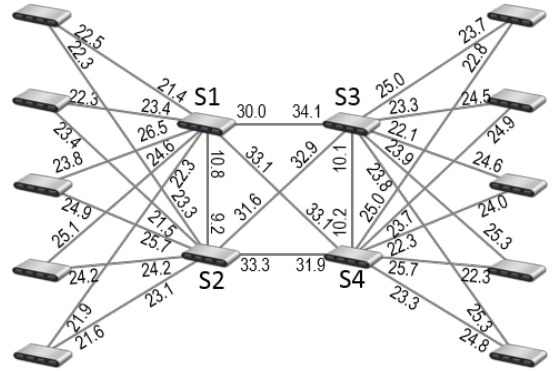
5.2. Packet latency

The second analysis deals with the end-to-end delay experienced by packets. For each host we compute the average delay experienced by packets sent to that host, but also the maximum delay measured at this node with $\lambda^{-1} = 0.4s$.

At first glance, average delays seem similar for both protocols, as shown in Fig. 8a and Table 3. Moreover, as the scenario chosen is tailor-made for SPB, it was not expected that ARP-Path could improve SPB performance. However, we noticed that a small number of destinations



(a) SPB.



(b) ARP-Path.

Figure 7: Load distribution for $\lambda^{-1} = 0.4s$.

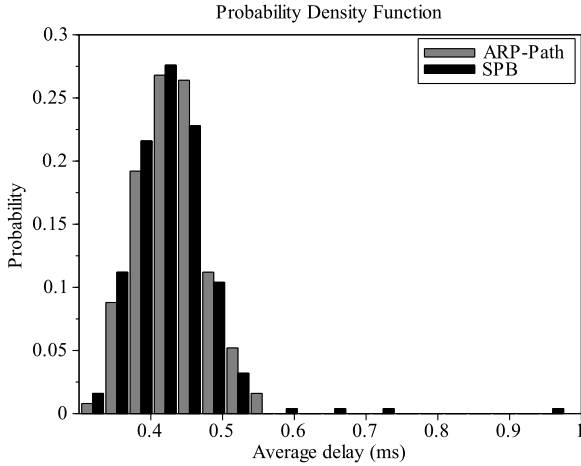
showed a noticeable higher average delay in SPB than in ARP-Path. To study this behavior in detail, in Fig. 8b we show a Q-Q plot that shows the percentiles of the average delay, where a point in the figure represents the n-th percentile for the average delay in ARP-Path (SPB) in the x-coordinate (y-coordinate). This figure shows that most of the percentiles are very similar for both protocols (they are very close to the line with slope equal to one), but there are some percentiles that fall far from this line, because they are higher in SPB than in ARP-Path. For that reason, we can conclude that the average delay in SPB for some destinations is undesirably higher than in ARP-Path, even in a scenario that is considered to be especially suited for SPB.

Figure 9 depicts the probability density function of the maximum delays experienced by packets for each destination. It can be concluded that the maximum delays in SPB are much higher than those obtained in ARP-Path (more than an order of magnitude). Moreover, maximum delays in ARP-Path are bounded, so it is more appropriate for time-sensitive applications. While for SPB the delay can become very high for some packets, which may be unacceptable for certain applications. Even when the number of packets that present this undesirable behavior is small, we show that ARP-Path does not present this drawback.

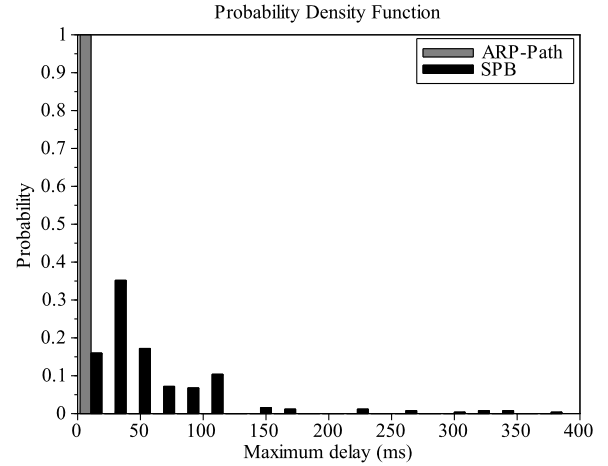
We have also studied the scenarios with $\lambda^{-1} = 0.8s$

Table 3: Comparison of delays in SPB and ARP-Path (expressed in milliseconds).

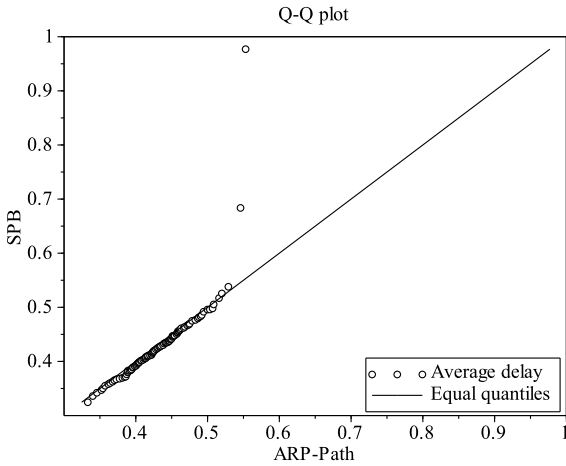
		$\lambda^{-1} = 0.4s$		$\lambda^{-1} = 0.8s$		$\lambda^{-1} = 1.6s$	
		SPB	ARP-Path	SPB	ARP-Path	SPB	ARP-Path
Average delay	Mean	0.43	0.43	0.39	0.39	0.34	0.34
	Std dev	0.06	0.05	0.07	0.06	0.07	0.07
	95th percentile	0.51	0.51	0.49	0.49	0.46	0.47
Maximum delay	Mean	58.12	1.32	18.15	0.95	1.88	0.81
	Std dev	60.77	0.13	27.34	0.08	2.42	0.07
	95th percentile	158.4	1.58	48.41	1.10	5.46	0.92



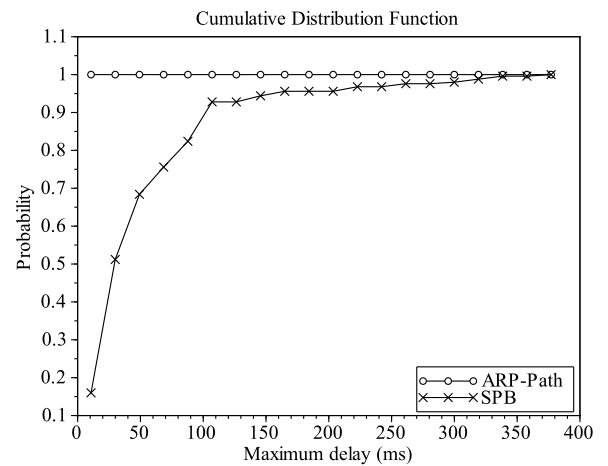
(a)



(a)



(b)



(b)

Figure 8: Average delay for $\lambda^{-1} = 0.4s$.

Figure 9: Maximum delay for $\lambda^{-1} = 0.4s$.

and $\lambda^{-1} = 1.6s$ (being λ the flow arrival rate) obtaining similar results. For the sake of brevity, we summarize those results in Table 3.

6. Experimental testbed

Any protocol enabling all redundant links at layer two must verify its robustness against path fails and broadcast

loops, which can produce network meltdown. ARP-Path has been successfully implemented in a variety of platforms such as Linux using *eatables* [23] and OpenFlow [24], and validated in real world scenarios with hosts connected to Internet via university campus networks. After validation in the previous platforms, ARP-Path protocol was implemented on NetFPGA [12]. The internal latencies obtained

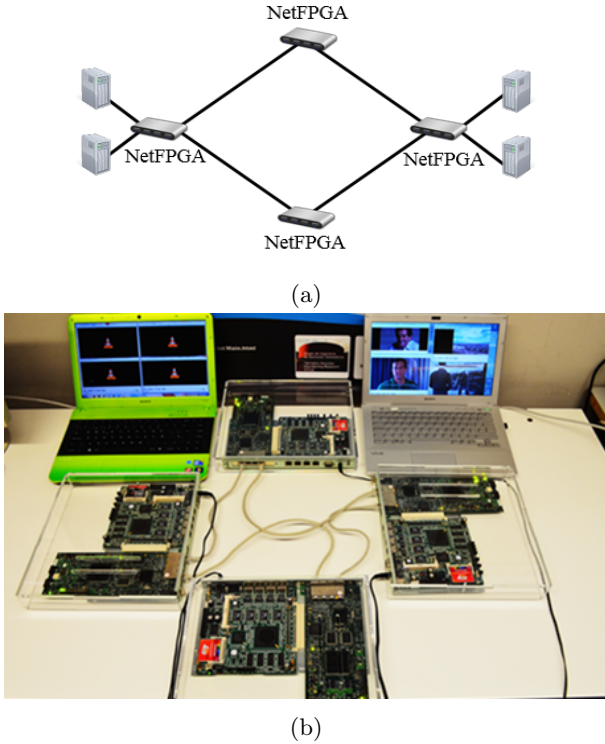
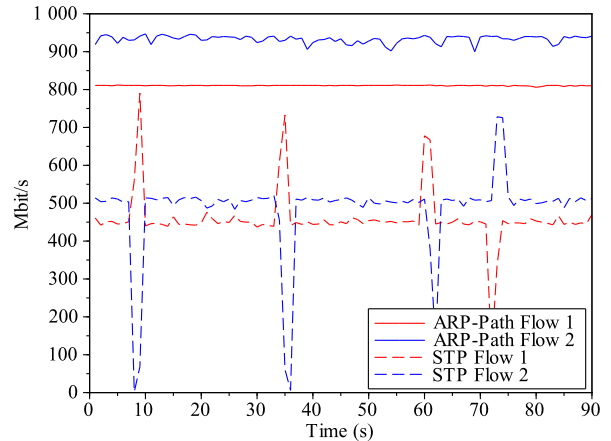


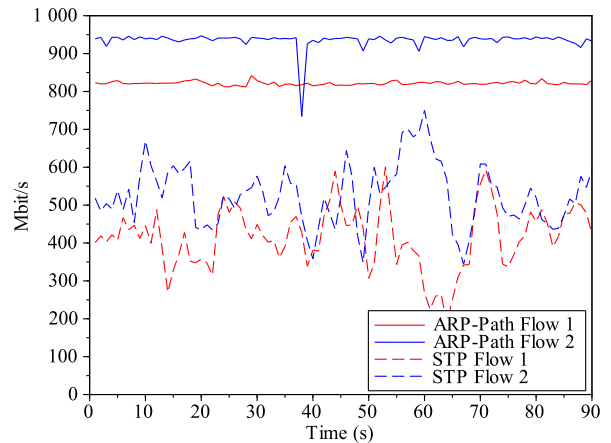
Figure 10: Four All-Path switches network on NetFPGAs.

are those typical of a switch implemented on a NetFPGA, and similar to commercial switches, such as D-link DGS-1008A.

The effect of load distribution has been verified in a single square four-switch network (Fig. 10). UDP and TCP flows from hosts connected at one switch to hosts attached to the opposite switch were established with *iperf* [25]. More precisely we have done an experiment sending two UDP flows (one flow from the upper host in the left to the upper host in the right and another flow from the lower host in the left to the lower host in the right) and another similar experiment but with TCP flows. It is important to note that flows were generated at 1 Gbit/s each and the NetFPGA model is 1G (supports up to 1 Gbit/s), however hosts limit the maximum throughput to around 800 – 950 Mbit/s. According to Fig. 11a, load can reach the maximum link limits with ARP-Path without significant packet loss (0.12%), being that possible as flows go through parallel paths in the network and do not compete for bandwidth. On the other side, when STP is used, one link is disabled to prevent loops, cutting one of the two parallel paths and thus limiting the maximum per flow capacity approximately to half (500 Mbit/s) as bandwidth must be shared between the two server links. For that reason in the scenario using STP and UDP packet losses probabilities are around 44% at each host. Similar results to those obtained for UDP but for TCP are shown in Fig. 11b.



(a) UDP.



(b) TCP.

Figure 11: Throughput at each receiver host over time using ARP-Path and STP.

7. Protocol Evolution and Variants

In this paper, we have focused our research, analysis and experiments on the first protocol of the All-Path family: ARP-Path. The ARP-Path protocol implements path set up at host level, that is, for each host we explore all possible paths and select one by snooping the information conveyed in the ARP messages exchanged previously to any communication. This selected path is the fastest one since slower copies of the ARP messages are blocked. At the same time, ARP-Path distributes load in the network because paths will usually be created over the most underutilized network resources (switches and links), which will provide the lowest latencies. In ARP-Path and as mentioned in section 4.1, paths to a destination host are also shared for different source hosts (once a path is created, new paths will just join it as if branches to a tree were added).

After implementing ARP-Path, several variants have been explored by changing different characteristics of ARP-Path. One of this variants is the second protocol of the All-Path family: Flow-Path. Flow-Path also learns from the exchange of ARP messages, but it creates unique paths per couple of hosts or per flow, that is, in Flow-Path paths to a destination host from different sources are not shared anymore and can be completely disjoint. This is an advantage of Flow-Path respect to ARP-Path which can be of great interest in networks with hot spots (i.e. hosts with high traffic or heavily loaded servers), but at the same time it requires a higher number of table entries because granularity in Flow-Path is higher than in ARP-Path.

The third protocol of the All-Path family is Bridge-Path, which creates paths per edge or ToR switch. Its main objective is reducing the number of table entries in the network, especially when the number of host per edge switch is very high. In Bridge-Path it is necessary to distinguish the edge switches with some identifier or MAC address, and to use some type of encapsulation. In this case we have designed Path-Moose [26], which uses hierarchical switch addressing of the form switchID:hostID, and MiM-Path, which uses MAC-in-MAC encapsulation as in SPBM [4]. In Bridge-Path, specific broadcast or multicast messages can be sent to create the paths between edge switches (instead of the ARP messages), so that edge switches can act as ARP proxies saving broadcast messages, and therefore only emitting the necessary messages to explore and build the paths.

8. Conclusions

All-Path bridging is a new routing paradigm for campus and data center networks based on simultaneous exploration of all network paths. Compared with path computation protocols, ARP-Path is simple, scalable and resilient. We have shown that the number of forwarding states per switch in ARP-Path respect to a link-state protocol is usually one or more orders of magnitude lower, so its implementation requirements are lower. Moreover, ARP-Path, without any additional load balancing mechanism, naturally distributes traffic among alternative paths more precisely than SPB and ECMP and with better average and maximum delays.

Path-probing mechanisms like ARP-Path for routing in layer two create a new design space for the evolution of switches focused on the simplicity of probing the network either reactively or proactively. Their simplicity, performance and resiliency make them attractive in a variety of scenarios like audio video bridges, enterprise and data center networks.

Acknowledgment

This article has been partially supported by the Madrid Regional Government through the TIGRE5-CM program (S2013/ICE-2919).

References

- [1] N. Maksic, A. Smiljanic, Improving utilization of data center networks, *IEEE Communications Magazine* 51(11) (2013) 32-38.
- [2] T. Benson, A. Akella, D.A. Maltz, Network traffic characteristics of data centers in the wild, In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement (IMC)*, (2010) 267-280.
- [3] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, VL2: a scalable and flexible data center network, *ACM SIGCOMM Computer Communication Review - SIGCOMM '09* 39(4) (2009) 51-62.
- [4] IEEE Shortest Path Bridging. <http://www.ieee802.org/1/pages/802.1aq.html>
- [5] D. Allan, N. Bragg, 802.1aq Shortest Path Bridging Design and Evolution: The Architect's Perspective, John Wiley & Sons (2012).
- [6] IETF Transparent interconnection of lots of links (TRILL) WG. <http://www.ietf.org/html.charters/trill-charter.html>
- [7] C. Kim, M. Caesar, J. Rexford, Floodless in SEATTLE: a scalable ethernet architecture for large enterprises, *ACM SIGCOMM Computer Communication Review* 38(4) (2008) 3-14.
- [8] D. Sampath, S. Agarwal, J.J. Garcia-Luna-Aceves, "Ethernet on AIR": Scalable Routing in very Large Ethernet-Based Networks," *IEEE 30th International Conference on Distributed Computing Systems (ICDCS)* (2010) 1-9.
- [9] C. Perkins, E. Belding-Royer, S. Das, RFC 3561: Ad hoc On-Demand Distance Vector (AODV) Routing. IETF (2003).
- [10] J. Farkas, Z. Arató, Performance analysis of shortest path bridging control protocols, In *Proceedings of GLOBECOM 2009* (2009) 4191-4196.
- [11] G. Ibáñez, J.A. Carral, J.M. Arco, D. Rivera, A. Montalvo, ARP Path: ARP-based shortest path bridges, *IEEE Communication Letters* 15(7) (2011) 770-772.
- [12] E. Rojas, J. Naous, G. Ibáñez, D. Rivera, J.A. Carral, J.M. Arco, Implementing ARP-path low latency bridges in NetFPGA, In *Proceedings of the ACM SIGCOMM 2011 conference* (2011) 444-445.
- [13] D. Bertsekas, A. Gallager, *Data Networks*, second ed., Prentice Hall, 1992.
- [14] G. Ibáñez, J.A. Carral, A. García-Martínez, J.M. Arco, D. Rivera, A. Azcorra, Fast Path Ethernet Switching: On-demand Efficient Transparent Bridges for Data Center and Campus Networks, In *Proceedings of 17th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN)* (2010).
- [15] A. Ghanwani, H. Shah, N. Bitar, ARP Broadcast Reduction for Large Data Centers. IETF Internet Draft (2012).
- [16] D. Medhi, K. Ramasamy, *Network Routing: Algorithms, Protocols, and Architectures*, Morgan Kaufmann Publishers, 2007.
- [17] M. Barbehenn, A Note on the Complexity of Dijkstra's Algorithm for Graphs with Weighted Vertices, *IEEE Transactions on Computers* 47(2) (1998) 263.
- [18] D. Allan, P. Ashwood-Smith, N. Bragg, D. Fedyk, Provider link state bridging, *IEEE Communications Magazine* 46(9) (2008) 110-117.
- [19] D. Allan, J. Farkas, S. Mansfield, Intelligent load balancing for shortest path bridging, *IEEE Communications Magazine* 50(7) (2012) 163-167.
- [20] B. Stephen, A. Cox, W. Felter, C. Dixon, J. Carter, PAST: Scalable Ethernet for Data Centers, In *Proceedings of CoNEXT'12* (2012) 49-60.
- [21] D. Li, P. Chen, Optimized Hash Lookup for Bloom Filter Based Packet Routing, 16th International Conference on Network-Based Information Systems (NBIS) (2013) 31-37.
- [22] R.S. Prasad, C. Dovrolis, Beyond the Model of Persistent TCP Flows: Open-Loop vs Closed-Loop Arrivals of Non-persistent Flows, In *Proceedings of 41st Annual Simulation Symposium 2008 (ANSS)* (2008) 121-130.
- [23] G. Ibáñez, B. De Schuymer, J. Naous, D. Rivera, E. Rojas, J.A. Carral, Implementation of ARP-path low latency bridges in Linux and OpenFlow/NetFPGA, In *Proceedings of IEEE 12th Interna-*

tional Conference on High Performance Switching and Routing (HPSR) (2011) 30-35.

[24] G. Ibáñez, J. Naous, E. Rojas, D. Rivera, J.A. Carral, J.M. Arco, A Simple, Zero Configuration, Low Latency, Bridging Protocol, In Proceedings of the 35th IEEE Conference on Local Computer Networks (LCN), demo (2010).

[25] Iperf: The TCP/UDP bandwidth measurement tool, <http://iperf.fr>

[26] G. Ibáñez, I. Marsá-Maestre, M.A. López-Carmona, I. Pérez-Ibáñez, J. Tanaka, J. Crowcroft, Path-Moose: A Scalable All-Path Bridging Protocol, IEICE TRANSACTIONS on Communications E96-B(3) (2013) 756-763.