TRAFFIC ENGINEERING FRAMEWORK WITH MACHINE LEARNING BASED META-LAYER IN SOFTWARE-DEFINED NETWORKS

Li Yanjun, Li Xiaobo, Yoshie Osamu

Graduate School of Information, Production and Systems, Waseda University, Fukuoka, Japan
liyanjun@ruri.waseda.jp, leesusu@ruri.waseda.jp, yoshie@waseda.jp

Abstract: Software-defined networks is an emerging architecture that separates the control plane and data plane. This paradigm enables flexible network resource allocations for traffic engineering, which aims to gain better network capacity and improved delay and loss performance. As we know, many heuristic algorithms have been developed to solve the dynamic routing problem. Whereas they lead to a high computational time cost, which results in a crucial problem whether such a heuristic approach to this NP-complete problem is of any use in practice. This paper proposes a framework with supervised machine learning based meta-layer to solve the dynamic routing problem in real time. We construct multiple machine learning modules in meta-layer, whose training set is consist of heuristic algorithm's input and its corresponding output. We show that after training process, the meta-layer will give heuristic-like results directly and independently, substituting for the time-consuming heuristic algorithm. We demonstrate, by analysis and simulation, our framework effectively enhance the network performance. Finally, the meta-layer architecture is quite universal and can be extended in numerous ways to accommodate a variety of traffic engineering scenarios in the network.

Keywords: software-defined networks, meta-layer, routing, traffic engineering, machine learning

1 Introduction

Software-defined networks (SDN) [1-3] is a new 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.

In this architecture, the controller has complete knowledge of network, including network topology, link capacities, current network traffic pattern (inferred from the measurements at the forwarding elements) and so on. Using these information, traffic engineering policy or management function residing at the controller can select routing according to our demands.

978-1-4799-4734-8 /14/$31.00 ©2014 IEEE

The routing problem is typically solved by shortest path first (SPF) algorithm with some simple criterion such as hop-count or delay. Although the simplicity of this approach makes it much quickly to calculate result and allows routing to scale to very large networks, it does not make the best use of network resources [4].

So in large Internet backbones, service providers have to concern with performance optimization of operational networks. However, due to its NP-completeness, an exact algorithm for traffic engineering, either is infeasible or works well in small networks only. So a lot of researches focus on traffic engineering, and have proposed some classical heuristic algorithms [5-7] to manage the traffic flow effectively, including in a partial deployment of SDN capability environment [8]. Whereas they lead to a high computational time cost, which results in a crucial problem that the heuristic approach could not calculate the optimal path in real time. That is the main obstacle for applying these advanced algorithms in practice.

The motivation for our problem arises from the need to get the heuristic-like results in real time. So in this paper, we consider traffic engineering in the case where there exists a powerful enough heuristic algorithm which has ability to calculate the optimal result satisfying the quality of service (QoS) request, even if it has a high time cost. In addition, we consider the routing problem in such a reliable network case where the topology connection condition will not change. In our proposed routing framework with machine learning (ML) based meta-layer, the input of heuristic algorithm and its corresponding output optimal result will integrate as our machine learning models' training set. After completing training process, the meta-layer can substitute for the complex heuristic algorithm in our framework. And under the new network state, when controller receives a dynamic routing request, the machine learning model can directly give a heuristic-like result in real time.

The rest of this paper is organized as follows. In Sect. 2 we briefly provide the description of network environment and formulate controller's traffic engineering optimization problem. Sect. 3 formally presents our proposed machine learning based meta-layer architecture. First we describe its structure and function in detail. Then we show how to use it to make the dynamic routing decision. Sect. 4 presents the

simulation results to evaluate the performance of the proposed approach. Finally we conclude this paper in Sect. 5.

2 Network description

In our paper, we consider a network topology where a centralized SDN controller connects with all the nodes via standardized protocol OpenFlow. For dynamic QoS routing problem it is crucial to select a cost metric and constraints where they both characterize the network state and support the QoS requirements. In our model, a network is represented as a directed graph $G = (V, E)$, where V is the set of switches and E is the set of links. The capacity of a link $(i, j) \in E$ is denoted by c_{ij}. In a reliable network, the graph G and the value of c_{ij} are constant, which are collectively called topology information. Refer to Ref. [9]'s idea and we define the following quantization rule. We select the t_{ij} and d_{ij} as cost metric for link $(i, j), \forall (i, j) \in E$, where t_{ij} denotes the current measured traffic amount on this link and d_{ij} is the delay measure. The set of all the links cost metrics is defined as current network state (CNS). Let P_{sd} denote the set of all the paths from source node s to destination node d. For any path $p \in P_{sd}$, we define traffic amount f_T and delay amount f_D measure as,

$$f_T(p) = \sum_{(i,j)\in p} t_{ij}$$

$$f_D(p) = \sum_{(i,j)\in p} d_{ij}$$

In order to balance network load and decrease delay, in our case, the dynamic routing problem can be stated as calculating a path p which minimizes the traffic amount function subject to the delay parameter to be less than or equal to a specified value $D_{\max}$. The objective function is shown as follows,

$$p^* = \arg\min_{p} \{f_T(p) | p \in P_{sd}, f_D(p) \le D_{\max}\}$$

A quality of service request for establishing path from s to d is defined as a tuple (s, d, b), where b is the bandwidth request of traffic engineering. All QoS requirements for the flow can be assumed to have been folded into the bandwidth b [10]. Usually if we use a certain heuristic algorithm, given a request (s, d, b), it can calculate the optimal path from s to d, which satisfies the QoS requirement b.

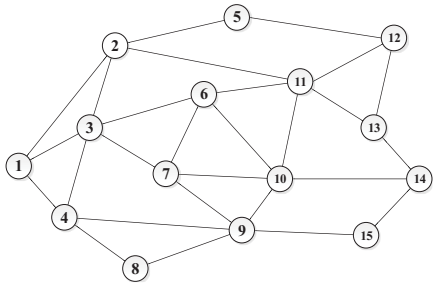

Figure 1 SDN topology

An example of SDN network is shown in Fig. 1, which is from Kodialam-Lakshman [10]. We call this network KL-graph. In it we assume that all the links are duplex, and all link capacity c_{ij} are identical. Without considering the propagation delay, for the shortest path computation, link weights are set to one. This network will be used to elucidate some concepts and model constructions that we outline in the rest of paper and it is one of the several networks on which we present our simulation results.

3 Machine learning based meta-layer architecture

Our proposed traffic engineering framework is shown in Fig. 2. It contains a heuristic algorithm layer and a machine learning based meta-layer. First we use the heuristic algorithm's input and its corresponding output to train the meta-layer. Then during the dynamic routing process, meta-layer will give heuristic-like results directly and independently, substituting for the time-consuming heuristic algorithm. Next we will describe the framework in detail.

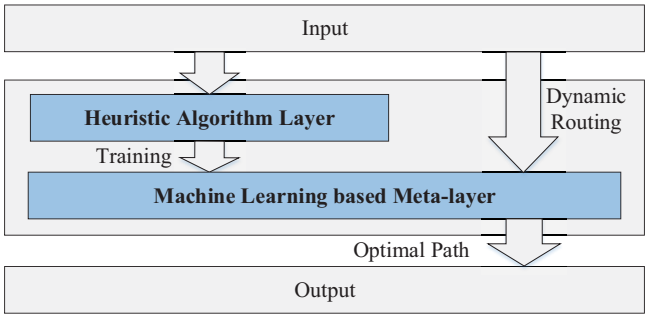

Figure 2 Traffic engineering framework with machine learning based meta-layer architecture

3.1 Pair based logical module

In the network, each node needs to be classified as an access node or a non-access one. Access nodes are those that end stations are directly attached to. According to the characteristic of path requirement, in our framework's meta-layer, we choose to construct a pair module for each access node pair(s, d), where $s \neq d$. 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. For example, when controller receives a flow request from s_i to d_i, this request is delivered to the pair module(s_i, d_i) in our meta-layer, which will give the optimal path finally. Each pair module contains a supervised machine learning module and a corresponding path database which consists of the paths from s_i to d_i that are calculated by heuristic algorithm.

So in the meta-layer, we logically abstract the network into several independent pair modules. Although they are detached to route for a certain ingress-egress pair, they share the same global network state in meta-layer, which ensures that machine learning module decision

will depend on same information used for the heuristic algorithm. To facilitate explanation, in the rest statement, we will use one pair specifically, pair (1, 14) to illustrate the construction process of its machine learning model.

3.2 Training data collection and path database construction

At the first step, we collect abundant training data and build a path database for each pair. The process is shown in the Fig. 3. As mentioned before, we assume that there exist some heuristic algorithms which have ability to take the CNS into account and calculate the optimal result to satisfy the QoS, regardless of its time cost. Since CNS represents the current traffic pattern in the whole network, which is changing all the time. In a series of time scales, even the same request is received, the algorithm's output may be totally different. During our training data collection, we sample the discrete n items of CNS from this time sequential network, according to the occasions when the QoS requests happen. We consider the input of the algorithm as stable topology information, discrete CNS and QoS request. And output is the optimal path from source node to destination one. Among these outputs, there exist a plenty of same paths. Then we compare all these n output optimal paths for this pair, and select the non-overlapping ones total of m to build the path database in this pair based module finally. So the path database contains all the unique optimal paths generated by heuristic algorithm, which are indexed by the path ID from 1 to *m*.

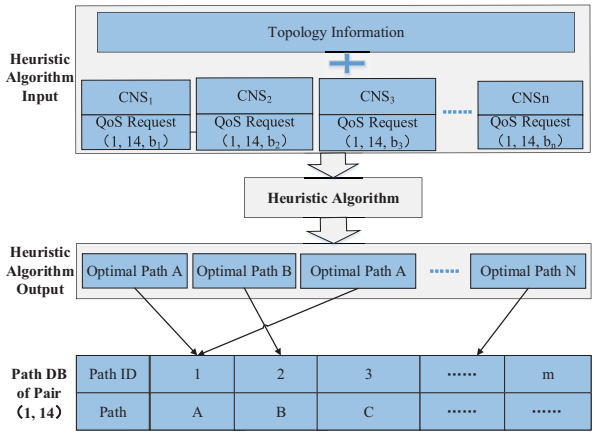

Figure 3 Training data and path database collection process

3.3 Model training

In our meta-layer, we adopt the supervised machine learning module that means a training sample should be consisted of features and its label. According to the preceding assumption, the topology information is unchanged. Meanwhile, in our experiment we learn from a heuristic algorithm which routes the flow through the least load path based on the current network state subject to a predefined invariable threshold value. So for our machine learning module, topology information and bandwidth requirement of QoS are the redundant information, which is not necessary to be considered as the sample's feature.

So first in order to ensure that our module can learn all the possible useful network information and generate approximate results compared with heuristic algorithm, in our simulation we constitute features with the CNS. As for sample label, because we have a complete mapping relation between the output path and path ID for each pair, we project the output optimal path to the numeric path ID, such as 1 or 2, which is considered as corresponding sample's label. With the sufficient n training samples collected from Sect. B, we structure our training set. As shown in Fig. 4, for each pair module, we use the training set to train until converges.

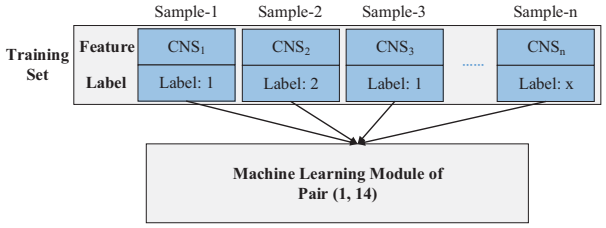

Figure 4 Module training process.

3.4 Dynamic routing decision

Once all the pair modules are converged after training process, they can directly give heuristic-like results independently, without the need of time-consuming heuristic algorithm any more. We illustrate the dynamic routing decision process with Fig. 5.

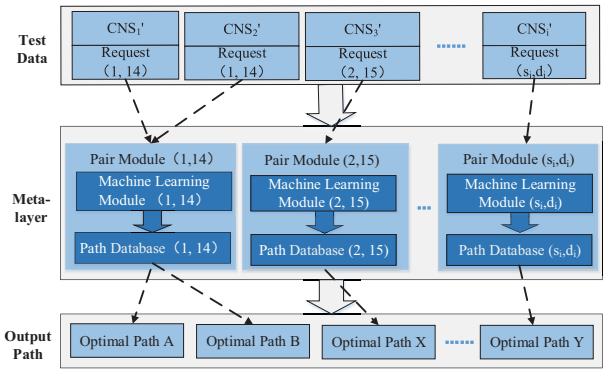

Figure 5 Dynamic routing decision process

After controller received a request, first we extract the source and destination node ID and determine which pair module is the computation unit. In the figure, first and second requests are delivered to the pair module (1, 14), third one is delivered to the pair module (2, 15). 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, given test data, the resulting classifier can directly assign path labels (path IDs) to them. Then using the path ID as index, we can easily get the corresponding path from its path database. Finally controller instructs the switches to forward the flow based on the path.

The above prediction process does not take any time, and it can provide heuristic-like optimal solutions. In other words, if we compare machine learning module with the heuristic algorithm, the higher accuracy of our machine learning module can obtain, the more similar results they can give. If the accuracy can reach 100%, that means these two approaches can give an identical path all the time. Whereas, their running time differs considerably. The heuristic algorithm needs plenty of iterative computations, which costs too much time that is not acceptable for the real network. On the contrary, prediction of machine learning module is a time-saving process that well satisfies the need of solving dynamic routing problem in real time. Then, in the next section, we run a series of experiments to underline the accuracy and effectiveness of our framework.

4 Experimental results

The heuristic algorithm and machine learning based meta-layer are implemented using C++ programming language, and the simulations are carried out on a computer with 3.4 GHz Intel Quad-Core processor and 8GByte memory. In our experiment, we use the following two topologies, one is the network topology shown in Fig. 1, and the other is NSFNet topology with 14 nodes and 21 links. For these two topologies, all the link capacities c_{ij} are assumed to be equal. According to our framework working process, we divide the simulation into three phases.

4.1 Training data collection and path database construction

In this phase, first we use the OMNeT++ simulator as the CNS generator. For a certain network topology, we assume every node attaches to end stations, and for each node pair in topologies has its own traffic demand. Abundant background traffic are generated by sending UDP traffic from each node to a random destination node with random chosen rates. Current OpenFlow protocol supports a counter corresponding to the destination node by packet length. Forwarding elements maintain a counter for each flow, these counters can be inquired and managed by controller. From these counter information, SDN controller can obtain the quasi real-time link load information t_{ij}, which we collect from the simulator. The delay parameter d_{ij} is set to 1 which corresponds to hop count. This is because the current OpenFlow switch do not support to collect delay information. So for machine learning model, d_{ij} is redundant information, which is excluded from input features. 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. With the change of background traffic, we record the 100 k items of discrete CNS. Meanwhile, we choose the classical max-min ant system (MMAS) algorithm [11] as our heuristic algorithm. MMAS is an improved Ant Colony Optimization algorithm deriving from ant system, which has been demonstrated as appropriate approach with high performance for dynamic routing and load balancing [12]. In our experiment, we configure the MMAS by setting the number of ants as 20, the iterative number is 500.

As mentioned above, in the stable network topology, we run the MMAS for each pair under 100 k different CNSs, then get the homologous 100 k optimal paths. Because the results' distribution is quite uneven, which is adverse to the following training process, we conduct the pre-process for the raw data. For example, for the pair module (1, 14) in KL-graph, we select top 9 optimal paths according to occurrences in 100 k results. Each optimal path category has 2.5 k samples. Then they are divided into two groups, one is for training with amount of 2 k, and the other is for testing with the rest 0.5 k. So for this pair, we collect 2 k*9 training samples and 0.5 k*9 test samples, both contain CNSs and path labels. Besides, we construct the path database of pair module (1, 14) with 9 labeled paths.

4.2 Model training

In our experiment, we choose the neural network as supervised machine learning module. According to the training sample's features and label, we adjust the parameters of neural network. For instance, for the pair model (1, 14) in KL-graph, with 2 k*9 training samples, we configure it with 600 units in hidden layer and set CNS as its input, the output layer consists of 9 units corresponding to 9 different labels.

4.3 Dynamic routing decision

In test phase, first we use the test set generated in Sect. 4.1 to measure the accuracy of our machine learning module. The configurations and results of some typical examples are shown in the Table I. The rest pair modules have similar configurations and accuracy.

Table I Parameter and accuracy of neural network

	KL-graph			NSFNeT		
	Module (1,14)	Module (2,15)	…	Module (1 ,14)	Module (2 ,14)	…
Number of features	54	54	…	42	42	…
hidden units	600	800	…	2 000	2 000	…
Number of labels	9	8	…	8	10	…
Weight decay	0.000 03	0.000 03	…	0.000 03	0.000 03	…
Accuracy	84.15%	88.2%	…	83.7%	83.2%	…

Next we carry out an experiment to measure the performance of our meta-layer relative to the shortest path algorithm based on minimum hop routing and MMAS, still using random generated background traffic pattern. Take the pair module (1, 14) in KL-graph as an example. We collect 4 k new discrete samples. We respectively use shortest path algorithm, MMAS, and machine learning module to calculate the optimal path under these CNSs. Here we focus on the network delay performance and plot the comparison among three approaches shown in the Fig. 6.

Note that when the network is lightly loaded, there is little difference between the SPF and MMAS. The beneficial effects of MMAS become evident when the network is partially congested. The delay of SPF obviously becomes larger than MMAS. In contrast, with different network states, ML's delays are almost as same as MMAS's, only in small number of cases they have deviations. That means our framework gives heuristic-like results under the new network states.

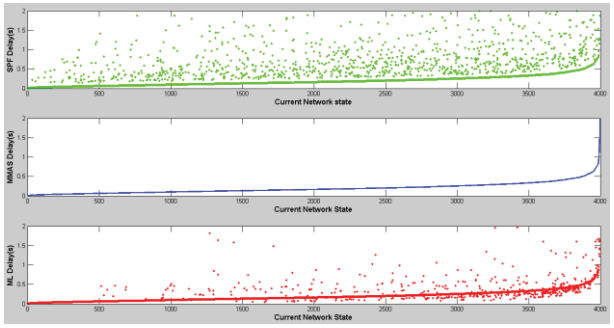

Table II Comparison of running time

Average running time for each CNS	SPF	MMAS	ML
KL-graph	0.011 ms	260.571 ms	0.042 ms
NSFNet	0.009 ms	243.150 ms	0.029 ms

Figure 6 Comparison of delay performance

In addition, another important measurement of dynamic routing problem is the running time of algorithm. The Table II shows the comparison of average running time for each CNS between MMAS and ML in the same computational environment.

Note that the executing time of ML approach is much shorter than MMAS, which is quite acceptable from practical point of view.

5 Conclusions

In this paper we present a framework with supervised machine learning based meta-layer for dynamic routing in real time. We demonstrate, by analysis and experiments that the routing phase of framework is as simple and computationally efficient as the commonly used min hop routing, but it has apparent advantages on network performance. On the other hand, our approach gives the optimal result almost like heuristic algorithm dose which considers the current network load, but it is substantially faster than the heuristic one.

So our framework can solve dynamic routing problem in real time and effectively enhance the network performance by taking all useful network information into account.

Besides, for these two functional layers contained in our framework, there are many advanced algorithms to implement as we mentioned before. So our framework is quite universal and can be extended in numerous ways to accommodate a variety of traffic engineering scenarios in the network.

References

[1] Casado M, Freedman M J, Pettit J, et al. "Ethane: Taking control of the enterprise." ACM SIGCOMM Computer Communication Review 37.4 (2007): 1-12.

[2] McKeown N, Anderson T, Balakrishnan H, et al. "OpenFlow: enabling innovation in campus networks."ACM SIGCOMM Computer Communication Review 38.2 (2008): 69-74.

[3] The Openflow Switch, openflowswitch.org

[4] Awduche D O, and Agogbua J. "Requirements for traffic engineering over MPLS." (1999).

[5] Fortz B, and Thorup M. "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.

[6] Sridharan A, Guerin R, Diot C. "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks."IEEE/ACM Transactions on Networking (TON) 13.2 (2005): 234-247.

[7] Dorigo, M. (Ed.). Ant Colony Optimization and Swarm Intelligence: 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4-7, 2006, Proceedings. Vol. 4150. Springer, 2006.

[8] Sugam Agarwal, Murali Kodialam, and T. V. Lakshman. "Traffic engineering in software defined networks." INFOCOM, 2013 Proceedings IEEE. IEEE, 2013.

[9] Egilmez H E, Dane S T, Gorkemli B, et al. OpenQoS: OpenFlow Controller Design and Test Network for Multimedia Delivery with Quality of Service[J]. eBook and USB produced by Sigma Orionis, 2012: 22.

[10] Murali Kodialam 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.

[11] Stützle, Thomas, and Holger H. Hoos. "MAX–MIN ant system." Future generation computer systems 16.8 (2000): 889-914.

[12] Sim, Kwang Mong, and Weng Hong Sun. "Ant colony optimization for routing and load-balancing: survey and new directions." Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on 33.5 (2003): 560-572.