A Trust Management Framework for Software Defined Network (SDN) Controller and Network Applications

Aliyu Lawal Aliyu, Adel Aneiba, Mohamed Patwary & Peter Bull
School of Computing & Digital Technology
Birmingham City University
Email: aliyu.lawalaliyu@mail.bcu.ac.uk

***Abstract*—The use of network applications to manage network operations by the controller in SDN architecture introduces a threat that makes the controller to be susceptible to several network attacks. This is possible because the network applications operate without any access control mechanism that authenticates or dictates what operations they can execute in the network. This consequently makes the network applications to take advantage of their ability to manipulate, change or modify network state to compromise network operations and resources. In order to address this problem this paper introduces a *token-based authentication* method that enables the controller to authenticate the various network applications. The application of this method builds an access permission zone where only legitimate network applications with the correct token credentials can have access to the network prior to implementing any network changes. This paper contributes in providing an authorisation method *Boolean Access Matrix* that enforces permission constraints on what the network applications can access or execute within the network. The authorisation method helps limits the unprecedented access the network applications have over the control layer resources, core services and the network operations. The paper introduces a novel method of evaluating the trust between the controller and the network application based on *Subjective Logic Reasoning* (SLR) which is a belief learning model. SLR is an advanced learning algorithm that is derived from Probability Calculus and Statistics. Experiments demonstrate the efficiency and scalability of the proposed algorithms in a large scale test environment.**

Keywords: SDN, Trust, Authentication, Authorisation, Security.

I. Introduction

Software Defined Networking (SDN) is an emerging paradigm that changes the way networks are managed by separating the control plane from the data plane and making networks programmable. The separation brings about flexibility, automation, orchestration and offers savings in both capital and operational expenditure [1].

The main problem of SDN stems from the benefits it provides [2]. Thus network programmability and innovation at the application layer. These introduce new faults and attack plane which consequently give way to new threats that were not present before or difficult to exploit [3]. These new threats are discussed in [4], where the identified severe threat among them is the trust between the controller and the network applications. In SDN environment, the controller is logically centralised and have the global view of the network state, these include topology, connected hosts, forwarding devices and links. The network applications request information about the network state and view from the logically centralised controller and transmit commands that dictates the forwarding behaviour of the data plane.

This form of communication between the controller and the network application is perceived as threat from a network security perspective. For example a network application with a malicious intent can take advantage of the global network state information to manipulate traffic flow based on the motive of the adversary. In addition poor application design can introduce security vulnerabilities that can otherwise compromise network operation.

The SDN paradigm supports third party development efforts and suffers from trust issues on deployed network applications. The violation of trust can leads to different types of attacks, where the consequences are severe and have impact on the entire network [5]. Therefore trust violation is a threat and resides between the controller and the application. SDN networks are programmed with policies that explicitly allow network applications to apply changes in the network and no formal verification technique or semantics to assess the trust of these applications [6][7]. Malicious applications can abuse these privileges and harm network operation.

When network applications are instantiated with the controller, they automatically inherit all the access rights to change, manipulate or modify network state [8]. This is a serious threat that receives little attention and sparsely explored within literature. This threat from network applications arises from how to verify the trustworthiness and reliability of a program module, because network applications have access to critical network resources like flow table, device configurations, CPU memory, RAM partitions, input and output ports.

Most network administrators that use third party applications assume same trust level similar to that of the controller on the network applications [9]. This is because controller modules must undergo series of tests and verification to make sure that they are reliable and fit for use in research and production network. However it is difficult to ascertain the reliability and trustworthiness of a third party application [10]. A malicious or compromised application can be a sink for various network based or host based exploits. This can give way to control plane attacks that can lead to code execution or information disclosure.

The design of the SDN architecture is in such a way that application layer contains the various various network applications that modify, change and manipulate network operations [11]. Each application has a distinct logic and function for which it was designed for, example switching, routing, firewall, load balancing and so on [12]. SDN applications run on top of the controller, and communicate with the network via the Northbound Application Programmable Interface (API). The core functionality is to manage flows that are contained in the forwarding elements using the controller's API [13]. With the help of the API applications can:

- Set up flows to route packet via the optimal best path between two endpoints.
- Traffic aggregation and loadbalancing across multipath.
- React to topology dynamism in case of device failure, change in location, device join or leave event.
- To redirect traffic for security related functions like authentication, deep packet inspection, intrusion detection and segregation for further analysis.

Within the scope of our knowledge there is absence of an access control mechanism that verifies the interaction and association of network applications with the controller. With the controller having full knowledge of the network under control, if a malicious application takes over the controller, the result would be catastrophic. Some of the implications are redirection of sensitive traffic, controller spoofing, DoS attack, network reconnaissance attack, tampering with flow rules etc. There is large state space control delegated to the controller that its compromise could leads to hijacking of the entire network operation [14]. Every secure communication network should guarantee confidentiality, integrity, availability, authentication and non-repudiation between communicating entities. This cannot be achieved without having concrete threat mitigation techniques in SDN architecture.

This paper contributes in providing a novel Trust Management Framework that:

- Authenticates the various network applications that are situated in the network function virtualisation layer.
- Apply authorisation privileges and constraint network application behaviour to conform with set network policy.
- Evaluate the trust relationship based on *Subjective Logic Reasoning* between the network applications and the SDN controller.

The mitigation of this threat that exist between the controller and the network application will be the primary focus of this work and this will aid towards building a secure and dependable trust framework for the SDN architecture. SDN is a new network paradigm at an early stage with the potential of becoming the next generation communication network, with this in mind there is growing need to have a dependable and reliabe architecture [15].

The rest of the paper is organised as follows: Section II presents the problem statement and motivation as to why there is need to mitigate the devastating threat between the controller and the network application. Section III presents related work on controller to network application security. Section IV presents the proposed design of the trust management framework and Section V provides detailed implementation, testing and evaluation of the framework. Section VI provides a comparative evaluation and validation with other frameworks that attempt to mitigate similar threat. Finally, section VII concludes and discusses future directions.

II. PROBLEM STATEMENT

The interaction between controller and the network applications takes place between the northbound interface [7]. Based on SDN secure design architecture, this interface should allow only trusted network applications to program and request services from the network [16]. The network interaction and the implementation of network changes is enabled by access permissions that give the network application ability to ***read, write,*** receive ***notification*** and make ***system calls*** to control network state [17].

The network applications interact with the shared SDN control plane state through service API calls and event callbacks. A network application can read from and write to one of the controller modules (e.g network information base) via a particular service and the service method [18]. For the event callback, a network application can register with the controller and subscribe to events of interest. In essence, the interaction is about control which involves reading network state, writing network policies, getting notification of events and access to resources via system calls.

A. Read Network State

The communication between the controller and the network application is implemented either via Representational Stateful Transfer (REST) APIs or the native controller API [8]. For example, java based controllers like OpenDayLight and Floodlight can use both native java based API or REST-APIs to enable interaction of network applications with the controller [14]. For a network application to read network state it can send a HTTP GET request to pull for example state information of the network topology or device configurations. The controller serves as an intermediary to relay the request by the network application to the forwarding devices at the data plane via any available southbound API majorly OpenFlow [19]. The forwarding devices at the data plane will then respond with the requested data which the controller interprets and forward back to the network application in form of a HTTP response. The query follows a similar method if the native controller API is used instead of REST API request.

B. Write Network Policy

In order to implement or effect a change in the network state, a HTTP POST request is sent by the network application to the controller. The controller serving as a proxy would translate the request as a flow modification or packet out instruction to be implemented in the flow table of the relevant switch in the data plane via the southbound API [20]. The network state change implemented by the HTTP POST method is been enabled by the *write* access permission [21]. Some functions allowed by the write access permission include but not limited to adding of flow rules, modifying flow rules, sending flows out to a designated port, assigning VLAN tags, flow queues, metering and deleting of flows. The ability to write network policies that can affect the global state of the network gives the network application an edge in terms of network control and management. When configuring an application, it is necessary to know precisely which access right is needed for the network application to function correctly without impacting the security and network operation [22].

C. Notify

Network application can subscribe to certain events in the network and receive notifications[17]. There is no limit to the number of events a network application can subscribe to and application can subscribe to as many events as necessary to accomplish its designated tasks [23] Example of some of the notification events include a switch join or leave event, switch port status (up or down state), link status etc. A network application can join or subscribe to network event of interest, a malicious network can subscribe to event in order to listen, monitor and observe network activity for reconnaissance purposes.

D. System Call

Network applications can access the local operating system resources, these are resources that are shared across all applications resident on that local machine [16]. These shared resources are CPU, RAM, Storage, processor threads, file system and kernel libraries [24]. Access to operating system resources should be monitored because the controller sits on the abstraction provided by the operating system. The operating system contains the kernel base code where tampering of system properties by the network application can be critical to network operations and the controller at large.

Analysing how the interaction of the network application and the controller takes place there are identified vulnerabilities in this approach:

- Network applications are not required to provide any kind of identity or credentials before sending network request. There is no method to verify the authenticity of network applications [16][21].
- There is no method to monitor, constrain or limit the set of requests or operations being executed by network applications. In essence no regulation of behaviour after installations, and network application can access and make changes to network operations without any hindrance [4][23].

From this assessment, the potential solution will be to provide:

1) A method of identifying network applications.
2) A method to control the level of access a network application has when managing network states.
3) To provide a trusted communication between the controller and the network application.

With exposed network resources and APIs at the disposal of the network applications. Developers can design and distribute applications. However this flexibility as a way of innovation can be constrained if an adversary decides to launch a malicious application with rogue intentions [25]. Figure 1a shows the threat that leverages the inability of the controller to completely identify the various network applications that make changes to network in the SDN architecture.

Network applications that control network states have all the access permissions *read, write, notify* and *system call* at their disposal. If a third party network application developer decides to implement a malicious application that can leverage these access levels to disrupt network operation and compromise network assets then there is no access control mechanism in place to stop the malicious intent [8]. The breach would go undetected. For example Figure 1b presents how a malicious network application is been installed and initiated in the network.

The focus of this work is to presents a method of identifying the network applications via authentication, and providing an authorisation method that constrains the behaviour and level of access every network application has on the network. And the last part entails using a trust method to evaluate how the network application interacts with the controller.

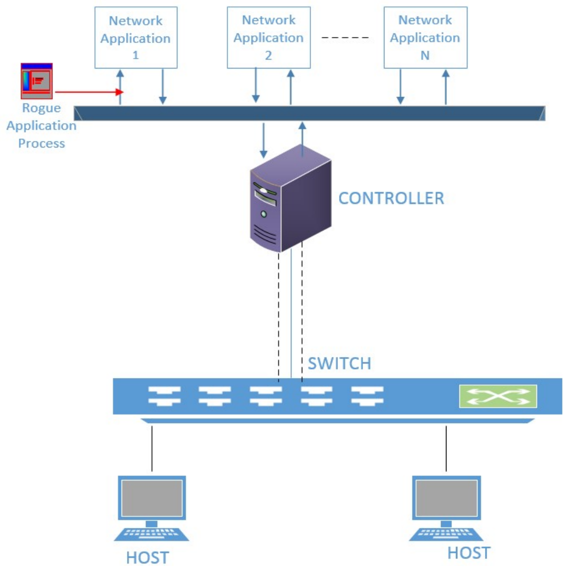
III. Related Work

This section analyses the previous work and attempts made by the research community to address this threat that exist between the controller and the network application. The threat is introduced due to the inability of the control layer to identify the network application and the level of authorisations they have when modifying network states.

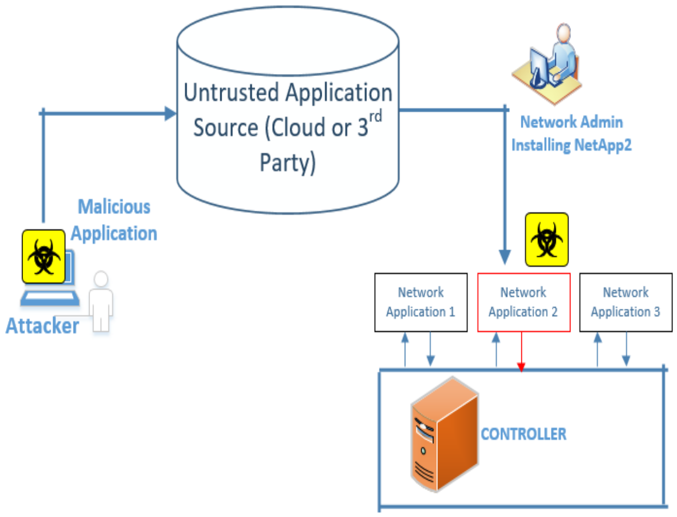
In [26] the authors propose FortNox which is one of the first attempt to implement an authentication mechanism for network applications within an SDN environment.The motivation of FortNox approach is based on flow-rule conflict and contradiction due to various network applications sending flow requests simultaneously to the SDN controller. The means of identifying a network application in FortNox is via generated flow rules, each flow rule is signed to make sure there is no overlap from a previous installed flow. The signing of the flow rule serves as the method of identifying the network application. FortNOX implements a real-time algorithm that detects and mediates rule conflict.

In FortNox, the authentication approach is to identify flows coming from various network applications in order to avoid rule overlap and rule bypass. FortNoX is not authenticating the network applications to be specific, however substantial amount of work has been done in identifying flows and implementing polices for granularity and control.

In [27] the authors propose Rosemary which is an SDN controller framework that is designed and centred on security. Rosemary builds upon the work conducted in FortNox because it is introduced by the same authors. Network application



(a) Application to Control layer threat



(b) Malicious Application Threat Scenario

Figure 1: Network threat and means of propagation in the SDN architecture.

identification is still via flow signing, however in this case, the authors introduce a method that authorises what network application can access or execute via system call access check module. The system call access check module runs in isolated containers called micro network operating system (microNOS) and intercept requests made by network application.

Rosemary aims at providing control layer resiliency in the event of a network application crashing due to instabilities or vulnerabilities. In some situations the failure of a network application may halt the operation of the control layer. Rosemary achieves this through application spawning that provides containment and resiliency. Rosemary does not provide trust between application and controller, but provides a means of threat containment in case of malicious application compromise. It has the capability to isolate the fault and keeps the network operating system functional.

PermOF is another approach proposed by authors in [17] that introduces a filtered fine grained authorisation permission system for network applications. They introduced a layer that restricts application to make direct calls to controller memory base. This layer is configured and managed by the controller so that unsolicited communications between network applications and the control layer is prevented.

PermOF makes a good proposal on authorisation strategy by providing a fine grained permission system that aid in authorising network applications. Though PermOF is not implemented and does have any authentication mechanism, but the authors put up a good authorisation mechanism to checkmate network application privileges and escalation of privileges.The approaches discussed do not provide a comprehensive framework that comprises of both authentication, authorisation and trust.

Another related attempt is our earlier work on trust between SDN controller and the network applications [28]. The research work identifies , highlights and give directions as to how the trust issues can be resolved between the controller and the network application. The aim is to address the vulnerability in the SDN architecture that exists when the network applications are interacting with the controller. The violation of trust by the network application can lead to different types of attacks and the consequences are severe and heavily impact the entire network operation.

The ability to segregate and isolate the different applications running on the controller in order to provide logical segmentation to support authentication of the applications and to enforce level of authorisation and privileges will be paramount to a secure and dependable control layer.

IV. Proposed Design of the Trust Management Framework

Considering the security problems discussed in Section II about the inability of the controller to identify network applications and authorise what operations they can execute in the network. This section presents the system design and the architecture of the trust management framework. The aim of the proposed trust management framework is to mitigate the identified security problem. The architecture is illustrated in Figure 2. Technically the framework is integrated as a module in the control layer. Matlab and Python scripts are used in developing the packages and incorporated directly at the control layer. RYU controller is used to demonstrate the feasibility of the proposed design.

The developed modules are compiled as part of the controller and have access to various controller classes, methods and data. The trust management framework contains various modules and submodules that interact with one another to provide a secure and trusted communication between the controller and the network application.

The system design of the trust management framework will satisfy the security requirement needed to mitigate the threat between the controller and the network application by:

1) Providing a method of identifying a network application via a credential.
2) Defining set of permission for network applications. The permission should include all application related task that are required for smooth operation of the network.
3) Evaluate the reliability and dependability of network applications based on the interaction with the controller.

The proposed framework aims at establishing a trusted relationship between the control layer and the network applications. Figure 2 shows the architecture of the framework and it is made up of three main blocks.

- Authentication Module
- Authorisation Module
- Trust Evaluation Module

Apart from the three main modules, there are however peripheral modules and interfaces that majorly make up the framework which include the controller core services modules. The framework involves implementing a controller with modules that assist in carrying out core functions of the proposed trust framework as seen in the architecture. The interfaces and modules are discussed as follows:

- **Application Interface**: Provides the chaining (service requests) between the controller and respective network applications.
- **Authentication Interface** -The interface that connects to the *authentication module*. It checks and verifies applications using *application instance* and a *unique token*.
- **Authorisation Interface** - The interface that connects to the *authorisation module* which applies permission constraints to authenticated applications.
- **Trust Interface** - The interface that connects to the *trust evaluation module* it aids in calculating the confidence and reliability of a network application.
- **IP** - The IP address of the controller for reachability .
- **Location** - The controller location whether remote, local or on the cloud.
- **Identity** - Identifies the controller in a cluster of controllers.
- **Event handler** - Handlers that listen for event and react based on the defined logic to carry out specific task.

This section provides detailed design and development of the main blocks starting with *authentication, authorisation* then *trust evaluation* module.

A. *Authentication Module*

This is one of the main modules of the framework, the authentication module checks, verifies and allows or blocks SDN applications on the grounds that they pass the network application credential checks. A unique identifier in terms of *Token* is assigned to every network application, which is used in the application permission management at the authorisation stage.

The authentication process is the first phase in the framework and is triggered when an application initiates a request to implement network changes via the control plane. Each instance of application that runs have a unique token, the token serves as basis of checks and verification. The network application in this context is a *view*, the view is the context through which the application in conjunction with the token is seen by the framework during authentication. This means that there is no physical exchange of credentials because the application is not a physical object like a user with credential but rather an automated batch process that is abstract and verified before execution of its task. Authentication always runs at the beginning before permission check for authorisation and before any other code is allowed to execute. The *Authentication Module* has two main functional blocks as seen from the architecture on Figure 2 that helps in successful execution of authentication, they are *Application Correctness* and *Token derivation*. These functional modules are analysed as follows:

1) ***Application Correctness and Verification****:* Checks for correctness and verification to make sure the authentication process goes smoothly. This is carried out by the *Application correctness* and *verification* block. The authentication module takes in two arguments, the first being the network application and the second argument is the token. This sets the stage for application verification and makes sure errors related to the authentication are addressed.

To demonstrate the feasibility of the concept two applications are used mainly:

- *Layer 2 MAC learning Application (L2 MAC)*
- *IP Blacklist Application*

These two applications will serve as the centre of the analysis and evaluation of the framework. A database is created for these applications where *App1* is mapped to L2 MAC and *App2* mapped to the the blacklist IP application together with their respective token.

$$Application\_database = \{App1,\ \ App2,\ ...\ App\_n\ \} \qquad (1)$$

Where $App\_n \in Application,\ and\ n \in \mathbb{N}$

$$Function\_desc = \{L2\_Mac,\ \ IP\_Blacklist,\ ...\ X\ \} \qquad (2)$$

Where description $X \leftarrow APPn$ and, $X \in Function\_desc$

$$Token = \{Token1,\ \ Token2,\ ...\ Token\_n\} \qquad (3)$$

Where $Token\_n \leftarrow APPn$ token and $n \in$ N

After the initiation the next step is to commence the verification where the authentication procedure *AuthenticateApp()* checks for incoming challenge request from network application.

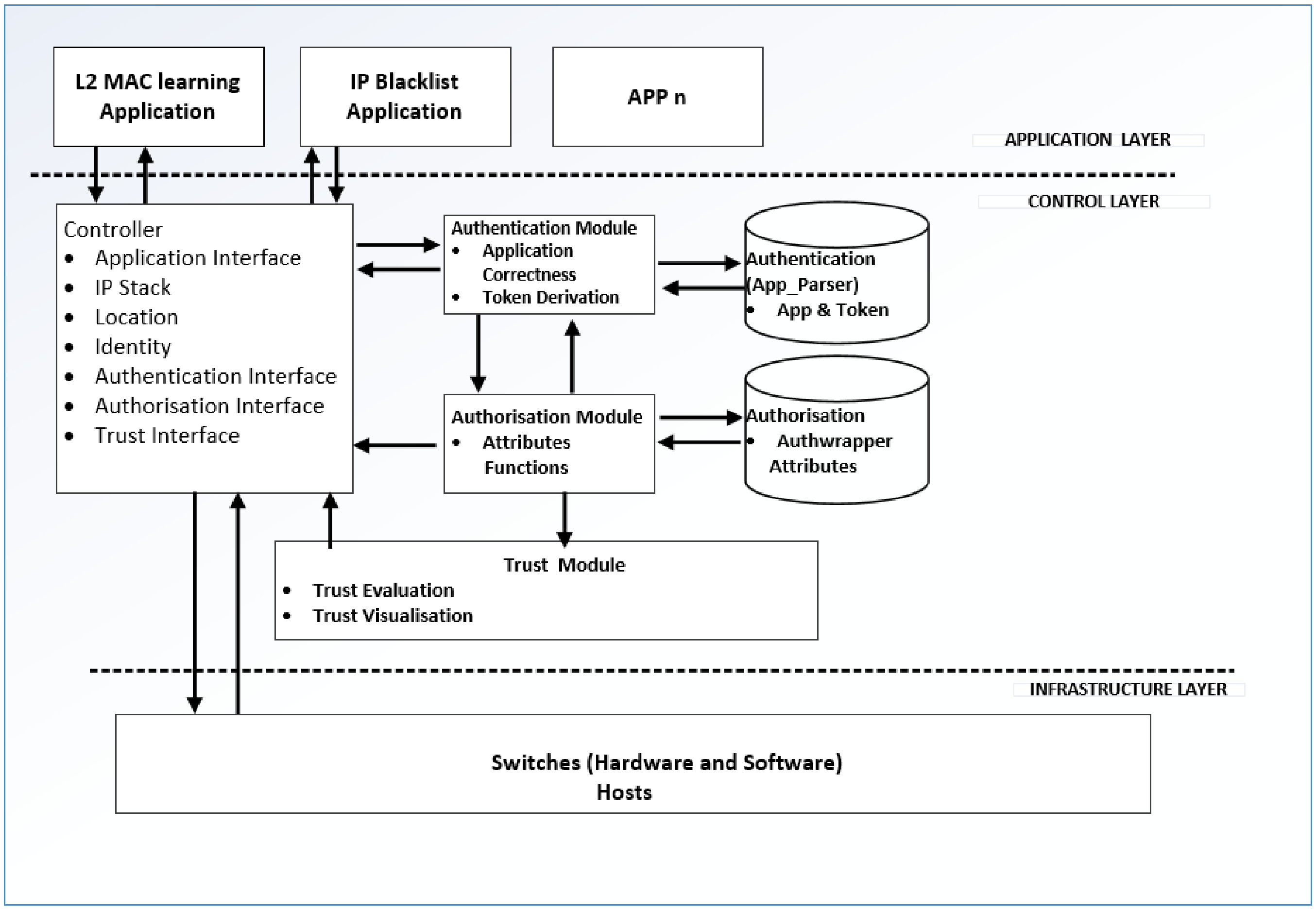


Fig. 2: Trust Management Framework

$$AuthenticateApp(arg\_app,\ token),$$

arg_app $\leftarrow$ Network applications as argument.
token $\leftarrow$ Token for the network application.

If there is a call to the *AuthenticateApp* module without any apparent argument. Thus if $nargin = \varnothing$, where *nargin* means number of arguments input, then the resulting output is to provide *more_arguments*.

To elaborate on the rest of the application correctness algorithm, the following conditionals are set:

1) ψ_1 = If *naragin* is numeric
2) ω_1 = Then error authentication failure resulting from condition ψ_1
3) ψ_2 = If the input is *arg_app_1*, where *arg_app* $\in$ Application
4) ω_2 = Then incomplete authentication, token required from condition ψ_2
5) ψ_3 = If the input is *arg_app_2*, where *arg_app* $\in$ Application
6) ω_3 = Incomplete authentication, token required from condition ψ_3

Therefore if $nargrin = 1$,

$$(\psi_1 \rightarrow \omega_1)\ \wedge\ (\psi_2 \rightarrow \omega_2)\ \wedge\ (\psi_3 \rightarrow \omega_3)$$

For the last part where *AuthenticateApp* takes in application that is not contained in the library of defined network applications thus (arg_app $\notin$ Application) then the resulting output should be:

7) ψ_4 = *arg_app*, where *arg_app* $\notin$ Application
8) ω_4 = Wrong application, valid application required from condition ψ_4

$$(\psi_4 \rightarrow \omega_4)$$

The token is obtained by set of defined algorithm that will be discussed in next section IV-A2. A snippet from the pseudo code of the Network application correctness and verification can be seen in Algorithm 1.

*2) **Token Derivation**:* Conventional methods mostly use heavy encryption ciphers that delay the process token exchange to the extent of timing out, to avoid such issues of downtime an efficient method with less cycles of processing is presented and implemented to achieve a successful verification of network applications [29]. Another challenge is the security

Algorithm 1 Network Application Correctness

```
1: Initialising checks, verification and correctness of the Application
2: Procedure **AuthenticateApp**(*Application, Token*)
3: $Application = \{APP1,\ APP2,\ \}$
4: $Function\_desc = \{L2\_MAC,\ IP\_BLACKLIST\}$
5: $Token = \{Token1,\ Token2\}$
6: $Application \leftarrow Application\_view$, Where $Application\_view$ is a process
7: $Token$ as application key of length(n), $keyspace(n) = 20$
8: if $(nargin == \varnothing)$ then
9:     display('More arguments required')
10:    EXIT
11: else if $(nargin == 1)$ then
12:    if isnumeric($arg\_app$) then
13:        $arg\_app \in \mathbb{N} \cap \mathbb{Z}$
14:        Action:= Deny
15:        error('Authentication_Failure')
16:        Controller_info('Application_view $\notin \mathbb{N} \cap \mathbb{Z}$ ')
17:    else if $App\_parser(App\_view,\ arg\_app)$ then
18:        $arg\_app \in Application$
19:        $arg\_app = APP1 \cup APP2$
20:        display('Incomplete Authentication, token required')
21:        error('Auth_Failure')
22:        Controller_info('$arg\_app$ and $token$ expected')
23:    else if $App\_parser(App\_view,\ arg\_app)$ then
24:        $arg\_app \notin Application$
25:        display('Application input expected')
26:        error('Auth_Failure')
27:        Controller_info('Input not allowed)
28:    end if
29: end if
    =0
```

issues of using some encryption ciphers. For example in some versions of SSL/TLS there are fatal vulnerabilities that exist like the POODLE [30] and BEAST [31] exploit. These exploits can decipher messages in transit and recover session information from a secure connection.

In this research work the token is unique to every network application and should change after a certain defined period. These tokens are generated locally and play a local significance in the ecosystem of the architecture. They cannot be exported nor used twice for different application. Figure 3 shows a process of token exchange between the controller and the network application.

The initiation of the token involves permutation, combination and randomisation of alphanumeric variables and set of natural numbers with defined boundaries. They have certain length which define their key space and strength against brute force. The initiation is as follows:

$$A = \{a,\ b,\ c,\ d,\ e\ .....\ z\}$$

$$B = \{0,\ 1,\ 2,\ 3,.....9\}$$

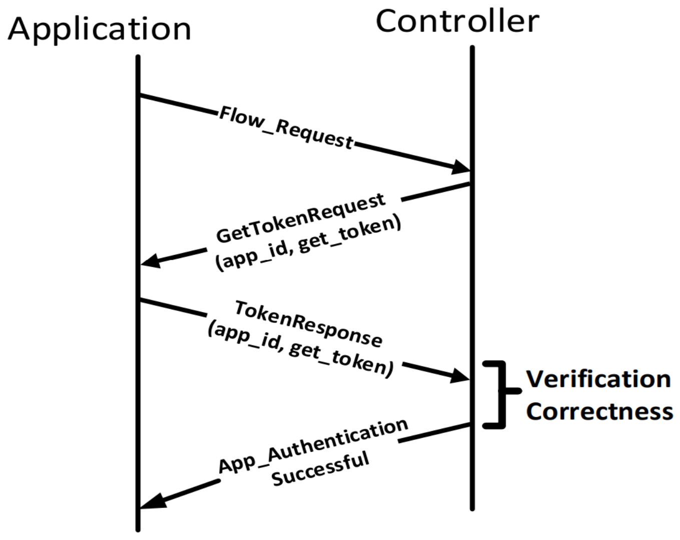

Application
Controller
Flow_Request
GetTokenRequest (app_id, get_token)
TokenResponse (app_id, get_token)
Verification Correctness
App_Authentication Successful

Fig. 3: Authentication Process

Therefore AB,

$$AB = \{x|\ x\ \subseteq (A\ \wedge\ B)\ \wedge\ x\ \in \mathbb{N}\ , 0 < x < 10\}$$

The set *AB* contains union of both set of *A* and *B*, these will serve as the input of the next stage of the token derivation. With the result of *AB*, a function *Zeroes* that increases entropy of randomisation will be used in conjunction with the length of *AB* .

$$AB = \{A\ \cup\ B\}$$

The function *Zeroes* takes two arguments *(M,D)*. The value for *D* is arbitrary chosen to be 20, for robustness and resiliency the higher the better. The defined variables:

- M = length of AB
- D = key size which is 20

$$Zeros(M,\ D)$$

The algorithm is built in stages with each output laying as the foundation of the next input function. The result of *Zeroes* is assigned as *Data*.

$$Data = Zeroes(length(AB),\ 20)$$

Data will serve as the upper boundary of the next procedure which is *size*. The lower boundary will be a chosen integer that fits within the required expected output and in this scenario 2 is arbitrarily selected.

$$Size(Data,\ 2) \tag{4}$$

For every output from equation 4, initiate a loop starting from integer *1*. The output serves as our Token pass which can be mapped to *application view* for authentication purpose.

$$\sum_{i=1}^{Size(Data,2)} randperm(length(AB))$$

The pseudo code for the token derivation is seen in Algorithm 2.

Algorithm 2 Token Derivation for Network Application

```
1: Declaring and instantiating variables for TOKEN key generation
2: A = $\{a, b, c, d, e ..... z\}$
3: B = $\{0, 1, 2, 3,.....9\}$
4: $AB = \{x| \ x \ \subseteq A \ \cap \ x \ \in \mathbb{N} \ , 0 < x < 10\}$
5: $AB = \{A \ \cup \ B\}$
6: Procedures **Zeros**(*M, D*)
7: *M* = length(*AB*)
8: *D* = 20, key space length
9: Data = Zeros(*length(AB), D*)
10: For loop initiating, Lowerbound = *1* and Upperbound = *Size(Data,2)*
11: Size(Data, 2) is a function and the (*Data,2*) are the input vectors
12: ∀ k(*Lowerlimit, Upperlimit*) **do**
13: Output = **randperm**(*Length(AB)*)
14: Token1 = {*ZPD5H580XJWLZ1RKDOE7' ,*}
15: Token2 = { *'HNHAKGA2ZMRDQCOH9JRV*}
16: **END**
    =0
```

The token to be used for the two applications respectively (L2 MAC and IP blacklist) are seen on Table I as the output from Algorithm 2.

TABLE I: Token for Applications

Application	Token
L2 MAC	*ZPD5H580XJWLZ1RKDOE7*
IP Blacklist	*HNHAKGA2ZMRDQCOH9JRV*

B. Authorisation Module

This module is very crucial in the evaluation and realisation of the framework. It serves as the second stage after which successfully authenticated applications will be assigned authorisation privileges. Authorisation deals with the specific permissions the network application is allowed to execute on the network after successful authentication.

These permissions are derived from the nature and behaviour of the application. What builds an application is the list of attribute actions in terms of **permission** it can execute within the network operating system.

The set of these permissions vary from one application to another based on the functionality and behaviour of the application. For example some identified attributes of a load balancing application are seen in Table II. From the list of the attributes of a network application, a wrapper is applied on the attributes via which access control and authorisation can be enforced.

Attributes	Function
Read_Buffers	This is an attribute that read buffers of ports and interfaces. Based on predefined limit and capacity a buffer can store packet half or full packet payload in a queue.
Read_Port	To map out connected ports of a data plane element, whether physical or virtual.
Select_Route	Choose the optimal route for traffic based on preconfigured policy. The procedure can be based on link saturation or to distribute traffic loads evenly across the different path.
Flow_Count	Take a record of total flows received or sent in a flowtable.

TABLE II: Load Balancing Attributes

In the authorisation stage the set of permissions for every network application are defined. Each network application has a unique identifier which is used to bind the related permissions of that application. A database structure uses the unique identifier to store the permission of every network application. The identifier can be used as a key to access all the permission of that application in the database. These permissions are made up of the related commands used by a network application to read state of network, write network policy, register for events or make system calls.

The actualise this concept, the authorisation model is made up of :

- Applications = $\{ App1, \ App2, \ App3 \ .... \ App_n\}$
- Objects = $\{ \ o_1, \ o_2, \ o_3, \ o_4 \ .... \ o_n\}$
- When interacting with network states, network applications make changes to objects. These objects are mapped to various instances of read, write, notify and system call objects. The level of access to these objects depends on the privileges of the network application.
- Set of Read permission $R = \{ \ r_1, \ r_2, \ r_3, \ .... \ r_n\}$
- Set of Write permission $W = \{ \ w_1, \ w_2, \ w_3, \ .... \ w_n\}$
- Set of Notify permission $N = \{ \ n_1, \ n_2, \ n_3, \ .... \ n_n\}$
- Set of System call permission $S = \{ \ s_1, \ s_2, \ s_3, \ .... \ s_n\}$
- Union of all permissions $P = \{ \ R \ \cup \ W \ \cup \ N\cup \ S\}$
- Example of an application with combinations of permission *APP1* = $\{ \ r_1, \ w_2, \ n_3, \ s_4, \ r_3, \ w_4, \ s_2 \ \}$
- At any instance, applications can only have a subset of *P* which means no application can have a global permission to read, write, receive notifications or make system calls to all network states thus $App \subseteq P$.

The concept that implements the presented authorisation model above is *Boolean Access Matrix* (BAM) [32][33], this method checks the privilege permissions of every network application with the saved application permission in a database. It carries out a Boolean comparison of the application permission and outputs either (0,1), where 1 means the application has that permission and 0 signifies a null permission.

In the authorisation module there is a method that helps network applications to query the controller in order to know the permission sets assigned to them. This is required considering the nature of network applications where the external applications that exercise control over the REST API and internal application that use local controller modules.

The controller provides set of Uniform Resource Identifiers (URIs) which a network application can use to specify its required resources and actions. The permission set should include all the relevant actions and resources needed by a network application to execute it intended design purpose.

When there is a network operation that is associated with one of the defined permissions of an application, the authorisation method known as BAM is called to determine whether that permission is allowed. If the permission is allowed then the operation will execute however if the application does not have the appropriate authorisation of that permission then that request will be denied in order to protect the network from malicious or unauthorised access. Section V elaborates more on the application and implementation of the Boolean Access matrix (BAM).

The authorisation module comprises of the authentication module and the respective allowed authorisation permissions for that application. Thus, an application has to be authenticated with its token before authorisation privileges are assigned to that application.

$$Authorisation = Authentication \;\|\; App\_privileges \tag{5}$$

$$\begin{aligned} &Authorisation = \\ &\qquad AuthenticateApp(argapp \;\wedge\; token\_argapp) \;\| \\ &\qquad func\_argapp \end{aligned} \tag{6}$$

*1) **Attributes**:* They are the functional building block of every application. They exist in every network application. They serve as the gauge via which the authorisation module can build a wrapper around the attributes and used it as a mode of access control. The analytical derivation is seen as follows:

$$\forall arg\_app \;,\; \exists func(i)[func \;\in\; arg\_app\_attributes]$$

$$\forall func(i) \;,\; \exists Operation[Read \vee write \vee system\_call \vee notify]$$

Attributes have the ability to carry out data modification actions like *read, write, system call and notify*. Careful observation and knowing the behaviour of an application can aid in identifying these various operations so that an authorisation control can be implemented against them. For instance a monitoring application that reports port events and gather statistics about link status should not be given access to system calls. Because that is not part of its primary function, so any suspicious request to *system calls* from that application should be denied and refused in the future. Algorithm 3 provides the pseudocode for authorisation.

Algorithm 3 Authorisation

```
1:  Initialising checks, verification and authorisation
2:  AuthoriseApp=AuthenticateApp((arg_app,token_argapp)
    || func_argapp(i))

3:  Application = {APP1,  APP2,  }
4:  Token = {Token1,  Token2}
5:   arg_app ∈ Application
6:   token ∈ Token
7:  if (AuthenticateApp() == False)  then
8:      Action := ('Deny Authorisation')
9:      Controller_info('Failed Authentication')
10: else if (AuthenticateApp() == True)  then
11:     Apply arg_app_func(n)
12:     n = number of allowed permissions
13: else
14:     Controller_info(Authorisation Denied)
15: end if=0
```

C. Trust Module

Trust is subjective and can be modelled as opinion that can be used as input arguments based on *Subjective Logic* reasoning models [34]. *Subjective Logic Reasoning* (SLR) is an advanced learning algorithm that is derived from Probability Calculus and Statistics [35]. It is a branch of artificial reasoning with a learning process at the initial stage of evaluating a proposition [36]. SLR extends standard Boolean Logic and provides room for making flexible decision that is beyond absolute (True or False) or (0,1) [37]. The end decision comes in three state which are (0,1, [0-1]), the first two are Boolean Logic (0,1) and the last value is a range from [0-1] that represents continuous uncertainty values since the expected outcome is either trusted, not trusted and the extreme values between trusted and untrusted relationship.

Trust can be seen as a directional graph relationship between two entities called the *trustor* and *trustee*. Subjective logic is a belief calculus specifically developed for modelling trust relationships [36][38][39].

$$Trustor \longrightarrow Trustee$$

In the context of this trust framework, the controller is *trustor* and the respective network application as *trustee*. It is modelled as moving from one edge of a graph to another.

$$Controller \longrightarrow Application$$

The *trustor* must have a sense of logical and intelligent decision and make assessment based on received information that can serve as input for decision making. These criteria fits into the model of a controller because it is the central logic of the network that oversees network operation. While the trustee is the agent which *trustor* relies on to provide vital service or information that will aid the *trustor* in making a global decision [40]. This goes well for the network application as trustee because it has task to execute in the network.

Trust opinions can be binomial opinions where absolutism is expected, because they can take two values as *True* or *False* or [*1,0*]. This domain can be denoted as $\mathbb{T} = \{t, \bar{t}\ \}$, so that the resulting decision can take either of the two between the controller C and the application A.

$$\mathbb{T} = \begin{cases} t: & \text{"The application is } \textit{trusted} \text{ ."} \\ \bar{t}: & \text{"The application is not } \textit{trusted} \text{ ."} \\ t: & \text{"Trust state is uncertain [0,1]."} \end{cases}$$

The analytical representation of trust is given as ω_{tA}. The preamble for the trust is to provide a mental map of how it is going to be applied in the framework. Based on the policy conditions set after the application pass authentication stage and authorisation, the next stage is the trust value which will give network application a more solid recommendation based on the conformity of application behaviour.

An application can be trusted or not, which is quite intuitive and absolute, however there are circumstances where the application can have trust value between these two extremes of trusted and not trusted. Subjective opinion provides a way of expressing the truth of propositions under degree of uncertainty. The notation ω_A^C is used to denote trust opinion in subjective logic, where the subscript A (*trustee*) in this context is the network application and denotes the variable or the proposition on which the opinion applies to [41]. The superscript C (*trustor*) is the subject that holds the opinion which is the controller.

The equivalent trust opinion between a controller and a network application is a composite function as follows:

$$\omega_A^C = (b_A^C, d_A^C,\ u_A^C,\ a_A^C) \tag{7}$$

TABLE III: Trust Mass

Mass	Representation
b_A^C	The controller belief mass on the application
d_A^C	The controller disbelief mass on the application
u_A^C	The controller uncertainty mass on the application
a_A^C	Base rate is a priori value assigned based on subjective assessment of the execution environment

Details are seen on Table III, and the proposition X from Table IV is the relationship between the controller and the network application. The concept of base rate *a_x* is tight to theory of probability. Given a domain D, with cardinality K , the default base rate is *1/k* . The base rate is used in deriving an opinion probability expectation value.

There is additive law for the trust evaluation and it is represented as follow:

$$(b_x + d_x +\ u_x) = 1 \tag{8}$$

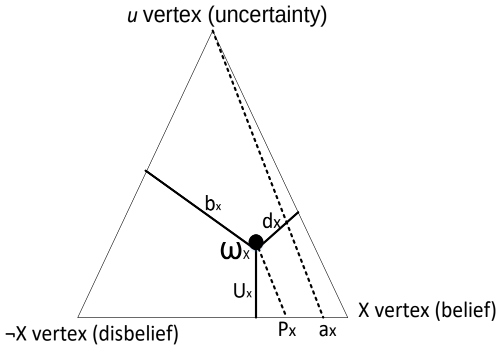


Fig. 4: Disbelief and belief visualisation

This translates to the sum of *belief , disbelief and uncertainty* must be equal to 1. In trust relationship the given probabiity based on binomial opinion at any given instance is :

$$P_x = b_x + a_x u_x \tag{9}$$

The functional trust (trust discounting[34]) that exist between the controller and the network application is given as follows:

$$\omega_x^{[C:A]} = \begin{cases} b_x^{[C:A]}(x) & = P_A^C b_C^X \text{ (x)} \\ d_x^{[C:A]}(x) & = P_A^C d_C^X \text{ (x)} \\ u_x^{[C:A]} & = 1 - P_A^C (b_C^X + d_C^X(x)); \\ a_x^{[C:A]}(x), & = a_x^C(x); \end{cases} \tag{10}$$

Since trust can be seen as extended graph from one edge to another, there is a way to visualise the equivalent result of the trust relationship via barycentric coordinate [42]. An example where the various trust parameters are represented on these coordinates, the visualisation of the opinion can be seen on the baryccentric triangle in Figure 4 . The point where both *bx, dx* and *ux* meet is the equivalent opinion ω_x. The projected probability can be obtained by running a line perpendicular on the opinion point. And the base rate *ax* is priori value. Table IV presents the various meaning and implication of the trust values, the following barycentric coordinates provide a visualised representation of the trust opinions. Equivalent trust values can be numeric (0, 0.1, 0.3) or based on severity (high,medium or low). In this work the trust assessment and evaluation will use numeric percentage.

1) Trust Handler Functions: These are the core critical components that help execute the real implementation of trust between the controller and the network applications. The handlers are as follows:

TABLE IV: Trust Opinion Truth Table

Mass Values	Representation
$b_x = 1$	Designates an absolute opinion equivalent to Boolean *True*
$d_x = 1$	Designates an absolute opinion equivalent to Boolean *False*
$u_x = 0$	There is no uncertainty and designates a dogmatic opinion
$0 < u_x < 1$	Opinion with some uncertainty
$u_x = 1$	This purely uncertain and designates a vacuous opinion

a) ReceiveRequest: This helper function interfaces with the authorisation module and is responsible for relaying the execution state of every network application. In the event that during execution an anomaly or breach is observed, it is the responsibility of this module to communicate such outcome to the *OpinionOperation* module. The output of the *ReceiveRequest* serves as the input to the *OpinionOperation* component.

b) OpinionOperation: This component works on the output of the *ReceiveRequest* and derives the trust equivalence using belief operation *belief (b), disbelief (d), uncertainty(u), baserate (a))* and probability calculus. When this operation is carried out this component reports on the reliability and dependability of the network application based on state execution. The module provides ground on which the trust can be visualised on graph, triangulation and probability distribution function.

c) Probability Distribution Function Equivalent (PDFEquivalent): The outcome from the *OpinionOperation* reports the outcome on a continuous range, and the most accurate and precise model to visualise the equivalent trust is to use the probability calculus. At any given state *PDFEquivalent* will show the trust value that can help the framework to make a solid decision on the trustworthiness of the application. More on probability is discussed later in this section.

d) TrustDB_handler: All derived decisions from network applications are cached in the TrustDB, this happens after the learning process based on the state of execution of the network application. This helps in fast tracking the trust process when the same network application is evaluated in future because the module already have the running attributes and anomalies if any.

e) LoginHandler: A log sink has been designed to report all unauthorised attempts and network state events. The logs are saved with time stamps and the ID of network applications the events is related to. Network management reports are dumped for analysis and further evaluation. Details of anomaly and threat detected are saved to this log. With the *LoginHandler* network admin can use threat quarantine mechanism or Advance Persistence Threat (APT) evasion mechanism to further safeguard the perimeter of the network.

A cognitive layer derived from Subjective Logic Reasoning is implemented in the control layer and this provides the controller with the ability to reason and analyse network application state based on combined opinion operations (*belief, disbelief, uncertainty and base rate*). The outcome from the combined opinion operations help designate whether a network application is trusted or not. This approach learns from giving input (network application state) and decides on the degree of trust of that application.

2) Trust Evaluation Methods: There are other relevant trust evaluation methods used in data and communication networks environment like the Recommender Trust System [43], Transitive (Indirect Trust)[34] and the Reputation Trust [44]. All of them provide a trusted relationship within the defined domain of use. However in the context of SDN networking these models can not provide the needed semantic and the flexibility to abstract and model a working relationship between the controller and the network application. How they derive the equivalent trust in their respective domain of application is as follows:

a) Recommender System: The equivalent trust is evaluated based on reliable transaction, process, or communication between a single entity (*trustor*) and multiple entities (*trustee*)[45]. The relationship can be depicted as many to one as seen in equation 11. The trustor holds a value or propositions (θ) that is needed by the trustees, these entities report on the reliability of the trustor after a successful transaction. A trust value is then compute to help prospective trustees make careful judgement when initiating a transaction with the trustor.

$$\sum_{i=1}^{n} User_1, User_2, User_3..User_n \xrightarrow{f(Trust)} User_x\theta \qquad (11)$$

Application of Recommender System is seen in buyer confidence used in Ebay, Amazon and onine auctions.

b) Transitive (Indirect Trust): In this method the trustor believes the reliability of a proposition based on a third party direct experience with the trustee. It differs from recommender systems, because there is no direct association. The trustor hinges the trust based on a third party relationship with the trustee.

$$A -- > B -- > C —> D$$

The trust equivalent between A and D is evaluated via the two transitive entities (B and C). There is no direct link between A and D, however with a single direct trust to D via the entity C, then A can trust D. The dash-line is transitive trust while the straight-line is direct functional trust. The transitive trust evaluation of [A,D] is seen as follows:

$$[A,D] = [A;B] : [B;C] : [C,D]$$

$$[A,D] = [A;C] : [C,D]$$

$$[A,D] = [A,D]$$

The evaluated transitive trust in the last stage grants A the full judgement to trust D.

c) Reputation Trust: In this method, trust is evaluated based on past behaviours of a trustee in a given domain, every trustee has a local trust value and the total trust of other entities is aggregated to obtain an equivalent trust mass for that domain [46]. Application of reputation trust can be seen in the Peer to Peer (P2P) networks. P2P is stateless relationship between peers, each peer has its own responsibility and power, it lacks central management system and as such peers can be malicious at will. The trust evaluation depends on aggregate local trust derived from each peer in the network.

All these trust evaluation methods are presented in order to justify why subjective logic is the best method to use in this research work, due to how the semantics and analytical derivation fits the concept of trust in the SDN network domain.

V. Evaluation and Implementation

This section presents the evaluation and implementation of the proposed Trust Framework which is an extension of our earlier work [28] on Controller to Network Application Trust . The implementation is carried out on Matlab simulator on a PC with Intel Core i5-4200M and 16GB RAM. The implementation is carried out as a set of Matlab scripts and functions with Object Oriented Concept which makes the project easy to use, extend and port to different implementation environment and testbed.

The controller core services and modules are explained in Section IV. The modules in conjunction with the main blocks of the proposed trust framework help in implementing and evaluating the framework. The evaluation will start with Authentication, Authorisation and then Trust.

A. *Testing of Authentication*

To test the feasibility of authintcation procedure, the two applications highlighted earlier will be used. And they are:

- L2 MAC learning Application
- IP Blacklist Application

*1) **Testing L2 MAC application for authentication**:* The L2 MAC learning is referred to as *App1* in the test and will be applied as *arg_app*, the token derived from Algorithm 2 will be put into action to actualise the objective of the framework.

$$Authentication(arg\_app\_App1 \parallel arg\_app\_token\_Ap1)$$

When there is an inbound request to install flows from the application, a challenge request is presented to the *application view*, this challenge is handled by the *app_parser* which picks the prospective application and the mapped token. The code snippet is seen on Algorithm 4. To evaluate the process let :

1) ψ_1 = *arp_parser* module with first argument as *App1.*
2) σ_1 = second argument of *app_parser* with argument token of *App1.*
3) σ_2 = second argument of *app_parser* with argument not *App1 token.*
4) τ = Successful authentication.
5) Ω = Error resulting from incomplete or failed authentication.

Therefore

$$(\psi_1 \ \wedge \ \sigma_1) \ \rightarrow \ \tau$$

iff *arg_app* $\in$ *Application*
and, *arg_app_token* $\in$ *Token*

The situation however will change if the expected *arg_app_token* in the *app_parser* module failed to mapped the entry contained in the database of Token as follows:

$$(\psi_1 \ \wedge \ \sigma_2) \ \rightarrow \ \Omega$$

iff *arg_app* $\in$ *Application*
and, *arg_app_token* $\notin$ *Token*

The other side of the authentication process entails if there is a failure to match any of the conditions ψ_1, σ_1 and τ.The result will be:

$$\neg(\psi_1 \ \wedge \ \sigma_1) \ \rightarrow \ \Omega$$

Algorithm 4 Authenticate Application (L2 MAC learning test)

```
1: Proceeding with application
2: Procedure AuthenticateApp(Application, Token)
3: Application = {APP1, APP2, }
4: Application_DB: Application ← Token
5: Token = {Token1, Token2} Token, derived from Algorithm 2
6: if app_parser(app_view, arg_app) ∩ tokencmp(Token, arg_pp_token) then
7:     arg_app ∈ Application, arg_app =1
8:     arg_app_token ∈ Token, arg_app_token=token1
9:     contorller_info: Authentication_Successful
10:    save(arg_app(1) ← arg_app_token(1))
11: else if app_parser(app_view, arg_app) ∩ tokencmp(Token, ¬arg_arp_token) then
12:    arg_app ∈ Application, arg_app = 1
13:    arg_app_token ∉ Token
14:    controller_info: (Wrong token for arg_app)
15:    error(Authentication_fail)
16:    controller_info(Access_Denied)
17:    Save(arg_app ← arg_app_token)
18: else
19:    error(Invalid Application)
20: end if
   =0
```

*2) **Testing IP Blacklist Application**:* This is the second test demonstrating how IP blacklist application is verified against the authentication module. The trigger occurs when there is an inbound IP packet to the controller looking for a route out of the network. The authentication module calls the *app_parser* module to carry out the authentication procedure on the Application. To test the process let:

1) ψ_2 = *arp_parser* module with first argument as *App2.*

2) σ_3 = second argument of *app_parser* with argument token of *App2*.
3) σ_4 = second argument of *app_parser* with argument not *App2 token.*
4) τ = Successful authentication.
5) Ω = Error resulting from incomplete or failed authentication.

Therefore

$$(\psi_2 \wedge \sigma_3) \rightarrow \tau$$

iff *arg_app* $\in$ *Application*
and, *arg_app_token* $\in$ *Token*

The situation however will change if the expected *arg_app_token* in the *app_parser* module failed to mapped the entry contained in the database of Token as follows:

$$(\psi_2 \wedge \sigma_4) \rightarrow \Omega$$

iff *arg_app* $\in$ *Application*
and, *arg_app_token* $\notin$ *Token*

The other side of the authentication process entails if there is a failure to match any of the conditions ψ_2, σ_3 and τ.The result will be:

$$\neg(\psi_2 \wedge \sigma_3) \rightarrow \Omega$$

The pseudo code for testing IP blacklist is similar to algorithm 4 that authenticates L2MAC, the only difference is that the network application of interest here is the IP blacklist. That is why the pseudo code serves as framework via which repetitive network applications can go through for the purpose of application and verification.

To further check for the correctness and precision of the network application detection by the trust framework. A matrix based on the *ground truth* of authentication is presented in Table V. The nomenclature of the matrix provides True Positive as (TP), True Negative (TN), False Positive (FP) and False Negative (FN). The results obtained in Table VI are derived from running algorithm 1 which is application correctness.

TABLE V: Nomenclature for Authentication Correctness

TP	There is a request for authentication by a legitimate network application and application authenticates successfully.
TN	There is a request for authentication by illegitimate application but the application fails due to wrong credentials.
FP	Failed authentication by a legitimate network application with correct credentials.
FN	Malicious application that authenticates successfully.

TABLE VI: Precision Test for Application with correct Credentials

Authentication	TP	TN	FP	FN
L2 MAC	50	0	0	0
IP Blacklist	50	0	0	0
Malicious_L2MAC	0	50	0	0

For the correctness and precision test, the two applications (L2 Mac and IP blacklist) were run with correct credential (*Token*) against the authentication module 50 times. During the test all the 50 trials were successful, there are no anomalies recorded only True Positive (TP) hits. The same test for a malicious L2 MAC application is initiated too, however the *Malicious_L2MAC* authentication would fail because it has the wrong *token* and this time around all the hits are on True Negative (TN).

B. ***Testing for Authorisation***

Evaluation and testing of the network application continues with the authorisation being the second part of the verification. The network applications in use are still the L2 MAC learning and the IP Blacklist. To authorise an application the logic and the intended purpose of the application has to be known and preconfigured beforehand. The test will begin with L2 MAC application, the prerequisite is that the application cannot proceed to authorisation if it fails authentication.

1) ***Authorising L2 MAC application****:* To authorise L2 MAC learning application, there are steps that must be followed and they are:

- The attributes have to be identified.
- And the attributes are derived from the logic and behaviour of L2 MAC learning application.

Typically in L2 MAC learning implementation and procedures, when packets arrive at the switch via the *packet_in* event, and the switch has no matching flow entry for that inbound packet. The *table_miss* is applied and the flow is forwarded to the controller. The logic of the L2 MAC learning has the ability to update the incoming flow because it has the flow and the source port where the flow originates. The incoming flow is saved in the flow table.

This process exhibits a function of *reading* and *writing*. However in the context of L2 MAC learning application the incoming flow is *src_mac* and attempting to establish a connection with the destination end (*dst_mac*). The next stage is to check whether the destination is contained in the flow table, if yes then the flow is forwarded. This exhibits a *unicast* message between two end points. However if the destination is not known then the switch will definitely flood to all connected ports, and this is a *broadcast*.

The host will respond on receiving the broadcast, this broadcast is further forwarded to the controller, the controller will now set flow for that destination. With both the source and destination contained in the flowtable now the controller can set a flow with timer for future flows to be forwarded without taking the control path.

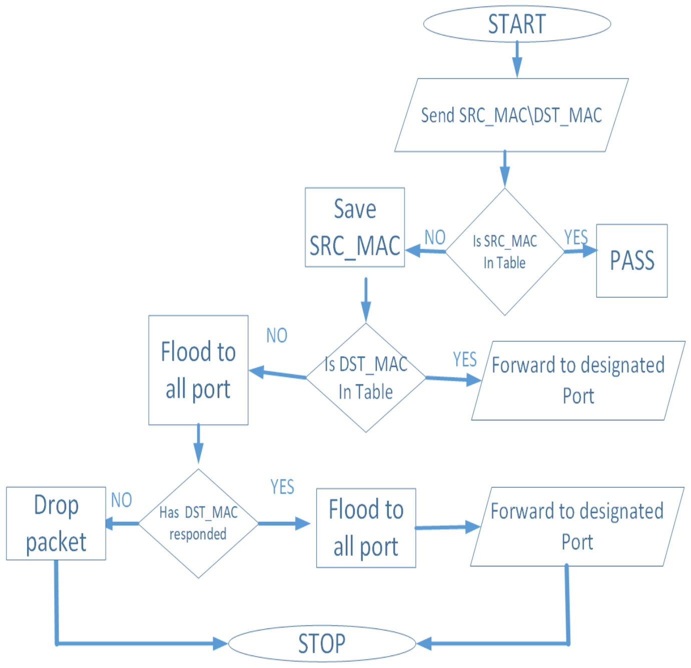


Fig. 5: High-level view of L2 MAC Learning Application

The high level representation of L2 MAC learning can be seen on Figure 5 and from careful analysis and evaluation Table VII shows the set of attributes highlighted in L2_MAC learning application.

TABLE VII: Attributes of L2 MAC learning Application

S/N	Attributes / Functions
1	*read_src_mac*
2	*read_dst_mac*
3	*unicast_forwarding*
4	*brdc_fwd*
5	*write_dst_mac*
6	*write_mem*
7	*multicast_fwd*
8	*write_src_mac*
N	*Attribute_n*

These attributes drive the successful execution of the application and as such can be used as authorisation antidote against unauthorised code execution.

Note: The listed functions are not an exhaustive list of L2_MAC attributes, but a high level and comprehensive functions that make up the application. The attributes are not sequential as shown and one can be called before the other. The analysis and evaluation will be limited to these defined sets to demonstrate feasibility of the framework. The attributes are modelled in functions.

To initiate authorisation, first the L2_MAC application has to pass authentication. After successful authentication, the authorisation privileges are passed to L2_MAC application. From equation 6, the authorisation is derived as follows:

$$Authorisation = AuthenticateApp(argapp \wedge token\_argapp) \, || \, func\_argapp$$

And the $arg\_app \leftarrow L2\_MAC$,

$$Auth_{L2\_MAC} = AuthenticateApp(L2\_mac \wedge L2\_mac\_token) \, || \, L2\_mac_{func(i)}$$

$Auth_{L2\_mac}$ can only complete if the first part:

$$AuthenticateApp(L2\_mac \wedge L2\_mac\_token) == True$$

If however it turns out to be *false* then $Auth_{L2\_mac}$ will fail because both the conditions must be *True* for the authorisation to proceed and complete.

The database of the authorisation must contain the defined set of allowed *L2_MAC_func*.In the implementation the database contains both for IP blacklist and the L2_MAC application.

$$Authorisation\_DB = \begin{cases} L2\_MAC_{func}, & \text{if } APP = L2\_MAC; \\ IP\_Blacklist_{func}, & \text{if } APP = IPBlacklist; \\ Arg\_AppN_{func}, & \text{if } APP = AppN; \end{cases}$$

The set of mapped elements contained in the authorisation database for L2_MAC are:

$$Authorisation\_database = L2\_MAC \leftarrow L2\_MAC_{func(i)}$$

Where in *func(i)*, i $\in \mathbb{N}$, $0 < i < 8$

$$\forall arg\_app \; \exists \; func(i) \; [func(i) \subseteq Authorisation\_DB]$$

Out of the listed L2 MAC attributes on Table VII, two attributes (*read_src_mac* & *read_dst_mac*) would be selected for brevity to demonstrate how the process for the authorisation are verified and evaluated by the framework. And at the end a threat will be introduced in to the network operation and with the application of the framework the execution of the threat would be stopped and halted.

*a) **READ_SRC_MAC** :*

$$L2\_MAC_{func(1)} \xrightarrow{f(read)} SRC\_MAC$$

The authorisation wrapper (*Auth_wrapper*) implements the Boolean Access Matrix (BAM) method of comparison to see whether if the *arg_arp* attribute is contained in the database of the Authorisation module. It is a Boolean function that returns *True* or *False* depending on the matching entry in Authorisation database.

$$Auth\_wrapper(arg\_app\_func(i),\ func(i))$$

$$BAM\_check = \begin{cases} Auth\_wrapper(x,y), & \text{if True}= Execute; \\ Auth\_wrapper(x,y), & \text{if False}= Deny; \\ otherwise, & \text{Deny;} \end{cases}$$

Where the *arg_app_func* = L2_MAC_func(i), and i=1 in this instance.

$$Auth\_wrapper(\ L2\_MAC\_func(1),\ func(1)\ ) \tag{12}$$

Execute if and only if output is *True* from equation 12, however $Auth_{L2\_MAC}$ will fail if the condition output is *false*.

b) ***READ_DST_MAC****:* From equation 13, where in this case *read_dst_mac* is considered. When the framework receives a request to execute a function regarding *read_dst_mac*, the framework checks within the authorisation database:

$$Auth_{L2\_MAC} = \\ AuthenticateApp(L2\_mac\ \wedge\ L2\_mac\_token)\ \| \\ L2\_mac_{func(i)} \tag{13}$$

$$Authorisation\_database = L2\_MAC \leftarrow L2\_MAC_{func(i)}$$

Where *func(i)*, i $\in \mathbb{N}$, $i = 2$

$$L2\_MAC_{func(2)} \xrightarrow{f(read)} DST\_MAC$$

$$Auth\_wrapper(\ L2\_MAC\_func(2)\ ,\ func(2)\ ) \tag{14}$$

From equation 14 if the output is *True*, then *read_dst_mac* will execute else if it is *false*.The execution will fail and return authorisation failure.

A threat is introduced in the L2 MAC learning application as follows:

c) ***Threat_1****:* Threat_1 is introduced to demonstrate how resilient and efficient the trust framework is in detecting external threat. The function *threat_1* is not contained or captured in the authorisation database. Even when the *arg_app* pass Authentication stage and attempts to execute functions that are not captured in the database they will be denied.

$$Authorisation\_database : L2\_MAC \leftarrow L2\_MAC_{func(i)}$$

The condition is *func(i)* $\in$ *L2_MAC(func(i))*, where this is not the case and *func(i)* $\notin$ *L2_MAC(func(i))*. Then that function is bound to be rejected based on the analytical derivation of the framework.

$$func(threat\_1) \xrightarrow{f(execute)} socket(src\_ip : port\ \wedge\ dst\_ip : port)$$

This is a *flow_mod* action where the controller is expected to retrieve such sensitive network information including IP address and their respective port association. This action is not a behaviour or attribute of L2_MAC.

$$Auth\_wrapper(\ L2\_MAC\_func(x)\ ,\ func(threat\_1)\ ) \tag{15}$$

From equation 15 the Boolean output cannot be *True*, because func(*threat_i*) is not contained in the authentication database. The execution will be halted and flows will be installed to deny similar future request.

The L2 MAC learning application is expected to be compliant with the defined set of rules. The threat here is making calls and listening to established socket connection between client and server. The calls will try to execute but due to the precision of the authorisation module and the predefined configurations of the authorisation database this request is denied and flow entries are initiated to refuse future requests, the algorithm is seen as follows in Algorithm 5.

Algorithm 5 Authorisation L2_MAC

```
1: Initialising checks, verification and Authorisation
2: arg_app = L2_MAC
3: token_argp_app = $token_{L2\_MAC}$
4: AuthoriseApp = **AuthenticateApp**((L2_MAC,$token_{L2MAC}$) || $func_{L2MAC(i)}$)
5: if (AuthenticateApp() == False) then
6:     Action := ('Deny Authorisation')
7:     Controller_info('Failed Authentication')
8: else if (AuthenticateApp() == True) then
9:     $L2\_mac_{func}$ = {read_src_mac, read_dst_mac, unicast_forwarding, write_src_mac,
       write_dst_mac, brdc_fwd, multicast_fwd, write_mem}
10:    Auth_wrapper( L2_MAC_func(i), func(i) )
11:    Where in func(i), i $\in \mathbb{N}$, $0 < i < 8$
12:    if Auth_wrapper() == True then
13:        Apply L2_MAC_func(i)
14:    else if Auth_wrapper() == False then
15:        Action := Deny Authorisation
16:        Controller_info(Possible Threat)
17:    end if
18: else
19:    Controller_info(Authorisation Denied)
20: end if=0
```

2) ***Authorising IP Blacklist Application*** *:* The blacklist IP application has attributes similar to that of the L2 MAC application.Attributes are behaviour descriptors and they vary from one application to another. Though blacklist IP operates differently than L2 MAC learning application. The framework

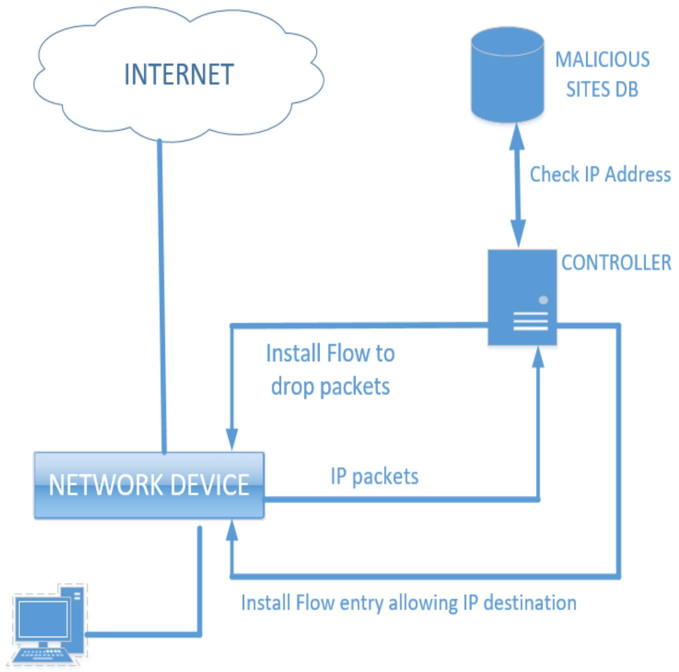


Fig. 6: IP Blacklist Logic

looks from a high level view of the application and implement the authorisation policy.

The IP blacklist application first set flows at edge switches, and these flows will wait for an incoming IP event to trigger the blacklist logic. The blacklist application then compares the incoming IP request with the set flows contained in the blacklist database, if there is a match the request is denied else the request is forwarded to the final destination.

The main logic of the application is seen on Figure 6 and highlighted as follows:

1) The controller set flows in edge devices with instructions to forward IP address request to the controller.
2) With the received request, the blacklist database is consulted with known malicious IP sites.
3) If the IP address is not contained in the blacklist database, the controller returns a flow with instructions to forward.
4) If there is a match with entries in the blacklist database then the request is denied.

From careful analysis and evaluation Table VIII shows the set of attributes highlighted in IP blacklist application.

TABLE VIII: Attributes of L2 MAC learning Application

S/N	Attributes / Functions
1	*read_src_ip*
2	*read_dst_ip*
3	*cmp_ip*
4	*drp_ip*
5	*fwd_dst_ip*
6	*set_flows*
7	*write_mem*
N	*Attribute_n*

These attributes drive the successful execution of the application and as such can be used as authorisation constraint against unauthorised code execution.

To begin the process of authorising IP blacklist application, the condition as per the framework is for the application to pass authentication which is the first stage. After successful application then privilege rights are assigned to the application as to what attributes or functions are allowed. This defines a perimeter through which policing of function execution can be carefully monitored and to prevent unauthorised access of network resources. The derivation for IP blacklist will be:

$$Authorisation = AuthenticateApp(argapp \land token\_argapp) \parallel func\_argapp$$

And the $arg\_app \rightarrow IP\_Blk$

$$Auth_{IP\_Blk} = AuthenticateApp(IP\_Blk \land IP\_Blk\_token) \parallel IP\_Blk_{func(i)}$$

Based on the framework condition, the authentication module *AuthenticateApp* must be *True*, if however it turns out to be *false* then $Auth_{IP\_Blk}$ will result in authorisation failure. The listed functions are not limited to what a blacklist IP application can execute, more attributes can be composed and must be captured in the database of the authorisation database. The entries for $IP\_Blk_{func(i)}$ are contained in the database as follows.

$$Authorisation\_DB = \begin{cases} L2\_MAC_{func}, & \text{if } APP = L2\_MAC; \\ IP\_Blacklist_{func}, & \text{if } APP = IP\_Blacklist; \\ Arg\_AppN_{func}, & \text{if } APP = AppN; \end{cases}$$

The mapped elements contained in the database for $IP\_Blk_{func(i)}$ are :

$$Authorisation\_database = IP\_Blk \leftarrow IP\_Blk_{func(i)}$$

Where $func(i)$, i $\in \mathbb{N}$, $0 < i < 7$

$$\forall arg\_app \ \exists \ func(i) \ [func(i) \subseteq Authentication\_DB]$$

An authorisation wrapper *Auth_wrapper* is used, which is the enabler of the Boolean Access (BAM) Method. It is a Boolean that checks for *arg_arp* attribute in the mapped authorisation database.

$$Auth\_wrapper(arg\_app\_func(i), \ func(i))$$

$$BAM\_check = \begin{cases} Auth\_wrapper(x,y), & \text{if True= } Execute; \\ Auth\_wrapper(x,y), & \text{if False= } Deny; \\ otherwise, & \text{Deny;} \end{cases}$$

Out of the many attributes listed from Table VIII, two will be evaluated based on the framework's authorisation procedure and they are *read_dst_mac* and *set_flows*.

a) ***READ_DST_IP**:* When the framework receives a request to execute a function regarding *reading* a destination IP address. The framework queries for a match in the request and the output will depend on the content of the authorisation database.

$$Auth_{IP\_Blk} = AuthenticateApp(IP\_Blk \ \wedge \ IP\_Blk\_token) \parallel IP\_Blk_{func(i)}$$

$$Authorisation\_database = IP\_Blk \leftarrow IP\_Blk_{func(i)}$$

Where *func(i)*, i $\in \mathbb{N}$, $i = 2$

$$IP\_Blk_{func(2)} \xrightarrow{f(read)} DST\_IP$$
$$Auth\_wrapper(\ IP\_Blk_{func(2)} \ , \ func(2) \ )$$

If the output is *True*, then *read_dst_ip* will execute else if it is *false*.The execution will fail and return authorisation failure.

b) ***SET_FLOWS**:* The edge switches in the infrastructure layer are the first point of contact to hosts making request to network resources. Flows are set to capture traffic that are perceived as rogue, malicious or threats. The switches subscribe to events and forward to the controller which triggers the logic that execute the filtering of IP blacklist. For this we have the relation as follows:

$$Auth_{IP\_Blk} = AuthenticateApp(IP\_Blk \ \wedge \ IP\_Blk\_token) \parallel IP\_Blk_{func(i)}$$

$$Authorisation\_database = IP\_Blk \leftarrow IP\_Blk_{func(i)}$$

Where *func(i)*, i $\in \mathbb{N}$, $i = 6$

$$IP\_Blk_{func(6)} \xrightarrow{f(set\_flows)} (dst\_ip \leftarrow \ target\_ipaddress)$$

$$Auth\_wrapper(\ IP\_Blk_{func(6)} \ , \ func(6) \ )$$

These flows will be installed in the Flow Table at edge switches in proactive mode, and once there is a match the switches invoke the control path where the blacklist IP logic which will trigger filtering of designated IP addresses.

c) ***Threat_2**:* This is a depiction of a live threat that is trying to execute its operation inside the network operating system. The framework as a policy correctness checker will serve as a boundary that separates the security of the internal network and external request from rogue sources. The behaviour of this threat is to dig deep in the operating system and make unsolicited calls to resources allocated for legitimate network operation. The threat is concatenated inside the working blacklist IP appication and will try to execute as part of the attributes of an IP blacklist application.

$$Auth_{IP\_Blk} = AuthenticateApp(IP\_Blk \ \wedge \ IP\_Blk\_token) \parallel IP\_Blk_{func(i)}$$

$$Authorisation\_database = IP\_Blk \leftarrow IP\_Blk_{func(i)}$$

The design of authorisation module is to make sure there is policy compliance. Even after successful application which is the first part of the module, if there are functions that are not captured in the authorisation database they cannot execute their operation.

$$func(i) \in IP\_Blacklist(func(i)) \qquad (16)$$

From equation 16 , if $func(i) \notin IP\_Blacklist(func(i))$, the authorisation database will invoke the *drp_ip* action so that the request will be denied because the function is not captured. The function call initiated by the threat is seen as follows:

$$\left(IP\_Blk_{func(threat\_2)} \xrightarrow{f(execute)} os\_call(version, \ buildtype, \ bios, \ config) \right)$$

$$Auth\_wrapper(\ IP\_Blk_{func(7)} \ , \ func(threat\_2) \ ) \qquad (17)$$

The authorisation database will come into play now and apply the pre-configured policy of allowed functions for a blacklist IP application. And from the captured functions this threat in particular does not match or correspond to any entry in the database of the authorisation module. The *authwrapper* function in equation 17 will perform the Boolean operation and output the result where the threat is not captured and the resulting action is to deny and log the report to monitoring log (*LoggingHandler*). Algorithm 6 shows the pseudo-code that initiates and executes the authorisation process successfully.

C. Performance Analysis

This section provides performance indices on how the authentication and authorisation module handle the two network applications. The computational cost of running the L2 MAC application from authentication to authorisation is analysed in Figure 7. The test is conducted using the *timeit* and *tic* functions for code profiling and stress testing. The mean (μ) execution time when the module is called

Algorithm 6 Authorisation of IP Blacklist

```
1: Initialising checks, verification and Authorisation
2: arg_app = IP_BLK
3: token_argp_app = $token_{IP\_BLK}$
4: AuthoriseApp = AuthenticateApp(($IP\_BLK, token_{IPBLK}$) $\|$ $func_{IPBLK(i)}$)
5: if (AuthenticateApp() == False) then
6:     Action := ('Deny Authorisation')
7:     Controller_info('Failed Authentication')
8: else if (AuthenticateApp() == True) then
9:     $IP\_Blk_{func(i)}$ = {read_src_ip, read_dst_ip, cmp_ip, drp_ip, fwd_dst_ip, set_flows, write_mem}
10:    Auth_wrapper( IP_BLK_func(i), func(i) )
11:    Where in func(i), $i \in \mathbb{N}$, $0 < i < 8$
12:    if Auth_wrapper() == True then
13:        Apply IP_BLK_func(i)
14:    else if Auth_wrapper() == False then
15:        Action := Deny Authorisation
16:        Controller_info(Possible Threat)
17:    end if
18: else
19:    Controller_info(Authorisation Denied)
20: end if=0
```

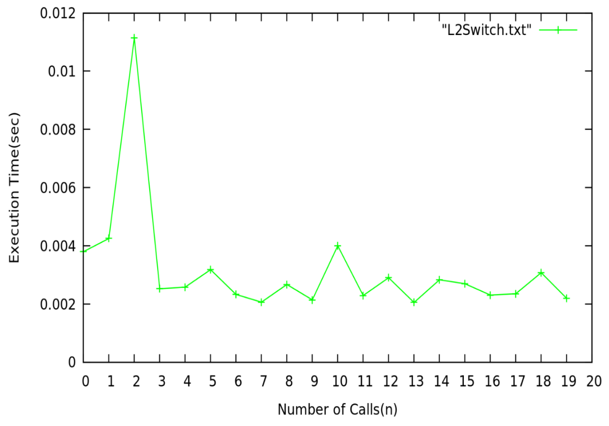


Fig. 7: Stress Test for L2 MAC Learning Application

is 0.00317sec (3.17ms) and the standard deviation (σ) is 0.00198sec (1.98ms). There is no significant overhead when running the application because the execution time is negligible. The minimum execution time is 2.06ms and the maximum is 11.14ms. From the standard deviation value (σ=1.98ms), a deduction of how the code uses the CPU execution time can be predicted, because adding the standard deviation to the mean would provide an execution time of 5.15ms which is still within acceptable range and would not impact the performance of the trust framework at large.

Figure 8 presents the code profiling test for the IP blacklist application, the test shows mean (μ) value of 24.79ms and 12.78ms as standard deviation (σ). The mean value is high

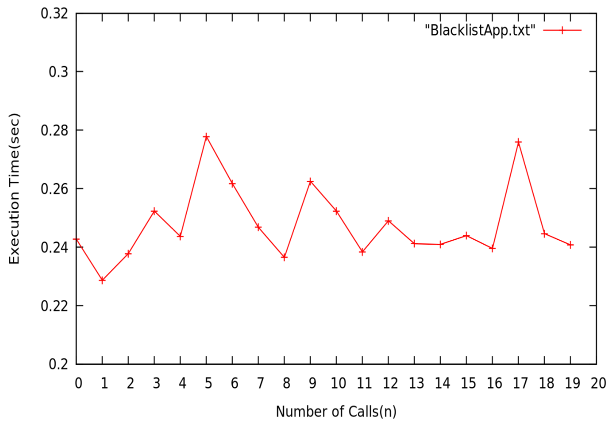


Fig. 8: Stress Test for the IP Blacklist Application

compared to other modules like the L2 MAC because there are more procedural steps in the IP blacklist application and is taking a tall on the code execution time. However 24.79ms as the mean execution time would not affect the efficiency and the performance of the trust framework . To evaluate the impact more, the standard deviation (σ) is applied to the mean(μ) and the resulting execution time is (26.08ms and 23.52ms). Both the min 22.87ms and max 27.79ms values are still within reasonable code execution time that would not impact the trust framework performance.

Table IX presents the summary of code profiling test for both the L2 Mac and IP blacklist application.

TABLE IX: Code Profiling Test

Application	Min(ms)	Max(ms)	Mode(ms)	Mean (μ)	Std (σ)
L2 Mac	2.06	11.14	2.33	3.17	1.98
IP Blacklist	22.87	27.79	24.31	24.79	12.78

The next section introduces the concept of trust in to the framework. After successful execution of authentication and authorisation, a numerical trust value is assigned to network applications based on how the functions conform to the policy of the network.

D. Trust Evaluation and Implementation

Applications can be trusted after successful execution of defined functions without any anomaly or deviation from list of specified tasks. Numeric trust values are assigned to applications based on how they conform to the network policy. If there is a successful execution of defined tasks by application, the application will have high trust mass. This will in turn makes the controller to associate with the network application without restrictions.

However when these defined sets of functions are not followed by the network application, then the controller will flag the network application and assign low trust value which consequently can lead to complete denial of the application to the entire network resources. The given trust values for any network application that executes successfully are given on

Table X, these values are default and are assigned based on trust framework design.

TABLE X: Default trust mass for applications that run successfully

Trust Opinion	Trust mass on local applications
b_E^A	0.9
d_E^A	0.1
u_E^A	0
a_E^A	0.90

These values indicate a strong trust on the running application because the belief mass (*b)* on the application is 0.9, and disbelief mass (*d*) is 0.1 which is very negligible. The uncertainty mass (*u*) is 0 signifying no ambiguity. The base rate (*a*) is a priori value for the execution environment and the execution environment is trusted with mass value of 0.9. These trust values (*b, d, u, a*) are carried by the network application, so when associating with the controller, the controller will evaluate the values through *trust discounting operation*. The controller also hold certain trust assumption on the application, section V-D1 will elaborates on that.

To begin trust evaluation the framework has three stages to certify and evaluate applications to make them trustworthy as follows:

1) Authentication
2) Authorisation
3) Trust evaluation

The first and second stages (authentication and authorisation) are prerequisite to the trust assignment stage. The trust evaluation will be carried out on the two applications L2 MAC and IP blacklist. And the evaluation will begin with L2 MAC application.

*1) **Trust Evaluation for L2_MAC application**:* For trust evaluation, the first two stages must be *True* for the output to be successful. If either the authentication or the authorisation is false, the trust vetting stage would not proceed which consequently leads to trust failure. The trust verification *Tv* is given as:

$$\forall Application \ \ \exists \ \ (Token \ \ \wedge \ \ Sets \ \ of \ \ functions)$$

And the representation for every application and its respective token is given as:

$$Authenticate\_DB = \begin{cases} \texttt{Token1}, & \text{if } APP = 1; \\ \texttt{Token2}, & \text{if } APP = 2; \\ \texttt{x}, & \text{if } APP = N; \end{cases}$$

And for the functions in each application, the relativism for authorisation is as follows:

$$\forall arg\_app \ \ \exists(\ func(i) \mid func \ \in \ arg\_app\_attribute)$$

To evaluate *Tv* based on L2 MAC application. The procedures are given as follows:

$$Tv = (Authentication \ \ \| \ \ Authorisation \parallel Trust)$$

$$\begin{aligned} Tv = \\ & Authenticate(arg\_app \ \wedge \ token\_arg\_app) \ \wedge \\ & Auth\_wrapper(\ arg\_app\_func(2) \ , \ func(i) \ ) \ \wedge \\ & Trust_{C\_L2MAC} \end{aligned}$$

Where *arg_app* = L2 MAC and $Trust_{C\_L2MAC}$ is the trust relaionship between the controller and the L2_MAC

$$\begin{aligned} Tv = \\ & Authenticate(L2\_MAC \ \wedge \ token\_L2\_MAC) \ \wedge \\ & Auth\_wrapper(\ L2\_MAC\_func(i) \ , \ func(i) \ ) \ \wedge \\ & Trust_{C\_L2MAC} \end{aligned} \tag{18}$$

The default trust value for network applications that execute successfully is given on Table X. The default trust that the controller assigns for every application that is developed and deployed locally is given as per Table XI based on the deign guide of the trust framework.

TABLE XI: Default trust mass on local applications

Trust Opinion	Trust mass on Controller
b_A^C	1
d_A^C	0
u_A^C	0
a_A^C	0.9

To calculate the trust between the controller and L2_MAC application to assess the trust relationship. The trust opinion values from Table X and that of the controller on Table XI have to undergo a *trust discounting operation* to get the equivalent trust based on *Subjective Reasoning* and *trust association* conditions:

$$Trust_{C\_L2MAC} = \omega_A^C \ \otimes \ \omega_E^A$$

$$Trust_{C\_L2MAC} = (b_A^C, \ d_A^C, \ u_A^C, \ a_A^C) \ \otimes \ (b_E^A, \ d_E^A, \ u_E^A, \ a_E^A)$$

$$Trust_{C\_L2MAC} = (1, \ 0, \ 0, \ 0.9) \ \otimes \ (0.9, \ 0.1, \ 0, \ 0.9)$$

$$Trust_{C\_L2MAC} = (0.9, \ 0.1, \ 0, \ 0.9)$$

The trust operator $\otimes$ is referred to as the discount operator. It is not a multiplication operation but a discounting operation where the equivalent values are obtained based on equation 10. The trust operator is used in *Subjective Logic* to model and derive trust relationships between services and components based on logic reasoning. Based on the derivation from equation 10 the equivalent $Trust_{C\_L2MAC}$ is (0.9, 0.1, 0 , 0.9).

Because trust is a directed graph relationship between two entities it can be visualised for evaluation. The graph visualisation for the derived trust $Trust_{C\_L2MAC}$ can be seen on

Figure 9. The two barycentric triangles on the left represent the plotted values of ω_A^C *and* ω_E^A. The equivalent trust on the right hand side is $Trust_{C\_L2MAC}$ and lies within the highly acceptable *trust* severity. Therefore for a local application that is running without any anomaly or inconsistency this is the equivalent derived trust.

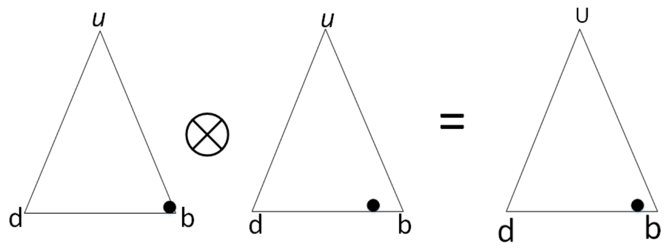


Fig. 9: Derived Trust of $Trust_{C\_L2MAC}$

And the probability distribution function (PDF) plot is seen on Figure 10. The graph peaks at the right hand side. It solidifies the trust derivation of the application to be highly dependable and reliable.

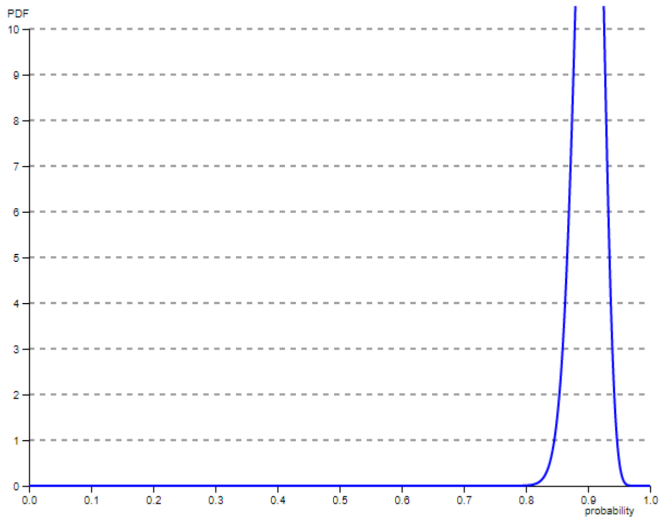


Fig. 10: Normal application behaviour $Trust_{C\_L2MAC}$ in graph notation

*2) **Trust Evaluation and Assessment on IP Blacklist Application for One Anomaly:*** For the evaluation of the IP Blacklist application, the test will be carried on when there is a violation of the set authorisation policy. The IP blacklist application has i^{th} set of defined functions that can execute and perform operations in the network. These i^{th} functions are well defined, captured and saved in the authorisation database for verification and checks. Failure to comply with these sets of defined i^{th} functions may lead to the IP Blacklist trust level to fall at a drastic rate which makes the controller to flag the network application as malicious.

Network policy instructions are set for safe, robust and efficient way of network operation. The IP blacklist application is registered with predefined set of rules, privileges and access rights. If there is any kind of anomaly or deviation from the norm, then the framework will react and correct the fault as necessary. This is achieved with the trust analytical evaluation. What happens in any situation is that the moment there is a trigger that indicates a deviation from the normal function, then the trust framework will adjust the belief mass (b) and the base rate (a). The effect will be that their values are halved.

$$\begin{aligned} Tv = {} & \\ & Authenticate(IP\_BLK \wedge token\_IP\_BLK) \wedge \\ & Auth\_wrapper(\ IP\_BLK\_func(i)\ ,\ func(i)\ ) \wedge \\ & Trust_{IP\_BLK2} \end{aligned}$$

Using the default derived trust for applications $\omega_x^{[A:E]}$ equals (*0.9, 0.1, 0, 0.9*). The trust can be deduced by applying the constraints laid out for any network application that deviates from the set of defined instructions. These constraints are that belief mass (b) and base rate (a) will be halved . Therefore $\omega_x^{[A:E]}$ will be $(\frac{1}{2}(b_E^A),\ d_E^A,\ u_E^A,\ \frac{1}{2}a_E^A)$.

$$\omega_E^A = (\ \frac{1}{2}(b_E^A),\ d_E^A,\ u_E^A,\ \frac{1}{2}a_E^A))$$

$$\omega_E^A = (\ \frac{1}{2}(0.9),\ d_E^A,\ u_E^A,\ \frac{1}{2}(0.9))$$

With the additive rule of trust where *b + d + u = 1,* the new value of d_E^A can be obtained.

$$d_E^A = 1 - \ (b_E^A + \ u_E^A)$$

$$\omega_E^A = (0.45,\ 0.55,\ 0.0,\ 0.45)$$

The derived trust for a single anomaly can be obtain by carrying out a trust discount operation with ω_A^C *and* ω_E^A.

$$Trust_{IP\_BLK2} = \omega_A^C\ \otimes\ \omega_E^A$$

$$Trust_{IP\_BLK2} = (b_A^C,\ d_A^C,\ u_A^C,\ a_A^C)\ \otimes\ (b_E^A,\ d_E^A,\ u_E^A,\ a_E^A)$$

$$Trust_{IP\_BLK2} = (1,\ 0,\ 0,\ 0.9)\ \otimes\ (0.45,\ 0.55,\ 0.0,\ 0.45)$$

$$Trust_{IP\_BLK2} = (0.45,\ 0.55,\ 0.0,\ 0.45)$$

The trust visualisation can be seen in the barycentric triangle as shown on Figure 11.

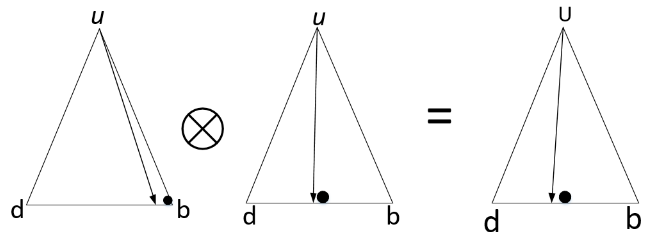


Fig. 11: The equivalent of the new $Trust_{IP\_BLK2}$ for one anomaly

As a result of the reduced trust obtained from applying the constraint, the confidence and reliability of the IP blacklist application reduced. The disbelief mass *(d)* has now increased to 0.55, where the belief mass (*b*) reduced drastically to 0.45. The visualisation graph shows the decision making point shifting towards the disbelief vertex. The base rate (a) is halved ($\frac{1}{2}(90)$), which signifies reduced confidence in the execution environment of the application.

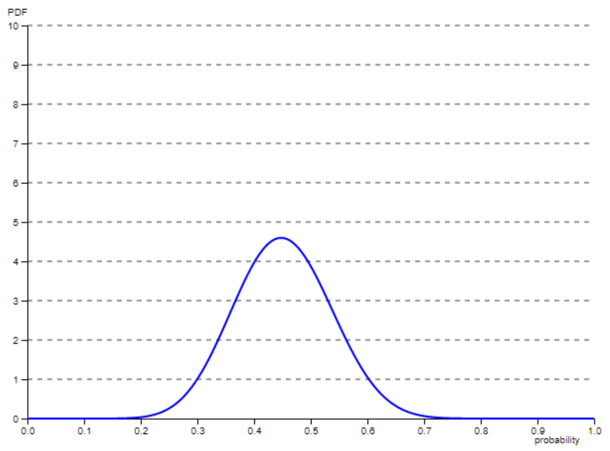


Fig. 12: When application exhibits strange behaviour $Trust_{IP\_BLK2}$ for one anomaly

The dependability on the application is in question based on the trust equivalent point on the barycentric triangle which shifts to the left. This signals a distrust and further more the PDF plot on Figure 12 shows another deviation of the trust from the right hand side. This gives a broader picture of how the trust deviates from an area of dense belief mass to disbelief position.

*3) **Further Test on L2 MAC and IP Blacklist Applications**:* The following tests show different scenarios of trust when the applications exhibit behaviours that are deem malicious within the network. Figure 13 for one anomaly in L2 MAC application exhibits a similar trust downfall just like the one in IP Blacklist. The belief mass and the base rate will be halved and the equivalent. The equivalent confidence and reliability of the application will reduce.

In Figure 14 the L2 MAC application exhibits a situation where multiple anomalies were recorded by executing the L2

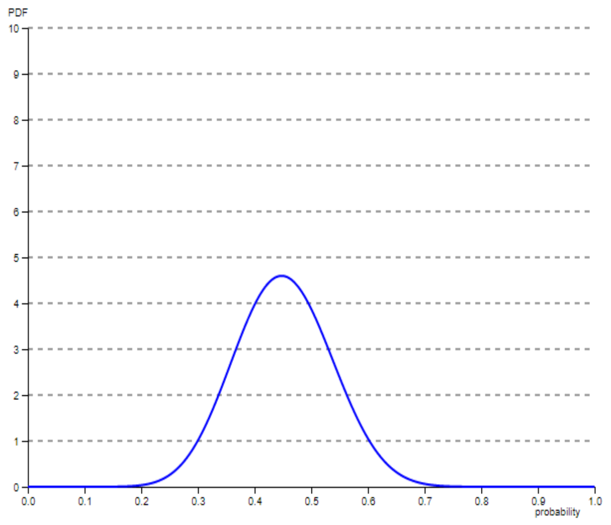


Fig. 13: When there is a strange behaviour on execution of (L2 MAC application) for one anomaly $Trust_{IP\_L2MAC}$

.

MAC application, the disbelief mass (d) decrease to 0.9 and the base rate that represents the subjective trustworthy of the execution environment will reduce to 0.225 based on the trust discounting operation of a second anomaly. The trust values are (*b=0.06, d=0.9, u = 0.04. a=0.225*), with disbelief mass as 0.9 the controller will tag the application as malicious and untrustworthy.

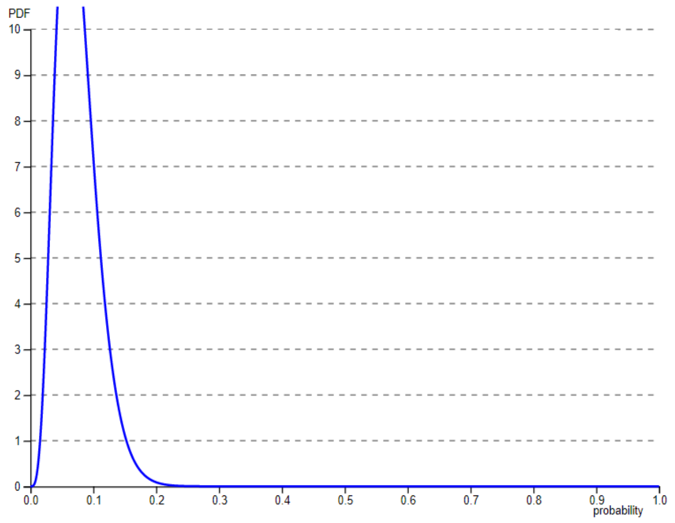


Fig. 14: Multiple violation of trust by L2 MAC application $Trust_{IP\_L2MAC}$ (multiple anomaly)

The trust representation on Figure 15 shows a trusted relationship with the controller and the IP blacklist application without any anomaly. The application trust equivalent is as follows (*b=0.9, d=0.1, u=0, a=0.9*), with the high belief mass the application is running based on the expected behaviour.

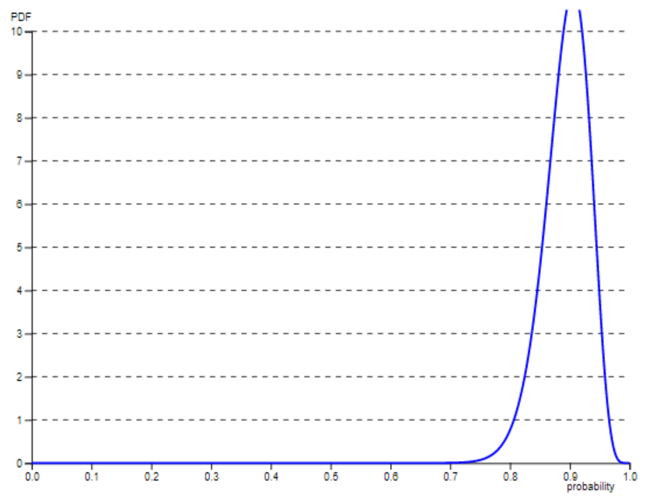


Fig. 15: Normal application behaviour of IP Blacklist application $Trust_{IP\_BLK}$

In Figure 16, the trust equivalent ω_k, is in the uncertain zone. If visualised in the barycentric triangle the point will be at the vertex near uncertainty mass (u). This signifies uncertainty with high confidence. The trust opinion values (b,d,u) show very low belief and disbelief mass with high uncertainty mass. For every proposition that lies within this range, the best possible deduction is to reject the trust outcome. The continuous straight line on the graph shows the uncertainty with high confidence, and the line will remain constant as more uncertainty increases.

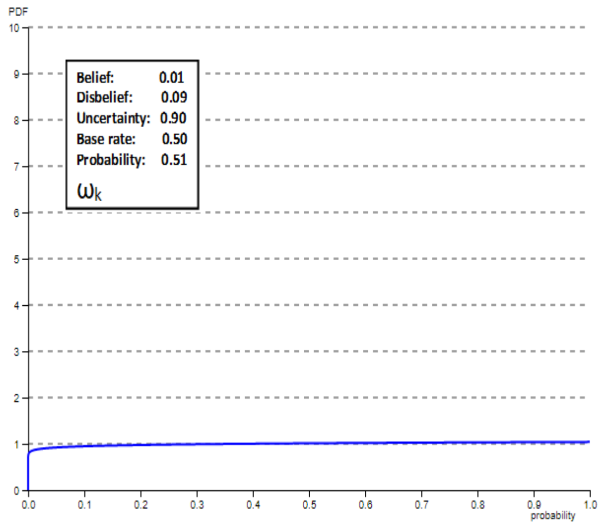


Fig. 16: When there is no trust in (L2 MAC application) $Trust_{IP\_L2MAC}$

.

Figure 17 presents an ambiguous trust point, it is not a complete uncertain trust equivalent. However, there is much at stake with very low confidence in the trust. And it is a bit away from the belief mass which can add some weight to the overall

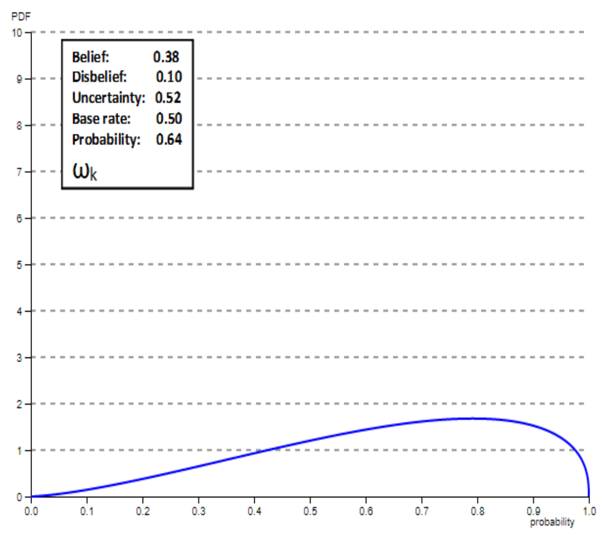


Fig. 17: When there ambiguous trust in (L2 MAC application) $Trust_{IP\_L2MAC}$

.

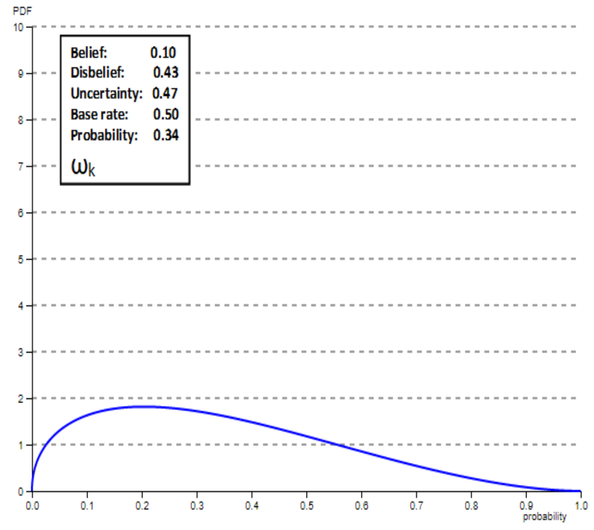


Fig. 18: When there ambiguous trust in (IP Blacklist application) $Trust_{IP\_BLK}$

.

trust derivation. The main action to take when faced with this kind of decision is to distrust the proposition. Trust can only be accepted when it has high confidence and reliability, and this can be realised when the belief mass is more than 90%, where both the disbelief and uncertainty mass are negligible.

The derived trust in Figure 18 that falls between the two extremes of disbelief and uncertainty mass has very low confidence, and consequently, the reliability is weak. Because the belief mass is low, which indicates that either the uncertainty or the disbelief or both are high. Equivalent trust like this should be avoided when considering the confidence and reliability of the proposition. The graph concentrates on the left-hand side and extends until it decays at the end. This signifies region between uncertainty or disbelief.

VI. Evaluation With Existing Models

This section provides a comprehensive evaluation of the proposed framework with other existing models that attempt to mitigate similar problem of controller to network application security. In every research work, it is very crucial to evaluate and validate the model or framework built. Verification and Validation (V&V) is paramount in evaluating the usefulness of any proposed framework in both research and production environment. The main objective of (V&V) is to carry out robust set of tests on the target framework and provide results in return [47]. The returned results assist in evaluating how useful, trustworthy, relevant and the impact of the proposed framework based on the problem it is trying to address.

In SDN environment the V&V platform used is called *Cbench* [48] [49]. It is a widely used model that checks how the SDN controller framework executes the series of assigned tasks and provides performance and latency constraint regrading any workload executed by the controller [50]. In this context the workload is our proposed trust framework. Cbench stands as the best candidate to carry out the V&V tests.

A. ***Framework Components and Cbench***

The proposed trust framework introduces three main modules mainly authentication, authorisation and the trust module. The Cbench will help provides a robust validation mechanism where by network applications are initiated and the Cbench tool will check for framework correctness and report on the performance constraint caused by introducing the modules in the SDN control layer. Cbench provides standardised tests for (*throughput* and *latency*) that work across frameworks built in SDN domain.

For the purpose of validation, the trust framework is integrated in RYU controller platform with the help of the pseudo codes. The pseudo codes provides high level view of the trust framework objectives and as such can be port to different programming environment. This modularity is enabled by Object Oriented Programming because it helps in building reusable and portable code base for any software implementation. Ryu is a component based SDN controller written in python, it is a modular framework with well defined APIs that simplifies network management and control applications [51] [5].

B. *Experiment Setup*

Three experiments will be carried out in order to verify the correctness and performance of the trust framework. The main essence of using RYU controller is to provide a fair contest between the trust framework and other frameworks that tackle similar problem Rosemary[27] and FortNox[52] respectively.These two frameworks as discussed in Section III (Related Work) are chosen because they are available in public domain and share similarities in terms of applied methods and techniques. They will be run in the same emulation environment and the same validation tool *Cbench* with attached switches and hosts.

The evaluation is carried out on RYU controller with firmware version 4.3.0 running inside Mininet [53] emulation environment and is hosted on a Virtual Machine (VM) Ubuntu 16.04 64-bit with processor Intel(R)Core(TM) i5-4200M, CPU processing speed of 2.50GHz and installed RAM capacity of 16.0 GB. The three experiments that will be conducted are:

- Experiment 1: Cbench throughput experiments that involves varying number of switches from (1-32).
- Experiment 2: Cbench latency experiment for switches (1-100).
- Experiment 3: Write intensive workload with hosts.
- Test for Reliability and Dependability

1) ***Experiment 1****:* The objective of this experiment is to stress test the correctness of the trust framework and determine the computational impact of running *authentication, authorisation* and *trust* between the controller and the network applications. To baseline the experiment an L2 MAC learning application (*simple_switch.py*) from the RYU controller is deployed to run without the trust framework in place. This initial experiment can help us understand the network behaviour and the initial performance because no additional workload on the side of the RYU controller.

The *simple_switch.py* is compiled and run in Cbench throughput mode and to minimise the extra overhead the logging functions are disabled. When Cbench is set to throughput mode, the command control of Cbench executes an algorithm that dictates if the buffers of the controller are not full, then keep forwarding *packet_in* requests and count successful installed flows (*flow_mod*) during that time. The experiment procedures are seen as follows:

```
ryu-manager simple_switch.py

cbench -p6653 -s [1, 4, ..32] -M -t
```

Name	Description
ryu-manager	Controller Module
Cbench	Validation tool
-p	Port number
-s	Number of Switch
-t	Throughput mode
-M	Number of Hosts
simple_switch	Application

TABLE XII: Experiment Arguments

The arguments used for the experiments are seen on Table XII. Figure 19a with lines-plot of (w/o framework) shows the average plot of flows installed wihtout the trust framework based on Cbench throughput test carried out for different number of switches (1, 4, 8, 16 and 32) respectively. Based on the obtained results it shows that there is decrease in the number of flows installed by the controller as more switches connect and send flow requests to be processed. The drop was not linear in practical, it shows a clear sign of reduced number of flows processed by the controller. This presents just the plain behaviour of the L2 MAC application within the RYU controller framework without the trust framework in place.

When the trust framework is involved in flow processing where verification and checks regarding how authentication, authorisation and trust are carried out. The command control module in Cbench that emulates flow requests will experience a processing delay in flow installation due to additional overhead introduced by the trust framework validations. There is no single case of control layer failure or crash due to sequence of verification carried out when *packet_in* are processed by the controller, despite several calls to APIs for network application verification introduced by the trust framework. This clearly shows a strong chaining and seamless integration of the modules within the control layer. There is a minimal drop (0.06%) in throughput due to the total number of recorded flows installed. The effect of having more active switches impact the throughput of flow processing, as the number of switches increase beyond certain threshold as seen in Figure 19a.

Running the same experiment for Rosemary and Fortnox, the evaluation from Cbench for Rosemary shows a drastic fall in throughput, this is possible due to numerous modules run by Rosemary to achieve threat containment and control layer resiliency. These modules are resource manager, security manager, kernel and the AppZone. There is a considerable amount of overhead when flows are traversing the control stack and Rosemary is checking for threats and at the same time implementing possible mitigation strategy. As more switches are connected to the network with massive flow requests the more the throughout in successful flow rule insertion reduces. However when no security checks are implemented Rosemary tend to perform better.

In the case of FortNox, when using different number of switches to test the computational impact of running the conflict analysis algorithm. At the initial stage FortNOX performs efficiently with high throughput, however as switch contention rate for resources increases with TCP connections set up and tear down the throughput degrades. There is quite intensive processing done by Fortnox which makes flow requests authentication to queue up as conflict verification engine implements the contradiction algorithm (ARR). The Cbench is constantly overwhelming the control channel with tremendous flow requests that is why the throughput for flow rule insertion degrades. The cumulative result for *experiment 1* is seen on Figure 19a.

A comparison of the average throughput (mean) and standard deviation of flows installed with varying number of switches for all the frameworks is presented on Table XIII. Taking the results of the trust framework for running 1 switch, the mean(μ) and standard deviation (σ) are 3100 and 220.30 respectively. Applying the data analysis method of *68-95-99.7* [54][55] through adding and subtracting the standard deviation(σ) from the mean(μ) will give an insight into how the flow rule insertion are distributed during the experiment. Adding the standard deviation (σ) will give 3220.30 and subtracting the (σ) from the mean (μ) gives 2879.7. This shows that 68% of the flows lies within these two bounds, by further adding (μ + 2σ) and subtracting (μ - 2σ) the standard deviation twice (2σ) the results give (2659.4, 3540.6) then the 95% of installed flow rules per second are within these two bounds. For the last bound applying same rule by adding and subtracting the mean three times (3μ) then 99.7% of the flow rules installed are within that range. The analysis applies to other data set for 4, 8, 16 and 32 switches.

*2) **Experiment 2:*** The aim of this experiment is to determine the latency incurred in processing number of successful flows per second when running the trust framework. This can be achieved through the Cbench latency mode, the algorithm works by sending several *packet_in* as defined in the experiment procedure, then wait for a return *flow_mod* (flow installation) and record the gap in response time. The experiment makes use 0 -100 emulated switches and provides the average delay in installing flow rules without the trust framework (default) and with the trust framework integrated.

The switches used in this experiment increment in the range of 10s. This will provide the flexibility in predicting the behaviour of the model with varying number of switches. In latency mode without the framework, Cbench initiates a synchronous flow requests to the controller, then the command control waits and calculates the time the controller takes to respond to those requests. Even without the trust framework there are default processes and jobs the controller executes which can take a toll against the response time. Some of these processes are:

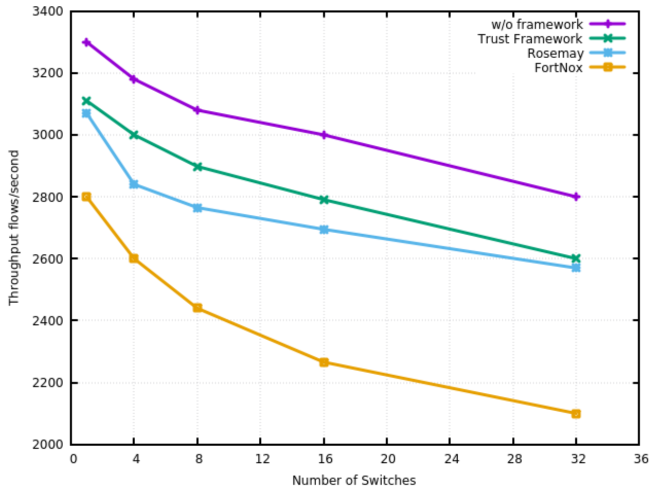
- Contention rate of TCP connections between switches and controller.
- Frequent hello messages for keep alive.
- Input and Output batch CPU process.
- Memory allocation.

Figure 19b shows the average latency when the trust framework is integrated, the setup environment and the experiment procedures are the same. In latency mode the Cbench command control will experience additional overhead due to the checks being carried out by the trust framework. For every iteration or loop that involves a new flow processing a verification is carried out whether that function is feasible as allowed by the trust framework. As the load increases the growing delay in flow response time increases in a near linear manner. The latency involves traversals of several flow processes twice from the networking stack on both the command control of the Cbench and that of the controller. The average response time of most SDN controllers is around 100-150 millisecond[49].

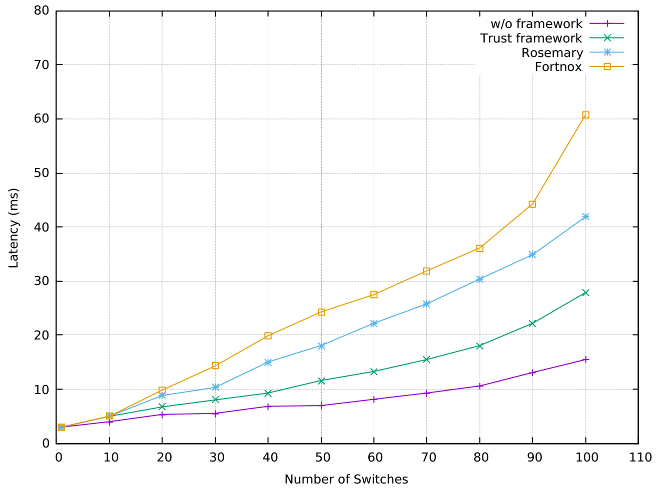
For the other frameworks starting with Rosemary there is a near latency relationship for both Rosemary and the trust framework at the beginning, this shows a correlation that they perform optimally at initial stage, however looking at Rosemary curve as more load is added the controller response time lags due to overhead processing caused by flow rule traversing several stacks of Rosemary modules. The correctness is still maintained because at no point during the time where Rosemary crashed, the processes of identifying

	1	4	8	16	32
W/o Framework	3378 ± 69.97	3181 ± 388	3067 ± 129.16	3053 ± 139.32	2800 ± 65.13
Trust Framework	3100 ± 220.30	3000 ± 365.4	2890 ± 415.89	2800 ± 366.73	2602 ± 485.6
Rosemary	3080 ± 84.15	2820 ± 105.4	2760 ± 335.9	2688 ± 256.3	2587 ± 154.21
FortNox	2804 ± 119.86	2598 ± 88.5	2411 ± 65.9	2240 ± 256.3	2092 ± 71.35

TABLE XIII: Mean and Standard Deviation of Flows Installed with Varying Number of Switches for Experiment 1

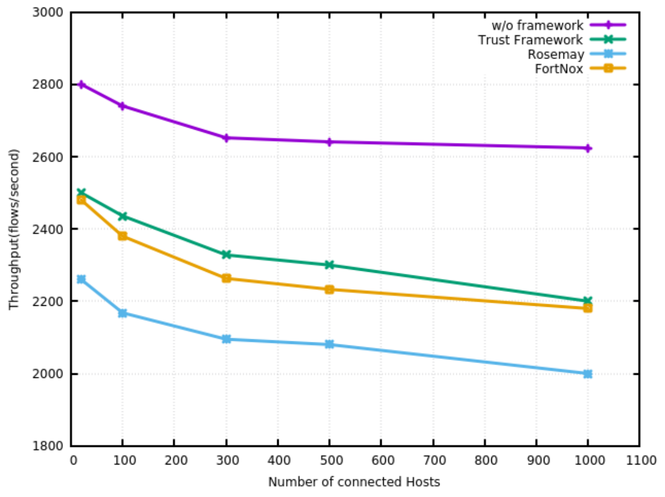


(a) Experiment 1

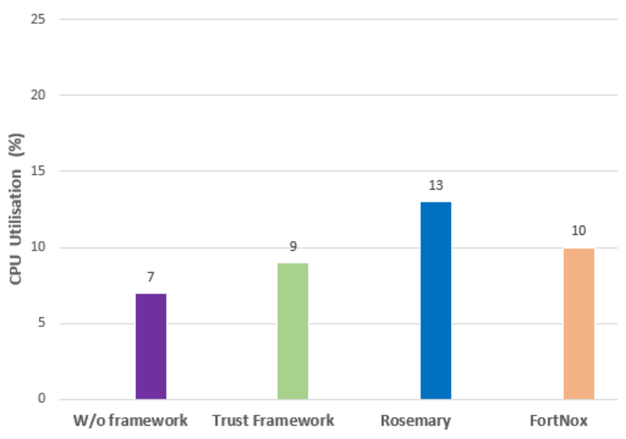


(b) Experiment 2

Figure 18: Average Throughput and Response Time (Latency) Comparison for Different Frameworks



(c)



(d)

Figure 18: (c) Write Intensive Workload, (d) CPU Utilisation on Runtime

network applications, checking their permissions and isolating malicious applications in containers affect the processing time.

In the case of FortNox the initial experiment with just one (1) switch turns out to have a swift response time, because that is the minimum response regarding growing number of switches. Up until the number of switches are 30, the flow installation response time is almost similar with both the trust framework and Fortnox. However at an interval when there is increasing number of switches then the response time starts to increase with more delay. The cumulative plot is seen on Figure 19b.

3) ***Write Intensive Workload:*** The objective of this experiment is to test the robustness of the trust framework against intensive workload when there are more connected hosts in the network. The answers that can come out of this experiment will be whether the trust framework will stand the overhead of several flow processing while still achieving the goal of security. The write intensive workload increases the contention in the network applications, with the L2 MAC application

having large number of hosts both Rosemary, FortNox and the trust framework are affected by the load condition introduced by the Cbench. However in this experiment the overwhelming flow requests initiated by Cbench at different iteration shows that Fortnox handles scaling number of hosts efficiently. The throughput turns out to be better than that of Rosemary, this is because if there is no flow mediation the flow installation will continue optimally.

	10	100	1000
W/o Framework	2803 ± 72.27	2724 ± 95.15	2695 ± 108.46
Trust Framework	2504 ± 32.45	2401 ± 341.66	2199 ± 87.37
Rosemary	2240 ± 125.35	2091 ± 41.66	2001 ± 54.37
Fortnox	2488 ± 119.86	2393 ± 87.14	2188 ± 124.04

TABLE XIV: Mean and Standard Deviation of Flows Installed with Varying Number of Hosts

For the trust framework when dealing with high number of hosts, the workload increases the contention in the network application. By probing the network application the steps involve in installing flows will vary due to additional overhead caused by authentication, authorisation and trust evaluation. The Cbench command control will experience a reduced number of flows processing per second due to the series of verification at the control layer before flows are installed as per host requests. Figure 19c presents the cumulative plot of write intensive workload for all the frameworks and Figure 19d presents CPU utilisation of compiling and executing the experiments for all the frameworks. Table XIV provides the mean and standard deviation of the successful flows installed for write intensive workload with different number of hosts (10,100,1000, Figure 19c provides more result data points for 300 and 500 hosts.

*4) **Test for Reliability and Dependability**:* To report on the reliability and dependability of the proposed trust framework, the ***Binomial Distribution Model*** is used to predict the probability of success and failure [56]. However there are ground rules that must be in place to actualise the application of Binomial Distribution Model and they are:

- *Independent*: The result of one experiment does not affect the result of another experiment.
- *Repetition*: Conditions for experiment are the same.

These two conditions align with the evaluation of the proposed Trust framework with Rosemary and FortNox based on the conducted experiments. The first condition is about independence where the individual experiments were carried out separately and they are all independent of each other. The experiments satisfy the second requirement of binomial distribution model which is *Repetition*. In this context the repetition translates to the condition of experiments, it should be the same and this has been satisfied as stated in section VI-B.

The experiments were all conducted under the same testing environment and condition. With the two conditions satisfied, the binomial distribution model is given as follows:

$$\binom{n}{r} p^r q^{n-r} = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$$

p = Probability of success.
q = Probability of failure.
n = Number of observations.
r = Number of success for n observations.
Where $p + q = 1$,
And $p = 1 - q$

To obtain the reliability metrics, the probability of success (P) must be established. Based on the experiments conducted the projected (P) for the different frameworks is seen on Table XV. The probabilities are derived from how many modules each framework has that address authentication, authorisation and trust.

Framework	Prob. of Success (p)
Rosemary	0.75
Trust framework	0.87
FortNox	0.60

TABLE XV: Experiment Arguments

To evaluate the reliability, if four (4) different experiments will be conducted for all the frameworks (Rosemary, FortNox and the Trust framework). The number four (4) is used just to get an approximate value of the reliability but the experiment can be run more than 4 times. The deductions based on the binomial distribution model will be to either get the probability of failure or success. However in this case we will start by evaluating failure first, from the obtained probability of failure the probability of success can be derived. The probability that any of the framework will fail that is $r=0$ for all is:

$$Trust = \binom{4}{0} p^0 q^{4-0} = \frac{4!}{0!(4-0)!} 0.87^0 (0.13)^{4-0} = 0.00028$$

$$Rosemary = \binom{4}{0} p^0 q^{4-0} = \frac{4!}{0!(4-0)!} 0.75^0 (0.25)^{4-0} = 0.0039$$

$$FortNox = \binom{4}{0} p^0 q^{4-0} = \frac{4!}{0!(4-0)!} 0.60^0 (0.40)^{4-0} = 0.0256$$

With the obtained values, it clearly shows how the chance of faiure in the Trust framework is very negligible then followed by that of Rosemary and finally FortNox. To get the probability of success:

- $p_{trust} = 1 - q_{failure}$
- $p_{rosemary} = 1 - q_{failure}$
- $p_{fortnox} = 1 - q_{failure}$

The probability of success for p_{trust} after application is 0.9997 and that of $p_{fortnox}$ is 0.9744 and finally for $p_{rosemary}$ is 0.9961. Looking at both the metrics of success and failure in terms of reliability and dependability, it can be concluded that the proposed trust framework has a very low chance of

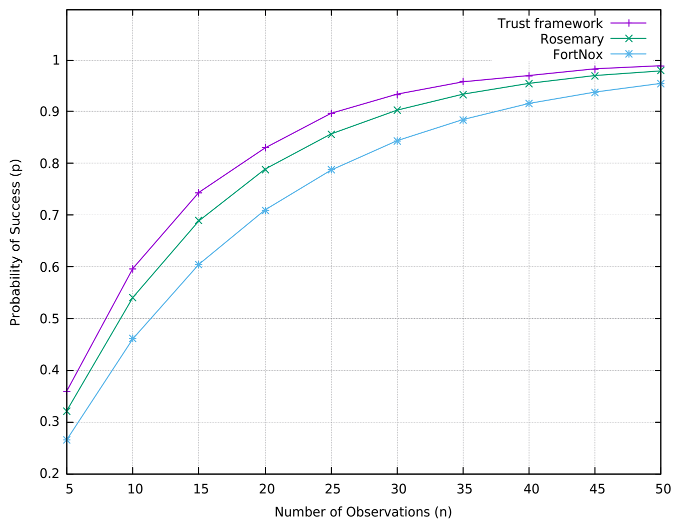


Fig. 20: The reliability metrics for Trust Framework, Rosemary and FortNOX.

failure and a very high success rate than both Rosemary and Fortnox.

For more robust and resilient tests for reliability and dependability, repeated experiments have been conducted where observations (n) ranges from 0-50 to ascertain the level of failure or success as seen on Figure 20. Several instances of the three frameworks (Trust, Rosemary and FortNox) in action and the Trust framework suffices to have more chance of being active without failure than Rosemary and FortNox. At the initial stage of the experiment the reliability and confidence level of trust framework is a bit higher, however towards the end the reliability seems to converge in to a near linear curve, though at that level the frameworks are all reliable however the Trust framework is more reliable.

VII. Conclusion & Future Work

The proposed trust framework aims at addressing a threat in the SDN architecture that exists between the controller and the network applications. This research work brings the abstract and design concept of the trust framework into implementable artefact that can be evaluated, tested and analysed. The proposed trust framework introduces methods through which the authenticity of network applications situated in Network Function Virtualisation layer are verified and respective privilege permissions for successfully authenticated applications are assigned. Trust evaluation and assessment are carried out based on how the network application conform and adhere to network policy, application can be flagged and denied if there is violation of network policy. A comparative and comprehensive analysis with other frameworks is presented in order to discover the impact, usefulness and how dependable the proposed trust framework is. Results shows the remarkable performance of the proposed trust framework and this is step in a right direction towards achieving a secure , dependable and reliable controller to network applications interactions.

Future work will look into implementing the framework in a multi controller environment focusing on establishing trusted relationship with network applications situated at neighbouring controllers for efficient and secure service delivery. In addition production environment implementation can go a long way in reporting about reliability, performance implication of the various stages of verification and the scalability of the proposed framework.

References

[1] K. Thimmaraju, B. Shastry, T. Fiebig, F. Hetzelt, J.-P. Seifert, A. Feldmann, and S. Schmid, "Taking control of sdn-based cloud systems via the data plane," in *Proceedings of the Symposium on SDN Research*. ACM, 2018, p. 1.

[2] K. Sood, K. K. Karmakar, V. Varadharajan, U. Tupakula, and S. Yu, "Analysis of policy-based security management system in software-defined networks," *IEEE Communications Letters*, 2019.

[3] N. Gray, T. Zinner, and P. Tran-Gia, "Enhancing sdn security by device fingerprinting," in *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*. IEEE, 2017, pp. 879–880.

[4] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *IEEE Journal & Magazine*, vol. 103, no. 1, pp. 14–76, jan 2015. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6994333

[5] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 393–430, 2018.

[6] S. Betge-Brezetz, G. B. Kamga, and M. Tazi, "Trust support for SDN controllers and virtualized network applications," *1st IEEE Conference on Network Softwarization: Software-Defined Infrastructures for Networks, Clouds, IoT and Services, NETSOFT 2015*, pp. 0–4, 2015.

[7] A. Lara and L. Quesada, "Performance analysis of sdn northbound interfaces," in *2018 IEEE 10th Latin-American Conference on Communications (LATINCOM)*. IEEE, 2018, pp. 1–6.

[8] S. Lee, C. Yoon, and S. Shin, "The smaller, the shrewder: A simple malicious application can kill an entire sdn environment," in *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*. ACM, 2016, pp. 23–28.

[9] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang, "Towards a secure controller platform for openflow applications," *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, p. 171, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2491185.2491212

[10] Y. Shi, F. Dai, and Z. Ye, "An enhanced security framework of software defined network based on attribute-based encryption," in *Systems and Informatics (ICSAI), 2017 4th International Conference on*. IEEE, 2017, pp. 965–969.

[11] A. L. Aliyu, P. Bull, and A. Abdallah, "Performance implication and analysis of the openflow sdn protocol," in *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. IEEE, 2017, pp. 391–396.

[12] J. L. G. Gomez, T.-C. Chang, C.-F. Chou, and L. Golubchik, "On improving the performance of software-defined networking through middlebox policies," in *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2019, pp. 1–4.

[13] J. Suárez-Varela and P. Barlet-Ros, "Sbar: Sdn flow-based monitoring and application recognition," in *Proceedings of the Symposium on SDN Research*. ACM, 2018, p. 22.

[14] A. R. Abdou, P. C. van Oorschot, and T. Wan, "Comparative analysis of control plane security of sdn and conventional networks," *IEEE Communications Surveys & Tutorials*, 2018.

[15] R. Hamed, B. Mokhtar, and M. Rizk, "Towards concern-based sdn management framework for computer networks," in *National Radio Science Conference (NRSC), 2018 35th*. IEEE, 2018, pp. 121–129.

[16] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A Survey of Security in Software Defined Networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 1–33, 2015. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7150550

[17] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang, "Towards a secure controller platform for openflow applications," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 171–172.

[18] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 13–18.

[19] C. Yoon, T. Park, S. Lee, H. Kang, S. Shin, and Z. Zhang, "Enabling security functions with sdn: A feasibility study," *Computer Networks*, vol. 85, pp. 19–35, 2015.

[20] ONF, "Version 1.0. 0 (wire protocol 0x01)," 2009. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf

[21] S. Scott-Hayward, C. Kane, and S. Sezer, "Operationcheckpoint: Sdn application control," in *2014 IEEE 22nd International Conference on Network Protocols*. IEEE, 2014, pp. 618–623.

[22] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software-defined networks," pp. 1–14, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2482626.2482629

[23] B. E. Ujcich, S. Jero, A. Edmundson, Q. Wang, R. Skowyra, J. Landry, A. Bates, W. H. Sanders, C. Nita-Rotaru, and H. Okhravi, "Cross-app poisoning in software-defined networking," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 648–663.

[24] F. Klaedtke, G. O. Karame, R. Bifulco, and H. Cui, "Access control for SDN controllers," *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN '14*, pp. 219–220, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2620728.2620773

[25] R. Nagai, W. Kurihara, S. Higuchi, and T. Hirotsu, "Design and implementation of an openflow-based tcp syn flood mitigation," in *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2018 6th IEEE International Conference on*, 2018, pp. 37–42.

[26] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN '12*, p. 121, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2342441.2342466

[27] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang, "Rosemary: A robust, secure, and high-performance network operating system," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. ACM, 2014, pp. 78–89.

[28] A. L. Aliyu, P. Bull, and A. Abdallah, "A trust management framework for network applications within an sdn environment," in *Advanced Information Networking and Applications Workshops (WAINA), 2017 31st International Conference on*. IEEE, 2017, pp. 93–98.

[29] K. Benton, L. J. Camp, and C. Small, "OpenFlow vulnerability assessment," *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*, pp. 151–152, 2013.

[30] N. Samarasinghe and M. Mannan, "Short paper: Tls ecosystems in networked devices vs. web servers," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 533–541.

[31] C. Cremers, M. Horvat, S. Scott, and T. van der Merwe, "Automated analysis and verification of tls 1.3: 0-rtt, resumption and delayed authentication," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 470–485.

[32] N. Khare, N. Adlakha, and K. Pardasani, "An algorithm for mining multidimensional association rules using boolean matrix," in *2010 International Conference on Recent Trends in Information, Telecommunication and Computing*. IEEE, 2010, pp. 95–99.

[33] V. Snášel, J. Platoš, and P. Krömer, "On genetic algorithms for boolean matrix factorization," in *2008 Eighth International Conference on Intelligent Systems Design and Applications*, vol. 2. IEEE, 2008, pp. 170–175.

[34] A. Jsang, "Subjective logic: A formalism for reasoning under uncertainty," 2018.

[35] M. Ivanovska, A. Jøsang, and F. Sambo, "Bayesian deduction with subjective opinions." in *ACM Proceedinds KR16 Fifteenth International Conference on Principles of Knowledge Representation and Reasoning*. ACM, 2016, pp. 484–493.

[36] A. Jøsang, "The consensus operator for combining beliefs," *Artificial Intelligence*, vol. 141, no. 1-2, pp. 157–170, 2002.

[37] F. Firoozi, V. I. Zadorozhny, and F. Y. Li, "Subjective logic-based in-network data processing for trust management in collocated and distributed wireless sensor networks," *IEEE Sensors Journal*, vol. 18, no. 15, pp. 6446–6460, 2018.

[38] S. Muhammad, L. Wang, and B. Yamin, "Trust model based uncertainty analysis between multi-path routes in manet using subjective logic," in *China Conference on Wireless Sensor Networks*. Springer, 2017, pp. 319–332.

[39] M. Ivanovska, A. Jøsang, L. Kaplan, and F. Sambo, "Subjective networks: Perspectives and challenges," in *International Workshop on Graph Structures for Knowledge Representation and Reasoning*. Springer, 2015, pp. 107–124.

[40] A. Jøsang, T. Ažderska, and S. Marsh, "Trust transitivity and conditional belief reasoning," in *IFIP International Conference on Trust Management*. Springer, 2012, pp. 68–83.

[41] A. Jøsang, S. Marsh, and S. Pope, "Exploring different types of trust propagation," in *International Conference on Trust Management*. Springer, 2006, pp. 179–192.

[42] A. Josang, "Conditional reasoning with subjective logic," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 15, no. 1, pp. 5–38, 2008.

[43] I. T. Javed, K. Toumi, and N. Crespi, "Trustcall: A trust computation model for web conversational services," *IEEE Access*, vol. 5, pp. 24 376–24 388, 2017.

[44] S. K. Awasthi and Y. N. Singh, "Generalized analysis of convergence of absolute trust in peer-to-peer networks," *IEEE Communications Letters*, vol. 20, no. 7, pp. 1345–1348, 2016.

[45] Y. Pan, F. He, and H. Yu, "An adaptive method to learn directive trust strength for trust-aware recommender systems," in *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design ((CSCWD))*. IEEE, 2018, pp. 10–16.

[46] H. S. Raju and S. Salim, "An efficient trust model for detecting trusted nodes in peer to peer systems," in *Emerging Technological Trends (ICETT), International Conference on*. IEEE, 2016, pp. 1–5.

[47] M. Zhao, F. Le Gall, P. Cousin, R. Vilalta, R. Muñoz, S. Castro, M. Peuster, S. Schneider, M. Siapera, E. Kapassa *et al.*, "Verification and validation framework for 5g network services and apps," in *Network Function Virtualization and Software Defined Networks (NFV-SDN), 2017 IEEE Conference on*. IEEE, 2017, pp. 321–326.

[48] C. Laissaoui, N. Idboufker, R. Elassali, and K. El Baamrani, "A measurement of the response times of various openflow/sdn controllers with cbench," in *Computer Systems and Applications (AICCSA), 2015 IEEE/ACS 12th International Conference of*. IEEE, 2015, pp. 1–2.

[49] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," *Proceeding Hot-ICE'12 Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, pp. 10–10, 2012. [Online]. Available: https://www.usenix.org/system/files/conference/hot-ice12/hotice12-final33_0.pdf

[50] B. Xiong, K. Yang, J. Zhao, W. Li, and K. Li, "Performance evaluation of openflow-based software-defined networks based on queueing model," *Computer Networks*, vol. 102, pp. 172–185, 2016.

[51] F. Tomonori, "Introduction to ryu sdn framework," *Open Networking Summit*, 2013.

[52] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN '12*, p. 121, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2342441.2342466

[53] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," *… Workshop on Hot Topics in Networks*, pp. 1–6, 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1868466

[54] D. S. Moore, *The basic practice of statistics*. Palgrave Macmillan, 2010.

[55] H. Zhang, Z. Liu, H. Huang, and L. Wang, "Ftsgd: An adaptive stochastic gradient descent algorithm for spark mllib," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2018, pp. 828–835.

[56] K. D. Trivedi, N. J. Patel, and P. R. Shah, “Binomial distribution approach for efficient detection of service level violations in cloud,” in *2013 Nirma University International Conference on Engineering (NUiCONE)*. IEEE, 2013, pp. 1–6.