# Minimizing Online Service Chain Scheduling Cost on Multi-Core Servers

Sihui Xu and Xiangyu Lin
Shanghai Jiao Tong University

**ABSTRACT:** Our project is in the field of Software Defined network (SDN). What we mainly focus on is Network Function Virtualization (NFV). NFV is a novel paradigm that enables flexible and scalable implementation of network services on cloud infrastructure. A key factor in the success of NFV is the ability to dynamically allocate physical resources according to the demand. This is particularly important when dealing with the data plane since additional resources are required in order to support the virtual switching of th packets between the Virtual Network Functions (VNFs). The exact amount of these resources depends on the way service chains are deployed and the amount of network traffic being handled. Thus, orchestrating service chains that require high traffic throughput is a very complex task. Prior art on the service chain deployment in a multi-core system can be impractical due to high overhead and exponential complexity. A central challenge for this problem is how to dynamically determine which service chain needs to be deployed onto which server and which VNF needs to be deployed onto which CPU core in an effective manner.

In this project, we propose a two-stage optimization framework to solve this problem. Our major contribution is the solution to the first stage, which aims to minimize the total number of consumed CPU cores. We then prove its NP-hardness. Based on the proof, we propose an near-optimal approximation algorithm (SA) and compare it with another approximation algorithm we designed (Greedy) to demonstrate its superiority.

For the completeness of the whole project, we also state the second stage. This stage aims to minimize the overall cost on each server subject to the CPU processing capacity constraints. We propose an online algorithm in this stage, which is our minor contribution to this project.

## 1. INTRODUCTION

**Background and Motivation:** Network Function Virtualization (NFV) can accelerate network innovation by reducing capital expenses and shortening the renewal cycles. It has the ability to dynamically allocate physical resources according to the demand. By implementing NFV on servers, massive network middle-boxes and network operators are able to provide network functions with much more efficient utilization of dedicate hardware resources.

A traditional network function can be replaced by a software service chain implemented on commodity servers with multiple CPU cores. A software service chain usually consists of a set of virtual network functions (VNFs).

Regulations and Internet protocols for NFV, such as OpenFlow in software defined networks (SDN), introduce the rules to dynamically process NFV service chains on servers.

Here are some deployment strategies to allocate VNF service chains onto servers.

One simple way of placement is the gather placement, which is demonstrated in fig.1. It allocates all the VNFs of one service chain onto one single server. Note that if we use the gather placement, there must be at least one server that can host all VNFs in one NFV service chain in order for the deployment to be feasible. Therefore, there could be situations where gather placement is not feasible.

The second strategy is the distribute placement, which is demonstrated in fig.2. In this strategy, no two VNFs belong to on NFV service chain can be allocated onto the same server.Note that given a service chain, the number of the servers must be larger than or equal to the number of VNFs in the NFV service chain in order for the deployment to be feasible. Therefore, there also could be situations where distribute placement is not feasible.

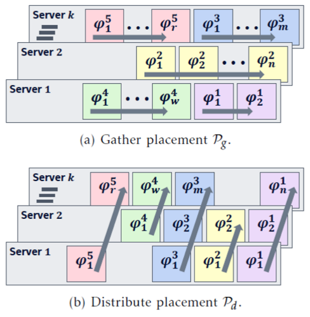

Server $k$
$\varphi_1^5$ ... $\varphi_r^5$ $\varphi_1^3$ ... $\varphi_m^3$
Server 2
$\varphi_1^2$ $\varphi_2^2$ ... $\varphi_n^2$
Server 1
$\varphi_1^4$ ... $\varphi_w^4$ $\varphi_1^1$ $\varphi_2^1$
(a) Gather placement $\mathcal{P}_g$.
Server $k$
$\varphi_r^5$ $\varphi_w^4$ $\varphi_m^3$ $\varphi_n^1$
Server 2
$\varphi_1^4$ $\varphi_2^3$ $\varphi_2^2$ $\varphi_2^1$
Server 1
$\varphi_1^5$ $\varphi_1^3$ $\varphi_1^2$ $\varphi_1^1$
(b) Distribute placement $\mathcal{P}_d$.

Fig. 1: Gather Placement and Distribute Placement

In this project, we adopt a more complex and flexible deployment strategy, the arbitrary placement [1],[2]. It is a combination of the gather placement and the distribute placement. We decompose a NFV service chain into sub-chains and deploy each sub-chain on a physical server.

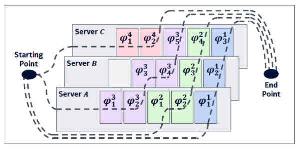


Fig. 2: Arbitrary Placement

However, the algorithm based on partition will result in an exponential runtime complexity. The exponential runtime complexity will be quite inefficient when confronting with large-scale inputs. In addition, [1] assumes that different sub-chains are not allowed to be processed on the same server, which is fairly inappropriate since the CPU process is able to migrate between servers in the same cluster freely.

In this project, we study a two-stage optimization framework of online NFV service chain scheduling problem on multi-core servers. Given a set of NFV service chains which come in an online manner, we schedule the VNFs and the flow demands in two stages. In the first stage, we divide the coming NFV service chains into sub-chains. Then we process them on different servers to confirm the minimal number of consumed CPU cores. In the second stage, the flow demands come from the solution of the first stage in an online manner. We dynamically steer and schedule the demands for different CPU cores in a server to minimize the overall cost, which involves provision cost and switching cost.

**Our Contributions:** We make two contributions in this project. First, we propose a two-stage optimization framework for NFV chain scheduling problem. In the first stage, we aim to minimize the number of occupied CPU cores when the service chains are divided and deployed on different servers. The second stage is dynamic adjustment on CPU cores, aiming at minimizing the overall cost when the demands for VNFs are scheduled on different CPU cores on a certain server.

Our second contribution is the solution to the first stage and the second stage. We prove the NP-hardness of the first stage and we design several approximation algorithms and evaluate their performance. Our algorithms run much faster than the algorithm proposed in [1]. What's more, we have also found an approximation algorithm with near-optimal performance in terms of the minimal cost. Furthermore, inspired by [3], we propose an online algorithm based on affine transformation to dynamically steer and schedule the VNF demands on CPU cores.

## 2. RELATED WORK

Due to the page-limitation, we will review the previous art on the NFV service chain scheduling problem briefly.
Ghaznavi et al. [4] present the concept of operational cost when VNFs are deployed on different servers and provide a method for flexible services. To observe the order in an NFV service chain, Ma et al. [5] design a heuristic algorithm to deploy interdependent VNFs, where the relations among these VNFs can be captured by a partially- or totally-ordered set. Breaking free from that one VNF processes all of the flows, Sang et al. [6] admits that one VNF can process a fraction of one flow while the others can process the rest, whose objective is minimizing the number of running VNF instances.

Recently in the research area for multi-core, NFVnice [7] deploys the VNFs of a service chain onto one network function platform with multiple CPU cores, where each VNF is deployed onto one CPU core and different VNFs can share the same CPU core. Besides, Luizelli et al. [1][2] implement to get VNF virtual switching cost and propose an arbitrary placement for NFV service chains, which is a combination of gather placement and distribute placement. On NFV orchestration in an online situation, Wang et al. [8] study online NFV orchestration and scaling over data centers and provide a learning approach to solve the problem. Shi et al. [3] put forward a computationally tractable framework for convex optimization with switching costs and ramp constraints and give an implementation on VNF orchestration problems.

## 3. PROBLEM FORMULATION

In this project, we propose an online two-stage optimization framework to solve the NFV chain scheduling problem.

### 3.1 Network Model

**Network:** In this section, we first introduce the network model. We use a directed graph $G = (V, E)$ to present the network, while $V$ represents the nodes in the network and $E$ represents the edges between the nodes in the network.
**Servers:** We use $S$ to represent the set of multi-core severs in the network, where $S \subseteq V$. Suppose there are totally M servers. For each server $k \in S$ $(1 \leq k \leq M)$ , we use $S_k$ to denote its number of CPU cores for service chain processing.

**NFV Service Chains:** Then we present our model for the NFV service chains. We use $\Phi$ to denote the set of the NFV service chains. Suppose there are totally N NFV service chains. Let $\varphi^i$ denote the $i$th NFV service chain, where $1 \leq i \leq N$. Then the set of NFV service chains can be formulated as:

$$\Phi = \{\varphi^1, \varphi^2, ..., \varphi^N\}$$

**VNFs:** An NFV service chain contains several virtual network functions (VNFs). The VNFs are ready to be processed on the servers in the server set $S$. Suppose there are $n_i$ different VNFs for each service chain $\varphi^i$. We use $\varphi^i_j$ to represent the $j$th VNF in NFV service chain $\varphi_i$, where $1 \leq j \leq n_i$. Then the set of VNFs in service chain $\varphi^i$ can be defined as:

$$\varphi^i = \{\varphi^i_1, \varphi^i_2, ..., \varphi^i_j, ...\varphi^i_{n_i}\} \tag{1}$$

**Flow Demand:** Then we are going to present the flow demand for each VNF service chain. We denote the number of packets per second for service chain $\varphi_i$ as $p_i$.
**Computing Resource:** Since different kind of VNFs calls for different amount of computing resource, we use $c^i_j$ to denote the number of consumed CPU cycles to process one packet. If VNF $\varphi^i_j$ is deployed on a CPU core, the consumed CPU cycles per second can be defined as:

$$T^i_j = c^i_j \times p_i$$

**Time Slots:** In this problem, the VNF service chains will come in an online manner. We can divide the working time for servers into various time slots. In each time slot, the NFV service chains arrive according to a sequence $t = \{1, 2, ..., N\}$. At each time t, one NFV service chain $\varphi_t$ comes from a client and the demand should receive response immediately at time t. Therefore, when t reaches N, there have been N NFV service chains in this time slot. And this time slot ends, then a new time slot starts.

### 3.2 Problem Definition

**Definition 1. the Two-stage NFV Service Chains Scheduling Problem:** We have a time slot $t = 1, 2, ..., N$, a set of k servers: $S = \{S_1, S_2, ...S_k\}$, and a sequence of service chains: $\Phi = \{\varphi^1, \varphi^2...\varphi^N\}$ which comes in an online manner. Each service chain is a set of VNFs: $\varphi^i = \{\varphi^i_1, \varphi^i_2, \varphi^1_3, ...\varphi^i_{n_i}\}$. We formulate the problem of the VNF service chains scheduling problem as a two-stage optimization problem. **First Stage:** In the first stage, we allocate the service chains onto the servers. This stage aims to minimize the number of occupied CPU cores when the VNF service chains are divided into several parts and each part is deployed on different servers. **Second Stage:** In the second stage, for each server, we dynamically adjust the demands for VNFs on its CPU cores. This stage aims to minimize the total operational cost.

### 3.3 Problem Formulation for the First Stage

Here are our problem formulation for the first stage. In this stage, we allocate NFV service chains to different servers to minimize the total number of occupied CPU cores.
**Sub-chains:** According to Eqn.(1), the NFV service chain comes at time $t$ can be denote as $\varphi^t = \{\varphi^t_1, \varphi^t_2, ..., \varphi^t_{n_t}, \}$. We divide the NFV service chain into $\Omega$ $(1 \leq \Omega \leq N)$VNF sub-chains. We use $\psi^t$ to present the set of sub-chains of $\varphi^t$:

$$\psi^t = \{\psi^t_{s_1 \to d_1}, \psi^t_{s_2 \to d_2}, ..., \psi^t_{s_w \to d_w}, ..., \psi^t_{s_\Omega \to d_\Omega}\}$$

where $\psi^t_{s_\omega \to d_\omega}$ denotes the $w$th sub-chain

$$\psi^t_{s_\omega \to d_\omega} = \{\varphi^t_{s_w}, \varphi^t_{s_w+1}, ..., \varphi^t_{d_w-1}, \varphi^t_{d_w}\}$$

where $s_w$ denotes the start of the sub-chain $\psi^t_{s_w \to d_w}$ and $d_w$ denotes the end of the sub-chain $\psi^t_{s_w \to d_w}$. Besides, to maintain the order of VNF sub-chains, we must follow the constraint:

$$s_{w+1} = d_w + 1, \quad \forall \quad 1 \le w \le \Omega - 1$$

Then we map each sub-chain to a server. We assume that each VNF can only be routed to one server. Therefore, the set of the sub-chains should be disjoint. Then we have:

$$\bigcup_{\omega \in [1,\Omega]} \Psi^t_{s_\omega \to d_\omega} = \boldsymbol{\varphi^t}$$

$$\bigcap_{\omega \in [1,\Omega]} \Psi^t_{s_\omega \to d_\omega} = \emptyset$$

**Allocation Function:** Then we will allocate the sub-chains to the set of servers $S$. We define the allocation function $\mathcal{F} : \psi^t \to S$. If a sub-chain $\psi^t_{s_w \to d_w}$ is allocated to server $k$, we can have $\mathcal{F}(\psi^t_{s_w \to d_w}) = k$. When sub-chain $\psi^t_{s_w \to d_w}$ is allocated to server $k$, $\psi^t_{s_{w+1} \to d_{w+1}}$ will not be allocate to the same server $k$, i.e., if $\mathcal{F}(\psi^t_{s_w \to d_w}) = k$ and $\mathcal{F}(\psi^t_{s_{w+1} \to d_{w+1}}) = k$, we will integrate the two sub-chains into one single sub-chain,i.e.,$\mathcal{F}(\psi^t_{s_w \to d_{w+1}}) = k$. To maintain the minimal cost policy, we had better allocate discontinuous sub-chains on one server.
**Cost of occupied CPU cores:** If we have $\mathcal{F}(\psi^t_{s_w \to d_w}) = k$, let $Q$ to represent the cost of occupied CPU cores. According to [1], the total cost $Q$ consists of $Q_v$ and $Q_h$. $Q_v$ denotes the required cost to operate VNFs. $Q_h$ denotes the required cost to operate the network traffic switching. According to [2], the cost $Q_h$ is defined as:

$$\log Q_h(\psi^t_{s_w \to d_w}, k) = \alpha_k \times \log(|d_w - s_w|) + \beta_k \times \log(p_t) + \gamma_k \tag{2}$$

where $p_t$ is the number of packets per second for service chain $\varphi^t$ and the parameter $(\alpha_k, \beta_k, \gamma_k)$ only depends on the property of server $k$ ($\alpha \in [0, 1]$).

We use $q^t_j(k)$ to denote the number of consumed CPU cores when VNF $j$ is allocated to server $k$. Then cost $Q_v$ is defined as:

$$Q_v(\psi^t_{s_\omega \to d_\omega}, k) = \sum_{j=s_\omega}^{d_\omega} q^t_j(k) \tag{3}$$

The total number of consumed CPU cores is:

$$Q_(\psi^t_{s_\omega \to d_\omega}, k) = Q_v(\psi^t_{s_\omega \to d_\omega}, k) + Q_h(\psi^t_{s_\omega \to d_\omega}, k) \tag{4}$$

where $Q_v(\psi^t_{s_\omega \to d_\omega}, k)$ and $Q_h(\psi^t_{s_\omega \to d_\omega}, k)$ is given by Eqn.(3) and Eqn. (4).
**Problem Formulation:** Now we introduce the following problem formulation for the first stage.

$$\min_{\Omega, (s_\omega, d_\omega), \mathcal{F}} \quad \sum_{k \in S} Q(\Psi^t_{s_\omega \to d_\omega}, k) \tag{5}$$

## 3.4 Problem Formulation for the Second Stage

In the second stage, for each VNF demand coming from the solutions of the first stage, we assign them on different CPU cores and dynamically adjust the processing flow rate on each core to confirm the minimal total cost during the time slot $t = \{1, 2, \cdots, T\}$ on each server.

Here we assume that there are totally $m$ kinds of VNFs and renumber them as $\{\varphi_1, \varphi_2, ..., \varphi_r, ..., \varphi_m\}$.
**Time Set:** We define a time set $T^k_r$ containing the time $t$s in which $\varphi_r$ is allocated to server $k$ in the first stage.

$$T^k_r(t) = \{t' \mid \mathcal{F}(\Psi^{t'}_{s_\omega \to d_\omega}) = k \; and \; \varphi_r \in \Psi^{t'}_{s_\omega \to d_\omega} \; and \; t' \le t\} \tag{6}$$

**Flow Rate:** We use $f^k_r(t)$ to denote the total flow rate (which means the total number of packets per second) for VNF $\varphi_r$ on server $k$, which accumulates as time $t$ flows by.Then $f^k_r(t)$ can be formulated as $f^k_r(t) = \sum_{t \in T^k_r(t)} p_t$
**Flow Demand:** For a simple CPU core $l$ ($1 \le l \le S_k$), we use $C_l$ to denote the number of cycles per second when the utility of this CPU core reaches maximum. We assign each demand $f^k_r(t)$ for VNF $\varphi_r$ on different CPU cores and adjust the amount of processing need on each core dynamically as time $t$ goes by. We use $x_{r,l}(t)$ to represent the amount of flow demand of VNF $\varphi_r$ which is processed on core $l$. To satisfy the demand, we have

$$\sum_{l=1}^{k} x_{r,l}(t) \ge f^k_r(t) \quad \forall r \in [1, m], \; \forall t \in [1, T] \tag{7}$$

Besides, we use $c_{r,l}$ to denote the consumed CPU cycles for processing one packet if VNF $\varphi_r$ is assigned to $l$. In order not to overload the make CPU cores, we have

$$\sum_{r=1}^{m} c_{r.l} \cdot x_{r,l}(t) \le C_l, \quad \forall l \in [1, S_k], \; \forall t \in [1, T] \tag{8}$$

**Provision Cost:** When the flow demand $x_{r,l}(t)$ are assigned to a CPU core, it will incur a provision cost. We assume that the server will incur a cost of $\pi_{r,l}$ when a unit of flow demand of $\varphi_r$ are assigned to server $l$. Then the VNF provision cost $W^k_p$ follows

$$W^k_p = \sum_{t=1}^{T} \sum_{r=1}^{m} \sum_{l=1}^{S_k} \pi_{r,l} \cdot x_{r,l}(t) \tag{9}$$

**Switching cost:** We have to endure a cost for the network rerouting when $x_{r,l}(t)$ changes by time $t$. Similar to [**?**], we use linear switching cost $W^k_s$ to describe the cost for workload balancing and network rerouting. Then we have

$$W^k_s = \sum_{t=1}^{T} \sum_{r=1}^{m} \sum_{l=1}^{S_k} \beta_{r,l} |x_{r,l}(t) - x_{r,l}(t-1)| \tag{10}$$

where $\beta_{r,l}$ denotes the switching cost for unit amount of change on $x_{r,l}$.
**Ramp Constraint:** We also introduce the Ramp Constraint in [3] to limit the amount of network rerouting:

$$|x_{r,l}(t) - x_{r,l}(t-1)| \Delta x, \quad \forall \, t, r, l \tag{11}$$

**Problem Formulation:**

$$\min \quad W_p^k + W_S^k, \quad \forall\, k \in [1, S_k] s.t.(9)(10)(13) \tag{12}$$

## 4. PROPOSED METHODS

### 4.1 Complexity Analysis for the first stage

**Theorem 1.** The feasibility of first stage is NP-hard.
*Proof*: First, we list out all possible sub-chains for service chain $\varphi^t$. We use P to denote the set of all possible sub-chains $P = \{\psi^t_{s_w \to d_w} | \psi^t_{s_w \to d_w} \subseteq \varphi^t\}$. Then we construct the set of subsets for a set cover problem. The set of servers is $S = \{1, 2, ..., k, ..., M\}$. We define a new set $Z$ which describes all possibilities of sub-chains deployment using set multiplication. $Z = P \times S = \{z | \forall \;\; \psi^t_{s_w \to d_w} \in P, \forall \;\; k \in S\}$, where $z$ is also a set: $z = (\psi^t_{s_w \to d_w}, k) = \{\varphi^t_{s_w}, \varphi^t_{s_{w+1}}, ..., \varphi^t_{d_w}, k\}$. We define a weight $w_z$ for each element $z \in Z$, then we have $w_z = Q_h + Q_v$. Since the number of consumed CPU cores can not exceed the total number of cores $S_k$, we set $w_z = \infty$ if $Q_h + Q_v > S_k$. Then we use the subsets $z$ in set $Z$ to cover the VNF set $\varphi^t$. In this way, we can reduce the first stage problem to the minimal weight set cover problem. Since the minimal weight set cover problem is NP-hard, the first stage problem is thus NP-hard.

### 4.2 Solution to the first stage

**Solution 1 - Greedy Algorithm:** It is intuitive for us to design the first algorithm in the greedy manner. In every iteration $h$, we choose the sub-chain z with the minimal $f(z)$, which is defined as $f_h(z) = w_z / N_h(z)$, where $N_h(z)$ is the number of uncovered elements in subset $z$ after $h-1$ iterations. If VNF $\varphi^t_j$ is covered by both $z_1$ and $z_2$ we use $z_1'$ to replace $z_1$, where $z_1' = z_1 - \{\varphi^t_j\}$. However, the performance of the greedy algorithm cannot live up to our expectation. Therefore, we design another algorithm, the simulated annealing algorithm to further improve the performance.

**Solution 2 - Simulated Annealing Algorithm:** We then try simulated-annealing for solving partition problems in our first stage. We use the partition of VNF set as the state to work on, note that the sub-chains have no overlaps and the sub-chains cover the whole chain:

$$\psi^t = \{\psi^t_{1 \to s_1}, \psi^t_{s_1+1 \to s_2}, ..., \psi^t_{s_{w-1}+1 \to s_w}, ..., \psi^t_{s_{n-1}+1 \to s_n}\}$$

where the partition is initialized as $s_1 = 1$, $s_i = s_{i-1} + 1 \; \forall i \in (1, n]$

For every iteration we randomly choose to combine or to divide with 0.5 probability respectively. "Combine" means randomly choosing two sub-chains, then merge them to form a new sub-chain, for example if we combine $\psi^t_{s_1 \to s_2 - 1}$ and $\psi^t_{s_2 \to s_3 - 1}$, the new sub-chain will be $\psi^t_{s_1 \to s_3 - 1}$. "Divide" means randomly choosing one sub-chains and divide it into two sub-chains. The dividing index is randomly generated. For example. we divide $\psi^t_{s_1 \to s_2 - 1}$ with index $i$, the new sub-chains are $\psi^t_{s_1 \to i}$ and $\psi^t_{i+1 \to s_2 - 1}$.

After combining or dividing, we have obtain an alternative partition $\varphi^t_{alt}$. Then we calculate the optimal cost $C'$ of the alternative partition $\varphi^t_{alt}$ and use simulated annealing method to decide whether to update $\varphi^t$ with $\varphi^t_{alt}$ or not.

**Algorithm 1** Simulated Annealing for the first stage

**Input:** The number of VNFs $n_t$ in service chain;the required CPU cores $q^t_j(k)$ to operate VNF $\varphi^t_j$;the number of packets per second $p_t$; the number of available CPU cores $S_k$ on the $k$th server; the parameters($\alpha_k, \beta_k, \gamma_k$) for server k.
**Output:** The partition results of the service chain $\Omega$,($s_w, d_w$); the allocation results $\mathcal{F}$

```
1: Initialize temperature $T$,limit $L$, decay $D$, max iterations per temperature $I$
2: Initialize partition of sub chain $\varphi^t = \{\{\varphi^t_1\}, \{\varphi^t_2\}, ...\{\varphi^t_{n_t}\}\}$
3: Initialize $C$ = optimal cost for $\varphi^t$
4: while $T > L$ do
5:     for Every iteration do
6:         $\varphi^t_{alt}$ = Randomly combine or divide partitions in $\varphi^t$ once
7:         $C'$ = the optimal cost for new partition $\varphi'$
8:         if $C' < C$ then
9:             $\varphi^t = \varphi^t_{alt}, C = C'$
10:        else
11:            Update $\varphi^t$ and $C$ with $\varphi^t_{alt}$ and $C'$ with probability $exp((C - C')/T)$
12:        end if
13:    end for
14:    $T = T \cdot D$ for every $I$ iterations
15: end while
16: Obtain $\Omega$, $(s_w, d_w)$, $\mathcal{F}$ according to $\varphi^t$
17: Return $\Omega$, $(s_w, d_w)$, $\mathcal{F}$
```

Now we are going to analyze the performance of our algorithm-Simulated Annealing algorithm.
**Theorem2.** The time-complexity of simulated annealing is $O(n_t^4 \cdot m)$.
*Proof*: There are $s_t \in (0, n_t]$ subsets in the current partition for simulated annealing, so the inner loop for every iteration is $M \cdot s_t$, also there are $s_t$ calculation for cost for every iteration. So the inner loop is $M \cdot s_t + s_t^2$. Suppose there are $I$ iterations for every temperature, and there are $a$ iterations for "while" loop, where $a \in [log_D \frac{L}{T} \cdot b, \frac{(1+n_t) \cdot n_t \cdot n_t)}{2})$ as $a$ is upper bounded by the total number of possible partitions. The total iteration is $log_D(\frac{L}{T}) \cdot b \cdot (M \cdot s_t + s_t^2) + (a - log_D(\frac{L}{T}) \cdot b) \cdot (M \cdot s_t + s_t^2)$, the total run time is:

$$T(n) \leq \frac{(1 + n_t) \cdot n_t \cdot n_t)}{2} \cdot M \cdot n_t = O(n_t^4 \cdot M)$$

### 4.3 Solution to the second stage

In this section, we design an online algorithm based on the framework[3] to dynamically schedule VNF demands on different CPU cores.

The value of flow rate $f^k_r(t)$ for VNF $\varphi_r$ changes by time $t = \{1, 2, \cdots, T\}$ according to the allocation results in the first stage. We must make the decisions $x_{r,l}(t)$ at each time $t$ to observe the demands from clients while minimizing the total provision cost and switching cost. Besides, the introduction of switching cost and the ramp constraint makes our decisions more difficult.

First, we formulate the flow rate $f^k_r(t)$ as a column vector $\boldsymbol{f}(t)$ with $m$ components, where $\boldsymbol{f}(t) = [\; f^k_1(t), f^k_2(t), \cdots, f^k_m(t) \;]^T$. Note that $\boldsymbol{f}(t)$ is not a completely arbitrary variable since $\boldsymbol{f}(t)$ is generated from the first stage. Although we are able to make reasonable predictions for $\boldsymbol{f}(t)$, we don't know it until time $t$.

Our decisions $x_{r,l}(t)$ can also be formulated as a column vector $\boldsymbol{X}(t)$ with $m \cdot S_k$ components.

$$\boldsymbol{X}(t) = [\; \boldsymbol{x}_{1,l}(t),\; \boldsymbol{x}_{2,l}(t), \cdots, \boldsymbol{x}_{r,l}(t), \cdots, \boldsymbol{x}_{m,l}(t) \;]^T, \tag{13}$$

w $\boldsymbol{x}_{r,l}(t)$ follows

$$\boldsymbol{x}_{r,l}(t) = [\, x_{r,1}(t),\ x_{r,2}(t), \cdots,\ x_{r,l}(t), \cdots, x_{r,S_k}(t)\,]^T. \quad (14)$$

$\boldsymbol{x}^T$ is the transpose of vector $\boldsymbol{x}$ and $\boldsymbol{x}_n$ is the $n$th components of vector $\boldsymbol{x}$.

At each time $t$, the environment updates the VNF flow demand vector $\boldsymbol{f}(t)$, we must update our decision vector $\boldsymbol{X}(t)$ and suffer from switching penalties and ramp constraint to confirm the global optimal cost during time $t = \{1, 2, \cdots, T\}$.

The variables $\pi_{r,l}$ and $\beta_{r,l}$ can also be formulated as vectors $\boldsymbol{\pi}$ and $\boldsymbol{\beta}$ (both of them possess $m \cdot S_k$ components) according to the regulations mentioned in Eqn.(13) and Eqn.(14).

The capacity $C_l$ of CPU core $l$ can also be formulated as a vector $\boldsymbol{C} = [C_1, C_2, \cdots, C_{S_k}]^T$.

Here is the standardized form of the second stage:

**Standardized problem:**

$$\begin{aligned}
\min \quad & W^k = \textstyle\sum_{t=1}^{T} \boldsymbol{\pi}^T \cdot \boldsymbol{X}(t) + \boldsymbol{\beta} \cdot |\boldsymbol{X}(t) - \boldsymbol{X}(t-1)| \\
s.t. \quad & \boldsymbol{A^r} \cdot \boldsymbol{X}(t) + \boldsymbol{A^{r'}} \cdot \boldsymbol{f}(t)0, \quad \forall\, r \in [1, m],\ \forall\, t, \\
& \boldsymbol{B^l} \cdot \boldsymbol{X}(t) + \boldsymbol{B^{l'}} \cdot \boldsymbol{C}0, \quad \forall\, l \in [1, S_k],\ \forall\, t, \\
& |\boldsymbol{X}_n(t) - \boldsymbol{X}_n(t-1)|\Delta x \quad \forall\, n \in [1, m \cdot S_k].
\end{aligned}$$

In the standardized problem, $\boldsymbol{A^r}$, $\boldsymbol{A^{r'}}$, $\boldsymbol{B^l}$ and $\boldsymbol{B^{l'}}$ are coefficient vectors we can easily calculate by comparing standardize problem with Program (12).

According to **affine policy** mentioned in [9], the vector $\boldsymbol{X}(t)$ can be expressed by the linear form of input flow demand $\boldsymbol{f}(t)$:

$$\boldsymbol{X}(t) = \boldsymbol{b}(t) + \boldsymbol{k}(t) \cdot \boldsymbol{f}(t) \quad (15)$$

where $\boldsymbol{b}(t) \in \boldsymbol{R}^{(mS_k)}$ and $\boldsymbol{k}(t) \in \boldsymbol{R}^{(m)\times(mS_k)}$. Therefore, we can determine $\boldsymbol{X}(t)$ from $\boldsymbol{f}(t)$ by finding suitable $\boldsymbol{b}(t)$ and $\boldsymbol{k}(t)$ at each time $t$. We can only optimize the objective value as much as possible, since we do not know the coming $\boldsymbol{f}(t)$ in advance.

Therefore, we consider to minimizing the competitive ratio $CR$:

$$CR = \max_{\boldsymbol{f}(t)\in \boldsymbol{F}} \frac{W^k(\boldsymbol{b}(t), \boldsymbol{k}(t), \boldsymbol{f}(t))}{W^k_{off}(\boldsymbol{X}(t), \boldsymbol{f}(t))}$$

where $W^k_{off}(\boldsymbol{X}(t), \boldsymbol{f}(t))$ denote the optimal value if we know $\boldsymbol{f}(t)$ beforehand. To get an excellent competitive ratio of our online algorithm, we use $CR$ as the objective function instead of $W^k$. Combining the constraints in Standardized problem, we can formulate Affine problem as follows:

**Affine problem:**

$$\begin{aligned}
\min_{\boldsymbol{b}(t),\boldsymbol{k}(t)} \quad & CR = \max_{\boldsymbol{f}(t)\in \boldsymbol{F}} \tfrac{W^k(\boldsymbol{b}(t),\boldsymbol{k}(t),\boldsymbol{f}(t))}{W^k_{off}(\boldsymbol{X}(t),\boldsymbol{f}(t))} \\
s.t. \quad & \boldsymbol{A^r} \cdot \boldsymbol{b}(t) + (\boldsymbol{A^r k}(t) + \boldsymbol{A^{r'}}) \cdot \boldsymbol{f}(t) \le 0, \forall\, r \in [1, m],\ \forall\, t, \\
& \boldsymbol{B^l} \cdot \boldsymbol{b}(t) + \boldsymbol{B^l k}(t) \cdot \boldsymbol{f}(t) + \boldsymbol{B^{l'}} \cdot \boldsymbol{C} \le 0, \forall\, l \in [1, S_k],\ \forall\, t, \\
& |\boldsymbol{b}_n(t) - \boldsymbol{b}_n(t-1) + (\boldsymbol{kf})_n(t) + (\boldsymbol{kf})_n(t-1)| \le \Delta x, \\
& \qquad \forall\, n \in [1, m \cdot S_k],\ \forall\, t.
\end{aligned}$$

It is hard to solve the affine problem because the objective function involves a ratio of convex functions. We can use Lemma 1 in [3] to transform it to an equivalent convex optimization problem.

First, we define $\mu(t)$ as the upper bound of switching cost $W^k_s$, where $\mu(t) = \sup_{\boldsymbol{f}(t)\in \boldsymbol{F}} \boldsymbol{\beta}^T |\boldsymbol{b}(t) + \boldsymbol{k}(t)\boldsymbol{f}(t) - \boldsymbol{b}(t-1) - \boldsymbol{k}(t-1)\boldsymbol{f}(t)|$ . Then we relax the objective function in Affine problem by replacing the switching cost $W^k_s$ in the numerator by $\mu(t)$, then we obtain the relaxed objective function as follows

$$\min_{\boldsymbol{b}(t),\boldsymbol{k}(t),\mu(t)} \quad CR' = \max_{\boldsymbol{f}(t)\in \boldsymbol{F}} \frac{W^k(\boldsymbol{b}(t), \boldsymbol{k}(t), \boldsymbol{f}(t)), \mu(t)}{W^k_{off}(\boldsymbol{X}(t), \boldsymbol{f}(t))} \quad (16)$$

Note that the optimal solution remains the same because $W^k_s$ is a convex function of $\boldsymbol{f}(t)$. It will not influence the optimal value when the objective function reaches minimal. Then we use Lemma 1 in [3] to transform it to a convex optimization problem:

**Convex problem:**

$$\begin{aligned}
\min_{\boldsymbol{b}(t),\boldsymbol{k}(t),\mu(t)} \quad & \max_{\boldsymbol{f'}(t)\in \boldsymbol{F}, u>0} \left\{ W^k_p(\boldsymbol{f'}(t)) + \textstyle\sum_{t=1}^{T} \mu(t) u \right\} \\
s.t. \quad & \boldsymbol{A^r} \cdot \boldsymbol{b}(t) + (\boldsymbol{A^r k}(t) + \boldsymbol{A^{r'}}) \cdot \boldsymbol{f}(t) \le 0, \\
& \boldsymbol{B^l} \cdot \boldsymbol{b}(t) + \boldsymbol{B^l k}(t) \cdot \boldsymbol{f}(t) + \boldsymbol{B^{l'}} \cdot \boldsymbol{C} \le 0, \forall\, l \in [1, S_k],\ \forall\, t, \\
& |\boldsymbol{b}_n(t) - \boldsymbol{b}_n(t-1) + (\boldsymbol{kf})_n(t) + (\boldsymbol{kf})_n(t-1)| \le \Delta x, \\
& \qquad \forall\, n \in [1, m{\cdot}S_k],\ \forall\, t, \\
& u \cdot W_{off}(\tfrac{\boldsymbol{f'}(t)}{u}) \le 1.
\end{aligned}$$

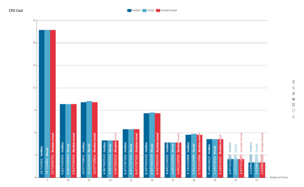
We can solve the convex problem through Interior Point Method (IPM) in polynomial time. After we get the optimal solution $\boldsymbol{b}(t)$ and $\boldsymbol{k}(t)$, we use the affine policy to determine $\boldsymbol{X}(t)$ according to $\boldsymbol{f}(t)$. The algorithm is shown as follows.

**Algorithm 2** An Online Algorithm for the second stage

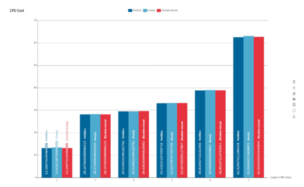
```
Input: The flow demand for each VNF $f(t)$ at time $t$; the set $F$
    containing all possible $f(t)$; the consumed CPU cycles for pro-
    cessing one packet $c_{r,l}$; the provision cost $\pi_{r,l}$; the switching
    cost $\beta_{r,l}$; the maximal CPU cycles $C_l$ for server $l$.
Output: The solution $\{x_{r,l}(t)\}$
1: Initialize the vectors $C(t)$, $\pi(t)$ and $\beta(t)$;
2: Solve the convex problem and acquire the optimal $b^*(t)$ and
   $k^*(t)$;
3: for $t = 1$ to $T$ do
4:     An $f(t)$ comes from $F$;
5:     Let $X(t) = b^*(t) + k^*(t)f(t)$;
6:     Get $x_{r,l}(t)$ according to (13) and (14);
7:     Return $x_{r,l}(t)$;
8: end for
```

## 5. EXPERIMENTS

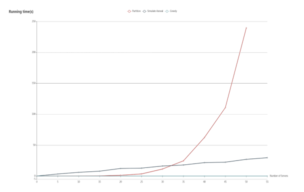
### 5.1 Experimental evaluation for the first stage



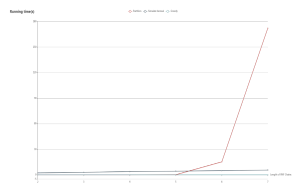
(a) CPU cost with different number of servers



(b) CPU cost with different length of NFV service chains



(c) Running time with different number of servers



(d) Running time with different length of NFV service chains

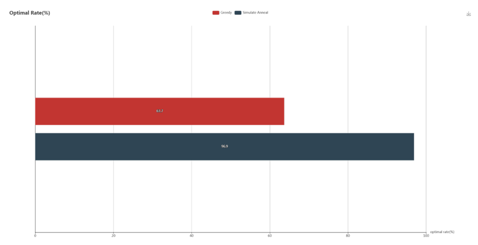


Fig. 3: Optimal rate of the two approximation algorithms

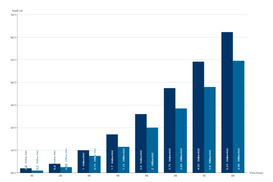
In first stage, we did two experiments: changing the number of servers with fixed NFV service chain length 3 and changing length of NFV service chain with fixed server number 10. We record the CPU cost and running time of three algorithms in both experiments.

We then compare the CPU cost and running time of partition, greedy and SA. Partition gives us optimal result all the time, but the running time will increase rapidly as number of servers or number of VNFs grows. Greedy runs the fastest but it suffers from the low optimal rate. SA has a very high chance of getting optimal cost (96.9%), and also has excellent running time when the number of servers or number of VNFs grows.

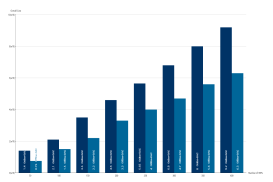
In addition, we did experiment to compare the chance of getting optimal result of greedy and SA with fixed number of server(17) and fixed length of NFV service chain(3) running for 500 times. Greedy has a 63.7% optimal rate while SA is near-optimal with 96.9 % optimal rate.

These results shows that greedy and SA are good approximations for the partition algorithm. Especially SA, which has both outstanding running time and near-optimal performance.

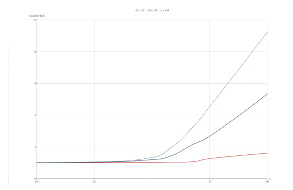
### 5.2 Experimental evaluation for the second stage



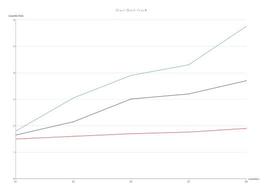
(a) Overall cost with different time horizons



(b) Overall cost with different number of VNFs



(c) Competitive ratio with the ratio between switching cost and provision cost



(d) Competitive ratio with the uncertainty level

In the second stage, we first fix $\epsilon = 0.1$ , $m = 100$ and $\gamma = 5$ to explore the relationship between time horizon $T$ and the overall cost of Online Problem and Offline Problem. As we can see, as T increases, the overall cost also increases but the online cost has never deviated from the offline optimal much. It is similar when we change the number of VNFs, which means our algorithm provides with an acceptable result.

In addition, we evaluate the effects of uncertainty $\epsilon$ and penalty ratio $\gamma$ on the competitive ratio with fixed T(100) and number of VNFs(50). As we can see, the competitive ratio rises when $\epsilon$ or $\gamma$ increases. However, in reality, $\epsilon$ and $\gamma$ cannot be that high. Therefore, our algorithm works well even with large penalty ratios and high uncertainty level.

## 6. CONCLUSION

In this project, we studied the problem of NFV chain scheduling on multi-core servers and proposed a two-stage optimization framework. We mainly focus on the first stage of the problem. We prove the hardness of the first stage and propose an approximation algorithm for it. Then we design an online algorithm for the second stage. Our evaluation results show that our algorithm for the first stage has excellent time performance while maintaining near-optimal property (96.9% optimal rate) and our algorithm for the second stage leads to significantly low competitive ratios.

## 7. ACKNOWLEDGEMENTS

We are grateful to professor Li Jiang who gives us the chance to work on this project by applying what we have learned in the Algorithm Design curriculum, such as the complexity analysis and the approximation algorithm design. We are also grateful to him for his inspiration on the design of the approximation algorithms.

We are also thankful the teaching assistants who help professor Li Jiang to organize the presentations of the project.

## REFERENCES

M. C. Luizelli, D. Raz, and Y. S. ’Ar. Optimizing NFV chain deployment through minimizing the cost of virtual switching. In INFOCOM, pages 1–9, 2018.

M. C. Luizelli, D. Raz, Y. Sa’ar, and J. Yallouz. The actual cost of software switching for NFV chaining. In IM, pages 335–343, 2017.

M. Shi, X. Lin, S. Fahmy, and D.-H. Shin. Competitive online convex optimization with switching costs and ramp constraints. In INFOCOM, pages 1–9, 2018.

M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba. Elastic virtual network function placement. In CloudNet, pages 255–260, 2015.

W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou. Traffic aware placement of interdependent NFV middleboxes. In INFOCOM, pages 1–9, 2017.

Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Ye. Provably efficient algorithms for joint placement and allocation of virtual network functions. In INFOCOM, pages 1–9, 2017.

S. G. Kulkarni, W. Zhang, J. Hwang, S. Rajagopalan, K. K. Ramakrishnan, T. Wood, M. Arumaithurai, and X. Fu. Nfvnice: Dynamic backpressure and scheduling for NFV service chains. In SIGCOMM, pages 71–84, 2017.

X. Wang, C. Wu, F. Le, and F. C. Lau. Online learning-assisted vnf service chain scaling with network uncertainties. In CLOUD, pages 205–213, 2017.

A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. Robust Optimization. Princeton University Press, 2009.