Latency and Reliability Aware SDN Controller: A Role Delegation Function as a Service

Dinah Wobuyaga*, Sisay T Arzo†, Harsh Kumar‡ Fabrizio Granelli 4§, Michael Devetsikiotis¶
*§Department of Information Engineering and Computer Science, University of Trento, Trento, Italy
§‖Deutsche Telekom Chair of Communication Networks, Technische Universität Dresden, Dresden, Germany
*†‡§¶‖Department of Electrical and Computer Engineering, University of New Mexico, USA
{*dinah.wobuyaga@studenti., §fabrizio.granelli}@unitn.it, {†sarzo, ‡hkumar, ¶mdevets}@unm.edu

2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC) | 979-8-3503-3286-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/CCWC57344.2023.10099225

***Abstract*—The emergency of machine type and ultra-reliable low latency communication is imposing stringent constraints for service provisioning. Addressing such constraints is challenging for network and cloud service providers. As a trending paradigm, software-defined networking (SDN) plays a significant role in future networks and services. However, the classical implementation of the SDN controller has limitations in-terms-of latency and reliability since the controller is decoupled from the forwarding device. Several research works have tried to tackle these challenges by proposing solutions such as Devoflow, DIFANE, and hierarchical and distributed controller deployment. Nonetheless, these approaches are not fully addressing these challenges. This paper tries to address the problem of latency and reliability by proposing a dynamic controller role delegation architecture for forwarding devices. To align with the microservice or multi-agent-based service-based architecture, the role delegation function as a service is proposed. The dynamic role delegation enables to predict and (pre-)installed flow rules in the forwarding devices based on various considerations such as network state, packet type, and service's stringent requirements. The proposed architecture is implemented and evaluated for latency and resiliency performance in comparison to the centralized and distributed deployment of the SDN controller. We used ComNetsEmu, a softwarized network emulation tool, to emulate SDN and NFV (Network Function Virtualization). The result indicated a significant decrease in latency and improved resilience in case of failure, yielding better network performance.**

Keywords: Software Defined Networks, Control Plane, OpenFlow, Role Delegation

I. Introduction

Network softwarization paves the way for network flexible and dynamic network management. Software-defined networking (SDN) and network function virtualization (NFV) are the two technologies driving network softwarization. They completely transformed how we design and deploy networks. However, future networks demand is increasing tremendously, especially with the arrival of machine type and ultra-reliable low latency communication. These stringent requirements should be addressed by future networks such as Beyond 5G and 6G networks.

The core concept of the SDN is to decouple the control and the data plane as a way of enhancing network management through global visualization of the entire network [1]. To achieve programmable networks that can easily be adjusted to suit the ever-changing requirements and to simplify management. Ironically, the advantage of the SDN controller is its inherent drawback. In other words, the decoupling would create a distance that induces a delay between the controller and the forwarding device. For example, a single centralized architecture, whereby the controller handles all the requests solemnly, results in long delays, congestion, and a single point of failure, which could result in network operations disruption or poor network performance. This limits the effective performance of the network reducing SDN's benefits as a promising network innovation.

A distributed and hierarchical SDN deployment is proposed to improve the latency and central point of failure. However, this is still not enough to reduce the latency and reliability to the expected level as a controller has to handle a number of forwarding devices and a controller failure also impacts the reliability in the particular domain of the controller managing. Most importantly, network state synchronization impacts the accuracy of the decision by the controllers.

Moreover, 6G networks are expected to introduce in-network intelligence [2] to have a dynamic network response. The approach introduced in [3] is an interesting example, where agents are used as a service design approach while introducing intelligence to the smallest network service functions. Therefore, revisiting the design principles and concept of SDN is necessary. In this regard, in [4] an SDN decomposition technique is presented, which splits the SDN controller functionalities into sub-functions that can be designed as microservices in a containerized environment. In this regard, in contrast to distributed/hierarchical controller deployment, instead of replicating the entire controller, it enables deploying only the required functionalities near the forwarding devices.

In line with this approach, in this paper, an additional role delegation function is designed to enable the forwarding device to dynamically take control of the forwarding function. The developed dynamic role delegation technique decreases the delay due to the communication between the controller and forwarding device, while also increases resilience in case of controller failure. The role delegation function reduces the latency and improves reliability in case of controller failure or congestion in the network that could potentially delay the communication between the controller and the forwarding devices. The suggested approach delegates control roles to forwarding devices which equips them with anticipated flow rules in the flow tables, which they use to route traffic across

the network. The role delegation could be designed as an agent or microservice that dynamically decides the role delegation scenario.

In general, this study suggests delegating control roles to forwarding devices as a way of offloading the controller and improving network recovery in the event of controller failure. The paper presents the following sections: Section I introduction and Section II reviews the literature. The proposed role delegation architecture is presented in Section III. Section IV is dedicated to performance evaluation. Conclusion and future works are provided in Section V.

II. Literature Survey

The quest to improve controller performance has been a point of concern by many researchers. Various solutions have been suggested over the years to counter the issue so as to improve SDN's performance. The two main architectural deployment scenarios are;

A. Centralized SDN Controller Architecture

This architecture comprises a single centralized SDN controller that manages the entire network. The forwarding device's role is only to route traffic across the network following instructions formulated and installed in their flow tables by the controller.

Centralized controller deployment of approaches is close to the initial concept of SDN as they use a single centralized controller but with multiple controllers connected to it. All the controllers play the same roles and in case one fails the others take over and continue managing network operations. It uses the master-slave mechanism whereby one controller acts as a master till it fails then the other assumes the role. This eliminates the challenge of a single point of failure. However, these solutions introduce other considerable drawbacks that range from load balancing, long network downtime in case the controller fails, latency constraints to mention a few [5]

B. Distributed/Hierarchical SDN Controller Architecture

Several suggested approaches consider distributed controller deployment, including HyperFlow [6]. These solutions consist of several instances of the controller performing the same roles. All the controllers must remain in synchronization to ensure consistency, nonetheless, this is still challenging and include controller positioning, which involves determining where exactly is suitable on the network to place the controller [7]. Furthermore, such solutions are difficult to configure, set up, troubleshoot issues, maintenance, communication inconsistency among controllers and long path length [5] [8]. These limit the effectiveness of the distributed controller approach.

The distributed controllers can be set up in a hierarchical architecture, with the root controller being at the top and the sub-controllers being at the bottom [9]. which mostly has two layers. To sustain communication, the root or main controller and the local or sub-controllers must always be inactive synchronization [10]. Before they may handle a set of forwarding devices near the local controllers, they must first get authorization from the root controller. They get updated instructions from the root controller on a regular basis and transmit them to the switches they govern [11]. Through delegation of power among multiple levels of the control hierarchy, this technique allows for logically decentralized control. Some management roles are delegated by the root controller to the sub-controllers [12]. The sub-controllers can only manage the sub-domain they are in charge of. Because Packet-IN messages from multiple forwarding devices are processed by the respective controller instead of being forwarded to the main controller, reducing the controller burden which improves the problem of a single point of failure. Even if the root controller fails, the local controllers respond to the forwarding devices for a period of time until the problem is resolved, ensuring network operations continuity. However, when the hierarchical levels increase, the path length tends to expand, which leads to the challenge of high levels of latency [13]. Furthermore, in large networks, root controllers are overworked because they receive intra-domain information from local controllers that is irrelevant to them. The hierarchical-based controller solution to SDN difficulties does not completely address these issues.

An interesting work that uses a blockchain approach is presented in [14] aiming at overcoming the problem of a single point of failure by implementing blockchain technology to develop multi-controller SDN topologies. With blockchain, you can create a distributed ledger and communicate with multiple nodes in the network without having to trust each other. It is possible to construct and run decentralized programs on several machines using this method. HyperFlow [6], is a NOX controller enhancement that improves the logical centralization of many controllers. Hyper flow localizes decision-making to individual controllers by passively synchronizing the views of the Openflow controllers. It is conceptually centralized but physically distributed. Another approach called KANDOO is discussed in [15], in which the flows are divided into two tiers in this model: small and elephant, with the elephant flows being managed by the main controller and the tiny flows by the sub-controllers. Each tier is made up of controllers that handle specific traffic. These controllers are completely independent of one another and no communication between them. This restricts their usefulness in the second layer, which may contain services that require a global view of the entire network. Onix [13] may operate in a restricted federated mode, which allows two Onix instances to exchange summary views of the networks they administer. This information sharing is limited to Onix instances under the same authority, and it allows the Onix NIB to represent enormous data planes in a compact manner. These two levels can only work effectively when this communication is maintained. However, they perform different functions.

Generally, the Hierarchical control plane approach seems to partially solve the problem at hand as their root control can still act as a bottleneck especially when the network consists of high levels of global flows. Hence calls for a better approach to this challenge. Besides most of these approaches are still faced with controller placement problems. As the question of

where to position each of the controllers to reduce the latency and ensure some level of reliability is still an open challenge. With this, there is a need for a better and simple approach to improve SDN performance.

C. Controller Role delegation Architecture

Several academics have proposed different data plane methodologies for offloading the SDN controller to improve the performance of SDN-based networks by lowering signaling overhead between switches and the controller and improving network resilience. P4 [16] proposes an approach to making the data plane stateful with the use of the P4. P4 is a programming protocol-independent packet processor [17]. It is an Open source domain-specific programming language for networking devices. In SDN, it specifies how data plane devices process packets. With this, every single switch is programmed with P4 which enables it to perform the port-knocking application and work as a fully authenticated unit on the network. It enables the delegation of authentication to the forwarding plane. It makes use of stateful memories to store information about various packets. Moreover, it keeps additional information between the processing stages.

The InSPired switches [18] approach aims at delegating the SDN control roles by redrawing the line that separates both the controller and switches roles. This approach enhances control and switches scalability. Besides, it requires fewer switch resources to interact with the controller via the OpenFlow protocol. However, this solution is limited due to the controller placement problem, which requires optimal placement of controllers to limit the challenge of long path length. In case of the long distance between the controller and the switches, there will be a huge latency. Other existing solutions suggest installing replicated controllers to solve the problem of a single point of failure. Nonetheless, these solutions pose other drawbacks with the use of passive controllers in case the main controller fails, which results in redundancy.

DevoFlow [19] applies rule cloning and switches local action to partially devolve control plane rules to the data plane. It also decouples global visibility from the controller. Devoflow uses wild-carded OpenFlow rules, this reduces the interaction between switches and the controller, as highly frequent events are handled by the switches, reducing controller visibility limits the whole network's effective management. DIFANE [20] handles traffic in the data plane by the use of intermediate authorized switches which contain pre-installed rules by the controller. The controller installs rules in the selected switches. These rules are based on route traffic across the network without consulting the controller. However, the solution has no clear basis on what to consider to divide the workload between the controller and the underlying authority switches, this limits the scalability of high-level policies.

III. Proposed Architecture

In this section, we present the proposed role delegation technique. The aim of the architecture is to enhance network latency and resilience. This work is based on the SDN controller decomposition architecture proposed in [4]. SDN controller functions are decomposed into sub-function that can be dockerized and placed in an edge/fog environment. And these functions are designed as microservices, which alternatively could also be developed as multi-agents [3], that can be deployed in a distributed environment.

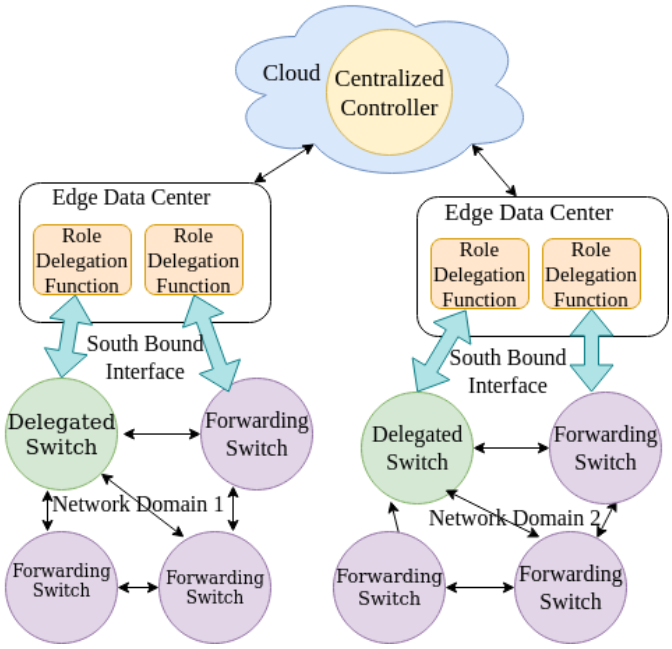

Fig. 1. Proposed Architecture

A. Architectural Design

Here we proposed an architecture, which is aimed at designing a role delegation function that offloads roles from the central controller to the forwarding switches in a particular domain. Figure 1 shows the proposed architecture. The architecture has three layers and can be considered an evolution of the hierarchical SDN controller deployment with a role delegation functionality, which is designed as a microservice. In other words, instead of replicating the entire SDN controller in each place, deploying only the required functions at the vicinity of the forwarding devices that the controller has to manage. In this case, replicating the role delegation function in each edge data center while instantiating backups functions for increased reliability. As can be seen from the figure, the top layer places the central controller hosting the global information for the network, including network state and other common functionalities such as end-to-end path or topology, inter-domain or intradomain information about the network.

The main principle behind the role delegation function is to dynamically determine the delegation rules so that the flow entry is pre-installed on the appropriate forwarding switches. This depends on the anticipated incoming traffic, network state, required QoS demand, and available path and bandwidth. The assumption is that incoming traffic has a set of demands and constraints so that traffic can be categorized into different

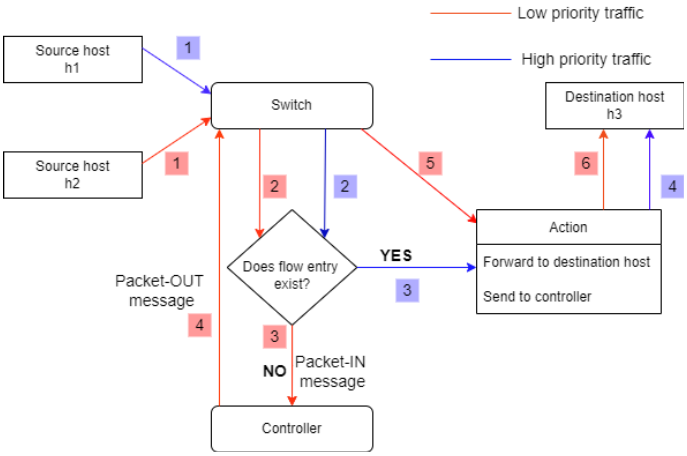

Fig. 2. Packet Flow

traffic classes. Highly prioritized representing critical service, medium priority class with better tolerant but with significant impact with delay and reliability, and low prioritized traffic comprising of services with least latency and reliability demand. Roles are delegated by pre-installing flow entry rules into the switch flow tables so as to enable them to forward traffic without requesting instructions from the controller.

The communication between the role delegation function and the forwarding switch can be the classic southbound interface such as OpenFlow interface. However, the communication between the role delegation function, which could be dockerized microservice, should be using RESTfull API/gRPC/Websocket as suggested in [4]. Moreover, in this architecture, the role delegation can respond dynamically which is developed as a modular service function that can be deployed in an edge/fog/cloud environment. This paves the way for in-network intelligence.

B. Packet Flow Life-Cycle

The main advantage of role delegation is the possibility of forwarding an incoming packet directly by using the proactively installed flow roles by the role delegation function. Specifically, whenever a packet arrives at the forwarding switches, it is forwarded directly to its destination since the switch holds the necessary instructions and details on how the packet should be treated. This is because the role delegation function provides pre-installs the flow roles to the appropriate forwarding switches, delegating the switch to handle the packet forwarding functionality.

Figure 2 depicts the packet flow in pre-installed forwarding switches. Whenever a packet arrives from the source host to s1 whose flow entry exists in the switch, s1 does not need to send a packet-In message to c1 instead routes the packet directly as indicated by the blue dotted arrow to its intended destination. We assume this packet was coming from a highly prioritized host that running critical services. In the flow table of the switch, there is a pre-installed flow entry that matches the packet received that consists of the src, dst, priority, and action to be performed. Comparing the flow of packets in SDN with a single controller, hierarchical control based architectures, and with packet flow after implementation of role delegation, several processes lead to high latency in the latter compared to the former, these processes result from the sole dependence of the entire network on the controller to carry out its operation. With delegating the control roles to forwarding devices step 3 and 4 will be eliminated in the packet flow cycle since there is no need of having packet-IN and packet-OUT messages. Besides the packet processing which would be done by the controller is now done by the switches with minimal delays that would result from setting up flow entry rules and installing them in the switch flow tables.

IV. Performance Evaluation and Simulation Scenario

Here we present the performance evaluation using latency and resilience using simulation as a proof of concept.

A. Implementation Scenario

The simulation is performed by setting up three scenarios of SDN using ComNetsEmu [21]. ComNetsEmu is an emulation software that combines a dockerized environment and Mininet. It enables an improved emulation of various computing applications in a given network. It is an extension of containernet project that implements its concepts. Mininet is used for emulating the topology and interacting with genuine virtual networks that run on real kernels, switches, and software code. For the virtual switches, OpenFlow is used as a southbound interface. Software-based virtual switches, hosts, and controllers are software-based duplicates of physical hardware devices. They behave similarly to discrete devices and perform the same network operations. Compared to Containernet, comNetsEmu extends the Mininet using a slightly different approach. The main aim is to utilize "sibling containers" in emulating network systems along with network computing. In comNetsEmu divers Computing In-Network computing applications are supported. On the other hand, the incorporated Mininet supports the SDN standard by providing both a CLI and an API for interactive commands and automation.

Using the above emulation environment, we have implemented three scenarios. In the first scenario, we show the basic notion of SDN controller where we used a single central controller managing forwarding devices. In the second case, we replicate the controller to implement a hierarchical topology. Finally, we deployed our solution of role delegation. In each of these scenarios, we use the ping command to test and assess the network's latency and dependability by sending ICMP requests to each node and monitoring the responses.

Here, two hosts are pinged, for example, h2 ping h4, and the packet reception delays are observed, and the inter-packet arrival rate from the source to the destination host is calculated. The pingall command is used to see which nodes are reachable to each other. In the first and second implementations of this research, this can only be successful if the controller(s) are running. However, in the suggested approach, the pingall is

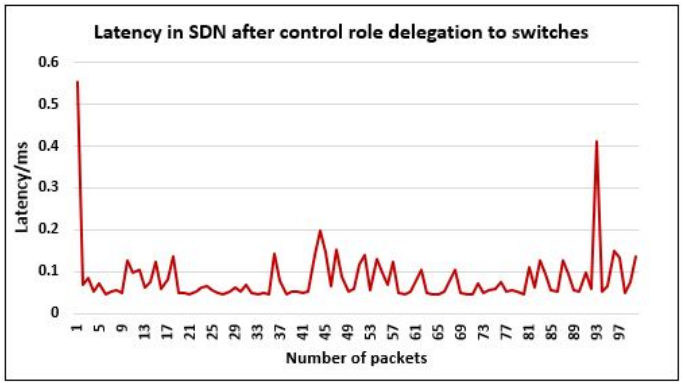

Fig. 3. Improved Latency Using the Role Delegation Technique

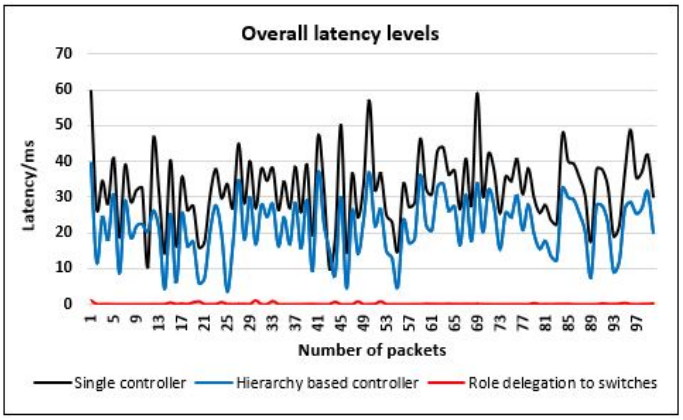

Fig. 4. Overall Latency For The Three SDN Cases

successful regardless of the controller state. The data was then analyzed and plotted, which are presented in the next subsection.

B. Evaluation Results and Discussion

By applying the role delegation algorithm, we have calculated the latency. Figure 3 depicts the average latency as a function of packets, showing the delay incurred by each packets.

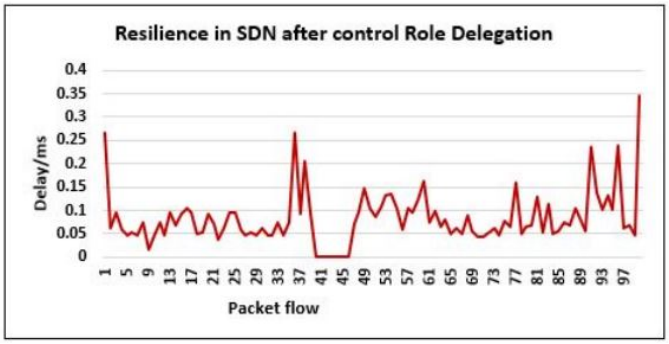

Fig. 5. Improved Resilience Using the Role Delegation Technique

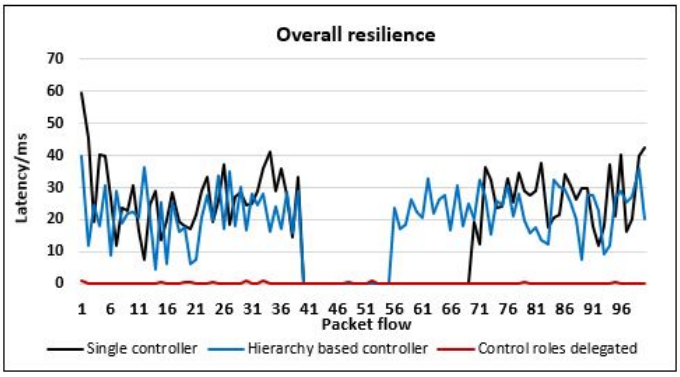

Fig. 6. Resilience Of The Three SDN Cases

Fig 4 represents the overall latency for the three different SDN deployment scenarios. The black line represents latency in a single centralized controller SDN. Blueline represents the hierarchical-based controller case. And red line indicates the latency in the case of the delegation scenario. As can be observed from the figure, the latency for the three cases varies, where the third case having a considerably low latency with an average of 0.157ms. The hierarchical controller deployment shows the next best with an average latency of 22.19ms. And the single controller case incurs the biggest average latency of 32.8ms among the implemented scenarios. In each case, the latency is measured in the same way, by pinging two hosts that intend to communicate with each other where one is considered as the source and the other as the destination. The variation in the level of latency is because of the difference in the architecture of each scenario with the main emphasis made due to the involvement and granularity in the packet handling decision of the controller(s).

As a reliability measure, we have tested the proposed architecture using the resilience parameter. Fig 6 depicts the resilience of the three emulated scenarios. Resiliency measures network downtime. As can be seen from the figure, the network downtime varies for each case. The single controller scenario is observed to have long downtime. Whereas the hierarchical-based deployment is the next to see longer network downtime. The least network downtime was observed for the control role delegation scenario. In the role delegation case, only a few (5) packets are lost as a result, whereas 15 packets and 30 packets of 100 packets are lost in the hierarchical and single controller case, respectively. This is due to the long time lag between controller failure and recovery for the central controller and hierarchical controller. The loss is 30% and 15% in the single controller and hierarchical controller scenarios. This significantly impacts the network performance. Especially, for network services such as ultra-reliable low latency (URLL) services with stringent requirements in terms of reliability and latency.

1) Latency: The overall latency results for the three SDN instances are summarized in Table I. According to the findings, the maximum level of latency encountered by SDN with a

TABLE I
OVERALL LATENCY LEVELS

Scenario	Highest	Lowest	Average
Single controller	59ms	10ms	32.44ms
Hierarchy based controller	35ms	5ms	21.79
Roles delegated to switches	0.4ms	0.05ms	0.157ms

TABLE II
OVERALL RESILIENCE LEVELS

Scenario	Total packets lost	Average packets lost
Single controller	30 packets	15 packets
Hierarchical based	15 packets	7packets
After role delegation	5 packets	2 packets

single centralized controller is 59ms, with the lowest level being 10ms and an average of 32.44ms. These levels are very high for a busy network environment such as data centers. The adoption of the hierarchical controller-based was thought to alleviate the problem by lowering these levels, but the latency remained quite high, with the maximum being 35ms and 5ms and the lowest and average being 21.79ms and 21.79ms, respectively. When compared to levels after control role delegation, the levels drop dramatically, with 0.4ms being the highest and 0.05ms, 0.157ms being the lowest and average, respectively. In comparison to the hierarchy technique, this is a tremendous improvement.

2) Resilience: Table II shows a summary of overall SDN resilience for the three simulated cases. According to the findings presented in the previous section, network downtime continues to decrease in each case, with the greatest reduction occurring after control role delegation, and only a few packets are lost as a result, whereas 30 packets and 15 packets of 100 packets are lost in the first and second cases, respectively, due to the long time lag between controller failure and recovery. The loss is 30% and 15% in each scenario, which is a significant amount, especially in networks with vital services operating over them.

V. CONCLUSION AND FUTURE WORKS

This work presented an architecture that enables delegating roles to forwarding switches. A role delegation function is designed that considers the network state within the domain while communicating with the central controller/network state database/ which could be placed in a cloud. This function could be designed as a microservice or agent and the it can be dockerized and deployed in an edge or fog environment. As proof of concept, a simulation scenario is designed using ComNetsEmu emulation tool. Using the simulation scenario, we have evaluated and compared three different cases. The three SDN scenarios were with a single centralized controller, hierarchical control and our proposed roles delegation deployment. We used the ping command to obtain results in each of the SDN scenarios, which measures the time lag of a single packet from its source to its intended destination. For latency, the number of packets in the flow was then analyzed corresponding to the delays experienced, while for resilience, the delay and number of dropped packets when the controller was down were measured. As a future work we are aiming to use a machine learning based role delegation prediction to enable autonomous role setting.

VI. ACKNOWLEDGMENT

This work has been partially funded by partially by the US National Science Foundation under the New Mexico SMART Grid Center - EPSCoR cooperative agreement Grant OIA-1757207.

REFERENCES

[1] O. A. A., E. Seun, A. A. O., and O. F. Y., "Introduction to software defined networks (sdn)," *International Journal of Applied Information Systems*, vol. 11, no. 7, pp. 10–14.

[2] Hexa-X. (2021, Apr.) D1.2 – Expanded 6G vision, use cases and societal values — including aspects of sustainability, security and spectrum. [Online]. Available: https://hexa-x.eu/wp-content/uploads/2021/05/Hexa-X$_D$1.2.*pdf*

[3] S. T. Arzo, R. Bassoli, F. Granelli, and F. H. P. Fitzek, "Multi-agent based autonomic network management architecture," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3595–3618, 2021.

[4] Arzo, Sisay Tadesse, Scotece, Domenico, Bassoli, Riccardo, Barattini, Daniel, Granelli, Fabrizio, Foschini, Luca, and Fitzek, Frank H. P., "Msn: A playground framework for design and evaluation of microservices-based sdn controller," *Journal of Network and Systems Management*, vol. 30, no. 1, pp. 1573–7705, 2021.

[5] X. A. Kala and S. Vinod, "Centralized controllers of sdn and its problem spaces," *International Journal of Applied Engineering Research ISSN*, vol. 12, 02 2017.

[6] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *HyperFlow: A Distributed Control Plane for OpenFlow*, 04 2010, pp. 3–3.

[7] J. Lu, Z. Zhang, T. Hu, P. Yi, and J. Lan, "A survey of controller placement problem in software-defined networking," *IEEE Access*, vol. 7, pp. 24 290–24 307, 2019.

[8] F. Bannour, S. Souihi, and A. Mellouk, "Distributed sdn control: Survey, taxonomy and challenges," *IEEE Communications Surveys Tutorials*, vol. PP, pp. 1–1, 12 2017.

[9] E. da Silva, O. Blial, M. Ben Mamoun, and R. Benaini, "An overview on sdn architectures with multiple controllers," *Journal of Computer Networks and Communications*, vol. 2016, no. 9396525, pp. 2090–7141, 2016.

[10] M. A. S. Santos, B. A. A. Nunes, K. Obraczka, T. Turletti, B. T. de Oliveira, and C. B. Margi, "Decentralizing sdn's control plane," in *39th Annual IEEE Conference on Local Computer Networks*, 2014, pp. 402–405.

[11] D. Giatsios, K. Choumas, P. Flegkas, T. Korakis, J. J. A. Cruelles, and D. C. Mur, "Design and evaluation of a hierarchical sdn control plane for 5g transport networks," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[12] E. Amiri, E. Alizadeh, and K. Raeisi, "An efficient hierarchical distributed sdn controller model," 02 2019, pp. 553–557.

[13] Y. Fu, J. Bi, Z. Chen, K. Gao, B. Zhang, G. Chen, and J. Wu, "A hybrid hierarchical control plane for flow-based large-scale software-defined networks," *IEEE Transactions on Network and Service Management*, vol. 12, pp. 1–1, 06 2015.

[14] T. Krishnamohan, J. Kugathasan, P. Peramune, and A. Ranaweera, "Blockflow: A decentralized sdn controller using blockchain," *International Journal of Scientific and Research Publications (IJSRP)*, vol. 10, p. p9991, 03 2020.

[15] S. Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," *HotSDN'12 - Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks*, 08 2012.

[16] A. Almaini, A. Al-Dubai, I. Romdhani, and M. Schramm, "Delegation of authentication to the data plane in software-defined networks," in *2019 IEEE International Conferences on Ubiquitous Computing Communications (IUCC) and Data Science and Computational Intelligence (DSCI) and Smart Computing, Networking and Services (SmartCNS)*, 2019, pp. 58–65.

[17] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 87–95, jul 2014. [Online]. Available: https://doi.org/10.1145/2656877.2656890

[18] R. Bifulco, J. Boite, M. Bouet, and F. Schneider, "Improving sdn with inspired switches," in *Improving SDN with InSPired Switches*, 03 2016, pp. 1–12.

[19] A. Curtis, J. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," 08 2011, pp. 254–265.

[20] Y. Minlan, R. Jennifer, J. Michael, Freedman, and W. Jia, "Scalable flow-based networking with difane," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, 2010.

[21] Z. Xiang, S. Pandi, J. Cabrera, F. Granelli, P. Seeling, and F. H. P. Fitzek, "An open source testbed for virtualized communication networks," *IEEE Communications Magazine*, vol. 59, no. 2, pp. 77–83, 2021.