network

MDPI

Article

Scalable and Reliable Data Center Networks by Combining Source Routing and Automatic Labelling

Elisa Rojas [1,*,†], Joaquin Alvarez-Horcajo [1,†], Isaias Martinez-Yelmo [1,†], Jose M. Arco [1] and Miguel Briso-Montiano [2]

[1] Departamento de Automática, Universidad de Alcalá, Alcalá de Henares, 28805 Madrid, Spain; j.alvarez@uah.es (J.A.-H.); isaias.martinezy@uah.es (I.M.-Y.); josem.arco@uah.es (J.M.A.)
[2] GMV Inc., Tres Cantos, 28760 Madrid, Spain; mbrisomontiano@gmv.com
* Correspondence: elisa.rojas@uah.es
† These authors contributed equally to this work.

Abstract: Today, most user services are based on cloud computing, which leverages data center networks (DCNs) to efficiently route its communications. These networks process high volumes of traffic and require exhaustive failure management. Furthermore, expanding these networks is usually costly due to their constraint designs. In this article, we present enhanced Torii (eTorii), an automatic, scalable, reliable and flexible multipath routing protocol that aims to accomplish the demanding requirements of DCNs. We prove that eTorii is, by definition, applicable to a wide range of DCNs or any other type of hierarchical network and able to route with minimum forwarding table size and capable of rerouting around failed links on-the-fly with almost zero cost. A proof of concept of the eTorii protocol has been implemented using the Ryu SDN controller and the Mininet framework. Its evaluation shows that eTorii balances the load and preserves high-bandwidth utilization. Thus, it optimizes the use of DCN resources in comparison to other approaches, such as Equal-Cost Multi-Path (ECMP).

Keywords: cloud computing; SDN; data center networks; Ethernet; source routing; automatic labelling

Citation: Rojas, E.; Alvarez-Horcajo, J.; Martinez-Yelmo, I.; Arco, J.M.; Briso-Montiano, M. Scalable and Reliable Data Center Networks by Combining Source Routing and Automatic Labelling. *Network* **2021**, *1*, 11–27. https://doi.org/10.3390/network1010003

Received: 20 May 2021
Accepted: 15 June 2021
Published: 18 June 2021

1. Introduction

The cloud computing paradigm has reached high relevance thanks to the flexibility and portability that it provides to a vast number of telecommunication systems [1]. The implementation of cloud services is performed through the deployment of data centers, composed of a set of hosts interconnected by a high-speed network. To guarantee that a cloud behaves transparently to final users, or network managers, its associated Data Center Network (DCN) should accomplish the four following requirements: scalability, flexibility, auto-configuration, and high availability (that is, resilience and fast recovery from any type of problem in the network).

During the last decade, different DCN architectures have been designed in the research community to fulfil those four desired features [2–7]. These solutions are focused on particular aspects of the previously mentioned, but none of them is able to achieve all of them at the same time. As an example, PortLand [2] guarantees easy configuration and high availability, but the flexibility of its design is limited to specific network topologies and its scalability depends on its implementation, as it is based on a logically centralized Software-Defined Networking (SDN) control plane [8].

In this article, we aim to prove that merging different existing technologies, we can completely fulfil the four requirements for an ideal DCN. In particular, we will combine a specific source routing protocol (Torii [9]) and an automatic labelling mechanism (GA3 [10]) to create what we have called enhanced Torii (eTorii). eTorii is scalable since it is implemented in a distributed manner and its routing tables are extremely reduced and independent of the amount of communication flows established in the network. At the

same time, it can be deployed in network topologies of different types and sizes, guaranteeing flexibility. It also follows a plug-and-play approach, achieving the auto-configuration feature, as it leverages the automatic labelling protocol for this purpose. Finally, the fourth requirement is accomplished thanks to its fast recovery, almost immediate (zero delay) in some scenarios, when some network elements fail.

The structure of this manuscript is as follows: First, we relate our proposal with the state of the art in Section 2. Second, Section 3 describes the main features of eTorii and its application to DCNs. Afterwards, Section 4 elaborates on the proof-of-concept of eTorii, implemented in the Ryu SDN controller, which is evaluated in Section 5. Finally, Section 6 recapitulates and concludes the work.

2. Related Work

DCNs are implemented based on two different strands, based on whether the control logic is centralized or distributed. On the one hand, centralized approaches usually leverage the SDN [11,12] paradigm. SDN allows a flexible definition of the behavior through a logically centralized controller connected to the network devices. Works like PortLand [2] o VL2 [3] are implemented with SDN. Nevertheless, this centralized control lacks scalability [8], specially to grant timely network monitoring to solve network failures [13], hence causing slow recovery times that hinders overall network performance.

On the other hand, distributed implementations try to overcome the scalability and robustness limitations of the centralized ones, but they are constrained in flexibility and ease of configuration. LESS [14] and FZS [15] follow a distributed approach and they are currently the only works that try to tackle most of these challenges. They are also the closest to eTorii, as it is based on source routing [16] together with label assignment. However, differently from eTorii, LESS requires the addition of an initial header to reach the destination, which reduces performance. Moreover, it also entails a second header containing alternative routes (even when no malfunction occurs) to overcome network failures, which is not needed in eTorii as alternative routes can be directly deduced from current routes thanks to its synchronized labelling mechanism. Finally, in the case of FZS, although it does not suffer from the disadvantages of LESS mentioned before, it is only applicable to certain DCNs, and it lacks additional features provided by eTorii such as on-the-fly path repair.

Additionally, some recent works in the state of the art focus on network resilience, which are worth mentioning. LetFlow [17] leverages flowlet switching to balance the load and to be resilient to asymmetry. This idea is supported by the elastic capabilities of flowlets to adapt to traffic. APS [18], CAPS [19] and QDAPS [20] follow a similar approach by dynamically separating flows based on different parameters, such as flow size (long vs. short) or queueing delay; thus, avoiding resource competition in congested networks. However, these previous approaches do not always guarantee timely re-routing, and are only focused on dynamically adapting the traffic in congested areas of the network. Hermes [21] acknowledges this problem and, for that reason, it defines an alternative approach based on network monitoring to carefully re-route traffic without causing additional problems due to abrupt changes. Although Hermes is the closest alternative to eTorii, the dependence on monitoring (not required in eTorii) might have potential scalability and performance issues in large DCNs.

Accordingly, the main contributions of eTorii are:

- Differently from most of the state of the art, it follows a distributed approach to maximize scalability and resilience.
- Packet forwarding follows a source-routing approach based on previously assigned labels, but no additional headers or fields are required, not even for network recovery upon failures.
- Thanks to the nature of its labels, routing tables are drastically reduced, independently of the active communication flows, and network recovery can be executed even on-the-fly (zero delay) in some scenarios.

- Labels are automatically assigned at startup and after any network change, granting auto-configuration.
- It is generalized and applicable to any hierarchical DCN, guaranteeing flexibility.

3. The eTorii DCN

The eTorii DCN is designed to accomplish the four desired features: auto-configuration, scalability, resilience and flexibility. To this purpose, in this Chapter we devote one section per feature to comprehensively define eTorii.

3.1. Auto-Configuration: Automatic Label Assignment Based on GA3

In order to route traffic (which will be described in the next sections), eTorii needs to assign at least one hierarchical label to each network device or switch. This assignment is only required when the network is deployed the very first time and upon network modifications (addition/deletion/movement of links/switches). Each label represents an Hierarchical Local MACs (HLMACs) address [9], which is used afterwards for masking real physical MAC addresses. An HLMACs contains 6 bytes, as a standard MAC, and each byte represents a position in the hierarchy. For instance, 01:04:07:00:00:00 stands for position 1 for the first level, 4 for the second and 7 for the third; for simplicity, it is expressed as 1.4.7, omitting the zeroes. In this way, addresses become meaningful and are able to convey information about the location of the different hosts and switches in the network, following a source routing approach.

In Torii, this assignment was performed leveraging the Rapid Spanning Tree Protocol (RSTP) with some modifications. As such, labelling was relatively slow and required a high number of control messages. Furthermore, RSTP did not allow the synchronization of the HLMAC suffixes, which is explained in the following paragraphs as one of the enhancements provided by eTorii. Therefore, this labelling is performed by GA3 [10] in eTorii instead, which follows the principles of meshed trees [22,23]. The control frames of both RSTP and GA3 are illustrated in Figure 1.

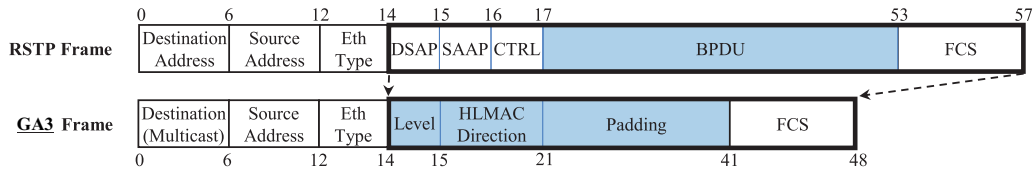


Figure 1. Frame comparison between eTorii and RSTP.

GA3 assigns one or more HLMACs to each switch in the network, which portrays one or more hierarchical positions of the switch. The procedure starts at the top-level network devices, also known as core switches. Each of these core switches obtains an initial HLMAC with only their first byte set to a value, such as 1 and 2, as depicted in Figure 2 for switches *S*1 and *S*2, respectively. This initial selection and assignment of the core switches is independent of the protocol, and could be any desired by the network manager.

Once the core switches start up, they flood GA3 frames through all their ports to start labelling the rest of nodes. These frames contain their HLMAC plus a suffix, which could be any value as far as it is unique per port. For example, in Figure 2, switch *S*1 has HLMAC with value 1 and, accordingly, it sends 1.7 to *S*3, 1.6 to *S*4, and so on; while *S*2 sends 2.2 to *S*3, 2.1 to *S*5, etc. Subsequent switches in the hierarchy will receive one or more HLMACs (e.g., *S*3 obtains 1.7 and 2.2). These switches follow the same procedure as soon as they receive their first HLMAC: they forward a GA3 frame containing their HLMAC plus a unique suffix per port. For instance, *S*3 will send 1.7.6 and 2.2.6 to switch *S*7. The flooding of GA3 frames is performed until all network nodes are reached. This process is not performed if the switches receive HLMACs longer than the ones previously obtained, as they discard them to avoid loops [22,23]. The resulting assignment is illustrated in Figure 2.

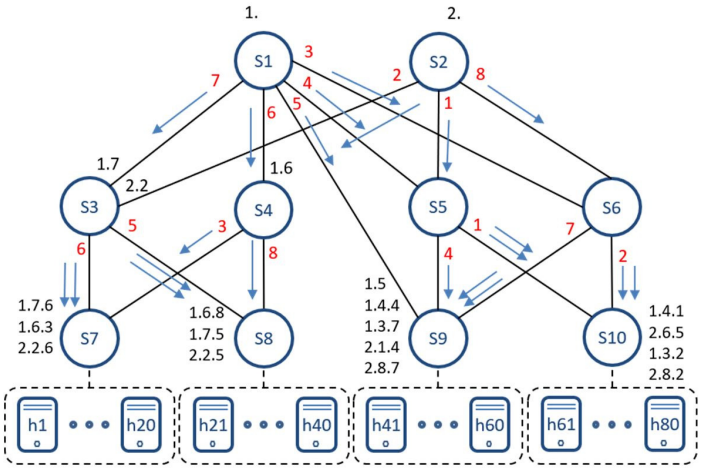


Figure 2. Address assignment example without synchronized suffixes.

The assigned addresses can have an additional property, which is synchronization of suffixes. GA3 achieves the synchronization of suffixes by evaluating the propagated HLMACs; instead of simply discarding longer HLMACs, a more complex priority rule is applied to discard or broadcast the addresses. For example, in Figure 2, node *S*3 obtains suffix 7 from *S*1 and suffix 2 from *S*2, learning HLMACs 1.7 and 2.2, respectively. This is because suffix selection is distributed, decided individually, so *S*1 and *S*2 decisions might (and will usually) be different. In order to match these suffixes, *S*1 and *S*2 could evaluate the broadcasting produced by *S*3. More specifically, *S*3 will propagate 1.7 (received from *S*1) plus a suffix to *S*2, and 2.2 (received from *S*2) plus a suffix to *S*1. When *S*1 and *S*2 receive these HLMACs, they would initially discard them (as they are longer than the ones they already have), but they could also leverage them to learn from the suffixes assigned by their neighbour core nodes and synchronize them. In the example, *S*1 will understand that *S*2 selected suffix 2, and *S*2 that *S*1 selected 7. Following certain priority, they could select a unique suffix and propagate it again, synchronising their values accordingly. For instance, *S*2 could consider that $1.7 < 2.2$ and then suffix 7 has a higher priority, hence propagating again its HLMAC to *S*3, but now as 2.7 (instead of 2.2), which is synchronized with 1.7. As a result, all network nodes obtain HLMACs with synchronized suffixes (for simplicity, the present article omits the full assignment procedure, but it can be checked in GA3 [10]).

Figure 3 shows an example of synchronized assignment. *S*7 obtains 11.7.6, 1.7.6 and 2.7.6 (all suffixes are 7.6), while in Figure 2 (not synchronized) it obtained 1.7.6, 1.6.3 and 2.2.6. Synchronization of suffixes requires additional control messages (as detailed in GA3 [10]), but it grants additional properties for faster recovery after failures, as we will describe in the following sections.

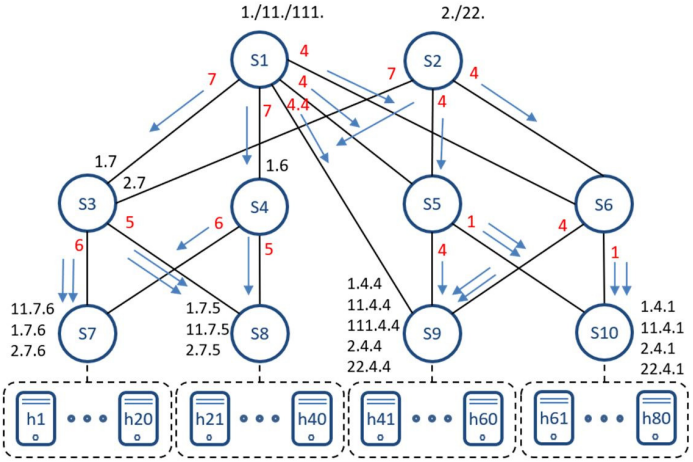


Figure 3. Address assignment example with synchronized suffixes.

Note that label assignment goes from top (core) to bottom (Top-of-Rack (ToR)) switches, while end devices (usually hosts or servers) are left unaltered. When a host requires the creation of a route, it will be the ToR switch the one in charge to translate the physical (real) MAC of the host into an HLMAC. For example, host `h1` could obtain label 11.7.6.1, 1.7.6.1 or 2.7.6.1, by simply adding a unique suffix (.1) to the assigned address of the ToR switch that serves it, which is *S*7.

Finally, it is worth noting that the bigger the network topology, the bigger the redundancy and the bigger the amount of HLMACs to be assigned, because each HLMAC represents a potential route. However, the network manager can limit the assignment of addresses to any value. For example, nodes could just learn a limited set of labels and stop propagating them once reached, which reduces the amount of alternative routes, but accelerates the convergence time of the assignment; hence, it is up to the network manager to analyze the trade-off between those two aspects: resilience and scalability.

3.2. Scalability: Zero-Table Multiple-Path Routing

Routing in eTorii is grounded on the fact that layer-2 addresses are meaningful, instead of randomly defined by the manufacturer. In other words, eTorii leverages the previously assigned HLMACs, conveyed now in the data frames instead of the original MAC addresses, to efficiently forward traffic. This principle of meaningful MAC addresses is supported by a recent amendment to IEEE Std 802, in which IEEE 802 MAC addresses are defined, published in 2017 [24]. Accordingly, eTorii is founded on a similar approach to source routing [14,25], as routing information is transported in the packets, allowing a reduction in the routing table sizes. However, the main advantage of eTorii is that it does not require any overhead as it reuses the already existing MAC fields of the frames. As a consequence, eTorii boosts scalability because it tends towards zero-entry tables.

The routing procedure is defined as follows: When a host, e.g., *h*1, starts a communication with another, such as *h*41, it will first send this traffic to its corresponding ToR, which is *S*7, according to Figure 3. Switch *S*7 will then be in charge of translating the real MAC address from *h*1, defined by the manufacturer, into a corresponding meaningful HLMAC, and it will apply the same action to the MAC address of *h*41, as defined in Torii [9]. For instance, in Figure 3, if *S*7 assigns the HLMACs with values 2.7.6.1 and 2.4.4.1 to *h*1 and to *h*41, respectively, the data packet will follow a path towards the core switch *S*2 in the topology (traversing *S*7 and *S*3), as it is the switch from which HLMACs with prefix 2 were populated, and then it will go down through port with ID 4 to switch *S*5, through port with ID 4 to switch *S*9, and through port with ID 1 until reaching *h*41. These 4 values (2, 4, 4 and 1) are basically the digits contained the HLMAC assigned to *h*41, which is carried out in the standard MAC address field of the packet. The first value (the prefix) is leveraged to send the packet up towards the correlated core switch, and the rest of values (the suffix) identify the list of switch ports traversed to reach the destination. For this reason, there is no need of routing tables, except for some entries to save the labels or IDs associated with each port, locally at each switch. These lists of saved IDs are independent of the number of active hosts communicating and might be configure to any size, up to the network administrator; the only implication is that the more labels a switch has, the bigger the amount of multiple alternative paths.

Each end host has a set of HLMACs, that is, a set of multiple paths to be reached. If, alternatively, *S*7 assigned a different HLMAC to *h*41 from the available set, like 1.4.4.1, the procedure would have been similar except for the traversed path, which would go through the core switch *S*1 first, instead of *S*2. That is, ToR switches are in charge of selecting the appropriate HLMACs (and hence paths) and, for this reason, they are capable of balancing the load in the network based on diverse criteria, which is an orthogonal aspect to the definition of eTorii, but proves that eTorii grants different behaviours based on this assignment of meaningful addresses.

3.3. Resilience: On-the-Fly Path Repair

Two main approaches exist to recover communication paths after network failures, namely: restoration and protection [26]. The former considers that, when a failure occurs, the network will trigger a mechanism to generate a new path, while the latter establishes multiple backup paths a priory, so that they could be directly leveraged when the main one is not available anymore. Although the second approach has a faster recovery time, it also consumes additional resources to keep an updated list of backup paths. Therefore, there should be a trade-off of both approaches depending on the specific network scenario.

In the specific case of eTorii, it benefits from the protection approach just by reusing the already assigned HLMACs. In particular, in eTorii, each HLMAC represents one possible path from a core switch to an end host; thus, multiple HLMACs represent multiple available paths, which means one main path and a set of backup paths, depending on the amount of stored HLMACs. For instance, in Figure 3, switch *S*9 has 5 assigned HLMACs, which indicates 5 different paths to reach the core switches and, more specifically, 3 paths to reach *S*1 and 2 to reach *S*2. Selecting an alternative path is as easy as modifying the destination HLMAC and this action could be performed at any point in the network. By limiting the amount of HLMACs assigned to each switch (as previously mentioned), we are just limiting the amount of alternative paths assigned. For example, limiting learning to 10 HLMACs will imply saving a list of only 10 labels and provides 1 main path to any node in the network plus 9 alternative routes, which seems to be a fair amount [27,28]. Furthermore, this amount of HLMACs is independent of network size, which grants the scalability of eTorii.

Additionally, as mentioned in Section 3.1, eTorii might assign HLMACs either with synchronized or without synchronized suffixes. Although synchronising suffixes implied additional control messages, one of the main advantages obtained is that path repair can be applied on-the-fly, when the failure occurs, as assigned HLMACs only differ in their prefix. Thus, when a route is not valid anymore, simply exchanging the prefix with another of the possible ones will instantaneously generate an alternative path, even if the message is in the middle of the network, which represents a minimum cost for rerouting.

Rerouting is performed in a hop-by-hop basis, hence the resulting alternative route does not necessarily have to be the shortest available one. More specifically the search for a new route is started in the very first node aware of the network failure. In the case that this node does not have a direct alternative route, the packet will be sent back (through the incoming port) to the previous hop, which will look again for an alternative route. This "hop-back" action is repeated until an alternative path is found.

Figure 4 illustrates how eTorii reroutes traffic in an example where two link failures occurs at the same time. In this example, host *h*1 is sending traffic to *h*60 and the selected HLMAC address for this destination is 11.4.4.20, which implies following the route traversing S7-S3-S1-S5-S9. When the traffic arrives at switch *S*3, the next link in the path (S3-S1) is unavailable, so switch *S*3 can directly select a new prefix for the destination HLMAC and that will automatically indicate the new alternative path. As only core switch *S*2 can be reached, the new prefix should be either 2 or 22. In the example from Figure 4, *S*3 chooses 22, i.e.,the new destination HLMAC is now 22.4.4.20 (instead of the original 11.4.4.20), which implies the new route is now S7-S3-S2-S6-S9. Accordingly, *S*3 would then reroute the traffic directly sending it to *S*2 with this new destination HLMAC, which would be a lossless on-the-fly path repair, hence minimizing the associated repair latency.

Following the example illustrated in Figure 4, once in *S*2, this switch will forward the traffic to *S*6, which finds a second down link towards the next hop, *S*9. To repair the path again, in this case, eTorii is going to select the new prefix 2, associated with route S7-S3-S2-S5-S9. As *S*6 does not belong to this new generated route, it sends the received traffic back through the incoming port until reaching *S*2, which in this case does belong to the new route. Consequently, *S*2 sends the traffic to *S*5, which will finally forward it to *S*9 and eventually to the destination host, *h*60.

As a conclusion, eTorii is capable to reroute traffic on the fly just via small local forwarding, even when multiple link failures occur at once. Furthermore, this rerouting is lossless and the processing cost is low as it only implies modifying one prefix in the MAC address. In particular, the advantage of this mechanism is that it can be immediately applied without requiring to wait for a new address assignment to overcome the failure.

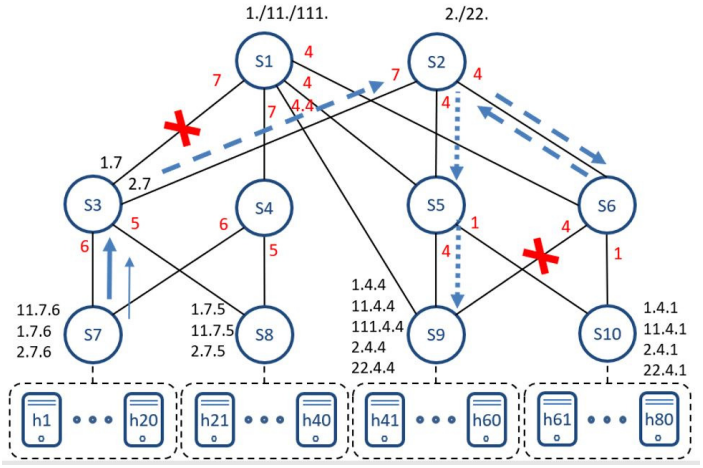


Figure 4. Example of on-the-fly path repair after two link failures.

3.4. Flexibility: Generalization to Any Hierarchical DCN

DCNs are usually deployed based on some kind of hierarchy (well-known examples include VL2 [3] or BCube [6]), and routing solutions are many times designed based on these specific topologies. For example, Torii was initially developed to be applied in the fat-tree defined by PortLand [2]. For that reason, one of the main objectives of eTorii was to expand the features of Torii to apply them in any hierarchical topology. GA3 is the cornerstone to accomplish this goal, as described in Section 3.1 and, accordingly, unicast forwarding and path repair in eTorii can be flexibly applied in any type of hierarchical topology, even non-symmetrical, as illustrated in Sections 3.2 and 3.3.

This approach is very flexible to guarantee, for instance, that routing still works when the DCN is being updated or some part of it fails, as a non-symmetrical topology could also represent a transient state in which the data center is deploying new hardware, progressively. In fact, thanks to GA3, eTorii is even capable of recognising some types of connections, which are usually considered very irregular or even faulty in DCNs, namely: peer-links (links that connect network devices in the same hierarchy level) and cross-links (links between non-adjacent hierarchy levels). This links could be ignored or could directly raise a notification if found and non desired. For example, the topology previously used to illustrate routing and path repair in Figures 2–4 contains a cross-link, as *S*1 and *S*9 are directly connected, skipping the intermediate level (comprised of *S*3, *S*4, *S*5 and *S*6). In this example, the cross-link is used in the routing scheme.

4. Implementation

In order to validate eTorii, we designed and developed an SDN application using the Ryu controller [29], which uses the OpenFlow protocol [30] to control the network devices. Although eTorii is a distributed protocol by definition, as explained in Section 3, the reason to implement it by using SDN (which is logically centralized) is that OpenFlow serves also to easily develop and test protocol proof-of-concepts in environments close to reality, as it can be deployed in hardware using real network traffic traces [30]. The implementation of eTorii has validated the four features described in previous sections, namely: auto-configuration, scalability (no routing table entries are needed apart from the node labelling), resilience (fast recovery after link failure) and flexibility (the protocol works with diverse hierarchical topologies).

As it can be seen in Figure 5, the developed application is composed of two main

logical parts. The first one, Figure 5a, on the left, is executed just right after starting the network, when the system has to discover and map the network topology that will be handled afterwards. In this initial step, the network controller will generate the HLMAC addresses, in a centralized way, to be assigned to each of the network switches. Additionally, the application also generates a traffic distribution tree for broadcast traffic. Finally, it enters an state in which it will be paying attention to additional events to be handled.

The second part, Figure 5b, on the right, is basically in charge of handling two types of events and its execution starts right after finalising Figure 5a. The events to be managed are basically of two types: (1) topological, or (2) PACKET_IN events. A topological events occurs as a consequence of a change in the network topology (for example, a new node is added or some link fails), while the PACKET_IN is an OpenFlow message triggered by a switch when no action is defined for a new packet arriving at it. In this latter case, the switch sends this message to the controller and awaits for the routing decision to be made by it. This event is usually generated with the first packets in any communication, which are usually ARP packets in IPv4-based networks. The actions to be applied for each event are as follows: in the case of the topological event, the controller checks whether the network topology has actually been modified and, if so, deletes obsolete forwarding rules and updates the traffic distribution tree; while if an PACKET_IN event, the controller proceeds to calculate and install the required rules in the switches to forward the traffic associated to that PACKET_IN message.

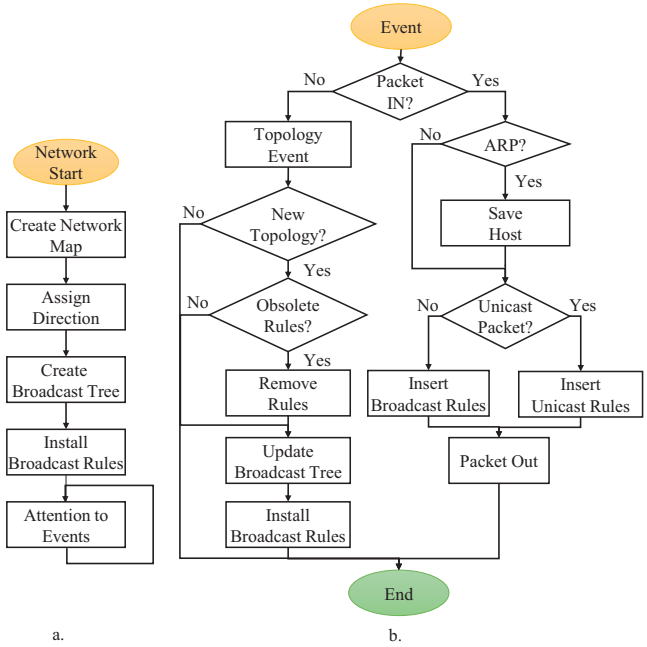


Figure 5. Logical implementation of eTorii.

In the following sections, we describe the implementation in detail.

4.1. Topology Discovery and HLMAC Address Assignment

Before routing any traffic, the network topology should be mapped. The controller discovers the network topology by using the Link Layer Discovery Protocol (LLDP) [31]. One the network mapping has finalized, the controller can proceed to assign the corresponding HLMAC to each network node, according to the procedure defined by GA3. This association of HLMAC addresses is only performed internally, in a logical way inside the controller, from which the actual OpenFlow forwarding rules are later on derived. The HLMAC association can be updated if topology changes are detected.

It is important to highlight that, in this implementation, GA3 has been developed as a centralized function for the sake of simplicity. However, in real networks, GA3 can

perfectly work in a distributed manner and, in fact, networks will benefit from it as the distributed behaviour is potentially much more scalable.

4.2. Host Registration

In eTorii, HLMAC addresses are assigned to network switches, but not to end hosts. Edge switches are usually responsible for registering their connected hosts and assigning them an HLMAC address, which is directly deduced from its own ones. In this regard, the developed application needs to register these hosts, associating an HLMAC to each physical MAC of the host, and saving its location together with the topological map. This action cannot be performed by LLDP, as this protocol is only capable of discovering network nodes directly connected to the controller, which is not the case of hosts. For this reason, the application registers the hosts indirectly via traffic snooping. In particular, our implementation considers any host will send an ARP message prior to any communication, hence the controller will monitor these events (as they will generate a PACKET_IN) and register the hosts accordingly.

4.3. Address Translation and Routing

By definition, edge switches in eTorii are responsible of translating the MAC addresses into HLMAC, and vice versa. To this purpose, the controller installs a set of rules at edge switches to modify the addresses in the Ethernet frames arriving at it.

Unicast routing in eTorii has been implemented in a reactive fashion, calculating the path switch by switch based on the destination HLMAC address contained in the frame and according to the eTorii logic. Broadcast routing is performed using the traffic distribution tree previously mentioned.

4.4. Recovery Mechanisms

When the controller detects a link from a path is not available, it examines the rest of links seeking for alternative routes, according to the definition of eTorii, as described in Figure 6. If an alternative route exists, the new link to be used should be associated with a specific prefix and, thus, it implies the modification of the HLMAC prefix in the frame, accordingly. Afterwards, the frame in sent through the associated port. Additionally, the source edge switch will remove the rule associated to the old prefix and will include the new one, so that future packets from the same flow are directly forwarded through the newer alternative path. Finally, the old prefix is excluded from the set of available prefixes until the link is back to normal again. In parallel, the HLMAC association may be updated triggered by a topology change detected by LLDP.

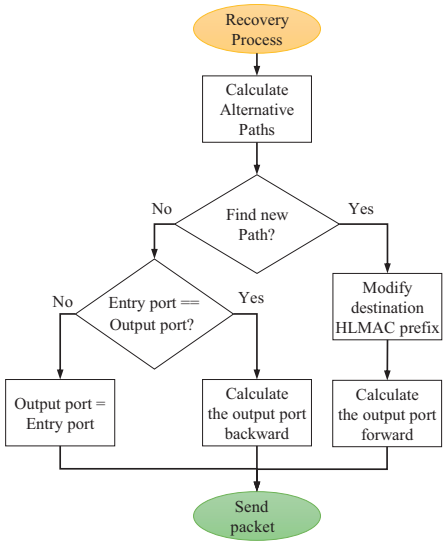


Figure 6. Recovery logic in the eTorii implementation.

The previous procedure is applied in the case of unicast routing. In the case of broadcast forwarding, the tree should be recalculated every time one of its links is affected, as previously mentioned.

5. Evaluation

In this section, we evaluate the implementation described before, based on a set of experiments. The testbed was based on Open vSwitch (OVS) switches [32], which were deployed with Mininet [33], as a network emulator, close to practical environments, in which we executed the eTorii application based on the Ryu controller. These environments were installed in two hosts Intel® Core™ i7 with 24 GB of RAM.

First of all, during the implementation phase, we checked the accomplishment of the four features of eTorii, described in Section 3, viz. auto-configuration, scalability, resilience and flexibility. To prove this, we built the scenario shown in Figure 4, among some others, and assessed the application of eTorii in all of them. First of all, we confirmed that the configuration was properly performed automatically thanks to GA3; while flexibility was acknowledged as the scenario in Figure 4 represents a non-symmetrical network. Resilience was proven via the simulation of multiple link failures (including the ones illustrated in Figure 4), which were correctly re-routed. Finally, scalability was mainly assessed on a qualitatively basis as, by definition, every HLMAC defines one possible route through the network and eTorii switches only need to save a small amount of them, independently of the network size. In particular, in all scenarios tested, no switch had to save more than 10 HLMACs, which is a very small amount of memory and assures scalability. Moreover, this amount of HLMACs was enough for most of the multiple-link failure scenearios. However, we only focused on that side of scalability and other parameters (such as convergence time or control message load) should be checked for a holistic view of it.

Once the four features of eTorii were assessed, the main objective of the evaluation was to prove that the performance of eTorii was still good even accomplishing these features. In particular, this performance evaluation was divided into two types of tests: the first one was designed to analyze flow routing, while the second was tailored to acknowledge the effective throughput achieved by the implemented application. Both of these tests measured the ability of eTorii to efficiently route traffic once HLMAC addresses are already distributed. In this way, we avoided any type of bias caused by the centralized implementation, as the evaluation focused strictly on the data plane, which works in a distributed manner.

The selected network topologies were two classical data center topologies, namely VL2 [3] and Spine-Leaf [4]. They are illustrated in Figures 7 and 8, respectively, in which switches (in this case OVS) are represented as circles (C_i for core, A_i for aggregate, T_i for ToR switches in VL2; and S_i for spine and L_i for leaf switches in Spine-Leaf), and hosts or servers (s_i) are located as groups in the inferior part of the topology, surrounded by dotted lines. All hosts belonging to the same ToR or edge switch are grouped accordingly. The importance of these topologies is well justified, as they have been leveraged by popular cloud providers like Google [34] or Facebook [5]. Additionally, using two types of networks let us prove the flexibility of eTorii, which is suitable for any hierarchical DCN.

To obtain representative results, we performed 10 executions per each average value calculated and, moreover, the confidence interval was also calculated to provide a delimitation of real values based on the experimental tests.

Finally, although not directly illustrated in any test or figure, we checked that the configuration was performed successfully in an automatic way, and that the amount of table entries for routing was smaller than 10 in all cases (independently of the network size or number of hosts), hence proving the scalability of eTorii.

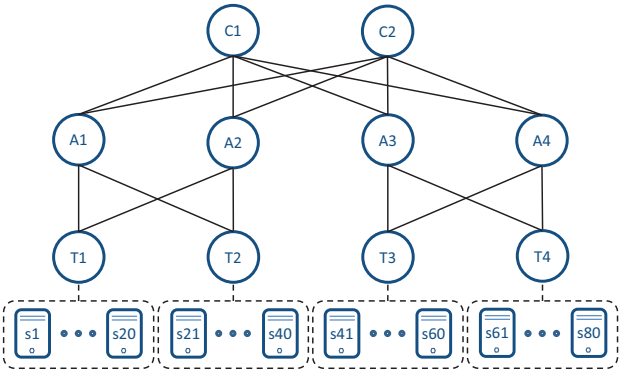


Figure 7. VL2 topology (parameters 2 × 4 × 4).

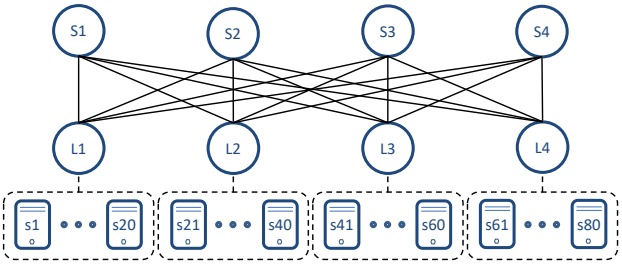


Figure 8. Spine-Leaf topology (parameters 4 × 4).

5.1. Test 1: Flow Routing Analysis

In this first test, we used the VL2 topology (Figure 7) to study the routing results obtained by the implemented eTori application. In particular, we generated traffic according to the following traffic matrix: h1→h16, h2→h15, h3→h14, ..., h14→h3, h15→h2, h16→h1, with random interarrival times. This matrix was designed to have flows that equally traverse the whole topology, i.e., all switches in the network.

The results obtained from executing this scenario are shown in Figure 9, which shows the percentage of paths grouped by common source ToR for certain destination ToR switches. Ideally, for a balanced distribution of traffic, this percentage should be uniformly distributed for each existing path.

For example, the green grouping in Figure 9 depicts the paths from ToR 1 to ToR 4. As four available paths exist between these two ToR switches, the traffic distribution should have a value around 25% per each of the possible routes, and we can see their values are indeed around 15% and 35% (with a deviation of 10%), which is quite close to the ideal scenario, particularly considering this values are measured only with 10 executions (a value that is not divisible by 4). A similar behaviour can be observed for the paths originating at ToR 2 and ToR 4, and only the ones with source in ToR 3 have a bigger deviation, but the actual percentage never surpasses the 50% of use in any case. This observation is because the different available paths (HLMACs) are randomly selected and the results depend on the quality of the random number generator.

Therefore, the results illustrate how eTorii balances the load among the diverse available paths between any source and any destination, which will improve the overall performance of the DCN.

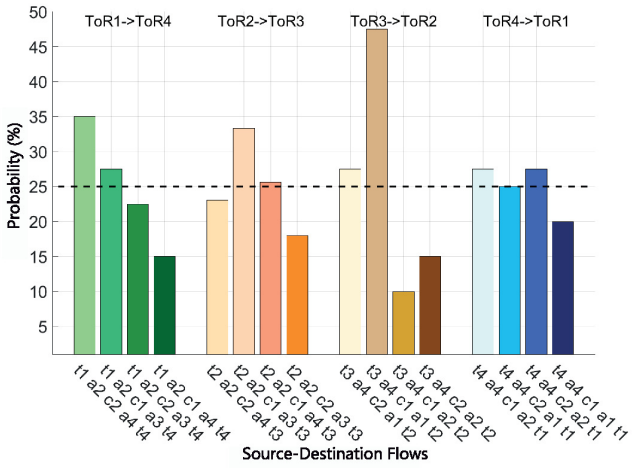


Figure 9. Percentage of selected routes, grouped by source ToR.

5.2. Test 2: Effective Throughput

To evaluate the performance of eTorii in terms of throughput, we compared it against Equal-Cost Multi-Path (ECMP) [35], a standardized and well-known protocol in the field of DCNs.

More specifically, we evaluated both protocols in a Spine-Leaf 4 $\times$ 4 topology (that is, with 4 Spine nodes *S* and 4 Leaf nodes *L*), with 20 hosts per Leaf node (as depicted in Figure 8). The traffic load is based in the flow patterns illustrated in Figure 10 with random inter-arrival times, and source-destination flows randomly generated in which the source and destination differ and belong to separate ToR switches, i.e., different Leaf nodes. Their flow size distributions are Web Search [2] and Data Mining [36] (derived from flow size measurements in previous works related to data centers). Furthermore, we defined four scenarios with increasing average traffic load in links (10%, 20%, 40% and 60% with respect to the their nominal capacity). This setup aims to prove that the relative performance is not worsen by the traffic increase in the network. Finally, all network links are set to 100 Mbps and the total execution time of the tests is 1800 s, considering a transitory time of 800 s, that is, we only evaluate the 1000 s afterwards, when the environment is stable. Each scenario is executed 10 times. Table 1 summarizes the setup.

The results are illustrated in Figure 11, which show that eTorii obtains similar throughput values to ECMP. In particular, the two columns depict the two types of traffic, divided into the three types of flows ("elephant", "rabbit" and "mouse"), and eTorii has a better throughput with the "elephant" (around 3% and 13% higher) and "mouse" flows (around 8% and 12% higher), while it remains similar or even lower for the "rabbit" flows. The advantage of eTorii is that DCNs are usually characterized by elephant and mouse flows, as they represent the majority of traffic, while rabbit flows remain less frequent. Therefore, the overall performance of eTorii would be better in DCNs if compared to ECMP.

The provided results are a lower bound since a distributed implementation of eTorii will obtain better results since it will not suffer from the delays derived from the SDN in regards to communication and processing time, which are at least equal to the Round Trip Time (RTT) between each device and SDN control plane, through the established control channel, to install the SDN rules derived from the HLMAC assignment.

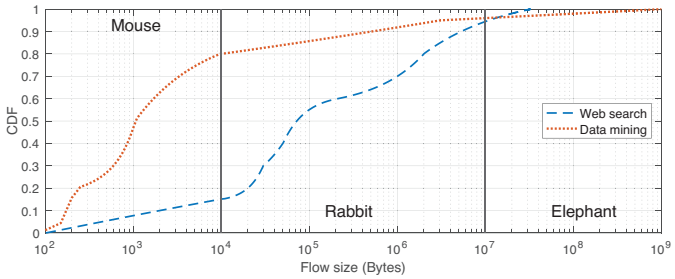


Figure 10. Data center flow size CDFs (Reprinted with permission from ref. [37]. Copyright 2017 IEEE).

Table 1. Summary of Test 2 testbed parameters.

Parameter	Value
Network topology	Spine-Leaf (4 × 4)
Hosts per Leaf	20
Flow distribution	Random Extra-Leaf
Flow size distribution	Web Search [2] & Data Mining [36]
Network traffic load (%)	10, 20, 40 y 60%
Link speed	100 Mbps
Execution time	1800 s
Transitory time	800 s
Number of executions	10

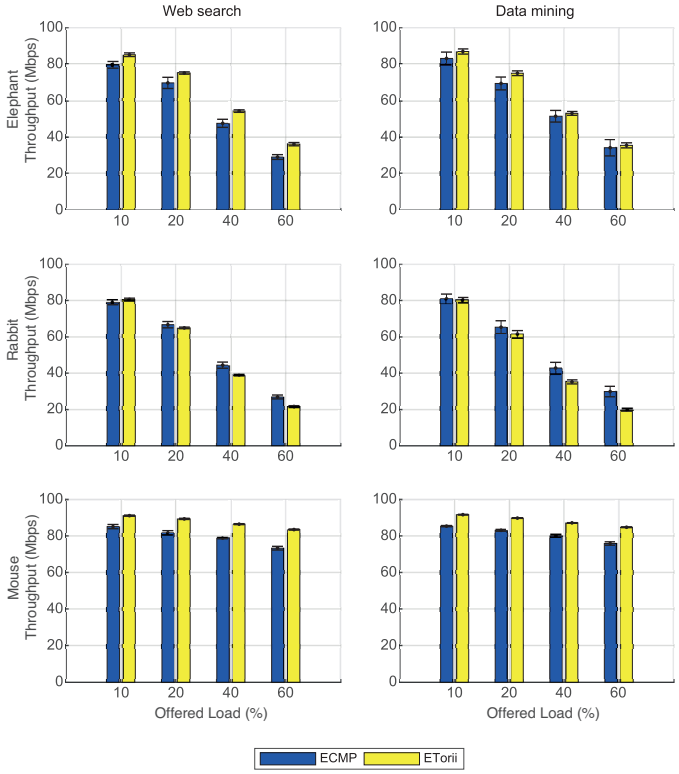


Figure 11. Average throughput classified by traffic type.

6. Conclusions

Along this article, we have defined eTorii, a protocol that allows routing traffic in DCNs in an efficient, scalable and flexible way. Despite being a distributed protocol, its configuration is automatic and it reacts fast to network failures and changes, as many alternative routes are usually deployed when the HLMAC address assignment is realized.

We have implemented and evaluated a prototype of eTorii in the Ryu SDN controller and with the Mininet emulator. The evaluation shows routing is correctly balanced and the obtained throughput directly competes with the standardized protocol ECMP, being even better for elephant and mouse flows, which are the most common in DCNs.

As future work, we envision an in-depth analysis of eTorii in diverse DCN topologies to measure the average amount of table entries required for routing to evaluate its scalability in detail, as well as the convergence time to prove its resilience and flexibility. In particular, we would like to focus on the effect of topological changes. Although eTorii is able to re-route at zero cost most of times, we should analyze in detail how the proposed recovery mechanism affects the overall throughput performance, packet loss and scalability. Additionally, a high-performance implementation, for example using P4 hardware targets or similar, could be advisable to show the application of eTorii to cloud computing networks.

Author Contributions: Conceptualization, E.R.; methodology, I.M.-Y.; software, M.B.-M.; validation, I.M.-Y. and J.A.-H.; formal analysis, E.R.; investigation, J.M.A.; resources, J.M.A.; data curation, J.A.-H.; writing—original draft preparation, E.R.; writing—review and editing, I.M.-Y.; visualization, J.A.-H. and J.M.A.; supervision, E.R. and I.M; project administration, E.R. and I.M.-Y.; funding acquisition, E.R. and I.M.-Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by grants from Comunidad de Madrid through Project TAPIR-CM (S2018/TCS-4496) and Project IRIS-CM (CM/JIN/2019-039), and from Junta de Comunidades de Castilla la Mancha through Project IRIS-JCCM (SBPLY/19/180501/000324).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

CDF	Cumulative Distribution Function
ECMP	Equal-Cost Multi-Path
HLMAC	Hierarchical Local MAC
LLDP	Link Layer Discovery Protocol
OVS	Open vSwitch
RSTP	Rapid Spanning Tree Protocol
SDN	Software-Defined Networking
ToR	Top-of-Rack

References

1. Milian, E.Z.; Spinola, M.M.; Carvalho, M.M. Risks and Uncertainties in Cloud Computing: Literature Review, Trends and Gaps. *IEEE Lat. Am. Trans.* **2017**, *15*, 349–357. [CrossRef]
2. Mysore, R.N.; Pamboris, A.; Farrington, N.; Huang, N.; Miri, P.; Radhakrishnan, S.; Subramanya, V.; Vahdat, A. PortLand: A scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.* **2009**, *39*, 39–50. [CrossRef]
3. Greenberg, A.; Hamilton, J.R.; Jain, N.; Kandula, S.; Kim, C.; Lahiri, P.; Maltz, D.A.; Patel, P.; Sengupta, S. VL2: A scalable and flexible data center network. *SIGCOMM Comput. Commun. Rev.* **2009**, *39*, 51–62. [CrossRef]
4. Al-Fares, M.; Loukissas, A.; Vahdat, A. A scalable, commodity data center network architecture. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 63–74. [CrossRef]
5. Alexey Andreyev. Introducing Data Center Fabric, the Next-generation Facebook Data Center Network. 2014. Available online: https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/ (accessed on 15 May 2021).
6. Guo, C.; Lu, G.; Li, D.; Wu, H.; Zhang, X.; Shi, Y.; Tian, C.; Zhang, Y.; Lu, S. BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers. *SIGCOMM Comput. Commun. Rev.* **2009**, *39*, 63–74. [CrossRef]
7. Singla, A.; Hong, C.Y.; Popa, L.; Godfrey, P.B. Jellyfish: Networking data centers randomly. Presented at the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12), San Jose, CA, USA, 25–27 April 2013; pp. 225–238.
8. Bhandarkar, S.; Behera, G.; Khan, K.A. Scalability Issues in Software Defined Network (SDN): A Survey. *Adv. Comput. Sci. Inf. Technol. ACSIT* **2015**, *2*, 81–85.

9. Rojas, E.; Ibanez, G.; Gimenez-Guzman, J.M.; Rivera, D.; Azcorra, A. Torii: Multipath distributed Ethernet fabric protocol for data centres with zero-loss path repair. *Trans. Emerg. Telecommun. Technol.* **2015**, *26*, 179–194. [CrossRef]
10. Rojas, E.; Alvarez-Horcajo, J.; Martinez-Yelmo, I.; Arco, J.M.; Carral, J.A. GA3: Scalable, distributed address assignment for dynamic data center networks. *Ann. Telecommun.* **2017**, 72, 693–702. [CrossRef]
11. Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. Available online: https://opennetworking.org/sdn-resources/whitepapers/software-defined-networking-the-new-norm-for-networks/ (accessed on 15 May 2021).
12. Kreutz, D.; Ramos, F.; Esteves Verissimo, P.; Esteve Rothenberg, C.; Azodolmolky, S.; Uhlig, S. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* **2015**, *103*, 14–76. [CrossRef]
13. Kempf, J.; Bellagamba, E.; Kern, A.; Jocha, D.; Takács, A.; Sköldström, P. Scalable fault management for OpenFlow. In Proceedings of the 2012 IEEE International Conference on Communications (ICC), Ottawa, ON, Canada, 10–15 June 2012; pp. 6606–6610.
14. Wang, F.; Gao, L.; Shao, X.; Harai, H.; Fujikawa, K. Towards reliable and lightweight source switching for datacenter networks. In Proceedings of the 2017 IEEE Conference on Computer Communications, INFOCOM 2017, Atlanta, GA, USA, 1–4 May 2017; pp. 1–9. [CrossRef]
15. Gonzalez-Diaz, S.; Marks, R.; Rojas, E.; de la Oliva, A.; Gazda, R. Stateless Flow-Zone Switching Using Software-Defined Addressing. *IEEE Access* **2021**, *9*, 68343–68365. [CrossRef]
16. Sunshine, C.A. Source Routing in Computer Networks. *SIGCOMM Comput. Commun. Rev.* **1977**, *7*, 29–33. [CrossRef]
17. Vanini, E.; Pan, R.; Alizadeh, M.; Taheri, P.; Edsall, T. Let It Flow: Resilient Asymmetric Load Balancing with Flowlet Switching. In Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), Boston, MA, USA, 27–29 March 2017; USENIX Association: Boston, MA, USA, 2017; pp. 407–420.
18. Liu, J.; Huang, J.; Lv, W.; Wang, J. APS: Adaptive Packet Spraying to Isolate Mix-flows in Data Center Network. *IEEE Trans. Cloud Comput.* **2020**, *2020*, 2985037. [CrossRef]
19. Hu, J.; Huang, J.; Lv, W.; Zhou, Y.; Wang, J.; He, T. CAPS: Coding-Based Adaptive Packet Spraying to Reduce Flow Completion Time in Data Center. *IEEE ACM Trans. Netw.* **2019**, *27*, 2338–2353. [CrossRef]
20. Huang, J.; Lyu, W.; Li, W.; Wang, J.; He, T. Mitigating Packet Reordering for Random Packet Spraying in Data Center Networks. *IEEE ACM Trans. Netw.* **2021**, *29*, 1183–1196. [CrossRef]
21. Zhang, H.; Zhang, J.; Bai, W.; Chen, K.; Chowdhury, M. Resilient Datacenter Load Balancing in the Wild. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM'17, Los Angeles, CA, USA, 21–25 August 2017; Association for Computing Machinery: New York, NY, USA, 2017; pp. 253–266. [CrossRef]
22. Lopez-Pajares, D.; Alvarez-Horcajo, J.; Rojas, E.; Asadujjaman, A.S.M.; Martinez-Yelmo, I. Amaru: Plug Play Resilient In-Band Control for SDN. *IEEE Access* **2019**, *7*, 123202–123218. [CrossRef]
23. Acharya, H.B.; Hamilton, J.; Shenoy, N. From Spanning Trees to Meshed Trees. In Proceedings of the 2020 International Conference on COMmunication Systems NETworkS (COMSNETS), Bangalore, India, 7–11 January 2020; pp. 391–395.
24. *IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture–Amendment 2: Local Medium Access Control (MAC) Address Usage;* IEEE, Posted 25 Augest, 2017. Available online: https://standards.ieee.org/standard/802c-2017.html (accessed on 15 May 2021).
25. Jin, X.; Farrington, N.; Rexford, J. Your Data Center Switch is Trying Too Hard. In Proceedings of the Symposium on SDN Research, SOSR'16, Santa Clara, CA, USA, 14 March 2016; ACM: New York, NY, USA, 2016; pp. 12:1–12:6. [CrossRef]
26. Cholda, P. Network recovery, protection and restoration of optical, SONET-SDH, IP, and MPLS [book review]. *IEEE Commun. Mag.* **2005**, *43*, 12. [CrossRef]
27. Nelakuditi, S.; Zhang, Z.L. On Selection of Paths for Multipath Routing. In Proceedings of the 9th International Workshop on Quality of Service, Karlsruhe, Germany, 6–8 June 2001; Springer: Berlin/Heidelberg, Germany, 2001; pp. 170–186.
28. Raiciu, C.; Barre, S.; Pluntke, C.; Greenhalgh, A.; Wischik, D.; Handley, M. Improving Datacenter Performance and Robustness with Multipath TCP. *SIGCOMM Comput. Commun. Rev.* **2011**, *41*, 266–277. [CrossRef]
29. Ryu SDN Framework. Available online: https://ryu-sdn.org/ (accessed on 15 May 2021).
30. McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S.; Turner, J. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 69–74. [CrossRef]
31. Link-Layer Discovery Protocol IEEE 802.1 AB: Link Layer Discovery Protocol (LLDP). Available online: https://www.ieee802.org/1/files/public/docs2002/lldp-protocol-00.pdf (accessed on 15 May 2021).
32. Pfaff, B.; Pettit, J.; Koponen, T.; Jackson, E.; Zhou, A.; Rajahalme, J.; Gross, J.; Wang, A.; Stringer, J.; Shelar, P.; et al. The Design and Implementation of Open vSwitch. In Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), Oakland, CA, USA, 4–6 May 2015; USENIX Association: Oakland, CA, USA, 2015; pp. 117–130.
33. Mininet. Available online: http://mininet.org/ (accessed on 15 May 2021).
34. Singh, A.; Ong, J.; Agarwal, A.; Anderson, G.; Armistead, A.; Bannon, R.; Boving, S.; Desai, G.; Felderman, B.; Germano, P.; et al. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. *ACM SIGCOMM Comput. Commun. Rev.* **2015**, *45*, 183–197. [CrossRef]
35. 802.1Qbp—Equal Cost Multiple Paths. Available online: https://www.ieee802.org/1/pages/802.1bp.html (accessed on 15 May 2021).

36. Alizadeh, M.; Greenberg, A.; Maltz, D.A.; Padhye, J.; Patel, P.; Prabhakar, B.; Sengupta, S.; Sridharan, M. Data Center TCP (DCTCP). In Proceedings of the ACM SIGCOMM 2010 Conference, SIGCOMM'10, Melbourne, Australia, 1–30 November 2010; ACM: New York, NY, USA, 2010; pp. 63–74. [CrossRef]
37. Alvarez-Horcajo, J.; Lopez-Pajares, D.; Arco, J.M.; Carral, J.A.; Martinez-Yelmo, I. TCP-path: Improving load balance by network exploration. In Proceedings of the 2017 IEEE 6th International Conference on Cloud Networking (CloudNet), Prague, Czech Republic, 25–27 September 2017; pp. 1–6. [CrossRef]

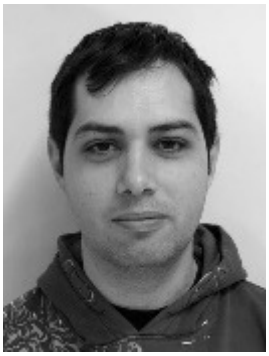
Short Biography of Authors

Elisa Rojas received her PhD in Information and Communication Technologies engineering from the University of Alcala, Spain, in 2013. As a postdoc, she worked in IMDEA Networks and, later on, as CTO of Telcaria Ideas S.L. She has participated in diverse projects funded by the EC, such as FP7-NetIDE or H2020-SUPERFLUIDITY. She currently works as an Assistant Professor in the University of Alcala where her current research interests encompass SDN, NFV, IoT routing, and high-performance Ethernet and data center networks.

Joaquín Álvarez-Horcajo obtained their PhD in Information and Communication Technologies engineering from the University of Alcala in 2020. After having worked at Telefonica as a test engineer for COFRE and RIMA networks, he was awarded a grant for university professor training (FPU) at the University of Alcala. The areas of research where he has worked include Software Defined Networks (SDN), Internet protocols, and new generation protocols. At present, he is especially interested in topics related to advanced switches and SDN networks. He has participated in various competitive projects funded through the Community of Madrid plan, such as TIGRE5-CM and TAPIR-CM.

Isaías Martínez-Yelmo received the Ph.D. degree in telematics from the Carlos III University of Madrid, Spain, in 2010. After working as a Postdoctoral Assistant with the Carlos III University of Madrid, he became and remains as a Teaching Assistant with the Automatics Department, Alcalá University, Spain. His research interests include peer-to-peer networks, content distribution networks, vehicular networks, NGN, and the Internet protocols. Nowadays, he is especially interested in advanced switching architectures, software-defined networks and P4. He has participated in various competitive research projects funded by the Madrid Regional Government (Medianet, Tigre5, TAPIR, IRIS), National projects (CIVTRAFF), and European projects (CONTENT, CARMEN, and so on). His research papers have been published in highimpact JCR indexed research journals, such as the IEEE Communications Magazine, Computer Communications, and Computer Networks, among others. He is currently Associate Editor in the Journal of Telecommunication Systems from Springer. In addition, he has been a Reviewer for high quality conferences (i.e., IEEE INFOCOM) and scientific journals (the IEEE Transactions on Vehicular Technology, Computer Communications, Computer Networks, the ACM Transactions on Multimedia Computing, and so on). He was a Technical Program Committee Member of IEEE ICON, from 2011 to 2013.

José Manuel Arco received their Ph.D. in Telecommunications engineering from the University of Alcala, Spain, in 2002. He is Associate professor in the Automatica Department (Telematic Area) at the University of Alcalá in Spain since 2002. His current research interests encompass SDN/NFV, high performance and scalable Ethernet, Traffic Engineering and data center networks. He also teaches SDN/NFV and data center networks. He has participated on different competitive research projects from Madrid regional government, National and European. His research activity has been published in high impact JCR indexed research magazines, conferences and workshops on networking technologies.



Miguel Briso-Montiano received their MSc in Electrical Engineering from the University of Alcala, Spain, in 2018. He is currently working at GMV as embedded software engineer for avionics and defence projects. His current research interests include data plane programmability with P4, SDN applications and IoT.