Orchestrating Service Chain Deployment with Plutus in Next Generation Cellular Core

Jiaqi Zheng†, Qiufang Ma†, Chen Tian†,
Haipeng Dai†, Wei Zhang‡, Guihai Chen†, Gong Zhang§
†State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
‡George Washington University, Washington, D.C., USA
§Huawei Research, China

ABSTRACT

Today's cellular core relies on a few expensive and dedicated hardware racks to connect the radio access network and the egress point to the Internet, which are geographically placed at fixed locations and use the specific routing policies. This inelastic architecture fundamentally leads to increased capital and operating expenses, poor application performance and slow evolution. The emerging paradigm of Network Function Virtualization (NFV) and Software Defined Networking (SDN) bring new opportunities for cellular networks, which makes it possible to flexibly deploy service chains on commodity servers and fine-grained control the routing policies in a centralized way.

We present a two-stage optimization framework *Plutus*. The network-level optimization aims to minimize the service chain deployment cost, while the server-level optimization requires to determine which Virtualized Network Function (VNF) should be deployed onto which CPU core to balance the CPU processing capability. We formulate these two problems as two optimization programs and prove their hardness. Based on parallel multi-block ADMM, we propose an $(\mathcal{O}(1), \mathcal{O}(1))$ bicriteria approximation algorithm and a 2-approximation algorithm. Large-scale simulations and DPDK-based OpenNetVM platform show that Plutus can reduce the capital cost by 84% and increase the throughput by 36% on average.

KEYWORDS

Cellular networks, SDN, NFV

ACM Reference Format:
Jiaqi Zheng†, Qiufang Ma†, Chen Tian†, Haipeng Dai†, Wei Zhang‡, Guihai Chen†, Gong Zhang§. 2019. Orchestrating Service Chain Deployment with *Plutus* in Next Generation Cellular Core. In *IEEE/ACM International Symposium on Quality of Service (IWQoS '19), June 24–25, 2019, Phoenix, AZ, USA*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3326285.3329045

IWQoS '19, June 24–25, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6778-3/19/06…$15.00
https://doi.org/10.1145/3326285.3329045

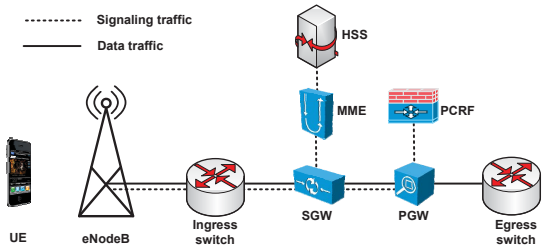


Figure 1: Today's cellular network architecture.

1 INTRODUCTION

Motivation: Cellular core is a critical piece of the infrastructure and provides fundamental cellular-specific functions such as user authentication, mobility management and session management, etc. It also requires to support various middlebox services to implement per-user accounting and charging rules of voice calls [16]. However, today's cellular infrastructure are experiencing explosive growth in mobile connected devices. A Report from Cisco suggested that there would be 3 billion IoT devices and around 11.6 billion mobile connected devices by 2020 [2]. In the meantime, the growth rate of signal traffic is more than 50% faster than that of data traffic [3], which together creates huge stresses on the cellular core.

On one hand, in order to response the rapid growth of cellular traffic, the providers have to purchase and deploy more expensive and dedicated hardware racks, which inevitably leads to unfavorable capital and operating expenses. On the other hand, current architecture heavily relies on these dedicated middleboxes [30] to connect the radio access network and the egress point to the Internet, which are geographically placed at fixed locations and use the specific routing configurations. They cannot perfectly react to the changing traffic volume and dynamic policies. The renewal cycles of service innovation have to be prolonged and hindered by vendor support. Furthermore, with more and more hardware racks are deployed, the cellular network protocols become complex and intractable, leading to high management overhead.

Existing works attempt to address these issues above from different angles. CleanG [18] and LTE-Xtend [21] design a simplified control protocol in SDN-based cellular architecture to support emerging mobile devices and services. Usually a set of cellular-specific functions can be virtualized as a service chain, which consists of a sequence of VNFs. The

work [4, 10, 25, 26] consider service chain embedding problem, i.e., determining which VNF can be deployed onto which commodity server such that the packets can be sequentially processed by these VNFs and comply with the service chain constraints. However, the routing selection and VNF deployment are optimized separately, which leads to suboptimal deployment cost in nature. In addition, the traffic has to route from an upstream VNF located on one server to the downstream VNF located in another server to perform a specific function defined by a service chain, which takes up expensive bandwidth resource due to the traffic transmission between two servers. To save the network bandwidth consumption, NFVnice [15] and NFP [31] integrate the whole service chain into a server with multiple physical CPU cores, where one VNF can be fine-grained deployed onto one physical CPU core and different VNFs can share the same one [33]. However, the optimization framework of such service chain deployment with multiple CPU cores has not been explored in the existing literature.

In this paper we initiate the study of orchestrating the service chain deployment with multiple CPU cores, aiming to minimize the provisioning cost, which has the potential to overcome the drawbacks above. The novelty of our work lies in a comprehensive exploration and design based on the multi-core CPU framework, which to our knowledge has not been done before. The cellular traffic in our framework can be dynamically managed by a logically centralized controller in a fine-grained manner. By virtualization techniques, the operator can accelerate the innovation by shortening the renewal cycles of service chain deployment, reducing the capital cost and improving the scalability. In the first stage, we focus on a joint optimization of traffic routing and the service chain deployment. Furthermore, for each VNF in a service chain, we seek to determine which VNF should be deployed onto which CPU core in the second stage, in order to balance the processing capacity and improve the throughput.

Our contributions: Firstly, we propose a two-stage optimization framework *Plutus* for Minimum Provisioning Cost Problem (MPCP) and Multi-Core Deployment Problem (MCDP). The optimization program in the first stage aims to minimize the total provisioning cost of service chains, where the required CPU resource and cost for each service chain are given, such that each link's load cannot beyond its capacity and each server's resource cannot be overbooked. The service chain consists of a set of VNFs. The program in the second stage needs to determine which VNF should be placed onto which CPU core to balance the CPU processing capacity.

Our second contribution is a set of algorithms to solve MPCP and MCDP. We prove that MPCP and MCDP are both NP-hard, and thus focus on designing approximation algorithms. Based on the multi-block ADMM, we first propose an $(\mathcal{O}(1), \mathcal{O}(1))$ bicriteria approximation algorithm and prove that it yields a constant approximation ratio of δ, while overbooking the CPU resource capacity at each server by at most a factor of 2, where δ is the number of pre-defined paths between source and destination. We further propose a local search algorithm with constant approximation ratio 2,

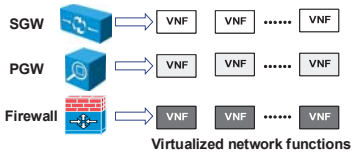


Figure 2: Legacy hardware devices can be virtualized into multiple software instances (virtualized network functions).

which improves upon the results of randomized rounding by greedily moving each VNF at each CPU core.

Our third contribution is a comprehensive performance evaluation of our algorithms. Large-scale simulations using synthetic cellular network topologies show that our algorithms can reduce the total provisioning cost by 84%. Meanwhile, our algorithms run faster compared to state of the art and can provide the near optimal solution. We also develop a prototype on the DPDK-based OpenNetVM platform [34]. Experimental results show that our solution can increase the throughput by 36% on average.

2 RELATED WORK

SDN-based cellular core: SoftCell [13] and SoftMoW [19] both present a SDN-based cellular core architecture, where the signal and data traffic are explicitly managed by a logically centralized controller. The main difference between them is that the former aims to minimize the number of forwarding rules in the core switches, while the latter focuses on improving the performance for latency-sensitive applications. Another line of this work advocate to improve the design of control plane protocols. For example, CleanG [18] develops a novel protocol customized for emerging IoT services. LTE-Xtend [21] extends the existing protocols to support M2M communication. ProCel [20] increases the EPC capacity by optimizing the interaction between eNBs and EPC.

EPC network function virtualization: The work in [7, 14, 24, 29] virtualize the cellular-specific functions and guarantee that they can provide backward compatible function. SCALE [6] re-organizes the MME functionality into a front-end load balancer and back-end virtualized processing cluster to improve scalability. KLEIN [25] routes the traffic to the available EPC instances located in geographically distributed data centers and manages virtualized EPC resources. PEPC [26] decomposes the traditional EPC functions into different components and reduces frequent communication by eliminating the duplicated device states.

VNF deployment and scheduling: VNF-P [17] propose a hybrid VNF deployment model to allocate physical resources, i.e., network services can be provided by a mixture of traditional dedicated hardware and VNFs. As for the fully virtualized environment, Ghaznavi et al. [12] present a model to minimize the operational cost and provide elastic services. Furthermore, Cohen et al. [10] develop an approximation algorithm to minimize the distance cost between the clients and VNFs such that the capacity constraint of single resource should be satisfied. To reduce the CPU overhead, E2 [22]

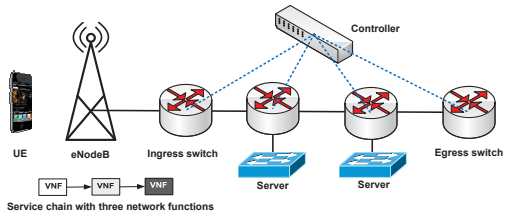


Figure 3: The network-level optimization in the first stage of Plutus.

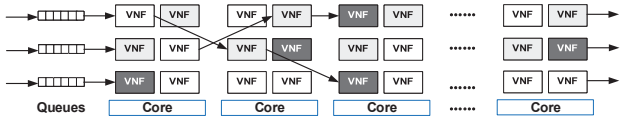


Figure 4: The server-level optimization in the second stage of Plutus.

integrates network functions into a shared address space and executes VNFs in a run to completion manner by one thread. NFP [31] accelerates the packet processing by orchestrating VNFs in a parallel manner. NFVnice [15] proposes a network functions scheduling framework on the shared CPU cores to achieve rate-cost fairness.

3 AN OPTIMIZATION FRAMEWORK

3.1 Cellular Core Background and Motivation

Today's cellular network architecture [1] is briefly shown in Fig. 1. The dotted lines and solid lines represent signaling traffic and data traffic, respectively. It mainly consists of the Radio Access Network (RAN) and the Evolved Packet Core (EPC). The RAN is a radio interface that connects User Equipment (UE) and eNodeBs (i.e., base stations). Once the traffic from UE arrives at eNodeBs, it will be forwarded to the EPC, where the EPC consists of a set of hardware racks such as Mobility Management Entity (MME), Serving Gateway (SGW), PDN Gateway (PGW), Home Subscriber Database (HSS) and Policy Charging Rules Function (PCRF). The MME handles all the signaling traffic from the UEs and the eNodeBs, and is responsible for user authentication, mobility management and session management. The SGW and PGW process all the data traffic. The SGW forwards the data traffic from the eNodeBs to the PGW, and PGW queries PCRF for setting the charging rules. The PGW then forwards the data traffic to a specific egress switch to the Internet.

This inelastic architecture suffers from poor scalability [27], high management complexity [5] and capital costs [23]. To address these issues, the legacy hardware devices can be virtualized into multiple software instances as shown in Fig. 2. Based on this, we propose a two-stage optimization framework. The network-level optimization in the first stage is based on SDN architecture shown in Fig. 3, where a logically centralized controller has a global view and is responsible for directing the mobile traffic passing through a service chain in the data plane. Given the flow demand and the pre-defined service policies, the controller requires to install the optimal routing and determine the service chain deployment with the objective of minimizing the total cost, such that each link's load in the network cannot beyond its capacity and each server's CPU resource cannot be overbooked. The server-level optimization in the second stage needs to determine which VNF should be placed onto which CPU core to balance the CPU processing capacity and improve the throughput.

3.2 Provisioning Model and Problem Formulation

Table 1: Key notations in this paper.

F	The set of flows f
V	The set of switches v
E	The set of links e
L	The set of servers l.
G	The acyclic directed network graph $G = (V \cup L, E)$
SC	The set of service chains i.
NF	The set of network functions j.
K_l	The set of CPU cores at the server l.
$r_{i,l}$	The required CPU resource for the service chain i at the server l
R_l	The total CPU resource at server l
$c_{i,l}$	The provisioning cost for the service chain i at the server l
b_e	The bandwidth capacity of link e
P_f	The set of possible paths for flow f from ingress switch v_f^+ to egress switch v_f^-
d_f	The demand of the flow f
$\alpha_{f,i}$	The indicator variable that equals 1 if flow f should pass through service chain i and 0 otherwise
$\beta_{i,j}$	The indicator variable that equals 1 if network function j belongs to the service chain i and 0 otherwise
σ_j	The consumed CPU cycles per packet for network function j
$x_{i,l}$	The indicator variable that equals 1 if service chain i locates at the server l and 0 otherwise
$y_{f,p}$	The fractional flow demand for flow f on path p

Before formulating the problem, we first present our network model. A network is a directed graph $G = (V \cup L, E)$, where V is the set of switches, L is the set of servers and E the set of links with capacities b_e. Each flow f is associated with a demand d_f, splitted at the ingress switch v_f^+ among the possible path set P_f and routed to the egress switch v_f^-. Before going out to the egress switch, each flow f should pass through a specific service chain i deployed at least a server. In the first stage of Plutus, the flow demand d_f and the service policy $\alpha_{f,i}$ (which flow should pass through which service chain) is known, and which service chain should be placed onto which server needs to be determined. In the second stage of Plutus, we need to determine which network function is required to be placed onto which CPU core so as to balance the processing capability of multiple CPU cores. For convenience, we summarize important notations in Table 1.

Based on the above model and definition, we first formulate the *Minimum Provisioning Cost Problem (MPCP)* as a program to solve the network-level optimization in the first stage of Plutus. The formulation is shown in (1) and that in the second stage is shown in (2). We will discuss them soon.

$$\text{minimize} \quad \sum_{l\in L}\sum_{i\in SC} c_{i,l}\cdot x_{i,l} \tag{1}$$
$$\text{subject to} \quad (1a), (1b), (1c), (1d), (1e), (1f).$$

The objective of formulation (1) aims to minimize the sum of provisioning cost $c_{i,l}$ in the whole network. Basically, we seek to find an optimal routing and service chain deployment schemes so as to minimize the total provisioning cost, such that each link's load cannot beyond its capacity and the CPU resource at each server cannot be overbooked.

$$\sum_{i\in SC} x_{i,l}\cdot r_{i,l} \le R_l, \quad \forall l \in L, \tag{1a}$$

For each server $l \in L$, constraint (1a) indicates that the sum of provisioning resource for each service chain must be less than or equal to the total resource R_l. The zero-one integer variable $x_{i,l}$ equals one when the service chain i is placed at the server l, and equals zero otherwise.

$$\sum_{f\in F} d_f \sum_{p\in P_f: e\in p} y_{f,p} \le b_e, \quad \forall e \in E, \tag{1b}$$

The LHS of constraint (1b) characterizes the load of total flows at link e, which must be less than or equal to its capacity. This optimization variable $y_{f,p}$ determines that the fractional flow demand for flow f on path p.

$$\sum_{i\in SC}\sum_{l\in p} \alpha_{f,i}\cdot x_{i,l} \ge y_{f,p}, \quad \forall f \in F, \forall p \in P_f, \tag{1c}$$

Constraint (1c) ensures that if the flow is routed on the path p, the service chain i corresponding to the flow f should be placed onto at least one of the servers on this path.

$$\sum_{p\in P_f} y_{f,p} = 1, \quad \forall f \in F, \tag{1d}$$

Constraint (1d) is the flow demand conservation constraint. The sum of all fractional flow demand among all the possible paths should equal to d_f.

$$x_{i,l} \in \{0,1\}, \quad \forall i \in SC, \forall l \in L, \tag{1e}$$

$$y_{f,p} \ge 0, \quad \forall f \in F, \forall p \in P_f, \tag{1f}$$

The zero-one integer variable $x_{i,l}$ indicates if service chain i can be placed onto the server l or not.

In the second stage of Plutus, we focus on server-level optimization. A service chain consists of a set of VNFs. The packets are sequentially processed from upstream VNF to downstream VNF. The maximum throughput of one service chain depends on that of the bottleneck VNF. Which service chain is deployed onto which server has been fixed in the first stage, we need to determine which VNF should be deployed onto which CPU core in the second stage. Given the solution $x_{i,l}$ and the computation cost (the product of packet arrival rate and consumed CPU cycles per packet) of different VNFs, the objective is to balance the processing capability of multiple CPU cores and improve service chain throughput. Once the server-level deployment is complete, the OS scheduler will assign CPU time for each running VNF proportional to its computation cost. Now we formulate the *Multi-Core Deployment Problem (MCDP)* as an optimization program shown in (2).

$$\text{minimize} \quad \max_{k\in K_l, j\in NF_l} w_j \cdot q_{j,k} \tag{2}$$
$$\text{subject to} \quad w_j = y_{f,p}\cdot \alpha_{f,i}\cdot \beta_{i,j}\cdot \sigma_j, \quad \forall j \in NF_l, \tag{2a}$$
$$\sum_{k\in K_l} q_{j,k} = 1, \quad \forall j \in NF_l, \tag{2b}$$
$$q_{j,k} \in \{0,1\}, \quad \forall j \in NF_l, \forall k \in K_l. \tag{2c}$$

The objective of formulation (2) aims to minimize the maximum CPU load on a physical core. The CPU load w_j is defined by the product of $y_{f,p}$ and σ_j.

$$w_j = y_{f,p}\cdot \sigma_j$$

where $y_{f,p}$ is the flow rate (packet arrival rate) and σ_j is the consumed CPU cycles per packet. The constant parameters $\alpha_{f,i}$ and $\beta_{i,j}$ in constraint (2a) describes the correlation among the flow f, service chain i and network function j. The zero-one integer variable $q_{j,k}$ indicates that which network function j should be placed onto which CPU core k.

3.3 Hardness Analysis

We establish the hardness of MPCP and MCDP below.

Theorem 3.1. *MPCP is NP-hard.*

Proof. Consider a special case of MPCP as illustrated in Fig. 5(b), where white nodes represent source or destination, and gray nodes represent the servers. All the flows from source to destination share one common path (gray nodes) and each server on this path has the identical resource capacity R. Each flow f is associated with one service chain i, which is required to be deployed onto at least one server l. If service chain i is deployed onto the server l, it will incur provisioning cost $c_{i,l}$. Our objective aims to minimize the total provisioning cost such that the CPU resource capacity at each server cannot be overbooked.

We construct a reduction with polynomial time from the Generalized Assignment Problem (GAP) [32] to the special case of MPCP. As shown in Fig. 5(a), the GAP aims to assign n jobs to m machines, where each job can only be assigned to exactly one machine. If the job j is assigned to the machine q, the processing time units and the incurred cost in machine q is $p_{q,j}$ and $c_{q,j}$ respectively. A processing time bound T for each machine is given to limit the total processing time. The objective of GAP is to find a feasible assignment strategy such that the total cost is minimized. The job j, machine i and processing time bound T in GAP correspond to the service chain i, server l and CPU resource capacity R, respectively. Therefore, any feasible solution of GAP corresponds to the special case of MPCP in Fig. 5(b), and vice versa. □

Theorem 3.2. *MCDP is NP-hard, even for a server only consisting of two CPU cores.*

Proof. Given a special case of MCDP with only two CPU cores, we construct a polynomial reduction from the

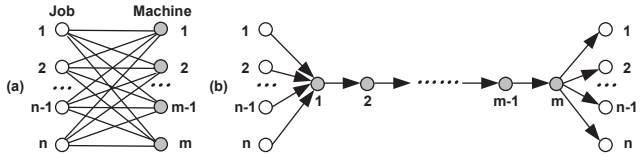


Figure 5: Reduction from GAP to MPCP.

set partition problem [9] to it. Consider a partition instance $\mathbb{A}$ consisting of n items, each with a value a_i, $a_j \in \mathbb{R}, j \in \{1, 2, \ldots, n\}$. The objective is to partition $\mathbb{A}$ into two subsets $\mathbb{A}_1$ and $\mathbb{A}_2$ ($\mathbb{A}_1 \cup \mathbb{A}_2 = \mathbb{A}$ and $\mathbb{A}_1 \cap \mathbb{A}_2 = \emptyset$) such that $|A_1 - A_2|$ is minimized, where A_1 and A_2 denote the sums of the elements in each of the two subsets $\mathbb{A}_1$ and $\mathbb{A}_2$. Accordingly, for each item in set $\mathbb{A}$ we introduce one network function j, where $w_j = a_j$. There are n items in total and thus we introduce n network functions in MCDP.

The partition results indicate that which network function should be placed onto which CPU cores. Therefore, any partition with minimum difference between set A_1 and A_2 corresponds to MCDP with only two CPU cores, and vice versa. The network functions placed onto the first CPU core forms one set of the partition, and that placed onto the second CPU core forms the other. □

4 ALGORITHMS

In this section, we design network-level and server-level optimization algorithms in Plutus, respectively.

4.1 Network-level optimization algorithm

Algorithm 1: A Proximal Jacobian ADMM Algorithm

Input : Network graph $G = (V \cup L, E)$; the set of flow f; the set of service chain i.
Output: A fractional solution $\{\tilde{x}_{i,l}\}$ and $\{\tilde{y}_{f,p}\}$ to the relaxed LP of (1)

1 Transform the program (1) to the program (3).
2 Initialize the variables $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, $\hat{\boldsymbol{x}}$, $\boldsymbol{y}$ and multipliers $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$, $\boldsymbol{\sigma}$ to zero.
3 **for** $t = 1, 2, \cdots$ **do**
4 Update $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, $\hat{\boldsymbol{x}}$, $\boldsymbol{y}$ from programs (5), (6), (7), (8), (9) in parallel.
5 Update $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$, $\boldsymbol{\sigma}$ from equations (10), (11), (12).

The mixed integer program (1) aims to minimize the total provisioning cost, which can be relaxed to a linear program by replacing the constraint (1e) with $x_{i,l} \geq 0$. Since constraint (1c) and (1d) hold, $x_{i,l}$ are in fact real numbers between 0 to 1. The optimal fractional solutions $\{\tilde{x}_{i,l}\}$ and $\{\tilde{y}_{f,p}\}$ of the relaxed LP of (1) can be obtained in polynomial time using standard solvers. However, solving this program is time-consuming especially in large-scale production networks with thousands of flows. Thus we set out to find a scalable algorithm to solve this program instead. Inspired by the framework of multiple-block ADMM [11], we develop a proximal Jacobian ADMM algorithm that can converge to an optimal solution at the rate of $o(\frac{1}{t})$ in Algorithm 1, where t is the number of iteration times. As parts of the constraints in program (1) are inequalities and the variables $x_{i,l}$ are coupled together, we require to transform program (1) to program (3) in order to apply 5-block ADMM (line 1). The detailed transformation is illustrated in Appendix A. Furthermore, we initialize the original variables $\boldsymbol{y}$, the introduced auxiliary variables $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, $\hat{\boldsymbol{x}}$, and multipliers $\boldsymbol{\theta}$, $\boldsymbol{\varphi}$, $\boldsymbol{\sigma}$ to zero (line 2) and solve each subprogram in parallel (lines 3-5).

Based on the fractional solution in Algorithm 1, we design a ($\mathcal{O}(1)$, $\mathcal{O}(1)$) bicriteria approximation algorithm that rounds the fractional solution to a feasible integer solution. The complete algorithm is shown in Algorithm 2. We now explain the high-level working of this algorithm. We first obtain the optimal fractional solution $\{\tilde{x}_{i,l}\}$ and $\{\tilde{y}_{f,p}\}$ to the relaxed LP of (1) (line 1). For the solution $\{\tilde{y}_{f,p}\}$, it is already a feasible solution, while for the solution $\{\tilde{x}_{i,l}\}$. For the solution $\{\tilde{x}_{i,l}\}$, it indicates that the service chain i can be fractionally placed onto all servers l on its path p ($l \in p$), and accordingly the required CPU resource is proportional to the fractional solution. We require to round it to an integer solution by constructing a complete bipartite graph (S, U, E') (lines 7-31). Initially, the set S and U are both empty set (line 2). We first add each solution $\tilde{y}_{f,p}$ into the set S (lines 3-6). And then, for each server, we assign k_l slots to accommodate the fractional solution $\{\tilde{x}_{i,l}\}$ one by one according to its required resource $r_{i,l}$ (line 8). Next we add u_l^j into set U, whose cardinality is the product of the number of servers l and the number of assigned slots k_l (line 11). Note that $\tilde{x}_{i,l}$ could be split into two adjacent slots, and we use notation $u_{i,l}^j$ to represent the corresponding parts. The value of $u_{i,l}^j$ is calculated from a loop procedure (lines 13-21). If $u_{i,l}^j$ is greater than zero, we add an edge $(\tilde{y}_{f,p}, u_l^j)$ with weight $c_{i,l}$ into E' and finish the construction procedure of complete bipartite graph (S, U, E') (line 22-26). Based on this graph, we compute a complete matching M with the minimum total weight (line 27). If there exists an edge in the matching M, we set $\hat{x}_{i,l}$ is equal to one that indicates the service chain i can be placed onto server l; otherwise, we set it to zero (lines 29-31).

Now we analyze the performance of the algorithm by introducing the related definition.

Definition 4.1. Let OPT_1 be the optimal solution to (1), which gives a lower bound of total provisioning cost.

THEOREM 4.2. *Algorithm 2 is an ($\mathcal{O}(1)$, $\mathcal{O}(1)$) bicriteria approximation algorithm, which has a constant approximation ratio of δ, while overbooking the resource capacity at each server by at most a factor of 2, where δ is the number of pre-defined paths between ingress and egress switch.*

The proof can be found in Appendix B.

4.2 Server-level optimization algorithm

Given the routing configurations and assigned flow rate from the network-level optimization, the server-level optimization seek to find a network function deployment solution to balance the processing capability of multiple CPU cores. As shown in Algorithm 3, we first obtain the optimal fractional solution $\{\tilde{q}_{j,k}\}$ to the relaxed LP of (2) by replacing the

Algorithm 2: A Bicriteria Approximation Algorithm

Input : Network graph $G = (V \cup L, E)$; the set of flow f; the set of service chain i.
Output: A solution $\{\hat{x}_{i,l}\}$ to (1).

1 Obtain the optimal fractional solution $\{\tilde{x}_{i,l}\}$ and $\{\tilde{y}_{f,p}\}$ to the relaxed LP of (1).
2 $S = U = \emptyset$
3 **for** *each* $f \in F$ **do**
4 **for** *each* $p \in P_f$ **do**
5 **if** $\tilde{y}_{f,p} > 0$ **then**
6 $S = S \cup \tilde{y}_{f,p}$
7 **for** *each* $l \in L$ **do**
8 $k_l = \left\lceil \sum_{i \in SC} \tilde{x}_{i,l} \right\rceil$
9 Sort $\tilde{x}_{i,l}$ in descending order according to $r_{i,l}$ into set X
10 **for** $j = 1$ *to* k_l **do**
11 $U = U \cup u_l^j$
12 $\Phi = 1$
13 **repeat**
14 Get the first element $\tilde{x}_{i,l}$ from set X
15 $\Delta = \max(\tilde{x}_{i,l}, \tilde{x}_{i,l} - \Phi)$
16 $\Phi = \Phi - \Delta$
17 $u_{i,l}^j = \Delta$
18 $\tilde{x}_{i,l} = \tilde{x}_{i,l} - \Delta$
19 **if** $\tilde{x}_{i,l} = 0$ **then**
20 Remove $\tilde{x}_{i,l}$ from set X
21 **until** $\Phi = 0$;
22 **for** *each* $\tilde{y}_{f,p} \in S$ **do**
23 Determine service chain i corresponding to flow f
24 **for** *each* $u_l^j \in U$ **do**
25 **if** $u_{i,l}^j > 0$ **then**
26 Add an edge $(\tilde{y}_{f,p}, u_l^j)$ with weight $c_{i,l}$ into E'
27 Construct a bipartite graph (S, U, E') and compute a complete matching M with the minimum total weight.
28 **if** *there exists an edge in the matching* M **then**
29 $\hat{x}_{i,l} = 1$
30 **else**
31 $\hat{x}_{i,l} = 0$

constraint (2c) with $q_{j,k} \geq 0$ (line 1). For each $j \in NF_f$, we apply randomized rounding to obtain an integer solution $\{\hat{q}_{j,k}\}$ (lines 2–10). To ensure that only one CPU core is chosen for a network function $j \in NF_f$, the optimal fractional solution can be viewed as partitioning the interval $[0, 1]$ to intervals of lengths $\{\tilde{q}_{j,k}\}$ (lines 4–7). A real number is generated uniformly at random in $(0, 1]$ and the interval in which it lies determines the CPU core (lines 8–10).

Before analyzing the performance of Algorithm 3, we introduce the following definition.

Definition 4.3. Let OPT_2 be the optimal solution to (2), which gives a lower bound of maximum CPU load.

THEOREM 4.4. *[32] Algorithm 3 outputs a feasible solution with maximum CPU load bounded by* $\mathcal{O}(\log |K_l|) \cdot OPT_2$ *with high probability, where* $|K_l|$ *is the number of CPU cores at the server* l.

The proof of Theorem 4.4 can be found in [32].

In spite of the guaranteed approximation ratio in Algorithm 3, it is not always efficient as it may occasionally produce a bad solution. The local search algorithm improves upon the solution of randomized algorithm by greedily moving each network function to another CPU core with less load. This algorithm achieves a constant approximation ratio of 2.

Algorithm 3: A Randomized Algorithm

Input : The set of CPU cores at the server l; the consumed CPU cycles for network function j; the indicator variables $\alpha_{f,i}$ and $\beta_{i,j}$.
Output: A solution $\{\hat{q}_{j,k}\}$ to (2).

1 Obtain the optimal fractional solution $\{\tilde{q}_{j,k}\}$ to the relaxed LP of (2).
2 **for** *each* $j \in NF_l$ **do**
3 $K_l' = \emptyset$
4 **for** *each* $k \in K_l$ **do**
5 $\hat{q}_{j,k} = 0$
6 $K_l' = K_l' \cup k$
7 $l_{j,k} = \sum_{k' \in K_l'} \tilde{q}_{j,k'}$
8 Generate a number r in (0,1] uniformly at random
9 Find $\hat{p}$ such that $r \leq l_{j,k}$ and $l_{j,k} - r$ is minimum
10 $\hat{q}_{j,k} = 1$

Algorithm 4: A Local Search Algorithm

Input : The set of CPU cores at the server l; the consumed CPU cycles for network function j; the indicator variables $\alpha_{f,i}$ and $\beta_{i,j}$.
Output: A solution $\{\hat{q}_{j,k}\}$ to (2).

1 Apply Algorithm 3 to obtain the initial solution $\{\hat{q}_{j,k}\}$.
2 **repeat**
3 $w^+ = \max_{k \in K_l} \{\sum_j w_j \cdot \hat{q}_{j,k}\}$
4 $w^- = \min_{k \in K_l} \{\sum_j w_j \cdot \hat{q}_{j,k}\}$
5 $g^+ = |K^+|$, where $K^+ = \{k | \sum_j w_j \cdot \hat{q}_{j,k} = w^+\}$
6 $g^- = |K^-|$, where $K^- = \{k | \sum_j w_j \cdot \hat{q}_{j,k} = w^-\}$
7 $\forall k^- \in K^-$
8 **for** *each* $k^+ \in K^+$ **do**
9 **for** *each* $j \in NF_l$ **do**
10 **if** $q_{j,k^+} = 0$ **then**
11 **continue**
12 Move network function j from k^+ to k^-
13 Re-calculate w^+ and g^+
14 **if** *the* w^+ *value or the* g^+ *value decreases* **then**
15 $q_{j,k^+} = 0$
16 $q_{j,k^-} = 1$
17 $NF_l = NF_l \setminus \{j\}$
18 **until** $NF_l = \emptyset$;

We are now ready to describe our local search algorithm shown in Algorithm 4. We first run Algorithm 3 and obtain an initial solution $\{\hat{q}_{j,k}\}$ (line 1). Next we iteratively move network function to another CPU core with less load to balance the CPU processing capability until we cannot find a better solution (lines 2-18). The notation w^+ and w^- indicate the maximum and minimum CPU load corresponding to the current solution $\{\hat{q}_{j,k}\}$ (lines 3-4), while g^+ and g^- indicate the number of CPU cores with maximum and minimum load, respectively (lines 5-6). For each network function j, we try to move it to CPU k^- since CPU k^- has the least CPU load currently (line 12). If this movement results in the decrease of w^+ value or g^+ value, we move network function j from CPU k^+ to k^- (lines 15-16). Finally we remove the network

function j from the set NF_l and the algorithm enters into the next loop (line 17). Based on the analysis above, we have Theorem 4.5 and its proof can be found in [32].

THEOREM 4.5. *[32] After at most $|NF_l|$ iterations, Algorithm 4 terminates and approximates MCSP with a factor of 2, where $|NF_l|$ is the number of network functions at server l.*

5 EXPERIMENTAL EVALUATION

We evaluate our two-stage optimization algorithm using both prototype implementation and large-scale simulation.

Benchmark schemes: We compare the following schemes with our algorithm.

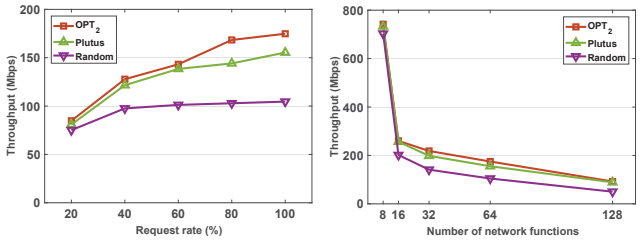
- **HW**: The network function of each type relies on traditional hardware middleboxes.
- **Greedy**: Each service chain is randomly deployed onto different servers.
- **Random**: For a service chain consisting of a set of VNFs, each VNF is randomly deployed onto multiple CPU cores.
- **Plutus**: Our approximation algorithm in Algorithm 2 and Algorithm 4.
- **OPT**: The optimal solution OPT_1 and OPT_2 for the MPCP and MCDP in the integer program (1) and (2) obtained using branch and bound.

Unless stated otherwise, we configure ρ, w and ι to be 0.1, 0.02 and 1.0 respectively in Algorithm 1 as suggested in [11].

5.1 Implementation and Testbed Emulations

Implementation: We develop a prototype of our algorithms on the DPDK-based OpenNetVM platform [34], where the polling mechanism is used in RX and TX threads for receiving and sending packets from NIC. Now we describe how to perform VNF deployment in the service chain in our experiments. We first obtain solutions to MPCP and MCDP using Algorithm 2 and 4 respectively. According to these solutions, we bind each VNF to a dedicated CPU core. The `CORELIST` parameter in OpenNetVM specify the index of CPU core, and the index parameters `SERVICE_ID` and `DST` indicate two adjacent VNFs in a service chain, i.e., once the packets have already been processed by an upstream VNF indexed by `SERVICE_ID`, they would be forwarded to a downstream VNF indexed by `DST`.

Testbed setup: In our experimental setup, we use two severs, each of which has dual Xeon(R) E5-2630 @ 2.40GHz CPUs (2x8 physical cores), an Intel 82599ES 10G dual port NIC and 128GB memory. Each server runs Ubuntu 14.04.3 with kernel version 3.19.0. We use `pktgen` to generate different UDP flows with 64 byte packet size in each run, where all of them are required to pass through a pre-defined ordered VNFs deployed onto different CPU cores. The VNFs we used perform forwarding and monitor function, which form a linear service chain.



(a) The throughput with different packet arrival rate (b) The throughput with different number of VNFs

Figure 6: The throughput comparison.

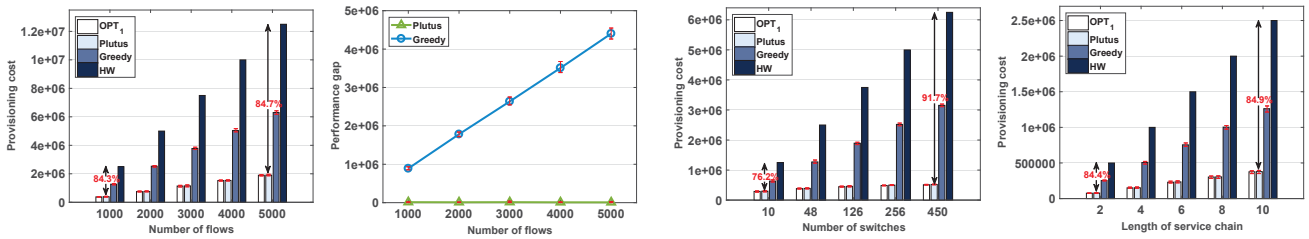
Experiment results: We study the achieved throughput (Mbps) with different packet arrival rate and different number of VNFs in Fig. 6. In Fig. 6(a), the number of running VNFs is fixed at 64. The 100% arrival rate corresponds to around 7 Gbps. We can observe that the achieved throughput of Plutus consistently outperforms that of Random by 31.5% on average when the packet arrival rate become larger. Specifically, Plutus can improve the throughput by 48.5% compared with Random when the number of packet arrival rate is 100%. Fig. 6(b) shows the throughput variations with different number of VNFs. We vary the number of VNF from 2^3 to 2^7 at the increment of double times. Plutus can reduce the throughput loss by 39.5% compared with Random.

5.2 Simulation

We also conduct extensive simulations to thoroughly evaluate our algorithms at scale.

Setup. In addition to the OpenNetVM experiments in our testbed, here we use a large-scale synthetic cellular network topology [28]. The topology can be divided into access layer, aggregation layer and core layer, where each layer includes a set of switches and servers. The access layer includes the base station clusters, each of which has 10 base stations interconnected into a ring. The aggregation and core layer are complete graphs with τ and τ^2 servers respectively. In the aggregation layer, the $\frac{\tau}{2}$ switches are connected to $\frac{\tau}{2}$ clusters in the access layer respectively. The remaining switches in the aggregation layer are connected to the switches in the core layer one by one. We generate different number of flows to measure the performance in our experiments. We run our algorithms on a server with Intel(R) Xeon(R) CPU E5-2650 and 64 GB memory. Each data point is an average of at least 30 runs.

Experiment results: We first investigate the total provisioning cost during service chain deployment generated by HW, Greedy and Plutus. In addition, we compare our algorithm against a branch and bound method that solves the program (1) optimally, denoted as OPT_1. We can see that in Fig. 7(a), as the number of flows increases, HW and Greedy yield significantly much provisioning cost, while that of Plutus is below 2.0×10^6 all the time and can achieve near optimal. Specifically, the provisioning cost for HW, Greedy, Plutus and OPT_1 is 1.25×10^7, 6.30×10^6, 1.92×10^6 and 1.90×10^6, when the number of flows is 5000. Furthermore, Plutus can



(a) Provisioning cost with different number of flows (b) Additive optimality gap with different number of flows (c) Provisioning cost with different number of switches (d) Provisioning cost with different length of service chain

Figure 7: Total provisioning cost comparison.

reduce the provisioning cost by 84.6% and 69.6% respectively, compared to HW and Greedy. This demonstrates that Plutus takes full advantage of NFV and reduce provisioning cost by flexibly deploying different VNFs.

Fig. 7(b) shows the additive optimality gap for Plutus and Greedy compared to OPT_1. For this simulation we vary the number of flows from 1000 to 5000. Intuitively, a larger additive optimality gap indicates more provisioning cost resulting from a worse solution. We can see that, as the number of flows increases, Greedy yields significantly larger optimality gap compared to Plutus, where Plutus can guarantee that the additive optimality gap is less than 1.3×10^4 and its provisioning cost is always less than 2.0×10^6, even though the number of flows become larger. The performance of our algorithms in large-scale networks is illustrated in Fig. 7(c), where the value of x-axis represents the number of switches and that of y-axis represents the provisioning cost. We fix the length of service chain at 10 in this setting. We can see that Plutus can reduce the provisioning cost by 86.0% and 72.4% compared to HW and Greedy respectively. Fig. 7(d) shows that the provisioning cost varies with the length of service chain.

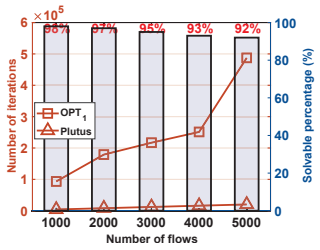


Figure 8: Solvable percentage for OPT_1 and numbers of iterations.

Finally we look at the percentage of solvability for OPT_1 and the number of solver iterations. In Fig. 8, the number of flows varies from 1000 to 5000 at the increment of 1000 for each run. We found that the number of solvable instances decreases when the number of flows becomes large. Specifically, when the number of flows is 5000, around 8% instances cannot be solved by standard solver. This demonstrates that OPT_1 cannot perfectly solve all instances, and it's going to get worse especially when the number of flows is large. Fig. 8 also shows the number of solver iterations for different schemes. We can see that the number of iterations for OPT_1 increases significantly than that for Plutus, when the number of flows become large. The convergence rate of Plutus in general can be faster than that of OPT_1.

6 CONCLUSION

We studied the problem of orchestrating service chain deployment in cellular networks. We propose a two-stage optimization framework and a set of algorithms to solve our problems. Evaluation results show that our algorithms can reduce the capital cost and increase the throughput.

ACKNOWLEDGMENT

This work was supported in part by the National Science and Technology Major Project of China under Grant Number 2017ZX03001013-003, the National Natural Science Foundation of China under Grant Numbers 61802172, 61602194, 61772265 and 61872178, the Collaborative Innovation Center of Novel Software Technology and Industrialization, Natural Science Foundation of Jiangsu Province under Grant Number BK20181251, the Fundamental Research Funds for the Central Universities under Grant 021014380079 and the Jiangsu Innovation and Entrepreneurship (Shuangchu-ang) Program.

REFERENCES

[1] 3gpp specifications. http://www.3gpp.org/.
[2] Cisco visual networking index. https://goo.gl/i6rd9e, 2016.
[3] Nokia. signaling is growing 50% faster than data traffic. http://goo.gl/uwnRiO, 2016.
[4] B. Addis, D. Belabed, M. Bouet, and S. Secci. Virtual network functions placement and routing optimization. In *CloudNet*, pages 171–177, 2015.
[5] A. Banerjee, J. Cho, E. Eide, J. Duerig, B. Nguyen, R. Ricci, J. E. van der Merwe, K. Webb, and G. Wong. Phantomnet: Research infrastructure for mobile networking, cloud computing and software-defined networking. *GetMobile*, 19(2):28–33, 2015.
[6] A. Banerjee, R. Mahindra, K. Sundaresan, S. K. Kasera, K. van der Merwe, and S. Rangarajan. Scaling the LTE control-plane for future mobile access. In *CoNEXT*, pages 19:1–19:13, 2015.
[7] A. Basta, W. Kellerer, M. Hoffmann, K. Hoffmann, and E. Schmidt. A virtual sdn-enabled LTE EPC architecture: A case study for s-/p-gateways functions. In *SDN4FNS*, pages 1–7, 2013.
[8] S. P. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.
[9] S. Chopra and M. R. Rao. The partition problem. *Math. Program.*, 59:87–115, 1993.
[10] R. Cohen, L. Lewin-Eytan, J. Naor, and D. Raz. Near optimal placement of virtual network functions. In *INFOCOM*, pages 1346–1354, 2015.

[11] W. Deng, M. Lai, Z. Peng, and W. Yin. Parallel multi-block ADMM with o(1 / k) convergence. *J. Sci. Comput.*, 71(2):712–736, 2017.
[12] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba. Elastic virtual network function placement. In *CloudNet*, pages 255–260, 2015.
[13] X. Jin, L. E. Li, L. Vanbever, and J. Rexford. Softcell: scalable and flexible cellular core network architecture. In *CoNEXT*, pages 163–174, 2013.
[14] J. Kempf, B. Johansson, S. Pettersson, H. Luning, and T. Nilsson. Moving the mobile evolved packet core to the cloud. In *WiMob*, pages 784–791, 2012.
[15] 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.
[16] Y. Li, Z. Yuan, and C. Peng. A control-plane perspective on reducing data access latency in lte networks. In *MOBICOM*, 2017.
[17] H. Moens and F. D. Turck. VNF-P: A model for efficient placement of virtualized network functions. In *CNSM*, pages 418–423, 2014.
[18] A. Mohammadkhan, K. K. Ramakrishnan, A. S. Rajan, and C. Maciocco. Cleang: A clean-slate EPC architecture and controlplane protocol for next generation cellular networks. In *CAN*, pages 31–36, 2016.
[19] M. Moradi, W. Wu, L. E. Li, and Z. M. Mao. Softmow: Recursive and reconfigurable cellular WAN architecture. In *CoNEXT*, pages 377–390, 2014.
[20] K. Nagaraj and S. Katti. Procel: smart traffic handling for a scalable software EPC. In *HotSDN*, pages 43–48, 2014.
[21] V. Nagendra, H. Sharma, A. Chakraborty, and S. R. Das. Ltextend: scalable support of M2M devices in cellular packet core. In *Proceedings of the 5th Workshop on All Things Cellular - Operations, Applications and Challenges, ATC@MobiCom*, pages 43–48, 2016.
[22] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker. E2: a framework for NFV applications. In *SOSP*, pages 121–136, 2015.
[23] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. G. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu, C. Kim, and N. Karri. Ananta: cloud scale load balancing. In *SIGCOMM*, pages 207–218, 2013.
[24] K. Pentikousis, Y. Wang, and W. Hu. Mobileflow: Toward software-defined mobile networks. *IEEE Communications Magazine*, 51(7):44–53, 2013.
[25] Z. A. Qazi, P. K. Penumarthi, V. Sekar, V. Gopalakrishnan, K. Joshi, and S. R. Das. KLEIN: A minimally disruptive design for an elastic cellular core. In *SOSR*, page 2, 2016.
[26] Z. A. Qazi, M. Walls, A. Panda, V. Sekar, S. Ratnasamy, and S. Shenker. A high performance packet core for next generation cellular networks. In *SIGCOMM*, pages 348–361, 2017.
[27] A. S. Rajan, S. Gobriel, C. Maciocco, K. B. Ramia, S. Kapur, A. Singh, J. Erman, V. Gopalakrishnan, and R. Jana. Understanding the bottlenecks in virtualizing cellular core network functions. In *LANMAN*, pages 1–6, 2015.
[28] N. Ron and T. Naveh. Wireless backhaul topologies: Analyzing backhaul topology strategies. In *Ceragon White Paper*, pages 1–15, 2010.
[29] M. R. Sama, L. M. Contreras, J. Kaippallimalil, I. Akiyoshi, H. Qian, and H. Ni. Software-defined control of the virtualized mobile packet core. *IEEE Communications Magazine*, 53(2):107–115, 2015.
[30] J. Sherry, P. X. Gao, S. Basu, A. Panda, A. Krishnamurthy, C. Maciocco, M. Manesh, J. Martins, S. Ratnasamy, L. Rizzo, and S. Shenker. Rollback-recovery for middleboxes. In *SIGCOMM*, 2015.
[31] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu. NFP: enabling network function parallelism in NFV. In *SIGCOMM*, pages 43–56, 2017.
[32] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.
[33] W. Zhang, J. Hwang, S. Rajagopalan, K. K. Ramakrishnan, and T. Wood. Flurries: Countless fine-grained nfs for flexible per-flow customization. In *CoNEXT*, pages 3–17, 2016.
[34] W. Zhang, G. Liu, W. Zhang, N. Shah, P. Lopreiato, G. Todeschi, K. K. Ramakrishnan, and T. Wood. Opennetvm: A platform for high performance network service chains. In *HotMiddlebox*, pages 26–31, 2016.

A A PROXIMAL JACOBIAN ADMM ALGORITHM

Now we reformulate the program (1) in order to apply ADMM. We first introduce slack variables α_l, β_p and γ_p to transform the inequality constraints (1a), (1b) and (1c) to equality constraints (3a), (3b) and (3c) required by ADMM. Second, all variables in the constraints of ADMM problem must be separable for each group of variables. To comply with this condition, we introduce auxiliary variables $\hat{x}_{i,l}$ and rewrite the constraints based on each pre-defined path. Towards this end, we add the original constraint (1d) and reformulate the program (1) to program (3) as follows.

$$\begin{aligned}
\text{minimize} \quad & \sum_{l\in L}\sum_{i\in SC} c_{i,l}\cdot \hat{x}_{i,l} && (3)\\
\text{subject to} \quad & R_l - \sum_{i\in SC} \hat{x}_{i,l}\cdot r_{i,l} - \alpha_l = 0, \quad \forall l, && (3a)\\
& b_p - d_f \cdot y_{f,p} - \beta_p = 0, \quad \forall f, p, && (3b)\\
& \sum_{l\in p} \hat{x}_{i,l} - y_{f,p} - r_p = 0, \quad \forall f, p, && (3c)\\
& (1d), \\
& \hat{x}_{i,l}, y_{f,p}, \alpha_i, \beta_p, \gamma_p \geq 0, \quad \forall i, l, f, p. && (3d)
\end{aligned}$$

The new program (3) is equivalent to the original program (1). The variables $\hat{x}_{i,l}$ in constraint (3c) ensure that the service chain i should be deployed onto server l (l is one of the nodes in path p) if and only if the flow f passes through the path p.

Let $\mathscr{L}_\rho$ be the augmented Lagrangian of program (3) with dual variables θ, φ and σ. i.e., introducing an extra $\mathscr{L}$-2 norm term into the objective:

$$\begin{aligned}
\mathscr{L}_\rho = & \sum_{l\in L}\theta_l\cdot\Delta_1 + \frac{\rho}{2}\sum_{l\in L}\Delta_1^2 + \sum_{f\in F}\sum_{p\in p(f)}\varphi_{f,p}\cdot\Delta_2 \\
& + \frac{\rho}{2}\sum_{f\in F}\sum_{p\in p(f)}\Delta_2^2 + \sum_{f\in F}\sum_{p\in p(f)}\sigma_{f,p}\cdot\Delta_3 + \frac{\rho}{2}\sum_{f\in F}\sum_{p\in p(f)}\Delta_3^2
\end{aligned}$$

where $\rho > 0$ is the penalty parameter. Note that $\mathscr{L}_0$ (when $\rho = 0$) is the standard Lagrangian for our problem. The reason why we introduce the penalty term is to speed up the convergence rate [8]. In addition, to simplify the notation, we introduce Δ_1, Δ_2 and Δ_3.

$$\begin{aligned}
\Delta_1 &= R_l - \sum_{i\in SC}\hat{x}_{i,l}\cdot r_{i,l} - \alpha_l \\
\Delta_2 &= b_p - d_f\cdot y_{f,p} - \beta_p \\
\Delta_3 &= \sum_{l\in p}\hat{x}_{i,l} - y_{f,p} - \gamma_p
\end{aligned}$$

Distributed 5-block ADMM. We initialize the variables α, β, γ, $\hat{x}$, y and multipliers θ, φ, σ to zero. For $t = 1, 2, \cdots$, repeat the following steps.

1. α-update: Each server l solves the following subproblem for obtaining α_l^{t+1}:

$$\begin{aligned}
\text{minimize} \quad & \theta_l^t\cdot\alpha_l + \frac{\rho}{2}\left(R_l - \sum_{i\in SC}\hat{x}_{i,l}^t\cdot r_{i,l} - \alpha_l\right)^2 \\
& + \frac{w}{2}\left(\alpha_l - \alpha_l^t\right)^2 && (5)\\
\text{subject to} \quad & \alpha_l \geq 0, \quad \forall l \in L. && (5a)
\end{aligned}$$

This per-server subproblem is a small-scale quadratic program and can be solved efficiently.

2. β-update Each generated p of the corresponding flow f solves the following subproblem for obtaining β_p^{t+1}:

$$\begin{aligned}\text{minimize} \quad & \sum_{f\in F}\varphi_{f,p}^t\cdot\beta_p+\frac{\rho}{2}\sum_{f\in F}\left(b_f-d_f\cdot y_{f,p}^t-\beta_p\right)^2\\ &+\frac{w}{2}\left(\beta_p-\beta_p^t\right)^2 \qquad (6)\\ \text{subject to} \quad & \beta_p\geq 0,\quad \forall p\in P(f). \qquad (6a)\end{aligned}$$

This subproblem can be solved by the standard solvers for quadratic program.

3. γ-update: Each generated p of the corresponding flow f solves the following subproblem for obtaining γ_p^{t+1}:

$$\begin{aligned}\text{minimize} \quad & \sum_{f\in F}\sigma_{f,p}^t\cdot\gamma_p+\frac{\rho}{2}\sum_{f\in F}\left(\sum_{l\in p}\hat{x}_{i,l}^t-y_{f,p}^t-\gamma_p\right)^2\\ &+\frac{w}{2}\left(\gamma_p-\gamma_p^t\right)^2 \qquad (7)\\ \text{subject to} \quad & \gamma_p\geq 0,\quad \forall p\in P(f). \qquad (7a)\end{aligned}$$

This subproblem can be solved by the standard solvers for quadratic program.

4. $\hat{x}$-update: Each server l solves the following subproblem for obtaining $x_{i,l}^{t+1}=(x_{1,l}^{t+1},x_{2,l}^{t+1},\cdots,x_{|SC|,l}^{t+1})$:

$$\begin{aligned}\text{minimize} \quad & \sum_{i\in SC}c_{i,l}\cdot\hat{x}_{i,l}+\theta_l^t\cdot\sum_{i\in SC}\hat{x}_{i,l}\cdot r_{i,l}\\ &+\frac{\rho}{2}\left(R_l-\sum_{i\in SC}\hat{x}_{i,l}\cdot r_{i,l}-\alpha_l^t\right)^2\\ &+\frac{w}{2}\left(\hat{x}_{i,l}-\hat{x}_{i,l}^t\right)^2 \qquad (8)\\ \text{subject to} \quad & \hat{x}_{i,l}\geq 0,\quad \forall l\in L. \qquad (8a)\end{aligned}$$

This subproblem can be solved by the standard solvers for quadratic program.

5. y-update: Each generated p of the corresponding flow f solves the following subproblem for obtaining $y_{f,p}^{t+1}=(y_{1,p}^{t+1},y_{2,p}^{t+1},\cdots,y_{|F|,p}^{t+1})$:

$$\begin{aligned}\text{minimize} \quad & \frac{\rho}{2}\sum_{f\in F}\left(b_p-d_f\cdot y_{f,p}-\beta_p^t\right)^2-\sum_{f\in F}\varphi_{f,p}^t\cdot d_f\cdot y_{f,p}\\ &+\frac{\rho}{2}\sum_{f\in F}\left(\sum_{l\in p}\hat{x}_{i,l}^t-y_{f,p}-\gamma_p^t\right)^2-\sum_{f\in F}\sigma_{f,p}^t\cdot y_{f,p}\\ &+\frac{w}{2}\left(y_{f,p}-y_{f,p}^t\right)^2 \qquad (9)\\ \text{subject to} \quad & \text{(1d)},\\ & y_{f,p}\geq 0,\quad \forall p\in P(f). \qquad (9a)\end{aligned}$$

This subproblem can be solved by the standard solvers for quadratic program.

6. Dual update: Each server j updates θ for the constraint (3a):

$$\theta_l^{t+1}=\theta_l^t+\iota\cdot\rho\cdot\left(R_l-\sum_{i\in SC}\hat{x}_{i,l}^{t+1}\cdot\hat{r}_{i,l}^{t+1}-\alpha_l^{t+1}\right) \qquad (10)$$

Each generated p of the corresponding flow f updates φ for the constraint (3b):

$$\varphi_{f,p}^{t+1}=\varphi_{f,p}^t+\iota\cdot\rho\cdot\left(b_p-d_f\cdot y_{f,p}^{t+1}-\beta_p^{t+1}\right) \qquad (11)$$

Each generated p of the corresponding flow f updates σ for the constraint (3c):

$$\sigma_{f,p}^{t+1}=\sigma_{f,p}^t+\iota\cdot\rho\cdot\left(\sum_{l\in p}\hat{x}_{i,l}^{t+1}-y_{f,p}^{t+1}-\gamma_p^{t+1}\right) \qquad (12)$$

where $\iota\cdot\rho$ is the step size for the dual update.

B PROOF OF THEOREM 4.2

Proof. We first introduce Theorem B.1 to facilitate our proof.

Theorem B.1. *[32] For any bipartite graph $B=(V,W,F)$, each extreme point of the feasible region has integer coordinates. Furthermore, given edge cost $c_{v,w}$, $(v,w)\in F$, and a feasible fractional solution $y_{v,w}$, $(v,w)\in F$, we can find, in polynomial time, a feasible integer solution $\hat{y}_{v,w}$ such that*

$$\sum_{(v,w)\in F}c_{v,w}\cdot\hat{y}_{v,w}\leq\sum_{(v,w)\in F}c_{v,w}\cdot y_{v,w}$$

Without loss of generality, we assume there are δ paths in path set P_f for flow f, i.e., the common node in the path set is at most δ. From Theorem B.1, we obtain,

$$\sum_{l\in L}\sum_{i\in SC}c_{i,l}\cdot\hat{x}_{i,l}\leq\delta\cdot\sum_{l\in L}\sum_{i\in SC}c_{i,l}\cdot x_{i,l}^*=\delta\cdot OPT_1 \qquad (13)$$

From the definition of complete matching, the provisioning resource Φ_l at each server l is

$$\Phi_l\leq\sum_{j=1}^{k_l}\hat{r}_l^j \qquad (14)$$

where $\hat{r}_l^j=\max\{r_{i,l}|u_l^j\in U\}$.

We give the upper bound of $\sum_{j=1}^{k_l}\hat{r}_l^j$ as following.

$$\begin{aligned}&\sum_{j=1}^{k_l}\hat{r}_l^j=\hat{r}_l^1+\sum_{j=2}^{k_l}\hat{r}_l^j\leq\hat{r}_l^1+\sum_{j=2}^{k_l}\sum_i u_{i,l}^{j-1}\cdot r_{i,l}\\ &\leq\hat{r}_l^1+\sum_{j=1}^{k_l}\sum_i u_{i,l}^j\cdot r_{i,l}\\ &=\hat{r}_l^1+\sum_i\sum_{j=1}^{k_l}u_{i,l}^j\cdot r_{i,l}\\ &=\hat{r}_l^1+\sum_i x_{i,l}\cdot r_{i,l}\end{aligned} \qquad (15)$$

From constraint (1a), both (16) and (17) hold.

$$\hat{r}_l^1\leq R_l \qquad (16)$$

$$\sum_i x_{i,l}\cdot r_{i,l}\leq R_l \qquad (17)$$

Combining (15), (16) and (17), we have the following inequation and conclude the proof.

$$\Phi_l\leq\sum_{j=1}^{k_l}\hat{r}_l^j\leq 2\cdot R_l \qquad (18)$$

□