

# PAST-HOAS: Fast Setup of Per-Address Sink Routing Trees by Reverse Flooding

Joaquin Alvarez-Horcajo, Diego Lopez-Pajares, Elisa Rojas, Guillermo Ibanez,  
Isaias Martinez-Yelmo and Maria Campos-Selfa

Departamento de Automatica, University of Alcalá, Alcalá de Henares (Madrid), Spain.

Email: j.alvarez@uah.es, diego.lopezp@uah.es, elisa.rojas@uah.es, guillermo.ibanez@uah.es,  
isaias.martinezy@uah.es and maria.campos@uah.es

**Abstract**—Scaling Ethernet switched networks to sizes of hundreds of thousands of hosts is a key requirement for data center networks. PAST is a simple and scalable OpenFlow-based layer-two architecture for data center Ethernet networks that outperforms Equal Cost Multipath Forwarding. PAST computes and configures one spanning tree for every MAC destination address. However, an important bottleneck of PAST is the time taken to perform the massive installation of rules at OpenFlow switches to set up the trees. PAST-HOAS aims to solve that obstacle, building trees instantly by path discovery, either triggered by the controller with a multicast probe frame from the edge destination switch or by the standard ARP Request frame; without flow installation and without tree computation. PAST-HOAS is loop-free, and provides 100-400 times faster network initialization, lower forwarding delays and increased robustness and scalability.

**Index Terms**—Shortest Path Bridges, Spanning Tree, Ethernet, Hybrid SDN.

## I. INTRODUCTION

Tree-based forwarding is particularly suited for data center traffic applications like MapReduce [1], involving one-to-many/many-to-one communications. PAST [2] is a tree-based proposal that leverages Software-Defined Networking (SDN) and implements OpenFlow switches that expose layer-two (L2) tables to the SDN controller to improve scalability in data center networks. Specifically, PAST's scalability is the consequence of using SRAMs (exact field matching) instead of the size-limited and power-hungry TCAMs required for looser matching. PAST computes and installs per-address spanning trees, one tree per MAC address. Its tree computation is configurable: it may be based on shortest-path with random link selection, random selection with link weighting to add a load effect, or may use non-minimal path random selection with or without link weighting. PAST is not only much simpler than ECMP but –surprisingly– its performance is also comparable [2].

Nevertheless, the main bottleneck of PAST is the deployment of rules at the OpenFlow switches. Installing rules in all switches to set up one tree per address (upon network initialization or reconfiguration) is sequential and takes significant time, not only at the controller, but also at the involved switch: from 2 to 12 ms to install a rule, leading to around 1600 rules installed per second maximum. Updating the paths for a 100K-host data center network requires up to one minute [2].

To prevent overloading the SDN controller, an alternative would be delegating part of the process to the switches, in charge of installing the per-address spanning trees, for instance via a multicast probe frame reaching all participant nodes in the network. Hybrid SDN switches, such as Hybrid OpenFlow ARP-Path Switch (HOAS) [3], allow this procedure while keeping OpenFlow compatibility. HOAS implements a cooperative mechanism where the controller triggers the setup of per-address spanning trees from the edge switches by sending a multicast probe frame flooded through the network to discover the forwarding tree. HOAS is based on the loop-free discovery protocols already implemented in NetFPGA [4], Linux [5], or OpenFlow [6].

This paper describes PAST-HOAS, a protocol which combines PAST forwarding based on per-address spanning trees (MAC learning) with the forwarding mechanisms of HOAS switches. PAST-HOAS improves PAST with faster setup of trees, increased robustness (frame forwarding operates even under controller failure) and increased scalability (by greatly reducing controller intervention). Additionally, PAST-HOAS guarantees no loops (PAST may have transient loops) [2].

## II. PAST-HOAS PER-ADDRESS SINK TREES

In this section, we describe how the PAST-HOAS protocol creates per-address sink trees. The first section explains its foundations and basic forwarding, while the following ones define how PAST-HOAS analyzes other issues, such as host mobility or compatibility with other switches.

### A. Tree set-up and forwarding

PAST-HOAS builds per destination address edge-based spanning trees. To achieve higher scalability than PAST, paths are dynamically built and deleted, instead of statically created after initialization. This procedure can be performed following two modes, namely *proactive* and *reactive*. For this process, PAST-HOAS requires a new packet, named *Set-Tree*, which is the multicast probe frame that creates the convergence tree towards the edge switch, illustrated in Fig. 1.

1) *Reactive mode*: In the reactive (on-demand) mode, the spanning tree is created (without any controller intervention) by flooding and applying MAC address learning at switches, using the standard ARP Request packets sent by every host as described in [3]. Alternatively, the controller might decide

when to start the process and trigger the creation of the trees just by sending to the hosts edge switch a multicast *Set-Tree* packet encapsulated in a `PACKET_OUT` standard OpenFlow message, as shown in Fig 2. In both cases, the edge switch forwards the frame through all switch ports towards the network. A sink tree rooted at the edge switch is created by *locking* the source address (i.e. the first port receiving the frame locks the association of this address to the port to prevent broadcast loops [4]).

Once created by the ARP Request or by the *Set-Tree*, sink trees stay active while the persistence timers of the address are active at the switches. The frames forwarded by the switches via the active tree branches towards the root of the tree (forward) refresh the path at every port traversed. Tree branches not traversed by traffic will be automatically erased after a timer expires, reducing the stored state in the switches.

2) *Proactive mode*: In the proactive mode, the controller needs to know the edge switch to which is connected every host in the network, to send the *Set-Tree* packet to it afterwards. To achieve this, every edge switch informs the controller of its active hosts directly connected. Each edge switch keeps an updated list of directly attached active hosts by monitoring the ARP Requests received and refreshing the corresponding timer associated with each MAC. Edge switches know which ports have directly connected hosts (host ports) because a port connected to a switch receives periodic *Hello* packets from its neighbor switch. Therefore, the switch concludes that the active ports of a switch not receiving *Hello* packets are connected to hosts and the switch keeps a list of active hosts directly connected to it. Every switch has a forwarding table that uses simple MAC addresses to select the output port. In HOAS [3], this table is learnt autonomously, fully separated from OpenFlow tables and is used by default to forward frames if no rule is already populated by the controller for this address.

If a unicast frame arrives to a switch and there is not a tree associated with its destination MAC, the above-mentioned controller assisted path recovery is performed: the frame is forwarded to the controller via a `PACKET_IN`; the controller internally searches the edge switch associated with this address and sends back (via a `PACKET_OUT`) a *Set-Tree* for path recovery. However, this situation will seldom occur because, in addition to the reactive tree set up or refresh performed with every ARP Request, the trees are refreshed periodically (e.g. every 30 minutes) for all active hosts, triggered remotely and sequentially from the controller, evenly-spaced across time.

#### B. New host connected / Host mobility / VM migration

When a new host is connected to the network, a new tree must be created, rooted at the host's edge switch. The associated tree towards this new host will be created when the

| Multicast Address | Source Address | PAST-HOAS EtherType |
|-------------------|----------------|---------------------|
| 0                 | 6              | 12                  |
|                   |                | 14                  |

Fig. 1: Set-Tree packet header.

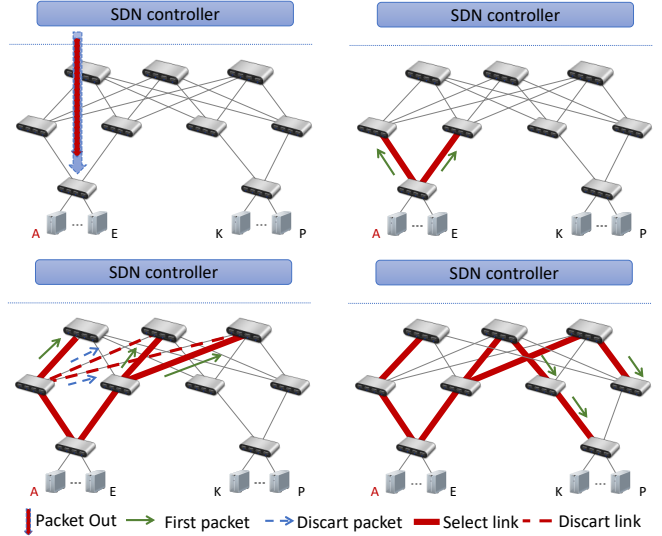


Fig. 2: Path building in PAST-HOAS.

host sends a normal ARP Request to locate a destination host. The ARP Reply from the interrogated host will set up the path to the destination host and the new branch will be added to the spanning tree of the destination host. The same will occur for the rest of the destination addresses.

If a host moves to a different edge switch, or a virtual machine is moved to another host, the gratuitous ARP Request will update the trees accordingly. More specifically, the controller will send a new *Set-Tree* packet to the edge switch, to force the creation of a new tree for the recently attached host. Every switch reached by the *Set-Tree* packet will unconditionally label the first port that receives it as root for the tree. In this way, the previous tree is replaced by the new one rooted at the new edge switch.

#### C. Compatibility and mixability

Like PAST, PAST-HOAS is transparent to hosts as it uses the standard Ethernet frame format and semantics. Overlays like VXLANs or NetLord [7] will operate without changes.

Regarding mixability of switch types, like in PAST, all network switches must support the same PAST-HOAS protocol. To alleviate this restriction and to provide mixability with standard OpenFlow switches, the controller may centrally execute the ARP-Path protocol for those non-HOAS switches. This allows several switches in the network to be exempt from supporting PAST-HOAS, although the latency and controller load advantages would be lost in those switches. This PAST-HOAS variant with fully centralized forwarding reuses an existing centralized implementation of the ARP-Path protocol [6]. In this case, the tree building through multicast flooding is performed by the OpenFlow controller (i.e. MAC learning and forwarding logic is fully implemented in the controller instead of the switch). A local controller (in place of a single central controller) could be collocated with every standard OpenFlow switch to manage locally the tree building

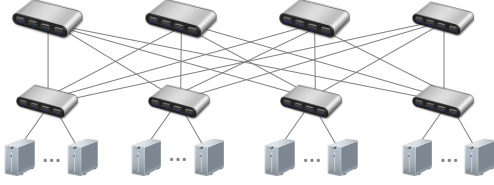


Fig. 3: Spine-Leaf 4-4-20 evaluation topology.

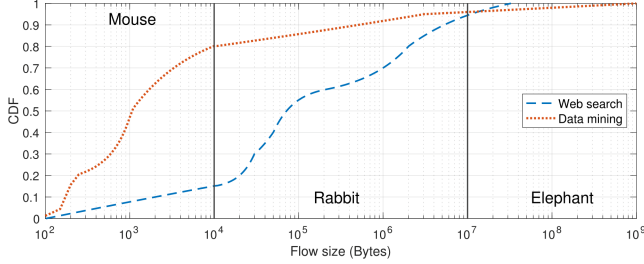


Fig. 4: Flow size distributions.

and maintenance, while the central controller takes charge of the ARP Request packet resolution and other management, configuration and monitoring functions. Neither the central controller nor the local controller need topology information, apart from the attached hosts list.

### III. EVALUATION

To evaluate PAST-HOAS, we implemented and simulated our protocol in *ns-3* [8]. In this section, we compare it with PAST in terms of throughput, setup time and scalability.

#### A. Testbed and Experimental Setup

The network topology simulated is a 4-4-20 *Spine-Leaf* [9], [10], consisting of two rows of 4 switches, connecting 20 servers per leaf switch for a total of 80, as depicted in Fig. 3.

Traffic flows are randomly distributed between any pair of servers attached to two different leaf switches. Flow size distributions follow [13]. Fig. 4 shows the CDF of both distributions and also illustrates how flows are classified according to their size. Flows with less than 10 KB of data are considered mouse flows, while those carrying more than 10 MB are considered elephant flows, as explained in [14]. Lastly, we inserted enough traffic to the network to establish an offered traffic of 10%, 20% and 40% of the maximum allowed by the network, according to either the *Web search* or

*Data mining* flow size distributions. Each experiment runs for 1800 seconds and is repeated 10 times to later compute 95% confidence intervals with an error smaller than 3.42%. Thus, further repetitions are not necessary. Additionally, we consider a warm-up time of 800 seconds to mitigate any transitory effect on the results. Table I summarizes the full setup of the conducted experiments.

#### B. Throughput

To evaluate the throughput of PAST-HOAS, we compared it to ECMP, since PAST claims to equal its performance [2]. Figure 5 illustrates the results obtained in the throughput tests. The left column represents the results for the *Web search* flow size distribution, while the right one shows the results for *Data mining*. The throughput obtained by PAST-HOAS is very similar to the one obtained by ECMP and, therefore, by PAST.

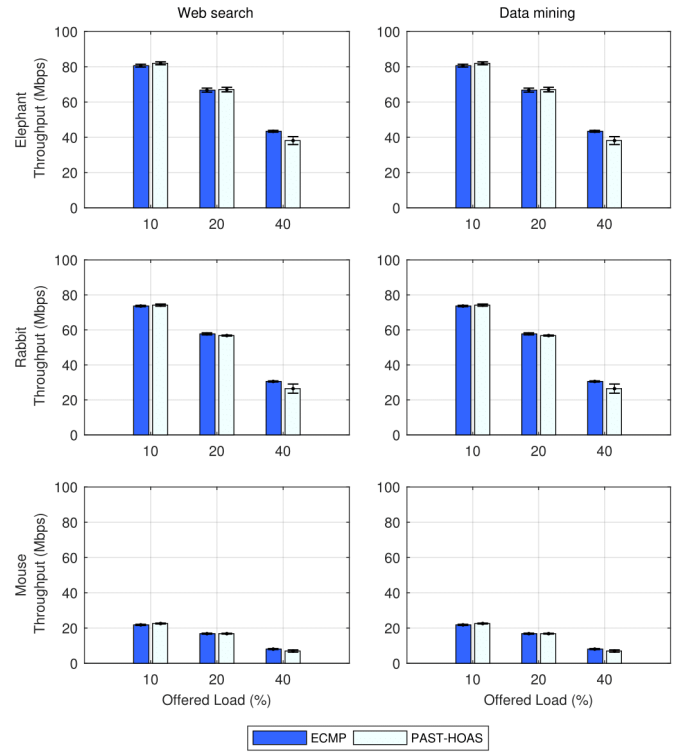


Fig. 5: Throughput on Spine-Leaf 4-4-20 100 Mbps topology.

#### C. Setup network time

PAST requires a relatively long setup time to install the routes, which might delay the initialization of the network estimated in 60 seconds [2]. Differently from PAST, PAST-HOAS does not consume initialization time as it builds the different trees upon request.

Furthermore, PAST calculates one tree per each connected host, while PAST-HOAS only creates it for the active network hosts. This logic of PAST causes that the performance decreases as the network increases, which provokes high recalculation times (i.e. 40 seconds for 100.000 hosts). Contrarily, PAST-HOAS does not suffer from this problem as it only

TABLE I: Experimental Setup

|                          |                                    |
|--------------------------|------------------------------------|
| Network topology         | Spine-Leaf (4 - 4) [9]             |
| Servers per leaf switch  | 20                                 |
| Flow distribution        | Random inter-leaf                  |
| Flow size distributions  | Web search [11] & Data mining [12] |
| Network offered load (%) | 10, 20 & 40%                       |
| Link speed (Mbps)        | 100Mbps                            |
| Run length (s)           | 1800 s                             |
| Warm up time (s)         | 800 s                              |
| Number of runs           | 10                                 |

depends on the active hosts and paths are created and deleted dynamically.

#### D. Scalability

In contrast to PAST, PAST-HOAS does not insert one entry per host in each switch (it installs one only per *active* host). As a consequence, the number of table entries per switch remains stable even when the number of hosts is increased. Figure 6 illustrates the number of table entries per switch for PAST and PAST-HOAS in relation to the number of hosts per leaf switch. Analyzing the graph, we can observe that PAST grows linearly with the number of hosts, while PAST-HOAS does not. Hence, PAST-HOAS guarantees higher scalability than PAST.

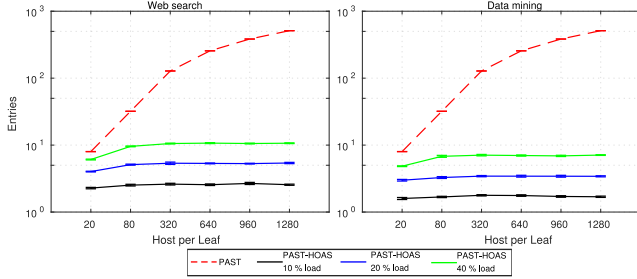


Fig. 6: Table size per switch in a Spine-Leaf 4-4 (100 Mbps).

#### E. Validation of results

We repeated all the testbed and experimental setup in a VL2 topology [14] in order to validate the previous results. The topology chosen has 2 core, 4 aggregated and 4 ToR switches, and in each ToR we connect from 20 to 1280 servers as the previous test. The throughput results show similar values as in the Spine-Leaf topology: a bandwidth decrease is observed when the network load grows, and elephants are the flow type that achieves the best bandwidth. Figure 7 shows the table size per switch when the number of servers increases. As the previous subsection, PAST grows linearly, while PAST-HOAS does not. These results confirm that PAST-HOAS guarantees the scalability regardless of the topology.

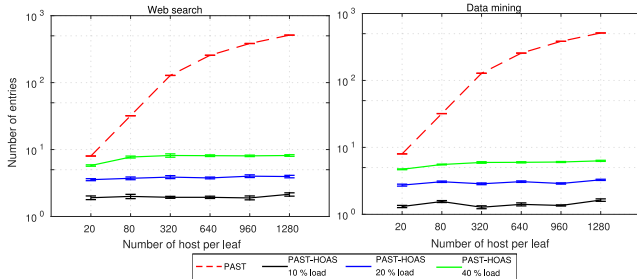


Fig. 7: Table size per switch in a VL2 4-4 (100 Mbps).

#### IV. CONCLUSION

In this article, we present PAST-HOAS, a protocol that creates per-address sink routing trees without controller intervention. PAST-HOAS might be developed as a stand-alone

switch and as a hybrid SDN switch, proving that the protocol is compatible with legacy or OpenFlow switches.

We implemented and evaluated PAST-HOAS in the ns-3 simulator and the results are promising, as expected. The dynamic building approach of PAST-HOAS drastically reduces setup time and table entries, thus boosting scalability compared to PAST, while ensuring high throughput.

#### ACKNOWLEDGMENT

This work was funded by a grant from Comunidad de Madrid through Project TIGRE5-CM (S2013/ICE-2919) and the “Formacion del Profesorado Universitario (FPU)” program from the University of Alcalá.

#### REFERENCES

- [1] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- [2] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, “PAST: Scalable Ethernet for Data Centers,” in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 49–60.
- [3] J. Alvarez-Horcajo, I. Martinez-Yelmo, E. Rojas, J. A. Carral, and D. Lopez-Pajares, “New cooperative mechanisms for software defined networks based on hybrid switches,” *Transactions on Emerging Telecommunications Technologies*, pp. e3150–n/a, 2017.
- [4] E. Rojas, G. Ibanez, J. M. Gimenez-Guzman, J. A. Carral, A. Garcia-Martinez, I. Martinez-Yelmo, and J. M. Arco, “All-Path bridging: Path exploration protocols for data center and campus networks,” *Computer Networks*, vol. 79, no. 0, pp. 120 – 132, 2015.
- [5] G. Ibanez, B. D. Schuymer, J. Naous, D. Rivera, E. Rojas, and J. A. Carral, “Implementation of ARP-path low latency bridges in Linux and OpenFlow/NetFPGA,” in *2011 IEEE 12th International Conference on High Performance Switching and Routing*, July 2011, pp. 30–35.
- [6] E. Rojas, J. Naous, G. Ibaez, and C. Recio, “A Small Data Center Network of ARP-Path Bridges made of Openflow Switches,” in *The 36th IEEE Conference on Local Computer Networks (LCN)*, Bonn, Germany, Oct. 2012.
- [7] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary, “Netlord: a scalable multi-tenant network architecture for virtualized datacenters,” in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 62–73.
- [8] *ns-3 simulator*. [Online]. Available: <https://www.nsnam.org/>
- [9] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, “CONGA: Distributed Congestion-aware Load Balancing for Datacenters,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 503–514, Aug. 2014.
- [10] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, “Presto: Edge-based Load Balancing for Fast Datacenter Networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 465–478, Aug. 2015.
- [11] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, “Data center tcp (dctcp),” in *ACM SIGCOMM computer communication review*, vol. 40. ACM, 2010, pp. 63–74.
- [12] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, “VL2: a scalable and flexible data center network,” *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, 2009.
- [13] J. Alvarez-Horcajo, D. Lopez-Pajares, J. M. Arco, J. A. Carral, and I. Martinez-Yelmo, “TCP-path: Improving load balance by network exploration,” in *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, Sep. 2017, pp. 1–6.
- [14] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, “VL2: a scalable and flexible data center network,” in *ACM SIGCOMM computer communication review*, vol. 39, no. 4. ACM, 2009, pp. 51–62.