



A Novel Approach for Standardizing Authentication Schemes in OpenDayLight Controller

Shakti Kumar¹, Sneha M.², Dr. Shobha G.³

Student¹, Faculty^{2,3}

Department of Computer Science and Engineering,
R V College of Engineering, Bangalore
INDIA

Abstract: Software Defined Networks (SDN) is bringing a drastic shift in the way people are looking upon networking, by separating the control plane from the data plane. With emerging new technologies dynamic network programming is becoming absolutely essential. SDN provides this capability to the network administrators, where they can change their network behavior dynamically and leverage the network utilization. With the advantages of SDN follow the security risks. Hence OpenDayLight provides AAA architecture for Authentication, Authorization and Accounting, which mandates users to provide username and password credentials in order to access the network resources. Here we propose a novel approach to standardize this authentication scheme, and allow users to use their own preferred credentials, rather than restricting them to username and password, to access the network through OpenDayLight controller. For evaluating the proposed methodology, we have used biometric (fingerprint) as one of the authentication schemes in this work.

Keywords: Software Defined Networks, OpenDayLight Controller, AAA service, Authentication.

I. Introduction

Networks have been an absolutely essential element in today's life, be it business or home. Whether the network is completely on-premises, cloud-based, or a hybrid of both, they have been providing critical communication links that organizations need to deliver required services, run applications etc. Software defined networking (SDN) has revolutionized the way we look at how networks control the resources and operate. This has been possible due to the concept of a centralized management plane which SDN leverages. This means that networks are controlled by a central software application running in an SDN controller rather than the traditional network management consoles and commands to manage each and every network node independently which required a lot of administrative overhead and were, extremely dreary to manage and maintain. A comparison between conventional networks and software defined networks [1] is clearly visible in Fig. 1.

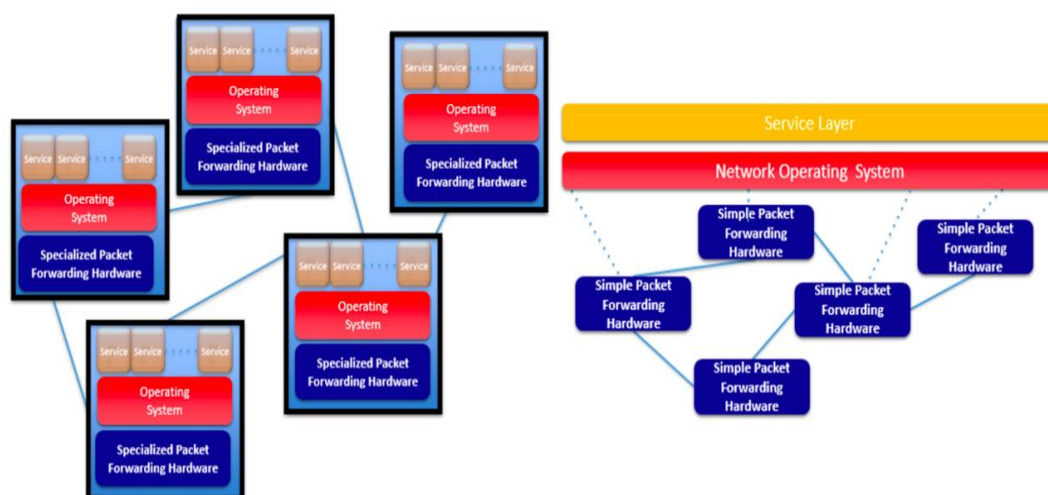


Figure 1 Conventional Networks versus Software Defined Networks

II. OpenDayLight Controller

OpenDaylight Project (ODL) is an open-source SDN project which aims at leveraging software-defined networking through support from communities of leading enterprises in the field of networks. Hosted by the Linux Foundation and with its excellent community support on features, releases and bugs, it brings up a common platform for all enterprises, and individuals with SDN goals to work in a synergistic manner and find new solutions for building programmable networks that are quick in response to organizations' and users' needs. OpenDaylight includes support for the OpenFlow protocol, the first SDN protocol that defined how the control and data plane elements would be separated and would communicate with each other. This not only provides enterprises the flexibility to adapt themselves to the changing demands of the networks, but also exercise improved control over their networks.

A. The AAA Service

AAA is acronym for Authentication, Authorization and Accounting. Put plainly authentication is asking, "Who or what are you?" Authorization asks, "What are you allowed to do?" And accounting poses the question, "What all did you do?". AAA is a term for a framework for decisively controlling access to any resource, enforcing policies, auditing the usage, and providing the information necessary to bill the services a client has made use of [2]. The OpenDayLight implements AAA as one of the layers in its management plane to monitor the network changes and verify the incoming clients. It also makes sure that the client is operating in the domain allowed to him and is not exercising a service outside his allowed rights. The core components of AAA for a general network which also includes ODL controllers are [2],

1) *Policy Enforcement Point (PEP)*: It is synonyms to a firewall, or a gateway that monitors and the access and usage of the network by the client. It is the Policy Enforcement Point that is responsible for applying specific rules as decided by the controller administrator to the traffic originating from the client.

2) *Policy Information Point (PIP)*: This node is basically a storehouse of information which is used to decide whether the incoming request to access the controller from the client should be allowed or not. There are different ways in which this Policy Information Point can be implemented, some examples being a database containing all device IDs which are allowed to access the controller; some server that provides one time passwords (OTPs) to the requesting entities; or user directories like the Lightweight Directory Access Protocol commonly used for authenticating requests in Linux systems.

3) *Policy Decision Point (PDP)*: This is the main decision point for the access of the client. Responsible for implementing the AAA service, this node collects the access request from the client via the enforcement point. It also collects any information about the requesting client from the stored data in the information points, and decides whether the access request should be allowed or not. Moreover, it also sends any policing rules to the policy enforcement point which should be used to filter out the clients' access terms.

4) *Accounting and Reporting System*: In order to keep track of the network usage the OpenDayLight uses a dedicated node for Accounting and Reporting System. In addition to providing services for tracking the changes made by different clients, this node also informs the administrator the access different clients had once they logged into the network.

B. AAA Call Flow

- 1) The client tries to connect to the controller and is prompted to present his credentials for authentication. The client presents his username and password to the Policy Enforcement Point discussed above to start the verification process.
- 2) The Policy Enforcement Point forwards this credential to the Policy Decision Point for the authentication.
- 3) The Policy Decision Point queries policy information point for any stored information about the requesting client and validates the credentials provided by the client against the information received from the policy information point.
- 4) At this stage, if the client is authenticated as valid, the Policy Information Point sends additional information about the requesting client to the decision point.
- 5) The decision point considers all the information it has learned about the client through the enforcement point, information point and makes an authorization decision, regarding the domain of authority the client has over the network resources.
- 6) The decision point sends the authentication result and authorization information to the enforcement point from where it is acknowledged to the client. Note that this information from the decision point to the enforcement point can also include any policing rules which should be used to filter out the clients' access terms.
- 7) The decision point sends the result of this process to the accounting and reporting system and starts accounting the clients access session.
- 8) The enforcement point start to apply the authorization policies received from the decision point to the messages originating from the client.
- 9) The client, on successful authentication begins accessing the network via the Policy Enforcement Point.

Fig. 2 summarizes the call flows between the different modules in a AAA protected network [2].

Now based on the location of the PEP, PIP and PEP, there are different ways in which the client's authentication can be done with ODL controller. The client can,

- 1) Provide credentials directly to the controller, where the PEP, PDP and PIP are embedded.
- 2) Get a token from a third application. This token is generated on the basis of the rights the client will get after he starts accessing the ODL controller.

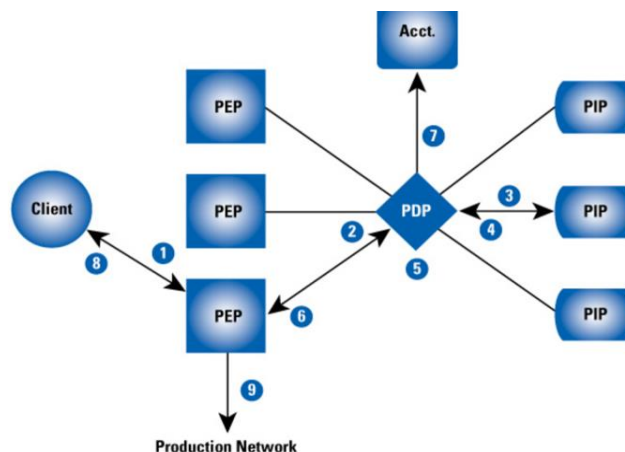
The first option is known as Direct Authentication, and the second option leads us to the concept of Federated Authentication.

C. Direct Authentication

Credential authentication uses user or client credentials (username/password) to authenticate directly with AAA and obtain an access token for accessing resources on the controller. The user request typically undergoes the following steps,

- 1) The user requests the controller administrator for a user account and sends his credentials which will be stored along with this user account. It is this information that the user will be checked against whenever he requests access to the controller.
- 2) Now for logging in to the controller and get access to network resources, the user submits his credentials to the controller.
- 3) This request is forwarded the controller token endpoint.
- 4) The controller token endpoint sends it to the authentication entity which returns a resource claim to the token entity, if at all the credentials match with the one associated with this respective client.

Figure 2 Client Connected to AAA protected network



- 5) The controller token entity converts this received claim into a token and acknowledges the token to the requesting client.

D. Federated Authentication

It is quite obvious that applications should not need to handle the burden of authentication and authorization. Introduction of wide variety of verification mechanisms burden the applications into throwing them away from their normal usage [3]. There are numerous identity providers (IdP) which we can use to verify the users. This leads us to the concept of a federated authentication, where a third-party IdP is used for authenticating the client. In essence, the controller trusts this third party and assumes that all incoming access requests from this IdP are already authenticated. Various example of third party IdPs are Linux SSSD (System Security Services Daemon), Openstack Keystone etc.

- 1) The user is authenticated by the trusted third party identity provider and claim is returned to the ODL authentication services.
- 2) This claim is mapped into users or roles and transformation of the claim takes place into a token which is then passed on to the user.
- 3) The request is forwarded on to the claim authentication broker that transforms it to a claim.
- 4) The ODL controller converts the claim into a token that is passed on to the user. The user is now ready to access the ODL with this token.

E. Lack of Different Authentication Schemes

It is by now clear and precise that the only support for direct authentication is the conventional (username, password) based method. This restriction is not of avail in an enterprise based system where clients may find other authentication schemes like iris recognition, fingerprint recognition easier. The managers of the network may also

find these biometric credential based methods or even other techniques such as OTP+PIN, digital certificates etc. more robust, secure and reliable in verifying the requesting client. Hence in this work a method is proposed for incorporating support for fingerprints in ODL direct authentication scheme.

F. Handling Fingerprints

Processing any biometric credentials has never been an easy task. The wide varieties of complexities involved in the conversion lead to the same biometric impressions giving variations in the final vector produced [4]. To tackle this difficulty, it is always the best approach to take into consideration a number of biometric impressions of the same subject and get the closest match with the credentials received for the authentication purpose.

A database of fingerprints was constructed with 10 fingerprint impressions of each subject. In this construction stage, each subject was asked to give 10 impressions of his fingerprints, then each image was processed as per the sequence listed.

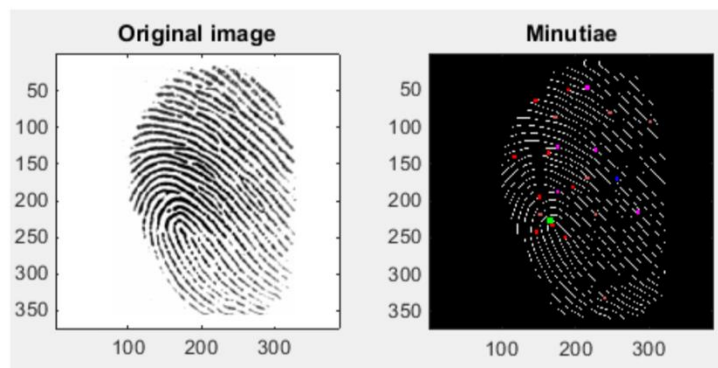
1) Image enhancement

The fingerprints' image which is retrieved using any sensor or scanner has many defects and noise. Hence enhancements for increasing the contrast between ridges and valleys for the fingerprint images is useful to make the next steps of processing easier. The fingerprint image was divided into sets of 6×6 , 12×12 , 24×24 blocks and then we applied Fast Fourier Transform (FFT) on it [4]. The application of fast FFT increased the presence of dominant frequencies in the specific blocks. During enhancement ridge regions in the image were also identified and returned a mask identifying this region. Enhancement also normalized the intensity values of the image so that the ridge regions had zero mean and unit standard deviation.

2) Thresholding

This steps basically reduced the image from 256 gray levels to just 2 gray levels, i.e. a black and white image, based on the value of a threshold gray level. This is required for the purpose of identifying the true ridges as ridges and true valleys as valleys. All the pixel values with gray level above threshold were taken as gray level 1, and rest all pixels were assigned with gray level 0. The main problem in performing in thresholding lies in the selection of the threshold value, which needs to be applied to the entire image. This specific value of threshold is difficult to find because of the absence of a single contrast characteristics for the entire image [4].

Figure 3 Finding Minutiae in the Fingerprint



An elegant solution of this problem is using adaptive local thresholding. Hence, we divided the image into blocks and the mean intensity value was calculated for each block. Next each pixel was pulled to 1 if its intensity value was larger than the mean intensity of the block in which the pixel existed, else it was dropped to 0.

3) Finding Minutiae

This phase involved 2 passes. In the first pass, ridge thinning was done, and in the second pass the minutiae points were marked. For ridge thinning redundant pixels were marked [4] in each 3×3 block and these pixels values were dropped to 0. The thinned ridge map was then filtered by morphological operations to remove isolated points and spikes and smoothen the entire image. Hence the single-point ridges or single-point breaks in a ridge got eliminated and the ridges in the fingerprint image came out to be a single pixel wide. In our work Raymond Thai algorithm was used for finding the Crossing Number (CN) [5]. The minutiae points are thus those which have a pixel value of one (ridge ending) as their neighbor or more than two ones (ridge bifurcations) in their neighborhood. Depending on the CN of the 3×3 block, we marked the points as one of the types of minutiae. Together with the minutia marking, all thinned ridges in the fingerprint image were labeled with a unique ID for further operation.

4) Removing False Minutiae

The minutiae points obtained in the above step contained many spurious minutiae. This frequently occurs due to insufficient amount of ink and ridge cross connections due to over inking. These false points significantly affect the final matching process if they are regarded as a valid feature point. The cases that can produce false minutiae are taken as mentioned in [4].

The final image produced after the entire processing was done is as shown in Fig. 3

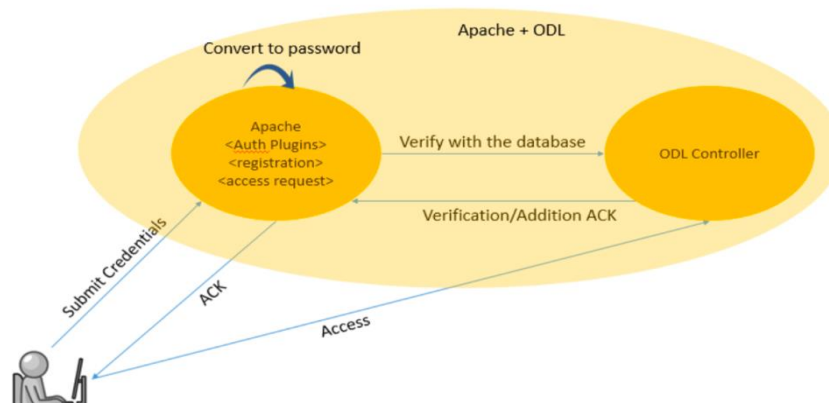
III. Towards the SDN Controller

Having handled the fingerprints of the incoming users for an access request to the controller, we now move onto the SDN controller. As depicted in the Fig. 4 the user logs onto an Apache server via a web application. The Apache server is hosted at the same machine as the ODL controller. This server hosts web applications to handle fingerprints as mentioned in the previous section. In our work we developed MATLAB codes for processing the fingerprints. The user clicks on the appropriate links to submit the type of credentials he has, example fingerprints, iris image, etc. This image is attempted to match with one of the images in the stored database. A score is returned from the fingerprints database, which denotes the similarity measure the uploaded credential has with the ones stored in the database. In our demonstration the image with which the similarity score was more than 60% was assumed to be matching. Now the web application where user submitted his credentials attempts establishing connection with the SDN controller with a username and password (obtained from an SQL database discussed below) that matches the identity of the requesting user.

A. Security of the stored credentials

All the users' credentials are stored, namely the username and password, in an SQL database at the Apache HTTP server. As soon the user's fingerprint image matches with one of the images stored in the database, the associated username and password is retrieved from this database, and forwarded to the ODL controller for a request access.

Figure 4 Call Flows in the Proposed Model



The fronting of the ODL controller with an Apache server was advantageous with respect to security of the stored fingerprints and passwords also. Apache supports a hardened security framework through many open-source modules, some of them being,

1) Protecting Server Files

Apache HTTP server provides ways to block access to the directory files through URLs. In general if the server can find its way to a file through normal URL mapping rules, it can serve it to clients. This may be a result of coding negligence in the server script which can lead to clients being able to walk through the entire file- system. Apache server provides 'all denied' option which can be included in the server configuration to disable access to the file-systems. The server configuration also supports access blocks to selective locations.

2) mod security

mod security is an open-source Apache module and helps protect websites from attacks like SQL injections, path traversal attacks, cross site scripting, and various other brute force attacks. This module blocks commonly known exploits by using rule sets and regular expressions. mod security can also work as a firewall for the web applications hosted on any Apache server and hence forms a barrier between a trusted and an untrusted network. If a request is blocked by the module mod security, the website sends a 406 response to the requesting client machine.

3) mod evasive

The mod evasive Apache module helps protect against Denial of Service (DoS), Distributed Denial of Service (DDoS) and brute force attacks on the Apache web server. It typically creates a table of IP addresses and blacklisting clients with behaviors, some of which are listed as

- a. Requesting the same page more than a few times per second.
 - b. Making any requests while temporarily blacklisted
- If any of the above conditions are met, a 403 response is sent and the IP address is logged as blacklisted.

B. Using Direct Authentication with Apache Server

As discussed previously, as soon as the querying user's fingerprint is matched with one of the stored fingerprints in the database, the associated username and password is forwarded to the ODL controller for access request. Note that this claim also contains a domain name and a role with which the user is trying to access the controller. In our work a shell script was written to forward this claim to the user. This command runs as shown below [6],

#create a token

```
$curl -ik -d 'grant_type=password&username=<username>&password=<password>&scope=<scope>'
http://localhost:8181/oauth2/token
```

#Use this token for the purpose of accessing the controller,

```
$curl -ik -H 'Authorization:Bearer<token value obtained in the previous
step>'http://localhost:8181/restconf/config/toaster:toaster
```

This command opens starts the ODL Command Line Interface (CLI) on the same machine where the ODL controller is up and running. The ODL CLI is a management interface to the ODL SDN controller platform. This CLI can be used for various management related tasks such as creating a node, configuring the controller, pushing policies into the controller etc. within the role and domain under which the claim was started [8]. This CLI needs to be echoed back to the user machine, so that the commands the user types on his remote machine, is executed on the server running the ODL. This functionality involves a prerequisite at the user end. The user's local machine from where he is trying to access the controller is assumed to be running a terminal emulator software. As soon as the user's CLI starts at the server machine the user is presented with a URL which he should use in the terminal emulator to connect to its CLI running at the ODL.

In our work, we used PuTTY with the IP address of the ODL controller which the user used to gain secured access to the controller. It is worth noting that the token based direct authentication access has a default timeout of 3600 seconds in ODL. Hence as soon as timeout occurs the user needs to authenticate itself again by presenting the credentials again via the starting web application as mentioned. This timeout was kept fixed at this default 3600 seconds in our work.

IV. Conclusion

This paper explained how to use fingerprints as credential instead of the conventional username and password based method for logging into ODL controller. It is worth noting that this method is applicable to any kind of credential system without loss of generality. The only thing the developer needs to change is the section mentioned in handling fingerprints in order to incorporate a different biometric credential like iris recognition, face recognition etc. This model presents a lucid way of handling different authentication methods, without changing anything in the existing architecture of either the OpenDayLight controller or SDN.

References

- [1] SriniSeetharaman, *OpenFlow/SDN Beginner's Tutorial*, Deutsche Telekom Innovation Center, June, 2013
- [2] Sean Convery, Network Authentication, Authorization, and Accounting, *The Internet Protocol Journal*, 10.1 (2007). Web
- [3] John Dennis, *Federated Authentication Utilizing Apache and SSSD*, Retrieved from <https://jdennis.fedorapeople.org/doc/sssd/configuration.pdf>
- [4] Zain S. Barham, *Fingerprint Recognition using MATLAB*, Faculty of Engineering and Information Technology, An-Najah National University, (2011).
- [5] RaymondThai, *FingerprintImageEnhancementandMinutiaeExtraction*, School of Computer Science and Software Engineering, The University of Western Australia, 2003.
- [6] OpenDayLight, "*OpenDayLight User Guide*", Oct. 2014.
- [7] OpenDayLight, "*OpenDayLight AAA Test Plan*"
- [8] OpenDayLight, "*OpenDaylight Developer Guide*", Oct. 2014.

Acknowledgments

This work is a part of sponsored research program by Cisco Systems (India) Pvt. Ltd. and R. V. College of Engineering, Bangalore. Authors would like to acknowledge the support of Cisco team consisting Mr. Ram Ramkumar, Mr. V C Gopalratnam and Mr. Anil Nileshwar. Any inferences, recommendations, opinions in this manuscript are those made by the author(s) and do not reflect the views of Cisco.