AutoVNF: An Automatic Resource Sharing Schema for VNF Requests

Wei Huang[1], Haoren Zhu[2], and Zhuzhong Qian[2*]
[1]School of Computer Engineering, Nanjing Institute of Technology, Nanjing 211167, China
wweihuang@sina.com.cn
[2]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China
haoren_zhu@dislab.nju.edu.cn, qzz@nju.edu.cn

Abstract

Nowadays, network function virtualization (NFV) receives widespread concerns from the whole society for its promising application. The main challenge for the deployment of VNF that comes along is the resource allocation of demanded network services in NFV-based network infrastructures. Current offline resource mapping and scheduling algorithms are impractical for continuous VNF requests in data centers, because of the high time consumption. In this paper, we firstly present a resource sharing schema for VNF, including the *automatic monitoring* and *fast switching* mechanisms, which support multiple VNFs effectively share resource in one node. This VNF resource sharing schema improves the acceptance ratio of VNF requests as well as the system utilization. And then, we propose an automatic NFV resource allocation mechanism AutoVNF to optimize the VNF deployment. AutoVNF includes mapping and scheduling algorithms, which automatically allocates available nodes to VNF requests and schedules the execution of VNFs in one VNF queue. Simulations show that AutoVNF has a good performance on acceptance ratio, average flow time, and total cost compared with well-known scheduling algorithms.

Keywords: virtual network function, resource sharing, service function chain

1 Introduction

Specialized physical middleboxes play an important role in network infrastructure. With the dramatic increasing of middleboxes, their drawbacks such as expensive price, high energy costs, less flexibility, and short lifecycle[6] become unacceptable. To prevent the inconveniences of middlebox, a new network architecture framework Network Function Virtualization (NFV) is proposed, and become more and more popular.

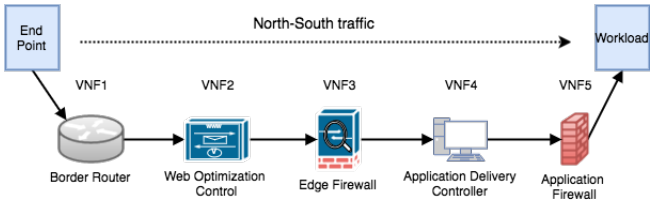

Figure 1: A Service Function Chain Example in A Data Center

Under the paradigm of NFV, traditional middleboxes are managed as single modules of software, named Virtual Network Functions (VNFs). Delivery of multiple VNFs in a sequence, in a data center or among data centers, constructs many requirements on the overall service delivery architecture. Such

Journal of Internet Services and Information Security (JISIS), volume: 7, number: 3 (August 2017), pp. 34-47
*Corresponding author: 163 Xianlin Avenue, Nanjing University, Nanjing 210023, China, Tel: +86-25-89686292

architectures may be termed as service function chaining architectures, and the list of VNFs applied to the traffic is a Service Function Chain (SFC). *Fig*.1 shows an example of north-south traffic virtual network SFC in a data center[7]. Each VNF in the SFC may be deployed in a high performance specific server or a general-purposed server according to different quality of service (QoS). However traffic does not always strictly flow through all the VNFs in that order, and each permutation of these VNFs represents a fully different service function chain.

Nowadays NFV is in a stage of rapid development, and there are still several challenges[6] in NFV, among which how to effectively monitor the resource usage and status of VNF as well as properly allocate resource to VNF are important issues to improve the performance. Resource allocation and scheduling problem (NFV-RA) has also been widely studied in both academic and industry. Traditional NFV-RA solutions are offline algorithms, whose precondition is often assumed that the service requests are known in advance, so orchestrator[4] has enough time to get the global optimal solution through the optimization algorithms and then accordingly deploy VNF allocating proper resources. But in a real system, service requests are submitted one by one, instead of knowing all the requests in advance. So orchestrator should handle incoming requests online, and deploy VNFs to proper servers immediately. To the best of our knowledge, few research focuses on online dynamic NFV-RA problem, and the proposed solutions often simplify the scheduling problem by relaxing some resource constraint.

According to surveys[10, 9, 5] in multi-tenancy mode data centers, each tenant has several different virtual network SFC requests. In order to make full use of resources, it is necessary to delete the VNFs which were in idle status for a long period of time. Science the lifecycle of each SFC various, some longtime running SFCs allow put off the start time while some SFCs may run very short time. So we construct a *resource sharing mechanism* which allows a single server accept multiple VNFs concurrently. Based on this mechanism, some NFV can be deployed to busy nodes and be triggered after waiting for a certain amount of time. That is, all the VNFs assigned to one server will wait in a queue, and run in a certain order. Accordingly, system could accept more virtual network SFC requests, although some of them may have to wait in a queue, it is much better to just reject the request. Actually, this resource sharing mechanism improves the overall performance and resource utilization.

To achieve this resource sharing mechanism, we design an *automatic monitoring mechanism* to effectively get the status of all nodes and a *fast switching mechanism* to quickly shut down and start up VNFs in one node. The *automatic monitoring mechanism* is based on the Finite State Machine (FSM), which converts original active monitoring to a passive monitoring and reduce the overhead of resource monitoring. The *fast switching mechanism* defines a Linux namespace as a minimal allocation unit, which is more lightweight and can reduce switching delay. The experimental results show that this mechanism not only guarantee the running efficiency, but also the acceptance ratio with a limited cost.

This paper presents an advanced virtual network function schema based on resource sharing *AutoVNF*. The contributions are summarized as follows:

(1) **Lower monitoring costs:** We design an automatic monitoring mechanism, so that AutoVNF can detect changes of resources in the substrate network in time without high monitoring costs.

(2) **Higher acceptance ratio:** First, with resource sharing, service requests can be embeded after waiting for a certain amount of time and then more service requests can be accepted. Second, the shceduling based on the fast switching mechanism can make requests more likely to be accepted by reasonable adjustment for the allocated resources.

(3) **Reasonable runtime and service delay:** The global max heap kept by the automatic monitoring mechanism allows AutoVNF to select candidate nodes fastly, and final runtime can be within reasonable runtime. The fast switching mechanism can also ensure service delay within a tolerable range.

The rest of the paper is organized as follows. Section 2 introduces the resource sharing schema and two system mechanisms. Section 3 gives the detailed model and problem description, and section 4 presents the resource allocation mechanism AutoVNF. Section 5 shows experiments results, and Section

6 gives some related work. Section 7 concludes the paper.

2 Resource Sharing Schema for VNF

If the substrate network could not meet the entire SFC resource requirement of VNF, the request will be rejected, even only one VNF could not be deployed. This strict resource allocation policy provides a good QoS guarantee for accept virtual network SFC Request, while it may reduce the resource utilization. Actually, some SFC requests are not delay-sensitive, which may allow a certain delay. For example, a start-up company use the cloud servers to provide online video transcoding services[6, 7], assuming that videos of different formats need to deploy different VNFs. It allows the cloud provider delay its VNFR requests for a certain time, because the waiting time is relatively small compared with the video processing time. With the development of NFV, the cost of deploying/removing NFV is getting lower. Thus, we design a VNF queue for each node and therefor construct a resource sharing schema, where system assigns several VNFs to one node simultaneously, which wait in the VNF queue of the node and will run one by one with FIFO.

In this section, we first define Hungry Index for VNF to evaluate capacity of both single VNF and VNF queue that is composed of several VNFs. And two important mechanisms are presented: *automatic monitoring mechanism* and *fast switching mechanism*. The former mechanism improves the speed of getting node states, thus mapping speed is significantly improved. The second mechanism is based on current sharing model, which reduces the switch delay in scheduling stage by avoiding launching a high-delay server.

2.1 Hungry Index of VNF Queue

Based on the resource sharing model for VNF, we have the following observation.

- If the length of queue is shorter, the number of waiting VNFs is smaller;
- The short total waiting time (T_w) of a VNF queue indicates that the relative processor is faster;
- If the VNF has been running for a long time, it indicates that it may finish soon.

After considering the above factors, here, we define the hungry index *HI* to evaluate whether the server is proper to load a new VNF, which is an important metric to choose available substrate nodes in the allocation or scheduling stage. The Hungry Index *HI* is defined as follows.

$$\mathbf{HI}(que_i) = \gamma \cdot \mathbf{L}(que_i) + \delta \cdot \mathbf{T}_w(que_i) - \eta \cdot \mathbf{T}_r(que_i) \tag{1}$$

where γ, δ and η are constants aimed at scaling the weights of above there factors. These constants are set as $\gamma = 0.4$, $\delta = 0.4$ and $\eta = 0.2$ in this paper.

In scheduling stage, we need to consider the HI of each VNF queue, so original $\mathbf{T}_w(que_i)$ needs to be adjusted as $\mathbf{T}_w(que_i, pos)$, where *pos* represents the position of the VNF in the queue.

Note: the queue length, running time and waiting time all need to be normalized using Min-Max Normalization. To distinguish the node(server) HI and the VNF HI, we denote former as NHI, and the latter as VHI.

2.2 Automatic Monitoring Mechanism

One of the main challenges to achieve online resource scheduling is how to effectively get the status of each node. General centralized implementation of the distributed system is utlizing one or more

control nodes to continuously poll information from each substrate node. However collecting status information is not suitable for current system, because neither long time interval of polling nor short interval cannot obtain a good result. If the time interval of polling is too long, system cannot detect the scheduling opportunity on time; otherwise, it will cause serious system load, and meanwhile frequent polling information will consume siginificant link resources.

We construct an FSM (Finite State Machine) for each node/queue, different operations of node will trigger the state transition of the corresponding FSM. Once the VHI of a VNF has exceeded the critical value, the FSM will trigger the scheduling state; Here all VNFs triggering the scheduling state will be recorded in a global queue, and the controller will complete the scheduling for the global queue by using specific scheduling algorithm. In addition, we maintain a global max heap (size S), used to record S nodes whose NHI values are smaller than others. Based on this monitoring mechanism, the speed of online resource scheduling will significantly improve, while the acceptance ratio is still high.

A simple mechanism example is illustrated in $Fig.2$. The corresponding FSMs for node $1,2,3$ calculate the VHI values of three nodes, shown in $Fig.2.a$. We assume the threshold of VHI and NHI is 20 and 50 respectively. The results calculated by FSM show that the VHI values of VNF2, VNF5 and VNF3 are larger than the threshold, so these three VNFs will be recorded in the global queue shown in subgraph $Fig.2.c$. At the same time, the max heap shown in subgraph $Fig.2.b$ records the last three NHI values. **Note:** if NHI value of each node is larger than the critical value means this VNFR should be rejected or more substrate nodes should be added.

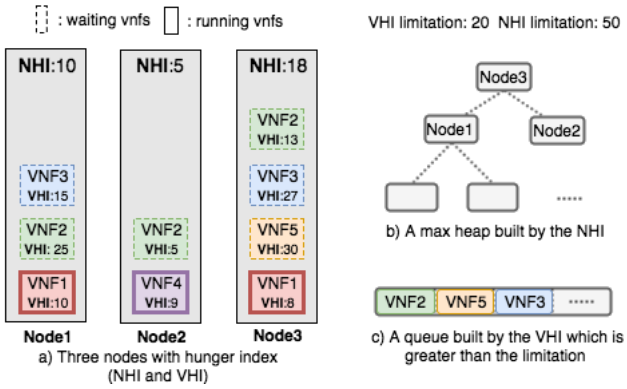

Figure 2: A Simple Introduction for Automatic Monitoring Mechnism.

2.3 Fast Switching Mechanism

Another important issue in resource sharing schema for VNF is how to reduce the overhead of frequently switching VNFs, which is also important for fast failure recovery. The experiments in [8] show that, in Openstack and Opendaylight, even if you launch a very lightweight virtual machine like ClickOS, it needs 4.2s in average, and a general virtual machine or a physic server requires much more time. Here based on our sharing model where a node (virtual machine or physic server) can share multiple VNFs using Linux namespaces for isolation, but all the VNFs should run sequentially.

We design a less costly mechanism, as shown in $Fig.3$. Supposed that data flow needs to pass through the NFVC: FW→DPI→NAT. When the FSM of DPI detects errors or scheduling, the MANO selects a low-NHI node after receiving the message, and then remotely run the DPI process in another namespace. After configuring network rules, the switch is successful. In our many experiments, the delay of starting the namespace remotely is far less than restarting a new virtual machine.

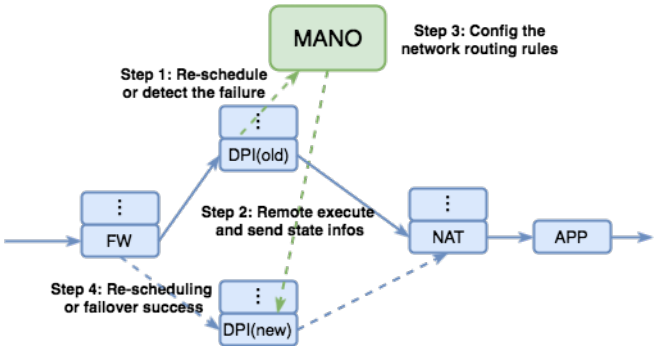

Figure 3: A Simple Introduction for Fast Switching Mechnism.

3 Virtual Network Function Requests and Resource Allocation

In this Section, we first introduce the request model of Virtual Network Function Requests (VNFR) and substrate network model, then we present the resource allocation problem.

3.1 The VNFR Model

Fig.4 depicts two virtual network SFC examples which are defined by VNFR model. In *Fig*.4 the life-cycle type is a very important decision parameter for VNF scheduling. For example, if two optional executing VNF queues involve only a short-lifecycle VNF and a long-lifecycle VNF respectively, generally we prefer the VNF queue with the short-lifecycle VNF. The VNF defined in the request has at least one instance, and its type, the required processing ability, and the number of branches are defined, which indicates the number of the outgoing data streams that are divided (such as the load balancer). The dashed arrows shown in *Fig*.4 indicates the dependencies among VNFs and the pointed VNF. Meanwhile, each links need to define the ratio of the output data rate to input. "*Ratio* : 60%" denotes that the output data rate is only 60% of the input.

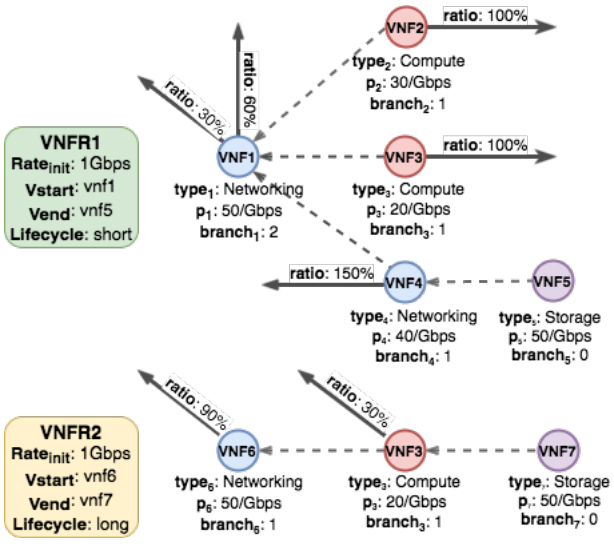

Figure 4: Two VNFR Examples

3.2 The Substrate Network Model

The substrate network is modeled in *Fig*.5.a. Each node in the substrate network represents a virtual machine or a physic server which is the minimal resource allocation unit. Each link shows the distance between two nodes, which means if there are two links between two nodes, the distance between the two nodes is two hops. Original approaches all try to allocate one VNF to a dedicated virtual machine or a high volume physical server, result in the high resource consumption, which eventually reduce the acceptance ratio of VNF requests. Actually, as we discussed above, most VNFRs have short life cycle and would be lightwight[10], while restart a new virtual machine or a physic server is time consuming[8]. Therefore, our resource sharing schema constructs each node as a queue which allows processing multiple VNFs one by one, if we won't make a new schedule.

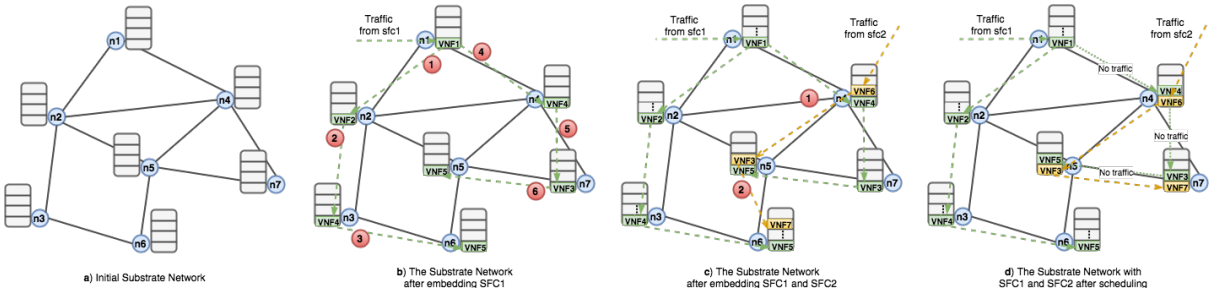

Figure 5: Four Different Periods for the Substrate Network. Subgraph **a)** shows the initial substrate network; Subgraph **b)** shows the substrate network after embedding SFC1; Subgraph **c)** shows the substrate network after embedding SFC1 and SFC2; Subgraph **d)** shows the substrate network with SFC1 and SFC2 after scheduling.

The following table is a summary of all the notations defined in this paper.

Table 1: Notation

Model	Notation	Description
VNFR	$VNFR_i/VNFR_s$	the ith request in the VNF request sequence($VNFR_s$)
	$type_i(VNFR_j)$	the type of ith VNF in $VNFR_j$
	$p^i_{vnf}(VNFR_j)$	the expected processing ability of ith VNF in$VNFR_j$
	$branch_i(VNFR_j)$	number of branches of ith VNF in $VNFR_j$
	$l_{in}(VNF_i)$	all the incoming links of ith VNF
	$l_{out}(VNF_i)$	all the outgoing links of ith VNF
	$ratio(VNF_i,VNF_j)$	the data flow ratio of ith and jth VNF
	$rate_{init}(VNFR_i)$	the initial data rate of ith VNF
	$V_{start}(VNFR_i)$	the first VNF of the ith VNFR
	$V_{end}(VNFR_i)$	the last VNF of the ith VNFR
Substrate Network	G=(N,L)	N denotes node and L denotes link
	$L(Que_i)$	number of VNF in ith queue
	$T_w(Que_i)$	the total waiting time of all the VNFs in $i$the queue
	$T_w(Que_i, pos)$	the waiting time of posth VNF in ith queue
	$T_r(Que_i)$	the processing time of first VNF in ith queue
	P^i_{node}	the total processing ability of ith node
	B^i_{link}	the bandwidth of ith link
	$Type_i$	the type of ith node, which could be one of computing, storage or netwok

3.3 Objectives of Resource Allocation

For a VNFR, system should allocate proper resources to all the VNFs. We should deploy all the VNFs to a node and wait in its VNF queue, which we should keep the high acceptance ratio and low latency. Our main objectives are minimizing the average flow time and cost, which are defined respectively.

Average Flow Time We define the average flow time AFT as a metric of average VNF processing speed. The average flow time AFT is defined as in equation (2).

$$\mathbf{AFT} = \frac{1}{n}\sum_{i=1}^{m}(t_{fin}^{i} - t_{arr}^{i}) \tag{2}$$

where n represents the number of VNFR, t_{fin}^{i} represents the time point when the processing of the last function of a service is completed and t_{arr}^{i} represents the time point when the service arrived.

Cost here we define the scheduling cost C as the main overhead when making a scheduling, both VNF buffer cost and time cost are in consideration. The cost C is defined as in equation (3).

$$\mathbf{C} = \sum_{i=1}^{m}(\alpha \cdot b_i + \beta \cdot (t_{fin}^{i} - t_{arr}^{i})) \tag{3}$$

where α and β are constants aimed at scaling the weights of buffer and time resources. These constants are set as $\alpha = \beta = 0.5$ in this paper. And b_i represents the total buffer size occupied by all the VNFs of the i-th VNFR.

The VNE problem is known to be NP-hard[1]. This VNF resource allocation problem is also NP-hard. In this paper we design an online heuristic mapping algorithm to allocate resource to related VNF and a fast schedule strategy which can get higher acceptance ratio with reasonable runtime and service delay.

4 Resource Allocation Algorithms for VNFR

In this section, we present the automatic resource allocation schema AutoVNF in detail and explain this mechanism through an example.

4.1 Resource Allocation Mechanism AutoVNF

The main resource allocation procedure is shown in Algorithm 1, including mapping and scheduling. For an incoming VNFR, system constructs the VNF dependance graph according to VNFR model (line 4); Then recursion method *RecProcessing* will complete the mapping. The scheduling process is a background running process, as long as the global queue is not empty (ie. there is some more VNFR need to scheduled), each VNF will be scheduled in turn (shown in algorithm 3).

The main mapping process (*RecProcessing*) shown in algorithm2 is a recursive function similar with the mapping algorithm (CoordVNF) shown in[2], the main difference is: CoordVNF selects substrate nodes only in limited search radius as candidate nodes for each VNF; Instead, we obtain the last n nodes of NHI values directly from the global max heap (according to the automatic monitoring mechanism), and sort the nodes in a specific order (line 6). Then, it chooses a node that meets all the constraints. Once the mapping is successful, the data rate of all the branches of the VNF will be calculated again (line 13) and recursively call *RecProcessing* to finish the mapping (line 14). *RecProcessing* mapping process will cover all the possible solutions, while the runtime is still in reasonable range.

```
Define M_s as a global mapping set.
for all VNFR_i ∈ VNFR_s, M_i ∈ M_s do
    M_i ← {}
    Construct the corresponding graph VNFFG_i
    RecProcessing(VNFFG_i,G,N_start,rate_init(VNFR_i),M_i)
end for
while True do
    highHungerVNF_s ← getHighHungerVNFsFromQue()
    if highHungerVNF_s ≠ ∅ then
        Schedule(G,VNFFG_s,M_s,highHungerVNF_s)
    end if
end while
```

Algorithm 1: AutoVNF(VNFR$_s$, G)

```
VNF_s ← nextVNF_s(VNFFG_i, M)
if VNF_s = ∅ then
    return M
end if
possibleSNode_s ← getPossibleSNodesFromMaxHeap()
sortPossibleSNodes(possibleSNode_s)
for each sn ∈ possibleSNode_s do
    M' ←mapNodeAndLinkDemands(sn,prevSNode,rate)
    if mapping is successful then
        success ← true, M'' ← M'
    else
        continue
    end if
    for all l ∈ l_out(currentVNF) do
        rate' ← rate · ratio(currentVNF, v)
        M_sn ← RecProcessing(VNFFG_i,G,sn,rate,M'')
        if M_sn ≠ ∅ then
            M'' ← M'' ∪ M_sn
        else
            success ← false, rollback(M''), break
        end if
    end for
    if success == true then
        return M''
    end if
end for
return ∅
```

Algorithm 2: RecProcessing(VNFFG$_i$, G, prevSNode, rate,M)

The main scheduling process shows in algorithm 3. Firstly, if the VNF V_r finished, V_r will be removed from the queue and the successor VNF V_m start running (line 4). Then, if V_r is idle, it will be moved to the tail of VNF queue and the successor VNF V_m start running (line 6-7). Finally, if the hungry index of some VNF VHI is higher than the threshold, the mapping result will be removed and a new mapping procedure will be triggered.

```
 1: Define V_r as a running VNF in a queue
 2: Define V_w as a waiting VNF in a queue
 3: if the lifecycle of the V_r ends then
 4:    remove V_r and start subsequent VNF, V_w in the queue
 5: else if V_r becomes idle then
 6:    place V_r in the back of the queue
 7:    start subsequent VNF, V_w in the queue
 8: else if there are VNFs with high hunger index then
 9:    for each v ∈ highHungerVNF_s do
10:       eraseMapping(v, M)
11:       sNode_s ← getPossibleSNodesFromMaxHeap()
12:       sortPossibleSNodes(sNode_s)
13:       for each sn ∈ sNode_s do
14:          M' ←mapNodeAndLinkDemands()
15:          if mapping is successful then
16:             M ← M' ∪ M
17:             return
18:          end if
19:       end for
20:       undoEraseMapping(v, M)
21:    end for
22: end if
```

Algorithm 3: Schedule(G,VNFFG$_s$,M,highHungerVNF$_s$)

4.2 An Example

The subgraph **b)** and **c)** in *Fig*.5 depict a simplified example when embedding $SFC1$ and $SFC2$ based on the scenario presented in *Fig*.4. Here we assume that there is no other SFCs in the data center when SFC1 arrives and SFC2 arrives after SFC1.

The whole process of embedding SFC1 involves several steps as follows. **(1)** We first embed the VNF1 in *Fig*.4. Assuming node n1 and its links can meet VNF1's demands, we directly embed VNF1 at node n1. **(2)** *Fig*.4 shows that VNF1 has two branch links, first we embed the first link. At the same time, according to the dependency, VNF1's optional next hop could be VNF2, VNF3 or VNF4, here we select VNF2 as its next hop. Then we need to select the substrate node, observing subgraph *Fig*.5.b, only node n2 or n3 meets demands, finally node n2 is chosen. **(3)** There is no VNF which has a dependency on VNF2, in order to avoid introducing unnecessary VNF, we hope to directly select the final VNF5. However VNF5 is dependent on VNF4, so the next two hops are VNF4 and VNF5, which are respectively embedded in node n3 and n4. **(4)** The next step we need to do is embedding the second branch links of VNF1, finally results are shown in subgraph. **(5)** Since VNF3 is still not be embedded, the next hop of VNF4 is VNF3 and the substrate node is n7. **(6)** Finally, the end of this branch, VNF5, is embedded in node n5.

After a period of time, the SFC2 reaches the data center. Like the steps of mapping SFC1, each VNF of SFC2 will be embedded in substrate nodes according the NHI value from the global max heap. Finally results are shown in subgraph *Fig*.5.c.

According to the fast switching mechanism, when the right branch of SFC1 becomes idle, each VNF of SFC2 will be re-scheduled and finally results are shown in subgraph *Fig*.5.d.

5 Evalution

In this section, we introduce the simulation setup and compare AutoVNF with well-known scheduling mechanism CoordVNF[2].

5.1 Simulation Setup

AutoVNF is implemented and evaluated based on the ALEVIN simulation framework [3]. The arrival number of VNFR follows the Poisson distribution, assuming that the total number of VNFRs is 150, and the average number of VNFR arrivals per time unit is 5. In addition, we define the ratio of long-lifecycle and short-lifecycle requests as 2 to 8. Other main parameters used in these simulations for creating substrate nodes and services are chosen randomly following a range from minimum to maximum values shown in *Table*.2. In adddtion, 5 VNFs are assigned to each VNFR. Dependencies between VNFs are randomly chosen. Initial data rate of the VNFRs is set to 50 bandwidth units, and the amount of required processing capacities is set to 1 capacity unit/bandwidth unit.

Table 2: Simulation Parameter Ranges

Parameter	**Minimum**	**Maximum**
Number of nodes	50	50
Capacity per node	50	100
Bandwidth per link	50	100
Queue length per node	1	3
Lifecycle per VNFR	1	10
Tolerated delay	1	3

5.2 Results Analysis

The results of the simulations are shown in *Fig*.6. Following is the detailed description of the results.

Time Complexity When each VNF in a SFC with length l is embeded, CoordVNF uses BFS to gain the candidate substrate nodes. Its time complexity is $O(|n|+|e|)$, where n and e represent respectively the number of nodes and the number of links traversed in each step of BFS. If the search radius is too large, CoordVNF will be slower; On the contrary, too small search radius will cause seriously decrease of the request acceptance ratio. So considering a tradeoff between speed and acceptance ratio, CoordVNF set the search radius as 2. In our experiment, AutoVNF uses the automatic monitoring mechanism. After each placement of VNFs, the global max heap will adjust itself and time complexity is $O(logS)$, where

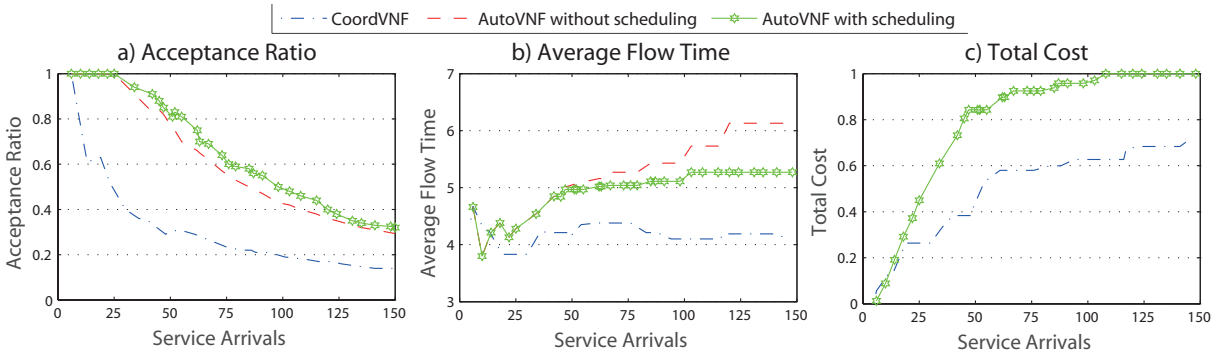

Figure 6: Performance comparison for three different aspects.

S is the size of the max heap. So AutoVNF can ensure global search field as large as possible, but also executing speed as fast as possible.

Acceptance Ratio *Fig.6.a* shows the variation of the VNFR acceptance ratio with service arrivals. Higher acceptance ratio means higher resource utilization and higher income for service providers. According to the results, it shows that AutoVNF has much higher acceptance ratio than CoordVNF. On the one hand, this is because AutoVNF allows a deferred execution of services, and put them in the node queue. The other is that when AutoVNF search for candidate substrate nodes, its search field is much larger and more available nodes can be obtained. In addition, with the usage of scheduling, the acceptance ratio increases by about 10%, mainly because the scheduling algorithm of AutoVNF makes some VNFRs in hungry state able to run, which shortens the length of the queue and then allows accepting more requests.

Average Flow Time *Fig.6.b* shows the variation of the average flow time with service arrivals. The average flow time reflects the average processing speed of VNFRs. Results show that the average flow completion time of the CoordVNF is four time units, AutoVNF without scheduling increases by about 2 time unit in the worst case because of the waiting time in queues. After using the scheduling, AutoVNF makes the average flow time drop by 1 time unit and final average flow time is about 5 time unit. We think this is reasonable and tolerable since each VNFR has a tolerance delay with a range from 1 to 3.

Total Cost *Fig.6.c* shows the variation of the total cost with service arrivals. In our experiment, we only think of two overhead factors: buffer and time resources. Here buffer size and time unit need to be normalized using Min-Max Normalization and the weights are respectively 0.5 and 0.5. Results show AutoVNF's overhead is much higher than the CoordVNF, but this is also essential for using the queuing mechanism.

6 Related Work

Now the NFV resource allocation problem can be divided into three sub-problems[6]: (1) virtual network SFC composition; (2) resource allocation (embedding); (3) task scheduling. And some existing approaches which aim at solving one or more sub-problems will be described briefly in this section.

Mehragdam *et al.*[9] aims at VNF chain composition and embedding. It defines these two problems as two NP-hard problems respectively: Location-Routing Problem(LRP)[11, 12] and Virtual Network Empebedding(VNE)[5]. Then a greedy heuristic method is proposed to solving the first sub-problem,

where the data rate of total flows will reduce by preferentially choosing the function of lower ratio of outgoing to incoming data rate. When the VNF forward graph has been composed using aforementioned method, the second sub-problem could be solved by using a Mixed Integer Quadratic Constrained Program (MIQCP) with regard for some different optimization criteria. Solving the second sub-problem with an exact algorithm will cause huge runtime overhead, so this method is only suitable for offline small-scale scenarios. Moreover, since solving the two problems separately, results got from the first step could not meet some limitation of the second step.

Beck *et al.*[2] propose a recursive heuristic algorithm to coordinately solve first two sub-problems, instead of considering them individually. With this method, each step, an optimal VNF selected according to the dependency in the VNFRs, will be chosen to compose a viable VNF-FG, and at the same time embedded in a substrate node which can satisfy all resource constraints. When a VNF cannot be embedded, performs backtracking by going back to the last successfully mapped VNF and looks for a another optional VNF in order to rapidly find a feasible solution. It should be noted that this method has a very important parameter called *maxPathLength* which limits the maximum distance between two connected VNF instances that are embedded into the substrate networks. For getting a faster runtime, a small value should be chosen for this parameter, which, however, will cause a lower VNFR acceptance, because a small value means the search area will be smaller and many possible substrate nodes are uncovered.

A few researches aiming at the last sub-problem (scheduling) of the NFV-RA problem. Riera *et al*. [14], [13] provide the first formalization of the scheduling problem in NFV as a Resource Constrained Project Scheduling Problem. However, no solution is proposed to solve the sub-problem in the two papers. Mijumbi *et al.*[10] proposes three greedy and meta-heuristic (tabu search) approaches to reducing the flow execution time. The VNF-CC sub-problem is not considered and the tabu search approach is only suitable for ideal situation.

7 Conclusion

In the paper, we present a novel resource sharing schema for VNF. We design the automatic monitoring and fast switching mechanisms, based on which we propose AutoVNF including an online heuristic resource allocation and a fast task scheduling strategy. We have extensive simulations to evaluate different aspects of AutoVNF including time complexity, acceptance ratio, average flow time, and total cost. And experiment results show that our algorithms can keep high acceptance ratio with reasonable runtime and service delay.

References

[1] E. Amaldi, S. Coniglio, A. M. Koster, and M. Tieves. On the computational complexity of the virtual network embedding problem. *Electronic Notes in Discrete Mathematics*, 52:213–220, June 2016.

[2] M. T. Beck and J. F. Botero. Coordinated allocation of service function chains. In *Proc. of the 2015 IEEE Global Communications Conference (GLOBECOM'15), San Diego, California, USA*, pages 1–6. IEEE, December 2015.

[3] M. T. Beck, C. Linnhoff-Popien, A. Fischer, F. Kokot, and H. De Meer. A simulation framework for virtual network embedding algorithms. In *Proc. of the 16th International Telecommunications Network Strategy and Planning Symposium (Networks'14), Funchal, Portugal*, pages 1–6. IEEE, September 2014.

[4] ETSI. European telecommunications standards institute, 2016. `http://www.etsi.org/technologies-clusters/technologies/nfv`, [Online; Accessed on August 1, 2017].

[5] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys & Tutorials*, 15(4):1888–1906, February 2013.

[6] J. G. Herrera and J. F. Botero. Resource allocation in nfv: A comprehensive survey. *IEEE Transactions on Network and Service Managemnet*, 13(3):518–532, August 2016.

[7] S. Kumar, M. Tufail, S. Majee, C. Captari, and S. Homma. Service function chaining use cases in data centers. IETF Internet-draft (work in progress), January 2016. `https://tools.ietf.org/html/draft-ietf-sfc-dc-use-cases-04`, [Online; accessed August-2017].

[8] X. Li and C. Qian. An nfv orchestration framework for interference-free policy enforcement. In *Proc. of the 36th IEEE International Conference on Distributed Computing Systems (ICDCS'16), Nara, Japan*, pages 649–658. IEEE, June 2016.

[9] S. Mehraghdam, M. Keller, and H. Karl. Specifying and placing chains of virtual network functions. In *Proc. of the 3rd IEEE International Conference on Cloud Networking (CloudNet'14), Luxembourg, Luxembourg*, pages 7–13. IEEE, October 2014.

[10] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and S. Davy. Design and evaluation of algorithms for mapping and scheduling of virtual network functions. In *Proc. of the 1st IEEE Conference on Network Softwarization (NetSoft'15), London, UK*, pages 1–9. IEEE, April 2015.

[11] G. Nagy and S. Salhi. Location-routing: Issues, models and methods. *European Journal of Operational Research*, 177(2):649–672, March 2007.

[12] C. Prodhon and C. Prins. A survey of recent research on location-routing problems. *European Journal of Operational Research*, 238(1):1–17, October 2014.

[13] J. F. Riera, E. Escalona, J. Batalle, E. Grasa, and J. A. Garcia-Espin. Virtual network function scheduling: Concept and challenges. In *Proc. of the International Conference on Smart Communications in Network Technologies (SaCoNeT'14), Vilanova i la Geltru, Spain*, pages 1–5. IEEE, June 2014.

[14] J. F. Riera, X. Hesselbach, E. Escalona, J. A. Garcia-Espin, and E. Grasa. On the complex scheduling formulation of virtual network functions over optical networks. In *Proc. of the 16th International Conference on Transparent Optical Networks (ICTON'14), Graz, Austria*, pages 1–5. IEEE, July 2014.

Author Biography

Wei Huang received the PhD degree in computer science in 2009. She is an associate professor and vice dean of the school of computer engineering, Nanjing Institute of Technology. Her research interests include distributed system, cloud computing and software engineering.

Haoren Zhu is a master student in the Department of Computer Science and Technology, Nanjing University. His research focuses on virtual network.

Zhuzhong Qian received the PhD degree in computer science in 2007. He is an associate professor in the Department of Computer Science and Technology, Nanjing University, P.R. China. He is also a research fellow of the State Key Laboratory for Novel Software Technology (http://keysoftlab.nju.edu.cn/). He is a member of the IEEE and ACM. His current research interests include cloud computing, distributed systems, and datacenter networking. He is the PI and chief member of several national research projects on distributed computing and networking. He has published more than 60 research papers in related fields.