Optimizing Packet Classification in Open vSwitch

Megha Arora
Carnegie Mellon University
marora@andrew.cmu.edu

Jeffrey Helt
Carnegie Mellon University
jhelt@andrew.cmu.edu

Rajesh Jayaram
Carnegie Mellon University
rkjayara@cs.cmu.edu

1. INTRODUCTION

Open vSwitch (OVS) [8] is a virtual software switch that allows users to run multiple virtual machines on a single physical host. It is an important abstraction for administrative purposes but also improves the efficiency of the system. Open vSwitch specifies a "virtual switching" layer, which applies a potentially very large set of classification rules to an incoming packet to determine where it should be routed. These rules are critical for security and are used for traffic filtering, acting as a firewall for the packets. The rule set is often modified within the system and is not a static table. Thus any implementation must be able to dynamically alter the rule set periodically.

As the number of machines and rules increase, the scalability of virtual switching becomes a crucial problem. Theoretically optimal algorithms require substantial pre-processing time which cannot accommodate frequent addition and deletion of rules. In practice, simpler implementations are used. In OVS, the classification algorithm is to hash the headers of each incoming packet into tables to determine which rules may apply to a given packet. Since a given rule may only apply to certain portions of the header (due to wildcard fields), such as the destination IP or the port, the packets must be masked to check if a rule applies to them. Since different rules apply to different wildcard masks, for each wildcard mask a distinct hash table is needed. Generally, there are around 20 to 30 distinct masks in an instance of OVS, resulting in the need to store just as many tables. Upon the arrival of each packet, each wildcard must be applied to its header and then hashed into the respective table. The current state-of-the-art implementation simply traverses the tables in a random order because it is difficult to predict which table will contain the matching rule.

In this paper, we consider optimization to the OVS packet classification algorithm. In particular, we consider adding an additional first level of indirection for packet classification prior to the table look-up to determine whether the look-up will find a match or not. Our protocol is based on the usage of Bloom Filters [1] to quickly test set membership for potential packet classification in a table prior to the look-up.

2. OVS CLASSIFICATION

OVS employs a multi-layer system for packet classification. The layers are ordered so that look-ups proceed in decreasing order of efficiency, yet in increasing probability of finding a match. Each rule applies to some header values of the packet, but not necessarily all. For instance, some rules will apply only to certain source or destination IP addresses or ranges of addresses, while some will apply to both and to the timestamp and/or the port. The result is a collection of *mask types*, where for each combination of fields to which the rule applies we obtain a different mask. To determine whether a rule applies to a given packet, the packet header must be masked by each type and checked. Thus incoming flows first proceed to an exact-match cache, which will recognize cached frequent flows and quickly classify them. If a miss is incurred in the exact match layer, the flow proceeds to the "megaflow" layer, wherein the aforementioned masking occurs.

The megaflow layer is where our optimizations are primarily concerned. In the megaflow layer, each mask type is applied to the packet header, and the result is hashed into a table which contains all the rules which apply to that mask type. The first hit in a table that is found is taken. Note that we are guaranteed that no more than one rule can apply to a given packet, so this behavior is correct.

Hash tables in OVS are implemented using an optimized implementation of Cuckoo hashing [11]. Unfortunately, while a single table lookup is fast, the number of required table lookups grows as the number tables increases. As a result, the aggregate time spent performing table lookups becomes a significant overhead when 20 or more tables are used. Further, table lookups become more costly as the number of rules increases. This is due to the additional size of the in-memory working set. Figure 1 shows our profiling results. We use Intel VTune to profile OVS while applying uniform traffic across all of the rules. Samples were collected over a period of 120 seconds in both cases.

3. BLOOM FILTER LAYER

Our proposed improvements rely primarily on the addition of a new layer of indirection prior to the hash table look-up. Instead of traversing through the hash tables until a hit is found, each hash table is instead