World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:13, No:9, 2019

Optimizing Network Latency with Fast Path Assignment for Incoming Flows

Qing Lyu, Hang Zhu

***Abstract*—**Various flows in the network require to go through different types of middlebox. The improper placement of network middlebox and path assignment for flows could greatly increase the network latency and also decrease the performance of network. Minimizing the total end to end latency of all the ows requires to assign path for the incoming flows. In this paper, the flow path assignment problem in regard to the placement of various kinds of middlebox is studied. The flow path assignment problem is formulated to a linear programming problem, which is very time consuming. On the other hand, a naive greedy algorithm is studied. Which is very fast but causes much more latency than the linear programming algorithm. At last, the paper presents a heuristic algorithm named FPA, which takes bottleneck link information and estimated bandwidth occupancy into consideration, and achieves near optimal latency in much less time. Evaluation results validate the effectiveness of the proposed algorithm.

***Keywords*—**Latency, Fast path assignment, Bottleneck link.

I. Introduction

FLOWS in network often change rapidly. Many of which are required to go through a chain of middlebox to meet the performance and security demand. The flows may suffer significant latency due to the improper assignment of the flow paths and load distributions, unbalancing the network resources. This may further cut down the network resource utilization, such that some middlebox will be loaded with too many ows and thus risk packets losses.

In the context of Software Defined Network (SDN) [1], [2] and Network Function Virtualization (NFV) [3], the devices of the network turn to be programmable, and network middlebox can be deployed as software modules/components. The flows' status are changing frequently, when some flows are finishing transmitting, some other flows are being formulated and begin to transmit at the same time. At the same time, a flow may require to go through several different middlebox or a ordered sequence of middlebox, i.e., service chain [4]. The path assigned for a specific flow should include specific types of middlebox or service chain. To keep pace with the rapidly changing flows and complex middlebox requirements, how to efficiently handle the incoming flows to optimize the utilization of network resource is a tough problem. In order that the flow paths adapt to the dynamic changes of the network environments, flow paths should be carefully assigned to accommodate the network topology and global information should be taken into consideration so that network resource utilization is optimized. Moreover, flows should be assigned among different middlebox to avoid the disable of the instance of middlebox and keep the network operating in a good condition.

Many recent works focus on middlebox placement as well as the path switching problem in the network [5]–[8]. [9] addresses the problem of load distribution in datacenter network when there are traffic spikes. A hybrid approach is proposed to handle the load distribution problem when there are traffic spikes. The system is consisted of a load distributing controller in the control plane and a load distributor in the data plane. For the flat flows, the controller will assignment paths for them according to the topology and bandwidth information. While for spiked flows, the load distributors are classied into several groups in advance to handle different flows. For a incoming flow, the controller will firstly checks the statuses of the distributors in its group and choose the idlest distributor to distribute the flow, if the distributors in the group are too busy to meet the constraint to handle the incoming flow, then the controller will guide the flow to other groups until it is effectively guided to the wanted destination. The grouping are updated periodically by the controller according to the load statuses of the distributors. SIMPLE [10] turns the policies in the middlebox to forwarding rules in SDN switches in the condition of load balance. SIMPLE consists of a resource manager, a modification handler and a rule generator. The resource manager takes care of the load balance among the middlebox, the modification handler deals with the changes caused by the middlebox, and the rule generator converts the policies in middlebox to rules in the SDN switches. Reference [11] considers the traffic rate changing effect of middlebox and puts forward a traffic aware middlebox placement scheme. It first sort the middlebox according to the traffic rate changing factor in the increasing order. For a single flow with predetermined path, it adopts a rule called Least-First-Greatest-Last (LFGL) to place the middlebox. While for multiple flows without predetermined paths, it is proved that the middlebox placement problem is NP-hard and can not be solved in polynomial time, a MinMax path guided heuristic is applied to address the problem. [12] takes into account the service chains and propose a solution to place the middlebox in the optimal locations when network information such as topology is known as well as the policies specifications. The authors firstly convert the placement problem to $0-1$ programming and prove that it is NP-hard, then two heuristic algorithms, greedy algorithm and simulated annealing based algorithm, are employed to obtain

Qing Lyu is with the Department of Automation and Research Institute of Information Technology, Tsinghua University, Beijing, China (email: lvq16@mails.tsinghua.edu.cn).
Hang Zhu is with Johns Hopkins University, USA.

TABLE I
A FLOW EXAMPLE

Source IP	Destination IP	Source Port	Destination Port	Protocol	Required Bandwidth	Service Chain
59.66.8.2/24	64.10.8.2/24	0-65535	80	TCP	600bps	FW,IDS

a sub-optimal solution.

Other works such as [13]–[15] target on the policy enforcement problem and distribute the policy on switches. Furthermore, [16] solves the network-wide middlebox extending problem by using FlowTags. Reference [17] considers the security problem in finding the optimal path as well as the updating problem. Reference [18] uses SDN to manage the middlebox placement.

The middlebox aware flow path assignment is formulated to a optimization problem in this paper. The optimization goal is to minimize the total end to end latency of the flows. However, the optimization goals can also be flexibly generalized to other metrics such as maximizing the utilization of bandwidth or minimizing the maximum link load ratio (the ratio of current link load to link bandwidth capacity) [11] in the network.

We put out a strategy to dynamically assignment path for each incoming flow. We firstly formulate the problem to a linear programming (LP) problem, but due to the difficulties of handling the path loops in the network, finding optimal solution to the LP problem is time consuming. Therefore, we come up with a heuristic algorithm FPA which takes into consider the bandwidth occupancy of each link and assigns the shortest path that does not impact the path assignment of the following flows to the current flow. We verify our proposed heuristic algorithm in three different network topologies by comparing to it the LP method and a naive greedy algorithm. Evaluation results show that FPA assigns latency optimized paths for arriving flows efficiently.

II. PROBLEM FORMULATION

We formulate the network as a directed graph $G(V, E)$, where G represents the nodes set while E represents the links set. The nodes include the hosts and forwarding devices, the links denote the connections the nodes. Flows can be transmitted from hosts to hosts. A flow example is given in Table I. The first 5 fields are the 5-tuple of a packet, the other 2 fields are the demanded bandwidth and service chain respectively. In this example, the specifications are firewall (FW) and intrusion detection system (IDS).

We define the variables in the network into 3 categories: topology variable, flow variable and assistant variable. Topology variable specifies the nodes and links in the network, nodes represents the hosts and forwarding devices. Denote (i, j) as the link from node i to node j, $Capa_{i,j}$ as the bandwidth capacity of link (i, j), L_{ij} as the latency of link (i, j), $Mloca_m$ as the positions of middlebox type m, a type of middlebox may have multiple instances in the network. M denotes the set of middlebox types, such as FW, IDS, Proxy, Deep Inspection (DI), Network Address Translation (NAT), Load Balance, etc [12]. Flow variables describe the information of flows. Denote F as the set of flows that need to assignment paths, (s, t) as a flow from source node s to destination node t, $M_{s,t}$ as the middlebox that flow (s, t) needs to go through, C_{st} as the bandwidth demanded by flow (s, t). Assistant variables are the decision variable which used in the linear programming, for example, $X_{i,j}^{s,t}$ is an indicating variable that defined as

$$x_{ij}^{st} = \begin{cases} 1, & \text{if } (s,t) \text{ go through link } (i,j) \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Similarly, y_i^{st} is used to indicate a flow (s, t) go through a node i, y_i^{st} can be represented as

$$\forall i \in V: \quad y_i^{st} = max \left\{ \sum_{j \in V,(j,i) \in E} x_{ji}^{st}, \sum_{j \in V,(i,j) \in E} x_{ij}^{st} \right\} \tag{2}$$

Equation (2) means that, for any flow and any node in the network, if the flow goes into the node (the number of ingress flow is 1) or leaves the node (the number of egress flow is 1), then the flow goes through the node, thus y_i^{st} equals to 1. Otherwise, the the number of ingress flow and the number of egress flow for the node both equal to 0, and the flow does not go through the node, thus y_i^{st} equal to 0.

u_i^{st} is a defined function of flow (s, t) and satisfies $1 \leq u \leq |V|$.

According to the forwarding action of a node, we have the following constraint

$$\forall s, t \in F: \sum_{i \in V,(i,j) \in E} x_{ij}^{st} - \sum_{k \in V,(j,k) \in E} x_{jk}^{st} = \begin{cases} -1, & j = s \\ 0, & j \neq s, t, j \in V \\ 1, & j = t \end{cases} \tag{3}$$

Equation (3) specifies the fact that for any flow (s, t) in the network, if $j = t$, it means that node j is the ingress node for flow (s, t), the number of ingress flows is 1 and the number of egress flows is 0, thus the number of ingress flows subtract the number of egress flows is 1, (3) holds. If $j = s$, vice versa, the subtract value is -1. If $j \neq s, t, j \in V$, whether flow (s, t) go through node j or not, the subtract value is 0. Equation (3) also determines that a flow is successfully routed.

To make sure there is no forwarding loop in the network, we make the following constraint

$$\forall (s,t) \in F, \forall (i,j) \in E: \quad u_i^{st} - u_j^{st} + |V| x_{ij}^{st} \leq |V| - 1. \tag{4}$$

From (4) we know that for any flow (s, t) and any link (i, j) in the network, if flow (s, t) does not go through link (i, j), then $x_{ij}^{st} = 0$, as we have assumed the assistant variable $1 \leq u \leq |V|$, thus (4) holds. Otherwise, flow (s, t) does go through link (i, j), $x_{ij}^{st} = 1$, the constrain (4) makes sure that $u_i^{st} - u_j^{st} \leq -1$. Which means that, for the node that a flow

goes through before, its node value u is less than that the flow goes through later. If there is a loop in the flow path, there should be a node that the flow goes through twice or more. Without loss of generality, assume there is a flow goes through nodes $\{a, b, c, d, e, f, a\}$ sequentially, we can obtain

$$\begin{aligned} u_a^{st} - u_b^{st} &\leq -1, \\ u_b^{st} - u_c^{st} &\leq -1, \\ u_c^{st} - u_d^{st} &\leq -1, \\ u_d^{st} - u_e^{st} &\leq -1, \\ u_e^{st} - u_f^{st} &\leq -1, \\ u_f^{st} - u_a^{st} &\leq -1, \end{aligned} \tag{5}$$

sum the items of Equation (5) by adding left to left side and right to right side, we can obtain

$$0 \leq -6, \tag{6}$$

which makes contradiction. Therefore, there should be no loop according to (4).

For the link bandwidth constrain, we have

$$\forall (i,j) \in E : \sum_{(s,t)\in F} x_{ij}^{st} \times C_{st} \leq Capa_{ij}. \tag{7}$$

Equation (7) specifies that the total demanded bandwidth of all flows that go through a link does not exceed the bandwidth capacity of that link.

Furthermore, a flow need to go through a specific type middlebox, we have the constraint

$$\forall (s,t) \in F, m \in M_{st} : \sum_{i \in Mloca_m} y_i^{st} \geq 1. \tag{8}$$

Every type of middlebox is placed to certain nodes, each type may have multiple middlebox instances, for any flow that need to go through middlebox of a certain type, Equation (8) configures that the flow go through at least 1 instance node of the middlebox, thus certain type's service for that flow is satisfied.

Lastly, the processing ability of the nodes that deployed with different types of middlebox should also be considered.

$$\forall i \in Mloca_m : \sum_{(s,t)\in F} y_i^{st} R(st) \leq R_i. \tag{9}$$

Where $R(st)$ is the processing resource that flow (s,t) needs. R_i is node i's processing resource capacity that can be spared to deal with the flows that routed to node i.

Considering the fact flows with higher bandwidth such as video may require higher priority, we take the bandwidth weighted end to end latency as our optimization target, and the whole optimization problem can be represented as

$$\begin{aligned} min \quad & \sum_{(s,t)\in F} \sum_{(i,j)\in E} x_{ij}^{st} L_{ij} C_{st} \\ s.t. \quad & (1), (2), (3), (4), (7), (8), (9). \end{aligned} \tag{10}$$

Algorithm 1 Find Bottleneck Link

Input: Topology, Mloca, flow set F_2, bandwidth
Output: Bottleneck link

```
function BOTTLENECK LINK(F_2)
    //Find the shortest latency path with demanded middlebox for flow set F_2
    for flow f in F_2 do
        //Get the location of middlebox for f: f_mb
        Calculate the shortest path that goes through f_mb
        path = Dijkstra(topology, f, f_mb)
        shortest path[f] = path
    end for
    //Calculate the demanded bandwidth for related links
    for link l in shortest path[F_2] do
        sum demanded bandwidth of flows go through l
        Calculate ratio of demanded BW and residue BW
    end for
    //Find the link with biggest ratio
    bottleneck link=link with Max ratio
    return result
end function
```

III. Algorithm Design

Solving the optimization problem is time consuming and can not meet the processing requirement in real deployment, which will be illustrated in the Section V. We turn to heuristic algorithms to assign paths for flows rapidly.

Due to the bandwidth constrain of the links, not every flow can be assigned to the best choice of path with the minimum latency. We consider both the bandwidth consumption information of each flow and the bandwidth residue of each link in the network, and assign path for each incoming flow aiming at minimizing the total end to end latency of all the flows.

A natural heuristic algorithm is the greedy algorithm. For the flow set remains to be assigned paths, we handle the incoming flows one by one. Calculate the path for the incoming flow with smallest latency that goes through needed middlebox by Dijkstra algorithm [19]. If the path can not meet the bandwidth constrain then go to the path with second smallest latency until the bandwidth constrain of all the links in the path are satisfied. Repeat the progress until all the flows have been assigned paths. It just considers the bandwidth capacity constraint when calculate the flow path can always chooses the path with least latency for the current flow.

We propose another heuristic algorithm called FPA. It considers the effect of the path assignment for the current flow to the later flows. Suppose there are multiple incoming flows waiting for path assignment, denote $F_1 = \{f_{11}, f_{12}, ..., f_{1K}\}$ as the set that flows have already been assigned, $F_2 = \{f_{21}, f_{22}, ..., f_{2L}\}$ as the set that flows waits to be assigned. As the bandwidth capacity of each link is limited, and a link may be demanded by multiple flows, we could not assign path with the least latency for every flow. We make effort to minimize the total end to end latency for all flows. Define the link ratio as the total expected bandwidth of all flows that prepare to go through the link under the shortest latency path condition with

TABLE II
A BOTTLENECK LINK EXAMPLE

Flow	Included Link	Link	Ratio
f_{20}	link1, link2	link1	$\frac{f_{20,dem}+f_{23,dem}}{BWresidue_{link1}}$
f_{21}	link3, link4	link2	$\frac{f_{20,dem}+f_{22,dem}}{BWresidue_{link2}}$
f_{22}	link2, link3	link3	$\frac{f_{21,dem}+f_{22,dem}+f_{23,dem}}{BWresidue_{link3}}$
f_{23}	link1, link3	link4	$\frac{f_{21,dem}}{BWresidue_{link4}}$

Algorithm 2 Calculate latency1

Input: Topology, Mloca, flow set F_2, bandwidth
Output: Latency1

function CALCULATE LATENCY1(F_2)
  //Calculate the latency when always assigning the shortest latency path for current flow
  for flow f in F_2 **do**
    current topology=topology
    for link l in current topology **do**
      if l's bw smaller than f's demanded bw **then**
        remove l from current topology
        curr topo=RemoveLinks(l)
      end if
    end for
    //Get the location of middlebox for f: f_{mb}
    Calculate the shortest path that goes through f_{mb}
    path=Dijkstra(curr topp, f, f_{mb})
    assignment the *path* to f
    latency1=0
    latency1=latency1+f's path latency
    update bandwidth residue
    move f to F_1
  end for
  return result
end function

Algorithm 3 Calculate latency2

Input: Topology, Mloca, flow set F_2, bandwidth
Output: Latency2

function CALCULATE LATENCY2(F_2)
  //Calculate latency when always assigning the shortest latency path without bottleneck link for current flow
  latency2=0
  for flow f in F_2 **do**
    current topology=topology
    for link l in current topology **do**
      if l's bw smaller than f's demanded bw **then**
        remove l from current topology
        curr topo=RemoveLinks(l)
      end if
    end for
    //Get the location of middlebox for f: f_{mb}
    Calculate the shortest path that goes through f_{mb}
    path=Dijkstra(curr topp, f, f_{mb})
    bottleneck link=BOTTLENECK LINK(F_2)
    if bottleneck link is not in f's path **then**
      assignment the path to f
      latency2=latency2+f's path latency
      update bandwidth residue
      move f to F_1
      continue
    else
      remove the *bottleneck link* from curr topp
      curr topo=RemoveLinks(*bottleneck link*)
      //Calculate the next shortest latency
      CALCULATE LATENCY2
    end if
  end for
  return result
end function

the demanded middlebox to the bandwidth residue of that link

$$linkRatio_{ij} = \frac{\sum_{f \in F_2,(i,j) \in p_f} f_{dem}}{BWresidue_{ij}}. \quad (11)$$

where f_{dem} denote the demanded bandwidth by flow f, p_f is the shortest latency path by Dijkstra algorithm for flow f. We define the link with highest link ratio as the *bottleneck link*.

Table II gives an example of the calculation of the bottleneck link, where the link with the highest link ratio will be the bottleneck link. The pseudo code to find the bottleneck link is shown in Algorithm 1.

For the current flow, if the shortest latency path with needed middlebox on it does not include the bottleneck link, then choose it as the path for the current flow and go to take care of the next incoming flow. However, if the shortest latency path for the current flow does include the bottleneck link, we have two options, the first one is we still assign the shortest path for the current flow and assign paths for the later incoming flows in the same way. Then calculate the end to end latency. Nevertheless, this may bring troubles to the later flows because the current flow could occupy the link bandwidth and the later flows can not choose the wanted paths as the link bandwidth capacity is exhausted. Under this strategy, the latency calculation is depicted in Algorithm 2. The second one is we do not select the path which has the shortest latency contains the bottleneck link for for the current flow, we turn to the second shortest latency path , if it still include a bottleneck link, then go to the next shortest latency path with demanded middlebox until there is no bottleneck link in the selected path for the current flow. If all available path for the flow contains bottleneck link, choose the first shortest latency for the flow. In this way the later incoming flows have larger space to choose the paths as the link bandwidth is relatively adequate. Update the link bandwidth residue information. Repeat doing this until all the flows have been assigned paths. Finally calculate the total end to end latency. Under this strategy, the latency calculation is depicted in Algorithm 3. Finally, we compare the total end to end latency of the two methods and choose the one with the lower value. The path assignment algorithm FPA is given in Algorithm 4.

IV. IMPLEMENTATION

We generate different network topologies to perform simulations on the proposed flow path assignment algorithm. The method described by [20] is used to generate network topology. The network topology generator takes multiple parameters as inputs. The number of connection (nodes and links) can be adjusted conveniently. The degree of connectivity in the network can be set as low, medium or high. We adopts three kinds of network topology, Class A, B, C. Specifically,

Algorithm 4 Path assignment

Input: Topology, Mloca, flow set F_2, bandwidth
Output: Path assignment

```
function PATH ASSIGNMENT(F2)
    //assignment path for flows in flow set F2
    for flow f in F2 do
        Compare Latency1 and Latency2
        if Latency1 is smaller than Latency2 then
            assignment the shortest latency path to f
        else
            assignment the shortest latency path without bottleneck link in it to f
        end if
        move f to F1
    end for
    return result
end function
```

Class A with 36 nodes and 108 links, Class B with 75 nodes and 196 links, Class C with 115 nodes and 384 links. For each topology, we allocate link latency and bandwidth constraints to the links. We generate flows from host to host based on the topology described above. We classify all the nodes in the topology into two categories, the edge nodes with only one connection and the intermediate nodes with multiple connection. Only the edge nodes can be chosen to deploy host, and intermediate nodes are chosen to deploy forwarding devices. We have implement 5 different types of middlebox. For a certain type of middlebox, several middlebox instances are randomly connected to the chosen forwarding devices. We randomly generate small flows between hosts and then merge the flows with the same source host and terminal host until a certain number of bandwidth is exhausted. In this way we can obtain a set of flows which can meet the bandwidth constraints. Total end to end latency is selected to evaluate the effectiveness of our flow path assignment algorithm. The algorithm assigns path for the incoming flows, the path latency can obtained by adding links' latency and processing latency. We calculate the total end to end latency by adding the path latency of all flows.

We implemented the path assignment algorithm in Python. Network libraries such as FNSS (Fast Network Simulation Setup) and NetworkX are adopted to generate the network topology and flows information. FNSS can setup the network, such as adding or removing links, acquiring link bandwidth and latency and obtaining the neighbouring nodes of the current node. Flows between two hosts can also be generated and removed. NetworkX offers various graph algorithms such as the shortest path calculation using Dijkstra algorithm. Furthermore, the formulated LP problem is solved by LP solvers [21]–[23].

V. Evaluation

Several simulation results are carried out to evaluate the proposed ow path assignment algorithm.

The evaluation is conducted under three kinds of topologies as described above. The topologies are shown in Table III.

With the rapid variation of network flows and network environments, the path assignment problem takes the overall

TABLE III
NETWORK TOPOLOGIES

Topology	Number of nodes	Number of links
A	36	108
B	75	196
C	115	384

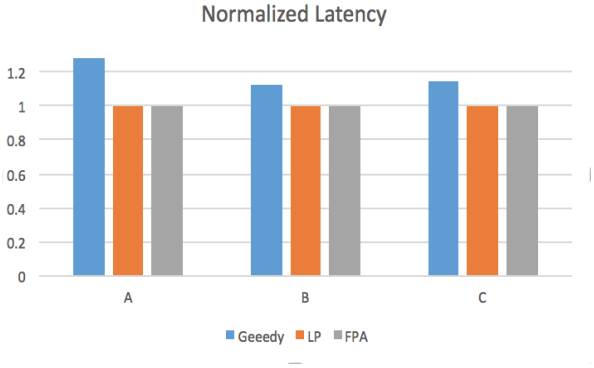

Fig. 1 Normalized total end to end latency under different topologies

end to end latency as the optimization target. How to rapidly assign paths for all the flows is the key indicator that should be cared about. The memory consumption of all the comparing methods is not a critical point here, so we focus the emphasis on the processing time of the algorithms.

To evaluate the effectiveness of the proposed path assignment algorithms FPA, LP and the naive greedy algorithm are taken as the baseline. The naive greedy algorithm always chooses the path with the smallest latency for the current flow under the bandwidth constraints. It does not consider the impact of the bandwidth consumption of the link, which may bring trouble to the path assignment of other flows. It occupies the link bandwidth no matter the link is required by the following flows or not. However, the link bandwidth is limited, therefore the rest flows might have to go through the path with longer latency if the bandwidth of the required link is exhausted. This algorithm does not take into account the overall demands of all the flows and always selects the optimal path for the current flow in each iteration, which may increase the total end to end latency.

Hereby, consider three critical questions: (1). What kind of performance can the algorithms achieve to minimize the total end to end latency when multiple flows need to be assigned paths. We compare the latency of the three methods: Linear Programming, proposed path assignment algorithm FPA and naive greedy algorithm. (2). How many processing time is needed to handle all the flows for the three methods, here we do not consider memory consumption of the algorithms as it is not a key index in the context. (3). What is the performance gap between the proposed path assignment algorithm and LP method, how close does the proposed path assignment algorithm obtain when comparing to the LP method which is the optimal solution that considers all the network constraints. Our methods can be convenient scaled to other optimization targets such as minimize the maximum

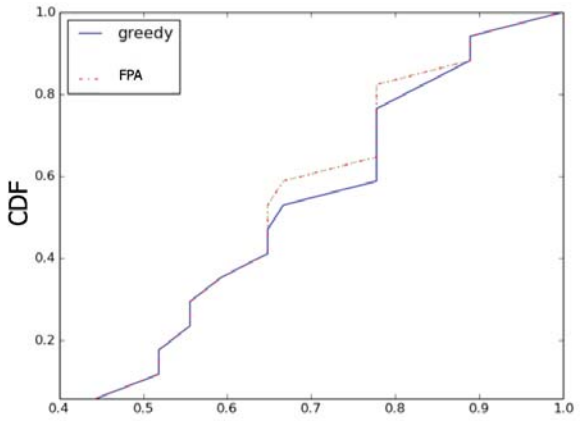

Fig. 2 CDF of the normalized latency for single flow under Topology A.

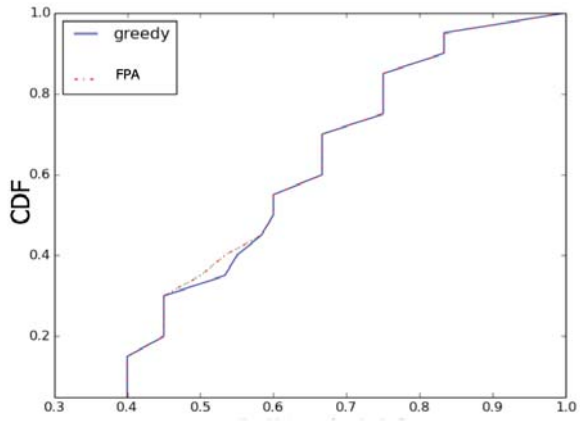

Fig. 3 CDF of the normalized latency for single flow under Topology B.

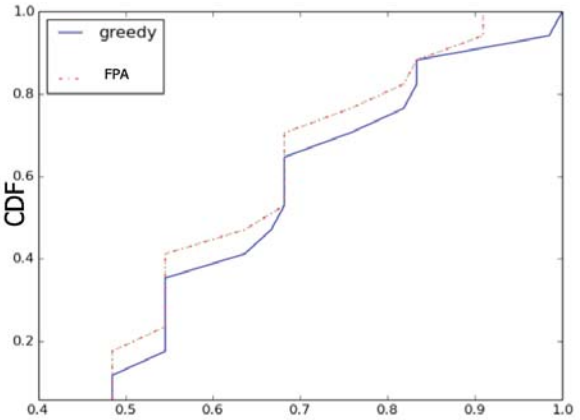

Fig. 4 CDF of the normalized latency for single flow under Topology C.

TABLE IV
PROCESSING TIME

Topology	A	B	C
LP	146.81	202.25	266.04
Greedy	0.18	0.31	0.56
FPA	0.28	2.22	22.01

link bandwidth consumption to link bandwidth capacity ratio or maximize the bandwidth utilization. All we need to do is to modify the optimization target, add or remove constraints under the current network model.

LP considers the overall network condition, and dedicates to find the optimal solution, therefore, the latency is the least among all the methods. To compare the performance of the algorithms more intuitively, the latency is normalized, where we calculate the ratio of the achievements of different algorithms to least latency solution, LP. The results are shown in Fig. 1, which can be obviously observed that, the latency of the naive greedy algorithm is the largest as it only consider the best path for the current flow. The proposed path assignment algorithm FPA can achieve almost the same performance of the LP algorithm. which also indicates that the performance gap between LP and FPA is quite small.

Under different kinds of network topology, the processing time of each algorithm is illustrated in Table IV. It can be obviously observed that the processing time of LP is much longer than that of the naive greedy algorithm and the proposed path assignment algorithm FPA. This is because LP is the globe optimal solution for the path assignment problem. To find the optimized path for every flow, it needs to consider multiple constraints while finding the optimal solution that meets all the constraint is very time consuming. The processing time of the naive greedy algorithm is the least as it just chooses the path with the smallest latency for the current flow, it only needs to consider the path goes through certain types of middlebox, and the links in the path meet the flow's bandwidth demand. It does not care about the bandwidth consumption of the links, which may bring trouble to the following flows when assigning paths for them. The processing time for the proposed path assignment algorithm is a little longer than that of the naive greedy algorithm. The proposed path assignment algorithm take the network condition into account and evaluates the impact it may bring to the following flows when assigning path for the current flow.

To further exhibit the details of the proposed path assignment algorithm and illustrate the path assignment strategy for single flow, we plot the CDF curve of the normalized latency of a single flow for the proposed algorithm and the naive greedy algorithm, where single flow's latency is normalized by calculating the ratio of it to the biggest flow latency. The results under different topologies are shown in Figs. 2-4,. From which the difference of the path assignment results of the proposed algorithm FPA and naive greedy algorithm can be observed. Under Topology A, the path assignment results of the two algorithms apart from each other in the middle stage, which means for flows with medium

latency, the two algorithms adopt different strategy to assign the paths and achieve different performance. While under Topology B, the path assignment results are the different for the flows with small latency. However, under Topology C, the path assignment results are quite different for single flow no matter the flow's path latency is small or big. This is because the proposed algorithm and naive greedy algorithm only choose the same path if the path for the current flow does not influence the following flows. However, for the flows which may affect other flows' path assignment, the two algorithms make different strategies which leads to the deviation of the curves.

VI. Conclusion

Flows in network often change rapidly. The improper assignment of the flow paths and load distributions may cause network to suffer from terrible congest. In this paper, we consider the problem of path assignment for multiple flows. How to minimize the total end to end latency while rapidly assigning paths of all flows need be be carefully addressed. We first formulate the problem to a LP problem but it is quite time consuming to find the optimal solution when multiple network constraints are needed to be considered. To reduce the processing time, the naive greedy algorithm is put forward, which always choose the path with smallest latency for the current. However, the total end to end latency for all the flows increases as it may cause the later flows to go through path with longer latency. Therefore, a heuristic algorithm FPA is proposed, which considers the affects that path assignment for current flow may bring to the following flows. The algorithm selects the path for current flow that considers the bandwidth condition and makes the strategy leads to the minimized total end to end latency. The evaluation results validate the effectiveness of the proposed algorithm.

VII. Acknowledgments

The authors benefit a lot from the fruitful discussions with colleagues in the Network Security Lab in Tsinghua University. The authors would also like to thank the anonymous reviewers for their efforts in the revision. This work is supported by National Natural Science Foundation (No. 61872212) and National Key Research and Development Program (No.2016YFB1000101).

References

[1] O. N. Fundation, "Software-defined networking: The new norm for networks," *ONF White Paper*, vol. 2, pp. 2–6, 2012.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[3] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[4] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan, *et al.*, "Steering: A software-defined networking for inline service chaining," in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pp. 1–10, IEEE, 2013.

[5] A. Gushchin, A. Walid, and A. Tang, "Scalable routing in sdn-enabled networks with consolidated middleboxes," in *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, pp. 55–60, ACM, 2015.

[6] A. Hari, T. Lakshman, and G. Wilfong, "Path switching: reduced-state flow handling in sdn using path information," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, p. 36, ACM, 2015.

[7] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, "Toward software-defined middlebox networking," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pp. 7–12, ACM, 2012.

[8] A. Abujoda and P. Papadimitriou, "Midas: Middlebox discovery and selection for on-path flow processing," in *Communication Systems and Networks (COMSNETS), 2015 7th International Conference on*, pp. 1–8, IEEE, 2015.

[9] Z. Liu, X. Wang, W. Pan, B. Yang, X. Hu, and J. Li, "Towards efficient load distribution in big data cloud," in *Computing, Networking and Communications (ICNC), 2015 International Conference on*, pp. 117–122, IEEE, 2015.

[10] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in *ACM SIGCOMM computer communication review*, vol. 43, pp. 27–38, ACM, 2013.

[11] W. Ma, J. Beltran, Z. Pan, D. Pan, and N. Pissinou, "Sdn-based traffic aware placement of nfv middleboxes," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 528–542, 2017.

[12] J. Liu, Y. Li, Y. Zhang, L. Su, and D. Jin, "Improve service chaining performance with optimized middlebox placement," *IEEE Transactions on Services Computing*, vol. 10, no. 4, pp. 560–573, 2017.

[13] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *INFOCOM, 2013 Proceedings IEEE*, pp. 545–549, IEEE, 2013.

[14] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the one big switch abstraction in software-defined networks," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pp. 13–24, ACM, 2013.

[15] X. Wang, W. Shi, Y. Xiang, and J. Li, "Efficient network security policy enforcement with policy space analysis," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2926–2938, 2016.

[16] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags.," in *NSDI*, vol. 14, pp. 543–546, 2014.

[17] J. Liu, Y. Li, H. Wang, D. Jin, L. Su, L. Zeng, and T. Vasilakos, "Leveraging software-defined networking for security policy enforcement," *Information Sciences*, vol. 327, pp. 288–299, 2016.

[18] Z. Qazi, C.-C. Tu, R. Miao, L. Chiang, V. Sekar, and M. Yu, "Practical and incremental convergence between sdn and middleboxes," *Open Network Summit, Santa Clara, CA*, 2013.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[20] J. McCauley, Z. Liu, A. Panda, T. Koponen, B. Raghavan, J. Rexford, and S. Shenker, "Recursive sdn for carrier networks," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 4, pp. 1–7, 2016.

[21] S. Mitchell, M. OSullivan, and I. Dunning, "Pulp: a linear programming toolkit for python," *The University of Auckland, Auckland, New Zealand, http://www. optimization-online. org/DB_FILE/2011/09/3178. pdf*, 2011.

[22] A. J. Mason, "Opensolver-an open source add-in to solve linear and integer progammes in excel," in *Operations research proceedings 2011*, pp. 401–406, Springer, 2012.

[23] A. Makhorin, "Glpk (gnu linear programming kit)," *http://www. gnu. org/s/glpk/glpk. html*, 2008.