NFVnice: Dynamic Backpressure and Scheduling for NFV Service Chains

Sameer G Kulkarni\textsuperscript{1}, Wei Zhang\textsuperscript{2}, Jinho Hwang\textsuperscript{3}, Shriram Rajagopalan\textsuperscript{3}, K.K. Ramakrishnan\textsuperscript{4}, Timothy Wood\textsuperscript{2}, Mayutan Arumaithurai\textsuperscript{1} & Xiaoming Fu\textsuperscript{1}

\textsuperscript{1}University of Göttingen \quad \textsuperscript{2}George Washington University
\textsuperscript{3}IBM T J Watson Research Center \quad \textsuperscript{4}University of California, Riverside.
Growing NFV Popularity..

- Diverse and Large # of middleboxes, on par with switches and routers in ISP/DC Networks [APLOMB, SIGCOMM’12]
- NFs are fast replacing the traditional middleboxes in ISP/Telco/DC networks.

*source: ETSI NFV June’13
Growing NFV Popularity..

- Diverse and Large # of middleboxes, on par with switches and routers in ISP/DC Networks [APLOMB, SIGCOMM’12]
- NFs are fast replacing the traditional middleboxes in ISP/Telco/DC networks.

Typical usage of NFs include Function chaining.

*source: ETSI NFV June’13
Growing NFV Popularity..

- Diverse and Large # of middleboxes, on par with switches and routers in ISP/DC Networks [APLOMB, SIGCOMM’12]
- NFs are fast replacing the traditional middleboxes in ISP/Telco/DC networks.

Typical usage of NFs include Function chaining.

Performance (Resource Utilization) and Scalability are the key for NFV deployment!

*source: ETSI NFV June’13
How to address performance and scalability for NFV platform?
How to address performance and scalability for NFV platform?

- Consolidation approaches
  - E2 [SOSP ‘15], NetBricks [OSDI’16]:
  - Consolidate NFs of a chain on single core.
How to address performance and scalability for NFV platform?

• Consolidation approaches
  – E2 [SOSP ‘15], NetBricks [OSDI’16]:
  – Consolidate NFs of a chain on single core.

• But, lot of different NFs and diverse NF chains >>> compute cores! Performance, Scalability Challenges Exist!!
How to address performance and scalability for NFV platform?

• Consolidation approaches
  – E2 [SOSP ‘15], NetBricks [OSDI’16]:
    – Consolidate NFs of a chain on single core.

• But, lot of different NFs and diverse NF chains >>> compute cores! **Performance, Scalability Challenges Exist!!**

• Multiplexing approach:
  – Flurries [CoNext ‘16], ClickOS [NSDI’14]
  – Multiplex NFs on same core.

**Performance?, Scalability**
How to address performance and scalability for NFV platform?

- **Consolidation approaches**
  - E2 [SOSP ‘15], NetBricks [OSDI‘16]: Consolidate NFs of a chain on single core.

- **But, lot of different NFs and diverse NF chains >>> compute cores!  ** Performance, Scalability Challenges Exist!!

- **Multiplexing approach:**
  - Flurries [CoNext ‘16], ClickOS [NSDI‘14]
  - Multiplex NFs on same core.

Performance?, Scalability
How to address performance and scalability for NFV platform?

- Consolidation approaches
  - E2 [SOSP ‘15], NetBricks [OSDI’16]:
    - Consolidate NFs of a chain on single core.
  - But, lot of different NFs and diverse NF chains >>> compute cores! **Performance, Scalability Challenges Exist!!**

- Multiplexing approach:
  - Flurries [CoNext ‘16], ClickOS [NSDI’14]
    - Multiplex NFs on same core.

**Performance?, Scalability**

Then, how to Schedule the NFs to optimize the performance?
How to address performance and scalability for NFV platform?

- Consolidation approaches
  - E2 [SOSP ‘15], NetBricks [OSDI ‘16]:
    - Consolidate NFs of a chain on single core.

- But, lot of different NFs and diverse NF chains >>> compute cores!  **Performance, Scalability Challenges Exist!!**

- Multiplexing approach:
  - Flurries [CoNext ‘16], ClickOS [NSDI ‘14]
  - Multiplex NFs on same core.

**Performance? Scalability**

Then, how to Schedule the NFs to optimize the performance?
Use Existing Linux Schedulers?

• Vanilla Linux schedulers:
Use Existing Linux Schedulers?

• Vanilla Linux schedulers:
  
  Completely Fair Scheduler
  • Normal or Default
  • Batch

  - Virtual run time
  - Nanosecond granularity

Do existing schedulers perform well?
Use Existing Linux Schedulers?

• Vanilla Linux schedulers:
  - Completely Fair Scheduler
    • Normal or Default
    • Batch
  - Virtual run time
  - Nanosecond granularity
  - Real Time Scheduler
    • Round Robin
    • FIFO
  - Time slice
  - Millisecond granularity

Do existing schedulers perform well?
OS Scheduler Characterization (Load)

3 Homogeneous NFs running on a same core with offered load 2:2:1.
OS Scheduler Characterization (Load)

3 Homogeneous NFs running on a same core with offered load 2:2:1.

Ideal Allocation (NF1, NF2)

Ideal Allocation (NF3)
OS Scheduler Characterization (Load)

3 Homogeneous NFs running on a same core with offered load 2:2:1.

Ideal Allocation (NF1, NF2)

Ideal Allocation (NF3)
OS Scheduler Characterization (Load)

3 Homogeneous NFs running on a same core with offered load 2:2:1.

<table>
<thead>
<tr>
<th>Flow</th>
<th>NF1 (X)</th>
<th>NF2 (X)</th>
<th>NF3 (X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow 1</td>
<td>6Mpps</td>
<td>6Mpps</td>
<td>3Mpps</td>
</tr>
<tr>
<td>Flow 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flow 3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ideal Allocation (NF1, NF2)

Ideal Allocation (NF3)

Ideal Throughput (F1, F2)

Ideal Throughput (F3)
OS Scheduler Characterization (Load)

3 Homogeneous NFs running on a same core with offered load 2:2:1.

Schedulers fail to account NF load!

Ideal Allocation (NF1, NF2)

Ideal Allocation (NF3)

Ideal Throughput (F1, F2)

Ideal Throughput (F3)
OS Scheduler Characterization (Cost)

3 Heterogeneous NFs (per packet processing cost 10:5:1) with equal load

~5Mpps ~5Mpps ~5Mpps
Flow 1 Flow 2 Flow 3

NF1 (10X) NF2 (5X) NF3 (X)

Core

Flow 1 Flow 2 Flow 3

~5Mpps ~5Mpps ~5Mpps

Flow 1 Flow 2 Flow 3

 NF1 | NF2 | NF3

| Normal | Batch | RR |

CPU Utilization (%)
OS Scheduler Characterization (Cost)

3 Heterogeneous NFs (per packet processing cost 10:5:1) with equal load.

- NF1 (10X)
- NF2 (5X)
- NF3 (X)

Flow 1: ~5Mpps
Flow 2: ~5Mpps
Flow 3: ~5Mpps

Ideal Allocation (NF1)
Ideal Allocation (NF2)
Ideal Allocation (NF3)
OS Scheduler Characterization (Cost)

3 Heterogeneous NFs (per packet processing cost 10:5:1) with equal load
OS Scheduler Characterization (Cost)

3 Heterogeneous NFs (per packet processing cost 10:5:1) with equal load

Ideal Allocation (NF1)
Ideal Allocation (NF2)
Ideal Allocation (NF3)

Ideal Throughput (F1, F2, F3)
OS Scheduler Characterization (Cost)

3 Heterogeneous NFs (per packet processing cost 10:5:1) with equal load

Schedulers fail to account NF Processing cost!

Ideal Throughput (F1, F2, F3)

Ideal Allocation (NF1)

Ideal Allocation (NF2)

Ideal Allocation (NF3)
OS Scheduler Characterization (Chain)

3 NF chain (all NFs running on same core):
OS Scheduler Characterization (Chain)

3 NF chain (all NFs running on same core):

Flow 1

10 Mpps

NF1 (X) → NF2 (2X) → NF3 (4X)

Core

![Bar chart showing throughput comparison for different scheduling methods.]

Throughput in Mpps

NORMAL | BATCH | RR

2.5 | 2.0 | 2.5
OS Scheduler Characterization (Chain)

3 NF chain (all NFs running on same core):

[Diagram showing 3 NFs (NF1, NF2, NF3) with throughput values: 7.1, 3.58, 3.52, 2.02]
OS Scheduler Characterization (Chain)

3 NF chain (all NFs running on same core):

<table>
<thead>
<tr>
<th>Core</th>
<th>NF1 (X)</th>
<th>NF2 (2X)</th>
<th>NF3 (4X)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7.1</td>
<td>3.58</td>
<td>2.02</td>
</tr>
</tbody>
</table>

10 Mpps
Flow 1

Graphs showing:
- Throughput in Mpps
- Ctx sw/s

Table:

<table>
<thead>
<tr>
<th>Ctx sw/s</th>
<th>NORMAL</th>
<th>BATCH</th>
<th>RR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>20K/s</td>
<td>2K/s</td>
<td>1K/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU %</th>
<th>NORMAL</th>
<th>BATCH</th>
<th>RR</th>
</tr>
</thead>
<tbody>
<tr>
<td>NF1</td>
<td>34%</td>
<td>15%</td>
<td>9%</td>
</tr>
<tr>
<td>NF2</td>
<td>34%</td>
<td>42%</td>
<td>37%</td>
</tr>
<tr>
<td>NF3</td>
<td>33%</td>
<td>43%</td>
<td>54%</td>
</tr>
</tbody>
</table>

Images and additional data points illustrate the performance metrics for each configuration.
OS Scheduler Characterization (Chain)

3 NF chain (all NFs running on same core):

Too many/too little context switches result in **overhead** and **in-appropriate** allocation of CPU.
OS Scheduler Characterization (Chain)

3 NF chain (all NFs running on same core):

Too many/too little context switches result in **overhead** and **in-appropriate** allocation of CPU.

Vanilla Linux schedulers result in sub-optimal resource utilization.

<table>
<thead>
<tr>
<th>Ctx sw/s</th>
<th>NORMAL</th>
<th>BATCH</th>
<th>RR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>20K/s</td>
<td>2K/s</td>
<td>1K/s</td>
</tr>
<tr>
<td>CPU %</td>
<td>NORMAL</td>
<td>BATCH</td>
<td>RR</td>
</tr>
<tr>
<td>NF2</td>
<td>34%</td>
<td>42%</td>
<td>37%</td>
</tr>
<tr>
<td>NF3</td>
<td>33%</td>
<td>43%</td>
<td>54%</td>
</tr>
</tbody>
</table>
OS Scheduler Characterization (Chain)

3 NF chain (all NFs running on same core):

Vanilla Linux schedulers result in sub-optimal resource utilization.

Too many/too little context switches result in **overhead** and **in-appropriate** allocation of CPU.

Need the schedulers to be Load, NF characteristic, & chain aware!
NFVnice

A user space control framework for scheduling NFV chains.
NFVnice

A user space control framework for scheduling NFV chains.

• NFVnice in a nutshell:
  – Complements the existing kernel task schedulers.
    • Integrates “Rate proportional scheduling” from Hardware schedulers.
    • Integrates “Cost Proportional scheduling” from software schedulers.
  – Built on OpenNetVM[HMBox’16, NSDI’14]: A DPDK based NFV platform.
    • Enables deployment of containerized (Docker) or process based NFs.
  – Improves NF Throughput, Fairness and CPU Utilization through:
    • Proportional and Fair share of CPU to NFs: Tuning Scheduler.
    • Avoid wasted work and isolate bottlenecks: Backpressure.
    • Efficient I/O management framework for NFs.
NFVnice: Building Blocks

Work-conserving and proportional scheduling (within each core)
NFVnice: Building Blocks

`cgroups`

Work-conserving and proportional scheduling (*within each core*)

Cgroups: (control groups) is a Linux kernel feature that limits, accounts for and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes.
Rate-Cost Proportional Fairness

• What is Rate-Cost Proportional Fairness?
  – Determines the NFs CPU share by accounting the:
    • NF Load (Avg. packet arrival rate, instantaneous Queue length)
    • NF Priority and per-packet computation cost (Median)

• Why?
  – Efficient and fair allocation of CPU to the contending NFs.
  – Provides upper bound on the wait/Idle time for each NF.
  – Flexible & Extensible approach to adapt any QOS policy.
Rate-Cost Proportional Fairness

Initialization

```
mkdir /cgroupfs/NF(i)
```

`cgroups`
Rate-Cost Proportional Fairness

**Initialization**

```
mkdir /cgroupfs/NF(i)
```

**Weight Computation**

\[
load(i) = \lambda_i \times S_i
\]

\[
Total\ Load(m) = \sum_{i=0}^{n} load(i)
\]

\[
NFShare(i) = Priority_i \times \frac{load(i)}{Total\ Load(m)}
\]
Rate-Cost Proportional Fairness

**Initialization**

```
mkdir /cgroupfs/NF(i)
```

**Weight Computation**

\[
load(i) = \lambda_i \times S_i \\
\text{Total Load}(m) = \sum_{i=0}^{n} load(i) \\
NFShare(i) = \text{Priority}_i \times \frac{load(i)}{\text{Total Load}(m)}
\]

**Update Cgroup Weight**

```
Write “NFShare(i)” to /cgroupfs/NF(i)/cpu.shares
```
Rate-Cost Proportional Fairness

Initialization

```
mkdir /cgroupfs/NF(i)
```

Weight Computation

\[
\text{load}(i) = \lambda_i \times S_i \\
\text{Total Load}(m) = \sum_{i=0}^{n} \text{load}(i) \\
\text{NFShare}(i) = \text{Priority}_i \times \frac{\text{load}(i)}{\text{Total Load}(m)}
\]

Every 10 ms

Update Cgroup Weight

```
Write “NFShare(i)” to /cgroupfs/NF(i)/cpu.shares
```
NFVnice: Building Blocks

- cgroups: Work-conserving and proportional scheduling *(within each core)*
- Chain-aware scheduling; Avoid wasted work *(within and across cores)*

NFVnice
Backpressure in NF chains

- Selective per chain backpressure marking.

- Only Flow “A” going through bottleneck NF (NF3) is back pressured and throttled at the upstream source NF1.
- while Flow “B” is not affected.
Scenario: No Backpressure
Scenario: No Backpressure

NF1 and NF2 contend for CPU, and steal the CPU cycles from NF3!
Scenario: No Backpressure

NF1 and NF2 contend for CPU, and steal the CPU cycles from NF3!

Lots of Wasted Work!!!
Traditional Backpressure

Stop! Stop!
Traditional Backpressure

Incurs feedback delay and offers no flow isolation.
Lots of Wasted Work
Incur Delay, No Isolation
Reacts Instantaneously
Packet Throttling per Chain
Lots of Wasted Work

Incur Delay, No Isolation

Reacts Instantaneously

Packet Throttling per Chain
Lots of Wasted Work
Incur Delay, No Isolation
Reacts Instantaneously
Packet Throttling per Chain
NFVnice: Building Blocks

- **cgroups**: Work-conserving and proportional scheduling \((\text{within each core})\)
- **Back pressure**: Chain-aware scheduling; Avoid wasted work \((\text{within and across cores})\)
- **ECN**: End-2-End Bottleneck/congestion control \((\text{across nodes})\)
ECN Marking

- ECN-aware NF Manager:
  - Per-NF ECN marking based on Active Queue Management (AQM) policies.
ECN Marking

- ECN-aware NF Manager:
  - Per-NF ECN marking based on Active Queue Management (AQM) policies.

Address NF bottleneck across chain of NFs in distinct nodes.
NFVnice: Building Blocks

- cgroups: Work-conserving and proportional scheduling \((\text{within each core})\)
- Chain-aware scheduling; Avoid wasted work \((\text{within and across cores})\)
- ECN: End-2-End Bottleneck/congestion control \((\text{across nodes})\)
- I/O Mgt.: Efficient Disk I/O Mgmt. Library
OpenNetVM [HMBox’16, NSDI’14] makes use of DPDK to enable fast packet processing.

NFVnice extends the **Data** and **Control** plane functionalities to facilitate efficient multiplexing and scheduling of NFs on same core.
- Resource monitoring and control functions.
• Resource monitoring and control functions.
  – *Wakeup Thread*
    • Wakeup notification to the NFs
    • Timer Management.
Resource monitoring and control functions.

- **Wakeup Thread**
  - Wakeup notification to the NFs
  - Timer Management.
• Resource monitoring and control functions.
  – **NF threads (libnf)**
    • Voluntary yield decisions.
    • Estimate per-packet processing cost.
- Resource monitoring and control functions.
  - **Monitor Thread**
    - periodically (1ms) monitors NF load.
    - computes the cpu share for each core.
    - Tracks EWMA of NFs Rx queue length and mark ECN.
• Testbed:
  – Hardware: 3 Intel Xeon(R) CPU E5-2697, 28 cores @2.6Ghz servers, with dual port 10Gbps DPDK compatible NICs.
  – NFVnice: built on top of OpenNetVM.

• Traffic:
  – Pktgen and Moongen: Line rate traffic (64 byte packets).
  – Iperf: TCP flows.

• Schemes compared:
  – Native Linux Schedulers with and w/o NFVnice.
  – Different NFs (varying computation costs) and chain configurations.
Performance: Impact of cgroup weights and Backpressure

Simple Three NF Chain

NF1 120 Cpp
Core-1

NF2 270 Cpp
Core-1

NF3 550 Cpp
Core-1

Cycles per packet

![Graph showing throughput in Mpps for different configurations: NORMAL, BATCH, RR(1ms), RR(100ms) vs Default.]}
Performance: Impact of cgroup weights and Backpressure

Simple Three NF Chain

NF1 120 Cpp
Core-1

NF2 270 Cpp
Core-1

NF3 550 Cpp
Core-1

Cycles per packet

Graph showing throughput in Mpps for different scenarios:
- Default
- Only CGroup

Scenarios:
- NORMAL
- BATCH
- RR(1ms)
- RR(100ms)
Performance: Impact of cgroup weights and Backpressure

Simple Three NF Chain

NF1 120 Cpp
Core-1

NF2 270 Cpp
Core-1

NF3 550 Cpp
Core-1

Cycles per packet

[Graph showing throughput in Mbps for different workloads and configurations]
Performance: Impact of cgroup weights and Backpressure

Simple Three NF Chain

- NF1 120 Cpp
- NF2 270 Cpp
- NF3 550 Cpp

Cycles per packet

Graph showing throughput in Mpps for different configurations:
- Default
- Only BKPR
- Only CGroup
- NFVnice

Throughput in Mpps:
- Normal
- Batch
- RR(1ms)
- RR(100ms)
Performance: Impact of cgroup weights and Backpressure

Simple Three NF Chain

NF1 120 Cpp
Core-1

NF2 270 Cpp
Core-1

NF3 550 Cpp
Core-1

Cycles per packet

Significant Reduction in Wasted Work!
Performance: Impact of cgroup weights and Backpressure

Simple Three NF Chain

NF1 120 Cpp
Core-1

NF2 270 Cpp
Core-1

NF3 550 Cpp
Core-1

Cycles per packet

Significant Reduction in Wasted Work!

CPU Allocation $\alpha$ Computation Cost

<table>
<thead>
<tr>
<th></th>
<th>CFS Normal Scheduler</th>
<th>Wasted Work (Packet Drops/sec)</th>
<th>NF Runtime (ms) (measured over 2s interval)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Default</td>
<td>Only BKPR</td>
<td>Only CG</td>
</tr>
<tr>
<td>NF1</td>
<td>Default</td>
<td>3.58M</td>
<td>58%</td>
</tr>
<tr>
<td></td>
<td>NFVnice</td>
<td>11.2K</td>
<td>&lt;0.3%</td>
</tr>
<tr>
<td>NF2</td>
<td>2.02M</td>
<td>12.3K</td>
<td>&lt;0.4%</td>
</tr>
<tr>
<td>NF3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Performance: Impact of cgroup weights and Backpressure

Simple Three NF Chain

NF1 120 Cpp
Core-1

NF2 270 Cpp
Core-1

NF3 550 Cpp
Core-1

Cycles per packet

Significant Reduction in Wasted Work!

CPU Allocation $\alpha$ Computation Cost

NFVnice improves throughput for all kernel schedulers.
Efficient Resource (CPU) Utilization

Three NF Chain (NF per core)

<table>
<thead>
<tr>
<th>Default</th>
<th>Svc. Rate</th>
<th>Drop rate</th>
<th>CPU Util</th>
</tr>
</thead>
<tbody>
<tr>
<td>NF1 (550cycles)</td>
<td>5.95Mpps</td>
<td>4.76Mpps</td>
<td>100%</td>
</tr>
<tr>
<td>NF2 (2200cycles)</td>
<td>1.18Mpps</td>
<td>0.58Mpps</td>
<td>100%</td>
</tr>
<tr>
<td>NF3 (4500cycles)</td>
<td>0.6Mpps</td>
<td>-</td>
<td>100%</td>
</tr>
<tr>
<td>Aggregate</td>
<td>0.6Mpps</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
# Efficient Resource (CPU) Utilization

## Three NF Chain (NF per core)

<table>
<thead>
<tr>
<th>Core</th>
<th>NF1 550 Cpp</th>
<th>NF2 2200 Cpp</th>
<th>NF3 4500 Cpp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core-1</td>
<td>(550cycles)</td>
<td>(2200cycles)</td>
<td>(4500cycles)</td>
</tr>
</tbody>
</table>

### Default

<table>
<thead>
<tr>
<th></th>
<th>Svc. Rate</th>
<th>Drop rate</th>
<th>CPU Util</th>
</tr>
</thead>
<tbody>
<tr>
<td>NF1 (550cycles)</td>
<td>5.95Mpps</td>
<td>4.76Mpps (80%)</td>
<td>100%</td>
</tr>
<tr>
<td>NF2 (2200cycles)</td>
<td>1.18Mpps</td>
<td>0.58Mpps (49%)</td>
<td>100%</td>
</tr>
<tr>
<td>NF3 (4500cycles)</td>
<td>0.6Mpps</td>
<td>-</td>
<td>100%</td>
</tr>
<tr>
<td>Aggregate</td>
<td>0.6Mpps</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Lots of Wasted Work! Burning CPU!!
# Efficient Resource (CPU) Utilization

## Three NF Chain (NF per core)

<table>
<thead>
<tr>
<th></th>
<th>Default</th>
<th>NFVnice</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Svc. Rate</td>
<td>Drop rate</td>
</tr>
<tr>
<td><strong>NF1</strong> (550cycles)</td>
<td>5.95Mpps</td>
<td>4.76Mpps (80%)</td>
</tr>
<tr>
<td><strong>NF2</strong> (2200cycles)</td>
<td>1.18Mpps</td>
<td>0.58Mpps (49%)</td>
</tr>
<tr>
<td><strong>NF3</strong> (4500cycles)</td>
<td>0.6Mpps</td>
<td>-</td>
</tr>
<tr>
<td><strong>Aggregate</strong></td>
<td>0.6Mpps</td>
<td>-</td>
</tr>
</tbody>
</table>

*Lots of Wasted Work! Burning CPU!!*
# Efficient Resource (CPU) Utilization

## Three NF Chain (NF per core)

### Default

<table>
<thead>
<tr>
<th>Svc. Rate</th>
<th>Drop rate</th>
<th>CPU Util</th>
</tr>
</thead>
<tbody>
<tr>
<td>NF1 (550cycles)</td>
<td>5.95Mpps</td>
<td>4.76Mpps 80%</td>
</tr>
<tr>
<td>NF2 (2200cycles)</td>
<td>1.18Mpps</td>
<td>0.58Mpps 49%</td>
</tr>
<tr>
<td>NF3 (4500cycles)</td>
<td>0.6Mpps</td>
<td>-</td>
</tr>
<tr>
<td>Aggregate</td>
<td>0.6Mpps</td>
<td>-</td>
</tr>
</tbody>
</table>

### NFVnice

<table>
<thead>
<tr>
<th>Svc. Rate</th>
<th>Drop rate</th>
<th>CPU Util</th>
</tr>
</thead>
<tbody>
<tr>
<td>NF1 (550cycles)</td>
<td>0.82Mpps</td>
<td>0.15Mpps 11%</td>
</tr>
<tr>
<td>NF2 (2200cycles)</td>
<td>0.72Mpps</td>
<td>0.07Mpps 64%</td>
</tr>
<tr>
<td>NF3 (4500cycles)</td>
<td>0.6Mpps</td>
<td>-</td>
</tr>
<tr>
<td>Aggregate</td>
<td>0.6Mpps</td>
<td>-</td>
</tr>
</tbody>
</table>

- **Lots of Wasted Work! Burning CPU!!**
- **Avoid Wasted Work! Optimize CPU utilization.**
# Efficient Resource (CPU) Utilization

Three NF Chain (NF per core)

<table>
<thead>
<tr>
<th></th>
<th>Default</th>
<th>NFVnice</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Svc. Rate</td>
<td>Drop rate</td>
</tr>
<tr>
<td>NF1</td>
<td>5.95Mpps</td>
<td>4.76Mpps</td>
</tr>
<tr>
<td>(550cycles)</td>
<td></td>
<td>80%</td>
</tr>
<tr>
<td>NF2</td>
<td>1.18Mpps</td>
<td>0.58Mpps</td>
</tr>
<tr>
<td>(2200cycles)</td>
<td></td>
<td>49%</td>
</tr>
<tr>
<td>NF3</td>
<td>0.6Mpps</td>
<td>-</td>
</tr>
<tr>
<td>(4500cycles)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aggregate</td>
<td>0.6Mpps</td>
<td>-</td>
</tr>
</tbody>
</table>

Lots of Wasted Work! Burning CPU!!

Achieves Same Throughput

Avoid Wasted Work! Optimize CPU utilization.

Lots of Wasted Work! Burning CPU!!
## Efficient Resource (CPU) Utilization

### Three NF Chain (NF per core)

<table>
<thead>
<tr>
<th></th>
<th>Default</th>
<th>SFVnice</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Svc. Rate</td>
<td>Drop rate</td>
</tr>
<tr>
<td>NF1 (550 cycles)</td>
<td>5.95Mpps</td>
<td>4.76Mpps</td>
</tr>
<tr>
<td>NF2 (2200 cycles)</td>
<td>1.18Mpps</td>
<td>0.58Mpps</td>
</tr>
<tr>
<td>Aggregate</td>
<td>0.6Mpps</td>
<td>-</td>
</tr>
</tbody>
</table>

- **Lots of Wasted Work! Burning CPU!!**
- **Achieves Same Throughput**
- **Avoid Wasted Work! Optimize CPU utilization.**

**NFVnice avoids wasted work; provides better resource utilization**
Performance + Resource Utilization

<table>
<thead>
<tr>
<th>NFVnice-NF1</th>
<th>NFVnice-NF2</th>
<th>Default-NF3</th>
<th>Default-NF4</th>
<th>Default-NF1</th>
<th>NFVnice-NF3</th>
<th>NFVnice-NF2</th>
<th>NFVnice-NF1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chain 1</td>
<td>Chain 2</td>
<td>Chain 1</td>
<td>Chain 2</td>
<td>Chain 1</td>
<td>Chain 2</td>
<td>Chain 2</td>
<td>Chain 1</td>
</tr>
</tbody>
</table>

# Packets processed in Mpps
Performance + Resource Utilization

Inefficient CPU utilization by NF1
Performance + Resource Utilization

Inefficient CPU utilization by NF1
Performance + Resource Utilization

Inefficient CPU utilization by NF1

Judicious utilization of NFs CPU

<table>
<thead>
<tr>
<th>NF</th>
<th>Chain 1</th>
<th>Chain 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default-NF1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NFVnice-NF1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default-NF2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NFVnice-NF2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default-NF3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NFVnice-NF3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default-NF4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NFVnice-NF4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# Packets processed in Mpps
Performance + Resource Utilization

Inefficient CPU utilization by NF1

Judicious utilization of NFs CPU

 NFVnice-NF4
 Default-NF4
 NFVnice-NF3
 Default-NF3
 NFVnice-NF2
 Default-NF2
 NFVnice-NF1
 Default-NF1

# Packets processed in Mpps

<table>
<thead>
<tr>
<th></th>
<th>Chain 1</th>
<th>Chain 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default-NF1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>NFVnice-NF1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Default-NF2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>NFVnice-NF2</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Default-NF3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>NFVnice-NF3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Default-NF4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>NFVnice-NF4</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Aggr. Throughput in Mpps

CPU Util. %

Default
NFVnice
Performance + Resource Utilization

Inefficient CPU utilization by NF1

Judicious utilization of NFs CPU

~2x Throughput Gain with efficient CPU utilization

Chain 1

Chain 2

NF1 ~270 cpp
Core-1

NF2 ~120 cpp
Core-2

NF3 ~4500 cpp
Core-3

NF4 ~300 cpp
Core-4

# Packets processed in Mpps

0 1 2 3 4 5 6 7 8

Default-NF1

NFVnice-NF1

Default-NF2

NFVnice-NF2

Default-NF3

NFVnice-NF3

Default-NF4

NFVnice-NF4

CPU Util. %

0 10 20 30 40 50 60 70 80 90 100

Default

NFVnice

Aggr. Throughput in Mpps

0 1 2 3 4 5 6 7 8
Performance + Resource Utilization

Inefficient CPU utilization by NF1

Judicious utilization of NFs CPU

~2x Throughput Gain with efficient CPU utilization

Flows get right amount of bandwidth and NF resources
Robustness: Chain Diversity

Robust across schedulers

Three NF Chain

NF1 (Low) Core-1
NF2 (Med) Core-1
NF3 (High) Core-1

Low-Med-High

Throughput in Mpps

Default    NFVnice
NORMAL BATCH RR(1ms) RR(100ms)
Robustness: Chain Diversity

Robust across schedulers

Three NF Chain

NF1 (Low) Core-1
NF2 (Med) Core-1
NF3 (High) Core-1

Core-1

Throughput in Mpps

Low-Med-High

Default NFVnice

Normal Batch RR(1ms) RR(100ms)

Med-High-Low

Default NFVnice

Normal Batch RR(1ms) RR(100ms)
Robustness: Chain Diversity

Three NF Chain

- NF1 (Low)
- NF2 (Med)
- NF3 (High)

Core-1

Robust across schedulers

Robust across chain diversity

Bar charts showing throughput in Mbps for different scenarios and schedulers.
Robustness: Chain Diversity

Three NF Chain

- NF1 (Low)
- NF2 (Med)
- NF3 (High)

Core-1

Robust across schedulers

Robust across chain diversity

Graphs showing throughput in Mpps for different scheduling methods (NORMAL, BATCH, RR(1ms), RR(100ms)) and NF chain configurations (Low-Med-High, Med-High-Low, High-Med-Low, Low-High-Med, Med-Low-High, High-Low-Med).
Robustness: Chain Diversity

- **Robust across schedulers**
- **Three NF Chain**
  - NF1 (Low)
  - NF2 (Med)
  - NF3 (High)

Core-1 Core-1 Core-1

- Robust across chain diversity

Consistently improves throughput regardless of chain characteristics
NF Processing cost variation

*Variable per packet processing cost [120 to 550 cpp]
NF Processing cost variation

Simple Three NF Chain

*Variable per packet processing cost [120 to 550 cpp]

NFVnice is resilient to cost variations!
TCP and UDP Isolation
TCP and UDP Isolation

TCP affected by UDP flows! Wastage of NF1,NF2 bandwidth
TCP and UDP Isolation

TCP affected by UDP flows! Wastage of NF1,NF2 bandwidth

Effectively isolates UDP and TCP flows

TCP W/O NFVnice, UDP W/O NFVnice, TCP With NFVnice, UDP With NFVnice

Throughput in Mbps

Time in Seconds
Conclusion

• NFVnice enables performance and scale

• A user space framework complementing the OS CPU schedulers.
  – Brings the best of Hardware packet schedulers and CPU schedulers to the NFV platform.
    • Weight adjustments and Backpressure help get better NFV performance.
  – Improves Fairness and Throughput by being chain-aware.

• Our work will be open-sourced soon:
  – Watch out for the link: https://github.com/sdnfv/NFVnice
Thank you!

CleanSky ITN: A EU FP7 Marie Curie Initial Training Network
A Network for the Cloud Computing Eco-System