network

MDPI

Article

ARP Optimization in SDN Using Controller-Independent Strategies for Data Center Networks

Jose Neftali Limon-Ortiz[1,2], Pedro David Arjona-Villicaña [1,*], Alejandra Guadalupe Silva-Trujillo [1], Francisco Javier Torres-Reyes [1] and Francisco Javier Ramirez-Aguilera [1]

[1] Facultad de Ingeniería, Universidad Autónoma de San Luis Potosí, San Luis Potosí 78290, Mexico; jlortiz@univ-pau.fr (J.N.L.-O.); asilva@uaslp.mx (A.G.S.-T.); francisco.torres@uaslp.mx (F.J.T.-R.); javier@uaslp.mx (F.J.R.-A.)
[2] Collège STEE, Universite de Pau et des Pays de l'Adour, Anglet 64600, France
* Correspondence: david.arjona@uaslp.mx

Abstract

Data Centers that implement Software-Defined Networks (SDN) are not required to employ the Address Resolution Protocol (ARP), but network hosts do. Therefore, there is a need to support this protocol without modifying the intrinsic functionality of the SDN controller. In this work, four strategies for handling ARP are evaluated using an SDN and OpenFlow rules. The strategies include disabling ARP at the host level, using static MAC addresses, introducing a fake gateway, and generating ARP replies using OpenFlow flows. To our knowledge, nobody has tested and compared the main characteristics and advantages offered by these four strategies. Experimental evaluation was conducted on a real SDN network and complemented with similar experiments using Mininet. Performance was assessed using metrics such as ping response time, address resolution response time, jitter, and packet loss ratio. The results show that OpenFlow-based ARP replies provide a good balance in terms of scalability, performance, and configuration effort. This strategy achieved the lowest average ping response time (0.641 ms) and ARP response time (0.6188 ms), while avoiding the manual configuration requirements of static approaches.

Keywords: Address Resolution Protocol; Software Defined Networks; Data Center networks; OpenFlow

1. Introduction

Designing and implementing a Data Center (DC) is a complex task that requires balancing the needs of the final users, the decisions taken by administrators, and the limitations imposed by current technologies. One of the most important factors that need to be considered is the allocation of IP addresses in the DC. By keeping all addresses in the same domain, a more flexible address distribution is possible, which in turn allows to easily move servers and virtual machines (VMs) around the DC. However, this approach also increases overhead traffic due to the use of broadcast packets in large networks [1]. Moreover, it has been reported that more than 88% of broadcast traffic in a DC is generated by the Address Resolution Protocol (ARP) [2]. Therefore, there is an incentive to limit the size of the address domain and generate subnets [3] that allow the implementation of an organized framework that improves traffic distribution and, at the same time, provides network isolation. Thus, creating a dilemma for network administrators between increasing flexibility and reducing network footprint.

Academic Editor: Firstname Lastname

Received:
Revised:
Accepted:
Published:

Another technology that has been increasingly considered for DCs is Software Defined Networks (SDN) [4], which defines a centralized network configuration and control entity, the *control plane*, but requires employing SDN-capable switches that forward data packets, and thus implement the *data plane*. Although SDN has become an attractive solution for DC networks because it allows centralized and streamlined network management [5], there are many other factors that need to be considered before its implementation.

One such factor is the role that MAC addresses play when forwarding packets in the DC network. Traditionally, hosts need to know the correspondence between the MAC and IP addresses of the destination hosts to which they send information; this is referred to as the *IP-MAC mapping* in this paper. This mapping is required before the host starts sending data packets and is usually stored in its ARP table. ARP [6] is the *de facto* protocol used to establish this mapping in IPv4, while Neighbor Discovery (ND) is the equivalent for IPv6 [7]. However, SDN networks do not require an IP-MAC mapping to send packets and may perform this task using only the IP address. Therefore, when using SDN, some implementations may decide to keep the mapping in order to support a single or very large subnet [8,9], while others may decide to use only the IP address for forwarding packets [10]. This latter approach requires implementing an alternative mechanism to provide the IP-MAC mapping needed by hosts.

In traditional networks, when a destination is outside the subnet, an IP-MAC mapping is defined with the interface of the router responsible for forwarding the packet to the following network. This router is called the *default gateway*. In order to support this mapping in SDN networks, some controllers implement a virtual default gateway, e.g., ONOS [11], but others, like OpenDaylight [12], do not. For the latter case, one possible solution is to implement a custom application that replaces the functionality provided by the gateway [8]. However, the time required to analyze, develop and test such an application, inside the existing code of a controller, quickly becomes prohibitive. Especially when initially testing the feasibility of an SDN deployment, or researching SDN characteristics that are not related to the ARP protocol.

This research has identified four solutions for the IP-MAC mapping problem that do not require modifying the basic functionality of an SDN controller. As such solutions do not modify the controller's code, we have termed them controller-independent strategies.

The main contribution of this work is to provide experimental proof of the behavior of each solution, as well as an analysis of the advantages and characteristics that need to be considered before deciding which one to adopt. The comparison highlights the strengths, limitations, and trade-offs associated with each approach. Additionally, this study includes a comparison between a physical implementation and an emulated environment for these solutions. Ultimately, it is up to network administrators to select a strategy based on their particular needs.

Two important implementation decisions need to be considered when reading the results in this work. The first one is that the DC network was only divided into two IP subnets, each implemented by an OpenFlow switch. Larger multi-subnet networks will increase the demand at the controller or introduce additional routing, which are outside the scope of this paper. Although this does not directly translate to the functionality provided by multi-subnet networks, this research will only focus on ARP behavior, which is contained within the subnet.

The second factor to consider is that all data packets are forwarded using IPv4 addresses exclusively. MAC addresses are only used to support the IP-MAC mapping required by each host. Although no experiments were implemented in IPv6, there is no evidence that the results should change significantly for networks that employ this protocol [13].

This paper is organized as follows. Section 2 provides background information used to develop this research. This includes a short description of ARP functionality and related work by other researchers. Section 3 describes the four different strategies employed to solve the IP-MAC mapping problem. Section 4 outlines the test scenarios and performance metrics used to evaluate each strategy. Section 5 presents the experimental results obtained, as well as a comparison of the performance between a physical implementation and an emulated environment. Section 6 provides an analysis of the benefits, considerations and security trade-offs provided by each strategy, and a comparison between the physical and emulation results. Finally, Section 7 summarizes the findings and provides conclusions for this article.

2. Background

2.1. The Address Resolution Protocol

In SDN networks, packets are processed and forwarded by the SDN switches (data plane) following the instructions, also known as *flows*, provided by the controller (control plane) [4]. One of the most common protocols for exchanging instructions between switches and the controller is OpenFlow, which allows filtering and processing of packets based on characteristics such as MAC address, IP address, TCP port, switch port, and others. Therefore, this type of network does not require the use of MAC addresses or the ARP protocol to forward packets. However, hosts need to satisfy the IP-MAC mapping before they can exchange data packets.

ARP is responsible for defining the IP-MAC mapping between hosts in a network [6]. To define this mapping between two nodes, let us call them host A and host B; A needs to send an *ARP request* packet to find the MAC address of host B, using B's known IP address. This packet has a special Ethernet type (0x0806), uses the broadcast MAC address (FF:FF:FF:FF:FF:FF hexadecimal) as the destination, and A's MAC address as the source address. An example of this packet is shown in Figure 1. Additionally, this packet includes four ARP-specific fields that are set as follows: the sender IP and sender MAC addresses are set using A's network configuration, the target IP address is set to B's IP address, and the target MAC address is just set to 00:00:00:00:00:00 hexadecimal. Since this request uses the broadcast MAC address as the destination, the packet is sent to all the hosts in the subnet.

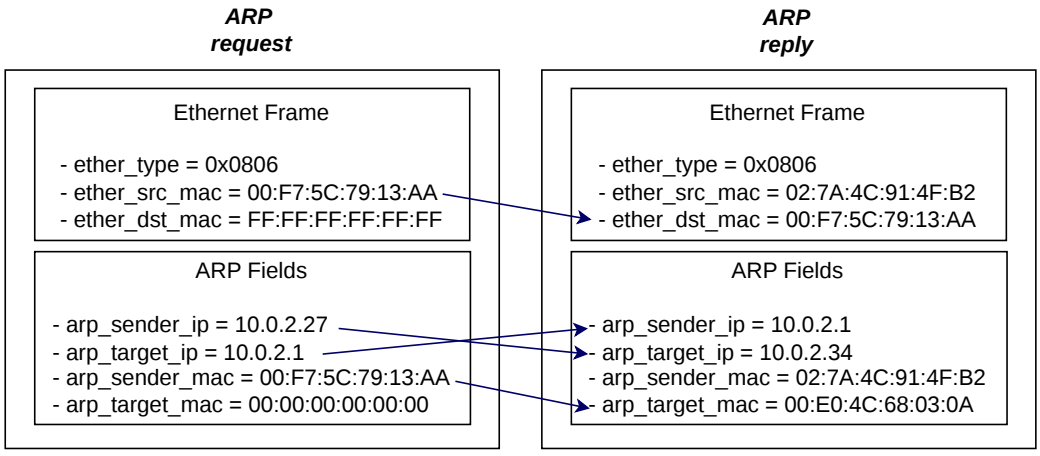

Figure 1. IP–MAC mapping process.

Host B responds to A's request with an *ARP reply* message. This packet also uses ARP's Ethernet type, host B's MAC address now becomes the source MAC address, and A's MAC address is used as the destination address. Since B has all the required information,

the four ARP-specific fields now contain the correct MAC and IP addresses for both hosts. The packet is sent back to A and both nodes update their ARP tables with the correct IP-MAC mapping.

If host B is not on the same subnet, host A must send the packet to its default gateway, which then forwards it toward the destination. In this case, host A uses B's IP address as the destination IP, but it sets the Ethernet destination address to the gateway's MAC address. However, before A can transmit the packet to the gateway, it must first obtain the gateway's MAC address using the procedure described earlier. Once the packet reaches the gateway, it is forwarded to host B's subnet, where the corresponding gateway must obtain host B's MAC address to deliver the packet.

2.2. Related Work

Existing approaches for handling ARP in SDN differ mainly in the network assumptions they adopt. Most proposals target a single domain where ARP optimization focuses on suppressing broadcast packets or centralizing the IP-MAC mapping. Accordingly, this review first analyzes a set of solutions that explicitly assume a single subnet environment. For example, Cho et al. [8] presented a centralized ARP proxy architecture implemented directly on the controller. In their design, the controller integrates a module that proactively builds and maintains an ARP table by retrieving IP-MAC mappings. When an ARP request arrives, the controller intercepts it, consults the internal table, generates the corresponding reply, and installs a flow rule on the switch to avoid future controller involvement. In their experiments, this approach reduced ARP traffic by up to 99%. This work represents a baseline for subsequent ARP management mechanisms in SDN.

For multiple controller deployments, Du et al. [15] proposed a suppression algorithm. In their approach, ARP reply messages are used to generate suppression entries, which are then synchronized across all controllers to ensure global awareness of resolved IP-MAC mappings. Once an ARP request matches an entry, the request is suppressed, thus limiting its propagation scope and reducing the controller overhead. This approach highlights that broadcast can be mitigated through control plane mechanisms, even when multiple controllers are involved. At the same time, it assumes that suppression information is correctly shared among the controllers. A key limitation, however, is that incomplete suppression information may prevent ARP requests from being correctly suppressed. Consequently, this solution highlights the dependence of multi-controller ARP handling mechanisms on a fully coordinated control plane state.

A similar approach requiring modifications to the SDN controller for handling destination MAC addresses not present in the switch forwarding table is proposed by Song et al. [16]. Their method detects frames with an unknown destination MAC address and installs targeted unicast forwarding rules by searching for known MAC locations across multiple switches. Although effective in reducing broadcast and energy consumption, it introduces additional control plane complexity and memory usage due to its dependency on maintaining multiple state tables and real-time path computation.

The strategies reviewed above rely on a single subnet configuration for the DC. In contrast, the following studies focus on multiple subnet environments. In this context, Di Lallo et al. [17] proposed an architecture that confines ARP traffic to network ingress points. Their approach allows hosts to initiate data transmission without knowledge of the destination MAC address, while the controller issues ARP requests from switches to obtain the destination MAC address, and then installs the forwarding rules. This work provides a reference model for ARP handling in multiple-subnet deployments, where ARP optimization is achieved through modifications to the SDN controller.

An alternative strategy, which avoids modifying the SDN controller by implementing ARP handling directly within OpenFlow switches, was introduced by Alharbi and Portmann [18]. This strategy was named *SProxy ARP*, and enables OpenFlow switches to generate ARP replies locally by rewriting incoming ARP request packets using pre-installed flow rules. These rules match ARP requests for specific IP-MAC mappings and use OpenFlow field-rewriting actions to transform the incoming ARP request into an ARP reply. This technique eliminates the need to forward requests to the controller during runtime, thereby significantly reducing ARP response time. A feature of this approach is that all necessary flow rules are installed by the controller during the network initialization phase, after which the data plane autonomously handles IP-MAC mapping. As a result, SProxy ARP provides an efficient data plane ARP handling mechanism. This strategy is revisited in Section 3.4.

Overall, the reviewed literature shows that most ARP optimization proposals in SDN rely on assumptions of a single subnet or a controller functionality that suppresses broadcast packets and manages IP–MAC mappings. While these approaches are effective under their own assumptions, they increase control plane complexity and make them dependent on specific controller implementations across different SDN deployments. In contrast, fewer studies explore ARP handling mechanisms that operate independently of controller alterations. This gap motivates the exploration of strategies that avoid controller modifications while remaining compatible with standard OpenFlow behavior.

3. Solving the IP-MAC Mapping Problem

This section describes four different strategies to solve the IP-MAC mapping problem. The first two, in Sections 3.1 and 3.2, eliminate ARP packets from the network, which means neither the controller nor the switches need to handle this type of packet. The last two allow transmission and handling of ARP packets. One requires full support from the controller (Section 3.4), while the other reduces demand at the controller at the expense of an additional host (Section 3.3). Table 1 summarizes the role of the host, switch, and controller in ARP handling for each strategy.

Table 1. Network element roles per ARP strategy.

Strategy	Host	Switch	Controller
Disable ARP	Configuration	–	–
Static Entries	Configuration	–	–
Fake Gateway	–	Follow OF rules	Install OF rules
SGAR	–	Follow OF rules	Install OF rules

The strategies included in this study allow to deliver packets originated in the same subnet. However, they cannot handle packets that come from another network because there is no real gateway to facilitate this operation. Therefore, Section 3.5 describes a technique to deliver data packets when they originate at a different subnet.

3.1. Disable ARP

The first strategy is to disable the ARP protocol on each host and interface. Thus, hosts stop producing ARP requests since the IP-MAC mapping is no longer employed. In practice, this configuration is only available on Linux operating systems using the following command for each network interface that requires disabling [19]:

```
ip link set dev <interface> arp off
```

Once ARP has been disabled, all data packets generated by the host are transmitted without prior MAC resolution. In practice, the host constructs the Ethernet frame with

a destination MAC address equal to its own MAC address, and the packet is forwarded using only the destination IP address and the OpenFlow rules installed in the switches.

The main advantage of this approach is the complete suppression of ARP, which significantly reduces broadcast packets and improves traffic predictability. Additionally, it simplifies analysis by avoiding unsolicited control messages. However, disabling ARP may interfere with applications or other protocols that require this protocol, potentially causing connectivity issues if the flow configuration is inconsistent.

3.2. Static Entries

In this strategy, ARP remains enabled on host devices, but its dynamic behavior is suppressed by manually configuring all subnet destinations into each host's ARP table. Instead of broadcasting ARP requests, IP-MAC mappings are explicitly defined using system commands such as

```
arp -s <ip-address> <mac-address>
```

in Windows [20], or

```
ip neigh add <ip-address> lladdr <mac-address> dev <interface>
```

in Linux [21].

Once the mappings have been completed, hosts bypass the ARP protocol for all destinations defined. During packet transmission, the destination MAC address is retrieved directly from the static ARP table, and the Ethernet frame is constructed using this address. Notice that an additional entry is needed for the default gateway's IP address.

This strategy eliminates ARP broadcast packets while keeping this protocol enabled at the host level, thus preserving compatibility with existing operating systems and applications. However, the static nature of this configuration limits scalability because any change in host addressing or topology requires manual updates across all the nodes in the subnet. Specifically, for a subnet with H hosts, the number of IP-MAC mappings required is H^2, and a new host h_{i+1} increases this number by $2H + 1$, since an additional entry is needed to consider the default gateway. For example, a network with three hosts ($H_1 = 3$), requires 9 IP-MAC mappings: each host needs two mappings to their neighbors and one to the default gateway. If this network grows to four hosts ($H_2 = 4$), the number of additional mappings increases by $2(3) + 1 = 7$, to make a new total of 16.

3.3. Fake Gateway

This strategy is implemented using an additional independent host which emulates the behavior of a default gateway, thus called a *fake gateway*. Once an IP address has been assigned to this fake device, it listens for ARP requests from the other hosts and automatically replies using its own MAC address, without any intervention from the controller. This fake gateway does not process or forward any IP traffic, its sole purpose is to resolve the IP-MAC mapping. Hosts employ this MAC address as destination when sending packets outside the subnet.

For this strategy, ARP packets need to be retransmitted to all hosts in the network. Therefore, a flow that broadcasts ARP packets needs to be added at every SDN switch that handles a subnet. OpenFlow's NORMAL reserved port [14] is usually used to broadcast packets to other ports in an SDN switch.

This setup enables IP-MAC mapping at the subnet without producing additional demands on the SDN controller. Adding new hosts to the subnet is straightforward, since they only need to configure the fake gateway's IP address. This strategy also provides compatibility with standard operating systems. However, ARP broadcast traffic is still present in the network. In addition, it introduces a dependency on the availability of the

fake system: If this device becomes unavailable or unresponsive, the IP-MAC mappings will fail. A final consideration is that at each subnet, one switch port must be reserved for the fake gateway.

3.4. Switch Generated ARP Replies

The last strategy is based on configuring OpenFlow flows that enable SDN switches to intercept ARP requests for specific destinations and generate their corresponding ARP replies. This approach is based on the SProxy ARP technique introduced by Alharbi and Portmann [18]. In this paper, it will be called Switch-Generated ARP Replies (SGAR). If the controller is configured in a proactive mode, forwarding rules may be installed in the switch before any traffic occurs, which will in turn reduce the controller's load.

For this strategy, flows modify ARP request header fields to generate a valid ARP reply. This packet is then sent back to the source host via the incoming port, using OpenFlow's IN_PORT reserved port. For example, when host 10.0.2.34/24 intends to send a packet to host 10.0.1.87/24, which is in a different subnet, it first needs to issue an ARP request to address 10.0.2.1, as it attempts to define the IP-MAC mapping for the default gateway. An example of this request message and its response is shown in Figure 2. Notice that the reply packet uses the gateway's MAC address as the sender address, which is 02:7A:4C:91:4F:B2 in this example. To generate this response, the controller requires a flow rule that filters the incoming ARP packet: Ethernet type (0x0806), source IP (10.0.2.34), target IP (10.0.2.1), and ARP operation 1 (request); and generates a response using the values shown in Figure 2. Notice that the ARP operation is now 2 (reply) and the output is set to IN_PORT. After receiving the reply, the source host installs this IP-MAC mapping, and it can now proceed to transmit regular data packets.

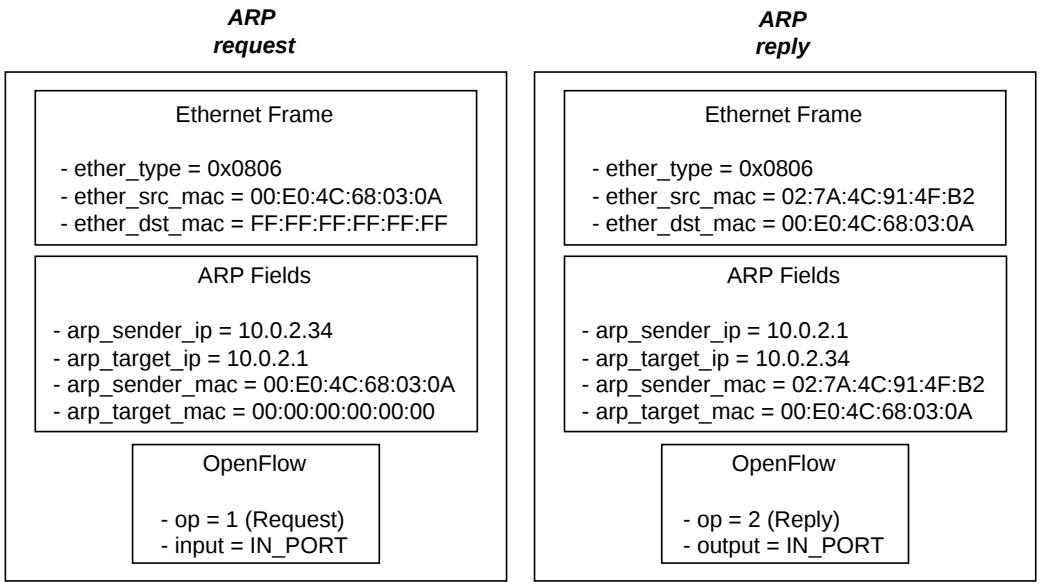

Figure 2. ARP packet handling SGAR strategy.

The main advantage of SGAR is that no additional host configuration or devices are required. It also aligns well with the SDN paradigm by maintaining centralized control logic while minimizing control plane interaction. However, this strategy may encounter implementation challenges: Switches that do not offer full OpenFlow support [22] may limit the availability of this method in production environments. Additionally, each destination requires a dedicated flow entry per host, which consumes switch memory in a quadratic proportion, just as for the Static Entries strategy. This is explicitly described at the end of Section 3.2.

3.5. Delivering Fake ARP Packets

The four strategies described before allow data packets to leave a subnet; however, they do not describe how to deliver such packets to a different destination subnet in the DC. In other words, hosts also need to accept data packets from their SDN switch without performing a real IP-MAC mapping. To achieve this, the following procedure may be used: Once the packet has reached the destination subnet, the SDN switch changes the packet's destination MAC address to the broadcast address and sends it only to the host's destination port. This operation is illustrated in Figure 3 and it ensures successful delivery based on a predefined flow that matches the destination IP with its corresponding output port. The number of flows needed to implement this delivery is equal to the number of hosts in the subnet.

In summary, the four ARP management strategies explored in this section offer trade-offs between simplicity, scalability, and implementation complexity; while all of them avoid modifying the controller to implement gateway functionality. These approaches enable controlled inter-subnet communication in SDN environments lacking native ARP support. The following section describes the experiments used to evaluate their performance.

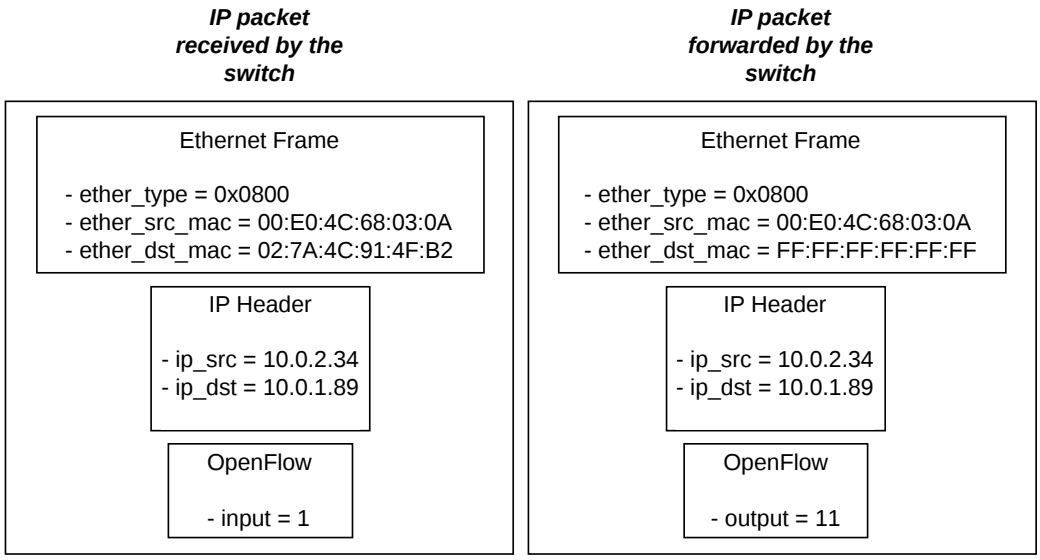

Figure 3. IP packet handling in the switch.

4. Experiments

Two series of controlled experiments were conducted to evaluate the effectiveness of the proposed strategies; the first one was conducted in a physical implementation, while the second was emulated in Mininet. Both series included three different types of experiments, which are further described in the following subsection. The experiments in Sections 4.1 and 4.3 were carried out for the four strategies included in this paper, while the experiment described in Section 4.2 only included the Fake Gateway and SGAR strategies.

All the experiments employed the spine-leaf network topology shown in Figure 4. This architecture consists of two leaf switches, $S3$ and $S4$, that connect hosts to the network; and two spine switches, $S1$ and $S2$, that interconnect the leaf switches [23]. Although this topology was selected for evaluation, paths are defined using static OpenFlow forwarding rules. As a result, similar behavior to that of a traditional tree topology is expected.

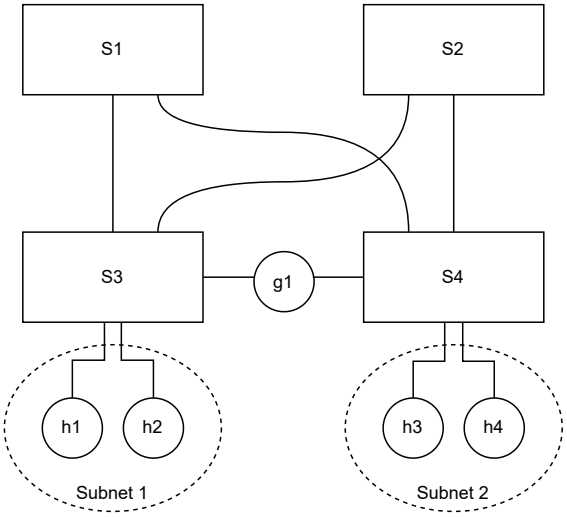

Figure 4. Spine–leaf topology with 5 Hosts (1 as gateway), 2 leaf switches, and 2 spine switches.

The first series of experiments was carried out on a physical network that used four Aruba switches, model 2930F (24G PoE+ 4SFP), which were managed by an OpenDaylight (ODL) controller, version Scandium-SR2. The OpenFlow plugin was enabled by installing the following ODL features on the controller [12]:

- `odl-openflowplugin-flow-services-rest;`
- `odl-openflowplugin-app-table-miss-enforcer;`
- `odl-openflowplugin-nxm-extensions.`

Figure 4 shows the topology, which consisted of two spine switches ($S1$, $S2$), two leaf switches ($S3$, $S4$), and four hosts, two per leaf switch. Each leaf switch provided service to a different subnet. For the Fake Gateway strategy, an additional host ($g1$) was introduced to act as the fake system (Section 3.3). Hosts were independent personal computers (PC) that ran different Linux distributions: Ubuntu 25.04 for $h1$, 24.04.2 for $h2$, and 24.04 for $h3$ and $h4$, while $g1$ used Lubuntu 20.04.

The second series of experiments emulated the same network topology, using Mininet version 2.3.0, which was also managed by an ODL controller, version Scandium-SR2, with the same OpenFlow plugin features installed. The emulation environment was deployed on a Dell Latitude 7390 laptop with 16 GB of RAM running Ubuntu 24.04.2. To emulate the physical setup more accurately, all virtual links in the Mininet topology were configured with a bandwidth limit of 1000 Mbps, which matches the 1 Gbps interfaces of the Aruba switches.

For both series and three different types of experiments, a common set of OpenFlow rules was defined for the controller and then installed at the SDN switches before transmitting any traffic. This set of rules provided basic IP connectivity using the following strategy: For each spine switch, flows were configured to forward packets received on one port to the opposite port, thus enabling bidirectional communication between the leaf switches. For the leaf switches, flow rules forwarded packets based on the destination IP by assigning a specific output port for each target IP address. Since the OpenFlow rules are common, the experiments focus on the measurement methodology and performance indicators described in the following subsections.

4.1. Ping Response Time

Ping Response Time (PRT) was used to assess communication delay between hosts. PRT was measured using the ping tool, which sends ping request messages and measures

the time required to receive the replies. For each test case, pings were sent sequentially, such that each host sent 11 ping requests to a given destination before proceeding to the next host. The complete sequence was repeated three times to ensure consistency.

Importantly, the ARP protocol runs only in the first ping and only for the strategies that exchange ARP messages (Fake Gateway and SGAR). Therefore, the ARP cache was cleared on all hosts before each sequence of experiments using the command

`ip neigh flush all` [21].

In contrast, for the strategies where the IP-MAC mapping is configured manually, ARP resolution does not occur at runtime.

4.2. ARP Response Time

ARP Response Time (ART) was measured to further quantify the delay introduced by the IP-MAC mapping. Measurements were performed using arping, a tool that sends ARP request packets and measures the time required to receive the replies. 20 ARP requests per host were generated using the following command:

`arping -i <interface> -c <count> <ip-address>`

Where `-i` specifies the network interface, and `-c` defines the number of ARP request packets to be transmitted. For each host, ARP requests were sent to the peer host in the same subnet and to the default gateway. ARP caches were cleared before each measurement to ensure equal initial conditions. This metric was only evaluated for Fake Gateway and SGAR, since ARP is not functional in the other two strategies. Inter-subnet measurements were not performed, since ARP resolution does not occur across different subnets.

Measurement of ART forces hosts and the network to process several ARP requests in a short period of time. Since this tool transmits approximately one request per second by default, the generated load was about 1 ARP packet per second per host during ART measurements. Therefore, this may be considered as a stress test for the two strategies evaluated with this tool.

4.3. Jitter and Packet Loss Ratio

Jitter and Packet Loss Ratio (PLR) were measured using iPerf, a traffic generation tool, in UDP mode. The receiving host was configured using the command

`iperf -u -s -p <port>`

and the traffic was generated from the sending host with

`iperf -u -c <ip-address> -p <port> -b <bandwidth> -t <time>`

where `-u` enables UDP mode, `-s` starts the receiving host, `-c` specifies the destination host IP address, `-p` defines the communication port, `-b` sets the target transmission rate, and `-t` indicates the duration of the experiment.

PLR was obtained directly from iPerf reports and represents the fraction of UDP packets lost as the traffic load increases. Jitter was also reported by this tool and reflects the variation in inter-packet delay, serving as an indicator of traffic stability. iPerf computes jitter following the definition of inter-arrival variation in RFC 3550, as the smoothed mean of differences in consecutive packet transit times [24].

Inter-subnet communication was evaluated by generating traffic simultaneously between the following hosts: $h1 \rightarrow h3$, $h3 \rightarrow h2$, $h2 \rightarrow h4$, $h4 \rightarrow h1$. For each traffic flow, the load was progressively increased in bandwidth and time. Test traffic started at 100 Mbps for 30 s, and then increased in steps of 100 Mbps up to a maximum of 1200 Mbps. The duration also increased by 30 s at each step, up to a maximum of 300 s. Although the bandwidth

of each switch port is 1000 Mbps (1 Gbps), the load was increased beyond this value to saturate the links and evaluate how each ARP strategy behaves under high load. Since each host handles two traffic flows, two different UDP ports were also used to avoid port conflicts. Packet loss and jitter were evaluated to assess whether the proposed strategies introduce any observable side effects during the initial communication phase. Since ARP is required only for the initial packet exchange in strategies that rely on IP–MAC mapping, its effect is limited to that initial stage.

5. Results

This section presents the evaluation of the four ARP management strategies implemented in the DC network. Experimental results are reported for the physical implementation (first series of experiments) and Mininet (second series of experiments). The comparison between hardware and emulation results is performed to observe how Mininet approximates the trends and behavior of a real SDN network.

As shown in Figure 5, the average PRT value obtained from the experiment described in Section 4.1 is very similar across all strategies for the physical implementation, ranging between 0.641 ms and 0.707 ms, indicating stable forwarding for different loads. Within this range, the SGAR strategy achieves the lowest average PRT. This figure also includes the 95% confidence interval for each strategy. During this experiment, the Fake Gateway and SGAR strategies experienced higher-than-usual PRT values only for the first exchanged packet. These outlier values are caused by the initial IP-MAC mapping that these strategies require. After the mapping is established, subsequent packets showed consistent PRT values. Therefore, the average PRT is calculated excluding the first outlier value per strategy, as it does not represent the typical PRT. These values are instead reported in Table 2. Since Disable ARP and Static Entries do not perform the mapping, their averages include all recorded values and the maximum is still reported in Table 2.

For the Mininet experiments, the PRT results summarized in Figure 6 exhibit a behavior consistent with the hardware experiments, although with significantly lower values. The average PRT values range from 55.7 µs to 64 µs, again reported with a 95% confidence interval. Similar to the hardware results, the SGAR strategy achieves the lowest average PRT. The lower values in Mininet are expected, as communication occurs in an emulated environment rather than across a physical network. The performance among the evaluated strategies remains consistent with the results from the physical implementation. However, in some tests, the first packet showed a higher value than subsequent packets, regardless of whether IP-MAC mapping was executed. This initial outlier occurs because Mininet uses Open vSwitch, which performs additional processing to establish forwarding rules when handling the first packet of a new flow, after which subsequent packets across all evaluated strategies experience more stable behavior, as described in the Switching–Performance section of the Open vSwitch documentation [25]. Consequently, and following the same criteria described for the physical implementation experiments, the maximum PRT values reported in Table 2 do not take these extra-processing values into consideration and instead report the packet that performed the IP-MAC mapping. In this sense, Mininet shows behavior consistent with hardware experiments, where the Fake Gateway and SGAR strategies, which require IP-MAC mapping, have higher maximum PRT values than the Disable ARP and Static Entries strategies.

The experiments measuring ART described in Section 4.2 only include the strategies that send ARP packets, namely Fake Gateway and SGAR. ART results are summarized in Figure 7, where average values are reported with a 95% confidence interval. For the physical implementation, the SGAR strategy shows lower average ART values than Fake Gateway, with average values of 0.6188 ms and 1.3856 ms, respectively. This behavior can

be explained by the number of packet hops required by each strategy. For Fake Gateway, an ARP request requires four hops: *host* → *switch* → *fake gateway*, and the response follows the same path in reverse. In contrast, with SGAR, the ARP reply is generated by the switch, so only two hops are needed: *host* → *switch* and back. This reduction from four to two hops is reflected in the measurements, where the average ART from SGAR is approximately half of that for Fake Gateway.

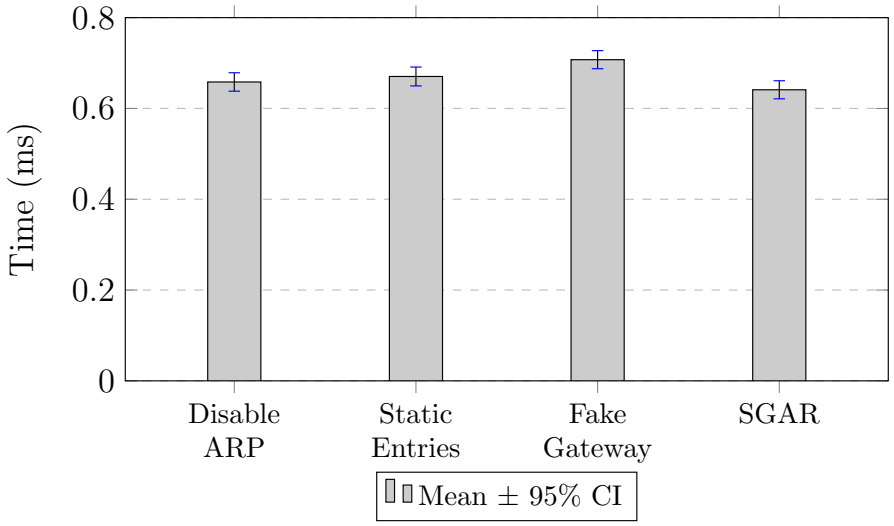

Figure 5. PRT results for physical implementation–Comparison by strategy.

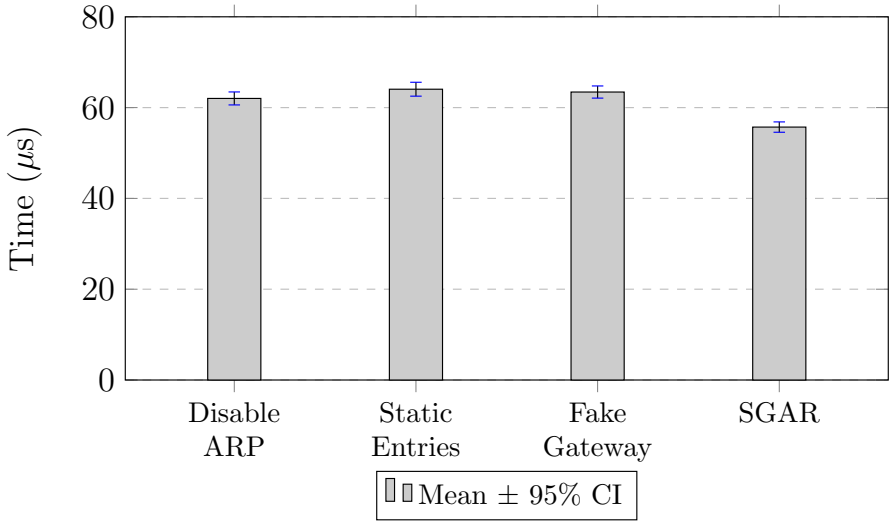

Figure 6. PRT results for Mininet–Comparison by strategy.

Table 2. Maximum PRT in milliseconds.

Strategy	Phys. Impl.	Mininet
Disable ARP	0.7578	0.0846
Static Entries	0.7782	0.0875
Fake Gateway	2.7713	0.9345
SGAR	1.6147	0.4950

In Mininet, ART behavior differs from that observed in hardware. Although Fake Gateway requires a higher number of hops, these are processed within the emulation environment, without traversing physical links, resulting in a very low average ART value of 0.0114 ms. In contrast, the SGAR strategy shows higher average ART values, reaching

0.1702 ms, as ARP handling requires explicit modification of packet fields, which introduces additional processing cost in Mininet [25].

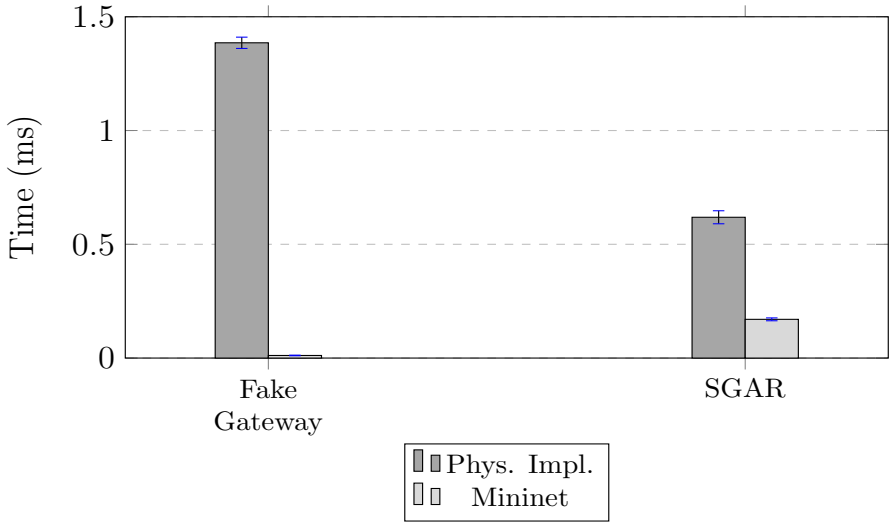

Figure 7. Average ART results for physical implementation and Mininet, with 95% confidence intervals–Comparison by strategy.

Figure 8 shows the jitter boxplots obtained from the experiment described in Section 4.3 for the physical implementation, which had very similar ranges across all four solutions, indicating that jitter values remain comparable regardless of the strategy. Median values remain close to 23 µs for all strategies. Higher-than-expected jitter values around 40 µs occasionally appear in all strategies except for Static Entries, but these are isolated cases and do not indicate an increase in jitter. Minor differences can be observed, such as slightly more consistent values for SGAR and lower average jitter for Static Entries. Overall, these variations are small and do not show a clear difference between them.

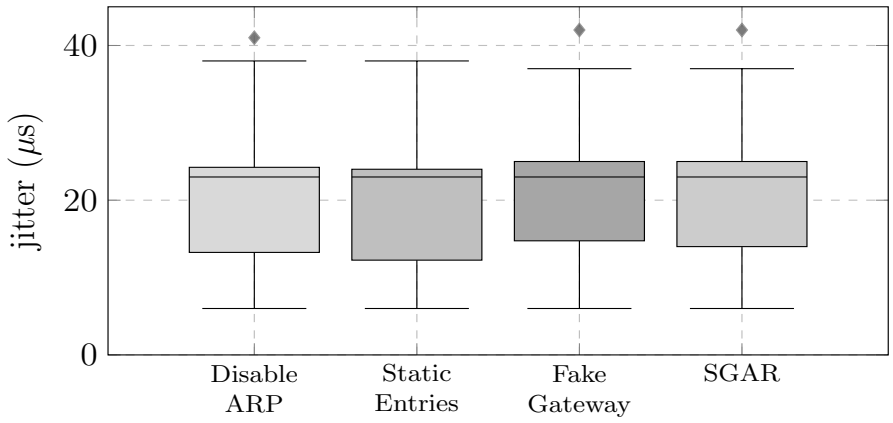

Figure 8. Jitter distribution for physical implementation–Comparison by strategy.

In Mininet, the jitter values remained at lower levels than those observed on the physical implementation. For the Disable ARP, Static Entries, and Fake Gateway strategies, only the first measurement deviates from the stable value, exhibiting a jitter value of 2 µs, as shown in Figure 9a whereas all subsequent jitter values remain constant at 1 µs. For the SGAR strategy, the first measurement exhibits a higher jitter, taking values of either 3 µs or 4 µs, followed by two measurements with a jitter of 2 µs. After these initial values, the jitter stabilizes at 1 µs, as reflected in Figure 9b. This behavior is consistent with the additional packet processing required for the SGAR strategy in the emulation

environment. Nevertheless, the jitter values remain similar to the behavior observed on hardware, indicating that jitter variation remains minimal across all strategies.

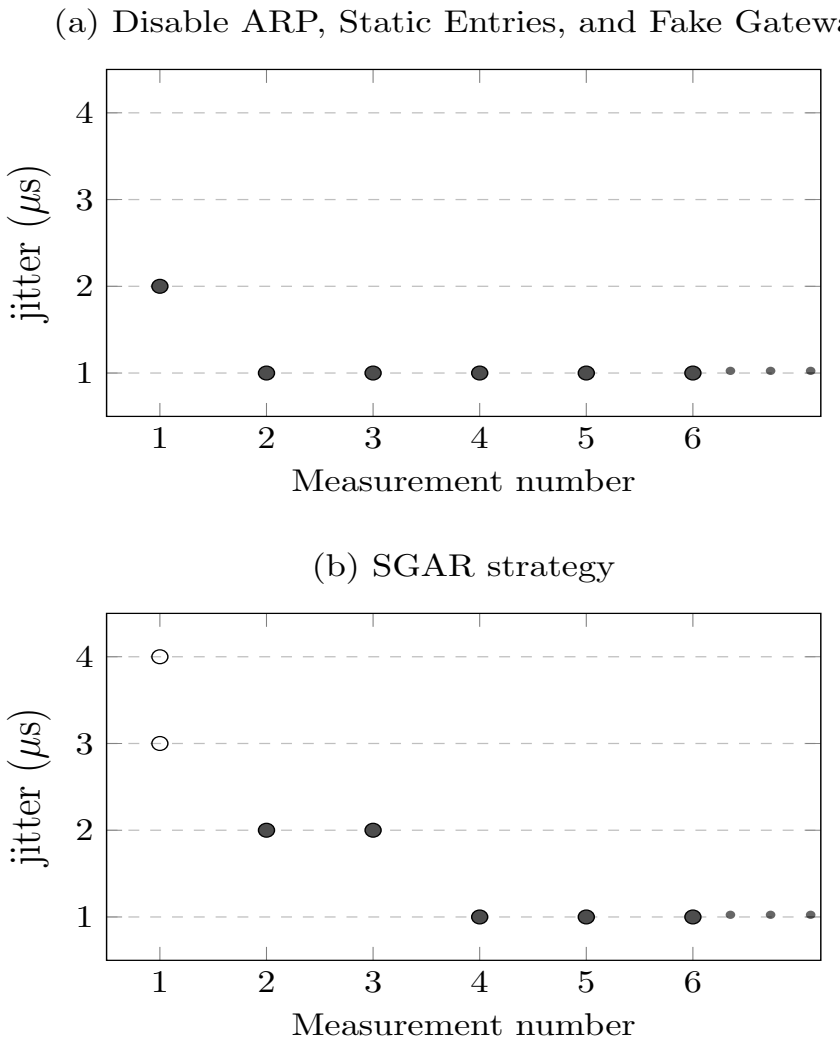

Figure 9. Jitter behavior for Mininet: (**a**) results for the first three strategies; (**b**) results for the SGAR strategy. The white dot indicates the point that may take two possible values.

In terms of PLR, the results obtained from the experiment described in Section 4.3 show a clear divergence between the hardware and emulated environment. On the physical implementation, PLR is minimal for all strategies evaluated, as summarized in Table 3, which reports both the average and the Standard Deviation (SD). The observed losses occur sporadically without any consistent pattern. This behavior indicates that, even under increased traffic load, the hardware provides reliable packet delivery and that the different strategies do not introduce any noticeable packet loss.

In Mininet, packet loss is consistently higher than in hardware experiments. The average PLR remains within the range from 3.25% to 3.60% across all strategies, with considerable variability across tests. This dispersion suggests that packet loss in the emulated environment is affected by characteristics related to software packet processing, rather than by the strategy employed. Among the evaluated strategies, Disable ARP shows the lowest average PLR value; the difference with respect to the other strategies is small and does not allow PLR to be directly attributed to a specific approach.

Table 3. PLR in percentage.

	Phys. Impl.		Mininet	
Strategy	**Avg**	**SD**	**Avg**	**SD**
Disable ARP	0.0008	0.0012	3.25	2.36
Static Entries	0.0011	0.0015	3.51	2.57
Fake Gateway	0.0069	0.0195	3.60	2.61
SGAR	0.0115	0.0249	3.46	2.46

6. Discussion

6.1. Comparing the Four Strategies

The results provided in the previous section need to be contextualized with the inherent properties of each of the solutions tested in this investigation. Table 4 provides a summary of the main characteristics of each strategy, which are further discussed in the following paragraphs.

Starting with Disable ARP, the experiments demonstrate that it has a very stable and predictable behavior since it avoids using the ARP protocol. However, in practice, deactivating ARP is only available in Linux operating systems. Therefore, this strategy can only be implemented in Data Centers that exclusively employ this operating system.

Static Entries avoid sending ARP traffic into the network, which produces the same benefits as disabling ARP. However, it requires manually configuring each host's ARP table. Section 3.2 has previously stated that the number of instructions required for this configuration is H_i^2, where H_i is the number of hosts in subnet i. Therefore, the total additional instructions (I_{SE}) for a network with S subnets is

$$I_{SE} = \sum_{i=1}^{S} H_i^2 \tag{1}$$

It is important to consider that there are automation tools, such as Ansible [26], which could provide automatic ARP table configuration for the hosts in a Data Center. However, the application and analysis of such tools are beyond the objectives of this study.

The Fake Gateway approach produced the longest PRT (Figure 5) because it provides full ARP support, including the associated cost produced by its broadcast traffic. Considering that a switch port needs to be reserved for the fake system, this is a very simple-to-implement strategy that only requires one additional flow at the SDN controller to move ARP packets to and from the gateway. Therefore, the number of additional flows (F_{FG}) is equal to the number of subnets (S) in the DC network.

$$F_{FG} = S \tag{2}$$

The SGAR strategy produced slightly better response times than Fake Gateway, since it does not fully implement the ARP protocol and does not produce all its associated traffic. However, as discussed in Section 3.4, it requires configuring additional flows at the SDN controller. The number of additional flows per subnet is H_i^2, where H_i is the number of hosts in subnet i. Therefore, the total additional flows (F_{SGAR}) for a network with S subnets is

$$F_{SGAR} = \sum_{i=1}^{S} H_i^2 \tag{3}$$

This expression is mathematically equivalent to Equation (1), although it represents OpenFlow rules instead of static ARP entries.

Additionally, the setup time demanded by each strategy varies depending on where the configuration occurs, the number of hosts, and the required flexibility in network address distribution. For small-scale environments, Disable ARP is a suitable solution, as it only requires disabling ARP directly on each host and interface. However, in networks with a large number of hosts, the Fake Gateway strategy becomes more suitable, as it uses a single ARP flow entry per subnet, avoiding configuring each host. In contrast, Static Entries and SGAR require that IP-MAC mappings be constantly maintained on each host. While Static Entries stores this mapping at each host, SGAR uses OpenFlow rules at the controller. This centralization simplifies management, since updates can be performed from a single point in the network.

It is also important to consider that the maximum number of OpenFlow entries supported by the Aruba 2930F switch in its flow tables is 16,000, a limit imposed by the hardware resources available [27]. In a scenario in which the 24 access ports in the switch are occupied, each host will require ARP resolution entries for all other hosts in the subnet; applying Equation (3), this corresponds to 576 possible ARP entries, which is still well below the maximum capacity supported by the switch.

Table 4. Summary of strategies: results, advantages, and disadvantages.

Strategy	Advantages	Disadvantages
Disable ARP	• Eliminates ARP broadcast traffic. • Easy to implement.	• Restricted to Linux operating systems.
Static Entries	• Eliminates ARP broadcast traffic.	• Requires H^2 instructions per subnet.
Fake Gateway	• Only one additional flow per subnet. • Easy to add hosts.	• Requires a dedicated host that consumes a switch port. • Produces ARP broadcast traffic.
SGAR	• Generates partial ARP traffic. • Handled exclusively by the SDN controller.	• Requires H^2 flows per subnet.

6.2. Security Trade-Offs

This subsection analyzes the security trade-offs associated with not employing ARP in an SDN network, as well as the vulnerabilities introduced by the four strategies.

ARP cache poisoning has been identified as a security problem in SDN networks [28,29]. This type of attack occurs when a host MAC address gets replaced by the attacker's address, also called address spoofing. The attacker may then use this replacement to hijack information or conduct a man-in-the-middle attack. Since the strategies analyzed here allow the DC network to forward packets using only the IP address, the attack surface is significantly reduced. Additionally, the MAC address is not used by the network for forwarding decisions, which reduces the effectiveness of MAC address hijacking attacks, although other attacks based on IP address spoofing remain possible.

However, the following additional vulnerabilities should be taken into consideration: Disable ARP may interfere with applications or protocols that employ ARP functionality, as stated in Section 3.1. Static Entries requires manually configuring the ARP table for each host, which may introduce human errors. Fake Gateway's main vulnerability is that the device may fail, either accidentally or by means of an attack, which will then prevent the IP-MAC mapping from completing successfully. Finally, the SDN controller is the weakest link for the SGAR strategy, since flows could be maliciously manipulated if this device is compromised.

6.3. Physical Implementation vs. Emulation Results

Beyond the characteristics of the strategies evaluated in this investigation, the experimental results provide a contrast between the physical and emulated environments. When examining the PRT metrics, both environments produced consistent behavior with bounded variability across all strategies. Moreover, although the maximum PRT differs between the environments, the trend is maintained and reflects that strategies that implement ARP (Fake Gateway and SGAR) have larger values than those that avoid ARP messaging (Disable ARP and Static Entries).

Regarding ART, hardware and emulated environments produced opposite results. For the Mininet experiments, although both strategies operate within the emulation environment, SGAR requires explicit packet field modifications, introducing additional processing overhead, whereas the Fake Gateway strategy does not. Therefore, the former produced worse results than the latter, which contradicts the results obtained by the physical implementation.

The observed jitter values are extremely low, remaining in the microseconds range in both hardware and emulation environments. These levels are well below the thresholds typically considered acceptable for latency-sensitive applications, such as videoconferencing or interactive services, where jitter values of several milliseconds are generally tolerated [30]. Therefore, the results indicate that none of the ARP management strategies evaluated generates a noticeable variation in packet delay, and thus does not negatively affect real-time traffic in SDN DC networks.

Regarding PLR, although physical and emulated environments produced different results, they are both consistent among the different strategies. Moreover, the results produced by emulation are consistent with prior findings by Hardin et al. [31], who showed that Mininet and iPerf may report misleading throughput and loss metrics, even in simple topologies, due to shared CPU resources, limited internal buffering, and timing artifacts under load.

In summary, although Mininet does not produce the same results as the experiments in the physical implementation, it captures the behavioral trends and interactions between the different strategies in most of the experiments performed.

7. Conclusions

Evaluation of four different ARP handling strategies in a DC SDN network reveals performance, operational and security trade-offs among them. Although all four solve the IP-MAC mapping problem without modifying the logic of the SDN controller or developing additional modules, their behaviors differ in terms of performance, scalability, and deployment complexity.

Across all hardware experiments, SGAR provided the most balanced performance. It exhibited low and stable response times, bounded jitter, negligible packet loss, and predictable behavior under increasing traffic load. These results indicate that handling ARP directly in the data plane aligns well with the capabilities of SDN switches, providing performance without relying on external hosts or controller intervention during runtime. However, this performance advantage comes at the cost of scalability, as it becomes necessary to install a quadratic number of ARP flows. Thus, its applicability in very large networks depends on the SDN controller capacity and memory limitations

The remaining strategies highlight alternative trade-offs. ARP Disable and Static Entries effectively eliminate broadcast traffic and simplify packet forwarding, but they have limited portability and poor scalability due to host configuration requirements. The Fake Gateway approach reduces host configuration effort and preserves operating system compatibility. However, it produces ARP broadcast traffic and introduces additional

network dependencies by requiring a dedicated device, without offering the same level of performance observed with SGAR.

Additionally, this study provides insights into the behavior of physical and emulated SDN environments. Although Mininet reproduces the general trends of all strategies, differences were observed in measurements related to packet loss under load. These observations suggest that emulation results should be complemented with physical experiments to draw valid conclusions.

Although the four strategies included in this study are not new, we are the first to perform a qualitative and experimental comparison between them. As such, this work may help network administrators and researchers to take informed decisions when implementing an SDN Data Center, without spending an excessive amount of time deciding how to configure their network and controller. Therefore, we expect this effort will help to promote the use of SDN networks and related experimental activities.

Author Contributions: Conceptualization, J.N.L.-O. and P.D.A.-V.; formal analysis, J.N.L.-O.; investigation, J.N.L.-O. and P.D.A.-V.; funding acquisition, P.D.A.-V.; project administration, P.D.A.-V.; supervision, F.J.R.-A. and A.G.S.-T.; validation, F.J.T.-R.; writing—original draft preparation, J.N.L.-O. and P.D.A.-V.; writing—review and editing, J.N.L.-O., P.D.A.-V., A.G.S.-T. and F.J.T.-R. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Consejo Potosino de Ciencia y Tecnología (COPOCYT), San Luis Potosí, Mexico.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data supporting the findings of this study are publicly available at GitHub: https://github.com/NeftaliLimonOrtiz/Data-ARP-Optimization-in-SDN-Using-Controller-Independent-Strategies-for-Data-Center-Networks (accessed on 10 April 2026).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ARP	Address Resolution Protocol
SDN	Software-Defined Networking
SGAR	Switch-Generated ARP Replies
PRT	Ping Response Time
ART	ARP Response Time
PLR	Packet Loss Ratio
SD	Standard Deviation

References

1. Narten, T.; Karir, M.; Foo, I. Address Resolution Problems in Large Data Center Networks. RFC 6820. 2013. Available online: https://www.rfc-editor.org/info/rfc6820 (accessed on 3 November 2025).
2. Elmeleegy, K.; Cox, A. EtherProxy: Scaling Ethernet By Suppressing Broadcast Traffic. In Proceedings of the IEEE INFOCOM, Rio de Janeiro, Brazil, 19–25 April 2009; pp. 1584–1592. https://doi.org/10.1109/INFCOM.2009.5062076.
3. Mogul, J.; Postel, J. Internet Standard Subnetting Procedure. RFC 950. 1985. Available online: https://www.rfc-editor.org/info/rfc0950 (accessed on 10 December 2025).
4. Peterson, L.; Cascone, C.; O'Connor, B.; Vachuska, T.; Davie, B. *Software-Defined Networks: A Systems Approach*; Systems Approach LLC: La Vergne, TN, USA, 2022.
5. Miguel-Alonso, J. A Research Review of OpenFlow for Datacenter Networking. *IEEE Access* **2023**, *11*, 770–786. https://doi.org/10.1109/ACCESS.2022.3233466.

6. Plummer, D. An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware. RFC 826. 1982. Available online: https://www.rfc-editor.org/info/rfc826 (accessed on 3 November 2025).
7. Narten, T.; Simpson, W.A.; Nordmark, E. Neighbor Discovery for IP Version 6 (IPv6). RFC 2461. 1998. Available online: https://www.rfc-editor.org/info/rfc2461 (accessed on 10 December 2025).
8. Cho, H.; Kang, S.; Lee, Y. Centralized ARP proxy server over SDN controller to cut down ARP broadcast in large-scale data center networks. In Proceedings of the International Conference on Information Networking (ICOIN), Siem Reap, Cambodia, 12–14 January 2015; pp. 301–306. https://doi.org/10.1109/ICOIN.2015.7057900.
9. Comer, D.; Karandikar, R.H.; Rastegarnia, A. DCnet: A new data center network architecture. In Proceedings of the IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 9–11 January 2017; pp. 1–6. https://doi.org/10.1109/CCWC.2017.7868382.
10. Alqahtani, J.; Hamdaoui, B. Rethinking Fat-Tree Topology Design for Cloud Data Centers. In Proceedings of the IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–6. https://doi.org/10.1109/GLOCOM.2018.8647774.
11. Friyanto, A. High Availability Aspects of SDN-IP Reactive Routing. *IOP Conf. Ser. Mater. Sci. Eng.* **2020**, *879*, 012070. https://doi.org/10.1088/1757-899X/879/1/012070.
12. OpenDaylight Project. OpenDaylight Documentation. Available online: https://docs.opendaylight.org/en/latest/index.html (accessed on 9 January 2026).
13. Zhou, X.; Jacobsson, M.; Uijterwaal, H.; Mieghem, V. IPv6 delay and loss performance evolution. *Int. J. Commun. Syst.* **2008**, *21*, 643–663. https://doi.org/10.1002/dac.916.
14. Open Networking Foundation. OpenFlow Switch Specification. Available online: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.2.pdf (accessed on 9 January 2026).
15. Du, J.; Gao, X.; Wang, J.; Liu, S.; Cao, W.; Song, Y.; Jiang, S. Research on an Approach of ARP Flooding Suppression in Multi-Controller SDN Networks. In Proceedings of the IEEE Intl Conf on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking (ISPA/BDCloud/SocialCom/SustainCom), New York City, NY, USA, 30 September–3 October 2021; pp. 1159–1166. https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00159.
16. Song, G.; Wang, H.; Hu, J. Using MSPG to Suppress Flooding in the Data-Link Layer for SDN. *Secur. Commun. Netw.* **2024**, *2024*, 9696548. https://doi.org/10.1155/2024/9696548.
17. di Lallo, R.; Lospoto, G.; Rimondini, M.; Di Battista, G. How to Handle ARP in a Software-Defined Network. In Proceedings of the IEEE NetSoft Conference and Workshops (NetSoft), Seoul, Republic of Korea, 6–10 June 2016; pp. 63–67. https://doi.org/10.1109/NETSOFT.2016.7502444.
18. Alharbi, T.; Portmann, M. SProxy ARP-Efficient ARP Handling in SDN. In Proceedings of the 26th International Telecommunication Networks and Applications Conference (ITNAC), Dunedin, New Zealand, 7–9 December 2016; pp. 179–184. https://doi.org/10.1109/ATNAC.2016.7878805.
19. Linux man-pages project. ip-link—Linux Manual Page. Available online: https://man7.org/linux/man-pages/man8/ip-link.8.html (accessed on 26 June 2026).
20. Microsoft Learn. ARP Command Usage in Windows. Available online: https://learn.microsoft.com/windows-server/administration/windows-commands/arp (accessed on 26 June 2025).
21. Linux man-pages project. ip-neighbour—Linux Manual Page. Available online: https://man7.org/linux/man-pages/man8/ip-neighbour.8.html (accessed on 26 June 2025).
22. Cisco Systems. Cisco Plug-In for OpenFlow Configuration Guide 1.3. Available online: https://www.cisco.com/c/en/us/td/docs-/switches/datacenter/nexus/openflow/b_openflow_agent_nxos_1_3/Cisco_Plug_in_for_OpenFlow.html (accessed on 28 May 2026).
23. Noormohammadpour, M.; Raghavendra, C.S. Datacenter Traffic Control: Understanding Techniques and Trade-offs. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 1492–1525. https://doi.org/10.1109/COMST.2017.2782753.
24. Schulzrinne, H.; Casner, S.L.; Frederick, R.; Jacobson, V. RTP: A Transport Protocol for Real-Time Applications. RFC 3550. 2003. Available online: https://www.rfc-editor.org/info/rfc3550 (accessed on 21 July 2025).
25. Open vSwitch Project. OVS Faucet Tutorial. 2023. Available online: https://docs.openvswitch.org/en/latest/tutorials/faucet (accessed on 18 January 2026).
26. Red Hat. Ansible Main Page. Available online: https://www.redhat.com/en/ansible-collaborative (accessed on 1 June 2026).
27. Hewlett Packard Enterprise. ArubaOS-Switch OpenFlow Administrator Guide for AOS-S 16.11. 2023. Available online: https://arubanetworking.hpe.com/techdocs/AOS-Switch/16.11/Aruba%20OpenFlow%20Administrator%20Guide%20for%20AOS-S%2016.11.pdf (accessed on 17 December 2024).

28. Meghana, J.S.; Subashri, T.; Vimal, K. A survey on ARP cache poisoning and techniques for detection and mitigation. In Proceedings of the Fourth International Conference on Signal Processing, Communication and Networking (ICSCN), Chennai, India, 2017; pp. 1–6. https://doi.org/10.1109/ICSCN.2017.8085417.
29. Shah, Z.; Cosgrove, S. Mitigating ARP Cache Poisoning Attack in Software-Defined Networking (SDN): A Survey. *Electronics* **2019**, *8*, 1095. https://doi.org/10.3390/electronics8101095.
30. Toral-Cruz, H.; Pathan, A.S.K.; Ramírez Pacheco, J.C. Accurate modeling of VoIP traffic QoS parameters in current and future networks with multifractal and Markov models. *Math. Comput. Model.* **2013**, *57*, 2832–2845. https://doi.org/10.1016/j.mcm.2011.12.007.
31. Hardin, B.; Comer, D.; Rastegarnia, A. On the Unreliability of Network Simulation Results from Mininet and iPerf. *Int. J. Future Comput. Commun.* **2023**, *12*, 5–13. https://doi.org/10.18178/ijfcc.2023.12.1.596.