Tag-And-Forward: A Source-Routing Enabled Data Plane for OpenFlow Fat-Tree Networks

Airton Ishimori, Eduardo Cerqueira, Antônio Abelém

Research Group on Computer Networks and Multimedia Communication
Federal University of Pará
Email: {airton, cerqueira, abelem}@ufpa.br

***Abstract*—Software-Defined Networking (SDN) has turned the Data Center Network (DCN) environment into a more flexible one by decoupling control plane from data plane, allowing an innovative and easily extensible network management solutions. Nowadays, OpenFlow is the most successful protocol for SDN. However, SDN based on OpenFlow protocol presents performance issues on forwarding table increasing and packet match cost. Our proposal named Tag-and-Forward (TF) is a data plane that reduces the number of flow table required in the Fat-Tree software-defined DCNs to optimize forwarding. The results noticebly outperformed RTT and packet transmission rate when compared to usual OpenFlow data plane.**

I. Introduction

Data centers have been hosting a large variety of service-oriented applications such as data storage, data processing and online business. These days, a simple web search request may touch more than 1000 servers [1]. Data storage and analysis applications interactively process petabytes of data on thousands of machines [2].

The advent of Software-Defined Networking (SDN) has turned the dynamic environment of Data Center Networks (DCNs) into a more flexible ones by decoupling control plane from data plane. Nowadays, OpenFlow is the most successfull control protocol for SDN. However, researches have pointed out that SDN based on OpenFlow protocol shows performance issues when control plane interacts with the data plane. The experiment results in [3] reports that statistics-pulling interferes in flow setup. When statistics are never pulled by the logically centralized controller, clients can make 275 connections/sec. and when they are pulled once a second, collecting counters for just under 4500 network state (forwarding entries), those clients achieve fewer than 150 connections/sec.

Moreover, the research proceeded in [4] shows that the OpenFlow programable switches can have significant differences on hardware design, such as on Ternary Content Access Memory (TCAM). This might affect network throughput over the large sets of forwarding rules. In addition, the research also presents that different vendors may have different cache replacement algorithms. Because of this, rules in the hardware table can have different TCAM configuration. For example, the highest-priority rule in the software table (flow table) can be inserted in a position with lower priority in the hardware table.

TCAM is a key component to achieve higher network throughput, but it has few memory space and is a "power hungry" element [5]. As discussed in [6], we can reduce the number of network states (forwarding rules) to decrease power consumption. According to [7], [8], the lookup operation on TCAM is the dominant factor in terms of the power consumption. Therefore, there should be approaches to make better use the flow table.

For example, the authors of FTRS (Flow Table Reduction Scheme) [9] and FFTA (Fast Flow Table Aggregation) [10] propose algorithms to reduce the flow table size. However, the compression algorithms tend not to be scalable in the dynamic environment of data centers, where the flow table updates commonly occurs at a high frequency. Others, such as GFlow [11], it makes use of the parallel processing capability of GPUs (Graphical Processor Units) to accelerate flow table lookups. However, this requires a lot of programming efforts and many of the vendors' switches do not support GPUs.

The aforementioned performance issues relate to the flow table, which indicates that the flow table is a critical point to evolve SDN scalability. In addition, in SDN there is also the flow setup delay problem. However, we can reduce the total amount of network latency if the network rely less on the flow table to forward packets. That is, the network state distribution occurs for a reduced number of switches in the network.

The proposal named Tag-And-Forward, or TF for short, is our source-routing inspired proposal for software-defined Fat-Tree DCN topology. TF limits the number of network state setup to smaller set of switches in contrast to the standard OpenFlow dataplane. This means that a smaller set of flow tables need to be updated to manage the entire course of the network flows. Hence, TF reduces the controller-switch interaction to setup forwarding paths for the network flows.

Our goals are two-folded: (a) to improve network forwarding capacity and (b) to reduce network latency. Our performance evaluation shows an improvement on network latency by roughly 60% when the flow table is overloaded at 20% of the total capacity. Moreover, the evaluation also shows an improvement around 40% on the network delivery capacity when 8K flows are sent in parallel.

The rest of the paper is organized as follows. Section II presents the related works followed by Section III that presents the TF design. Section IV presents an example of the TF working. In Section V, we presents our simulation results. Lastly, we conclude the paper in Section VI.

II. Related Works

Tag-in-Tag [7] replaces the hundreds of packet header bits by a shorter one to match against the flow table. Therefore, the flow table only stores such tag number. Tag-in-Tag is closer to the VLAN and MPLS working in which all the flow tables need to store a common label. Our proprosal uses a reduced number of flow tables.

KeyFlow [12] tag/detag a key calculated by Chinese Remainder Theorem (CRT) in the packets. The key is a code that identifies the forwarding path where switches decode such key to determine the output port. The main difference with our proposal is that KeyFlow uses flow tables on the first and latest switch of the forwarding path, respectively, to tag and detag the key. Besides, the key is generated by multiplying prime numbers successively, so the result number tend to be huge which the packet header might lack space.

Packets in SourceFlow [13] carries an index counter and list of OpenFlow actions. Although, the short paper does not provide a deeper details on the proposal, but it seems that there is a linear search cost on such list. In SiBF [14] the packets carry Bloom filters having a sequence of output ports. SiBF uses 96 bits from MAC fields to store the bloomed code. The problem is that updating 96 bits whenever the packets ingress and egress the network, such operation comes with delay cost.

III. TF Design

A. Overview

TF is inspired by the traditional source-routed networks, where the source routers attach a routing information to the packet when they are entering the network and remove such information when they are leaving the network. The routing information is computed at the source routers. However, in our proposal, the remote control plane is responsible for computing routes and building flow tables.

In our proposal, the network controller programs the flow tables with rules that intruct switches to attach TF-tag (TF header) to the packets. Basically, TF-tag has a number to implicitly indicate the output gate (network interface card) to the next hop. Once a packet is forwarded, the following switches read TF-tag and run a particular function defined in the proposal to determine the output gate. For example, if a path is formed by the sequence of switches $es1 \rightarrow as1 \rightarrow cs2 \rightarrow as7 \rightarrow es8$ to reach a server S, the source switch $es1$ associates TF-tag with the packet while the others $as1$, $cs2$, $as7$ and $es8$ forward the packet by reading the tag. That is, the source switch uses flow table while the others do not.

B. Gate Configuration Properties

OpenFlow ports are wrapped by gates. A gate can be interpreted as label for the actual OpenFlow port. In the Fat-Tree topology presented in Figure 1, the numbers beside switches are gates. It is important to keep in mind that OpenFlow ports are not replaced by gates, i.e., if an OpenFlow port is 2 and the gate is 16, both represent the same network interface card of the OpenFlow switch.

The gate configuration phase consists in enumerating network cards of switches according to the following methodology: i) the enumeration starts in the edge layer and ends in the core layer (bottom-up enumeration) and ii) the enumeration starts in the core layer and ends in the edge layer (top-down enumeration). Moreover, the enumeration process has some properties as we state in the paragraphs below.

Property 1 (P1). Given a set of switches $s_1, s_2, ..., s_{j-1}, s_j$ with j switches and m interfaces per switch $\{i_{1(m-k)}, i_{1m}\}$, $\{i_{2(m-k)}, i_{2m}\}$,..., $\{i_{(j-1)(m-k)}, i_{(j-1)m}\}$, $\{i_{j(m-k)}, i_{jm}\}$, the gates are generated by Arithmetic Progression (AP) having the initial term of the sequence an even number $g_1 > 0$ and the common difference number d among the terms of the sequence, then we have the n-th term g_n:

$$\begin{gathered} g_n = g_1 + (n-1) \times d \\ \therefore \\ AP(g_1, d, n) = \{g_1, g_2, .., g_n\} \\ \text{, where } 1 \leq n < j \times m \end{gathered} \tag{1}$$

Property 2 (P2). Having the association of gates with interfaces, where $g_1 = i_{11}, ..., g_2 = i_{(j-1)(m-k)}, ..., g_n = i_{jm}$. For each switch s_j having m interfaces, there is a set of gates $G_{s_j} = \{g_1, g_2, ..., g_m\}$, where $g_1 \neq g_2 \neq ... \neq g_m$.

Property 3 (P3). Being $G_1 = \{g_1, g_2, ..., g_n\}$ the sequence of gates of the first layer, G_2 the sequence of the second layer and G_3 the sequence of the third layer, we must have $G_1 \supset G_2 \supset G_3$. That is, G_3 must be formed by the gates of G_2, where G_2 must be formed by the gates of G_1. The gates $g_1, g_2, ..., g_n$ for G_2 and G_3 must be chosen by the relation:

$$\begin{gathered} g_n = \max\ G_{s_j} = \max\ \{g_1, g_2, ..., g_m\} = g_m \\ \text{, where } 1 \leq n < j \times m \end{gathered} \tag{2}$$

Property 4 (P4). A switch s_j having the sequence of gates $G_{s_j} = \{g_1, g_2, ..., g_m\}$, the common difference number $d = g_m - g_{(m-1)}$ must be equal among other switches in the same level. That is, $d = d_{s_1} = d_{s_2} = ... = d_{s_j}$.

The gate configuration properties apply for the bottom-up and top-down enumeration phases. In the bottom-up phase, only the m network interfaces cards of the switches that connect the current layer to the lower layer are considered, i.e., the interfaces that connect edge switches with servers. While in the top-down phase, only the m interfaces that connect switches to the upper layer are considered, i.e., the interfaces that connect core switches with the external network.

C. Forwarding Function

TF data plane process packets in Linux kernel space to forward packets. The output gate is the returned value of the function $f(g, d)$, where the parameter g is the number tagged on the packet and d is the common difference number that exist among gates. The function is defined as follow.

$$f(g,d) = \begin{cases} g, & \text{if } g \bmod d = 0 \\ \left(\lfloor \frac{g}{d} \rfloor + 1 \right) \times d, & \text{otherwise} \end{cases} \tag{3}$$

The computational complexity of $f(.)$ is $O(1)$ on average, because the parameters g and d are known and they are given to the function. Therefore, no iteration takes place in the function to increase the computational complexity.

The function $f(.)$ returns g if, and only if, g is perfectly divisible by d. Otherwise, the result of the operation $\left(\lfloor \frac{g}{d} \rfloor + 1\right) \times d$ is returned. In both cases, the number returned by $f(.)$ is the output gate through where the received packet should be forwarded. This function is only used by switches that do not rely on flow table during the forwarding process. That is, the switches that are not the source switch of the packet.

D. Gate Flags

G_EXT and G_SRV are two flags of gates defined in the proposal. Gates of the core switches that have access to the external network must have the gates configured with the flag G_EXT. Similarly, the gates of the edge switches that connect the data center servers must have the flag G_SRV.

These flags are used by the core and edge switches to forward tagged packets through the gates that have access with the external network or a server of the data center. If the flag exist, then the switch discards TF-tag before forwarding the packet. When the tag is removed, the packet returns to its original format, that is, the packet with original TCP/IP protocol headers.

The removal is necessary because the server is not capable of processing TF-tagged packets. Moreover, we cannot assure that an entity (i.e., IP router) out of the DCN is capable of processing such packets.

There is one more gate flag called G_ASS. This is a special flag that assists aggregation switches to forward some packets to the core layer. When an aggregation switch receives the packet, it checks if the fields UP and DOWN are both not zero. If the receiveing gate has the flag and the assertion is true, then the forwarding function $f(.)$ defined in Eq. (3) determines the output gate from the number that UP holds.

The aggregation switches are the only switches that should send packets upward or downward without any modifications to the packets. On the other hand, the core and edge switches modifies the packets (drop TF-tag) when they need to send packets, respectively, to the external network and data center server.

E. TF Header

The TF header fits in Layer 2 (L2) after the MAC addresses of the packet header, where the header is composed by four fields named EthType (16 bits), QID (3 bits), UP (32 bits) and DOWN (13 bits). EthType must be 0xff1f to indentify the tagged packet. This allows switches to use the forwarding function $f(g, d)$, where the parameter g receives the number from UP or DOWN. The field UP is used to send the packet to the upper layer and DOWN is used to send the packet to the bottom layer.

UP and DOWN are used to perform stateless packet forwarding. DOWN must have the latest gate in which the packet should go through to reach its destination, while UP must have the gate that connect an aggregation switch with a core switch. QID indicates a queue identification number to be used for traffic shapping.

F. Implementation

TF is an extension of OVS version 2.4 in which new codes are programmed in the user-space module ovs-vswitchd and mostly, in the kernel-space module openvswitch.ko. These modules from the OVS architecture exchange information through the Netlink library. In our current code development, we have not programmed the control plane. That is, no programming interfaces of TF are available in any OpenFlow controller yet. Nevertheless, we do have extended the command-line tool called ovs-ofctl to allow users to setup flow entries and gates.

Gates are configured in the kernel module. More specifically, in the data structure from the original OVS source code called vport. This data structure is an abstracted representation of the network interface card and it has an identification number to indicate the OpenFlow port. In our implementation, we added three more integer variables in the data structure to be used to configure the gate, the gate flag and the common difference number.

However, in order to configure the gates, we should pass the parameters through the user-space module of the OVS architecture. Therefore, ovs-vswitchd receives the gate configuration parameters from ovs-ofctl and it translates these parameters into the equivalent Netlink message to send to the modified openvswitch.ko. In this way, the kernel module becomes aware of the existence of the gates.

IV. TF Working

Figure 1 shows an example of the TF working. We start by configuring the gates according to the properties defined in Subsection III-B. The bottom-up and top-down enumeration phases are interchangeable since that the properties are respected. In our example, we first run the bottom-up phase and then, the top-down phase.

i) Bottom-up enumeration: We enumerate the network interfaces of $es1, es2, ..., es8$ that connect the servers. We must choose the initial term and the common difference number d for AP. We attribute the number 2 as the initial term for the sequece and the variable d, so AP generates the sequence $2, 4, 6, ..., 32$ by the property P1. Therefore, the gates of the edge switches are $G_{es1} = \{2, 4\}$, $G_{es2} = \{6, 8\}$, ..., $G_{es8} = \{30, 32\}$.

The next step is the enumeration of the network interfaces of $as1, as2, ..., as8$ that connect the edge switches. By the property P3, the gates must be formed by the gates that have been configured for the edge switches. At the same time, the properties P2 and P4 must hold true. Therefore, the gates of

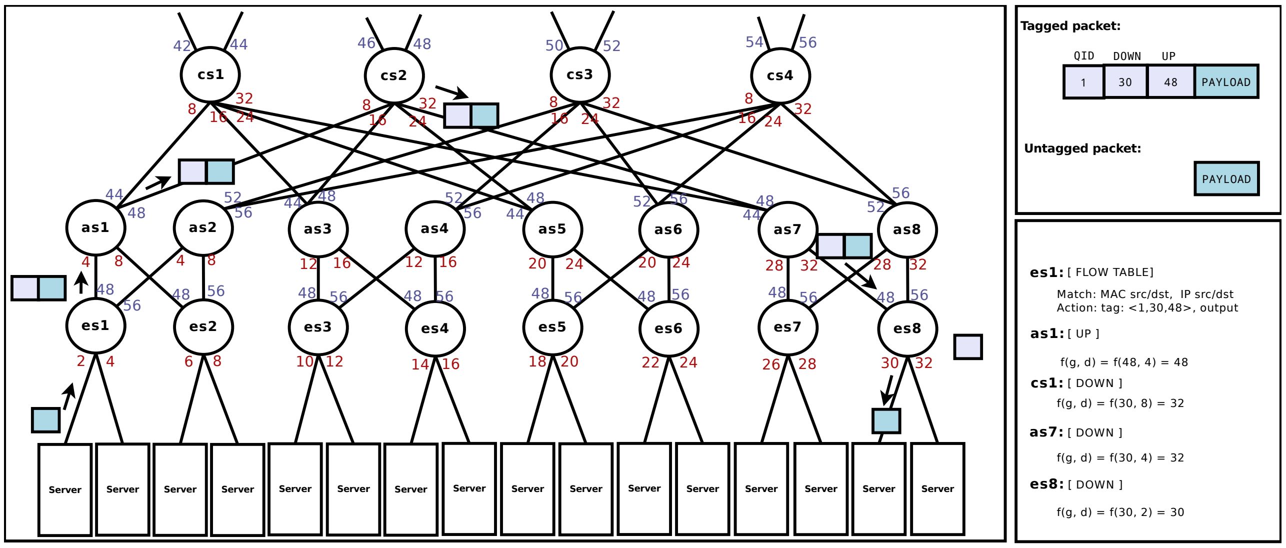


Fig. 1: TF working by example case. Packets are forwarded by processing the TF header tagged in the packets.

the aggregation switches are $G_{as1} = G_{as2} = \{4, 8\}$, $G_{as3} = G_{as4} = \{12, 16\}$, $G_{as5} = G_{as6} = \{20, 24\}$ and $G_{as7} = G_{as8} = \{28, 32\}$, where d is 4 by the property P4.

The same procedure applies for the enumeration of the network interfaces that belong to $cs1, cs2, cs3, cs4$ by following the properties P2 and P3. Therefore, the gates of the core switches are $G_{cs1} = G_{cs2} = G_{cs3} = G_{cs4} = \{8, 16, 24, 32\}$ and d is 8 by the property P4. In this way, the bottom-up phase is complete. We must continue enumerating the rest of the interfaces by following the top-down phase.

ii) Top-down enumeration: By following the same logic used in the bottom-up phase, we enumerate the interfaces of $cs1, cs2, cs3$ and $cs4$ that connect the external network. However, AP must generate gates that are not be equal to any of those gate that have already been used in the bottom-up phase in order to respect the property P2. That is, the initial term of the sequence must be an even number greater than 32. In our example (Figure 1), we attribute the number 42 as the initial term of the sequence and the common difference number d as 2. Therefore, AP generates the sequence $42, 44, ..., 56$ by the property P1, which are the gate numbers of the core switches $G_{cs1} = \{42, 44\}$, $G_{cs2} = \{46, 48\}, G_{cs3} = \{50, 52\}$ and $G_{cs4} = \{54, 56\}$.

The next step is the enumeration of the network interfaces of $as1, as2, ..., as8$ that connect the core switches. By the properties P2 and P3, the gates must be formed by the gates that have been configured for the core switches. That is, the gates of the aggregation switches are $G_{as1} = G_{as3} = G_{as5} = G_{as7} = \{44, 48\}$ and $G_{as2} = G_{as4} = G_{as6} = G_{as8} = \{52, 56\}$, where d is 4 by the property P4.

Similarly, the procedure to enumerate the network interfaces of the edge switches $es1, es2, ..., es8$ that connect the aggregation layer must follow the properties P2 and P3. Therefore, the gate numbers for the edge switches are $G_{es1} = G_{es2} = ... = G_{es8} = \{48, 56\}$, where d is 8 by the property P4.

Once the network is properly configured, the data center servers are able to make connections among them. In our example in Figure 1, the server connected to $es1$ wants to communicate with the server connected to $es8$. So, if the shortest path is $es1 \mapsto as1 \mapsto cs2 \mapsto as7 \mapsto es8$ then the packets should flow through the gates 48, 48, 32, 32 and 30. To this end, $es1$ must associate TF-tag with the packets properly.

In our example, $es1$ associates with all the incoming packets that match IP src/dst addresses of the servers and with the configuration [EthType = `0xff1f`, QID = `1`, DOWN = `30`, UP = `48`]. The field UP holds the output gate of $as1$ while DOWN holds the output gate of $es8$ (latest gate to traverse). We simplified the packet representation in Figure 1 just for more clarity.

The number `0xff1f` allows $as1, cs2, as7$ and $es8$ to indetify TF-tagged packets. Hence, $as1$ takes the values from UP and the common difference number d, and passes as argument to the forwarding function $f(.)$, presented in Subsection III-C, to determine the output gate. Therefore, $f(48, 4)$ is 48 which is the output gate returned by the function.

Once the packet reaches $cs2$, the forwarding procedure is not different. However, $cs2$ takes the field DOWN (instead of UP) and the common difference number d to determine the output gate. Thus, we have $f(30, 8) = 32$. Similarly, in $as7$ and $es8$ we have, respectively, $f(30, 4) = 32$ and $f(30, 2) = 30$, but $es8$ additionally removes TF-tag because it is the latest switch in the path. Due to the number of page limitation of the paper, we will leave the pseudocode that shows in more details how switches choose header fields to use and when to remove TF-tag.

To summarize, $as1$ used flow table while others remained stateless. If the destination server wants to reply messages to the sender server, so $es8$ and $es1$ "change their role". For example, consider that the shortest path is formed by the sequence of switches $es8 \mapsto as7 \mapsto cs2 \mapsto as1 \mapsto es1$ (it could be a different shortest path) which now means that $es8$ insert TF-tag while $es1$ removes it. That is, the source switch is $es8$ and uses flow table while others remain stateless.

Even though $es1$ has rules in the flow table, the packets will not be deviated to the flow table because the Ethertype will be `0xff1f`. Therefore, a switch only uses the flow table when a packet does not have the TF header.

We have ommited in mentioning QID so far to keep focus on the forwarding process. TF uses QID for traffic shapping in the same way that OVS uses. The difference is that there is no flow table involved to enqueue the packets into the queue identified by QID. In our proposal, TF enqueues packets to the corresponding queue of the output gate, determined by the forwarding function $f(.)$, before sending the packets.

V. Performance Evaluation

A. Simulation Setup

We used Mininet prototyper version 2.3 that runs on Ubuntu 14.04 server with kernel 3.16.0 to build the fat-tree DCN topology that Figure 1 shows. In the performance evaluation, we compared the performance of the regular OVS 2.3 with our proposal TF. The server machine, where the experiments were carried, is the Blade server having 32 Intel Xeon CPU E5-2650 2.00GHz with 8 cores each, 64GB RAM and cache size of 2MB.

We generated ICMP packets to evaluate round-trip time (RTT) under different packet sizes. We also evaluated the network throughput by using Pktgen to generate 8K concurrent UDP flows where each flow is 20-second long. In both evalutation, the flow table of the switches is saturated at 20% of the total capacity (maximum is 200K [15] in kernel). Moreover, we measured the time spent to configure the gates.

We used the utility `ovs-ofctl` to populate the flow tables of the user space `ovs-vswitchd` daemon. To make sure, we performed preliminary tests to check if the rules were properly being copied to the kernel module `openvswitch.ko` by using the tool `ovs-dpctl`.

OVS and TF results were averaged with the confidence interval of 90% under 100 runnings. The data samples showed a small skew from the average, so we think 100 runnings is a good size to statisticaly validate the evaluation.

B. Round-Trip Time

The plot in Figure 2a shows RTT values under different packet sizes. We can see that the performance dropped as the packet size increased. However, TF showed a better performance. The performance gain is about, respectively, $61\% - 63\%$ for the packet sizes smaller than the default Ethernet MTU size which is 1500 bytes and $56\% - 65\%$ for packet sizes bigger than the MTU size.

RTT increases due to the fact that the network devices take longer transmission time. However, as TF does not waste time with packet parse and successive forwarding table lookups, in which both operations involve many memory operations, TF can avoid wasting time where OVS does. That is, the output gate is generated by the function $f(.)$ (Subsection III-C) and not retrieved from memory (action list) like OpenFlow port is when the match is found in memory (match rule).

The large packets sent was used as a proof of concept that our implementation works for packet sizes above MTU. Although, TF has nothing to do on the packet segmentation, we wanted to make sure that TF is capable of sending those packets. As expected, the result shows an increase on RTT values because the sending host spends more time on the packet transmission, because of the packet segmentation delay.

We should mention that the network interface driver has to be aware of the addional header size to avoid dropping the frame, because of the excess bytes. At first attempt, TF was not being able to send the tagged packets, the total amount of bytes in the packet was greater than the MTU size. Hence, the network interface card was dropping these packets.

C. Packet Transmission Rate

The results in Figure 2b shows that TF improved the network performance by roughly 40% when compared to OVS. As we can see both OVS and TF drop transmission capacity as the packet size increases, but the TF performance is still noticebly much better. As the transmission rate is the ratio between the number of packets sent over time, the performance degradation occurs due to the increase on the packet transmission delay. However, as OVS spends time on packet parse operations, this sums a fraction of delay cost to the overall delay as the packets traverse the network.

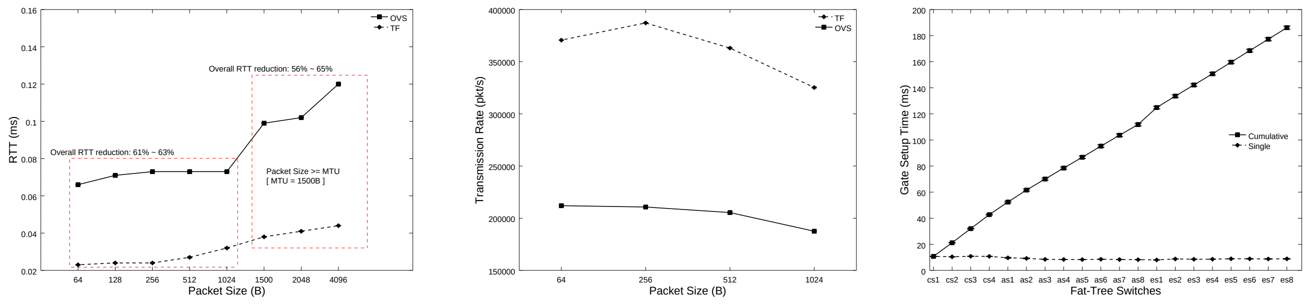
In addition, there is also the kernel and userspace communication delay when the forwarding rules have not been cached in the kernel module (`openvswitch.ko`). Such delay always occurs when the first packet of each flow traverses the network along the forwarding path. The others does not suffer with such delay because the forwarding rules have already been copied from the userspace's table to the kernel's table, unless the rule expires.

D. Gate Setup Time

Figure 2c shows the average time that switches take to configure the gate numbers. The gate enumeration procedure follows the example presented in Section IV where the bottom-up enumeration procedure comes first and then, the top-down enumeration procedure comes later. That is, switches are configured sequentially. Therefore, $cs1$ is the first switch in the sequence that had all the network interfaces configured and sequencially we had $cs2$, $cs3$, $cs4$, $as1$, ..., $es8$ configured as the horizontal axis of the plot in Figure 2c shows.

The plot shows two curves where one is indicated by the squared points and the other one is indicated by the circled points. They represent, respectively, the total amount of time spent to configure the fat-tree topology and the time spent to configure one switch at a time as the gate enumeration procedure goes.

As we can see, the core switches spend slightly more time than the others. The core switches spend about $10ms$ while the



(a) Round-trip time (b) Packet transmission rate (c) Gate setup delay

Fig. 2: Average performance of TF. (a) shows the RTT under different packet sizes, (b) shows the achieved packet transmission rate when the flow table is overloaded at 20% and (c) shows the time needed to configure the gate numbers.

aggregation and edge switches spend about $8ms$ on average to have their interfaces configured. The core switches take longer because they have 6 interface cards and the others have 4 interface cards to be configured. These time spent is roughly 5% of $186ms$, which is the total amount of time spent to configure the network. That is, each switch is contributing with 5% in the gate configuration process which is an acceptable delay to configure 4 to 6 interface cards that each switch has in the topology.

Moreover, the configuration time of $186ms$ to configure 104 network interface cards was better than an expected and it is a reasonable time spent, because it took less than 2 seconds. Surely, the total amount of time spent varies as the fat-tree network grows.

VI. Conclusions

We presented the proposal named Tag-and-Forward (TF) which is a source-routing enabled OpenFlow dataplane for the fat-tree software-defined DCNs. TF reduces the control plane dependency and the need for having network state distribution over all available flow tables in the data plane.

Our results showed that TF reduced RTT by roughly 63% and improved the network packet transmission rate by roughly 40% when compared to the normal OpenFlow data plane performance, which are a satisfactory improvement for data center network applications.

Acknowledgment

The authors would like to thank SERPRO (brazilian government company) for the financial support and partnership.

References

[1] B. Wang, Z. Qi, R. Ma, H. Guan, and A. V. Vasilakos, "A survey on data center networking for cloud computing," *Computer Networks*, vol. 91, pp. 528–547, 2015.

[2] K. Chen, C. Hu, X. Zhang, K. Zheng, Y. Chen, and A. V. Vasilakos, "Survey on routing in data centers: insights and future directions," *Network, IEEE*, vol. 25, no. 4, pp. 6–10, 2011.

[3] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 254–265.

[4] A. Lazaris, D. Tahara, X. Huang, E. Li, A. Voellmy, Y. R. Yang, and M. Yu, "Tango: Simplifying sdn control with automatic switch property inference, abstraction, and optimization," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, 2014, pp. 199–212.

[5] H. Huang and S. Guo, "Multi-flow oriented packets scheduling in openflow enabled networks," in *Communications (ICC), 2015 IEEE International Conference on*. IEEE, 2015, pp. 5753–5758.

[6] S. Veeramani, M. Kumar, and S. N. Mahammad, "Minimization of flow table for tcam based openflow switches by virtual compression approach," in *Advanced Networks and Telecommuncations Systems (ANTS), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1–4.

[7] 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*. IEEE, 2014, pp. 109–117.

[8] K. Kannan and S. Banerjee, "Scissors: Dealing with header redundancies in data centers through sdn," in *Proceedings of the 8th International Conference on Network and Service Management*. International Federation for Information Processing, 2012, pp. 295–301.

[9] B. Leng, L. Huang, X. Wang, H. Xu, and Y. Zhang, "A mechanism for reducing flow tables in software defined network," in *Communications (ICC), 2015 IEEE International Conference on*. IEEE, 2015, pp. 5302–5307.

[10] S. Luo, H. Yu *et al.*, "Fast incremental flow table aggregation in sdn," in *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on*. IEEE, 2014, pp. 1–8.

[11] K. Qiu, Z. Chen, Y. Chen, J. Zhao, and X. Wang, "Gflow: Towards gpu-based high-performance table matching in openflow switches," in *Information Networking (ICOIN), 2015 International Conference on*. IEEE, 2015, pp. 283–288.

[12] M. Martinello, M. Ribeiro, R. E. Z. de Oliveira, and R. de Angelis Vitoi, "Keyflow: a prototype for evolving sdn toward core network fabrics," *Network, IEEE*, vol. 28, no. 2, pp. 12–19, 2014.

[13] Y. Chiba, Y. Shinohara, and H. Shimonishi, "Source flow: handling millions of flows on flow-based nodes," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 465–466, 2011.

[14] C. A. Macapuna, C. E. Rothenberg, and M. F. Magalhaes, "In-packet bloom filter based data center networking with distributed openflow controllers," in *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*. IEEE, 2010, pp. 584–588.

[15] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 117–130.