GEN: A GPU-Accelerated Elastic Framework for NFV

Zhilong Zheng, Jun Bi, Chen Sun, Heng Yu, Hongxin Hu, Zili Meng, Shuhe Wang, Kai Gao, Jianping Wu*

ABSTRACT

Network Function Virtualization (NFV) has the potential to enhance service delivery flexibility and reduce overall costs by provisioning software-based service function chains (SFCs) on commodity hardware. However, we observe that existing CPU-based SFC solutions cannot achieve both high performance and high elasticity simultaneously. To address such a critical challenge, we seek beyond CPU and exploit the capability of Graphics Processing Unit (GPU) to support NFV. We propose `GEN`, a GPU-based *high performance* and *elastic* framework for NFV. As opposed to pipeline-based SFCs in existing GPU-based NFV systems, `GEN` proposes to support RTC-based SFCs to improve processing performance. Meanwhile, `GEN` offers great elasticity of network function (NF) *scaling up* and *down* by allocating a different number of fine-grained GPU threads to an NF during runtime. We have implemented a prototype of `GEN`. Preliminary evaluation results demonstrate that `GEN` improves performance with RTC-based SFCs, and supports adaptive, precise, and fast NF scaling for NFV.

CCS CONCEPTS

• **Networks** → *Middle boxes / network appliances*; *Programmable networks*;

KEYWORDS

NFV; Service chain; GPU; P1erformance

*Zhilong Zheng, Jun Bi, Chen Sun, Heng Yu, Zili Meng, Shuhe Wang, Kai Gao, and Jianping Wu are with Institute for Network Sciences and Cyberspace, Tsinghua University and Beijing National Research Center for Information Science and Technology (BNRist). Hongxin Hu is with the School of Computing, Clemson University. Jun Bi (junbi@tsinghua.edu.cn) is the corresponding author.

APNet '18, August 2–3, 2018, Beijing, China
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6395-2/18/08.
https://doi.org/10.1145/3232565.3234510

ACM Reference Format:
Zhilong Zheng, Jun Bi, Chen Sun, Heng Yu, Hongxin Hu, Zili Meng, Shuhe Wang, Kai Gao, Jianping Wu. 2018. GEN: A GPU-Accelerated Elastic Framework for NFV. In *APNet '18: Asia-Pacific Workshop on Networking, August 2–3, 2018, Beijing, China* , 8 pages. https://doi.org/10.1145/3232565.3234510

1 INTRODUCTION

Network Function Virtualization (NFV) [7] was recently introduced to address the limitations of traditional proprietary middleboxes [26]. By leveraging virtualization techniques, NFV could achieve the elasticity of NF scaling during runtime to adapt to network traffic dynamics. In operator networks [23], data centers [14, 18], mobile networks [9], and enterprise networks [25], operators often require traffic to pass through a sequence of NFs (e.g. firewall+IDS+proxy) [2, 10, 22], which is commonly referred to as a service function chain (SFC) [10, 23].

Supporting SFCs in NFV should achieve two major goals including *high performance* and *high elasticity*. However, Current CPU-based SFCs fail to simultaneously achieve the above two goals in two aspects.

Low performance with negative improvement expectation: Current CPU-based NFV systems suffer from a much lower performance in both latency and throughput comparing to traditional dedicated hardware [28]. This limitation prohibits the effective support for a wide range of applications with tight performance requirements such as algorithmic stock trading and high performance distributed memory caches [3]. Furthermore, the end of Moore's Law reveals a pessimistic expectation of the improvement of CPU computation power, which weakens the capability of CPU to satisfy the performance requirement when handling growing volumes of traffic. Although scaling out to more servers can improve throughput, we still have to cope with the problem of additional cost of *NF migration* [5, 24].

Coarse-grained scaling: To achieve high elasticity, an NF instance has to be scaled *out* when it is overloaded by the increasing network traffic, and scaled *in* when traffic rate drops. However, NF scaling out/in in CPU exposes two major shortcomings. First, the *resource utilization efficiency* is compromised. Suppose we have to scale out an NF instance if its load exceeds 80% of its capacity. For an NF with 85% load, we have to introduce another whole CPU core to carry only

Table 1: Test system hardware specification (total $3,333)

Item	Specification	Qty	Unit price
CPU	Intel Xeon E5-2650 v4 (12 cores, 2.2GHz)	1	$1199
RAM	Samsung DDR4 16G (2440MHz)	2	$302
NIC	Intel X520 (Dual 10Gbps ports)	2	$165
GPU	NVIDIA TITAN Xp	1	$1200

5% of the load, with the rest 95% CPU core power wasted. Second, to scale out an NF, states have to be migrated and flows have to be redistributed among NF instances, which could incur *milliseconds* of latency overhead [5, 24]. The intrinsic shortcomings of NF scaling out/in limit CPU from achieving high elasticity.

To address the limitations of CPU-based SFCs in NFV, we seek beyond CPU and argue that another type of commodity computation resource, Graphics Processing Unit (GPU), has the potential to achieve both *high performance* and *elasticity* when supporting NFV. While recent works have proposed to use GPU to support NFV [32, 33], we focus on discussing the potential of GPU to achieve fine-grained scaling and the interconnection of SFC to achieve higher performance.

Concerning performance, GPU offers extreme thread-level parallelism with thousands of GPU cores. And many prior works have shown that GPU can provide higher performance than CPU [11, 30, 33]. Moreover, such a significant performance gain comes with a small cost (i.e., cost-effective [11]). Table 1 shows that GPU is even cheaper than CPUs and only occupy a small portion of the total hardware budget. With respect to elasticity, GPU could allocate a different number of fine-grained computation resources, i.e. GPU cores/threads, to different tasks during runtime [19]. Therefore, GPU could *scale up/down* NF instances to adapt to dynamic network traffic volumes. In this way, GPU resources could be precisely allocated according to the incoming traffic, which leads to near 100% resource utilization efficiency. At the same time, states and flows no longer need to be migrated for NF scaling. This could fully avoid the overheads caused by NF *scaling out/in*, showing the potential of GPU to offer great elasticity for NFV.

In this paper, we propose GEN, a GPU-accelerated Elastic framework for NFV. As shown in Figure 1, GEN consists of two major components. The *orchestrator* is responsible for SFC construction and modification. The *infrastructure* supports SFCs with GPU. Each server runs an *SFC Manager* to communicate with the orchestrator and manages local SFCs. Each SFC is equipped with an in-CPU *SFC Controller* that steers packets among NFs and forwards packets after processing. The SFC Controller can scale up/down NFs to adapt to network traffic dynamics.

- We present the background and related research efforts of running NFs inside GPU, and identify design challenges to support SFCs in GPU for NFV. (§ 2)

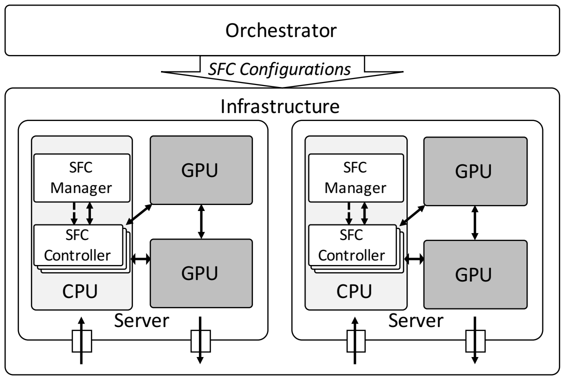


Figure 1: GEN framework overview

- We propose GEN, a GPU-based high performance and elastic framework for NFV. We articulate how GEN achieves both high performance by supporting *run-to-completion (RTC) model* in GPU and high elasticity by NF scaling up/down. We further enable GEN to support some advanced NFV features, such as runtime SFC modification. (§ 3)
- We have implemented a prototype of GEN. Preliminary evaluation results (§ 4) demonstrate the high performance of RTC-based SFC and high elasticity offered by GEN.

2 BACKGROUND AND CHALLENGES

We first introduce background on the GPU architecture and workflow, and recent research efforts on exploring GPU for packet processing. We then identify the design challenges of using GPU to support a *chain* of NFs in NFV.

2.1 Background and Related Work

A modern programmable GPU acts as a co-processor that receives the code (called "kernels") and data from the host CPU. GPU has on-board device memory, so data must be *copied* in from the server's main memory ("host memory") over the PCIe. Then GPU processes the data and informs CPU to copy processed data back to host memory. The PCIe is also used for CPU-GPU communication such as launching GPU kernels. Each time a CPU launches a GPU kernel, CPU assigns a different number of threads to this kernel, which can be regarded as *dynamic scaling up/down* of the kernel. GPU uses a SIMT (Single Instruction, Multiple Thread) execution model, in which a group of cores share a single program counter and process a *batch* of data. More details about GPU and the Compute Unified Device Architecture (CUDA) environment we adopted can be found in [19].

Some research efforts [6, 11, 17, 29, 30] have proposed to accelerate packet processing tasks with GPU. PacketShader [11] proposed a high-performance software router framework for tasks such as IPv4/IPv6 packet forwarding with GPU. Snap [29] integrated and extended the Click Modular Router to flexibly construct one static packet processing pipeline in GPU.

GASPP [30] proposed a processing framework that supported stateful network functions such as flow tracking and TCP reassembly in GPU. NBA [17] performed optimized load balancing between heterogeneous processors including CPU and GPU for tasks such as forwarding and IDS. Above research efforts mainly focused on the performance and programmability of supporting *one NF* or *a static NF pipeline* using GPU. GPUNFV [32] also proposed to support stateful SFC using GPU. However, its per-flow in one thread processing model can suffer from heavy performance overhead. G-NET[33] uses an SFC-based scheduling schema to improve system utilization. However, we are targeting at a different goal of exploiting the benefits of GPU to build a *high performance and elastic* framework for NFV. We next present the design challenges of the GEN framework.

2.2 Challenges

GEN could employ advanced orchestrators such as Stratos [4], E2 [20], etc. In this paper, we focus on the design of the infrastructure to support NFV with GPU. We encounter three major challenges in the design of the GEN infrastructure.

SFC modeling in GPU: We need to select an appropriate model (pipelining or RTC) for SFCs in GPU. Although GPU can support both, we reveal the intrinsic limitations of the pipelining model with respect to unique features of GPU. Thus, we select RTC to support SFCs in GPU for both high performance and high elasticity. We introduce this in § 3.2.

Elastic NF scaling: NF scaling up / down in GPU is challenged to decide *when* to scale, calculate *to what extent* the NF should be scaled, and *quickly* realize scaling. For an NF in a CPU, we could easily decide if it is overloaded based on its utilization. However, GPU allocates multiple cores to an NF, which makes it difficult to monitor the load. Furthermore, precisely allocating resources to NFs to process varying traffic volumes is also a critical challenge. Finally, NF scaling should be quickly carried out to timely handle network dynamics. In response, GEN supports adaptive, precise, and quick scaling of NFs through dynamic adjustment of the batch size according to the SFC buffer occupancy. We introduce this in § 3.3.

Runtime SFC Modification: Operators of NFV networks may need to dynamically add/modify an SFC, or add a new NF to the SFC *during runtime* [5, 23]. However, NFs are written as kernels and compiled into binary codes to run in GPU, which makes it difficult to modify SFCs on the fly. We address this challenge in § 3.4.

3 GEN DESIGN

We present the GEN infrastructure design in Figure 2. We first provide an overview of each module in the GEN infrastructure.

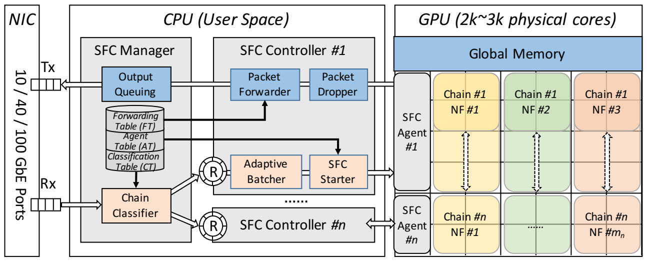


Figure 2: GEN infrastructure design

Then, we focus on the design choices of GEN to achieve both high performance and high elasticity for NFV.

3.1 Overview

GEN deploys multiple NFs that can form *multiple SFCs inside one GPU* to fully exploit the abundant resource of GPU, which typically comprises 2k to 3k physical cores. For SFC management, GEN designs two major modules in CPU. We introduce an *SFC Manager* for each server to communicate with the Orchestrator and manage SFCs in local GPU. Furthermore, we prepare an individual *SFC Controller* to control each SFC inside the GPU.

Specifically, once packets enter a server, they need to be grouped based on the SFC they belong to in order to match the SIMT execution model of GPU. Therefore, we design a *Chain Classifier* in the SFC Manager to classify packets by consulting a *Classification Table (CT)* (Figure 3(a)). Then, each *SFC Controller* receives classified packets and stores them in its own *Ring Buffer* [12]. Inside the controller, the *Adaptive Batcher* prepares packet batches for the SFC. The *SFC Starter* launches the SFC according to the *Agent Table (AT)* (Figure 3(b)) and copies the batch into GPU. The *Packet Dropper* would filter processed packets from GPU, and the *Packet Forwarder* will send them out according to the *Forwarding Table (FT)* (Figure 3(c)). The SFC Manager could receive SFC configuration messages from the Orchestrator and maintain local tables accordingly.

Furthermore, GPU communicates with the host server through PCIe buses. Since one physical server is equipped with multiple PCIe buses, we could install more than one GPU to maximize the processing capacity of a single server with incremental costs. The SFC Manager should then be able to classify and deliver packets to SFCs on different GPU inside one server. In response, we enable each SFC Controller to start its SFC according to the GPU ID in AT, while exposing the same Ring Buffer to the SFC Manager.

In the rest of this section, we first introduce the selection of the SFC model inside a GPU in § 3.2. Then we present how GEN could offer elasticity of NF scaling in § 3.3. Finally, we

Match	SFC_ID
Src_IP = 192.168.*	1

(a) Classification Table (CT)

SFC ID	GPU ID	Agent	NFs
1	*#1*	*Pointer 1*	[1, 2, 3]
2	*#1*	*Pointer 2*	[3, 4]
3	*#2*	*Pointer 3*	[5, 6, 7]

(b) Agent Table (AT)

SFC ID	Target
1	output (*port #1*)
2	output (*port #2*)
3	output (*port #1*)

(c) Forwarding Table (FT)

Figure 3: GEN table structures

elaborate how GEN supports advanced NFV features such as runtime SFC modification in § 3.4.

3.2 SFC Modeling in GPU

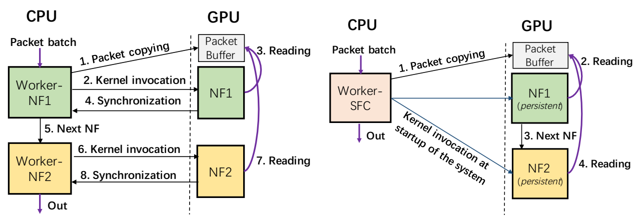
The pipelining model in CPU allocates isolated CPU cores for different NFs in an SFC. All NFs in the SFC have to run simultaneously to form a packet processing pipeline. In the context of GPU, NFs are programmed as GPU kernels. Recent researches have proposed two ways to support pipeline-based SFC in GPUs. However, we observe that the two approaches suffer from either low performance or costly elasticity.

Low SFC performance: Figure 4(a) shows the first way to support pipeline-based SFC in GPU [8, 33]. It copies packet batches from CPU to GPU and sequentially invokes the NF kernel functions in an SFC to process the packets. However, as introduced in § 2.1, there is performance overhead every time CPU launches a GPU kernel ($\sim 5\mu s$ according to our testing). This could be even worse for NFV since an SFC could be composed of seven or more NFs, which might largely compromise the performance acceleration brought by GPU.

Costly SFC elasticity: Figure 4(b) shows the second way [8] to support pipeline-based SFC in GPU. It benefits from the latest techniques of GPU including *Multi-Process Service (MPS)* and *persistent threads*, which support running multiple concurrent kernels simultaneously and constantly in an endless loop. In this way, the need for frequently invoking GPU kernels is eliminated and therefore could protect the high performance brought by GPU. Unfortunately, this approach cannot exploit the capability of GPU to dynamically scale up/down a kernel function by allocating more or fewer threads to the kernel each time the CPU launches the NF kernel. As threads inside the NFs are persistent in this approach, in order to provision more resources for a specific NF, CPU has to launch another kernel of this NF and balance the load between the two instances, which falls back to the scaling out solution and suffers from costly state migration and NF management issues. Therefore, this approach may suffer from costly elasticity.

3.2.1 Supporting RTC Model in GPU.

For the RTC model in CPU, all NFs in an SFC share the same amount of resource, i.e. the CPU core. At any time, only one NF exclusively occupies the entire core to process the packets. Similarly, in GPU, we support the RTC model by allocating the same number of threads to every NF in an SFC. Suppose



(a) Sequenced invocations [8, 33] (b) Persistent kernels [8]

Figure 4: Two ways to support pipelining SFC in GPU

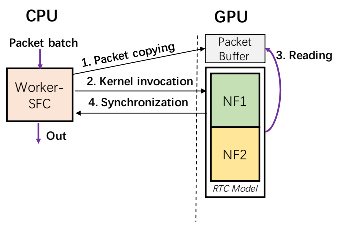


Figure 5: RTC-based SFC in GEN

the size of the packet batch is 32, in which case 32 threads are allocated to the SFC. NFs in the SFC *in turn fully occupy* the 32 GPU threads to process the packet batch, which leads to almost 100% utilization efficiency of the GPU threads allocated to the RTC-based SFCs. In this way, RTC model can fully utilize GPU resources

Regarding elasticity, we could dynamically adjust the batch size according to traffic volumes and allocate a different number of threads to the SFC to achieve the elasticity of scaling up/down, which achieves a cheaper elasticity comparing to Figure 4(b). Meanwhile, as Figure 5 shows, we only need to invoke the kernel once for an SFC no matter how long it is, which could improve the performance comparing to the pipelining model shown in Figure 4(a). Therefore, we select the RTC model to support SFCs in GPU for both high performance and elasticity.

The core idea of RTC model is consolidating several operations in multiple threads into one thread (i.e., *all-in-one* [13]). Researches including NetBricks [21] and Metron [16] consolidated all NFs into one core to boost the performance of SFC in CPU-based NFV. Kargus [13] adopted the RTC model to consolidate in-CPU functional modules including networking I/O operation (i.e., packet acquisition) and preprocessing (e.g., copying packets to GPU) to implement a high-performance NIDS in GPU. In brief, above efforts all proved that the RTC model could achieve higher performance than the pipelining model. In comparison, besides the performance benefits, GEN also reveals the potential of the in-GPU RTC model to achieve elasticity by dynamically scaling up/down, which is also a significant feature of NFV.

3.2.2 Efficient NF Chaining for RTC-based SFCs.

We learn that a GPU kernel with a __*global*__ declaration can launch another function with a *__device__* declaration with negligible overhead [31]. Therefore, as shown in Figure 1, GEN introduces a *__global__ SFC Agent* for each SFC that runs inside GPU as an agent of the CPU. *Every time* a new packet batch is ready for processing, the *SFC Starter* in CPU looks up the Agent Table in Figure 3(b), launches the corresponding SFC Agent, and passes all the NFs in the SFC as a $List < NF >$ parameter to the agent. The SFC Agent would then start *__device__* NFs in turn on behalf of the CPU. In this way, GEN requires the CPU to start *only one* GPU kernel (the SFC Agent) for an SFC of any length to process a batch.

It should be noted that there is no requirement for packet delivery across NFs when adopting an RTC-based model for SFCs. As shown in Figure 5, when compiling into an SFC Agent, NFs are treated as sequenced parts (similar to the term *element* in NetBricks [21]) of this program and share the same packet buffer. As a result, when an NF completes code execution, subsequent NFs use the same program context and pointers to packet buffer to execute their code. Hence, we can see that passing packets explicitly across an SFC is not required.

3.3 Elastic NF Scaling

Similar to NF scaling out/in inside CPU, a *strawman solution* for scaling up/down an NF in GPU is to keep monitoring the load of the NF, and adjust the batch size and GPU threads allocated to the NF once the NF is detected to be overloaded. However, as mentioned in § 2, scaling an NF in GPU is *challenged* to figure out when to scale, to what extent the NF should be scaled, and how to quickly carry out NF scaling. As GPU allocates multiple (possibly 1,000 or more) GPU threads to an NF, it becomes difficult to inspect the NF load by monitoring the utilization efficiency of GPU cores. Even if we could obtain the NF load, we cannot easily decide how many GPU threads should be provisioned to precisely handle the current traffic volume. Finally, NF load measurement and resource provisioning calculation may prevent the scaling process from being carried out quickly.

To address the above challenges, we consider the *goal of NF scaling* from another perspective. Due to the adoption of the RTC model, scaling an NF in GEN is equal to scaling the entire SFC. If the processing capability of an SFC is below the packet rate, incoming packets will pile up in the buffer of the SFC, which could incur large performance overhead or even packet dropping. Therefore, the key point of SFC scaling is *keeping the buffer occupancy at a low level.*

Therefore, we design an *Adaptive Batcher* module to adjust the packet batch size to adapt to network traffic dynamics. After sending a packet batch into the SFC Starter, the Adaptive Batcher will keep waiting until the batch has been processed. Then the Adaptive Batcher fetches the *maximal number of packets from the Ring Buffer* into the new batch, and divide the batch into multiple mini-batches (a static size of 256 packets in our current implementation) and assign them to multiple SMs (Streaming Multiprocessors) to reduce the latency induced by a large batch. With the adaptive batcher, GEN no longer needs to monitor the NF load. Instead, GEN simply keeps the Ring Buffer of the SFC at a low occupancy by consuming the buffer as many packets as possible to handle more loads. Now, every thread allocated to the new packet batch could be utilized to process a packet, resulting in zero resource waste. Furthermore, the batch size could be adjusted every time a new batch is prepared, indicating that NF scaling could be carried out very quickly.

3.4 Runtime SFC Modification

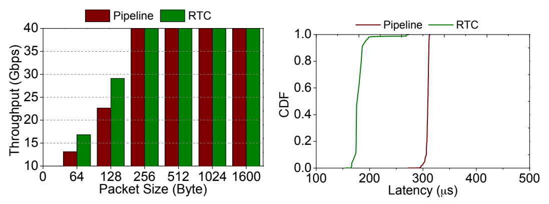
To achieve flexible service provisioning, NFV operators might need to dynamically add an SFC, reorder NFs in an SFC, or inject new NFs into an SFC. We next introduce how GEN could support above runtime SFC modifications.

SFC addition: To add an SFC for a new group of packets, the SFC Manager would insert an entry in the Classification Table to record the SFC ID and matching patterns of packets. Furthermore, it records the GPU ID where the SFC is deployed and NFs in the SFC in the Agent Table. Finally, the SFC Manager adds an entry in the Forwarding Table to record where to send packets after processing.

SFC modification: To modify an SFC, such as changing NF sequence, removing NFs, or including NFs that are already deployed in GPU, the SFC Manager could simply modify the *NFs* field of the Agent Table to record the updated NF sequence. The SFC Starter will inform the SFC Agent of the new set of NFs when processing a new packet batch, which could realize runtime fast SFC modification.

New NF addition: NFs are written as kernels, compiled to binary codes, and deployed in GPU. A straightforward approach to adding new NFs in an SFC is to pause GPU, recompile the kernels with the new NFs, and redeploy them in GPU. However, this approach could incur significant performance overhead since all SFCs in GPU have to be interrupted. In response, GEN explores the *ptx* feature of CUDA [19], which supports compiling new kernels and injecting binary codes into GPU during runtime.

However, *ptx* only supports dynamic compilation and injection of kernels declared with *__global__*, NFs in GEN are declared as *__device__* functions. Therefore, we directly write a *new SFC Agent* that includes the code of the new NF, compile this new Agent kernel, and install it into GPU. We then modify the *NFs* field in the Agent Table to record the new NF sequence. Upon start, this new SFC Agent will then launch and execute NFs including the newly added NF.



(a) Throughput (b) Latency CDF

Figure 6: Performance of RTC vs. pipeline in GPU

4 PRELIMINARY EVALUATION

We have implemented a prototype of GEN on a testbed composed of two servers. One server carries the GEN infrastructure and is equipped with components shown in Table 1. The other server runs a DPDK [12] based packet generator that sends packets to (up to 40Gbps) and receives packets from the directly connected infrastructure server. GEN infrastructure also uses DPDK to fetch packets from the NIC to the userspace. Both servers run Ubuntu 14.04 with Linux kernel 3.16.0-30-generic and DPDK 17.11. We use three NFs for evaluation including an IPv4 Router (1k entries in the forwarding table), a NIDS (about 3k rules from the Snort [27] community [27]), and an IPSec (based on HMAC-SHA1 and AES-128-CBC algorithms).

Performance improvement from RTC: We first evaluate the performance improvement from RTC-based SFC as opposed to pipeline-based SFC (using the way shown in Figure 4(a)) in GPU-accelerated NFV. As Figure 6(a) shows, RTC-based SFC improves throughput by 29.2% and 28.1% when the packet size is 64B and 128B, respectively. Figure 6(b) shows that the latency can be reduced by 33.7% at the 95th percentile at a traffic rate of 20 Gbps with the packet size distribution derived from [1], when comparing to the pipeline-based SFC. This demonstrates that with proper selection of SFC models, GPU could achieve high performance in both latency and throughput.

Fast Elasticity: We next evaluate whether GEN could adaptively, precisely, and quickly adjust the batch size to handle varying traffic volumes. Starting from 10Gbps, we increase the input traffic rate to the SFC by 10Gbps every 5 seconds until the packet rate reaches 40Gbps. Then we decrease the traffic rate to 10Gbps with the same speed. As shown in Figure 7, GEN can dynamically adjust the batch size with the variation of input traffic rate, to adaptively scale out and in for different traffic volume requirements. Furthermore, each time the input packet rate increases, it quickly converges to a stable value that requires just enough GPU threads to handle the traffic. This could demonstrate GEN's capability to precisely and quickly adapt to traffic volume changes.

5 CONCLUSION AND DISCUSSION

This paper presents GEN, a GPU-accelerated elastic framework for NFV. GEN exploits GPU to push one step towards building an NFV framework with both performance and elasticity for SFC in mind. We next present our insights on some future research directions towards this target.

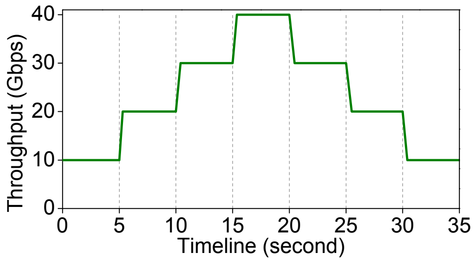


Figure 7: Elasticity of GEN to adapt to traffic dynamics

SFC performance enhancement in GPU: There exist some potential approaches to further enhance performance when supporting NFV with GPU. First, instead of copying the whole packets, a possible enhancement would be only extracting packet fields visited by NFs and sending them between CPU and GPU, in order to reduce latency. Second, packets fetched by DPDK are first stored in huge pages, then copied to the host memory [19], and finally copied to GPU. As a potential acceleration, we could benefit from CUDA Direct [19] or driver modification [6] to reduce memory copying. Finally, NFP [28] and ParaBox [34] proposed to run independent NFs in parallel in CPU-based SFCs. GEN runs SFCs with GPU, which could naturally integrate and benefit from techniques such as NF parallelism.

Coordination between CPU and GPU: GEN could benefit from this coordination in two aspects. First, we could deploy an NF on both CPU and GPU and perform load balancing as proposed in NBA [17]. This approach can fully utilizing the processing power of a commodity server. However, we are challenged to design a common programming abstraction for NFs on both CPU and GPU as well as the load balancing mechanisms. Second, we could only offload partial NFs that are suitable to be supported by GPU for high performance [6, 15]. We will study the both as our future work.

Impacts of dynamic traffic load: In the current version of GEN, we haven't fully tested whether there exist packet drops under bursty traffic. Also, the performance of GEN under heterogeneous traffic workloads has not been evaluated. As our future work, we will perform thorough analysis and evaluations about the impacts of dynamic traffic loads.

ACKNOWLEDGMENTS

We would like to thank our shepherd KyoungSoo Park and anonymous reviewers for their thoughtful feedback. This work is supported by the National Key R&D Program of China (2017YFB0801701), and the National Science Foundation of China (No.61472213).

REFERENCES

[1] Theophilus Benson, Aditya Akella, and David A Maltz. 2010. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 267–280.

[2] Anat Bremler-Barr, Yotam Harchol, and David Hay. 2016. OpenBox: a software-defined framework for developing, deploying, and managing network functions. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 511–524.

[3] Rohan Gandhi, Hongqiang Harry Liu, Y Charlie Hu, Guohan Lu, Jitendra Padhye, Lihua Yuan, and Ming Zhang. 2015. Duet: Cloud scale load balancing with hardware and software. *ACM SIGCOMM Computer Communication Review* 44, 4 (2015), 27–38.

[4] Aaron Gember, Anand Krishnamurthy, Saul St John, Robert Grandl, Xiaoyang Gao, Ashok Anand, Theophilus Benson, Vyas Sekar, and Aditya Akella. 2013. Stratos: A Network-Aware Orchestration Layer for Virtual Middleboxes in Clouds. *arXiv preprint arXiv:1305.0209* (2013).

[5] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. 2014. OpenNF: Enabling innovation in network function control. In *ACM SIGCOMM Computer Communication Review*, Vol. 44. 163–174.

[6] Younghwan Go, Muhammad Asim Jamshed, Young-Gyoun Moon, Changho Hwang, and KyoungSoo Park. 2017. APUNet: Revitalizing GPU as Packet Processing Accelerator. In *USENIX Symposium on Networked Systems Design and Implementation*. 83–96.

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

[8] Kshitij Gupta, Jeff A Stuart, and John D Owens. 2012. A study of persistent threads style GPU programming for GPGPU workloads. In *Innovative Parallel Computing (InPar), 2012*. IEEE, 1–14.

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

[10] J Halpern and C Pignataro. 2015. Service Function Chaining (SFC) Architecture. *draft-ietf-sfc-architecture-07 (work in progress)* (2015).

[11] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. 2010. PacketShader: a GPU-accelerated software router. In *ACM SIGCOMM Computer Communication Review*, Vol. 40. ACM, 195–206.

[12] Intel. 2018. Data Plane Development Kit (DPDK). http://dpdk.org

[13] Muhammad Asim Jamshed, Jihyung Lee, Sangwoo Moon, Insu Yun, Deokjin Kim, Sungryoul Lee, Yung Yi, and KyoungSoo Park. 2012. Kargus: a highly-scalable software-based intrusion detection system. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 317–328.

[14] Dilip A Joseph, Arsalan Tavakoli, and Ion Stoica. 2008. A policy-aware switching layer for data centers. In *ACM SIGCOMM Computer Communication Review*, Vol. 38. ACM, 51–62.

[15] Anuj Kalia, Dong Zhou, Michael Kaminsky, and David G Andersen. 2015. Raising the Bar for Using GPUs in Software Packet Processing.. In *USENIX Symposium on Networked Systems Design and Implementation*. 409–423.

[16] Georgios P Katsikas, Tom Barbette, Dejan Kostic, Rebecca Steinert, and Gerald Q Maguire Jr. 2018. Metron: NFV Service Chains at the True Speed of the Underlying Hardware. In *USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association.

[17] Joongi Kim, Keon Jang, Keunhong Lee, Sangwook Ma, Junhyun Shim, and Sue Moon. 2015. NBA (network balancing act): A high-performance packet processing framework for heterogeneous processors. In *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 22.

[18] S Kumar, M Tufail, S Majee, C Captari, and S Homma. 2015. Service Function Chaining Use Cases in Data Centers. *IETF SFC WG* (2015).

[19] CUDA Nvidia. 2010. Programming guide.

[20] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. 2015. E2: a framework for NFV applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, 121–136.

[21] Aurojit Panda, Sangjin Han, Keon Jang, Melvin Walls, Sylvia Ratnasamy, and Scott Shenker. [n. d.]. NetBricks: Taking the V out of NFV. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), USENIX OSDI*, Vol. 16.

[22] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. 2013. SIMPLE-fying middlebox policy enforcement using SDN. In *ACM SIGCOMM computer communication review*, Vol. 43. ACM, 27–38.

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

[24] Shriram Rajagopalan, Dan Williams, Hani Jamjoom, and Andrew Warfield. 2013. Split/merge: System support for elastic execution in virtual middleboxes. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation*. 227–240.

[25] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K Reiter, and Guangyu Shi. 2012. Design and implementation of a consolidated middlebox architecture. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 24–24.

[26] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. 2012. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review* 42, 4 (2012), 13–24.

[27] Snort. 2018. Snort community rules. https://www.snort.org/

[28] Chen Sun, Jun Bi, Zhilong Zheng, Heng Yu, and Hongxin Hu. 2017. NFP: Enabling Network Function Parallelism in NFV. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 43–56.

[29] Weibin Sun and Robert Ricci. 2013. Fast and flexible: parallel packet processing with GPUs and click. In *Architectures for Networking and Communications Systems, ACM/IEEE Symposium on*. IEEE, 25–35.

[30] Giorgos Vasiliadis, Lazaros Koromilas, Michalis Polychronakis, and Sotiris Ioannidis. 2014. GASPP: A GPU-Accelerated Stateful Packet Processing Framework.. In *USENIX Annual Technical Conference*. 321–332.

[31] Tsung Tai Yeh, Amit Sabne, Putt Sakdhnagool, Rudolf Eigenmann, and Timothy G Rogers. 2017. Pagoda: Fine-Grained GPU Resource Virtualization for Narrow Tasks. In *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 221–234.

[32] Xiaodong Yi, Jingpu Duan, and Chuan Wu. 2017. GPUNFV: a GPU-Accelerated NFV System. In *Proceedings of the First Asia-Pacific Workshop on Networking*. ACM, 85–91.

[33] Kai Zhang, Bingsheng He, Jiayu Hu, Zeke Wang, Bei Hua, Jiayi Meng, and Lishan Yang. 2018. G-NET: Effective GPU Sharing in NFV Systems. In *USENIX Symposium on Networked Systems Design and Implementation*. USENIX, 187–200.

[34] Yang Zhang, Bilal Anwer, Vijay Gopalakrishnan, Bo Han, Joshua Reich, Aman Shaikh, and Zhi-Li Zhang. 2017. ParaBox: Exploiting Parallelism for Virtual Network Functions in Service Chaining. In *Proceedings of the Symposium on SDN Research*. ACM, 143–149.