Sieve

Source code of Sieve, SDN-based flow scheduling framework. This framework is a distributed solution where some modules run in SDN control plane and other parts inside the devices of the data plane. Sieve consists of three layers, namely, the first layer resides in the data plane and it is implemented in OVS switches by OpenFlow13 flow tables, group buckets and proactive flow entries. We employ a sampling algorithm using OpenFlow group bucket, which sends a portion of the arrived flows at edge layer switches to the SDN controller to be scheduled by Sieve’s first layer. On the other hand, the remaining flows, will be scheduled by ECMP. Consequently, we mitigate Sieve load, sample a portion of the network flows and reduce ECMP-caused packet collisions. The second layer runs as a part of the control plane, its main purpose is finding the shortest paths and installing OpenFlow13 flow entries for new flows after receiving *Packet-in* messages from data plane. The third layer detects elephant flows on edge switch ports upon threshold hits, then it tries to find alternative routes for the detected elephant flows to be rescheduled.

It consists of the following source code files:

The first layer:

* fattree.py: creates the topology used to evaluate Sieve's performance. The employed topology is Fat Tree whose scale factor is 4.

The second layer:

* sieve.py: receives the packet-in from the OVS switches of the data plane. It tries to find the shortest path between the source and destination. It installs new flow entries. In addition, it invokes the network monitoring and discovering functions.

The third layer:

* network\_monitor.py: this module probes switches in the data plane, to get statistics about ports and flows. It detects and reschedules elephant flows upon threshold hits. The threshold hits when the available BW on an edge layer switch port decreases below 25% of the link capacity.

Network topology discovering:

* network\_awareness.py: discovers network topology and provides topology information to other modules. In addition, it monitors the network situation.

Topology is based on OVS switches. Since Fat Tree topology is part of the data plane, we use the following OVS and OpenFlow programs to create and configure OVS switches:

* set OpenFlow protocol version 1.3:

sudo ovs-vsctl set bridge %s protocols=OpenFlow13

define proactive openflow flow entries:

ovs-ofctl add-flow sw -O OpenFlow13 \

'table=0,idle\_timeout=0,hard\_timeout=0,priority=1000,arp, \

nw\_dst=10.1.0.1,actions=output:portnumber

* define group buckets:

ovs-ofctl add-group sw -O OpenFlow13 \

'group\_id=3,type=select,bucket=output:portnumber1,bucket=output:portnumber2

In the control plane, we employ APIs provided by RYU controller to implement Sieve functionalities. The used RYU APIs comply with OpenFlow 1.3 specifications, so Sieve’s functionalities can be applied by any SDN controller provides similar APIs given that the data plane consists of OpenFlow capable devices. In this context, most of the open-source and commercial SDN controllers provide such APIs.

Sieve employs the following OpenFlow13 messages:

* OPF\_PORT\_STATS: sent from the controller to switches to get port statistic
* OPF\_PORT\_DESCRIPTION: sent from the controller to switches to get port description
* OPF\_FLOW\_MOD: sent from the controller to switches to add new flows
* Packet-in: sent from a switch to the controller in case of table miss
* Port-status: sent form a switch to the controller upon port status changes.

The following figure depicts the solution architecture:

OVS

Sieve modules

Data plane

Control plane

Openflow v1.3

For further information, you can read the article: https://www.sciencedirect.com/science/article/pii/S0140366421000761