electronics

MDPI

Review

Mitigating ARP Cache Poisoning Attack in Software-Defined Networking (SDN): A Survey

Zawar Shah [1,2,*] and Steve Cosgrove [3]

[1] School of Information Technology, Whitireia Community Polytechnic, Auckland 1010, New Zealand
[2] Department of Business Information Systems, Australian Institute of Higher Education, Sydney NSW 2000, Australia
[3] School of Information Technology, Whitireia Community Polytechnic, Wellington 5022, New Zealand; steve.cosgrove@whitireia.ac.nz
* Correspondence: zawar.shah@whitireia.ac.nz

Received: 16 August 2019; Accepted: 26 September 2019; Published: 28 September 2019

Abstract: Address Resolution Protocol (ARP) is a widely used protocol that provides a mapping of Internet Protocol (IP) addresses to Media Access Control (MAC) addresses in local area networks. This protocol suffers from many spoofing attacks because of its stateless nature and lack of authentication. One such spoofing attack is the ARP Cache Poisoning attack, in which attackers poison the cache of hosts on the network by sending spoofed ARP requests and replies. Detection and mitigation of ARP Cache Poisoning attack is important as this attack can be used by attackers to further launch Denial of Service (DoS) and Man-In-The Middle (MITM) attacks. As with traditional networks, an ARP Cache Poisoning attack is also a serious concern in Software Defined Networking (SDN) and consequently, many solutions are proposed in the literature to mitigate this attack. In this paper, a detailed survey on various solutions to mitigate ARP Cache Poisoning attack in SDN is carried out. In this survey, various solutions are classified into three categories: Flow Graph based solutions; Traffic Patterns based solutions; IP-MAC Address Bindings based solutions. All these solutions are critically evaluated in terms of their working principles, advantages and shortcomings. Another important feature of this survey is to compare various solutions with respect to different performance metrics, e.g., attack detection time, ARP response time, calculation of delay at the Controller etc. In addition, future research directions are also presented in this survey that can be explored by other researchers to propose better solutions to mitigate the ARP Cache Poisoning attack in SDN.

Keywords: ARP cache poisoning attack; Software-Defined Networking (SDN); Denial of Service (DoS) attack; Man in the Middle (MITM) attack

1. Introduction

Software-Defined Networking (SDN) is a new approach to computer networks. Traditional networking relies on control decisions being made in each network device, based on information that uses the same paths (or 'plane') as network data. SDN separates the data and control planes. A central controller manages the networking, using the control plane to send commands to network devices. The separation of the control plane and the data plane is realized by means of a well-defined programming interface between the switches and the controller. The separation of the forwarding hardware from the control logic in SDN allows easier deployment of new protocols and applications along with straightforward network visualization and management [1–3]. Nowadays, a major concern to network administrators and engineers is the security of the network. Traditional Networks suffer from many security issues and attacks. Although SDN has solved some of the security vulnerabilities

and attacks present in traditional networks, those networks are still vulnerable to many security attacks. One such security attack that needs to be mitigated is the Address Resolution Protocol (ARP) Cache Poisoning attack [4–6].

ARP is a widely used protocol for resolving Internet Protocol (IP) addresses into Media Access Control (MAC) addresses. However, ARP suffers from many vulnerabilities, e.g., ARP has no built-in mechanism by which a receiving node can authenticate the sender from which the packet originated. Moreover, ARP is a stateless protocol, i.e., ARP replies can be sent by nodes without a corresponding ARP request. These vulnerabilities in ARP are exploited by attackers to carry out the ARP Cache Poisoning attack. In traditional networks, this attack is carried out by poisoning the cache of a host by inserting false or spoofed IP to MAC address mappings in victim's ARP cache table. However, in SDN, this attack is carried out either by poisoning the ARP cache table maintained by the Controller or by poisoning ARP cache table of the hosts in the network (as with traditional networks). It is important to mitigate ARP Cache Poisoning attack in both traditional networks and SDN as it can be used by the attacker to further launch other attacks, i.e., Denial of Service (DoS), Man-In-The-Middle (MITM) and Host Impersonation attacks [7–9].

Some solutions to mitigate an ARP Cache Poisoning attack in traditional networks are proposed in the current literature. Dynamic ARP Inspection (DAI) is one such solution that is widely used in the traditional networks. However, network administrators have to enable DAI on every switch, which is cumbersome. Moreover, proprietary nature and high cost are another two issues associated with DAI [10]. Another solution discussed in the literature is Secure ARP (S-ARP). S-ARP uses public key cryptography and relies on each node to maintain a public/private key pair. However, key management is a serious issue in S-ARP. Moreover, it only protects from spoofing of ARP reply messages and is vulnerable to spoofing of ARP requests [11]. Static configuration of ARP mappings is another solution to mitigate this attack. This solution provides no chance to the attacker to spoof the other host's ARP cache. However, this solution is error prone and not scalable [12]. Recently, solutions to mitigate the ARP Cache Poisoning attack in SDN have also been discussed in the literature [4–6,9,10,13].

In this work, a detailed survey is carried out of various solutions proposed in the current literature to mitigate the ARP Cache Poisoning attack in SDN. We classify the various SDN based solutions into three broad categories, i.e., Flow Graph based solutions, Analysis of Traffic Patterns based solutions and Maintaining IP-MAC Address Bindings based solutions. The principles and working details of all these three types of solutions are explained in this research. Moreover, all solutions are critically evaluated in terms of their strengths and weaknesses. All SDN-based solutions are also compared and evaluated for various performance metrics, e.g., delay at the Controller, attack detection time, scalability etc. Future research directions that outline promising investigation areas are discussed. We note that a few surveys [7,14–16] have been carried out in the existing literature regarding the ARP Cache Poisoning attack. However, surveys carried out in [7,14–16] are focused on non-SDN based solutions for ARP Cache Poisoning attack and do not discuss SDN based solutions. In [17], the authors have carried out a survey of SDN based solutions for ARP Cache Poisoning attack; however, this study is limited to explanation of few solutions with no critical evaluation. Moreover, classification of solutions into various categories and future research directions are also missing from [17]. To the best of our knowledge, there are limited studies available in the existing literature that classify and critically evaluate various SDN based solutions to the ARP Cache Poisoning attack.

The main contributions of this work are: (1) to categorize various solutions proposed in the existing literature to mitigate the ARP Cache Poisoning attack in SDN. (2) To discuss working principles and details of various solutions proposed in the existing literature to mitigate the ARP Cache Poisoning attack in SDN. (3) To critically evaluate various categories of solutions proposed in the current literature to mitigate the ARP Cache Poisoning attack in SDN. (4) To compare different solutions to mitigate ARP Cache Poisoning attack in SDN based on various performance metrics. (5) To propose future research directions that can be explored as a source of motivation towards development and deployment of new SDN based solutions for the ARP Cache Poisoning attack.

The rest of this survey is organized as follows. In Section 2, articles that have carried out similar surveys on ARP Cache Poisoning attack are discussed. Section 3 outlines key aspects of ARP protocol handling and ARP Cache Poisoning. First in traditional networks, then in SDN networks. Types of ARP Cache Poisoning attacks in SDN are explained further in Section 3. In Section 4, different types of solutions are discussed and critically evaluated. Future research directions and comparison of various solutions are discussed in Section 5. Finally, conclusion of the survey is provided in Section 6.

2. Related Work

Some studies [7,14–20] in the current literature have carried out a survey to classify and analyze various solutions to mitigate the ARP Cache Poisoning Attack in traditional networks. In [7], authors carry out a survey on MITM attacks in traditional networks. Various MITM attacks are divided into four categories by authors in [7]. These are: Spoofing-based MITM; Secure Sockets Layer (SSL) MITM; Border Gateway Protocol (BGP) MITM; and False Base Station-based MITM. ARP Cache Poisoning attack is discussed as part of spoofing-based MITM attacks. Moreover, different solutions to mitigate ARP Cache Poisoning attacks are classified into various categories in [7], i.e., cryptographic solutions, hardware solutions, server-based solutions, host-based solutions. All these solutions are briefly explained by authors in [7]; however, critical evaluation of various solutions is limited and future research directions are not proposed. A survey on various detection and mitigation solutions to the ARP Cache Poisoning attack is also presented in [14]. A key feature of research presented in [14] is the proposal of three future research directions to detect and mitigate this attack. These future research directions include minimal usage of cryptographic techniques by solutions so that the ARP process is not slowed down; new solutions should be practical with minimal changes in the way traditional ARP works and more preventive techniques should be used instead of just detection, as detection increases the workload of network administrators because of the generation of many alarms. The authors in [15–18] also discussed many MITM attacks including the ARP Cache Poisoning Attack. A critical evaluation of few solutions to mitigate ARP Cache Poisoning attack is presented in [15–18], but the solutions are not divided into various categories. In [19], the authors briefly explain and critically evaluate six different schemes to mitigate ARP Cache Poisoning Attack in wireless networks. Some of these schemes include DAI, Static ARP entries, SSL certificate validator etc. However, the research conducted in [19] explains only six schemes with inadequate critical evaluation. Another survey conducted in [20] discusses four solutions to mitigate ARP Cache Poisoning attack in traditional networks. These four solutions include cryptographic solutions, hardware solutions, server based solutions and host based solutions. Future research directions are not proposed in [20].

Surveys conducted in [7,14–20] are focused on traditional networks and do not take into account solutions to mitigate ARP Cache Poisoning attack in SDN. To the best of our knowledge, few studies exist in the existing literature that have carried out a detailed survey of various solutions that have been proposed in the last few years to mitigate the ARP Cache Poisoning attack in SDN. Only work conducted in [17] has briefly explained working principles of only six solutions to mitigate this attack in SDN and fails to take into account many other solutions available in the existing literature. Moreover, the survey conducted in [17] does not divide the solutions into different categories and also no future research directions are presented. This study aims to fill in this research gap by categorizing and critically evaluating various solutions to mitigate ARP Cache Poisoning attack in SDN. Another novelty of this work is to compare different solutions in terms of various performance metrics and propose future research directions.

3. Background

The ARP protocol is a key part of the TCP/IP protocol suite. It is intended to be simple and fast [7]. This section reviews how ARP traditionally works, then outlines variations being used to accommodate modern threats, and changing network environments.

3.1. ARP Handling in Traditional Networks

The traditional ARP protocol is simple and very lightweight [7]. A host needs to send a packet to a particular IP address. Network information held by the sending host verifies that the destination host is on the same network. In order to send the packet, the MAC address is needed. The host checks its ARP cache table and an ARP request packet is broadcast if the corresponding MAC address is not present in the ARP cache table. The network switch forwards that request to all hosts on a particular network.

The destination host recognizes the IP address in the request, and sends a unicast packet to the host, thereby advising the host of its MAC address [8,9]. ARP handling in traditional networks is shown in Figure 1.

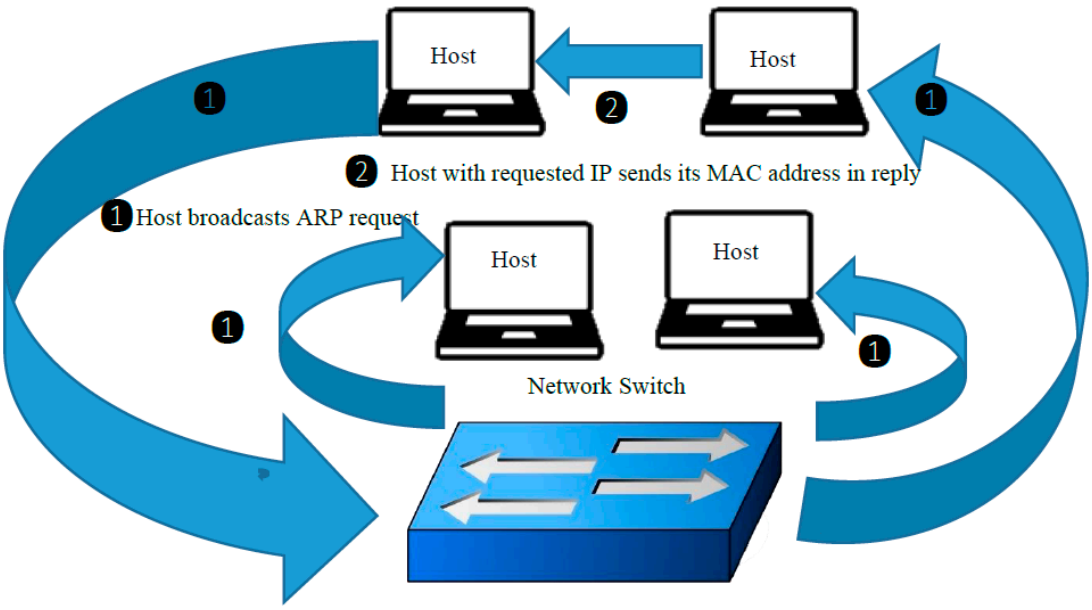

Figure 1. ARP (Address Resolution Protocol) in Traditional Networks.

ARP Cache Poisoning Attack in Traditional Networks

ARP protocol is stateless and has no built-in authentication mechanism. It can be easily exploited by attackers to poison the ARP cache table of hosts present in the network by sending spoofed ARP messages onto the Local Area Network (LAN). Attackers can reply to selected, or all, ARP requests with their own MAC address, regardless of the IP address requested. In this way, attackers can poison the ARP cache table maintained by hosts who then send data to attackers instead of other hosts. Attackers can also use fake gratuitous ARP packets to poison the ARP cache table maintained by hosts on a LAN. Attackers on receiving data packets from hosts can carry out further attacks by modifying data pockets (MITM attack) or by dropping them (DoS attack) [7–9].

3.2. ARP Handling in SDN

There are two ways of handling ARP in SDN. These two ways are called as Proxy ARP and Regular ARP [4,5,21]. We will explain the working of both these types in this section.

3.2.1. Regular ARP

Regular ARP in SDN (shown in Figure 2) works in a similar manner as in traditional networks (explained in Section 3.1). The host sends a broadcast ARP request on the network and the corresponding host replies with its MAC address via unicast ARP reply. In regular ARP, switches are instructed to forward all ARP packets that do not have matching flow entries to the Controller as an OpenFlow Packet-in message. The Controller runs a learning switch application that manages those packets that

did not match any flow on the switch. Controller then forwards received ARP packets as a Packet-out OpenFlow message to the appropriate switch so that it is delivered to the right host. Controller only has a forwarding functionality in regular ARP. However, in regular ARP, the learning switch application on the Controller installs two flows on the switches, i.e., one flow related to source MAC and second is related to destination MAC. This reduces the traffic from switches to the Controller [4,5,21]. Regular ARP handling is supported by a few Controllers, e.g., Faucet [22].

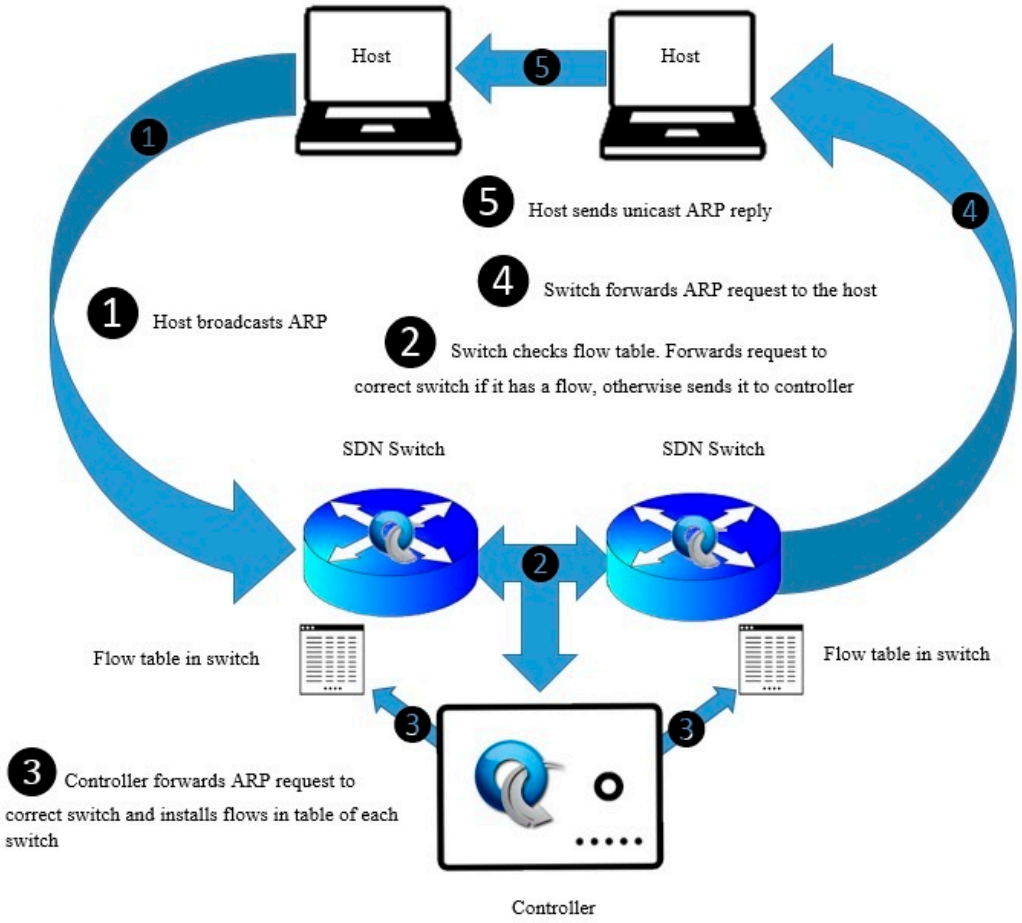

Figure 2. Regular ARP in SDN (Software Defined Networking).

3.2.2. Proxy ARP

Proxy ARP takes advantage of the centralized architecture of SDN and lets the Controller reply to all the ARP queries in the network. The switches on receiving ARP packets forward them to the Controller. The Controller builds and maintains a database of IP-MAC address bindings and on receiving ARP request sends an ARP reply as an OpenFlow Packet-Out message, which is ultimately delivered by the switch to the host attached to the corresponding port [4,5]. Proxy ARP technique is widely used by many SDN Controllers, e.g., Python-based OpenFlow Controller (POX), RYU and Open Network Operating System (ONOS) [21]. Proxy ARP in SDN is shown in Figure 3.

3.2.3. ARP Cache Poisoning in SDN

ARP Cache poisoning attack in SDN can be carried out against both Proxy ARP and Regular ARP [4,5,21]. In this subsection, we will discuss how attacker can carry out the attack in both cases.

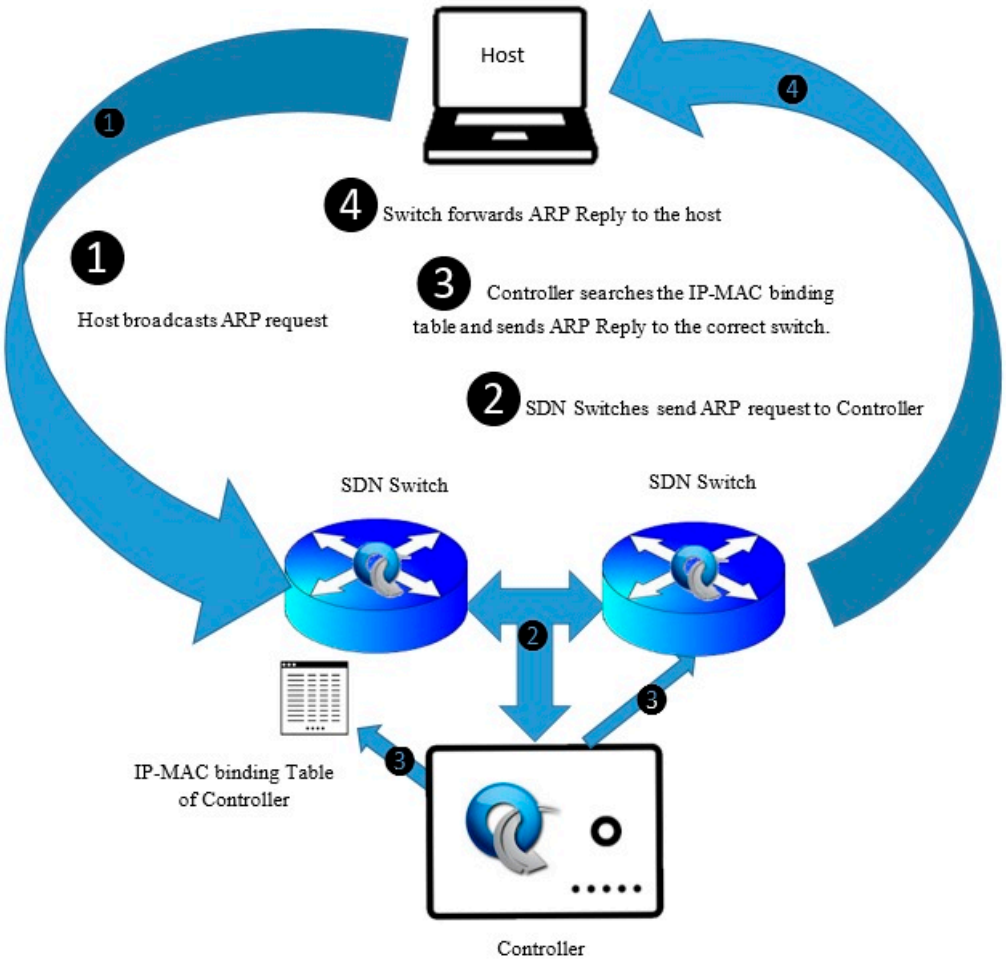

Figure 3. Proxy ARP in SDN.

ARP Cache Poisoning in Proxy ARP

In this attack, the aim of the attacker is to poison the ARP cache of the Controller. This can be done by sending fake ARP request or reply packets to the Controller. These fake ARP request or reply packets can poison the ARP cache table of the Controller. Moreover, as ARP is a stateless protocol (as mentioned in the Section 3.1), this attack in SDN can also be carried out by just sending fake gratuitous ARP packets to the Controller. Controller is the brain of the network and poisoning its ARP cache can have dangerous consequences for the entire network, e.g., this attack can result in MITM attack which can enable the attacker to extract sensitive information from the traffic. This can also result in a DoS attack in which the real recipient will never receive the ARP message and consequently will not be able to communicate on the network. It is to be noted that there is no mechanism in the Controller to protect against this attack; neither the switch learning application, nor OpenFlow provides protection against this attack [4,5,21].

ARP Cache Poisoning in Regular ARP

ARP cache poisoning in regular ARP can also lead to many attacks including MITM and DoS. In this attack, the aim of the attacker is to poison the cache of the hosts in the network. This attack is carried out in the same way as in traditional networks in which an attacker sends fake ARP request packets to corrupt the ARP cache of the hosts in the network. Fake ARP replies and gratuitous ARP packets can also be sent to carry out this attack in the network. It is to be noted that in regular ARP, the controller just acts as a forwarder and does not check or verify the IP or MAC addresses of sending hosts. This makes it easy for the attacker to carry out the attack [4,5,21].

4. Solutions to Mitigate ARP Cache Poisoning Attack in SDN

In this research, our focus is on the articles published in the last few years that have provided a solution to mitigate ARP Cache Poisoning attack in SDN. Studies that have not explicitly addressed this attack in SDN are not considered in this research. Many researchers have also proposed solutions to mitigate ARP flooding attack in SDN; however, flooding attack is not the focus of this work. There are four important features of our work. Firstly, we classify the solutions available in the current literature into various categories. Secondly, working details and principles of various solutions to mitigate ARP Cache Poisoning attack in SDN are provided. Thirdly, we evaluate the solutions to determine whether they provide protection against both types of ARP Cache Poisoning attacks in SDN (i.e., Proxy and Regular) carried out either by spoofed ARP requests or replies. Fourthly, we critically evaluate the available solutions to highlight the advantages and issues in them.

We classify the various solutions discussed in the literature into three broad categories, which are Traffic Patterns Based Solutions, IP-MAC Address Bindings Based Solutions and Flow Graph Based Solutions. IP-MAC address Bindings Based Solutions are further divided into two categories depending on whether the IP-MAC address bindings are maintained on the Controller or on some other network entity. This classification is shown in Figure 4.

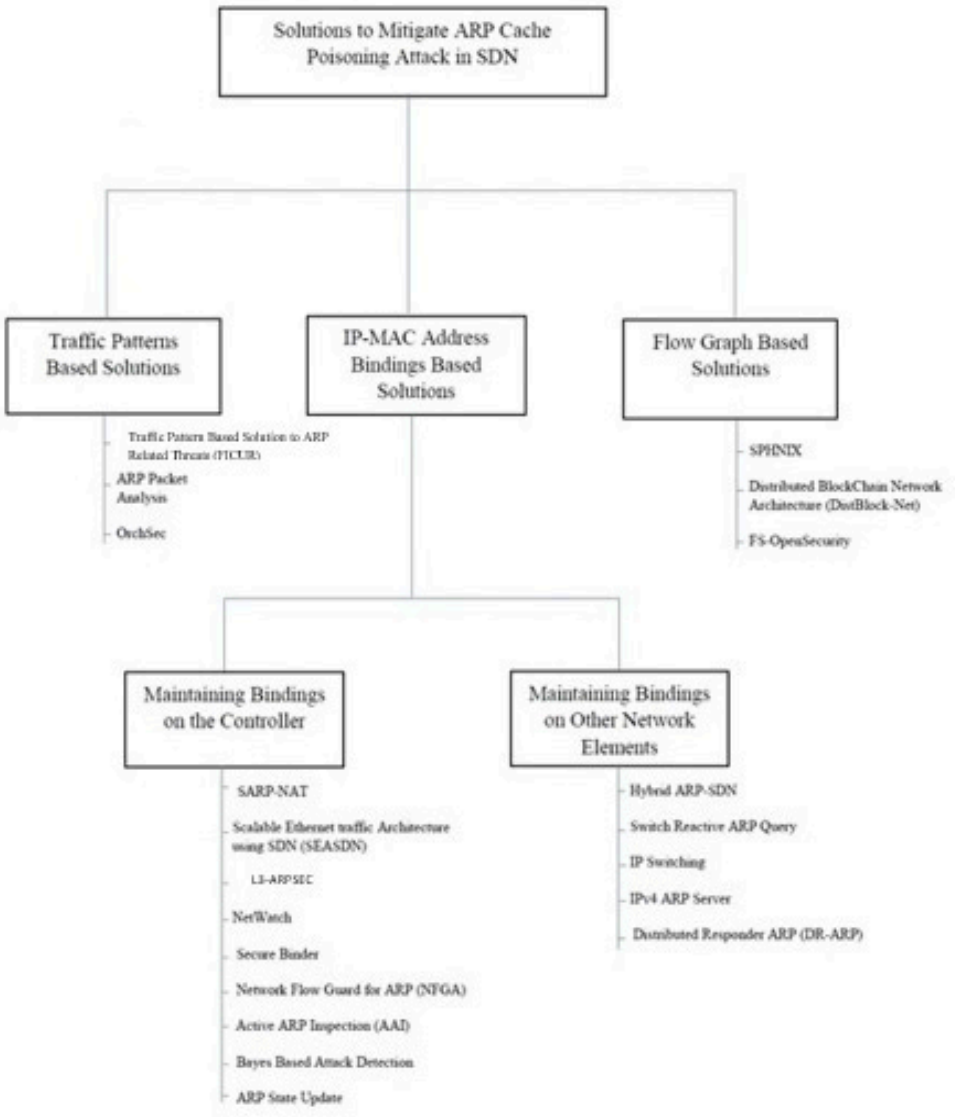

Figure 4. Classification of Solutions for ARP Cache Poisoning Attack in SDN.

4.1. Solutions Based on Analysis of Traffic Patterns

The key idea of these solutions is to use the Controller to monitor and analyze ARP traffic passing through the network. The Controller monitors the ARP traffic and detects the ARP Cache Poisoning attack if analysis of ARP traffic meets certain pre-defined traffic patterns. These pre-defined traffic patterns include, e.g., ARP request packets with source MAC address in Ethernet frame different than the source MAC address of ARP header etc. These solutions are discussed below.

4.1.1. Traffic Pattern based Solution to ARP Related Threats (FICUR)

FICUR is proposed in [12], in which authors explain that analysis of traffic pattern can be used to detect an ARP poisoning attack. For example, if traffic analysis shows ARP request packets with different IP addresses and the same MAC address, then it shows an ARP Cache Poisoning attack. Similarly, authors are of the view that if many ARP request packets are received at the same port of the switch then it also shows an ARP Cache Poisoning attack. FICUR is implemented as an application on the POX Controller. It monitors and analyses all ARP requests packets in the network and once any anomaly in traffic pattern is detected then that port is blocked to filter ARP packets. FICUR can handle attacks against Proxy ARP and Regular ARP attacks. This is because analysis of traffic patterns by the Controller not only prevents the attacker from poisoning the cache of Controller but also enables the Controller to drop suspicious packets (that match a certain traffic pattern) to reach other hosts in the network. Simulations and real experiments carried out in [12] using POX Controller with three hosts show that FICUR can detect ARP cache poisoning attack in 16.056 msec. However, only three hosts are considered by the authors in their experimental topology, therefore, the results obtained may not hold true for a large network. Another important metric of processing delay at the Controller caused by analyzing ARP packets is also not considered by the authors. Moreover, authors only considered fake ARP requests and fail to take into account spoofed ARP replies, which can also lead to an ARP Cache Poisoning attack.

4.1.2. ARP-Packet Analysis

Another study conducted by authors in [23] proposes a solution called as ARP-Packet Analysis, which also takes into account packet analysis to detect and mitigate an ARP Cache Poisoning attack. In [23], the Controller runs the Dynamic Host Configuration Protocol (DHCP) server and keeps track of IP addresses that are leased to various hosts. ARP packets received on the ports of edge switches are passed to the Controller for processing. This solution is also implemented as an application on the POX Controller. This application handles ARP requests and replies that are received at the Controller and considers them spoofed based on a few conditions, e.g., ARP request packet is spoofed if the source MAC address in Ethernet frame and source MAC address of ARP header are not the same. Similarly, an ARP reply packet is spoofed if destination MAC address in Ethernet frame and destination MAC address of ARP header are not same or destination MAC address of the Ethernet header is FF:FF:FF:FF:FF:FF. On finding spoofed ARP packets, Controller directs the switch to stop the malicious traffic. Like FICUR [12], this solution can also handle attack against both Proxy ARP and Regular ARP. This solution can mitigate the ARP Cache Poisoning attack using gratuitous ARP as it flags Ethernet packets that have a destination MAC address of FF:FF:FF:FF:FF:FF (present in the gratuitous ARP packet). This solution does not require any major modifications in the SDN infrastructure. However, topology considered by authors in [23] is very simple with only three hosts and single Controller. Moreover, time required to complete ARP requests is not considered by the authors.

4.1.3. OrchSec

Another solution based on analysis of traffic patterns is called as OrchSec and is proposed in [24]. OrchSec is an architecture to enhance network security and according to authors it can be used to tackle different attacks in SDN including the ARP Cache Poisoning attack. The most important component of

this architecture is Orchestrator, which is the brain of the network and runs on top of the Controller. Orchestrator commands the Controller to observe certain traffic patterns and then forwards the traffic related to these patterns to it for further inspection. An application called the Orchestrator Agent is installed in Controller that enables the Controller to communicate with the Orchestrator. For detecting an ARP Cache Poisoning attack, Controller sends all ARP traffic to the Orchestrator, which forwards it to a detection logic module for examination. The detection module analyzes the ARP traffic to find any unusual patterns in traffic. On detection of the attack, the Orchestrator sends a request to the Controller to install flows on the switches to block the traffic that is being sent by the attacker. An advantage of this architecture is that it can be deployed on many SDN Controllers. Authors in [24] provided no experimental or simulation results related to successful detection and mitigation of ARP Cache Poisoning attack in SDN. This architecture, however, can theoretically protect against both Proxy ARP and Regular ARP attacks. Many parts of this architecture need more explanation, e.g., how it handles spoofed ARP requests and replies. Additionally, authors should explain the detailed working of the logic module. Moreover, in OrchSec, the Orchestrator is the brain of the network rather than the Controller. This is against the standard SDN architecture and will make its deployment difficult in the real world.

4.1.4. Issues with Solutions Based on Traffic Patterns

The key benefit of these solutions is that they are easy to implement. However, scalability is a concern in all these solutions. This is because in all these solutions the Controller has to process all the ARP packets in real time and then identify any anomaly in traffic patterns. This is suitable for small number of hosts but in the presence of many hosts in the network this requires extra processing on the Controller which may ultimately affect its performance. Moreover, the network administrator has to identify various pre-defined traffic patterns accurately and carefully; otherwise, false alarms may be generated by these solutions. The key findings related to these solutions are summarized in Table 1.

Table 1. Summary of Traffic Patterns Based Solutions.

Solution	Advantage	Issue	Proxy ARP Attack Prevention	Regular ARP Attack Prevention	Spoofed ARP Requests Handling	Spoofed ARP Replies Handling
FICUR [12]	Quickly detects the attack	Controller CPU utilization is not considered	Yes (only for spoofed ARP Requests)	Yes (only for spoofed ARP Requests)	Yes	No
ARP-Packet Analysis [23]	Low CPU utilization at the Controller	Tested on a small network	Yes	Yes	Yes	Yes
OrchSec [24]	Can be deployed in a Multi-Controller Architecture	Attack detection time is not calculated	Yes	Yes	Not explained	Not explained

4.2. Solutions Based on Flow Graphs

Other types of solutions proposed in the literature to mitigate ARP Cache Poisoning attack are called Flow Graph based solutions. A flow is defined as a traffic pattern that is observed over specified ports between two hosts that have different MAC addresses. A flow graph is graph theory representation of the switches and the flow metadata. The switches are nodes and the edges are the flow metadata in the graph. The solutions that belong to this category build an incremental flow graph to approximate the operation of the network to detect attacks. Validation (in real time) of all updates in the network and restrictions on every flow graph in the network is possible with the construction and maintenance of these flow graphs. Flow graphs provide a simple and easy mechanism that helps in detection of violations of various restrictions for network topology and data plane forwarding in SDNs [4,6]. All solutions that belong to his category perform three steps: (1) they intercept OpenFlow

packets in the network; (2) they build an incremental flow graph from the information captured from these packets; (3) they validate the metadata of any flow against allowed values of metadata captured over time for that particular flow or with policies allowed by the administrators [4,6]. The working principles and critical analysis of all solutions that belong to this category are given below.

4.2.1. SPHINX

SPHINX is proposed in [4] and it builds incremental flow graphs to detect an ARP cache poisoning attack. Creation of flow graph enables SPHNIX to maintain and continuously update comprehensive information about the network. This information is gathered from various OpenFlow messages, i.e. Packet_in, Stats_reply, Features_reply and Flow_mod. This information includes various bindings (e.g., IP-MAC, MAC-Port and Switch-Port), flow statistics (e.g., bytes transferred), paths in the network used by each flow etc. SPHINX validates all flow graphs against two constraints: (1) administrator defined security policies (to defend against known attacks); (2) those learnt over time for a specific flow (to defend against unexpected and unsafe network updates). SPHNIX provides protection against Proxy ARP attack and Regular ARP attack as fake ARP requests are detected by looking into the various bindings and flow metadata maintained by it. SPHINX is implemented as an application (written in Java) that resides between the Controller and the switches. In [4], authors carried out the performance analysis of SPHNIX in their experimental network that consisted of 10 servers connected to 14 switches. Results show that SPHINX can quickly detect the ARP cache poisoning attack carried out by spoofed ARP requests (attack detection time of 44 µsec). However, results also reveal that attack detection time is dependent on length of SPHNIX ingress queue and processing time of Packet_in messages. The authors, however, do not show clearly how an ARP Cache Poisoning attack carried out by spoofed ARP replies is successfully prevented. Another advantage of SPHNIX is that it can easily be deployed over four different Controllers (i.e., OpenDaylight (ODL), Floodlight, POX and Maestro) with slight modification in the code. A major limitation of SPHINX is that it may not be able to detect attacks that last for a short time. This is because it takes time (up to few seconds) to collect and update various metadata statistics related to various flow graphs.

4.2.2. Distributed BlockChain Network Architecture (DistBlockNet)

In [25] authors propose a distributed secure SDN architecture using BlockChain technology to prevent security attacks on IoT devices. The architecture takes advantage of programmability feature of SDN and also utilizes the inherent advantage of BlockChain that helps to create a distributed peer-to-peer architecture with no central Controller. The architecture is scalable as BlockChain technology enables many IoT devices to interact in a verifiable manner with each other without the need of a trusted intermediate entity. In this distributed BlockChain architecture, Controllers maintain the updated flow table rules which can be obtained by the IoT devices by sending request to the Controllers. The main module of DistBlockNet related to providing security is the Shelter which creates a graph network with traffic flow by analyzing and parsing OpenFlow packets. It maintains a metadata feature set and network topological state that is obtained from OpenFlow packet headers, measurements of traffic flow etc. Shelter, just like SPHINX [4], compares this metadata related to each flow with the set of acceptable metadata values defined by the administrative rules or with the flow activities it has maintained over time. It generates an alarm when it detects an untrusted device that tries to modify the existing flow behavior or when the existing flow tries to resist the rules specified by the administrator. To gauge the effectiveness of DistBlockNet, authors in [25] carried out simulations by using POX Controller and an OpenFlow Switch. They carried out an ARP Cache Poisoning attack throughout the distributed BlockChain network. The results show that DistBlockNet can detect and block the ARP Cache Poisoning attack as soon as the Packet-in messages were parsed and processed. Similar to SPHINX, DistBlockNet can provide protection against Proxy ARP and Regular ARP attacks. However, precise attack detection time was not calculated by the authors in [25]. Similar to SPHNIX, it is not clearly mentioned in [25] how DistBlockNet handles spoofed ARP replies

that can also cause ARP Cache Poisoning attack. Moreover, the authors did not mention what type of BlockChain (i.e., Public or Private) they have used in their research. Private BlockChain has scalability issues [26] and pubic BlockChain is vulnerable to Sybil attack in which hackers attempt to control the peer network by using fake identities [27,28]. Studies suggest that the Sybil nodes can carry out many attacks including the ARP Cache Poisoning attack [27,29]. Additionally, this architecture is proposed specially for IoT based networks and cannot be readily used in the traditional networks.

4.2.3. FS-OpenSecurity

In [6], authors proposed a new security architecture called as FS-OpenSecurity. This architecture also makes use of Flow Graphs to detect and block the ARP Cache Poisoning attack. The most important component of this architecture is SQUEAK which like the other two solutions monitors the OpenFlow messages to create an incremental flow graph. All traffic belonging to new flows is marked as suspicious by switches and is first sent to SQUEAK for processing. No direct communication between switches and Controller is allowed. When SQUEAK receives new packets, it creates an incremental flow graph and validates the information against administrator defined security policies and with those learnt over time for a specific flow. If the validation results in indication of an attack it then directs the Controller to take steps to block the attack. However, authors do not clearly explain how spoofed ARP requests and replies are handled in this solution. Moreover, FS-OpenSecurity is just a theoretical architecture as no details of its implementation are given by the authors. Simulation and experimental results are needed to further gauge the effectiveness of this architecture.

4.2.4. Issues with Solutions Based on Flow Graphs

The solutions in this category have a broader scope and can be used to detect a variety of attacks in SDN including the ARP Cache Poisoning attack. Hence, how to mitigate ARP Cache Poisoning attack using these solutions was mentioned briefly by authors in [4,6,25]. More specific experimental and simulation results related to mitigating ARP Cache Poisoning attack are needed to find the effectiveness of these solutions. However, theoretically, these solutions can provide protection against both Proxy ARP and Regular ARP attacks. Moreover, these solutions suffer from two critical issues. Firstly, these solutions assume that the OpenFlow messages sent by the SDN switches are trustworthy. However, a compromised switch may add false information related to various flows and distort the flow graphs. This will affect the functionality of these solutions in terms of detecting ARP Cache Poisoning attack. Secondly, if the topology is dynamic and changing rapidly then various properties of flows (i.e., various bindings, path taken, bytes transferred etc.) will change rapidly. This may result in generation of false alarms by all these solutions. Important features of these solutions are summarized in Table 2.

Table 2. Summary of Flow Graph Based Solutions.

Solution	Advantage	Issue	Proxy ARP Attack Prevention	Regular ARP Attack Prevention	Spoofed ARP Requests Handling	Spoofed ARP Replies Handling
SPHINX [4]	Can be deployed over four different Controllers	Attack detection time is dependent on processing time of Packet_In messages	Yes	Yes	Yes	Not explained
DistBlockNet [25]	Scalable Architecture	Attack detection time is not calculated	Yes	Yes	Yes	Not explained
FS-OpenSecurity [6]	Quickly detects the attack	No simulation or experimental results are provided	Yes	Yes	Not explained	Not explained

4.3. Solutions Based on IP-MAC Address Binding

Previous sections have considered analysis of Traffic Patterns and use of Flow Graphs as solutions to ARP Cache Poisoning attacks in SDN. Other SDN-based solutions discussed in the literature take advantage of the fact that maintaining IP-MAC address bindings can help to detect the ARP cache poisoning attack in SDN. These solutions can be further divided into two types. The first type maintains IP-MAC address bindings on the Controller and the second type maintains IP-MAC address bindings on other network elements such as the server or SDN switch. We discuss the working principles and critically evaluate both types of solutions in this section.

4.3.1. IP-MAC address Bindings on the Controller

The solutions in this category maintain IP-MAC address bindings on the Controller and are run as a separate application or a module on the Controller. Our survey shows that solutions in this category are more in number compared to solutions proposed for other categories. These solutions are discussed below.

SARP_NAT

SARP-NAT is proposed in [21]. It makes ARP stateful by maintaining a list of hosts that have sent an ARP request message. When a host h1 sends an ARP request, it is first sent to the Controller where SARP-NAT component is running. This component stores each ARP request in a list of requests that are still pending. It stores the original MAC address and IP address of the sending host and then replaces them with dummy (safe) values. The ARP request message is then sent to the target host h2. The host h2 replies with its MAC address and sends the ARP reply to the SARP-NAT component. The SARP_NAT checks to see whether the reply is to a genuine ARP message sent earlier by looking into the pending list. If an entry is not present in the pending list, then it is a gratuitous ARP, which is not reliable, and is therefore dropped by the Controller. However, on finding an entry in the list, SARP-NAT component sends an ARP reply message to the host h1 in which it replaces the dummy values with the original values that were stored by it earlier. In this way, SARP-NAT stops the attacker from poisoning the ARP cache of hosts by sending spoofed gratuitous ARPs. The above explanation is to protect against Regular ARP; however, this mechanism can also be used to mitigate an attack against Proxy ARP. In Proxy ARP, the Controller has to be protected from spoofing attacks. For this purpose, SARP-NAT module running at the Controller adds IP-MAC bindings in the ARP table only if ARP reply is received in response to a genuine ARP request. Experiments conducted in [21] showed that SARP_NAT has less CPU utilization of the Controller than regular ARP. It also takes 32% less time to handle ARP requests compared to time taken by regular ARP. However, authors in [21] only used 64 hosts in their experiments, which is not a large number. In large networks, maintaining states of hundreds of hosts will put an extra burden on the Controller which may result in large delay to handle an ARP request. Additionally, care has to be taken that dummy values are not used by the actual hosts on the network. This is tedious to do and may also result in error if the network is large and scalable.

Scalable Ethernet Traffic Architecture Using SDN (SEASDN)

Another solution called as Scalable Ethernet traffic Architecture using SDN (SEASDN) is proposed in [30]. In this solution, an ARP handling module is present on the Controller. This module enables the Controller in SEASDN to maintain IP and MAC address mappings about the hosts in the network and it stores this information in a table. SEASDN module gets this information from the DHCP packets that are exchanged between hosts and the DHCP server. These DHCP packets are sent to the Controller by the SDN switches. When an ARP request is received, the IP and MAC address pair is checked in the table. If a match is found, then the ARP request is sent to the destination host. However, if no entry is found in the table then that packet is dropped. Additionally, on receiving an ARP reply, the SEASDN ARP module checks if the IP and MAC address pair present in the ARP reply are present in the table.

The packet is dropped if such pair is not present in the table. This is done to prevent the attacker from sending a spoofed MAC address in an ARP reply packet. In [30], simulations were carried out in Mininet which show that SEASDN does not take much CPU usage of the POX Controller even though the number of hosts was varied from 25 to 625. However, more results are needed to judge the effectiveness of this solution, e.g., time taken to handle ARP in SEASDN should be compared with regular ARP.

L3-ARPSEC

L3-ARPSEC [31] is another IP-MAC address binding based solution. This solution also runs a module on the Controller. Similar to SEASDN, this module maintains a table of IP and MAC bindings of all hosts present in the network. However, this table is dynamically created when ARP request messages are received, unlike SEASDN, which creates the table by listening to DHCP messages. When an ARP request packet is received, the ARP reply message containing the MAC address is forwarded to the requesting host, if an entry is found in the table. A packet is broadcasted to all ports of the switch if an entry is not found in the table. In the case of an ARP reply message, Flooding-ARP reply module is invoked. This module calculates the average time interval for last five ARP reply packets. If this average value is less than 3 s, it assumes that an attack is being attempted, as it is unusual to see so many ARP reply packets in short span of 3 s. The switch is then directed by the Controller to block packets from the attacker. Authors in [31] are of the view that attacker can send many ARP reply packets in fraction of seconds and they may be difficult to detect. Therefore, unlike other IP-MAC binding based solutions, L3-ARPSEC scans the IP and MAC bindings table every 13 s to find entries that have different IP addresses but same MAC addresses. This enables L3-ARPSEC to detect and block the attacker. Simulations with three hosts and one switch in Mininet were carried out by the authors in [31]. The results show that L3-ARPSEC can effectively detect and block the attacker. However, only three hosts were used by the authors in their simulations, which is a small number. Additionally, the values of 3 s and 13 s are used by authors with no clear explanation. These values may provide better results in the simulation environment used by the authors. However, in other scenarios (e.g., with greater number of hosts) these values may need to be modified. Therefore, more work is needed to find optimal values in various scenarios.

NetWatch

NetWatch is another IP-MAC address binding based solution proposed in [32]. NetWatch is implemented as an application on the Controller. This application first collects all flow statistics by directing the switches to send all ARP packets to the Controller so that a database of IP-MAC address bindings is maintained by the Controller. Another component of NetWatch is called a Flowstats handler, which monitors the number of ARP packets in the network. If high numbers of ARP packets are detected, this module sends these packets to the Controller to check if the IP and MAC address present in the ARP packets is present in the database of the Controller. The packet is dropped if no entry is present in the database of the Controller. NetWatch then directs the switches to drop all packets from the attacker for a certain fixed duration. Authors in [32] carried out simulations in Mininet to test the performance of NetWatch (which is run as a module on the POX Controller). Results show that for both linear and tree topologies (that are commonly used in data centers), NetWatch provides less delay in detecting the ARP spoofing attack compared to the standard POX Controller with no defense mechanism. However, an issue with NetWatch is that it only detects an attack if the ARP packets are high in number as directed by the Flowstats handler. It may be possible that the attacker only sends few spoofed packets. This solution fails to detect the ARP spoofing attack in such a scenario. Additionally, more results and analysis of this solution is needed to judge its effectiveness, e.g., load on the Controller is not determined by the authors. Moreover, only four hosts were used in the simulations, which is a small number.

Secure Binder

The authors in [33] were focused on enhancing security of various identifier bindings in the network. The authors proposed a new defense mechanism called Secure Binder that prevents attacks against different identifier bindings. This defense mechanism utilizes IP-MAC address bindings along with IEEE 802.1x protocol to establish root-of-trust. Secure Binder is run as a privileged application on the Controller that is responsible to handle all Packet_in messages. It consists of two important modules i.e., mediator and the device authenticator. The role of the mediator is to differentiate between binding traffic (e.g., ARP traffic) and normal data plane traffic and send all traffic to the Controller. At the Controller this binding traffic is checked against the bindings maintained by the Controller. The mediator performs additional verification, e.g., it checks that an old identifier is no longer available before allowing rebinding if an attempt is made to rebind the identifier that is already bounded (e.g., an IP address to a different MAC address). Additionally, the mediator requires the binding requests to originate at the same location as the known identifier. The device authenticator, on the other hand, uses IEEE 802.1x technology that enables a port only if an authorized client is connected to it. Traditionally, IEEE 802.1x uses Remote Authentication Dial-In User Service (RADIUS) to verify the client but it does not check the MAC address. However, the authors in [33] extended IEEE 802.1x to check the MAC address of the device as well before it is allowed to send data in the network. Secure Binder makes use of multiple tables, i.e., table 0 for identifier binding traffic (IP-MAC address bindings) and Table 1 for other data plane traffic. Secure Binder is implemented on the ONOS 1.5.1 Controller. Simulations with three hosts and Open vSwitch were carried out in Mininet. Results show that ARP attack was successfully detected and blocked by Secure Binder as soon as victim's traffic passes through the Controller. It took 41 s in the author's simulation setup to detect this attack which authors explain depends on the workload of the machine of the victim. The results also show that Secure Binder has a high host join latency (42% more than the ONOS Controller with no defense mechanism), which is defined as the time taken by host to join the network including DHCP negotiation, 802.1x authentication etc. Additionally, using modified IEEE 802.1x to stop an attacker from impersonating another device causes an extra overhead on the network. Moreover, Secure Binder requires changes to the way the standard SDN architecture works as it involves the use of modified IEEE 802.1x with SDN. This will make it less practical to be widely deployed in networks.

Network Flow Guard for ARP (NFGA)

NFGA is another IP-MAC address binding based solution, proposed in [34]. NFGA is also run as an application on the Controller. Its main purpose is to monitor the DHCP messages (offers, requests and acknowledgements) in the network and then build a dynamic table of MAC-IP-Port mappings of all hosts in the network. An ARP validator module is part of a NFGA that on receiving an ARP reply searches the table for a match and forwards the packet only if there is a matching entry in the table. If no matching entry is found in the table, then NFGA detects this as an ARP spoofing attack and blocks the port. Simulations conducted in Mininet with POX Controller and six hosts show that NFGA can detect and stop the spoofed ARP traffic at the port. However, the authors discussed only spoofed ARP replies and did not explain how spoofed ARP requests are handled in NFGA. An attacker can send spoofed ARP requests to poison bindings maintained by the NFGA leading to Proxy ARP attack. CPU utilization of the Controller running NFGA is also not determined by the authors in [34].

Active ARP Inspection (AAI)

Authors in [13] proposed AAI to mitigate an ARP Cache Poisoning attack in SDN. This solution is deployed as an application on the Controller. All ARP messages in the network are forwarded to the Controller for processing. AAI application running on the Controller then passes these packets to the demultiplexer, which either forwards packets to the reply processor module or the request processor module. ARP cache module running in the ARP reply processor module maintains binding of IP

address, MAC address, Switch ID and Switch port in a table. ARP reply messages are checked with the entries of this table and are forwarded to the destination if no conflict is found. If conflict is found, the packet is processed by the ARP Inspector. ARP inspector constructs an ARP request packet and sends it to the host corresponding to the suspicious ARP reply message. ARP reply processor than waits for the ARP reply message from the conflicting host. If no ARP reply message is received, the host will be considered as non-existent and the ARP reply processor enters Timeout mode until another ARP reply message is received. The ARP reply processor module enters Judgement mode when all ARP packets from conflicting host are received or timeout. In this mode, ARP reply processor checks to see if any one of the IPs, MACs or switch IDs are conflicting, suggesting the ARP packet is sent by the attacker. It then sends an alarm to the Controller that installs a packet-dropping rule in the corresponding switch. Authors use Cbench (an open source OpenFlow Controller performance testing tool) to test the performance of POX Controller with and without AAI module. The results show that ARP response time with AAI module on the Controller is very low, even with 1000 hosts. However, authors provide no working details of the ARP request processor module. Moreover, sending ARP requests to the conflicting host even when the entry is not present in ARP cache is not properly justified by the authors and seems to be an overhead. Attack detection time is also not computed by the authors in [13].

Bayes-Based Attack Detection

The authors in [5] proposed a solution that maintains IP-MAC bindings on the Controller. However, the authors were of the view that sometimes transmission errors in the network and changes in the configuration of nodes may lead to issues in the IP-MAC bindings on the Controller. To overcome this issue, the authors in [5] proposed an algorithm based on Bayesian theorem that enables the Controller to accurately detect the attacker node. Authors defined various ARP attacker features, e.g., attacker will send ARP frames with a MAC address that is different from that in the link layer frame of the same network packet, an IP-MAC binding table entry, where more than one IP-MAC binding will appear. The authors utilized the IP-MAC binding table information and the attacker features in the algorithm to calculate the probability of the node being an attacker. If the value of probability calculated by the algorithm is above a certain threshold value then the algorithm declares the host as an attacker and the Controller then takes appropriate action to block the host. The authors implemented their solution on the Floodlight Controller and performed simulations in Mininet with 1000 hosts. They also compared their solution with MR-ARP (a solution proposed in [35] to mitigate an ARP cache poisoning attack in traditional networks). Results show that this solution has little impact on network traffic compared to MR-ARP. Moreover, unlike MR-ARP this solution over a long period of time reduces the number of poisoned hosts in the network. This solution increases accuracy to detect the attacker; however, if the attack frequency is low, this solution cannot detect the attacker. Additionally, it is not clear how the value of threshold probability is calculated by the authors.

ARP State Update

ARP State Update is another solution proposed in [36]. The authors in [36] were of the view that the ARP spoofing attack tools generate repetitive ARP reply packets, which can be used to detect an ARP Cache Poisoning attack. This solution is run as a module on the Controller and it maintains IP-MAC bindings in a table. A variable called ARP_STATE (whose default value is 0) is also maintained by this solution. This solution on receiving ARP reply packet checks the MAC address in the received packet with the MAC address in the table. The value of ARP_STATE is changed to 1 if the MAC address in the APR reply packet is different than that in the IP-MAC bindings table. It then starts a timer and waits for another ARP reply. It detects an ARP Cache Poisoning attack if another ARP reply is received before the timer expires. Simulations carried out in Mininet with POX Controller and four hosts show that the proposed solution successfully detects the ARP Cache Poisoning attack. However, how this solution detects ARP Cache Poisoning attack carried out via ARP request packets is

not explained in [36]. Additionally, authors in [36] have chosen 1 s or 2 s as the timer value with no explanation. Moreover, attack detection time is not calculated by the authors.

Issues with IP-MAC address bindings on the Controller Based Solutions

All solutions in this category (except SARP_NAT) are focused on preventing Proxy ARP attack in which attackers try to poison the cache of the Controller. However, Regular ARP attack in which attackers try to poison the cache of the other hosts is not considered. SARP-NAT is the only solution that provides a protection mechanism against both Proxy ARP attack and Regular ARP attack. Moreover, NFGA and AAI are vulnerable to a Proxy ARP Cache Poisoning attack via ARP request messages as both solutions only check ARP replies. All the solutions in this category suffer from two issues. Firstly, it is not clear how to create the IP-MAC bindings table in the beginning as ARP packets sent initially could already be spoofed. Additionally, some solutions [30,31,34] in this category build the IP-MAC bindings based on the DHCP messages in the network. However, this way of building the table is not effective in mitigating the ARP Cache Poisoning Attack. This is because of the fact that the attacker can carry out a DHCP spoofing attack [37] which can corrupt or poison the IP-MAC bindings maintained by the Controller. Hence, creation of trusted IP-MAC bindings table in all these solutions is a major issue. Secondly, these solutions require the construction and maintenance of the IP-MAC bindings table, which may cause delay and extra processing on the Controller if a large number of hosts are present in the network. Important features of these solutions are summarized in Table 3.

Table 3. Summary of Maintaining IP-MAC Address Bindings on the Controller Based Solutions.

Solution	Advantage	Issue	Proxy ARP Attack Prevention	Regular ARP Attack Prevention	Spoofed ARP Requests Handling	Spoofed ARP Replies Handling
SARP-NAT [21]	Less time taken to handle ARP request compared to regular ARP	Too much load on the Controller for large number of hosts	Yes	Yes	Yes	Yes
SEASDN [30]	Less CPU utilization of the Controller	Time taken to detect ARP spoofing attack is not computed	Yes	No	Yes	Yes
L3-ARPSEC [31]	Can effectively detect and block the attacker	Overhead of maintaining Timers	Yes	No	Yes	Yes
NetWatch [32]	Quickly detects the attack than standard POX Controller	Cannot detect the attack if number of spoofed ARP packets is small	Yes	No	Yes	Yes
Secure Binder [33]	Protect against identifier bindings based attacks	Overhead to implement IEEE 802.1x protocol.	Yes	No	Yes	Yes
NFGA [34]	Detect spoofed ARP replies	CPU utilization of the Controller is not determined	Yes (Spoofed ARP replies only)	No	No	Yes
AAI [13]	Low ARP response time even with 1000 hosts	Attack detection time is not computed	Yes (Spoofed ARP replies only)	No	No	Yes
Bayes Based Attack Detection [5]	Increases the accuracy to detect the attacker	Cannot detect the attacker if attack frequency is low	Yes	No	Yes	Yes
ARP State Update [36]	Detects the attack carried out by spoofed ARP replies	Attack detection time is not calculated	Yes (Spoofed ARP replies only)	No	No	Yes

IP-MAC Address Bindings on the Other Network Elements

The solutions in this category maintain IP-MAC address bindings on other network elements like a server or SDN switch. These solutions are discussed in the following sections.

ARP in Hybrid SDN

Authors in [38] proposed a solution to mitigate the ARP Cache Poisoning attack in hybrid SDN. A hybrid network consists of legacy switches and SDN infrastructure, i.e., Controller and SDN enabled switches. The solution proposed by the authors consists of many components (e.g., component to get topology information, component to install flow rules in the switches etc.) and is implemented on a server that handles all ARP requests in the network. Both legacy switches and SDN switches share all topological information (i.e., IP to MAC bindings etc.) with this server. Flow rules are installed in the SDN switches by the Controller to forward all ARP traffic to the server for analysis. Similarly, legacy switches are also configured to direct all ARP related traffic. The legacy switch, on receiving the ARP request, forwards it to the nearest SDN switch. The SDN switch then forwards this request to the server for handling. The server only entertains this ARP request if the IP address in the ARP packet belongs to the current network otherwise the packet is dropped. If the IP address in the ARP packet belongs to the current network then the IP-MAC address mapping is checked in a table maintained by the server. If IP and MAC binding maintained by the server is the same as that present in the ARP packet, then the ARP request is replied to; otherwise, it is dropped. Gratuitous ARP packets with fake IP addresses are also detected by finding the IP-MAC address bindings in the table maintained by the server. Simulations of a hybrid network that consists of POX Controller and multiple SDN and legacy switches were conducted in Mininet to check the effectiveness of this solution. The authors in [38] compared their results with those obtained by [23]. Results show that this solution detects the ARP.

Cache Poisoning attack quicker than the solution proposed in [23]. Another advantage of this solution is that it does not put load on the Controller; however, for this solution a separate server has to be maintained. Moreover, all ARP requests by legacy and SDN switches have to be forwarded to this server which will generate a lot of traffic in the network. According to the authors, this solution can also be used in a Multi-Controller SDN network; however, no details or results are provided to support this claim.

Switch Reactive ARP Query

Another solution that maintains IP-MAC address binding is called Switch reactive ARP query and is proposed in [39]. The author in [39] was of the view that maintaining IP-MAC address database at the Controller creates a lot of load on the Controller therefore they propose an alternate way of maintaining this database at the switch. IP-MAC and MAC-port mappings are maintained at the switch in the form of OpenFlow flows. When a host sends an ARP packet then the IP-MAC address in mapping table is checked. If mapping is found, the host is validated and a corresponding ARP reply is sent to the host. If this is a new host then the flow entry will not be present in the table. In such a case, the switch will itself send an ARP query request packet to validate the host. If the host is validated correctly, a Packet-in event is sent to the Controller so that it can install a matching rule on the switch. The packet is dropped if the host is not validated. The author in [39] also pointed out that if an attacker binds the IP address of the victim's machine with its own MAC address then in such a scenario packets coming from the victim's machine will be dropped. This issue can be mitigated by installing proactive rules for all 'safe hosts' in advance. The author in [39] carried out simulations in Mininet with Ryu Controller to gauge the effectiveness of proposed scheme. The results show that this mechanism can effectively stop the ARP Cache Poisoning attack. This solution does reduce load on the Controller but transfers it to the switches as they now have to maintain the IP-MAC bindings table. This solution also requires that switches send an ARP query by themselves to validate the hosts which will cause an extra overhead on switches and may affect their performance in the presence of large number of hosts. The authors also did not explain how the table is created by the Switch (e.g., by monitoring DHCP packets).

IP Switching

IP switching is a solution proposed in [9]. In this solution, authors are of the view that two different levels of addressing (IP and MAC) are responsible for many vulnerabilities and attacks in SDN. They, therefore, propose that only IP addresses should be used for switching in SDN. This solution involves an ARP server that is running on the Controller. The main aim of the ARP server is to reply to all ARP requests with a virtual non-existent MAC address. The clients in the network use this MAC address and IP address of the other hosts when sending Ethernet frames. The switches forward these packets to the Controller that uses an IP address to find a route for the packet and also to create matching entries in the switches. The last switch in the route of the packet maintains an IP-MAC binding in a table. This switch then adds the corresponding MAC address that matches the IP address and forwards the packet towards the destination host. As this solution uses virtual MAC addresses, all nodes in the network only know their own MAC address and IP addresses of other systems. Simulations carried out in Mininet with Ryu Controller show that IP switching can effectively block an ARP Cache Poisoning attack. However, the authors did not explain how spoofed ARP replies are handled by this solution. The disadvantage of this solution is that SDN switches now have to search in the table and then replace the virtual MAC addresses with the original MAC address in the Ethernet frames. This will incur more delay in forwarding IP packets. The authors have not calculated this delay. Moreover, the IP-MAC bindings now have to be maintained in all the switches, which is an overhead for the system. The authors also failed to explain how the IP-MAC binding table is created on switches. IP spoofing can also lead to security issues in this solution and this fact is not considered by the authors.

IPv4 ARP Server

The authors in [40] proposed a solution to prevent an ARP Cache Poisoning attack in smart home Internet of Things (IoT) networks. Their solution is simple and it consists of setting up an ARP server as a Network Functions Virtualization (NFV) security service in the IoT network. The Controller instructs all switches to forward ARP reply and request messages to the server. In this way, all ARP requests are generated by a trusted entity which prevents the ARP Cache Poisoning attack. Authors carried out simulations in Mininet and used Faucet Controller in their network. In order for faster processing of packets, they implemented ARP server in Data Plan Development Kit (DPDK). Results showed that all fake ARP packets are dropped by the server. Authors also determined the ARP response time using the DPDK server and traditional ARP (without ARP server). They found that the DPDK server provides 50 µsec higher response time compared to the traditional ARP. DPDK server can easily handle a load of 50 ARP requests. However, maintaining one ARP server is a single point of failure and can stop the functioning of the whole network if it is down for any reason. Moreover, this solution is specifically proposed for a smart home IoT network and cannot be readily used in traditional networks. The authors should have considered how Neighbor Discovery Spoofing is detected and blocked in IPv6, as smart home IoT networks are widely using IPv6.

Distributed Responder ARP (DR-ARP)

DR-ARP is proposed in [41] and it is another solution to maintain IP-MAC bindings. DR-ARP responder service operates separately from the Controller and is written in the Python language. The Controller in SDN directs all switches to forward a copy of all traffic to the DR-ARP responder so that it can create and continuously update an IP-MAC bindings table. DR-ARP also requires that some meta-data (e.g., identifiers of switches and ports) should be attached with ARP requests for enhancing security. The DR-ARP responder replies to all ARP requests based on the entries in the IP-MAC table. There can be many ARP-responders in the network and they share their IP-MAC bindings table with each other. However, the authors in [41] only focused on spoofed ARP requests and did not explain how spoofed ARP replies are handled by DR-ARP. Simulations were carried out in [41] using OpenvSwitch and a simplified Controller written in Python. Results show that in the

simulation environment of the authors, DR-ARP takes 5–6 msec to complete an ARP query which is slightly higher than standard ARP. This solution is also scalable as multiple DR-ARP responders can handle many hosts. However, this solution requires maintenance of DR-ARP modules which is an overhead. Additionally, implementing DR-ARP violates the standard SDN architecture which makes this solution less feasible to be deployed in real world. Moreover, attack detection time is not computed by the authors.

Issues with Maintaining IP-MAC Address Bindings on the Other Network Elements Based Solutions

The main aim of these solutions is to reduce load on the Controller and also to protect it from the ARP cache poisoning attack. These solutions mitigate the Proxy ARP attack on the Controller; however, none of the solutions have considered mitigating the Regular ARP attack. Moreover, switches in some of these solutions [9,39] now have the extra task of maintaining the IP-MAC bindings table which may affect their performance. Additionally, a single server as proposed in [38,40] can be a single point of failure. Moreover, these solutions deviate from standard SDN architecture in which the Controller is the main brain of the entire network. Salient features of these solutions are summarized in Table 4.

Table 4. Summary of IP-MAC Address Bindings on Other Network Elements Based Solutions.

Solution	Advantage	Issue	Proxy ARP Attack Prevention	Regular ARP Attack Prevention	Spoofed ARP Requests Handling	Spoofed ARP Replies Handling
ARP in Hybrid SDN [38]	Detects attack in a hybrid network	Server can be a single point of failure	Yes	No	Yes	Yes
Switch Reactive ARP Query [39]	Less load on the Controller	Overhead to send ARP packets to validate hosts.	Yes	No	Yes	Yes
IP Switching [9]	Using IP address for switching	Delay to replace virtual MAC address with real MAC address	Yes (only from spoofed ARP requests)	No	Yes	No
IPv4 ARP Server [40]	Easily handle 50 parallel ARP requests	Solution cannot be used in traditional networks	Yes	No	Yes	Yes
DR-ARP [41]	Less ARP completion time even with large number of hosts	Attack detection time is not computed	Yes (only from spoofed ARP requests)	No	Yes	No

5. Comparison of Different Solutions and Future Work

In the last section various SDN solutions to mitigate ARP Cache Poisoning attack in SDN are discussed and different issues related to them are highlighted. However, it is also important to compare different solutions in terms of various performance metrics. These performance metrics will help to determine the usefulness and effectiveness of these solutions. This comparison will also help to propose future research directions that can be utilized to propose better solutions to mitigate ARP Cache Poisoning attack in SDN. A comparison of different solutions in terms of various performance metrics along with future research directions are discussed in this section. Table 5 shows a summary of performance comparison of various solutions.

Table 5. Comparison of Various Solutions.

Taxonomy of Solutions	Solution	Attack Detection Time	ARP Response Time	Controller Utilization	SDN Architecture Modified	Number of Hosts	Hybrid SDN	Multi-Controller Architecture
Traffic Patterns Based Solutions [12,23,24]	FICUR	Yes	No	No	No	03	No	No
	ARP Packet Analysis	Yes	No	Yes	No	03	No	No
	OrchSec	No	No	No	Yes	N/A	No	Yes
Flow Graph Based Solutions [4,6,25]	SPHINX	Yes	No	Yes	No	1000	No	No
	DistBlock-Net	No	No	Yes	Yes	6000	No	Yes
	FS-Open Security	No	No	No	No	N/A	No	No
IP-MAC Address Bindings on the Controller Based Solutions [5,13,21,30–34,36]	SARP-NAT	No	Yes	Yes	No	03	No	No
	SEASDN	No	No	Yes	No	50	No	No
	L3-ARPSEC	No	No	No	No	03	No	No
	NetWatch	No	Yes	No	No	04	No	No
	Secure Binder	Yes	No	Yes	Yes	04	No	No
	NFGA	No	No	No	No	06	No	No
	AAI	No	Yes	Yes	No	1000	No	No
	Bayes Based Attack Detection	No	No	No	No	1000	No	No
	ARP State Update	No	No	No	No	04	No	No
IP-MAC Address Bindings on Other Network Elements Based Solutions [9,38–41]	Hybrid ARP-SDN	Yes	No	Yes	Yes	09	Yes	No
	Switch Reactive ARP Query	No	No	Yes	Yes	02	No	No
	IP Switching	No	No	No	Yes	02	No	No
	IPv4 ARP Server	No	Yes	No	Yes	50	No	No
	DR-ARP	No	Yes	No	Yes	48	No	No

Future research directions suggested in this section should be read alongside the comparison provided in Table 5. Researchers can use our analysis to identify target metrics applicable to their research, and aim to improve on those metrics. In a broader sense, research on SDN is limited only by the imagination of the research community. It is time for the research community to move beyond solutions based on the traditional ARP protocol, and develop software that implements alternative ways to map an IP address to a physical device.

5.1. Attack Detection Time

Attack detection time is defined as the time required to detect the ARP Cache Poisoning attack. All solutions should be evaluated on how quickly they can detect the attack; however, our survey results show that only [4,12,23,33,38] have taken this performance metric into account when evaluating their solution. Future researchers should take this metric into account and design the solutions that can quickly detect the ARP Cache Poisoning attack.

5.2. Attack Mitigation Time

Another important metric is the attack mitigation time which is defined as the total time lapsed from detecting the attack to completely blocking the attacker. Our survey results show that none of the solutions have taken this metric into account. Most solutions, on detecting the ARP Cache Poisoning attack, instruct the Controller to install flows in the switches to block the attacker, but the exact time taken for this whole process is not calculated. Future researchers should consider this metric when evaluating their solutions.

5.3. ARP Response Time

ARP response time is defined as the time taken by the solution to handle an ARP request. The solutions proposed to mitigate ARP Cache Poisoning attack should not only quickly detect and block the attacker but also provides less delay in handling ARP requests. This value of ARP response time provided by any solution should be compared with the ARP response time in normal SDN, i.e., in the absence of any solution. This will enable the researchers to calculate the overhead involved in using their proposed solution. Our survey results show that only [13,21,32,40,41] have taken this metrics into account when evaluating their proposed solution. This metric is important and must be considered by future researchers.

5.4. CPU Utilization of the Controller

Critical evaluation of all the solutions show that Controller plays a significant role in implementing most of the solutions, e.g., running applications that maintain IP-MAC address bindings [21,30–34], running applications that analyze all ARP packets [12,23,24] etc. However, it should be noted that the Controller is the brain of the network and performs many other tasks. The proposed solution may involve too much utilization of CPU which may affect its performance. Therefore, it is important to calculate how much CPU is utilized by any solution. As shown in Table 5, many solutions, e.g., [5,6,12,31,32,36,40,41] have not considered this metric. Future research in this area should consider calculating CPU utilization of the Controller for their proposed solution.

5.5. Modification of SDN Architecture

Our survey results show that some solutions modify the SDN architecture, e.g., all maintaining IP-MAC address bindings on other network elements based solutions [9,38–41] modify the SDN architecture (as mentioned in Section 4.3.1). This will make these solutions impractical to be deployed in real world. Future researchers should propose solutions that do not modify the standard SDN architecture so that they can be easily deployed.

5.6. Number of Hosts

Scalability is another important metric that must be considered when designing solutions to mitigate the ARP Cache Poisoning attack. Even in the presence of a large number of hosts in the network, a scalable solution should provide lower ARP response time and quick detection and mitigation of the attack. Our survey results show that many authors have evaluated their solutions with few hosts, e.g., FICUR [12] quickly detects the attack; however, results are based on only three hosts. On the other hand, some authors have considered large number of hosts, e.g., the authors in [25] considered 6000 IoT nodes but they did not calculate ARP response time, attack detection and mitigation time; therefore, it is not clear whether the solution is scalable. To determine scalability of any solution, future solutions should calculate the ARP response time, attack detection time and attack mitigation time in the presence of large number of hosts.

5.7. Hybrid SDN

Studies [42–44] suggested that many organizations are reluctant to deploy SDN because of various reasons including budget constraints, fear of downtime due to attacks against the centralized Controller etc. This makes it highly unlikely that there will be a sudden change from legacy networks to SDN. To overcome this issue, many researchers have proposed hybrid SDN as a solution in which both traditional network and SDN infrastructure can co-exist [42,43]. However, many security attacks (including ARP Cache Poisoning attack) can also occur in hybrid SDN [43,44]. Our survey results show that only authors in [38] have proposed a solution to mitigate ARP Cache Poisoning attack in hybrid SDN. Future research should also be directed in designing solutions that detect and block ARP Cache Poisoning attack in hybrid SDN.

5.8. Multi-Controller Architecture

Single Controller based SDN architecture suffers from issues like scalability and single point of failure [45,46]. To overcome these issues, many studies [45–48] have proposed the use of multi-Controller SDN architecture. In multi-Controller architecture, each Controller is a master of few switches and all Controllers share information with each other to manage the entire network. However, studies suggest that presence of many Controllers can lead to configuration inconsistencies and conflicts which can be exploited by attackers to launch many security attacks (including the ARP Cache Poisoning attack) on the network [47,48]. As shown in Table 5, none of the researchers have proposed solutions that can mitigate ARP Cache Poisoning attack in a multi-Controller architecture. Only DistBlockNet [25] has provided some results of many IoT nodes managed by multiple Controllers. Moreover, authors in [24] mentioned that OrchSec can support a multi-Controller architecture, but no results are provided. Future researchers should propose solutions that can detect and block this attack in a multi-Controller SDN architecture.

5.9. Wireless SDN

Deployment of wireless data capability has been very common in consumer and commercial networks for a number of years [49]. In spite of this, very little research is available that focuses on wireless data networks using SDN [50]. A few manufacturers (e.g., Allied Telesis [51]) have implemented OpenFlow support in their production wireless Access Points (APs). An alternative approach is to implement OpenFlow in a Wireless Network Controller. Aruba [52] and Cisco [53] are manufacturers who take this approach. None of the solutions surveyed for this paper made reference to wireless SDN. In the research community, the SWIFT project [50] is an example of use of commodity wireless APs to implement open source SDN management of a wireless network. ARP handling is an area identified by [50] as requiring particular attention. Client isolation, described in [50], is implemented in a variety of different ways, depending on the chipset the manufacturer has chosen to use in a particular model of AP. The lack of wireless APs with native OpenFlow support,

and complexity in adding OpenFlow support, has resulted in there being little available literature focusing on standards-based wireless SDN. Therefore, mitigating ARP Cache Poisoning attack in wireless SDN devices is an opportunity for future research.

5.10. Comparison with Other Solutions

It is interesting to note that few solutions [5,21] discussed above have compared their work with other solutions. In order to gauge the effectiveness of any solution it is important to carry out a performance comparison of that solution with other solutions in terms of various performance metrics, e.g., attack detection time, attack mitigation time, ARP response time etc. It is found in this survey that some studies have compared their work with the standard ARP protocol, e.g., the authors in [21] compared the results of SARP_NAT with standard ARP protocol in terms of ARP response time and CPU utilization. Similarly, the authors in [5] compared their results with MR-ARP [35]. Future research work should carry out comparison with other solutions so that the best among all the solutions can be identified.

5.11. SDN Based IoT Networks

SDN is particularly advantageous in IoT networks because central control and programmability enables researchers and practitioners to modify network behavior in ways not possible in conventional networks. Centralized control can make management of many IoT devices simpler, including flexibility to tune components within IoT so that performance and security can be continuously enhanced [54–56]. It is therefore important that solutions should be proposed that can mitigate ARP Cache Poisoning attack in SDN based IoT networks. Our survey results show that DistBlockNet [25] and IPv4 ARP Server [40] are the two solutions that mitigate this attack in SDN based IoT networks. However, both DistBlockNet [25] and IPv4 ARP Server [40] suffer from various issues and do not address all the performance metrics as shown in Table 5. Future researchers should propose better solutions to mitigate ARP Cache Poisoning attack in SDN based IoT networks. IPv6 is widely used in IoT based networks which unlike IPv4 networks use the Neighbor Discovery Protocol (NDP) for address resolution. Studies [57,58] suggest that NDP is vulnerable to many attacks (e.g., spoofing, DDoS etc.) and future research is needed in this direction.

Moreover, it should be noted that future research directions suggested in this section can be applied equally to conventional IoT networks, with researchers identifying and using metrics particularly applicable to their network.

6. Conclusions

In this survey, various solutions proposed in the existing literature to mitigate ARP Cache Poisoning attack in SDN are classified into three types: Flow Graph based solutions; Analysis of Traffic Patterns based solutions; Maintaining IP-MAC Address Bindings based solutions. The working principles, advantages and issues associated with all the solutions are discussed in detail. All the solutions are also critically evaluated and compared in terms of various performance metrics. It is noted from this survey that IP-MAC bindings based solutions such as SARP_NAT [21], AAI [13], Secure Binder [33], ARP in Hybrid SDN [38] and Flow Graph based solution like SPHINX [4] perform better than other solutions as they address most of the performance metrics. However, each one of them has some issues which need to be addressed in future research works. Various future research directions are discussed in this survey which will enable future researchers to address the limitations in the current solutions to effectively mitigate ARP Cache Poisoning Attack in SDN. Future researchers can also take into account various performance metrics given in this research to design better solutions to mitigate the ARP Cache Poisoning attack in SDN.

Author Contributions: Conceptualization, Z.S.; methodology, Z.S. and S.C.; investigation, Z.S. and S.C.; resources, Z.S. and S.C.; writing—original draft preparation, Z.S. and S.C.; writing—review and editing, Z.S. and S.C.; project administration, Z.S. and S.C.

Funding: This research received no external funding.

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

References

1. Nunes, B.; Mendonca, M.; Nguyen, X.; Obraczka, K.; Turletti, T. A Survey of Software-defined Networking: Past, Present, and Future of Programmable Networks. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 1617–1634. [CrossRef]
2. Kreutz, D.; Ramos, F.; Verissimo, P.; Rothenberg, C.; Azodolmolky, S.; Uhlig, S. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* **2015**, *103*, 14–76. [CrossRef]
3. Shah, Z. Mitigating TCP Incast Issue in Cloud Data Centres using Software-Defined Networking (SDN): A Survey. *KSII Trans. Internet Inf. Syst.* **2018**, *12*, 5179–5202.
4. Dhawan, M.; Poddar, R.; Mahajan, K.; Mann, V. SPHINX: Detecting Security Attacks in Software-Defined Networks. In Proceedings of the Network and Distributed System Security Symposium, San Diego, CA, USA, 8–11 February 2015; pp. 8–11.
5. Ma, H.; Ding, H.; Yang, Y.; Mi, Z.; Yang, J.Y.; Xiong, Z. Bayes-based ARP Attack Detection Algorithm for Cloud Centers. *Tsinghua Sci. Technol.* **2016**, *21*, 17–28. [CrossRef]
6. Sung, Y.; Sharma, P.; Lopez, E.; Park, J. FS-OpenSecurity: A Taxonomic Modeling of Security Threats in SDN for Future Sustainable Computing. *Sustainability* **2016**, *8*, 919. [CrossRef]
7. Conti, M.; Dragoni, N.; Lesyk, V. A Survey of Man in the Middle Attacks. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 2027–2051. [CrossRef]
8. Hijazi, S.; Obaidat, M.S. A New Detection and Prevention System for ARP Attacks Using Static Entry. *IEEE Syst. J.* **2018**, *13*, 2732–2738. [CrossRef]
9. Rietz, R.; Cwalinski, R.; König, H.; Brinner, A. An SDN-Based Approach to Ward Off LAN Attacks. *J. Comput. Netw. Commun.* **2018**, *2018*, 4127487. [CrossRef]
10. Sasan, Z.; Salehi, M. SDN-based Defending Against ARP Poisoning Attack. *J. Adv. Comput. Res.* **2017**, *8*, 95–102.
11. Bruschi, D.; Ornaghi, A.; Rosti, E. S-ARP: A Secure Address Resolution Protocol. In Proceedings of the 19th Annual Computer Security Applications Conference, Las Vegas, NV, USA, 8–12 December 2003; pp. 66–74.
12. Nehra, A.; Tripathi, M.; Gaur, M.S. FICUR: Employing SDN Programmability to Secure ARP. In Proceedings of the IEEE 7th Annual Computing and Communication Workshop and Conference, Las Vegas, NV, USA, 9–11 January 2017; pp. 1–8.
13. Xia, J.; Cai, Z.; Hu, G.; Xu, M. An Active Defense Solution for ARP Spoofing in OpenFlow Network. *Chin. J. Electron.* **2019**, *28*, 172–178. [CrossRef]
14. Hijazi, S.; Obaidat, M. Address resolution protocol spoofing attacks and security approaches: A survey. *Secur. Priv.* **2019**, *2*, e49. [CrossRef]
15. Prajapati, S.; Noorani, Z. A Survey on ARP Poisoning and Techniques for Detection and Prevention. *Int. J. Adv. Res. Innov. Ideas Educ.* **2017**, *3*, 594–601.
16. Hingne, A.; Jain, S. A Survey on Various Detection and Prevention Mechanism for MITM and ARP Attacks. *Int. J. Innov. Res. Comput. Commun. Eng.* **2016**, *4*, 19918–19924.
17. Meghana, J.; Subashri, T.; Vimal, R. A Survey on ARP Cache Poisoning and Techniques for Detection and Mitigation. In Proceedings of the IEEE International Conference on Signal Processing, Communication and Networking (ICSCN), Chennai, India, 16–18 March 2017; pp. 1–6.
18. Kapil, J.; Manoj, J.; Jay, B. A Survey on Man in the Middle Attack. *Int. J. Sci. Technol. Eng.* **2016**, *2*, 277–280.
19. Jaideep, S.; Vinit, G. A Survey of Different Strategies to Pacify ARP Poisoning Attacks in Wireless Networks. *Int. J. Comput. Appl.* **2015**, *116*, 25–28.
20. Bhushan, B.; Sahoo, G.; Rai, A. Man-in-the-Middle Attack in Wireless and Computer Networking—A Review. In Proceedings of the IEEE International Conference on Advances in Computing, Communication & Automation, Dehradun, India, 15–16 September 2017; pp. 1–6.
21. Alharbi, T.; Durando, D.; Pakzad, F.; Portmann, M. Securing ARP in Software Defined Networks. In Proceedings of the IEEE Conference on Local Computer Networks (LCN), Dubai, UAE, 7–10 November 2016; pp. 523–526.
22. Bailey, J.; Stuart, S. Faucet: Deploying SDN in the enterprise. *ACM Commun.* **2017**, *60*, 45–49. [CrossRef]

23. AbdelSalam, A.; El-Sisi, A.; Reddy, V. Mitigating ARP Spoofing Attacks in Software-Defined Networks. In Proceedings of the International Conference on Computer Theory and Applications (ICCTA), Alexandria, Egypt, 25–27 October 2016.
24. Zaalouk, A.; Khondoker, R.; Marx, R.; Bayarou, K. OrchSec: An Orchestrator-Based Architecture for Enhancing Network-Security Using Network Monitoring and SDN Control Functions. In Proceedings of the Network Operations and Management Symposium (NOMS), Krakow, Poland, 5–9 May 2014; pp. 1–9.
25. Sharma, P.; Singh, S.; Jeong, Y.; Park, J. Distblocknet: A distributed blockchains-based secure SDN architecture for IoT networks. *IEEE Commun. Mag.* **2017**, *55*, 78–85. [CrossRef]
26. Karame, G. On the Security and Scalability of Bitcoin's Blockchain. In Proceedings of the ACM Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 1861–1862.
27. Gramoli, V. On the Danger of Private Blockchains. In Proceedings of the Workshop on Distributed Cryptocurrencies and Consensus Ledgers, Chicago, IL, USA, 25 July 2016.
28. Saad, M.; Spaulding, J.; Njilla, L.; Kamhoua, C.; Shetty, S.; Nyang, D.; Mohaisen, A. Exploring the attack surface of blockchain: A systematic overview. *arXiv* **2019**, arXiv:1904.03487.
29. Gupta, A.; Sukheja, D.; Tiwari, A. Impact of Sybil Attack and Security Threat in Mobile Adhoc Network. *Int. J. Comput. Appl.* **2015**, *124*, 5–12. [CrossRef]
30. Jehan, N.; Haneef, A. Scalable Ethernet Architecture Using SDN by Suppressing Broadcast Traffic. In Proceedings of the IEEE International Conference on Advances in Computing and Communications (ICACC), Kochi, India, 2–4 September 2015; pp. 24–27.
31. Oliveira, R.; Shinoda, A.; Schweitzer, C.; Iope, R.; Prete, L. L3-ARPSec–A Secure Openflow Network Controller Module to Control and Protect the Address Resolution Protocol. In Proceedings of the XXXIII Simpósio Brasileiro Telecomunicações, Juiz de Fora, Brazil, 25 February 2015; pp. 158–162. [CrossRef]
32. Balagopal, D.; Rani, X. NetWatch: Empowering Software-Defined Network Switches for Packet Filtering. In Proceedings of the IEEE International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), Davangere, India, 29–31 October 2015; pp. 837–840.
33. Jero, S.; Koch, W.; Skowyra, R.; Okhravi, H.; Nita-Rotaru, C.; Bigelow, D. Identifier Binding Attacks and Defenses in Software-Defined Networks. In Proceedings of the USENIX Security Symposium, Vancouver, BC, Canada, 16–18 August 2017; pp. 415–432.
34. Cox, J.; Clark, R.; Owen, H. Leveraging SDN for ARP Security. In Proceedings of the IEEE SoutheastCon, Norfolk, VA, USA, 30 March–3 April 2016; pp. 1–8.
35. Kumar, S.; Tapaswi, S. A centralized detection and prevention technique against ARP poisoning. In Proceedings of the IEEE International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), Kuala Lumpur, Malaysia, 26–28 June 2012; pp. 259–264.
36. Kim, Y.; Ahn, S.; Thang, N.; Choi, D.; Park, M. ARP Poisoning Attack Detection Based on ARP Update State in Software-Defined Networks. In Proceedings of the IEEE International Conference on Information Networking, Kuala Lumpur, Malaysia, 9–11 January 2019; pp. 366–371.
37. Mallik, A.; Ahsan, A.; Shahadat, M.; Tsou, J. Man-in-the-middle-attack: Understanding in simple words. *Int. J. Data Netw. Sci.* **2019**, *3*, 77–92. [CrossRef]
38. Fahad, U.; Rashid, A.; Faisal, U.; Muwar, I. Mitigating Address Spoofing Attacks in Hybrid SDN. *Int. J. Adv. Comput. Sci. Appl.* **2017**, *8*, 562–570.
39. Solomon, N. Mitigating Layer 2 Attacks: Re-Thinking the Division of Labor. Master's Thesis, School of Computer Science, The Interdisciplinary Center, Herzliya, Israel, 2015.
40. Al-Shaboti, M.; Welch, I.; Chen, A.; Mahmood, M. Towards Secure Smart Home IoT: Manufacturer and User Network Access Control Framework. In Proceedings of the IEEE International Conference on Advanced Information Networking and Applications (AINA), Krakow, Poland, 16–18 May 2018; pp. 892–899.
41. Matties, M. Distributed Responder ARP: Using SDN to Re-Engineer ARP from Within the Network. In Proceedings of the IEEE International Conference on Computing, Networking and Communications (ICNC), Santa Clara, CA, USA, 26–29 January 2017; pp. 678–683.
42. Sinha, Y.; Haribabu, K. A survey: Hybrid SDN. *J. Netw. Comput. Appl.* **2017**, *100*, 35–55.
43. Amin, R.; Reisslein, M.; Shah, N. Hybrid SDN networks: A survey of existing approaches. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 3259–3306. [CrossRef]
44. Vissicchio, S.; Vanbever, L.; Bonaventure, O. Opportunities and research challenges of hybrid software defined networks. *ACM Comput. Commun. Rev.* **2014**, *44*, 70–75. [CrossRef]

45. Blial, O.; Ben, M.; Benaini, R. An overview on SDN architectures with multiple Controllers. *J. Comput. Netw. Commun.* **2016**, *2016*, 9396525. [CrossRef]
46. Phemius, K.; Bouet, M.; Leguay, J. Disco: Distributed Multi-Domain SDN Controllers. In Proceedings of the IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland, 5–9 May 2014; pp. 1–4.
47. Qi, C.; Wu, J.; Hu, H.; Cheng, G.; Liu, W.; Ai, J.; Yang, C. An Intensive Security Architecture with Multi-Controller for SDN. In Proceedings of the IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), San Francisco, CA, USA, 10–14 April 2016; pp. 401–402.
48. Shu, Z.; Wan, J.; Li, D.; Lin, J.; Vasilakos, A.; Imran, M. Security in software-defined networking: Threats and countermeasures. *Mob. Netw. Appl.* **2016**, *21*, 764–776. [CrossRef]
49. Zander, J.; Mähönen, P. Riding the data tsunami in the cloud: Myths and challenges in future wireless access. *IEEE Commun. Mag.* **2013**, *51*, 145–151. [CrossRef]
50. Hätönen, S.; Savolainen, P.; Rao, A.; Flinck, H.; Tarkoma, S. SWIFT: Bringing SDN-Based Flow Management to Commodity Wi-Fi Access Points. In Proceedings of the IEEE Networking Conference and Workshops, Zurich, Switzerland, 14–16 May 2018; pp. 1–9.
51. Software Defined Networks. Available online: https://www.alliedtelesis.com/solutions/software-defined-networks (accessed on 28 September 2019).
52. Wi-Fi & SDN ... Why? Available online: https://blogs.arubanetworks.com/industries/wi-fi-sdn-why/ (accessed on 28 September 2019).
53. Cisco Wireless LAN Controller. Available online: https://www.cisco.com/c/en/us/products/wireless/wireless-lanController/index.html (accessed on 28 September 2019).
54. Zarca, A.; Garcia-Carrillo, D.; Bernabe, B.; Ortiz, J.; Marin-Perez, R.; Skarmeta, A. Enabling Virtual AAA Management in SDN-Based IoT Networks. *Sensors* **2019**, *19*, 295. [CrossRef] [PubMed]
55. Bedhief, I.; Kassar, M.; Aguili, T.; Foschini, L.; Bellavista, P. Self-Adaptive Management of SDN Distributed Controllers for Highly Dynamic IoT Networks. In Proceedings of the IEEE International Wireless Communications & Mobile Computing Conference (IWCMC), Tangier, Morocco, 24–28 June 2019; pp. 2098–2104.
56. Salman, O.; Elhajj, I.; Chehab, A.; Kayssi, A. IoT survey: An SDN and fog computing perspective. *Comput. Netw.* **2018**, *143*, 221–246. [CrossRef]
57. Ahmed, S.; Hassan, R.; Othman, E.; Ahmad, I.; Kenish, Y. Impacts evaluation of DoS attacks over IPv6 neihbor discovery protocol. *J. Comput. Sci.* **2019**, *15*, 702–727. [CrossRef]
58. Barbhuiya, A.; Bansal, G.; Kumar, N.; Biswas, S.; Nandi, S. Detection of neighbor discovery protocol based attacks in IPv6 network. *Netw. Sci.* **2013**, *2*, 91–113. [CrossRef]