Hello Martin, hope you had a great vacation. Below is my quick recap on the project’s progress.

1. I have generated different sets of synthetic classification rules: 10, 97, 516 and around 10000. I used a classbench open-source tool for this: <https://github.com/classbench-ng/classbench-ng> . Then I used a python script to convert these rule-sets into iptables rules format.
2. I have ran a few performance tests on my own machine, then along with the virtual machine installed in it, and on the machine in the lab, with these rule-sets for:
   1. standard Linux iptables module
   2. pcn-iptables (XDP based analog of iptables): <https://polycube-network.readthedocs.io/en/latest/components/iptables/pcn-iptables.html>

The test involves the following steps:

1. On the server (separate Linux network namespace or VM):

– Install all rules from the set to the filter module

– Start the nginx server

1. On the client side run wrk – HTTP load generator (<https://github.com/wg/wrk>) to stress the server network stack. Collect wrk numbers (query per second), plus perf report and system statistics (mpstat).

So far, I did a “positive” test – only valid traffic was sent, so none of the rules in these sets were hit

1. A few observations were made:
   1. The ipfilter performance degrades significantly, as number of rules in the set grow: from ~400K QPS with no rules to ~37K QPS with 10K rules.
   2. pcn-iptables in contrast showed very modest performance slowdown even with significant number of rules: ~350K QPS with 5000 rules.

However, I have encountered several quit significant obstacles while trying to use pcn-iptables:

* + - It doesn’t support rules with a range of source/destination ports.
    - As the number of rules increases, the speed of adding new rule drops dramatically – in order to add around 10k rules, it took around 22 hours for the VM to do so and then it hanged.

Due to this, I have tested pcn-iptables only with the first 5000 rules.

As I understood, this happened due to the pcn-iptables design principle: internally, it is a cascade of a eBPF MAP tables where every single table manages a single matching protocol field of the current rule-set. Each table contains the list of unique values for that field present in the given rule-set, in addition, a wildcard for rules that do not care for any specific value. Each value in the table is associated with a bit-vector of length N equal to the number of rules, in which the ith ‘1’ bit tells that rule i can be possibly matched when the field assumes that value.

So adding a new rule causes the increase of the bit-vector size and re-generation of all the values within the internal MAP table.

More details on pcn-iptables internal design can be found at: <https://mbertrone.github.io/documents/21-Securing_Linux_with_a_Faster_and_Scalable_Iptables> .

1. Such limitations look very serious and I think made it impossible to use pcn-iptables in a real production environment.

So, I went ahead and tried to implement my own XDP based program that would provide functionality similar to a stateless iptables rules. To do that I used DPDK ACL library as a starting point: <https://doc.dpdk.org/guides/prog_guide/packet_classif_access_ctrl.html>. It takes a whole rule-set at once and generates a set of internal tables that can be used later by data-plane to perform fast rules matching.

Internally it contains a set of 8-bit tries that are represented as a flat array of what they call “transitions”. During the search it performs a walk over this trie (jump through array of transition values) to find a matching rule with the highest priority.

So I did three main things:

* 1. I modified the DPDK test-acl user-space program (<https://github.com/napatech/dpdk/blob/master/app/test-acl/main.c> ) to load the generated by DPDK ACL lib “transitions array” into the BPF MAP array table.
  2. Wrote an XDP program that implements search algorithm over loaded BPF MAP array table.
  3. Verified that given search works as expected (all rules can be matched successfully).

The results are quite promising with the same 10K rules, running nginx/wrk test I got ~350K QPS.

1. Things that are missing:
   1. Apart from a “positive” test with only valid traffic, I would also like to add a “negative” tests, where along with valid traffic from wrk, I plan to generate some malicious traffic that would hit all the rules added (simulate DDoS attack on the server) and collect the performance statistics. This is generated using created pcap files from the trace files for each rule-set and performing the test using scapy.

I plan to do that again for all three filter implementations:

* + - standard Linux ipfilter
    - pcn-iptables
    - XDP ACL program

for all the previously generated rule-sets.

* 1. Compile the final project report.

Right now, due to the exams I am trying to balance between them and working on my project.

However, once I will have completed my last exam on August 8th, I will concentrate completely on the project and will try to produce a first draft of the final report by the August 12th.