FleXam: Flexible Sampling Extension for Monitoring and Security Applications in OpenFlow

Sajad Shirali-Shahreza
Department of Computer Science
University of Toronto, Canada
shirali@cs.toronto.edu

Yashar Ganjali
Department of Computer Science
University of Toronto, Canada
yganjali@cs.toronto.edu

Categories and Subject Descriptors

C.2.3 [**Communication Networks**]: *Network monitoring*

Keywords

OpenFlow, Sampling, Software-Defined Network.

1. Introduction

Software-Defined Networking (SDN) aims at simplifying and enhancing network control and management, while making it easy to implement new applications. For some networks, it might be desirable to implement security services such as intrusion detection systems (IDS) and network monitoring as controller applications in the SDN, due to simplicity and agility of such a solution. For instance, in small-sized networks, the cost of adding extra security boxes can be significantly reduced if we can implement such services as controller applications.

However, the feasibility of such a design is not obvious. Network security applications, such as IDS, usually require access to packet level information such as packet content and inter-arrival times, but packet level information might not be readily available in SDN controllers. OpenFlow is primarily designed for routing applications [3], and mostly deals with flows rather than individual packets. There are three major information channels for the controller in the current OpenFlow specification:

1) Event-based Messages. Event-based messages are sent by the switches at events such as state change of a link or port, and usually deliver information about changes in network structure and topology.

2) Flow Statistics. Flow statistics (received packets, received bytes and duration in the current specification) are collected by the switches and pulled by the controller. This is the only way for the controller to collect information about active flows.

HotSDN'13, August 16, 2013, Hong Kong, China.

3) Packet-in Messages. Switches send a packet-in message either because they did not know what to do with a packet (no matching entry found in the flow table) or as a result of a send-to-controller action in the matching flow entry. The switches may buffer the packet and only include part of the packet (usually the first 128 bytes) in the message.

These information channels are designed to provide flow-level (as opposed to packet-level) information to the controller. The first two do not provide any packet-level information, and the third one only provides limited access to such information. Even in the original OpenFlow proposal [3], it is suggested to direct flows that require further packet-level analysis to a separate machine dedicated to this purpose. As a result, it is difficult and inefficient, if not impossible, to implement applications that need packet-level information as OpenFlow controller applications with current specification.

There are two ways for an OpenFlow controller application to access packet-level information of a given flow. The first option is to not install any flow entries for the desired flow on one of the switches on the path. Every packet of the flow will be a table miss at that switch, triggering a packet-in message from the switch to the controller. The controller then needs to tell the switch to send out these packets on the correct port. This option was used in SDN-based port-scan detection system proposed by Mehdi et al. [4]. This approach has two major limitations. First, the controller effectively sits on the packet delivery path, potentially creating a bottleneck, and leading to increased packet delivery time. Second, the switch may, and probably prefer to, buffer the packet locally and only send part of it to the controller, which will limit the amount of packet content that the controller can access.

The second option, which was suggested in [3], is to ask the switch to send a copy of each packet to another machine (probably a monitoring host other than controller, considering the load and scalability issues of the controller), during the forwarding process. This option might have a high overhead which is not tolerable for all applications. The monitoring machine needs to be extremely powerful given the significant load resulting from applications such as IDS that need to process all connections, which can lead to major increases in cost and complexity.

2. FleXam: Flexible Sampling Extension

In this paper, we propose FleXam, a flexible sampling extension for OpenFlow that enables the controller to access packet-level information. Simply stated, the controller can define which packets should be sampled, what part of packet should be selected, and where they should be sent. Packets can be sampled stochastically (with a predetermined probability) or deterministically (based on a pattern), making it flexible for different applications. At the same time, it is simple enough to be done entirely in the data path. The controller can also request switches to only send part of packets that are needed (e.g. headers only, payload, etc.) and define where they should be sent, make it possible to easily manage and distribute the load.

FleXam includes two types of sampling: (1) select each packet of the flow with a probability of ρ, and (2) select m consecutive packets from each k consecutive packets, skipping the first δ packets. The first case is the stochastic sampling. The second case is a generalized version of the deterministic sampling. For $m=1$, it is equivalent to the normal one out of k, or every k^{th} packet sampling. If an application needs more than one consecutive sample, it can set m to a value more than one. By choosing a very large k, an application can ensure it will only receive the first m consecutive packets. This is usually what security applications such as intrusion detection need. Finally, by changing the value of δ, the application can skip the first few packets of each flow, e.g. to exclude small and short flows (mice flows). Considering that sending full packets could impose a significant load on the network, and not all applications need full packet contents, the controller can define what parts of packets (e.g. IP header) should be sent.

Like other per-flow sampling schemes, FleXam has the advantages of generating a more accurate estimation of traffic statistics [1] in comparison to per-packet sampling methods like sFlow. It is also a better fit for security applications (such as IDS) that need data about short-lived flows, which can be missed by uniform sampling [5] or flow-based sampling techniques that focus on heavy-hitters [2].

2.1 OpenFlow Specification Modification

We define sampling as a new action (OFPAT_SAMPLING) that could be assigned to each flow. This new action is similar to OFPAT_OUTPUT, which sends the sampled packet to the controller. The action has six parameters: *scheme*, ρ, m, k, δ and *destination*. The *scheme* parameter defines the sampling scheme that identifies which parts of the sampled packets will be sent. The m, k, and δ parameters define deterministic sampling, and ρ defines stochastic sampling. The *destination* parameter defines the host that sampled packets should be sent to. It could be the controller (e.g. for small networks that the controller can run the application that uses the sampled data) or another host that will process the sampled packets and communicate with the controller.

Representing sampling as an action has two main advantages. First, it can be easily added to current OpenFlow implementations without the need to modify the overall processing structure (such as matching with the flow table or performing different actions). Second, there is no overhead for flows that do not need sampling.

2.2 Switch Implementation

Implementing per-flow sampling in traditional networks and in solutions such as ProgME [6] requires significant changes to the hardware and packet processing flow in the switches. However, identifying the flow that the packet belongs to and keeping a record of the identified flows – a major task for per-flow sampling methods in traditional networks such as ProgME [6] – is an essential part of OpenFlow. As a result, our sampling extension could be implemented easily without requiring any significant hardware changes or software modifications.

The stochastic case is relatively straightforward to implement: for each packet, a random number is generated. If it is less than ρ, the packet will be selected for sampling. The deterministic case can also be implemented easily, without any additional memory or counter. OpenFlow switches maintain a number of counters for each flow. One of them is the Received Packets counter, which counts the number of packets received for each flow. Deterministic sampling of m first packets from each k packets after ignoring δ packets could be done using this counter: if $((Received\_Packet\_Counter-\delta)\ \%\ k)<m$, then the packet will be sampled. This simple expression could be executed in the data-path at line rate without the need for any new memory for sampling.

3. REFERENCES

[1] Hohn, N., and Veitch, D. 2006. Inverting sampled traffic. IEEE/ACM Trans. Netw. 14(1). 68-80.

[2] Mai, J., et al. 2006. Is sampled data sufficient for anomaly detection? In IMC '06. 165-176.

[3] McKeown N., et al. 2008. OpenFlow: enabling innovation in campus networks. Comput. Commun. Rev. 38(2). 69-74.

[4] Mehdi, S.A., Khalid, J., and Khayam, S.A. 2011. Revisiting traffic anomaly detection using software defined networking. In RAID'11. 161-180.

[5] Salem, O., et al. 2010. A scalable, efficient and informative approach for anomaly-based intrusion detection systems. Int. J. Netw. Manag. 20(5). 271-293.

[6] Yuan,L.,et al. 2011.ProgME:towards programmable network measurement. IEEE/ACM Trans. Netw. 19(1).115-128.