Flow Scheduling of Service Chain Processing in a NFV-based Network

Yang Chen, *Student Member, IEEE,* and Jie Wu, *Fellow, IEEE*

Abstract—Network Function Virtualization (NFV) is changing the way we implement the network functions from expensive hardware to software middleboxes, called Virtual Network Functions (VNFs). Flows always request to be processed by several middleboxes in a specific order, which is known as a service chain. Most researches study the middlebox placement problem and few of them pay attention to the flow scheduling of a deployed service chain, resulting in poor control of flow completion times. However, the flow completion time is an extreme metric to evaluate the performance of a network. Therefore, we focus on the service chain scheduling problem. We aim to minimize the flow completion time in two aspects: the longest completion time (makespan) and the average completion time. First, a transmission and processing delay model is proposed to formulate the communication latency behaviour of flows being processed by middleboxes. When there are only two middleboxes in the service chain, we propose one optimal solution for each aspect, respectively. For a service chain with an arbitrary length, we prove the NP-hardness of our problem in both aspects and two corresponding heuristic algorithms are designed, which are extended from our proposed optimal solutions for a service chain with a length of two. Real testbed experiments and extensive simulations are conducted to evaluate the performance of our proposed algorithms in various scenarios.

Index Terms—Middlebox, transmission delay, processing time, service chain.

1 Introduction

The role of modern networks has transformed to provide various types of network services (such as security, performance optimization, cross-protocol interoperability, and value-added services) beyond providing basic connectivity services. Realizing such network services usually requires multiple network functions, also called middleboxes, to be chained together in some order, which is known as the service chain [1]. For example, in operator networks [2], data centers [3], mobile networks [4], and enterprise networks [5], network operators often require traffic to pass the service chain: NAT, Firewall, and IDS in the sequence order [6], which is shown in Fig. 1. Virtual Network Functions (VNFs) become the modern implementation of network functions as the replacement of expensive traditional hardwares[7]. These middleboxes are executed on switch-connected servers. Middleboxes play an increasingly important role in modern networks [8]. As Software Defined Networking (SDN) emerges, so does a tendency to incorporate SDN and NFV in concerted ecosystems [9, 10]. With the intense requirement of high performance networks, SDN manoeuvres through NFV traffic and schedules flows to be processed by middleboxes [11]. However, most current works focus on the VNF deployment and pay little attention to flow scheduling (order of flows to be processed by a service chain), resulting in poor control of the flow completion time.

The flow completion time is important to evaluating the performance of the network, which is highly demanded nowadays [12]. The service chain with multiple middleboxes makes the flow processing order problem more complicated. This is because different flows on each middlebox of the service chain have various processing times. A flow cannot start (or finish) being processed in a middlebox of a service chain before its starting (or finishing) time of its previous middlebox plus the transmission delay between these two middleboxes. Additionally, the link transmission delay between the locations of deployed middleboxes is necessary to take into consideration, which further complicates our problem.

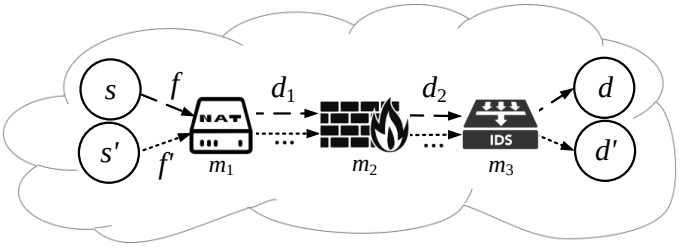


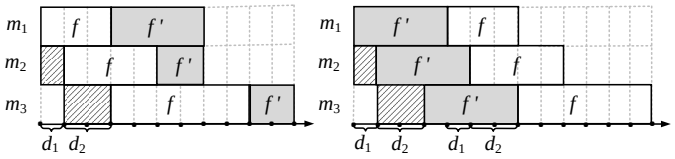
Fig. 1: An illustrating example.

We illustrate the importance of the flow processing order in an example, shown in Fig. 1. Circles denote the switches and rectangles denote middleboxes deployed on switch-connected servers. Suppose there are three middleboxes m_1, m_2, and m_3 in the service chain and two flows f (source s, destination d, and the dash-line path) and f' (source s', destination d', and the dot-line path) request to be processed by the service chain. We omit the drawings of servers and switches with no middlebox deployed. The processing time of each flow on each middlebox is shown

• *Yang Chen and Jie Wu are with Center for Networked Computing, Temple University, USA. E-mail: {yang.chen, jiewu}@temple.edu.*

Flows \ Middleboxes	m_1	m_2	m_3
f	3	4	6
f'	4	2	2

TABLE 1: Processing times.



(a) f before f'. (b) f' before f.

Fig. 2: Different scheduling orders.

in the Tab. 1. The transmission delay between m_1 and m_2 is $d_1 = 1$ and the transmission delay between m_2 and m_3 is $d_2 = 2$. Because of the transmission delay between two middleboxes, a flow cannot start (or finish) being processed in the middlebox m_i, before its starting (or finishing) time of the middlebox m_{i-1} plus the transmission delay between these two middleboxes. It means that f' is "delayed" to be processed on m_2 and m_3 by the processing on m_1. More specifically, the completion time of f' in middlebox m_2 cannot be less than the completion time of f' in middlebox m_1 plus the delay d_1, which is $4+1 = 5$. The same situation occurs to the completion time of f' in middlebox m_3, which is $5 + 2 = 7$. This illustrates that the processing times of f' in middleboxes m_2 and m_3 are longer than the given processing times of m_2 and m_3, because of the transmission delays. Thus, if flow f is scheduled before flow f', the makespan is 11, which is shown in Fig. 2(a). If flow f' is scheduled before flow f, the makespan is 14, which is shown in Fig. 2(b). The difference comes from the constraint of the completion times between adjacent middleboxes in the service chain.

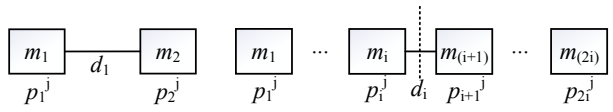
In this paper, we aim at minimizing the total time of transmission delay and processing delay in two aspects: makespan (the longest completion time) and the average completion time (the total completion time of all flows divided by the number of flows). We build a flow transmission time and processing delay model to formulate its latency behavior and control the flow processing sequence in the network. We are given a deployed service chain with the processing times of flows and transmission delays between middleboxes. We propose two optimal solutions when there are only two services in the service chain. With a service chain of an arbitrary length, two heuristic algorithms are designed with insights. Extensive simulations and experiments are conducted to evaluate the performance of our proposed algorithms in various scenarios.

The remainder of this paper is organized as follows. Section 2 surveys related works. Section 3 describes the model and formulates the problem. Section 4 introduces our optimal algorithms to arrange flows for a service chain with only two middleboxes. In Section 5, we handle cases with an arbitrary number of middleboxes in a service chain and propose two heuristic algorithms with insights. Section 6 includes the testbed experiments. Simulations are conducted in Section 7. Section 8 concludes the paper.

2 Related Work

Most researches on the middlebox problem focus on efficiently deploying a chain of middlebox instances whose middleboxes conform to a strict serving sequence as a totally-ordered set [13]. Rost et al. [14] prove the NP-completeness and inapproximability of the service chain deployment under different constraint settings, extended from the virtual network embedding problem. [15] initiates the study of approximation algorithms and proposes a performance-guaranteed solution under the offline setting (given multiple flows), based on randomized rounding of Linear Programming, to maximize the total profit of satisfied flows. In [16], they propose a context-free language to formalize the chaining of network functions and describe the middlebox resource allocation problem as a mixed integer quadratically constrained program. Rami et al. [17] locate middleboxes in a way that minimizes both the new middlebox setup cost and the distance cost between flows' paths and middleboxes and provide near optimal approximation algorithms to guarantee a placement with a theoretical-proven performance. Flowtag [10] uses SDN to support service chaining by redefining certain packet header fields as tags to track flows. Both [18] and [19] aim to maximize the number of requests for each service chain. [18] proposes a systematic way to tune the proper link and server resource usages in the joint problem of middlebox placement and path selection. Li et al. [19] present the design and implementation of NFV-RT, a system that dynamically provisions resources in an NFV environment to provide timing guarantees so that the assigned flows meet their deadlines. Stratos [20] is proposed as a controllable and scalable framework for the efficient deployment of virtual middleboxes. OpenNF [21] enforces the functions of NFV with SDN, and provides a rich set of NFV/SDN APIs (move, copy, share, etc.) for software middleboxes management, which makes it feasible to dynamically schedule the middleboxes according to the changing traffic. ClickOS [22] is put forward to improve the running efficiency of virtual middleboxes by optimizing the underlying Virtual Machines. ClickOS can launch the middlebox software within about 30ms, which makes it possible for dynamic deployment (addition, deletion, etc.) of middleboxes based on online changing traffic. [23] studies the joint optimization on functions mapping and preemptive scheduling in NVF, with the objective to minimize the total completion time. [24] considers different user-level SLAs, such as latency and cost, while scheduling such services. Their aim is to reduce overall turnaround time for the complete end-to-end service in service function chains and reduce the total traffic generated with a novel fair weighted affinity-based scheduling heuristic to solve this problem. However, both these works do not include a performance-guaranteed solution.

However, many data center applications are sensitive to latency. One source of latency is network congestion as throughput-intensive applications cause queuing at switches that delays traffic from latency-sensitive applica-



(a) Two ordered services. (b) Multiple ordered services.

Fig. 3: A service chain.

tions. Existing techniques to combat the queuing problem are to prioritize flows such that packets from latency-sensitive flows can jump the queue [25], to centrally schedule all flows for every server so that no flow has to queue [26], or to pace end host packets to achieve guaranteed bandwidth for guaranteed queuing [27]. These techniques assume the shortest path forwarding. Today's data center fabrics have rich path-redundancy in nature, so non-shortest paths can be exploited to use path redundancy and spare capacity for mitigating network congestion [28]. As policy rules chaining can effectively shape the network traffic (packets need to follow policy paths), they can be chained over non-shortest paths to mitigate congestion-led queuing since propagation delay on physical links is predictable and smaller than queueing delay [29]. [30] proposes an online orchestration framework for cross-edge service function chaining, which aims to maximize the holistic cost efficiency, via jointly optimizing the resource provisioning and traffic routing on-the-fly. [31] propose an online algorithm for the dynamic service chain deployment problem with an objective of minimizing the cost for provisioning VNF instances on the go, and provide a bandit-based online learning algorithm to place the VNF instances which minimizes the congestion. However, both these papers do not consider the processing delay incurred by the VNFs.

Our paper is motivated by the classic flow shop problem [32]. In the flow shop problem, it assumes that all phases are set up in series and that jobs have to follow the same route to be executed. Our problem resembles the flow shop, but there is transmission delay between every two phases. [33] provides an optimal solution for minimizing the makespan with only two phases in a flow shop problem. It also proves that the general flow shop problem with k phases ($k > 2$) is NP-complete. In contrast to the makespan objective, results with regard to the average completion time objective are much harder to obtain. Minimizing the average completion time with two phases is already strongly NP-hard. Almost all existing works focus on heuristic algorithms [34, 35]. Zheng et al. in [36] provide an optimal solution when all jobs can be paired, which is restrictive. Furthermore, all above works assume that there is no transmission delay between phases. In this paper, we include the transmission delay between phases, which is different from the classic flow shop problem. And we aim at minimizing the total time of transmission delay and processing delay in both two aspects: makespan and the average completion time.

3 Model and Formulation

In this section, we first propose our network model and then formulate our problem.

3.1 Network Model

Before formulating the problem, we first present our model of the directed network, $G = (V, E)$, where $V = \{v\}$ is a set of vertices (i.e., switches) and $E = \{e\}$ is a set of directed edges (i.e., links). We use v to denote a single vertex, and $e_{vv'}$ as the edge from vertex v to vertex v'. Middleboxes are deployed in the network. A service chain is an ordered middlebox set, where each flow needs to be processed in a fixed order. We are given a set of unsplittable flows $F = \{f_j\}$, because flow splitting may not be feasible for applications that are sensitive to TCP packet ordering (e.g. video applications). Additionally, split flows can be treated as multiple unsplittable flows. We use f_j to denote a single flow j and p_i^j to denote the processing time of f_j on a middlebox m_i. The completion time of flow f_j on a middlebox m_j is represented as C_i^j. We denote C^j as the completion time of flow f_j across all the middleboxes in its service chain and all the links in its path. When flows are transmitting through edges, there are delays. We assume the delay between two middleboxes m_i and m_{i+1} in a service chain is d_i, which is identical for all flows. In this paper, we only consider the deterministic processing behavior of packets in the queuing model [37], whose processing time of each flow f is a constant.

To analyze processing behaviors of flows in middleboxes, a queuing model is employed in [38], and we extend it in this paper. Furthermore, the serving behavior in our queuing model would be more complex, as we would consider various middleboxes for different network functions. Therefore, we adopt two queuing models according to different processing behaviors of packets: deterministic model and exponential model [37]. In the deterministic model, the processing time of a single flow j on a middlebox m_i, denoted by p_i^j, is a fixed constant given value. In the exponential model, it follows the exponential distribution with a rate λ_i^j [39]. The expected processing time is $1/\lambda_i^j$.

3.2 Problem Formulation

In this paper, we schedule flows to be processed by a service chain. We have two different objectives: (1) minimize the makespan; (2) minimize the average completion time. The definitions are as follows:

Definition 1. The makespan C_{max} is defined as $\max_{f_j \in F}(C^j),\ \forall f \in F$, equivalent to the completion time of the last flow to finish being processed by the last middlebox in the service chain.

Definition 2. The average completion time is defined as $\frac{1}{|F|}\sum_{f_j \in F} C^j$, which is the total completion time of all flows divided by the number of flows. ($|\cdot|$ denotes the set cardinality)

We formulate our problem as follows:

$$\min \quad C_{max} \quad \text{or} \quad \frac{1}{|F|}\sum_{f_j \in F} C^j \tag{1}$$

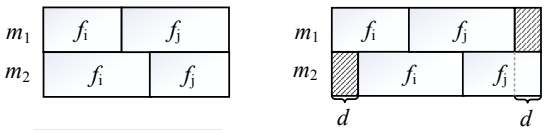
4 A Service Chain with Two Middleboxes

In this section, we study the case of the service chain that only includes two middleboxes, which is shown in Fig. 3(a). The processing time of flow f_j on middlebox 1 is p_1^j and its processing time on middlebox 2 is p_2^j.

Algorithm 1 Two Set Order Schedule (TSOS)

In: Flow processing times p_1^j and p_2^j, $\forall f_j \in F$ and the transmission delay d;
Out: The flow scheduling order;

1: Calculate the value of $p_2^j - p_1^j,\ \forall f_j \in F$;
2: Sort flows in decreasing order of $p_2^j - p_1^j$;
3: **return** The flow scheduling order.



(a) A pair. (b) A pair with a delay.

Fig. 4: Pairing flows.

Algorithm 2 Pairwise Schedule (PS)

In: Flow processing times p_1^j and p_2^j, $\forall f_j \in F$ and the transmission delay d;
Out: The flow scheduling order;

1: Sort all flows in increasing order of $\max\{p_1^j, p_2^j\}$;
2: **for** each subset of flows with the same $\max\{p_1^j, p_2^j\}$ **do**
3: Reorder flows by iteratively taking out a pair of flows of $\arg\max_i(p_2^j - p_1^j)$ and $\arg\max_i(p_1^j - p_2^j)$;
4: **return** The flow scheduling order.

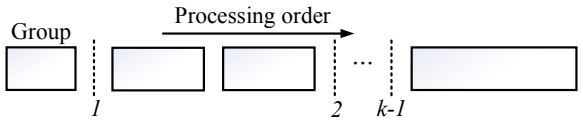
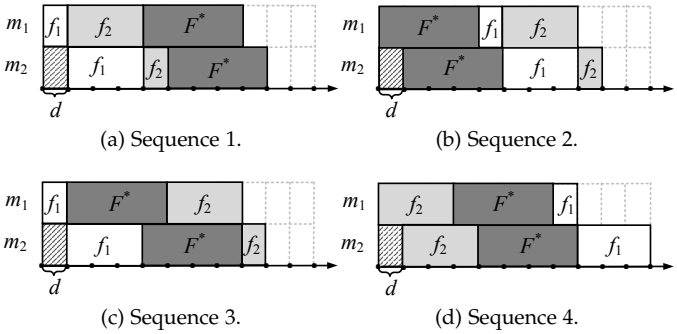


Fig. 5: Multiple flows.



(a) Sequence 1. (b) Sequence 2.
(c) Sequence 3. (d) Sequence 4.

Fig. 6: An illustration for the proof of Theorem 2.

4.1 Minimizing the makespan

To minimize the makespan, we propose an optimal algorithm, called Two Set Order Schedule (TSOS), in Alg. 1. We are given the processing time of each flow on the two middleboxes. We need to decide the processing order of all flows on the service chain with two middleboxes. The order of flows is returned in line 3. As the sorting of flows costs $|F|\log|F|$, its time complexity is $O(|F|\log|F|)$.

Theorem 1. TSOS algorithm is optimal for scheduling flows in a service chain with only two middleboxes.

Proof: The shortest makespan is no less than $d + \max\{\sum_{f_j \in F} p_1^j, \sum_{f_j \in F} p_2^j\}$. This is because at least all flows need to be processed by both middleboxes in order and the transmission delay d is the extra time. We need to find a schedule that is able to keep at least the second middlebox busy, which is the bottleneck of the scheduling. Moreover, we also need to avoid prolonging the processing time because of the transmission delay. If the completion time on the second middlebox is larger than the completion time on the first middlebox plus d, there is no need to prolong the time and at the same time the second middlebox keeps busy. Thus, we sort the value of $p_2^j - p_1^j,\ \forall f_j \in F$ in increasing order to extend the start time on the second middlebox. Thus, our algorithm has the least extension of the completion time of the last flow on the second middlebox, i.e. the minimum makespan. ■

4.2 Minimizing the average completion time

We present a pair-based scheduling policy. For clear presentation, the following definitions are introduced:

Definition 3. Flows f_i and f_j are a pair, if $p_1^i = p_2^j$ and $p_2^i = p_1^j$.

The definition of pair comes from [36], which is for the MapReduce optimization [40]. The pair in [36] is shown in 4(a), in which there is no delay between the starting time of one flow in two stages. However, there is delay between two middleboxes in a service chain. In this paper, the pair with a delay is shown in 4(b); there is a delay between the starting time of one flow in two stages. In order to minimize the average completion time, we propose an optimal algorithm, called Pairwise Schedule (PS), in Alg. 2. Line 1 sorts flows in the increasing order of $\max\{p_1^j, p_2^j\}$. Lines 2-3 pair flows by iteratively taking out a pair of flows of $\arg\max_i(p_2^j - p_1^j)$ and $\arg\max_i(p_1^j - p_2^j)$. The order of flows is returned in line 4. The processing order is illustrated in Fig. 5. As the sorting of flows costs $|F|\log|F|$, the time complexity of PS algorithm is $O(|F|\log|F|)$. It works well when a large portion of flows can be paired. Its optimality is stated as follows:

Theorem 2. The proposed Alg. 2 is optimal for scheduling flows if all flows can be pairwise executed in the optimal schedule. For each pair, the flow with $p_2^j > p_1^j$ executes before the flow with $p_2^j < p_1^j$.

Proof: We prove by induction. Let us start with a basic case, where F only includes two flows that can form a pair (denoted as f_1 and f_2). Suppose f_1 has $p_1^1 < p_2^1$ and f_2 has $p_1^2 > p_2^2$. We have two schedules: schedule one executes f_1 before f_2, and schedule two executes f_2 before f_1 . Then, the flow makespans of f_1 and f_2 are shown as follows:

Completion time	C^1	C^2
f_1 before f_2	$d + p_2^1$	$\max(p_1^1 + p_1^2, p_1^1 + p_2^2) + d$
f_2 before f_1	$d + p_1^2 + p_2^1$	$d + p_2^2$

We have $p_2^1 = p_1^2$ according to the definition of the pair. Since f_1 is $p_1^1 < p_2^1$, we have $p_1^1 + p_1^2 < p_2^1 + p_2^2$.

Hence, schedule one has a smaller average flow makespan by executing the flow with $p_1^j < p_2^j$ before the flow with $p_1^k > p_2^k$. For induction, let us consider an existing schedule of S. It executes flows that can form a pair. Let F^* denote a subset of F is are consecutively and pairwise executed in S. Let τ denote the average flow makespan of F^*, but be calculated from the execution time of F^*. Let t^* denote the total processing time of middlebox m_1. Since flows in F^* are strongly paired, t^* is also the total processing time of middlebox m_2. The induction step adds one more pair of flows to schedule S (i.e. f_1 with $p_1^1 < p_2^1$ and f_2 with $p_1^2 > p_2^2$). As shown in Fig. 6, there exist four possible sequences to incorporate f_1 and f_2 into S: S_1 executes f_1 and f_2 before F^*; S_2 executes f_1 and f_2 after F^*; S_3 executes f_1 before F^*, and f_2 after F^*; S_4 executes f_2 before F^* and f_1 after F^*. S_1 and S_2 execute f_1 and f_2 in a pairwise manner, while S_3 and S_4 execute f_1 and f_2 in an interwoven manner. If f_1, f_2, and F^* are scheduled at time t_0, then their flow makespans are:

Time	C^1	C^2	C^*
S_1	$t_0 + p_2^1 + d$	$t_0 + p_2^1 + p_2^2 + d$	$t_0 + p_2^1 + p_2^2 + \tau + d$
S_2	$t_0 + p_2^1 + t^* + d$	$t_0 + p_2^1 + p_2^2 + t^* + d$	$t_0 + \tau + d$
S_3	$t_0 + p_2^1 + d$	$t_0 + p_2^1 + p_2^2 + t^* + d$	$t_0 + \tau + p_2^1 + d$
S_4	$t_0 + p_2^1 + p_1^2 + t^* + d$	$t_0 + p_1^2 + d$	$t_0 + p_1^2 + \tau + d$

It is trivial that S_4 is always worse than S_3, due to its underutilization of middlebox m_1. Meanwhile, the average flow completion time of S_1, S_2, and S_3 are shown as follows:

$$
\begin{aligned}
S_1 &: t_0 + \frac{\left[|F^*|\cdot(p_1^1+p_1^2)\right] + \left[p_2^1+(p_1^1+p_1^2)+|F^*|\cdot\tau\right]}{|F^*|+2} + d \\
S_2 &: t_0 + \frac{\left[\quad 2t^* \quad\right] + \left[p_2^1+(p_1^1+p_1^2)+|F^*|\cdot\tau\right]}{|F^*|+2} + d \\
S_3 &: t_0 + \frac{\left[\ |F^*|\cdot p_2^1+t^*\ \right] + \left[p_2^1+(p_1^1+p_1^2)+|F^*|\cdot\tau\right]}{|F^*|+2} + d
\end{aligned}
\tag{2}
$$

Here, $|F^*|$ denotes the number of flows in F^*. A notable point is $|F^*| \cdot (p_1^1 + p_1^2) < |F * | \cdot 2p_2^1$ according to the definitions of f_1 and f_2. We have the following inequality: $\frac{|F^*|\cdot(p_2^1+p_2^2)+2t^*}{2} \leq \frac{|F^*|\cdot 2p_2^1+2t^*}{2} = |F^*| \cdot p_2^1 + t^*$. The mean of two unequal numbers is always larger than the smaller one of these two numbers. Therefore, we have: $\min\{|F^*|\cdot(p_2^1+p_2^2), 2t^*\} \leq |F^*| \cdot p_2^1 + t^*$. The two equations indicate that either S_1 or S_2 has the smallest average flow makespan. Hence, f_1 and f_2 should be pairwise executed when being incorporated into F^*. By induction, flows that can form a pair should be pairwise executed in the optimal schedule. We also conclude that for each pair, the middlebox m_1 is executed before middlebox m_2. Therefore, the proof of the theorem is complete. ■

Theorem 3. The proposed Pairwise Schedule algorithm is optimal when all flows are simultaneously $p_2^j < p_1^j$ (or $p_2^j > p_1^j$ or $p_2^j = p_1^j$).

Proof: When all flows in F simultaneously have $p_1^j > p_2^j,\ \forall f_j \in F$, middlebox m_2 has almost no impact on the flow makespan except for transmission delay. This is because the middlebox m_2 is always underutilized for each flow. At this time, Alg. 2 schedules flows according to their p_1^j. It is trivial that flows with shorter processing time p_1^j should be executed earlier to minimize the average flow makespan, since the smaller flows can finish earlier. When all flows in F are simultaneously $p_1^j = p_2^j,\ \forall f_j \in F$ or $p_1^j < p_2^j,\ \forall f_j \in F$, the scenario is similar, and thus, the proof is complete. ■

We can also construct a new flow set of F' from F. Each pair of flows in F (say f_i and f_j) is mapped to a flow in F'. The mapped flow in F' has map and shuffle workloads of $p_1^i + p_1^j$ and $p_2^i + p_2^j$, respectively. By the definition of the pair, we have $p_1^i + p_1^j = p_2^i + p_2^j$. Therefore, each flow in F' is balanced. Basically, F' is constructed by merging each pair of flows in F. According to Theorem 1 in [36], when F can be decomposed to pairs of flows, flows that can form a pair are pairwise executed in the optimal schedule of F. The optimal schedule for F' is the same as the one for F. While each flow in F' is balanced, it is trivial that flows with lighter workloads should be executed earlier to minimize the average flow makespan, since the smaller flows can finish earlier. If a flow with a heavier workload is executed before a flow with a lighter workload, then a swap of their execution order always leads to a smaller average flow makespan. Hence, Alg. 2 is optimal, when F can be decomposed to pairs of flows or all flows are simultaneously $p_2^j < p_1^j$ (or $p_2^j > p_1^j$ or $p_2^j = p_1^j$).

A potential problem of Alg. 2 is the flow processing time granularity when flows cannot be perfectly strongly paired. Line 3 in Alg. 2 pairs jobs with the same dominant processing time, i.e., the same $\max(p_1^j,\ p_2^j)$. If each flow has a unique dominant processing time, then the pairing process is skipped and thus becomes useless. To control the granularity, we additionally introduce a discretization process before applying Alg. 2. Let δ denote the discretization step, and the first and second processing times on the two middleboxes of each flow being rounded to the nearest multiple of δ. A larger δ represents a coarser processing time granularity, with more flows sharing the same dominant processing time. A smaller δ brings fine-grained processing time, where fewer flows share the same dominant processing time.

5 A SERVICE CHAIN WITH MULTIPLE MIDDLEBOXES

In this section, we study the case in which the service chain includes more than two middleboxes (multiple middleboxes), which is shown in Fig. 3(b). We illustrate the relationship among processing times, delays, and two flows in Fig. 7.

5.1 NP-hardness

We prove the NP-hardness of scheduling flows in a service chain with more than two middleboxes.

Theorem 4. Scheduling flows in a service chain with multiple middleboxes to minimize the makespan is NP-hard even when there is no transmission delay.

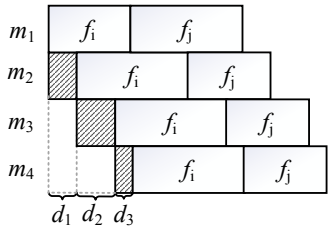


TABLE 2: Processing times.

	m_1	m_2	m_3	m_4
f_1	1	2	3	4
f_2	2	1	3	4
f_3	2	2	4	6
f_4	1	5	5	3

Fig. 7: Multiple middleboxes in a service chain.

Algorithm 3 Slope Heuristic Algorithm (SHA)

In: Flow processing times p_i^j, $\forall f_j \in F, m_i \in M$ and the transmission delay $d_i \in D$;
Out: The flow scheduling order;

1: $p_i^j = p_i^j + d_i$;
2: Sort flows in decreasing order of $A_j = -\sum_{i=1}^{|M|}(|M| - (2i-1))p_i^j$;
3: **return** The flow scheduling order.

Proof: We prove the NP-hardness of a special case where no transmission delay exists between any pair of middleboxes. In this case, our problem is equivalent to the classic flow shop problem, which is known to be NP-hard. Thus, scheduling flows in a service chain with more than two middleboxes with the minimum makespan is NP-hard. We prove the NP-hardness from the classic NP-hard problem 3-partition. Given integers $a_1, ..., a_{3t}, b$, under the usual assumptions, let the number of flows n equal $4t+1$ and let $p_1^0 = 0, p_2^0 = b, p_3^0 = 2b$; for flows $f_1, ...f_{t-1}$, we have $p_1^j = 2b, p_2^j = b, p_3^j = 2b$; for flow f_t, we have $p_1^t = 2b, p_2^t = 2b, p_3^t = 0$; for flows $f_{t+1}, ...f_{4t}$, we have $p_1^j = 0, p_2^j = a_j, p_3^j = 0$. Let $z = (2t+1)b$. A makespan of value z can be obtained if the first $t+1$ flows are scheduled according to sequence $0, 1, ..., t$. These $t+1$ flows then form a framework, leaving t gaps on middlebox m_2. Flows $t+1, ..., 4t$ have to be partitioned into t sets of three flows each and these t sets have to be scheduled in between the first $t+1$ flows. A makespan of value z can be obtained if and only if 3-PARTITION has a solution. ■

It is harder to obtain results of minimizing the average completion time, which is also NP-hard. In the following, we propose two heuristic solutions for the different objectives.

5.2 Minimizing the makespan

For more than two middleboxes in the chain, it proves NP-hard to optimally schedule the flows. We propose an algorithm, called Slope Heuristic Algorithm (SHA), which is extended from the TSOS algorithm. The transmission delay between two adjacent middleboxes are accumulated, shown in Fig. 7. A slope index is computed for each flow. The slope index A_j for flow f_j is defined as $A_j = -\sum_{i=1}^{|M|}(|M| - (2i-1))(p_i^j + d_i)$. The flows are then sequenced in decreasing order of the slope index. The insight is extended from the TSOS algorithm. It is already clear that flows with small processing times on the first middlebox instance and large processing times on the second middlebox instance should be positioned more towards the beginning of the sequence. Similarly, flows with large processing times on the first middlebox instance and small processing times on the second middlebox instance should be positioned more towards the end of the sequence. The slope index is large if the processing times on the downstream middlebox instances are large relative to the processing times on the upstream middlebox instances; the slope index is small if the processing times on the downstream middlebox instances are relatively small in comparison with the processing times on the upstream middlebox instances. We are given the processing times of each flow on all middleboxes in the service chain as well as the transmission delay on each link. We need to decide the processing order of all flows on the service chain. Line 1 handles the link transmission delay as part of the processing time on the middlebox before that link. Line 2 sorts flows in the decreasing order of $A_j = -\sum_{i=1}^{|M|}(|M|-(2i-1))(p_i^j+d_i)$. The order of flows is returned in line 4. As the sorting of flows costs $|F|\log|F|$, the time complexity of SHA is also $O(|F|\log|F|)$.

For better understanding, we use a service chain with four middleboxes as an example, shown in Fig. 7. The transmission delays are $d_1 = 2, d_2 = 3$, and $d_3 = 1$. There are four flows, whose processing times are shown in Tab. 2. By applying the SHA algorithm, we calculate each new processing time value such as $p_1^1 = p_1^1 + d_1 = 1 + 2 = 3$. Then we calculate the slope index for each flow such as $A_1 = -(4-1)\cdot p_1^1 - (4-3)\cdot p_2^1 - (4-5)\cdot p_3^1 - (4-7)\cdot p_4^1 = -3\cdot 3 - 5 + 4 + 3\cdot 4 = 2$. Similarly, we get $A_2 = 0, A_3 = 6$, and $A_4 = -2$. Next we sort the slope indices in decreasing order and return the order as f_4, f_2, f_1, and f_3. The makespan is 23.

5.3 Minimizing the average completion time

In order to minimize the average completion time, we propose a heuristic algorithm, called Pairwise Heuristic Schedule (PHS), in Alg. 4, which is extended from our PS algorithm. We are given the processing time of each flow on each middlebox of the service chain. We need to decide the processing order of all flows on the service chain. The insight of the algorithm is to pair flows with the sum of processing times and transmission delays in each half part of the service chain. In Alg. 4, lines 1-2 define two metrics in order to handle the middlebox processing time and the link transmission delay. Line 3 sorts the flow orders based on the two metrics. Lines 4-5 pair flows. The order of flows is returned in line 6. As the sorting of flows costs $|F|\log|F|$, the time complexity of PHS algorithm is $O(|F|\log|F|)$.

For better understanding, we also use a service chain with four middleboxes as an example, shown in Fig. 7. The settings are the same as the last subsection. By applying the PHS algorithm, we calculate B_1^j and B_2^j for each flow. For example, $B_1^1 = p_1^1 + d_1 + p_2^1 + d_2 = 1+2+2+3 = 8$. Similarly, we get $B_2^1 = 8, B_1^2 = 8, B_2^2 = 8, B_1^3 = 9, B_2^3 = 11, B_1^4 = 11$, and $B_2^4 = 9$. We sort $\max\{B_1^j, B_2^j\}$ in increasing order as f_1, f_2, f_3, and f_4. Next we make f_1 and f_2 as a pair, and f_3 and f_4 as a pair. The scheduling order is f_1, f_2, f_3, and f_4. The average completion time is 16.75.

Algorithm 4 Pairwise Heuristic Schedule (PHS)

In: Flow processing times p_i^j, $\forall f_j \in F, m_i \in M$ and the transmission delay $d_i \in D$;
Out: The flow scheduling order;

1: $B_1^j = \sum_{i=1}^{|M|/2}(p_i^j + d_i)$;
2: $B_2^j \sum_{i=(|M|/2+1)}^{|M|}(p_i^j + d_i)$
3: Sort all flows in increasing order of $\max\{B_1^j, B_2^j\}$;
4: **for** each subset of flows with the same $\max\{B_1^j, B_2^j\}$ **do**
5: Reorder flows by iteratively taking out a pair of flows of $\arg\max(B_2^j - B_1^j)$ and $\arg\max(B_1^j - B_2^j)$;
6: **return** The flow scheduling order.

6 Testbed Evaluation

We use a combination of experiments and simulations for performance evaluation. We have built a prototype to validate the solutions in realistic environments, and have also conducted simulations to obtain performance data in large scale networks. We present experiment results and extensive simulation to show the effectiveness of our design.

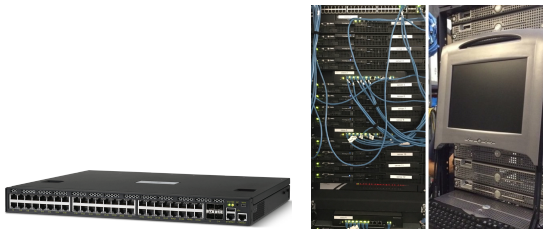
6.1 Settings

We do realistic transmission experiments on the testbed in our lab, whose photos are shown in Figure 8. The testbed contains two Cisco switches (8 ports) and three Pica8 P-3297 switches (48 ports). Each Pica switch connects with two 64 bits Dell Power Edge R210 servers. Each server has 2.4 GHz CPU and 4 GB memory and is accessible via the connections offered by the switches. $Grnlntrn$ is the controller, which is constructed by a Dell Power Edge R210 server and runs with the open-source SDN controller Ryu [41]. The capacities of all physical links in our testbed are 1 Gbps.

We pick four network services for the tested service chain: (1) Load balancer: acts as a reverse proxy and distributes network or application traffic across a number of servers. We utilize the virtual port transmission function of the Pica8 SDN switch. (2) Firewall: monitors incoming and outgoing network traffic and decides whether to allow or block specific traffic based on a defined set of security rules. We utilize the filter configuration function of Pica8 SDN switch. (3) Intrusion Detection System (IDS): monitors a network or system for malicious activity or policy violations. We utilize the filter configuration function of the Pica8 SDN switch. (4) Network address translation: remaps one IP address space into another by modifying network address information in the IP header of packets while they are in transit across a traffic routing device. We utilize the routing configuration of the Pica8 SDN switch. All Pica8 switches run in Open vSwitch (OVS) Mode. Due to space limitation, the detailed description is omitted. We change the number of flows to be served by the same service chain from 2 to 12 with an increment interval of 2. The flow size is randomly generated between 100 Mb and 200 Mb.

6.2 Comparison algorithms and metrics

Based on our knowledge, the model of our work has not been studied before. As a result, the following three algorithms are used for comparison:



(a) Pica8 P-3297. (b) Rack.

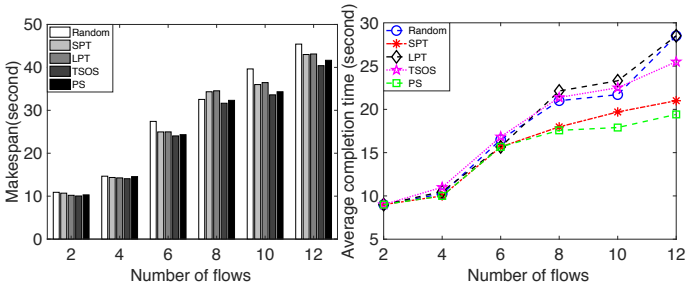
Fig. 8: The views of the testbed.

1) Random: ranks flows randomly.
2) Shortest Processing Time first (SPT): ranks flows by their sum of processing times on each middlebox in an increasing order.
3) Longest Processing Time first (LPT): ranks flows by their sum of processing times on each middlebox in a decreasing order.

In addition, experiments present Algorithms 1 to 4 as TSOS, PS, SHA, and PHS, respectively. Two metrics are used for comparison. The first metric is the *makespan*, which is the interval between the starting time of the first flow on the first middlebox and the finishing time of the last flow on the last middlebox of the service chain. The second metric is the *average completion time*, which is the sum of each flow's completion time divided by the number of flows. The completion time of each flow is the interval between its starting time on the first middlebox of the service chain and its ending time on the last middlebox. The times are measured as the transmission delay of each flow, which is calculated as the difference between the recorded completion and starting time.

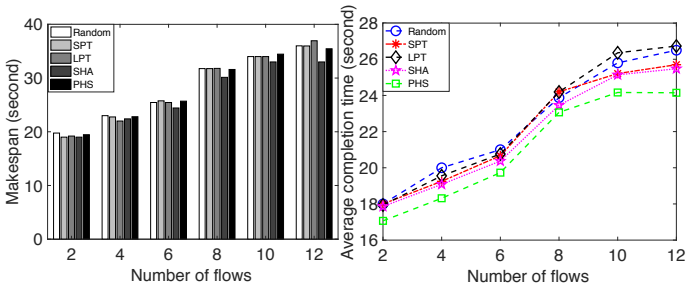
6.3 Results

We show the results of makespan and the average completion time in the service chain with only two middleboxes in Fig. 9. Specifically, Fig. 9(a) uses the bar plot to indicate the makespans of flows by applying the five algorithms, while Fig. 9(b) shows the corresponding average completion times. The increasing tendency of all lines in both figures is similar, which indicates the correctness of our results since the average completion time demonstrates the largest flow completion time (makespan) to some degree. In Fig. 9(a), our TSOS algorithm performs the best with the smallest makespan among the five algorithms. Our proposed PS algorithm for the minimum average completion time objective has a satisfactory result in terms of the makespan objective. Although LPT algorithm is optimal for a single stage flow scheduling [42], its performance is not outstanding, which indicates that simply adding up all processing times as the ordering metric is not enough. Additionally, we find out that the difference among the performances of the five algorithms is not so obvious. It can be explained that the lower bound of the makespan is the sum of all processing times of flows in the second middlebox plus the transmission delay. Moreover, its upper bound is the sum of the bigger processing time of each flow on the two



(a) Makespan. (b) Average completion time.

Fig. 9: A service chain with only two middleboxes.



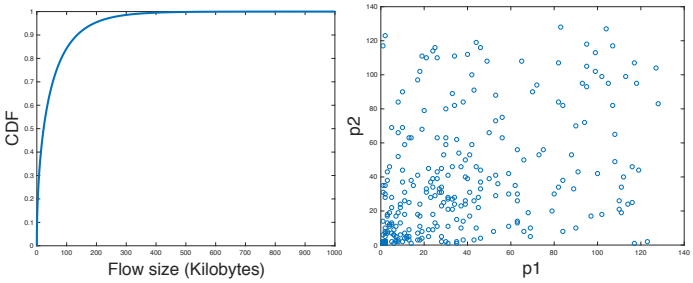
(a) Makespan. (b) Average completion time.

Fig. 10: A service chain with four middleboxes.

middleboxes plus the transmission delay times the number of flows. The difference between lower and upper bounds is not large and is directly proportional to the number of flows. Thus, the performance differences of algorithms are not so obvious.

Fig. 9(b) shows the results of the average completion time in seconds. Though not all flows can be paired, our proposed algorithm PS outperforms others with the minimum average completion time, especially when the number of flows is large. It is worth mentioning that SPT algorithm also has excellent performance. This is because SPT algorithm is optimal for minimizing the average completion time in a single middlebox [43]. Algorithms TSOS and LPT have the largest average completion time since they schedule flows with the longer processing time first. This results in a larger completion time for the majority of flows compared to PS algorithm. When the number of flows is relatively small, the sequence choices are limited, which makes the sequences of different algorithms the same. Even though the realistic transmissions are not completely steady, the results of the five algorithms are quite similar, indicating the stable state of our testbed.

We show the results of makespan and the average completion time in the service chain with six middleboxes in Fig. 10. Though our proposed algorithms SHA and PHS are heuristic, they achieve the best performance in terms of the makespan and the average completion time, respectively. Comparing this with the results with only two middleboxes in the service chain in Fig. 9, we find the basic tendencies of each bar and each line to be similar. In Fig. 10(a), with the same number of flows, the increment of the makespan is not proportional to the number of middleboxes. This is because



(a) Flow size distribution. (b) Processing time distribution.

Fig. 11: An illustration of flow information.

the processing of different flows on different middleboxes can be done at the same time. In terms of the average completion time, Fig. 10(b) shows that PHS algorithm has the lowest average completion time for all situations. When there are 12 flows, its average completion time is 11.7% less than the algorithm LPT.

7 Simulation Evaluation

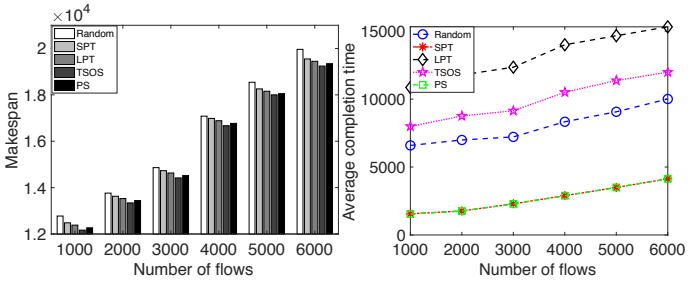
Large-scale simulation experiments are conducted to evaluate the performances of our proposed algorithms. After we present the network and flow settings, the results are shown from different perspectives to provide insightful conclusions. The simulation results show that our proposed optimal and greedy algorithms empirically perform much better than other algorithms.

7.1 Settings

We conduct simulations based on Facebook Data-center Network flow distribution [44], which has been widely used to report the realistic traffic in Facebook's data centers. The flow size distribution is shown in Fig. 11(a) based on the result of [44]. The processing time of each flow on the middleboxes of the service chain is generalized from the size distribution. We conduct the flow scheduling on three service chains. One is with only two middleboxes, another one is with four middleboxes, and the last one is with six middleboxes. We use Fig. 11(b) as an example to illustrate the flow processing time information of the service chain with only two middleboxes. The transmission delays between every two middleboxes are randomly chosen ranging from 1 to 10, which is scaled based on the relationship of the transmission delay and the flow processing time in our experiments. We also measure the makespan as well as the average completion time in the simulations based on the varied number of flows being served in the same service chain. The total number of flows ranges from 1,000 to 6,000 with a stride of 1,000.

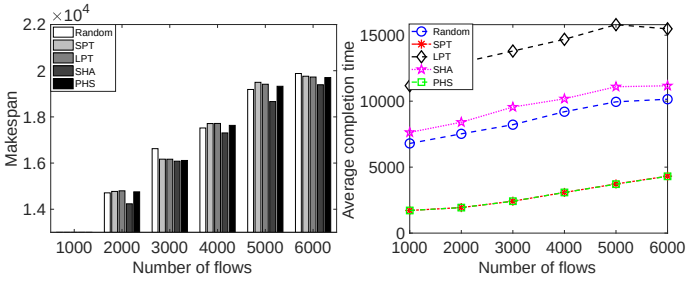
7.2 Results

We show the simulation results of the makespan and the average completion time in the service chain with two middleboxes in Fig. 12. Compared to the experiments in Section VI, the results are similar while the tendency becomes more smooth. Since our proposed algorithm TSOS is optimal, it



(a) Makespan. (b) Average completion time.

Fig. 12: A service chain with only two middleboxes.

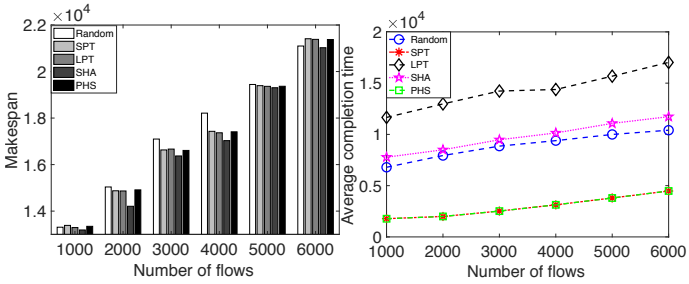


(a) Makespan. (b) Average completion time.

Fig. 13: A service chain with four middleboxes.

achieves the best performance in the makespan, shown in Fig. 12(a). It is worth mentioning that our proposed PS algorithm has the second best performance, which indicates its excellence. In terms of the average completion time, Fig. 12(b) shows that PS algorithm has the lowest average completion time for all situations though the results of the algorithm SPT are similar. This indicates the importance of the total processing times when we schedule flows aiming at minimizing the average completion time. Additionally, the average completion time of PS algorithm is at least 60.1% less than SPT algorithm's when there are at least 1,000 flows.

Figs. 13 show the results for a service chain with four middleboxes. Our proposed algorithm SHA achieves the best performance in the makespan while the PHS algorithm has lowest average completion time. The performance difference among algorithms is not obvious. The tradeoff between the time complexity and the performance makes our proposed algorithms worthy to apply. Fig. 13(a) shows the results of the makespan. When there are 3,000 flows, the increment of the makespan is larger than in Fig. 12(a). Additionally, when the number of flows changes from 5,000 to 6,000, the makespan of TSOS algorithm increases by 18.8% in Fig. 13(a) while the makespan of SHA algorithm in Fig. 12(a) only increases the makespan by 14.5%. In Fig. 13(b), the performance of our proposed algorithm SHA is better than in Fig. 12(b). The average completion times of PHS and SPT algorithms are again quite close to each other. The algorithms LPT and SPT always have the reverse performances when we minimize the makespan and the average completion time. Additionally, the average completion time of PS algorithm is at least 45.1.1% less than SPT algorithm's if the number of flows is no less than 2000. It is worth mentioning that our proposed PS algorithm has the second best performance, which indicates its excellence.



(a) Makespan. (b) Average completion time.

Fig. 14: A service chain with six middleboxes.

The results for when there are six middleboxes are shown in Fig. 14. Though our proposed algorithms SHA and PHS are heuristic, they achieve the best performance in the makespan and the average completion time, respectively. The performance difference among algorithms is not obvious, for reasons similar to that in Section VI(C). However, as the time complexity of all our proposed algorithms is low, it is still reasonable to apply our algorithms. Specifically, Fig. 14(a) shows the results of the makespan. When there are 3,000 flows, the increment of the makespan is larger than in Fig. 12(a). Additionally, when the number of flows changes from 5,000 to 6,000, the makespan of TSOS algorithm increases by 17.5% in Fig. 14(a) while the makespan of SHA algorithm in Fig. 12(a) only increases the makespan by 12.7%. In Fig. 14(b), the performance of our proposed algorithm SHA is better than in Fig. 12(b). It is worth mentioning that algorithm Random is not the worst in all experiments. This is because the two objectives are conflicting with each other; for example, algorithms LPT and SPT always have the reverse performances when we minimize the makespan and the average completion time, respectively. The average completion times of PHS and SPT algorithms are again quite close to each other.

To sum up, our proposed four algorithms perform outstandingly in their corresponding settings. Our simulation results basically have the same tendency as the testbed results, indicating the proper selection of the parameter settings. Small scale experiments in Section 6 verify the correctness of our proposed algorithms while large scale simulations in Section 7 demonstrate their effectiveness. Additionally, we notice that the transmission time of flows in the testbed is not steady enough, resulting in relatively fluctuating performance lines. In Section 7, we conduct simulations with the real flow distribution of Facebook data centers, which makes our results significantly more convincing.

8 Conclusion

Flows always request to be processed by several middleboxes in a specific order, which is known as a service chain. Most researches study the middlebox placement problem and few of them pay attention to the flow scheduling of a deployed service chain, resulting in poor control of the flow completion times. However, the flow completion time is an extreme metric to evaluate the performance of a network.

Therefore, this paper focuses on the service chain scheduling problem. In this paper, we focus on the flow scheduling problem of being served by a service chain in order to improve the quality of service. We aim at minimizing the transmission delay and processing delay in two aspects: makespan and the total completion time. We build a transmission and processing delay model to formulate latency behaviors and control the flow processing sequence in the network. We propose two optimal solutions when there are only two services in the service chain. With a service chain of an arbitrary length, two heuristic algorithms are designed with insights. Extensive experiments as well as large-scale simulations are conducted to evaluate the performance of our algorithms in various scenarios. Experiment and simulation results show that our proposed optimal and greedy algorithms empirically perform much better than other algorithms.

In future, we tend to study the flow completion time among multiple service chains. Additionally, various transmission delay of different flows can also be took consideration in the network model. It is also interesting to formulate the delay behaviour based on a stochastic model.

9 Acknowledgment

This research was supported in part by NSF grants CNS 1824440, CNS 1828363, CNS 1757533, CNS 1629746, CNS 1651947, and CNS 1564128.

References

[1] M. Ghaznavi, E. Jalalpour, B. Wong, and R. Boutaba, "Fault tolerance for service function chains," *arXiv preprint arXiv:2001.03321*, 2020.

[2] P. Quinn and T. Nadeau, "Service function chaining problem statement," *draft-ietf-sfc-problem-statement-07 (work in progress)*, 2014.

[3] S. Kumar, M. Tufail, S. Majee, C. Captari, and S. Homma, "Service function chaining use cases in data centers," *IETF SFC WG*, p. 10, 2015.

[4] W. Haeffner, J. Napper, M. Stiemerling, D. Lopez, and J. Uttaro, "Service function chaining use cases in mobile networks," *Internet Engineering Task Force*, 2015.

[5] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, 2012, pp. 323–336.

[6] A. Bremler-Barr, Y. Harchol, and D. Hay, "Openbox: a software-defined framework for developing, deploying, and managing network functions," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 511–524.

[7] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: Enabling innovation in network function control," in *SIGCOMM 2014*.

[8] J. Sherry and S. Ratnasamy, "A survey of enterprise middlebox deployments," EECS Department, University of California, Berkeley, Tech. Rep., 2012.

[9] D. Zeng, L. Gu, S. Pan, and S. Guo, "Software defined networking ii: Nfv," in *Software Defined Systems*. Springer, 2020, pp. 77–100.

[10] S. Fayazbakhsh, V. Sekar, M. Yu, and J. Mogul, "Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions," in *HotSDN 2013*.

[11] P. P. Zave, R. Ferreira, X. Zou, M. Morimoto, and J. Rexford, "Dynamic service chaining with dysco," in *SIGCOMM 2017*.

[12] L. Qu, C. Assi, and K. Shaban, "Delay-aware scheduling and resource optimization with network function virtualization," *IEEE Transactions on Communications*, vol. 64, no. 9, pp. 3746–3758, 2016.

[13] Z. Zhou, Q. Wu, and X. Chen, "Online orchestration of cross-edge service function chaining for cost-efficient edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1866–1880, 2019.

[14] M. Rost and S. Schmid, "Np-completeness and inapproximability of the virtual network embedding problem and its variants," Technical Report, Tech. Rep., 2018.

[15] "Virtual network embedding approximations: Leveraging randomized rounding," *arXiv preprint arXiv:1803.03622*, 2018.

[16] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *CloudNet 2014*.

[17] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *INFOCOM 2015*.

[18] T. Kuo, B. Liou, K. Lin, and M. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," in *INFOCOM 2016*.

[19] Y. Li, L. T. X. Phan, and B. T. Loo, "Network functions virtualization with soft real-time guarantees," in *INFOCOM 2016*.

[20] A. Gember, A. Krishnamurthy, S. St. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, "Stratos: A network-aware orchestration layer for middleboxes in the cloud," in *arXiv 2013*.

[21] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "Opennf: Enabling innovation in network function control," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 163–174, 2014.

[22] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *USENIX NSDI 2014*, pp. 459–473.

[23] Yao, Hong and Xiong, Muzhou and Li, Hui and Gu, Lin and Zeng, Deze, "Joint optimization of function mapping and preemptive scheduling for service chains in network function virtualization," *Future Generation Computer Systems*, vol. 108, pp. 1112–1118, 2020.

[24] Bhamare, Deval and Samaka, Mohammed and Erbad, Aiman and Jain, Raj and Gupta, Lav and Chan, H Anthony, "Multi-objective scheduling of micro-services for optimal service function chains," in *2017 IEEE international conference on communications (ICC)*. IEEE, 2017, pp. 1–6.

[25] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. M. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Queues don't matter when you can JUMP them!" in *NSDI 2015*.

[26] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized "zero-queue" datacenter network," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, 2014.

[27] K. Jang, J. Sherry, H. Ballani, and T. Moncaster, "Silo: Predictable message latency in the cloud," in *SIGCOMM 2015*.

[28] F. P. Tso, G. Hamilton, R. Weber, C. S. Perkins, and D. P. Pezaros, "Longer is better: Exploiting path diversity in data center networks," in *ICDCS 2013*.

[29] S. Bera, S. Misra, and A. Jamalipour, "Flowstat: Adaptive flow-rule placement for per-flow statistics in sdn," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 530–539, 2019.

[30] Z. Zhou and Q. Wu and X. Chen, "Online Orchestration of

Cross-Edge Service Function Chaining for Cost-Efficient Edge Computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1866–1880, 2019.

[31] X. Wang and C. Wu and F. Le and F. C. M. Lau, "Online Learning-Assisted VNF Service Chain Scaling with Network Uncertainties," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, 2017, pp. 205–213.

[32] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included," *Naval Research Logistics Quarterly*, vol. 1, 1954.

[33] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 2008.

[34] H. Hoogeveen, L. van Norden, and S. van de Velde, "Lower bounds for minimizing total completion time in a two-machine flow shop," *Journal of Scheduling*, vol. 9, no. 6, pp. 559–568, 2006.

[35] F. D. Croce, V. Narayan, and R. Tadei, "The two-machine total completion time flow shop problem," *European Journal of Operational Research*, vol. 90, no. 2, pp. 227 – 237, 1996.

[36] H. Zheng, Z. Wan, and J. Wu, "Optimizing mapreduce framework through joint scheduling of overlapping phases," in *ICCCN 2016*.

[37] P. Duan, Q. Li, Y. Jiang, and S. T. Xia, "Toward latency-aware dynamic middlebox scheduling," in *ICCCN 2015*.

[38] A. Abdou, A. Matrawy, and P. C. van Oorschot, "Taxing the queue: Hindering middleboxes from unauthorized large-scale traffic relaying," *IEEE Communications Letters*, vol. 19, no. 1, pp. 42–45, 2015.

[39] M. Pinedo, *Scheduling - Theory, Algorithms, and Systems*, 2008.

[40] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[41] Ryu component-based software defined networking framework. [Online]. Available: http://osrg.github.io/ryu/

[42] J. Kleinberg and E. Tardos, *Algorithm Design*, Boston, MA, USA, 2005.

[43] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Springer International Publishing, 2016.

[44] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *SIGCOMM 2015*, pp. 123–137.