Scalable controller implementation for heterogenous wireless network management

1st Keerthana SanalaPrakash
Department of Mathematics and Computer Science
University of Antwerp, Belgium
keerthana.sanalaprakash@student.uantwerpen.be

***Abstract*—There is an increase in the number of consumer devices using LAN, which can connect to the Internet using divergent wireless network technologies. After a thorough study, we discover that the existing solutions are unfit to subsist with this divergence. For instance, dynamic inter-technology switching is user or application-based. To manage this problem, we study and utilize the ORCHESTRA framework as proposed in the Paper 'ORCHESTRA'. The proposed framework manages the different devices in heterogeneous wireless local area networks (WLANs) and introduces capabilities such as packet level dynamic and intelligent handovers (both inter- and intra-technology), load balancing, replication, and scheduling. The framework consists of a controller that is capable of communicating with both existing Software-Defined Networking (SDN) and also newly proposed devices like virtual Medium Access Control (MAC) layer. This paper mainly focuses on the implementation of this controller with the proposal of more features like efficient storage for gathered information, communication with other controllers, communication with wired Software-Defined Networking (SDN) controller, Switch and Access Point (AP). The experimental evaluations carried out on different machines demonstrate the benefits of our approach to manage the network traffic across VMACs and handle communication between devices asynchronously across the network.**

I. Introduction

The diversity of devices is becoming more provident in using LANs for connecting to the internet using different network technologies such as Bluetooth. In the future, the probability of an increase in the variety of devices and technology which leads to all kind of Internet of Things (IoT) devices and new technologies like Bluetooth 5.0, WiFi 6 and IEEE 802.11ax. These devices lead to a strong need for diverse quality requirements like high throughput and very sensitive to network disruptions. Managing a wide range of diverse devices is critical to handle. While serving the right quality is still needed.

At present, the lower layers of OSI stack are unfit to handle the heterogeneity. Since, there is a wide range of wireless networks such as WLANs, WPANs, WMANs, and WWANs. Switching between these technologies or load balancing is consigned to the application layer. This follows a very standard way of management of wireless networks, making it hard to respond in a timely fashion to the dynamic network changes. The Existing solution such as SDN, having scalability performance issues like controllers has to be synchronized which is hard to set up. Technology change, which needs a particular software to run. Also, it is having more problems handling network data flow. Now, We understand the need for inter-technology and access point (AP) routing and dynamic inter-technology gives support to load balancing and multi-path routing.

All these problems discussed are overcome using this proposed framework named ORCHESTRA [8]. Where it introduces the software-defined framework that relies on network virtualization to cope with the heterogeneous challenges and can support inter-technology management. Although there is still a controller lack. This will be the main focus of this paper which will be discussed below.

The contribution of this paper is two-fold: First, we utilize the introduced ORCHESTRA framework [8]. Second, the proposal for improving the central controller to distributed and improving upon single point failure. Also, it includes the distributed database for storing topological information.

The rest of the paper is organized as follows. We start by giving an introduction in Section I. Next, the implementation of the controller in the developer perspective as a flow diagram, Existing database, different types of database and suitable database supporting this implementation and some related works in Section II. Next, giving an overview of the existing ORCHESTRA framework, architecture in Section III. Next, we start by illustrating how the proposed controller can be scaled with other devices and databases in the architecture Section IV. Lastly, We provide the results supporting the controller activities and heterogeneity in Section V. Finally, we conclude in Section VI.

II. RELATED WORK

In this section, we will discuss existing work on various controllers. 5g-Empower is one the controller to take into consideration [16,22,23]. The EmPOWER is responsible for the management of the heterogeneous RAN. The EmPOWER Runtime supports multiple virtual networks, or Tenants, on top of the same physical infrastructure. A Tenant is a virtual network with its own set of WTPs, VBSes, and CPPs. Network Apps run on top of the run-time in their slice of resources and

exploit their programming primitives through either a REST API or a native Python API.

The only persistent information stored at the run-time is the client's authentication method (currently only ACLs are supported) and the list of network slices currently defined. All the state is kept within the network in a distributed fashion and is synchronized when the devices connect to the run-time. As a result, the run-time can be hot-swapped with another instance without affecting the active clients. Moreover, the network itself can still function at its last known state even if the run-time becomes unavailable. Except for the logging subsystem, every other task supported by the run-time is implemented as a plug-in (i.e., a Python module) that can be loaded at run-time. Examples of such plugin are the module implementing the data-path run-time protocol, the RESTful web interface and the mobility/load-balancing applications used for the demos[12,24].

Another Controller to discuss would be SDN, As our network scales, it becomes more and more inflexible as more and more devices have to be configured for every change. Eventually, it can stagnate and we will need a new solution, or our administrators would not want to change anything because it would simply be such a huge undertaking. Another aspect of this model is that the control and forwarding planes of devices are usually in the same box [1,2]. The control plane is the part of the device that makes decisions about how it will handle traffic, the forwarding plane is the part that processes and forwards traffic based on how the control plane is configured.

An administrator has to access the device, using CLI or SNMP or other access methods, and configure the control plane to influence new behavior in the forwarding plane. Whether that means modifying an ACL, turning on a routing protocol, adding new HSRP neighbors, really anything, it must be configured on the device. So once network scales as our business grows, every change and addition will require more and more configuration tasks on existing network devices, to guarantee uniform performance and with more requirements for flexible networking for server provisioning, as well as cloud usage, this can quickly get out of hand. But there is a better solution. Enter SDN, The big strategy behind SDN is that it decouples the control plane from the forwarding plane. No longer will the decisions for how to handle traffic be made on the same device that is forwarding it. One of the most popular tools for this is the OpenFlow protocol[9]. So how does OpenFlow work? It will periodically gather information from network devices concerning their status as well as issue commands concerning how to handle the traffic. The information it gathers will be passed in an abstract format to a network controller OS, examples of which are NOS or Cisco's XNC [11,14]. On the controller, an administrator can programmatically define how programs and application traffic should be handled.

Odin [25] is one such controller to discuss it further, this is implemented as an application on top of the Floodlight OpenFlow controller. It uses an in-band control channel to invoke commands on the agents. In its current form, Odin commands can add and remove LVAPs [6,7,10] and query for statistics. The master, through Floodlight, uses the OpenFlow protocol to update forwarding tables on the AP and switches. Odin applications (i.e. Mobility Manager and Load Balancer) execute as a thread on top of the Odin Controller. Applications can view the statistics exposed by the Controller in a key-value format.

To summarize, The discussed controllers lag in distribution between controllers and there is a possibility for single point failure. It lags in load balancing. Our solution would manage and provide a significant approach.

III. ARCHITECTURE

In this section, we define the functionality of the proposed ORCHESTRA framework [8] and describe the controller's functionality and the proposal of added contribution to it.

A. Functionality of the ORCHESTRA

The proposed solution offers two operations for the respected clients on the network. The first operation corresponds to devices that have the proposed virtual layer (OVL) implemented. This provides flexibility and high control.

1) Load balancing: Packets are balanced over multiple modes of production. The goal is to increase throughput and can be differentiated by applications to support different requirements.

2) Replication: Packets are replicated over multiple technologies to increase reliability. This increases throughput or reduces delays in hard environments.

3) Scheduling: The time-division multiple access (TDMA) based scheduling is applied on top of access control of the technology. Which allows for good transmission and cope up with interference from surrounding stations.

B. ORCHESTRA virtual layer

Recently connectivity is handled on an interface basis. Changing the interface leads to connection loss. This issue is been solved by introducing a virtual layer and abstracting connectivity from users and applications. This is been proposed in this article [8].

C. Controller

The controller enables management and combines all network logic. This is a distributed controller to avoid a single point of failure. This includes the assignment of clients to endpoints.

We can see the architecture of the controller in Figure 1, The Controller provides the interfaces to the SDN controllers for both wired and wireless networks. It is higher in the hierarchy than SDN controllers. It also provides communication with switches, APs and client devices. It also utilizes the distributed storage system for storing the collected monitoring information.

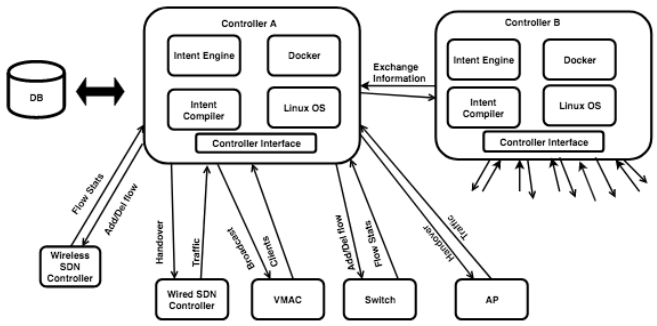

Fig. 1. Controller Architecture with Controller A and B if the center of network management and coordinates existing SDN-based controllers, Switch, AP and clients through different protocols

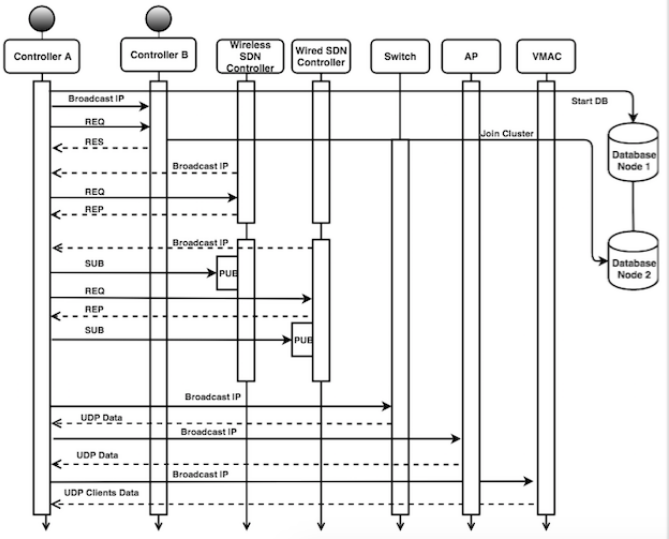

Fig. 2. Control flow via sequence diagram.

1) VMACs and controller: For the communication between VMACs and Controller, we use the UDP (User Datagram Protocol) which is a communication protocol for establishing low-latency and loss-tolerating connections between applications on the internet. As shown in the flowchart in Figure 2, The controller has the capability of letting multiple clients communicate with it. We have a client one on one of the Linux machine and using Multi-threading concept, we are first listening for the broadcast of the controller IP address. And as soon as we receive the controllers broadcast we receive the IP address and connect to it and start establishing the communication.

Now, Controller sends a rule set in a JSON or dictionary format with what to filter on and what to apply. IP addresses, transport protocol or packet type serve as filters. Messages from controller include handovers to a different technology or AP. The minimum message includes the current AP. One thread for listening for broadcast, one thread is for sending the data and one main thread to start up the Client. Whereas the clients send the flow details which include the source IP, Destination IP and throughput for AP. In Figure 3 the sequence diagram Showing the workflow of controller and clients. These systems link clients with controllers via a network to support distributed data management, processing, and presentation. The extent to which servers share these functions with clients determines the range of potential applications to distributed databases. We test the network traffic and the consistency of the controllers through this.

2) Controller A to Controller B: We use two controllers for this framework. The idea behind choosing two controllers is To tackle the challenge of scalability, our controller is distributable as well. State information about common devices is shared between controllers. This includes the traffic requirements and more importantly, the RSSI values to estimate the distance. Only information of devices that both controllers have information on will be shared among the controllers to reduce overhead. For example, a device sees two APs and is connected to one of them. One of the APs is in the region of the first controller and the other AP is in the region of the second controller. As both controllers have information on a device, it is shared among them to consider them when a handover is needed. If a handover is needed because a newly computed assignment would place it in the region of another controller, the controller currently responsible for the device informs the other flow stats add/del flow, RSSI, traffic, handover, flow stats add/del flow controller to take over the device. The new controller then updates its flow rules and AP configuration and acknowledges the handover. After that, the old controller deletes the flow rules and the AP configuration and only monitors the device. It is also useful when starting up the database and sends commands to join its cluster of nodes. The functionality of both the controller is pretty much the same.

The first step is to start with broadcasting to discover the listening controllers and other devices. Broadcasting is nothing but to transmit a packet that will be received by every device on the network. In our case, we broadcast to the subnet mask so it's more secure, which will only discover the devices in the subnet. For this, we use the socket broadcasting technique and for the communication between the controller, we follow the REQ-RES messaging pattern. Besides, we are not storing any controller communication into the database.

3) Controller and Wireless SDN Controller: This section describes the communication between the controller and the wireless SDN controller. The wireless controller needs some implementation such as to be able to listen to the controller broadcast and able to send the broadcast message. Then be able to send flow status and handle the add/delete flow. we picked up the messaging pattern of REQ-REP. The requestor sends a request message to a replier system that receives and processes the request, ultimately returning a message in response. This is a simple, but powerful messaging pattern that allows two applications to have a two-way conversation with one another over a channel.

The information transferred during this communication was modeled using Yang data modeling language. At the wireless

SDN controller, we acknowledge this by sending a response. We then store this packet in a file system.

For the communication from wireless SDN controller to Controller, we chose to use PUB-SUB messaging pattern where we just send data packets and do not care about the acknowledgment. The publisher is a Wireless SDN controller here and Controller being Subscriber. We store all packets from the SDN controller into the database.

4) Controller to Wired SDN Controller: This section describes the communication between controller to wired SDN controller. This wired SDN controller component includes the listening sockets and broadcasting phase. It can handle Handovers and traffic. We chose the messaging pattern of REQ-REP. The information transferred during this communication was modeled using Yang data modeling language.

For the communication between wired SDN to the controller, we prefer to use PUB-SUB messaging pattern and data is modeled using Yang. The controller receives all the data from wired SDN and is pushed to the database.

5) AP and Switch to Controller: For AP and Switch to controller communication, We use regular UDP socket communication. We start by listening to broadcasting from the controller and then establish a connection. Further, we start sending the flow status which includes the source and destination IP and throughput. After sending data packets, we wait for an acknowledgment. This can be further improved by using OpenSSL and Netconf. But it is out of scope for this research.

IV. IMPLEMENTATION

In this section, we will discuss how the proposed controller architecture is implemented in terms of a programming perspective. The implementation of the ORCHESTRA controller is quite challenging but easily deployable on any operating machines. The controller was built and tested on the virtual machines. But later, the virtual machines are replaced with Physical servers. Figure 3. gives an overview of communication and how the packets are transferred across the network.

The first implementation starts with choosing the database, Initially SQL was taken into account but later to distribute the data across several machines and avoid single point failure, we chose a distributed database. We defined a function to start up a database where shell scripting is used to write commands to setup database nodes. This implementation was time-consuming by using regular database setup so we used a docker for speeding up the process. Where it provides containers for each node. After this step, the instructions for joining the database cluster is sent to the other controller via Request-Response messaging pattern using ZMQ library. The controllers have access to communicating with different devices. For defining the rule set and modeling the data, we used the YANG modeling language. The YANG modeling can be challenging if we need to dump some information and pack it for usage. The YANG model first needs to be modeled based on the criteria and later needs to be converted to a python class to use in functions for dumping some data. We then either use XML or JSON format for dumping some information on the modeled scripts. For this experiment, we modeling 10 different scripts as follows, from the controller to five different devices and five devices to the controller.

After all these steps, we move on to the communication part. There are three types of broadcasting as follows; The Broadcasting function is initiated by calling sockets where it has broadcasting support. For communicating between different devices, we chose to use messaging patterns like REQ-RES and PUB-SUB and finally for clients we chose to use UDP communication protocol because UDP is faster than TCP, and the simple reason is because its nonexistent acknowledge packet (ACK) that permits a continuous packet stream, instead of TCP that acknowledges a set of packets, calculated by using the TCP window size and round-trip time (RTT).

To test the network traffic and the consistency of the controllers and get the use out of controllers. As seen in the flowchart in Figure 2 the controller has the capability of letting multiple clients communicate with it.

V. DATABASE

In this section, we will discuss the challenges of having scalable architecture and the reason for choosing a certain database. The controller has another part besides communication. Which is a data storage system where all information is aggregated and combined. This model consists of topology information from the SDN controller, APs and throughput of each flow on each switch. The challenge of having scalable architecture is to overcome the database failure and be able to hold all information. But this is not possible when we use a Relational Database. For example, if one controller has one database at the persistence layer and it needs to inform the other controller whenever there is a data. This is entirely invisible for other controller and it needs to wait until unless it is informed. If there was a database failure then it is nowhere. We lose all the information stored. To avoid this, we chose to use a distributed database. Let's have a look at different distributed databases and the reason for choosing a certain type from them.

A. Non-Relational Database

Pros: They scale out horizontally and work with unstructured and semi-structured data. Some support ACID transactional consistency. Schema-free or Schema-on-read options. High availability. While many NoSQL databases are open source and so "free", there are often considerable training, setup, and development costs. There are now also numerous commercial products available. Cons: Weaker or eventual consistency (BASE) instead of ACID. Limited support for joins. Data is denormalized, requiring mass updates (i.e. product name change). It does not have built-in data integrity (must do in code). Limited indexing.

Some of the most known NoSQL or non- relational DBs that are MongoDB, DocumentDB, Cassandra, Coachbase, HBase, Redis, and Neo4j. Hadoop is also a part of this entire

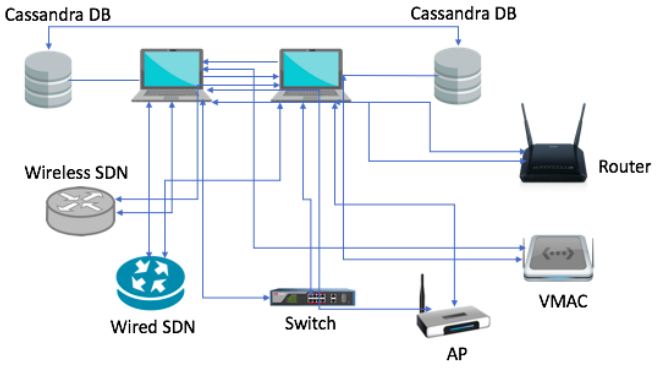

Fig. 3. Implementation setup of the experiment

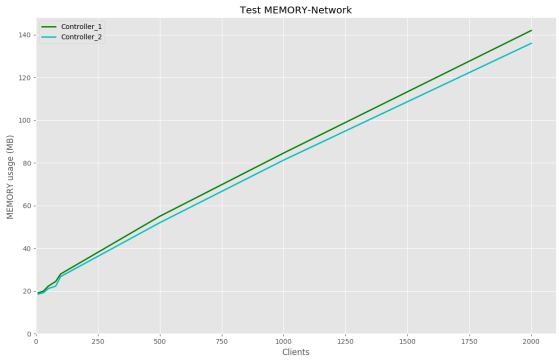

Fig. 4. Memory usage of the network

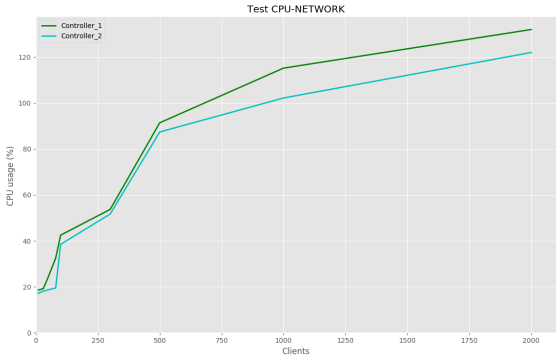

Fig. 5. CPU Usage

discussion. But, we should know that it is a file system with components made up of HDFS, Yarn, and MapReduce.

For this research, we use Cassandra, Its write performance is higher than most other NoSQL DBS. Cassandra follows a peer to peer architecture, as opposed to the master-slave architecture of MongoDB and most RDBMS. That means you can write to any peer and Cassandra will take care of data synchronization. That's why it is faster. Which was easily installed using Docker, which allocates its container to one of the nodes (seed node). With this seed node, we can let any number of nodes to join its cluster. After joining the cluster, the database updates can be seen from any remote system irrespective of its location.

VI. RESULTS AND DISCUSSION

This section explains the experimental setup of the framework in Figure 3 and results in Figures 4, 5, 6, 7, 8. The prototype consists of multiple parts: first, there is a controller A running on Linux machine 16.04 and Other controller B running on Linux device 18.04. Devices such as Wireless, Wired SDN controller, APs and Switch are run on Virtual Machine (Linux 18.04). Which is installed on top of Windows Machine? Furthermore, Docker is used for Database installation and Cassandra 2.0 is installed on top of Docker, which allocates one of the containers for each node. All simulations were conducted on personal laptops. We can discuss the results clearly in sub-sections.

A. Load distribution

To improve the solution in vast networks, with more than thousands of devices, we experiment and investigate the impact of distributing the computational and management load across different controllers. In this experiment, we emulated 1000 to 2000 clients and used two controllers to distribute the load on random. The results can be seen in Figure 5 where one controller reaches up to 125 percent average CPU usage for 1000 clients. This means that there is not much scope for the computation and the computation takes longer or the clients are served with a delay which results in the worst experience. Since we have multiple controllers, we can see that controller two provides 120 percentage of CPU usage less CPU usage than the Controller one, this is because of the distribution of clients among the controller. The CPU usage on both the test is bit high but this is because of the Linux device was maxing out of memory.

Both memory usage is shown in Figure 4. While the memory usage starts with 20 MB for ten clients and there is a spike in memory because of generating more clients and storing more data and we can see that there is less consumption of memory in the graph for controller B, this is because of using multiple controllers. Figure 6. shows the response time of the controller, which is pretty good that it takes more or less 0.2 to 4 seconds for several increases in clients. This could also take less time when tested on a more powerful device.

Finally in figure 7. we can see the database synchronization time. we to test how much time passes between one controller storing information and the other controller having that fresh information available. we synchronize clocks between the controllers and time-stamp the storing and on the other controller we pull and check when the information changes.

VII. CONCLUSION

In this paper, We introduce a scalable controller with implementation for heterogeneous wireless network management.

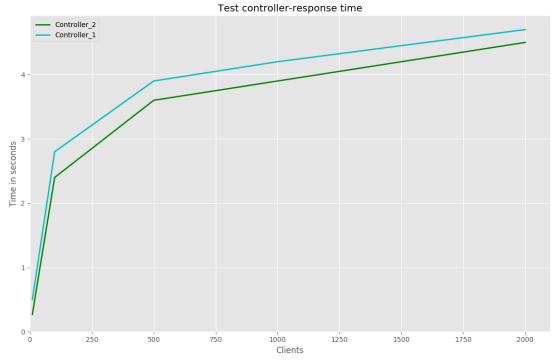

Fig. 6. Response time from the controller to client

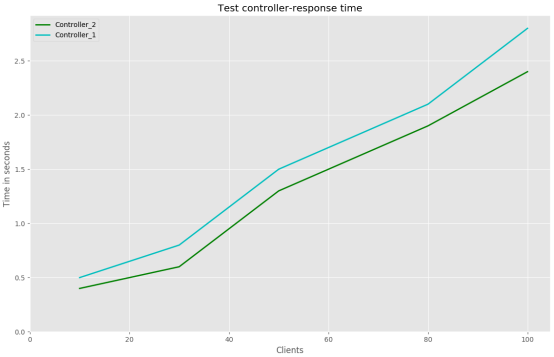

Fig. 7. fig:Responsetime from the controller to client between 10 to 100 clients

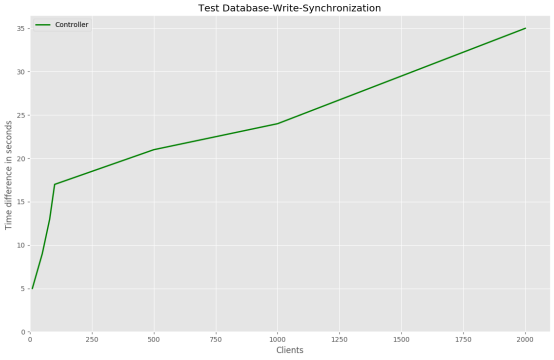

Fig. 8. Database Synchronization time between two nodes

The controller itself has two parts besides the communication interfaces. The first part consists of a data store where all information is aggregated and combined into one state model. This model consists of the topology information from the SDN/NFV controller responsible for the backbone network, including the APs as transition points to the wireless network, and the throughput of each flow on each switch. For this part, We used a powerful distributed database. Then the communication between several devices on the network such as wireless SDN controller, wired SDN controller, VMACs, AP, and Switch. The communicated is modeled using data modeling language and communication is established using a strong messaging pattern. Thus, The distribution of controllers helps to improve scalability and Distributed database helps in distributing data across two nodes, which leads to maintaining the data consistency and even if one controller is down, the data is still available at another node.

REFERENCES

[1] M. Monaco, O. Michel, and E. Keller, '" Applying Operating System Principles to SDN Controller Design," Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks, ACM, 2013

[2] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado and R. Sherwood, '" On controller performance in software-defined networks," USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE), vol. 54. 2012.

[3] Lior Okman, Nurit Gal-Oz, Yaron Gonen, Ehud Gudes, Jenny Abramov- Security Issues in NoSQL Databases IEEE2011.

[4] Alexandru Boicea, Florin Radulescu, Laura Ioana Agapin - MongoDB vs Oracle - database comparison. IEEE2012.

[5] Zhu Wei-ping, Li Ming-xin- Using MongoDB to Implement Textbook Management System instead of MySQL IEEE2011.

[6] Gast, Matthew,802.11 Wireless Networks: The Definitive Guide,2nd Edition, O'Reilly Media, Inc.,2005.

[7] Ni, Qiang, Romdhani, Lamia, and Turletti, Thierry," A Survey of QoS Enhancements for IEEE 802.11 Wireless LAN", Journal of Wireless Communication and Mobile Computing, Vol.4, No.5,2004,pp547-566.

[8] Ensar Zeljkovic, Tom De Schepper, Patrick Bosch, Ian Vermeulen, Jetmir Haxhibeqiri, Jeroen Hoebeke†, ORCHESTRA: Virtualized and Programmable Orchestration of Heterogeneous WLANs

[9] IEEE Std. 1905.1-2013, "IEEE standard for the convergent digital home network for heterogeneous technologies," 2013.

[10] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," Tech. Rep., 2013.

[11] S. Dandapat, B. Mitra, R. Choudhury, and N. Ganguly, "Smart Association Control in Wireless Mobile Environment Using Max-Flow," IEEE Transactions on Network and Service Management, vol. 9, pp. 73–86, 2012.

[12] N. Soetens, J. Famaey, M. Verstappen, and S. Latre,´ "SDN-based management of heterogeneous home networks," in 11th International Conference on Network and Service Management (CNSM), 2015, pp. 402–405.

[13] K. Xu, X. Wang, W. Wei, H. Song, and B. Mao, "Toward software defined smart home," IEEE Communications Magazine, vol. 54, no. 5, pp. 116–122, 2016.

[14] T. De Schepper, P. Bosch, E. Zeljkovic, K. De Schepper, C. Hawinkel, S. Latre, and J. Famaey, "Sdn-based transparent flow scheduling for heterogeneous wireless LANs," 2017.

[15] G. P. Fettweis, "A 5G Wireless Communications Vision," Microwave J., Dec. 2012, pp. 24–36.

[16] Q. C. Li et al., "5G Network Capacity: Key Elements and Technologies," IEEE Vehic. Tech. Mag., vol. 9, no. 1, Mar. 2014, pp. 71–78.

[17] Rothlisberger David et al., "Exploiting dynamic information in IDEs improves speed and correctness of software maintenance tasks", Software Engineering IEEE Transactions, vol. 3, no. 38, pp. 579-591, 2012.

[18] Herb Sutter, "The Free Lunch Is Over", Dr. Dobb's Journal, 30(3), March 2005. http://www.gotw.ca/publications/concurrency-ddj.htm.

[19] R. Riggio, M. K. Marina, J. Schulz-Zander, S. Kuklin- ski, and T. Rasheed, “Programming abstractions for software-defined wireless networks,” IEEE Transactions on Network and Service Management, vol. 12, no. 2, pp. 146–162, 2015.acm.org article.cfm?id=2618393.

[20] David Beazley, ”Understanding the Python GIL”, PyCON Python Conference, Atlanta, Georgia, 2010. http://www.dabeaz.com/python/UnderstandingGIL.pdf.

[21] Michael McCool, Arch Robison, James Reinders, ”Am- dahl’s Lawvs. Gustafson-Barsis’ Law”, Dr. Dobb’s Parallel, October 22, 2013. http://www.drdobbs.com/parallel/amdahls- law-vs-gustafsonbarsislaw/240162980

[22] IEEE Std. 1905.1-2013, “IEEE standard for the convergent digital home network for heterogeneous technologies,” 2013. [20] A. Ford, C. Raiciu, M. Handley, and O. Bonaven true, “TCP extensions for multipath operation with multiple addresses,” Tech. Rep., 2013.

[23] ACM Std. 2016, ”The EmPOWER mobile network operating system,” 2016. Riggio, Roberto, ACM.

[24] IEEE Std, 2014, ”Opendaylight: Towards a model-driven Sdn controller architecture,” 2014. [10] Medved, Jan and Varga, Robert and Tkacik, Anton and Gray, Ken, ” Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks,” Tech. Rep., 2014

[25] Elsevier, 2004, ”A global output-feedback controller for stabilization and tracking of underactuated ODIN: A spherical underwater vehicle,” 2004. [10] Do, Khac D and Jiang, Zhong-Ping and Pan, Jie and Nijmeijer, Henk, ”. Rep., 2004