DP-SPOOF: A Dynamic Spoofing Attack Prevention Based SDN-Switch Microservice Framework for Secure Fog Computing

Md. Julkar Nayeen Mahi[a,*], Esraq Humayun Eisty[a], Alistair Barros[b], Rajkumar Buyya[c] and Md Whaiduzzaman[b]

[a]Daffodil International University, Dhaka, Bangladesh
[b]Queensland University of Technology, Australia
[c]University of Melbourne, Australia

ARTICLE INFO

Keywords:
Resource Utilization
Task Scheduler
Cloud Computing
Fog Computing

Abstract

Providing a secure internet for dedicated users sharing a communication link has become an important issue nowadays. Among several kinds of spoofing activities spread all over the communication system, ARP poisoning is a preferable SDN security module that Protects against ARP spoofing. Dynamic port allocation has been used to detect and prevent ARP attacks. SDN deploys a controller which can be a pox controller, a Ryu controller, or a nox controller. The controller serves as the brain of software-defined networking. Fixed value assignment for states may make it easier for the attacker to drudge the network. Hence random value assignment for conditions may solve this problem. An attacker may also approach multiple arrival requests to verify himself as a legitimate host. So arrival request should be rejected if the DHCP packet arrives more than a specific number of times. Mininet is used for setting up NFG implementation. NFG used an apyretic query. Moreover, our framework's essential contribution affirms secured ARP with nominal involvement by network operators while assisting static and dynamic port allotment, demanding no changes to the current network topology or protocols and no consumer software installation.

1. Introduction

In today's world digital services are shared and taken by cloud platforms. Clouds are data sharing based digital computing platforms that harnesses the ease of deploying complex services. Cloud specifically scales over storage, databases, networking, software, analytics, intelligence while delivering system resources and distributing computing power without the direct or active participation of users. Fog is a part of the middleware architecture of cloud computation stages that carry out communications nearer to the edge devices locally and triggers the routing around the cloud backbone. As a system backbone cloud needs to handle large amount of traffic, communication loopholes, deadlock or network bottlenecks because of occupying with various user requests in different communication switching channels. Hence, security preservation is essential for efficient response time and crucial data delivery across the whole network.

In this day and age of Ethernet, spoofing the dynamic host configuration protocol, more often addressed to as ARP, has become one of the most significant security concerns that may occur.An attacker may manufacture the address of a target host by using a technique known as ARP spoofing. This technique takes use of flaws in the ARP protocol. Because of this, the attacker is able to read communications that are supposed to be sent to the victim. Despite the fact that it is simple, this kind of technique has been there ever since before the invention of Ethernet; yet, it has been difficult to design a successful mitigations against it up to this point. The address resolution protocol, or ARP, is a kind of communication protocol that is employed for the purpose of locating the link-layer address, such as a MAC address, that is associated with a certain internet layer address (commonly known as an IPv4 address).The source IP address, as well as the MAC address and destination IP address, are included in a request.ARP assaults take place when the attacker responds to a request by providing the IP address of the destination.By inspecting the IP/MAC addressing mapping of a suspicious target's cache table, it is possible to easily identify behaviors related to ARP spoofing.For purposes of ARP spoofing, the target MAC address will often be one that is mapped in the ARP store table to more than one IP address.Some of the procedures that may be taken to put a stop to spoofing operations include dynamic ARP scanning and intrusion detection systems. Software-based preventative measures against ARP assaults are more meticulous than their hardware-based counterparts. Spoofing an ARP address has consequences that are comparable to those of other types of spoofing, which including email spoofing.If the attacker uses a denial-of-service attack, the results of ARP spoofing may vary from being completely undetectable to causing the victim to lose all connection to the network. It is dependent on the objectives of the hacker whether or not the impacts of the assault are seen.If here only objective is to monitor or manipulate information sent between hosts, it is possible that they will not be discovered.Those who will also want to stay undiscovered if we want to trigger a subsequent attack, which is frequently the approach with ARP spoofing assaults but may also apply to other types of attacks.However, after

*Principal corresponding author
Queensland University of Technology, Australia (Md. Julkar Nayeen Mahi)
ORCID(s):

we have accomplished their final objective, these kinds of assaults will become clear.

When a packet comes, we have to determine if it is an offer from DHCP or an acknowledgement from DHCP.It is necessary for us to search through the cache table in order to identify whether or not the hostname is already included in the dynamic counter top.When a static port is used, the packet will not be processed (dropped).The value of the port's status is compared to the value that was allocated to it, including that that the connection is active.In the event that the IP is discovered to be a match, the verified state will be located.An attacker has the ability to modify the state values of the network, which is one of the most dangerous risks it faces; an attacker may also attempt to guess the proper state value. The fact that perhaps the intruder may break the measured values with several tries is another disadvantage of the method that is already in place.To the highest of our knowledge, there is no known remedy to this issue at this time.Therefore, measures need to be established so that a user may only submit a single request at a time.It will no longer be feasible for any host to make a request once a certain number of tries have been made.The amount of tries that are made by an IP may be tracked with the help of a counter.In the event where the values of the assigned states are simple to hypothesize, the adversary will immediately access the confirmation state.Therefore, the values of the states are determined by a random number generator.The approach that is currently used operates on static allocation.This indicates that the values of the state will be carried over to each subsequent stage in the process.On the other hand, we are interacting with a dynamically port allocation, which enables us to pass parameters at each stage of the process.

Arp Spoofing is shown in Figure 3

The purpose of this effort is to ensure that a safe ARP packet will be sent from the sender to the destination regardless of any problems that may arise.This is the thinking behind doing this research.In the following, we will go over some of the fundamental elements that are of the utmost importance:

- Techniques for detecting ARP that have been utilized in the past sometimes involve convoluted procedures, and the detection process itself is often painfully slow.In this article, we have shown a strategy that is based on the dynamic allocation of ports rather than the static allocation of ports.Switches each have a port that may connect to the network, and hosts use that port to connect to the network.The majority of previous studies were based on a static method for allocating ports; when spoofing activities in the cache were discovered, it was necessary to manually disconnect the port.Instead, we suggest using a mechanism that dynamically allocates ports and states.

The main contributions of this paper are to present a faster detection of malicious ARP replies. We optimized some metrics to deliver a better solution against spoofing activities.

  - **Random State Allocation:** Random values are assigned as state values at each step of verification. To let this happen, the port has to be dynamic. So, we have ensured that the port has not been defined as static.
  - **Request Blocking:** An intruder may try several times to estimate the correct state value to access the network. Hence, we provide a limited number of trial requests for any host. When the limit is exceeded, that particular IP will be unable to send any request further.
  - **Network Optimization:** Our implementation procedure uses a mininet (a realistic virtual network) utilization. An extensive scale network that is hard to implement on the whole system is implemented using mininet. We use a virtual machine installed with mininet for reconfiguring the system (modification of topologies).

The rest of the part of this paper is organized in the following manner. Section **2** describes the background or previous works that are related to our proposed system. Section **3** provides a overall description of our proposed methodology as well as explains the experimental setup and implementation of our proposed system. Section **4** investigates a comparative study based on the factors of proposed systems. Finally, we discuss the conclusion and future works in section **5**, and **6** respectively.

2. Background and Rationale

The safety of networks is becoming an increasingly pressing issue. In addition, hackers often push the capabilities of the technologies that are accessible to network administrators.Several different mechanisms are carried out.Inspection of dynamic ARP packets is one of them. There are many different kinds of anti-spoofing skills that have been created by developers. Some examples are intrusion detection systems, port security, dynamic ARP inspection, and others.These solutions are not correct nor adequate in any way. Therefore, these solutions need ever more direction in order to prevent the attacker from engaging in spoofing terminologies.

- Background of the problem: As breakthrough networking technologies like Development tools Computer networks (SDN) and Service Function Chaining (SFC) continue to gain in popularity, new security problems have lately surfaced.The use of SDN and

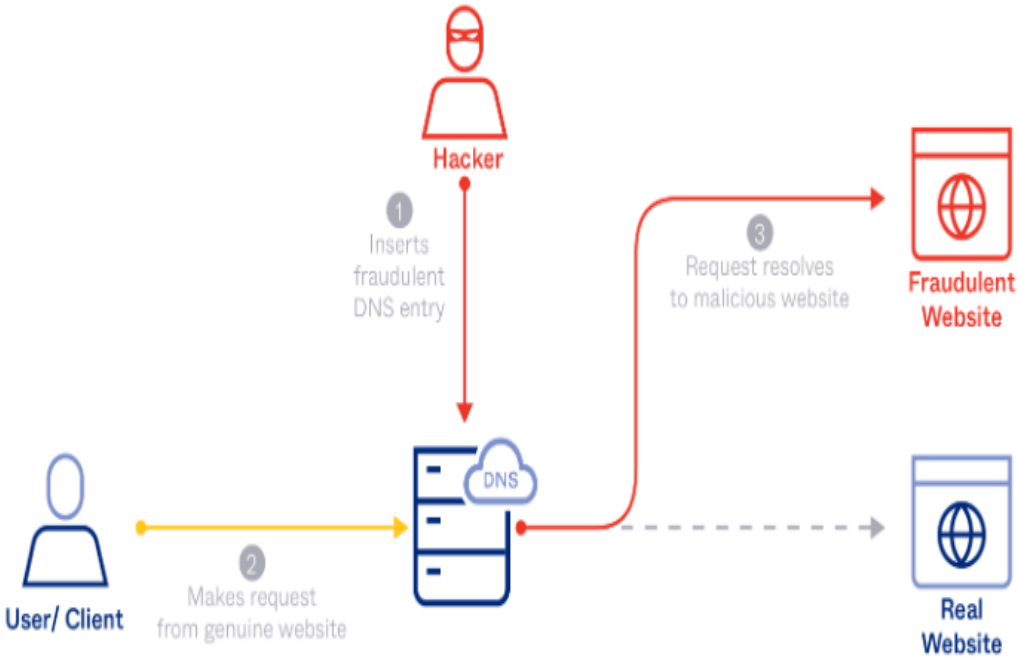

Figure 1: Arp spoofing in Cloud Orchestrated Network

SFC is also becoming more common. Because cyber assaults may originate from a number of places and be aimed at a range of targets, it is necessary to analyze network data from various ipv4 is a connectionless protocol in order to identify cyber attacks that spread from one area to another.The development of a coordinated and widespread ARP Spoofing detection system will be one of the major components. This system will be used to correlate suspicious occurrences across a large number of network locations that are working together in order to identify coordinated assaults.

- Network Intrusion Detection System: The Intrusion Detection System (ids, also referred to as NIDS, is a piece of software that employs a variety of information mining methods in order to automatically identify invasions of privacy into workstation networks and systems. NIDS is also known as the Intrusion Detection Systems.This methodology allows use of an unsupervised classification technique that assigns a point total towards each computer network depending on how anomalous it is. The score is determined by the degree to which the connection deviates from normal behavior.In addition, this technique makes use of a module that is based on association pattern analysis and is responsible for providing a summary of the communication networks which the anomaly detection module identifies as being extremely anomalous.
- Drawback : The volume of data generated by network traffic monitoring is usually quite immense volume Because network data is temporal (streaming), constructing a real-time intrusion detection system necessitates the development of algorithms for mining data streams..
- Intrusion Detection System : Let us take a specific example. Imagine a scenario in which a user is prevented from using a particular application.Therefore, the firewall will get warning about the user's unauthorized access each time he or she attempts to connect into that particular application.Therefore, the user is unable to access the website since doing so is against the terms of the ban imposed on them. The user is prevented from gaining unauthorized access via the firewall. That indicates that the request will not be sent in any form. Now, the user could be crafty enough, and he or she might employ a proxy server in order to acquire unauthorized access. However, if an intrusion detection system is installed into the network, the user may be prevented from accessing that website. This is possible due to the fact that the user is required to use the gateway of the network where he or she is currently located in order to browse the website in question.It is necessary to have a network administrator because an intrusion detection system will sound an alert for the network administrator, which will prompt him or her to be concerned that a user is attempting to access a website that has not been authorized.
- Present state of the problem :

A. attack case 1:

The puzzling problems about the existing method is that the update of state values are fixed there. Using a fixed value for state update provides a risk for the users that one may guess the correct state value and then he may attack the network. Thus using a constant update value may lead to a problem.

B. attack case 2:

The another problem is that we are checking whether dhcp offer or acknowledgement is true or false. This checking is based on the request arrival. When a request arrives we need to check the value for ensuring an offer or an acknowledgement. After that if we find that the request is from a suspicious ip, then we are going to ignore the packet. Here if the attacker tries several times, then he will be succeeded to guess a correct state value. So what we can do is to set a limited number of time and this time indicates how many time an ip can request for offer or acknowledgement. Finally , if request from an ip arrives exceeding that limited number of times, then we will ignore the packet.

We discussed about the drawbacks of the existing algorithm in the background of the problem. In this section we are going to add solutions to the previously mentioned problems. For solving the problems mentioned above, we need to set a counter like variable that can count how many time a request has arrived. Next if a request arrives more than the counter's value ; then we are going to ignore the request. Thus one deficits of the existing algorithm can be solved. In this part, we are going to discuss about the second problem which was described in the previous section. The problem is, if the state value is almost same for all the request then one time the attacker will somehow manage to guess the state value. So we should consider such methods that will prevent the attacker from guessing a correct value for state. In this case state can be updated as we are using dynamic port allocation. Moreover, to solve the above issue, we are going to assign the state with random values. When a request arrives we will be updating its state value with random values. As random values are difficult to guess, so security is assigned to the ARP protocol. The main issue is that , we are using dynamic port[18] allocation. It supports updating state values for ensuring security. The mentioned two problems are threats for the existing method. We are solving these two problems by following the methods mentioned in this section.

- Openflow:

  Openflow protocol is the core of the SDN technology . Openflow adds flexibility and scalability to the network. SDN requires a programmable network protocol as communication between controller and switches needs to be maintained by that protocol. Openflow is that desired protocol which is used for the above purposes. Programming of networking devices is separated from underlying hardware using openflow protocol. openflow makes that network adaptable so that changes in network can be quickly accepted. Openflow offers a centralized and programmable network.

  Because OpenFlow switches lack specific topology finding capabilities, this feature is solely dependent on the SDN controller. Although there is no official standard that defines a specific approach, most SDN controllers use an unofficial method known as OFDP (OpenFlow Discovery Protocol), which uses Link Layer Discovery Protocol (LLDP) to allow information to flow between nodes.

- SDN Controllers:

  SDN is a networking method in which the intelligent control plane is decoupled from networking devices and a distinct entity termed a "controller" is established to govern the behavior of the data plane on physical networking devices. Due to the continuous expansion and growth of SDN controllers in the market, the goal of this work is to give a comprehensive analysis of the performance and scalability of several open source SDN controllers currently accessible in the literature. This work builds on prior research by providing current data and official benchmarking methodology. The study provides a framework based on IETF (Internet Engineering Task Force) suggested standards that will be used by the SDN community to benchmark different SDN controllers.

  The following controllers are included in this survey based on their popularity: • Floodlight • ONOS • POX • OpenDayLight • Ryu • OpenMUL

  plane and control plane have been separated from the similar plane. Data plane introduces the network traffic. Control plane introduces the routing tables for each routers. Number of hardware is reduced as SDN has been introduced. Furthermore SDN introduces us with open flow switch. Open flow switch is a kind of switch that makes switch chip sets to be controlled using a software. That means controlling switches is possible using a software. That is a huge revolutionary contribution from SDN to networking. Thus ARP poisoning attacks are detected and prevented using this methods. Virtualization of networking enables to configure necessary networking tools that were previously done manually. Now measurements that need to performed were done by network administrator for preventing and detecting ARP attack. But now-a-days SDN provides with virtualization of networking that can be configured easily. This procedure is saving much more time[4]. Instead, software defined networking, a new paradigm that decouples the control plane from the switch's data plane to provide

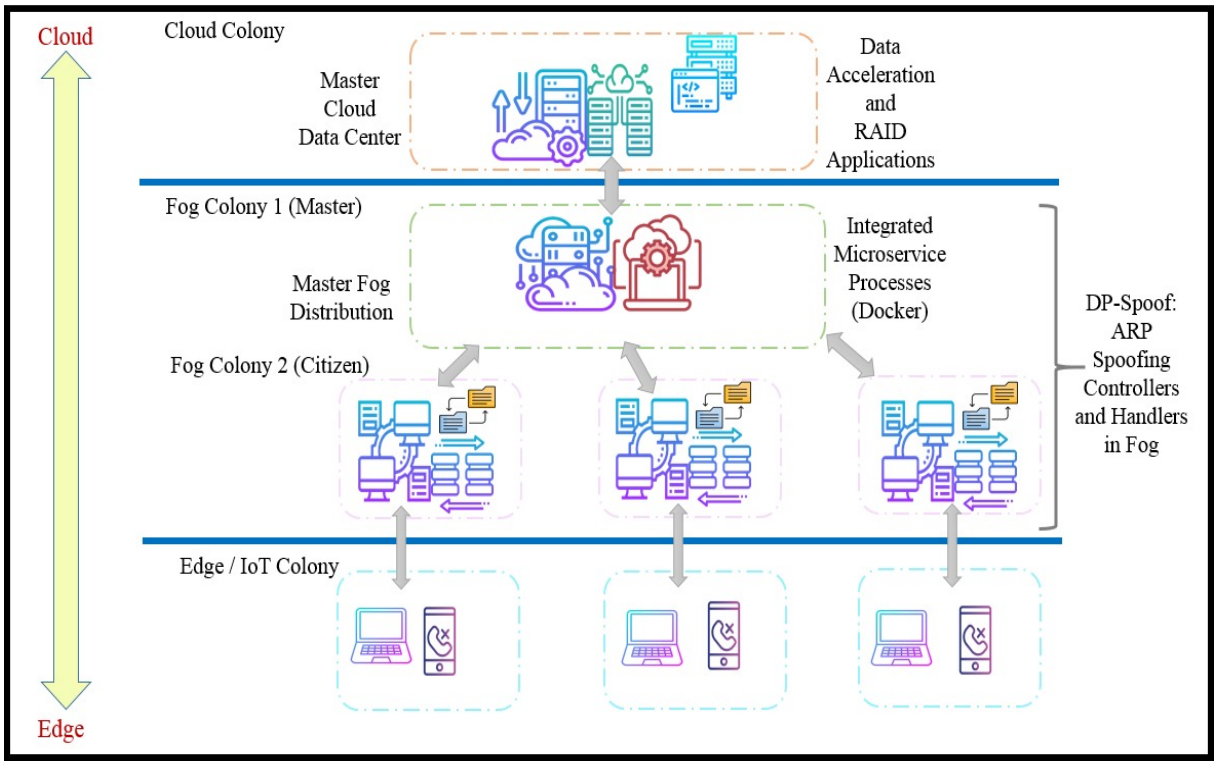

Figure 2: Proposed Method

security against ARP spoofing attacks. Network flow guard is a modular application that augments an SDN controller to detect and prevent ARP-replies from unauthorized hosts. This method requires no topology changes, authentication services, cryptographic keys, or significant network operator support, instead NFG monitors dynamic host configuration protocol (DHCP)Offers, requests, and acknowledgements from valid DHCP server to construct a dynamic table consisting of MAC: IP: port: fixed: state associations for each device on a given LAN. Doing so leverage the capabilities of SDN to prevent ARP poisoning attempts as they occur. NFG's key contribution is that it provides ARP security while requiring little network operator invention, no additional equipment or host software, and no changes to the network's current topology or protocols[5].

3. System Model and Methodology

3.1. Components Description

Figure 3 shows the components of our solution and their interconnected communication around the network.

3.1.1. Master Fog

.

3.1.2. Master Fog Manager

3.1.3. Master Decision Manager

3.1.4. Fog Identifier Interface

3.1.5. Citizen Fog Manager

3.1.6. Citizen Fog 1

3.1.7. Citizen Fog 2

3.1.8. Citizen Fog 3

4. Mathematical Derivation Procedure for Cloud, Fog and IoT Hierarchy

4.1. Time Interval of Process Execution for Fog-Cloud-IoT continuum

From Lebesgue Measure we know that, for any interval $I = [a, b]$ having n continuous periods and a, b as finite elements in the set of R real numbers Let, $l(I) = b - a$, denoting the time length for any subset $E \subseteq R$ the expectation of outer measure $\lambda * (E)$ is defined as lower bound or an infimum. Therefore, $\lambda * (E) = \int (\int_0^n [\sum_{n-1}^n l(I_n)])$ where $(I_n) \in N$is a sequence of open intervals in N dimensional space with $\subset \sum \mathbb{U} (n-1)^n (I_n)$ Since the expectation probability, tends to lie in exponential data deliveries hence our data distribution relies upon multiserver queuing kendal model and process execution generated by poisson process, where each process executes independently to one another.

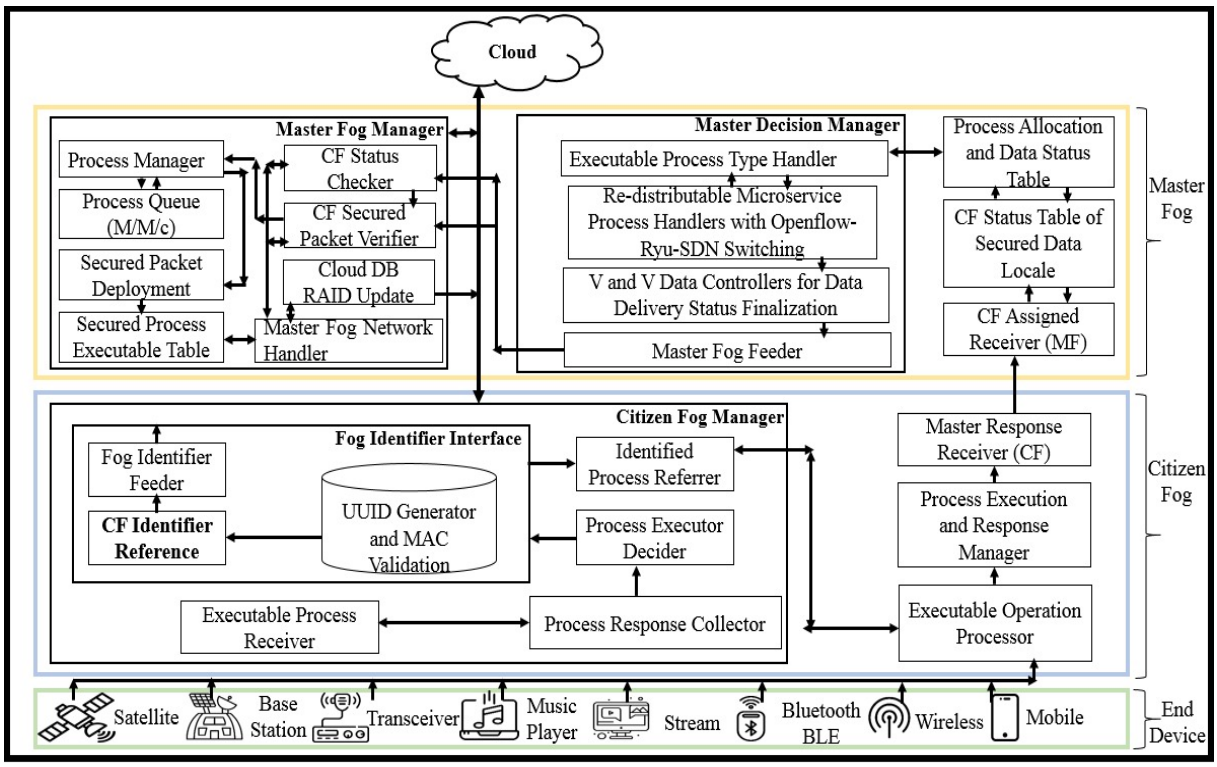

Figure 3: Framework

In the Cloud-fog-IoT continuum process arrivals (data in stack) occur at rate λ in an incremental phase from i to $i+1$ in N-dimensional space. where the service delivery through communication level supports finite mean μ remaining servers in idle states if process delivery contains fewer than c jobs. If exceeding c jobs the Fog level communication increments to a buffer level. Wehereas, c number of jobs are defined as a server communication stack interpreters around the 3 servers. The buffer is generically infinite but we controlled it in a finite level of 2 ms, hence, the communication has a sufficient data delivery level according to server availability, where the birth and death process flow contains for N-dimensional space that equals

$$Q = [\ (-\lambda \quad 2\lambda \quad 4\lambda \quad 8\lambda \quad \cdots \quad \lambda \quad \lambda$$
$$\mu \quad -(\mu+\lambda) \quad -(2\mu+\lambda) \quad -(3\mu+\lambda) \quad \lambda$$
$$2\mu \quad 3\mu \quad 4\mu$$
$$\lambda \quad \lambda$$
$$-(c\mu+\lambda) \quad -(c\mu+\lambda) \quad \lambda\ )]$$

This process queue in Fog-Cloud-IoT generalizes and continues state space N = [0,1,2,3...] and utilization of server $\rho = \lambda/(c\mu)$ and works as a stable measure in the stack queue if ρ<1 satisfies.

4.2. Buffer Allocation in the Fog-Cloud-IoT continuum

Since we have tuned the architecture in 3-tier distribution of N-dimensional continuous process delivery in periodic sessions, these data queues occupied with traffic intensity and buffer allocated delivery is needed to be configured for easy access and cumulative process executions. We have selected the bound for a 3-way server utilization scheme (i.e.: Fog-Microservice, Cloud, IoT). The generalized bound server utilization in Fog-Microservice, $\rho = \lambda/(c\mu) < 1$, then the system delivers stationery process execution of steady-state data delivery at individual cumulative service rate. Where, Π_0 defines a steady service rate and c supports 3-way exchange services from servers.

$$\Pi_0 = 1/\left[\left(\sum_{n-0}^{c-1} c*\rho)^n/n! \quad + (c*\rho)^c/c! \quad + 1/1-\rho\right)\right]$$

Where $\Pi_n = \left[\Pi_0 = (c*\rho)^n/n!, if 0 < n < c\ \right]$
$\Pi_n = \left[\Pi_i = (c*\rho)^n/c! * (c-n), if c \leq n\ \right]$
Here the average number of data buffers in the system queue delivers ρ/ 1 - ρ * C(c, λ/μ) + c *ρ

4.3. Data Delivery Entropy Threshold

Let N be exponentially distributed through random variable x with arrival rate parameter λ, then Probability Density Function is $f(x) = \lambda e^{-\lambda x}, where\ x \geq 0$.

Since, our arrival rate is exponential and poisson process accumulated procedure hence the entropy is

differential. Therefore,

$$\begin{aligned} \mathrm{h}_e(N) &= -\int_0^n [\lambda e^{-\lambda x} log(\lambda e^{-\lambda x})] dx \\ &= -(\int_0^n (log\lambda)\lambda e^{-\lambda x} dx + \int_0^n (-\lambda x)\lambda e^{-\lambda x} dx \\ &= -log\lambda \int_0^n f(x) dx + \lambda E[x] \\ &= -(log\lambda + 1) \end{aligned}$$

Here, λ = mean arrival rate in poisson
μ = mean service rate
Traffic intensity $\rho = \lambda/\mu$
Expected number of data deliveries in system (length) $L_s = \lambda/\mu - \lambda$
Expected queue length $L_q = L_s - \lambda/\mu = \lambda^2/\mu(\mu - \lambda)$
Expected data waiting in queue $W_q = \lambda/\mu(\mu - \lambda)$
Expected data stack in system $W_s = W_q + 1/\mu = 1/\mu - \lambda$
Data need to wait in queue $(w/w > 0) = 1/\mu - \lambda = 1/\mu(1 - \rho)$

$$\int (\int_0^\infty \sum_{i=1}^{n} R_c)\, dt = \int (\int_0^\infty \sum_{i=1}^{n} C_{cc} + \int_0^\infty \sum_{i=1}^{n} F_{pd})\, dt \quad (1)$$

$$\int (\int_0^\infty \sum_{i=1}^{n} R_d)\, dt = \int (\int \sum_{i\to 1}^{i\to n} {}_{cd} + \int \sum_{i\to 1}^{i\to n} P_i)\, dt \quad (2)$$

B: Algorithmic Steps

- C:Experimental Setup

C: Implementation

ARP Poisoning:

here

Shows the gateway ip, also compares gateway ip after the reverse networking operation.

Sniff traffic:

The network capturing is also stopped until all the ips has been processed for detection measure.

DHCP :

C:Experimental Results:

WIRESHARK:

5. Performance Evaluation

6. Discussion

In our experiment we have created the master fog by creating AWS EC2 instance on which we set up ubuntu operating system. On the ubuntu operating system we installed Apache server from which we transferred data as rest to the citizen fog. Here data flow between master fog to all three citizen is taking place. The configuration is created by installing docker and postgre sql database inside it. The data from postgre sql is fetched by using php and sql query . Later on these data are used as rest api in the citizen fog to collect and operate communication through http protocol. Inside htdocs we created a index.php file in which we run fetch the data from database. Then we convert the data in rest api by using php get request. When citizen fog is attacked using ARP spoofing the system than flooded and jammed in existing network . In this way linux kernel system got busy and could not handle adjacent request due to enormous communication handling issue generated by anonymous users. But this problem need to define for smooth communication in any backend based architecture and plateform. To get rid of these problems we have used SDN based open flow litepd ryu controller for switching the inline and outline communication. Inline communication primary works for the established data processing in a network architecture and outline communication refers to the newly generated data process accumulated from outside the network. We have traced both the inline and outline communication through Wireshark modular activity specifically the ongoing attack scenario, accumulated attack scenario and after attack scenario, we have also checked the network performance parameter simplifying latency delay packet drop count runtime activity frame generation, packet delivery and network delay. Latency delay and packet drop counts around the end-to-end gateway. The performances are checked towards the consideration off end to end gateway coupled hope to hope count. In these fog layer around the 3 tier architecture middleware plays a vital role to develop and establish a communication around the network background. In three tier architecture the network efficiency counts through middleware communication placement and delivery opportunities. These middleware need to keep free from anonymous chaos creation and data offload for enhancing network efficiency in terms of exaggerate issues of data offload and mismanagement middleware plays an efficient role so do the fog layer. Fog layer inherits the cloud attributes to maximize the scaling efficiency for service deliveries when erroneous and enormous process handling p needs to sum up from the fog and anonymous end user request support. In dealing with enormous request coming from the edge layer fog services specifically citizen fog process overflow occurs hence provides an extra load in jamming the process execution runtime of the master fog controllers. The anonymous request mainly counterfeits the master fog load balancer adopting Bogos information and

certainly interprets additional hashing misconception that arises identity disclosure problem for the connected user within the network. This misconception actually generated eves dropping or man in the middle attack feasibility within the network backbone. For a good and secure architecture an establish framework is essential to get rid of these problems .Hence thinking of these problematic scenario in network architecture we have established a microservice based SDN switching framework for time critical communication. Time critical communication actually refers when we need both of the data delivery in a socket based channel delivery when a network is under attack. Because if we do not considered the attack all of our valuable data in that time maybe lost due to data drop or network draining and the communication counterfeit may hampered us in predicting a stable dataflow around the network. Moreover, we may lose some of ours existing users process data when the network is in act or running in its shared communication pattern so in that case process optimization is also needed task to get rid from these time critical phenomena. In our framework microservice or docker platform in fog layer supports the time critical data delivery smoothly by switching the attacked communication in different level container based on user priorities. The user priorities actually are priority schedulers that optimize the conducted process within the network and tracks the outside process to get the essential information for that time. The optimizers helps us to augment the data delivery in these attack situation to keep the stability within the network. The attacked timing information directly moved to define containers in microservices to tackle the spoofed node and save the other active feasible nodes around the networks. These container data sorts out the data packet from the eves dropper or spoofed identifier and delivers the main microservice to keep track the eves dropper to shut down the communication from fog to edges around the hope to hope. The primary microservice sends out the information to cloud backend for establishing a process lock form. The cloud then track and verify the spoofs identity of the device and data packets through its data center and delivers a shutdown confirmation towards the fog microservice. The clouds verify and validate the spoofed identifier and completely block out the ip information around the network within administrative access permission. The primary fog microservice get the administrative access permission from the cloud backend and generates a log file for future disclosure to eradicate or ban these user from the existing network. After the validation confirmation from the cloud the fog microservice releases the information from the container having attacked data packet and keeps it free for future reference. After while the primary microservice checks the existing available containers for the registered active user data packets mismatched withing attacked data packets for an efficient delivery around the network. If there is existing attacked data mixed with register user data then again the attacked packet will be delivered within the define container deal with attack packets. After the attack packet sorting registered user packet are delivered to the respective containers for the effective channel to channel communication. In other words after the verification from the cloud if the primary fog microservice does not find any attacked packet mixed with the registered user packet the microservice trace the define container for the channel based packet allocation and aligned them based on assigned users for an effective data delivery across the channel to channel end . The define (attacked and registered based) microservice data delivery primarily works using preemption schedulers to track the data packet and linearized them. These preemption pipeline works through microservice customized pipeline chaining according to process bar size. Bar size specifically works in that particular time and if the process again needs to work or complete within the same time period in a different data slot these bar size problem in scheduling occupies or added additional delay to the existing timeframe hence we face delay augmentation in a communication channel. Our pipeline based scheduling in microservices uses sorted bar size of pivot element which successfully detects the process sizes to deliver early or in an ascending order or lately. Our modified pipelining scheduling process in microservice uses the quick sort searching strategy for an effective bar size calculation and early data delivery within the end to end network. These strategy suppresses the existing network delay issue and reduce the delay in a minimal tolerance having a highly supportive backend with an elastic data delivery support. These effective process delivery changes the define microservice container from one channel to another channel using microservice migration support. These migration support usually helps to eradicate channel overflow delay when the active user channel remain busy. The overall migration support and effective highly pipeline support defines our framework as a particular suitable mechanism for an elastic cloud fog IOT backend.

7. Conclusion

In this paper, we presented a three-layer framework to enhance resource utilization in a fog computing paradigm. Our SDFC framework includes a Master Fog layer that resides in between the general-purpose fog nodes called Citizen Fogs and the Cloud. The Master Fog is dedicated to bear all the computing overheads to schedule the tasks and allocating the highest priority task to the most eligible CF immediately. The Master Fog uses the CAA algorithm to

schedule the tasks based on their priority and the LASA algorithm to sort the citizen fogs according to their available computing ability. Moreover, SDFC assures that the CF layer's resources are utilized adequately, hence reducing the dependency on the cloud. Consequently, not only the task execution time decreases but also the delay in real-time based services reduces. We validated the impact of our SDFC framework and integrated algorithms by simulating in iFogSim. We found that the dependency on cloud lessens by $15\% - 20\%$, and the total execution time reduces by $45\% - 50\%$. This reduction in execution time and cloud dependency is implicitly decreasing delays in real-time services.

Acknowledgements

The authors have declared no conflict of interest. The authors acknowledge that this research is supported through the Australian Research Council Discovery Project: DP190100314, 'Re-Engineering Enterprise Systems for Microservices in the Cloud'.

References

Algorithm 1: Master Fog Panel Process Allocation Fog-Cloud-Microservice-end(regular and irregular [i.e.: intruder] data execution)

Input: SDN Switching and Sorting Stack Generation From 1 to n
Output: Identify Spoofing Stacks and Execute Microservice Configured Schedulers

```
/* isEntered = A boolean value defines whether
   it is an regular or intruder scenario.      */
/* StackList = Sorted Priority Scheduled
   Ascending Stack List                        */
/* Stack = A Particular Renewal Stack From the
   StackList                                   */
```

foreach *Stack in StackList* **do**
  Step 1: Actor: *Master Decision Manager*
    $StackList \leftarrow$ $Stack$.getDataStatusTableInfo()
    if *Stack is NOT from CFStatusTable() AND StackList () Process allocation with verification is failed* **then**
      $isEntered \leftarrow$ true
      $n \leftarrow StackList$.size()
      $Stack$.number $\leftarrow n + 1$
      Send the $Stack$ To CitizenFogManager()
    else
      $isEntered \leftarrow$ false
      Check For Verified data with Captured Location info of the $Stack$
      Assign Priority Scheduler to Insert
      Validated data to MasterFogFeeder()
  if $isEntered$ **then**
    Step 2: Actor: *Master Decision Manager*
      Regenerate Stack Block with Indexing Overlaps
      $n_{cs} \leftarrow$ current Indexed executing Stack number traverse To (n_{cs})
      $Stack$.number $\leftarrow n_{cs}$+1
      Increment Other Stack Numbers By One
  else
    Step 2: Actor: *Master Decision Manager*
      Assign and Deliver New Stacks To the CF Status Checker of MasterFogManager()
    Step 3: Actor: *MasterFogManager*
      if *CF Secured Packet is Verified* **then**
        Continue AND Response to MasterFogNetworkHandler()
        Get ProcessQueueUpdate() From SecuredPacketVerifier()
        AND SceduledProcessManager()
        Compile SecuredPacketDeploy() AND ExecutableProcessTable()For CloudWatchDBUpdate()

Algorithm 2: Citizen Fog Panel Process Allocation Fog-Microservice-IoT-end(regular and irregular [i.e.: intruder] data execution)

Input: SDN Switching and Sorting Stack Generation From 1 to n
Output: Identify Spoofing Stacks and Execute Microservice Configured Schedulers
/* $isEntered$ = A boolean value defines whether it is an **regular** or **intruder** scenario. */
/* $StackList$ = Sorted Priority Scheduled Ascending Stack List */
/* $Stack$ = A Particular Renewal Stack From the $StackList$ */

foreach *Stack in StackList* **do**
  Step 1: Actor: *Fog Identifier Interface*
    $StackList \leftarrow$ $Stack$.getDataStatusTableInfo()
    if *Stack is NOT from MAC Validation() AND StackList() Process allocation with ProRef() is failed* **then**
      $isEntered \leftarrow$ true
      $n \leftarrow StackList$.size()
      $Stack$.number $\leftarrow n + 1$
      Send the $Stack$ To CitizenFogManager()
  else
    $isEntered \leftarrow$ false
    Check For Verified data with Captured Location index of the $Stack$
    Assign Priority Sorted Scheduler to Insert Validated data in ExecOpProcessor()
if $isEntered$ **then**
  Step 2: Actor: *Citizen Fog Manager*
    Regenerate Stack Block with Indexing Overlaps
    $n_{cs} \leftarrow$ current Indexed executing Stack number traverse To (n_{cs})
    $Stack$.number $\leftarrow n_{cs}+1$
    Increment Other Stack Numbers By One
    AND Get ProcRef() data Deliver to MasterResRec()
else
  Step 2: Actor: *Master Response Receiver*
    Assign and Deliver New Stacks To the CF Assigned Receiver
    Get Verified DataStatusLocale() to secure Process Allocation
  Step 3: Actor: *Process Allocation Table*
    if *CF Assigned Secured Packet Receiver is Verified* **then**
      Get CFStatusTableUpdate() From CFAssignRec() AND DataStatusTable()
      Compile ExecProTypeHandler() For ExecMasterFogFeed()

Table 1
System descriptions for simulation environments

System properties	Specification
Operating System	Windows 10
Processor	i3-4100M, CPU 2.50GHz
Storage	RAM 16 GB, SSD 256 GB
Number of Cores	4
System Architecture	64 bit, x64
Simulator	iFogSim
Programming Language	Java
IDE	Intellije IDEA 2019.2.4

Table 2
Citizen Fog and Master Fog Resource and System Specification

Attribute	Specification
Number of CF under a MF	5
MIPS	1024 * Random(8, 12)
RAM	512 * Random(1, 3)
UpBW	10000
DownBW	270
Busy Power	87.59
Idle Power	82.44
Number Of PE	Random(1, 3)
Storage	1000000
Bandwidth	1024 * Random(8, 12)
Architecture	x86
OS	Linux
VMM	xen

Table 3
Required Resource specification of Task

Attribute	Specification
Number of Task	500
RAM	32
MIPS	1024
TaskLength	1024
Bandwidth	128