DeepMDR: A Deep-Learning-assisted Control Plane System for Scalable, Protocol-independent, and Multi-domain Network Automation

Deyun Li, Hongqiang Fang, Xu Zhang, Jin Qi, and Zuqing Zhu, *Senior Member, IEEE*

***Abstract*—This article discusses DeepMDR, which is a deep learning (DL) assisted control plane (CP) system to realize scalable and protocol-independent path computation in multi-domain packet networks. We develop DeepMDR based on ONOS, make it support protocol-oblivious forwarding (POF) in the data plane (DP) and facilitate a hierarchical CP architecture for multi-domain operations, and propose a DL model to achieve fast and high-quality path computation in each domain. Simulation results verify that our DL-assisted routing module achieves better trade-off between path computation time and routing performance than existing approaches. The effectiveness of our proposed DeepMDR is also demonstrated with experiments, which show that it serves inter-domain flow requests quickly with a processing capacity of $\sim 166,000$ messages/sec or higher.**

***Index Terms*—Deep learning (DL), Software-defined networking (SDN), Multi-domain networks, Network automation.**

I. Introduction

OVER past decades, both the number of network devices and the volume of traffic in the Internet have been increasing dramatically. This complicates the tasks of network control and management (NC&M) in at least two aspects, *i.e.*, they need to manage more and more connected devices, and satisfy more sophisticated quality-of-service (QoS) demands in a timely manner [1, 2]. Hence, people are seeking for scalable, efficient, and intelligent NC&M techniques. Following this trend, software-defined networking (SDN) was proposed [3], which decouples the control plane (CP) and data plane (DP) of a network to enhance network programmability. Although the initial success of SDN has confirmed its great potential, further studies are still necessary to address the requirements from practical implementations. For instance, the design of CP needs to be improved to adapt to the multi-domain scenario that divides the DP into multiple administrative domains.

The rationale behind considering the multi-domain scenario is multi-fold. Specifically, it handles the situation where the DP elements are operated by multiple operators, ensures the scalability of NC&M when the DP consists of many network elements or/and covers a relatively large geographical area, and resolves the inter-operability issues when deploying network elements from different vendors. Previously, various CP systems have been developed to support the multi-domain scenario. They allocated at least one controller to each domain (*i.e.*, the domain controller), and organized the controllers in either the peer-to-peer (P2P) or hierarchical manners [4, 5]. The P2P architecture lets domain controllers collaborate distributedly to calculate and set up cross-domain paths [4]. Although the P2P architecture can guarantee the autonomy of each domain, the efficiency of NC&M can be improved if we adopt the hierarchical architecture [5], which places a global controller to supervise the domain controllers.

However, most of the existing multi-domain-capable CP systems were developed based on OpenFlow, which is known to be protocol-dependent. Specifically, as OpenFlow defines the match fields and actions based on existing network protocols, the CP systems based on it have difficulty to be future-proof (*i.e.*, new protocols cannot be introduced on-demand). This issue can be resolved by considering the programmable data plane (PDP), which enables an operator to customize the packet processing pipelines in switches. PDP can be realized with the programming protocol-independent packet processors (P4) [6] or protocol-oblivious forwarding (POF) [7]. Here, P4 defines the methods for writing and compiling packet processing programs, and with POF, one can program a PDP switch in runtime by installing protocol-oblivious flow tables and constructing packet processing pipelines with them.

In addition to supporting PDP, another important NC&M task for a multi-domain-capable CP system to address is path computation. Note that, path computation can be challenging in large-scale networks. For instance, the path computation in an optical network needs to solve the well-known routing and spectrum assignment (RSA) problem [8–10], which is known to be intractable for large-scale problems. Meanwhile, the path computation in a packet network can also be complex, when a batch of flows need to be served simultaneously in a relatively large topology, or/and sophisticated end-to-end QoS demands have to be satisfied [11, 12]. Hence, we need to consider methods other than conventional optimization techniques to solve the path computation in a large multi-domain network. Recently, deep learning (DL) has demonstrated its powerful capability on making timely and smart decisions to tackle complex optimizations in dynamic environments, which motivated researchers to leverage DL to solve path computation [13].

Inspired by the aforementioned advances on PDP and DL-based network automation, we, in this article, discuss the design and experimentally demonstrations of DeepMDR, which is a DL-assisted CP system that can achieve scalable and protocol-independent path computation in large multi-domain

D. Li, H. Fang, X. Zhang, and Z. Zhu are with the School of Information Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, P. R. China (email: zqzhu@ieee.org).

J. Qi is with the Zhongxing Telecommunication Equipment (ZTE) Corporation, Nanjing, Jiangsu 518057, P. R. China.

Manuscript received on August 10, 2020.

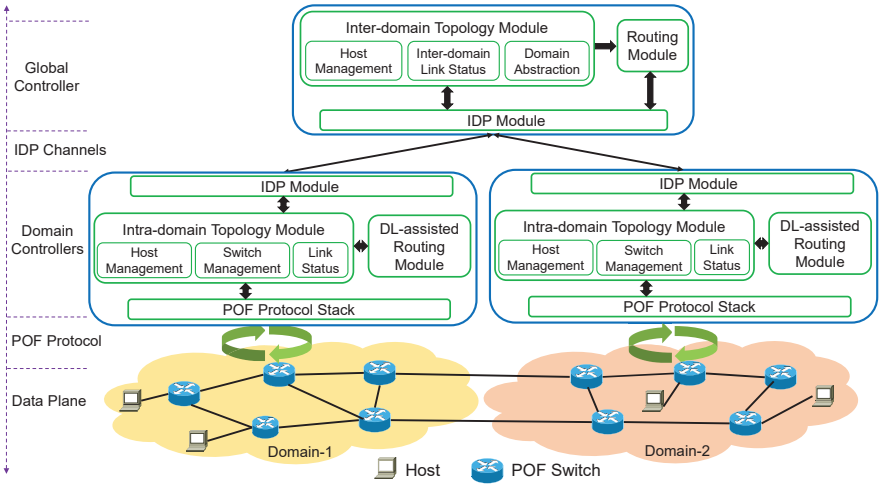

Fig. 1. Overall system architecture of a multi-domain network with DeepMDR.

packet networks. Our DeepMDR is developed based on the well-known ONOS platform. It has the following derived benefits for service providers. Firstly, DeepMDR leverages a hierarchical CP architecture to not only manage a multi-domain network efficiently but also preserve the privacy of each domain. Secondly, DeepMDR uses a DL-assisted routing modules that can achieve better tradeoff between path computation time and routing performance than existing approaches. Finally, our proposal supports POF [7] in DP, and thus it enables service providers to customize their packet processing to adapt to various or even time-varying service demands (*i.e.*, enhanced flexibility and application-awareness).

The rest of the article is organized as follows. Section II describes the overall design of DeepMDR, while the DL model is presented in Section III. We discuss the performance evaluations with simulations and experiments in Section IV. Finally, Section V summarizes the paper.

II. System Design

Fig. 1 shows the overall system architecture of a multi-domain packet network that is supervised by DeepMDR. The DP consists of multiple autonomous domains built with POF switches. We allocate at least one domain controller (DCtrl) to manage each domain, and place a global controller (GCtrl) on top of all the DCtrls to coordinate their operations. Both the DCtrl and GCtrl are developed based on ONOS. Specifically, we add the POF protocol stack in ONOS as a new SDN southbound protocol, with which DCtrls can manage POF switches, while for the control channels between DCtrls and the GCtrl, we design an inter-domain protocol (IDP). Note that, as the major components of DeepMDR are realized based on ONOS, its deployability is similar as that of ONOS.

A. Data plane

We consider the data plane as a multi-domain packet network that consists of end hosts and POF switches. Similar to the well-known P4-based PDP switches [6], POF switches also do not restrict network protocols as known ones. Specifically, POF represents each packet field with a tuple {*offset*, *length*}, where *offset* denotes the start bit-position of the field in a packet and *length* tells the length of the field in bits [7], and it also defines a forwarding instruction set (POF-FIS) based on this type of representation to operate on packet fields. Hence, with POF, flow tables can be composed to process arbitrary packet fields and build protocol-independent packet processing pipelines. Meanwhile, this process is runtime-programmable [7], because an SDN controller can modify the packet processing pipelines in POF switches in runtime by updating the corresponding flow tables dynamically.

B. Domain Controller

When the first packet of a flow arrives at a POF switch, it encodes a *PacketIn* message to send to the DCtrl in its domain. Then, the DCtrl first determines whether the flow is an intra-domain one or not. If yes, it calculates and sets up a routing path for the flow directly. Otherwise, it escalates the *PacketIn* message to the GCtrl for inter-domain path computation. To realize these operations, we implement two key modules in each DCtrl. Specifically, as shown in Fig. 1, the intra-domain topology module maintains the status about the domain, based on which the DL-assisted routing module realizes fast and adaptive path computation. We will discuss the design of the DL-assisted routing module in Section III.

The intra-domain topology module includes the host management, switch management, and link status submodules. The host management records the addresses of hosts and handles address resolution. For instance, if the network uses TCP/IP protocol stack, it deals with the address resolution protocol (ARP). Specifically, it records the IP and Ethernet addresses of a host, as well as the ID and port of the switch to which the host is attached, as an ARP information entry. If it finds out that the IP address in an ARP request is not in its own domain, it will forward the request to the GCtrl using IDP. The switch management controls the switches in the domain. In system initialization, it discovers each switch in the domain and assigns an ID to it. After that, it monitors the switches' status, and installs, updates, and deletes flow tables in them.

The link status submodule utilizes the link layer discovery protocol (LLDP) to detect all the links that are related to the domain. Note that, although ONOS supports LLDP, it does not distinguish between inter- and intra-domain links. Hence, we extend the LLDP packet to include a field about the domain ID. Then, for link status collection, a DCtrl encapsulates the LLDP packet in a *PacketOut* message, and sends it to each switch in the domain, which in turn forwards the message to all of its neighbors. Upon receiving the LLDP packet, a neighbor switch encodes the information regarding the packet's incoming link (including the ID and domain ID of the neighbor switch, and port ID of the link) in a *PacketIn* message to the DCtrl in its domain. Then, by parsing the *PacketIn* message, the DCtrl gets the information of the link, and it determines whether the link is an inter-domain one or not by checking the domain ID. If yes, it will report the information to the GCtrl. Meanwhile, the link status submodule also collects link usages.

C. Global Controller

As illustrated in Fig. 1, the GCtrl also includes two key modules. The inter-domain topology module collects the global topology about the whole multi-domain network. In this module, the host management receives host information from the host management in each DCtrl, and builds a global ARP table that records the IP and Ethernet addresses and domain ID of a host as an entry. Here, each entry is created in the on-demand way and associates with a life time, to maintain the scalability of the global ARP table. The inter-domain link status stores the information regarding all the inter-domain links. The domain abstraction collects the abstracted topologies from all the DCtrls, and combines them with inter-domain links to build the global topology of the multi-domain network. With the global topology, the routing module calculates the domain-level path of each inter-domain flow with an existing routing algorithm, *e.g.*, the weighted Dijkstra algorithm.

To reduce the information exchange between the GCtrl and DCtrls and protect the privacy of each domain, we design each DCtrl to aggregate the links and nodes in its domain to come up with an abstracted topology, and report the abstract topology to the GCtrl. In the abstract topology, each node represents a border switch in a domain (*i.e.*, switches on the source and destination nodes are also considered as border ones), while each link denotes a feasible path between a pair of border switches. To assist inter-domain path computation, the attributes of each link record the hop-count and available bandwidth of the feasible path represented by the link.

D. Inter-Domain Protocol

To coordinate the DCtrls and GCtrl for inter-domain path computation and setup, we design an IDP that includes several types of control messages. When a host needs to transmit packets to another one that is not in its domain, it sends an ARP request to the switch that it is attached to. The switch encapsulates the ARP request in a *PacketIn* message to its DCtrl, which in turn forwards the ARP request to the GCtrl with an IDP message. The GCtrl searches the global ARP table built by its host management submodule, and returns

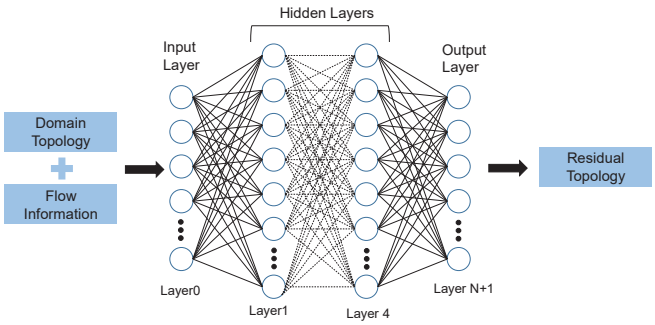

Fig. 2. DNN in our DL-assisted routing module.

the result to the DCtrl, which encodes the ARP reply in a *PacketOut* message to the source switch. Then, the source switch sends the *PacketIn* message for path setup to the DCtrl, which escalates it to the GCtrl using an IDP request. Upon receiving the IDP request, the GCtrl calculates the inter-domain path (*i.e.*, the domain sequence and the input/output link of each related domain), and instructs the DCtrl of each related domain to compute the corresponding intra-domain path segment. Finally, each related DCtrl calculates a path segment with its DL-assisted routing module, encodes the obtained path segment in *FlowMod* messages, and sends them to the corresponding switches to setup the inter-domain path.

III. DL-Assisted Path Computation

A. Network Model for Intra-Domain Path Computation

The DL-assisted routing module in each DCtrl calculates paths and path segments within its domain for intra- and inter-domain flows, respectively. For a DCtrl, the topology of its domain can be modeled as a graph, where the nodes and edges represent the switches and links in the domain, respectively. The graph also records the available bandwidth on each link as a link attribute. Then, the path computation of the DCtrl is to find a proper routing path in the graph, which can connect the source and destination of a flow with sufficient bandwidth. Note that, if the flow is an intra-domain one, the source and destination are just the real ones. Otherwise, the source and destination are the border switches determined by the GCtrl.

Theoretically speaking, the aforementioned path computation can be solved exactly and time-efficiently with the well-known Dijkstra algorithm. Nevertheless, at least two additional considerations should be addressed in practical implementations. Firstly, in a real-world network, flow requests can come in batches, and thus the DCtrl needs to find the routing paths of multiple flows simultaneously. In this situation, the order of the path computations can affect the overall performance of the results, especially when the bandwidth usages on links are relatively high or unbalanced. However, the best order cannot be determined in polynomial-time, because for n requests, the number of orders to check in an exhaustive search is the factorial of n. Secondly, the time complexity of Dijkstra algorithm increases with the size of a topology, while a practical domain can contain many switches and links.

B. Design of DL Model

Hence, we design a DL-assisted routing module to address these practical issues, *i.e.*, not only obtaining a proper order of path computations but also limiting the complexity of each path computation. Specifically, the routing module uses the deep neural network (DNN) in Fig. 2 to analyze the domain topology and information about the flows, and lets it cut certain links to generate a residual topology for each flow. As the time complexity of Dijkstra algorithm is proportional to the total number of links and nodes in a topology, it will run faster in the residual topology. Meanwhile, as the flows are handled one by one with link usage updates in between, the residual topologies are different for different flows. Hence, the DNN actually reserves bandwidth for subsequent path computations to minimize contentions. In this sense, the residual topologies also optimize the order of path computations implicitly.

In Fig. 2, the DNN consists of an input layer, four hidden layers, and an output layer, where the neurons are all fully-connected in between layers. Each neuron in the output layer is binary and corresponds to the status of a link in the residual topology, *i.e.*, if the neuron outputs 1, the corresponding link is kept, and the link is cut, otherwise. The first three hidden layers are residual neural networks [14], which are introduced to expedite the training process and relieve the vanishing gradient and exploding gradient problems, especially for the cases where the size of domain topology is relatively large. The fourth hidden layer is a fully-connected one. Note that, the DNN needs to be trained and verified with realistic data. Specifically, before deploying the DL-assisted routing module in a domain, we need to give it the right domain topology, and train and verify its DNN with the realistic flow information extracted from traces taken in the domain.

C. Training and Operation of DL Model

We obtain the training and testing sets for the DL model in each DCtrl by emulating the path computations of dynamic flows in various network states. Specifically, it is done by leveraging a discrete-time simulation that runs in iterations. In each iteration, we first free the bandwidth occupied by expired flows, and then randomly generate a batch of flow requests. Next, we utilize a simple genetic algorithm (GA) to optimize the path computation order of the requests, and get the routing path of each request under the optimized order. Here, the fitness function of the GA is defined to evaluate both the total bandwidth usage and blocking probability of the requests. Then, the requests are served with the paths obtained by the GA, and if a request can be served, we record its path and other information about it and the domain as a sample. The aforementioned procedure is repeated until we have accumulated enough samples (*e.g.*, $\geq 50,000$ samples).

We put 80% and 20% of the obtained samples in the training and testing sets, respectively. Similar to the residual topology from the DNN, the routing path of each flow can also be represented by a set of binary variables, each of which corresponds to a link in the domain. Hence, the accuracy of the DNN can be obtained by comparing the binary representations of the paths got with it to those of the paths in the samples. Specifically, we define the accuracy as the ratio of the number of matched bits to that of total bits. We train the DNN until its average accuracies over the training and testing sets converge. Then, the trained DNN is put in the DCtrl for online operation. Specifically, for each flow request, the DNN generates a residual topology, which is close to the proper routing path, based on the current network status, and thus it effectively speeds up the path computation with Dijkstra algorithm.

IV. Performance Evaluations

The performance evaluations use both numerical simulations and experimental demonstrations to verify the effectiveness and benefits of our proposal.

A. Numerical Simulations

Our simulations consider 4 domain topologies, which include $\{50, 100, 150, 200\}$ nodes and have their average node degrees as $\{2.95, 3.06, 3.11, 3.12\}$, respectively. In each simulation, we select one domain topology to generate a multi-domain network that consists of 5 identical domains. When the domain topologies with $\{50, 100, 150, 200\}$ nodes are chosen, the obtained multi-domain networks include $\{12, 18, 26, 26\}$ inter-domain links, respectively. The bandwidth capacities of intra- and inter-domain links are set as 100 and $1,000$ units, respectively. Note that, the simulation setup mentioned above is based on the realistic topology data from a major telecommunication equipment vendor in China (*i.e.*, the Zhongxing Telecommunication Equipment (ZTE) Corporation).

The ratio between intra- and inter-domain flow requests is set as $1{:}4$, and the source and destination of each request is selected according to the realistic traffic data provided by ZTE. Also based on the realistic data, we make sure that 90% of the requests have bandwidth demands within $[2, 6]$ units, while the demands of the remaining ones are within $[70, 80]$ units. For the multi-domain networks that have domain topologies with $\{50, 100, 150, 200\}$ nodes, each of their simulations serves $\{480, 925, 1530, 1700\}$ requests, respectively. In addition to our DeepMDR, we consider two benchmarks. The first one replaces the DL-assisted routing module in each DCtrl with Dijkstra algorithm, and is referred to as MD-Dijkstra. The second one (*i.e.*, G-Dijkstra) addresses each multi-domain network as a single-domain one, and uses Dijkstra algorithm to calculate routing paths based on the global information. To ensure sufficient statistical accuracy, we run 100 independent simulations and average the results to get each data point. We perform the simulations in Python 3.6 on a computer with Intel Core i5-7400 CPU and 16 GB memory.

For the DL-assisted routing module in each DCtrl, we train and test it with $40,000$ and $10,000$ samples, respectively. Note that, its accuracy converges fast in the training. For instance, for the largest domain topology considered in the simulations (*i.e.*, the one includes 200 nodes), the training converges within 100 iterations and only takes $\sim$1000 seconds, when the average accuracies of the DL-assisted routing module on the training and testing sets are 99.96% and 99.67%, respectively.

Fig. 3 shows the results on the blocking probability of flow requests, which indicate that G-Dijkstra always provides the

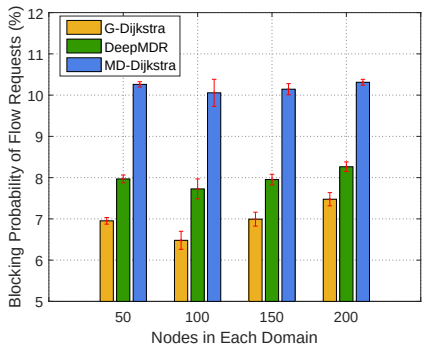

Fig. 3. Simulation results on blocking probability.

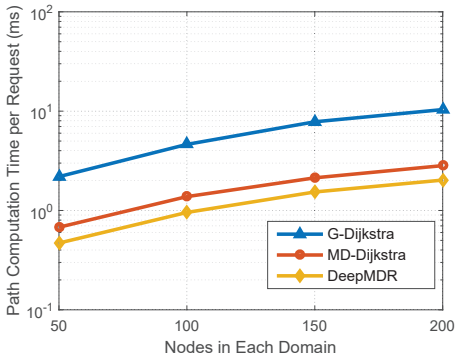

Fig. 4. Simulation results on path computation time.

lowest blocking probability, followed by DeepMDR, while MD-Dijkstra performs the worst. This is because G-Dijkstra has global information for path computation. As for DeepMDR and MD-Dijkstra, each DCtrl only computes the path segment in its own domain while the GCtrl only optimizes the inter-domain routing scheme. Hence, it is difficult for them to achieve the global optimum. However, although G-Dijkstra achieves the lowest blocking probability, it calculates routing paths in large global topologies, which leads to the longest path computation time (as shown in Fig. 4). The results in Figs. 3 and 4 suggest that DeepMDR performs slightly worse than G-Dijkstra in terms of blocking probability, but it runs much faster than G-Dijkstra. Meanwhile, we also confirm that the average path segment lengths in each domain from DeepMDR and G-Dijkstra are almost the same. Therefore, the simulations verify the effectiveness and scalability of our proposal.

B. Experimental Demonstrations

To further verify the performance of DeepMDR, we implement it in a CP system that consists of 5 DCtrls and a GCtrl, each of which is deployed on an independent commodity server. Here, each server equips a 2.10 GHz Intel Xeon Silver 4110 CPU and 16 GB memory, and runs Linux Ubuntu 16.04. The common hardware and software configurations of the commodity servers confirm the deployability of our DeepMDR. Meanwhile, to emulate a practical multi-domain DP, we run Mininet on another server, and use it to generate a multi-domain network that consists of 5 domains. Here, the domains include $\{12, 15, 15, 17, 18\}$ POF switches, respectively, and the average node degree in the multi-domain network is 5.2. Each domain is managed by a DCtrl, and all the DCtrls report to the GCtrl. We first conduct an experiment to show the procedure of setting up an inter-domain flow in the CP system with DeepMDR. Fig. 5 shows the messages related to the procedure, where the first capture is collected on the GCtrl while the others are collected on the DCtrls of the source, an intermediate and destination domains of the flow, respectively.

The first capture in Fig. 5 shows that upon receiving the IDP request, the GCtrl only takes ~ 12 msec to finish the inter-domain path computation. Then, it sends an IDP message to the DCtrl of each related domain and instructs it to calculate the corresponding intra-domain path segment. Next, the DCtrls leverage their DL-assisted routing modules for intra-domain path computation, and set the whole inter-domain path up by sending *FlowMod* messages to related switches. This is done in parallel by the DCtrls. According to the Wireshark captures in Fig. 5, the DCtrls accomplish the path setup within 76 msec.

We then conduct a few stress tests to measure the message processing capacity of the CP system with DeepMDR. Specifically, we leverage a Cbench tool that supports POF [15] to generate and flood a large number of *PacketIn* messages to the CP system, and see how many of them can be processed successfully. In the experiments, the sending rate from the Cbench tool to all the DCtrls is fixed at $800,000$ messages per second, and we change the ratio of inter-domain requests in them from 0 to 100%. Fig. 6 shows that the message processing capacity decreases with the ratio of inter-domain requests. This is because the CP system needs to use more operations to process it if a request is for an inter-domain flow, and resulting communications between the GCtrl and DCtrls also restrict the message processing capacity. However, even for the worst case scenario where all the *PacketIn* messages are for inter-domain requests, the processing capacity is still above $\sim 166,000$ messages per second. Hence, our implementation of DeepMDR has satisfactory performance on control message processing, which suggests that it can fit into the requirements on the CP system of a real multi-domain network.

V. Discussion And Conclusion

In this paper, we designed and experimentally demonstrated DeepMDR, which is a DL-assisted CP system for scalable path computation in multi-domain packet networks. We discussed the design of overall system architecture and functional modules in detail, and elaborated on our proposals for inter- and intra-domain path computations. Our work made contributions in both systemic and algorithmic aspects. On the system side, we developed DeepMDR based on ONOS, extended it to support POF in the DP, and facilitated the hierarchical architecture for multi-domain CP operations. On the algorithm side, we designed and optimized a DL model to achieve fast and high-quality path computation in each domain.

Meanwhile, we hope to point out that DeepMDR can still be improved from the following three perspectives. Firstly, its current path computation only considers bandwidth as the QoS demand, while in a practical network, there could be more

No.	Time	Source	Destination	Protocol	Lengtl	Info
6339	348.766248051	192.168.109.213	192.168.109.224	IDP Protocol	301	Type:Flow_Request
6341	348.778338997	192.168.109.224	192.168.109.213	IDP Protocol	97	Type:Flow_Reply
6345	348.778926795	192.168.109.224	192.168.109.205	IDP Protocol	97	Type:Flow_Reply
6348	348.779088230	192.168.109.224	192.168.109.95	IDP Protocol	97	Type:Flow_Reply

Message collected on GCtrl

38598	310.318294143	192.168.109.208	192.168.109.213	POF	196	Type:POFT_PACKET_IN
38622	310.335042994	192.168.109.213	192.168.109.224	IDP Protocol	301	Type:Flow_Request
38624	310.347285849	192.168.109.224	192.168.109.213	IDP Protocol	97	Type:Flow_Reply
38644	310.394120853	192.168.109.213	192.168.109.208	POF	1514	Type:POFT_FLOW_MOD

Message collected on DCtrl of source domain

27078	353.965620588	192.168.109.224	192.168.109.205	IDP Protocol	97	Type:Flow_Reply
27095	354.013647702	192.168.109.205	192.168.109.208	POF	2258	Type:POFT_FLOW_MOD

Message collected on DCtrl of an intermediate domain

26523	341.331839295	192.168.109.224	192.168.109.95	IDP Protocol	97	Type:Flow_Reply
26555	341.375874849	192.168.109.95	192.168.109.208	POF	2258	Type:POFT_FLOW_MOD

Message collected on DCtrl of destination domain

Fig. 5. Wireshark captures of messages about setting up an inter-domain flow.

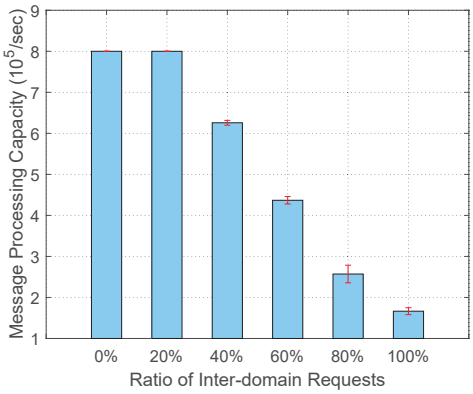

Fig. 6. Message processing capacity of DeepMDR (with five domains).

types of QoS demands (*e.g.*, delay and jitter). Hence, the path computation can be improved to consider various QoS demands jointly. Secondly, the DL-assisted routing module uses offline training, which means that it needs to adopt transfer learning when the network environment changes dramatically. However, if we upgrade the DL model to a deep reinforcement learning (DRL) model whose adaptivity is guaranteed with online training, the hassle of transfer learning can be avoided. Last but not the least, we can integrate network telemetry and related data analytics in DeepMDR to make its functionalities more comprehensive for network automation.

Acknowledgments

This work was supported in part by the ZTE Research Fund PA-HQ-20190925001J-1, NSFC projects 61871357, SPR Program of CAS (XDC02070300), and Fundamental Funds for Central Universities (WK3500000006).

References

[1] P. Lu *et al.*, "Highly-efficient data migration and backup for Big Data applications in elastic optical inter-datacenter networks," *IEEE Netw.*, vol. 29, pp. 36–42, Sept./Oct. 2015.

[2] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.

[3] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, Mar. 2008.

[4] Z. Zhu *et al.*, "Demonstration of cooperative resource allocation in an OpenFlow-controlled multidomain and multinational SD-EON testbed," *J. Lightw. Technol.*, vol. 33, pp. 1508–1514, Apr. 2015.

[5] X. Chen *et al.*, "Incentive-driven bidding strategy for brokers to compete for service provisioning tasks in multi-domain SD-EONs," *J. Lightw. Technol.*, vol. 34, pp. 3867–3876, Aug. 2016.

[6] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

[7] S. Li *et al.*, "Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability," *IEEE Netw.*, vol. 31, pp. 12–20, Mar./Apr. 2017.

[8] Z. Zhu, W. Lu, L. Zhang, and N. Ansari, "Dynamic service provisioning in elastic optical networks with hybrid single-/multi-path routing," *J. Lightw. Technol.*, vol. 31, pp. 15–22, Jan. 2013.

[9] L. Gong *et al.*, "Efficient resource allocation for all-optical multicasting over spectrum-sliced elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. 836–847, Aug. 2013.

[10] Y. Yin *et al.*, "Spectral and spatial 2D fragmentation-aware routing and spectrum assignment algorithms in elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. A100–A106, Oct. 2013.

[11] N. Xue *et al.*, "Demonstration of OpenFlow-controlled network orchestration for adaptive SVC video manycast," *IEEE Trans. Multimedia*, vol. 17, pp. 1617–1629, Sept. 2015.

[12] K. Joshi and K. Kataoka, "PRIME-Q: Privacy aware end-to-end QoS framework in multi-domain SDN," in *Proc. of NetSoft 2019*, pp. 169–177, Jun. 2019.

[13] X. Chen *et al.*, "DeepRMSA: A deep reinforcement learning framework for routing, modulation and spectrum assignment in elastic optical networks," *J. Lightw. Technol.*, vol. 37, pp. 4155–4163, Aug. 2019.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of CVPR 2016*, pp. 770–778, Jun. 2016.

[15] POF Cbench Tool. [Online]. Available: https://github.com/USTC-INFINITELAB/POFSwitch.