Energy Efficient SDN Commodity Switch based Practical Flow Forwarding Method

Amer AlGhadhban and Basem Shihada
CEMSE Division
KAUST, Saudi Arabia
{amer.alghadhban, basem.shihada}@kaust.edu.sa

***Abstract*—Software Defined Network (SDN) has shown a great potential as a framework for providing fine-grained management of network resources. However, recent SDN practices suffer from over-accumulation of unhealthy flow-load. Instead, we leverage the SDN controller network view to encode the end-to-end path information into the packet address. Our solution *EncPath* significantly reduces the flow-table size and the number of control messages. Consequently, the power consumption of network switches is in orders of magnitude less than other evaluated solutions. It also provides flow management flexibility and scalability. We compare *EncPath* with single and multipath routing solutions and single path solution. Also, we operated them in proactive and reactive modes. We find that *EncPath* flow entries in core switches in a multihomed fat-tree with 144 hosts is approximately 1000 times smaller than Equal-Cost MultiPath (ECMP) and random routing. Additionally, the number of control messages to setup the network is reduced by a factor of $200\times$. This, consequently, affords data-plane and control-plane devices space to process other tasks.**

Keywords*—*OpenFlow switch; Data center; Software defined network; Flow-table size

I. Introduction

SDN framework is proposed to provide fine-grained control of networks and has demonstrated potential for mitigating several network management challenges by abstracting primary network functions into high level APIs. SDN decouples the control-plane from the data-plane. The control-plane makes destination decisions about arriving traffic and the data-plane delivers traffic to the destination [1]. Unfortunately, SDN solutions which use the OpenFlow protocol as the underlying paradigm suffer from several unexpected challenges, such as thousands of flow-entries, controller messages and unacceptable flow-setup delay [2] [3]. The challenges created by OpenFlow hinder SDN from providing fine-grained flow control of the network in current network services and middleboxes management complexity, (e.g., frequent configuration and errors) [3]. The aforementioned complexities increase the flow-table entries in the orders of magnitude. This is evinced in [4] where the authors found a 10 to 1 ratio of flows to each host. This is in agreement with previous findings on OpenFlow switch implementation, where they found that its flow-table contains 78K flow-entries [2], which is close to legacy data center routing table numbers [4]. This quantity of flow-entries requires 15 seconds to collect its flow statistics [2] and hinders the migration of flow-entries to data-plane devices to reduce the flow-setup time. Unfortunately, the current hardware switches, which provide line rate packet forwarding, have a limited amount of expensive and power hungry Ternary Content Addressable Memory (TCAM). Thus, TCAM with an average size of 1.5 Mbits, is not capable of accommodating the large amount of flow entries of current data center network [2] [5]. Moreover, the incremental engagement between fine-grained flow control and the number of flow-entries needs to be positively abstracted without losing granularity.

We propose a new solution that aims at reducing the load and the power consumption of network switches. One of our goals is to make our solution systematic and easily fabricated in hardware chips in the future. In our solution, defined as *EncPath*, we exploit the ability of a controller to get complete information of a network path before installing the flow. The path information are interpreted to a packet address. *EncPath* combines a reactive and a proactive flow-entries' installation. Being reactive at the edge devices aims at providing enough visibility to the controller to perform necessary functionalities, such as multipath routing. Being proactive aims at reducing the load on the controller and on data-plane devices. Furthermore, the controller needs only to communicate with the edge devices in response to network incidents or for security precautions.

EncPath introduces several advantages. First, it has a very small flow-table size. The number of flow-entries in every switch in our solution is linearly proportional to the number of switch ports. When the number of flow entries is reduced, TCAM deployments are positively influenced in two domains: the required size of TCAM to accommodate these flow-entries becomes small and the energy consumption of TCAM or any other deployed memory is reduced. The second advantage is that the controller preserves sufficient visibility over network flows. The third advantage of *EncPath* is that it enhances network performance. Due to most flow-entries are proactively installed and only edge devices' flow-entries are reactively added, the number of control messages is reduced. Finally, it has the flexibility to work in the network and data-link layers, and it also has the flexibility to work with multipath and single path routing algorithms.

II. Related Work

SDN faces significant challenges including: scalability, performance, robustness, and security. In this work we aim to elaborate the performance of OpenFlow switches by reducing

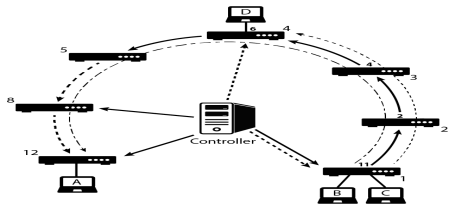


Fig. 1: Host B communicates with Host A, in 4 hops linear topology, and Host C communicates with Host D in 12 hops linear topology.

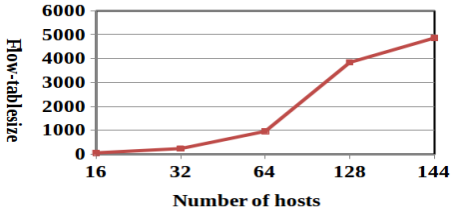


Fig. 2: Flow table size according to different number of hosts.

the number of flow-entries and the controller messages. In order to reduce the flow-table size and expedite the reading of flow-entries, two possible approaches were proposed in the literature. First, by enhancing the performance of data-plane functions or, second, using flow management with wild-card match-based techniques and flow tagging [6] [7] [2] [5]. In this section we provide a summary of both research approaches. In [6] the authors improve the performance of data-plane by using programmable packet processors together with the regular OpenFlow switch hardware. In [7] the authors proposed a reconfigurable match tables (RMT) which facilitates the implementation of programmable Match-Action processing in hardware. On the second research approach, authors in [5] propose a Tag-in-Tag scheme to reduce the flow table size. In their solution they reduce the flow table size using a method to route multiple flows through deterministic paths and they reduce the energy consumption of TCAM by 35% compared to L2 switches and 80% compared to OpenFlow switches. A similar work proposed in [8] to reduce the number of flow-entries in TCAM and relax the complexity of core network by building a label switching network. A DevoFlow in [2] was proposed to reduce the control messages, flow-setup delay as well as the use of TCAM. The authors of [9] proposed a SDN source routing algorithm to reduce state disseminations by the controller to data-plane devices in the network, in addition, the efficiency of their solution is increased as the network diameter increases. In their solution the exchange packets are embedded with a new header contains path information and each switch should read and modify the new header contents which requires a modification to existing OpenFlow switch. They aim to reduce the controller messages and provide an efficient controller placement solution. Finally, authors in [3] suggested the use of flow tagging for flow tracking and network policy enforcement. The focus herein was changing how the addresses are represented in the network and how the data-plane reads the addresses. *EncPath* reduces the flow-table size and control messages without concerning with traffic detection and classification overhead or adding new tag or header. Moreover, the controller in *EncPath* preserve enough visibility on the network flows and *EncPath* idea is simple, which makes it has a potential to be fabricated in a dedicated chip.

III. The Overhead of Flow-Entries

In this subsection we build a simple experiment to measure the number of flow-entries needed by proactive-based solutions, such as [2], to avoid the overhead of control-plane devices in flow setup, and enable network switches to use the proactive flow-entries to forward incoming traffic. However, the performance figures of reactive flow requests are evaluated in previous studies, such as [2] [10].

We build our investigation on an OpenFlow executed on Openvswitch 1.10 in mininet which is installed on top of Intel Xeon CPU X5550 2.67GHz 16 cores with 48GB memory and the latest version of POX controller. The POX controller and mininet are both installed in the same machine. In this experiment we build a fat-tree topology by using Ripcord [11] and Riplpox [12]. Since, ECMP is usually used by proactive based solutions [2], we adopted it in this experiment. We measured the number of flow-entries needed by ECMP to enable a single edge switch to directly handles the network flows without involving the controller. We start the experiment with 16 hosts and stop at 144 hosts. The results are shown in Fig. 2. We can see how the number of ECMP flow-entries are increased exponentially with the increase in the number of hosts.

On the other hand, we measured the effect of different sizes of flow-tables on the flow status request delay. We start the experiment with 1000 until 100,000 flow-entries. The results of this experiments are shown in Fig. 3. The time to pull the flow is increased dramatically with the increase in the number of flow-entries. The experiment demonstrates how proactive based solutions need thousands of flow-entries to run the operated network and how these flow-entries cause an unacceptable delay to collect their flow statistics. Although, most of the proactive based solutions deploy flow sampling solutions to collect the flow statistics, in real data center where the number of hosts are in order of magnitude compared to this experiment, the sampling process will be complex and time consuming [13].

IV. EncPath Design

In this section we explain the *EncPath* algorithm as well as its main functions as presented in Algorithm 1.

A. EncPath Operation Steps

The *EncPath* functionality can be divided into two main functions: the edge devices functions and the in-path switches

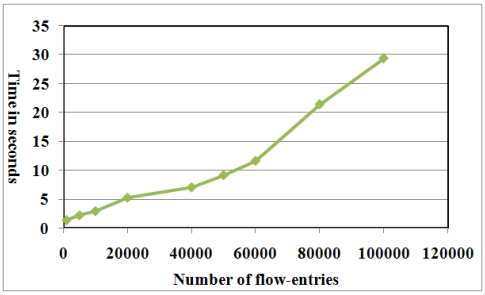


Fig. 3: The response time of flow status requests according to different flow table sizes.

functions. The edge devices are responsible in communicating with the controller to setup the flow and write outgoing port numbers into the octets of the used address, (i.e., IP or MAC address). The in-path switches are responsible in reading the IP_{TTL} value to know which address octet contains the current designated outgoing port number.

B. OpenFlow-based EncPath

In OpenFlow-based *EncPath*, the controller uses OpenFlow Discovery Protocol (OFDP), which is similar to Link-Layer Discovery Protocol (LLDP), to build a complete view of the network topology; how network switches are connected with each other and how to reach end-hosts in the network. In this work, the controller communicates with the edge devices where it is responsible in rewriting the address of every passes packet with *EncPath* address. After we investigated the OpenFlow features and specifications, we found OpenFlow (v1.1 and later versions) support arbitrary netmask in IP and MAC addresses where the ones and zeros can be inserted arbitrarily in any octets of the netmask. In this work we utilize this feature and IP_{TTL} value, as a hop counter, together to point to the right address octet that contains the outgoing port number. In Fig. 1 an example of the basic *EncPath* process wherein Host A aims to send a message to Host B. The traffic should go through port 11 in switch 1 (SW1), port 2 in switch 2 (SW2), port 4 in switch 3 (SW3), and final the out-port 6 in switch 4 (SW4). In *EncPath*, the controller encodes the outgoing port numbers into an IP or MAC address and sends two rewrite flow entries, one to SW1 to rewrites the address of outgoing packets into *EncPath* and another to SW4 to returns it back into its original address. SW1 rewrites the source IP address of the packet into srcIP = 11.2.4.6 and in this example destination IP will not be changed. SW1 forwards the packet via port 11, switch 2 via port 2 and so forth. The last switch in the path will rewrite the source IP addresses, (i.e., *EncPath* address), of the packet back to its original value.

In *EncPath*, the controller installs in each in-path switch proactive flow-entries, as displayed in Table 1, consist of the outgoing port number in the IP octet which represents the switch location in the path. SW4 forwards any incoming packet via the outgoing port that has the same value as the value in the fourth octet of the incoming packet's IP address. The TTL value is used to indicate the switch position along the path, (e.g., when the TTL=255 means the switch need to read the first octet in the IP address). Although, in previous example we used IP address octets as a container to *EncPath* information, defined here as *IPEncPath*, we also use MAC address space for *EncPath* address, defined here as *MACEncPath*, which provides a flexibility to our solution and gains even better performance.

Algorithm 1: *EncPath* execution steps

Input:{Packet Address and TTL Max_TTL=64};
if $Pkt\_IP_{TTL} = 52$ **then**
  $\_OF\_(packet)$;# Call EncPath OpenFlow
  $setIP_{TTL} = Max_{TTL}$; $return$;
else
  $ttl\_index = Packet\_IP_{TTL}\&\&255$;
  $Netmask[ttl\_index] = 255$ # creates netmask
  $OutPort = Netmask\&\&EncPath\ IP_{TTL}$ - 1#
  dec_ttl $returnOutPort$;
end

TABLE I: Flow-entries

table=0,ip,nw_dst=0.1.0.0/0.255.0.0,ttl=63,dec_ttl,output:1,
table=0,ip,nw_dst=0.2.0.0/0.255.0.0,ttl=63,dec_ttl,output:2,
table=0,ip,nw_dst=0.3.0.0/0.255.0.0,ttl=63,dec_ttl,output:3,
table=0,ip,nw_dst=0.4.0.0/0.255.0.0,ttl=63,dec_ttl,output:4,
table=0,ip,nw_dst=0.0.1.0/0.0.255.0,ttl=62,dec_ttl,output:1,
table=0,ip,nw_dst=0.0.2.0/0.0.255.0,ttl=62,dec_ttl,output:2,
table=0,ip,nw_dst=0.0.3.0/0.0.255.0,ttl=62,dec_ttl,output:3,
table=0,ip,nw_dst=0.0.4.0/0.0.255.0,ttl=62,dec_ttl,output:4,
table=0,ip,nw_dst=0.0.0.1/0.0.0.255,ttl=61,dec_ttl,output:1,
table=0,ip,nw_dst=0.0.0.2/0.0.0.255,ttl=61,dec_ttl,output:2,
table=0,ip,nw_dst=0.0.0.3/0.0.0.255,ttl=61,dec_ttl,output:3,
table=0,ip,nw_dst=0.0.0.4/0.0.0.255,ttl=61,dec_ttl,output:4,

1) EncPath Power Model: Since, the ICT consumes about 10% of global annual power and the global community struggling to reduce the carbon-dioxide emissions footprint. In addition, the number of switches in data center network is more than 10K device. When their power consumption is reduced by small fraction the total sum of their power reduction will be significant. Since, TCAM has a limited capacity, where it can not accommodate the expected flow-entries of data center network devices. We measure the energy efficiency of our solution and others based on the energy consumption of DRAM. To measure the switch power consumption of our solutions and others, (i.e., random, spanning-tree and ECMP routings), we use the model explained in the work of T. Vogelsang [14] and M. Kim et al. [15].

The power consumption of a DRAM is the sum over all charging and discharging operations of a capacitance C to voltage V multiplied with their number of occurrence, (i.e., the clock speed),f, where K is the number of flow entries that are examined to find a match. Since, this is a random number, we take its expected value $\overline{K}$ in power consumption evaluations.

$$P_{DRAM} = \sum_{i=1}^{K} \omega \Longleftrightarrow \omega = \frac{1}{2} C V^2 f \quad (1)$$

Then:

$$P_{DRAM} = \omega \overline{K} \quad (2)$$

C. Offload Flow-Setup

In our solution the edge switches are responsible for communicating with the controller and handle flow setup messages as well as rewriting flow packets with *EncPath* address. However, one of our goals is to relax all physical switches in the network, (i.e., edge, aggregation and core switches), and makes them ready to exploit the simplicity of *EncPath* solution. We test *EncPath* in the network topology shown in Fig. 4. We find that the number of flow entries is proportional to the number of the flows initiated from end hosts, as shown in Fig. 5(c). The switch power consumption increased with the increase in the flow-table entries. We aim at making its flow-entries systematic and proportional to the number of ports. The problem of relaxing edge switches is not new in the literature [2] [16] [7] [6], where they were motivated by previous studies that found that the OpenFlow switches can setup only few hundreds of flows per second [10] [2].

On the other hand, servers in a data center runs with high performance processors and high capacity memories as well as has a fast scale-up procedures not like network switches. Servers hardware prices, such as CPU and memory, are cheap compared to network devices. In order to reduce the load on edge switches and utilize these available resources we build a direct OpenFlow southbound communication between the servers and the controller. In this configuration the flow setup load on the edge switches are offloaded and spread among servers in the same subnet. The servers, in this configuration, are similar to authority switch in DIFANE [16], however, they handle only their traffic. When a server needs only to handle its traffic flows, the number of flow entries in its table are proportional to its traffic-load. Hence, the overloaded servers in the subnet will not overload the edge switch with flow setup requests and in consequence affect the other servers in the same subnet. The load of this communication on the CPU and memory are evaluated by using a testbed and the results are shown in Fig. 8.

D. Path Length Challenge

The average path length is 5 hops in data center network which is accommodated in the number of octets of source/destination IP and MAC addresses [17]. Nevertheless, *EncPath* provides scalable mechanism to avoid the long-path challenge. When the traffic is planned to be forwarded through a path has a number of hops shorter than the number of octets of used address, (i.e., IP or MAC address that used to implement *EncPath*), in this case the *EncPath* algorithm simply pads zeros into empty octets. In another case, when the path has number of hops more than 8 hops. Any type of address,

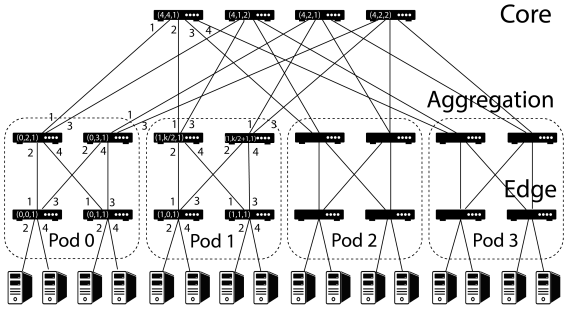


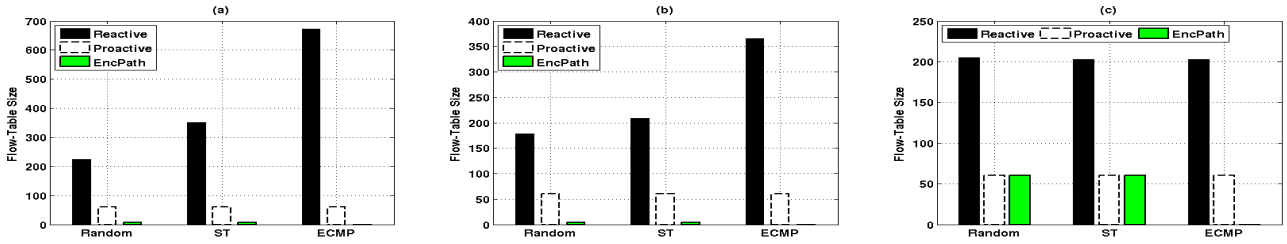
Fig. 4: Fat-Tree multihomed,k=4, topology.

(e.g., MAC and IPv6), will face similar limitations, however, we focus on IPv4 and MAC address in this work.

The controller, when it receives a new flow request, reads the host information and topology databases to discover the outgoing path, when the path is longer than 8/12 hops, at the same time the controller inserts the rewriting entries in the edge devices, it also, inserts a rewriting entry in the 8th/12th switch to again rewrites the packet address of that flow with the subsequent path information. The second challenge is how the 8th/12th switch recognizes the packets that belongs to the longer-path flow? In *EncPath*, we use the combination of source and destination MAC addresses, (i.e., in case of IP*EncPath*), TTL value and *EncPath* information as a Flow-ID which is used by the 8th/12th switch to recognize the targeted flow. For more clarification, in Fig. 1, when Host A sends traffic to Host C, the path between them consists of a number of hops greater than 8-hops. The controller instructs the 8th switch in the path to rewrite a source/destination IP address again with a new *EncPath* that contains the remaining 4-hops outgoing ports number. The 8th switch recognizes the designated flow's packets by the tuples of its MAC and *EncPath* addresses. However, our solutions is designed to be implemented in data center network where it has a structured topology and average number of hops is less than 8.

V. Validation and Results

In order to validate our solution, we use riplpox [12] based on Ripcord [11] and mininet [18] emulator. The propose solution is validated in two different scenarios to prove its robustness; three-layer homogeneous fat-tree,k=4, as shown in Fig. 4. and a linear topology with 12 switches and 4 hosts as shown in Fig. 1. Also, we built a testbed contains of 7 virtual machines to represent a single pod of the fat-tree topology and use it as an example of real implementation of our solution. In the testbed each virtual machine has an openvswitch [19] installed in it and the controller is installed in a physical machine. We select these topologies to validate *EncPath* in facing the identified challenges, like path length and flow identification, we created an Iperf flow between hosts and we dumped the flows of edges and intermediate switches, while the results are discussed in incoming subsections.



(a) Flow-table size in core switch (b) Flow-table size in aggre. switch (c) Flow-table size in edge switch

Fig. 5: Flow-table size of *EncPath* and other solutions in the multihomed fat-tree topology with 16 hosts.

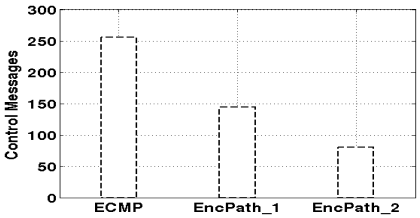


Fig. 6: The network setup load.

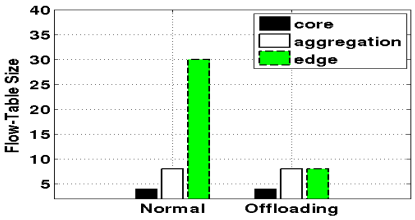


Fig. 7: The flow-table of *EncPath* in all switches.

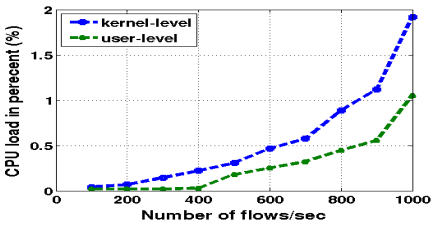


Fig. 8: The CPU load of different Nmap flows.

A. *Three-Layer Homogeneous Fat-Tree Results*

In Fat-Tree topology, we compare our solution with three different routing algorithms; ECMP, Spanning Tree (ST), and Random Routing (RN) algorithms. Furthermore, everything runs in a proactive and reactive modes.

1) EncPath Flow-Entry: We evaluate *EncPath* with other solutions by using a fat-tree with 16 hosts and the results are posted in Fig. 5. Respectively, Fig. 5(a) displays the flow-table size of one of the core switches, where the flow-table size of *EncPath* core switch contains only 4 entries and it is $15\times$ smaller than others in proactive mode and in reactive mode it outperform others by a factor of $45\times$ to $92\times$. Fig. 5(b) displays the size of flow-table in aggregation switches and Fig. 5(c), shows the size of flow-table in edge switches. Additionally, we evaluate *EncPath* and the other solutions by using 144 hosts in the same topology and 33% of the hosts initiate flows between each other. We find the number of flow entries of other solutions in proactive mode reached 4,860 entry in each switch, in contrast, the number of flow entries in *EncPath* switches is as what they are when tested with 16 hosts except the edge switches which is 576 flow-entries. Although, the edge switches have flow-entries smaller than other solutions, we need to reduce it further and makes it proportional to the number of ports. This is the purpose of *edge switches offloading* technique.

2) EncPath Setup: EncPath needs to install proactive flow-entries, as shown in Table 1, in network switches in order to inform the in-path switches how to handle exchanged packets. In this subsection we show how many control messages are needed by *EncPath* to install these proactive messages and compare it with other solutions in proactive mode. Since the solutions in reactive mode do not need to prepare the network before installing any flow, we compare *EncPath* with other solutions in proactive mode. The solutions in *proactive mode* need 256 control messages to prepare the network and this number for only 16 hosts as shown in Fig. 6, and 20,736 control messages when we test them with 144 hosts. In contrast, *EncPath* requires 80 control messages in first configuration, when the OpenFlow communication is between edge switches and the controller, and 144 control messages in second configuration, when the OpenFlow communication are performed between the servers and the controller. Consider that *EncPath* needs 2 reactive messages to setup the end-to-end flow which is not needed for network setup.

3) EncPath Energy Efficiency: Since, the ECMP routing and other solutions in proactive mode insert large number of flow-entries to setup the network, their average power consumption in proactive mode is 2.7 times of reactive mode, as illustrated in Fig.9 and Fig.10. Due to EncPath's flow-table isn't affected by a change in the number of hosts, its flow-table as well as power consumption stay constant, unless, there is a topology change. The power consumption of EncPath switches are shown in Fig.11, which is in orders of magnitudes less than other solutions.

B. *Edge Switches Flow Offloading Results*

In this experiment we evaluate the possibility of creating an OpenFlow communication between a virtual server and a controller, also the load of this communication on the CPU and memory is measured. We aim to utilize the available resources

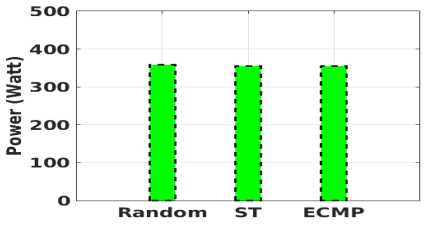


Fig. 9: The power consumption of edge switches in reactive mode.

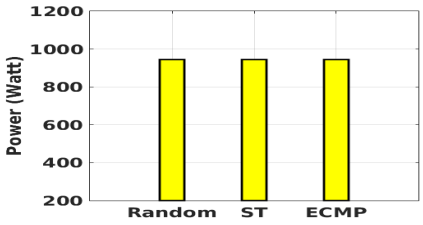


Fig. 10: The power consumption of edge switches in proactive mode.

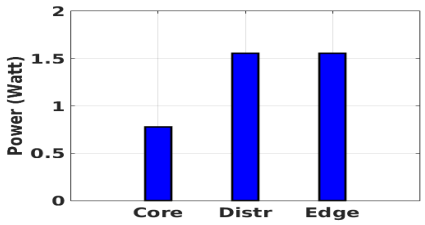


Fig. 11: The power consumption of EncPath.

and the scale-up facility of the servers to make the flow-entry in edge switches proportional to the number of ports.

To represent one pod of the fat-tree we connected 7 virtual machines which are 5 switches and 2 servers, where a controller is installed in a separated machine. The source server has a 4 cores CPU and 4GB memory. We use Nmap software in the source server to initiate several flows from 100 to 1000 flows per second, where the TCP ports being varied in order to have new flow-entry for each Nmap flow. The flow table size of the edge switches and others are shown in Fig. 7, normal case means when the OpenFlow communication is between edge switches and the controller, while offloading case means when the communication is between the servers and the controller.

On the other hand, the load of OpenFlow communication between the server and the controller on the CPU and memory are measured during Nmap flows. We find that in worst case the load on the CPU is less than 2% and the results are shown in Fig. 8, where the load on the memory oscillated between 0 to 0.3%. The same experiments are repeated to measure the effect of OpenFlow communication when it runs in kernel and user levels, the results are displayed in the same figure. Although, the performance of kernel-level is better in general, in this experiment both levels provide same TCP throughput when we evaluate them by using Iperf.

VI. EncPath Discussion

In this section, we explain the additional features and future work of *EncPath* solution. The main challenge is the path length challenge, although, there are multiple solutions to solve this challenge, we are searching for a scalable and simple solution that has an accepted impact on the operation, (i.e., in the order of few flow-entries execution time). It should be noted that this work is designed to be implemented in data center network.

A. Flexibility and Scalability

The proposed solution needs to provide some flexibility and elasticity to meet network engineers expectations. They consider the deployment of new technologies, as an intrusive solution and degrading the performance of the network operation. *EncPath* solution provides flexibility and scalability in the following domains:

Flexibility in deployment: *EncPath* has the ability to be implemented in network and in data-link layers which gives it the resilience to the variability of data center traffic needs. We measured the round trip time of IP*EncPath* and MAC*EncPath* mechanisms in a multihomed fat-tree as shown in Fig. 4 and the results are displayed in Fig. 12. In Fig. 12, the x-axis displays the two routing algorithms, multipath and single path routing, which proves the ability of *EncPath* to work as a multipath or a single-path routing solution and its flexibility with routing solutions.

Scalability and flexibility in flow-entries: since we designed *EncPath* to work in data center networks whereas the network topological structure shows the same pattern, in general, the flow-entries illustrated in Table 1 are enough for all switches to handle all passing traffic. Sometimes the number of ports in one switch or more in the same topological layer are not the same, in *EncPath* the maximum number of flow-entries among the number of ports of switches in the same layer is considered. To prove the flexibility of *EncPath* we inserted the same flow-entries in all switches in all layers, even the edge switches, and we tested our solution in the fat-tree topology, k=4. We find no change in flow-setup time and round trip time, and this because the flow-entries that are needed by each upper-layer switch to setup the network are 4 to core switches and 8 for aggregation switches.

Flexibility in flow-entries: for different reasons, such as link failure and congestion avoidance, the traffic needs to go through unplanned routes, i.e., non-proactively set-up route. For example, in Fig. 13, the traffic is planned to go through Path#1 and Path#2 and the switches are pro-actively prepared to forward incoming traffic through these paths, however, if the links between SW4 and SW6 is overloaded or goes down, the traffic is better to go through the link in-between SW4 and

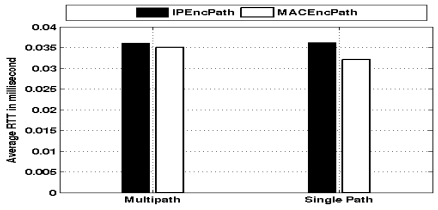


Fig. 12: *EncPath* implemented as IP and MAC-layers solution in multipath and single path routing.

SW5 than forward all the traffic via Path#1. Unfortunately, in previous work, such as ECMP, even though the controller knows the existence of alternative paths and have them in their table, the path need to be built a prior and its information are pro-actively inserted on the switches before it can be used to forward traffic through it. In contrast, in our solution the pro-actively inserted information are enough to use any path in the network when the controller knows about it and has it in his table. In EncPath the controller needs only to encode the required path in the packet header by instructing the source edge switch, SW1 in the figure, to utilize that link and forward the traffic through it. In this work we aren't proposing a link-failure solution, however, we are showing the flexibility of our solution in changing the flow direction for any reason and we use the link failure as an example.

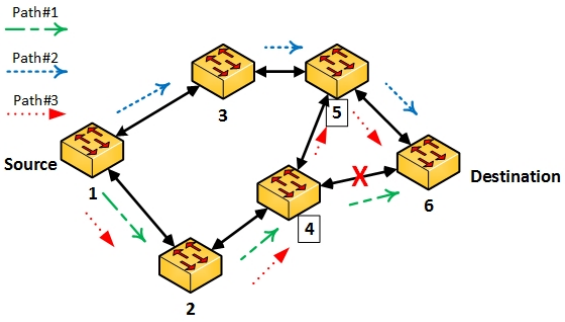


Fig. 13: Three possible paths two equal-cost (path#1 and path#2) and one unequal-cost path (path#3).

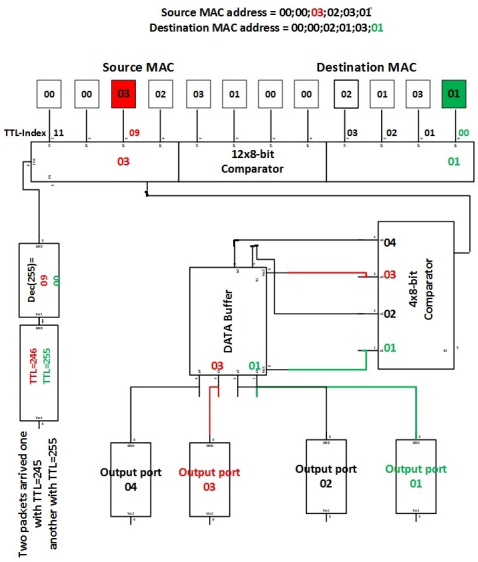


Fig. 14: The overlay hardware chip of *EncPath*.

B. Hardware Chip Design as a Future Work

In this section we show how *EncPath* can be implemented in hardware chip. The pseudo code in Algorithm 1 explains the main steps that are going to be considered in the hardware chip design. In first step the algorithm reads the TTL value to indicate which address octets contains the outgoing port value. then, the algorithm reads the value inside the indicated octets, then and it with wild-card mask to get the outgoing port. Finally, sends the packets through that outgoing port and decrement the TTL value. Fig. 14, displays our overlay hardware chip design of *EncPath*. The diagram shows how *EncPath* handles two packets from two different flows, one with a source MAC address (00:00:02:01:03:01) with TTL = 255 which means the first octet is going to be read by the chip and considered as an output port. The value of first octet is 01 which means the output port is 01. The destination MAC address is (00:00:03:02:03:01) with TTL=249 which means the output port is the value of octet number 09 which is 03. This chip gets the source and destination MAC address as one input and the TTL value as another input, where the TTL value works as an index indicating which octet the chip needs to read to select the output port. We assume the hardware solution will be implemented in a data center network where the maximum path length is less than 13 hops. Overall, out of this assumption when the path length is longer than 13 hops and the packet reaches the switch number 13, identified by the TTL value, it will be dropped or forwarded to the controller as an unknown flow. It should be noted the path length challenge is not new in the literature the RIPv1 has a limited number of hops = 15.

VII. Conclusion

In this work, we introduced a novel solution, named as *EncPath*, that reduces the flow-entries in OpenFlow switches to a number close to its number of ports. The flow path port numbers are interpreted as an IP or MAC address. In this case, the network switches need only to read the right address octet to understand where to forward the packet. To achieve this, few flow-entries are proactively inserted in the switches to indicate to the right address. To this extend, OpenFlow switches need only few flow-entries to read the right address octet. When flow entries decreased, the number of control messages as well as the time to pull flow statistics when it is needed are decreased, as well. *EncPath* solution provides flexibility and scalability in several areas of deployment; network layers, routing mode and changing the path direction. Also, its idea is simple, which makes it has a potential to be fabricated in a dedicated hardware chip. The performance figures of *EncPath* is evaluated in detailed set of experiments with different configurations and scenarios including: linear topology and multihomed fat-tree, k=4, topology as well as a testbed of 7 virtual machines. We find that *EncPath* outperforms other tested solutions in the number of flow-entries, control messages and throughput.

References

[1] A. Lara et al, "Network innovation using openflow: A survey," *Communications Surveys Tutorials, IEEE*, vol. 16, no. 1, pp. 493–512, First 2014.

[2] A. R. Curtis et al, "Devoflow: Scaling flow management for high-performance networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11, 2011, pp. 254–265.

[3] S. K. Fayazbakhsh et al., "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14, 2014, pp. 533–546.

[4] S. Kandula et al., "The nature of data center traffic: Measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '09, 2009, pp. 202–208.

[5] S. Banerjee and K. Kannan, "Tag-in-tag: Efficient flow table management in sdn switches," in *Network and Service Management (CNSM), 2014 10th International Conference on*, Nov 2014, pp. 109–117.

[6] S. Zhouet al., "A flexible and scalable high-performance openflow switch on heterogeneous soc platforms," in *Performance Computing and Communications Conference (IPCCC), 2014 IEEE International*, Dec 2014, pp. 1–8.

[7] P. Bosshart et al., "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13, 2013, pp. 99–110.

[8] K. Agarwal et al., "Shadow macs: Scalable label-switching for commodity ethernet," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14, 2014, pp. 157–162.

[9] M. Soliman et al., "Source routed forwarding with software defined control, considerations and implications," in *Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop*, ser. CoNEXT Student '12, 2012, pp. 43–44.

[10] C. Rotsos et al, "Oflops: An open framework for openflow switch evaluation," in *Proceedings of the 13th International Conference on Passive and Active Measurement*, ser. PAM'12, 2012, pp. 85–95.

[11] M. Casado et al, "Ripcord: A modular platform for data center networking," EECS Department, University of California, Berkeley, Tech. Rep., Jun 2010.

[12] Riplpox. https://github.com/MurphyMc/riplpox.

[13] C. Barakat et al., "Ranking flows from sampled traffic," in *Proceedings of the 2005 ACM Conference on Emerging Network Experiment and Technology*, ser. CoNEXT '05, 2005, pp. 188–199.

[14] T. Vogelsang, "Understanding the energy consumption of dynamic random access memories," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '43, 2010, pp. 363–374.

[15] K. Minjoong et al., "A simple model for estimating power consumption of a multicore server system," vol. 9, no. 2, pp. 153–160, 2014.

[16] M. Yu et al., "Scalable flow-based networking with difane," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 351–362, Aug. 2010.

[17] J. Chabarek et al, "Networks of Tiny Switches (NoTS): In Search of Network Power Efficiency and Proportionality." in *Proc. of the 5th Workshop on Energy-Efficient Design*, 2013.

[18] N. Handigol et al, "Reproducible network experiments using container-based emulation," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12, 2012, pp. 253–264.

[19] B. Pfaff et al, "Extending networking into the virtualization layer," in *In: 8th ACM Workshop on Hot Topics inNetworks (HotNets-VIII).New YorkCity,NY(October 2009*, 2009.