Software-defined load-balanced data center: design, implementation and performance analysis

Ahmadreza Montazerolghaem

Software-defined Load-balanced Data Center: Design, Implementation and Performance Analysis

Ahmadreza Montazerolghaem*

Department of Computer Engineering, Quchan University of Technology, Quchan, Khorasan Razavi, Iran

Abstract

Data centers are growing densely and providing various services to millions of users through a collection of limited servers. That's why large-scale data center servers are threatened by the overload phenomenon. In this paper, we propose a framework for data centers that are based on Software-defined networking (SDN) technology and, taking advantage of this technology, seek to balance the load between servers and prevent overloading on a given server. In addition, this framework provides the required services in a fast time and with low computational complexity. The proposed framework is implemented in a real testbed, and a wide variety of experimentations are carried out in comprehensive scenarios to evaluate its performance. Furthermore, the framework is evaluated with four data centers including Three-layer, Fat-Tree, BCube, and Dcell data centers. In the testbed, Open vSwitch v2.4.1 and Floodlight v1.2 are used to implement switches and OpenFlow controllers. The results show that in all four SDN-based architectures, the load balances between the servers is well maintained, and a significant improvement has been made in parameters such as throughput, delay, and resource consumption.

Keywords: Software-defined networking (SDN), Data center design, Load balancing, PID controller, Throughput, Delay

*Corresponding author
Email address: ar.montazer@qiet.ac.ir (Ahmadreza Montazerolghaem)

1. Introduction

Recently, introduced Software Defined Networking (SDN) is a major trend in the telecommunication industry that can enhance the data center networks [1]. The Software defined data center integrates the software defined concepts into data center technology. Data center networks are the backbone of the Internet services. SDN significantly simplifies and improves this networks management [2]. In SDN, the control and data planes are separated and logically centralized using the OpenFlow protocol[3, 4, 5]. It can meet the requirements of the data center communication network [6]. In this case, each network switch simply forwards the traffic and enforces policy according to instructions received from the controller. This makes the network programmable in a way that promises to be more flexible, scalable, and secure than that of traditional networks [7]. SDN has attracted attention for developing different data center applications which require a higher degree of network awareness [8, 9, 10, 11].

Data centers are being transformed into service-oriented networks as fully connectivity oriented networks. This new paradigm changes the framework and architecture of networks and changes their design and construction. In the new paradigm, instead of layering switches and routers, it is attempted to reduce the complexity of their components and create new capacities by homogenizing, simplifying configurations and developing programmable capabilities in data centers. The best way to quickly and easily achieve this goal is to achieve a specific operating model with more coordinated services, thereby guiding us towards the greatest change and flexibility available, Software Defined Networking. Deploying SDN and other new approaches are the biggest challenges in next-generation data center architecture that wants to move networks towards being service-oriented. Traditionally data center scaling was done by building a subnet and adding a new domain. In contrast, SD-DC[1] method controls the resources according to demand through a centralized view. This approach prevents the

[1]Software-Defined Data Center

saturation of server resources, overload, and loss of service quality.

1.1. Related work

Darabesh et al. [1] introduce an experimental simulation (not implementation) framework to provide a virtualized testbed environment for data center based on SDN. This work is built based on the Mininet simulator, where its core components, the host, the switch and the controller, are set. [12] describes a technique based on Domain Name System (DNS) for providing distributed server load balancing over resources across multiple data centers. [13] explores micro-level power management in Software defined data center and assess the feasibility of this new power management paradigm by characterizing the resource and power impact of various management operations. [10] proposes a novel algorithm for use in an intelligent, user-aware SDDC which performs runtime analysis of user storage system activity in a manner that has a minimal impact on performance and provides accurate estimations of future user activity. [14] implements the SDN based routing in CamCube data center topology that is able to outperform the traditional OSPF protocol in the CamCube infrastructure network. [15] designs new physical layer control instances and combine SDN control with encoding in a data center. Yifei Xu et al. [16] present a management model for data center networks managing information is three levels: network managers, regional controllers, and tenants. The authors in [17] design a novel network fabric and shortest path algorithm for data center networks based on SDN. New SDN-based TCP slow start mechanism for data center is proposed in [18]. In this method, the available bandwidth can be obtained by the SDN controller in every time slot. Then the slow start threshold and the congestion window can be tuned suitably. An OpenFlow and SDN-based QoS traffic method to achieve effective usage of data center network resources is proposed by [19]. In [20], authors attempt to enhance the QoS of the multi-tenanted data center network clouds. E^3MC [21] is a method to improve data center energy efficiency by leveraging SDN. In this regard, the energy optimization for both forwarding and control plane are investigated by dynamic control mapping. [22]

introduces the dynamic timeout for SDN-based data centers, which can assign appropriate timeout to various flows under their characteristics. In [23], Hwang et al. introduce an approach for fast failover of both the control and data plane in the data centers based on SDN. An dynamic load management method based on SDN technology for optimizing data center link utilization by flow priority is proposed in [24].

While much prior research has suggested the potential benefits of applying SDN in computer networks in order to facilitate network management, there has only been few studies about the practical approaches of applying SDN in data center, practically. Additionally, the whole concept of SD-DC is in its infancy and standardization efforts in terms of framework, protocols, applications, and assessment tools are still underway. Also, as discussed earlier, the proposed ideas and related work are mostly preliminary proposals about softwarization of WSNs; or they focus on security and big data challenges of IoT. Here, we pay particular attention to management of resources as well as QoS of data centers. To the best of our knowledge, there are no studies concerning a comprehensive approach for combining data center server load balancing with QoS mechanism with operational view. Therefore, the exploration of such an approach is timely and crucial, especially considering the rapid development of data center applications and the emergence of SDN. In this paper, we propose a SDN-based architecture for data center applications, so that both the path and server selection can be managed together to improve QoS for the users, and to balance traffic between servers simultaneously.

According to the above studies, there are no comprehensive studies on an SDN-based data center load balancing. So, in this paper, we propose an SDN-based architecture for data center to balance the load between servers and prevent overloading. Also, the proposed framework is implemented in a real testbed, and a wide variety of experimentations are carried out under various scenarios.

1.2. Motivation

Why do we need a scalable DC communication infrastructure? Knowing that the DC infrastructure is a key service in next generation network, it is vital to design and develop a high performance and scalable communication infrastructure with quality of service support. Recently, the DC has been utilized for supporting new services. In addition, the DC is rapidly developing and spreading in different geographical locations. Moreover, for the future large-scale DC, it will be necessary to collect and transfer data. Therefore, this will create a huge traffic load which should be efficiently routed and balanced across the network and servers. To summarize, with the overall modern network road map, both the number of users and data will increase dramatically. Consequently, a huge amount of data will go through the servers, which imposes a great challenge on the scalability of the traditional DC communication network. Over the next few years not only will DC servers face overload, but also the communication network will fail to satisfy the QoS of traffic. So this article addresses this challenge and proposes a scalable SDN-based DC framework.

1.3. Contributions

Our study mainly aims at managing the traffic through the concept of the SDN. In this regard, by applying the SDN in the data center, we provide a resource and QoS conscious framework. In this framework, we seek to choose the best path among the existing paths for each of the data center traffic classes in a way that the load balance of data center servers is established and the QoS of the traffic class is satisfied. To this end, we design a modular controller that uses PID system. In this regard, we propose the system for resource managment based on PID. In other words, PID is used to decide how to load balances between servers.

So, the main innovation is *the architectural design of a modular controller based on SDN technology according to Proportional-Integral-Derivative (PID), Least Connection and Least Response Time algorithms to achieve a scalable*

data centers with a higher quality of service. In this regard, the main contributions of this paper can be summarized as follows:

- **Theoretical aspect**

- Designing a novel SDN-based control and management framework for the data center (for avoiding the overload occurrence on data center servers together with increasing the QoS),
- Designing a proactive heuristic load balancing method based on PID system,
- Designing modules of the proposed framework,

- **Implementation aspect**

- Designing and preparing a real test platform to evaluate the performance of the proposed framework (is applied to the well-known topologies).
- Designing comprehensive scenarios to evaluate the efficiency of the proposed framework.

1.4. Organization

The rest of this article is organized as follows: in Section 2, we propose a data center framework as well as an SDN controller for proper distribution of loads between servers. In Section 3 we discuss the details of the implementation of the proposed schemes and their evaluation. Section 4 presents the conclusion and further work.

2. Designing an SDN based enterprise data center framework

The framework consists of two parts: infrastructure and control. We first present the overall architecture of the four important data centers and then propose the controller details of the framework.

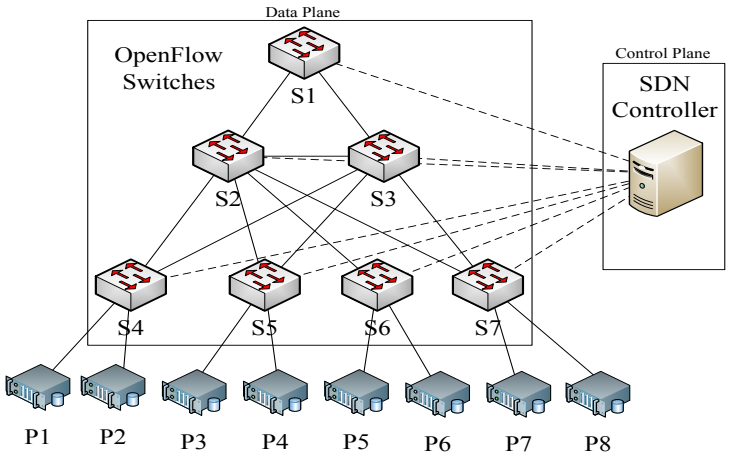


Figure 1: SDN-based three-layer data center architecture

2.1. Three-layer architecture

The three-layer architecture based on SDN technology is illustrated in Fig.1. The architecture consists of three layers of switches which establish the communication between the servers. These switches use OpenFlow technology and are controlled by an SDN controller. Flows between the servers are also managed by commands sent from the controller to the switches. The traditional structure of this well-known data center architecture is illustrated in Fig.2.

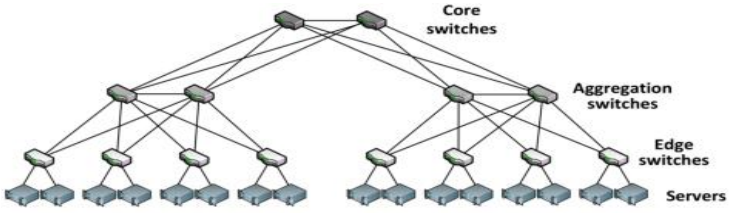


Figure 2: Traditional three-layer data center architecture

As can be seen in this architecture, there are three layers of edge, aggregation, and core, each layer having a desired number of switches with different capacities. For example, in this example, there are 8 edge switches, 4 aggregation switches, and 2 core switches.

2.2. Fat-Tree architecture

Fat-Tree architecture is illustrated in Fig.3. The topology of this architecture is tree-like.

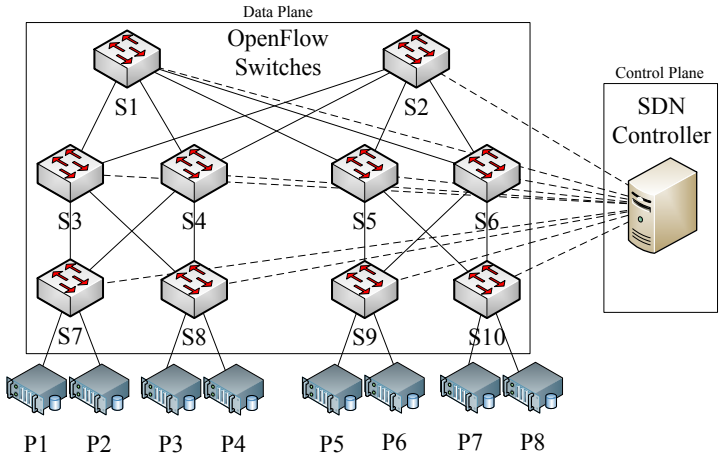


Figure 3: SDN-based data center tree architecture

The traditional architecture of this model of the data centers is shown in Fig.4. As can be seen, the data center consists of 4 pods, each of them consisting of 8 aggregation and edge switches. A total of 4 core switches establish the communication between the pods.

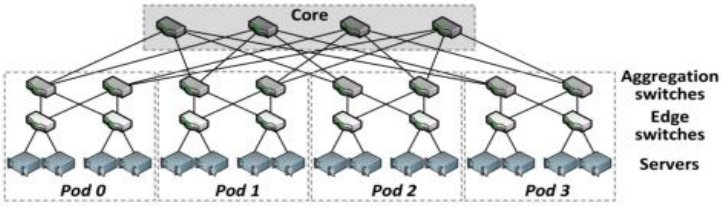


Figure 4: Traditional data center tree architecture

2.3. BCube architecture

BCube architecture is illustrated in Fig.5. As can be seen, the two levels of switch establish communication between the servers. Communications between all four adjacent servers are provided via low-level switches and communications between non-adjacent servers are provided via high-level switches.

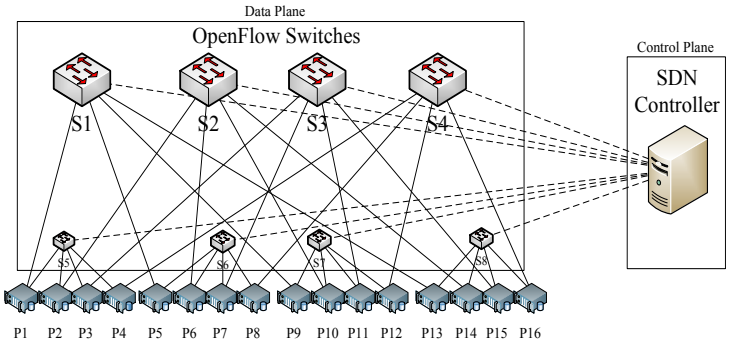


Figure 5: SDN-based BCube data center architecture

The traditional architecture of this type of data center is shown in Fig.6. The architecture has two levels. At the lower level, 4 four-port switches are connected to 16 adjacent servers. At the top level, 4 four-port switches provide communication between non-adjacent servers.

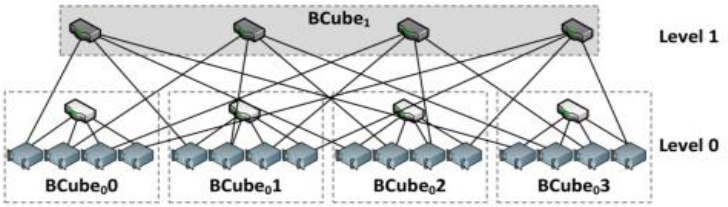


Figure 6: Traditional BCube data center architecture

2.4. Dcell architecture

Dcell architecture is illustrated in Fig.7. The framework of this architecture consists of several cells. Each cell contains one switch and four servers. Communications between servers of different cells are provided through switches. The SDN controller also controls the flows and switches.

The traditional architecture of a Dcell data center is also shown in Fig.8.

In the traditional architecture, the communication between cells is provided through traditional switches (not OpenFlow), which increases the complexity of the infrastructure level, hence makes the architecture unscalable. In traditional networks, once the flow management (forwarding policy) has been defined, the

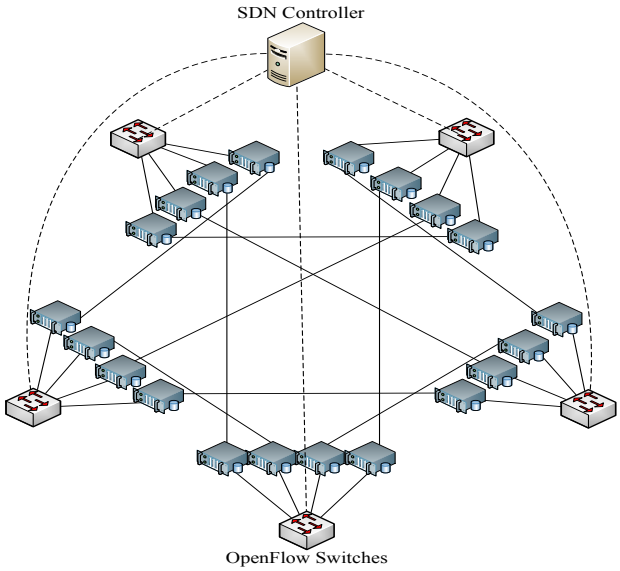


Figure 7: SDN-based Dcell data center architecture

only way to make an adjustment to the policy is via changes to the configuration of the network equipment. This has proven restrictive for network operators who are keen to scale their networks in response to changing traffic demands, increasing the use of mobile devices.

2.5. Designing the SDN controller

In this section, we present the proposed architecture of the SDN controller for balancing server loads in the aforementioned schemes. As shown in Fig.9, there are two layers of infrastructure and control. In the infrastructure layer, there are servers, switches, and computers. The data flow is distributed through the switches between the servers. In this regard, the SDN controller plays a role in the control layer. The *Network Statistics* module collects traffic information from switches. Server load information is also collected by the *Servers Load Monitoring* module. In the SDN controller, *Proportional-Integral-Derivative (PID)* controller is used to decide how to load balances between servers. In this regard, the PID controller seeks to regulate the servers load within the target load.

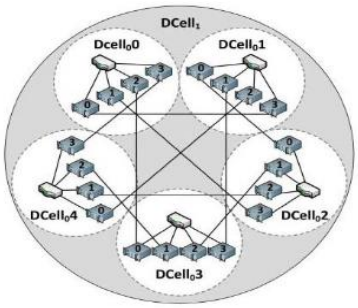


Figure 8: Traditional Dcell data center architecture

In order to address this issue, we propose here a control theory approach that dynamically (re)adjusts load threshold depending on the system behavior and the desired utilization levels, allowing the SDN controller to learn over time depending on current system behavior. We evaluate a PID controller that readjusts load threshold, i.e., change the level of load that the system is willing to face, to achieve a desired level of resource utilization for each dimension (CPU, memory) in a safer way. r denotes the desired load level, y the measured level, e the difference between the current load level and the target one, and finally u is the increment or decrement in the actual load threshold needed to achieve the target utilization. The u value at time t can be obtained from Equation (1):

$$u(t) = K_P e(t) + K_I \int_0^t e(t)dt + K_D \frac{de(t)}{dt} \tag{1}$$

In this equation, increasing the P component (K_P) leads to faster response but also to overshooting and oscillation problems. Increasing I component (K_I) reduces stationary errors but at the expense of larger oscillations. Finally, the D component (K_D) reduces the oscillations but may lead to slower response. Only setting these values turned out to be insufficient to achieve a good performance (stable and fast response nal value ($P + I + D$)). This filter keeps threshold load levels between certain values, i.e., between 0.2 and 0.8 instead of the [0,1] range. Limiting the spectrum of feasible values for the load threshold reduces fluctuations caused by fast switching between accepting too many and too few services.

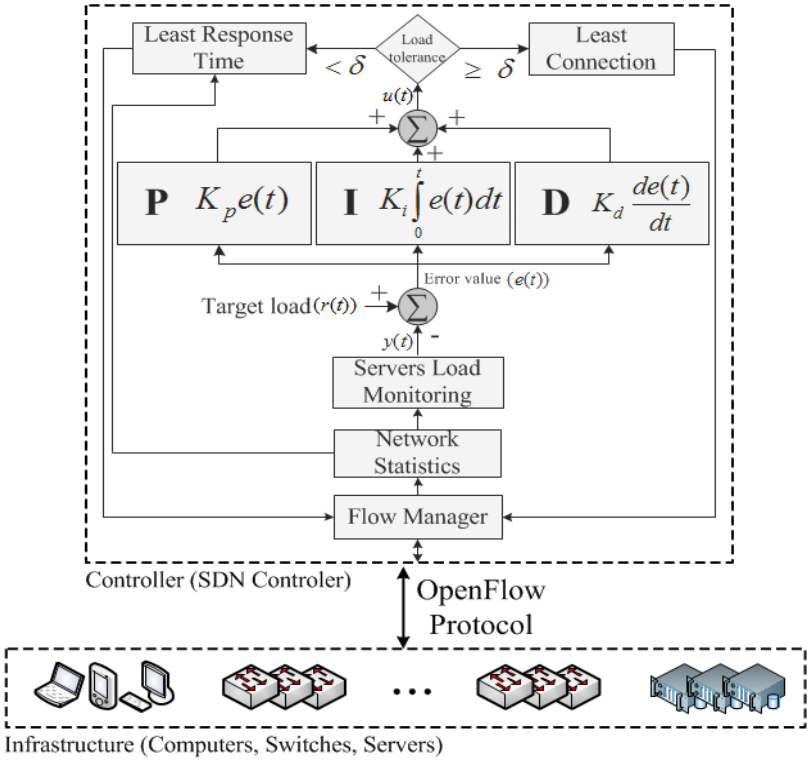


Figure 9: The proposed architecture of the SDN controller

Finally, if the server load tolerance is lower than the threshold (δ), the server with the lowest response time is selected, and otherwise, the server with the lowest connection is selected.

Least response time method uses the response information from the *Network Statistics* module to determine the server that is responding fastest at a particular time. In least connection method the current request goes to the server that is servicing the least number of active sessions at the current time. The reason is that if the load tolerance is high and the servers' load is more imbalanced, the least connection method can better detect less efficient servers and balance the servers by injecting load into them, in comparison to the least response time method. If the server load tolerance is low, the least response time method can better and faster keep the server load status balanced.

In the end, depending on the selected server, the *Flow Manager* module adjusts the appropriate path of flows to that server by installing the appropriate rolls on the OpenFlow switches. Algorithm 1 below shows the details of the proposed load balancing approach.

Algorithm 1: Load Balancing

Result: Select the most appropriate server

Data: r denotes the desired load level;

y the measured load level of server;

e the difference between the current load level and the target one;

u is the increment or decrement in the actual load threshold needed to achieve the target utilization.

while *as long as the traffic is flowing* **do**

- The *Flow Manager* module receives the incoming packets like Packet-In message;
- The *Network Statistics* module extracts the necessary information such as time from the messages;
- Response time is given to the *Least Response Time* module;
- The *Servers Load Monitoring* module monitors and records y;
- The *PID Controller* module calculates u with regard to r;

if $u \geq \delta$ **then**

Least Connection method runs;

The server with the least load is selected;

else

Least Response Time method runs;

The server with the least response time is selected;

end

end

The *Flow Manager* module is in charge of sending LLDP (Link Layer Discovery Protocol) packets to all the connected switches through Packet Out messages. These messages instruct the switches to send LLDP packets to all ports. Once a switch receives the Packet-Out message, the LLDP packets are sent out among all the ports. If the neighbor device is an OpenFlow switch, it will perform a flow lookup. Since the switch does not have a flow entry for this LLDP message, it will send this packet to the controller by means of a Packet-In message. When the controller receives the Packet-In, it analyses the packet and creates a connection in its discovery table for the two switches. All remaining switches in the network will similarly send a packet into the controller, which would create a complete network topology. LLDP messages are periodically exchanged and events are brought to the controller when links go up/down, or new links are added/removed. Information on switches and links are maintained in the *Network Statistics* module. Servers load information and load level of servers (y) is also recorded by the *Servers Load Monitoring* module. Load monitoring is defined in one place for the entire network - in the control plane module. It is easy to scale by replicating the control plane. As mentioned, PID module is used to decide how to load balances between servers. So, the PID controller seeks to regulate the servers load (y) with the target load (r). The PID controller output is u. u is the increment or decrement in the actual load threshold needed to achieve the target utilization. The u value at time t can be obtained from Equation 1. Finally, if the server load tolerance (u) is upper than the threshold (δ), the server with the least connection is selected, and otherwise, the server with the least response time is selected. Least connection load balancing algorithm selects the server with the fewest active connections. Neither round robin or random take the current server load into consideration when distributing messages. The least connection algorithm considers the current server load. In return, the least response time algorithm uses the response data from a server to determine the server that is responding fastest at a deadline.

3. Testing and evaluating the proposed schemes

In this section, we first describe the implementation details. We then evaluate the performance and present the results of the proposed approach in several of the following subsections.

In the testbed, we employ Open vSwitch v2.4.1 and Floodlight v1.2 to implement the OpenFlow switch and controller and modify them as described in the previous section. Floodlight is a Java-based controller. The PID controller is also coded and tuned as a module in Floodlight in Java. Open vSwitch is also a virtual and software switch that supports the OpenFlow protocol. We use SIPp to inject traffic. Oprofile software is also used to observe consumables. If we need to inject background traffic between the servers, we use iperf to send packets at a fixed rate. We run each experiment three times and reported the mean as a result.

3.1. Implementation testbed

The testbed provided in the IP-PBX Laboratory of Ferdowsi University of Mashhad[2] for the implementation of the proposed architectures is given in Fig.10.

The list of equipment needed to implement the proposed architectures is as follows:

- HP G9 DL580 server
- Open vSwitch v2.4.1 Switch
- Floodlight v1.2 controller
- SIPp for traffic injection

[2]http://voip-lab.um.ac.ir/index.php?lang=en

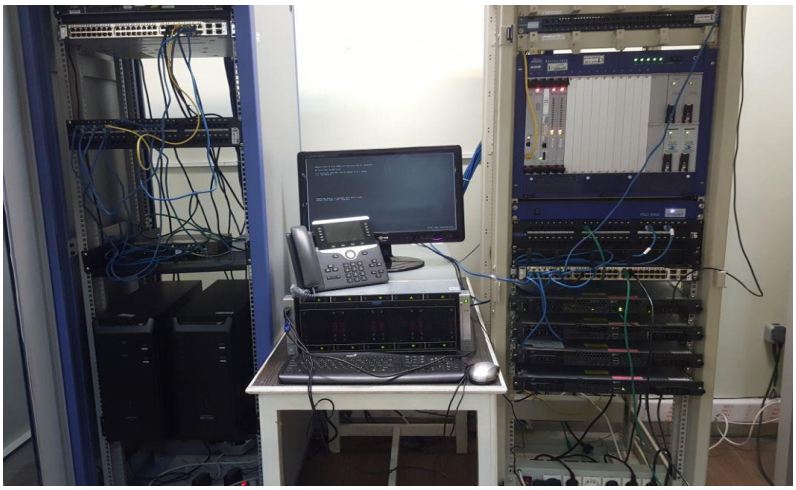


Figure 10: Implementation platform in the IP-PBX type approval laboratory of Ferdowsi University of Mashhad

3.2. Performance evaluation and experimental results

In the following, we present experimental results for the Three-layer architecture[3]. According to Fig.1, we use 7 OpenFlow switches and 8 servers to test the architecture.

3.2.1. Experiment 1: Fixed load

The first experiment involves two scenarios with different background traffic. In Scenario 1, each server's background traffic is equal to 500 bps. In Scenario 2, the background traffic for Servers 1 to 4 (P1 to P4) is 1000, and for Servers 5 to 8 (5P to P8) is 500 bps. A fixed offered-load of 400 seconds at a rate of 1500 requests per second (rps) is also injected into the system by a traffic generator. Fig. 10 shows the performance of the servers.

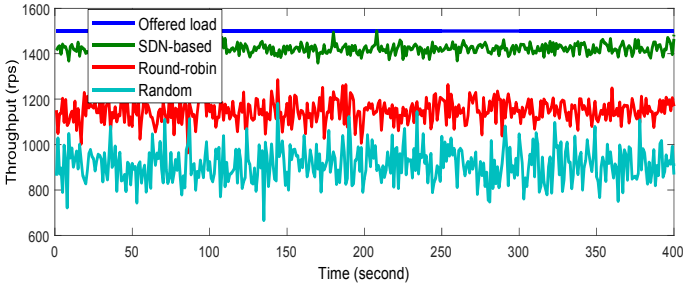
Three different methods are used to select a server: Round-robin, Random, and SDN-based (the proposed approach). Servers' throughput (the number of serviced requests by servers per time), their average response time (the time between sending a request from the computer and receiving the acknowledgment

[3]The results of the other architectures are given in the appendix.

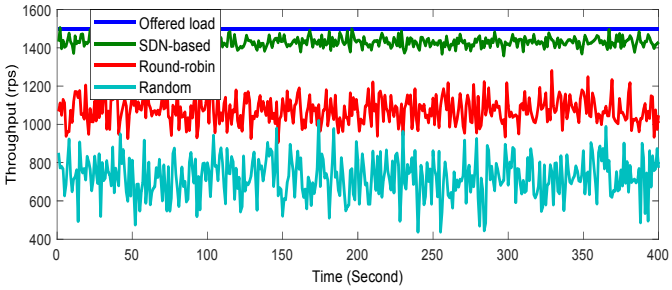
by the server), and the resources consumed by the servers are among the evaluation criteria. The goal is to achieve the maximum throughput and minimum response time without overloading and with respect to resources.

The proposed method has achieved better results than the Round-robin and Random methods in both scenarios (Fig. 10). In addition, the proposed method results are similar in both scenarios, however, the results of Random-Robin and Random in Scenario 2 are worse than in Scenario 1. The reason is that the different background traffic of servers in Scenario 2 has made the blind load distribution of the two methods worse over time. As can be seen from the comparison of Fig. 11a to 11b, the SDN-based method in both scenarios is able to achieve near offered-load throughput, as it can have a good estimate of servers' load using both, response time and the number of connections. Not only consuming the resources in Round-Robin and Random methods is more than SDN-based method, but also their average response time is longer. The servers' resource utilization rate in the SDN-based approach is approximately equal, indicating a conscious and fair distribution of the load of this method (Fig. 10e to 10p).

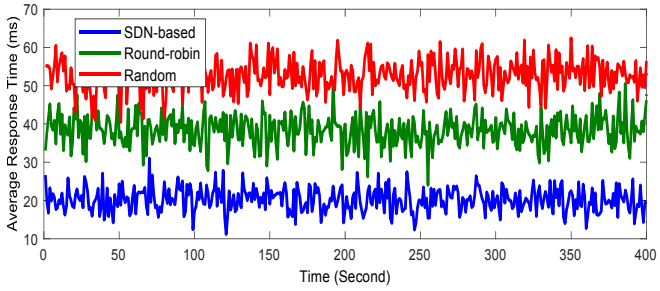
In Round-Robin and Random methods unequal distribution of load over *time* causes unanswered messages in the server queue. This is because the redundant retransmissions and manipulation of the timers increase CPU and memory occupation and worsen server overload. In other words, failure to perform one task in due time affects subsequent tasks, causing an overload of CPU and memory. The same is true for subsequent tasks. As a result, the server processor and memory are always involved in a large number of previous or retransmission messages. All of this is due to the lack of fair distribution. Besides that, the flow of *new requests* eventuated to overflowing of the queue and losing the packages. In this regard, an SDN-based framework is proposed which uses the global view to distribute fairly and based on *servers capacity*. On the contrary, in the random and round-robin approaches, the capacity of the servers is neglected. This over time causes the queues to become full and also overloaded.



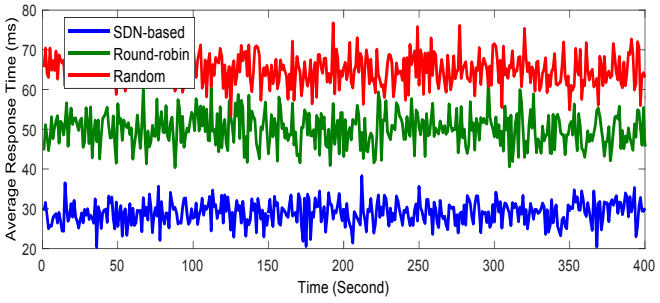
(a) Servers' throughput in Scenario 1



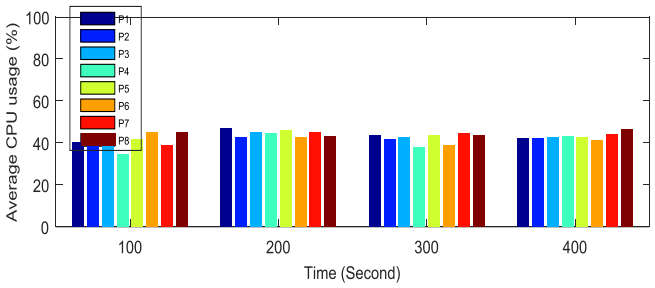
(b) Servers' throughput in Scenario 2



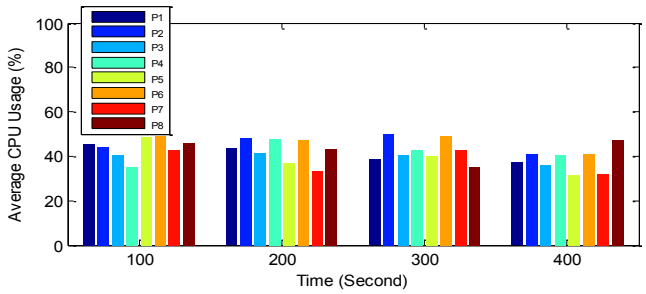
(c) Servers average response time in Scenario 1



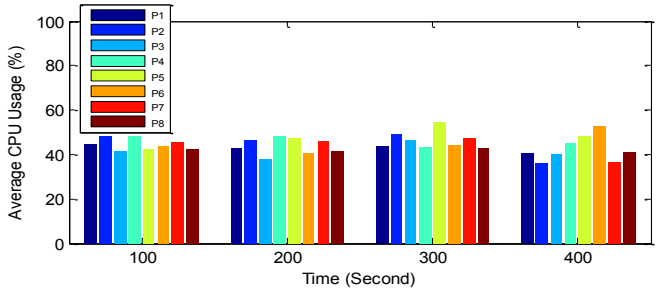
(d) Servers average response time in Scenario 2



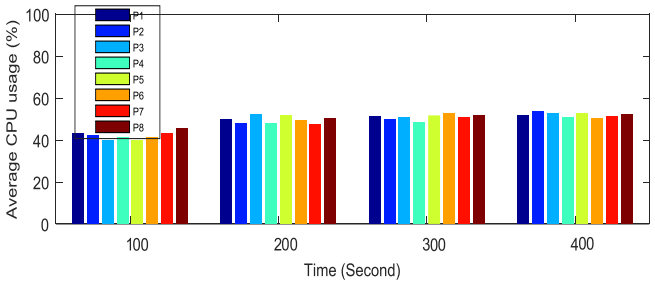
(e) Servers average CPU usage in Scenario 1 (SDN-based)



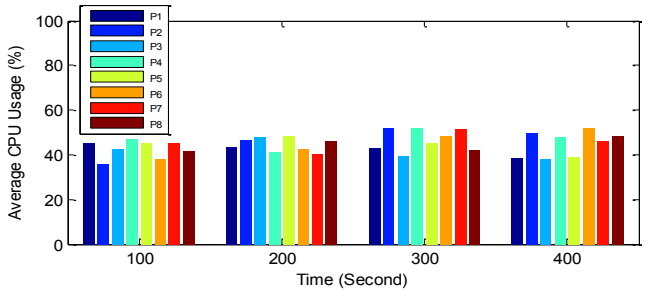
(f) Server average CPU usage in Scenario 1 (Round-robin)



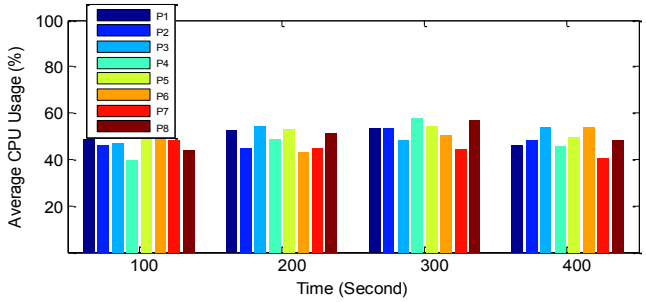
(g) Server average CPU usage in Scenario 1 (Random)



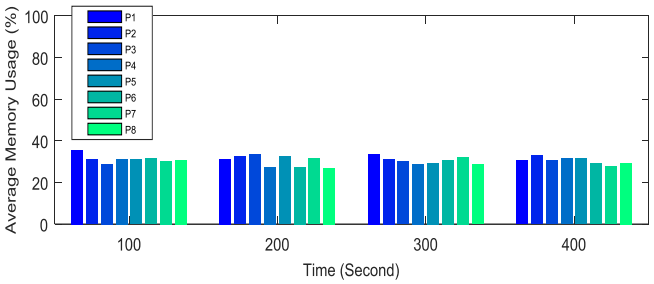
(h) Server average CPU usage in Scenario 2 (SDN-based)



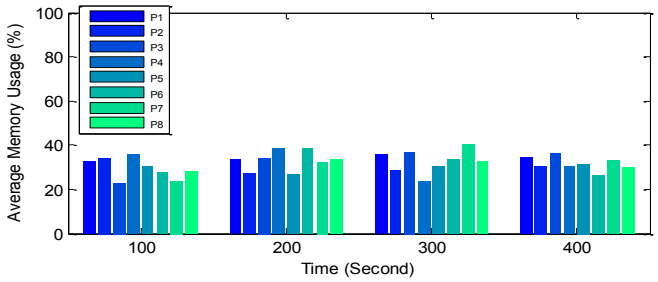
(i) Server average CPU usage in Scenario 2 (Round-robin)



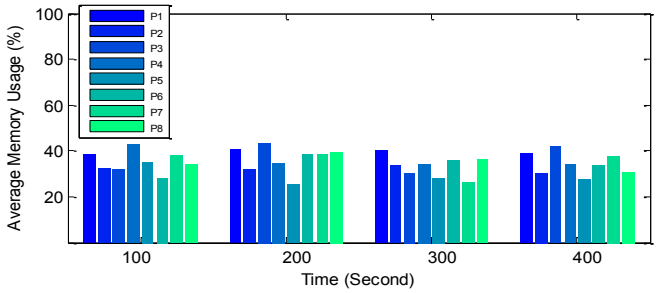
(j) Server average CPU usage in Scenario 2 (Random)



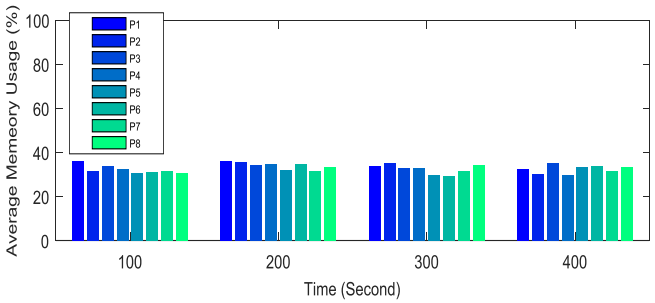
(k) Server average memory utilization in Scenario 1 (SDN-based)



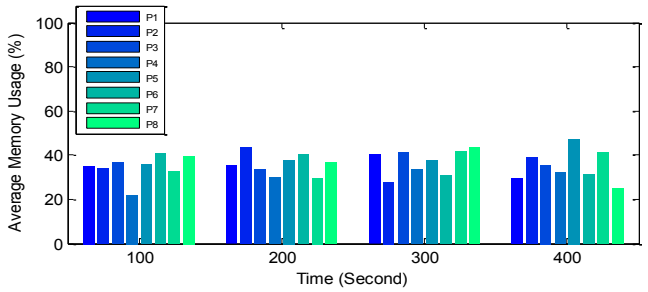
(l) Server average memory utilization in Scenario 1 (Round-robin)



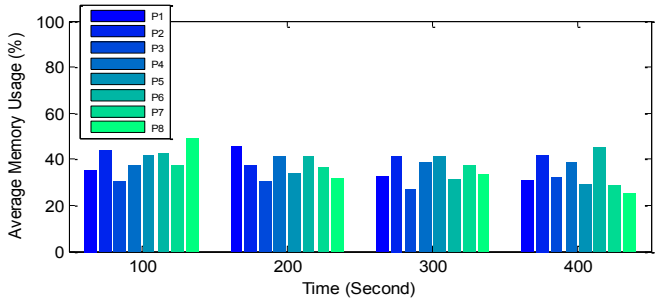
(m) Server average memory utilization in Scenario 1 (Random)



(n) Server average memory utilization in Scenario 2 (SDN-based)



(o) Server average memory utilization in Scenario 2 (Round-robin)



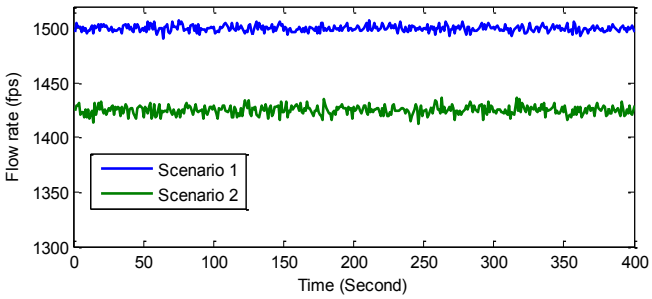
(p) Server average memory utilization in Scenario 2 (Random)

Figure 10: Comparison of the data center servers' performance in two scenarios

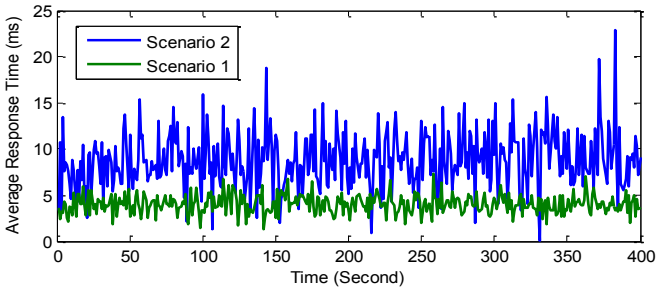
Fig. 11 shows the performance of the proposed controller. The controller throughput is denoted by the number of serviced flows per time, and the average controller response time is denoted by the time between sending a Packet-In message from the switch to receiving the Flow-Mod by the controller.

As shown in Fig. 11a and 11b, the controller performance is scenario-independent, with an average throughput of approximately 1450 fps and an average response time of approximately 7 ms. This indicates that the controller has been able to achieve high throughput and low delay. However, its modules do not overload resources. Consuming fewer resources by the controller than the servers is because the server is responsible for establishing and terminating all requests, while the controller is responsible for managing a limited number of switches. This is also deduced from Fig. 12. This figure shows that the number of packets processed per server is approximately 7 times that of the controller. As you know, at least 7 messages are involved in establishing a connection, while the process of managing the rules on the switches by the controller is initiated with the Packet-In message and terminated with the Flow-Mod message. So the overload on the controller is much less likely than on servers. Also, servers will not be exposed to overload by the fair distribution of the load by the controller.

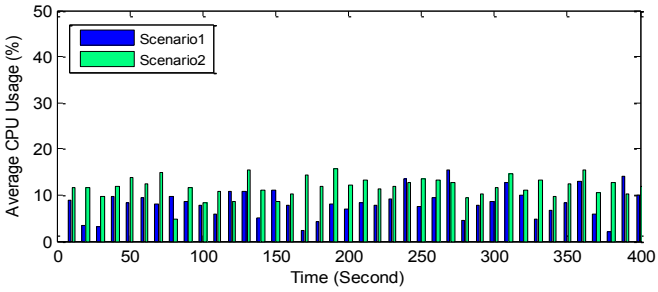
Other reasons the servers may be in trouble are the sudden failure of network components and the sudden loss of capacity. This failure may be an imposed corrupted server load on other servers. In order to test this situation, let P1 to P4 fail at second 80 and recover at second 160, in the first scenario. Similarly, let P1 to P4 fail at second 240 and recover at second 320, in the second scenario. Fig. 13 shows the performance under these conditions. In Scenario 1, the controller is able to transfer the entire load, from second 80 to 160, to P5 to P8, indicating the speed of the controller action, which, despite the sudden failure of P1 to P4, still maintained the entire system throughput near to offered-load. The consumption of resources P5 to P8 has also increased during this period. The controller resources consumption has also increased in this period. Under normal circumstances, server resource usage is almost equal and controller resource consumption is very low. In Scenario 2, the same process is repeated in



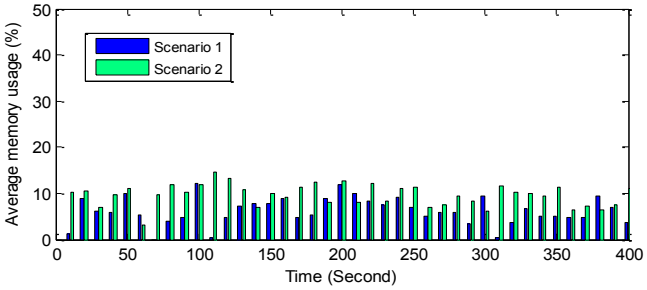
(a) Controller throughput



(b) The average response time of the controller



(c) The average controller CPU usage



(d) The average controller memory usage

Figure 11: Comparison of the controller performance in two scenarios

seconds 240 to 320, except that the load to P1 to P4 is not equal to P5 to P8, yet the controller is able to transfer the entire load to the set P5 to P8 in second 240, and redistribute the load across all servers in second 320.

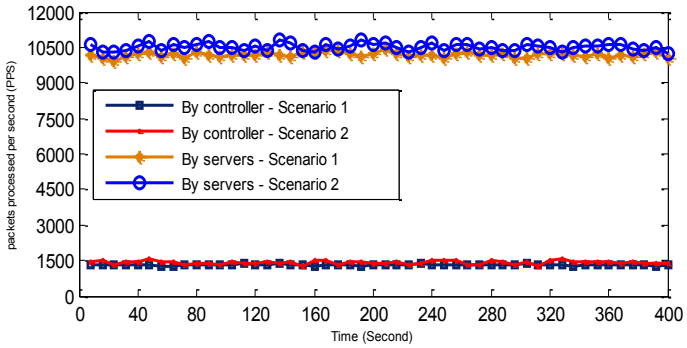
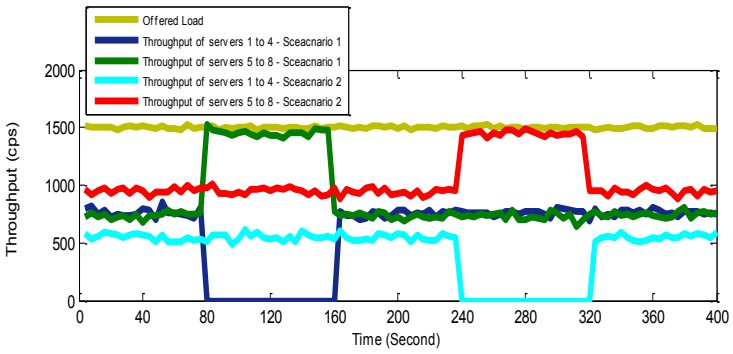


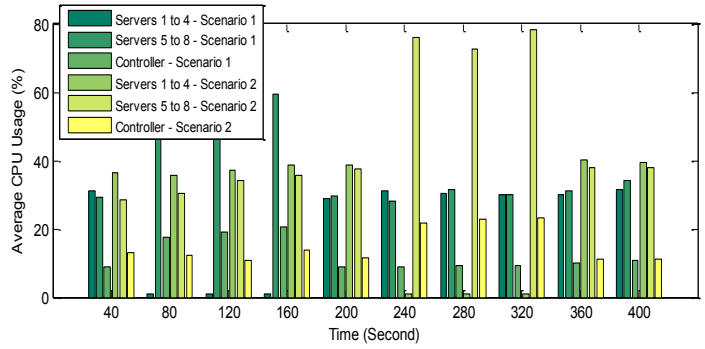
Figure 12: Number of packets processed per second

3.2.2. Experiment 2: Variable load

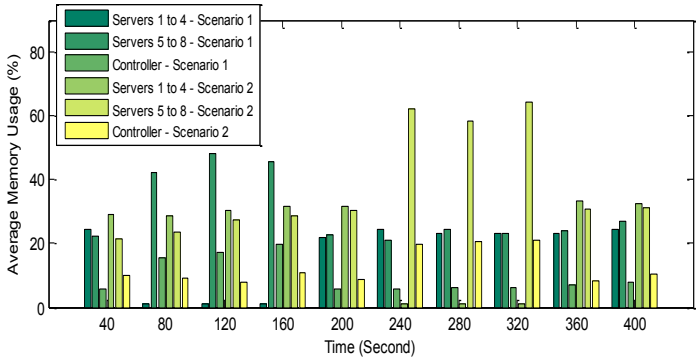
In the previous section, a constant load of 1500 requests per second (1500 rps) is injected into the system, but in this section, we evaluate variable load performance. In the previous section, the servers are not exposed to overload because they had a capacity exceeding 1500 rps (and thus did not exposed to resource shortage). But in this section, as the load increases, we introduce the overload phenomenon and test the performance of the servers and controllers. The results are shown in Fig. 14. The load starts in 1500 and increases to 6000 requests per second (up to 400 seconds) in four steps. Then, as a result of a sudden loss in the second 400, it again reaches 1500 requests per second. In the second 500, it jumps to 6000 requests per second with a sudden jump and reaches 3000 requests per second in the last 100 seconds. Before the second 200, the server and controller throughputs are very close to the offered-load. In the second 200, the servers become overloaded until the second 400. During these 200 seconds, the average throughput of the servers is approximately 3000 rps and the rest of the load provided by the servers is rejected. Servers' overload occurs due to a lack of resources especially the processor (Fig. 14c and 14d).



(a) Throughput



(b) Average CPU usage



(c) Average memory usage

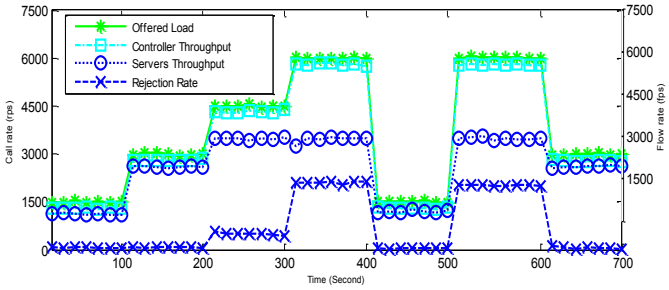
Figure 13: Controller performance when P1 to P4 servers fail

For example, Fig. 14c shows the server CPU saturation at seconds 250, 350 and 550. Usually, in traditional architecture, the server CPU saturation leads to approaching the throughput to zero, however, SDN architecture has been able to prevent the severe throughput loss in the situation of overloading and utilize the maximum server capacity (3000 rps). Unlike servers, the controller does not overload at the offered pick load, and its throughput is always in line with the offered-load. For example, its average throughput in seconds 300 to 400 is approximately 5978 fps. Depending on the throughput, the servers' response time also varies (Fig. 14b). With overload occurring, response times also increase by several times. The overload control algorithm may not be able to restore servers' throughput to normal after the load is returned to less than server capacity (return to the state without any overloading), but when the overload terminates (in second 400), SDN-based architecture has again been able to provide system throughput to the offered-load using the proper load distribution. Unlike the incremental load presented in the first 400 seconds, immediate congestion occurs at second 500. At the same time, despite the 100 percent busy servers' processor (Fig. 14c), the SDN-based architecture has still been able to maximize server throughput. Immediate congestion occurs when a large number of computers request simultaneously. The high throughput of servers in the sudden load fluctuations indicates the stability of the system. Finally, in the last 100 seconds, the rejection call rate is negligible, and the maximum throughput and the response time is approximately 10 milliseconds.

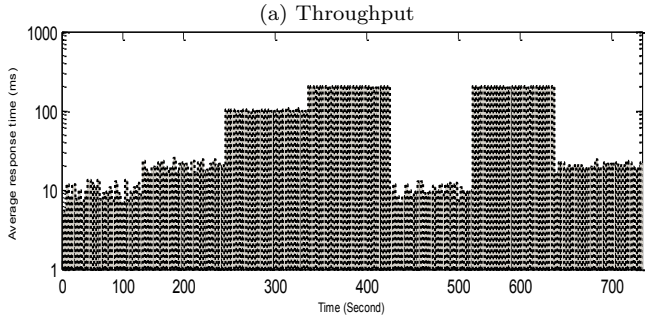
It is worth noting that in a time interval of seconds 200 to 400 or 500 to 600 in which the input load is more than the network resources, we can achieve to the throughput near to offered-load by increasing server resources and removing their hardware limitations.

3.2.3. Experiment 3: Comparison with the traditional Three-layer architecture

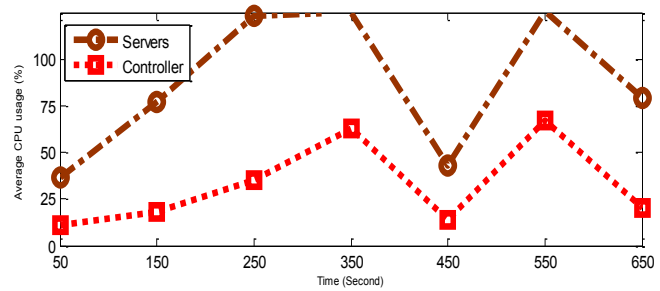
Table 1 provides a comparison of two traditional Three-layer and SDN-based architectures in terms of throughput, delay, and resource consumption. As can be seen, SDN technology has been able to significantly improve the quality of



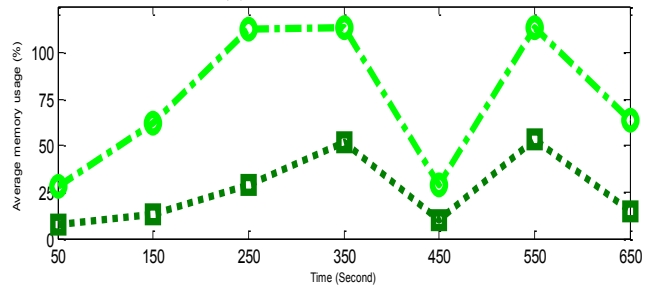
(a) Throughput



(b) Average response time of servers



(c) Average CPU usage



(d) Average memory usage

Figure 14: Performance over time and with different offered-loads

Table 1: Comparison of traditional Three-layer and SDN architectures

	1500 rps		2000 rps		2500 rps	
	Traditional	SDN	Traditional	SDN	Traditional	SDN
Throughput (rps)	956	1432	1432	1897	1998	2456
Delay (ms)	28	12	57	13	98	14
Average CPU Usage (%)	25	16	34	18	44	23
Average Memory Usage (%)	27	17	37	20	55	24

service of server requests including throughput, delay, and resource consumption.

3.3. Discussion

The higher performance of our proposed method comes from an efficient and integrated framework based on SDN concept, which decouples the network control from the data forwarding by direct programming. With its inherent decoupling of control plane from data plane, SDN offers a greater control of a network through programming. This combined feature would bring potential benefits of enhanced configuration, improved performance, and encouraged innovation in network architecture and operations. Especially, SDN offers a promising alternative for traffic steering by programmatically configuring forwarding rules. So, our proposed architecture is indebted to the SDN concept and global view of entire network for performance improvement. Compared to SDN-based approach, the traditional development of approach leads to inefficiency in the use and management of infrastructure hardware resources. As opposed to traditional hardware-centric products where control and data planes are embedded into a *closed* equipment, SDN allows the control plane logic to be decoupled from the network hardware (data plane), and it moves the control logic to a programmable software component, the *SDN controller*[4]. SDN

[4]we want the control of the network to be centralized rather than having each device be its own island, which greatly simplifies the network discovery, connectivity and control issues that have resulted with the current system. Having this overarching control actually makes

controller and APIs (such as OpenFlow) are capable of L2/3/4-based policy enforcement.

So overall, by implementing a new orchestration level, SDN can tackle the inflexibility and complexity of the traditional network. SDN provides enterprises with the ability to control their networks programmatically and to scale them without affecting performance, reliability, or user experience. The data and control-plane abstractions constitute the immense worth of SDN. By eliminating the complexity of the infrastructure layer and adding visibility for applications and services, SDN simplifies network management and brings virtualization to the network. It abstracts flow control from individual devices to the network level. Network-wide data-flow control gives administrators the power to define network flows that meet connectivity requirements and address the specific needs of discrete user communities.

4. Conclusion

In this paper, we proposed an SDN-based framework for load balancing in data centers. This framework was applied to the well-known topologies Three-layer, Fat-Tree, BCube, and Dcell, each implementing a real platform and conducting multiple experiments on each.We also proposed a proactive load balancer mechanism that uses a PID controller for load balancing between data center servers. This mechanism is used as a SDN controller module. The features of this controller are that it is applicable to all four data center architectures. In the testbed, we used Open vSwitch v2.4.1 and Floodlight v1.2 to implement the OpenFlow switch and controller. We experimented each architecture under three tests. The first experiment is the fixed load inject to the data center, the second experiment is the variable load inject to the data center, and the third experiment is to compare traditional architecture with SDN based

the whole network programmable instead of having to individually configure each device every time an application is added or something moves.

architecture. Observations and experiments show that all four SDN-based architectures have been able to well distribute the load across algorithms such as the proposed method or even Round-Robin. They also achieve higher evaluation criteria than traditional architecture and have significant improvements in parameters such as throughput, delay, and resource consumption. One of the future works of this paper is to present a framework based on SDN and NFV (Network Functions Virtualization) technologies in combination, to present virtual servers in data centers in a VNF (Virtual Network Function) manner and optimize their network communications by SDN controller. In this case, the energy consumption of data centers can be optimized. Mathematical modeling of this framework will also be followed in our future work.

Acknowledgment

This work was supported by the Quchan University of Technology (Grant Nos. 11942).

References

[1] A. Darabseh, M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, M. Vouk, A. Rindos, Sddc: A software defined datacenter experimental framework, in: 2015 3rd International Conference on Future Internet of Things and Cloud, 2015, pp. 189–194. doi:10.1109/FiCloud.2015.127.

[2] T. Hu, Z. Guo, P. Yi, T. Baker, J. Lan, Multi-controller based software-defined networking: A survey, IEEE Access 6 (2018) 15980–15996.

[3] Y. Zhang, L. Cui, W. Wang, Y. Zhang, A survey on software defined networking with multiple controllers, Journal of Network and Computer Applications 103 (2018) 101–118.

[4] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, Y. Liu, A survey of machine learning techniques applied to software defined networking (sdn):

Research issues and challenges, IEEE Communications Surveys & Tutorials 21 (1) (2018) 393–430.

[5] V. S. N. Amulothu, A. Kapur, K. Khani, V. Shukla, Adaptive software defined networking controller, uS Patent App. 10/257,073 (Apr. 9 2019).

[6] K. TaKeaWays, The software-defined data center is the future of infrastructure architecture, strategies.

[7] M. Carlson, A. Yoder, L. Schoeb, D. Deel, C. Pratt, C. Lionetti, D. Voigt, Software defined storage, Storage Networking Industry Assoc. working draft (2014) 20–24.

[8] U. Bayram, D. Divine, P. Zhou, E. W. Rozier, Improving reliability with dynamic syndrome allocation in intelligent software defined data centers, in: 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, IEEE, 2015, pp. 219–230.

[9] V. Törhönen, Designing a software-defined datacenter, Master's thesis (2014).

[10] E. W. Rozier, P. Zhou, D. Divine, Building intelligence for software defined data centers: modeling usage patterns, in: Proceedings of the 6th International Systems and Storage Conference, ACM, 2013, p. 20.

[11] U. Paščinski, J. Trnkoczy, V. Stankovski, M. Cigale, S. Gec, Qos-aware orchestration of network intensive software utilities within software defined data centres, Journal of Grid Computing 16 (1) (2018) 85–112.

[12] L. Cui, X. Hailing, D. Chen, Distributed global load-balancing system for software-defined data centers, uS Patent 9,998,530 (Jun. 12 2018).

[13] Y. Hu, C. Li, L. Liu, T. Li, Hope: Enabling efficient service orchestration in software-defined data centers, in: Proceedings of the 2016 International Conference on Supercomputing, ACM, 2016, p. 10.

[14] R. Touihri, S. Alwan, A. Dandoush, N. Aitsaadi, C. Veillon, Novel optimized sdn routing scheme in camcube server only data center networks, in: 2019 16th IEEE Annual Consumer Communications Networking Conference (CCNC), 2019, pp. 1–2. doi:10.1109/CCNC.2019.8651677.

[15] Mingwei Yang, H. Rastegarfar, I. B. Djordjevic, Physical-layer adaptive resource allocation in software-defined data center networks, IEEE/OSA Journal of Optical Communications and Networking 10 (12) (2018) 1015–1026. doi:10.1364/JOCN.10.001015.

[16] Y. Xu, Y. Yan, Z. Dai, X. Wang, A management model for sdn-based data center networks, in: 2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2014, pp. 113–114. doi:10.1109/INFCOMW.2014.6849181.

[17] Wei Hou, L. Shi, Yingzhe Wang, Fan Wang, Hui Lyu, M. St-Hilaire, An improved sdn-based fabric for flexible data center networks, in: 2017 International Conference on Computing, Networking and Communications (ICNC), 2017, pp. 432–436. doi:10.1109/ICCNC.2017.7876167.

[18] H. Yao, W. Muqing, L. Shen, An sdn-based slow start algorithm for data center networks, in: 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2017, pp. 687–691. doi:10.1109/ITNEC.2017.8284820.

[19] Jian Di, Quanquan Ma, Design and implementation of sdn-base qos traffic control method for electric power data center network, in: 2016 2nd IEEE International Conference on Computer and Communications (ICCC), 2016, pp. 2669–2672. doi:10.1109/CompComm.2016.7925182.

[20] P. Kathiravelu, Software-defined networking-based enhancements to data quality and qos in multi-tenanted data center clouds, in: 2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW), 2016, pp. 201–203. doi:10.1109/IC2EW.2016.19.

[21] K. Xie, X. Huang, S. Hao, M. Ma, P. Zhang, D. Hu, E^3 mc: Improving energy efficiency via elastic multi-controller sdn in data center networks, IEEE Access 4 (2016) 6780–6791. doi:10.1109/ACCESS.2016.2617871.

[22] H. Zhu, H. Fan, X. Luo, Y. Jin, Intelligent timeout master: Dynamic timeout for sdn-based data centers, in: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015, pp. 734–737. doi:10.1109/INM.2015.7140363.

[23] R. Hwang, Y. Tang, Fast failover mechanism for sdn-enabled data centers, in: 2016 International Computer Symposium (ICS), 2016, pp. 171–176. doi:10.1109/ICS.2016.0042.

[24] U. Zakia, H. Ben Yedder, Dynamic load balancing in sdn-based data center networks, in: 2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2017, pp. 242–247. doi:10.1109/IEMCON.2017.8117206.

5. Appendix: Supplementary experiments and results

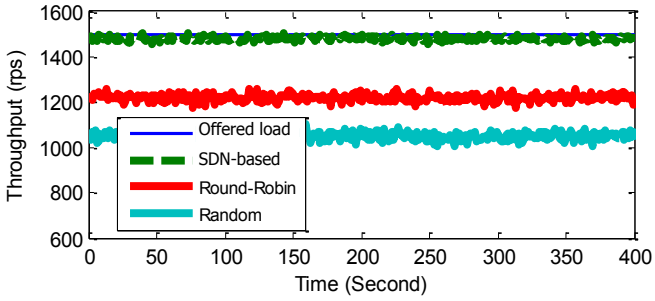
5.1. Experimental results of the Fat-Tree architecture

We use 10 OpenFlow switches and 8 servers according to Fig.3 architecture and repeat the experiments as before to test this architecture.

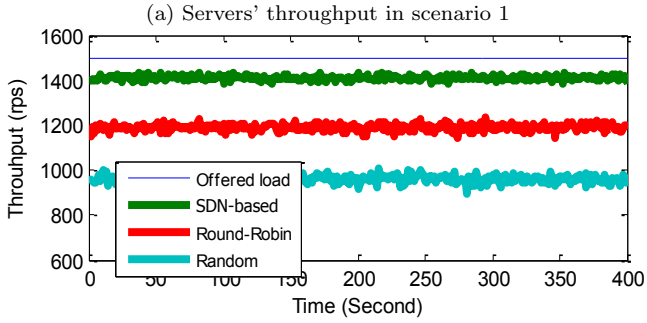
5.1.1. Experiment 1: Fixed load

There are two scenarios in this experiment. In Scenario 1, each server's background traffic is equal to 500 bps. In Scenario 2, the background traffic for Servers 1 to 4 (P1 to P4) is 1000 and for the Servers 5 to 8 (5P to P8) is 500 bps. A fixed offered-load of 400 seconds at a rate of 1500 requests per second is also injected into the system by a traffic generator. Figures 15 and 16 show the performance of the servers.

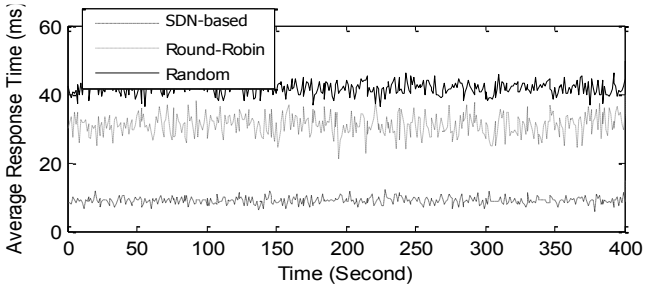
As can be seen from Figures 15 and 16, the throughput and average delay in both scenarios are slightly improved compared to the Three-layer architecture.



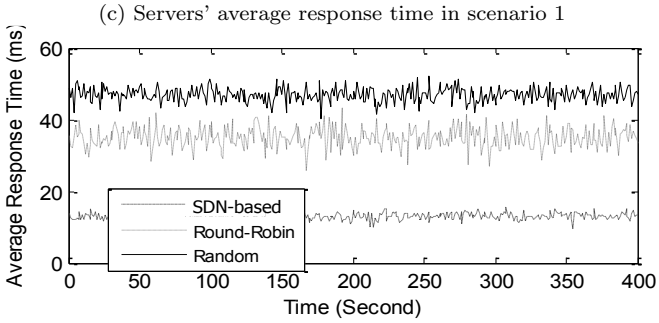
(a) Servers' throughput in scenario 1



(b) Servers' throughput in scenario 2



(c) Servers' average response time in scenario 1



(d) Servers' average response time in scenario 2

Figure 15: Comparison of the performance of data center servers in two scenarios

Table 2: Comparison of traditional Fat-Tree architecture with SDN

	1500 rps		2000 rps		2500 rps	
	Traditional	SDN	Traditional	SDN	Traditional	SDN
Throughput (rps)	844	1389	1376	1787	1997	2497
Delay (ms)	24	11	55	12	96	12
Average CPU Usage (%)	26	15	33	16	42	20
Average Memory Usage (%)	23	14	35	18	53	21

The reason is that the Fat-Tree architecture (compared to the Three-layer architecture) is divided into two general sections. OpenFlow switches are divided into two subsystems in this architecture, which will result in better load balancing between the 8 servers. As in the previous sections, the results of Scenario 2 is slightly worse than the results of Scenario 1 due to the lack of background traffic.

5.1.2. Experiment 2: Variable load

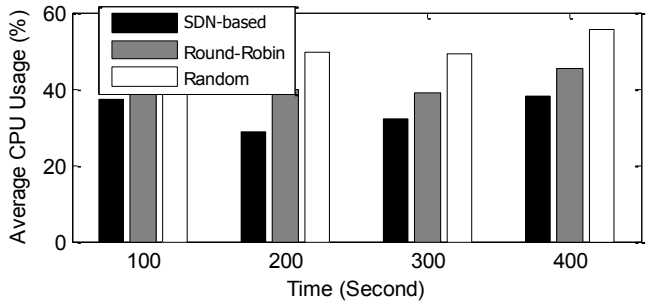
In the previous section, a constant load of 1500 requests per second (1500 rps) is injected into the system, but in this section, we evaluate variable load performance. The results are shown in Fig. 17. These results are almost similar to those obtained from the Three-layer architecture.

5.1.3. Experiment 3: Comparison with the traditional Fat-Tree architecture

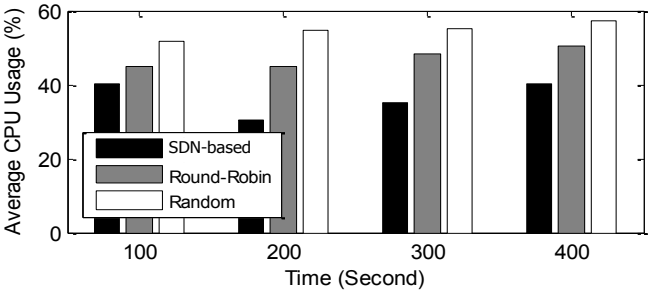
Table 2 provides a comparison of traditional Fat-tree and SDN-based architectures in terms of throughput, delay, and resource consumption. As can be seen, SDN technology has been able to significantly improve the quality of service of server requests including throughput, delay, and resource consumption.

5.2. Experimental results for BCube architecture

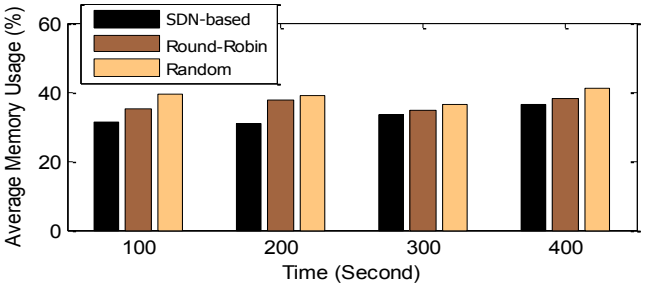
We use 8 OpenFlow switches and 16 servers in accordance with Fig. 5, to test the BCube architecture.



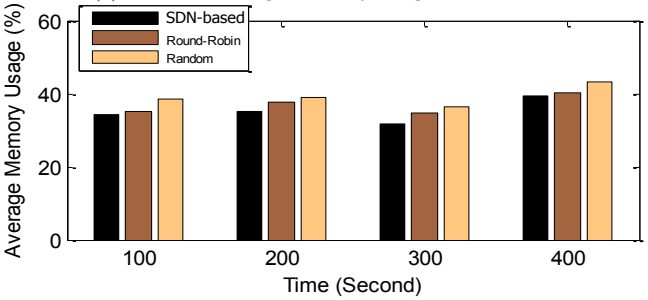
(a) Servers' average CPU usage in scenario 1



(b) Servers' average CPU usage in scenario 2

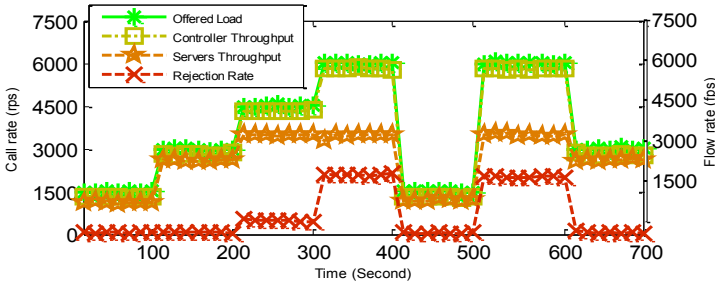


(c) Servers' average memory usage in scenario 1

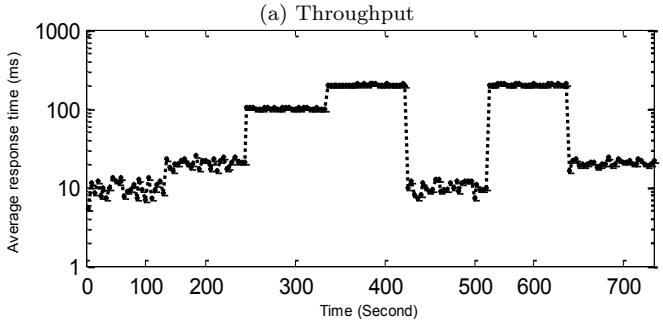


(d) Servers' average memory usage in scenario 2

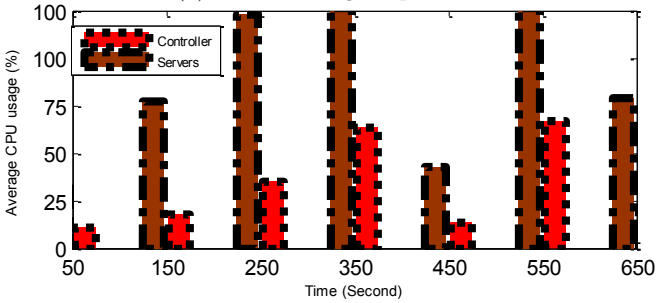
Figure 16: Comparison of data center servers' resource usage in two scenarios



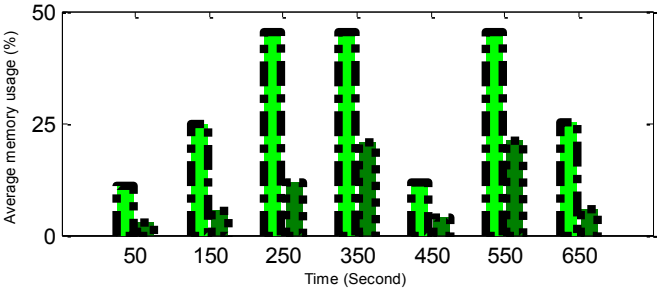
(a) Throughput



(b) Servers average response time



(c) Average CPU usage



(d) Average memory usage

Figure 17: Performance over time and with different offered-loads

5.2.1. Experiment 1: Constant load

The first experiment, as described in the previous sections, consists of two scenarios with different background traffic. In Scenario 1, each server's background traffic is equal to 500 packets per second. However, in the second scenario, the background traffic of the servers are not equal. The results are shown in Fig. 18.

As shown in Fig. 18 and the figures in the previous sections, servers are less efficient in the BCube architecture than in the Fat-Tree architecture but relatively better than the Three-layer architecture. This is also illustrated in the resources usage by the servers in Fig. 19.

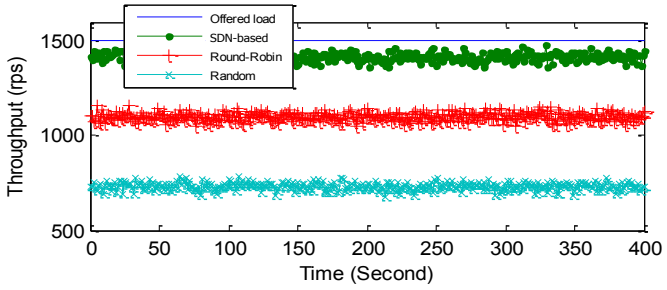
As can be seen, the servers resource usage in this architecture is slightly more than in the Fat-Tree architecture. This is because, in the Fat-Tree architecture, all the components are divided into two parts, and access to each server with fewer jumps (links) is possible, while in the BCube architecture, access from the source server to the destination server is possible with more jumps.

It is worth noting that all three examined architectures have correctly distributed loads, and the process of the three SDN-based, Round-Robin and Random algorithms are consistent in all three architectures, having no significant difference. For example, Scenario 2 consumes more resources than Scenario 1 (Fig. 19).

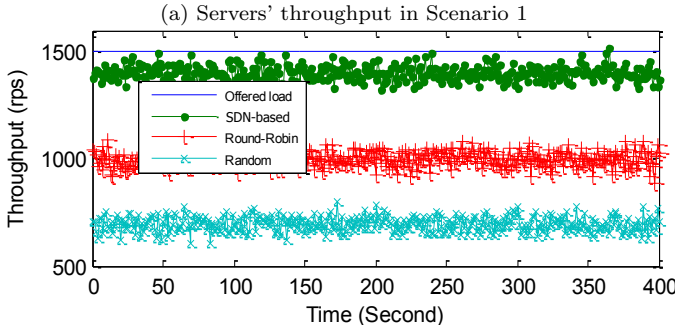
5.2.2. Experiment 2: Variable load

In the previous section, a constant load of 1500 requests per second (1500 rps) is injected into the system, however, in this section, we evaluate variable load performance. The results are shown in Fig.20. These results are almost similar to those obtained from previous architectures.

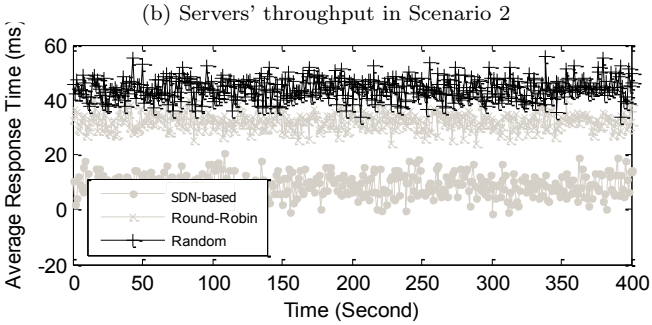
As can be seen from Fig. 20c and 20d, the amount of resources consumed by the controller is not comparable to the servers, and therefore the probability of controller bottlenecks is very low.



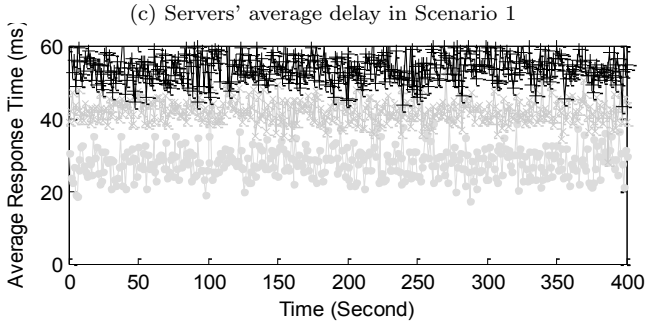
(a) Servers' throughput in Scenario 1



(b) Servers' throughput in Scenario 2

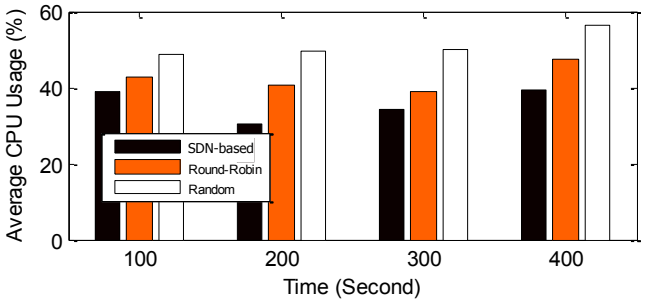


(c) Servers' average delay in Scenario 1

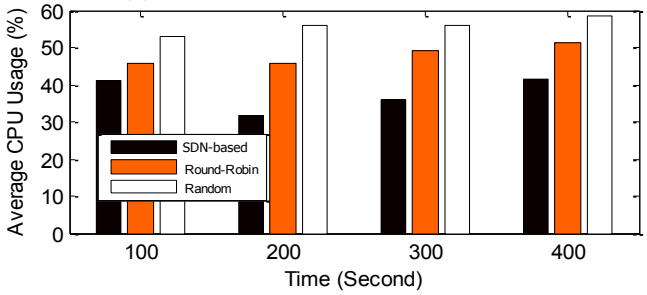


(d) Servers' average delay in Scenario 2

Figure 18: Comparison of the performance of data center servers in two scenarios



(a) The average CPU usage in Scenario 1



(b) The average CPU usage in Scenario 2

Figure 19: Comparison of resource usage by the data center server in two scenarios

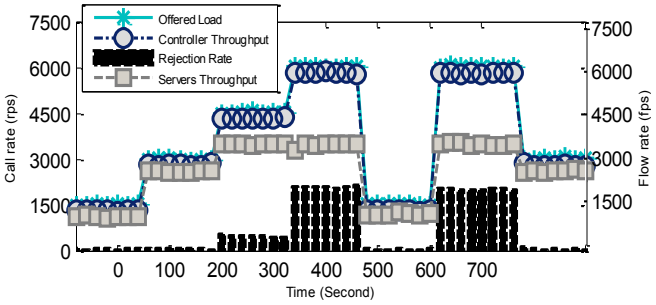
5.2.3. Experiment 3: Comparison with traditional BCube architecture

Table 3 provides a comparison of traditional BCube and SDN-based architectures in terms of throughput, delay, and resource consumption.

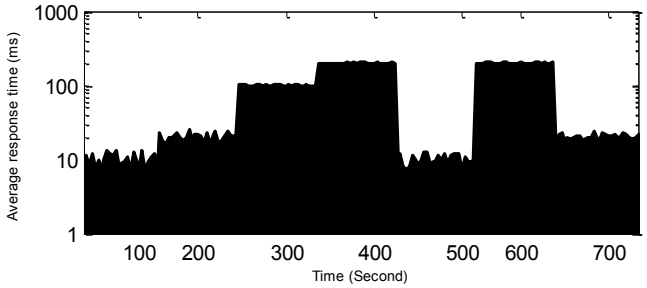
As can be seen, SDN technology has been able to significantly improve the quality of service of server requests including throughput, delay, and resource consumption.up to here, in all three studied architectures, SDN technology has had a great impact on server performance.

Table 3: Comparison of traditional BCube architecture with SDN

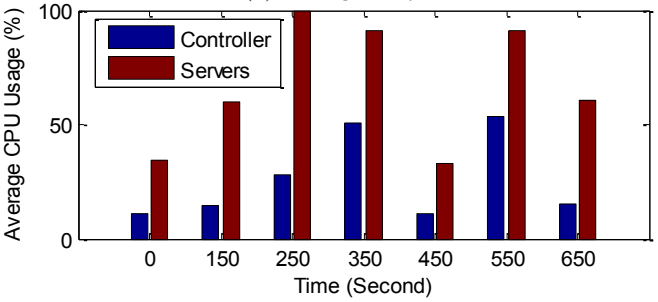
	1500 rps		2000 rps		2500 rps	
	Traditional	SDN	Traditional	SDN	Traditional	SDN
Throughput (rps)	824	1442	1354	1987	2021	2496
Delay (ms)	23	12	56	13	97	11
Average CPU Usage (%)	25	14	34	17	41	23
Average Memory Usage (%)	27	17	36	20	54	22



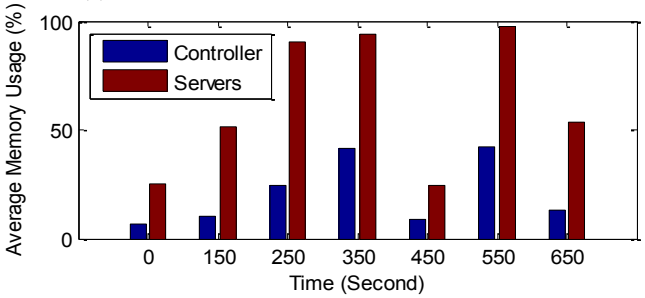
(a) Throughput



(b) Average delay



(c) Average CPU usage of controllers and servers



(d) Average memory usage of controllers and servers

Figure 20: Performance over time and with different offered-loads

5.3. Experimental results of Dcell architecture

We use 5 OpenFlow switches and 20 servers in accordance with the architecture in Fig. 7, to test the Dcell architecture.

5.3.1. Experiment 1: Constant load

This experiment is also a repetition of the first experiment in the previous sections. The first experiment involves two scenarios with different background traffic. In Scenario 1, each server's background traffic is equal to 500 bps. In Scenario 2, the background traffic for servers 1 to 10 (P1 to P10) is 1000 and for servers 11 to 20 (P11 to P20) is 500 bps. Fig. 21 illustrates the performance of SIP servers in this architecture.

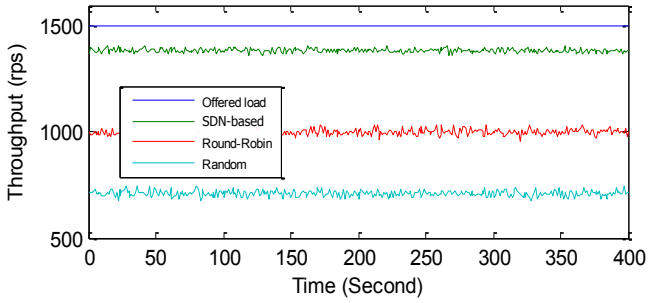
As can be seen, the delay difference of the three SDN-based, Round-Robin and Random methods is less than in the previous architectures. In other words, the performance of these three algorithms in this architecture is close to each other (compared to the previous three).

The results of consuming resources such as CPU and memory are almost similar to the BCube architecture and we refuse to explain them here.

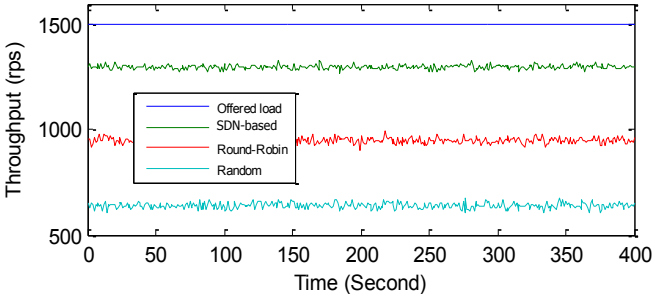
However, as can be seen from the results above, the SDN controller has been able to distribute the load with high throughput and low latency between the servers.

5.3.2. Experiment 2: Variable load

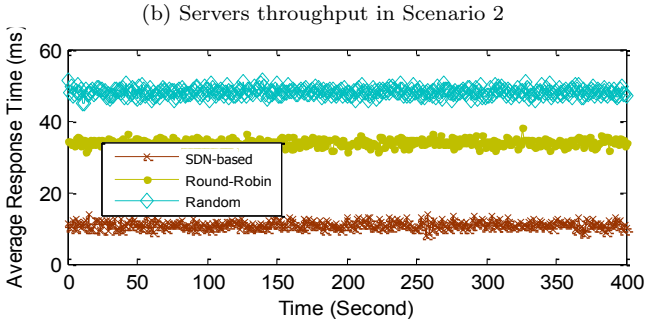
This section evaluates the performance of variable loads. However, since the results of this section are similar to those of the BCube architecture, we refuse to explain them again. It should be noted that the intended architecture for Dcell consists of five cells and the transfer of packets from one cell to another is slightly delayed compared to other architectures. For example, according to our observations, this architecture had a slightly longer delay than the previous architectures in sending loads from server 1 in cell 1 to server 2 in cell 2. It should be noted, however, that SDN-based architectures have greatly improved service quality than traditional architectures.



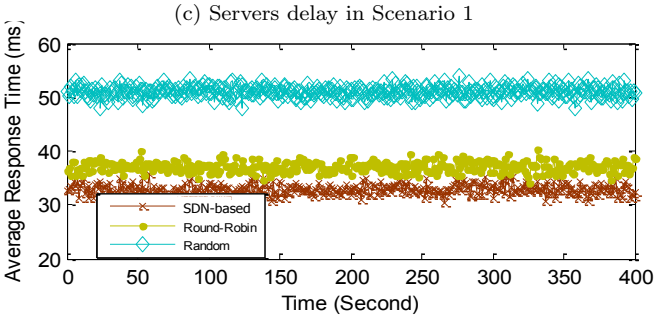
(a) Servers throughput in Scenario 1



(b) Servers throughput in Scenario 2



(c) Servers delay in Scenario 1



(d) Servers delay in Scenario 2

Figure 21: Server performance over time

Table 4: Comparison of traditional Dcell architecture with SDN

	1500 rps		2000 rps		2500 rps	
	Traditional	SDN	Traditional	SDN	Traditional	SDN
Throughput (rps)	1163	1445	1451	1998	2065	2498
Delay (ms)	22	12	55	11	98	12
Average CPU Usage (%)	24	11	33	18	42	22
Average Memory Usage (%)	25	16	33	18	55	23

5.3.3. Experiment 3: Comparison with traditional Dcell architecture

Table 4 provides a comparison of traditional Dcell and SDN-based architectures in terms of throughput, delay, and resource consumption.

As can be seen, SDN technology has been able to significantly improve the quality of service of server requests including throughput, delay, and resource consumption.