CS6250 Project MileStone 3

Yeli Zhu, Ruiming Lu, Chengwei Li, Mengfan Jiang, Lixi Zhao

Topic: Traffic engineering framework with machine learning in SDN

1 Introduction

Software-defined networks (SDN) is a paradigm that separates the control plane and data plane. In the control plane a central software program, called controller, makes all control decisions and manages the overall network behavior. While in the data plane, network devices become simple packet forwarding units, called forwarding elements. Controller communicates with these network-wide distributed forwarding elements via standardized interfaces like OpenFlow protocol.

The routing problem is typically solved by the shortest path first algorithm with some simple criterion. Although this method is fast and allows large scale problem, it does not consider the current flow state within the whole network. That means it will make the usage of network inefficient and even course the heavy load network much more worse. Some scholars proposed some NP-complete algorithms to route the network considering the current network load and do the load balance to make use of the network. But the classical heuristic algorithms takes a long time to calculate the routing path. In our project, we design a machine learning algorithm by learning the data (network topology, link capacities, current network traffic), then speedup the routing path calculation to make sure we get a good performance within short time.

In this Milestone, we completed the designs of topology of the network, traffic generation and current network traffic measurement. We also discussed the heuristic routing algorithms and did some researches on the artificial neural network. We finished the blocks to build out top algorithms and try to finished the algorithm design in the next Milestone.

2 Progress

2.1 Overview

Since the goal is to partially replicate the results in [1] using Mininet, we have identified some building blocks that we have to use to accomplish the final goal,

which will be explained in 2.2 - 2.6 in this section, and the plan about how to put things together using those building blocks will be discussed in 2.7.

2.2 Topology

We use 2 different network topologies to run our experiments, exactly as what [1] is using. One topology is called KL-graph [6], which has 15 hosts and 27 links. The other one is NSFNet topology, which has 14 nodes and 21 links.

We build our topologies on Mininet using its API, where we create a switch for each node in the graph and create a link between 2 switches for every edge in the graph. Also, every switch is attached with a different host by the same kind of link. Following are some configurations we use currently:

Link Bandwidth: 10M

Link Delay: 10ms

CPU Bandwidth Limit on Hosts: 0.15

Controller: a remote controller based on pyretic and pox (explained later)

2.3 Pyretic

Since we have loops in both of the network topologies, we need something more than the default OVS controller. Also, if we use a spanning-tree-like forwarding policy, some of the links in the network will never be used, which will cause a problem that the CNS data collected by us will be biased. Therefore, we need something by which we can implement dynamic policies.

We use Pyretic [8] to implement a customized controller, where we can define its initial behavior to be using a simple spanning tree for forwarding, but it can be dynamically changing according to the results of heuristic algorithms, through which we can generate a more realistic set of data.

2.4 Traffic generation

After successfully created the experiment network topology, we assume for each node pair in topologies has its own traffic demand. We generated abundant traffic flow by sending UDP traffic from each node to a random destination node with random chosen rates. Through this way, we are able to simulate the real network environment.

In order to allow every node to send and receive data traffic, we config them as server mode. All nodes will be listening to port 5001. We use mininet Python API to control each node's behavior. For each node in our topology network, we use "iperf -u" command to generate UDP traffic flow to a random node .

Pseudocode are described below:

Algorithm 1 Pseudocode for generating traffic flow

```
1: procedure GENERATE-TRAFFIC-FLOW
2:     for each node i ∈ N do
3:         set node as UDP server, listening to port 501
4:     end for
5:     for each node i ∈ N do
6:         send packet to a random node
7:     end for
8: end procedure
```

2.5 Traffic Measurement

The measurement of the traffic would determine the current network status of the network, Abundant background traffic are generated by sending UDP traffic from each node to a random destination node with random chosen rates. And each time for the calculation of the path for the choosen pair, the SDN controller would first obtain the quasi real-time link load information t_{ij}. We use a ubuntu command line tool for collection of current network status.Since the topology is represented as a directed graph, for KL-graph with 15 nodes and 27 links, the CNS contains 54 dimensions and for NSFNet it contains 42 dimensions.These status would be output into files at a set interval we set to allow for previous calculation of the path to finish.

The collection of the background traffic would take place while the calculation of the path of sending data to destinated switch.We would record 100K items of discrete CNS alone with the 100K times calcution of the forwarding path for one pair of node.

After the measurement, the machine learing algorithm would take these 100K input for the training of the neuronetwork. One potential problem would be the uneven distribution of the calculation of the paths, as the small scale of the calculated graph. We would select a fixed number of optimal paths with regarding to the occurance frequency in the result. And divide the pre-processed data into trainning sample and test sample.

The path itself would be a lable for the classification, and CNS would be the features of the network. The collected feature is the link state of the graph.

Currently we have built up the topology of the network in Mininet-VM and could get the basic link rate to construct the CNS of the graph.The next step would be adopt the heuristic algorithm to get the training data for the machine learning module.

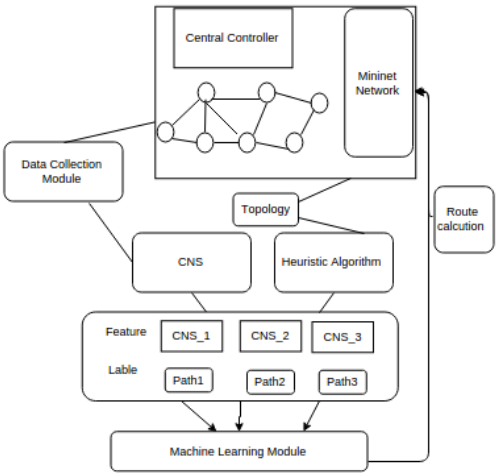


Fig 1 Measurement workflow

2.6 Heuristic routing algorithm

There are some researches on the classical heuristic algorithms for dynamic routing: 1) optimizing the OSPT weight [7]; 2) using ant colony optimization and swarm intelligence [8]; 3) MIN-MAX LOAD algorithm [9].

To implement the Heuristic routing algorithm, we choose a simple way. Because we already know our constrains:

$$f_T(p) = \sum\nolimits_{(i,j)\in Paths(I)} t_{ij} \tag{1}$$

$$f_D(p) = \sum\nolimits_{(i,j)\in Paths(I)} d_{ij} \tag{2}$$

where d_{ij} as cost metric for link $(i, j) \in E$, t_{ij} denotes the current measured traffic amount on this link. On the other word, d_{ij} is the delay measurement. Our goal function is:

$$p^* = argmin(f_T(p)|p \in P_{sd}, f_D(p) < D_{max}) \tag{3}$$

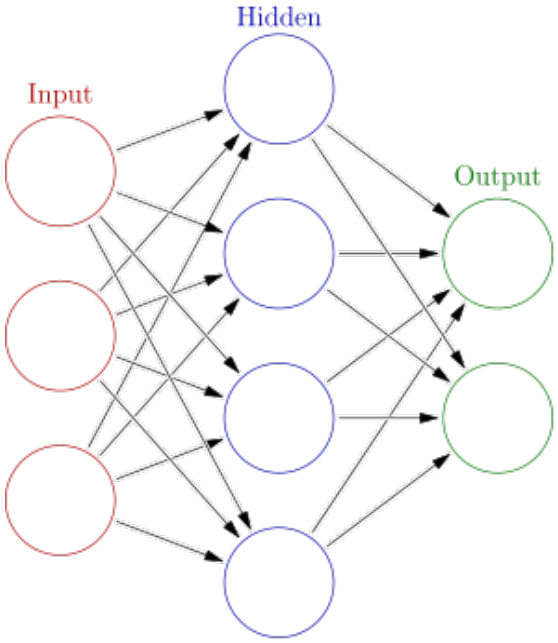
where p_{sd} denote the set of all the paths from source node S to destination node d. By solving above goal equation with the constrains, we could apply DFS algorithm and setting the prune condition.

2.7 Artificial neural network algorithm

In machine learning, artificial neural networks (ANNs) are a family of models inspired by biological neural networks and are used to estimate or approximate functions that can depend on a large number of inputs and are generally

unknown. Artificial neural networks are generally presented as systems of interconnected "neurons" which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning.

For our task here, a neural network for dynamic routing is defined by a set of input neurons which may be activated by link states of current network state. After being weighted and transformed by a function, the activation of these neurons are then passed on to other neurons. This process is repeated until finally, an output neuron is activated. This determines which path to be chosen.



In our experiment, for each access node pair (s, d) where $s \neq d$, we train a module. That means, in a network which contains n access nodes, we will construct A_n^2 pair modules. Each pair module is responsible for the routing decision of corresponding source and destination nodes. As a supervised machine learning module, we adjust the parameters of neural network according training sample's features and label.

Once all the pair module are converged after training process, they can directly give heuristic-like results independently, without the need of time consuming heuristic algorithm any more. After controller received a request, first we extract the source and destination node ID and determine which pair module is the computation unit. Then each machine learning module receives the corresponding requests, and takes the global current network state as its input. Since our supervised learning algorithm has already built a concise model of the

distribution of path labels in terms of current network state features. So in the prediction phase, the resulting classifier can directly assign path labels to them. Then using the path ID as index, we can easily get the corresponding path from its path database.

2.8 Putting Things Together

After we've got all the pieces above, we are going to

1. Generate random traffic on each of the 2 graphs periodically
2. Measure CNS periodically (only bandwidth used on each link)
3. For each pair of nodes and each CNS, feed the node and CNS to heuristic algorithm and compute paths
4. Take the input and output of heuristic algorithm, and feed it into neural network module
5. Calculate the test error of neural network
6. Measure the speed of heuristic algorithm and trained neural network algorithm and compare
7. * If we have enough time, we can let the controller take the output of heuristic algorithm and change the forwarding policy, to generate a set of more realistic CNS data

3 Summary

In this project, we present a framework with supervised machine learning based meta-layer for dynamic routing in real time. Unlike classical heuristic algorithms, which lead to high computational time cost, our machine learning based model can directly give a heuristic-like result in real time. On the other hand, our framework effectively enhance the network performance by taking all useful network information into account, which commonly used min hop routing does not.

References

[1] Li Yanjun, Li Xiaobo, "Traffic engineering framework with machine learning based meta-layer in software-defined networks" *Network Infrastructure and Digital Content (IC-NIDC), 2014 4th IEEE International Conference*, pp. 121-125, Sep 2014.

[2] Lan F.Akyildiz, Ahyoung Lee, "A roadmap for traffic engineering in SDN-OpenFlow Networks" *Computer Networks*, pp. 121-125, Sep 2014.

[3] A Sridharan, R Guerin and C. Diot, "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks" *IEEE/ACM Transactions on Networking (TON) 13, vol. 2*, pp.234-247 2005

[4] T. V. Lakshman, "Traffic engineering in software defined networks" *INFOCOM, 2013 Proceedings IEEE*

[5] Kwang Mong Sim and Hong Sun Weng,"Ant colony optimization for routing and load-balancing: survey and new directions" *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, vol. 33, no. 5*, pp.560 -572 2003

[6] Kodialam, Mudi, and T. V. Lakshman. "Minimum interference routing with applications to MPLS traffic engineering." *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*. Proceedings. IEEE. Vol. 2. IEEE, 2000. APA

[7] Fortz, Bernard, and Mikkel Thorup. "Internet traffic engineering by optimizing OSPF weights." *INFOCOM 2000. Nineteenth annual joint conference of the IEEE computer and communications societies*. Proceedings. IEEE. Vol. 2. IEEE, 2000.

[8] Dorigo, Marco, et al., eds. Ant Colony Optimization and Swarm Intelligence: 6th International Conference, ANTS 2008, Brussels, Belgium, September 22-24, 2008, Proceedings. Vol. 5217. Springer, 2008.

[9] Kodialam, Mudi, and T. V. Lakshman. "Minimum interference routing with applications to MPLS traffic engineering." *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*. Proceedings. IEEE. Vol. 2. IEEE, 2000.

[10] Pyretic: http://frenetic-lang.org/pyretic/