Application Layer Flow Classification in SDN

Hamid Farhadi, Akihiro Nakao
The University of Tokyo
{farhadi, nakao}@nakao-lab.org

***Abstract*—Software Defined Networking (SDN) increasingly attracts more researchers as well as industry attentions. Most of current SDN packet processing approaches classify packets based on matching a set of fields on the packet. We propose a tag-based packet classification architecture to reduce filtering and flow management overhead.**

I. Introduction

Software Defined Networking (SDN) increasingly attracts more researchers as well as industry attentions. Last year, Google announced they start using SDN to improve their internal network between data centers. An out of the box outline of SDN roadmap shows significant improvements in network control and management. Network measurement which is the key stone of network management has applications beyond controlling the network such as providing different APIs for service providers. An example of such a service can be network measurement specifically for accounting, billing and charging procedures of the end-user. Such an application layer classification, in-network, is a resource intensive task. The key challenge when we deal with software defined network measurement, is the tradeoff between flexibility (or generality and programmability) and efficiency. By the flexibility we mean the ability to measure sophisticated features of the traffic. Currently using application layer metrics in the form of a generic measurement solution received minor attention. An example application of such a measurement can be a generic usage-based charging as a service. In short, *we need a more flexible measurement which is software defined and written by service providers customized for their own specific use, transparent from their internal applications and end-users.*

The rest of the paper is organized as follows. Next we present related work and then we review our system architecture following by evaluation and finally we conclude the paper.

II. Related Work

We can divide current state of the art to two main classes of methods based on the way they filter the target traffic to measure. Different approaches use variant solutions to initiate, store and maintain flow information. In the following we mention two classes of methods that follow different goals but share their core functionality:

A. Hashing Approach

Most of todays measurement and forwarding solutions apply a <*match,action*> rule to every packet passing through the system. The *match* section means a pattern or attribute that the packet includes. After classifying and matching packets with specific filters, an *action* is applied to the packet. Examples of the action can be *forward/drop* for packet switching solutions or *count* for a measurement solution. The latter action means the counter for that specific matching criterion should be incremented based on the counter type. A counter could be a packet counter, a flow size counter or a timer etc.. In the hashing approach, basically the packet is parsed to find some special fields targeted by the *match* rule. In many cases such as OpenFlow, NetFlow and sFlow, we may have 10 to 15 distinct fields to be parsed and involved in the classification. These fields can be located anywhere in the packet. The main bottleneck of many-field parsing is the number of memory accesses which grows as the load increases. Finally, a sort of *hashing function* is applied to the parsed fields to generate a *key*. The key is used to search a list of records containing flow information. In case of packet forwarding solutions (e.g., OpenFlow) the table is the forwarding table and usually implemented using a *hashtable* data structure with lookup complexity of *O(1)*.

OpenFlow, DevoFlow [1] and Hedera [2] are a group of SDN APIs primarily designed to control the forwarding logic of network efficiently. They place as a logically centralized control plane in the network and install <match,action>rules on forwarding devices to manage the network. They are all designed to work on commodity hardware and they have some measurement features such as raw packet counters or approximated counters. DevoFlow supports limited data plane programmability and all of them support control plane programmability.

OpenSketch [3] is an SDN API dedicated for network measurement using sketches. The classification process in this system is also similar to other hashing approaches. The main drawbacks of this approach is a) all measurement task should be reducible to binary sketches otherwise the measurement should happen at the controller which is not feasible for real world traffic loads and b) it is installed in a centralized point and all the measurement happens at a single point and more importantly c) sketches which receive the output of hash functions, and the main programmability of the system is claimed on the sketches while the classification part has limited programmability. Since any classifier should be implementable to TCAM binary entries plus hash function. Obviously, any reducing and application layer classification mechanism (e.g., parsing http header) is not a trivial task using hash functions plus binary TCAM entries. In short, the classification part of OpenSketch follow the same approach OpenFlow has and is limited by the same predefined packet fields.

B. Tagging Approach

This approach received a limited attention from the community in comparison to hashing method. The major work in this area is MPLS. MPLS assigns a label to every packet and use it to forward packets in contrast to IP switching in which a sophisticated prefix matching decides the destination port of each packet. MPLS is dependent on a fixed layer-2 protocol.

In addition to old contributions, recently some new requirements motivate researchers to revisit the same idea. OpenTag [4] is a slicing mechanism for network virtualization that supports both performance and security isolation. In OpenTag, the user injects a slice ID tag per packet that denotes which slice the packet belongs to.

LIPSIN[5] is a forwarding platform using source routing method for publish/subscribe networks. The whole path a packet should pass through (i.e., virtual link), is encoded using some hash functions in a fixed-sized label within the packet.

III. System Architecture

In this section we discuss the proposed architecture in details. Figure 1 illustrates the top level system architecture. Before explaining our method, we explain our assumptions about the environment.

The main assumption is that we are deploying our system in a software defined network that is already virtualized. The service provider rents a slice from infrastructure provider and has the ability to modify all SDN routing and switching logic within its own slice. The user joining process is simply connecting and authenticating on a switch at the edge of the network. This process is already implemented in many networks using tunnels and authentication protocols such as 802.1x port authentication.

There are two types of contents in the service provider domain that an end-user can access: paid and free; The system distinguishes and meters the amount of access to paid content and report it to the financial management system of the corresponding slice for each end-user. For simplicity we focus on pre-paid charging model of Online Charging Systems. That is, each end-user has an account and deposit money. As s/he use paid content, the system measures and reduces the equivalent charge from the account. When the account balance reaches zero, the system denies user to access paid account.

The network has two edges: source edge and destination edge. The former means the switch that connects the application server to the network. Destination edge means where the packet leaves the network and it is the same switch in which end-user authentication happens.

When an end-user accesses to a paid content, packets including the paid content flow in the network from the source edge. The source edge switch classifies packets carrying a paid value and puts a *Value Tag (vTag)* on every packet. The classification is an application layer classification that distinguishes among applications with different payment criterion. Contents with different charging policy receive different vTags. Therefore, every vTag type represents a class of content with the same metering criteria. After inserting appropriate vTags to related classes of packets, the regular network routing and

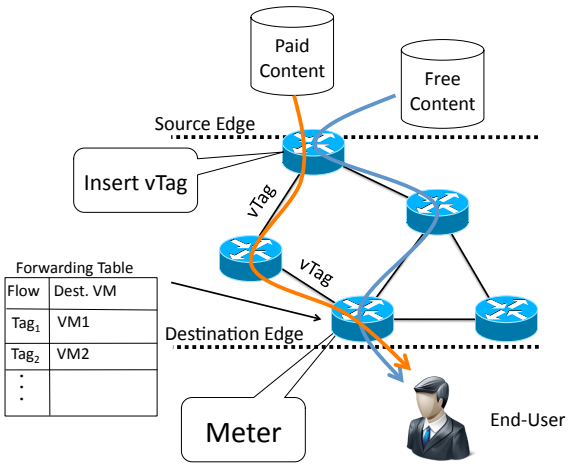

Fig. 1. System Architecture

switching algorithms transmits the packet to the destination edge. The switch at the destination edge (i.e., the Meter in the figure) checks the vTag and depending on the user session, updates corresponding metering values of the end-user. Then removes the vTag from the packet and let it leave the network for destination. Metering values are count down counters that are fetched by the edge switch from slice financial management system once the user logs in. Different slices have different instances of financial management system that take care of their own user accounts. When the user is authenticating, relevant metering parameters are fetched by the authenticating edge switch.

Figure 1 indicates the forwarding table at the destination edge switch. The forwarding table has two columns: flow identifier and destination port. We use tags as the flow identifier. In our scenario, the network is sliced using tags on the packet. The vTag is an additional tag to slice ID tag which both can be combined in one tag. The second column which is the destination VM means to which switch virtual machine (VM) the packet should be forwarded. Since the switch HW is virtualized, each slice has its own switch VM and packets carrying slice ID tag, go to the corresponding switch VM where the switching and metering happens.

IV. Evaluation

As the evaluation we show the difference between hashing approach and tagging approach with network-wide view and conclude that there is a considerable difference in resource usage between these two approaches. Then we argue we can exploit such a processing resource for application layer classification.

Our evaluation and all calculations as well as projections diagrams are based on a previous work in [6] which is offloading classification feature of OpenFLow to network interface card armed with an embedded hardware classifier. Recent NICs such as those with an Intel 82559 10GbE controller are able to classify traffic based on packet fields including a few commonly used fields for OpenFlow. Figure 2 (a) shows the pure amount of processing we need to classify using hashing approach[6]. In particular, this diagram is based on

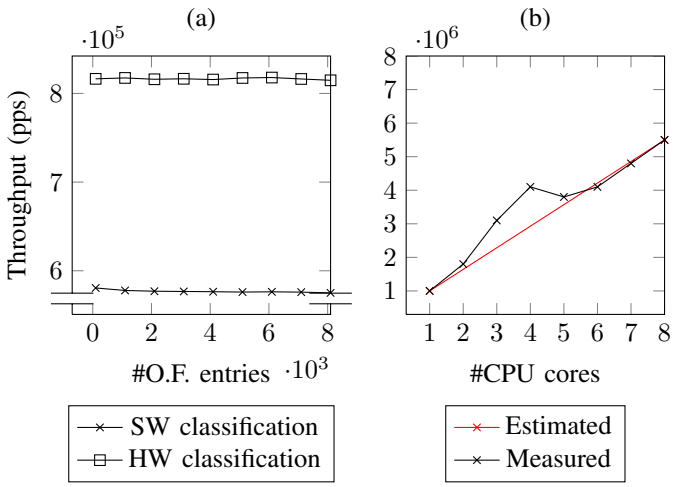

Fig. 2. (a) Flow classification using hardware vs software in OpenFlow environment (b) Forwarding throughput versus number of CPU cores

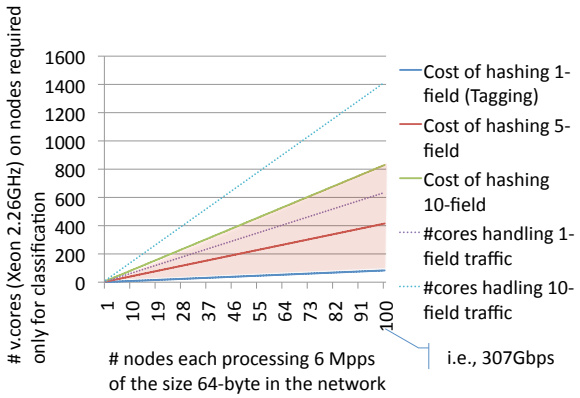

Fig. 3. 1-field versus 10-field clasification efficientcy

a measurement using a Xeon Quad core 5520 processor with 64-byte packets. The architecture consist of simply a packet generator that sends traffic to the server with a dual port NIC and receives packets back to calculate the performance. The line corresponding with SW classification indicates a OpenFlow switch completely implemented in software and the other line illustrates the same scenario but with the classification functionality offloaded to the HW. Particularly, the NIC classifier is the HW classifier in this scenario.

The difference between two lines shows the pure classification cost of hashing approach in case of 5-field hashing using only one core of the CPU. The x-axis of the graph shows the number of entries in OpenFLow forwarding table. The near zero slope of both lines show the forwarding is almost independent from the number of rules. The difference between two lines is more than 200 Kpps which is a large difference. The reason behind this difference is that the classification process consists of two steps: parsing and lookup. As the lookup process has negligible overhead, the main cause of difference is parsing. That is, accessing delay to different parts of the packet on the memory grows very fast as the number of packets coming to the system grows. So a 10-field packet classification rule causes double overhead of classification comparing to 5-field in the worst case. Building on this intuition, we project the diagram to a 10-field packet classification scenario which can be the case for many hashing approach classifiers.

Figure 2 (b) simply illustrates the affect of using multicore processing on the throughput of the system[6]. Since we want to consider a network-wide view of nodes each processing a couple of Mpps, we need to consider multicore processing scenario. The diagram shows the throughput of 8 cores. To keep the calculation simple we consider a linear increase with the number of cores shown by the red line in the diagram.

Figure 2 consists of 5 lines. The red line is the result of applying Figure 1 (a) on the multicore environment of Figure 1 (b) which is noted as the cost of 5-field classification using hashing approach. Note that this is only the *cost* of classification meaning the set of resources we need to *only* classify the traffic excluding the forwarding overheads. As depicted on the x-axis we consider a network with 100 nodes each processing 6Mpps of the size 64-byte. On the y-axis we have number of virtual cores we need for classification. As an example, for classification of 6Mpps of traffic (i.e., a network with only one node) we need 10 virtual cores.

The blue and green lines are the projection of red line (i.e., 5-field classification) to 1-field and 10-field classification (only), respectively. For the total cost of handling packets with different approaches (i.e., classification and forwarding) see the dotted lines. We can see the same difference between two dotted line and between red and blue line. The main intuition behind Figure 2 is difference between the classification cost of hashing and tagging. The red area shows this difference. We conclude that we can use this huge amount of processing resources as application layer classifiers at the source edge switches to filter and assign vTags to packets. We leave an example application layer classification implementation as the future work.

V. Conclusion

We proposed a simple tag based distributed classification method as the replacement for hashing techniques. This method saves processing power and provides a room for more complex classification mechanism such as the classification we need for in-network usage based charging and billing of users.

References

[1] A. R. Curtis, J. C. Mogul, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," *ACM CCR*, vol. 41, no. 4, p. 254, 2011.

[2] M. Al-Fares, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proceedings of USENIX NSDI*, 2010.

[3] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," Tech. rep., USC, Tech. Rep., 2012.

[4] R. Furuhashi and A. Nakao, "Opentag: Tag-based network slicing for wide-area coordinated in-network packet processing," in *IEEE ICC*, 2011.

[5] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, "Lipsin: line speed publish/subscribe inter-networking," in *ACM CCR*, vol. 39, no. 4. ACM, 2009, pp. 195–206.

[6] V. Tanyingyong, M. Hidell, and P. Sjodin, "Using hardware classification to improve pc-based openflow switching," in *IEEE HPSR*, 2011.