ISSN: 0067-2904

Layer-4 Load Balancer for Flow Size Prediction with TCP/UDP Separation Using P4

Sanaa Alaa Hussein*, Mustafa Ismael Salman
Department of Computer Engineering, College of Engineering, University of Baghdad, Baghdad, Iraq

Received: 20/9/2020 Accepted: 21/11/2020

Abstract

Nowadays, datacenters become more complicated and handle many more users' requests. Custom protocols are becoming more demanded, and an advanced load balancer to distribute the requests among servers is essential to serve the users quickly and efficiently. P4 introduced a new way to manipulate all packet headers. Therefore, by making use of the P4 ability to decapsulate the transport layer header, a new algorithm of load balancing is proposed. The algorithm has three main parts. First, a TCP/UDP separation is used to separate the flows based on the network layer information about the used protocol in the transport layer. Second, a flow size prediction technique is adopted, which relies on the service port number of the transport layer. Lastly, a probing system is considered to detect and solve the failure of the link and server. The proposed load balancer enhances response time of both resources usage and packet processing of the datacenter. Also, our load balancer improves link failure detection by developing a custom probing protocol.

Keywords: P4, load balancing, FatTree

موزع احمال بالطبقة 4 بإستخدام تقنية P4 قادر على توقع حجم حزمة البيانات و التفريق بين TCP و UDP

سناء علاء حسين* , مصطفى إسماعيل سلمان
قسم هندسة الحاسبات, كلية الهندسة, جامعة بغداد, بغداد, العراق

الخلاصة

في الوقت الحاضر, مراكز البيانات اصبحت اكثر تعقيد و تعالج عدد هائل من طلبات المستخدمين. البروتوكلات الخاصة اصبحت مطلوبة بشكل اكبر, و اصبح موازن حمل البيانات المتقدم اساسي لمعالجة طلبات المستخدمين بصورة سريعة و كفوءة. تقنية P4 استحدثت طريقة جديدة لمعالجة عناوين حزم البيانات. بالإستفادة من هذه التقنية لمعالجة عنوان الطبقة الثالثة من طبقات الشبكات تم اقتراح طريقة جديدة في توزيع الاحمال. الموزع المقترح يتكون من ثلاث اجزاء رئيسية. الجزء الاول هو تفريق TCP و UDP عن طريق معلومات طبقة الشبكة. الثاني هو تفريق توقع حجم حزمة البيانات. و الاخير هو الفحص لتحسس الاربطة او الخوادم المقطوعة. موزع الاحمال المقترح حسّن استعمال الموارد المتوفرة ضمن مركز البيانات, بالاظافة على تحسين وقت الاستجابة. علاوة على ذلك, المقترح طور طريقة اكتشاف قطع الاربطة التي تربط الاجهزة داخل مركز البيانات عن طريق بناء بروتوكول جديد.

*Email: sanaa.mot@gmail.com

1 Introduction

With the increase of internet users, services, and cloud computing, the datacenters (DCs) are becoming more complicated. Switches within a DC have multiple connections between each other as well as between them and the server. These redundant links can either be used to speed up traffic flows or as a backup in a link failure case. Sometimes, without using an effective method, these links may cause flow forward confusion. It is the responsibility of the load balancers (LBs) to evenly distribute traffic on the links. Employing effective LB results in better throughput, higher response time, and better resource utilization [1]. Also, the case of servers and network devices scaling up and down must be considered in an LB implementation.

The network devices, virtually, are composed of two planes, which are data- and control- planes. The data-plane is responsible for data forwarding, while the control-plane is responsible for system configuration and management [2]. Compared to the data-plane, the control-plane cannot manipulate a traffic in line rate speed. As a result, the detection of changing links and servers in a DC is slower, causing more flow interruptions and jams [3]. On the other hand, the P4 devices have a line rate speed detection because it happens at the data-plane [4].

By adopting the P4 technology, many LBs were proposed, like Hula [5], which solves CONGA [6] memory limitation and hardware implementation problems. Hula was the first P4 LB presented. However, Hula records only the next best path. This may lead to congest the path.

Another research of P4 LB is W-ECMP [3]. It depends on the probability to choose a path rather than the best path. Although W-ECMP, compared to Hula and ECMP, improves the file completion time [7], this LB does not introduce any link failure detection and solution. Also, the end-to-end path selection may increase the transmission interruption.

An Efficient Multipath Mechanism based on the Flowlet Abstraction and P4 [8] is also a P4 LB. In this LB, the path selection is random, considering only the information of the congestion of the last route that the data was sent on. Also, the selected path is end-to-end. There are two main problems in this LB. First, it suffers from a high probability of traffic interruption due to the end-to-end route selection. Second, it misses the link failure recovery.

In this paper, a new LB is proposed based on a modified Round Robin (RR) scheduling using the P4 language to implement it within a FatTree [9] emulated DC. It uses Flow Size Prediction (FSP) technique alongside the TCP and UDP separation to distribute the flows.

The rest of this paper is organized as follows. Section 2 gives a brief description of the emulation tools and FatTree topology. In section 3, the implementation and methodology of the proposed LB is described. In section 4, the routing mechanism is explained, while section 5 discusses the results.

2 System Design and Implementation

This work is designed and implemented using the Bmv2 switch simulator based on a P4 technology. Each P4 switch, in addition to its main functions, acts as an LB. Accordingly, the proposed LB is a hop-by-hop route selection.

The proposed LB is based on the typical RR load balancer, with three main enhancements, as shown in Figure-1. These are TCP/UDP separation, FSP technique, and probing. They improved the LB such that the evaluation metrics have been boosted.

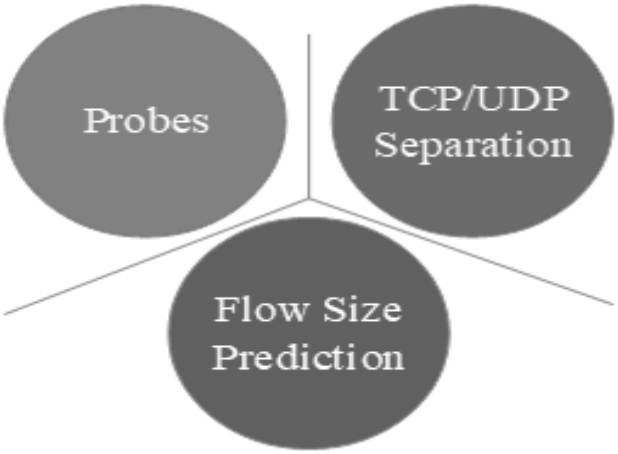

Figure 1- The LB Main Enhancements

2.1 TCP/UDP Separation

It is known that each flow from a source to a destination is segmented into small segments then encapsulated with headers, according to the Open Systems Interconnection (OSI) reference model [10]. The header of the network layer (layer 3) has multiple fields, one of them indicates whether the packet belongs to a TCP or UDP flow. For IPv4 encapsulation, the field is called "Protocol".

The protocol field contains numbers that range from 0 to 255. These numbers represent the upper layer protocol, which is the transport layer. Thus, number "6" states that the packet belongs to a TCP flow, whereas number "17" indicates that it belongs to a UDP flow [10].

By combining the information of the network layer header with the P4 ability to parse header fields, it can be possible to distinguish between the TCP and UDP flows. Hence, the combination technique is adopted in this work. The TCP flows are directed to the leftmost switch of the core switches of the FatTree DC, then to the next left switch, and so on, as in a RR fashion. Similarly, the UDP flows are directed to the rightmost switch. This TCP/UDP separation process is done in each switches level, that is, in the core, aggregation and edge levels. This separation may reduce the time overhead to choose the best next route, especially when the DC receives a few number of requests. Rather than checking a switch and turning to the next if it is founded to be busy with an elephant flow, and so on, there is a chance that the switches from the other sides are lightly loaded.

2.2 The Probes: Link Failure Detection

After the routing direction is decided by the separation technique and before proceeding with the FSP, the switch that has the RR turn must be checked as to whether it has a connectivity with an active server or not. This is achieved by probes. The probes are special frames that serve in advertising the active servers and switches. In this work, the probes are divided into two types, namely servers probe (SP), and switches probe (WP). The SP is used by the servers to announce their activities and to inform the connected edge switch about the server CPU utilization. The WP is used by the switches to announce their activities and whether they have a route to a functional server or not. The SP and WP together are used for link and server failure detection.

The header of the ethernet frame, which is a layer 2 protocol, consists of three fields: source MAC address, destination MAC address, and the Ether Type. The third field identifies the protocol that was used to encapsulate the payload of the frame.

By using Scapy [11], a new protocol is produced in this proposal, called the probing protocol. The protocol has two Ether Types, which are the 0x0111 for SP, and 0x0112 for WP. These type numbers are custom and not reserved. Also, the probing protocol has only a one-byte header and no payload, and it does not require passing through the upper OSI layers. Accordingly, this protocol prevents time-wasting, that could result from decapsulating all encapsulation headers of the layers, and reduces processing resources consumption. The header of the probing protocol is defined by P4 into the switches. This way, whenever a probe frame is received by a switch, it can be manipulated and parsed correctly.

2.2.1 Servers Probe (SP)

Every physical server within the proposed DC has two virtual servers: one for service port number 80 and the other for service port number 554, as shown in Figure-2.

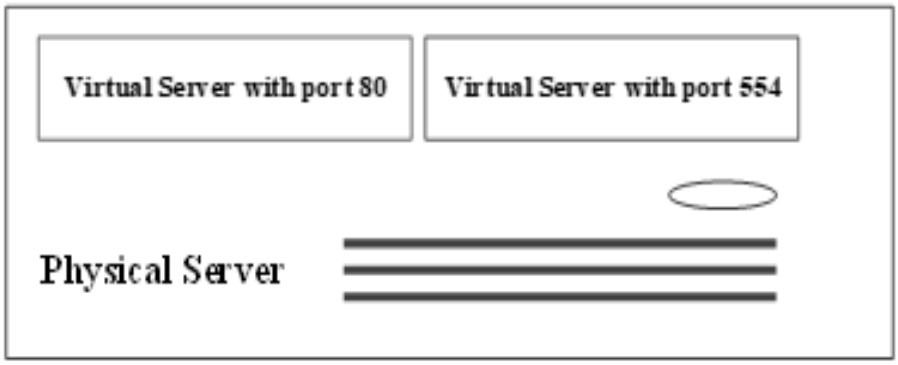

Figure 2- A Physical Server with Two Virtual Servers

The SP is generated by the physical servers. It is a single frame in which the payload is encapsulated by the probing protocol. The SP is generated every 10ms then sent to the edge switch that is connected to its server. For the SP, the one-byte header of the probing protocol is used to represent the CPU utilization of the physical server.

By the P4 code running on the switches, after an edge switch received a frame, it checks the Ether Type. If the Ether Type is 0x011, the frame is identified as an SP frame. To it, the SP frame is passed through the corresponding parser written in P4.

2.2.2 Switches Probe (WP)

By the time of an edge switch received an SP, it will generate the second type of probes, i.e. the WP. For the edge switches, they do not generate any WP if they do not have a connection with an active physical server. For the aggregation and core switches, they never generate a WP, rather they only replicate the received WPs. Therefore, not receiving a WP from a switch in the lower tier means that there is no route to an active physical server through that switch.

To prevent probes' loops within the DC, each pod is isolated from the others. Besides, the WP probes can be passed from the lower to the upper tier of switches, from aggregation switches core to switches, or from aggregation switches to edge switches. Figure-3 clarifies the probing system (considering the existence of only two servers).

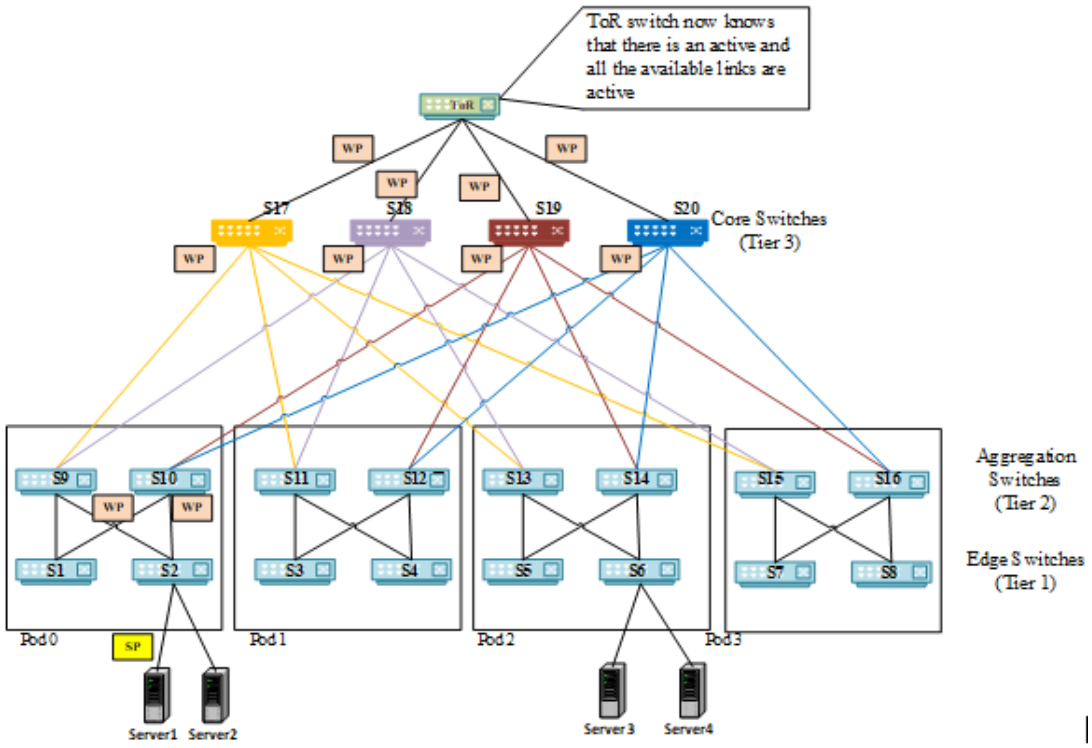

Figure 3- The Probing System for Link and Server Change Detection

2.2.3 The Probes Mechanism

In the proposed DC, every edge switch has a one-bit register for each connected server to mark the functionality of this server by setting the corresponding register to 1. Similarly, the other switches' tiers have also a one-bit register for each connected switch in the lower tier. Besides, the time threshold for a switch to wait for receiving an SP or WP is 12ms, then the switch or server will be marked as failed.

The probes mechanism can be clarified by the following procedure:

1. A physical server sends an SP to the connected edge switch every 10ms.
2. When the edge switch receives an SP from a server, the corresponding register is set to 1.
3. Whenever an edge switch receives an SP, it creates a WP. Then, it sends it to the connected aggregation switches.
4. In turn, when the aggregation switch gets a WP from an edge switch, it replicates this WP to the connected core switches. As well, the aggregation switch sets the P4 register of the corresponding edge switch to 1.

5. Consequently, the core switches replicate the WP produced from the aggregation switches to the ToR switch and set the P4 register of the sender aggregation switch to 1. At this point, the ToR switch knows the active switches that have routes to active physical servers.
6. In case the edge switch does not receive an SP from a connected physical server within 12ms, the server is stamped as failed by resetting the register to 0. Subsequently, no request will be forwarded to it.
7. If an edge switch does not receive any SP from the two connected physical servers, it will not generate a WP to send it to the connected aggregation switches. Accordingly, these aggregation switches will replicate nothing to the core switches. If a core switch receives no WP from all the connected pods, it will not send a WP to the ToR switch. In this scenario, that core switch will be marked as failed. Any the switch does not receive a WP from the connected switches in the lower tier of the switches, the corresponding registers are reset to 0.

Table-1 summarizes the types of probes that each switch tier receives and generates.

Table 1- Switch Tiers Probes

Switch Tier	Receive SP	Receive WP	Generate WP	Replicate WP
Edge	Yes	No	Yes	No
Aggregation	No	Yes	No	Yes
Core	No	Yes	No	Yes

2.3 Flow Size Prediction

After making sure that the switch chosen by the TCP/UDP separation is active, the prediction method is started to check if the switch is the best choice or the RR counter should be incremented to the next switch. In the term of networking, the flows are classified into two categories depending on the size of the data transfer. The first category is the elephant flow and the second is the mice flow [12].

An elephant flow is a large data transfer that occupies a huge amount of the bandwidth and requires relatively a long time to finish. Examples of elephant flows are video streaming and software updates. On the other hand, a mice flow is a small size flow that requires a little time to complete. Examples of mice flows are web-browsing and internet searching [13].

Each service on the internet is basically an application layer protocol that has an identifier called a service port number (SPN). The headers of the layer 4 protocols, i.e. TCP and UDP, have a field for the SPN. As a result, any layer 4 devices within the network can recognize the flow layer 7 protocol.

Since each protocol has its own SPN, then by checking the SPN of the TCP and UDP header, the layer 7 services (or protocol) flow can be identified. As a result, the flow size can be predicted depending on the SPN.

In this paper, the DC is supposed to provide a mixture of both elephant and mice flows. More specifically, each physical server has two virtual servers. The virtual servers have the port numbers of 80 and 554. Respectively, these two ports ensure that the DC provides two services, one for web paging and the other for video streaming. The port number 80 uses TCP, while 554 uses both TCP and UDP. The supported port numbers can be modified to any port numbers and extended to more port numbers by coding.

3 Routing Mechanism

When a host request a mice flow service and sends it to the ToR switch, the ToR switch forwards it directly to the next core switch using RR scheduling. In contrast, if the issued request is for an elephant flow service, the ToR switch checks the core switch, which it is in turn on the RR. If the core switch is busy with a mice flow or free of any loads, the request will be forwarded to that core switch. But, if the core switch is loaded with an elephant flow, the ToR switch will check whether the flow is marked as a timeout or not. If the flow is timeout, then the request will be forwarded to that core switch. Otherwise, the next core switch on the RR will go through the same procedure. When all the core switches are busy with elephant flows, the request will be forwarded to the next core switch on the RR.

The core and aggregation switches, as well as the ToR switch, do the same checking before routing any request to a switch in the next lower tier of switches.

When an edge switch receives a request, it will forward it to the server with the lowest CPU usage. The CPU usage of the servers is broadcasted from the servers to the edge switches. Whenever

the request is received by a server, the server will respond to it on the same path that the request was routed on. Figure-4 shows a flowchart of the complete system routing.

3.1 Flow Completion/Timeout

Before forwarding any host request, the ToR, core, and aggregation switches must check the current switch on the RR in the next tier of switches. It must be ensured that the chosen switch is either having a mice flow (or completely free) or it has a timeout elephant flow.

3.1.1 TCP Flow Timeout

For a TCP elephant flow to be tagged as a timeout flow, one of the following conditions must be met:

- Flow completion
- Flow interruption

If a switch detects that a TCP elephant flow is completed or interrupted, the switch will consider the flow as timeout. As a result, a new host request can be forwarded to this switch.

[1] Detection of TCP Flow Completion: The TCP is a connection-oriented protocol. An established connection between any two peers is maintained until one of the peers terminates the connection. The termination procedure begins by setting the FIN bit in the TCP header by a peer, and the other peer responds by setting the ACK bit. Then, the same procedure is repeated by the other peer. After that, the TCP connection is closed.

Accordingly, a TCP flow completion can be detected by checking the FIN bit. Whenever the FIN bit is found to be 1, the connection is marked as a completed flow. Subsequently, the CITO register will be set to 1. Also, the TFTO register will be set to one as well.

Note that, in this work, the acknowledgment (ACK) of the FIN is not waited to be received. In the instant that a FIN of a TCP flow is detected, a corresponding completion/interruption timeout register is set to 1. This is because the data transferring is done, and there is no need for further delay for waiting the full termination procedure to be finished.

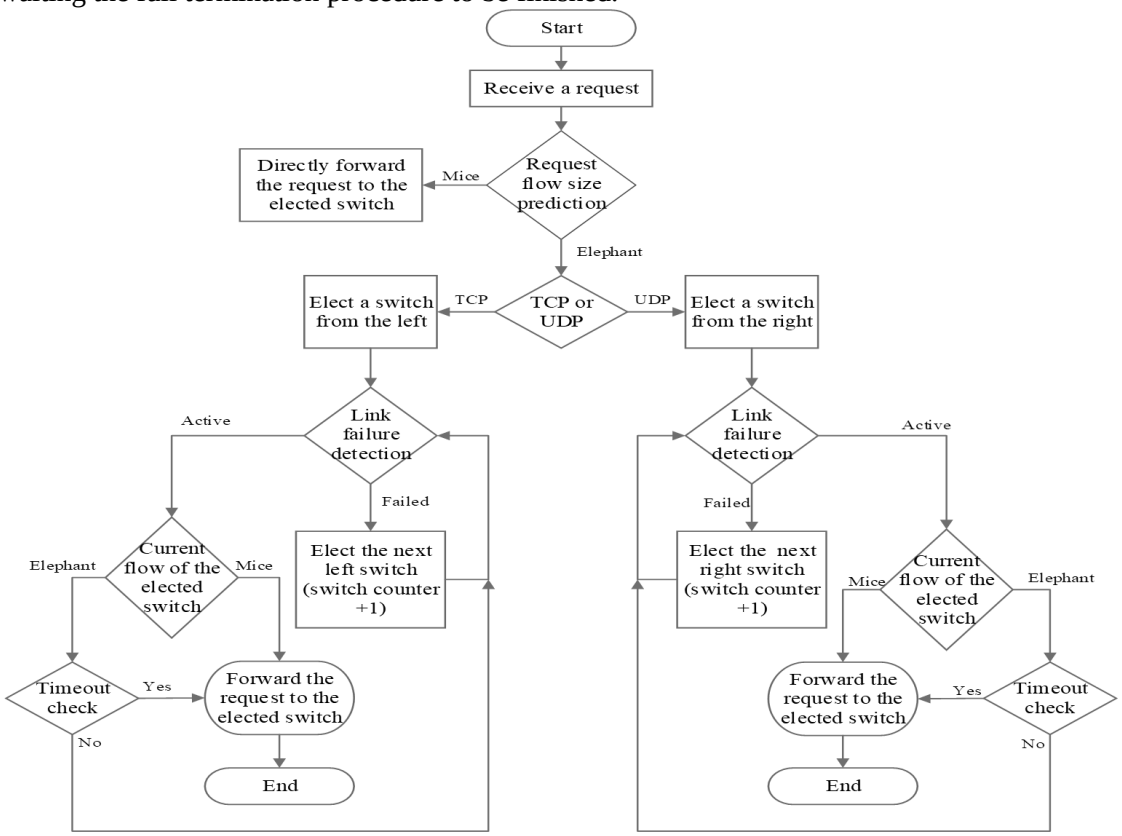

Figure 4- The Proposed System Flowchart

Moreover, updating a switch to itself to delete flow information can take a bit of time. For this reason, a completion/interruption timeout register is used. The switches in the upper tier can check the

completion/interruption timeout register to make forwarding decisions without waiting for the deletion time, which may cause a fault routing.

[2] Detection of TCP Flow Interruption: Due to some reasons, a flow may be interrupted before the completion. In such a case, the FIN bit will never occur. Therefore, the switch, where the interruption has happened, will always be found to be loaded with an active elephant flow. Furthermore, no host requests will be routed to that switch.

The interruption situation can be solved by setting a threshold time. This threshold is the interframe gap between two received frames of the same flow at the server-side. The threshold is chosen to be 115µs. Thus, when a switch receives a frame that belongs to a flow, the switch will wait for 115µs. If the threshold time was passed without receiving the next frame, then the flow is timeout, and the corresponding completion/interruption timeout register will be set to 1.

3.1.2 UDP Flow Timeout

Since the UDP is not a reliable protocol and has no header field for a connection termination information, it cannot be detected when a UDP traffic ends. For that reason, a simple method is used to announce that a UDP flow is completed. This is achieved by considering that the interruption and the completion are the same. That is, whenever the threshold of 115µs is elapsed, the UDP flow is announced as completed or interrupted. Consequently, the corresponding completion/interruption timeout will be set to 1, and the UDP flow is announced to be a timeout.

4 Results and Discussion

To implement the proposed LB, the P4 language is used [14] with its switch simulator, i.e. the behavioral-model version 2 (Bmv2). The DC FatTree network is built using the network emulator Mininet [15]. Compiling a P4 program causes the Bmv2 simulator python code to be executed in the Mininet switches. Afterward, all the switches within the network, which is created using Mininet, become P4 based switches.

For FatTree network construction, a factor called k must be known, which represents the number of ports of the switches within the FatTree. The number of core switches equals $(k/2)^2$. The network, also, has k pods. Each pod has (k/2) aggregation and edge switches. Within a pod, each aggregation switch is connected to the (k/2) core and edge switches. The edge switches, within a pod, are connected to the (k/2) aggregation switches and (k/2) servers [16]. Figure-5 shows a FatTree topology with k=4, which is the network of this work.

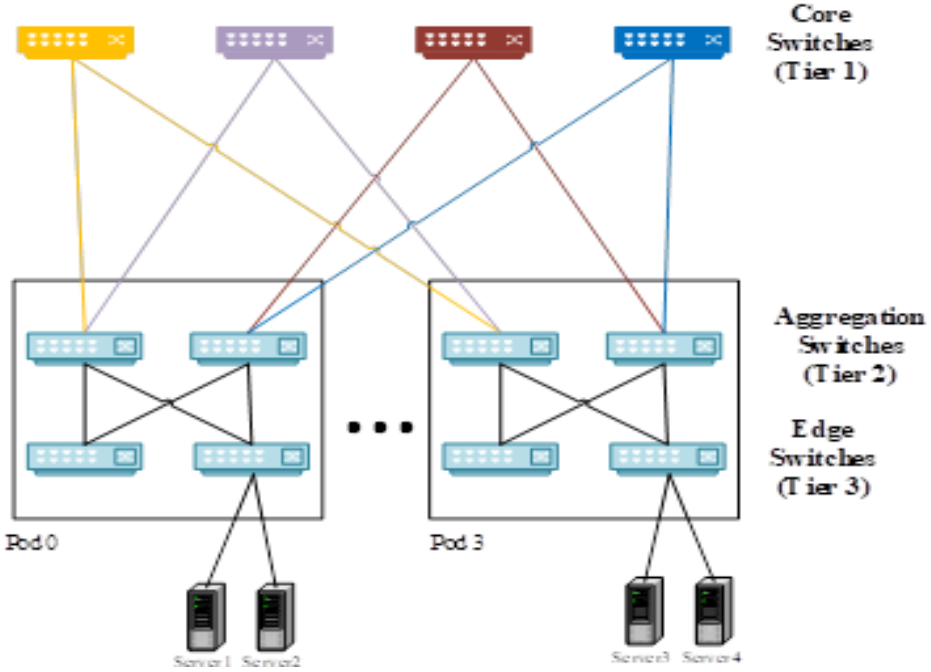

Figure 5- FatTree DC with k=4

To build the proposed LB, a virtual machine (VM) of the Linux Ubuntu operating system is created with Windows 10 as the host or physical machine. The specifications of both of the machines

are shown in Table-2. The Mininet version 2.3.0d3 is used for DC network building, and the P4 compiler version is 1.1.0-rc1.

Table 2- Physical and Virtual Machines Specifications

Machine Specification	**Physical Machine**	**Virtual Machine**
Operating System	Microsoft Windows 10 Pro 64-bit	Ubuntu 16.04.6 LTS (Xenial Xerus) 64-bit
Processor	Intel(R) Core(TM) i5-3317U CPU @ 1.70GHz ×6	Intel(R) Core(TM) i5-3317U CPU @ 1.70GHz ×2
Disk	250 GB SSD	30 GB SSD
Memory	8 GB	2 GB

The overall performance of the LB is limited by the aforementioned specifications of both the physical and the virtual machines. Those specifications achieved a bandwidth of about 1Mbps for all the links between the switches.

The probes were tested for both encapsulation and decapsulation times to show the efficiency of the proposed probing protocol. Consequently, some performance metrics were measured for this work. At first, the scalability, which shows the benefits of the probes system was measured. Then, the CPU utilization of the switches that are located within the DC was estimated. Afterward, the response time for packet processing was calculated. Furthermore, two more tests were conducted, one for throughput and the other for latency. Lastly, the Request Handling Time (RHT) was calculated to evaluate the benefits of the TCP/UDP separation technique.

4.1 Probes Evaluation

As described before, the failure detection system presented by the SP and WP is a one byte system that is encapsulated by only the Ethernet header (layer-2 header). Thus, the time of both the encapsulation and decapsulation processes is less than that time for a frame with full headers encapsulation. Figure-6 demonstrates the enhancement of the probes in terms of reducing the processing time.

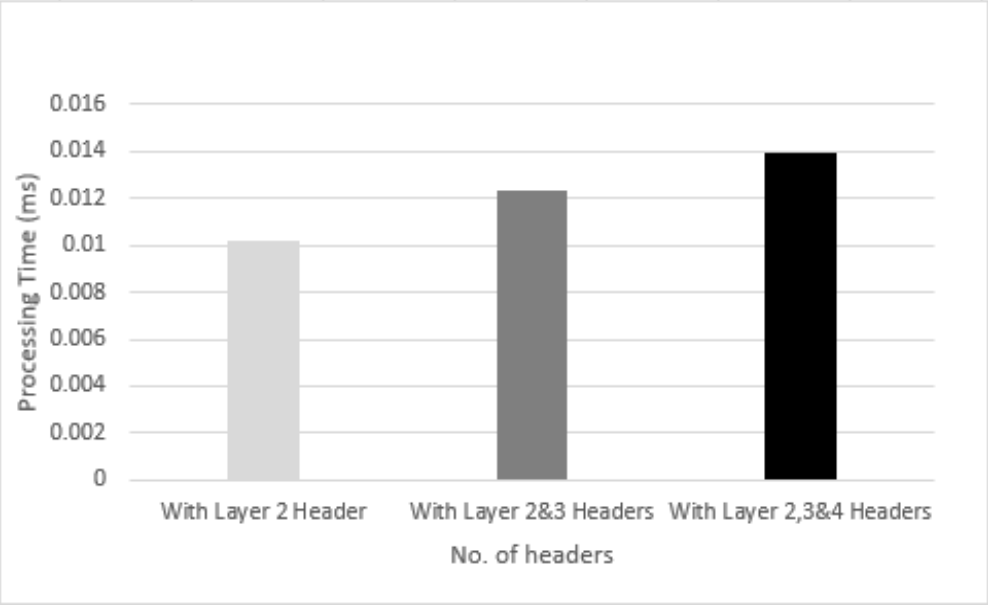

Figure 6- Probes Frames Encapsulation and Decapsulation Time

4.2 Scalability

For a DC, scalability is the ability of the DC to work in case of a device (network device or server) is scaled in or out. The scaling out may imply the failure of a device or a link between devices [1].

In the scenario of DC extension with additional switches or servers, and after the right

configuration, flows can be directed to that device. This is because the probes advertise the presence of this device to the connected switches in the upper tier.

Table-3 shows a scenario of several elephant flow requests (denoted as R_n) arrived at the ToR switch and routes. The requests must be forwarded on based on the proposed algorithm (referring to Figure-3).

Table 3- A Scenario of Some Elephant Requests and the Chosen Paths

R_n	TCP/UDP	Route
R_1	TCP	S17-S9-S2-Server3
R_2	UDP	S20-S14-S6-Server12
R_3	TCP	S18-S9-S2-Server4

The link between S17 and S9 is down, as well as the link between S6 and s14. In addition, a new server (called Serverx) is added to the edge switch S2 between Server3 and Server4. As a result, the new routes are as shown in Table-4.

Table 4- Chosen Paths After the addition of Link Failure and Serverx

R_n	Change in DC	Route
R_1	S17-S9 link is down	S18-S9-S2-Server3
R_2	S6-S14 link is down	S18-S13-S6-Server12
R_3	Serverx is added	S19-S10-S2-Serverx

Without the probing system, there is no way to failover the failed links, which will cause a connection interruption, and the newly added server will not be noticed.

4.3CPU Usage

As aforementioned, the Mininet is used to build the DC network. Since the Mininet is an emulator, each device within the DC utilizes a fraction from the CPU allocated for the VM.

The CPU usage of every switch can be recorded by monitoring the overall CPU utilization of the VM. The monitoring process showed that the average CPU usage of each switch is about 14% in the no-load case.

We considered the proposed LB works only with the TCP/UDP separation, without the FSP technique. Also, the sequence of the elephant requests shown in Table-5 is received at the ToR switch. R1 is completed just before the arrival of R5. Since the LB works without FSP, the R5 will be directed to S18, causing it to be busy with two elephant flows (R3 and R5). By proceeding with such a scenario, the S18 will be busy with more than two elephant flows.

Table 5- Requests Scenario for CPU Test

R_n	Protocol	Next Hop Switch
R1	TCP	S17
R2	UDP	S20
R3	TCP	S18
R4	UDP	S19
R5	UDP	S18/S17

Figure-7 highlights the issue of S18 processing more than one elephant flow and how would that affect the CPU usage. Also, the figure shows how the FSP technique reduces the CPU usage by reducing the number of flows on S18 to fix it to one as long as possible, or at least to the possible minimum number.

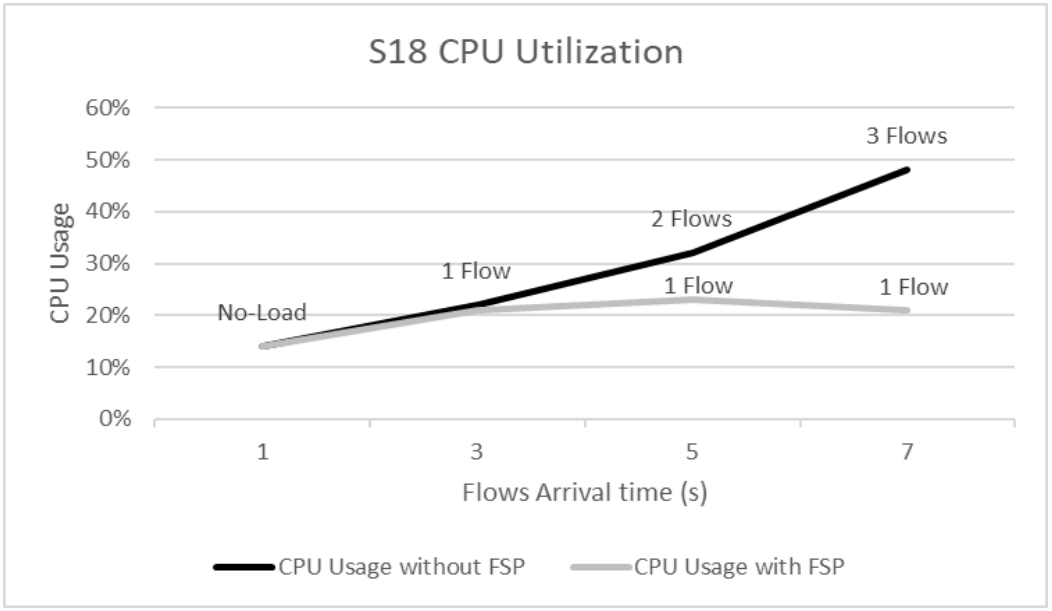

Figure 7- S18 CPU usage with and without FSP

With the absence of the prediction, and according to the TCP/UDP separation, the LB decision for R5 is to forward it to S18, which is already busy with R3. On the other hand, the FSP causes the ToR switch to check S18, and the checking results will indicate that S18 is busy with an elephant flow. Afterward, the RR counter will be incremented by one, and then the LB will choose S17 as the next hop, because S17 is not busy with an elephant flow anymore.

4.4 Packet Processing Response Time (PPRT)

When a packet is received on any of the P4 switches, the proposed LB algorithm will make the right decision in choosing the proper output port. The time from the moment that a P4 switch receives a packet to the time that it forwards this packet to an output port can be called Packet Processing Response Time. The average PPRT of each switch within the DC, when it is busy with only one flow, is 0.018µs.

The PPRT is proportional to CPU usage. The increase in CPU usage causes the PPRT to be increased. The same scenario shown in subsection 4.3 is considered here, in which the CPU utilization increases with the increase in the load on S18, when the switch operates without FSP. Shortly, a busy switch means a higher PPRT value. Figure-8 shows the response time of S18 when it operates with and without FSP.

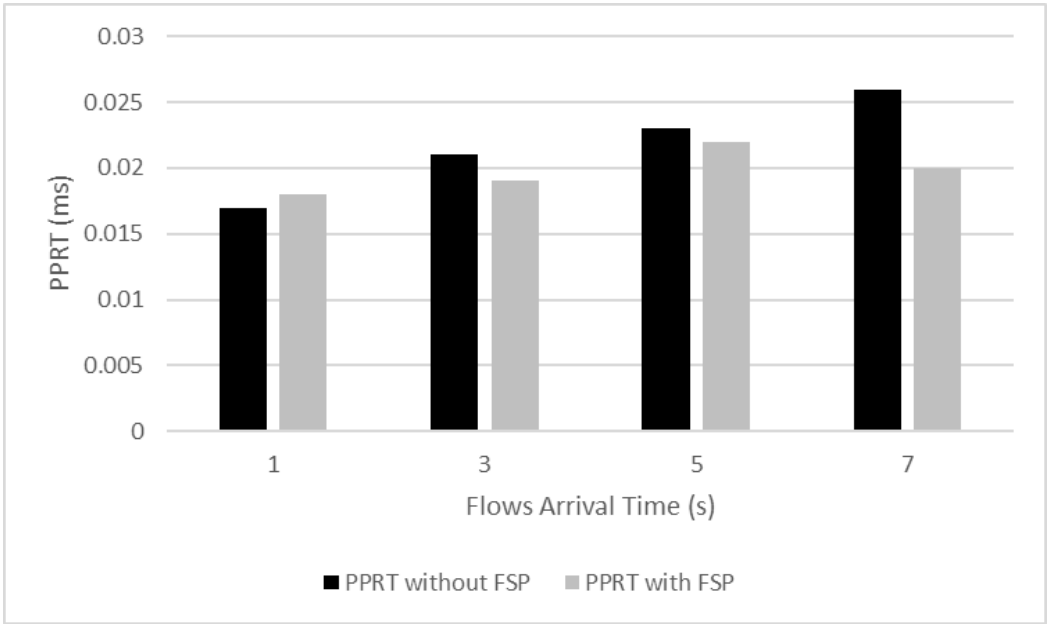

Figure 8- PPRT of S18 With and Without FSP

From Figure-8, it is clear that when S18 handles only one elephant flow, the PPRT value is much better than when it handles multiple flows. The PPRT affects not only the elephant flows, but also the mice flows. The TCP/UDP separation does not guarantee the minimum number of flows on a switch while, oppositely, FSP guarantee that. As a result, the proposed LB attempts to make a P4 switch to be busy with a minimum number of flows, which leads to lower CPU usage and, consequently, lower PPRT.

4.5 Throughput

To measure the efficiency of the FSP technique of the proposed LB, multiple requests for a video file of size 10MB are sent to the DC from many different hosts that are connected to the ToR switch. The switches of the DC choose the path to which each request is forwarded. A server will respond to a request on the same chosen path. Then, the throughput is measured to the response flow for the first request, once when the path is occupied by only the flow and another when the path is shared by multiple flows. The throughput can be measured by Equation-1.

$$Throughput = FileSize/TransmisionTime \quad \textbf{Equation 1}$$

Figure- 9 shows the throughput for the first request when the LB is working with and without FSP. From the figure, it can be concluded that the FSP technique enhanced the throughput of the flow, because the FSP technique utilizes the available bandwidth by distributing the load and placing the least possible number of elephant flows on each path.

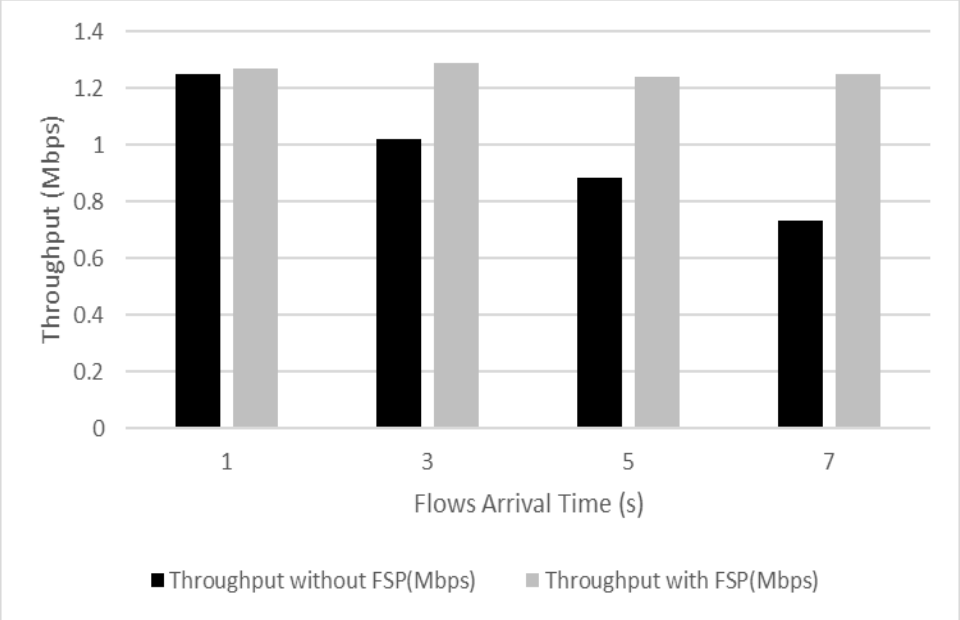

Figure 9- Throughput with and without FSP

4.6 Latency

In networking, latency is the time required for the data or request to go from the source to the destination and then return to the source [17]. This can be measured by using an edited version of the Ping command-line to evaluate the round trip time, which represents the latency. The edited version ensures that Ping can work with the service port numbers 80 and 554 in both TCP and UDP protocols.

The latency is measured by using the Ping to send a stream of packets of size 64 bytes from different hosts to the servers in two scenarios. The first scenario is integrating the LB with FSP and the other is without FSP. Figure-10 shows the latency results recorded from the test.

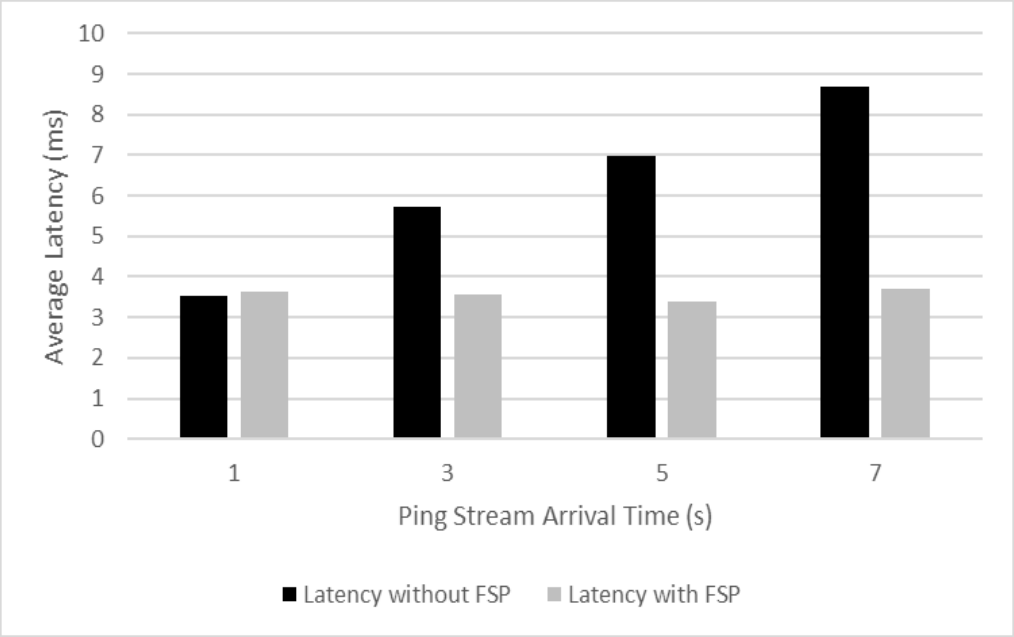

Figure 10- The Average Latency with and without FSP

4.7 Effects of TCP/UDP separation on request handling time

To demonstrate the benefits of the TCP/UDP separation in reducing the required time of forwarding a request, the RHT metric is needed to be calculated.

RHT is the required time for a switch to select the next one in a route for an incoming request. Lower RHT values represent better performance. This is because low RHT means less time in forwarding a request to the next switch.

The RHT value depends on the time required to check a switch in RR turn, i.e. whether it has an elephant flow or not, and consequently the time to increase the RR counter to the next switch if the result of the flow checking is true.

In a scenario such as that shown in Table-6, when the ToR switch receives the elephant flow requests, it must check each core switch before placing the request. At the time that R5 is received, R4 is already finished. As a result, and according to the FSP system, R5 should be forwarded to S20. With or without the TCP/UDP separation, S20 will be chosen as the next hop, but the separation saves time in choosing S20.

Table 6- Request Scenario for RHT Test

R_n	Protocol	Next Hop Switch
R1	TCP	S17
R2	TCP	S18
R3	TCP	S19
R4	TCP	S20
R5	UDP	S20

To calculate RHT, Equation-2 is used, where (S) is the time when a received request is placed at an output port of a switch after choosing the next hop, (R) is the time when the request is received at an input port of the switch, and 0.015ms is the average time of encapsulation/decapsulation.

$$RHT = S - R - 0.015ms \qquad \textbf{Equation 2}$$

As shown in Figure-11, the RHT value for R5 without the TCP/UDP separation is higher than that with TCP/UDP separation. This can be explained as follows. When R5 is received at an input port of the ToR switch, the ToR switch will check the core switch which is in turn in the RR scheduling, which is S17. Then, the checking process will start from S17. Since it is busy with R1, the ToR switch will jump to S18 and check it. S18 and S19 are both busy with R2 and R3, respectively, and as a result, S20 will be checked and chosen as the next switch because R4 was completed just before

receiving R5. In contrast, using the separation technique, and since R5 is UDP, S20 will be chosen directly. In this scenario, the consumed time is just for checking S20, without time-wasting in incrementing the RR counter to the next switch and checking other switches.

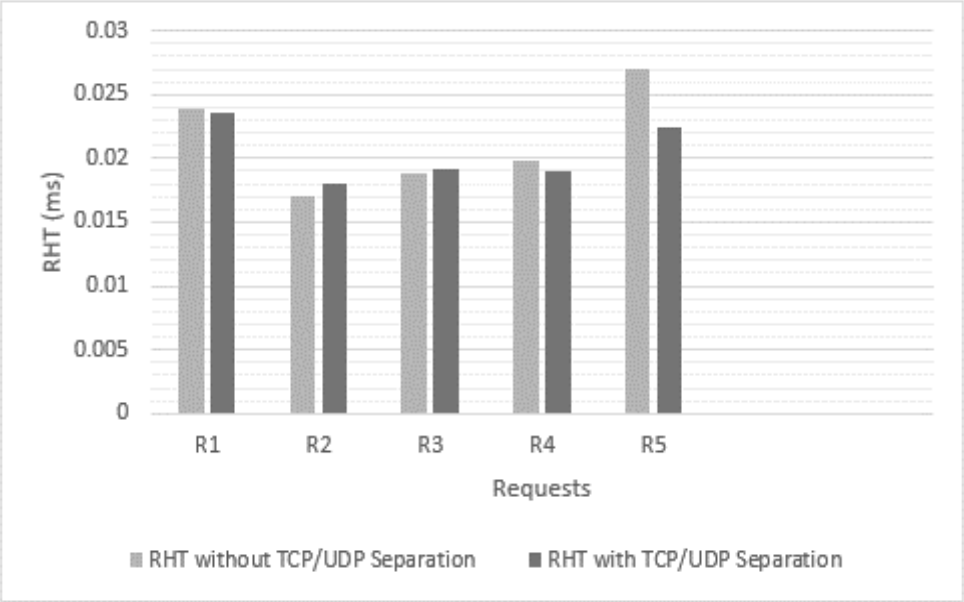

Figure 11- RHT for ToR Switch with and without TCP/UDP Separation

Despite the benefits of the FSP technique, it consumes time in checking a switch then jumping to the next. But by combining the TCP/UDP separation with FSP, the RHT can be reduced.

The TCP/UDP separation technique may give L4FSA a benefit in reducing the RHT without adding any extra load, because the time required for the separation is trivial.

5 Comparative Study

To show the improvements and enhancements of the proposed LB, it was compared to the aforementioned related works by some important factors. These factors are:

A. Link/Server Failure: Hula uses probes for link failure detection, but they cannot detect server failure. The probes of this proposal can detect both of the failures. Besides, the Hula probes are 64 bytes in size, while the proposed probes are only 15 bytes. In the work of Efficient Multipath, the failure cannot be detected.

B. Flow/Flowlet: All the discussed related works are Flowlet-based. Despite its benefit of reducing the connection interruption, it can add an overhead of splitting a flow into flowlets and choosing the right slicing time to prevent flowlets reordering at the client-side. Thus, the proposal utilizes a flow-based LB relying on the probes for fast link failover.

C. Up/Down Stream Switch Notation: All the related works are supposing a DC with up and down switch notation. Only Hula adopted the separated notation to prevent probes forever loops. The proposed work, and like the other related works, is using the combined notation, which is more real and popular.

D. Supported Topologies: This work is designed and tested in a FatTree DC. However, any other DC topology can be adopted. The related works can be run with any topology, except for W-ECMP, which requires a topology with an equal number of hopes.

E. Next Path Holding: Except for Hula and Efficient Multipath proposals, in which a switch holds only one path, the other related works, as well as this work, use a switch that can provide all the available next paths.

F. Congestion Information technique: For Hula, the congestion information is carried by the probes, while for the other works it is carried over the regular traffics. Due to the full-layers header encapsulation of the probes in Hula, the congestion may be noticed lately. On the other hand, the other works deliver the congestion information over regular traffics, which means additional unnecessary overhead due to adding extra encapsulation headers. In contrast, this proposal records the traffics in-

formation in each switch with incoming requests arrival then uses these pieces of information to determine the congested links.

The comparison between this proposal and the related works in terms of evaluation metrics cannot be accurate. This is because of two reasons: either the authors of those works used a simulator to build the network, while we emulate the network to achive realistic performance, or we could not meet the very high specification of the VM that some of them have used in their researches. In both cases, their possibly achieved results are not realistic and not comparable.

6 Conclusions

This paper introduced a P4 based load balancing algorithm for DC environment. The proposed LB has many benefits. It reduced the time of choosing the next hop by implementing the TCP/UDP separation technique. While the FSP technique enhanced the PPRT, which resulted in a faster overall traffic processing. Furthermore, the CPU usage for each switch was reduced by making the switch busy with few elephant flows. Also, the consumed bandwidth was decreased by distributing the elephant flows on all the available links. Besides, the proposed probing protocol achieved a best-effort link/server failure detection. The results of this study suggest that, by combining the P4 technology with the separation and prediction techniques, a better load balancing method can be achieved in terms of bandwidth and resource utilization. The present work opens the door for future load balancing research of classifying the traffics into mice and elephants using P4 technology.

References

1. J. Zhang, F. R. Yu, S. Wang, T. Huang, Z. Liu, and Y. Liu. **2018**. Load balancing in data center networks: A survey. *IEEE Commun. Surv. Tutorials*. **20**(3): 2324–2352. doi: 10.1109 / COMST.2018.2816042.
2. A. Dixit, F. Hao, S. Mukherjee, T. V Lakshman, and R. Kompella. **2013**. Towards an elastic distributed SDN controller. Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking. 7–12. doi: 10.1145/2491185.2491193.
3. J. Ye, C. Chen, and Y. Huang Chu. **2018**. A Weighted ECMP Load Balancing Scheme for Data Centers Using P4 Switches. IEEE 7th International Conference on Cloud Networking (CloudNet). 1–4. doi: 10.1109/CloudNet.2018.8549549.
4. P. Bosshart *et al.* **2017**. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Comput. Commun. Rev.* **44**(3)**:**87–95. doi: 10.1145/2656877.2656890.
5. N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford. **2016**. Hula: Scalable load balancing using programmable data planes. Proceedings of the Symposium on SDN Research. 1–12. doi: 10.1145/2890955.2890968.
6. M. Alizadeh *et al.* **2014**. CONGA: Distributed congestion-aware load balancing for datacenters. Proceedings of the 2014 ACM Conference on SIGCOMM. 503–514. doi: 10.1145/ 2619239 .2626316.
7. N. Dukkipati and N. McKeown. **2006**. Why flow-completion time is the right metric for congestion control. *ACM SIGCOMM Comput. Commun. Rev.* **36**(1):59–62, 2006. doi: 10.1145/ 1111322.1111336.
8. M. Pizzutti and A. E. Schaeffer-Filho. **2018**. An Efficient Multipath Mechanism Based on the Flowlet Abstraction and P4. IEEE Global Communications Conference (GLOBECOM). doi: 10.1109/GLOCOM.2018.8647887.
9. C. Gómez, F. Gilabert, M. E. Gómez, P. López, and J. Duato. **2008**. RUFT: Simplifying the fat-tree topology. Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS).153–160. doi: 10.1109/ICPADS.2008.44.
10. Y. Li, D. Li, W. Cui, and R. Zhang. **2011**. Research based on OSI model. IEEE 3rd International Conference on Communication Software and Networks. 554–557. doi: 10.1109/ ICCSN.2011.6014631.
11. "Welcome to Scapy's documentation! — Scapy 2.4.4. documentation." [Online]. Available: https://scapy.readthedocs.io/en/latest/. [Accessed: 06-Nov-2020].
12. W. Wang, Y. Sun, K. Salamatian, and Z. Li. **2016**. Adaptive path isolation for elephant and mice flows by exploiting path diversity in datacenters. *IEEE Trans. Netw. Serv. Manag.* **13**(1):5–18. doi: 10.1109/TNSM.2016.2517087.
13. K. Chandrasekaran. **2014**. *Essentials of cloud computing*. Boca Raton. CrC Press.

14. M. Budiu and C. Dodd. **2017**. The P416 programming language. Operating Systems Review (ACM*)*. **51**(**1**):5–14. doi: 10.1145/3139645.3139648.
15. "Mininet Overview - Mininet." [Online]. Available: http://mininet.org/overview/. [Accessed: 30-Jun-2020].
16. M. Al-Fares, A. Loukissas, and A. Vahdat. **2008**. A scalable, commodity data center network architecture. *ACM SIGCOMM Comput. Commun. Rev.* **38**(4):63–74. doi: 10.1145/1402946.1402967.
17. C. E. Werner and F. Rauscher. **2011**. Network latency analysis packet and method. *Google Patents*. https://patents.google.com/patent/US7961635B2/en.