A Simplified Network Access Control Design and Implementation for M2M Communication Using SDN

Almulla Hesham* Fragkiskos Sardis*, Stan Wong*, Toktam Mahmoodi*, Mallikarjun Tatipamula†
* Centre for Telecommunications Research, Dep. of Informatics, King's College London, UK
† F5 Networks, San Jose, CA, USA
{almulla.hesham, fragkiskos.sardis, stan.wong, toktam.mahmoodi}@kcl.ac.uk; m.tatipamula@f5.com

***Abstract*—Network access control is an established security method that only grants access to authorised devices. The 802.1X standard defines the mechanisms for implementing network access control in legacy networks. However, its implementation requires hardware and software components that are not commonly found in small deployments and also adds to the complexity of the network and the deployment time. The Software-defined Networking (SDN) offers an opportunity to develop new network access control solutions, among other things, that can be used to simplify and speed up the implementation and deployment of small, localised sensor networks. In this paper we demonstrate how SDN can be used to develop a basic network access control service without using 802.1X software and hardware while also offering the ability to adjust the available bandwidth and network access policy per device. The proposed solution provides quick and flexible network deployments for IoT sensor networks and other types of M2M communication. We implement our proposed architecture using OpenDaylight and OpenFlow switches and evaluate its performance by running tests with multiple client devices and various policies. Our results show that the studied approach can be a valid approach for small to medium sized networks that requires quick and low-cost deployment and high flexibility in terms of adding new devices.**

***Keywords*—Software Defined Networking, SDN, Quality of Service, QoS, Network Access Control, security, 802.1X, Machine to Machine, M2M, Internet of Things, IoT.**

I. Introduction

Network Access Control (NAC) is considered as one of the most significant aspects in network security since it provides a mechanism for ensuring the authenticity of devices before they are allowed to connect to the network. NAC also provides a means for controlling network access levels for each device or groups of devices so that different users and devices may access different subnets of the network while being prevented from accessing resources outside their access rights. NAC solutions are typically encountered in industrial networks where security and reliability are important factors, alongside more traditional network security solutions such as firewalls and user authentication.

The Software-Defined Networking (SDN) paradigm separates the control plane and data plane (forwarding plane) enabling the abstraction of the underlying infrastructure [1]. The separated control plane resides on a centralized controller (SDN controller) that connects to the forwarding devices (e.g. switch) via a common set of protocols. The controller provides a programmatic interface for business applications through northbound APIs while the communication with network devices occurs via southbound APIs (e.g. OpenFlow protocol). This makes the SDN controller a candidate for consolidating NAC functionality along with other functions such as traffic prioritisation for Quality of Service Management [2]-[5].

OpenFlow (OF) [6], is a southbound protocol for SDN that facilitates the configuration of forwarding rules on switches. An OpenFlow-compliant switch maintains flow tables where the controller can add, update and delete flow entries in each table via the OpenFlow protocol. Each flow table contains flow entries that describe how packets should be processed by the switch according to values in their protocol headers. Meter tables, also stored on the switch, contain bandwidth meter entries that define a bandwidth threshold and an action. When a flow is assigned to a meter, the meter checks whether the throughput of the flow (packet/sec or bits/s) exceeds its threshold. If exceeded, the meter will act on the flow packets either by dropping them in order to enforce the threshold or remark their DSCP value so that other operations can take place such as reducing or increasing the flow priority on the network. These functions present the core functionality of SDN and along with queuing, form the basic mechanisms that enable fine-grained Quality of Service (QoS) control in SDN infrastructures. This flexibility offered by SDN allows us to build NAC systems that combine device and user authentication with a set of network access levels and a level of QoS. Such a solution could potentially consolidate NAC with QoS management in industrial networks, where Machine-to-Machine (M2M) communication plays a critical role in production[7]. The envisioned solution would enable QoS and access control management from a single entity inside the network and would also enable a more flexible approach at changing the applied NAC and QoS policies per device.

This paper presents a novel SDN solution aimed at providing different access level and bandwidth rate for users and clients based on predetermined policies through a simplified and easy to implement Authentication and Authorisation entity that communicates with the SDN controller via its northbound interface. We experimentally evaluate a proof-of-concept prototype and measure the performance of the system in terms of user authentication delay and flow instantiation delay. The rest of the paper is structure as follows: Section II covers the related work, and highlight the gap in the literature where our work is placed. The system architecture is elaborated in Section III, and the details of the implementation are included in Section IV. Section V presents the evaluation and results' analysis. Finally, Section VI concludes this paper and discusses the potential for future development.

II. Background

Network security implemented over SDN is fairly new, but several efforts are taking place mainly focusing on firewall implementation of the SDN controller. In this section, we look at some of the more prominent work done in this field. The SDN Routing Firewall Application [8] performs both routing and firewall function on layer 3 traffic. This is achieved by creating a network application using Python programming language to interact with the controller to route and filter incoming packets. The application was tested on three hosts in three different subnets using a white list where the default permission is to block all traffic except traffic from authenticated IP addresses. The drawback of this application is that it does not authenticate endpoints. Therefore, IP spoofing is possible in this case to reach trusted subnets.

Suh et. al. [9] propose another approach where a firewall application is developed at the switch level thus each packet's headers are checked against the firewall rule from highest to lowest priority, and performs a specified action once matching fields are found before it is delivered to the controller for further processing. This approach is more secure than the above since the firewall rule matching fields are more restricted. On the other hand, its still not secure enough to prevent malicious attack (e.g. IP/MAC spoofing) since no user authentication is enforced. FlowIdentity [10] is a virtualized network access control function using OpenFlow protocol. It implements 802.1X framework in SDN, combined with an authorization method through a role-based firewall. This solution is considered as integration between the traditional port-base access control using an external authenticator (i.e. RADIUS) and SDN architecture with policy enforcement through a role-based firewall. Another similar solution is FlowNAC [11] where a modified version of 802.1X is used to authenticate the users and service level access control based on proactive deployment of flows for user authorization. The difference between them is the method followed for user authorization. Although they provide a great level of security in terms of authentication and authorization, many resources with specific requirements must be available in order to implement such a solution.

The open source SDN controller ONOS consortium [1] is also making an effort at implementing standard 802.1x on the controller in order to embed AAA/RADIUS functionality for wired and wireless networks. The application is being developed as part of the Central Office Re-Architected as a Datacenter (CORD) initiative which is aimed at integrating advanced functionality commonly found in the datacenter to smaller networks such as small and medium businesses and homes. Jung Wan Shin et al. [13] have implemented NAC for Wireless LAN in ONOS using traffic prioritisation policies based on the MAC addresses of devices. In their paper they demonstrate a working prototype based on open source applications and target their solution towards software defined wireless networks. To a similar end, where wireless and wired access technologies are used in tandem, the HP VAN controller also implements similar functionality with the Campus Security module which is aimed at campus networks and provides NAC through the controller and allows for a more flexible management of university networks which largely operate on BYOD principle due to the number of students connecting to them [2]. SDN-Driven Authentication and Access Control System [11] is an AAA (authentication, authorization and accounting) design based on Software Defined Networks (SDN) structure. It uses SDN features in order to protect the traditional network infrastructure. In other words, SDN is considered as an additional access layer where user traffic is forwarded to the controller that registers and authenticates the switches, authenticates the hosts and bind them to the switches (and ports), provides the authentication of users and also manages data flows and users/hosts mobility. However, one downside of this technique is that too many packets can be delivered to the controller and take up a large portion of its resources, therefore it is more efficient to block unnecessary packets at the switch level.

Efforts are also being made by the SDN community to address M2M communication requirements. The OpenDaylight IoT Data Management (IoTDM) project provides an open source implementation of the OneM2M IoT Data Broker which acts as a data collector for sensors. Embedding this functionality on the SDN controller enables tight integration in sensor networks and reduces the number of devices required to implement an IoT platform. Furthermore, this integration effectively turns the SDN controller into a single entity that can manage the network of sensors and collect data from IoT devices. This functionality could be extended in the future in such way that the network management aspects will take into consideration the events registered by the sensors so that communication can be prioritised or compromised devices can be isolated automatically.

III. System Design

The aim of our proposal is to enforce access control in SDN connected clients. Our work is similar to FlowNAC and FlowIdentity, however our intention is to show how this can easily be implemented using simpler methods that don't rely on 802.1X and are more suitable for smaller networks such as in small and medium businesses or at home. Furthermore, we leverage SDN's capability in controlling the QoS per client and use it along the defined access policies. Due to complexities in dealing with QoS over wireless access networks, our proposed architecture addresses only the wired part of the network. In the cases of wireless sensor networks, we assume all of these functions to take place behind the IoT gateways. In order to achieve this, our solution must be able to:

- Authenticate clients based on their set of credentials (username, password and host MAC-address) when they request to log in.
- Authorize access to other network hosts and allocate a bandwidth rate for the authenticated clients.
- Install the appropriate flows on the controller.
- Delete the flows on the controller when the client logs out.

Fig. 1 illustrates the high-level system design. End-users such as IoT devices communicate with the authentication and authorization service through an application that sends the users credentials. The authentication and authorisation service first checks the user credentials and upon successful authentication, retrieves the assigned policy for the user. It then installs the corresponding access rules and bandwidth capacity on the SDN controller for that user's policy. Packets belonging to the authenticated devices are identified through a series of fields in their headers such as source and destination MAC/IP/Ports as well as transport protocol ID or even physical switch ports if desired.

The User and Policy database contains the information required for authentication and authorization. The User database stores user and host credentials (i.e. binding user name, password and MAC address) used by the service to authenticate the users and pair them to a host. The Policy database stores the policies that are enforced on authenticated users. These policies define the access level (access to subnets and hosts) and bandwidth rate allocated to the user. Each architecture block can reside on a

[1]Open Network Operating System. Available at: http://onosproject.org/

[2]https://www.hpe.com/asia_pac/en/product-catalog/networking/networking-software/pip.hp-network-protector-sdn-application-series.6903004.html

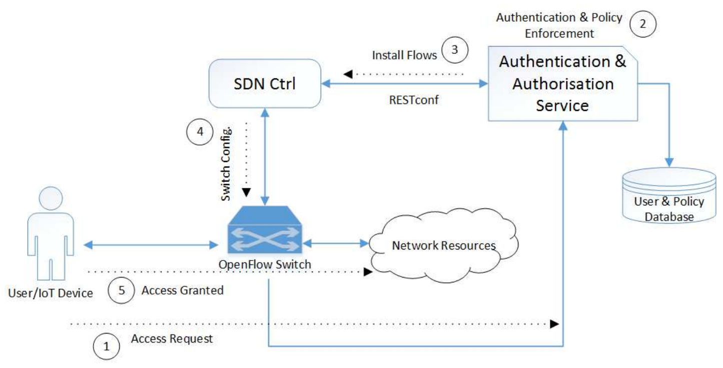

Fig. 1. Block diagram of the investigated system architecture. The architecture implies in-band control and data planes in the SDN architecture as the clients require access to the Authentication and Authorisation Server which resides on the same plane as the Controller.

separate physical host and it can exchange information with other blocks using a secure connection over the network. Transport or application layer security can be used to secure the connection between architecture components if they reside on different hosts. For example, the authentication server can communicates with the SDN controller through a secure interface to add or delete configurations. Thereafter, the SDN controller connects to the switch via OpenFlow protocol to add or delete the flows.

Our solution uses OpenFlow meters in order to control the available bandwidth for each user and also measures the network utilization reported by each meter for monitoring purposes. Those flow entries and meters comprise certain match fields that are required to authorize incoming packets. The match fields used are:

– **Ethernet Source and Destination Address:** it defines the user host's MAC-address as a source on Outbound flows and as a destination on Inbound flows.

– **IPv4 Source and Destination Address:** it defines the subnet allowed for a user to access as a source on Inbound flows and as a destination on Outbound flows.

– **Ethernet In-port:** it defines the switchs Ethernet-port that is allowed for the users host to connect to.

– **Band Rate:** it defines the bandwidth rate allocated for a user.

IV. Implementation

We developed a prototype of our architecture using Python in order to demonstrate how the system function and to perform some basic performance measurements in terms of how quickly users can be authenticated and granted access to the network. For the SDN controller, we use OpenDaylight (ODL) [10] because it is a controller that is widely supported by a big community and is considered stable for production. We use Pica8 P3290 OpenFlow switches in our testbed with an in-band topology for the control and data network.

The P3290 management interface connects to a legacy GbE switch. The managed interfaces also bridge to the same legacy switch which also connects directly to the university's network. As illustrated in Fig 2, the SDN controller and the host running the authentication service also connect on the same switch. The laptop plays the role of the client device and therefore runs the client application and requests access to Kings college internal network and resources. In the initial switch setup, we pre-install flows that will allow the laptop to contact the authentication service but no other hosts on the network.

The implementation setup assumes a manual network bootstrap to enable initial communication between new clients and the authentication server. However, this process can be easily automated via the SDN controller using a script that installs a basic configuration as soon as the controller starts or a new switch is added to the topology. Furthermore, while our experimental setup makes use of a in-band topology (merged control and data planes) to enable communication between clients and authentication service, this is not the mandatory deployment mode in a real setup as the authentication server could use two independent interfaces (one for each plane) in order to maintain the separation between control and data planes if desired.

Two Python scripts have been created in our solution; the first one runs on the client machines and communicates with the authentication service through TCP/IP socket to send the user name and password in addition to the MAC address and the host. The second script runs on the authentication server and connects to the user and policy databases, and SDN controller. The User and policy databases are implemented as plain text files residing locally on the same host machine as the authentication service and contacts the SDN controller over the network. Upon receiving input, the authentication service executes the following functions:

– ***Login***: the user will be asked to provide a name and password that is sent along with the host's MAC-address to the server. The server checks the user database and responded back to the client with its logging statues (authenticated or unauthenticated). If the authentication is successful, the servers application will instruct the SDN controller to add the flows and allocate a bandwidth rate to enforce the assigned policy.

– ***Logout***: the servers application will instruct the SDN controller to delete the assigned flows and bandwidth.

V. Experimental Results

The prototype is implemented in the infrastructure demonstrated in 2, where the Pica8 Switch is set up in OpenFlow mode and managed by OpenDaylight controller running on a separate server. We tested the prototype by firstly running the server application to create the communication session, then we ran the client application on the laptop where username, password and MAC-address was sent to the server. After the user was successfully authenticated, the policy had been successfully enforced by adding new flows and allocating bandwidth on the OpenDayLight controller and pushed to the switch as shown in Fig. 3.

Authentication and authorization times are important metrics to evaluate the performance and the access delay of this prototype. The performance has been evaluated against 50, 100 and 1000 number of clients and policies. To obtain precise results, the performance test was repeated 5 times for each number.

The authentication time, in micro seconds (μsec) is the time between a request received by the server and a response sent back to the client. Upon successful authentication, the authorization time, in μsec, is the time the system takes to find the corresponding policy within the total flow installation time. The flow installation time, in milliseconds (msec), is defined as the time it takes for the controller to receive the flow installation message from the authorisation service and configure the switch. This process is an order of magnitude slower than the authorisation

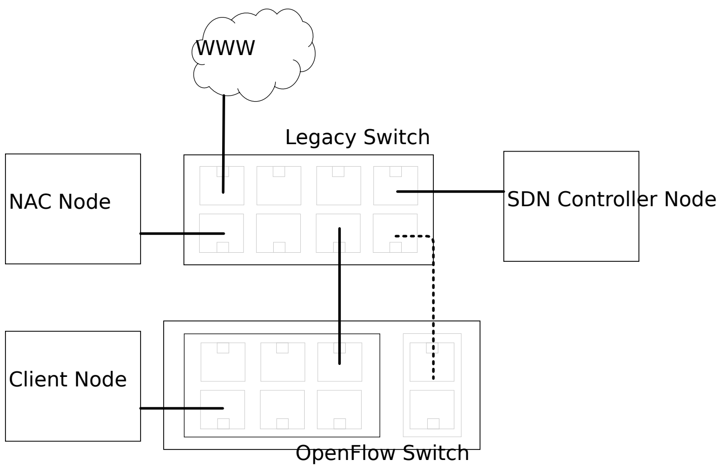

Fig. 2. Prototype trial network topology with in-band control and data planes. The legacy switch provides the control plane and is also bridged to the SDN switch's data plane to allow authentication communication to new clients as well as Internet access to authenticated clients.

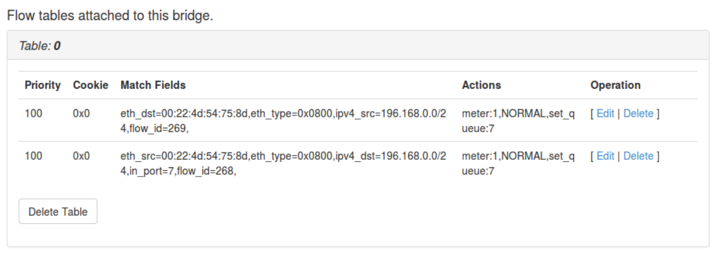
Flow tables attached to this bridge.

Table: ***0***

Priority	Cookie	Match Fields	Actions	Operation
100	0x0	eth_dst=00:22:4d:54:75:8d,eth_type=0x0800,ipv4_src=196.168.0.0/24,flow_id=269,	meter:1,NORMAL,set_queue:7	[Edit | Delete]
100	0x0	eth_src=00:22:4d:54:75:8d,eth_type=0x0800,ipv4_dst=196.168.0.0/24,in_port=7,flow_id=268,	meter:1,NORMAL,set_queue:7	[Edit | Delete]

Fig. 3. Flows installed on the switch as appearing on the switch management UI.

time and therefore poses the main delay in the second step of the experiment.

TABLE I
AUTHENTICATION TIME (IN μSEC) FOR 50M 100 AND 1000 CLIENTS

Iteration	Auth.T 50c	Auth.T 100c	Auth.T 1000c
1	81.06	131.85	816.11
2	102.04	143.05	969.89
3	99.90	139.00	849.01
4	109.91	134.94	872.85
5	108.96	137.09	828.98
Average	**100.37**	**137.19**	**867.37**

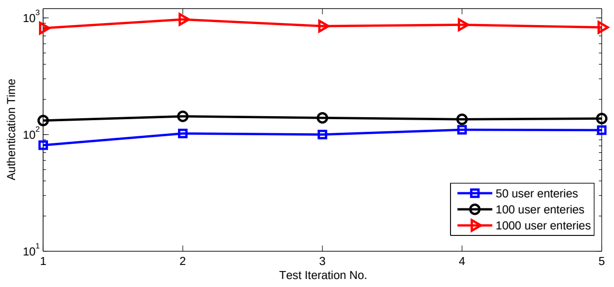

Fig. 4. Authentication time (in msec) for 50, 100 and 1000 user entries in the database over 5 testing iterations from a single client. Logarithmic scale is used for further clarity of presentations.

The results in Table I and Fig. 4 show the authentication delay for 50, 100 and 1000 client entries in the database. As expected, the time increases as the number of clients increases. Although, there is a large gap between the numbers of client IDs tested in the database, the average times are quite small (less than 1 msec). We expect these results to vary depending on the type of hardware, database and encryption used in a production-ready implementation, however, we need to highlight that the performance achieved does not present a severe bottleneck on the network and this type of service can be also used in M2M communication where quick response times are required.

TABLE II
AUTHORIZATION AND FLOW INSTALLATION TIME (50 POLICIES)

Iteration	Authorization Time(μsec)	Installation Time(msec)
1	104.10	104.02
2	117.59	117.49
3	66.62	66.49
4	88.86	86.22
5	180.15	180.05
Average	**111.46**	**110.85**

TABLE III
AUTHORIZATION AND FLOW INSTALLATION TIME (100 POLICIES)

Iteration	Authorization Time(μsec)	Installation Time(msec)
1	94.96	94.83
2	77.79	77.68
3	85.74	85.64
4	61.93	61.80
5	84.20	84.09
Average	**80.93**	**80.81**

TABLE IV
AUTHORIZATION AND FLOW INSTALLATION TIME (1000 POLICIES)

Iteration	Authorization Time(μsec)	Installation Time(msec)
1	105.96	105.51
2	93.32	92.82
3	95.71	95.21
4	77.64	77.14
5	121.12	120.64
Average	**98.75**	**98.26**

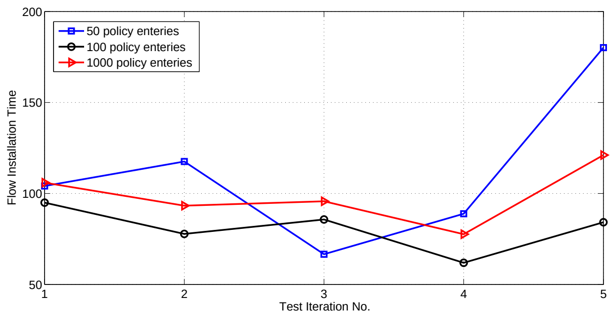

Fig. 5. Flow installation time (in msec) for 50, 100 and 1000 policy entries in the database over 5 testing iterations from a single client.

The results for authorization and flow installation in Tables II, III, IV and Fig. 5 show the authorization and network configuration delay for a different number of policies. Unlike authentication, the number of policies doesnt impact the overall flow installation time severely due to the fact that most of the authorization time is spent on installing the flows which means that network configuration is the main cause of authorization delay. Therefore, the overall flow installation time is heavily impacted by the SDN controller itself and the OpenFlow switches used on the network, as they are the two main components involved in the installation of new flows on the network. It is also worth noting that the autorization time does not scale similarly with the authentication time as the number of entries in the database increases. This is due to the fact that for the authorisation, we check user names and passwords as well as MAC addresses, whereas in the authorization, we only check for the appropriate policy ID that belongs to the user. Therefore, the check for the policy is quicker to perform and is a single check as opposed to the authentication check. This is purely an implementation-specific quality of our setup and we expect that real world deployments could behave differently depending on how the database is designed and implemented.

From our experimental results, we can conclude that such a solution can provide adequate performance for small and medium networks and given more powerful equipment it could scale well to large networks also. Considering M2M communication in industrial networks, the proposed architecture and authentication scheme can enable dynamic re-allocation or network resources and subnet access for devices that feature multiple sensors and send their data at different intervals to collectors. An example of such devices is LoRA [3], where narrowband sensors transmit their data to the LoRA base station using different codes in order to identify the application. These codes can be used to extend the network partitioning to the wired network via the gateway so that each sensor can send and receive data in an isolated end-to-end connection. The presented results can be further improved by implementing the policy and authentication database in Apache Cassandra which is optimised for thousands of transaction per second and could provide better scalability when introducing multiple new devices simultaneously on the network (i.e. bootstrapping a new subnet). From our results, we also conclude that flow installation takes the most time and for a system that requires fast reconfiguration, this needs to be improved by optimising the SDN controller and switches. This is something that largely depends on the OpenFlow implementation of the SDN controller and the software architecture and implementation of the controller and switches as well as the load on the controller and the control network, therefore, it is difficult to propose ways that this could be achieved within the scope of this paper. However, one solution that could improve the flow installation time would be to build the authentication and authorisation services within the controller as controller modules which would interact directly with the flow database of the controller to install client flows. This would bypass the northbound interface of the controller which could result in reduction of the total flow installation time. This solution would sacrifice some of the flexibility and ease of development since these modules would be specific to the SDN controller used in the deployment and would have to use the controller's internal APIs which would require a developer to know the inner workings of the controller and also restrict the developer to the programming language used by the controller's developers.

VI. Conclusion

The paper presents a simple and easy to implement NAC solution that makes use of SDN technology for controlling network traffic based on predetermined security rules that provides different access levels and bandwidth rates for users. We showed how NAC can be implemented using SDN without deploying 802.11X and we envision an integrated solution for small and medium sized networks where more security functionality can be implemented via the controller. Our system is able to authenticate users successfully, register their connected device and bind a particular IP and MAC combination to a predefined level of QoS and a subnet or host that the client may access. The preliminary performance results show that the performance bottleneck lies in the flow installation time, however, the authentication process can be further optimised using alternative methods for storing and accessing the policies. We believe that the proposed solution is a step in the right direction and in line with the industry's steps towards an integrated security solution on the controller.

[3]A Technical Overview of LoRAWAN. Available at: https://www.lora-alliance.org/portals/0/documents/whitepapers/LoRaWAN101.pdf

VII. Acknowledgement

This work has been supported by the 5G-PPP VirtuWind [14] (www.virtuwind.eu) and 5G-NORMA [15] (5gnorma.5g-ppp.eu) projects under the European Union's H2020 research and innovation programme.

References

[1] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks", Open Networking Foundation White Paper, April 2012.

[2] Amani, M., Mahmoodi, T., Tatipamula, M., Aghvami, H., "SDN-based Data Offloading for 5G Mobile Networks", ZTE Communications, no. 2, July 2014, pp. 34-40.

[3] Amani, M., Mahmoodi, T., Tatipamula, M., Aghvami, H., "Programmable policies for Data Offloading in LTE Network", IEEE International Conference on Communications, June 2014, pp. 3154-3159.

[4] Mahmoodi, T., Seetharaman, S., "Traffic Jam: Handling the Increasing Volume of Mobile Data Traffic", IEEE Vehicular Technology Magazine, vol., no. 3, September 2014, pp. 56-62.

[5] Mahmoodi, T., Seetharaman, S., "On Using a SDN-based Control Plane in 5G Mobile Networks", Wireless World Research Forum (WWRF), 32nd Meeting, May 2014.

[6] Open Networking Foundation, "OpenFlow Switch Specification-v-1.3.0", June 2012.

[7] Condoluci, M., Araniti, G., Mahmoodi, T., Dohler, M., "Enabling the IoT Machine Age with 5G: Machine-Type Multicast Services for Innovative Real-Time Applications", IEEE Access, vol. 4, May 2016, pp. 5555-5569.

[8] Kaur, K., Kaur, S., Gupta, V., "Software-Defined Networking based Routing Firewall", International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT), March 2016, pp. 267-269.

[9] Suh, M., Park S. H., Lee, B., Yang, S., "Building Firewall over the Software-Defined Network Controller", International Conference on Advanced Communication Technology, February 2014, pp. 744-748.

[10] Sadiq T. Yakasai, Chris G. Guy, "FlowIdentity: Software-Defined Network Access Control", IEEE Network Function Virtualization and Software Defined Network (NFV-SDN), November 2015, pp. 115-120.

[11] Matias, J., Jacob, E., Toledo, N., Mendiola, A. and Garay, J., "FlowNAC: Flow-based Network Access Control", European Workshop on Software-Defined Networks, December 2014, pp. 7984.

[12] OpenDaylight, Available at: http://opendaylight.org/ Last access: August 2016.

[13] Shin, J. W., Lee, H. Y., Lee W. J., Chung, M. Y., "Access control with ONOS controller in the SDN based WLAN testbed", Int'l Conference on Ubiquitous and Future Networks (ICUFN), July 2014, pp. 656-660.

[14] Mahmoodi, T., et. al., "VirtuWind: Virtual and Programmable Industrial Network Prototype Deployed in Operational Wind Park", Transactions on Emerging Telecommunications Technologies, vol. 27, no. 9, September 2016, pp. 1281-1288.

[15] Droste, H., Rost, P., et. al., "An adaptive 5G multiservice and multitenant radio access network architecture", Transactions on Emerging Telecommunications Technologies, vol. 27, no. 9, September 2016, pp. 1262-1270.