Virtual Function Placement and Traffic Steering in Flexible and Dynamic Software Defined Networks

Ali Mohammadkhan*, Sheida Ghapani†, Guyue Liu‡, Wei Zhang‡, K. K. Ramakrishnan*, and Timothy Wood‡
*University of California, Riverside
‡The George Washington University
Email: {amoha006, sghap001, kk}@ucr.edu and {timwood, guyue, zhangwei1984}@gwu.edu

***Abstract*—The integration of network function virtualization (NFV) and software defined networks (SDN) seeks to create a more flexible and dynamic software-based network environment. The line between entities involved in forwarding and those involved in more complex middle box functionality in the network is blurred by the use of high-performance virtualized platforms capable of performing these functions. A key problem is how and where network functions should be placed in the network and how traffic is routed through them. An efficient placement and appropriate routing increases system capacity while also minimizing the delay seen by flows.**

In this paper, we formulate the problem of network function placement and routing as a mixed integer linear programming problem. This formulation not only determines the placement of services and routing of the flows, but also seeks to minimize the resource utilization. We develop heuristicsto solve the problem incrementally, allowing us to support a large number of flows and to solve the problem for incoming flows without impacting existing flows.

***Index Terms*—Integer linear programming, middlebox placement, network function virtualization, software defined networks.**

I. Introduction

Software defined networking (SDN) introduces the concept of separation between the data plane and control plane, to provide more flexibility in how individual flows are handled [1], [2]. At the same time, improved techniques for packet processing in virtualized platforms running on commercial off the shelf (COTS) systems make it possible to run network functions on software based platforms rather than purpose-built hardware appliances [3], [4], [5]. This approach called network function virtualization (NFV) further enables the network to be dynamic and flexible, by exploiting software environments for common network resident functions. Thus, SDN and NFV provide flexibility and dynamic capability in the control and data planes. Key to this is the dynamic instantiation and placement of network functions (NF) in the network and flexible routing.

A service provider network rarely consists of just forwarding entities like switches and routers [6]. Current networks commonly include middlebox functions such as firewalls, proxies, caches, policy engines, etc. Switches and middlebox functionality can also coexist on the same COTS platform with the use of NFV. Flows have to be routed through these network functions in a pre-defined order, and SDN provides the necessary power and flexibility to achieve this. A network function (NF) can be dynamically instantiated in a host as long as there is enough computational power for hosting the service. Flows steered through switches and NFs, with the goal of executing the needed service functions in the required order. This could potentially result in the flow having to traverse a given link multiple times (i.e. even having loops as perceived by the network layer, [7]). Thus, the placement of the NFs and steering flows through them need to be done judiciously. The focus of this paper is on placing the NFs and routing of flows, ensuring that each flow's path starts from an entry switch, meets all the necessary NFs in sequence, and ends at the exit switch.

978-1-4673-6762-2/15/$31.00

Multiple studies on middlebox or virtual machine (VM) placement consider the placement problem independent from how flows utiize these functions and the routing of flows. Some, such as [8], [9] consider both placement and steering of flows, but solve them separately. For instance, a heuristic is used for placement and use its result as an input for flow steering. However, a placement solution that does not leverage the information about the flows can be inefficient.

In this paper we have formulated the service placement and flow steering problems jointly in a single mixed integer linear problem (MILP) formulation. This formulation results in the optimal placement of services and the routing of the flows. It seeks to minimize the maximum link and CPU core utilization and the maximum delay of flows in the network. This approach also offers the opportunity to solve problems incrementally, as flows are added. This means that instead of solving a large problem which may be intractable, the problem is partitioned and solved in smaller pieces, and the final result is still close to the optimal solution. Another benefit of the incremental solution is that after adding new flows to the network, we only have to solve the partial problem of newly added flows without impacting existing flows.

II. System Description

Network nodes in our software-based network play multiple roles – providing the conventional role of forwarding packets, and supporting network services such as firewalls, proxies, policy engines etc.. A COTS system CPU comprising multiple cores could be assigned to forwarding or to provide network functions. To avoid the overheads of Non-uniform memory

access (NUMA), we assume that a core is dedicated to a single VM that supports a network service or forwarding function.

The desired functions are instantiated in the network based on the requirements of each flow and the placement decisions are made based on the estimated per-packet computation requirement for the function and the link bandwidth as well as the maximum tolerable delay for the flow. In addition, the set (or chain) of network services and their order is the same for all packets of the flow.

Two examples of the services chains are depicted in Figure 1. Each service has its computational requirements, so we set a limit on maximum number of flows a service can support on a specified hardware. Making placement decision for services need to take all of these factors into consideration. For example, in Figrue 2 we have a network with 8 switches (we will henceforth use the term switches and network nodes interchangeably). All the switches except $S4$ have just one free core, while switch $S4$ has two available cores. The assigned services at each switch is shown in parentheses. The service chain for $F1$ is "ABDEFIC" and the service chain for $F2$ is "DEFGHI" (each letter represent a service like a firewall or a proxy.) The difficulty for efficient placement increases dramatically with the growth in the number of flows or network nodes.

III. MILP Formulation

A goal of the formulation is to obtain an 'efficient' placement of services and routing of the flows without violating the constraints of the maximum capacity of the links and tolerable delays of flows. The 'efficient' placement seeks to minimize the utilization of the links and of the available CPU cores, thus maximizing system capacity. This is especially important when we need to solve the problem incrementally as new flows and functionality may be dynamically added to the network. Our proposed problem formulation is as follows:

Minimize U subject to:

$\forall k \in Flows, \forall i, m \in Switches, \forall j \in Services, \forall l \in O_k, \forall l' \in O'_k$:

$$\sum_j M_{ij} \leq C_i \tag{1}$$

$$X_{K_{il}} = M_{ij} S_{k_{jl}} \tag{2}$$

$$N_{k_{il}} = X_{K_{il}} \circ W_{k_{il}} \tag{3}$$

$$\sum_i N_{k_{il}} = \sum_j S_{k_{jl}} \tag{4}$$

$$F_k = [I_k N_k E_k] \tag{5}$$

$$\sum_m V_{k_{l'_{im}}} - \sum_m V_{k_{l'_{mi}}} = F_{k_{il'}} - F_{k_{i(l'+1)}} \tag{6}$$

$$\sum_{l'} \sum_{i,m} V_{k_{l'_{im}}} D_{im} \leq T_k \tag{7}$$

$$A_k = S_k N_k^T \tag{8}$$

$$\sum_k A_{k_{ji}} \leq P_{ji} \tag{9}$$

$$\sum_k \sum_{l'} (V_{k_{l'_{im}}} \circ B_k) \circ (1/H_{im}) \leq U \tag{10}$$

$$\sum_k A_{k_{ji}} / P_{ji} \leq U \tag{11}$$

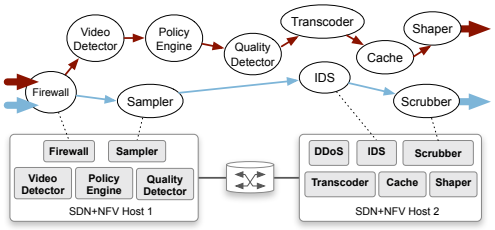

Fig. 1. NF Service Graph Map across Multiple Hosts

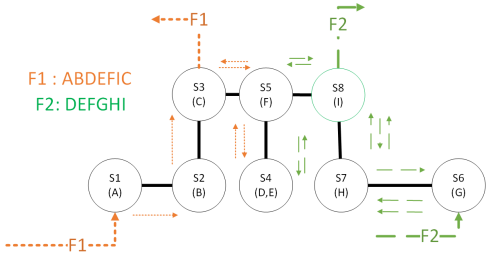

Fig. 2. An example of service placement for two flows

The definition of variables in this formulation is provided in Table I. After obtaining the solution, the placement result is stored in variable M and the routing steps in V. In this formulation we are minimizing the maximum utilization of the links and CPU cores of the network nodes (i.e., of the bottleneck). Core utilization is the number of flows using a CPU core over the maximum number of flows that can be simultaneously supported by a service on that CPU core. By minimization of the utilization, load is distributed more evenly in the network, avoiding hot spots and increasing residual system capacity. It also results in lower overall delay for flows.

Equation (1) is a constraint on the maximum number of services that can be supported on a switch. To avoid NUMA overhead, at most one service is assigned to a core. Consequently the number of services on a switch should be less than or equal to the number of free cores on that switch. The next equation, Equation (2) reflects the process of selecting the necessary switches for a flow. M represents the available

Var.	Definition
U	Maximum utilization of links and switches
M_{ij}	Number of running instances of service j on switch i
C_i	Number of available cores on switch i
$N_{k_{il}}$	Selected switch for order l of the flow k's service chain
$S_{k_{ij}}$	This value is one, if i is the j^{th} service in flow k's service chain; zero otherwise
W_k	Binary decision variable for satisfying constraint (4)
I_{ki}	Equal to 1, if i is the entrance switch for flow k
E_{ki}	Equal to 1, if i is the exit switch for flow k
D_{ij}	Delay of the link between switches i and j
T_K	Maximum delay tolerated by flow k
P_{ij}	Maximum number of supported flows, if switch i runs service j
B_K	Bandwidth usage of flow k
H_{ij}	Capacity of the link between switch i and j
O_k	A range from 1 to length of service chain for flow k
O'_k	A range from 1 to length of service chain for flow k plus one
$V_{k_{l_{im}}}$	Is one, if the link between switches i and m is used, to reach to the l^{th} service in service chain of flow k
$A_{k_{ji}}$	Is one, if switch i processes service j for flow k

TABLE I
DEFINITION OF VARIABLES OF MILP FORMULATION

services on switches, and S stores the necessary services and their order of execution. Thus, X represents the possible switches for each order of execution of a particular flow (k). For each order, only one switch is needed for a service, but multiple choices may exist in X. Equation (3) selects one instance using the binary variable W. A limit on the number of selected switches is set in (4). Although for readability and clarity we show Equation (3) as a multiplication, it is not a non-linear constraint, because it can be re-written as follows:

$$N_{k_{il}} \leq X_{K_{il}}$$
$$N_{k_{il}} \leq W_{k_{il}}$$
$$N_{k_{il}} \geq X_{K_{il}} + W_{k_{il}} - 1$$

The legitimacy of this conversion is because both sides of the multiplication are binary variables. For the flow k, the selected switches for each order are stored in N. Entry and exit switches are added to the selected switches N, resulting in F as shown in Equation (5). For each order in flow k, the right hand side of Equation (6) is equal to zero except for the source (+1) and destination (-1) switches. The left hand side of this equation shows the difference between out-degree and in-degree of each switch, so each zero or one for V shows the selection of a link between two switches for a particular order and flow. To make sure that selected routes in Equation (6) are not very long and they do not exceed maximum tolerable delay for a flow, Equation (7) is used.

The maximum number of flows using a service on a switch simultaneously depends on the nature of a service such as the computation needed for that service, and the hardware capabilities of the switch running it. P reflects this for each combination of service and switch. This may be specified a priori or obtained experimentally. Equation (8) stores the mapping between switches and services for the flow k in variable A. A is limited to the maximum number of flows defined for a service in Equation (9). Equation (10) and Equation (11) set the variable U to the maximum value of link or core utilization and finally Link utilization is enforced by Equation (10).

A. Variations in Formulation

The formulation above can be the foundation, and we consider a few alternatives below.

1) Consider only Link or Core utilization: Equation (11) or Equation (10) may be omitted so that we seek to just optimize either the core or link utilization. This variation is helpful if one of the objectives, link or core utilization, has a high value and cannot be decreased at all. Hence it is better to just minimize the other one and increase the available capacity in the network.

2) Combined formulation with a penalty function for delay: The combined formulation, which covers link and core utilizations at the same time does not seek to minimize the delays experienced by flows, but just ensures the delay is below the tolerable value. After minimizing the utilization of the links and cores, it may be desirable to minimize the delay as well. For this, we can change the objective function to the following:

$$\text{Minimize: } U + MaxDelay/LV$$

LV is a large enough integer to reduce the effect of MaxDelay to an amount lower than the minimum variance of U. For example if the finest granularity of variance in U is equal to 0.01, LV may be two orders larger than the possible MaxDelay. Therefore the effect of the delay will be limited to one percent. With an equal value of U, the solution with smaller MaxDelay is chosen. MaxDelay, the maximum delay observed by a flow, is:

$$\sum_{l'} \sum_{i,m} V_{k_{l'_{im}}} D_{im} \leq MaxDelay \tag{12}$$

This constraint reflects the fact that MaxDelay is larger or equal to the total delay of any flow.

IV. Developing simple Heuristics

The solution time for the optimal placement with the MILP grows exponentially with the number of flows, thus limiting the scale of the problem that can be solved. Moreover, once the optimal placement is arrived at, any changes in the set of flows or the assigned services at the switches requires the problem to be solved all over again, which may not be practical . We seek heuristic approaches to solve these problems.

3) Heuristic-A: This heuristic is a multi-step greedy algorithm without using the MILP. At first we select flows one by one and try to place their required services on free cores along their shortest path. In the next step, we seek to share the already assigned cores on the shortest path. In step three, we then look further at the neighboring switches and use their free cores to accommodate necessary services. In the next step we try to share already assigned cores in the neighboring switches with flows whose requirements are not yet satisfied. If after all these previous steps a flow still does not have all the necessary services, Heuristic-A adds a node from the neighboring switches to the shortest path and repeats all the aforementioned steps.

4) Heuristic B, B+, and B+COR: For these heuristics, instead of solving the problem for all the flows at the same time, we divide the flows into the groups. We start from the first group, and solve the optimization problem for it. Based on the solution, the problem is updated again and we solve the updated problem for the next group. We continue this process until all flows are supported. We call this heuristic B. To be able to use information from a previous step, we have defined new set of variables and have changed some of the constraints of the MILP formulation. We call these variables $preM$, $preUL$, and $preUC$. The first, $preM$, represents the union of assignments of services to the cores in the previous rounds. The M variable in the formulation is replaced with $M + preM$ throughout. $preUL$ and $preUC$ represent the link utilization and core utilization respectively. $preUL$ is added to the left hand side of (11) and $preUC$ is added to the left hand side of (9).

In the MILP formulation we are not minimizing the number of used cores, so as a result some cores may be assigned but not used. To make this heuristic more efficient, we add a pre-processing phase which eliminates unused cores from $preM$. In the rest of the paper, this enhanced version is called B+. It is efficient in curing the both drawbacks of original formulation: if we do the processing in small groups, the problem is solved very fast; and this method can be used to avoid solving the problem all over again. Just the problem for the added flows can be solved.

After studying the results with B+, we found that other methods of partitioning flows may help get better results. The most effective was $B + COR$. In first step, the shortest path between entry and exit switches for different flows is chosen. Then, the number of flows who have a common switch in their shortest path is counted and assigned to that switch. Then switches are sorted in ascending order based on the number of flows passing through them. Less crowded switches are selected first. The reason for efficiency of this algorithm is that trying to minimize utilization of bottleneck switches overuse lots of resources of neighboring switches in the hope of lowering the overall utilization. But if we place flows at crowded switches last, the necessary resources at neighboring switches are allocated and are used to satisfy the needs of hotspot flows. In other words, starting from the least crowded switches, helps us to have a more even distribution of resources in the network.

5) Heuristic C: The default formulation doesn't support minimizing the number of used cores, because the original formulation seeks to minimize the utilization at the bottlenecks. But with Heuristic C, similar to the B, flows are processed in groups. Therefore having more unassigned cores provides a greater opportunity for placing subsequent flows. It provides the flexibility to define the type of a service that has to run on a core for incoming flows. So Heuristic C is similar to B, except that the objective function is changed to: *Minimize U+ (MaxDelay / LV) + (TotalCores / VLV)*

The value of $TotalCores$ is calculated based on the following expression:

$$\sum_{i,j} M_{ij} \leq TotalCores$$

VLV is defined similarly to LV. The LV should be large enough that the maximum variance of TotalCores over LV should be smaller than minimum variance of MaxDelay over LV.

V. Evaluation

We evaluate the effectiveness of our MILP formulation on an example network topology, and initially use a default where all the flows have the same service chain comprising 5 services. All services support up to 10 flows on a single core, with the exception of $Service_4$ which only supports up to 4 different flows on a switch core. We use an off-the-shelf solver to solve the MILP and the non-optimization components are developed in Java. All the switches have homogeneous processing capability with 2 available CPU cores. The default topology in experiments is the network topology of AS-16631

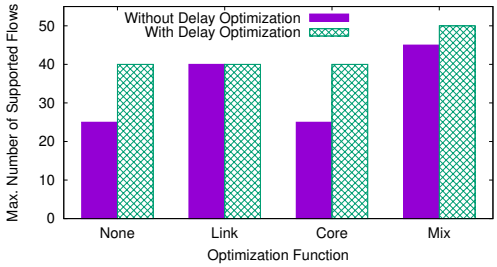

Fig. 3. Maximum number of fitted flows for different objective functions

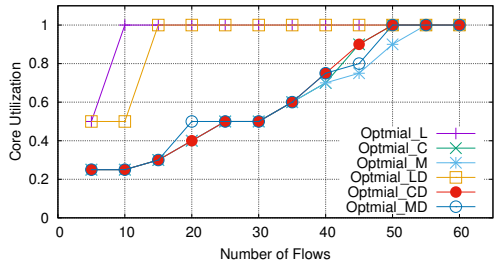

Fig. 4. Max. core utilization for problems with different objective functions

from Rocketfuel [10] with 22 nodes and 64 links. We run a total of 12 experiments, with varying number of flows: 5 to 60 in steps of 5.

First we investigate different objective functions, $\_C$, $\_L$ and $\_M$, each minimizing CPU load, traffic load on the links or both CPU and link load, respectively. They can also minimize the maximum flow delay, after minimizing its main objective and this is represented by a $\_D$ suffix. To show that the optimization approach can support additional flows as they arrive, we generated random flows in groups of five each. We then placed these flows using the MILP with the different objective function. If the placement was feasible, another group of five was added to the network. The results of this experiment is depicted in Figure 3. Each objective function with and without maximum delay minimization are shown as distinct bars. 'None' represents a placement without any optimization, and 'Mix' shows the results with the objective function $\_M$ combining link and core utilization. The significant improvement in capacity by minimizing delay in the 'None' and 'Core' cases shows that minimizing the maximum delay can be useful. However, minimizing the maximum link utilization is effective as seen with the $\_M$ (Mix) objective function, which has a higher capacity than both the 'Core' cases. Enhancements by adding delay minimization to the Mix case shows even better performance. Figures 4, 7 and Figure 8 show the maximum utilization of the most highly utilized core, link and the link or core for the solution with the 3 different objective functions $\_C$, $\_L$ and $\_M$. When only one metric (core or link utilization) is minimized, the utilizations of the other resource grows quickly, resulting in that resource also not being used by additional flows and reducing the flexibility for the placement algorithm. The combined objective function ($\_M$) is able to compensate for the shortage in one of the resources by increase in the usage of the other resource. The core and link utilization thus increase together.

As shown in Figure 8, when delay minimization is also used, we observe higher utilization compared to the case where there

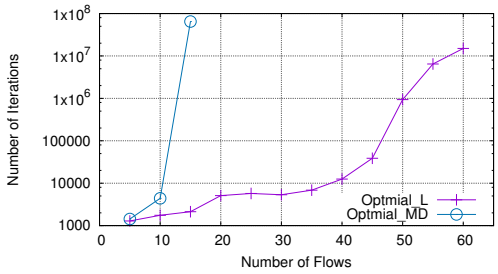

Fig. 5. Number of solver iterations for different objective functions

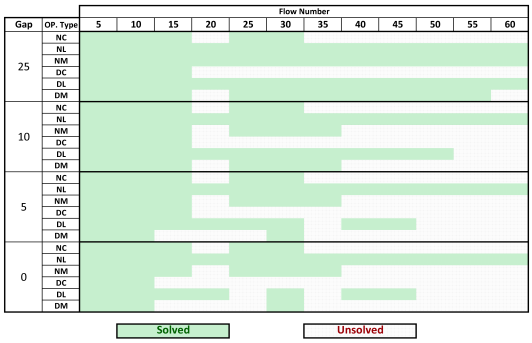

Fig. 6. Status of found solution for each problem

is no delay minimization (even though delay minimization is a second level optimization). This is because we only achieve a suboptimal MILP solution. In some cases, especially for large number of flows, the solver is not able to reach to the optimal solution even after running the solver for a long time. In these cases, we have reported the best achieved (suboptimal) solution for that problem. Figure 6 shows which problems produced solutions within a "gap" percentage of the optimal solution. For example if we reached to a solution with an objective function with 17% higher value than the possible optimal answer, we mark it as 'solved' if the gap is 25%. It is marked as 'unsolved' for a gap of 0, 5, and 10 percent. The figure shows that the optimal solutions are reachable for all the cases up to 10 flows. However, we are not able to find the optimal solution even for 15 flows in some cases. This challenge is because of the exponential nature of this problem. The number of iterations needed for getting the optimal solution is shown in Figure 5. $Optimal\_MD$ needs substantial computation time even for 15 flows. As a result, we propose heuristics to overcome this scalability challenge.

Figure 9 shows the maximum delay of flows in the network, which is below 100 (the value set in the constraint for the maximum delay tolerated). Using the delay in the objective

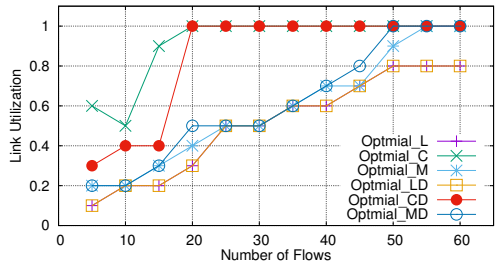

Fig. 7. Max. link utilization for problems with different objective functions

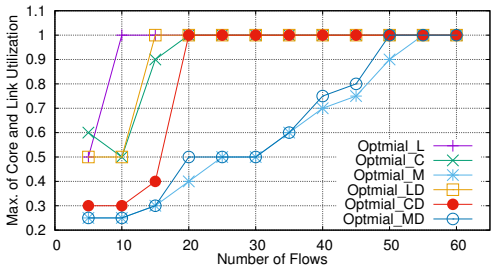

Fig. 8. Max. core and link utilization for different objective functions

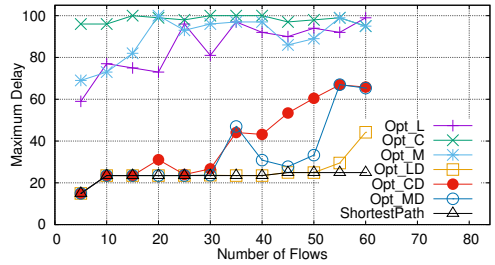

Fig. 9. Maximum delay of flows for different objective functions

function does help in reducing the maximum delay of flows, and the maximum delay is less that three times the maximum delay of the shortest path even in the worst case.

A. *Evaluation of Heuristics*

Our main goal is to support as large a number of flows as possible in a network. The maximum number of supported flows by each method is illustrated in Figure 10. We are able to solve the original optimization problem, which returns the optimal placement, for up to 60 flows in this network. However, the time needed for computation of optimal solution in this network with 22 switches and 60 flows, is more than a day even on a server.

We therefore look at using the various heuristics described in the previous section. The most scalable method, $B+COR$, can fit 55 flows in the network, while solving the placement in a matter of seconds. Based on the bandwidth required for flow, the link capacities in the network and the maximum number of flows supported on each switch, the maximum number of flows that can be carried in this network is 60 flows, even with optimal placement and steering,. With our heuristics, we could fit up to 55 flows (92% of optimal solution). To show the scalability of our heuristic approach, we increased the capacity of network by a factor of 10 and then 100, by changing the available bandwidth and the flow capacity of each service. The resulting number of fitted flows in the network by heuristic $B+$ is shown in Figure 11.

The comparison between maximum delay of algorithms and optimal solution is in Figure 12 shows the delay of the proposed heuristics is comparable to delay value of optimal solution (even lower in some cases). Since the optimizer first optimizes the core and link utilization and delay only as a second step.

VI. Related Work

A large body of work exists for object placement and traffic steering. In our context, some recent approaches are:

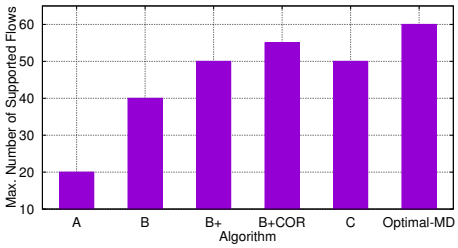

Fig. 10. Maximum number of fitted flows for different algorithms

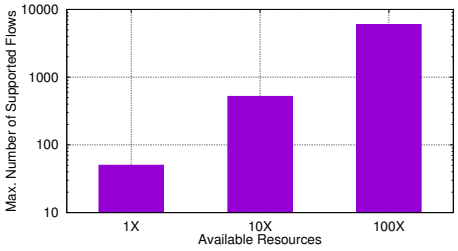

Fig. 11. Maximum number of fitted flows for different network capacities

1) CoMb[11]: CoMb is an architecture supporting consolidated middleboxes. Its optimization problem places services along the pre-defined paths of flows, leveraging the common parts of different services. One consequence is that it lacks the dynamic nature of placing NFVs in the network and the corresponding routing.

2) Stratos [9]: Stratos is an orchestration layer for virtual middleboxes in clouds. It uses an ILP formulation to decide how to steer flows through middleboxes, with a binary cost function between switches. Decisions about placement are made by an online rack-aware heuristic.

3) SIMPLE [12]: They use both an online and offline formulation. The main focus of the offline formulation is to keep limiting the size of forwarding rules due to the limits in the TCAM memory of SDN switches. The online formulation is for online load balancing on the available switches. However, it does not address the possibility of dynamic instantiation of services.

4) StEERING [8]: This work describes a system to dynamically instantiate a service at a desired network node and route traffic through such a service. The placement of the service is primarily through a heuristic.

5) T-Storm [13]: T-Storm is an online scheduler for Storm stream processing. There are some similarities between this scheduler and the placement and steering needed in an SDN-NFV network. For example, assigning executors to slots on workers resembles the service assignment to switches. However the proposed algorithm in T-Storm does not satisfy our needs as it primarily allocates executors based on the incoming traffic load and does not consider the network topology for decision making.

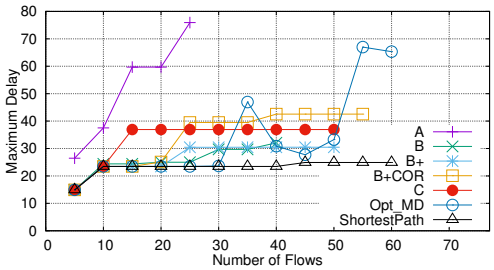

Fig. 12. Maximum flows' delay for different algorithms

VII. Conclusions

NFVs supported by an SDN protocol suite, provide a great opportunity to have higher level of flexibility and dynamicity in networks. However they introduce new challenges of joint service placement and traffic steering. In this paper, we provided a MILP formulation for this problem, which not only determines the placement of services and routing of the flows, but also seeks to minimize network link and network node core utilizations. We have devised heuristics to provide the opportunity to perform the placement incrementally without imposing a significant penalty. Our ongoing work is to enhance the scalability of our solution approach and to implement and demonstrate the use of the placement approach in practice.

References

[1] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to sdn. *Queue*, 11(12):20:20–20:40, December 2013.

[2] Ian F. Akyildiza, Ahyoung Leea, Pu Wangb, Min Luoc, and Wu Chouc. A roadmap for traffic engineering in sdn-openflow networks. *Computer Networks (Elsevier)*, 2014.

[3] European Telecommunications Standards Institute. Network functions virtualization (nfv): Architectural framework. *White Paper*, 2014.

[4] Jinho Hwang, K. K. Ramakrishnan, and Timothy Wood. Netvm: High performance and flexible networking using virtualization on commodity platforms. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 445–458, Berkeley, CA, USA, 2014. USENIX Association.

[5] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. Clickos and the art of network function virtualization. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 459–473, Seattle, WA, April 2014. USENIX Association.

[6] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K. Reiter, and Guangyu Shi. Design and implementation of a consolidated middlebox architecture. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 24–24, Berkeley, CA, USA, 2012. USENIX Association.

[7] Mayutan Arumaithurai, Jiachen Chen, Edo Monticelli, Xiaoming Fu, and Kadangode K. Ramakrishnan. Exploiting icn for flexible management of software-defined networks. In *Proceedings of the 1st International Conference on Information-centric Networking*, INC '14, pages 107–116, New York, NY, USA, 2014. ACM.

[8] Ying Zhang, Neda Beheshti, Ludovic Beliveau, Geoffrey Lefebvre, Ravi Manghirmalani, Ramesh Mishra, Ritun Patney, Meral Shirazipour, Ramesh Subrahmaniam, Catherine Truchan, et al. Steering: A software-defined networking for inline service chaining. In *ICNP*, pages 1–10, 2013.

[9] Aaron Gember, Anand Krishnamurthy, Saul St. John, Robert Grandl, Xiaoyang Gao, Ashok Anand, Theophilus Benson, Aditya Akella, and Vyas Sekar. Stratos: A network-aware orchestration layer for middleboxes in the cloud. *CoRR*, abs/1305.0209, 2013.

[10] S Neil, M Ratul, and A Thomas. Quantifying the causes of path inflation. In *Proc. ACM SIGCOMM'03*, 2003.

[11] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K Reiter, and Guangyu Shi. Design and implementation of a consolidated middlebox architecture. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 24–24. USENIX Association, 2012.

[12] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. Simple-fying middlebox policy enforcement using sdn. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 27–38. ACM, 2013.

[13] Jielong Xu, Zhenhua Chen, Jian Tang, and Sen Su. T-storm: Traffic-aware online scheduling in storm. In *Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on*, pages 535–544. IEEE, 2014.