Unified model towards Scalability in Software Defined Networks

Amit Nayyer, Aman Kumar Sharma and Lalit Kumar Awasthi

Software Defined Network is a paradigm that enables the network administrators to manage and control the network from a centralized location using software programs. The limitations and complexities of the traditional network are handled by separating the control plane from the data plane in this setup. The main idea is to have centralized control over network devices. Scalability is one of the main concerns in such a paradigm. Various independent solutions to improve scalability are available in the literature. In this paper, two approaches for the solutions of scalability are studied and implemented: Topology based solutions and Routing based solutions. Different evaluation parameters are selected for evaluating a framework combined with a specific routing protocol. Frameworks from different categories are implemented along with different routing protocols. Putting the routing protocols one by one in a single framework, nine such models are implemented for evaluation. Results are provided for consideration before network setup for the network administrators. Furthermore, the discussions based on the results are presented regarding the combination of a particular framework with routing solution to get better results in specific conditions.

Keywords: Control plane, Framework, Routing Algorithm, Scalability, Software Defined Network

1. INTRODUCTION

Software defined network (SDN) is a novel network paradigm that has the potential to ease the management, programmability and control of the network. Research community and industries have grown attention for various advantages of this model. The conventional networks are complicated and hard to manage. New trends like mobile, cloud, big data etc. are providing new challenges that are difficult to handle with stereotype approaches as discussed in W. Xia et al. [2015]. Traditionally, distributed protocols provide distributed management and fault tolerance. Any network having switches and routers performing independent control decisions makes debugging and controlling of such networks very difficult. The existing internet model poses various challenges to the deployment and evolution of protocols. SDN carries the opportunity to innovate with the simple and easy configuration in network computing. The vertical integration of control decision and forwarding elements of traditional network is broken to provide the centralized network control and flexibility to the program. In traditional network setup, each vender has its firmware and other software to operate such devices in a proprietary and closed way. This model halts innovation in network technology, increases the cost of operating and controlling such networks as discussed in D. Kreutz et al. [2015]. SDN paradigm decouples the forwarding hardware from the control decision. Broadly, it is characterized by its two distinct features as, decoupling of the data plane from the control plane and providing programmability in the network. In SDN, devices like switches only forward the packets as per rules provided by the controller. A controller can oversee the underlying network; therefore, it provides an efficient platform to implement various services and applications as disscussed in H. Farhadi et al. [2015]. The separation of both planes can be realized by using a well-defined Application Programming Interface (API) in the network. SDN is taken as hardware-independent technology of next-generation, where software-based programmable controller plays its role of controlling the network that consists of devices from different vendors. The controllers and OpenFlow supported switches are the prime components in SDN paradigm.

Authors' addresses: Amit Nayyer, Aman Kumar Sharma, Department of Computer Science, H.P University SummerHill, Shimla, Himachal Pradesh India; Lalit Kumar Awasthi, Department of Computer Science and Engineering, NIT Hamirpur, Himachal Pradesh, India.

OpenFlow as proposed by Open Networking Foundations [2015], defines communication between switches and the controller in the network. It is considered as a building block of SDN. It allows creating a global view of the network at the control plane and provides a system-wide, consistent programming interface to the centrally programmed network devices. OpenFlow was initially proposed to experiment in the campus network. Open Networking Foundation is an industrial driven organization that promotes SDN and control the standards for OpenFlow protocol. Both SDN and OpenFlow are related to each other in such a way that the success of one is promoting the success of other. The deployment of the network become instant with the use of OpenFlow enabled switches. It is also known as the southbound interface, as it allows interaction between the control plane and the forwarding devices in the network. OpenFlow protocol allows convenient manipulation of flow table entries like insertion, deletion, modification etc. through secure TCP channel.

The separation of the control plane from the data plane leave the devices of data plane as merely forwarding devices. The processing capabilities are taken out from such devices, and therefore they work as slave devices to follow the instructions given by the controller. With the target to reduce the number of incoming requests towards itself, the controller sometimes installs proactive rules in the forwarding devices. The flow-based paradigm also prevents the controller from overloading, as once a rule for a particular flow is provided to the switch, it will not interrupt the controller until a new flow arrives at the switch or the existing one expires. Some researchers proposed to add functionality in the data plane for increasing scalability of the control plane. These proposals increase the load of the data plane. Consequently, other issues of concern like limited memory in data plane arises. Furthermore, these proposals violate the generality of forwarding devices.

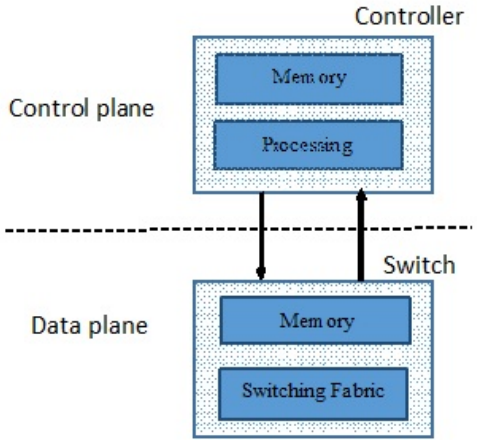

Figure. 1: The Controller and Switch model in SDN

Decoupling data plane and control plane brings scalability concerns in both the planes. The burden is shifted more towards the control plane and therefore its scalability is of more concern to researchers. The controller can become a bottleneck to handle a large number of requests from the switches, as the paradigm pushes most of the responsibilities towards the control plane. Flow setup messages can choke the controller in setups like data centers, where large number of flow initiation rate of messages at a particular duration can occur. The solutions for increasing scalability of the control plane in a topological manner can be broadly divided as Horizontal model-based solutions and Vertical model-based solutions. These solutions work on different ways of arranging the devices to control the network. Hyperflow, Onix, Kandoo, Laman as provided in M. Karakus et al. [2017] and A. Nayyer et al. [2019] are few to name that focuses on providing scalability by utilizing different topological setups. To reduce the load of the controller

and to make it more scalable, some proposals are designed to shift the responsibilities of the control plane towards the data plane. Other methods like rule cloning, sampling, local actions and triggers are used to reduce the flow requests and flow statistics between control and data plane. Broadly, the packetin messages and flowstatistics messages are the two types of messages that can overload the controller. The packetin messages are requests received by the controller, and they need urgent processing and reply based on statistics available. A new statistic can be requested for creating a reply to packetin message. Therefore, handling of packetin messages requires instant and additional efforts. Processing of packetin messages more efficiently will result in improving the scalability of the controller and network thereof. The OpenFlow protocol independently tries to focus on scalability concerns and provide features in its different versions to provide the solutions. Group table mechanism, Multiple controller support, Meter Table and Control Message Quenching are some of the features of OpenFlow designed to scale the network.

The remaining paper is organized as follows. In section 2, motivation for work is provided. Section 3 discusses related material to this work. Unification and selection of various system frameworks with routing protocols are provided in Section 4. Section 5 presents evaluation experiments and discusses results. Section 6 presents the conclusion and future scope.

2. MOTIVATION

Whatever large may be the capabilities of the controller are, it may not scale well as the network grows. Even if the controller handles the flows, the QoS cannot be maintained during large flows by a sizeable network, and it cannot guarantee the same level of services. As the network is flow-based, initialization delay should be as small as possible for the smooth functioning of the network. The controller can easily become a bottleneck when excess messages are triggered by the data plane. The performance of the controller is focused mainly to increase the scalability in the network. The performance is accessed on flow setup time, link utilization, statistics availability etc. of the controller. Throughput of the control plane is one of the performance parameters to measures the scalability of the network. Different solutions related to topological setups are worked out by researchers without considering the best routing solution for it. Similarly, different routing algorithms are provided independently of the frameworks to provide the best results. The enhancement of controller capability may not be enough for the situation with high flow rates. In various research articles such as D. Erickson et al. [2013]., it is stated that 10 million flows can be produced by a 100 switches network. NOX claims that it can handle 10k requests per second. These statistics show that the network is prone to scalability issues when large number of requests are generated.

3. RELATED WORK

The combination of SDN and OpenFlow provide flexibility in managing the network. There are additional attributes that you can work upon in SDN as compared to the traditional network. The software-based controller proven to be more easily handled and provide better functionality by monitoring the data plane under its control. Control-Data plane interface and Application-Control plane interface are used to provide communication between various devices and application for different services. Some articles such as M. Karakus et al. [2017] also discussed scalability by using the concept of parallel approaches to execute the programs. Due to the reason that these approaches are dependent on the architecture available and may not work with all types of systems, they are not considered in this article for evaluation purposes. As this article is exclusively focused on the scalability of the control plane, the methods to increase the scalability at the data plane are not considered. Different factors, like processing power, available memory, and software implementation of data plane devices, can affect scalability in this plane. The solution approaches for scalability is broadly divided into two categories. The discussion on control plane scalability is organized into Topological based solutions and Routing based solutions as in figure 2.

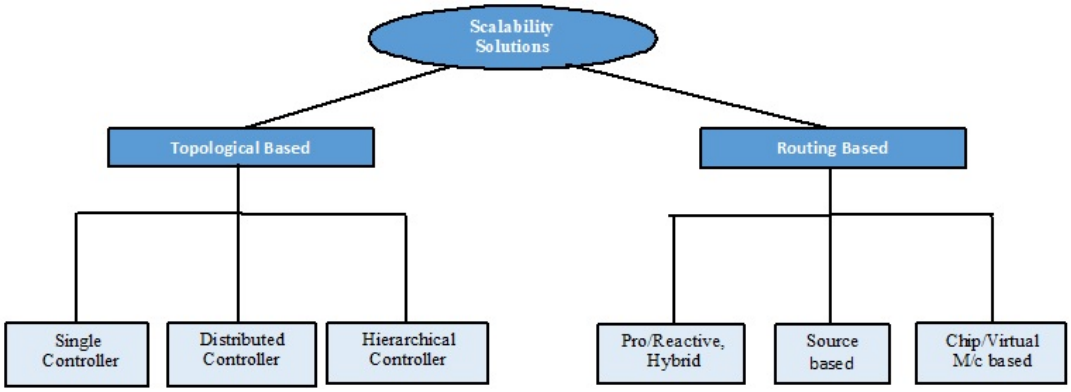

Figure. 2: Control plane approaches for scalability in SDN

3.1 Topological based solutions.

Topological solutions are those where the impact of a problem is reduced by arranging devices in different ways within the network. In SDN to increase scalability, the controller arrangements are considered, and various frameworks are designed. In a single controller centralized system, there is only one controller in the network with visibility of all the events. The responsibility of network management lies totally on it. The design is simple and easy to manage. It suits only for the requirement of a small network as it may fail due to scalability issues when the load increases. The solution came in the form of distributed topologies where multiple controllers are used to distributing the load among them. Scalability and failure handling are the main advantages of distribution. Issues like communication overhead between controllers, latency due to synchronization, consistency between controller etc. are there in the distributed setup. In another type of arrangement, i.e. hierarchical setup, the lower controller in the hierarchy maintain the local view, whereas the upper one maintains the global view.

NOX in N. Gude et al. [2008] allows programs to be written as a centralized one with high-level names. It was proposed as an operating system for better network management of large and complex system. The packets with the same header are considered as a flow, and flow-based granularity helps in improving scalability issues of the network. Due to the limited processing capability of a single controller, flow setup time rises drastically with an increase in the number of requests in a network. NOX extends the work of ETHANE in M. Casado et al. [2007] as it attempts to scale the centralized paradigm, and it provides a general programming interface that makes it easier to support current management tasks.

HyperFlow in A. Tootoonchian et al. [2010] is logically centralized but a distributed system. One of the major design goals is to provide a more scalable solution to the single controller-based solution. It localizes the decision making and minimizes the response time of the control plane. The topology supports any number of the controller in the system and allows the sharing of network-wide view with its neighbors to construct a global view. It is implemented as an application of NOX and allows reusing its applications with minor changes. A different system of publish/subscribe paradigm is implemented for cross controller communication. For a network-wide view in HyperFlow, the events that change the state of a system are published. Another controller subscribes the published events to reconstruct the state. HyperFlow enables OpenFlow deployment in the data center and enterprise network. It allows the deployment of any number of controllers. Synchronization is the main overhead for this topology solution.

ONOS in P. Berde et al. [2014] is a distributed and open-source SDN platform that provides scalability and performance of the networks. It allows horizontal scaling to eliminate single point of failure. An event notification framework is added that made changes to the data-store, and a caching layer is introduced. It creates a useful abstraction, global network view on the system to

run on multiple servers, to provide control plane scale-out and fault tolerance feature. Applications read from the global view to take the appropriate policy and decision making. Applications update OpenFlow managers regarding the changes made by them. Scalability is the responsibility of individual ONOS instance at each controller for propagating state changes between the switches. It provides fault tolerance by distributing the jobs among available controllers in case of failure of any instance. ONIX in T. Koponen et al. [2010] was the first distributed controller that provides global network view, and ONOS was influenced by it.

Elasticon in A. Dixit et al. [2014], controllers can be added or removed dynamically based on the network conditions. A switch migration process is proposed, which transfer a switch from one controller to another. The pool for the controllers can shrink or expand dynamically as per the load of the controller. The load of the controller is checked periodically, and in case the aggregated load exceeds the maximum capacity, the resources are grown by adding new controllers. The resource pool shrinks in case the load falls below a particular threshold. A load measurement module is maintained on each controller to periodically report the CPU utilization and network I/O rates. A switch is assigned to a nearby controller to reduce the control plane latency. Elasticon provides switch migration, controller load balancing and elasticity in the SDN.

Kandoo in S. Hassas et al. [2012] framework provide scalability by creating two-layers of the controllers. At the bottom layer, the controllers are responsible for switches under their control. Controllers are not connected to each other at the same layer; instead, they are only connected to the controller at the upper layer. At the top layer, a logically centralized controller maintains the global network view and distributes the state to other controllers under it. Most of the events are local and handled at bottom layers of the setup; this excludes the top layer controller to be unnecessarily overloaded for requests that can be handled locally. The Kandoo focuses on the idea that frequent events should be handled closer to the data plane.

Orion in Y. Fu et al. [2014] is a hybrid hierarchical control plane for SDN. It divides the network into domains. A domain contains area controllers and a domain controller. The area controller is responsible for routing within its area as in Kandoo. A domain controller is the in-charge of few areas, and it synchronizes itself with other domain controllers as in Hyperflow setup. In Orion, the bottom layer is called the physical layer, and it is composed of OpenFlow switches. The middle layer contains an area controller that is responsible for collecting all information regarding physical devices, managing intra-area topology and processing intra-area routing. Top layers consist of domain controller which synchronize and create a global-wide network view through distributed protocols. Orion reduces the computational complexity of control plane form super-linear to linear.

In A. Nayyer et al. [2019], Laman is a supervisor based framework that divides the workload of the main controller by implementing application-specific controllers in the topology. It distributes the load of the main controller, achieves the goal of scalability and reduces the load of the supervisor controller. The additional controllers in the topology are called contributed controllers, and the number of such controllers may vary depending upon the situation. By analyzing the most demanding service in the network, the supervisor controller assigns the role of handling that service to a contributory controller. The installation of an extra controller with the supervisory controller may incur extra cost, but it provides more scalability to the network. A supervisor controller can respond to the request when there is less load in the network, which is not possible in case of load balancers.

3.2 Routing based solutions.

Designing an efficient routing protocol is a big challenge in SDN. The routing solution should take into consideration the constraints of links, flow tables, bandwidth, congestion, etc., before selecting a particular route. Research work is also done to provide some relief to scalability problems of the network through optimized routing. Reducing the control plane events is one of the common methods to increase the scalability of the control plane. The scalable routing solutions are designed in such a way that they try to reduce the generation of control plane

Table I: Different frameworks available with their attributes from different studies in the literature

Framework	Type	Controller	Challege Addr.	Campus	Enterp.	Cloud	DC	WAN
Ethane	Single	Ethane	Large control plane events	✓	✓	×	×	×
NOX	Single	NOX	Manageability of network	×	✓	×	×	×
Hyperflow	Distributed	NOX	Flow setup latency	✓	×	×	✓	×
Onix	Distributed	Onix	Consistent network state	✓	✓	×	✓	✓
Elasticon	Distributed	Any	Uneven Load distribution	×	✓	×	✓	×
ONOS	Distributed	Any	Scalability and failure	×	×	×	×	✓
Kandoo	Hierarchical	Kandoo	Overhead at ctrl. plane	✓	×	✓	✓	×
Log. X-Bars	Hierarchical	Any	Centralized ctrl. plane	×	×	×	×	✓
D-SDN	Hierarchical	Any	Distribution of ctrl. plane	×	×	×	×	✓
Laman	Hierarchical	Laman	Events overhead control	×	✓	✓	✓	✓
Flow Broker	Hierarchical	Any	Load balancing	×	×	×	×	✓
Orion	Hybrid	Any	Network management	×	×	×	×	✓

events in the network. The proposals aim for better scalability by producing fewer events in the routing scheme. The OpenFlow support reactive routing as proposed by Open Networking Foundations [2015], in which for each message generated towards the control plane, the reply is generated and sent immediately. For each new flow request, the reply message is generated by the controller. When a new packet arises in the network, each switch that is from source to destination path of the packet generates a flow request to the controller, and the controller replies the switches accordingly. The reactive mode of operation by the OpenFlow is not scalable in any way as a large number of messages are generated in this model. The OpenFlow specification itself provide different features like Group Table and Meter Table that can be used by the protocols to increase the performance of the system.

Source routing is developed, keeping scalability in mind. Flow request messages generated in large number from switches in the path of the source to destination, unnecessarily overload the controller. In the SDN paradigm, the controller is aware of topology and forwarding devices in the network; therefore, a variant of source routing is proposed for SDN in M. Soliman et al. [2012]. The path information is inserted in the packet so that the intermediate nodes fetch the next-hop address from the packet and forward it accordingly. Network performance and congestions are not handled by this routing. The Reverse-path can also be calculated by switches inserting its incoming port in the packet after fetching the output port from it.

Explicit Routing in SDN (ERSDN) in H. Owens et al. [2014] is another routing technique focused upon reducing the number of network events by the control plane. The controller needs to modify the flow tables of the switches to configure the routes in the network. The solution is provided by updating the switches that lie in the path of the source to the destination of a packet before the packet reaches these switches. The controller on receiving a flow request from the ingress switch calculates the path and update the intermediate switches along with the ingress switch in the network. The algorithm is evaluated and compared to a normal video over SDN controller, and the results provide decent performance for ERSDN.

The Box covering based routing (BCR) algorithm in L. Zhang et al. [2017] is developed for SDN to the renormalization of networks by dividing it into subnets. The shortest paths are calculated using the Dijkstra algorithm. The number of nodes and edges are decreased without changing the structural characteristics of the network. The network is divided into the boxes, and the shortest paths for intra-links and interlinks between different boxes are calculated. The algorithm reduces the complexity of routing in large networks. The evaluation is done using an implementation in Mininet with Ryu controller; the results depict low latency performance and suitability to large SDN.

To improve the reliability in SDN, an algorithm is proposed based on the clustering of switches by controllers in K. Adrianfar et al. [2017]. It reduces the delay in the network. A hybrid type of architecture is assumed, where a central controller monitors the distributed controllers. Each

controller chooses the path within its own cluster based on metrics such as traffic, bandwidth and distance. The simulation results show the acceptable performance of this method.

Hybrid routing is one in which both the proactive and reactive flow installations are used for routing. Both proactive and reactive schemes have their own advantages and limitations. Proactive flow installation provides a fast response time for a packet, whereas reactive provides a dynamic routing that depends upon the present situation of the network. Reactive flow installation reduces the chances of congestion and failure in the network. In A. Nayyer et al. [2020], Hybrid uses proactive rules at the initialization of the network, and when the links are congested, the switch is programmed to ask the controller for a reactive flow instead of dropping the packet. The controller again finds out the optimal route, keeping scalability in mind. The rerouting is performed in such a way that the proactive flow rules are used wherever possible in the network.

Dynamic Reconfigurable Processor (DRP), QuagFlow and RouteFlow in M. Karakus et al. [2017] are different projects that focus on increasing the scalability and performance of the network by exploiting network-on-chip and routing engines on virtual machines.

4. UNIFICATION MODEL

The SDN is a centralized paradigm where the controller handles the requests of various hosts through switches. Scalability is one of the main issues in this paradigm as discussed in A. Nayyer et al. [2018]. The objective is to perform different combinations of frameworks with routing solutions and evaluate their performance. Instead of proposing any new topology or any optimized routing protocol, the article tests the scalability performance of already proposed frameworks in combination with already proposed routing protocols. This article provides an insight into the complete solution for scalability of the control plane in SDN. The framework or topology itself does not provide a solution to the exiting problem unless a routing protocol is implemented in it. Similarly, a routing protocol requires a topology to work upon. Various approaches are already designed for scalable frameworks and routing optimization, but the research focuses on these two approaches independently. As per our knowledge, this article is a first of its kind that provides the performance comparison by unifying frameworks with different routing protocols.

4.1 Selection of Framework for Routing protocols

From different categories of the topological based solution as in table 1, one solution is taken from each category for the purpose of evaluation. NOX, a famous and earliest single controller system, is implemented for evaluating the performances of routing protocols in a single controller setup. Although a single controller setup is not very famous, it can be a choice for small networks like the campus-wide network. HyperFlow is taken from the distributed one, due to its clean setup description and features. From the hierarchical frameworks, the latest framework Laman is selected.

4.2 Selection of Routing protocols for Frameworks

A reactive approach is the primary approach of OpenFlow networks, and therefore it is taken as the first choice for implementing in different frameworks. Explicit Routing in SDN (ERSDN) is a source-based routing developed specifically to confront scalability issues, it is implemented for testing its performance in different frameworks. Hybrid Routing also provides an impact on increasing the scalability of the network, and therefore it is also selected for evaluation.

4.3 Selection of performance parameters

The performance of a network is measured by different QoS provided by the network. Among different parameters for measuring the performance, the prominent are Traffic measurement, Response Time, Throughput and Consistency.

Traffic measurement: By monitoring the traffic under a controller, different decisions can be made feasible. When the network is in operational mode, there are different packets/messages that are moving in the network. Traffic measurement provides data to the controller for taking

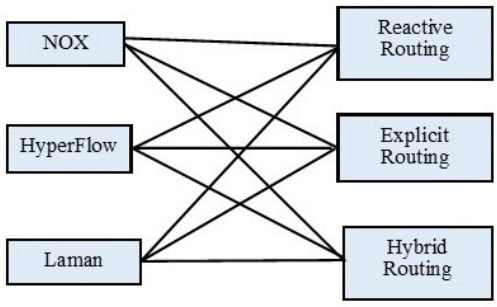

Figure. 3: Selected Frameworks and Routing protocols

the appropriate decision. For example, the large number of packets for a particular type of service may be dealt with providing dedicated and large bandwidth to that service in the network.

Response Time: The heath of a controller can be easily guessed by checking the response time of the controller. Round trip time is generally taken as a famous method to check the response time of the controller as it can be done from a single location.

Throughput: The parameter refers to the rate at which messages are processed by the controller at a particular time duration. High throughput is a desirable parameter for any network as the performance of the network is dependent on this parameter.

4.4 Selection of Architecture

The architecture depicted in figure 4 is the one that is implemented in all the frameworks for setup. Under a single controller, switches are attached to each other using multipath. Only the number of hosts under a switch are incremented for evaluating the performance of the network on different parameters. Under distributed controller architecture, different domains are created with the same architecture and connection between them is made. In the hierarchical setup, the lowest level controller is arranged in the network, as per this architecture too. Selecting a similar architecture for evaluation purpose to all frameworks provides equal opportunity and fairness.

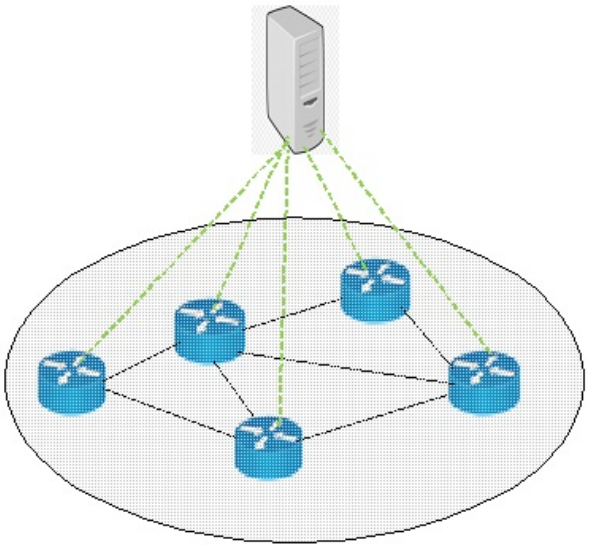

Figure. 4: Architecture with controller-switch connection

5. PERFORMANCE EVALUATION

In this section, the experimental setup of the network is discussed. A network using Mininet emulator from B. Lantz et al. [2010] is deployed to find the effectiveness of different unification models on various parameters. The simulation setup is provided, followed by an analysis of the test case results.

5.1 Simulation Setup

Before implementing a new idea, simulation is performed to check the validity of an idea on network performance. To evaluate the effect on unification of frameworks and routing protocols, networks are deployed in a famous emulator. Mininet is a network emulator used to create virtual hosts, switches, links and controllers. It is used to deploy the setup on a single virtual machine, as shown in figures 5,6 and 7. It is one of the common method used for the academic purpose as hardware testbeds incur high costs. The network is implemented on intel core i-5-7200 2.5 GHz processor with 8 GB RAM on Ubuntu 20.04 with Oracle VM virtual box 6.1. The controllers used are external controllers, with a similar type of virtual OpenFlow switches viz. OVS. Table 2 summarizes the network setup.

Table II: Network Setup

Software	**Version**
Mininet version	2.2.2
Ryu controller version	4.3.0
Open VSwitch version	2.13
OpenFlow version	1.0, 1.3
Ubuntu version	20.04
Processor	$2.5GHz$
RAM	$8GBDDR$
Number of switches	15
Number of hosts	$15 - 15000$

Ryu as available by Ryu [2020] controller is configured and managed as per the requirements. In case of different controllers setup, the controllers are implemented to listen on different port numbers. OpenFlow version 1.3 is configured on Ryu for the purpose of evaluating the response time and throughput, version 1.0 for OpenFlow is used with Cbench as provided by cbench [2020]. Mgen from mgen [2020] is an open-source traffic generator tool, using which the size and pattern like periodic, Poisson, burst etc. can be configured for the generated packets. The packet rates can be easily modified, and logs are created for later use. Different test cases are provided for the evaluation of selected parameters, and the performance results are discussed with each test case.

The network starts initially with 15 switches and 5 hosts connected to each switch, 1 controller is used for a single controller platform setup as in figure 5. In the initial setup, a total of 76 devices (including controller) are deployed. The number of switches remains constant throughout the performance evaluation for each setup in figures 5,6 and 7. Only the number of hosts are increased with a constant factor to each switch. The increase in the number of hosts helps to generate new packetin messages towards controller for flow requests. In distributed and hierarchical setup, 15 switches are divided under 3 different controllers, whereas other setup criteria remain the same during evaluation. In hierarchical setup the framework viz. Laman, requires to add additional controllers attached with the supervisor controller. Three such additional controllers are implemented for evaluation purpose. The performance parameters that are used for the evaluation of different network setup are provided in section 4.

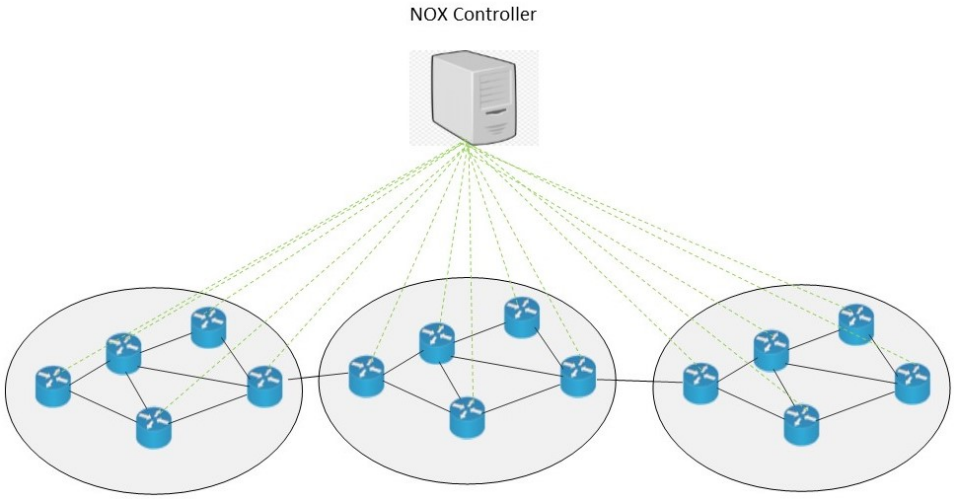

Figure. 5: Simulation setup for Single Controller Framework

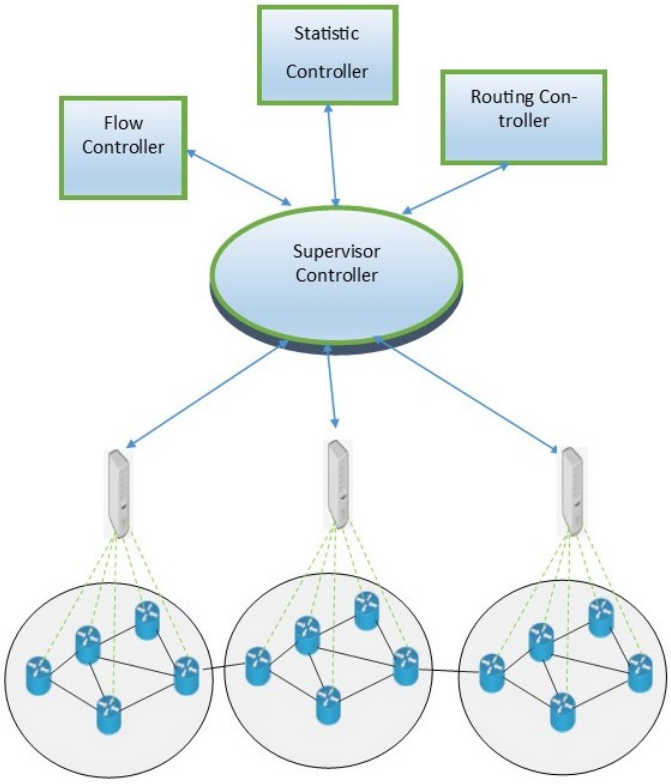

Figure. 6: Simulation setup for Hierarchical Framework

5.2 Results

Test case 1: The first experiment is performed for computing the total number of packet in messages generated towards the control plane in the network. Large number of messages effects

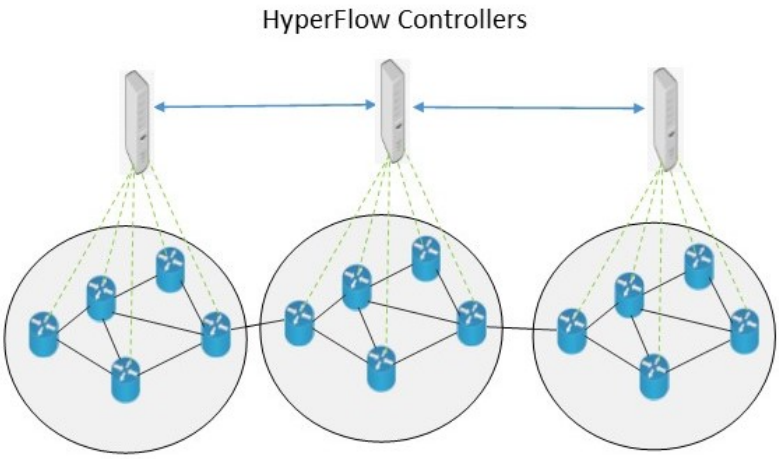

Figure. 7: Simulation setup for Distributed Framework

parameters of the controller like response time, throughput, latency etc. In the experiment, the number of messages is counted starting from the initialization of the network until the end of flow request generation by the hosts. Each host pings to all other hosts in the network, therefore being a new flow request it is forwarded to the controller by the switches. Large number of messages are generated due to the ping request between the hosts. The ARP and ICMP packets towards the controller also increase the number of packets in the total. Firstly, the experiment is performed on the architectural setup shown in figure 4, which is the base setup for the evaluation. The performance of Reactive protocol, Explicit Routing in SDN (ERSDN) protocol and Hybrid protocol is evaluated. The results of implementing three routing protocols in the setup as in figure 4 are depicted in figure 8.

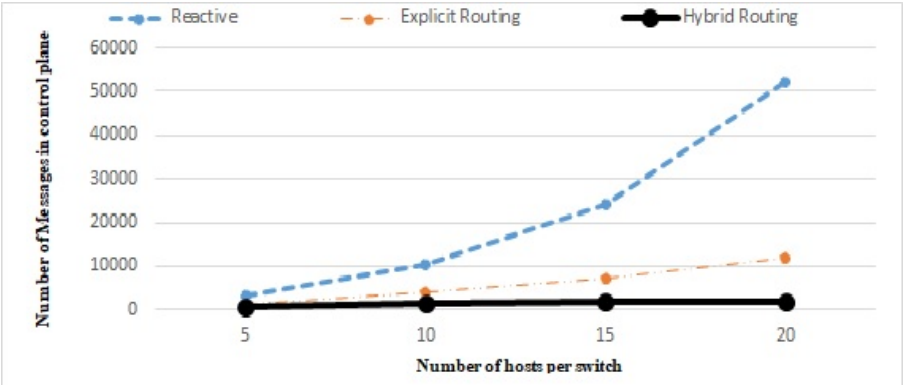

Figure. 8: Total messages generated as per setup in figure 4

Next, the performance of all three routing protocols, i.e. Reactive, ERSDN and Hybrid routing is evaluated in a similar manner on three different frameworks viz. NOX, for single controller setup as in figure 5, HyperFlow for distributed controller setup as in figure 6 and Laman for hierarchical controller setup as in figure 7. The messages are summed up at three distributed controllers for the HyperFlow framework. In Laman, the performance of the Supervisor controller is of concern, but the total number of messages that are received by all controllers are taken into consideration in this test case. The messages that are handled at the level of area controllers are

also considered for performance evaluation. The comparison and discussions are provided based on the results in figures 9,10 and 11.

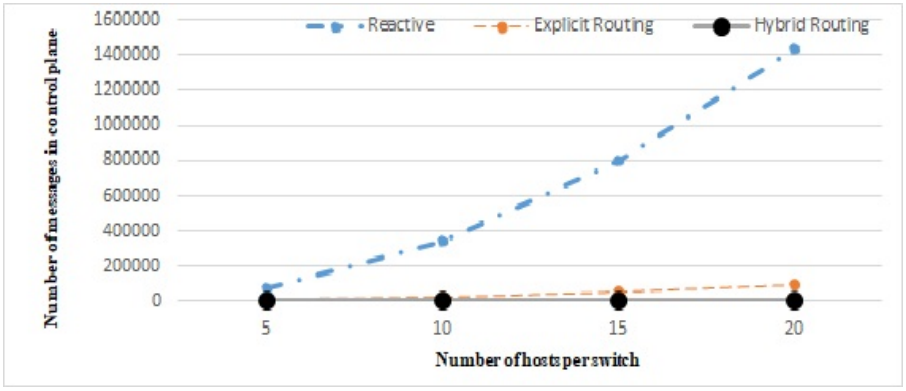

Figure. 9: Total messages generated in a Single Controller setup as in figure 5

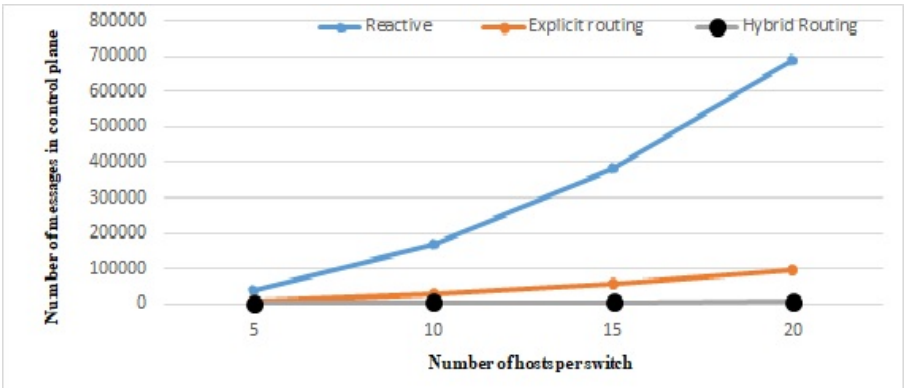

Figure. 10: Total messages generated in Distributed Controller setup as in figure 6

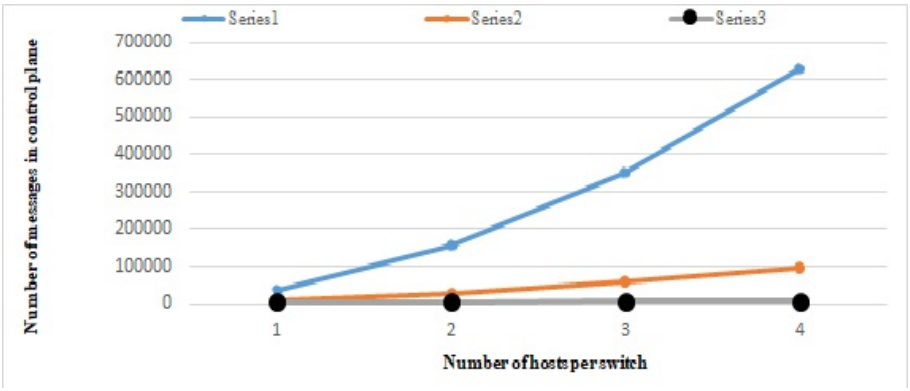

Figure. 11: Total messages generated in Hierarchical Controller setup as in figure 7

The figures 8,9,10 and 11 are exclusively focused on the number of packetin messages that are received by different controllers. When a single controller is used as in figure 8 and figure 9, the

number of messages during initialization is generated at a high pace. Both figures 8 and 9 are based on a single controller setup with a difference in the number of switches attached to the controller. The line chart in both figures follow the same curve, only the number of messages generated are larger in figure 9 due to the large number of hosts attached in comparison to that of the results in figure 8. Distributed controller setup is delivering slightly better results for the number of messages towards the controller in contrast to hierarchical setup. In Laman, i.e. a hierarchical framework, the total number of messages at all controllers are slightly higher as compare to the Hyperflow framework, i.e. distributed framework. The reason for the marginally large number of messages is due to the additional controller in the hierarchical setup. In the test case, the supervisor controller is not overloaded and handled all the packetin request itself for Laman. In case the supervisor controller is overloaded, the contributory controller will be used to forward the packets from the supervisor controller, that further increases the number of packets in the network. Therefore, the results of experiments show that distributed setup, i.e. Hyperflow framework with hybrid approach generates a slightly small number of packetin messages towards the control plane.

Test case 2: Response time is one of the basic parameters used to check the performance of the controller in any network. This experiment is used to check the response time of the controllers. For each packetin message, the controller needs to generate a reply generally in the form of flow rule to a switch. The short response time is always a desirable feature of any controller. The response time of a controller largely depends on the load of the controller. In this test case, no extra load is created at the controller, i.e. it is not in an overloaded condition. The tests are generated on low load conditions for the controller. Only 1 host is added to each switch in the first case, then incremented by 1 host to keep the load as low as possible in the network. The response time of three different frameworks is provided in figures 12,13 and14.

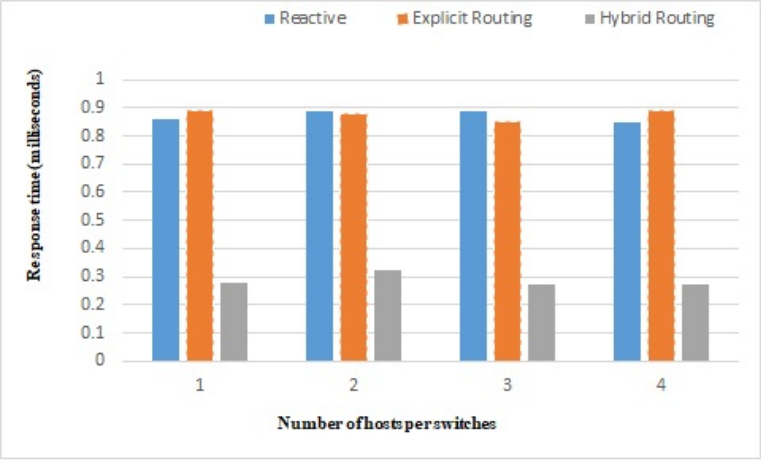

Figure. 12: Response time of a Single Controller at low load conditions

The figures depict that under low load conditions, the response time of reactive and explicit routing is almost similar in all the frameworks considered for evaluation. Addition of 1 host per switch does not affect the response time of the controller. Due to the proactive rule installation, hybrid routing has proven better in all the models. The low load conditions did not provide a valuable result, but the performance can be used to compare the response time for overloaded conditions in the network.

Test case 3: In this test case, the response time of the controller is evaluated similarly to test case 2, but the controllers are overloaded this time. To evaluate the response time, for each packetin message generated, the host waits for the reply of the controller before submitting a

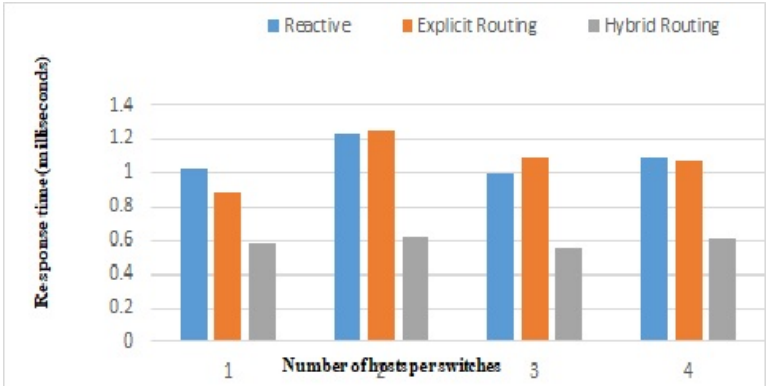

Figure. 13: Response time of Distributed Controller at low load conditions

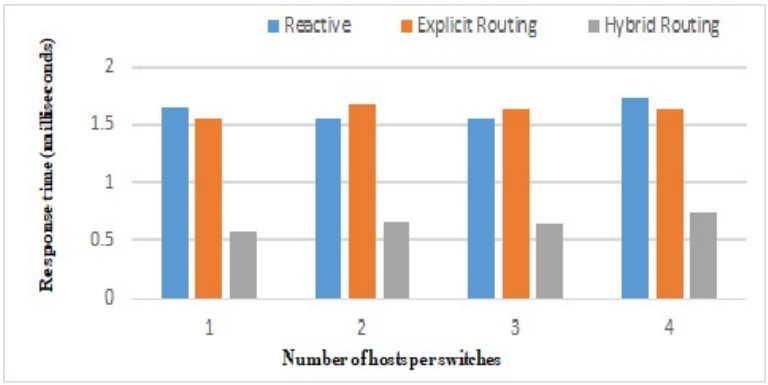

Figure. 14: Response time of Hierarchical Controller at low load conditions

new request for the flow. The test is run for 5 seconds, and the average of 5 tests is taken for the results. The total number of hosts in the network is increased from 50 to 10000 gradually. Figure 15 shows the experimental results on a single controller setup. In hybrid routing to overload the controller, 50:50 ratios are used for proactive and reactive rules. 50 percent flows from the total generated flows are replied at the switch level, and the other 50 percent are forwarded to the controller.

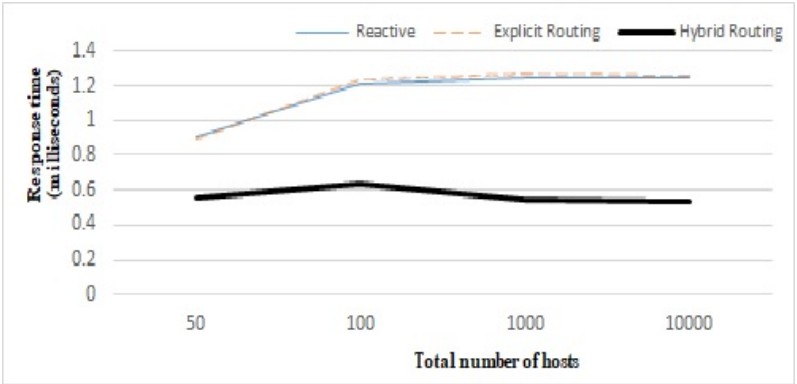

Figure. 15: Single Controller average response time under heavy load

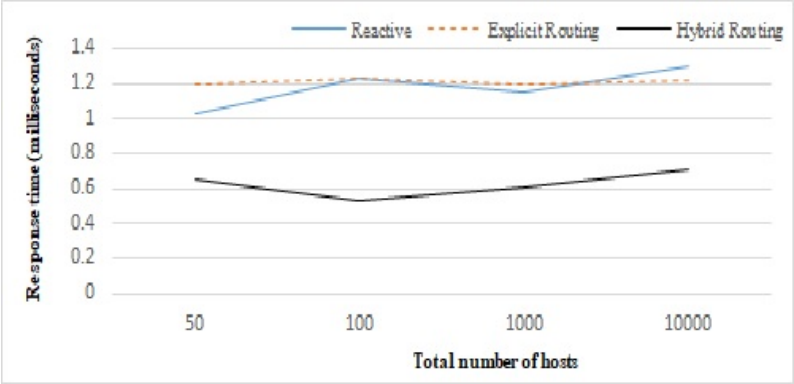

Figure. 16: Distributed Controller average response time under heavy load

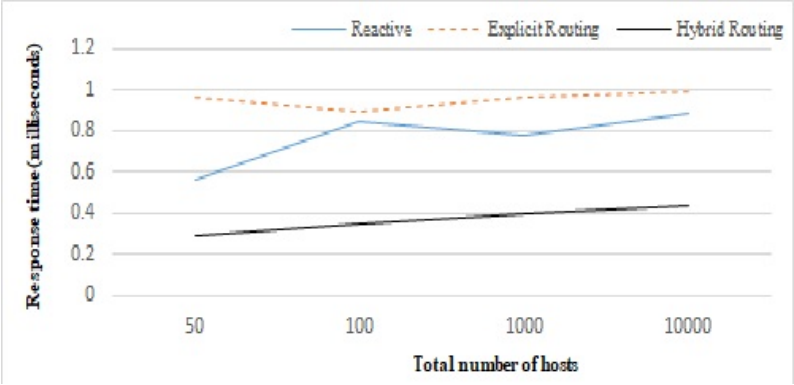

Figure. 17: Hierarchical Controller average response time

Even in case of heavy load at the controller, the results are in favor of hybrid routing protocol in all frameworks evaluated. In the single controller setup, reactive and explicit routing increase load of the controller and its response time thereof. The single overloaded controller produces large response time. Both reactive and explicit routing protocols are providing almost similar response time. In hierarchical controller setup, the reactive routing is even little better as compared to the explicit routing. The distributed controller distributes the load among different controllers. The packet needs to traverse other controllers, and the response time increases as compare to the single controller setup. The hierarchical setup provides the best response time for each of routing method due to availability of a special controller for the routing purpose. The hybrid routing algorithm here also proves its ability to provide the best results as compared to other routing methods. The results provided in test case 2 and test case 3 provides a solution that hybrid routing protocol is better as compared to others if response time is one of the parameters of concern to the network administrators. It provides the best results in all types of network setup and in both underload and overload conditions in the network.

Test case 4: This test case estimates the throughput, i.e. the total number of responses for different requests generated by the hosts in the network. The total responses considered in the results are the sum of responses by all the controller as well as the switches in the network. The responses handled by a single controller are limited; increasing the number of controllers also increase the number of responses. In test case 4 to calculate the throughput, messages are pushed back to back to find the number of messages that the controller can handle at a particular moment. The test is run for 5 seconds, and the average of 10 tests is taken for the results. The total number of hosts in the network is increased from 150 to 15000 gradually. Figure 18, 19

and 20 depict the results on a single controller, distributed controller and hierarchical controller setup. In hybrid routing, the switch responds to about 50 percent of request and forward rest towards the controller. In hierarchical setup, a contributed controller for routing is attached to the supervisory will further increase the number of responses handled.

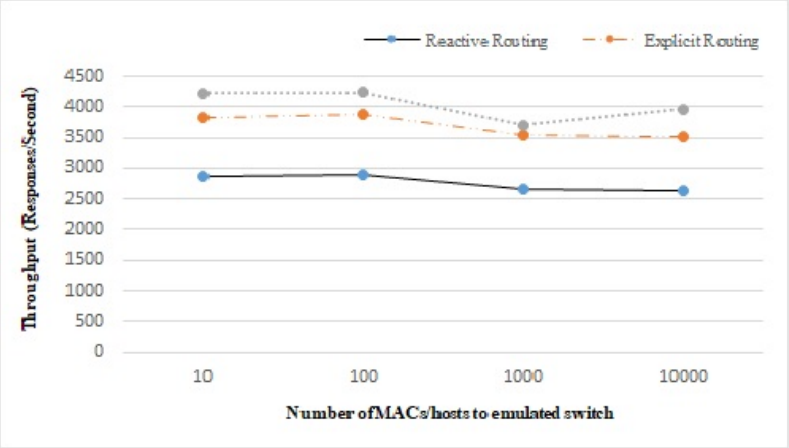

Figure. 18: Average maximum throughput with different number of hosts in a Single Controller setup

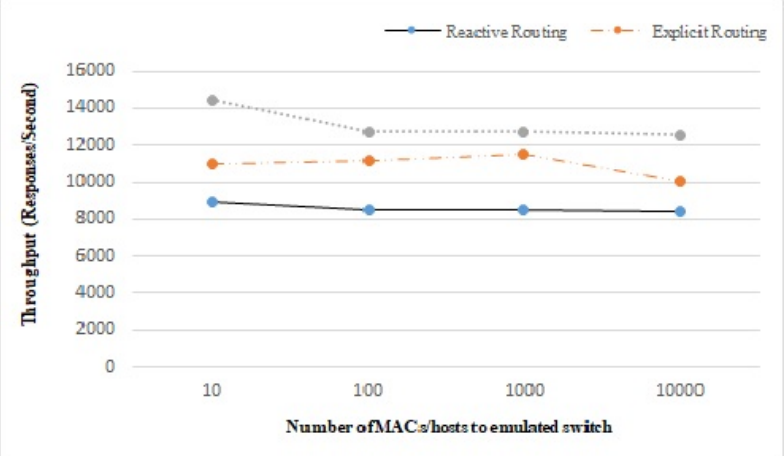

Figure. 19: Average maximum throughput with different number of hosts in Distributed Controller setup

The results in figures 18,19 and 20, encourages the use of hybrid routing as compared to other routing protocols. In these figures, the throughput of the hybrid routing protocol is better than the others. In a single controller setup, the performance of hybrid routing is utmost, but the gap between the number of requests handled by hybrid routing and explicit routing is small. The single controller is not able to handle more requests, and the throughput is restricted by the number of requests handled by the controller. In the distributed controller, the hybrid routing provides greater throughput. The number of messages handled in the network is larger as compared to single controller setup, due to the availability of multiple controllers in distributed and hierarchical setup. The results in figure 19 and 20 show the total number of requests handled by multiple controllers and switches in the network. In hierarchical setup, the throughput provided by hybrid routing is best to all the other routing protocols as in figure 20. The additional controller in the

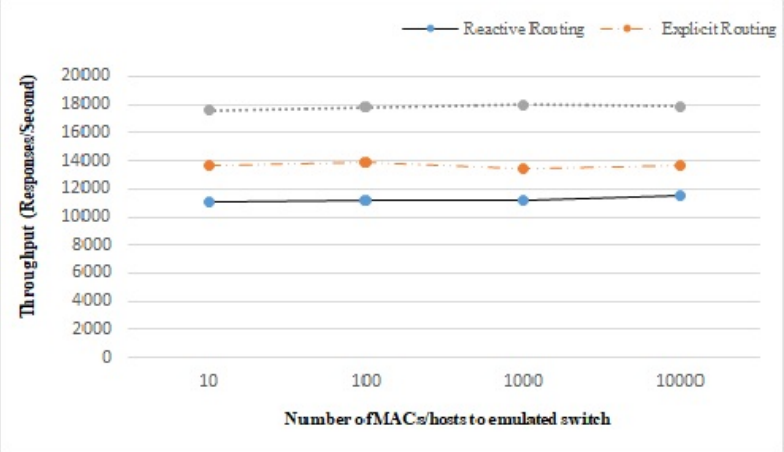

Figure. 20: Average maximum throughput with different number of hosts in Hierarchical Controller setup.

form of supervisor controller and contributory controller implementation in hierarchical setup contributes toward the large throughput in the system.

6. CONCLUSION

In SDN, different solutions are evolved to increase the scalability of the network. A network administrator needs to choose the framework and a routing protocol as per the performance demands of a network. This article provides experimental based solutions to increase the scalability of the network. On the basis of results, it is concluded that when the number of packets at the controller is in consideration, distributed setup along with hybrid routing protocol provides marginally better results in comparison to others. While considering response time, the hybrid routing provides the best performance in all the frameworks considered, both in the underload and overload conditions. The response time of the controller is an important factor that tweaks the network performance. Therefore, this factor should be considered critically before the implementation of any network. In case of throughput of the system, performance evaluation also reports the supremacy of hierarchical framework with the hybrid protocol. This work may be extended to include more number of frameworks and routing protocols. Further, this work may be expanded to incorporate parallelism based solutions along with routing solutions to evaluate the performance of unification.

REFERENCES

Adrianfar K., and Adrianfar K. 2017. A new method for routing in SDN network to increase the delivery rate of sent packets. Intenational Journal of Computer Science and Network Security.17(6) (2017) 266-272.

Berde P.,Gerola M., Hart J. ,Higuchi Y.,Kobayashi M., Koide T., Oconnor B., Radoslavov P. ,Snow W., and Parulkar G.. 2014. ONOS: Towards an Open, Distributed SDN OS. HotSDN, ACM, 99 (2014) 569-576

Casado M.,Freedman M.J., Pettit J., Luo J., McKeown N. and Shenker S. 2007. Ethane: taking control of the enterprise SIGCOMM Comput. Commun. Rev. 37 (2007) 1-12.

cbench https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/ pages /1343657/Cbench+New. (accessed 11 May 2020)

Dixit A., F. Hao, S. Mukherjee,T. Lakshmanand R. Kompella, 2014. Elasticon: An elastic distributed SDN controller. In *Proceeding of 10th ACM/IEEE Symp. Architect. Netw. Commun. Syst, (2014) 17-28.*

Erickson D. The beacon openflow controller. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, in: HotSDN 13, (2013) 1318.*

Farhadi H., Lee H., and Nakao A. Software-defined networking: A survey. Computer Networks, 81 (2015) 7995.

Fu Y., Bi J., Gao K., Chen Z.,Wu J. and Hao B. 2014. Orion: a hybrid hierarchical control plane of software-defined networking for large-scale networks. INetwork Protocols (ICNP), IEEE 22nd International Conference, (2014) 569576.

Gude N.,Koponen T.,Pettit J.,Pfaff B.,Casado M.,McKeown N., and Shenker S. 2008. Nox: towards an operating system for networks, SIGCOMM Comput. Commun. Rev. 38 (2008) 105110.

Karakus M. and Durresi A. 2017. A survey: Control plane scalability issues and approaches in software-defined networking (SDN), Comput. Netw., 112 (2017) 279-293.

Koponen T., Casado M., Gude N.,Stribling J.,Poutievski L.,Zhu M.,JRamanathan R.,Iwata Y.,JInoue H.,JHama T. and YShenker S. 2010. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation, (2010) 1-6.*

Kreutz D., JRamos F. M. V., erissimo P. E.,Rothenberg C. E.,Azodolmolky S. and Uhlig S. 2015. Software-defined networking: A comprehensive survey, Proc. IEEE, 103(1) (2015) 14-76.

Lantz B.,IHeller B. and McKeown N. 2010. A network in a laptop: Rapid prototyping for software-defined networks. In *Proceedings of 9th ACM SIGCOMM Workshop HotNets-IX, (2010) 19:119:6.*

mgen http://cpham.perso.univ-pau.fr/ENSEIGNEMENT/QOS/mgen.html (accessed 13 June 2020)

Nayyer A., Sharma A. K. and Awasthi L. K. 2020. Game Theory-based Routing for Software-Defined Networks, IJRASET, XII (8) (2020) 1007-1014.

Nayyer A., Sharma A. K. and Awasthi L. K. 2018. Issues in software-defined networking. In *Proceedings of of the Second International Conference on Communication, Computing and Networking, Lecture Notes in Networks and Systems, (46), Springer, (2018) 989997 .*

Nayyer A., Sharma A. K. and Awasthi L. K. 2019. Laman: A supervisor controller based scalable framework for software defined networks, Computer Networks, 159 (2019) 125-134.

Open Networking Foundation, OpenFlow Switch Specification, Version

Owens H. and Durresi A. 2014. Explicit routing in software-defined networking (ersdn): addressing controller scalability. In*Proceedings of Network-Based Information Systems (NBiS), 17th International Conference (2014)*

Ryu. [Online]. Available: https://ryu.readthedocs.io/en/latest/getting started.html. (accessed 4 Dec 2019)

Soliman M., Nandy B.,Lambadaris I. and, Ashwood-Smith P. 2012. Source routed forwarding with software defined control, considerations and implications. In*Proceedings the 2012 ACM Conference on CoNEXT Student Workshop, in: CoNEXT Student 12, (2012) 4344.*

Tootoonchian A. and Ganjali Y. 2010. Hyperflow: A distributed control plane for open- flow. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, (2010) 1-6.*

A survey on software-defined networking, IEEE Commun. Surveys Tuts., 17(1) (2015) 27-51.

Yeganeh S. H. and Ganjali Y. 2012. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on Hot topics in software defined networks. (2012) 1924.*

Zhang L., Deng Q. Su Y. and Hu Y. 2017. A box-covering-based routing algorithm for large-scale SDNs, IEEE Access, 5(1), (2017) 4048-4056.

Amit Nayyer received his B.Tech in Computer Science and Engineering (2004) from PTU Jalandhar, India and his M.Tech in Computer Science and Engineering (2009) from National Institute of Technology, Hamirpur, India. He is currently pursuing his Ph.D. in Computer Science under the supervision of Dr. Aman Kumar Sharma and Dr. Lalit Kumar Awasthi at Himachal Pradesh University, Shimla, India. His research interests include Software Defined Networking, Machine Learning and Sensor Networks

Dr. Aman Kumar Sharma is a Professor of Computer Science in Faculty of Physical Sciences at Himachal Pradesh University, Shimla, India. Currently he is Chairman of the Computer Science Department. His research interest includes Cloud Computing, Engineering Quality and Software Engineering . He chaired many National and International Conferences. He has published over 90 papers in International Journals and over 120 papers in conference proceedings.

Dr. Lalit-Kumar-Awasthi is a Professor of Computer Science and Engineering at National Institute of Technology, Hamirpur, India. Currently, he is at the position of Director at Dr. B.R Ambedkar NIT, Jalandhar, India. His research interest includes Check pointing, Mobile computing, Sensor Networks and P2P Networks. He has published over 100 papers in Journals and over 220 papers in conference proceedings.