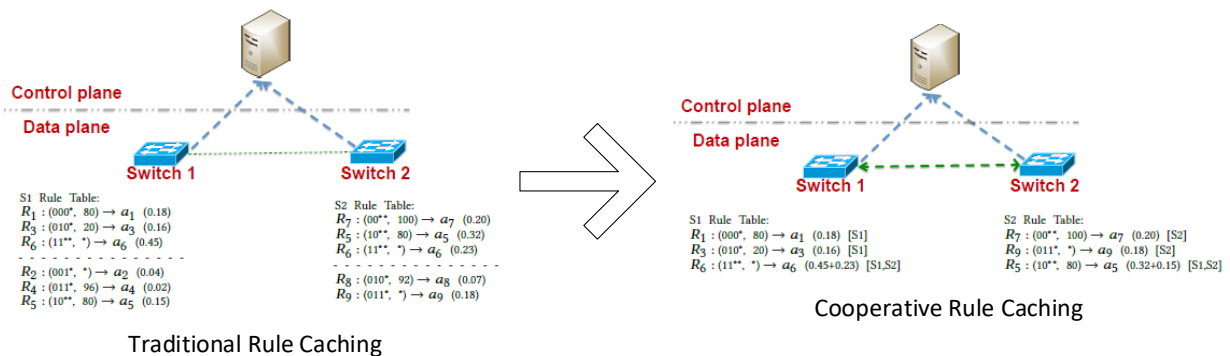# Cooperative Rule Caching in Software Defined Network (SDN) Switches

Abstract:

Classification is a building block in many network services such as monitoring and policy enforcement. In commodity switches, classification is performed by rule tables of various matching patterns (exact match, longest prefix match or ternary match). The memory components are fast and power-hungry, but at the same time are of limited size. To deal with the memory limits, rule caching approach is implemented in traditional switches, where it is performed independently in each of them.

A new approach is taking advantage of the similarity between the policies expressed in the various network switches through cooperative rule caching. Cooperative caching improves the performance of a distributed caching system by coordinating the cache policies among clients.

In addition, the centralized control of SDNs naturally motivates cooperative caching. By allowing packets to be forwarded to other switches to complete the classification process within the data plane, the load on the control plane and the time to resolve requests are reduced.

Control plane
Data plane
Switch 1   Switch 2

S1 Rule Table:
R_1 : (000*, 80) $\to a_1$ (0.18)
R_3 : (010*, 20) $\to a_3$ (0.16)
R_6 : (11**, *) $\to a_6$ (0.45)

R_2 : (001*, *) $\to a_2$ (0.04)
R_4 : (011*, 96) $\to a_4$ (0.02)
R_5 : (10**, 80) $\to a_5$ (0.15)

S2 Rule Table:
R_7 : (00**, 100) $\to a_7$ (0.20)
R_5 : (10**, 80) $\to a_5$ (0.32)
R_6 : (11**, *) $\to a_6$ (0.23)

R_8 : (010*, 92) $\to a_8$ (0.07)
R_9 : (011*, *) $\to a_9$ (0.18)

Traditional Rule Caching

Control plane
Data plane
Switch 1   Switch 2

S1 Rule Table:
R_1 : (000*, 80) $\to a_1$ (0.18) [S1]
R_3 : (010*, 20) $\to a_3$ (0.16) [S1]
R_6 : (11**, *) $\to a_6$ (0.45+0.23) [S1,S2]

S2 Rule Table:
R_7 : (00**, 100) $\to a_7$ (0.20) [S2]
R_9 : (011*, *) $\to a_9$ (0.18) [S2]
R_5 : (10**, 80) $\to a_5$ (0.32+0.15) [S1,S2]

Cooperative Rule Caching

Goals:

1. *Read the paper Cooperative Rule Caching in SDN Switches (Ori Rottenstreich)*
2. **Raise a topology with OpenFlow Switches using Mininet. For example:**
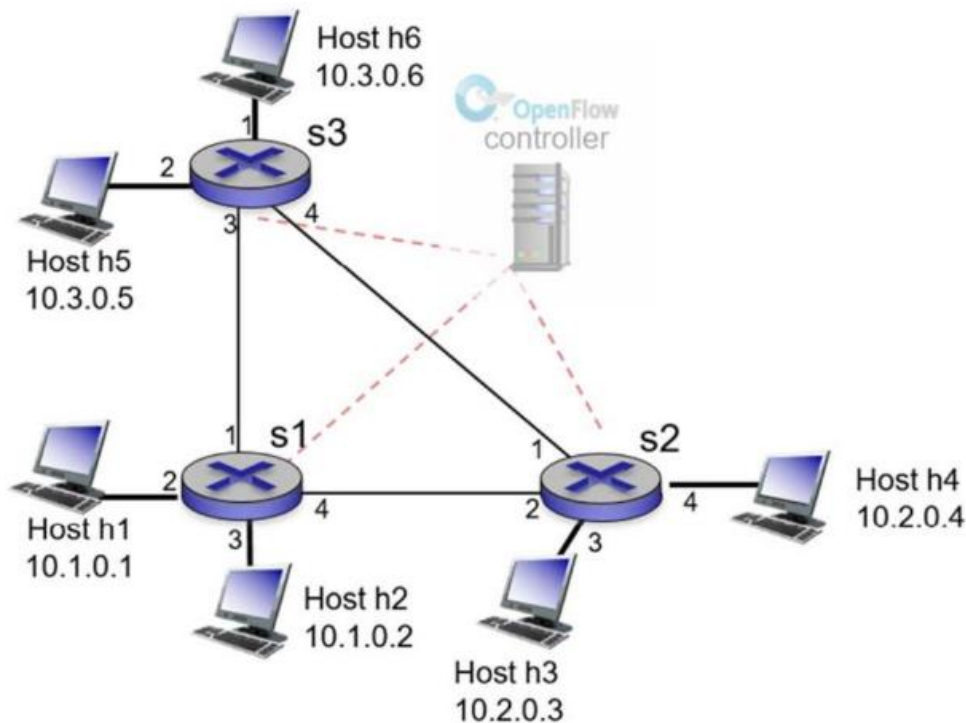
3. Install the ClassBench-ng tool from https://github.com/classbench-ng/classbench-ng and use it for synthetic rules generation to inject the switches.
4. Implement the Cooperative Rule Caching algorithm and evaluate it on the topology setup. The evaluation should verify traffic performance for various scenarious:
   - Rule dependencies (exact match, prefix match, wildcard match)
   - Number of switches in the setup

Requirements:

Introduction to Networking (Must), Internet Networking (Optional)

Programming Language:
C++ or Python

Guided by:
Prof' Ori Rottenstreich