Heterogeneous Network Policy Enforcement in Data Centers

Lin Cui*, Fung Po Tso†, Weijia Jia‡,
*Department of Computer Science, Jinan University, Guangzhou, China
†Department of Computer Science, Loughborough University, UK
‡Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai China.
Email: tcuilin@jnu.edu.cn; f.p.tso@lboro.ac.uk; jia-wj@cs.sjtu.edu.cn

***Abstract*—With the emergence of network function virtualization, data center start to deploy a variety of network function boxes (NFBs) in both physical and virtual form factors in order to combines inherent efficiency offered by physical NFBs with the agility and flexibility of virtual ones. However, existing schemes are limited to exclusively consider physical or virtual NFBs, which may reduce the performance efficiency of services running atop. In this paper, we propose a Heterogeneous NetwOrk Policy Enforcement scheme (HOPE) to overcome these challenges. An efficient algorithm that can closely approximate optimal latency-wise NF service chaining is proposed. The experimental results have also shown that HOPE can outperform greedy algorithm by 25% in terms of network latency and is 56x more efficient than naive depth-first search algorithm.**

I. Introduction

Cloud data centers deploy a great variety of network functions (NFs), such as firewall (FW), intrusion prevention/detection system (IP/DS), deep packet inspection (DPI), load balancer (LB), and etc., at various points in the network topology to safeguard networks and/or improve application performance [1]. Each NF is responsible for specific treatment of received packets, including forwarding, dropping, rate-limiting, inspecting and so on. Various permutations of NFs form an ordered chain, i.e., service chain [2], which is defined by network policy [3] and must be applied to packets.

Nowadays, NFs are either embedded in purpose-built proprietary hardware middleboxes or run as virtual instances on top of commodity servers through network function virtualization (NFV). We term both hardware middleboxes and virtualized servers for NFs as Network Function Boxes (NFBs). Physical NFBs are more efficient because they are built with dedicate hardware for optimizing the performance of specific functions but are proprietary and hence less extensible. On the other hand, virtualized NFBs have the agility for rapid on-demand deployment and programmability for software automation but they are less efficient due to virtualization overhead, resource sharing, and general-purpose hardware they sit atop [4]. Obviously, a NF can be independently allocated to different NFBs in the network or collocated with other NFs within a single NFB [2]. In addition to hardware middleboxes and general-purpose NFV servers, some simple NFs such as firewall and NAT can also be efficiently implemented in SDN switches [5].

Clearly, these distinctively different NFBs present an enormous opportunity for data centers by adopting mixture of both form factors in order to captialize on both efficiency and flexibility [2]. Nevertheless, coming with this heterogeneous are significant challenges on the correct implementation of network policies:(1) Support for deployment of network policies is limited exclusively to physical or virtualized NFBs, there is no tools for supporting mixture of both [3];(2) Migration of virtual instances of NFs will involve change of end-to-end path that will break legacy VLAN partition [6]; (3) The performance of virtual NFs are subject to the compute and storage capacities of commodity servers. This will lead to unpredicted performance such as end-to-end latency [7].

While the first and second challenges can be partially tackled with small tweaks on top of our previous work Sync [6], the third challenge remains the most prominent and needs solving urgently because data centers host applications that are largely latency-sensitive and are prone to unpredictable slowdown along the end-to-end links [8]. Existing works only combat latency in network switches and/or protocols running atop and overlook latency arises from NFBs [9].

In this paper, we propose a Heterogeneous NetwOrk Policy Enforcement scheme (HOPE) which meets this requirement. Our experimental evaluation demonstrates that HOPE can achieve optimal placement of NFs amongst heterogeneous NFBs.

The remainder of this paper is structured as follows. Section II describes the problem formulation and the model of HOPE. An efficient greedy scheme is proposed in Section III, followed by the performance evaluation of HOPE in Section IV. Section V concludes the paper.

II. Problem Modeling

A. Overview

We consider a heterogeneous environment where NFs can be implemented different kinds of NFBs: (1) *Hardware middleboxes* are vendor specific, proprietary boxes for providing specific NFs. They are optimized for performance and less extensible. (2) *NFV servers* are virtualized that can run multiple, and theoretically, any types of virtual NFs. (3) Some simple NFs can also be implemented on *switches or routers* such as VPN, simple firewalls, LBs. They are amongst hardware middleboxes. However, SDN also allow us to exploit the OpenFlow switches to increase the performance of service chain by installing some rules (i.e., NF) to their tables [5].

We anticipate that the heterogeneous implementation of NFs will exist for the foreseeable future.

Denote $\mathbb{B} = \{b_1, b_2, \ldots\}$ to be the set of all NFBs in a data center. For a NFB b_i, $b_i.capacity$ denotes the maximum processing capability of b_i, measuring in number of packets per second (*pps*), e.g., 3800 *pps* [10]. $b_i.typeset$ specifies the set of supported NF types on b_i. NFV servers theoretically support all types of NF, while hardware MBs and switches can only support one or few types of NFs. We assume that memory space of NFBs are enough to accommodate states information of all NFs, i.e., bottleneck is the processing capacity.

Let $\mathbb{N} = \{n_1, n_2, \ldots\}$ be the set of all NF instances in data center. For a NF n_i, the property $n_i.type$ defines the function of n_i, e.g., *IP/DS*, *LB*, or *FW*. $n_i.capacity$ is essentially the processing capacity requirement of n_i in *pps*. $n_i.location$ is the NFB that currently hosts n_i. This set of NFs in $\mathbb{N}$ may belong to different applications, and are deployed and configured by a centralized *Policy Controller* [11].

Traffic in data center is largely flow-based [12]. In light of this, we define data center traffic as $\mathbb{F} = \{f_1, f_2, \ldots\}$. For each flow $f_i \in \mathbb{F}$, $f_i.src$ and $f_i.dst$ specify the source and destination VMs of f_i respectively, e.g., $f_i.src = v_1$ and $f_i.dst = v_2$. The data rate of $f_i.rate$ is represented by data exchanged from VM $f_i.src$ to VM $f_i.dst$ per time unit[1].

The set of network policies is $\mathbb{P}$, which can be defined by administrators. For each $f_i \in \mathbb{F}$, there is a policy p_i. $p_i.chain$ defines the sequence of NFB types that all flows matching policy p_i should traverse in order, e.g., $p_i.chain = \{n_1, n_2, n_3\}$, where, for example, $n_1.type = FW, n_2.type = IPS, n_3.type = Proxy$. Specially, $p_i = \emptyset$ means f_i is not governed by any policies. $p_i.len$ is the length of $p_i.chain$.

$p_i.chain$ must be assigned to appropriate NFBs beforehand, and we assume there are enough NFBs to accommodate all required NFs. Since we consider heterogeneous NFs, there are various possible locations for each NF in $p_i.chain$. For example, in the above example of p_i, $n_1.location$ could be a core router, $n_2.location$ could be a hardware NFB, and $n_3.location$ could be a NFV server. An example of service chain is given in Figure 1.

B. Delays with network functions

There are many metrics to measure the efficiency of NF placement (service function chaining) for a policy such as communication cost [6][14]. In this paper, we mainly focus on the latency of a policy flow.

The total delay of a flow includes transmission delay among adjacent NFs and processing delay of NFs in the service chain.

1) Transmission delays: In order to steer traffic to the service chain, either Policy Based Routing (PBR) or VLAN stitching can be used in data centers [2]. For either case, *the intended solution in this paper should be unaware of these schemes and is general and applicable to the schemes.* So, we do not consider the detailed routing between two NFBs.

[1]There are a handful of research literature, e.g., [13], about deriving real time traffic matrices in data center networks.

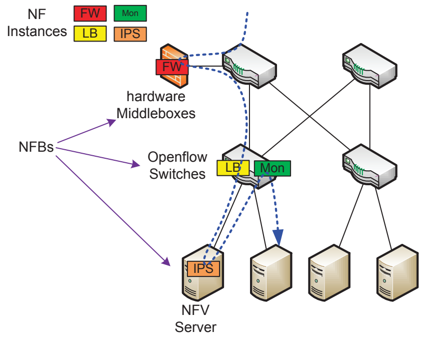


Fig. 1: Service Chain Example: $FW \rightarrow LB \rightarrow IPS \rightarrow Monitor$

Since, in production data centers, the transmission delay of links in its path are relatively stable and can be easily obtained/estimated through large-scale measurement [15], we assume the transmission delay between two NFs is known and can be obtained through the controller. The controller will maintain a transmission delay matrix D, $D(n_i, n_j) = D(n_j, n_i)$ is the delay between n_i an n_j. $D(n_i, n_j) = -1$ if the delay is unknown or they are unreachable.

2) Processing delays of network functions: We define service time t_s^i as the time that n_i takes to process a packet. Since many NFs such as firewalls and load balancers only process packet headers of which sizes are fixed, ignoring variable length data payloads. Thus, the service time t_s^i is a constant [16]. Specially, considering the processing capacity $n_i.capacity$ of n_i, $t_s^i = 1/n_i.capacity$.

C. Heterogeneous network policy enforcement problem

The expected delay for flow constrained by policy p_i is:

$$\begin{aligned} T(p_i) &= D(f_i.src, p_i.chain[1]) \\ &+ \sum_{j=1}^{p_i.len-1} (D(p_i.chain[j], p_i.chain[j+1]) + t_p(p_i.chain[j])) \\ &+ D(p_i.chain[p_i.len], f_i.dst) \end{aligned} \tag{1}$$

We aims to reduce the total delay by efficiently placing NFs onto heterogeneous NFBs while strictly adhering to network policies. Let A be an allocation of NFs. $A(n_i)$ is the NFB which hosts n_i, and $A(b_j)$ is the set of NFs hosted by b_j.

The *Heterogeneous Network Policy Enforcement problem* is defined as follows:

Definition 1. *Given the set of* $\mathbb{F}$, $\mathbb{P}$, *NFBs* $\mathbb{B}$ *and* D, *we need to find an appropriate allocation of NFs* A, *which that minimizes the total expected end-to-end delays of the network:*

$$\begin{aligned} &\min \sum_{f_i \in \mathbb{F}} T(p_i) \\ &s.t. \quad A(n_i) \neq \emptyset \ \&\& \ |A(n_i)| = 1, \forall n_i \in p_k.chain, \forall p_k \in \mathbb{P} \\ &\qquad \textstyle\sum_{n_i \in A(b_j)} n_i.capacity < b_j.capacity, \forall b_j \in \mathbb{B} \end{aligned} \tag{2}$$

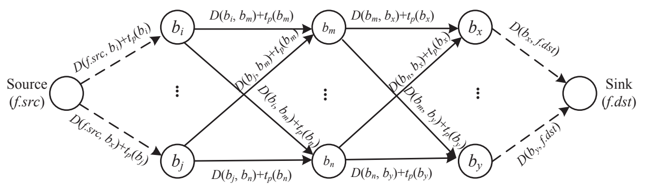


Fig. 2: Example of service chain network for flow f and $l = 3$.

The first constraint ensure that NFs of all service chains are appropriately accommodated by one NFB. The second constraint is the capacity constraint of all NFBs. It can be easily proven that the above problem is NP-Hard, by reducing from the Multiple Knapsack Problem (MKP).

III. Heterogeneous Policy Enforcement

In this section, we introduce *HOPE*, a Heterogeneous NetwOrk Policy Enforcement scheme.

A. Service chain network

We consider an online solution which process one service chain at a time when a new policy/flow requirement arrives. Suppose a flow f_i, constrained by p_i, arrives and we need to find appropriate NFBs to accommodate all NFs in $p_i.chain$ with an objective to minimize its total expected delay $T(p_i)$. We define its *Service Chain Network* $G = (V, E)$, which is a $p_i.len$-tier directed graph. Nodes in the jth tier are NFBs defined by B_j:

$$B_j = \{b_k | p_i.chain[j] \in b_k.typeset \text{ and } \sum_{n \in A(b_k)} n.capacity + capacity \leq b_k.capacity, \forall b_k \in \mathbb{B}\} \tag{3}$$

For a node x in jth ($j \leq p_i.len - 1$) tier and y in $(j+1)$th tier, there is a directed edge from x to y if y is reachable from x and weight of the edge is $D(x, y) + t_p(y)$. It is possible that both x and y are the same NFB. In this case, $D(x, y) = 0$. Flow originates from the source ($f_i.src$) and terminate at the sink ($f_i.dst$). For a node x in 1st tier, the weight of edges from $f_i.src$ to x is $D(f_i.src, x) + t_p(s)$. For a node y in lth tier, the weight of the directed edges from y to $f_i.dst$ is $D(y, f_i.dst)$. Figure 2 shows an example of service chain network.

B. Shortest service chain path

Clearly, the route with smallest expected latency for a flow is the shortest path from source to sink, which we referred as ssp (Shortest service chain path). However, since nodes in different tiers of the service chain network can be the same NFB with limited capacity, we can not simply re-use traditional shortest first path algorithms, e.g., Dijkstra, Floyd.

The difficulty here is that two nodes that belong to different tiers in the service chain network, say x and y, may be in the same NFB and share the same capacity. If we assign f_i to x, it may saturate the NFB such that y can not further accept f_i. In this case, we call them *conflict nodes*. A path from the source will be *blocked* by the latter one of the *conflict nodes*.

Algorithm 1 SSP:Shortest Service Chain Path

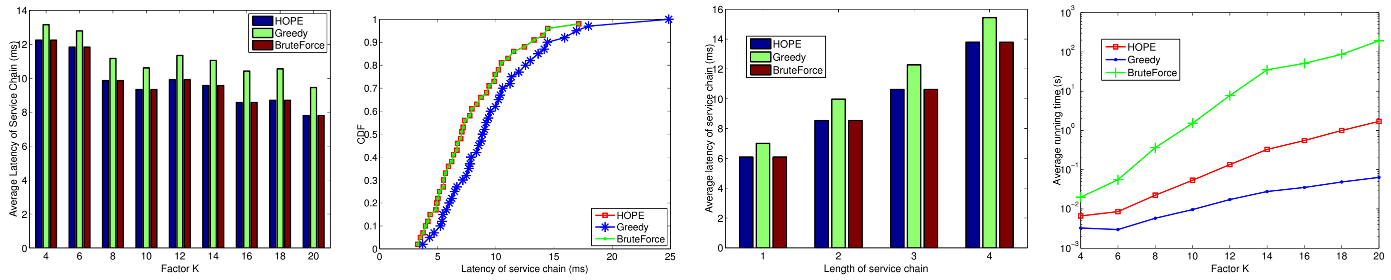
```
Input:  Service chain Network $G(V, E)$, $f_i$
Output: shortest service chain path to $f_i.dst$
 1: $S \leftarrow \emptyset$
 2: $d(v) \leftarrow \infty, \forall v \in V$
 3: $prev[v] \leftarrow$ undefined, $\forall v \in V$
 4: $d(f.src) \leftarrow 0$
 5: while $S \neq V$ do
 6:     $u \leftarrow argmin_{v \in V \setminus S}\ d(v)$
 7:     if $u == f_i.dst$ then
 8:         break
 9:     end if
10:     $S \leftarrow S \cap \{u\}$
11:     $n_k \leftarrow$ NF in $p_i.chain$ that will be placed in $u$
12:     for each neighbor $v$ of $u$ do
13:         if $d(v) > d(u) + D(u, v)$ then
14:             if $v \notin getPath(prev, u)$ or $\sum_{n_j \in A(v)} n_j.capacity + n_k.capacity \leq v.capacity$ then
15:                 $d(v) \leftarrow d(u) + D(u, v)$
16:                 $prev[v] \leftarrow u$
17:             end if
18:         end if
19:     end for
20: end while
21: return $getPath(prev, f_i.dst)$
```

Hence, we design the SSP (Shortest Service Chain Path) to find the shortest path in this situation, as shown in Algorithm 1. Conflict nodes are handled in line 14. The shortest service chain path are maintained in $prev$ and can be obtained through function $getPath()$. We can easily derived that Algorithm 1 can always output a shortest service chain path. Due to page limitation, detailed proof is not shown here.

IV. Evaluation

We have evaluated the performance of HOPE through extensive simulations in a fat-tree data center topology with factor k ranged from 4 to 20 meaning that there are at most 2000 servers and 500 switches in these setups. Each NFB in our simulations is modeled with random residual capacity (number of packets it can process per second) and a set of NF types that it supports. NFBs are implemented through OpenFlow switches, hardware middleboxes or NFV servers. Each service chain is comprised of 1∼4 NFs including FW, IPS, RE, LB and (traffic) Monitor [2]. A centralized controller is implemented to collect all network information that is needed, as defined in Section II to perform the HOPE scheme. To compare the performance of HOPE, we have also implemented two other approaches: *Greedy*, which always tries to assign NFBs with the lowest estimated latency, and *Brute-force*, which gives the optimal results but is not suitable for large-scale network.

Figure 3a shows the average latency of all service chain under different network scales with the factor k of fat-tree ranging from 4 to 20. It shows that on average HOPE can



(a) Average latency for various network scale (b) Latency of service chain for $k = 20$ (c) Average latency for various length of service chain (d) Average running time

Fig. 3: Performance Comparison

always find a service chain path with the same latency as Brute-force, which is optimal. The Greedy approach can deviate from both HOPE and Brute-force by up to 23%.

Figure 3b shows that HOPE and Brute-force schemes have identical CDF of latency for all policies for a large scale network when $k = 20$. Particularly, they can outperform Greedy scheme by 38% at 99 percentile. Figure 3c reveals that average latency increase linearly with the length of service chain when all NFBs have sufficient capacity for accommodating all NFs.

Figure 3d shows the average total running time to process a policy increases exponentially for all schemes. Greedy is the fastest methods, consuming only $50ms$ and $63ms$ for $k = 18$ and $k = 20$ respectively to complete a cycle. On the contrary, HOPE can complete within $1s$ for most scenarios, and Brute-force is much more slower. The results indicate that HOPE can be nearly 9 times slower as compared to Greedy but is 56 times faster than Brute-force. However, given the remarkable performance gain of HOPE over Greedy, we believe this trade-off in system running time is acceptable.

V. Conclusion

Network policies and service chains are important for the security and reliability of data centers. NFs of policies can be deployed in different environment, e.g., OpenFlow switches, hardware middleboxes and NFV servers. Such heterogeneous environment for policy allocation remain unexplored in previous works. In this paper, we study the Heterogeneous Policy Enforcement Problem with a focus on the latency. We proposed HOPE, which can find the optimal service chain path for each policy. Extensive simulation results have demonstrated the effectiveness and optimality of HOPE.

VI. Acknowledgements

This work is partially supported by Chinese National Research Fund (NSFC) Project No. 61402200; NSFC Key Project No. 61532013; National China 973 Project No. 2015CB352401; the UK Engineering and Physical Sciences Research Council (EPSRC) grants EP/P004407/1 and EP/P004024/1; Shanghai Scientific Innovation Act of STCSM No.15JC1402400 and 985 Project of SJTU with No. WF220103001.

References

[1] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan *et al.*, "Steering: A software-defined networking for inline service chaining," in *ICNP 2013*. IEEE, pp. 1–10.

[2] Surendra, M. Tufail, S. Majee, C. Captari, and S. Homma, "Service function chaining use cases in data centers," Internet Draft, IETF SFC WG, Tech. Rep. draft-ietf-sfc-dc-use-cases-04, January 2016.

[3] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang, "PGA: Using graphs to express and automatically reconcile network policies," in *ACM SIGCOMM*, 2015.

[4] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, Feb 2015.

[5] H. Mekky, F. Hao, S. Mukherjee, Z.-L. Zhang, and T. Lakshman, "Application-aware data plane processing in sdn," in *HotSDN*. ACM, 2014.

[6] L. Cui, R. Cziva, F. P. Tso, and D. P. Pezaros, "Synergistic policy and virtual machine consolidation in cloud data centers," IEEE INFOCOM 2016.

[7] T. Lukovszki, M. Rost, S. Schmid, and S. Schmid, "It's a Match! Near-Optimal and Incremental Middlebox Deployment," *ACM Sigcomm Computer Communication Review*, vol. 46, no. 1, pp. 30–36, 2016.

[8] K. Jang, J. Sherry, H. Ballani, and T. Moncaster, "Silo: predictable message latency in the cloud," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 435–448, 2015.

[9] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft, "Queues dont matter when you can jump them!" in *NSDI 2015*.

[10] Z. Liu, X. Wang, W. Pan, B. Yang, X. Hu, and J. Li, "Towards efficient load distribution in big data cloud," in *IEEE ICNC*, 2015, pp. 117–122.

[11] A. Gember-Jacobson, C. P. Raajay Viswanathan, R. Grandl, J. Khalid, S. Das, and A. Akella, "OpenNF: enabling innovation in network function control," in *Proc. of ACM SIGCOMM*, 2014, pp. 163–174.

[12] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible data center network," in *ACM SIGCOMM computer communication review*, vol. 39, no. 4. ACM, 2009, pp. 51–62.

[13] Z. Hu, Y. Qiao, and J. Luo, "Coarse-grained traffic matrix estimation for data center networks," *Computer Communications*, vol. 56, pp. 25–34, 2015.

[14] L. Cui, F. P. Tso, D. P. Pezaros, W. Jia, and W. Zho, "Policy-aware virtual machine management in data center networks," IEEE ICDCS 2015.

[15] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. ACM, 2015, pp. 139–152.

[16] P. Duan, Q. Li, Y. Jiang, and S.-T. Xia, "Toward latency-aware dynamic middlebox scheduling," in *Computer Communication and Networks (ICCCN), 24th International Conference on*. IEEE, 2015, pp. 1–8.