Which Flows Are Hiding Behind My Wildcard Rule? Adding Packet Sampling to OpenFlow

Philip Wette
University of Paderborn
Warburger Straße 100
33098 Paderborn, Germany
philip.wette@uni-paderborn.de

Holger Karl
University of Paderborn
Warburger Straße 100
33098 Paderborn, Germany
holger.karl@uni-paderborn.de

ABSTRACT

In OpenFlow [1], multiple switches share the same control plane which is centralized at what is called the OpenFlow controller. A switch only consists of a forwarding plane. Rules for forwarding individual packets (called *flow entries* in OpenFlow) are pushed from the controller to the switches.

In a network with a high arrival rate of new flows, such as in a data center, the control traffic between the switch and controller can become very high. As a consequence, routing of new flows will be slow. One way to reduce control traffic is to use wildcarded flow entries. Wildcard flow entries can be used to create default routes in the network. However, since switches do not keep track of flows covered by a wildcard flow entry, the controller no longer has knowledge about individual flows. To find out about these individual flows we propose an extension to the current OpenFlow standard to enable packet sampling of wildcard flow entries.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: [Network communications, Packet-switching networks];
C.2.4 [**Distributed Systems**]: [Network operating systems]

General Terms

Software-Defined Networks, Packet Sampling

Keywords

OpenFlow

1. INTRODUCTION

OpenFlow [1] is a protocol for communication between the forwarding plane of switches and a (logically) centralized control plane in a Software-Defined Network (SDN). In the OpenFlow model, switches only consist of a forwarding plane that is equipped with what is called a *flow table*. A flow table is a collection of flow entries that identify traffic flows and determine how each packet of a flow is processed. There are 9 packet header fields to match on, where matching can either be done exact (where each single field has to match) or wildcarded (where particular header fields can be omitted). Wildcarded flow entries thus group several individual flows together and treat them in the same way.

Whenever a switch does not have an appropriate flow entry for an incoming packet, this packet is encapsulated in a `PACKET_IN` message and sent to the controller. The controller then computes an appropriate route for the flow and pushes the corresponding flow entry to the switch. This way, all subsequent packets of that flow are handled by the newly installed rule. For a network with a high arrival rate of new flows, the amount of `PACKET_IN` messages is very high. However, even current high-end ToR switches like the IBM G8264 are only capable of creating up to 200 `PACKET_IN`s per second [2], which massively restricts its usage in such networks. To cope with these limitations, wildcard flow entries have to be installed in the switch. Thus, default routes for different groups of flows are defined in a proactive manner. But due to grouping, the fine-grained control of the network is lost. The only way to see which single flows are hiding behind a wildcarded flow is to delete the wildcard flow entry from the switch. Then, each subsequent packet that was previously covered by the wildcard flow entry creates a `PACKET_IN` that is sent to the controller. This is clearly a very inefficient way to look behind a wildcard flow entry.

This paper proposes to add a packet sampling mechanism to the OpenFlow standard to efficiently unveil which individual flows are hiding behind a wildcard flow entry. Since we use packet sampling, the gained results are only approximations. Unlike in conventional packet sampling techniques (like sFlow), samples are not taken from all incoming packets but only from the wildcarded flow entries that are under suspicion. This is more economical: for a given level of accuracy, less samples are required.

2. OPENFLOW PACKET SAMPLING

2.1 Protocol Extension

To add sampling support to OpenFlow, we propose the following extensions to the standard: To invoke the sampling process on a switch for a specific flow entry we extend the `ofp_stats_type` by the new type `OFPST_SAMPLING`. The body of such a message is defined in Figure 1. On reception of this message, a switch marks every matching wildcard entry with the requested sampling probability and duration. In the message, `sampling_period` specifies the average sampling

SIGCOMM'13, August 12–16, 2013, Hong Kong, China.
ACM 978-1-4503-2056-6/13/08.

```
struct ofp_flow_sampling_request {
    struct ofp_match match;  /* Fields to match. */
    uint8_t table_id;        /* Table ID (from ofp_table_stats) */
    uint8_t pad;             /* Padding */
    uint16_t bucket_size     /* Bucket size */
    uint32_t max_samples;    /* Max number of samples */
    uint16_t out_port;       /* Output port. */
    uint16_t sampling_period;/* Average sampling period */
    uint16_t duration;       /* Sampling duration in ms */
};
```

Figure 1: `ofp_flow_sampling_request` message.

```
struct ofp_sample_flow {
    struct ofp_header header;
    uint16_t total_len;     /* Full length of frame. */
    uint16_t sample_len;    /* Length of sampled packet. */
    uint16_t in_port;       /* In port */
    uint16_t bucket_empty;  /* Indicator, if bucket empty */
    uint64_t cookie;        /* Opaque controller-issued ID. */
    uint8_t data[0];        /* Ethernet frame */
};
```

Figure 2: `ofp_sample_flow` message.

period. If set to p, 1 out of p packets are sampled and sent to the controller. `duration` specifies the sampling duration in milliseconds.

Sampled packet headers are sent to the controller in a `OFPT_FLOW_SAMPLE` message (Figure 2), where `total_len` is the length of the encapsulated Ethernet frame (without payload) and `sample_len` is the length of the originally sampled packet (including payload). `in_port` is the port on which the packet was received. The message additionally contains the `cookie` of the wildcard entry that was matched. This way, it is possible to have one switch sample multiple rules at a time without large processing overhead at the controller. If the wildcard flow entry from which a packet is sampled does not have a cookie, the value -1 (0xffffffffffffffff) is used.

When a wildcard flow suddenly increases its data rate during a sampling period, the controller is flooded with `OFPT_FLOW_SAMPLE` messages. To counteract this, we use the *token bucket* algorithm to limit the emission of samples. The token generation rate is defined as $r = \frac{\texttt{duration}}{\texttt{max\_samples}}$, where a token is added to the bucket every $1/r$ seconds. `bucket_size` denotes the size of the bucket. When the bucket gets empty, the controller is informed by sending a `ofp_sample_flow` with `bucket_empty` set to 1 (0x0001); by default, `bucket_empty` is always set to 0 (0x0000).

The packet sampling extension is very simple to implement since all required building blocks are already used in the OpenFlow implementation. Only a random generator is required in addition. Our extension requires 208 bits of storage overhead per installed wildcard flow entry. We implemented this proposal in the OpenFlow 1.0 user space reference implementation. Figure 3 plots the forwarding performance of our implementation. We installed two wildcard rules on a switch connecting two hosts. The plot shows the maximum achievable data rate between both hosts under different sampling periods. The switch was emulated using mininet on an Intel Core i7 QM 2.2 GHz with 8 GB memory.

2.2 Sample Application 1: Default Routing

When operating latency-critical networks with OpenFlow, it is not preferable to install flows reactively (by sending a `PACKET_IN` to the controller each time a new flow arrives). Thus, the switches have to be populated with wildcard entries to speed up forwarding. These wildcard entries create default routes in the network. In case these default routes get

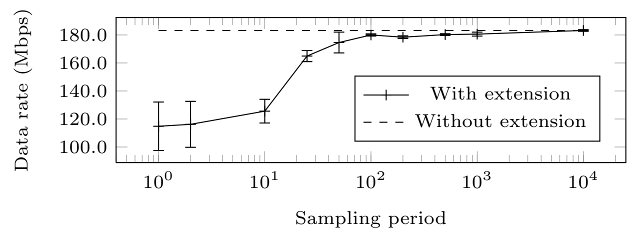

Figure 3: Throughput over different sampling rates with confidence level of 95%.

congested, it is preferable to reroute some of the flows to less utilized links. This, however, would require knowledge about the flows which are hiding behind the wildcard flow entry. With our extension, the wildcard flow entry can be sampled and a new route for the extracted flows can be installed.

2.3 Sample Application 2: Garbage Collection

Flow tables are stored on ternary content-addressable memory (TCAM), which is expensive and therefore a limited resource in OpenFlow switches. As a consequence, the amount of flow entries that can be stored on a single switch is very limited. After the limit has been reached, it is not possible for the switch to process any more additional flows. In such a situation it is favorable to group multiple flow entries and represent these by using a wildcard flow entry.

To match a packet on wildcard flow entries, in most practical implementations the corresponding packet headers have to be compared to each installed wildcard flow entry of the switch. Thus, with increasing wildcard flow entries at the switch, the latency increases, too. To counteract this, wildcard flow entries covering only a small number of individual flow entries should be removed from the switch and reinstalled as individual flow entries if possible. This, however, requires a function to estimate the number of flows before deletion. To accomplish this, our packet sampling extension can be used.

3. CONCLUSION

With our OpenFlow packet sampling extension it is possible to efficiently find out which individual flows are hiding behind a wildcard flow entry. This is a powerful tool when building OpenFlow applications that reside on wildcard flow entries, but requires detailed knowledge of the traffic information after a wildcard flow entry is installed.

4. ACKNOWLEDGMENTS

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).

5. REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[2] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter. Past: scalable ethernet for data centers. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 49–60. ACM, 2012.