Adaptive Load Balancing Scheme For Data Center Networks Using Software Defined Network

Shavan **K. A**skar
Electrical and Computer Engineering Department, Collage of Engineering, University of Duhok, Kurdistan Region-Iraq.
(Accepted for publication: December 21, 2016)

Abstract:

A new adaptive load balancing scheme for data center networks is proposed in this paper by utilizing the characteristics of Software Defined Networks. Mininet was utilized for the purpose of emulating and evaluating the proposed design, Miniedit was utilized as a GUI tool. In order to obtain a similar environment to the data center network, Fat-Tree topology was utilized. Different scenarios and traffic distributions were applied in order to cover as much cases of the real traffic as possible. The suggested design showed superiority over the traditional scheme in term of throughput and loss rate for all the evaluated scenarios. Two scenarios were implemented; the proposed algorithm presented a loss-free performance compared to 15% to 31% loss rate in the traditional scheme for the first scenario. The proposed scheme showed up to 81% improvement in the loss rate in the second scenario. In term of throughput, the proposed scheme maintained the same level of throughput in the first scenario compared to an average of 5Mbps reduction in the throughput when using the traditional scheme. While in the second scenario, the new scheme outperformed the traditional scheme by showing an improvement of up to 16.6% in the throughput value.

Keywords: *Software Defined Network; Data center; POX controller; Fat-Tree; Mininet; miniedit, Load Balancing, Datacenter.*

I. INTRODUCTION

Data Center Networks (DCN) witnessed an unprecedented development over the past few years in an attempt to accommodate the huge increase and requirements' change in the traffic. To handle such big data, special consideration has to be taken for traffic monitoring and management because any disruption in the service or presenting undesirable QoS parameters would lead to massive revenue loss (Yang et al., 2016; Shavan et al., 2011).

Traffic of networks is mainly comprises of control plane traffic and data plane traffic. The majority of load balancing schemes deal with the data plane traffic as its percentage is far more than the control plane traffic. In present, Data centers deploy hierarchical network architecture with multi-path characteristics such as Fat-Tree topology. The existence of multi-path routes facilitates having different routes to the same destination and this will help having a better load balancing options. Fat-Tree topology has been implemented in many modern DCs such as (Heller et al., 2010, Mohammad Al-Fares et al., 2010). Figure 1 shows a Fat-Tree topology with four pods.

Although there is more than one rout into a particular destination in a Fat-Tree network, however, the classical distance vector and link state routing protocols cannot utilize this multi-path property. Internet routing protocols usually routes and forwards packets based on the destination IP address. As a consequent, packets with the same intended destination address will be routed at the same path (Shubhi, 2015; James and Keight, 2012).

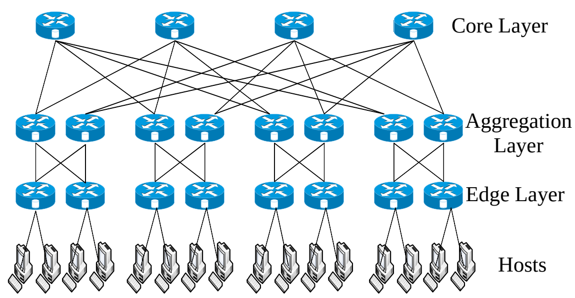

Figure 1: 4-Pod Fat-Tree topology

Undoubtedly, there are some routing protocols that have equal cost multipath (ECMP) characteristic; however, they split traffic statically depending on the information obtained from a packet's header. As a result, there will be no consideration for traffic flow's requirements in term of QoS parameters; in addition, the status of the overall network load is not taken into consideration. In other words, those kinds of ECMP algorithms are merely capable of selecting among multiple paths that have equal least cost (Heller et al., 2010; James and Keith, 2012).

The main difference between routing of DC traffic and internet traffic is that; internet routing protocols often emphasize on selecting the shortest path to reduce the delay. Whereas, DCs are composed from servers that usually located in close distances, therefore, the concerns is more than just the latency, it is about balancing the huge traffic. The pre-mentioned bandwidth balancing function is not attainable in traditional DCNs because of the nature of the traditional switches utilized in those kinds of networks. The switches that are deployed in traditional DCNs do not have a global view on the entire network resources such as the remaining link bandwidths and alternative paths in a real time manner (Dan Li et al., 2015; Liming and Gang, 2016; Feilong et al., 2016).

An adaptive load balancing DCN is proposed in this paper by means of utilizing SDN switches and controller. The main difference between the SDN network structure and the traditional network is that in SDN, the forwarding process is conducted in a centralized manner by means of a controller and forwarding switches and this is considered as the main advantage for conducting an efficient load balancing over the traditional DCNs. Figure 2 shows a simple architecture of the SDN network. The SDN controller has a comprehensive overview on the type of flows, links' utilization, and the available paths to the intended destination. These kinds of information help in performing more efficient load balancing algorithms than if it is limited to distributed protocols for routing and traffic monitoring as it is the case with the traditional network architectures (Zhaogang et al., 2016).

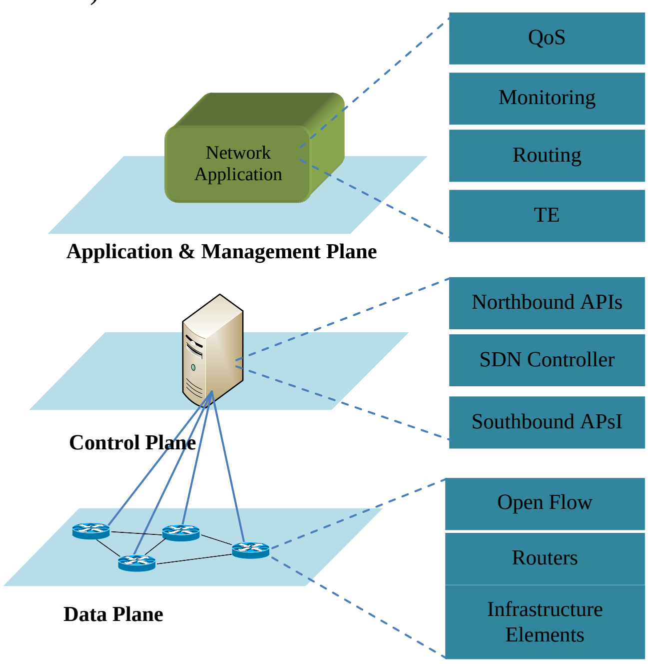

Figure (2): SDN Architecture

As shown in Figure 2, SDN networks consist of three main layers; data layer, control layer, and application\management layer. The data layer comprises of network devices such as routers, OpenFlow switches, and wireless devices. The operation of these devices differs from their function at traditional networks; in SDN, they are merely forwarding devices while the intelligence unit that is responsible for making decisions is located at the controller. The case is different with traditional networks that come with network devices with their software or control unit built inside them. SDN allows network administrators of configuring and managing network's traffic which contributes into better utilization for network resources. The concept of SDN was originally proposed by Stanford University (Sixto, 2013). SDN separates the control plane from the data plane on its network devices; in addition, it allows having an entire overview on the network resources that supports making changes globally not in a centralized manner as in traditional networks. This new network technique is implemented utilizing some open standards such as OpenFlow. OpenFlow is one of the most important protocols that are capable of configuring, managing, and interoperating between different network devices (ONF, 2015). As shown in Figure 2, SDN networks consist of two major elements which are namely; the controller (control plane) and the forwarding devices (data plane). The forwarding device could be a switch or a router that is in charge of forwarding packets only. On the other hand, the controller is considered as brain of the network, it is simply software operating on a specific hardware platform. The controller is communicated with the OpenFlow switches via a secure channel that runs an OpenFlow protocol. SDN controller inserts flow entries, modify flow entries, query, and has an overview of the whole network resources. OpenFlow forwarding switches keep statistics of each flow and port such as the total number of transferred bytes and the duration time of each flow. The forwarding switches and controller coordinate their work as follows; if the path of the flow is already known (not the first packet of the flow), then the forwarding switch would not need to consult the controller and it can forward packets on the fly. However, for first packet case (the income packet does not match any flow entries of the Ternary Content Addressable Memory table), the switch needs to consult the controller to find a suitable outgoing port (Xuan-Nam et al., 2016; Andreas et al., 2016; Ian et al., 2016).

The proposed scheme aims at adaptively balancing the load by means of re-routing into an alternative path based on information obtained from the SDN controller.

The rest of this paper is organized as follows; Section II gives a description for the previous research on providing load balancing schemes for data centers and some of the early attempts on using SDN for this purpose. Section III describes the proposed adaptive load balancing scheme using SDN architecture. Section IV presents the obtained results and analyzes them. Section V concludes the paper.

II. Related Work

Load balancing problem is one of the major issues in DCs in their different shapes, whether they are physical DCs or virtual DCs. DCs usually allow multiple paths routing for the purpose of improving the tolerance to faults in addition to increasing network's throughput by means of sorting out the problem of congestion. Layer 2 and Layer 3 are capable of running multipath routing; however, each layer deploys it based on its protocols. For instance, spanning tree is utilized by Layer 2; therefore, only one path would be available for a pair of sender receiver nodes at a time. There are some proposals to support multipath with Layer 2 such as the one conducted by (Jayaram et al., 2010). They proposed exploiting the redundant paths in the network using an algorithm that calculates a set of available paths and combines them into another set of trees. On the other hand, at Layer 3, routers support ECMP by implementing static load separation between the available flows. Switches that have their ECMP property enabled would have more than one path in each subnet. Upon receiving an incoming packet, switches utilize the hash function (interpreting packet header) in order to select one of the available paths for forwarding purpose. However, ECMP does not take into consideration the flow bandwidth when selecting paths which may results in overloading links unnecessarily where other links may already be available as it is shown in Figure 3. In addition, ECMP has a problem in its practical implementation because the available paths for selection are either 8 or 16 paths which are much lower than the needed paths for the purpose of providing bisection bandwidth, in particular, when dealing with big data as it is the case with DCNs.

Figure 3 depicts a scenario where ECMP is utilized and where it can't utilize network's links in an efficient way because of the phenomena mentioned above, that is not counting for flow bandwidth. One of the major drawbacks of ECMP is that long flows may contend on the same output port based on their hash values, this would consequently lead into bottleneck (Wei et al., 2016).

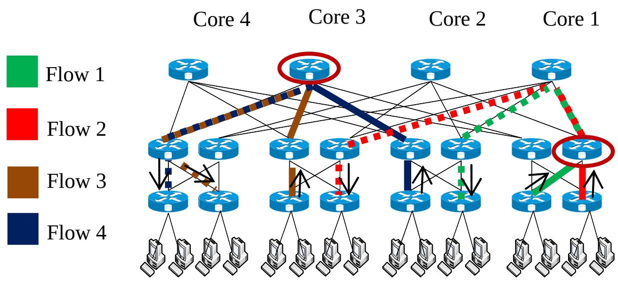

Figure (3): Scenario depicting ECMP problem

Figure 3 shows a scenario of Fat-Tree topology in which all networks' links are of 10 Gbps bandwidth. Flow 1 and Flow 2 sending traffic with 10 Gbps each, because of the hashing, they contend at the aggregation level (encircled with red colour) for the same output port that routes to the core level. This collision results in halving the throughput of each of them. The other collision is happened between Flow 3 and Flow 4 at the core level. Obviously, their throughput is halved as the link requires carrying their overall traffic which is equal to double of the link capacity. A Fat-Tree Topology with four pods as depicted in Figure 3 should allow for four different paths for each host, however, an efficient algorithm that can utilize this property is needed. This means that with an existence of the right load balancing scheme, the four flows would have transferred traffic in a rate of 10Gbps instead of 5 Gbps. This could have been happened if Flow 1 was directed into Core 2 and Flow 3 into Core 4 (Wei et al., 2016; Zhiyang et al., 2015).

A research is conducted in (Wang et al., 2016) to improve the hash algorithm by distributing the data flow. A detection algorithm is utilized to find out the occupancy duration for the purpose of identification weights of each load and their dense points. Another research was conducted in (George et al., 2003) in which a shared memory for network data flow was proposed by means of multiprocessor model. Priority and weight schemes were deployed in order to evenly distribute network flows to the processor. However, in addition to the lack of an overview of flow bandwidth, one of the drawbacks of the abovementioned algorithms is that their systems are closed. In addition, their software and hardware is tightly coupled, therefore they are not suitable for the high development growth of Internet.

III. ADAPTIVE LOAD BALANCING SCHEME

In addition to the above mentioned issues with ECMP, traditional load balancing techniques come with a dedicated hardware that is in charge of conducting the function of load balancing as depicted in Figure 4.

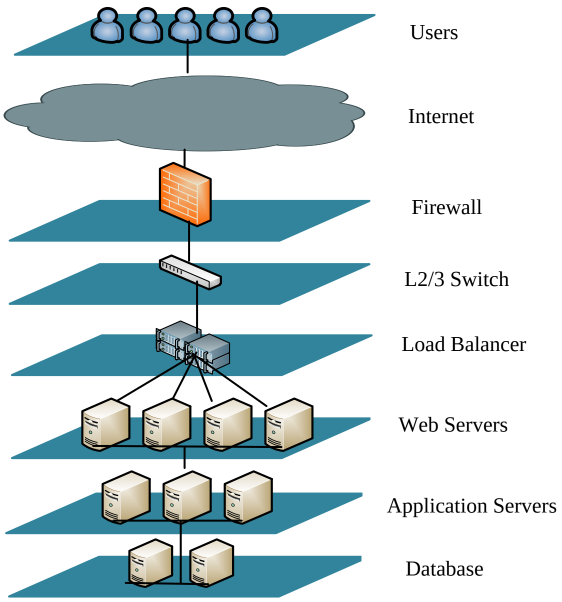

Figure (4): Traditional load balancer for DCN

When users try to access backend servers shown in Figure 4, it would be the role of the load balancer to check the list of backend servers out and select an appropriate load balancing algorithm for the purpose of distributing clients' access into the available servers. Therefore, the load balancer should keep track of all the established sessions; in addition, all the packets that have the same TCP\UDP addresses would be forwarded into the backend servers no matter to what flow they belong. In this case, the load balancer should has\and executes network address translation by updating the source; port number, and the IP addresses of the outgoing packets while conducting an opposite job when receiving packets by matching the destination; IP and port number addresses for the incoming packets with its table (Senthil and Ranjani, 2015). The dedicated load balancer has more drawbacks than the above-mentions ones; it is an expensive solution, not a flexible technique, undergoes from the problem of having single point of failure, and leads to bottleneck for the whole system (Senthil and Ranjani, 2015; Mao and Shen, 2015).

A generic overview of the proposed adaptive load balancing system is shown in Figure 5. The main difference between the proposed system and the traditional one is that there is no dedicated hardware for the purpose of load balancing. Instead of a dedicated load balancer and traditional switches, the proposed scheme utilizes OpenFlow switches that could be programmed to work under any needed function whether as a router, switch, or hub. OpenFlow switches works under the supervision of a controller that is connected to all the switches and has an entire overview of the whole network and its resources. The property of the controller is exploited for the sake of having an efficient load balancing scheme, this is conducted by deploying the load balancing algorithm inside the POX controller. The role of the controller of a DCN is to manage requests received from clients and forward them into a specific path to a particular server based on the information of the entire network that is already gathered by the controller. SDN controller is capable to adaptively add, delete, and modify entries of the flow table of the OpenFlow switches for the sake of balancing the load of the network.

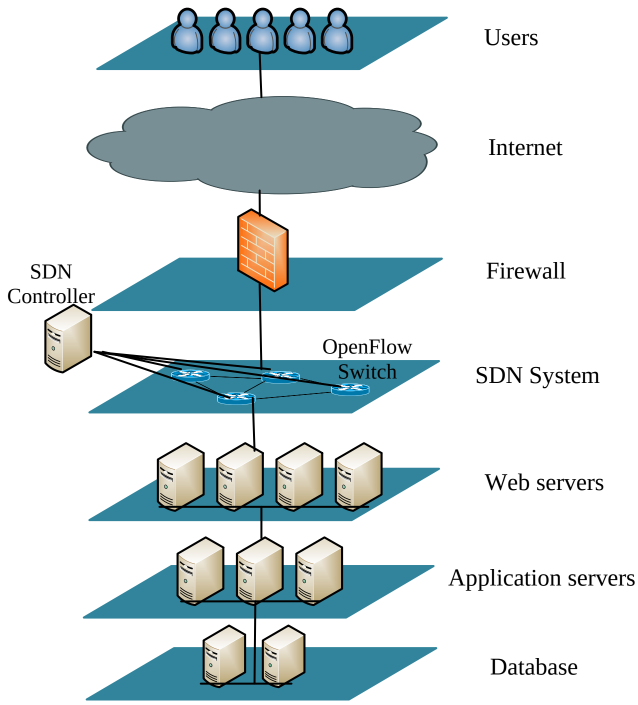

Figure (5): Generic Architecture for the proposed Adaptive SDN load balancing scheme.

The proposed architecture aims at adaptively balancing the load of the DCN based on some triggering parameters that could be set either manually (DCN administrator) or dynamically based on service requirements, in both cases, the network status plays a major role in initiation the load balancing algorithm. To meet a reliable evaluation for the proposed scheme, two aspects have been taken into consideration. First, is to utilize exactly the same network topology that is deployed by DCNs that is, a Fat-Tree network topology. Secondly, to utilize the most reliable emulator for SDN networks, that is Mininet emulator (Mininet, 2016; Faris and Shavan, 2015).

The proposed DCN scenario is evaluated by means of a Fat-Tree network with k=4, the proposed architecture is emulated utilising Mininet emulator as shown in Figure 6 that represents a snapshot of the emulated network. Fat-Tree topology is built with K ports switches and it consists of K-pods. Each pod has two layers, aggregation and edge as indicated in Figure 1. The available paths between any two hosts in a K-pods Fat-Tree network is $(K/2)^2$, this means that there are four routing paths between any two servers of the network shown in Figure 6. In addition, the entire K-pods should be connected into $(K/2)^2$ Core switches (4 core switches) (Jun and Yuanyuan, 2016).

The proposed adaptive load balancing algorithm is programmed inside a POX controller that belongs to the SDN based DCN. The triggering parameter for the proposed algorithm is bandwidth and loss which are the two most important factors when dealing with DCNs. Accordingly, when a received throughput is decreased under its expected value or in case there is an increase in the loss value in one or couple of the DCN links (throughput and loss are interrelated and gives that same indication), then the proposed algorithm takes action. The pre-mentioned scenario is when there are already established connections and there is an increase in the traffic that leads to loss, however, if the connections among servers and clients started with high bandwidth requirements, then the algorithm will find optimal path at the beginning of creating the connections. The initiation starts with the controller which has an entire overview on the whole network resources as shown in red lines in Figure 6. The controller exploits this facility and finds alternative paths for the reduced throughput traffic or for the traffic that undergoes of high loss rate.

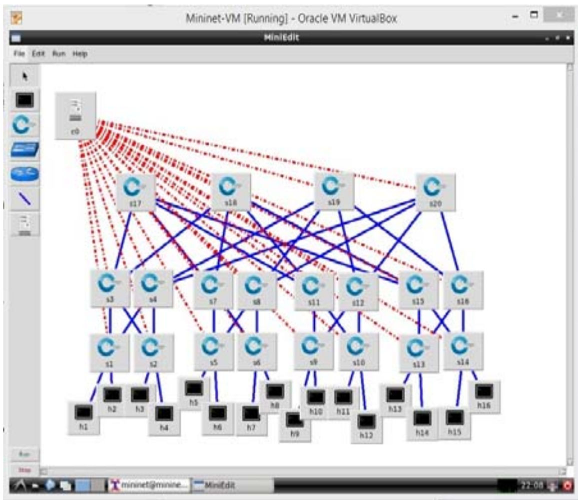

Figure (6): The emulated Fat-Tree DCN utilizing Mininet Emulator.

Figure 7 shows a flow chart of the proposed adaptive load balancing algorithm. Two cases are considered; the first case where there is a new joining client, whereas, the second case is where there is an already established connection between two pairs and there is a demand to increase the throughput which may affect other communication parties. It is assumed that the proposed scheme collects the throughput requirements for specific applications and keeps that information in the controller. Once there is a contention in one of the links, the throughput of those applications may goes lower than their pre-specified threshold value; therefore, the algorithm will be initiated to conduct load balancing in order to attain the original required throughput. Because the Fat-Tree network utilized in the proposal has 4 pods, there will be four routes between any two hosts (servers). Therefore, the controller will search for the rest of the three $((K/2)^2-1)$ alternative paths to find out the best one as described in Figure 7. The same scenario is applied when there is an increase in a demand between two already connected parties, this increase in demand will be examined whether it would lead to reducing the throughput below its threshold value or if it cause any increase in the loss value. If any of the two pre-mentioned cases are met, there will be a need to change into another route.

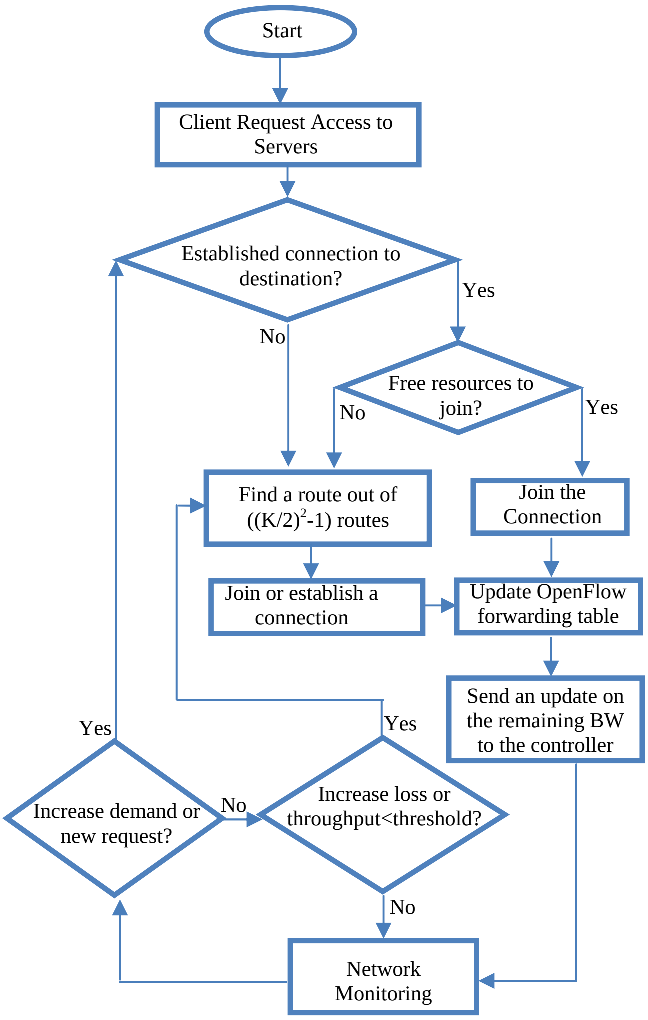

Figure (7): The proposed Algorithm

Upon changing the path, the controller updates the OpenFlow forwarding table of the OpenFlow switches. Simultaneously, information about the remaining bandwidth of the new and former links is sent to the controller so that the controller is aware of the available network resources in case of future reservation for other parties. The controller exploits the opportunity of having an overview of the entire network; accordingly it performs two kinds of tasks which are namely; network monitoring and allocation of resources. Monitoring is conducted by sending requests to all the switches in a periodic manner. Up on receiving requests, an analysis is conducted for the reply packets in order to determine the best route to the intended destination. Monitoring would not add much overhead because the request and reply messages are too small; the request packet length is 8 Bytes while the reply packet length is 104 Bytes (Yi-Chih et al., 2015).

IV. Emulation and Results

This Section describes the emulation environment, emulation tools, and the obtained results. All experiments were conducted utilizing HP ENVY dv6 PC with core i7 intel (R) processor Core (TM) i7-3630QM CPU 2.40GHz, 6 GB RAM, and 64-bit Windows 8 operating system. Virtual Box Oracle VM version 5.0.16 was utilized, in addition, the guest OS of the VM was installed with Linux OS Ubunto 14.04 32-bit and 1GB RAM. Mininet 2.2.1 emulator was installed on this VM, with POX 2.0 controller. The emulated DCN is of Fat-Tree type with four pods, 8 aggregation OpenFlow switches, 8 edge OpenFlow switches, 4 core OpenFlow switches, and 16 hosts. In order to obtain more realistic and reliable results; small packets and relatively small link capacities bandwidth were utilized because the performance of Open Virtual Switch (OVS) and OpenFlow controller created by Mininet is effected by underlying OS, available processor and the allocated memory (Alexander et al., 2015). Accordingly, all link bandwidths have a capacity of 10Mbps.

Traffic generation and throughput measurement was conducted by means of Iperf tool which is a network testing tool that can generate Transmission Control Protocol (TCP) and User Datagrams Protocol (UDP) packets in order to measure the throughput of a network (Iperf, 2016). For the purpose of evaluating the proposed scheme, two scenarios were investigated. The first scenario (Scenario A) is depicted in Figure 8 where at the beginning, two hosts, namely H16 and H10 send traffic with a rate of 8Mbps (Flow 2 in red colour) and 7Mbps (Flow 4 in blow colour) to Hosts H8 and H1 respectively. Mininet was utilized as an emulation tool for the purpose of designing and evaluating the proposed scheme. In addition, Mininet was used in order to feed the network with traffic and measure the throughput via the command Iperf. Mininet is programmed using Python programming language.

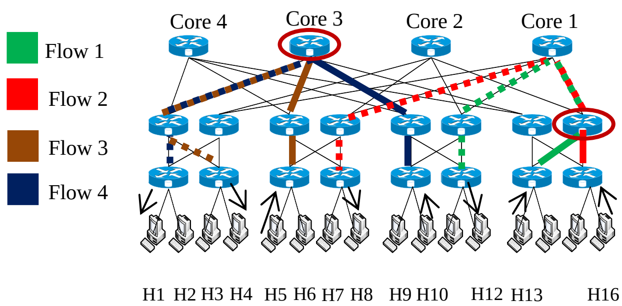

Figure (8): Emulation of the first Scenario

The emulation period is 20 seconds where loss, throughput, and delay are recorded every second at the receiver. At time 0 Sec, H16 and H10 start sending their traffic to their intended destinations (H16-H8, H10-H1). At time 5 Sec, Flow 3 starts when H5 start sending traffic of 5Mbps rate to H4. Relying on the hash way for routing and load balancing, Flow 3 and Flow 4 contend for the same outgoing port and accumulate an overall traffic of 12Mbps that leads to reducing the throughput and increasing the loss rate. At time 10Sec, H13 starts sending traffic of 5Mbps rate (Flow 1 in Green colour) to H12 that would apparently contends with Flow 2 and they together make traffic of 13Mbps.

Figure 9 shows the obtained results of throughput when the traditional hashing method is utilized, as it could be noticed, up to the fifth Second of the emulation period, H16 and H10 were sending an average traffic rate of 8Mbps and 7Mbps respectively. Then H5 joins with 5Mbps traffic rate so apart from its intended receiver (H4), it affects H1 only because it contends with the traffic sent by H10 at the core level as depicted in Figure 8. Therefore, their received throughputs are reduced as depicted in Figure 9. At time 10 Seconds, H13 starts transmitting traffic to H12 with 5Mbps rate. Again, there will be a collision with the traffic of Flow 2 but this time it will occurred at the aggregation level. This leads to dropping the throughput of hosts H12 and H8 as shown in Figure 9.

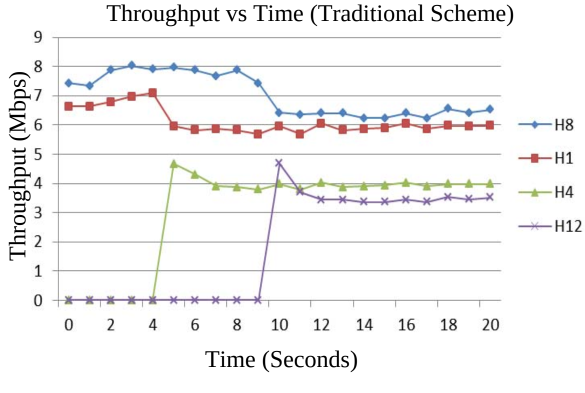

Figure (9): Throughput versus emulation time for Scenario A when utilizing traditional hash load balancing technique.

When deploying the proposed adaptive load balancing scheme to the same scenario and traffic distribution, then neither the new joined hosts nor the already transmitting hosts will be affected as shown in Figure 10. The reason is that the load balancer has a full overview over the entire network and once it receives a packet that belongs to a new flow, it allocates free resources to it without undergoing any loss. The controller inserts a new entry in the OpenFlow forwarding tables to establish a connection of the new joined server. However, the case may be different in the Second Scenario when there is an increase in demands for an already established connection, in this case, there will be some affect that lasts very short time as it will be depicted later.

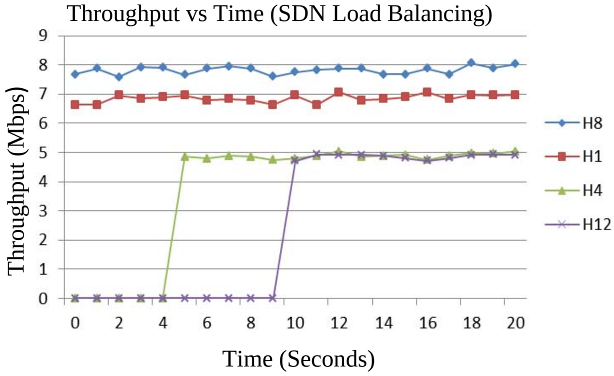

Figure (10): Throughput vs Emulation time for Scenario A when utilizing the proposed adaptive load balancing algorithm.

Figure 11 shows the loss results when traditional technique is deployed; obviously there is not any loss in the case of the proposed adaptive load balancing method. As indicated in Figure 11, the loss rate starts gradually for hosts H1 and H4 at time 5 Seconds when H5 joins by sending traffic via the network and cause collision at the core level. However, hosts H8 and H12 start undergoing from loss at time 10 Seconds when H13 joins the network that leads to congestion at the aggregation level as depicted in Figure 8.

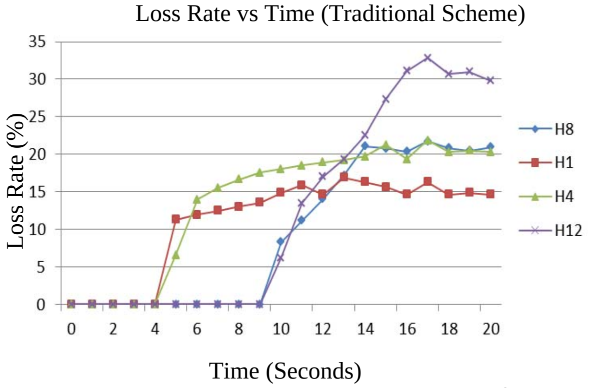

Figure (11): Loss Rate versus emulation time for the traditional scheme

As mentioned earlier, the SDN controller can fully control and prevent any loss in such a case because H5 and H13 are new to the network and their flow will be optimally allocated by the controller, therefore, the loss rate is equal to 0% for such a case when utilizing the proposed adaptive load balancing scheme. Nevertheless, there would be some loss if the connection is already established as it will be explained in the Second Scenario (Scenario B).

For the sake of simplicity for the reader, the same topology and sender-receiver pairs that are shown in Figure 12 are assumed for Second Scenario. However, the starting sending rate is way lower than the First Scenario, where, H16 and H13 send traffic rates of 5Mbps and 4Mbps respectively, they utilize the same route to their intended destinations, H8 and H12 respectively. It is also assumed that H5 and H10 send traffic with 2Mbps and 6Mbps rates to H1 and H4 respectively. The main difference between the two scenarios is that in the second Scenario, flows are already established; therefore, in case that the demand for capacity goes beyond link's capacity, the controller will call the adaptive load balancing algorithm to conduct load balancing. Whereas, in the first scenario, the flow were not established when it was required to send traffic higher than link's capacities.

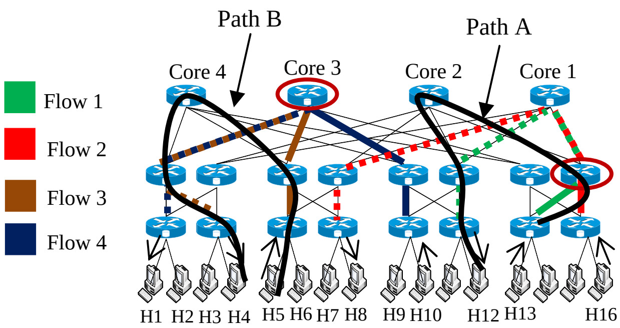

Figure (12): Emulation of the Second Scenario

H5 (destination H4) increases its demand from 2Mbps to 6Mbps at time 5Sec as shown in Figure 13. For the case of the traditional scheme, there will be a contention between Flow 3 and Flow 4 which leads to degrading the throughput and increasing the loss rate for H4 and H1 as they share the same route as it is depicted in Figure 13. The expected throughput of H4 is supposed to be 6Mbps, however, as it is depicted in Figure 13 (green colour), it does not exceed the average of 4.3Mbps. In addition, the contention affect H1 by reducing it is already established connection's throughput from 6Mbps into around 5.5 Mbps as depicted in Figure 13 (blue colour). On the other hand, when utilizing the adaptive load balancing scheme, the contention triggers the proposed algorithm to take an action as there is an increase in the loss rate. The controller takes the initiation and dictates OpenFlow switches to change their forwarding table into a new route based on the information that the controller has about the entire network. Therefore, it re-route Flow 3 into Path B as shown in Figure 12. In addition, H13 (destination H12) increases its sending rate from 4Mbps into 8Mbps at time 10Sec as depicted in Figure 14; this would have consequences on Flow 1 and Flow 2. The controller triggers the adaptive load balancing algorithm to choose an alternative path from the available three paths; it selects Path A to forward the traffic of Flow 1 as depicted in Figure 12. The algorithm re-routes the traffic sent by H13 into Path A; similarly it changes the route of the traffic sent by H5 into Path B as depicted in Figure 12. As it is depicted in Figure 14, the increase in demands would have an effect for a very short time; afterwards, the expected throughput is attained as depicted in blue and red coloured curves of the same Figure. Figure 14 shows that in the case of a traditional scheme, the increase of Flow 1 will have a devastating effect on Flow 2 as shown in blue and green coloured curved.

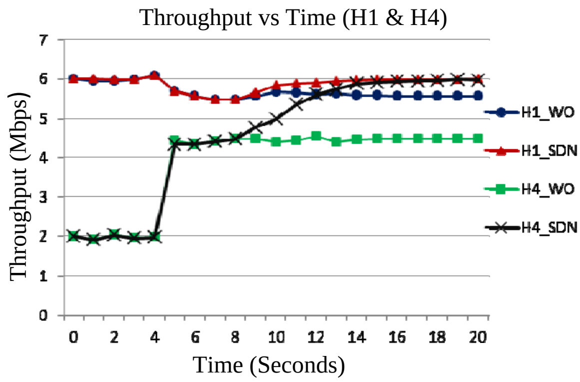

Figure (13): Scenario B, throughput comparison between the traditional scheme and the adaptive load balancing scheme for for H1 and H4.

Figures 15 and 16 depict the loss rate versus the emulation time for the second Scenario (Scenario B). It could be noticed how the throughput and loss values are degraded only for very short times when utilizing the adaptive scheme.

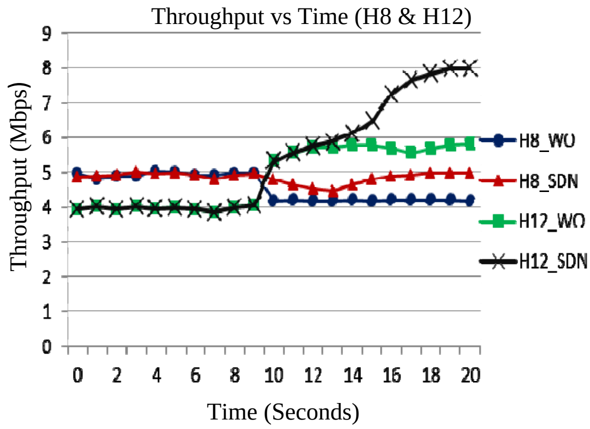

Figure (14): Scenario B, throughput comparison between the traditional scheme and the adaptive load balancing scheme for H8 and H12.

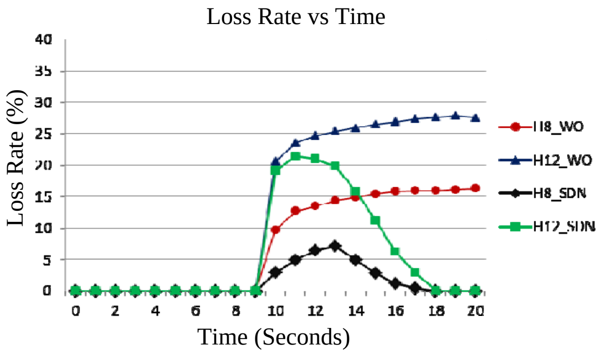

Figure (15): Scenario B, loss rate comparison between the traditional scheme and the adaptive load balancing scheme for H8 and H12.

The results showed that the proposed algorithm has considerable superiority over the traditional load balancing algorithm and it remarkably improves the performance of data centre networks.

The summary of improvement is depicted in Table 1 that records the average throughput, average loss for the traditional and the proposed algorithm. In addition, it shows the amount of improvements, whereas, there was up to 81% improvement in the loss rate. Throughput improvements hit 16% on average (it is calculated from the time of joining a new host until the end of the simulation time), obviously, this percentage could be increased remarkably by increasing the emulation time as the throughput of the proposed algorithm will reach a maximum (expected).

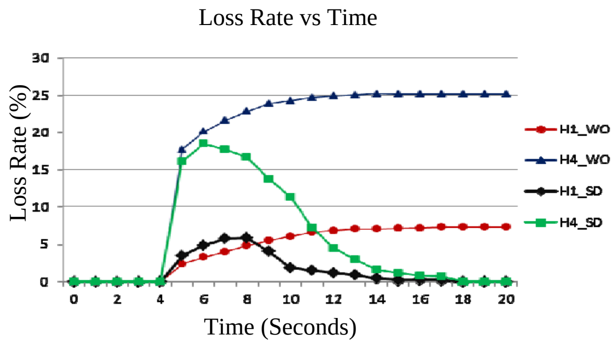

Figure (16): Scenario B, loss rate comparison between the traditional scheme and the adaptive load balancing scheme for H1 and H4.

Table (1): Summary of loss and throughput results for Scenario B

	H1	H4	H8	H12
Avg. Loss Traditional	6.031	23.8270	14.5763	25.7851
Avg. Throughput Traditional Mbps	5.58186	4.45845	4.19199	5.66897
Avg. Loss SDN (%)	1.87940	7.017	2.76798	10.6599
Avg. Throughput SDN (Mbps)	5.82659	5.34617	4.77856	6.64272
Loss Improvement	68.84%	70.548%	81.010%	58.658%
Throughput Improvement	4.2003%	16.604%	12.275%	14.659%

V. CONCLUSION

This paper proposes a new mechanism to conduct load balancing for data center networks in order to improve their efficiency. To obtain realistic and reliable results, specific kind of network topology was chosen, the one that is most utilized topology in data centers which is called Fat-tree network topology. Fat-Tree network topology was utilized with 4 pods, 8 edge OpenFlow Switches, 8 aggregation OpenFlow switches, 16 hosts, 4 core OpenFlow switches and a controller. The proposed algorithm suggests utilizing SDN technique for the purpose of load balancing in order to maintain a minimum loss and maximum throughput. For the evaluation purpose, the most reliable SDN emulator was utilized which is called Mininet emulator with Miniedit GUI tool. Two scenarios were emulated; the scenarios were chosen carefully in order to cover all the expected cases and the result in both of them was that the proposed scheme showed a remarkable improvement over the traditional scheme. Whereas, for the first scenario, the proposed scheme showed 0% loss rate compared to a loss rate that ranged from 15% to 34% when using the traditional scheme, whereas in the second scenario, the proposed scheme showed a loss rate improvement that ranges between 58% and 81% depending on the amount of contending traffic and the additional traffic beyond links' capacity.

In term of throughput, hosts utilizing the proposed scheme maintained the same level of throughput without any degradation when new flows joined the network and added additional traffic in the first scenario. On the other hand, hosts that utilizing the traditional scheme underwent from a remarkable reduction in their throughput, the overall reduction in the throughput hits more than 5Mbps, whereas for the second scenario, the proposed scheme outperforms the traditional mechanism, whereas the improvement in throughput recorded amounts that range between 4.2% and 16.6%.

In general, this paper suggests utilizing\deploying SDN networks for designing data center network in order to improve their performance. Taken into consideration that OpenFlow devices are already widly available in the market and many data center networks are using it as a network switching fabric, therefore, the proposed scheme is ready for implementation in such networks. In addition, the proposed algorithm is simple to implement and support more flexibility to the data center network.

REFERENCES

Alexander Craig, Biswajit Nandy, Ioannis Lambadaris, "Load Balancing for Multicast Traffic in SDN using Real-Time Link Cost Modification, " IEEE ICC-Next Generation Network Symposium, 2015.

Andreas Blenk, Arsany Basta, Martin Reisslein, Wolfgang Kellerer, "Survey on Network Virtualization Hypervisors for Software Defined Networking", IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 655-685, Firstquarter 2016.

Dan Li, Yunfei Shang, Wu He, and Congjie Chen, "Greening Data Center Network with Software Defined Exclusive Routing", IEEE Transaction on Computers, Vol. 64, No. 9, pp. 2534-2544, 2015.

Faris Keti, and Shavan Askar "Emulation of Software Defined Networks Using Mininet in Different Simulation Environments. " IEEE International Conference on Intelligent Systems, Modelling and Simulation, Malizia, 2015.

Feilong Tang, Laurence T. Yang, Cang Tang, Jie Li, Minyi Guo, "A Dynamical and Load-Balanced Flow Scheduling Approach for Big Data Centers in Clouds", IEEE Transactions on Cloud Computing , Vol. 99, pp.1-14, 2016.

George Kornaros, T. Orphanoudakis and N. Zervos, "An efficient implementation of fair load balancing over multi-CPU SOC architectures", Symposium on Digital System Design, pp. 197-203, Belek-Antalya, Turkey, 2003.

Heller Brandon, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick Mckeown, "Elastic Tree: Saving Energy in Data Center Networks", Proceeding of the 7th USENIX Conference on networked systems design and implementation, San Jose, California, USA, 2010.

Ian F. Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, Wu Chou, "Research challenges for traffic engineering in software defined networks", IEEE Network, vol. 30, no. 3, pp. 52-58, May-June 2016.

Iperf, https://iperf.fr/. Accessed in August 2016

James F. Kurose, Keith W. Ross, "Computer Networking: A Top-Down Approach", 6th Edition, Pearson, 2012.

Jayaram Mudigonda, Praveen Yalagandula, Mohammad Al-Fres, Jeffrey Mogul, "SPAIN: COTS data-center ethernet for multipathing over arbitrary topologies", 7th USENIX Symposium on Networked Systems Design and Implementation, 2010.

Jun Duan, Yuanyuan Yang, "Placement and performance Analysis of Virtual Multicast

Networks in Fat-Tree Data Center Networks", IEEE Transactions on Parallel and Distributed Systems, Vol. 99 , No. 1, pp. 1-14, Janurary 2016.

Liming Wang, and Gang Lu, "The dynamic sub-topology load balancing algorithm for data center networks", International Conference on Information Networking (ICOIN 2016), Kota Kinabalu, pp. 268-273, 2016.

Mao Qilin and Shen Weikang, "A Load Balancing Method Based on SDN, " IEEE International Conference on Measuring Technology and Mechatronics Automation, China, 2015.

Mininet. http://mininet.org. Accessed in August 2016

Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, Amin Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks", Networked Systems Design and Implementation (NSDI 2010) Symposium on Networked Systems Design and Implementation, San Jose, California, USA, 2010.

ONF TS-025, "OpenFlow Switch Specification", Open Networking Foundation, Version 1.5.1, March 2015.

Senthil Ganesh N, and Ranjani S., "Dynamic Load Balancing using Software Defined Networks", International Journal of Computer Applications (0975-8887), 2015.

Shavan Askar, Georgios Zervas, David K. Hunter, Dimitra Simeonidou, "A Novel Ingress Node Desgin for Video Streaming over Optical Burst Switching Networks", Optics Express Journal, Vol. 19, No. 26, B191-B196, 2011.

Shavan Askar, Georgios Zervas, David K. Hunter, Dimitra Simeonidou, "Evaluation of Classified Cloning Scheme with Self-similar Traffic", 3rd International Conference on Computer Science and Electronic Engineering (CEEC 2011), pp. 23-28, 2011.

Shavan Askar, Georgios Zervas, David K. Hunter, Dimitra Simeonidou, "Service Differentiation for Video Applications over OBS Networks", 16th European Conference on Networks and Optical Communications (NOC 11), pp.200-203, Newcastle upon Tyne, UK, 2011.

Shubhi Prashant Shukla, "Comparative Analysis of Distance Vector Routing & Link State Protocols", International Journal of Innovative Research in Computer and Communication Engineering, Vol. 3, No. 10, pp. 9533-9539, October 2015.

Sixto Ortiz, "Software-defined networking: On the verge of a breakthrough?", IEEE Computer Society, Vol. 46, No. 7, pp. 10-12, July 2013.

Wang Yong, T. Xiaoling, H. Qian and K. Yuwen, "A Dynamic Load Balancing Method of Cloud-Center Based on SDN". in China Communications, vol. 13, no. 2, pp. 130-137, Feb. 2016.

Wei Wang, Yi Sun, Kave Salamatian, and Zhongcheng Li, "Adaptive Path Isolation for Elephant and Mice Flows by Exploiting Path Diversity in Datacenters", IEEE Transaction on Network and Service Management, Vol. 13, No. 1, pp. 5-18, March 2016.

Xuan-Nam Nguyen, Damien Saucez, Chadi Barakat, and Thierry Turletti, "Rules Placement Problem in OpenFlow Networks: A Survey," IEEE Communications Surveys & Tutorials, Vol. 18, No. 2, pp. 1273-1286, Secondquarter 2016.

Yang Peng, Kai Chen, Guohui Wang, "Towards Comprehensive Traffic Forecasting in Cloud Computing: Design and Application", IEEE/ACM Transactions on Networking, Vol. 24, No. 4, pp. 2210-2222, August 2016.

Yi-chih Lei, Kuochen Wang, Yi-Huai Hsu , "Multipath Routing in SDN-based Data Center Networks, " IEEE European Conference on Networks and Communications , Paris, 2015.

Zhaogang Shu; Jiafu Wan, Jiaxiang Lin, "Traffic Engineering in Software-Defined-Networking: Measurement and Management", IEEE Access, Vol. 4, pp. 3246-3256, 2016.

Zhiyang Guo, and Yuanyuan Yang, "On Nonblocking Multicast Fat-Tree Data Center Networks with Server Redundancy", IEEE Transactions on Computers, Vol. 64, No. 4, pp. 1058-1073, April 2015.

کورتیا لێکولینێ:

رێکەکا نویێ یا هه فسه نگاندنا گرانییێ یا گونجاییی بو داتا سەنتەر هاته پێشنیارکرن د دڤێ ڤەکولینێدا . کو ئەوژی ب رێکا مفا وەرگرتنی ژ کەرکتارێن تورێین ییت هاتێنە بیناسەکرن بریکا سوفتویرد. مینینێت یا هاتیە بکارینان بو ڤی ئەگەرێ کو ئەوژی ژبو ریپرودیوسکرنی و دیفچوونا دانانا دیزانی. هەروەسا مینیدێت یا هاتیە بکارئێنان وەك ئامیرێ (جی –یو –ئای). بو بدستفه ینانا ژینگەهەکا وەکهەڤ بو سەنتەرێ داتا یی تورا یێنترنیتێ (فات – تری– تیپولوجی) هاتیە ب کارئێنان. ئەو دیزاینێ هاتیە بکارئێنان دیاردکەت "ئەوت پیرفورمانس" بەرامبەر سکیمێ کلاسیك ئانکو یا " ترادشن" کو ئەو ژی ژلایی بەرهەمینانێ ، و لەزاتییێ ، و هەروەسا رێژا دست چوونا هەمی سیناریویا یێت هاتینە هەڤسەنگاندن. دوو سیناریو یێت هاتینە ب کارێینان : ئەو ریکا هاتیە پێشنیارکرن پیرفورمانسەکێ ئەنجام دەت بێ هێچ دەست چوون بەرامبەر 15٪ تا 31٪ ژ رێژا دەست چوونێ د دەمێ سکێمێ " ترادشن" دهێتە بکارئێنان ل سیناریوییێ ئێکێ. ئەو ریکا هاتیە پێشنیارکرن باشتر ئەنجامددەت ژ ریژا دەست جوونێ ب ریژا کو دگەهیتە 81٪ ل سیناریوییێ دووێ . سەبارەت بەرهەمینانێ، ئەو ئالگوریسما هاتیە پێشنیارکرن رێژا بەرهەمینانێ پاراست ل سیناریویی ئیکێ دا بەرامبەر کیمبوونا نیزیکێ(5) میگا بت هەر چرکەکێ دا یێ بەرهەمینانێ د سکیمێ ترادشن. ژ لایەك دیڤە، د سیناریوییێ دووی دا بەرهەمینانەك باشتر دیاردکەت ب ریکا سکیمێ هاتیە پیشنیارکرن کو ئەژی نزیکی 16.6 توماردکەت لسەر سکیمێ ترادشن.

کلیلا لێکولینێ: *تورا نیاسنا پروگرامی ، داتا سەنتەر، بوکس کنترول ر ، تورا فات تری ، مینی نێت ، مینی ایدت، هەفسەنگادنا گرانییێ.*

طريقة موازنة الحمل التكييفية لمراكز البيانات بأستخدام الشبكات المعرفة برمجيا

الخلاصة:

طريقة موازنة حمل تكييفية جديدة لمراكز البيانات اقترحت في هذا البحث بأستخدام خصائص الشبكات المعرفة برمجيا. تم استخدام مينينيت لغرض محاكاة وتقييم التصميم المقترح، ميني ادت استخدم كواجهة المستخدم الرسومية. لغرض استحصال بيئة مشابهة لمركز البيانات، بنية فات تري استخدمت. تم تطبيق سيناريوهات وتوزيع احمال مختلفة لغرض تغطية اكبر عدد ممكن من الاحتمالات للأحمال الحقيقية. التصميم المقترح ابدى اداءا متفوقا على نظيره التقليدي من ناحية الانتاجية والخسائر لكل السيناريوهات المقيمة. تم تطبيق سيناريوهين مختلفين: الطريقة المقترحة اظهرت اداء خالي من اية خسائر مقارنة ب 15٪ الى 31٪ من الخسائر عند استعمال الطريقة التقليدية للسيناريو الاول. الطريقة المقترحة اظهرت تحسين في نسبة الخسائر تصل الى 81٪ في السيناريو الثاني. اما من ناحية الانتاجية، الطريقة المقترحة حافظت على نفس مستوى الانتاجية في السيناريو الاول مقارنة بتقليل بمعدل 5 ميكا بت في الثانية في الانتاجية للطريقة التقليدية. بينما في السيناريو الثاني، الطريقة الجديدة تفوقت على الطريقة التقليدية بأظهار تحسين يصل الى 16.6٪ في قيمة الانتاجية.

الكلمات المفتاحية: *الشبكة المعرفة برمجيا، مركز البيانات، المسيطر بوكس، شبكة فات تري، ميني نيت، ميني ايدت، موازنة الحمل.*