Minimizing Transmission and Processing Delay in a NFV-based Network

Yang Chen and Jie Wu
Center for Networked Computing, Temple University, USA
Email: {yang.chen, jiewu}@temple.edu

***Abstract*—Software middleboxes, also called Virtual Network Functions (VNFs), are replacing expensive traditional hardwares in implementing network services. Multiple middleboxes processing flows in a specific order form a service chain. Current works mostly focus on the service chain deployment problem and pay little attention to the flow scheduling of a deployed service chain, resulting in a poor control of the flow completion times. However, a high performance network always has a strict requirement of the flow completion time. In this paper, we build a transmission and processing delay model to formulate the communication latency behavior of flows being processed by middleboxes. We aim to minimize the flow completion time in two aspects: the longest completion time (makespan) and the average completion time. We propose an optimal solution for each aspect when there are only two middleboxes in the service chain. With a service chain with an arbitrary length, we first prove the NP-hardness of our problem in both aspects and then design two corresponding heuristic algorithms, which are extended from our proposed optimal solutions for a service chain with a length of two. Extensive simulations are conducted to evaluate the performance of our algorithms in various scenarios.**

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

I. Introduction

Network Function Virtualization (NFV) is changing the way we implement the network functions from expensive hardwares to software middleboxes, called Virtual Network Functions (VNFs). These (software) middleboxes are executed on switch-connected servers. Middleboxes play an increasingly important role in modern networks [1]. As Software Defined Networking (SDN) emerges, so does a tendency to incorporate SDN and NFV in concerted ecosystems [2]. With the intense requirement of high performance networks, SDN manoeuvres through NFV traffic and schedules flows to be processed by middleboxes [3]. However, most current works focus on the VNF deployment and pay little attention to flow scheduling (the order of flows to be processed by a service chain), resulting in a 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 [4]. 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.

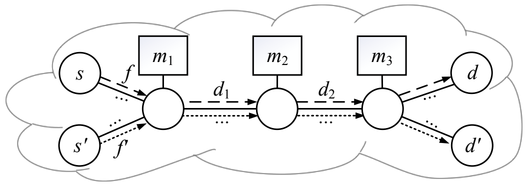


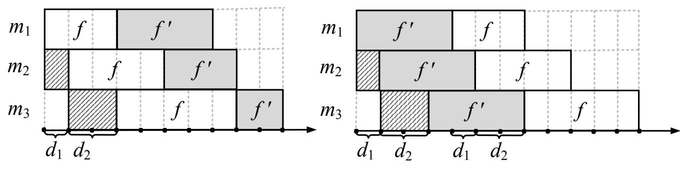
Fig. 1: An illustration example.

Flows \ Middleboxes	m_1	m_2	m_3
f	3	4	5
f'	4	3	2

TABLE I: Processing times.

We illustrate the importance of the flow processing order in an example, shown in Fig. 1. Circles denote 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) that 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 in Tab. I. 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 there is a delay for f' to process 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 its given processing times, because of the transmission delays. Thus, if flow f is scheduled before flow f', the makespan is 10, which is shown in Fig. 2(a). If flow f' is scheduled before flow f, the makespan is 12, which is shown in Fig. 2(b). The difference comes from the constraint on the completion times between adjacent middleboxes in the service chain.

In this paper, we aim at minimizing the total transmission



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

Fig. 2: Different scheduling orders.

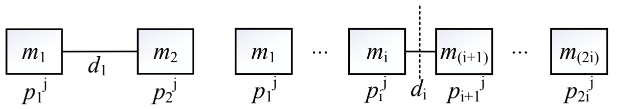
delay and processing delay in two aspects: makespan (the longest completion time) and the average completion time (the total completion time of all flows divided by the number of flows). We build a flow transmission time and processing delay model to formulate the latency behavior and control the flow 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.

The remainder of this paper is organized as follows. Section II surveys related works. Section III describes model and problem formulation. Section IV introduces our optimal algorithms to arrange flows for a service chain with only two middleboxes. In Section V, we handle cases with an arbitrary number of middleboxes in a service chain and propose two heuristic algorithms with insights. Section VI includes simulations. Section VII concludes the paper.

II. Related Work

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 applications. Existing techniques to combat the queuing problem include prioritizing flows such that packets from latency-sensitive flows can "jump" the queue [5], centrally scheduling all flows for every server so that no flow has to queue [6], and pacing end host packets to achieve a guaranteed bandwidth for guaranteed queuing [7]. These techniques assume that the shortest path forwarding is used. Data center fabrics have rich path-redundancy, so non-shortest paths can be exploited to use path redundancy and spare capacity for mitigating network congestion [8]. As policy chaining rules 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. This is possible because propagation delay on physical links is predictable and smaller than queueing delays.

A classic problem, flow shop [9], inspires our work. Flow shop 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. [10] 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.



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

Fig. 3: A service chain.

Minimizing the average completion time with two phases is already strongly NP-hard. Zheng et al. in [11] 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.

III. Model and Formulation

A. 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^j. 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 [12], in which the processing time of each flow f is a constant.

To analyze processing behaviors of flows in middleboxes, a queuing model is employed in [13], 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: the deterministic model and the exponential model [12]. In the deterministic model, the processing time is denoted by p^j. In the exponential model, it follows an exponential distribution with a rate λ^j [14]. The expected processing time is $1/\lambda^j$.

B. Problem Formulation

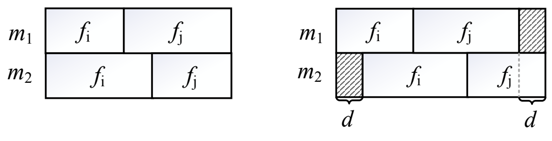
In this paper, we schedule flows to be processed by a service chain. We have two different objectives: (1) minimizing the makespan; and (2) minimizing the average completion time. The definitions are as follows:

***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.

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.

***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:

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

IV. A SERVICE CHAIN WITH TWO MIDDLEBOXES

In this section, we study the case of the service chain that only includes two middleboxes 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.

A. Minimizing the makespan

To minimize the makespan, we propose an optimal algorithm, called the 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:* The 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 the least, all flows need to be processed i order by both middleboxes 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 the second middlebox is kept 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. ■

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.
```

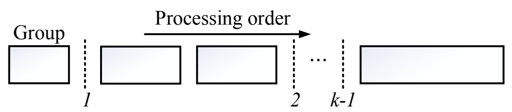


Fig. 5: Multiple middleboxes.

B. Minimizing the average completion time

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

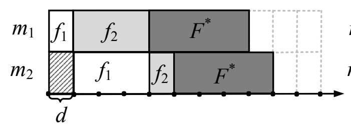
***Definition** 3:* f_i and f_j are a pair, if $p_1^i = p_2^j$ and $p_2^i = p_1^j$. The definition of pair comes from [11], which is for MapReduce optimization [15]. The pair in [11] is shown in 4(a), in which there is no delay between the starting time of one flow in two stages. However, there is a delay between two middleboxes in a service chain. In this paper, the pair with a delay is shown in 4(b); the delay is between the starting time of one flow in two stages. In order to minimize the average completion time, we propose an optimal algorithm, called the 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 the 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 executed pairwise in its optimal schedule. For each pair, the flow with $p_2^j > p_1^j$ is executed before the flow with $p_2^j < p_1^j$.

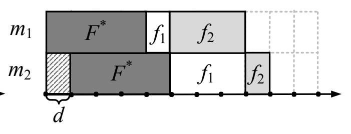
Proof: We prove by induction. Let us start with a basic case, where F only includes two flows that can form a pair (denoted as f_1 and f_2). Suppose f_1 has $p_1^1 < p_2^1$ and f_2 has $p_1^2 > p_2^2$. We have two schedules: schedule one executes f_1 before f_2, and schedule two executes f_2 before f_1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

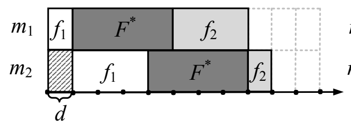
We have $p_2^1 = p_1^2$ according to the definition of pair. Since f_1 is $p_1^1 < p_2^1$, we have $p_1^1 + p_1^2 < p_2^1 + p_2^2$. Hence, schedule one has a smaller average flow makespan by executing the flow with $p_1^j < p_2^j$ before the flow with $p_1^k > p_2^k$. For induction, let us consider an existing schedule of S. It executes flows



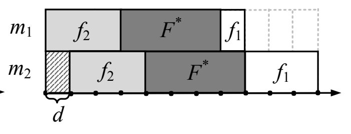
(a) Sequence 1.



(b) Sequence 2.



(c) Sequence 3.



(d) Sequence 4.

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

that can form a pair. Let F^* denote a subset of F that is consecutively executed pairwise in S. Let τ denote the average flow makespan of F^*; however, it is 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 exists 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. Suppose f_1, f_2, and F^* are scheduled at time t_0 then their 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 under-utilization of middlebox m_1. Meanwhile, the average flow completion time of S_1, S_2, and S_3 are shown as follows:

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

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: $\frac{|F^*|\cdot(p_2^1+p_2^2)+2t^*}{2} \le \frac{|F^*|\cdot 2p_2^1+2t^*}{2} = |F^*| \cdot p_2^1 + t^*$. The mean of two unequal numbers is always larger than the smaller of these two numbers. Therefore, we have: $\min\{|F^*|\cdot(p_2^1+p_2^2), 2t^*\} \le |F^*| \cdot p_2^1 + t^*$. 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, 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 the 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 a shorter processing time p_1^j should be executed earlier to minimize the average flow makespan, since these 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 a 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 [11], when F can be decomposed to pairs of flows, flows that can form a pair are executed pairwise 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 these 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 orders always leads to a smaller average flow makespan. Hence, Alg. 2 is optimal when F can be decomposed to pairs of flows or when all flows are simultaneously $p_2^j < p_1^j$ (or $p_2^j > p_1^j$ or $p_2^j = p_1^j$).

A potential problem in 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 let the first and second processing times on the two middleboxes of each flow be 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.

V. A SERVICE CHAIN WITH MULTIPLE MIDDLEBOXES

In this section, we study the case of the service chain including 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.

A. Minimizing the makespan

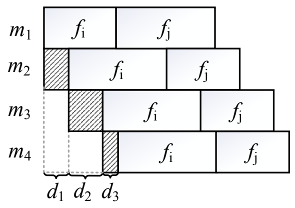


Fig. 7: Multiple middleboxes in a service chain.

TABLE II: 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

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.
```

For more than two middleboxes in the chain, it is 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 their slope indices. 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 downstream middlebox instances are relatively small compared to those 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 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 $d_3 = 1$, respectively. There are four flows, whose processing times are shown in Tab. II. 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.

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.
```

B. Minimizing the average completion time

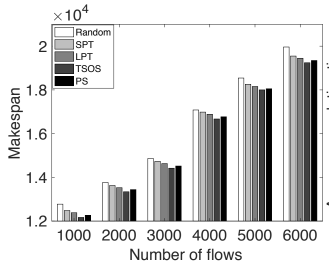
In order to minimize the average completion time, we propose a heuristic algorithm, called the Pairwise Heuristic Schedule (PHS), in Alg. 4, which is extended from our PS algorithm. We are given the processing time of each flow on each middlebox of the service chain. We need to decide the processing 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 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 a pair, and f_3 and f_4 a pair. The scheduling order is f_1, f_2, f_3, and f_4. The average completion time is 16.75.

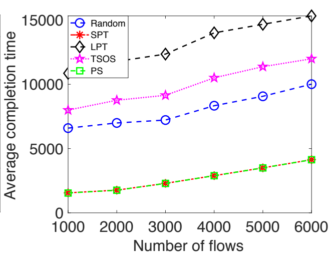
VI. Simulation evaluation

A. Settings

We conduct simulations based on Facebook Data-center Network flow distribution [16], which has been widely used to report the realistic traffic in Facebook's data centers. We conduct the flow scheduling on two service chains. One is with only two middleboxes and the other one is with six 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

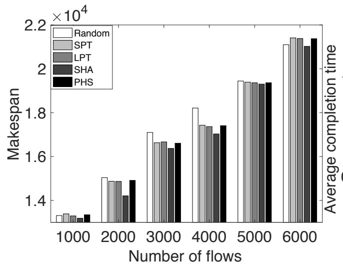


(a) Makespan.

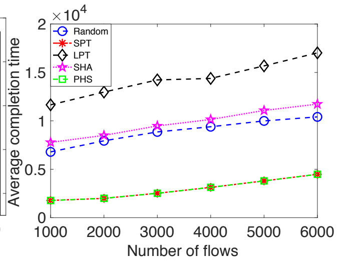


(b) Average completion time.

Fig. 8: A service chain with only two middleboxes.



(a) Makespan.



(b) Average completion time.

Fig. 9: A service chain with six middleboxes.

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.

B. Results

We show the results of the service chain with two middleboxes in Fig. 8. Since the TSOS algorithm is optimal, it achieves the best performance in the makespan, shown in Fig. 8(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. 8(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 time when we schedule flows that aim to minimize the average completion time. Additionally, the average completion time of the PS algorithm is at least 60.1% less than SPT algorithm's when there are at least 1,000 flows.

The results for when there are six middleboxes are shown in Fig. 9. The performance difference among algorithms is not obvious. However, as the time complexity of all our proposed algorithms is low, it is still reasonable to apply our algorithms. Specifically, Fig. 9(a) shows the results of the makespan. When there are 3,000 flows, the increment of the makespan is larger than in Fig. 8(a). Additionally, when the number of flows changes from 5,000 to 6,000, the makespan of the TSOS algorithm increases by 17.5% in Fig. 9(a) while the makespan of SHA algorithm in Fig. 8(a) only increases its makespan by 12.7%. In Fig. 9(b), the performance of our proposed algorithm SHA is better than in Fig. 8(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, LPT and SPT always have reverse performances when we minimize the makespan and the average completion time. The average completion times of PHS and SPT are quite close.

VII. Conclusion

We study the flow scheduling problem of being served by a service chain in order to improve the quality of service. We aim to minimize 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.

VIII. Acknowledgment

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

References

[1] Y. Chen and J. Wu, "Nfv middlebox placement with balanced set-up cost and bandwidth consumption," in *ICPP 2018*.

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

[3] Y. Chen, J. Wu, and B. Ji, "Virtual network function deployment in tree-structured networks," in *ICNP 2018*.

[4] 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.

[5] 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*.

[6] 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.

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

[8] 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*.

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

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

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

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

[13] 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.

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

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

[16] 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.