G-NET: Effective GPU Sharing In NFV Systems

Kai Zhang*, Bingsheng He^, Jiayu Hu#, Zeke Wang^,
Bei Hua#, Jiayi Meng#, Lishan Yang#

*Fudan University
^National University of Singapore
#University of Science and Technology of China
Network Function Virtualization (NFV)

- **Network Functions**: nodes on the data path between a source host and a destination host
  - Firewall, NIDS, IPS, Gateway, VPNs, Load Balancers, etc.

- **NFV** is a network architecture concept: **hardware** => **software**
  - Based on **virtualization techniques**
  - **Easier to manage/deploy, higher flexibility, higher scalability, easier to validate**, etc.
  - Construct **service chains** to provide specific services to meet different demands
GPUs in Accelerating Network Functions

- **GPUs** are proven to be a good candidate for accelerating network functions
  - **Router** - PacketShader [Sigcomm’10]
  - **SSL reverse proxy** - SSLShader [NSDI’11]
  - **NIDS** - Kargus [CCS’12], MIDeA [CCS’11]
  - **NDN Router** - [NSDI’13]

1. **Massive Processing Units**
   - Network functions — large number of packets
   - GPU — thousands of cores for parallel processing

2. **Massively Hiding Memory Access Latency**
   - Network functions — large number of memory accesses in processing packets
   - GPUs can effectively hide memory access latency with massive hardware threads and zero-overhead thread scheduling (a GPU hardware support)
GPU-Accelerated Network Functions

RX → Pre-Processing → GPU Processing → Post-Processing → TX

GPU
GPU-Accelerated Network Functions

RX
- Packet parsing, batching, etc.
- Pre-Processing

TX
- Construct/filter packet, etc.
- Post-Processing

Compute/memory-intensive tasks

Parallel Processing in GPUs
Why GPUs Have not Been Utilized in NFV Systems?

- **Current GPU-based NFs - *Exclusive Access* to GPU**
  - The GPU is only accessed by **one network function**
Why GPUs Have not Been Utilized in NFV Systems?

- **Temporal GPU Sharing** - Only kernels from one VM can run on the GPU at a time

Inefficient

```
Firewall  NIDS  IPsec  Router
```

Virtualization

- Bit vector linear search
- Aho-Corasick algorithm
- AES and SHA1
- DIR-24-8-BASIC

GPU
Current Way of GPU Virtualization is Inefficient

- **Temporal GPU Sharing** - Only kernels from one VM can run on the GPU at a time
- **GPU underutilization**

Input: **20 Mpps**

GPU capability: **70 Mpps**
Current Way of GPU Virtualization is Inefficient

- **Temporal GPU Sharing** - Only kernels from one VM can run on the GPU at a time
  - **GPU underutilization**
  - **Higher latency**

![Diagram showing exclusive access and temporal sharing](image)
Spatial GPU Sharing

(1) **Exclusive access**

(2) **Temporal sharing**

(3) **Spatial sharing**

- **Spatial GPU sharing** — multiple kernels run on the GPU **simultaneously**
  - Minimize the **interference** of kernel executions from other NFs (Latency)
  - Enhance utilization - Kernels from VMs can run on the GPU simultaneously (Throughput)
Hyper-Q for Spatial GPU Sharing

- **Hyper-Q** for spatial GPU sharing
  - A technique that enables GPU kernels from the **same GPU context** to execute on the GPU simultaneously

**Challenges**
1. VMs have **different GPU context** => **Cannot** utilize Hyper-Q directly
2. Kernels utilizing Hyper-Q can **access the entire memory space** => **Security issue**
3. NFs are **not aware of** the existence of other NFs; An NF tries to maximize its resources would influence other NFs => Demanding **scheduling** and **resource allocation** schemes
The Goal of **G-NET**

Network Function Virtualization

- Flexibility
- Scalability
- Agility
- Easy to manage/deploy
- Easy validation

**GPU**

- Spatial GPU Sharing
- Security
- Development
- Resource Allocation
- Scheduling

**G-NET: NF-Hypervisor Co-Design**
G-NET: GPU Virtualization

A proxy creates a common context in the hypervisor for spatial GPU sharing.

Use API remoting to launch GPU operations.

Receive requests, perform GPU ops, send response.
G-NET: System Workflow

Zero-copy principle applied in system implementation

Framework

Switch

Manager

Common Context

NIC

GPU

VMs

Hypervisor
Achieve Predictable Performance

**How to control the performance of an NF?**

- Throughput
- Latency

**GPU Kernel Performance**

- Quantity of Work
- Compute Resource

**Batch Size**

**How to guarantee the performance of a service chain?**

- Firewall
- NIDS
- IPsec
- Router

- Virtualization

**How to allocate GPU resources?**

- GPUs utilize fast **thread switching** to enhance hardware utilization
- GPUs have **massive hardware threads** (#thread >> #core)
- Unlike CPUs, a GPU thread is **unable** to be bound to a specific core
Achieve Predictable Performance

**How to control the performance of an NF?**

- Throughput
- Latency

**GPU Kernel Performance**

**How to guarantee the performance of a service chain?**

- Firewall → NIDS → IPsec → Router

**Batch Size**

- Quantity of Work
- Compute Resource
- Thread block

- **Our Approach**
  - **Streaming Multiprocessor (SM)** as the basic unit for resource allocation
  - One thread block can only run on one SM; A thread block would be scheduled to run on an idle SM when there are available ones
  - A thread block is allocated with one SM when

\[
\text{Total #thread blocks} \leq \text{Total #SMs}
\]
Service Chain Based Scheduling

• How to optimize the performance of a service chain with limited compute resources
  • NFs have different processing tasks

• Service chain based scheduling and resource allocation
  • Locate the bottleneck NF (the NF with the lowest throughput $T$)
  • Allocate resources for all NFs in the service chain to achieve throughput $T \times (1+P)$ \hspace{1cm} (0<P<1)

<table>
<thead>
<tr>
<th>#SM</th>
<th>Firewall</th>
<th>NIDS</th>
<th>IPsec</th>
<th>Router</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Throughput (Mpps)</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

Throughput improves by $P$ in each round
Service Chain Based Scheduling

- **NF1**
  - Framework
  - B1
- **NF2**
  - Framework
  - B2
- **…**
- **NFn**
  - Framework
  - Bn

**Switch**
- **Scheduler**
  - 1. Batch size
  - 2. #Thread blk
- **Manager**
  - Common Context
  - NF1:5
  - NF2:4
  - NF3:7

**NIC**

- **VMs**
  - Hypervisor
  - Streaming Multiprocessor
IsoPointer for GPU Memory Isolation

![Diagram of IsoPointer for GPU Memory Isolation]

- *P* → Memory Region
  - Base Address (B)
  - Memory Size (S)

Pointer access checking:
B ≤ P < B+S

IsoPointer: guarantee GPU memory isolation
NF Development

- Repetitive development efforts
  - CPU-GPU pipeline
  - Manage CPU threads
  - Communicate with Manager
  - Packet I/O with Switch
  - … …

- Framework handles all common operations
NF Development

- Repetitive development efforts
  - CPU-GPU pipeline
  - Manage CPU threads
  - Communicate with Manager
  - Packet I/O with Switch
  - ...

- Framework handles all common operations
- Abstraction to simplify NF development
NF Development

NF Spec.
Abstraction
Framework

NF1

Pre-Processing

• pre_pkt_handler

called for each pkt

GPU Processing

• memcpy_htod
• set_kernel_args
• memcpy dtoh

called for each kernel

Post-Processing

• post_pkt_handler

called for each pkt

Implementation = Significantly reduces development efforts + GPU Kernel

CPU code (Router)

Significantly reduces development efforts

Core functions
Evaluation

- **Hardware**
  - **CPU**: Intel Xeon E5-2650 v4
  - **GPU**: NVIDIA GTX TITAN X
  - **NIC**: Intel XL710 40Gbps

- **Software**
  - **Virtualization**: Docker 17.03.0-ce
  - **NIC Driver & Library**: DPDK 17.02.1
  - **OS**: CentOS 7.3.1611, Linux kernel 3.8.0-30

- **Service Chains**
  - **2 NFs**: IPsec → NIDS
  - **3 NFs**: Firewall → IPsec → NIDS
  - **4 NFs**: Firewall → IPsec → NIDS → Router
Throughput

- Comparison with **Temporal GPU Sharing**

<table>
<thead>
<tr>
<th>Packet Size (Byte)</th>
<th>Temporal Share</th>
<th>G-NET</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>128</td>
<td>0.75</td>
<td>1.25</td>
</tr>
<tr>
<td>256</td>
<td>1</td>
<td>1.5</td>
</tr>
<tr>
<td>512</td>
<td>1.25</td>
<td>1.75</td>
</tr>
<tr>
<td>1024</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>1518</td>
<td>1.75</td>
<td>2.25</td>
</tr>
</tbody>
</table>

(a) IPSec+NIDS  
up to 23.8%

(b) Firewall+IPSec+NIDS  
up to 25.9%

(c) IPSec+NIDS+Router  
up to 21.5%

(d) Firewall+IPSec+NIDS+Router  
up to 70.8%

More Resource Competition with four NFs
Scheduling

• Service chain scheduling scheme comparison
  • **G-NET**: optimize the performance of the service chain
  • **FairShare**: Evenly partition compute resources among NFs
  • **UncoShare**: Each NF tries to maximize its resource allocation

![Graph showing throughput improvement with packet size](image)

**Firewall + IPSec + NIDS + Router**

- **FairShare**
- **UncoShare**
- **G-NET**

**10.7% average improvement over FairShare**
**80.5% average improvement over UncoShare**
Latency

(a) 50th percentile latency

(b) 95th percentile latency

Normalized Latency

50th: ~20%

95th: 25% - 44%
Conclusion

• **G-NET:**
  • An NFV system that *efficiently* utilizes GPUs with *spatial GPU sharing*
  • **Service chain** based scheduling and resource allocation scheme
  • **Memory isolation** with IsoPointer
  • **Abstraction** that simplifies building GPU-accelerated NFs

• Experimental Results (Compare with temporal GPU sharing)
  • Enhances throughput by up to **70.8%**
  • Reduces latency by up to **44.3%**

**G-NET**
High-efficient platform for building GPU-based NFV systems