ParaBox: Exploiting Parallelism for Virtual Network Functions in Service Chaining

Yang Zhang*† Bilal Anwer* Vijay Gopalakrishnan* Bo Han*
Joshua Reich* Aman Shaikh* Zhi-Li Zhang†
*AT&T Labs – Research †University of Minnesota

ABSTRACT

Service Function Chains (SFCs) comprise a sequence of Network Functions (NFs) that are typically traversed in-order by data flows. Consequently, SFC delay grows linearly with the length of the SFC. Yet, for highly latency sensitive applications, this delay may be unacceptable—particularly when the constituent NFs are virtualized, running on commodity servers. In this paper, we investigate how SFC latency may be reduced by exploiting opportunities for parallel packet processing across NFs. We propose ParaBox, a novel hybrid packet processing architecture that, when possible, dynamically distributes packets to VNFs in parallel and merges their outputs intelligently to ensure the preservation of sequential processing semantics. To demonstrate our approach's feasibility, we implement a ParaBox prototype on top of the DPDK-enabled Berkeley Extensible Software Switch. Our preliminary experiment results show that ParaBox can not only significantly reduce the service chaining latency, but also improve throughput.

CCS CONCEPTS

•Networks →Middle boxes / network appliances; Traffic engineering algorithms;

KEYWORDS

Network Function Visualization; Service Function Chaining

1 INTRODUCTION

A Service Function Chain (SFC) defines a sequence of Network Functions (NFs), such as firewalls and load balancers (LBs), and stitches them together [14]. SFC has been a key enabler for network operators to offer diverse services and an important application of Software Defined Networking (SDN) [10, 28, 33]. Recently operators have begun to apply Network Functions Virtualization (NFV) [15] to SFC, using virtualized NFs running on commodity servers. While NFV ameliorates some of the challenges operators face in deploying SFC (*e.g.,* elastic service provisioning [30]), it exacerbates others. In particular, traffic traversing virtualized SFCs may suffer from reduced throughput and increased latency [13, 19, 24**?**]. Moreover, it is likely that the flexibility offered by the combination of SDN and NFV will result in SFC length increasing as networks become ever more highly automated—making this challenge ever more relevant.

In response, we present ParaBox, a novel packet processing architecture, that, when possible, mirrors packets to NFs in parallel and then intelligently merges together the output. To ensure correctness, the traffic emitted by our merge function must be identical to that which would have been emitted had the traffic traversed the NFs in the traditional sequential manner. As not all VNFs are parallelizable, ParaBox identifies opportunities for parallelism through its analysis function (see Section 2). In a nutshell, ParaBox is a hybrid architecture that leverages both sequential and parallel packet processing, as shown in Figure 1. In this example, the Intrusion Detection System (IDS) and Traffic Shaper are parallelizable, while the VPN gateway and Router are not. When using ParaBox, data packets first traverse the Virtual Private Network (VPN) gateway, then the IDS and Traffic Shaper in parallel, and finally the router.

While parallelism has been well studied in the wider space of computer networks [7, 12, 22, 23, 32], to the best of our knowledge, we are the first to explore parallel packet processing to reduce SFC latency. It is an important problem for network operators as the latency of a service chain with multiple VNFs may be unacceptable for latency-sensitive applications. In the following, we focus on applying ParaBox to service chains with VNFs on the same physical server. The scenario of service chains on a server has been studied in the literature [19] and it has use cases in real-world networking services [1]. We discuss how to extend to Physical Network Functions (PNFs) and VNFs across servers in Section 6.

SOSR'17, Santa Clara, CA.

DOI: http://dx.doi.org/10.1145/xxxxxxxxxxxxxxx

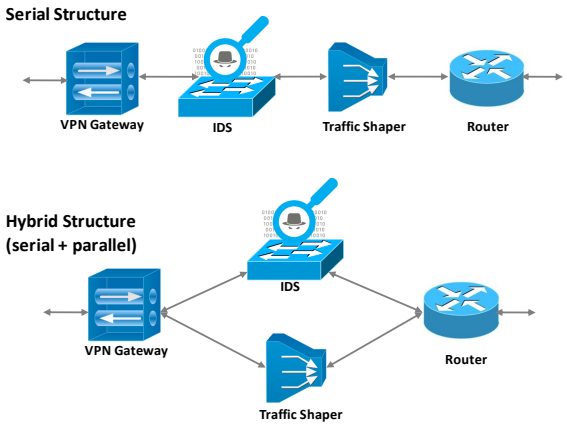


Figure 1: Hybrid service function chain.

Enabling parallel packet processing among VNFs is challenging due to following reasons. First, the mirror and merge functions should be lightweight, avoiding introducing too much latency. Otherwise, the benefit of parallel packet processing will be negated by this extra delay. Second, we need to determine what VNFs are parallelizable by carefully analyzing the VNF order dependency in SFC. Finally, to enable incremental deployment, ParaBox should not require any changes to VNFs.

We address the above challenges by making the following contributions in this paper.

- We investigate the feasibility of VNF-level parallel packet processing by carefully examining their order dependency in SFC (Section 2).
- We design ParaBox, a hybrid architecture that supports both sequential and parallel processing (Section 3).
- We implement a proof-of-concept of ParaBox by building its key components in a virtual switch (Section 4).
- We evaluate the performance of ParaBox and demonstrate that it can reduce the latency by up to 37.7% for a service chain consisting of two VNFs (Section 5).

2 PARALLELIZING NFS

In this section, we present an introduction to SFC and answer the key question of whether it is possible to parallelize VNFs.

2.1 Introduction to SFC

Network functions such as firewall, Network Address Translation (NAT), Intrusion Prevention System (IPS), WAN optimizer (WANX), *etc.* are generally deployed as inline services and end users are usually unaware of their existence. Such a set of NFs construct a service chain with use cases in various networks [13, 25**?**]. There are multiple ways of steering data flows through a service chain. The basic one is to physically wire an NF in a dedicated hardware middlebox and statically place them at manually induced intermediate points. It is hard to reconfigure such a predefined service chain, which is prone to errors and increases the management complexity of network operators. The advent of NFV and SDN has greatly facilitated traffic steering in SFC [10, 28, 33], by leveraging logically centralized control plane and providing the programmability of forwarding plane.

VNFs	HDR R/W	PL R/W	Add Bits	Examples
Probe	T/F	F/F	F	Flowmon
IDS	T/F	T/F	F	Snort/Bro
Firewall	T/F	F/F	F	iptables
NAT	T/T	F/F	F	iptables
L4 LB	T/F	F/F	F	iptables
WANX	T/T	T/T	T	WANProxy
Shaper	T/F	F/F	F	tc
Proxy	T/F	T/T	F	Squid

Table 1: NF operations on packet header (HDR) and payload (PL).

2.2 Order Dependency of NFs

At a high level, we can parallelize packet processing among NFs only if they are independent of each other in a service chain. Otherwise, we may break the correctness of network and service policies.

There are multiple factors that impact the NF order dependency for service chaining: 1) the read and write operations of NFs on data packets; 2) termination of flows (*e.g.,* dropped by a firewall) in an NF that affects the correctness/efficiency of the next NF; 3) packet reconstruction (*e.g.,* merged by a WAN optimizer); and 4) a load balancer before multiple instances of the same NF.

In Table 1, we examine the read and write operations on both packet header and payload (beyond the TCP header) for NFs commonly used in the literature. Some NFs, *e.g.,* WANX, may add extra bits to packets. This table shows operations performed by the listed NF examples on a per-packet basis (to the best of our knowledge). The read/write behavior of an NF can change from one implementation to another. Similarly configuration of individual NFs can impact their packet operations. This table represents an abstraction that can be used to perform order-dependency analysis of NFs for service chains.

The following relationships can be present between NFs based on their operations of packet data, Read after Read (RAR), Read after Write (RAW), Write after Read (WAR) and Write after Write (WAW) [17]. Two NFs that perform RAR and WAR operations can be safely parallelized. Two VNFs that perform WAW and RAW operations cannot be parallelized if the packet data that is being written/read in the second NF overlaps with what is written in the first one.

We use an example to illustrate the problems caused by flow termination and the other two cases (*i.e.,* packet reconstruction and multiple NF instances after an LB) are easy to

	Probe	IDS	Firewall	NAT	L4 LB	WANX	Shaper	Proxy
Probe		Y	Y	Y	Y	Y	Y	Y
IDS	Y		Y	Y	Y	Y	Y	Y
Firewall	N	N		Y	N	Y	Y	N
NAT	N	N	N		N	N	N	N
L4 LB	N	N	N	N		N	N	N
WANX	Y	X	X	X	X		Y	X
Shaper	N	Y	Y	Y	Y	Y		Y
Proxy	Y	Y	Y	Y	Y	N	Y	

Table 2: Pairwise NF order dependency. The NF in the leftmost column is the first one in a chain and the one in the top row is the second.

understand. When there is a firewall before a proxy or an IDS, parallelization will cause them to generate reports for flows that might be dropped by the firewall, which affects correctness. If there is an LB after a firewall, parallel processing will send dropped flows to the LB which impacts the efficiency of its load balancing algorithm. For other cases, such as a firewall before an NAT, parallelization may increase the resource utilization on the NAT. We can fall back to the sequential processing when the firewall drops a large number of flows.

Table 2 shows if various two-NF chains can be parallelized using ParaBox. The first NF of the chain is in the leftmost column and the second one is in the top row. The service chains that can be parallelized by ParaBox are marked with a "Y". Those that cannot be parallelized or do not have a deployment case are marked as "N"/"X" respectively. Note that, all the service chains that have a NAT as the first hop are not parallelizable. The reason is that the policy of next hop (*e.g.,* firewall rules) may be defined based on modified IP address, which brings in dependency between NAT and the next hop. In the ParaBox approach, data packets arriving at the next hop have unmodified IP address, which makes the regular rules not effective. However, if the policy configuration is ParaBox-aware (*e.g.,* defining the firewall rules on the original source IP address, instead of the one assigned by NAT), many of these chains can be parallelized. We assume that WANX is applied to outgoing network traffic and thus should not be deployed before IDS, firewall, NAT *etc.* Nonetheless, there are a number of NF pairs that are parallelizable.

3 DESIGN OF PARABOX

In this section, we describe the system architecture and key components of ParaBox.

3.1 Overview

As mentioned before, there are three key requirements for ParaBox. First, the additional components added to service chains should be lightweight without adding extra noticeable latency and require minimal knowledge of the VNFs for scalability. Second, we need a service orchestrator and controller

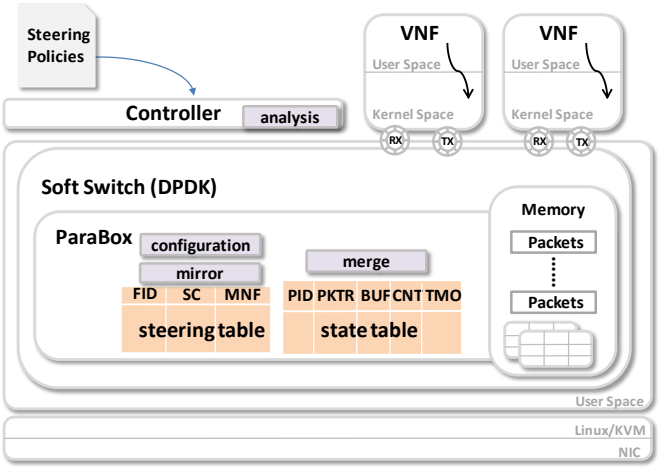


Figure 2: System architecture of ParaBox.

to analyze VNF order-dependency in a defined chain. Third, we should not require changes to network functions, in order to leverage existing VNFs from various vendors and deploy ParaBox incrementally. Modifying commercial off-the-shelf VNFs to adapt to new proposals is non-trivial.

To satisfy these requirements, ParaBox consists of three major components: the order-dependency analysis function in a controller, and the mirror and merge functions on a software switch. We show the architecture of ParaBox in Figure 2. The analysis function takes SFCs as input and examines whether data packets in certain service chain can be processed in parallel. Based on the output of the analysis function, the mirror function sends copies of data packets to parallelizable VNFs. The merging function then combines packets after they are processed in parallel by these NFs.

3.2 Dependency Analysis Function

The order-dependency analysis function of ParaBox controller is responsible for generating a SFC layout with parallel components. This layout is sent to the configuration module of ParaBox which uses it to determine how to mirror the traffic to VNF instances. To decide what parts of a service chain can be parallelized, the order-dependency analysis

function takes into account the principles based on NF models, as summarized in Section 2. It also considers the actions performed by NFs. For example, firewall would terminate a session, but it should not modify the packets. In contrast, NAT would rewrite a packet header, but not terminate a session. ParaBox controller sends selected information of VNFs to the configuration module that is required by the merge function, as to be shown next.

3.3 Mirror and Merge Functions

The mirror function is simple. Based on the service chain, if the next hop is a parallel component, it will create a copy of the packet and send it to each VNF in parallel. For the merge function, we model network packet as a $\{0|1\}^*$, *i.e.,* a sequence of bits. We first discuss the case where VNFs do not insert extra bits into packets. Assume P_O is the original packet, and there are two VNFs A and B in the chain with P_A and P_B as their outputs. The final merged packet $P_M = [(P_O \oplus P_A)|(P_O \oplus P_B)] \oplus P_O$. We *xor* every output packet of a VNF with the original one to get the modified bits and keep this result in an intermediate buffer. Since parallelizable VNFs do not modify the same field of a packet, we can get all modified bits from multiple VNFs by combining (*or*) the above *xor* results incrementally. For example, assume P_A arrives first. We will get its modified bits $P_O \oplus P_A$. After the merge function receives P_B, it will *or* B's modified bits $P_O \oplus P_B$ with A's. The operations are done when the merge function receives packets from all parallel VNFs, which triggers the *xor* of all modified bits with P_O. After that, checksum gets updated before packet is sent out. The advantage of this approach is that the merge function does not need to know in advance which field a VNF modifies. For VNFs that insert extra bits, the merge function needs to first remove these bits and add them back to the above P_M. Note that there will be a mirror and merge for every parallel component of a service chain.

Based on the above description of the mirror and merge functions, we summarize the structure of two tables that are required by them. The first one is a *traffic steering table* which describes service chains. There are three fields in this table: 1) flow ID; 2) service chain; and 3) description of VNFs if necessary. For example, we can write a hybrid service chain as A, {B, C}, D, {E, F, G}, H with two parallel components and three sequential NFs A, D, and H. We need the description for VNFs that add data to packets (*e.g.,* L7 LB and WANX). The second one is a *packet state table* which has five fields: 1) per-packet unique ID; 2) packet reference; 3) intermediate packet buffer; 4) VNF counter array; and 5) timeout. We use the packet ID as the key of each item in the table and for the mapping among packets in the merge function. Packet reference is a pointer to the memory address of the original packet, which we keep for the merge function. We use the packet buffer to hold the intermediate results of the merge function. The VNF counter array records the number of VNFs in each parallel component of a service chain. For instance, the array for the above example will be {1, 2, 1, 3, 1} (in which non-parallelizable NF is taken as a special case in parallelization). After a packet goes through a VNF, the corresponding counter will decrease by 1. When a counter reaches 0, it would trigger the final merge operation. We use timeout to handle packet drops.

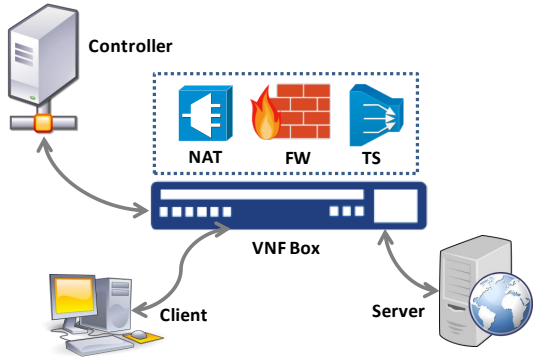


Figure 3: Experiment setup (TS stands for Traffic Shaper).

4 IMPLEMENTATION

We have implemented a prototype of ParaBox. The software switch used in the current implementation is BESS (Berkeley Extensible Software Switch) [2], a modular framework natively integrated with DPDK [4]. The reason we chose BESS is because of its flexibility and high performance. BESS leverages batch processing to improve efficiency and developers can define their own customized logic. Note that we can also use other software switches, such as Open vSwitch [5], mSwitch [18], Vector Packet Processing [3], *etc.*

We implement the mirror and merge functions as BESS modules. Our extensions to BESS have around 1100 lines of code (150 for the mirror function, 600 for the merge function, 200 for the configuration module, and 150 for others). The controller utilizes a customized protocol to communicate with ParaBox's configuration module. In the current implementation, we use the hash value of selected bytes of a packet as its ID. Following the recommendation by Henke *et al.* [16], we use the one-at-a-time hash function and choose high entropy bytes based on the protocols (*e.g.,* the IP ID field, TCP sequence and acknowledgment numbers, *etc.*).

5 PERFORMANCE EVALUATION

In this section, we present our preliminary experimental results to demonstrate the feasibility and efficiency of ParaBox.

The experimental setup is shown in Figure 3. We use three NFs: a Network Address Translator (NAT), a firewall (FW), and a Traffic Shaper (TS). We use Linux kernel **`iptables`** for the NAT and FW. For the Traffic Shaper we use the Linux **`tc`**

	Uplink			Downlink		
	Mean	MED	Stdev	Mean	MED	Stdev
S-C1	**32**	20	33	**74**	25	71
P-C1	**21**	17	13	**45**	21	62
S-C2	**117**	118	58	**292**	316	80
P-C2	**96**	71	85	**252**	264	68
S-C3	**141**	157	47	**319**	319	82
P-C3	**107**	87	68	**284**	305	88

Table 3: Comparison of the latency in microseconds for ParaBox **and serial service chaining. S stands for serial processing and P for** ParaBox**. C1 is NAT → FW, C2 is NAT → Traffic Shaper and C3 is NAT → FW → Traffic Shaper.**

utility. We run each in its own docker container with dedicated CPU. Each container is bound to a BESS *VPort* that uses an efficient ring-based driver to exchange data buffers. The machine running ParaBox is equipped with Intel(R) Xeon(R) CPU E5-2620 (2.00GHz, 12 cores, hyperthreading off) and 32GB memory. There are two other machines. The second which acts as a client, downloads a large file from a server running on the third machine. These three machines are directly connected via Ethernet cables.

We use two metrics to evaluate the performance of ParaBox: (a) service chain latency, and (b) end-to-end throughput. We evaluate the performance for three service chains: NAT → FW, NAT → TS, and NAT → FW → TS. We assume FW rules are based on host original ip, which decouples the dependency between NAT and FW. We calculate the latency for both uplink and downlink. Note that this is the latency of the service chain with all VNFs on the same server.

We compare ParaBox with sequential chaining and present the CDFs of the per-packet downlink latency for the three chains in Figure 4. For the first chain, the latency of 80% downlink packets in ParaBox is within 30 microseconds, while only 50% for the serial service chain. Similarly for the second and third chains we can see marked improvement in latency for 90% of the packets when compared with serialized service chaining. As we can see from these figures, ParaBox can reduce the latency for all three service chains.

We also report the mean, median and standard deviation (stdev) of all data packets in Table 3. This table shows ParaBox can reduce the latency in both directions. Note that the downlink takes more time because of the larger packet size (data packets vs. ACKs). For the service chain of NAT → FW, ParaBox can reduce the service chain latency by up to 37.7% and increase the downloading throughput by up to 10.8% (729 vs. 658 Mbps). For the other two chains, the latency reduction is around 14.9% as the Traffic Shaper throttles traffic and acts as a bottleneck.

6 DISCUSSION

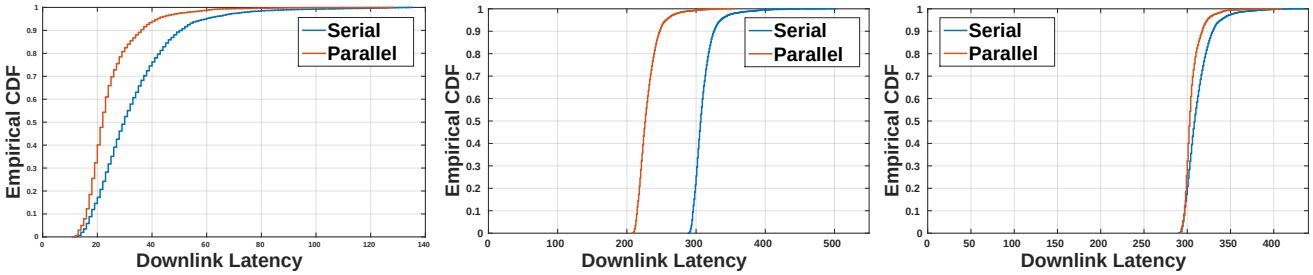
In this section, we discuss how to extend ParaBox and highlight several open issues as future directions.

Dependency Analysis. To analyze the VNF order dependency, we need to have a formal model to describe diverse functionality and deployment configurations of network functions [20]. For real-world use cases we also need to consider different implementations of the same VNF type within its context of deployment to perform correct order-dependency analysis. The current VNF configurations are generated by network administrators or SDN controllers with an assumption that VNFs are not parallelized. This results in many chains in Table 2 not parallelizable (*e.g.,* all chains with NAT at the first hop). We can build a plugin for controllers that is ParaBox aware and is able to transform the configurations of other VNFs accordingly when parallelized with a NAT. This controller plugin will ultimately result in more VNF pairs that can be parallelized in Table 2.

Merge Function. We aim to minimize the VNF specific information conveyed to the merge function. We can further improve the performance of ParaBox by exposing more information to it. For example, if the merge function knows that a flow is dropped by a firewall, it does not need to use the timeout for packets from other VNFs. For the mapping of packets in the merge function, the ID of a packet is the hash value of its selected bytes. Other options include the recently proposed Network Service Header [29] which defines per-packet service metafuturedata. ParaBox can generate global unique per-packet ID (within a packet processing platform) and keep it in the metadata. We can also borrow the existing fields in packet headers, *e.g.,* VLAN Identifier and IPv6 Flow Label, if they are not used for other purposes.

Across Multiple Physical Servers. In this paper, we focus on the scenario where multiple VNFs of a service chain are running on the same physical server. The VNF-level parallel data processing is also applicable for VNFs across multiple servers and PNFs. We can potentially leverage the port mirroring of hardware switches for ParaBox's mirror function. We can implement the merge operation as a network function and chain it together with the parallel VNFs/PNFs as their immediate next hop. Another option is to select the server running one of the VNFs as the merge point and steer the outputs of other VNFs to it.

Layer 3 VNFs and Mobility VNFs. We have considered layer 4+ VNFs for parallel data processing. There are also layer 3 VNFs, such as virtual routers, and mobility VNFs, including virtual Packet Data Network Gateway, Serving Gateway, Mobility Management Entity, *etc.* In general, we cannot parallelize routers and gateways with other VNFs, as the routers determine the traffic path and the gateways create/handle various tunnels. However, benefiting from the hybrid nature of ParaBox, we can still chain them together with parallelizable VNFs, as shown in Figure 1. We will investigate mobility VNFs, *e.g.,* various Call Session Control Functions in the IP Multimedia Subsystem in future work.



(a) NAT → FW (b) NAT → Traffic Shaper (c) NAT → FW → Traffic Shaper

Figure 4: Empirical CDF of downlink latency in microseconds for different VNF chains.

Performance Evaluation. The performance evaluation of ParaBox is preliminary. We examined the performance of ParaBox for short service chains with two VNFs. We did not evaluate the scenarios where multiple service chains sharing common VNFs run at the same time. We used open source software running in containers for the evaluation, having not conducted experiments on production VNFs. We plan to address these limitations in our future work.

7 RELATED WORK

Parallelism in Computer Networks. Parallel processing is a well-established technology in various areas of computer networks. P4 [7] is a domain-specific protocol-independent language for programming packet forwarding dataplanes, which supports parallel table lookups. ClickNP [22] utilizes parallelism in FPGA to optimize packet processing. MapReduce [9] is a programming model (plus an associated implementation) for large data set processing, which parallelizes jobs on a cluster of servers. Graphene [12] is a cluster scheduler for improving job completion time in a directed acyclic graph with many parallel chains. WebProphet [23] and WProf [32] expedite web page load time by parallelizing object downloading based on dependency graph analysis. In contrast, the parallelism target of ParaBox is network functions in service chains, which has not been exploited.

Service Function Chaining. Service function chaining has been extensively studied in the literature. The Delegation-Oriented Architecture [31] facilitates incremental middlebox deployment and chaining by leveraging global identifiers in packets and endpoint-based delegation. PLayer [21] realizes a policy-aware switching layer to explicitly steer traffic through middleboxes. SIMPLE [28] is a policy enforcement platform for efficient middlebox specific traffic steering. STEERING [33] provides a flexible way to route traffic for inline services based on SDN schema. FlowTags [10] offers flow tracking capability by using tags associated with necessary middlebox context to ensure consistent policy enforcement. PGA [27] proposes a network policy expression and leverages graph structure to resolve service chaining policy conflicts. Slick [6] implements an integrated platform which handles both middlebox placement and traffic steering for efficient use of network resources. The above traffic steering and policy enforcement applied in SFC are orthogonal to ParaBox whose goal is to improve chaining latency.

Network Functions Virtualization. There is a plethora of work on improving the performance and manageability of VNFs. FreeFlow [30] provides a systems-level and state-centric abstraction, called Split/Merge, for elastic execution of VNFs. NetVM [19] allows zero-copy packet delivery across a chain of VMs within a trust boundary on the same physical server. ClickOS [24] proposes a high-performance virtualized software platform to reduce VM instantiation time. OpenNF [11] is a framework that offers coordinated control of network function state. E2 [26] proposes an NFV framework which makes developers focus on core application logics by automating and consolidating common management tasks. OpenBox [8] decouples control plane from data plane of VNFs and allows reuse of software modules across network functions. Differently from the above work, ParaBox leverages VNF-level parallel packet processing to reduce service chain latency.

8 CONCLUSIONS

To the best of our knowledge, we have proposed the first hybrid service function chaining architecture, called ParaBox, that processes data packets among VNFs in parallel when possible. ParaBox has three key components, dependency analysis function that determines whether VNFs are parallelizable, mirror function that distributes copies of packets to multiple VNFs and merge function that combines their outputs. To demonstrate its feasibility and effectiveness, we have implemented a prototype of ParaBox mainly using the BESS virtual switch. Our initial experiment results show that ParaBox can reduce latency by up to 37.7% and increase throughput by up to 10.8% when parallelizing data processing between two VNFs (*e.g.*, a NAT and a firewall). The work that has been illustrated in this paper so far is only a starting point of VNF-level parallel data processing for service function chaining. It serves as our initial attempt towards a full-fledged realization of the ParaBox architecture.

9 ACKNOWLEDGMENTS

We thank all SOSR reviewers for their valuable comments. This research was supported in part by NSF grants CNS-1411636, CNS 1618339 and CNS 1617729, DTRA grant HDTRA1-14-1-0040 and DoD ARO MURI Award W911NF-12-1-0385.

REFERENCES

[1] AT&T Universal Customer Premises Equipment (uCPE). https://www.business.att.com/content/productbrochures/universal-customer-premises-equipment-brief.pdf, 2016.
[2] Berkeley Extensible Software Switch. http://span.cs.berkeley.edu/bess.html, 2016.
[3] Cisco's Vector Packet Processing. https://wiki.fd.io/view/VPP, 2016.
[4] DPDK. http://dpdk.org/, 2016.
[5] DPDK OVS. https://clearlinux.org/documentation/ac-ovs-dpdk.html, 2016.
[6] B. Anwer, T. Benson, N. Feamster, and D. Levin. Programming Slick Network Functions. In *Proc. SOSR*, 2015.
[7] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming Protocol-Independent Packet Processors. In *SIGCOMM CCR*, 2014.
[8] A. Bremler-Barr, Y. Harchol, and D. Hay. OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions. In *Proc. SIGCOMM*, 2016.
[9] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. OSDI*, 2004.
[10] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags. In *Proc. NSDI*, 2014.
[11] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. OpenNF: Enabling Innovation in Network Function Control. In *Proc. SIGCOMM*, 2014.
[12] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni. GRAPHENE: Packing and Dependency-Aware Scheduling for Data-Parallel Clusters. In *Proc. OSDI*, 2016.
[13] W. Haeffner, J. Napper, M. Stiemerling, D. R. Lopez, and J. Uttaro. Service Function Chaining Use Cases in Mobile Networks. Internet-Draft draft-haeffner-sfc-use-case-mobility-02, IETF, 2014.
[14] J. M. Halpern and C. Pignataro. Service Function Chaining (SFC) Architecture. RFC 7665, 2015.
[15] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. Network function virtualization: Challenges and opportunities for innovations. In *IEEE Communications Magazine*, 2015.
[16] C. Henke, C. Schmoll, and T. Zseby. Empirical Evaluation of Hash Functions for Multipoint Measurements. In *SIGCOMM CCR*, 2008.
[17] J. L. Hennessy and D. A. Patterson. Computer architecture: a quantitative approach. Elsevier, 2011.
[18] M. Honda, F. Huici, G. Lettieri, and L. Rizzo. mSwitch: A Highly-Scalable, Modular Software Switch. In *Proc. SOSR*, 2015.
[19] J. Hwang, K. K. Ramakrishnan, and T. Wood. NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms. In *Proc. NSDI*, 2014.
[20] D. Joseph and I. Stoica. Modeling middleboxes. IEEE Network: The Magazine of Global Internetworking, 2008.
[21] D. A. Joseph, A. Tavakoli, and I. Stoica. A Policy-aware Switching Layer for Data Centers. In *Proc. SIGCOMM*, 2008.
[22] B. Li, K. Tan, L. L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen. ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware. In *Proc. SIGCOMM*, 2016.
[23] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. Greenberg, and Y. Wang. WebProphet: Automating Performance Prediction for Web Services. In *Proc. NSDI*, 2010.
[24] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici. ClickOS and the Art of Network Function Virtualization. In *Proc. NSDI 14*, 2014.
[25] T. Nadeau and P. Quinn. Problem Statement for Service Function Chaining. RFC 7498, 2015.
[26] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker. E2: A Framework for NFV Applications. In *Proc. SOSP*, 2015.
[27] 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 *Proc. SIGCOMM*, 2015.
[28] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. SIMPLE-fying Middlebox Policy Enforcement Using SDN. In *Proc. SIGCOMM*, 2013.
[29] P. Quinn and U. Elzur. Network Service Header. Internet-Draft draft-ietf-sfc-nsh-10, IETF, 2016.
[30] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield. Split/Merge: System Support for Elastic Execution in Virtual Middleboxes. In *Proc. NSDI*, 2013.
[31] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes No Longer Considered Harmful. In *Proc. OSDI*, 2004.
[32] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. Demystifying Page Load Performance with WProf. In *Proc. NSDI*, 2013.
[33] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvret, R. Manghirmalani, R. Mishra, R. Patney, M. Shirazipour, R. Subrahmaniam, C. Truchan, and M. Tatipamula. StEERING: A software-defined networking for inline service chaining. In *Proc. ICNP*, 2013.