This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2800003, IEEE Internet of Things Journal

PRSFC-IoT: A Performance and Resource Aware Orchestration System of Service Function Chaining for Internet of Things

Junxiao Wang, Heng Qi, Member, IEEE, Keqiu Li, Senior Member, IEEE, and Xiaobo Zhou

***Abstract*—Nowadays, Service Function Chaining (SFC) becomes more and more widespread and profound to implement flexible and economical virtual network infrastructures for the Internet of Things (IoT). With the benefits of SFC, the IoT service providers can steer massive traffic through a sequence of heterogeneous Virtual Network Function (VNF) instances based on their business logic. SFC is viewed as an attractive solution for building virtualized IoT-dedicated network. However, the SFC orchestration in IoT is still a challenge problem. Existing work usually focuses on the performance guarantee while ignoring the issue of resource idleness. To meet the sharp increase in IoT traffic amounts and the diversification of IoT traffic requirements, it is necessary to implement the performance and resource aware SFC orchestration system. Motivated by this, we propose a novel linear programming model and an effective approximation optimization algorithm for SFC orchestration, in order to achieve performance guarantee while avoiding resource idleness. Based on the proposed model and algorithm, a new prototype system named PRSFC-IoT is built upon OpenStack for online SFC orchestration. A large number of simulation experiments show that the PRSFC-IoT outperforms existing solutions for SFC orchestration in IoT.**

***Index Terms*—Internet of Things, Network Function Virtualization, Virtualized Network Functions, Service Function Chaining, Performance Guarantee, Resource Idleness.**

I. INTRODUCTION

As an emerging technology, the Internet of Things (IoT) is developing at an astonishing speed in recent years. Gartner estimates that the number of IoT devices is expected to be more than 20 billion in 2020, while IDC predicts that 10 percent of all data on the planet will be generated by IoT devices in 2020. The rapid growth in the number of IoT devices and the amount of data generated by these devices will cause a sharp increase in the network traffic, thereby posing a huge challenge to network infrastructures [1]–[4]. Existing work show that many in-network hardware middleboxes such as firewalls, Intrusion Prevention Systems (IPSs) and WAN optimizers are used in IoT applications to ensure security and improve performance [5], [6]. However, with more and more IoT applications emerging, traditional hardware middleboxes obviously cannot match the diversification of IoT traffic requirements [7], [8]. To tackle these challenges, Software Defined Networking (SDN) [9] and Network Function Virtualization (NFV) [10] are introduced to provide programmable network and flexible network resource management. Based on SDN and NFV, a IoT-dedicated network can be built to promote the development of IoT [11].

As a typical and successful case of SDN and NFV, the Service Function Chaining (SFC) draws more and more attentions from the industry and the academia. In IoT-dedicated network, the customers can use SFC service to define a sequence of heterogeneous Virtual Network Function (VNF) instances (e.g. IDS/IPS, firewall instances) for processing the massive data flows belonging to their applications flexibly [12], [13]. These service chains can make the service network of IoT more efficient, more scalable and more economical. The massive data flows of the IoT terminals is repaired, filtered, compressed by the SFC service, and the aggregated data is sent to the cloud data center at the back end, which not only can take full advantage of existing computing power, enhancing service availability and reliability, but also can save valuable bandwidth.

However, it is a big challenge to give effective solutions for integrating SFC orchestrations with special requirements of IoT scenarios. On one hand, performance guarantee of IoT network is a key issue of the SFC orchestration. When deploying SFC in IoT environment, the performance guarantee in network and the capacities of VNF instances should be first considered. It includes maximum tolerated deadline and peak packet rate, which are claimed by the IoT tenants of chains. Once the guarantee is violated, the SFC services with low network performance might cause serious response timeout or packet loss, thereby affecting service qualities of applications. On the other hand, the capacities of VNF instances also concerns closely in the SFC orchestration since resource efficiency of IoT network is another target. A reasonable SFC orchestration should occupy as few resource as possible to meet required performance and avoid producing any idleness of occupied resource, especially for crucial computation resource. In fact, there are always a few VNF instances become the bottlenecks in the SFC processing while other instances with more computation resources sit idle, leading to an ineffective SFC service with resource idleness.

For the problem of SFC orchestration, many previous

This work was supported in part by the National Key Research and Development Program of China No. 2016YFB1000205, in part by the State Key Program of National Natural Science of China under Grant 61432002, in part by the NSFC under Grants 61772112, 61672379, U1701263, 61702365, 61425002, 61751203, in part by the Dalian High-level Talent Innovation Program under Grant 2015R049.

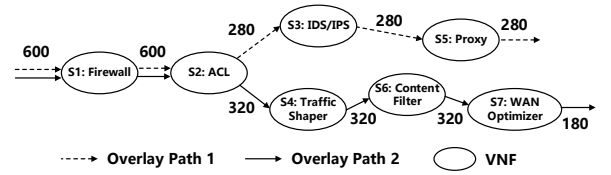
J. Wang, H. Qi and K. Li are with the School of Computer Science and Technology, Dalian University of Technology, China. E-mail: wangjunxiao.dalian@gmail.com, hengqi@dlut.edu.cn, likeqiu@gmail.com.

X. Zhou is with the School of Computer Science and Technology, Tianjin University, China.

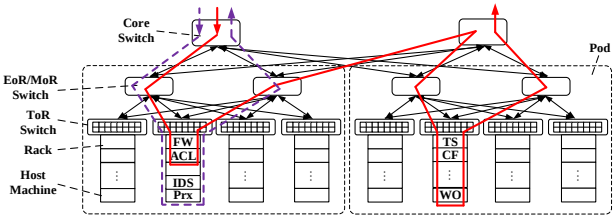
Corresponding authors: Heng Qi and Keqiu Li.

2327-4662 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2800003, IEEE Internet of Things Journal



(a) The logical orchestration of a service chain



(b) The physical deployment of a service chain

Fig. 1. An example of service chains in the cloud data center.

research efforts have been made. Sekar et al. [14] proposed a dynamic service chain deployment approach, but not considered service performance guarantee. Motivated by this work, Gember et al. [15] proposed a reactive service chain deployment scheme to adapt with the bottleneck of service performance. Li et al. [16] further proposed a proactive service chain orchestration system to reserve the deadline of service request. However, there is little work which focuses on resource usage. To overcome this drawback, we first propose a joint optimization solution which takes both performance guarantee and resource usage into consideration. We propose an effective and efficient method of SFC orchestration for the IoT environment to achieve network performance guarantee in chain level while avoiding resource idleness. Moreover, we also build a novel SFC orchestration system named PRSFC-IoT to evaluate our method. The main contributions of this paper are as follows:

(1) We formulate the problem of performance and resource aware SFC orchestration with an optimization model. Given a set of service chains in which each consists of some network functions, we propose an approximation optimization algorithm to achieve the guarantees of deadline and packet rate while avoiding resource idleness.

(2) To evaluate our algorithm, we built a prototype system named PRSFC-IoT upon OpenStack for online SFC orchestration. Compared to existing greedy based solution, the PRSFC-IoT has significant advantages in deadline satisfying, computation resource effective utilization, deployment acceptance rate and delay guarantee.

The rest of the paper is organized as follows. We introduce the model of SFC and the design of PRSFC-IoT in §II and §III, respectively. Then, we describe the implementation and evaluation of PRSFC-IoT prototype system in §IV and §V. We give a review of related work in §VI. Finally, we give a conclusion of this paper in §VII.

II. SFC MODEL

Preliminary: According to the networking revolution driven by network function virtualization (NFV) [10], elastic scaling and pooling services are enabled to be deployed in a more agile and cost-effective fashion than traditional hardware based appliances in the cloud. These network services are referred to as Virtualized Network Function (VNF) and are migrated from costly hardware appliances to virtualized instances deployed on generic servers (e.g., x86 based systems) [6], [17]. This service architecture increases service flexibility and scalability for IoT, as these virtualized services can be instantiated and scaled on demand using cloud scaling technologies. For the scenario of NFV, virtualized network services are rarely used in isolation, since dedicated network management must consider the chain level service. The concept of Service Functions Chaining (SFC) seems to remagnetize the academics and industry [12]. In practice, the NFV-based in-the-cloud SFC orchestrations are typically structured as logical network service chains, where a given flow is processed by a sequence of heterogeneous VNF instances that lie on critical forwarding paths. Since these VNF instances are on the data path, their network footprints more critically impact service performance and resource usage, respectively, compared to common virtual compute services. The two aspects are also the key challenges of SFC orchestration under the IoT environment since the orchestrated network services need to coordinate the requirements of IoT applications on low-latency response, high-throughput processing and cost-effective resource usage. Service Function Chaining is the technique that enables organizations to flexibly orchestrate custom virtual network functions into a service chain according to their particular needs. Predefined rules would be installed into each VNF to execute particular network function, and packets would be steered through different VNFs in particular order [13].

Fig.1(a) shows an example of logical service chains which consists of two subchains. Packets are divided into two paths by service classifier according to their types of service. VNFs on each subchain are orchestrated into a service to accomplish a traffic processing function. As we known, the perceptual layer is at the most front-end of information collection and object identifying, which plays a fundamental role in the IoT. Subchain 1 as the VPN gateway can accomplish secure access from the perceptual layer to the network layer of IoT, e.g. GPRS, Internet and LTE. With the help of subchain 2, the data traffic collected by the perceptual layer devices can be accelerated for higher speed transmition.

Fig.1(b) shows a practical cloud datacenter-based deployment of the logical service chain of Fig.1(a), in which the datacenter has two disjoint pods that are connected via core switches. Each pod consists of a number of EoR/MoR switches and their connected racks, in which each rack consists of a ToR switch and several host machines. We model the collection of host machines within a rack m as an aggregated node with CPU time period cpu_m, where CPU time period is the amount of time slice allocated to one service by CPU, and cpu_m is the total amount of CPU time period in the rack m. The overlay path e has a capacity of $bandwidth_e$. It connects a source point-of-presence (POP) with a destination POP through EoR/MoR switches and core switches. All pathes are implemented with overlay technologies, where the source POP and the destination POP can be located in any rack

2327-4662 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2800003, IEEE Internet of Things Journal

inside of cloud, or even outside of cloud. $bandwidth_e$ is the total available bandwidth in this path, adapting with practical underlay networks.

In actual SFC orchestration, each logical service chain is often orchestrated into at least one physical chain instance. This helps adapt to dynamically scaling in or scaling out, and makes it flexible for timely assigning workload of service chains into various chain instances. Therefore, PRSFC-IoT considers performance guarantee and resource idleness in the view of chain instance instead of logical service chain.

A. Performance Guarantee of Chain Instances

As shown in Fig.2, given a service chain with a set of executable NFV services, we denote the worst-case execution time (WCET) [18] by $wcet_s$ that a service s takes at most to process a packet. $wcet_s\,(P_s)$ indicates the corresponding $wcet_s$ under assigned CPU time period P_s. In general, given longer CPU time period P_s would bring lower $wcet_s(P_s)$, in which if P_s is doubled, corresponding $wcet_s(P_s)$ would halve.

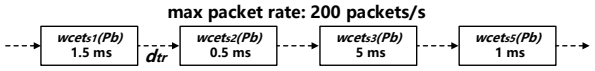


Fig. 2. Performance guarantee problem.

Unlike common virtual compute services, resource consumption of service chains can be quite diverse and workload dependent [19], e.g. a high packet rate at peak time while a low rate at trough time. The profile of a logical service chain i is defined as $chain_i(T_i,<s_i^1, s_i^2, ..., s_i^{c_i}>, d_i)$, where T_i is a set of packet rate demands on different time periods; $<s_i^1, s_i^2, ..., s_i^{c_i}>$ is the service composition with length c_i that packets need to be routed through; d_i is the longest tolerable packet granularity end-to-end delay. Specially, the deadline of a chain instance contains both a delay of VNF processing and a delay of traffic transmission.

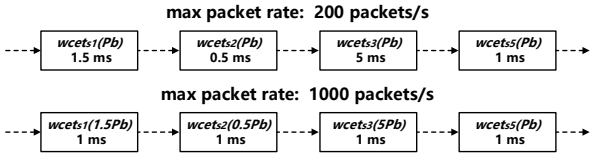


Fig. 3. Resource idleness problem.

For each chain instance of $chain_i$, towards tenant deadline guarantee, we restrict that

$$d_{tr}(c_i+1)+\sum_{w=1}^{c_i} wcet_{s_i^w}(P_{s_i^w}) \leqslant \beta d_i \ ,$$

where d_{tr} is the delay of traffic transmission, β is a fraction close to 1, indicating the tolerance coefficient of tenant deadline; and towards non-blocking pipelining processing guarantee in which each packet is processed in series and the total processing rate remains stable, we define the max packet rate of incoming request as $\frac{1000}{\max\limits_{1\leqslant w\leqslant c_i} wcet_{s_i^w}(P_{s_i^w})}$ packets/s, which indicates the maximal allowed workload in this chain instance.

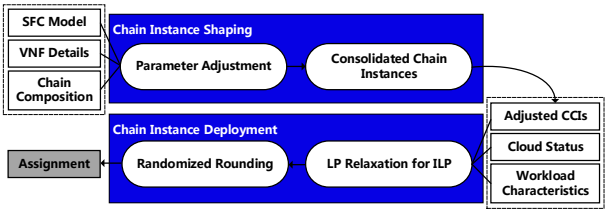


Fig. 4. The overview of PRSFC-IoT.

In practical running, the defined max packet rate of chain instances can provide persistent traffic processing with real-time and strict performance guarantee in the granularity of packet.

B. Resource Idleness of Chain Instances

On the basis of hypothesis in Section II(A), as shown in Fig.3, we find that different assignment for CPU resource can lead to different degrees of idleness condition in chain level.

For example, there are two kinds of chain instance implementation against one same logical service chain. In the first kind of resource allocation case along with 200 packets/s max packet rate and 4 units of CPU resource, we need 20 units of CPU time period to meet up with the max packet rate of 1000 packets/s, while in the second case, we only need 8 units. It means that 60% of assigned CPU resource in the first case sits idle. To indicate how idle a chain instance of $chain_i$ is, we define $\frac{\left(\sum\limits_{w=1}^{c_i} P_{s_i^w}\right)\left(\max\limits_{1\leqslant w\leqslant c_i} wcet_{s_i^w}\left(P_{s_i^w}\right)\right)}{1000P_b}$ as the quantitative criteria of idleness, where $\sum\limits_{w=1}^{c_i} P_{s_i^w}$ is the total amount of CPU time period in this chain instance. P_b is the reference value of P_s and we assume that $wcet_s(P_b)$ is known in a common scenario.

***Theorem* I:**

For CPU resource allocation in chain instances of $chain_i$, when and only when $P_{s_i^1} : P_{s_i^2} : \cdots : P_{s_i^{C_i}} = wcet_{s_i^1}(P_b) : wcet_{s_i^2}(P_b) : \cdots : wcet_{s_i^{c_i}}(P_b)$, the idleness is the minimum.

***Proof*:**

Based on the known conditions, it can be concluded that: for $\forall w \in [1, c_i]$, $\frac{P_{s_i^w}}{P_b} = \frac{wcet_{s_i^w}(P_b)}{wcet_{s_i^w}(P_{s_i^w})}$. On account of $P_{s_i^1} : P_{s_i^2} : \cdots : P_{s_i^{C_i}} = wcet_{s_i^1}(P_b) : wcet_{s_i^2}(P_b) : \cdots : wcet_{s_i^{c_i}}(P_b)$, it can be concluded that $wcet_{s_i^1}(P_{s_i^1}) = wcet_{s_i^2}(P_{s_i^2}) = \cdots = wcet_{s_i^{c_i}}(P_{s_i^{c_i}})$. In this case, the idleness indicator of the chain instance can be expressed as $\frac{\sum\limits_{w=1}^{c_i} wcet_{s_i^w}(P_b)}{1000}$. So the uniqueness is proven.

As for optimality, since $\frac{\left(\sum\limits_{w=1}^{c_i} P_{s_i^w}\right)\left(\max\limits_{1\leqslant w\leqslant c_i} wcet_{s_i^w}\left(P_{s_i^w}\right)\right)}{1000P_b}$ $= \sum\limits_{w=1}^{c_i} \frac{\left(wcet_{s_i^w}(P_b)\right)\left(\max\limits_{1\leqslant w\leqslant c_i} wcet_{s_i^w}\left(P_{s_i^w}\right)\right)}{1000wcet_{s_i^w}\left(P_{s_i^w}\right)}$, and due to that $\frac{\max\limits_{1\leqslant w\leqslant c_i} wcet_{s_i^w}\left(P_{s_i^w}\right)}{wcet_{s_i^w}\left(P_{s_i^w}\right)} \geqslant 1$. Therefore, we can logically find that $\frac{\left(\sum\limits_{w=1}^{c_i} P_{s_i^w}\right)\left(\max\limits_{1\leqslant w\leqslant c_i} wcet_{s_i^w}\left(P_{s_i^w}\right)\right)}{1000P_b} \geqslant \frac{\sum\limits_{w=1}^{c_i} wcet_{s_i^w}(P_b)}{1000}$. So the optimality is also proven.

2327-4662 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2800003, IEEE Internet of Things Journal

Algorithm 1: iterative consolidation for chain instances

Input: chain instances with the standardized form, $< wcet_{s_i^1}(P_b), wcet_{s_i^2}(P_b), ..., wcet_{s_i^{c_i}}(P_b) >$.
Output: profiles of instances with the minimal idleness, $< wcet_{s_i^1}(P_b), wcet_{s_i^2}(P_b), ..., wcet_{s_i^{c_i}}(P_b) >$, Max Packet Rate $rate_i$.
$z, w \in \{1, 2, ..., c_i\}$;
$Resource_{MaxUnit}$ is the predefined threshold of $P_{s_i^w}$;
while *not at end of the list of chain instances* **do**
  $instance_i$ = getCurrentInstance( );
  foreach s_i^w *in* $instance_i$ **do**
    $wcet_{s_i^w}(P_{s_i^w}) = \frac{\beta d_i - (c_i+1)d_{tr}}{c_i}$;
    $P_{s_i^w} = \frac{wcet_{s_i^w}(P_b)c_i}{\beta d_i - (c_i+1)d_{tr}} P_b$;
    $< s_i^z, s_i^{z+1} >$ = getPairWithMinSumP($instance_i$);
    if $\sum_{z \leqslant w \leqslant z+1} P_{s_i^w} \geqslant Resource_{MaxUnit}$ **then**
      $rate_i = \frac{1000 c_i}{\beta d_i - (c_i+1)d_{tr}}$;
      break;
    else
      consolidateChain($< s_i^z, s_i^{z+1} >$);
      $c_i = c_i - 1$;
    end
  end
end

III. DESIGN OF PRSFC-IoT

To tackle the problem that jointly involves both the performance guarantee and idleness avoidance, we propose a solution called PRSFC-IoT to orchestrate the IoT-dedicated network services in an effective and efficient way. Fig.4 shows the overview of PRSFC-IoT. It contains two interacting stages: (i) chain instance shaping stage, which consists of the parameter adjustment and the chain instance consolidation based chain instance optimization for obtaining deployable consolidated chain instances; and (ii) chain instance deployment stage, which is composed by the LP relaxation and randomized rounding based chain instance deployment for deploying consolidated into cloud infrastructure properly.

The stage of chain instance shaping takes three sets of inputs: (1) the SFC model proposed in Section II, (2) a set of detailed profiles of different type of VNFs, and (3) a set of constitution of different service chains, in which different service chains may not arrive to be deployed at unique time slot. These inputs are together processed through the stage of chain instance shaping and eventually output as a set of deployable and consolidated chain instances.

A. Chain Instance Shaping

In this stage, PRSFC-IoT focuses on how to transform an original chain instance into a deployable one.

1) Parameter Adjustment: During the practical chain processing, besides the factors in our proposed service chain model, there are still a few factors like packet size changing may exert influence. For instance, when traffic goes through VNFs with service functions like compressing/decompressing or encryption/decryption, the size of the output packet can differ from that of the input packet. Towards this, we need to adjust the parameters which related to these factors into one standardized form to eliminate influence.

TABLE I
NOTATIONS

Symbol	Meaning
$e_{i,w,j,k}$	The overlay path connecting s_i^w to s_i^{w+1} in $chain_i$ whether is assigned into the underlying network between $rack_j$ and $rack_k$
b_i	The bandwidth demand of $chain_i$; $b_i = size_{init} \cdot rate_i$, where $size_{init}$ is the packet size of incoming request from chain instances of $chain_i$
$B_{j,k}$	The available bandwidth capacity between $rack_j$ and $rack_k$
$cpu_{i,w}$	The cpu resource demand of s_i^w in $chain_i$; $cpu_{i,w} = P_{s_i^w}$
$CPU_{j/k}$	The available cpu resource capacity of $rack_j$ or $rack_k$
$\mathbb{I}$	The set of service chains which to be deployed
$\mathbb{M}$	The set of available racks; $rack_j$ and $rack_k \in \mathbb{M}$

Algorithm 2: ILP for the deployment of chain instances

$$\min \ MaxUtilizationAmongRacks = \lambda$$

$$\text{s.t.} \begin{cases} \forall j,k \in \mathbb{M}, \sum_i \sum_w e_{i,w,j,k} \cdot b_i \leqslant \lambda B_{j,k}; \\ \forall j,k \in \mathbb{M}, \sum_i \sum_w e_{i,w,j,k} \cdot cpu_{i,w} \leqslant \lambda CPU_{j/k}; \\ \forall i,w,\forall k \in \mathbb{M}, \sum_j e_{i,w,j,k} = \sum_j e_{i,w+1,k,j}; \\ \forall i,w, \sum_j \sum_k e_{i,w,j,k} = 1; \\ \lambda \in (0,1], i \in \mathbb{I}, j,k \in \mathbb{M}; \\ w \in \{1,2,...,c_i\}, e_{i,w,j,k} \in \{0,1\}. \end{cases}$$

We formulate $(\prod_{z=1}^{w-1} \varphi_{s_i^z} \cdot \sigma_{s_i^z}) wcet_{s_i^w}(P_b)$ as the standardized form of $wcet_{s_i^w}$, where $\sigma_{s_i^z} = L_{s_i^z}^{out}/L_{s_i^z}^{in}$ and $\varphi_{s_i^z} = R_{s_i^z}^{out}/R_{s_i^z}^{in}$ are the relative change ratio on the z_{st} hop of the packet size and the packet rate, respectively. $L_{s_i^z}^{in}$ and $R_{s_i^z}^{in}$ is the packet size and the packet rate before chain processing, $L_{s_i^z}^{out}$ and $R_{s_i^z}^{out}$ is the packet size and the packet rate after chain processing, respectively.

With the standardized form of $wcet_{s_i^w}$, we can eliminate the influence caused by the relative change of packet size and packet rate in per-hop chain processing, and make it feasible to conduct chain instances consolidation.

2) Chain Instance Consolidation: On the basis of parameter adjustment, we present an algorithm named iterative consolidation (IC). As shown in Algorithm 1, it's used for chain instance consolidation. IC is an iterative procedure which contains three parts: i) Given a set of chain instances with the standardized form, IC first optimizes the resource idleness of instances according to the method proposed in Theorem I. ii) Then, for each instance, IC finds out two adjacent VNFs from that instance, i.e., s_i^z and s_i^{z+1} with the minimal $\sum_{w=z}^{z+1} P_{s_i^w}$, and consolidate the two VNFs into a single one. Thus, the network transmission delay between the original

2327-4662 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2800003, IEEE Internet of Things Journal

Algorithm 3: randomized-rounding for LP relaxation

Input: fractional solution $e_{i,w,j,k}*$ to chain deployment; i.e, the output of the LP relaxation for ILP.
Output: feasible solution $e_{i,w,j,k}{}'$ to chain deployment; i.e, the solution to standard ILP for chain deployment.
$E \ni e_{i,w,j,k}*$;
let $E' \leftarrow \{\}$ (E' contains the decided $e_{i,w,j,k}{}'$.);
while E *is not empty* **do**
  foreach $e_{i,w,j,k}*$ *in* E **do**
    let $E \leftarrow E - \{e_{i,w,j,k}*\}$;
    $e_{i,w,j,k}{}' \sim B(e_{i,w,j,k}{}', \min(e_{i,w,j,k}*, 1))$;
    if $\forall i, w, \forall k \in \mathbb{M}, \sum_j e_{i,w,j,k}{}' \neq \sum_j e_{i,w+1,k,j}{}'$ **then**
      let $E \leftarrow E + \{e_{i,w,j,k}*\}$;
      continue;
    else
      if $\forall i, w, \sum_j \sum_k e_{i,w,j,k}{}' \neq 1$ **then**
        let $E \leftarrow E + \{e_{i,w,j,k}*\}$;
        continue;
      else
        let $E' \leftarrow E' + \{e_{i,w,j,k}{}'\}$;
      end
    end
  end
end
if $\lambda(e_{i,w,j,k}{}') \in (0, 1]$ **then**
  accept;
end

two VNFs can be eliminated by aggregating the fragmented VNFs. iii) At the end of each iteration, IC recomputes a new profile of resource allocation for that chain instance, which possess a optimized resource idleness. Repeating the iteration procedure mentioned above, desirable chain instances can be obtained before the iterations fall into the stopping rules, i.e., the amount of assigned CPU time period in one consolidated VNF exceeds the predefined threshold.

The stage of chain instance deployment takes three sets of inputs: (1) the detailed status of cloud infrastructure, (2) a series of consolidated chain instances produced in Section III(A), and (3) the service chains' time-dependent workload characteristics. These inputs are together processed through the stage of chain instance deployment and eventually output as a set of assignment of chain instances.

B. Chain Instance Deployment

In this stage, PRSFC-IoT intends to deploy the consolidated chain instances into underlying cloud infrastructure.

1) ILP and LP Relaxation: Towards this, we first formulate the deployment of consolidated chain instances as an Integer Linear Program (ILP) model. As shown in Algorithm 2, the decision variable of the program problem is $e_{i,w,j,k}$, which decides the overlay path connecting s_i^w to s_i^{w+1} in $chain_i$ whether is assigned into the underlying network between $rack_j$ and $rack_k$. Note that overlay paths are end-to-end links that connect two network functions of a logical service chain and the underlay network contains the detailed physical routing capacities of relative overlay paths between racks. Minimizing λ is the optimization objective of the program problem for balancing the workload, essentially we intend to minimize the maximum of resource utilization ratio among available different racks, in which these available racks mean the specific racks could be used for chain instance deployment in entire cloud infrastructure and the resource includes both the cpu resource inside racks and the bandwidth resource between racks. In addition, we also take the restrictive conditions e.g. available capacity of CPU and bandwidth into consideration. The detailed meanings of notations used in the program problem are shown in table I.

Moreover, different service chains might encounter various workloads during different periods since the total traffic rate of each IoT tenant captures a bursty nature, and almost these workloads are regular and periodic. Therefore, before setting up a new deployment, the appropriate demands for the deployed chain instances need to be determined in advance. We define $\sum_f rate_i^f \geqslant Load_i^t$ as a supplementary restriction of demands, where $rate_i^f$ is the max packet rate provisioned by one fragment instance f of the logical service chain i. $Load_i^t$ is the demand of total workloads of chain i during the time period t. Towards workloads scaling in or scaling out, PRSFC-IoT has to timely increase or decrease the amount of chain instances, so as to constantly meet up with demands of specified time period.

Due to integer programming is NP-hard, specially, 0-1 ILP of this model that unknown decision variables are binary and all restrictions are satisfied, is one of Karp's 21 NP-complete problems [20], in which directly searching in solution space would be inefficient. We thus relax the ILP to a Linear Program (LP) by removing the integrality constraints, allow the decision variables to take on non-integral values, and solve the linear program, then round and obtain a set of LP solutions to the original ILP problem. In particular, we let the decision variables $e_{i,w,j,k}$ to be chosen within the range $[0, 1]$ instead of integers $\{0, 1\}$, and compute the solutions of the LP version.

2) Randomized Rounding: By the mean of relaxing ILP to LP, we can obtain a set of optimal fractional solutions $e_{i,w,j,k}* \in E_{solution}$. To round the fractional solution $e_{i,w,j,k}*$ to the integer solutions $e_{i,w,j,k}{}'$ of the ILP, for each overlay path connecting s_i^w to s_i^{w+1} in $chain_i$, we choose to assign it into the underlying network between $rack_j$ and $rack_k$ with the probability $\min(e_{i,w,j,k}*, 1)$.

As shown in Algorithm 3, through a series of rounding operations, we can get a set of feasible integer solution $e_{i,w,j,k}{}'$ to support decisions in the deployment of chains. We can show that this LP with randomized rounding technique would produce a solution with objective value $\lambda(e_{i,w,j,k}{}')$ increased by a factor of at most $1+\alpha$, for some small $\alpha > 0$, compared to $\lambda(e_{i,w,j,k}*)$ from the original LP fractional solutions.

***Proof of the Approximation*:**

Since the expectation of each $e_{i,w,j,k}{}'$ is $e_{i,w,j,k}*$. By linearity of expectation, we can find that the expectation of $\lambda(e_{i,w,j,k}{}')$ is $\lambda(e_{i,w,j,k}*)$.

Therefore, by Markov's inequality that $P(|x| \geqslant a) \leqslant \frac{E(|x|)}{a}$

2327-4662 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2800003, IEEE Internet of Things Journal

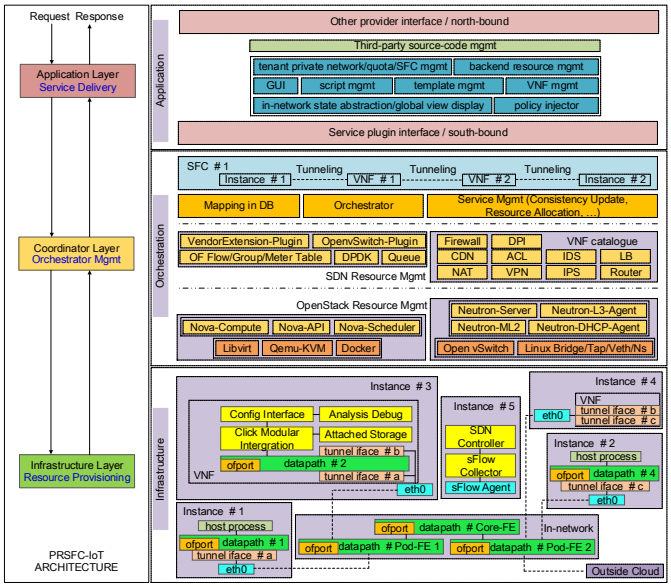


Fig. 5. The architecture overview of PRSFC-IoT.

while $a \geqslant 0$, we can substitute $|x| = |\lambda(e_{i,w,j,k}{}')|$ and $a = (1+\alpha)\,|\lambda(e_{i,w,j,k*})|$ into the expression to get

$$P\left(\frac{|\lambda(e_{i,w,j,k}{}')|}{|\lambda(e_{i,w,j,k*})|} \geqslant (1+\alpha)\right) \leqslant \frac{E\left(|\lambda(e_{i,w,j,k}{}')|\right)}{(1+\alpha)\,|\lambda(e_{i,w,j,k*})|}\ .$$

Since $E\left(|\lambda(e_{i,w,j,k}{}')|\right) = |\lambda(e_{i,w,j,k*})|$, we can infer that

$$\left(\frac{|\lambda(e_{i,w,j,k}{}')|}{|\lambda(e_{i,w,j,k*})|} \geqslant (1+\alpha)\right) \leqslant \frac{E\left(|\lambda(e_{i,w,j,k}{}')|\right)}{(1+\alpha)\,E\left(|\lambda(e_{i,w,j,k}{}')|\right)}$$

and then we can simplify the inequality as:

$$P\left(\frac{|\lambda(e_{i,w,j,k}{}')|}{|\lambda(e_{i,w,j,k*})|} \geqslant (1+\alpha)\right) \leqslant \frac{1}{(1+\alpha)}\ ,$$

$$P\left(|\lambda(e_{i,w,j,k}{}')| \geqslant (1+\alpha)\,|\lambda(e_{i,w,j,k*})|\right) \leqslant \frac{1}{(1+\alpha)}\ .$$

For some small $\alpha > 0$, the probability that a solution with objective value $\lambda(e_{i,w,j,k}{}')$ increased by a factor of at most $1+\alpha$ is less than $\frac{1}{1+\alpha} \approx 1$. Therefore, with positive probability, the approximation of this algorithm is proven.

IV. PROTOSYSTEM OF PRSFC-IoT

In order to evaluate our approach, we have built a protosystem for PRSFC-IoT, in which several techniques are integrated for the top-down SFC orchestration. The implementation of the protosystem of PRSFC-IoT consists of three parts: (1) the hierarchical design of architecture; (2) the encapsulation formation for steering traffic; (3) the policy for CPU resource allocation inside of racks.

A. Hierarchical Design of Architecture

Fig.5 shows the architecture overview of PRSFC-IoT, consisting of three hierarchical components:
(1) Infrastructure layer, which is responsible for provisioning resource from underlying cloud infrastructure, e.g. creating VNF instances with assigned CPU time period and creating virtual networks connecting VNF instances with assigned bandwidth. In the aspect of resource provisioning, our protosystem makes use of well-known, widely used tools, such as OpenStack [21], OpenFlow [22], DPDK [23], Click [24], and sFlow [25], integrated into a common architecture. Specially, OpenStack platform is used as a framework to operate cloud infrastructure, OpenFlow and DPDK are added for obtaining superiority on controllability and performance; Click helps modularize service functions of VNFs; sFlow is responsible for the task of sniffing the real-time in-network state.
(2) Orchestration layer, which is responsible for service chain orchestration, e.g. managing available pooled resource including computation resource, network resource and storage resource. The orchestration engine realizes the algorithms of PRSFC-IoT that transforms logical service chains into deployable consolidated chain instances and gives the feasible solutions to the deployment.
(3) Application layer, which is responsible for receiving requests and configs from IoT tenants or organizations of service chains, e.g. updating the mappings between the capacity of physical cores and the wcet; updating profiles of service chains and constitutions of VNFs.

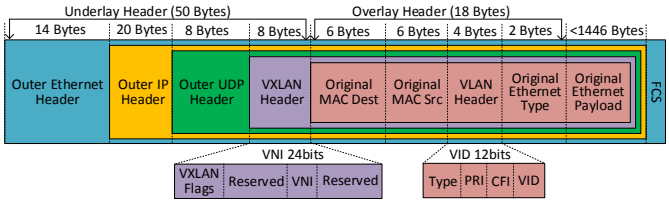


Fig. 6. The encapsulation format for steering traffic.

B. Encapsulation Formation for Steering Traffic

As for the implementation of steering chain traffic, instead of directly using Network Service Header (NSH) [26], we adopt a more backward compatible way, since NSH as a private software patch attached to Open vSwitch [27] so far is completely incompatible with the most of off-the-shelf in-network hardwares or softwares. As shown in Fig.6, we choose to encapsulate the original packets of traffic with a customized nested header.

Specially, the ethernet payload of original traffic would be encapsulated into the data part of a standard ethernet packet. In front of the data part, a classical VLAN tag instead of an ultra-new NSH tag is attached for distinguishing different next-hop VNFs during chain processing. To ensure clearer understanding the spirit of our design, we would illustrate with a example. Assuming there is aggregated traffic destining for two different subchains both with two hops, the first hop of the two subchains share the same VNF, and the second hop respectively is an isolated VNF. In this case, the aggregated traffic would be tagged with different VIDs before steered through the shared VNF, in order to distinguish which isolated VNF is the accurate next hop. A VLAN tag with the 12-bits identification field named VID can help us distinguish at most 4094 different next hop VNFs.

2327-4662 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2800003, IEEE Internet of Things Journal

Outside of the encapsulated header with VLAN tag, we further add an encapsulation of VXLAN [28]. The technique of VXLAN can offer support for distinguishing a huge number of different service chains (in fact, more than 1.6×10^7) with the 24-bits identification field named VNI. Besides, VXLAN transmission is also a kind of overlay path implementation based on a reduced UDP protocol, compatible with dynamic traffic steering as long as the next-hop IP is reachable. In this way, the traffic which belongs to different service chains can be tagged with different VNIs. A specific VNI and VID as well can determine the only one subchain, in which the bandwidth management of underlay network can be accomplished in a granularity of subchain.

In addition, it is worth noting that the encapsulated packet in transit might be sliced into a number of IP fragmentation by any device located on the overlay path, and the VTEP devices of VXLAN might drop the sliced IP fragmentation by mechanism, eventually result in packet loss of original traffic. So we need to make sure the bad events would not happen in our case. Assuming that the MTUs of all devices located on the overlay path is by default set as 1500, which means the data part of the encapsulated outer IP packet must be within 1480. According to the defined encapsulation formation, we can calculate the size of original ethernet payload as at most 1446. Therefore, at the source point of traffic incoming, IoT tenants or organizations have to limit the MTUs within 1446, so as to avoid the bad influence caused by packet fragmentation.

C. Policy for CPU Resource Allocation inside of Racks

On the basis of the chain instance deployment in Section III(B), we can determine each chain instance is assigned into which rack m with the total CPU time period cpu_m. But in practical running, the chain instances must locate in the host machines inside racks. Towards this, the part of chain instances inside each rack need to be orchestrated into various host machines which belong to this rack, and assign the chain instances with appropriate CPU time period.

In this respect, we formulate the orchestration between the chain instances and the host machines inside each rack as bin packing problem. We use the general method of bin packing to pack these chain instances into corresponding host machines. Besides, the requests of service chain deployment might arrive in real time, when most of host machines inside one rack have encountered constrained CPU time period, the original orchestration inside this rack need to be repacked.

As for assigning the chain instances inside each host machine with appropriate CPU time period, PRSFC-IoT interacts with Linux CGroup [29] to divide the CPU time period. The functions of controlling CPU resource in CGroup can be basically decomposed into two subsystems: `cpu` and `cpuset`, where the subsystem `cpu` is used for controlling the scheduling of different VNF processes and setting the relative weight, thus certain CPU resource can be occupied by specific VNF process; the subsystem `cpuset` is used for controlling the binding between CPU cores and VNF processes. The relative weight could be set via updating the parameter `cpu.shares` and the CPU affinity could be set via updating the parameter `cpuset.cpus`. Besides, the parameter `cpu.rt_runtime_us` and the parameter `cpu.rt_period_us` can work together to control the threshold of CPU utilization ratio. We can place different VNFs into various defined cgroup, and effectively control the CPU time period occupied by different VNFs.

V. Evaluation of PRSFC-IoT

To judge the effectiveness and the serviceability of PRSFC-IoT, we evaluate it on the protosystem, in which our evaluation is based on both the simulation and the actual testbed.

A. Actual Testbed

As shown in Fig.7, our actual testbed is running on two racks, in which each rack contains four embedded machines. In order to set up a cloud infrastructure with the same topology as shown in Fig.1(b) (standard k=4 fat tree), we use each actual rack to represent one pod and use each embedded machine inside the actual rack to represent one rack. For building the cloud in-network, we use one divisible switch which offers support for dividing various ports into an isolated group, meanwhile each ports in the isolated group can independently reach a line rate performance, i.e. working like a smaller switch with fewer ports. We use two 4-ports smaller switches to represent two core switches, use four 4-ports smaller switches to represent four EoR switches, use the vSwitch [27] in each embedded machine to represent the ToR switches. Each embedded machine has 8 Intel Xeon(E5-2630v3) 2.60GHz CPU cores with 64GB RAM, running with Centos Server 7, 64-bit. Each 4-ports smaller switch is divided from one Pica8 3297 Openflow-enabled switch with 48-ports 1000BASE-T RJ45, running with PicOS 2.3.0. Every port of 4-ports smaller switch has an independent line rate of 1Gb/s bandwidth.

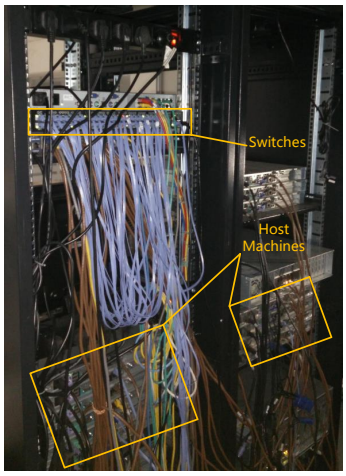


Fig. 7. The actual testbed.

We compare our approach against a greedy based algorithm which as our baseline. In order to detect chain processing bottlenecks. The status of cloud, especially the CPU resource utilization ratio need to be constantly monitored in the greedy strategy. Once a VNF instance with outstanding CPU resource utilization ratio, i.e. over 95% is found, a new VNF instance would be spawned in the same rack to assist the overloaded VNF instance. If the CPU time period in this rack are used up, the new VNF instance would be tentatively placed in one rack that are one hop away before considering other racks, the extra workloads are then split into the new spawned VNF instance. In order to calculate the acceptance rate of SFC orchestration, it checks

2327-4662 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2800003, IEEE Internet of Things Journal

whether can assign the new service chain into the cloud infrastructure when a new request of SFC orchestration arrives at run tim. If applying the assignment violates resource constraints of the cloud infrastructure, i.e. no location can be found for this service chain, if so, the request will be rejected.

We implement two type of VNF instance: Network Address Translation (NAT) and Firewall (FW). The FWs match connection state of TCP packets and the NATs match source address of IP packets according to a list of predefined rules. FWs do no modification to the original packets but NATs do modify the TCP source port and IP source address. Each VNF has a parameter that specifies the number of rules it needs to match, e.g. FW10000 and NAT50000 respectively stand for FW with 10K rules and NAT with 50K rules. Four different VNF instances are used in our testbed: FW10000 with $wcet(P_b) \approx$ 2ms, FW30000 with $wcet(P_b) \approx$ 6ms, NAT10000 with $wcet(P_b) \approx$ 4ms, and NAT50000 with $wcet(P_b) \approx$ 20ms. The $wcet$ is measured when the P_b is 12.5% of the total CPU time period of a single CPU core.

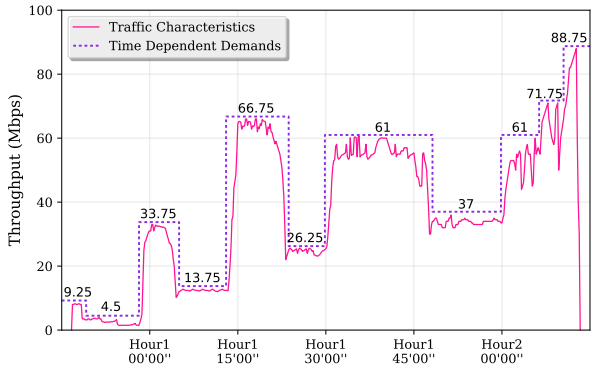


Fig. 8. Traffic workloads and time-dependent demands.

Moreover, a traffic generator has been implemented with Scapy [30] to introduce proper traffic rate for service chains. The traffic characteristics and corresponding time-dependent demands for each service chain as shown in Fig.8. Within the arranged one hour, we let the two strategies constantly assign chain instances into cloud infrastructure for three service chains, based on the time-dependent demands. Each service chain sequentially contains one FW10000, one NAT50000, one FW30000 and one NAT10000.

1) Deadline Satisfaction: For evaluating the deadline satisfaction degree of service chains between PRSFC-IoT and predefined baseline, we measure ahead of time the delay of traffic transmission in our testbed, and find out the delay of transmitting an encapsulated packet from one VNF to its next hop VNF is at most 3 ms. The deadline of each service chain is set as 25 ms, and the tolerance coefficient of the deadline is set as 0.8. During the experiment, each IoT flow i that enters the chain instance is calculated its end-to-end latency (completion time) ct_i at the export of the service chain, in which $ct_i = t_{out} - t_{in}$, t_{in} and t_{out} respectively are the time of flow entering or flow quitting chain instance. As shown in Fig.9, we can find that 93.6% of the traffic entering the service chains orchestrated via PRSFC-IoT meet the deadline

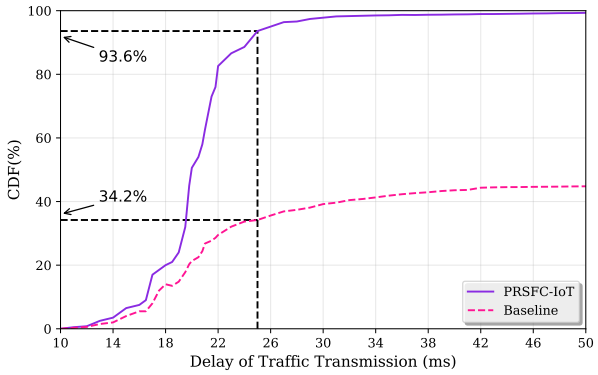


Fig. 9. Deadline satisfaction between PRSFC-IoT and baseline.

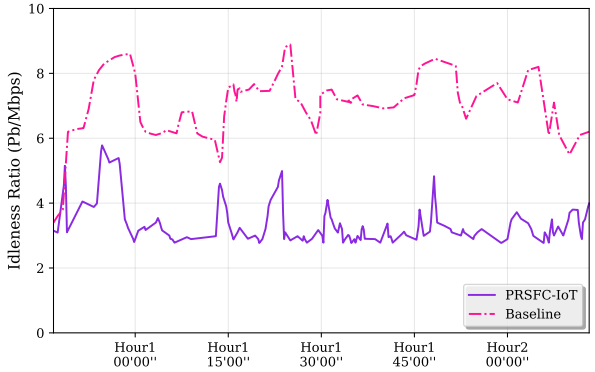


Fig. 10. Resource Idleness between PRSFC-IoT and baseline.

satisfaction, while only 34.2% for baseline. We also find that over 99% of traffic with PRSFC-IoT achieve 50 ms deadline, compared to at most 45% for baseline. In contrast, the end-to-end latency under PRSFC-IoT is almost bounded, and only a very small fraction i.e. less than 1% of the traffic have a end-to-end latency larger than 2 times the deadline.

2) Resource Idleness: For evaluating the effect of PRSFC-IoT on idleness avoidance, we calculate the indicator of CPU resource idleness against deployed chain instances during the experiment. The indicator of resource idleness for each chain instance is represented as $\frac{total\_period}{realtime\_rate}$, where $total\_period$ is the total amount of used CPU time period for one chain instance and $realtime\_rate$ is the total real time rate of served traffic for this chain instance. The results of evaluation are shown in Fig.10.

To zoom in the gap, at each time point we collect a number of indicators both for the two cases, in which for the case of PRSFC-IoT, the maximum indicator is kept while for the case of baseline, the minimum indicator is kept. These selected indicators reform the two curves in the figure. The results show almost all of the chain instances orchestrated by PRSFC-IoT with a lower indicator of resource idleness than that orchestrated by baseline, which reflects lighter degree of resource idleness in the chain level. In addition, we further observe that abrupt increments of indicator are always to be seen during the workload switching of service chains, e.g. scaling in or scaling out, whose reasons cloud be considered in our future work.

2327-4662 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2800003, IEEE Internet of Things Journal

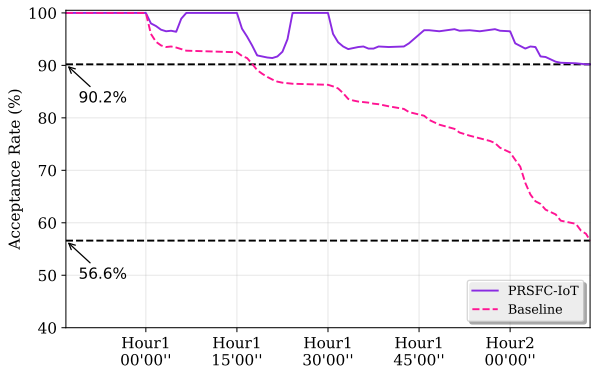


Fig. 11. Request acceptance rate between PRSFC-IoT and baseline.

3) Deployment Acceptance Rate: A stress test environment is built in this experiment for evaluating the acceptance rate of SFC deployment request. Accordingly, we increase the amount of service chains which to be deployed from three to five, and remain the workloads and constitutions of VNF in each service chain unchanged. The request acceptance rates between PRSFC-IoT and baseline are shown in Fig.11.

We can find that PRSFC-IoT always outperforms the baseline in this respect, and the performance gap increases with time goes by. The results are expected because the requests would constantly accumulate in the cloud over time, and it becomes more difficult for a greedy strategy to fully utilize resource. Note that a smaller difference at the beginning is also expected, because the cloud has a lot of resource available, allowing the baseline to consume resource aggressively. When the resource in the cloud is almost saturated i.e. at the first peak, the performance of baseline begins to drop gradually. In contrast, PRSFC-IoT from this time starts to be demonstrated as advanced to handle requests effectively under limited resource. In the most obvious gap, PRSFC-IoT based orchestration accepts almost 2 times requests in the time slot, compared to the baseline based orchestration.

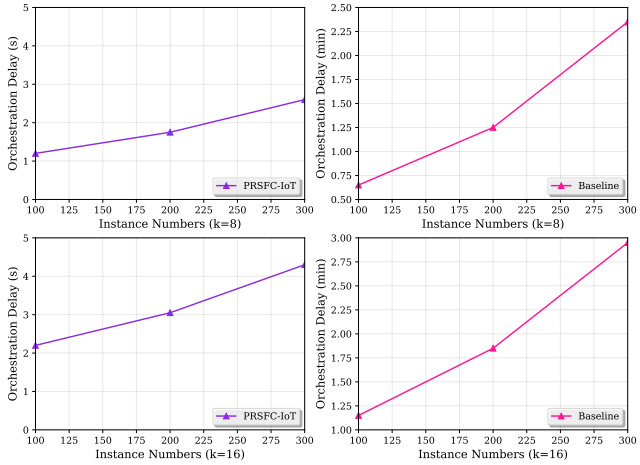


Fig. 12. Deployment delay between PRSFC-IoT and baseline.

B. Simulation

Our approximation algorithm on chain instance deployment is evaluated by the method of simulation which implemented with Python. We simulate a cloud with the same topology as standard k fat tree where k is 8 and 16, respectively. The cloud contains $5k^2$ machines, each of which with $32P_b$ CPU time period, divided into k pods. Each pod contains $\frac{k}{2}$ racks and $\frac{k}{2}$ EoR switches. There are $(\frac{k}{2})^2$ core switches that are shared by all pods. Every link in the cloud has a bandwidth of 10Gb/s. We orchestrate three different amount, i.e., 100, 200 and 300 of chain instances into cloud, respectively. Each instance with length at most 10, with starting and ending position of served traffic randomly picked. The max packet rate of each instance is randomly chosen from 1000 to 4000 packets/s. Hence, the largest traffic rate is about 48Mbps. For comparison, we set a baseline strategy based on a traditional ILP method. In order to obtain the LP solution of chain instance deployment in Section III(B), Gurobi [31] is used in our simulation for solving LP and ILP. The results are shown in Fig.12, showing that our approximation optimization based resource allocation algorithm is highly effective. The deployment delay of PRSFC-IoT is less than 4.5s, which is several orders of magnitude better than the traditional ILP method with almost 3min for the offline planning, meaning that PRSFC-IoT can accomplish a agile and scalable chain instance deployment through a real time mode in the IoT cloud environment.

VI. RELATED WORKS

Most prior works on SFC orchestration have not taken both the performance guarantee and the resource idleness into consideration to implement a joint optimization solution to support dynamic deployment of service chains in a general virtualized cloud environment. For instance, PLayer [32] and SIMPLE [33] show effectiveness on routing traffic between static middleboxes, but have ignored cloud virtualization and dynamic middlebox assignment. Similarly, CoMb [14] although considers dynamic middlebox assignment from the premise that routing pathes are dedicated, but with the target of determining which host machines on the path should be chosen to undertake the functions of middleboxes. Stratos [15] uses the method of monitoring to detect the cloud status and resource bottlenecks, so as to reactively reply workload scaling-out by replicating or migrating VMs. However, the reactive nature of Stratos leads to poor performance guarantees, especially for the deadline guarantees. Before a bottleneck is detected, several deadlines might have been failed. NFV-RT [16] is a real time system which supports proactively deploying service chains into RT-Xen [34] based cloud environment but ignores to avoid resource idleness in chain level. In contrast, PRSFC-IoT assigns the cloud resource to service chains cost-effectively while guarantees the performance of chain processing dynamically, thus achieving both better in performance guarantees and resource idleness avoidance.

In addition, there still exists extensive literatures on resource management in cloud environment [35]–[37] but they are not for real time scenarios. Techniques on the deployment and

2327-4662 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2800003, IEEE Internet of Things Journal

migration for virtual machine are also been followed [38]–[40]. However, these techniques cannot reconcile both the deployment for virtual machines and the steering for traffic, eventually achieve non optimal result.

VII. Conclusion

In this paper, we present the design, implementation and evaluation of PRSFC-IoT, a cloud SFC orchestration system under the IoT scenarios, taking both service performance and resource usage into consideration. PRSFC-IoT integrates with several meaningful methods, such as chain instance consolidation and linear programming with approximation optimization to achieve effective deadline and packet rate guarantee while avoiding resource idleness. The evaluation results show that PRSFC-IoT has significant advantages in deadline satisfying, computation resource effective utilization, deployment acceptance rate and delay guarantee, compared to the baseline strategy.

References

[1] J. Swetina, G. Lu, P. Jacobs, F. Ennesser, and J. Song, "Toward a standardized common m2m service layer platform: Introduction to onem2m," *IEEE Wireless Communications*, vol. 21, no. 3, pp. 20–26, 2014.

[2] M. Villari, A. Celesti, M. Fazio, and A. Puliafito, "Alljoyn lambda: An architecture for the management of smart environments in iot," in *Proc. of IEEE SMARTCOMP*, 2014.

[3] X. Wang, Z. Sheng, S. Yang, and V. C. Leung, "Tag-assisted social-aware opportunistic device-to-device sharing for traffic offloading in mobile social networks," *IEEE Wireless Communications*, vol. 23, no. 4, pp. 60–67, 2016.

[4] X. Wang, Y. Zhang, V. Leung, N. Guizani, and T. Jiang, "D2d big data: Content deliveries over wireless device-to-device sharing in realistic large scale mobile networks," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 1–10, 2018.

[5] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi, "The middlebox manifesto: enabling innovation in middlebox deployment," in *Proc. of ACM HotNets*, 2011.

[6] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 13–24, 2012.

[7] J. Sherry, S. Ratnasamy, and J. S. At, "A survey of enterprise middlebox deployments," 2012.

[8] J. Kim, J. Lee, J. Kim, and J. Yun, "M2m service platforms: Survey, issues, and enabling technologies." *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 61–76, 2014.

[9] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[10] R. Guerzoni, "Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action, introductory white paper," in *SDN and OpenFlow World Congress*, 2012.

[11] M. Naohisa, T. Kenki, H. Sho, A. Hiroki, and A. Aiko, "Iot network implemented with nfv," *NEC Technical Journal*, vol. 10, no. 3, pp. 35–40, 2016.

[12] W. Haeffner, J. Napper, M. Stiemerling, D. Lopez, and J. Uttaro, "Service function chaining use cases in mobile networks," *draft-ietf-sfc-use-case-mobility-01*, 2014.

[13] P. Quinn and T. Nadeau, "Service function chaining problem statement," *draft-ietf-sfc-problem-statement-10 (work in progress)*, 2014.

[14] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proc. of USENIX NSDI*, 2012.

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

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

[17] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, "Toward software-defined middlebox networking," in *Proc. of ACM HotNets*, 2012.

[18] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra *et al.*, "The worst-case execution-time problemłoverview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, p. 36, 2008.

[19] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica, "Multi-resource fair queueing for packet processing," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 1–12, 2012.

[20] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[21] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, 2012.

[22] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[23] G. Pongrácz, L. Molnár, and Z. L. Kis, "Removing roadblocks from sdn: Openflow software switch performance on intel dpdk," in *Proc. of IEEE EWSDN*, 2013.

[24] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.

[25] P. Phaal, S. Panchen, and N. McKee, "Inmon corporation's sflow: A method for monitoring traffic in switched and routed networks," Tech. Rep., 2001.

[26] P. Quinn and U. Elzur, "Network service header," *draft-quinnsfc-nsh-01*, 2014.

[27] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, and P. Shelar, "The design and implementation of open vswitch." in *Proc. of USENIX NSDI*, 2015.

[28] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks," Tech. Rep., 2014.

[29] F. Checconi, T. Cucinotta, D. Faggioli, and G. Lipari, "Hierarchical multiprocessor cpu reservations for the linux kernel," in *Proc. of OSPERT*, 2009.

[30] P. Biondi, "Scapy, a powerful interactive packet manipulation program," 2010.

[31] Gurobi, "Gurobi optimizer reference manual," 2014.

[32] D. A. Joseph, A. Tavakoli, and I. Stoica, "A policy-aware switching layer for data centers," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, 2008, pp. 51–62.

[33] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 27–38, 2013.

[34] S. Xi, J. Wilson, C. Lu, and C. Gill, "Rt-xen: Towards real-time hypervisor scheduling in xen," in *Proc. of IEEE EMSOFT*, 2011.

[35] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea, "Chatty tenants and the cloud network sharing problem," in *Proc. of USENIX NSDI*, 2013.

[36] D. Shue, M. J. Freedman, and A. Shaikh, "Performance isolation and fairness for multi-tenant cloud storage," in *Proc. of USENIX OSDI*, 2012.

[37] D. B. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, M. K. Aguilera, and H. Abu-Libdeh, "Consistency-based service level agreements for cloud storage," in *Proc. of ACM SOSP*, 2013.

[38] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proc. of IEEE INFOCOM*, 2010.

[39] V. Shrivastava, P. Zerfos, K.-W. Lee, H. Jamjoom, Y.-H. Liu, and S. Banerjee, "Application-aware virtual machine migration in data centers," in *Proc. of IEEE INFOCOM*, 2011.

[40] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *Proc. of USENIX NSDI*, 2007.

2327-4662 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.