Hindawi
Wireless Communications and Mobile Computing
Volume 2018, Article ID 3027269, 15 pages
https://doi.org/10.1155/2018/3027269

Research Article

IoT-B&B: Edge-Based NFV for IoT Devices with CPE Crowdsourcing

He Zhu and Changcheng Huang

Department of Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada

Correspondence should be addressed to He Zhu; hzhu@sce.carleton.ca

Received 23 August 2017; Revised 6 December 2017; Accepted 13 December 2017; Published 4 February 2018

Academic Editor: Kuan Zhang

For embracing the ubiquitous Internet-of-Things (IoT) devices, edge computing and Network Function Virtualization (NFV) have been enabled in branch offices and homes in the form of virtual Customer-Premises Equipment (vCPE). A Service Provider (SP) deploys vCPE instances as Virtual Network Functions (VNFs) on top of generic physical Customer-Premises Equipment (pCPE) to ease administration. Upon a usage surge of IoT devices at a certain part of the network, vCPU, memory, and other resource limitations of a single pCPE node make it difficult to add new services handling the high demand. In this paper, we present IoT-B&B, a novel architecture featuring resource sharing of pCPE nodes. When a pCPE node has sharable resources available, the SP will utilize its free resources as a "bed-and-breakfast" place to deploy vCPE instances in need. A placement algorithm is also presented to assign vCPE instances to a cost-efficient pCPE node. By keeping vCPE instances at the network edge, their costs of hosting are reduced. Meanwhile, the transmission latencies are maintained at acceptable levels for processing real-time data burst from IoT devices. The traffic load to the remote, centralized cloud can be substantially reduced.

1. Introduction

Customer-premises equipment (CPE) devices, such as routers, switches, residential gateways, and set-top boxes, have been deployed at the subscriber's premises to originate, route, and terminate communications between the customer premises and the central office [1]. In the wake of cloud computing and Network Function Virtualization (NFV) [2, 3], Service Providers (SPs) leverage virtual Customer-Premises Equipment (vCPE) as Virtual Network Function (VNF) instances on top of generic physical Customer-Premises Equipment (pCPE), in search of rebuilding a dynamic revenue stream [4]. There can be enough resources for pCPE to deploy VNFs locally [5], while pCPE can also coordinate with the cloud if VNF scale-out is needed to accommodate heavier usage.

Taking advantage of centralized cloud services in the core networks has benefits [6] because of scalable and flexible computing capabilities. However, large-number deployments of Internet-of-Things (IoT) devices bring challenges to VNFs running in a centralized cloud, as the network traffic load would be drastically increased by transmitting data between the core and the edge of the network. Such traffic overhead can become unacceptable with excessive data transmission, causing high processing delay or even service outage due to the congestion of the network. Meanwhile, high usage of the cloud networks would jack up the price per usage, resulting in a higher-than-expected operating expense (OPEX).

Recent research has been aware of the explosive growth of devices in the edge of the networks. The concepts of fog computing [7] and edge computing [8] were proposed to move the initial handling of raw data to the edge for IoT devices. Although the fog can mitigate the load of the core network, the power of the Customer Edge (CE), namely, the computational capabilities of CPE, is buried. While a single pCPE node has limited resources and typically serves a designated location, the aggregated computing capabilities of pCPE nodes across the edge of a network can be powerful: pCPE nodes have time-varying resource usage that does not always reach full workloads. For instance, the home gateways

typically have significantly lower usage in business hours as their users leave for work, while office gateways become idle after hours. If the spare resources of pCPE can be shared within the network edge, VNFs will be able to roam around the edge. Both SP and users will benefit from the considerable capabilities of the sharable resources. Meanwhile, the VNFs deployed on the pCPE nodes keep most traffic within the edge and reduce the traffic to the core network.

It certainly sounds interesting to utilize the time-varying computational resources. However, CPE resource sharing faces challenges:

(i) It is unclear how much SP can benefit from spare resources of pCPE nodes compared to traditional virtualization without resource sharing.

(ii) Service availability becomes a concern. A pCPE node's availability can be jeopardized if it no longer has enough resources to host VNFs. It can also be down due to power outages or user pulling the plugs. The availability of offloaded VNFs must be ensured by enforcing proper redundancy.

(iii) The users need to be motivated to consent to contributing their pCPE nodes for resource sharing. They would not do so without incentives or discounts. Incentives returned to users are required in order to benefit both the SP and its end users.

In this paper, we propose an architecture to allow sharing resources of pCPE within the network edge, namely, IoT-B&B. We discuss the scenario that SP deploys VNFs, which are vCPE instances, to both the cloud and the available pCPE nodes participating in the resource sharing program. As Figure 1 shows, when a sharable pCPE node is not actively used by its owner, it will be treated as a "bed-and-breakfast" place for vCPE instances to "stay." SP will have the permission to utilize free resources through the resource manager to deploy VNFs of other users from the same edge network. The following contributions are made for enabling crowdsourcing at the network edge by utilizing resources of pCPE nodes:

(i) We propose an architecture extended from ETSI NFV architecture and interfaces [9] to support resource sharing of pCPE nodes. The pCPE nodes at the network edge are treated as compute hosts which can have VNF instances deployed. They are grouped together and abstracted into the NFV Infrastructure (NFVI) layer. A resource manager is embedded in the NFVI and can leverage placement algorithms to make placement decisions.

(ii) A model is presented to evaluate the cost of assigning a VNF instance to a pCPE node and to the remote cloud. Multiple factors are considered to determine the cost, including remaining resources, network transmission delay, and availability requirements.

(iii) A placement algorithm called "IoT-B&B Algorithm" is also presented to assign vCPE instances to pCPE nodes with the goal of finding a cost-efficient pCPE node for each VNF.

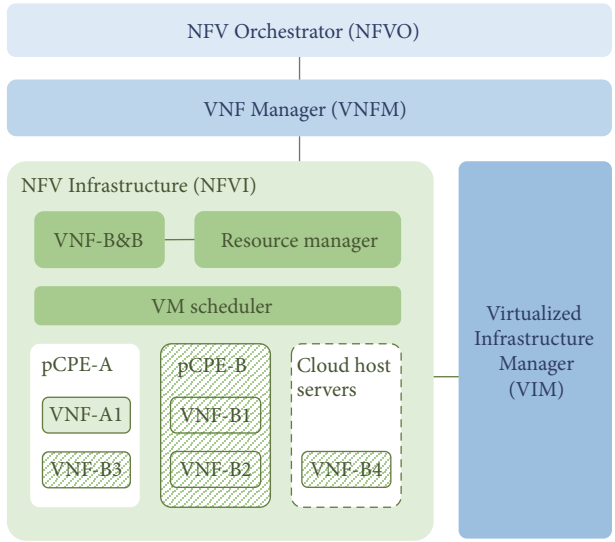

Figure 1: The system architecture is extended from ETSI NFV architecture and interfaces [9]. It has a resource manager integrated with NFVI to schedule VNF placement, leveraging the VNF-B&B algorithm. The architecture enables pCPE resources to be part of the NFVI along with cloud host servers, where each pCPE node can have VNFs deployed for multiple users in the edge network. For instance, when pCPE-B needs more VNFs to handle its overloaded use, they can be deployed on both pCPE-A (VNF-B3) and the cloud (VNF-B4).

(iv) We implement a system with the IoT-B&B architecture with steps to set up the NFVI and the system's life cycle. Numerical results are shown to demonstrate the placement algorithm's effectiveness.

We divide the contents into the following sections. Section 2 formulates the problem. Section 3 proposes the IoT-B&B Algorithm based on the problem formulation. Then, Section 4 is presented, covering the actual implementation of the system, followed by the numerical results in Section 5. The related work is illustrated in Section 6. Section 7 concludes the paper and lists future work items.

2. Problem Formulation

In this section, we formulate the problem by modeling the resource properties and constraints of the network edge. The resource types we discuss are limited to CPU, memory, and network bandwidth. We believe these three types of the resources are most representative for cost estimation and optimization. Adding consideration of more resource types will not necessarily change the optimization model. Then, the properties of the VNF instances are defined and annotated. All notations defined and used are also summarized in the Notations Used in Problem Formulation. Note that the terms "VNF"; "VNF instance"; and "vCPE instance" in this paper are interchangeable.

2.1. Connected pCPE at Network Edge. We discuss one particular network edge, which includes all pCPE nodes under

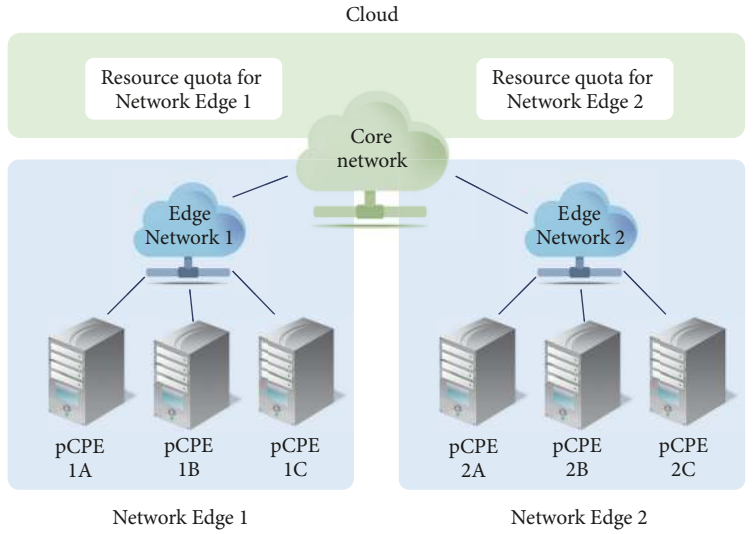

Figure 2: Topologies at network edges. All pCPE nodes can communicate with each other if they are within the same network edge.

it. A network edge is defined as the networks connecting all pCPE nodes under it. We model a network edge to have a topology such that each pCPE node within it can communicate with another. An example with two network edges can be found in Figure 2. The nodes of pCPE 1A, 1B, and 1C group as one network edge connected to each other via Edge Network 1, while the other one consists of the nodes 2A, 2B, and 2C, connected by Edge Network 2.

Based on the definition above, the pCPE nodes and their links at the network edge can be modeled as a directed graph $G = (V, L)$. Set V represents all pCPE nodes at the network edge, while L is the set of all connected links from one vCPE node to another. A pCPE node in V is denoted by v, and there is $v \in V$. Define the total number of pCPE nodes to be n_V. Let v_i be a specific pCPE node in V, such that

$$v_i \in V, \quad \forall i \in [1 \cdots n_V]. \tag{1}$$

Since all pCPE nodes are connected to each other, G is strongly connected. For any data transmitted from one node v_i to another node v_j in V, there exists a link l_{ij}, such that

$$l_{ij} \in L, \quad \forall i, j \in [1 \cdots n_V], \ i \neq j. \tag{2}$$

The network edge is connected to the core network via a logical link, denoted by l_c. The capacity of l_c is limited due to budget reasons: the network edge has a certain bandwidth quota. The total bandwidth to the cloud must be kept within the quota. In reality, l_c can be a group of links connecting the remote cloud.

2.2. VNF Types and Resource Requirement Profiles (Flavors). The network functions serving the IoT networks have been encapsulated into various types of VNFs. Each type provides a user with a specific service. When the demand increases for a certain type of VNF to a certain level that it exceeds the maximum capacity of current VNF instances, the VNF scales out by increasing the number of Virtual Machine (VM) instances, so that more requests can be processed at the same time. Depending on the purpose, different types of VNF have different resource requirements. The term "flavor" is used to describe the resource requirements profile of an instance of the VNF, including the number of vCPUs, the amount of memory, the size of the disk. We use f to define a VNF instance.

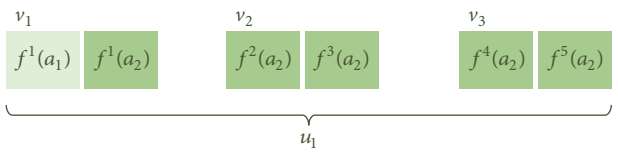

Figure 3: An example of VNF instances grouped by u_1.

Assign a to identify a specific type of network function and n_a to be the number of network function types. A VNF instance with type "a" can then be represented by $f(a)$. The CPU, memory, and bandwidth requirements of $f(a)$ are then denoted by $U(f(a))$, $M(f(a))$, and $B(f(a))$.

2.3. The User of a VNF Instance. Another property of a VNF instance is the user who owns and uses it. Suppose each pCPE node has one unique owner. This is a valid assumption as the actual device on customer premise is typically not shared and belongs to the entity who pays for the service. For a pCPE node v, its owner u uses a set of VNF instances to satisfy its needs, regardless of where the instances are deployed. When referring to a specific node v_i, $i \in [1 \cdots n_V]$, we represent the node's owner by u_i, where $i \in [1 \cdots n_V]$.

Figure 3 illustrates an example of VNF instances grouped by the user u_1. There is 1 instance of $f(a_1)$ and 5 instances of $f(a_1)$. The only instance of $f(a_1)$, denoted by $f^1(a_1)$, is on v_1, along with $f^1(a_2)$. Instances $f^2(a_2)$ and $f^3(a_2)$ are on v_2. Instances $f^4(a_2)$ and $f^5(a_2)$ are on v_3. The numbers and types of VNF instances grouped by their users are determined by

user activities and can change dynamically according to user demands.

Based on the user of VNF instances, the annotation of f can be extended from $f(a)$ to $f(a,u)$, where u is the user who owns and uses f.

2.4. Places to Deploy VNF Instances. Placement decisions are made based on the flavors, which are the resource requirement profiles, of VNF instances and the resource capacities of pCPE nodes. A VNF instance of a certain user u_i, denoted by $f(a,u_i)$, can be deployed at either of the locations below.

2.4.1. B&B Deployment. For a VNF instance $f(a,u_i)$, any pCPE node within the same network edge can be considered as a candidate place to deploy, also known as B&B deployments. A B&B deployment is performed when $f(a,u_i)$ is deployed on a pCPE node v_j. For a B&B deployment of $f(a,u_i)$, its notation can be extended to $f(a,u_i,v_j)$, where v_j is the place to deploy f. We use the set F_v to define all VNF instances deployed on the pCPE node v.

Particularly, when there are enough resources on the pCPE node v_i of u_i, $f(a,u_i)$ can be deployed on v_i locally. For a local deployment of $f(a,u_i)$, its notation becomes $f(a,u_i,v_i)$, where v_i is the place to deploy f.

2.4.2. Cloud Deployment. Besides B&B deployments, the remote cloud can be chosen as an alternative place to deploy VNF instances. For a cloud deployment of $f(a,u_i)$, its notation can be extended to $f(a,u_i,c)$, where c is the place to deploy f, and c stands for the remote cloud location. We use the set F_c to define all VNF instances deployed on the cloud.

2.5. Factors to Impact Placement Decisions. The following factors will impact the placement decisions.

2.5.1. pCPE Resource Capacity. Every pCPE node v has its own resource capacity to host a limited number of VNF instances. As a compute node, the resources for VNF instances are as follows: virtual CPUs (vCPUs) and memory. We assume that there is a plenty of disk space on each pCPE node for any virtual instances deployed. Therefore, the disk space of the pCPE nodes is not in the scope of discussion. Let $U(v)$ denote the number of vCPUs v can provide. Let $M(v)$ be the total amount of memory for VNF instances from v. Figure 4 provides an example of the resource capacity for a pCPE node with 20 vCPUs and 10 GB memory in total. There are currently 2 instances of VNF $f(a_1)$ and 3 instances of $f(a_2)$ deployed on it.

2.5.2. Edge Network Transmission Delay. Comparing with the core network transmission delay to be discussed in the next section, the transmission delay between pCPE nodes at the network edge can be ignored as the bandwidth between edge nodes is considered plenty and the transmission delay is small enough to be ignored in the discussion.

2.5.3. Core Network Transmission Delay. The core network transmission delay of a VNF instance f offloaded to the cloud is defined as the time consumed by offloading the VNF instance to the cloud and is denoted by $t(f)$, while $T_{\max}(f)$ is the maximum allowed network delay for a specific network function instance. The actual delay must not exceed this limit, or the requests would eventually overflow the buffer and cause malfunction of the VNF.

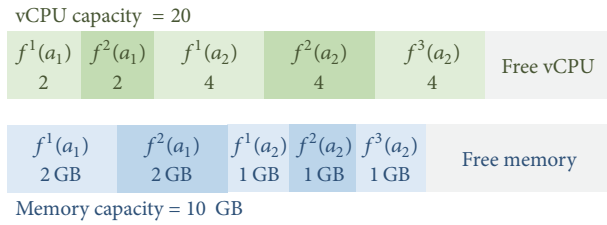

Figure 4: Capacities of the number of vCPUs and the amount of memory of a pCPE node. At the moment, there are 2 instances of VNF $f(a_1)$ and 3 instances of VNF $f(a_2)$.

There are many factors that may affect the core network transmission delay. As stated in [10], the transmission delay to the cloud can be calculated by

$$\text{Transmission delay} = \frac{\text{Message size}}{\text{Network bandwidth}}. \quad (3)$$

For the same message, the less available bandwidth left from the network edge to the cloud, the longer transmission delay will be. During the peak hours, more VNF instances are requested by users concurrently across the network and would congest l_c. The severity of direct oversubscription is reflected by the residue bandwidth of the link from the edge switch to the cloud, denoted by $R(l_c)$, which is the link's total bandwidth $B(l_c)$ less the mean bandwidth usage for all remote VNF instances. The smaller residue bandwidth $R(l_c)$ there is, the bigger core network transmission delay $t(l_c)$ we should expect.

Because all network function instances offloaded to a remote cloud share the same link l_c that connects the edge network to the remote cloud and the same cloud environment, we assume the core network transmission delay is the same for all offloaded network function instances to simplify the modeling process. Let $B(l_c)$ be the total bandwidth of l_c and $t(l_c)$ be the transmission delay of l_c. VNF instances take up the bandwidth of l_c to communicate with the pCPE nodes. The latency of the core network is therefore highly correlated to the bandwidth consumption of l_c.

Let F_c denote the set of VNF instances deployed in the remote cloud. $R(l_c)$ can be calculated by

$$R(l_c) = B(l_c) - \sum_{f \in F_c} B(f), \quad \forall f \in F_c. \quad (4)$$

Even if l_c is not congested, the remote cloud environment may be degraded by other sources. For example, overloaded VNFs from other users or applications of a shared cloud could affect other VNFs on the same host because of overcommitting. To better utilize the resources on a host, overcommitting is enabled by default [11]. However, the performance could be jeopardized if some VNFs are taking up most of the resource

[12]. We define the delay not directly caused by the network edge as a T_d, where the value of T_d changes according to the load of the cloud environment.

Oversubscription will result in a higher core network delay $t(l_c)$. For all VNF instances offloaded to the cloud, there must be

$$t(l_c) \le T_{\max}(f), \quad \forall f \in F_c. \tag{5}$$

The equation above ensures the functionality of all VNFs offloaded to the cloud with the existence of $t(l_c)$. It draws a limit of how much VNF offloading can be done, since an oversubscribed cloud environment would increase $t(l_c)$. Based on the calculation of transmission delay, we further model $t(l_c)$ to be inversely proportional to $R(l_c) + b$ and proportional to T_d with b as a constant scoping the maximum core network delay when the bandwidth of l_c is depleted:

$$t(l_c) = \frac{T_d}{R(l_c) + b}, \quad 0 \le R(l_c) \le B(l_c). \tag{6}$$

Combining (5) with (6), we have

$$\frac{T_d}{R(l_c) + b} \le T_{\max}(f), \quad \forall f \in F_c. \tag{7}$$

When T_d gets higher or $R(l_c)$ becomes lower, the value of $t(l_c)$ would exceed $T_{\max}(f)$ of one or more offloaded VNF instances.

2.6. Cost of Offloading to Edge Network. By enabling the resource sharing of pCPE nodes across the network edge, SP benefits from extended containers hosting the VNFs. The costs of leveraging these resources include the following.

2.6.1. Incentives Returned to End Users. By encouraging the users to participate in the resource sharing program and to consent to share, it is necessary to give incentives to the users based on the amount of resource shared. We denote the unit incentive of CPU, memory, and bandwidth usage for pCPE node v as $w_U(v)$, $w_M(v)$, and $w_B(v)$, respectively. One exception is that when the pCPE node is hosting VNF instances used by its own user, the incentives do not apply.

2.6.2. Extra Redundancy. Since the availability of the pCPE nodes is lower than the cloud, more standby VNF instances are needed. We define the redundancy factor γ to be the mean number of standby VNF instances needed for one VNF instance on B&B nodes.

Based on the two factors above, the cost of offloading a VNF instance f to any of the pCPE nodes, denoted by $S(f, v)$, is calculated as below:

$$S(f, v) = (1 + \gamma) \cdot \left[w_U(v) U(f) + w_M(v) M(f) + w_B(v) B(f)\right]. \tag{8}$$

2.7. Cost of Offloading to Cloud. As defined earlier, we use c to represent the remote cloud in general to deploy VNF instances, to make it distinct from B&B deployments. Although the resources in the cloud, especially in the public cloud, can be considered infinite [13] due to the elasticity of the amount of resources available, for a specific edge network, there are budgets for resources assigned to it. Therefore, resources in the cloud to an edge network are limited when we model them.

The total amounts of vCPUs, memory, and network bandwidth assigned to the edge network we discuss in the cloud are denoted by $U(c)$, $M(c)$, and $B(c)$, respectively. The cost of offloading to the cloud depends on its usage. In general, the less resources left in the cloud for the edge network, the higher unit price our model gives, because the cloud needs to be available as an alternative place to host vCPE instances. Allowing the cloud resources to be drained too early will jeopardize the flexibility of placement and do harm to the service availability.

Let $w_U(c)$ stand for the unit cost for consuming the cloud's CPU resource. We model $w_U(c)$ to be inversely proportional to the cloud's remaining vCPUs with the constant of proportionality W_U. The remaining number of vCPUs is denoted by $R_U(c)$. The total cost of vCPUs for a VNF instance f to be offloaded to the cloud, denoted by $S_U(f, c)$, is then

$$\begin{aligned} S_U(f, c) &= w_U(c) U(f) = \frac{W_U}{R_U(c) + \delta} U(f) \\ &= \frac{W_U U(f)}{U(c) - \sum_{f' \in F_c} U(f') + \delta}. \end{aligned} \tag{9}$$

In (9), δ is a small positive number to avoid dividing by zero.

Also, let $w_M(c)$ represent the unit cost of the memory resource in the cloud. Like vCPUs, $w_M(c)$ is modeled to be inversely proportional to the cloud's residue memory with the constant of proportionality W_M. The residue memory resource of the cloud is denoted by $R_M(c)$. Similar to the induction of (9), we have the total cost of cloud memory for a VNF instance f denoted by $S_M(f, c)$, where δ is a small positive number to avoid dividing by zero:

$$\begin{aligned} S_M(f, c) &= w_M(c) M(f) = \frac{W_M}{R_M(c) + \delta} M(f) \\ &= \frac{W_M M(f)}{M(c) - \sum_{f' \in F_c} M(f') + \delta}. \end{aligned} \tag{10}$$

Let $w_B(c)$ denote the unit cost of the remote cloud's network bandwidth. The variable $w_B(c)$ is defined to be proportional to the core network delay $t(l_c)$ with the constant of proportionality W_B. We define the total cost of bandwidth used between the VNF instance f and the cloud as $S_B(f, c)$. As defined previously, b is a constant representing the maximum

core network delay when the bandwidth of l_c is depleted. Then we have

$$\begin{aligned} S_B(f,c) &= w_B(c)\,B(f) = W_B t(l_c)\,B(f) \\ &= W_B \frac{T_d}{R(l_c)+b} B(f) \\ &= \frac{W_B T_d B(f)}{B(l_c) - \sum_{f' \in F_c} B(f') + b}. \end{aligned} \tag{11}$$

From (9), (10), and (11), the cost of offloading a VNF instance f to the cloud, denoted by $S(f,c)$, is then calculated as

$$\begin{aligned} S(f,c) &= S_U(f,c) + S_M(f,c) + S_B(f,c) \\ &= \frac{W_U U(f)}{U(c) - \sum_{f' \in F_c} U(f') + \delta} \\ &\quad + \frac{W_M M(f)}{M(c) - \sum_{f' \in F_c} M(f') + \delta} \\ &\quad + \frac{W_B T_d B(f)}{B(l_c) - \sum_{f' \in F_c} B(f') + b}. \end{aligned} \tag{12}$$

2.8. Objective and 0-1 Integer Programming Formulation. SP would like to reduce the total cost of deploying and running VNF instances for all users across the network edge. The VNF instances can be deployed either to the remote cloud location c or to the pCPE location v. Based on where the VNF instances are offloaded, we identify two portions of costs offloading the VNF instances: (1) to the cloud and (2) to B&B nodes. The objective of the optimization is to minimize the total offloading cost of the SP.

Variables

(i) $X(f,v)$: a group of Boolean variables representing if each VNF instance f is deployed on the B&B node v.

(ii) $X(f,c)$: a group of Boolean variables representing if each VNF instance f is deployed on the remote cloud c

$$X(f,v) = \begin{cases} 0, & f \text{ not deployed on } v; \\ 1, & f \text{ deployed on } v. \end{cases}$$
$$X(f,c) = \begin{cases} 0, & f \text{ not deployed on cloud}; \\ 1, & f \text{ deployed on cloud}. \end{cases} \tag{13}$$

Objective

$$\text{Minimize} \quad \sum_{f \in F} \sum_{v \in V} S(f,c)\,X(f,c) + S(f,v)\,X(f,v) \tag{14}$$

Constraints

$$X(f,c) + \sum_{v \in V} X(f,v) = 1, \quad \forall f \in F, \tag{15}$$

$$U(c) - \sum_{f \in F_c} U(f) \geq 0, \tag{16}$$

$$M(c) - \sum_{f \in F_c} M(f) \geq 0, \tag{17}$$

$$\frac{T_d}{R(l_c)+b} \leq T_{\max}(f), \quad \forall f \in F_c, \tag{18}$$

$$U(v) - \sum_{f \in F_v} U(f) \geq 0, \quad \forall v \in V, \tag{19}$$

$$M(v) - \sum_{f \in F_v} M(f) \geq 0, \quad \forall v \in V. \tag{20}$$

Remarks

(i) Function (14) is the objective function. It minimizes the total cost of offloading VNFs instances to the cloud and to B&B nodes.

(ii) Constraint (15) ensures that every VNF instance $f \in F$ is only deployed at one place.

(iii) Constraints (16) and (17) are the capacity bounds of the CPU and memory of the cloud. Each type of the three resources leveraged by all VNF instances offloaded to the cloud must not exceed the cloud's allocated resource capacity for the network edge.

(iv) Constraint (18) sets the bottom line of the residue bandwidth for l_c between the network edge and the cloud, which is essentially setting a limit for the number of VNFs offloaded to the cloud.

(v) Constraints (19) and (20) are the capacity bounds for CPU and memory of every pCPE node. Each type of the resources used by all VNFs offloaded to the vCPEs must not exceed these bounds.

3. IoT-B&B Heuristic Placement Algorithm

From the 0-1 integer programming in the previous section, we design a heuristic algorithm to achieve a lower cost by choosing the first valid candidate place to deploy new VNF instances, after the candidate places are sorted by the remaining resources.

3.1. Preliminary Resource Check. For every request of deploying a new VNF instance, we first use Algorithm 1 to check the placement eligibility of every pCPE node, as well as the remote cloud. If the place does not meet the resource constraints of the instance, it will be excluded from the list of candidate places. By calling the function GetCandidates(f), a list of candidate places will be returned from the input of a specific VNF instance f and the current resource level. The list will be sorted by considering the lowest percentage of

```
(1)  function GetSortedCandidates(f)
(2)      create an empty list candidates
(3)      for all v ∈ V do
(4)          if IsResourceEnough(v, f) then
(5)              add v to candidates
(6)          end if
(7)      end for
(8)      sort candidates by remaining resources descending
(9)      if IsResourceEnough(c, f) then
(10)         add c to candidates
(11)     end if
(12)     return candidates
(13) end function

(14) function IsResourceEnough(v, f)
(15)     if v is c then                                  ▷ Check delay for cloud
(16)         link_bw_left ← B(l_c) + b
(17)         for all f' ∈ F_c do
(18)             link_bw_left ← link_bw_left − B(f')
(19)         end for
(20)         delay ← T_d / link_bw_left
(21)         if delay < T_max(f) then
(22)             return false                            ▷ Too much delay
(23)         end if
(24)     end if
(25)     if R_U(v) < U(f) then
(26)         return false                                ▷ vCPU not enough
(27)     end if
(28)     if R_M(v) < M(f) then
(29)         return false                                ▷ memory not enough
(30)     end if
(31)     if R_B(v) < B(f) then
(32)         return false                                ▷ bandwidth not enough
(33)     end if
(34)     return true                                     ▷ validation passes
(35) end function
```

Algorithm 1: IoT-B&B resource eligibility check algorithm.

remaining resource type, in a descending order. For example, if a pCPE node has 90% of vCPU left but only 20% of memory left, then the remaining memory will be used for sorting.

3.2. Cost Estimation. With the list of eligible candidate places for a VNF instance f, we can further estimate the cost of f deployed at each place. Algorithms 2 and 3 provide implementation of the cost model from Section 2. Algorithm 3 defines the function to choose the place for VNF instance f at the lowest cost, namely, ChoosePlace(f). The function first calls GetCandidates(f) in Algorithm 1 to get the places eligible for deploying f. Then, for each eligible place, the cost is checked based on the type of the place based on Algorithm 2. If the place is the cloud, CloudCost(f) is invoked for cost; if the place is a pCPE node, the function BnbCost(f, v) is called instead. After iterating all eligible places, the place with the lowest cost is selected and returned.

3.3. Time Complexity. Algorithm 1 has the time complexity of $O(n\log(n))$ because of sorting the pCPE nodes by remaining resources (assuming merge sort is used). Algorithm 2 has time complexity of $O(1)$. Algorithm 3 will always compare the first candidate pCPE node with the cloud and choose the destination with the lower cost, which has the time complexity of $O(1)$. Combining the three algorithms, the time complexity of IoT-B&B Algorithm is $O(n\log(n))$.

If the exhaustive algorithm is used, which does not sort the candidate pCPE nodes, it would have to check all candidates and find out the one with the lowest cost. Such algorithm would increase the time complexity to $O(n^2)$. Figure 5 shows the time consumed using the two different algorithms (Algorithms 2 and 3) with up to 50 pCPE nodes. From the results, we can see that VNF-B&B algorithm scales well compared to the exhaustive algorithm, where the time consumed is less than 1000 ms for 50 nodes, while the exhaustive algorithm takes more than 9000 ms.

3.4. Access to IoT-B&B Algorithms. We have implemented the Java version of the IoT-B&B Algorithm library. The library source code is published under the MIT License and is

```
(1)  function CloudCost(f)
(2)      cpu_left ← U(c) + δ
(3)      memory_left ← M(c) + δ
(4)      link_bw_left ← B(l_c) + b
(5)      cost ← 0
(6)      for all f' ∈ F_c do
(7)        cpu_left ← cpu_left − U(f')
(8)        memory_left ← memory_left − M(f')
(9)        link_bw_left ← link_bw_left − B(f')
(10)     end for
(11)     cost ← cost + W_U U(f)/cpu_left
(12)     cost ← cost + W_M M(f)/memory_left
(13)     cost ← cost + W_B T_d B(f)/link_bw_left
(14)     return cost
(15) end function

(16) function BnbCost(f, v)
(17)     cost ← 0
(18)     if user of f does not own v then
(19)         cost ← cost + w_U(v)U(f)
(20)         cost ← cost + w_M(v)M(f)
(21)         cost ← cost + w_B(v)B(f)
(22)         cost ← cost × (1 + γ)
(23)     end if
(24)     return cost
(25) end function
```

Algorithm 2: IoT-B&B cost estimation algorithm.

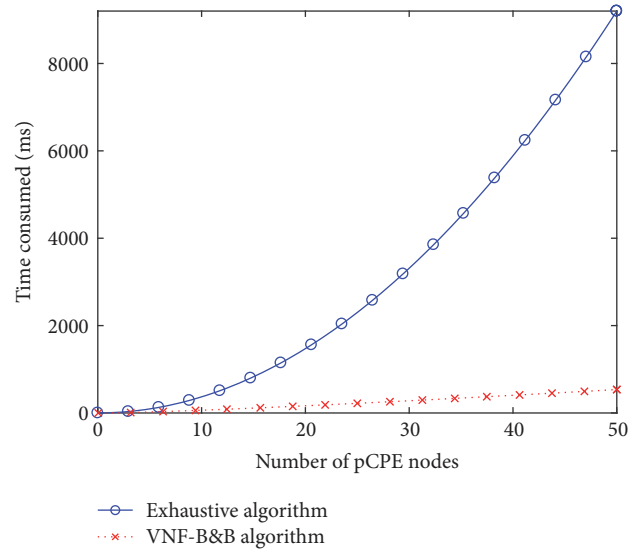

Figure 5: Time complexity comparison between IoT-B&B heuristic algorithm and the exhaustive algorithm checking all pCPE candidate nodes.

downloadable from the following URL: https://github.com/zhuheec/iot-bnb.

4. System Implementation

We have implemented a system following the architecture illustrated in the previous section. The system provides a platform to practice and evaluate the IoT-B&B algorithm.

4.1. Hardware Configuration of pCPE Nodes. For flexibility and scalability, we use VMs instead of bare metal machines as pCPE nodes. Up to 99 VMs are deployed, and each acts as a pCPE node with 8 Cores of CPU, 16 GB of memory, and 40 GB of disk space. Every pCPE node can communicate with any other one via a private virtual network, to mimic that these pCPE nodes are at the same network edge.

4.2. NFVI Setup. Each pCPE node has CentOS 7 [14] installed as its operating system. It has its essential functionalities running as CPE. We use the OpenStack Kolla Project [15] to deploy OpenStack services across multiple pCPE nodes as well as PE, such that

(a) the OpenStack services on a pCPE node runs as Docker containers. They can be spun up and torn down with minimal overhead;

(b) with container-based OpenStack services, a pCPE node can be converted into an OpenStack compute node and then register to the controller to be one of the available hosts;

(c) when the pCPE is no longer available to be a compute node due to high usage, the OpenStack services on it can be stopped to free up resources.

Figure 6 reflects the architecture we use to provision IoT-B&B service upon container-based OpenStack.

4.3. IoT-B&B Algorithm as Filter Scheduler. To leverage the proposed IoT-B&B algorithm in the system, we implement it as a *filter scheduler* used by `nova` service, with the name `VnfBnbFilter`. The IoT-B&B algorithm matches the filtering-weighting mechanism of the *filter scheduler*.

Figure 7 shows how IoT-B&B algorithm works as a filter scheduler to rank places to deploy. Suppose there are four places to choose: v_1, v_2, v_3, and c. Algorithm 1 is first invoked to filter out ineligible places that do not have enough resources. In the example, v_2 is filtered out as a result of resource shortage. Then, Algorithms 2 and 3 are used to rank the places according to the cost to deploy the VNF instance. They determine that v_3 has the lowest cost to deploy the instance.

4.4. System Life Cycle. We define a set of system states and events for IoT-B&B. The system transitions its state based on the events according to the actual demand, in order to adjust the scaling of the VNF. As Figure 8 shows, the states and events are listed as follows.

4.4.1. Up State. The state indicates the system is up and running as expected. System load level is acceptable to satisfy the needs. The system is expected to stay in this state if it runs properly.

```
(1)  function ChoosePlace(f)
(2)      candidates ← GetSortedCandidates(f)
(3)      for all candidate in candidates do
(4)          if candidate is c then
(5)              current_cost ← CloudCost(f)
(6)          else
(7)              current_cost ← BnbCost(f, candidate)
(8)          end if
(9)          if current_cost < cost then
(10)             return place
(11)         end if
(12)     end for
(13)     return none
(14) end function
```

Algorithm 3: IoT-B&B place selection algorithm.

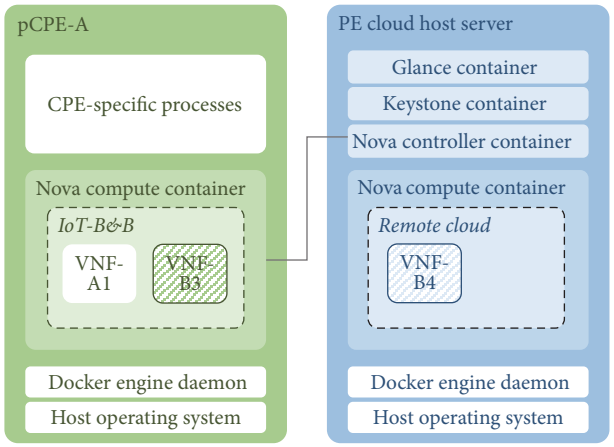

Figure 6: IoT-B&B service backed by container-based OpenStack.

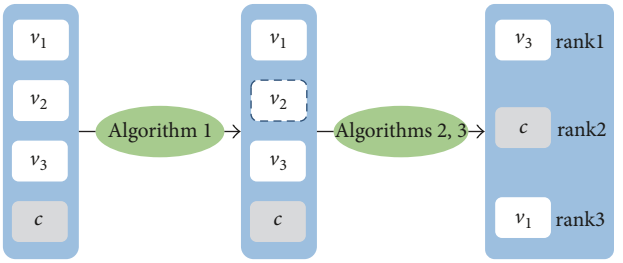

Figure 7: IoT-B&B algorithm in OpenStack as a *filter scheduler*.

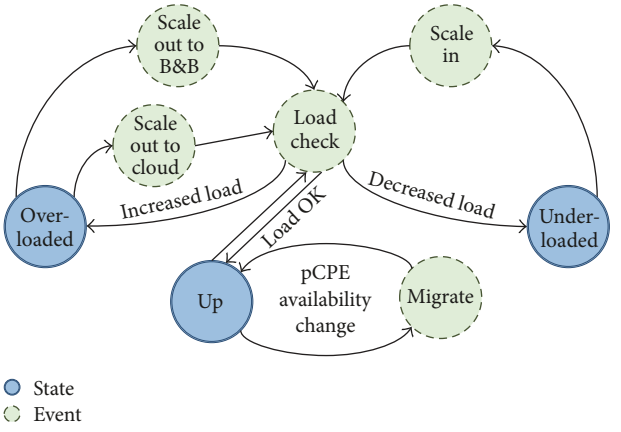

Figure 8: IoT-B&B service system life cycle.

4.4.2. Load Check Event. This is the event to update the system load level. If the updated load level triggers a change, the system may enter Overloaded or Underloaded state or remain in Up state, depending on the threshold to determine them.

4.4.3. Overloaded State. The state indicates the system is overloaded by a higher volume of requests. A scale-out is pending. The placement scheduler will be invoked to determine the place to scale out: B&B or the cloud and then triggers the actual event to scale out.

4.4.4. Scale-out to B&B Event. This is the event to trigger a new VM to be deployed on a B&B, that is, a CPE deployment.

4.4.5. Scale-out to Cloud Event. This is the event to trigger a new VM to be scaled out in the remote cloud environment.

4.4.6. Underloaded State. The state indicates the system is underloaded because of a lower volume of requests. A scale-in is pending. If the number of the VMs has already reached the minimum number required, then the system would not enter this state.

4.4.7. Scale-in Event. This is the event to trigger an existing VM to be scaled in from either the B&B or the remote cloud.

4.4.8. Migration Event. This is the event that is triggered by pCPE availability changes. When a pCPE is no longer capable of hosting a VM because of higher usage from its user, it will cease to be a B&B and be removed from the list of available

Table 1: Constant configurations.

Constant	Value
T_d	50000 ms
W_U	1000
W_M	1000
W_B	1000
δ	1
b	1
γ	1

hosts. Meanwhile, the migration event will be added to the system to move the existing VMs deployed.

4.5. Typical Use Cases. With the definition of the system life cycle, we describe typical uses cases leveraging the IoT-B&B algorithm.

4.5.1. Launch of New Application. The dynamic nature of IoT-based services allows the user to launch new applications which are processed by new VNF instances. A new VNF instance does not automatically get deployed on-site, that is, the pCPE node of its user. The reason can range from lack of enough resources to higher cost being deployed on-site. The IoT-B&B algorithm will be called to determine the place to deploy the new VNF instances.

4.5.2. Scaling out due to Higher Load. When the user applications have more significant activities, resulting in a higher load of the existing instances, the system's periodical load check daemon will detect the load increase. If the load is above the threshold raising flags of performance, extra VNF instances are needed for processing the larger amount of requests. The IoT-B&B algorithm will be called to determine the place to scale out new VNF instances.

4.5.3. Migration for Lower Cost. The VNF instances in the cloud may start with low cost. However, it does not last forever. As more VNF instances are deployed to the remote cloud, fewer resources are available and the unit resource becomes more expensive. At some point, a migration from the cloud to B&B nodes, or vice versa, is reasonable to lower considerable cost.

5. Numerical Results

The numerical results based on simulations are shown in this section. From the numerical results, our goals are to verify the benefits of leveraging B&B nodes, compared to using a centralized cloud alone. Table 1 lists the values of the constants used in the algorithms.

5.1. Host Nodes Setup. We create 99 pCPE nodes with random levels of initial resources. As all pCPE nodes and the cloud are used to deploy VNF instances, in our configuration, there is a total of 100 nodes for deployments. As seen in Figure 9, the nodes are arranged as a 10 × 10 matrix. Each node is represented by a cell, and is given a horizontal and a vertical coordinate from 1 to 10. The last element, which has the coordinates (10, 10), represents the remote cloud. The colors of the cells reflect the remaining resource levels of the nodes. For the ease of demonstration, the CPU, memory, and bandwidth resources are all broken into 20 levels ranging from 1 to 20. Deploying VNF instances on a pCPE node follows the Law of the Minimum [16], meaning that the capacity of a pCPE node to host instances is determined by its scarcest resource. Therefore, we color the cells according to the resource type of the lowest level of a node. For example, if the remaining CPU level of a node is 20, while the memory level is 3, the cell representing the node will be colored at Level 3. From the initial resource levels, it can be learned that the pCPE nodes have various levels of resources available. Meanwhile, the cloud starts with the maximum level of resources for deployment.

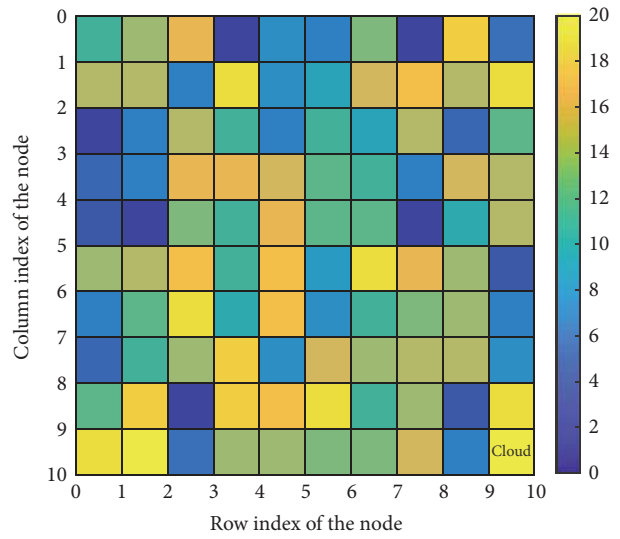

Figure 9: Initial resource levels of the 100 nodes used for experiments. The nodes are arranged as a 10 × 10 matrix, where the last element represents the cloud.

5.2. VNF Resource Requirement Profile (Flavor) Types. We predefine 10 types of VNF resource requirement profiles (flavors), as shown in Table 2, with different requirements of resources and max delays allowed. A VNF instance to be deployed will have a flavor from the 10 predefined ones. Templating VNF flavors is based on the real use cases as users will need VNF instances from limited kinds of images for serving known functionalities.

5.3. Placement Configuration Modes. In order to compare the effectiveness of the IoT-B&B algorithm, we configure the simulated system to keep deploying new VNF instances of a specific flavor with one of the three modes below, until the resources are depleted:

(i) Local mode: deploying only on the pCPE node the user owns. The cloud and B&B nodes are not allowed.

Table 2: Predefined flavor types for simulation. Resource requirements in units.

Name	CPU	Memory	Bandwidth	Max delay
F1	1	1	1	1000 ms
F2	2	2	2	100 ms
F3	2	2	2	1000 ms
F4	2	2	2	10000 ms
F5	4	4	4	100 ms
F6	4	4	4	1000 ms
F7	4	4	4	10000 ms
F8	8	8	8	100 ms
F9	8	8	8	1000 ms
F10	8	8	8	10000 ms

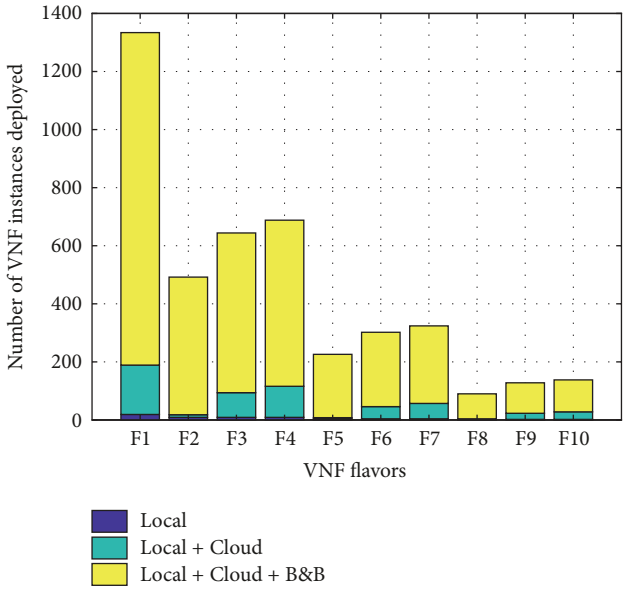

Figure 10: Total number of VNF instances deployed for the network edge with various flavors and placement configuration modes.

(ii) Local + cloud mode: deploying locally on the pCPE node the user owns, as well as on the remote cloud.

(iii) Local + cloud + B&B mode: local, cloud, and B&B deployments.

5.4. Extended VNF Instance Capacity. Figure 10 shows the numbers of instances of deploying each of the 10 predefined flavors with the 3 modes. From the results of Figure 10, we learn that the numbers of instances deployed for all 10 flavors have dramatically increased. Taking F3 as an example, in *Local Mode*, only 9 instances are deployed. When using *Local + Cloud Mode*, the number jumps to 85. For *Local + Cloud + B&B Mode*, the number skyrockets to 550. Therefore, the most beneficial part of the system is to extend the total capacity of hosting VNF instances. If using the remote cloud alone, the capacity of the cloud for VNF instances is limited by the core network delay, even if other resources are assumed to be unlimited. This bottleneck is greatly relieved by the B&B nodes hosting instances, because the instances on B&B nodes do not put extra traffic to the core network.

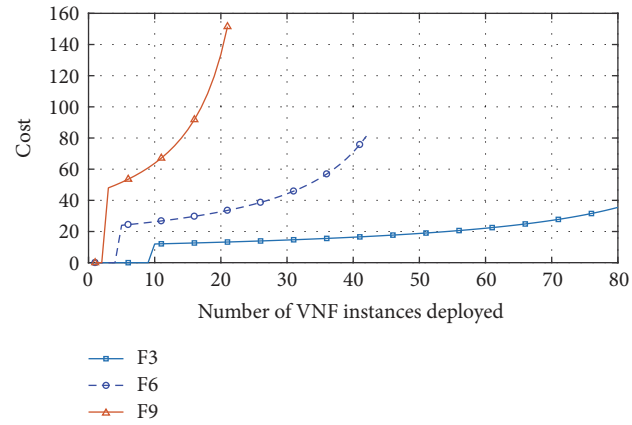

Figure 11: Cost hikes when the cloud load increases: *Local + Cloud Mode*.

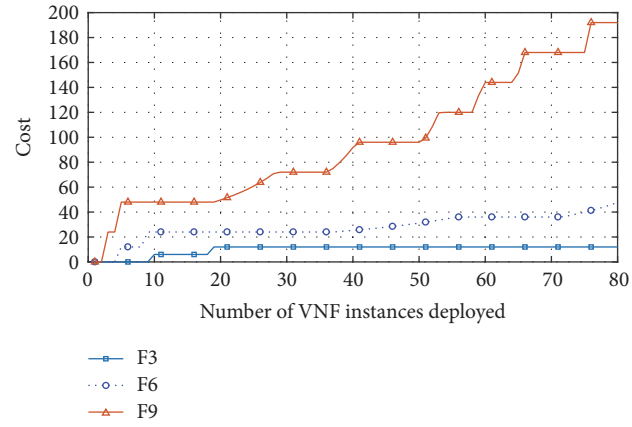

Figure 12: Cost hikes when the cloud load increases: *Local + Cloud + B&B Mode*.

5.5. Cost Hike by Cloud Load Increase. We pick the three flavors: F3, F6, and F9, to investigate the trends of cost increase as more VNF instances are deployed in the system. Figures 11 and 12 demonstrate the changes of costs to deploy

a new VNF instance on the cloud in two different modes, as the numbers of deployed instances go up.

Using *Local + Cloud Mode*, the cost is first 0 as the instances are deployed on the local pCPE nodes. As the loads increase, the remote cloud starts to be picked and the cost to deploy an instance increases as the numbers of deployed instances climb. For F6 and F9, the numbers of deployed VNF instances stop at 42 and 21, respectively.

Comparing Figure 12 with Figure 11, in *Local + Cloud + B&B Mode*, the costs are lower when deploying the same numbers of instances in the system. For instance, when deploying 20 instances with flavor F9, the cost using *Local + Cloud Mode* is about 140. Meanwhile, when deploying the same number of instances with the same flavor, the cost under *Local + Cloud + B&B Mode* is only around 50.

The results above have demonstrated the ability of the B&B nodes to redirect the load off the cloud and to reduce the overall cost, even if offering incentives to the users.

5.6. Impact from outside the Network Edge. As discussed in Section 2, when the cloud load from outside the network edge gets higher, that is, the value of T_d is higher, the ability of the cloud hosting VNF instances may be reduced. To verify how much the impact will be, under *Local + Cloud + B&B Mode* and for the three flavors F3, F6, and F9, we increase the level of T_d by 1 each time and repeat the deployment for 10 times. The numbers of VNF instances deployed are shown in Figure 13. From the results, the capacity of the system is affected by the increase of T_d. However, the impact becomes less significant as the level of T_d increases. With the considerable buffer of the B&B nodes, the impact from T_d is reduced.

5.7. Remaining Resource Levels. In *Local + Cloud + B&B Mode*, all B&B nodes participate in hosting VNF instances. We examine the resource levels after the system resources are depleted. When all VNF instances deployed are of flavor F1, the resource levels after the maximum number of instances is deployed are displayed in Figure 14.

The dark colors of all cells indicate that the remaining resource levels are low across all pCPE nodes. The cloud resource levels are also low because of the link/delay bottleneck from the network edge to the core network. The results have demonstrated the ability of the IoT-B&B algorithm to extract the resources to deploy more instances following the best-effort basis.

6. Related Work

NFV has been a key role to accelerate usage-based changes and to reduce OPEX by both SPs and vendor. Much of the recent NFV research relies on cloud computing as the underlying infrastructure [17].

By leveraging generic cloud computing Infrastructure-as-a-Service (IaaS) frameworks, such as OpenStack [18] and VMWare [19], research on cloud-based NFV has been done to ensure that VNFs run at optimum levels in the cloud [20]. Soares et al. presented a platform for VNFs called Cloud4NFV [3], which is compliant with the ETSI [9] NFV

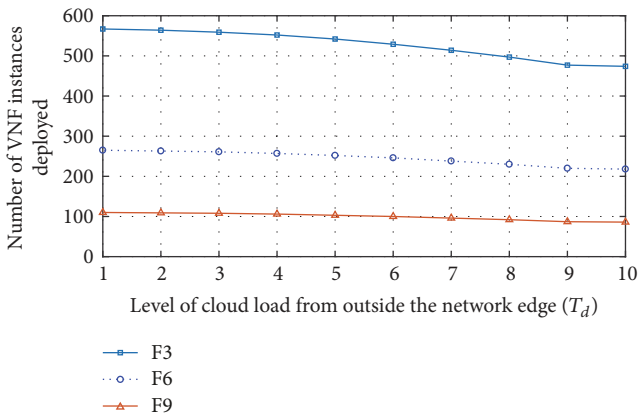

Figure 13: Cost changes when the cloud load increases due to tasks outside the target network edge.

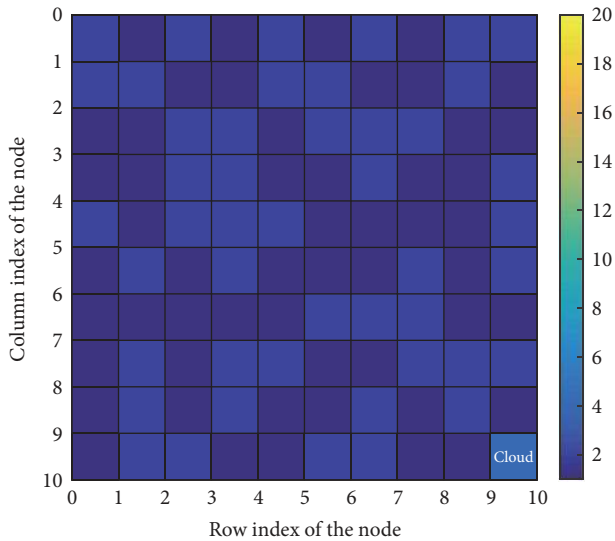

Figure 14: Remaining resource levels after all B&B nodes have been used.

architectural specification. Two approaches were discussed to virtualize NF: full virtualization moved all control and user plane functional entities to the cloud, while partial virtualization still forwarded user traffic to physical hardware. With the implementation of service provisioning and end-to-end inventory management, vConductor [21] was presented by Shen et al. that enabled users to plan the virtual network services using its data model. The systems and architectures above focus on deploying VNF instances into the generic cloud infrastructure, rather than the edge of the network.

Due to the nature of varying cost of resources in the cloud, cloud-based resource allocation problems have been studied to reduce the cost and to help evenly distribute the workload. We have analyzed vulnerability of mobile apps in [22] to keep sensitive information in local mobile device, while offloading secured computing-intensive modules to the cloud. Xiao et al. [23] presented a system leveraging virtualization to dynamically allocate resources in datacenter and to optimize the number of servers in use. While these

solutions did help better use cloud resources, they keep the computing remotely in the cloud and will not move it to the edge of the network.

The concept of fog computing was proposed in [7] and was anticipated to become an essential part of cloud computing with the volume of Internet-of-Things (IoT) growing explosively. Vaquero and Rodero-Merino [24] proposed a comprehensive definition of the fog covering its features and impact, including device ubiquity, challenges on service and fog-based network management, levels of device connectivity, and privacy. Edge clouds were presented as entry points for IoT, which could be parts of the Enhanced Packet Core (EPC). The scenarios of fog computing in several domains were discussed in [25], including Smart Grid, IoT, and SDN, with topics about security, privacy, trust, and service migration. The work above has pointed the research direction of leveraging the edge of the network from high levels. Based on fog computing, crowdsourcing becomes an option as fog nodes can be updated dynamically with the participation of the third party. The security and privacy challenges were illustrated in [26], where a general architecture was presented to model crowdsourcing networks, including crowdsourcing sensing and crowdsourcing computing. The security concerns were captured from the characteristics of the architecture.

Virtualization in edge networks as a form of fog computing, including NFV, have been given a close look. Manzalini et al. visioned potential value chain shifts and business opportunities in [27] by emerging paradigms such as SDN and NFV. The paper pictured a massive number of virtualized network and service functions running at the edge of the network, making the processing power more distributed globally. The service chaining in the cloud-based edge networks was analyzed in [28] by programming actions into OpenFlow switches to achieve dynamic service chaining. A platform called Network Functions At The Edge (NetFATE) was proposed in [29] as a proof of concept (PoC) of an NFV framework at the edge of a telco operator networks. Each CPE node was realized with a generic-purpose computer installed with a hypervisor and virtual switches. This made the CPE node capable of deploying VNFs on itself. The focus of this paper was to prove that deploying VNFs on the edge of the network is feasible. However, the benefits of resource sharing across different CPE nodes are not mentioned.

7. Conclusions

In this paper, we have presented the architecture and the algorithms to share resources of pCPE nodes across the network edge. When a sharable pCPE node has enough resources, SP will utilize its free resources as a bed-and-breakfast place to deploy VNFs of other users from the same network edge for a certain period. The users can get incentives by allowing SP to leverage the free resources.

By applying the VNF-B&B architecture, the capacity of VNF instances for the network edge is greatly increased. The cost of offloading to the centralized cloud is reduced. By keeping the VNFs at the network edge, the delay is reduced for better processing of real-time data burst from IoT devices. Meanwhile, the traffic load to the core network is substantially reduced with the same number of VNF instances deployed.

Making better use of the network edge is an interesting topic and has a massive potential. While the paper ends here, we are continuing to beef up the architecture, including the following:

(i) Explore the availability to factor in the service up and down of B&B nodes. This paper has used a constant factor to model the backup VNF instances. The modeling can be improved so that it is closer to the real-world scenario.

(ii) Consider more factors impacting the deployment placement besides vCPUs, memory, and network bandwidth. Also, consider detailed factors that can indirectly impact the cost and the core network delay of the remote cloud.

Future work of this paper will consider the items listed above with the aim of obtaining a practical and effective framework of virtualizing and utilizing the network edge.

Notations Used in Problem Formulation

v, v_i, n_V:	v is a pCPE node. v_i is a specific pCPE node by its index, where $i \in [1 \cdots n_V]$. n_V is the total number of pCPE nodes in the network edge
c:	The remote cloud location to deploy
u, u_i:	u is a user. Each user owns one pCPE node. u_i is a specific user by the pCPE index, where $i \in [1 \cdots n_V]$
$l_{ij}, l_c, R(l_c)$:	l_{ij} is the link between pCPE nodes v_i and v_j, where $i, j \in [1 \cdots n_V]$, $i \neq j$. l_c is the link between the network edge and the core network. $R(l_c)$ is the remaining network bandwidth of l_c
a, n_a:	a is a VNF type/flavor. n_a is the total number of VNF types
$f, f(a_k)$:	f is a VNF instance. $f(a_k)$ is an instance of type a_k, $k \in [1 \cdots n_a]$
$f(a_k, u_i)$:	A VNF instance of type a_k and user u_i, $k \in [1 \cdots n_a]$, $i \in [1 \cdots n_V]$
$f(a_k, u_i, v_j)$:	A VNF instance of type a_k, user u_i, and deployed on pCPE node v_j, $k \in [1 \cdots n_a]$, $i, j \in [1 \cdots n_V]$
F_v, F_c:	F_v is the set of all VNF instances deployed on pCPE node v. F_c is the set of all VNF instances deployed on the cloud
$U(v), M(v), B(v)$:	The number of vCPUs, the amount of memory, and the network bandwidth capacity that can be provided by the pCPE node v, respectively

$U(f), M(f), B(f)$: The number of vCPUs, the amount of memory, and the network bandwidth required by f, respectively

$t(f), T_{\max}(f), T_d$: $t(f)$ is the core network delay by offloading f to the cloud. $T_{\max}(f)$ is the maximum delay allowed by f. T_d is the core network delay not caused by the network edge

b, δ: b is the maximum core network delay when the bandwidth of l_c is depleted. δ is a very small positive number used as part of the denominators in (9), (10), and (12) to avoid dividing by 0

$S(f, c), S(f, v)$: Cost of f deployed on the cloud c and the pCPE node v, respectively

$X(f, c), X(f, v)$: They equal 1 if f is on c/v and 0 if not.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

The authors would like to thank Yiwen Wang who kindly reviewed the manuscript of this paper and shared comments.

References

[1] Customer premises equipment (CPE), 2017, http://www.thenetworkencyclopedia.com/entry/customer-premises-equipment-cpe/.

[2] S. Beereddy and K. Sirupa, "NFV use case—delivering virtual CPE with multi-vendor VNF orchestration," in *Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN '15)*, pp. 25–27, San Francisco, Claif, USA, November 2015.

[3] J. Soares, M. Dias, J. Carapinha, B. Parreira, and S. Sargento, "Cloud4NFV: a platform for Virtual Network Functions," in *Proceedings of the 3rd IEEE International Conference on Cloud Networking (CloudNet '14)*, pp. 288–293, Luxembourg, Luxembourg, October 2014.

[4] E. Telecom, White Paper: The Definitive Guide to vCPE, 2017, https://www.ecitele.com/media/1703/white-paper-the-definitive-guide-to-vcpe.pdf.

[5] T. Taleb, M. Corici, C. Parada et al., "EASE: EPC as a service to ease mobile core network deployment over cloud," *IEEE Network*, vol. 29, no. 2, pp. 78–88, 2015.

[6] N. Herbaut, D. Negru, G. Xilouris, and Y. Chen, "Migrating to a NFV-based Home Gateway: Introducing a Surrogate vNF approach," in *Proceedings of the 6th International Conference on the Network of the Future (NOF '15)*, pp. 1–7, Montreal, Canada, October 2015.

[7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the 1st Mobile Cloud Computing Workshop on Mobile Cloud Computing (MCC '12)*, pp. 13–16, Helsinki, Finland, August 2012.

[8] M. Satyanarayanan, "The emergence of edge computing," *The Computer Journal*, vol. 50, no. 1, pp. 30–39, 2017.

[9] ETSI Industry Specification Group (ISG) NFV, ETSI GS NFV 002 V1.2.1: Network Functions Virtualisation (NFV): Architectural Framework, 2014.

[10] K. Jang, J. Sherry, H. Ballani, and T. Moncaster, "Silo: predictable message latency in the cloud," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*, pp. 435–448, London, UK, August 2015.

[11] OpenStack: Overcommitting CPU and RAM, 2017, https://docs.openstack.org/arch-design/design-compute/design-compute-overcommit.html.

[12] S. A. Baset, L. Wang, and C. Tang, "Towards an understanding of oversubscription in cloud," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE '12)*, San Jose, Claif, USA, 2012.

[13] Amazon EC2, 2017, https://aws.amazon.com/ec2/.

[14] The CentOS Project, 2017, https://www.centos.org/.

[15] OpenStack Kolla Project, 2017, https://wiki.openstack.org/wiki/Kolla/.

[16] A. N. Gorban, L. I. Pokidysheva, E. V. Smirnova, and T. A. Tyukina, "Law of the minimum paradoxes," *Bulletin of Mathematical Biology*, vol. 73, no. 9, pp. 2013–2044, 2011.

[17] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *IEEE Communications Magazine*, vol. 51, no. 11, pp. 24–31, 2013.

[18] OpenStack, 2017, http://www.openstack.org/.

[19] VMWare, 2017, http://www.vmware.com/.

[20] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: state-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[21] W. Shen, M. Yoshida, T. Kawabata, K. Minato, and W. Imajuku, "VConductor: an NFV management solution for realizing end-to-end virtual network services," in *Proceedings of the 16th Asia-Pacific Network Operations and Management Symposium (APNOMS '14)*, pp. 1–6, Hsinchu, Taiwan, September 2014.

[22] H. Zhu, C. Huang, and J. Yan, "Vulnerability evaluation for securely offloading mobile apps in the cloud," in *Proceedings of the IEEE 2nd International Conference on Cloud Networking (CloudNet '13)*, pp. 108–116, San Francisco, Claif, USA, November 2013.

[23] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1107–1117, 2013.

[24] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review Archive*, vol. 44, no. 5, pp. 27–32, 2014.

[25] I. Stojmenovic and S. Wen, "The fog computing paradigm: scenarios and security issues," in *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS '14)*, pp. 1–8, IEEE, Warsaw, Poland, September 2014.

[26] K. Yang, K. Zhang, J. Ren, and X. Shen, "Security and privacy in mobile crowdsourcing networks: challenges and opportunities," *IEEE Communications Magazine*, vol. 53, no. 8, pp. 75–81, 2015.

[27] A. Manzalini, R. Minerva, F. Callegati, W. Cerroni, and A. Campi, "Clouds of virtual machines in edge networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 63–70, 2013.

[28] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Dynamic chaining of virtual network functions in cloud-based edge networks," in *Proceedings of the 1st IEEE Conference on Network Softwarization (NETSOFT '15)*, IEEE, London, UK, April 2015.

[29] A. Lombardo, A. Manzalini, G. Schembra, G. Faraci, C. Rametta, and V. Riccobene, "An open framework to enable NetFATE (Network Functions at the edge)," in *Proceedings of the 1st IEEE Conference on Network Softwarization (NETSOFT '15)*, pp. 1–6, London, UK, April 2015.