Network functions in virtualized GPU environment

Avinash Kumar Chaurasia
Computer Science and Engineering
Indian Institute of Technology Bombay, India
Email: avinashk@cse.iitb.ac.in

Uday Kurkure, Hari Sivaraman and Lan Vu
Performance Engineering
VMware, USA
Email: {ukurkure, hsivaraman, lanv}@vmware.com

Sairam Veeraswamy
Innovation Program
VMware, India
Email: sveeraswamy@vmware.com

Bhaskaran Raman
Computer Science and Engineering
Indian Institute of Technology Bombay, India
Email: br@cse.iitb.ac.in

***Abstract*—Network functions virtualization (NFV) is an emerging and important research topic in the telecom domain to provide innovative solutions that reduce cost and improve network processing & computing efficiency. Attaining high throughput is a key challenge and an important goal in the deployment of NFV. While horizontal scaling can improve the throughput, it increases the management complexity for cloud native NFVs. Another approach is using accelerators to enhance the NFV throughput, which can help reduce cost and simplify the large-scale deployment of NFV. In the cloud environment, the increasing use of virtualized GPU by large cloud providers requires GPU-based NFV solutions to leverage virtual GPUs for efficient cloud deployment. However, most researches in NFV acceleration are done for non-virtualized GPU while utilizing virtualized GPU for NFV is still under-investigated in the industry and research community. In this paper, we present our study that addresses this problem by exploring virtualized GPU to maximize the benefits of NFV and analyze NFs behavior with respect to virtual GPU in multiple use cases. We demonstrate using virtual GPU increases GPU utilization and provides higher performance and throughput for GPU-based NFV. Our experimental results show that virtual GPU can help NFV deliver up to 288% in throughput comparing to using passthrough GPU.**

***Keywords*—NFV, GPU, vGPU, hardware assisted virtualized GPU.**

I. Introduction

Network functions (NFs) such as firewall, HTTP proxy, IPSec, etc., usually come in the form of special hardware known as application specific integrated circuits (ASICs). These ASICs are very costly and deployed in almost every generation of telecommunications(telecom) network. Since every generation of the cellular network requires a different set of NFs, telecom companies had to invest massive capital with each generation change to upgrade the network processing. For example role of mobility management entity (MME) in 4G is taken over by three NFs in 5G, namely mobility management function (AMF), session management function (SMF), and authentication server function (AUSF). This huge upfront cost is one of the important reasons why every telecom company is hesitant to bring new technologies and want to continue with the old generation network as much as possible. Over the last decade, researchers tried to bring down the cost with network function virtualization(NFV). With NFV, ASICs are supposed to be replaced by NFs software, and NFs are executed over generic compute units such as x86 cores. This helps in cost reduction with respect to deploying new networking hardware. Furthermore, it minimizes maintenance efforts because the new upgrade is mostly done at the software level and can be automated. Therefore, the advent of NFs makes telecommunications more agile and rapidly adapt to new demands from the customers. However, NF deployment over x86 lacks the processing power of ASICs because of the many inherent software overheads. One of the overhead lies with the TCP/IP stack [5]. The kernel allocates a buffer for each incoming packet, then copies it into the kernel buffer, copies it further into the application buffer, and later deallocates the kernel buffer once it is no more required. Another problem is the throughput obtained per CPU core. For a compute intensive NF, per core throughput is very low. In such cases, horizontal scaling can improve throughput. Horizontal scaling means packets need to be transferred over network fabric in the data center, causing extra latency overhead. Furthermore, managing horizontally scaled network function is another important research topic. Recent research tries to address a few of these problems with possible software optimizations such as DPDK [17] and netmap [15]. Both DPDK and netmap provide zero-copy via bypassing traditional TCP/IP stack and directly copy the packet in application mapped buffers.

Recent research is more focused on using accelerators such as GPU [5]–[7], [18] and FPGA [8] for achieving better throughput. However, using GPU for high-speed NF processing introduces a different set of challenges. Discrete GPUs suffer from PCI bus performance, as each packet needs to move between GPU memory and CPU main memory in both directions over the PCI bus. GPU Direct technology solves the problem by offering direct packet transfer between NIC and GPU memory, thus avoiding the CPU's main memory; however, even with this technology, packets are transferred over a PCI bus.

Nowadays GPUs are an integral part of the cloud offered by various vendors [2], [4], [11]. GPUs demand in cloud setup is mostly related to solving compute intensive problems having SIMD (Single Instruction Multiple Data) nature since

it can solve them relatively at ease with their higher compute cores. These problems span across multiple areas such as High Performance Computing (HPC), Machine learning, Deep Learning, etc. In all the GPU enabled cloud infrastructures [2], [4], [11], GPU is provisioned using a PCI passthrough mode. However, provisioning this way negated the advantages of virtualization. In passthrough mode, a GPU is directly assigned to a VM and cannot be multiplexed across multiple VMs. Many software based multiplexing techniques such as API-level interception [16], remote GPU abstraction [19] and kernel slicing [9] have been recommended for the purpose. However, to provide a better performance, NVIDIA introduced virtual GPUs to provide hardware assisted multiplexing. The technology allows a GPU to be virtualized and exposed as multiple vGPUs such that each VM gets a separate vGPU. NVIDIA driver schedules these VMs on GPU on a temporal basis.

In a cloud setup, both of these two modes (passthrough and virtual GPU) are available. However, when it comes to NFV acceleration research, a significant portion of literature covers non virtualized GPUs [3], [5], [6] or passthrough mode GPU [21]. There is little to no work that provides NFV acceleration in a cloud setup, especially hardware-assisted virtual GPU mode. Since passthrough mode is very similar to using GPU in a non-virtualized manner with a little overhead, both these setups' performance would be the same. However, virtual GPU setups are different, and without a study, predicting what could be a better way to utilize GPU for NFV acceleration is difficult.

Furthermore, virtual GPU setups provide two fine-tuning knobs for application performance: scheduling algorithm and vGPU profile. Selecting an appropriate scheduling algorithm and vGPU profile affects the performance of workloads. Cloud providers need to know which scheduling algorithm and vGPU profile are most suitable for better GPU utilization concerning the workload. In this work, we are trying to answer that for NFs deployment, as we evaluate and characterize the performance of NVIDIA hardware assisted GPU virtualization using two compute intensive NFs. Our work helps achieve line-rate throughput by providing inputs on the selection of scheduling algorithms and vGPU profile. As a part of this work, the following are our contributions:

- A detailed description of hardware assisted GPU virtualization provided by NVIDIA, and it's working and limitations.
- We present our design of optimized versions of two compute-intensive NFs for the GPUs.
- Detailed experimental evaluation of these NFs on two of the most promising scheduling algorithms and different vGPU profiles.
- Method of selecting an appropriate scheduling policy and vGPU profile for optimal NF throughput using hardware assisted virtualized GPU in cloud setup.

The next section will explain recent works on NFV acceleration using GPU. Then in section 3, we will explain GPU virtualization. In section 4, we will present our choice of network function, design, and implementation. Section 5 describes the experimental evaluation of these NFs over two different scheduling algorithms and vGPU profiles. Finally, section 6 concludes our work with specific input on NF deployment over virtualized GPUs.

II. Related Works

Network function virtualization aims to reduce capex and opex, has a faster deployment, and improves innovation. Another purpose is to minimize the dependencies on the proprietary hardware to avoid vendor lock-in. Click [12] provides a modular approach for building network functions. It uses elements as basic building blocks that can be combined to perform complex network functions. Most of the research regarding NFV are concentrated on achieving high throughput and low latency over x86. However, few works in the past tried to improve performance with accelerators' help (FPGA and GPU). ClickNP [10] is a framework that provides 40Gbps line-rate throughput and low latency by accelerating NFs over FPGA. ClickNP provides basic elements similar to Click [12] for modular NF development. Furthermore, it provides C-like high level language abstraction instead of complex low-level HDL (hardware description languages). PacketShader [5], SSLShader [7], Kargus [6] and G-NET [21] are GPU accelerated NFV frameworks. PacketShader [5], SSLShader [7] and Kargus [6] are specially designed while keeping a particular NF in mind. PacketShader [5] accelerates software router over GPU while optimizing many system software for better throughput. SSLShader [7] is designed for cryptographic computation over GPU and Kargus [6] is intrusion detection system offloaded over GPU. Compared to these three, G-NET [21] is a generic framework and provides NFV chaining as additional functionality. G-NET [21] uses HYPERQ enabled GPU to support the spatial sharing of GPU. With the help of spatial sharing, it gives a complete NFV chaining functionality over the GPU. With G-NET's exception, all of the GPU accelerated NF framework do not use hardware assisted virtualization. Even G-NET [21] uses GPU in passthrough mode. In the next section, we will explain hardware assisted virtualized GPU.

III. Background

Initially, GPUs can be part of virtual machines (VMs) only in passthrough mode, i.e., each VM must have a dedicated GPU. The benefits of virtualization are relatively low when VMs have a dedicated GPU. Cloud provider's ability to provide cheaper resources comes from over provisioning the resource. Since providers cannot over provision GPU in passthrough mode, it limits the provider's ability to provide GPU infrastructure in cost effective manner. Hardware assisted virtualization provided by NVIDIA is an attempt to provide similar over provisioning benefits over GPU.

Hardware assisted virtualized GPU or vGPU is comparatively a newer feature which divides a single GPU into multiple virtual GPU (vGPU) devices as shown in the figure 1. Each

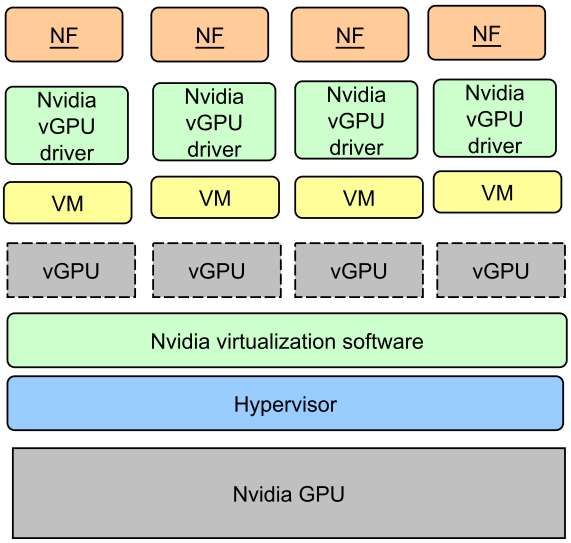


Fig. 1: Hardware assisted virtual GPU stack

of these vGPUs can be assigned to VMs[1]. GPU virtualization is managed via the drivers installed inside the VM and the hypervisor [13]. It exposes vGPUs to VMs and shares a physical GPU across multiple VMs (Figure 1), and VM uses non virtualized GPUs with the illusion created by hypervisor and NVIDIA drivers. Two parameters that configure vGPUs are the scheduling algorithm and the vGPU profile. These parameters required configuration at the hypervisor level with NVIDIA virtualization software as a part of NVIDIA driver. The vGPU profile is responsible for deciding the memory reserved per vGPU that a hypervisor should expose among these two configuration parameters. This reserved memory further determines how many vGPUs should be exposed. For example, if GPU consists of a 16GB memory and 4GB is reserved per vGPU based on the vGPU profile, then a total of 4 vGPUs (16/4) are possible, each having a 4GB memory. Table I lists all the possible vGPU profile for NVIDIA Tesla P-100.

TABLE I: vGPU profiles supported by NVIDIA Tesla P-100

vGPU profile	memory per vGPU	number of vGPUs exposed
1q	1 GB	16
2q	2 GB	8
4q	4 GB	4
8q	8 GB	2
16q	16 GB	1

The second configuration parameter is the scheduling algorithm. The scheduling algorithm is configured at the hypervisor using the NVIDIA driver and provides temporal sharing among VMs. Based on the algorithm chosen, the driver allocates the GPU resources for VM processing. Currently, the driver supports three types of scheduling algorithms:

[1]only one vGPU can be accessed by the VM

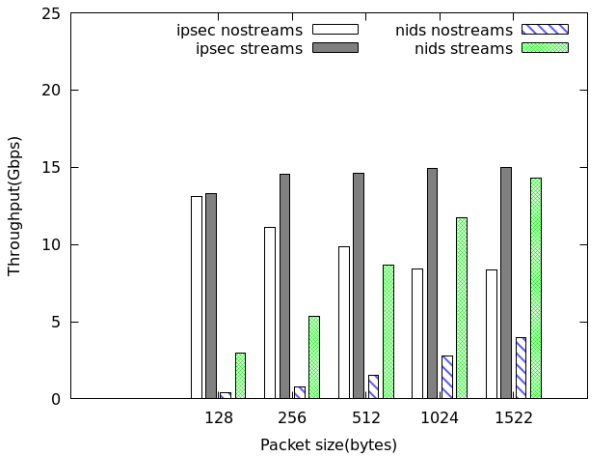
- Best effort scheduler: This scheduler utilizes GPU as much as possible, i.e., if there are four vGPU enabled VMs, and only 3 VMs need GPU computation, then the scheduler will allocate all the time slots among 3 VMs only.
- Equal share scheduler: This scheduler reserves the time slot as per active VMs (powered on VMs). If the equal share scheduler is used in the scenario mentioned above, the scheduler will divide time into four identical slots, and one slot will waste without any computation. However, if one VM is powered off, it will divide the time slot into three equal parts.
- Fixed share scheduler: This scheduler divides the time slot based on the vGPU profile, i.e., if four vGPUs can be exposed, then GPU time is divided into four slots. It does not matter whether VM is active or not.

Best effort scheduler is supposed to perform equally or better than the other two schedulers as it can dynamically adjust the time slots allocated to the VM based on circumstances. Furthermore, VMs are scheduled on GPU using a round-robin mechanism for *equal share scheduler* and *fixed share scheduler*. It is expected that *best effort scheduler* with only 1 VM should have similar performance (little lower throughput due to virtualization overhead) as passthrough mode GPU. Despite all the performance benefits vGPU can offer, there are certain limitations in using vGPU: (1) Currently, vGPU profile must be selected separately for each VM in offline mode, (2) Scheduler is chosen at the hypervisor and every time scheduler is changed, the host needs to be restarted, (3) It is not possible to power-on more VMs than calculated based on the vGPU profile since memory is kept reserved per VM, (4) Heterogeneous vGPU profile is not allowed, i.e., vGPU profile should be consistent across all the active VMs (powered on) and (5) Many GPU optimizations such as Unified Virtual Memory (UVM), CUDA visual profilers, etc. are not present in virtualized mode.

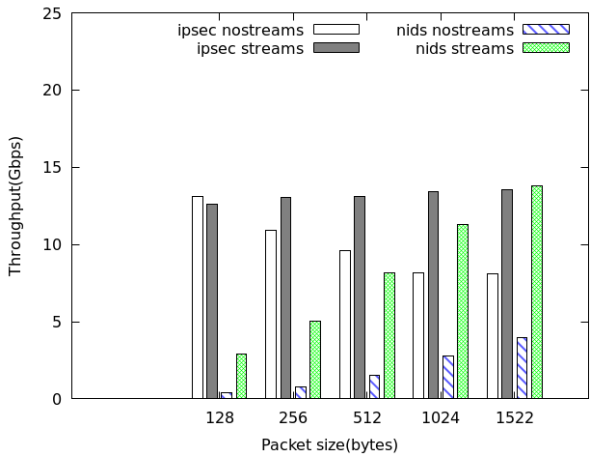
Since GPUs perform well for a compute intensive task, we will discuss two compute intensive network functions for performance evaluation of network function over GPUs in passthrough mode and virtual mode in the next section.

IV. Design and Implementation of Network Functions

Network functions can be classified into two categories: IO intensive and IO-compute intensive. Since every network function performs computation over every incoming packet, they are always IO intensive. Some NFs aren't compute intensive such as the IPv4 router. The router needs to look up the table based on the packet header's destination address and outputs the next port the packet needs to be forwarded. Whereas other NFs such as network intrusion detection system (NIDS) and Internet Protocol Security (IPSec) are compute intensive. These NFs perform computation over the payload segment of the packet. IPSec performs both HMAC and AES operation on each packet; both algorithms (cryptography) are



(a) Throughput performance over GPU in passthrough mode



(b) Throughput performance over a single vGPU

Fig. 2: Throughput of IPSec and NIDS with respect to packet size (with and without streams)

considered compute intensive. NIDS performs string matching against a predefined set of rules for intrusion detection.

We implemented both IPSec and NIDS in CUDA. Our IPSec uses HMAC-SHA1 and AES-128 bit in CBC mode. Openssl [14] AES-128 bit CBC encryption and decryption algorithm is rewritten in CUDA as part of our implementation. NIDS is implemented using Aho-Corasick algorithm [1]. It is based on deterministic finite automata (DFA). However, we have used only 147 rules for building the DFA state. Our design allocates a CUDA thread per packet. In each round, we first copy the packets in GPU memory, then, kernel is launched where CUDA threads perform computing on their respective packets. On completion, the kernel terminates, and the result is copied back to host memory. To optimize the performance of these NFs, we heavily used constant memory for read-only data access. These read-only data are copied to GPU constant memory at the initialization stage. The memory footprint of these data varies according to NF. For instance, IPSec accesses big tables for encryption/decryption, and keeping these tables in cache-friendly constant memory boosts the performance manifold. Whereas, NIDS does not have much of predefined read-only data.

To provide a further performance boost, we used multiple CUDA streams for NF computing. In such scenarios, we always used an asynchronous mechanism of data copy between host and device. Furthermore, the total number of CUDA threads are equally divided among the streams. Data (packets and results) are also divided equally among streams and asynchronously transferred between host and device, to leverage parallel execution and data copy. We also found that the performance of NF does not monotonically increase with the increase of CUDA streams. When we increase the number of CUDA streams, NF performance also increases until it reaches a certain optimal level; after that, it stays the same or drops. In our experiments, the number of CUDA streams is kept at this optimal value for the best achievable throughput. Our experiments in the later sections show that using multiple CUDA streams provides better performance than using the default CUDA stream for computing.

V. Experimental Evaluation

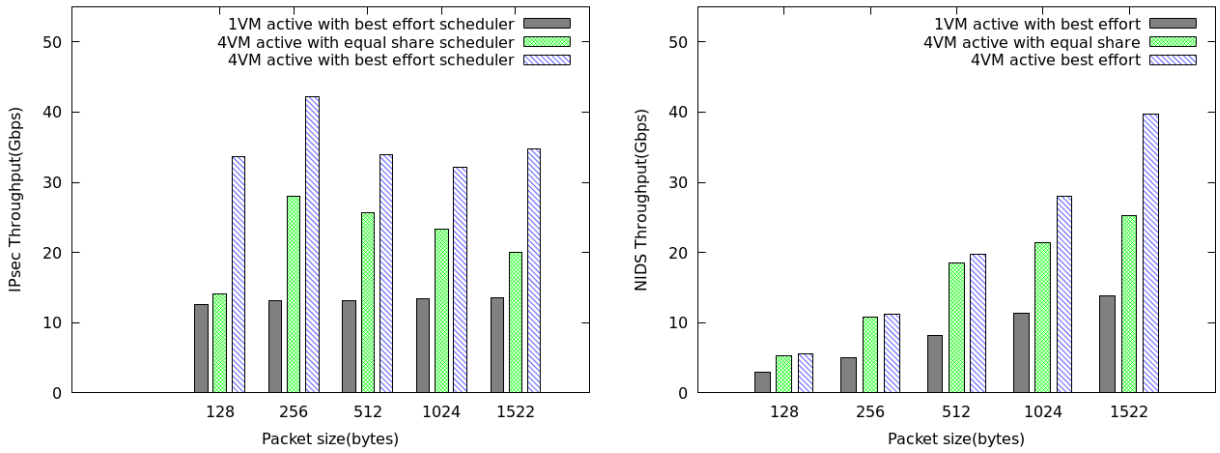
Our test setup includes a host machine (PowerEdge R740xd model) consist of 32 Intel Xeon cores operating at 2.30 GHz with 766GB memory. The host is running over an ESXi 6.5 and NVIDIA tesla P-100 GPU. CUDA version 9 is used to perform the experiments. The configuration of each VMs is as follows: Ubuntu 16.04 64 bit OS, 32 GB ram, and eight vCPU core each. Packets are generated and kept in memory so that IO overhead (NIC to main memory and main memory to NIC) does not act as a variable in our analysis.

In the next few subsections, we analyze the experimental results of NFs (IPSec and NIDS) over vGPU and compare it with passthrough mode performance. These experiments help in answering many questions for cloud provider: (1) Whether passthrough mode is better or virtual GPU, (2) How many vGPU should be activated for better throughput, (3) What will happen to NF throughput if the cloud provider wishes to multiplex different types of workload such as machine learning workload with NFs.

In most cases, we are using *best effort scheduler* unless explicitly mentioned. Additionally, we are using the term nostream or without stream when the default CUDA stream is used, as every CUDA program uses a default stream 0. When we mentioned streams in experiments, it is for the cases where we specially programmed it to use multiple CUDA streams.

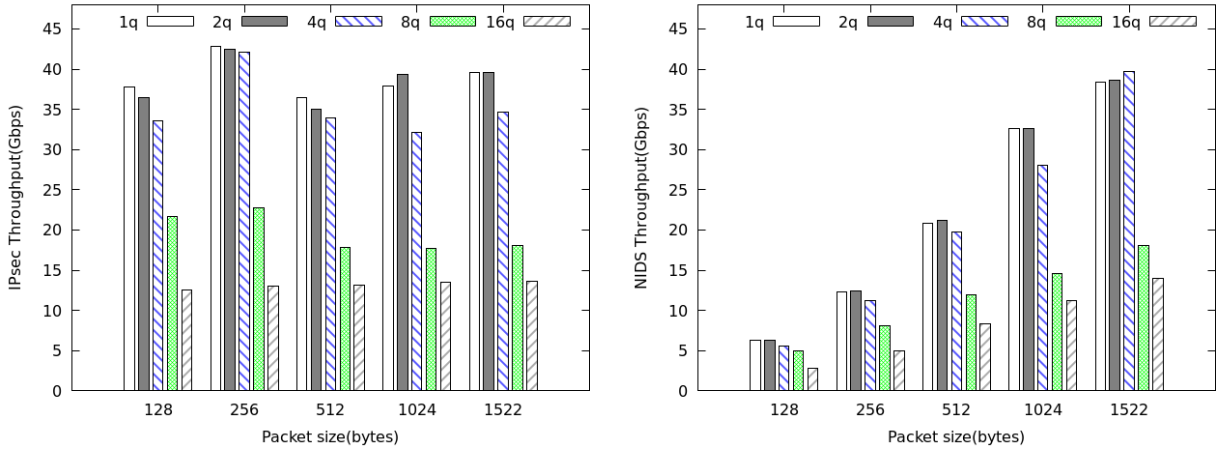
A. Passthrough mode

The throughput of IPSec and NIDS network function when GPU is in passthrough mode is presented in figure 2a. Both NFs behave differently with respect to packet size and CUDA streams. While IPSec throughput decreases with increased packet size for default CUDA stream (nostreams), whereas with multiple CUDA streams, throughput more or less stays



(a) IPSec throughput (b) NIDS throughput

Fig. 3: Comparison of NFs throughput with respect to schedulers and number of online VMs



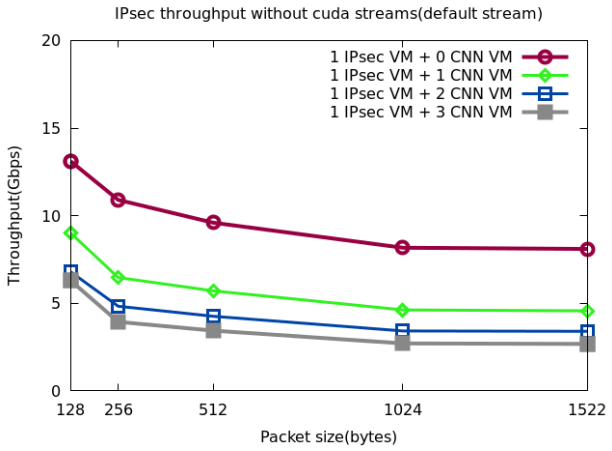
(a) IPSec throughput (b) NIDS throughput

Fig. 4: Combined NF throughput with respect to vGPU profile

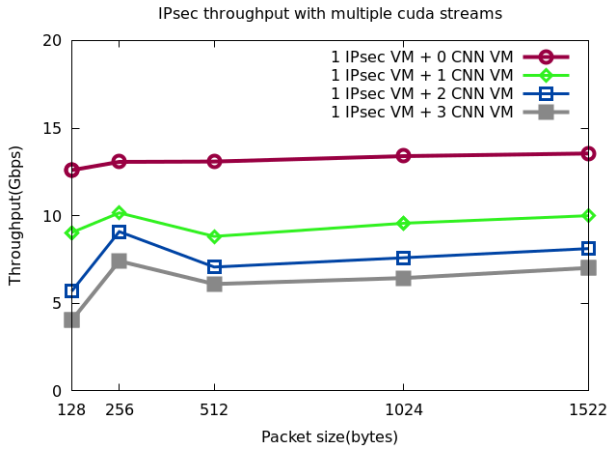
stagnant. However, NIDS NF throughput rises with an increase in packet size. Usage of streams (multiple CUDA streams) does not change the trend; the only difference is, NIDS performance is much better with streams. Figure 2a shows that NIDS throughput is dependent on packet size, which means it processes the packet in constant time regardless of packet size. However, lower throughput than IPSec for smaller packet sizes means the compute requirement for NIDS is higher. IPSec performance is mostly restricted by PCI transfer speed. The larger the packet size, the more bytes need to be transferred hence lowering the throughput. CUDA streams help in alleviating this bottleneck by asynchronously moving the packets in the background. For a packet size of 1522B, IPSec obtained a throughput of 8.32Gbps when multiple CUDA streams are not used (i.e., default CUDA stream is being used) in comparison to 15Gbps when multiple CUDA streams are used. NIDS obtained a throughput of 3.96Gbps without stream usage (default CUDA stream) and 14.29Gbps with streams (multiple CUDA streams) usage for 1522B packets.

B. NFs baseline performance in vGPU mode

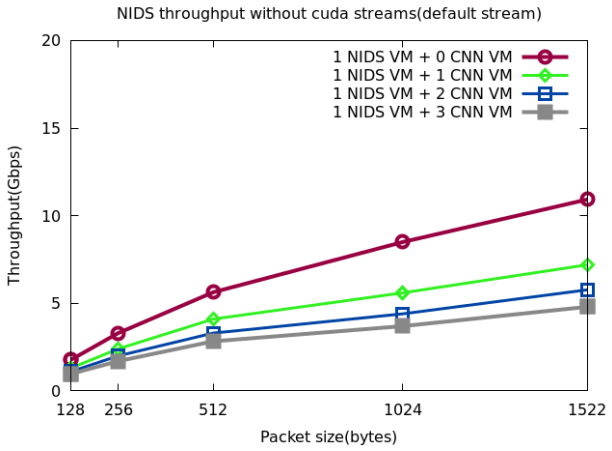
This section studies the NFs performance execution in a single vGPU enabled VM. For this experiment, we have used a 4q vGPU profile and *best effort scheduler*. Figure 2b shows that vGPU closely mimics the performance of IPSec and NIDS NFs in passthrough mode. IPSec throughput is just 9.7% lower than IPSec on passthrough mode as 13.55Gbps throughput is obtained in vGPU configuration against 15Gbps passthrough mode. This is caused by the overhead of virtualization introduced by vGPU. It proves our hypothesis that single vGPU enabled VM performance with best effort scheduler is close to passthrough mode performance. Moreover, the key benefits of vGPU for NFs are in having multiple vGPUs sharing a single physical GPU, which we will show in the section V-C.



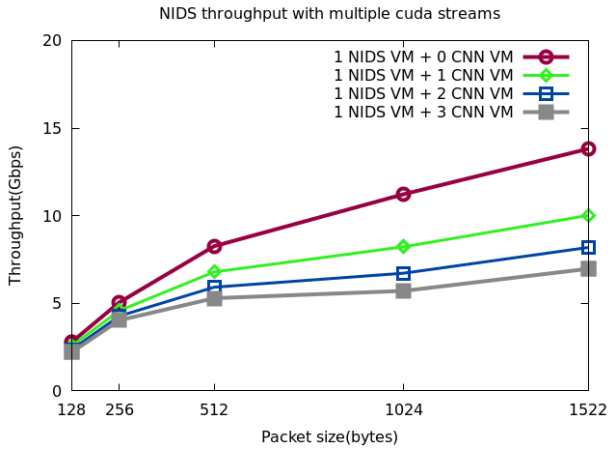
(a) IPSec performance without CUDA streams (default stream)



(b) IPSec performance with multiple CUDA streams



(c) NIDS performance without CUDA streams (default stream)



(d) NIDS performance with multiple CUDA streams

Fig. 5: NF performances when CNN workload are used simultaneously

C. NF throughput improvement with parallel usage of vGPU

Although the drop in NF throughput is sublinear for single vGPU, the combined throughput of all the vGPUs exceed passthrough mode throughput even though the underlying hardware is the same. Figure 3 shows the experimental result of combined NF throughput of all vGPUs. Combined throughput means the sum of throughput obtained by executing NF in all the possible vGPU enabled VMs. When we used the 4q vGPU profile for our experiments, we observed that the combined throughput of either scheduler (*equal share scheduler* and *best effort scheduler*) exceeded passthrough mode throughput or one active VM throughput. With *equal share scheduler*, we achieved 148% and 182% throughput for IPSec and NIDS, respectively, compared to their respective one active VM throughput. Whereas, with *best effort scheduler*, we achieved 256% and 288% throughput for IPSec and NIDS, respectively, compared to their respective one active VM throughput. Furthermore, *best effort scheduler* outperforms *equal share scheduler* in every possible scenario. *Best effort scheduler* with 4 VM outperforms *best effort scheduler* with 1 VM because in 1 VM scenario kernel waits for the packet transfer to complete before performing computation over that (even with multiple streams). However, the same is not the case in scenarios where NFs are computed using four vGPU enabled VM, NVIDIA best effort scheduler schedules the VM that is ready to perform computation, and packet transfer for the rest of the VMs will continue in parallel. In a scenario when *equal share scheduler* is used, time slots get reserved per active VM. Hence, if VM gets scheduled when the packet is being transferred to (or from) GPU, it will lead to wastage of some GPU cycles as the kernel have to wait for packet transfer for useful computation. Since we have already established it as the fact that NFs with streams perform better than NFs without stream (default stream), we are skipping the performance results of the NFs without stream. However, results follow the same trends in those cases, i.e., for 4q vGPU profile, the combined throughput of 4 vGPU enabled VM using *Best effort scheduler* outperforms other scenarios.

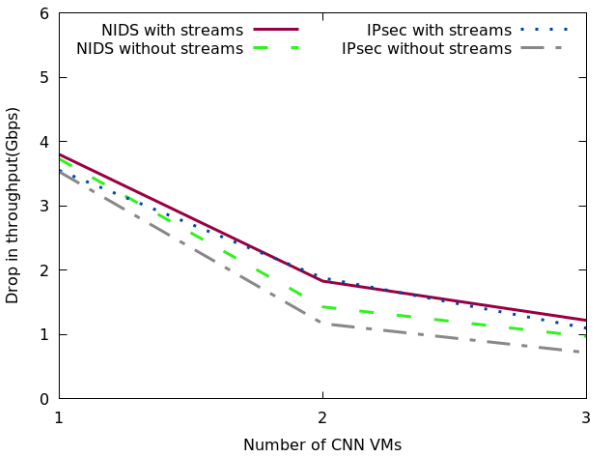


Fig. 6: Drop in throughput with respect to increase in CNN VMs. 1522B packet size is used for respective NF computation

D. Nfs performance with respect to vGPU profile

Figure 4a and figure 4b shows respective combined throughput achieved by IPSec and NIDS NF for different vGPU profiles. *Best effort scheduler* is being used for the experiment as it excels in performance compared to *equal share scheduler*. From the figures 4a, 4b we can observe that throughput gets better as number of vGPUs in the systems increases. It corroborated our claim that lesser the time VM waits for useful computation, the better will be throughput, i.e., when VM gets scheduled, it should be computing instead of waiting for data transfer to get completed. Though we have not faced any memory allocation restriction for a P-100 Tesla NVIDIA GPU, it is completely possible for some other GPU where vGPU has lesser reserved memory. NF throughput over such vGPUs might suffer from memory allocation problem, hence, lowering the throughput as all packets might not fit in the reserved vGPU memory. Hence, the vGPU profile must be selected based on NF performance on the particular GPU for optimal throughput.

E. NF performance with mixed workload

This section analyzes the impact of NFs performance when it shares GPU with other machine learning workloads using vGPU. In this case, our experimental setup is as follows: one vGPU enabled VM is allocated for NF, and the rest of the vGPU enabled VMs are executing TensorFlow [20] workload. Tensorflow workload is used for convolutional neural network (CNN) benchmark. We used the 4q vGPU profile for the experiment. The throughput performance of IPSec and NIDS (when CNN workload is present) with respect to packet size is presented in the figure 5. Tensorflow (CNN) workload lowered the throughput of the NFs. As the number of VMs with TensorFlow workload increases, the throughput of NF decreases. However, a reduction in throughput for either NFs (IPSec and NIDS, with or without CUDA streams) is not proportional to an increase in the number of TensorFlow (CNN) VMs (Figure 5). Furthermore, as it can be observed from figure 6, drop in throughput, more or less, remains the same across both NFs(with or without CUDA streams). As per our analysis, this sublinear decrease (Figure 6) in throughput with an increase in TensorFlow (CNN) VMs is attributed to the high IO nature of NF workloads. Since time slots are distributed among multiple VMs, it is possible that when CNN workloads are executed over GPU, packets from the main memory gets transferred to vGPU memory for the NF. Hence, NFs waiting time for packet reception decreases with an increase in vGPU enabled VMs hosting CNN workload, thus a sublinear decrease in throughput. In the real world use cases, where cloud providers want a flexible deployment of multiple GPU-based workloads (like both NFs and machine learning) on a single server, we demonstrate such deployment be possible with vGPU. Since machine learning jobs like CNN are very compute intensive, they consume all GPU cycles assigned to them, reducing the GPU cycles used for NFs. This explains the reduction of NFs throughput as the number of VMs running CNN jobs increase. Hence, we suggest the optimal use of NFs with vGPU is sharing multiple vGPUs with the same NFs functions, as shown in section V-C for the optimal performance of NFs using vGPU.

VI. Conclusion

As most of the computing moves toward a hybrid cloud based environment, network function (NF) deployment also picked up the pace towards cloud based virtual deployment. Virtualization benefits are manifolds and have limitations in throughput, latency, and data locality. Accelerators like GPUs in hybrid cloud setup, both in dedicated as well as disaggregated deployment models, could help alleviate this problem. Our experiments revealed that for IO intensive NFV workloads passthrough mode is sub-optimal as the network IO overheads lower the GPU compute utilization, thus impacting the effective performance of the NFV stack. Since many modern GPUs come with hardware assisted virtualization, leveraging them can further enhance NF performance in a cloud setup. We demonstrated with our experiments that correct selection of vGPU profile (lower the better as long as memory does not become the bottleneck) and scheduling algorithm could provide better throughput and help achieve 40Gbps line-rate throughput, which is more than double of what passthrough mode can provide. Our analysis would significantly help telecommunication operators overcome some of their mind-blocks related to lower throughput with NFs deployments, thus could improve the quality of service at a lower CapEx point by avoiding proprietary hardware based solutions. Our work alleviates that fear with experimental proofs and encourages NFV deployment.

Acknowledgment

This work has been done as a part of internship at VMware, Inc. Authors would like to thank VMware, Inc for funding this work and supporting us with hardware and software required.

REFERENCES

[1] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, June 1975.

[2] Build powerful machine learning applications on the most advanced and highest performing gpu-accelerated cloud infrastructure, July 2020. https://aws.amazon.com/nvidia/.

[3] Younghwan Go, Muhammad Jamshed, YoungGyoun Moon, Changho Hwang, and KyoungSoo Park. Apunet: Revitalizing gpu as packet processing accelerator. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, pages 83–96, Berkeley, CA, USA, 2017. USENIX Association.

[4] Cloud gpus, July 2020. https://cloud.google.com/gpu/.

[5] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. Packetshader: A gpu-accelerated software router. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 195–206, New York, NY, USA, 2010. ACM.

[6] Muhammad Asim Jamshed, Jihyung Lee, Sangwoo Moon, Insu Yun, Deokjin Kim, Sungryoul Lee, Yung Yi, and KyoungSoo Park. Kargus: A highly-scalable software-based intrusion detection system. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 317–328, New York, NY, USA, 2012. ACM.

[7] Keon Jang, Sangjin Han, Seungyeop Han, Sue Moon, and KyoungSoo Park. Sslshader: Cheap ssl acceleration with commodity processors. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 1–14, Berkeley, CA, USA, 2011. USENIX Association.

[8] Christoforos Kachris, Georgios Ch. Sirakoulis, and Dimitrios Soudris. Network function virtualization based on fpgas: A framework for all-programmable network devices. *CoRR*, abs/1406.0309, 2014.

[9] *PPoPP '17: Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, New York, NY, USA, 2017. Association for Computing Machinery.

[10] Bojie Li, Kun Tan, Layong (Larry) Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, pages 1–14, New York, NY, USA, 2016. ACM.

[11] Gpu optimized virtual machine sizes, July 2020. https://docs.microsoft.com/en-us/azure/virtual-machines/sizes-gpu.

[12] Robert Morris, Eddie Kohler, John Jannotti, and M. Frans Kaashoek. The click modular router. *SIGOPS Oper. Syst. Rev.*, 33(5):217–231, December 1999.

[13] Nvidia virtual gpu technology, July 2020. https://www.nvidia.com/en-us/data-center/virtual-gpu-technology/.

[14] OpenSSL. Tls/ssl and crypto library. https://github.com/openssl/openssl, 2019.

[15] Luigi Rizzo. netmap: A novel framework for fast packet i/o. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pages 101–112, Boston, MA, 2012. USENIX Association.

[16] L. Shi, H. Chen, J. Sun, and K. Li. vcuda: Gpu-accelerated high-performance computing in virtual machines. *IEEE Transactions on Computers*, 61(6):804–816, 2012.

[17] Aayush Shrut. DPDK for Layman. https://www.linkedin.com/pulse/dpdk-layman-aayush-shrut. Accessed: 2019-03-06.

[18] Mark Silberstein, Sangman Kim, Seonggu Huh, Xinya Zhang, Yige Hu, Amir Wated, and Emmett Witchel. Gpunet: Networking abstractions for gpu programs. *ACM Trans. Comput. Syst.*, 34(3):9:1–9:31, September 2016.

[19] Federico Silla, Sergio Iserte, Carlos Reaño, and Javier Prades. On the benefits of the remote GPU virtualization mechanism: The rCUDA case. *Concurrency and Computation: Practice and Experience*, 29(13), 2017.

[20] Tensorflow benchmarks, July 2020. https://github.com/tensorflow/benchmarks.

[21] Kai Zhang, Bingsheng He, Jiayu Hu, Zeke Wang, Bei Hua, Jiayi Meng, and Lishan Yang. G-net: Effective gpu sharing in nfv systems. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation*, NSDI'18, Berkeley, CA, USA, 2018. USENIX Association.