Software-Based Networks: Leveraging high-performance NFV platforms to meet future communication challenges

K.K. Ramakrishnan
University of California, Riverside
(kk@cs.ucr.edu)

Joint work with: Timothy Wood (GWU) , our students, collaborators

UNIVERSITY OF CALIFORNIA, RIVERSIDE

A Middlebox World


ad insertion


BRAS


WAN accelerator


carrier-grade NAT


transcoder


session border controller


IDS


load balancer


firewall

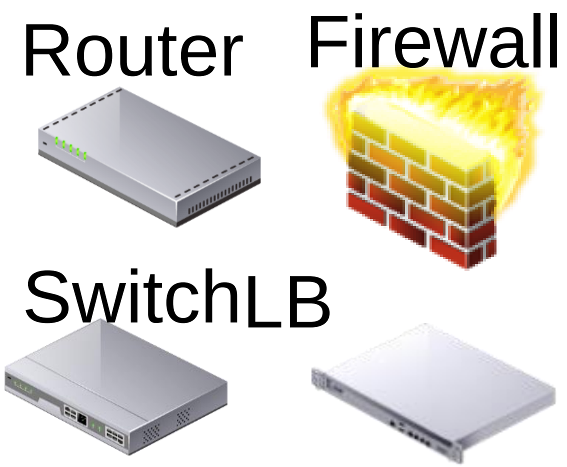
DDoS protection

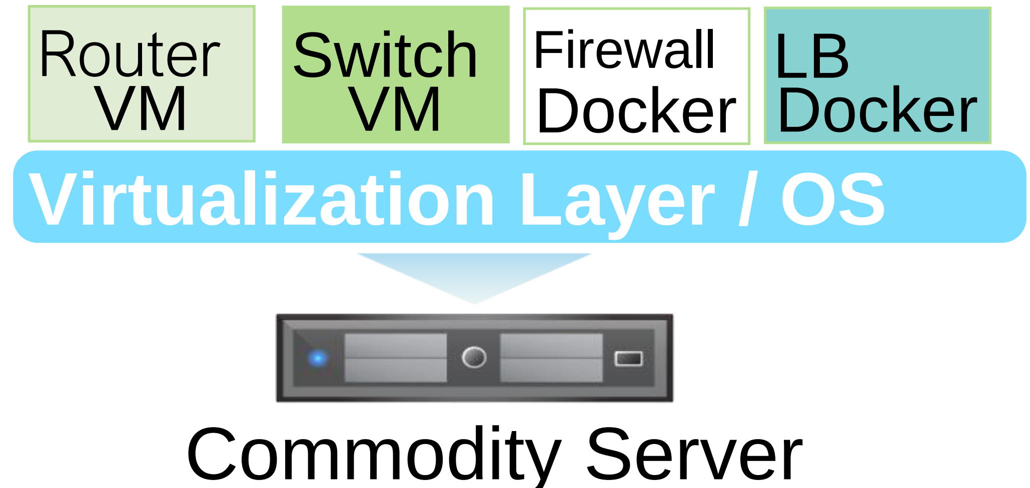
QoE monitor


DPI

Network Function Virtualization

- Run network functions in software
    - Driven by a confluence of factors: system capabilities; OS evolution
    - Requirements of new services + evolution of the network infrastructure



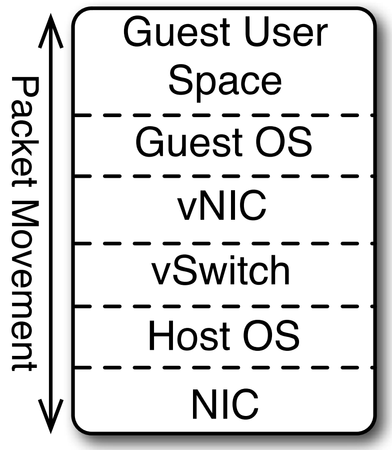


- More flexible than hardware
    - Easy to deploy NFs; quick instantiation
    - Easier to manage NFs

- Network Service Providers are migrating towards a software-based networking infrastructure: AT&T “55% of network functions have been virtualized”

Virtualization Overheads

- **Virtualization layer** provides (resource and performance) isolation among virtual machines
- **Isolation** involves many functions such as access permissions (security), ability to schedule and share etc.
- **Network overhead** (packet delivery) is one of the most critical concerns
- A generic virtualization architecture includes several critical boundaries – host OS, virtual NIC, guest OS, and guest user space–getting packet data there includes **memory copies**



Jinho Hwang, K.K. Ramakrishnan, and Timothy Wood, "NetVM: High Performance and Flexible Networking using Virtualization on Commodity Platforms," NSDI '14.

Contributions: OpenNetVM

1. **A virtualization-based high-speed packet delivery platform**

   - for flexible network service deployment that can meet the performance of customized hardware, especially when involving complex packet processing

2. **Network shared-memory framework**

   - that truly exploits the DPDK (data plane development kit) library to provide zero-copy delivery to VMs and between VMs (containers)

3. **A NF-Manager: provides switching and overall NF management**

   - dynamically adjust a flow's destination in a state-dependent and/or data-dependent manner

4. **High speed inter-NF communication**

   - enabling complex network services to be spread across multiple NFs

5. **Security domains**

   - that restrict access of packet data to only trusted NFs

Key Technical Challenges

- Achieving high performance:
  - Wire-speed throughput
  - Low Latency

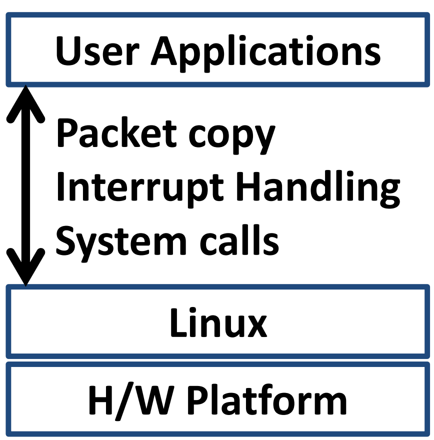
→ Function as a ‘bump in the wire’

- Flexibility to support software functionality from many sources/vendors
- Customization – can we have network functions customized for each packet flow?
  - Flurries (CoNext 2016), Microboxes (Sigcomm 2018)
- Scalability and effectively using resources: NFVnice (Sigcomm 2017)
  - Scheduling
  - Managing congestion
- Failure Resiliency
  - Reinforce (Conext 2018)

Achieving High Performance

Linux Packet Processing

- Traditional networking:
  - NIC uses DMA to copy data into kernel buffer
  - Interrupt when packets arrive
  - Copy packet data from kernel space to user space
  - Use system call to transmit packet from user space



Generic Packet Processing

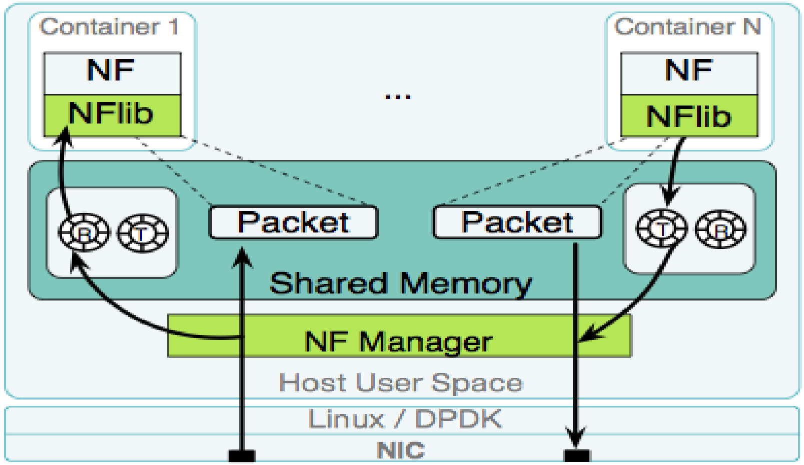
Can it handle being interrupted 14 million times per second?

Data Plane Development Kit (DPDK)

- High performance I/O library
- Poll mode driver reads packets from NIC
- Packets bypass the OS and are copied directly into user space memory

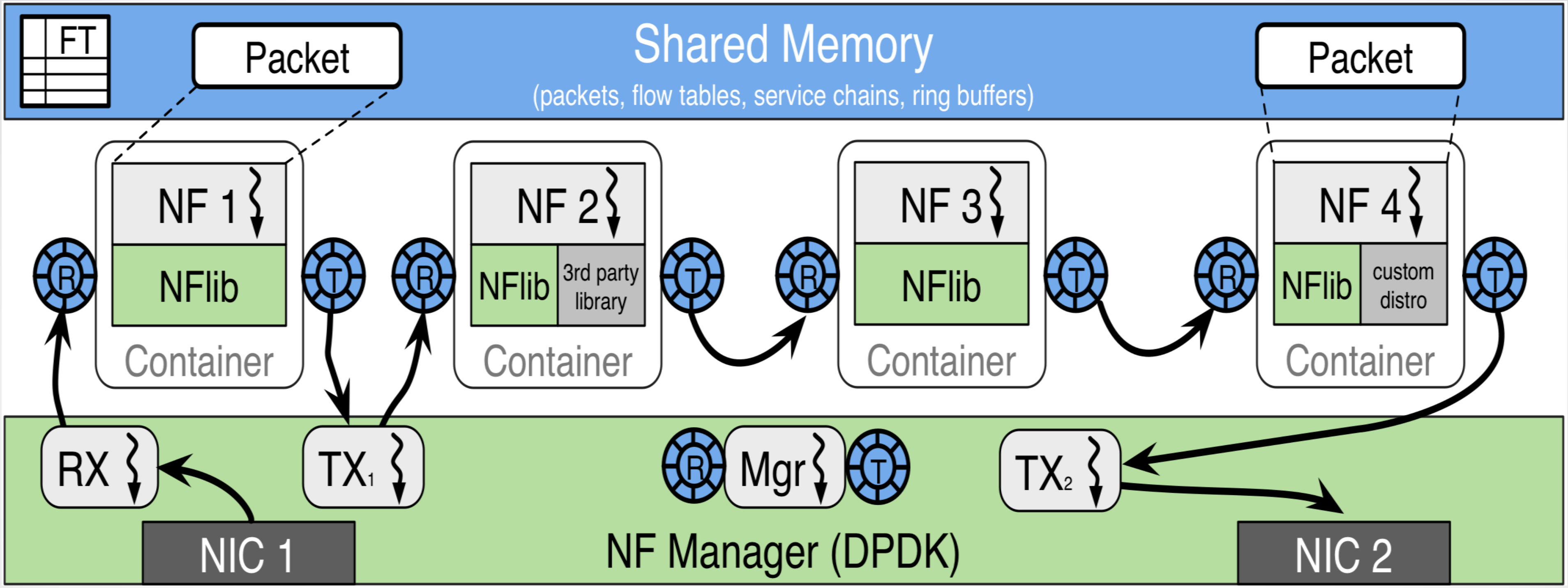
- Low level library... does not provide:
  - Support for multiple network functions
  - Interrupt-driven NFs
  - State management
  - SDN-based control
  - TCP/IP protocol stack

OpenNetVM Architecture



- **NF Manager** (with DPDK) runs in user space
- **NFs** run inside Docker containers
  - **NUMA-aware** processing
  - **Zero-copy** data transfer to and between NFs
  - **No Interrupts** using DPDK poll-mode driver
  - **Scalable** Multiple Rx and Tx threads in manager
    - Each NF has its own ring to receive/transmit a packet descriptor
  - NFs start in 0.5 second; throughput of 68 Gbps w/ 6 cores; base forwarding latency < 10 μsecs

OpenNetVM – System Architecture



- NF Manager uses DPDK library to allocate memory pools in huge pages for packets
- NFs map memory regions using same base virtual address as NF manager- descriptor access and shared memory

How to Eliminate / Hide Overheads?

Interrupt Context Switch Overhead

Kernel User Overhead

Core To Thread Scheduling Overhead

Polling
(dedicated core)

User Mode Driver
(DPDK)

Pthread Affinity
(pin threads to cores)

4K Paging Overhead

PCI Bridge I/O Overhead

Huge Pages

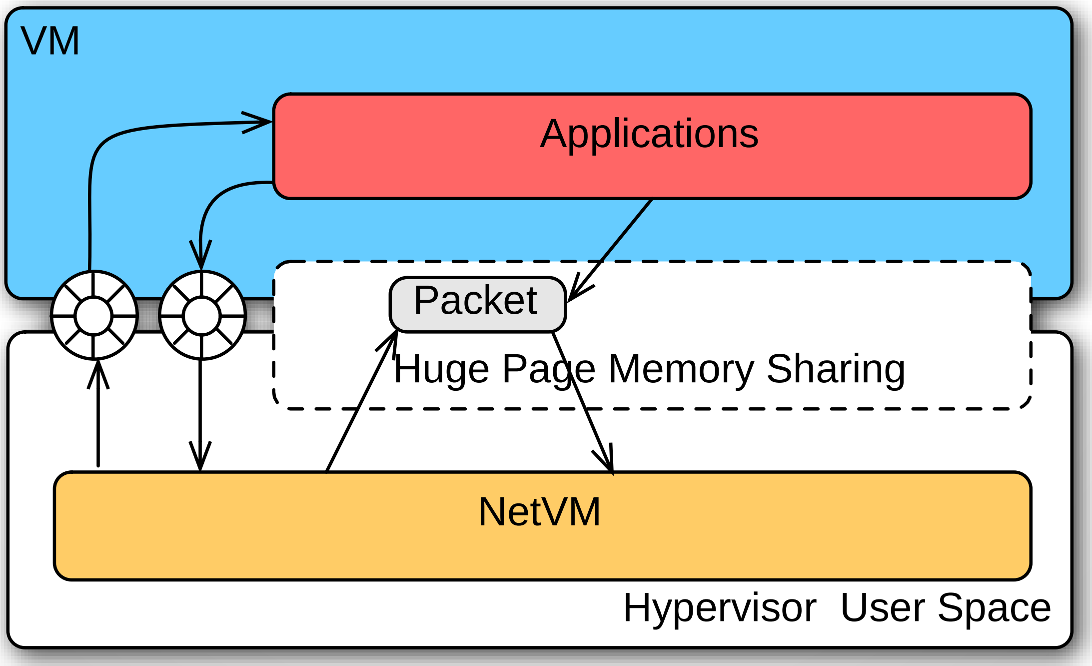
(1GB huge pages)

Lockless Inter-core Communication
(ring buffer based message queues)

High Throughput Bulk Mode I/O calls

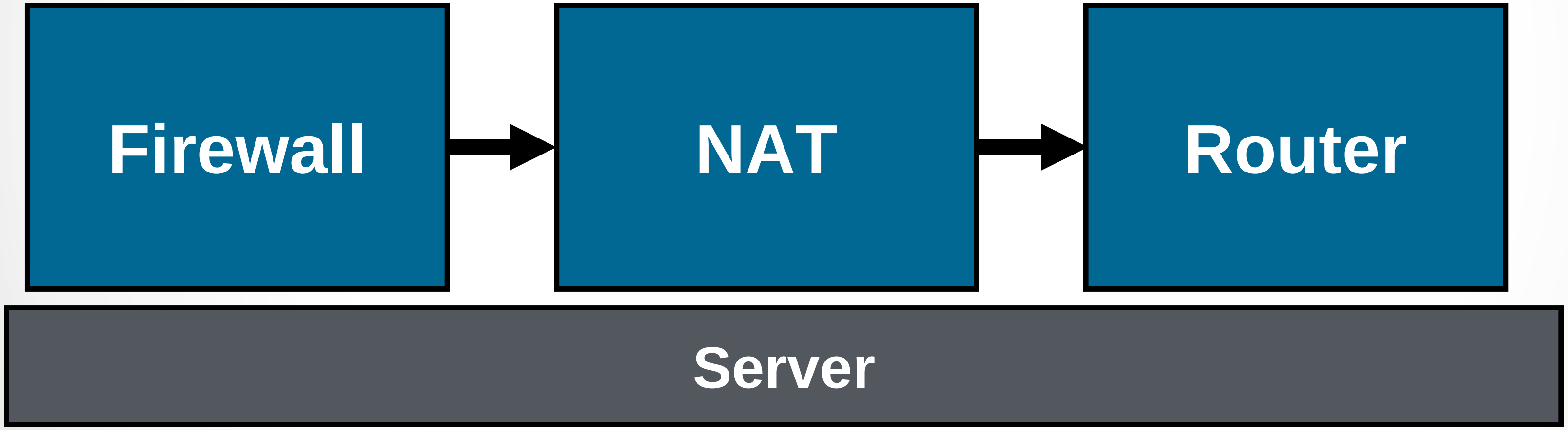
Zero-Copy Packet Delivery

- Packet directly DMA-ed into huge page memory by NIC (take advantage of DPDK)
- Applications in Container receive references (location) via the shared descriptor ring buffer
- Packet content can be modified by NF application



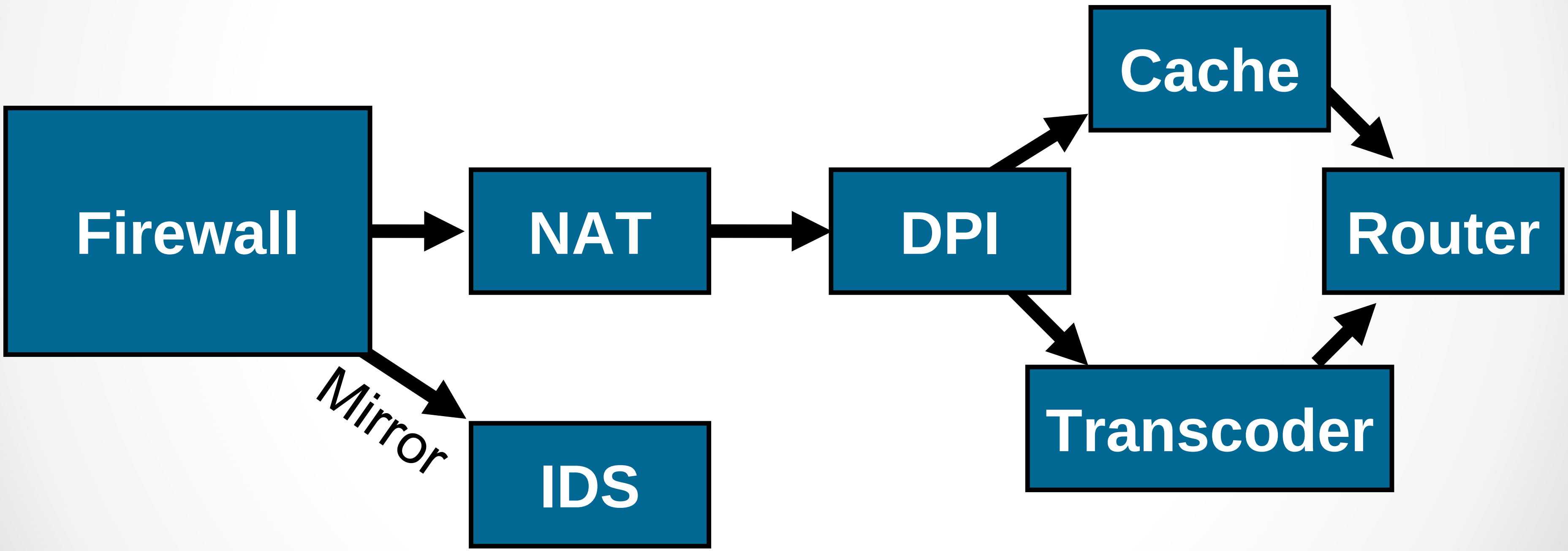
Service Chains

- Chain together functionality to build more complex services
  - Need to move packets through chain efficiently



Service Chains

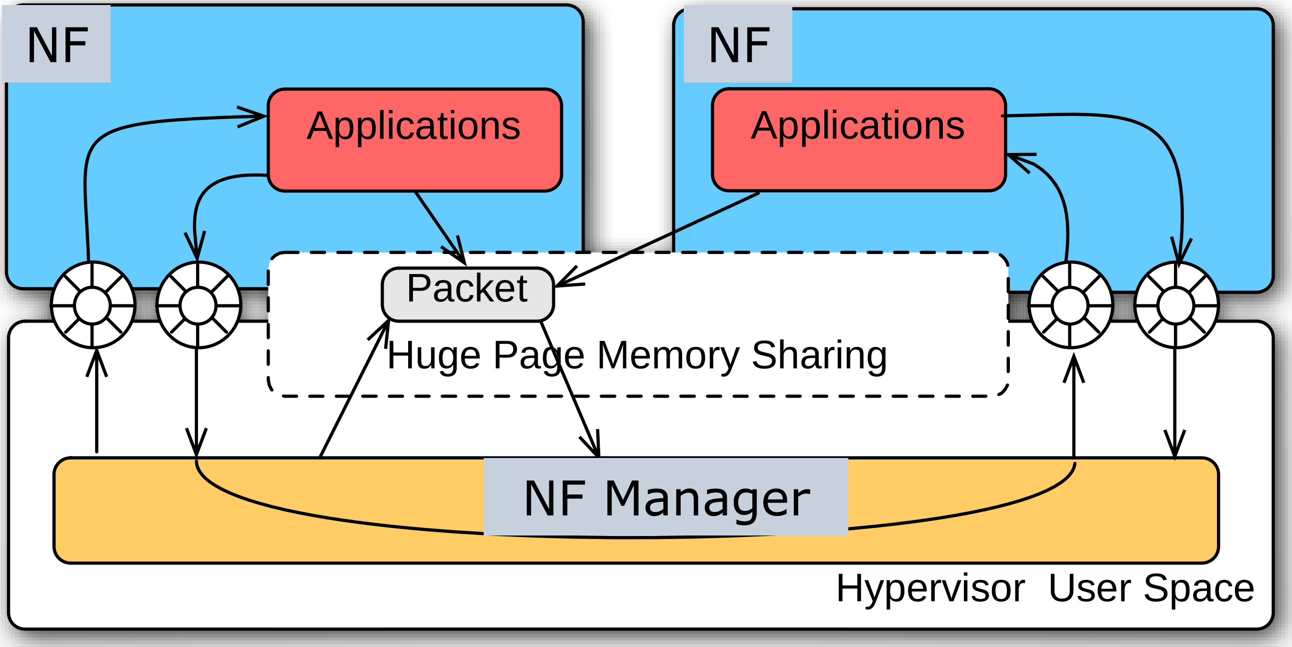
- Chain together functionality to build more complex services
  - Need to move packets through chain efficiently



- **Can be complex with multiple paths!**

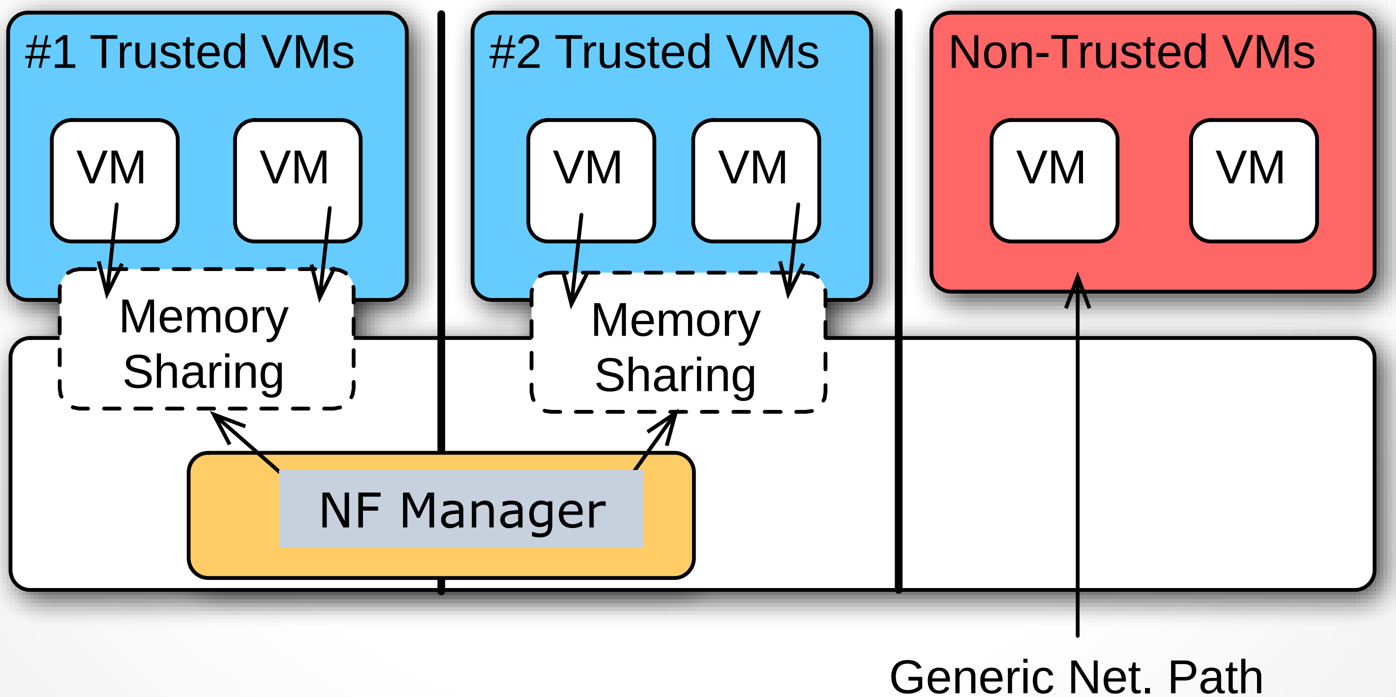
Chained Packet Delivery

- Packets in memory do not have to be copied
- Applications in containers pass packet references to others: NF→NF – through the descriptor ring
- Only one application can access a given packet at any time for writing – avoid locks
  - Allow multiple readers (e.g., packet monitoring)



Trusted and Untrusted Domains

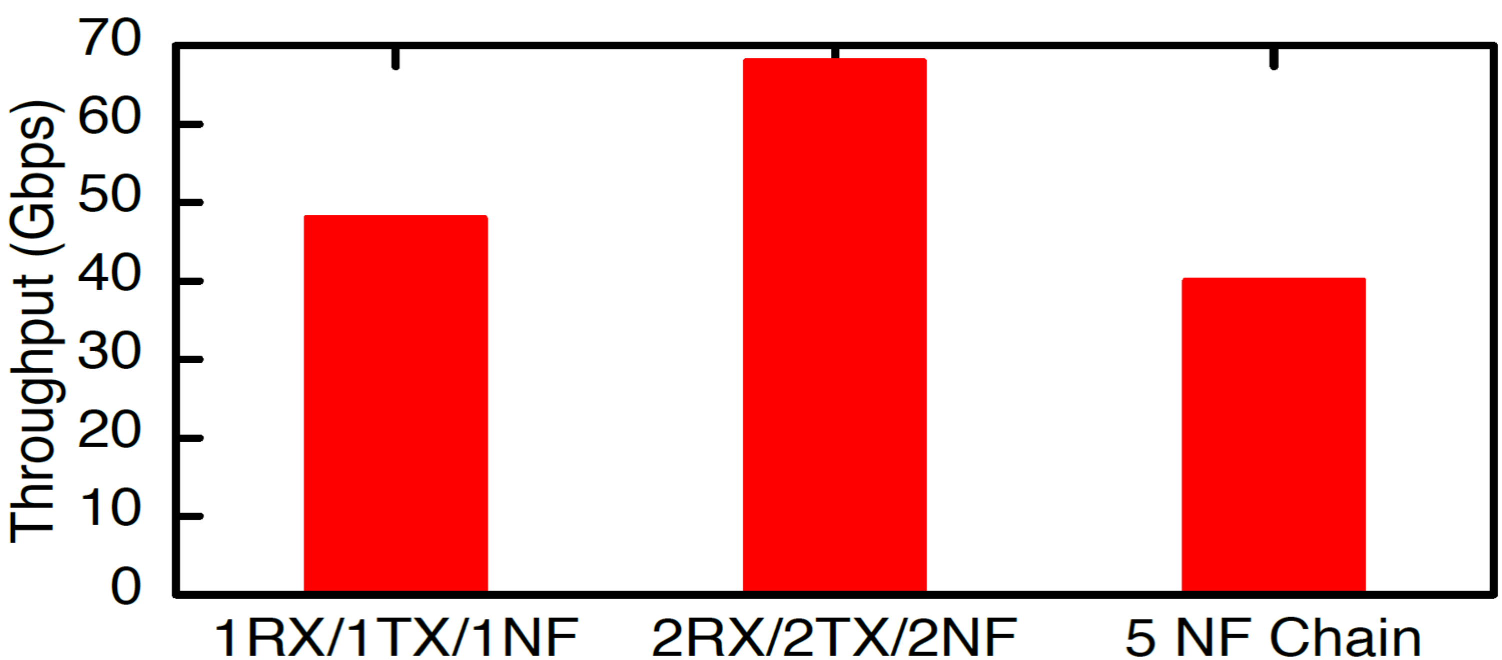
- Virtualization should provide security guarantees among VMs/containers
- OpenNetVM provides a security boundary between trusted and untrusted NFs
- Grouping of trusted NFs via huge page separation
- Untrusted NFs cannot see packets from OpenNetVM



UCR

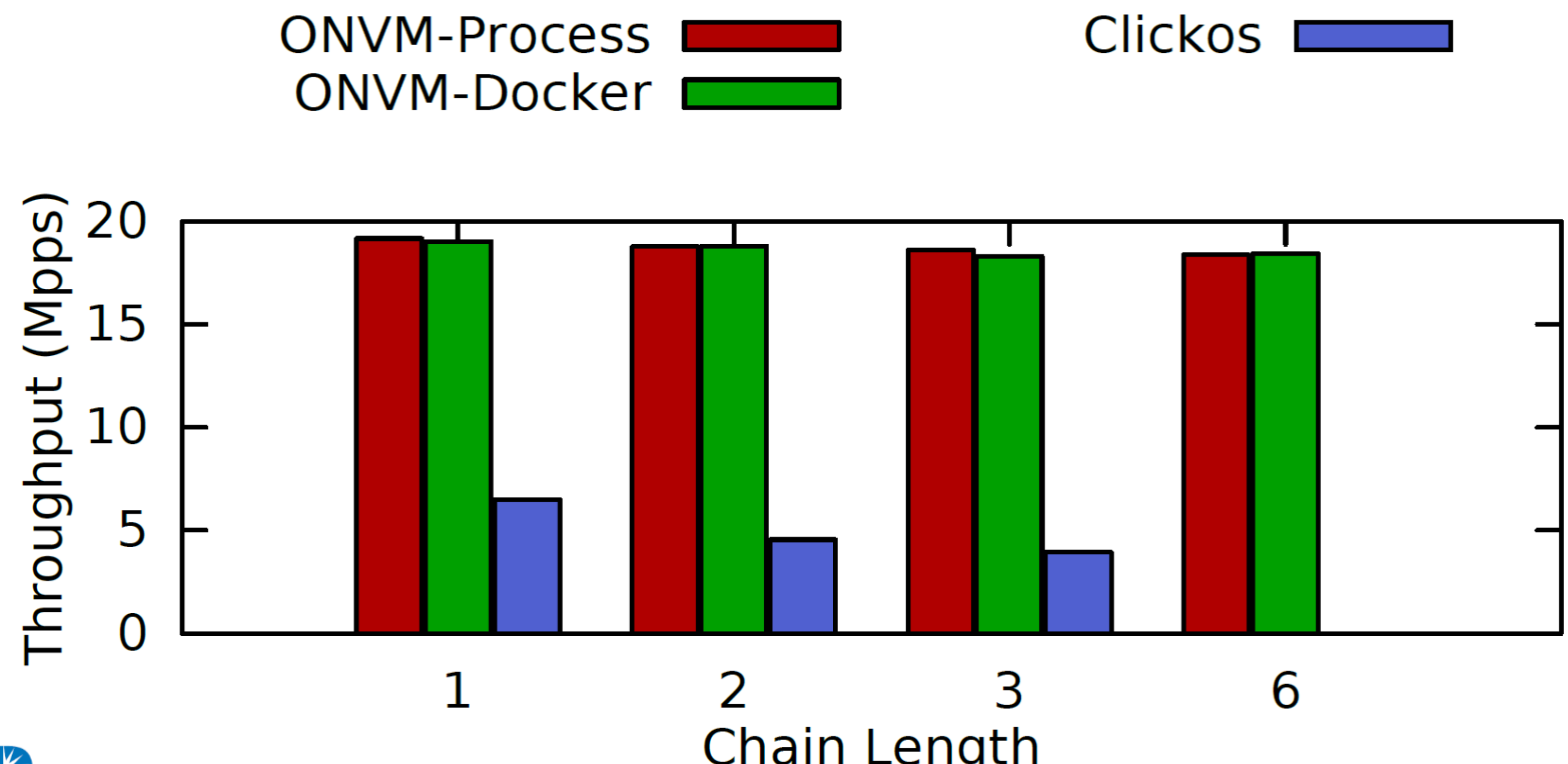
Performance w/ Real Traffic

- Send HTTP traffic through OpenNetVM
  - 1 RX thread, 1 TX thread, 1 NF = 48Gbps
  - 2 RX threads, 2 TX threads, 2 NFs = 68Gbps (NICs bottleneck?)
  - 2 RX threads, 5 TX threads, chain of 5 NFs = 38Gbps
- Fast enough to run software-based edge/core router; Middleboxes function as a ‘bump-in-the-wire’



Service Chain Performance

- Negligible performance difference between processes and containers.
  - OpenNetVM sees only a 4% drop in throughput for a six NF chain, while ClickOS falls by 39% with a chain of three NFs.



OpenNetVM – NFV Open Source Platform

http://sdnfv.github.io

- Network Functions run in Docker containers
- DPDK based design, to achieve zero-copy, high-speed I/O
  - **Key: Shared memory across NFs and NF Manager**
- An open source platform
- Multiple industrial partners evaluating/using OpenNetVM
  - Of course, there are many competitors (e.g., Fast Data Project (fd.io), NetMap, BESS (E2), ClickOS, etc.)

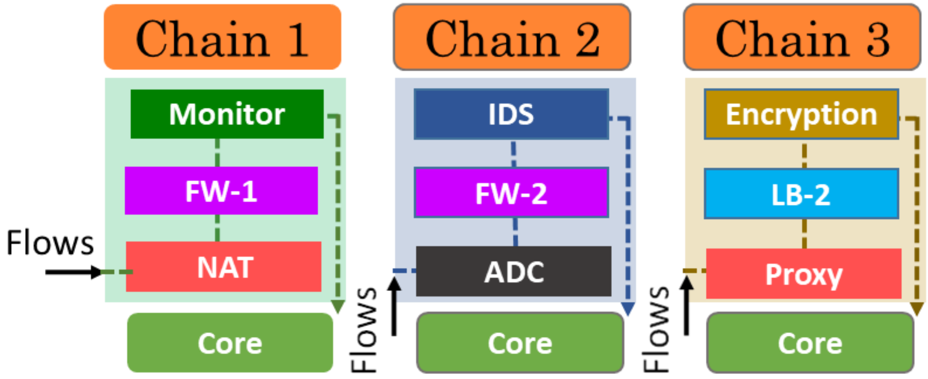
Scalability, Performance, Fairness: NF Scheduling and Backpressure - NFVnice

Sameer G Kulkarni, Wei Zhang, Jinho Hwang and Shriram Rajagopalan, K. K. Ramakrishnan, Timothy Wood and others

(ACM Sigcomm 2017)

How to address performance and scalability for NFV platforms?

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

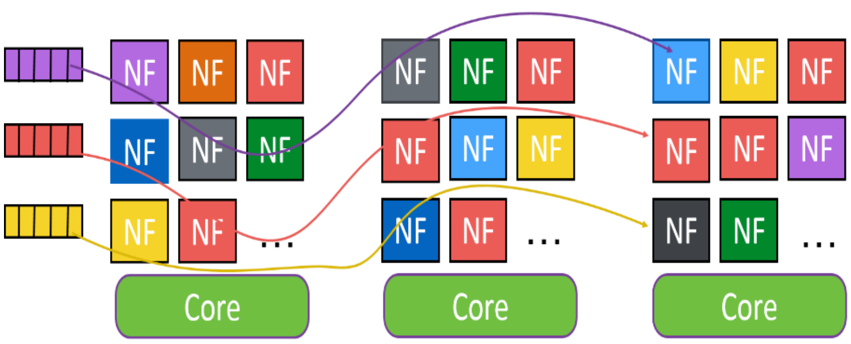


- But, lot of different NFs and diverse NF chains >>> compute cores!
  ☑Performance; ☒Scalability! ☒ Chain Deployment Flexibility!!

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

☑Scalability; ☒Performance?

☑Chain Deployment Flexibility;



How to Schedule the NFs to optimize the system utilization?

Scalability through NF Scheduling

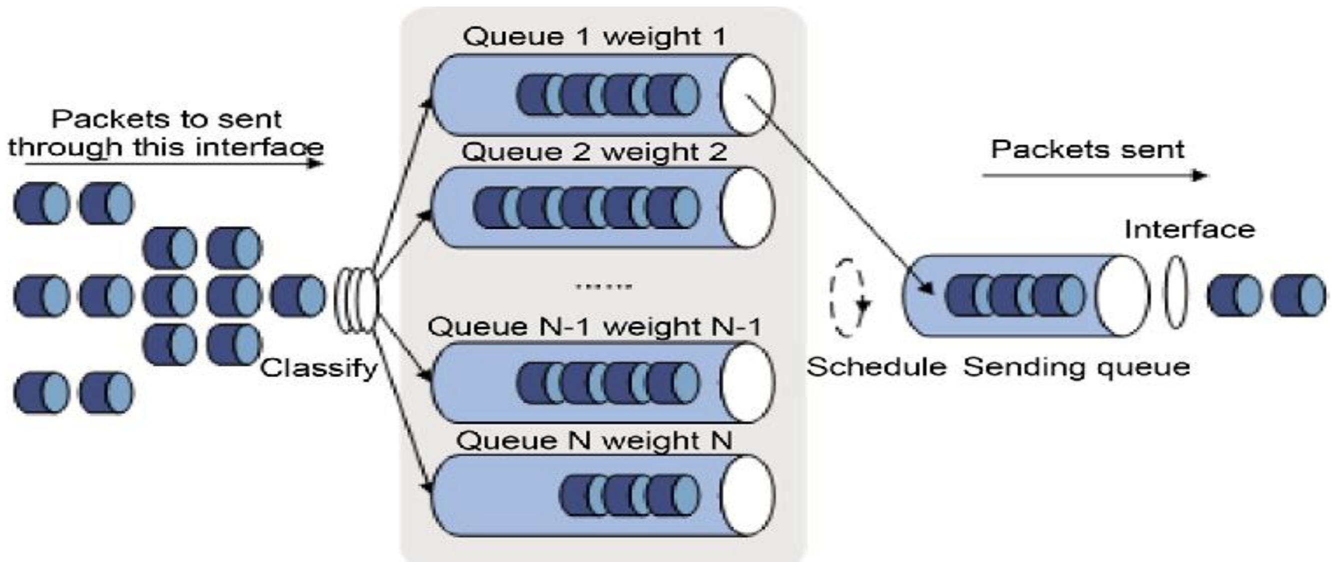
- Why schedule/multiplex more than 1 NF on same core?
  - Not all NFs need full CPU → Poll mode results in wastage of CPU bandwidth.
    - *I/O bound NFs result in sporadic/intermittent usage of CPU.*
  - Diverse & large number of network functions; and diverse NF policies (chains).
    - *# NFs and # NF Chains >> # Cores on Servers*
  - Varying traffic cha’ → not all NFs process packets all the time.
  - Cross core chaining of NFs result in cross-core NUMA overheads.

- To ***support diverse NFs and NF chains***, **to efficiently utilize CPU bandwidth**, and **to avoid CC-NUMA overheads**, multiplexing multiple NFs on the same core may be beneficial.

NF Scheduling: Fundamentals

- NF Scheduling: combines both the ***hardware packet-scheduling*** and ***software process-scheduling*** concepts.
    - NFs/flows sensitive: minimize packet-drops and provide predictable latency

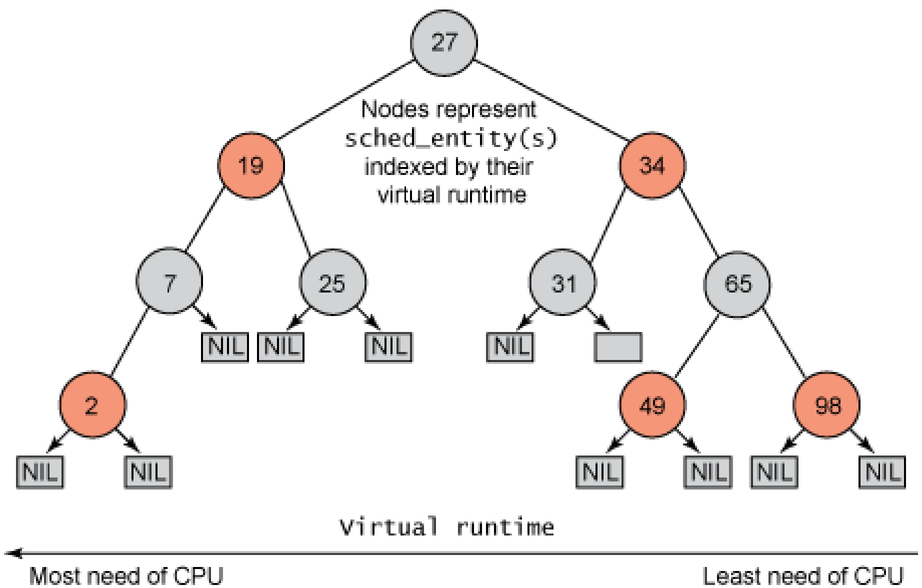
- **Hardware packet-scheduling**:
  - Determine the order of packet transmission at the output link.
  - Examples: FIFO, Round Robin, Weighted RR, Class based, Priority, etc.



  - In NF Scheduling, corresponds to selection of packets for processing at each of the NFs.

NF Scheduling: Fundamentals

- **Software NF Scheduling**: OS Schedulers allocate CPU to the competing NFs.
- Different OS Schedulers: NORMAL (CFS, Batch), Real Time (Round Robin, FIFO).



- CFS: "Virtual-runtime" based fair scheduler. (numbers in tree represent the total vruntime accrued by each process).
- "Red-black" tree: efficient, self-balancing data-structure to store/sort the tasks.
- Left-most task (least vruntime) is scheduled.
- Space needed for Red-black tree is **O(n)**.
- Search, Insert, Delete in the order: **O(log$n$)**.

- Diverse (heterogeneous) NFs have different per-packet computation costs.
- Per-packet processing cost of the NF can vary → resulting in different CPU utilizations over time.
- NFs can be part of one or multiple service chains → order of scheduling the NFs can impact the packet processing.

NF Scheduling: Desiderata

- Multiplexing more NFs on same core:
  - Diverse network functions: I/O intensive or different computation costs; scheduling them on a single CPU can impact the overall performance (throughput and latency).
  - Flows can be of varying rates, going through different NF chains.
  - Ensure fair-sharing of the CPU resource across all the competing NFs.
    - ***What is the right notion of fairness for allocating CPU to the NFs?***
- Honor both packet-level scheduling requirements (per-flow QoS) and task-level process scheduling requirements (per-NF QoS).
  - ***What is/are the right knobs(s) to tune and control the OS schedulers?***
- In addition, packet processing at NFs needs to follow the order in the service graph. Scheduling order can have a large impact on throughput and latency.
  - ***What is the right sequence/order for scheduling the NFs?***

Use Existing Linux Schedulers?

- Vanilla Linux schedulers:

Completely Fair Scheduler
- Normal or Default
- Batch (longer time scales)

- Virtual run time
- Nanosecond granularity

Real Time Scheduler
- Round Robin
- FIFO

- Time slice
- Millisecond granularity

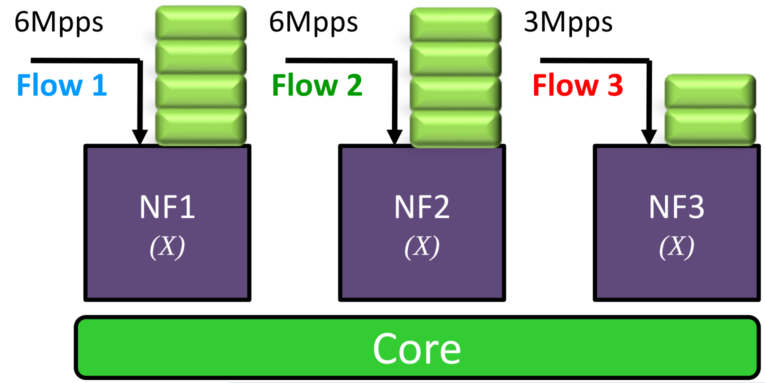
- We want:
  - High **throughput**
  - **Fairness** across NFs
  - Low **latency**
  - Low **context switch** overheads

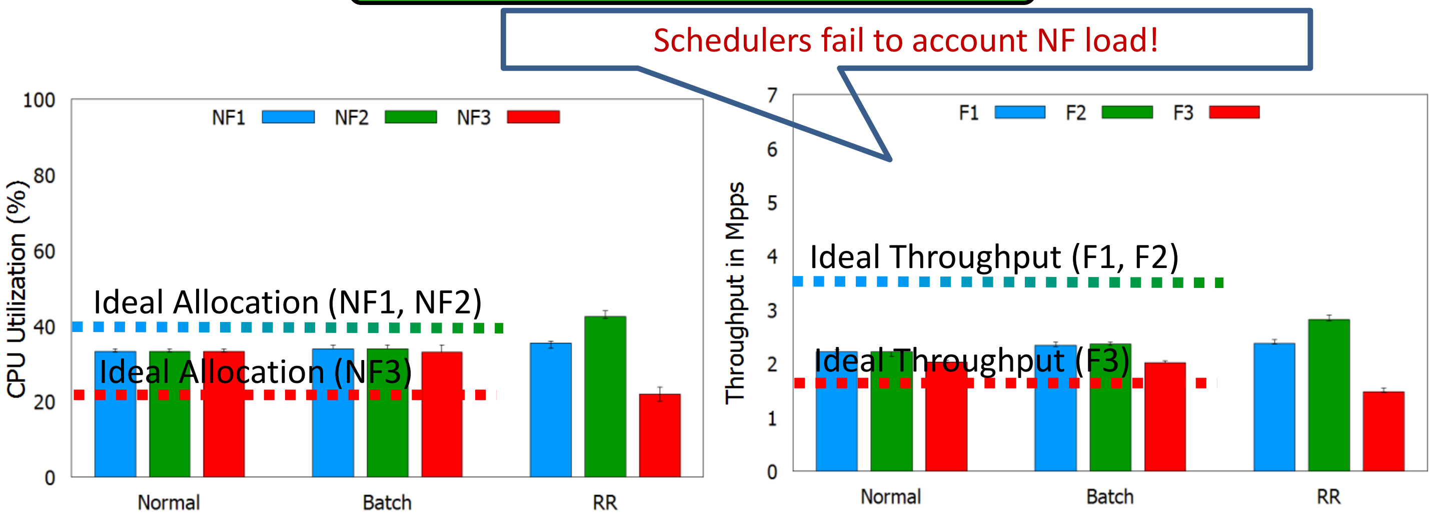
Do existing schedulers perform well?

Do schedulers account for:
- Offered Load?
- NF cost heterogeneity?
- Chaining sequence?

OS Scheduler Characterization (Load)

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

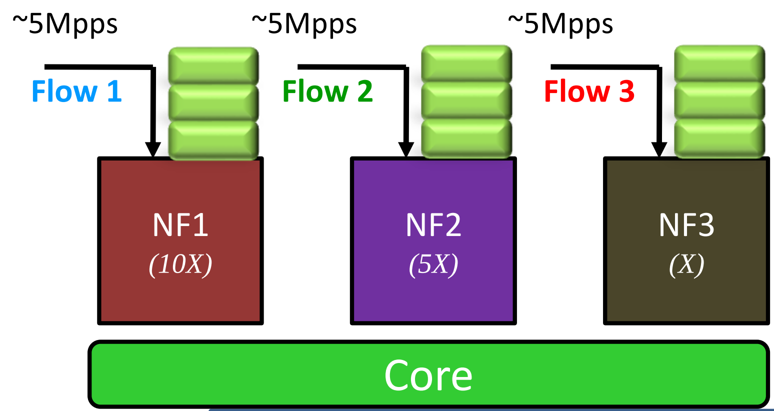




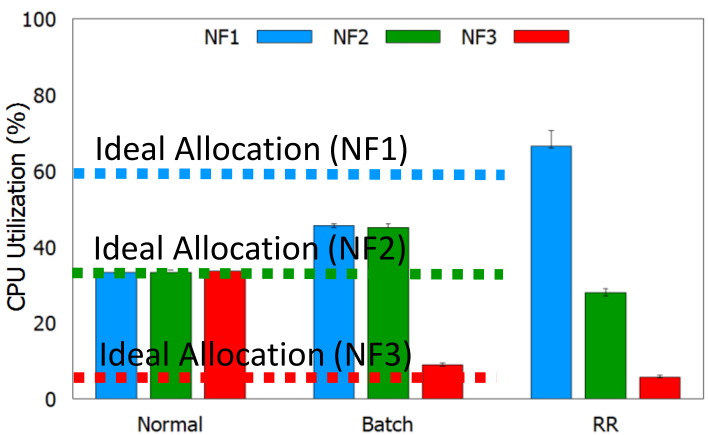
- Normal and Batch CFS always allocate CPU equally.
- RR: CPU allocation depends on voluntary yield within the allotted time-slice.
- Ideally, rate proportional allocation for NF1:NF2:NF3 must be 2:2:1.

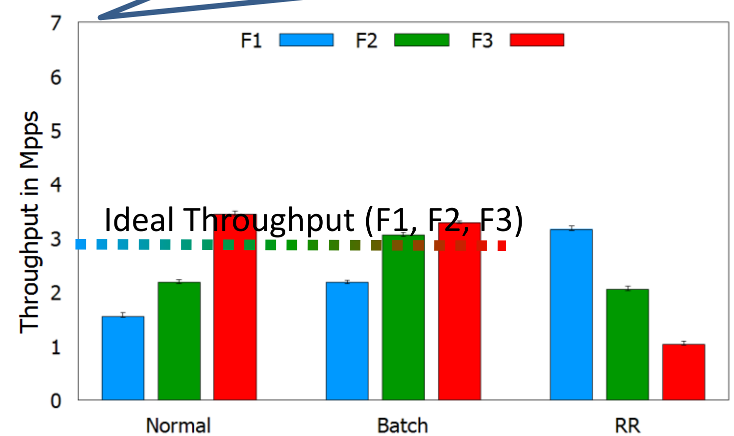
OS Scheduler Characterization (Cost)

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



Schedulers fail to account NF Processing cost!

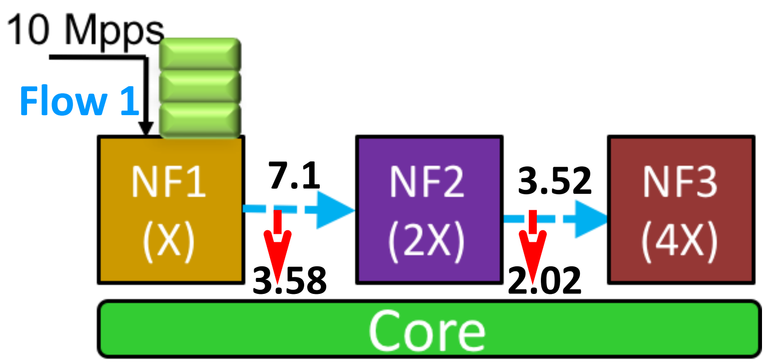


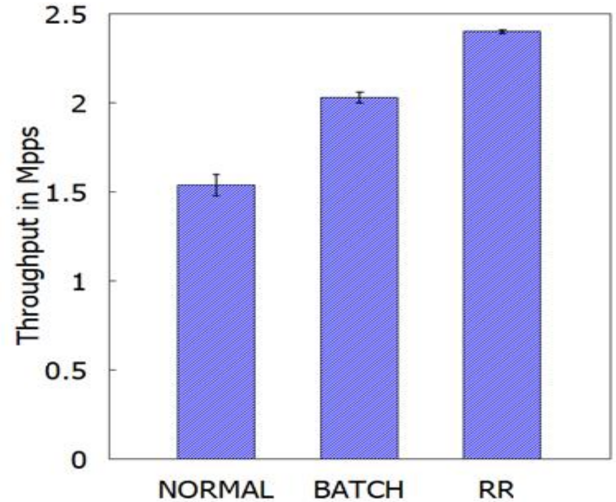


- Again, Normal CFS allocates CPU equally; Batch CFS is coarser allocation (better)
- RR: allocation depends on voluntary yields in allotted time-slice.
- Ideally the cost proportional allocation for NF1:NF2:NF3 needs to be 10:5:1

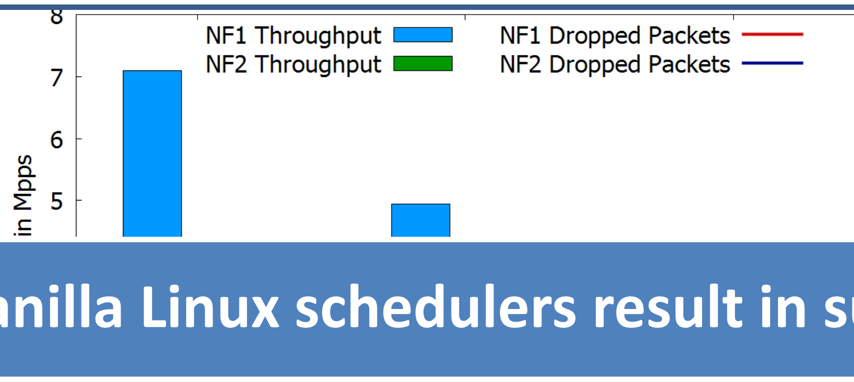
OS Scheduler Characterization (Chain)

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





Too many/too little context switches result in **overhead** and **inappropriate** allocation of CPU



Ctx sw/s	NORMAL	BATCH	RR
Total	20K/s	2K/s	1K/s

Vanilla Linux schedulers result in sub-optimal resource utilization.

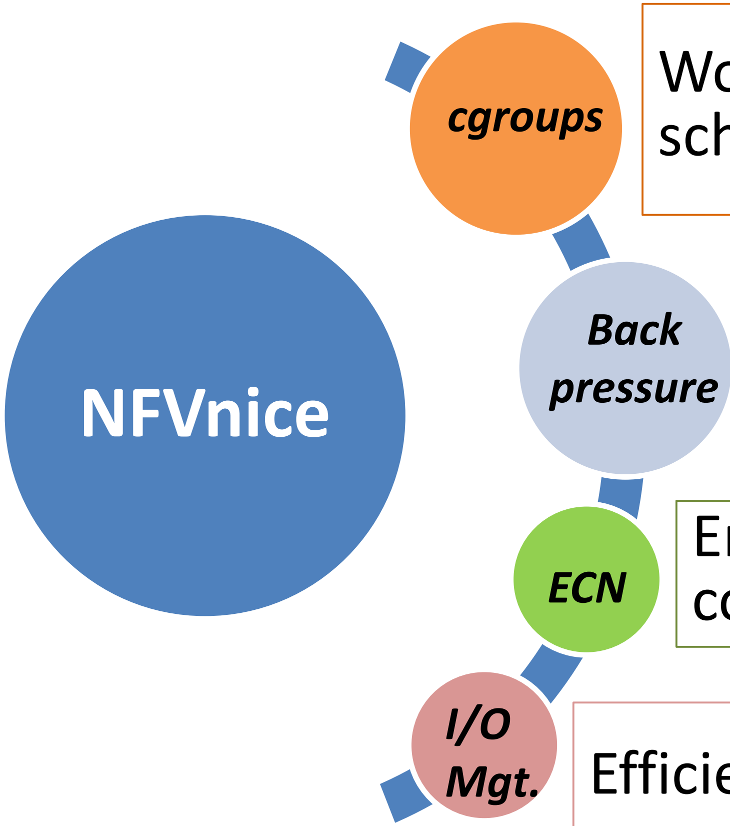
Need the schedulers to be: Load, NF characteristic, & chain aware!

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 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)*

Chain-aware scheduling; Avoid wasted work *(within and across cores of a node)*

End-2-End Bottleneck/congestion control *(across nodes)*

Efficient Disk I/O Mgmt. Library

cgroups

Rate-Cost Proportional Fairness

- **What is Rate-Cost Proportional Fairness?**
  - Determines the NFs CPU share by accounting for:
    - NF Load (Avg. packet arrival rate, instantaneous Queue length)
    - NF Priority and the median per-packet computation cost.
- **Why?**
  - Efficient and fair allocation of CPU to the contending NFs.
  - Flexible & Extensible approach to adapt to any QOS policy.
- **How?**
  - **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.

cgroups

Rate-Cost Proportional Fairness

Initialization

mkdir /cgroupfs/NF(i)

Weight Computation

$$load(i) = \lambda_i \times S_i$$

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

$$NFShare(i) = Priority_i \times \frac{load(i)}{Total\ Load(m)}$$

Update Cgroup Weight

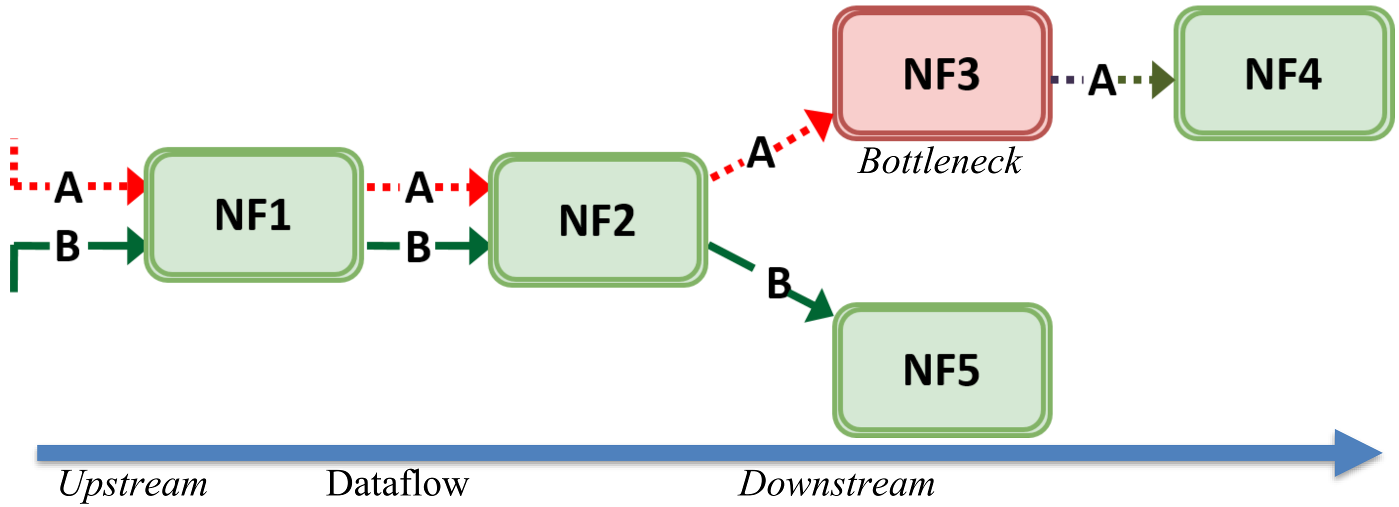
Write "NFShare(i)" to /cgrougfs/NF(i)/cpu.shares

Every 10 ms

Back pressure

Backpressure in NF chains

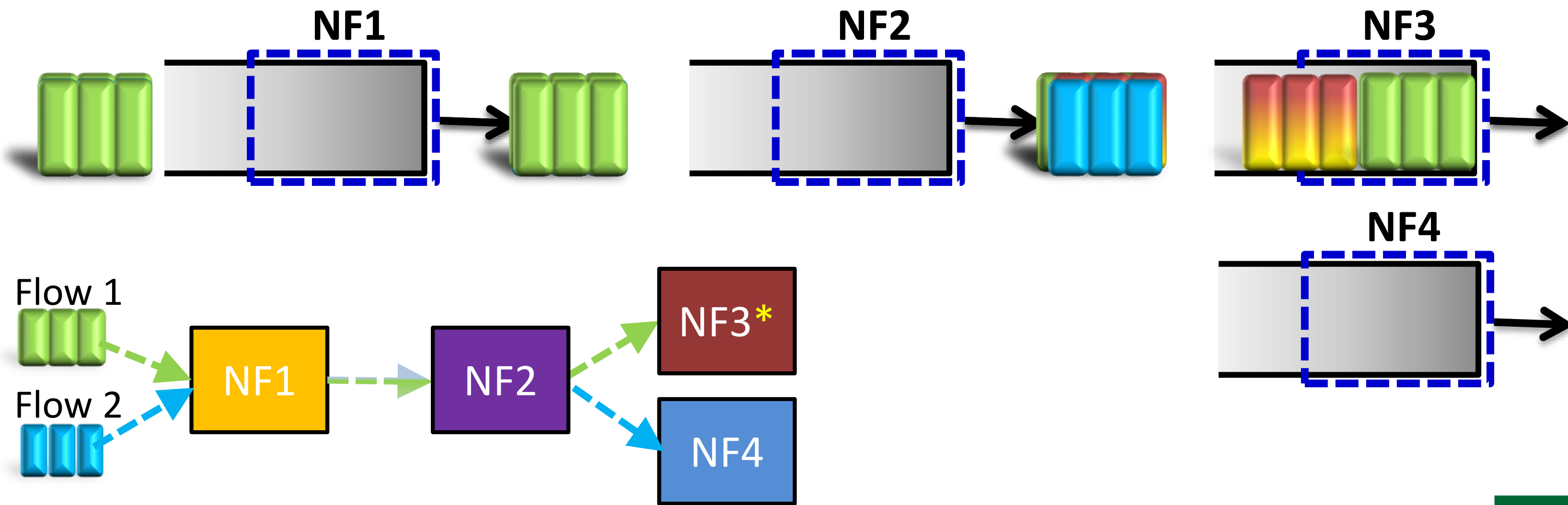
- Selective per chain backpressure marking.



  - Only Flow “A” going through bottleneck NF (NF3) is back-pressured and throttled at the upstream source NF1.
  - while Flow “B” is not affected.

Back pressure →

NFVnice Backpressure

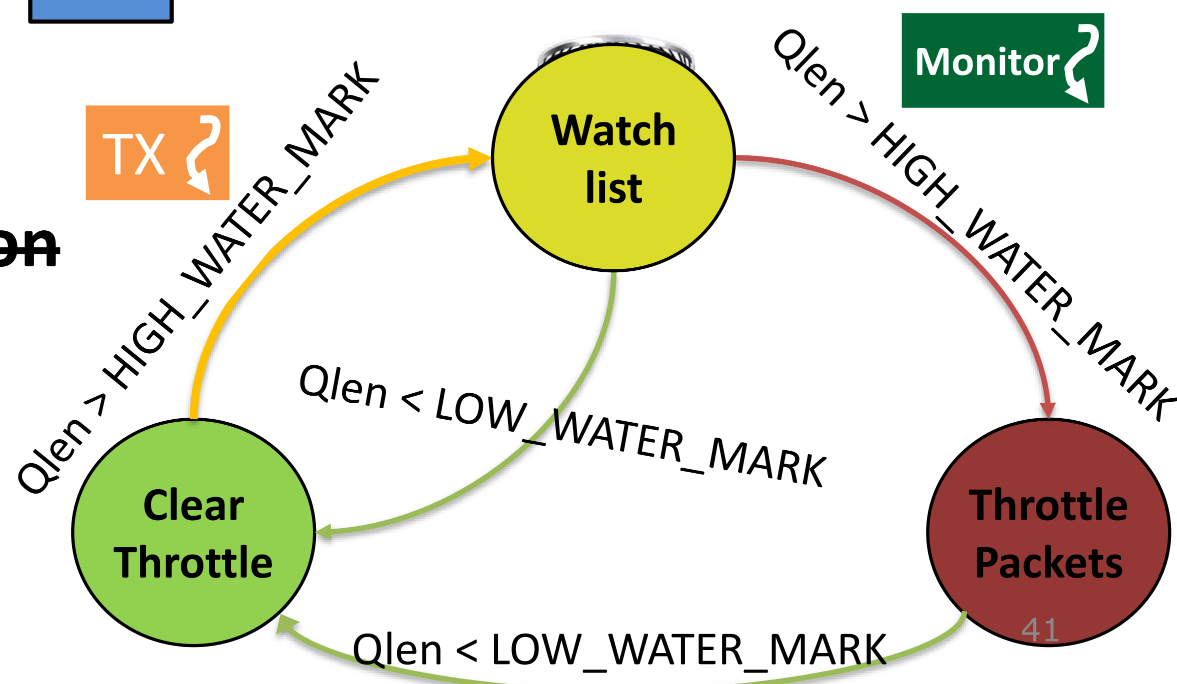


~~**Lots of Wasted Work**~~

~~**Incurs Delay, No Isolation**~~

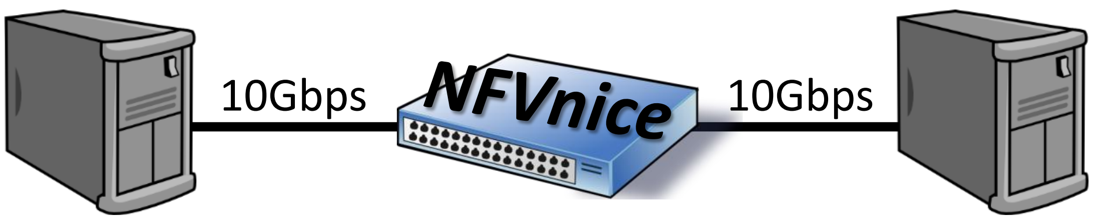
Reacts Instantaneously

Chain Isolation



Evaluation

- Testbed:
  - Hardware: 3 Intel Xeon(R) CPU E5-2697, 28 cores @2.6Ghz servers, with dual port 10Gbps DPDK compatible NICs.
  - Software: Linux kernel 3.19.0-39-lowlatency profile.
  - NFVnice: built on top of OpenNetVM.



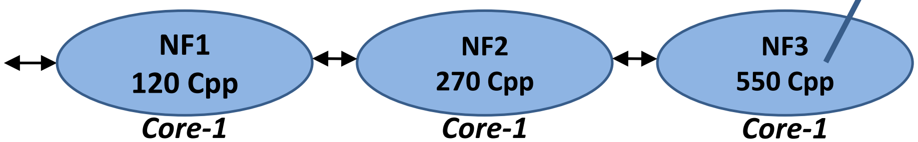
- Traffic:
  - Pktgen and Moongen: Line rate traffic (64 byte packets).
  - Iperf: TCP flows.

- Schemes compared:
  - Native Linux Schedulers with and w/o NFVnice.

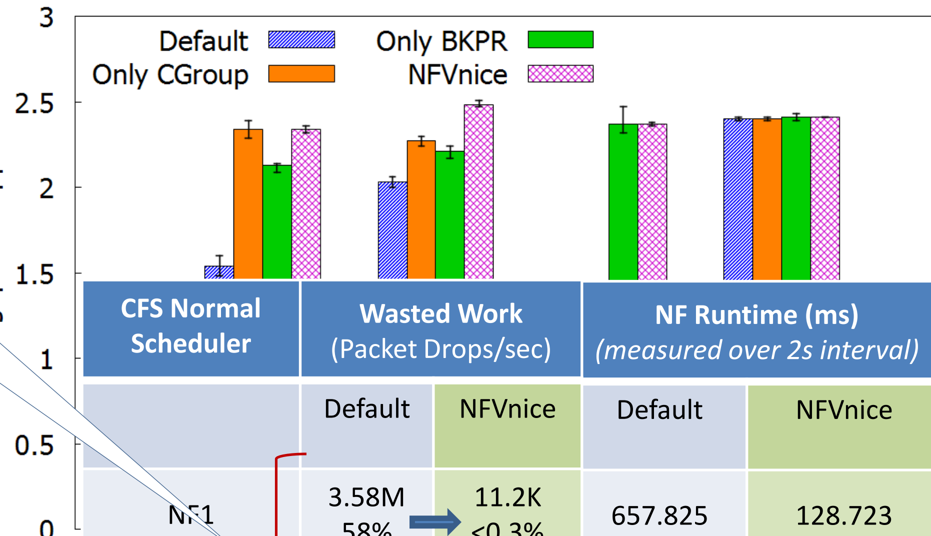
  Different NFs (varying computation costs) and chain configurations.

Performance: Impact of cgroup weights and Backpressure

Simple Three NF Chain



Cycles per packet



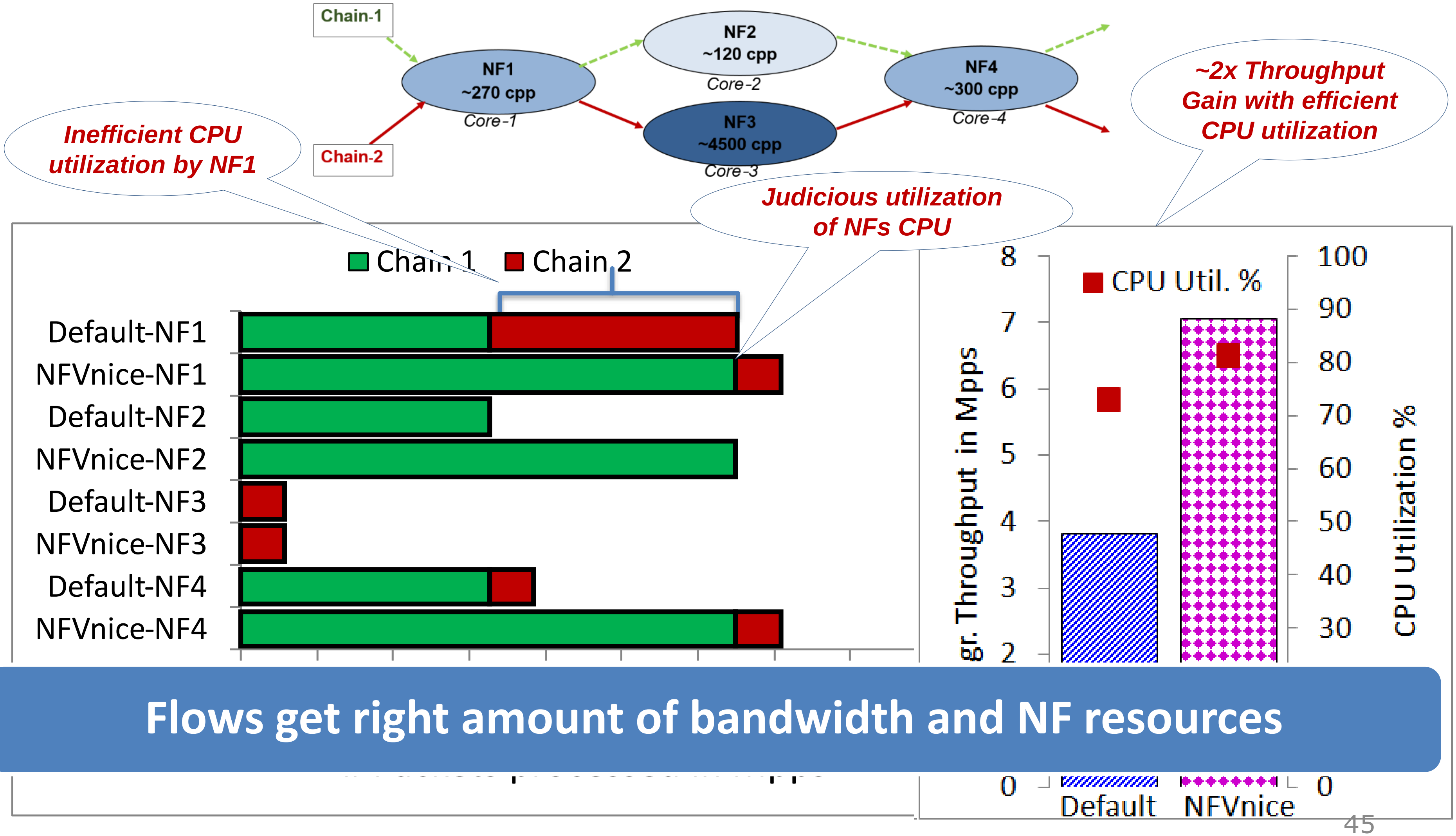
Significant Reduction in Wasted Work!

CFS Normal Scheduler	Wasted Work (Packet Drops/sec)		NF Runtime (ms) *(measured over 2s interval)*	
	Default	NFVnice	Default	NFVnice
NF1	3.58M 58%	11.2K <0.3%	657.825	128.723
NF2	2.02M	12.3K	602.285	848.922

CPU Allocation α to **Computation Cost**

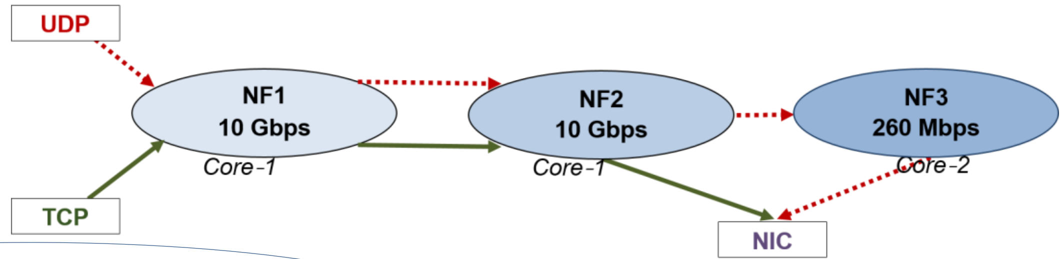
NFVnice improves throughput for all kernel schedulers.

Performance + Resource Utilization



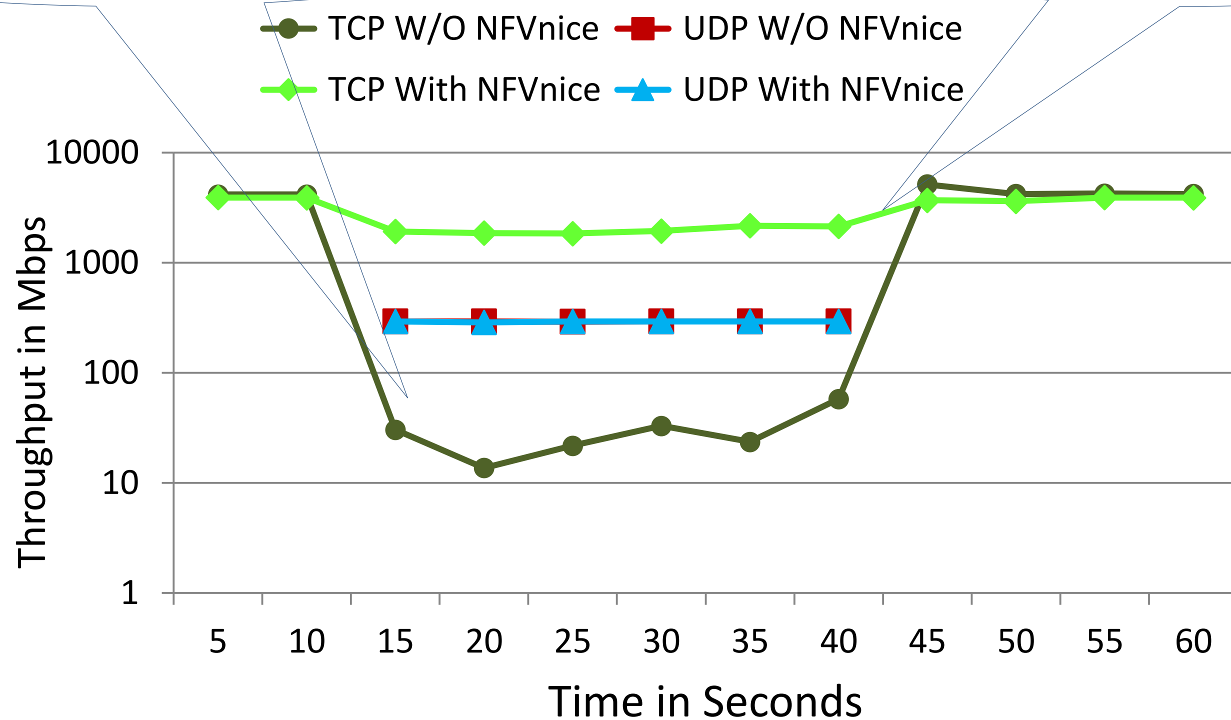
Flows get right amount of bandwidth and NF resources

TCP and UDP Isolation

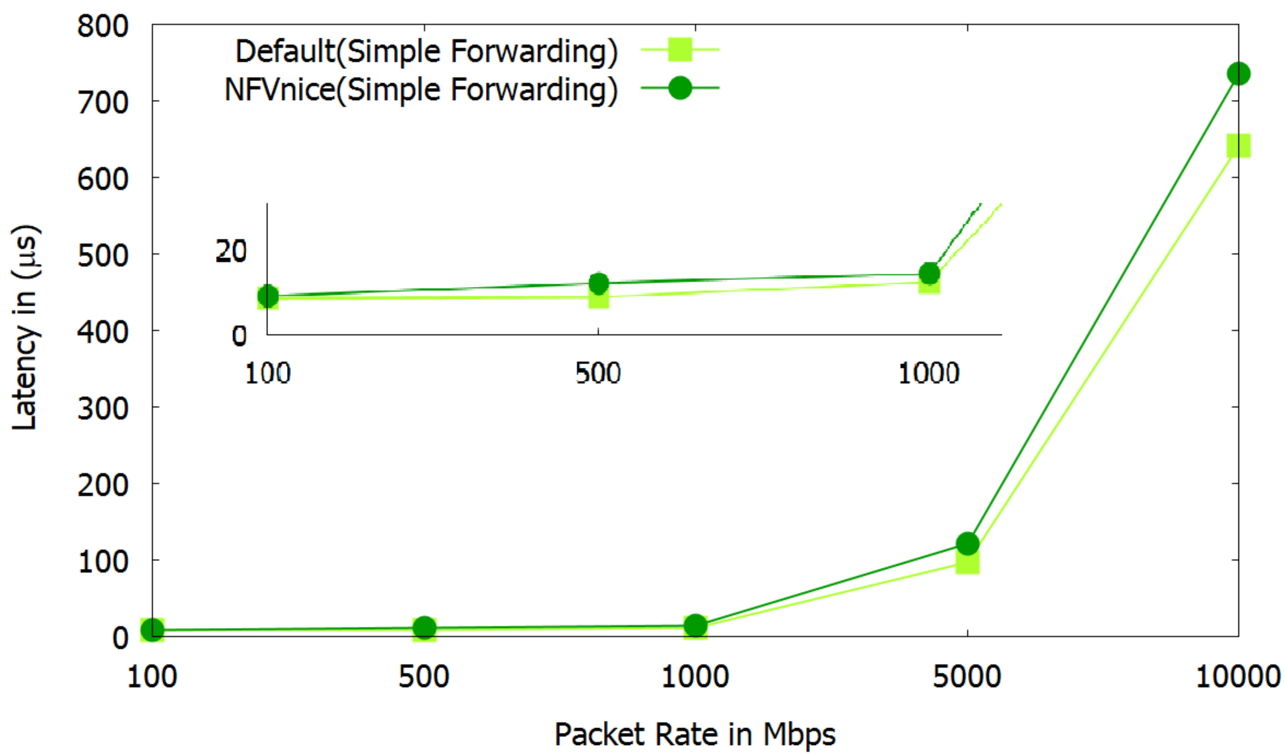


TCP affected by UDP flows!
Wastage of NF1,NF2 bandwidth

Effectively isolates UDP and TCP flows

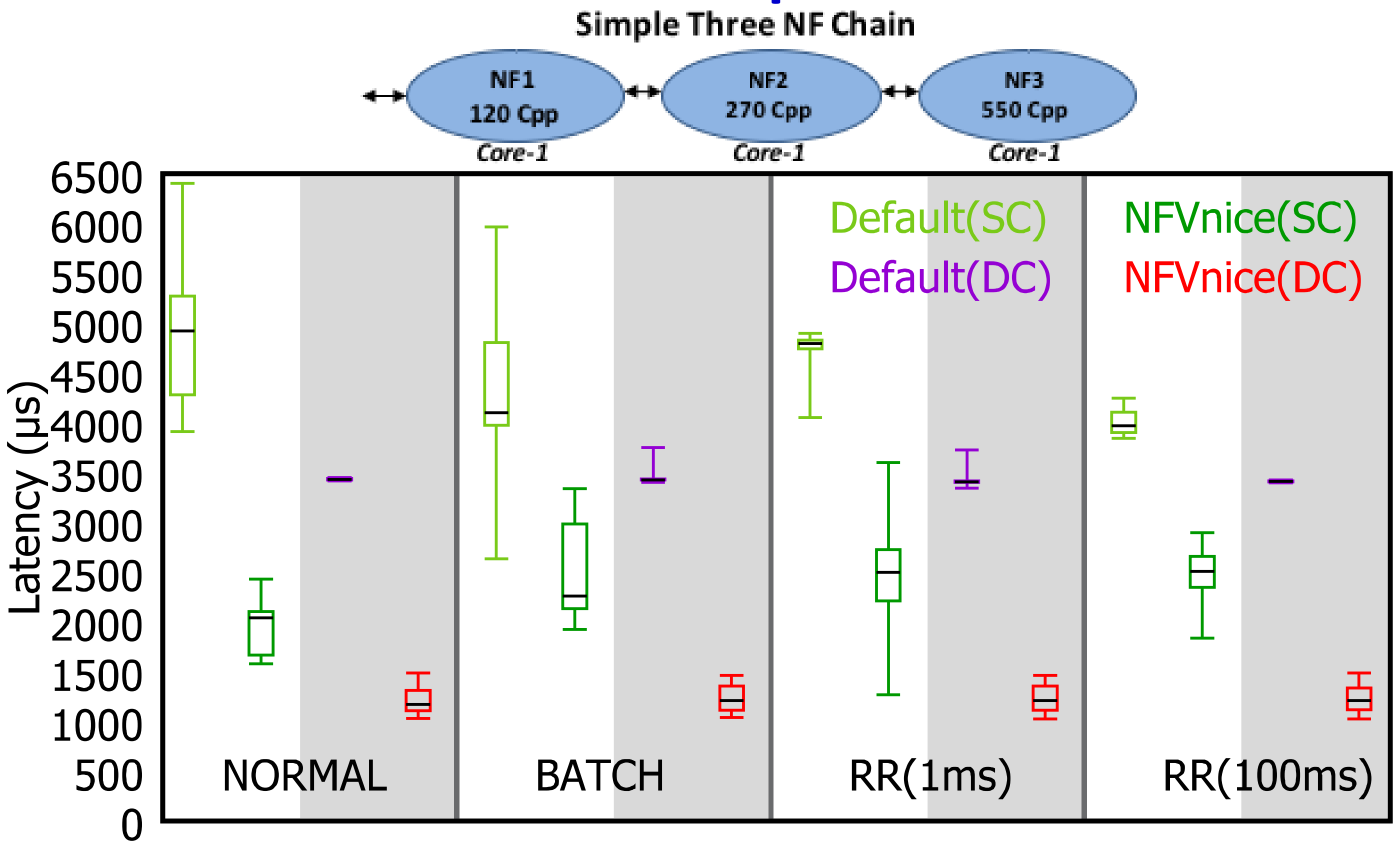


Impact on Simple Forwarding Latency



- Latency varies with packet rate (due to queuing at higher rates).
  - Under low load the latency is 8~20 µs.
  - At higher load latency increases to around 650 µs.
  - NFVnice processing has minimal impact on base forwarding latency across all load conditions.

OS Scheduler impact on Latency



- Same core(SC) scheduling vs pinning NFs to different cores (DC).
  - Underlying scheduler significantly impacts the chain latency.
- NFVnice improves latency across all the schedulers.
  - NFVnice benefits even when NFs pinned to different cores.

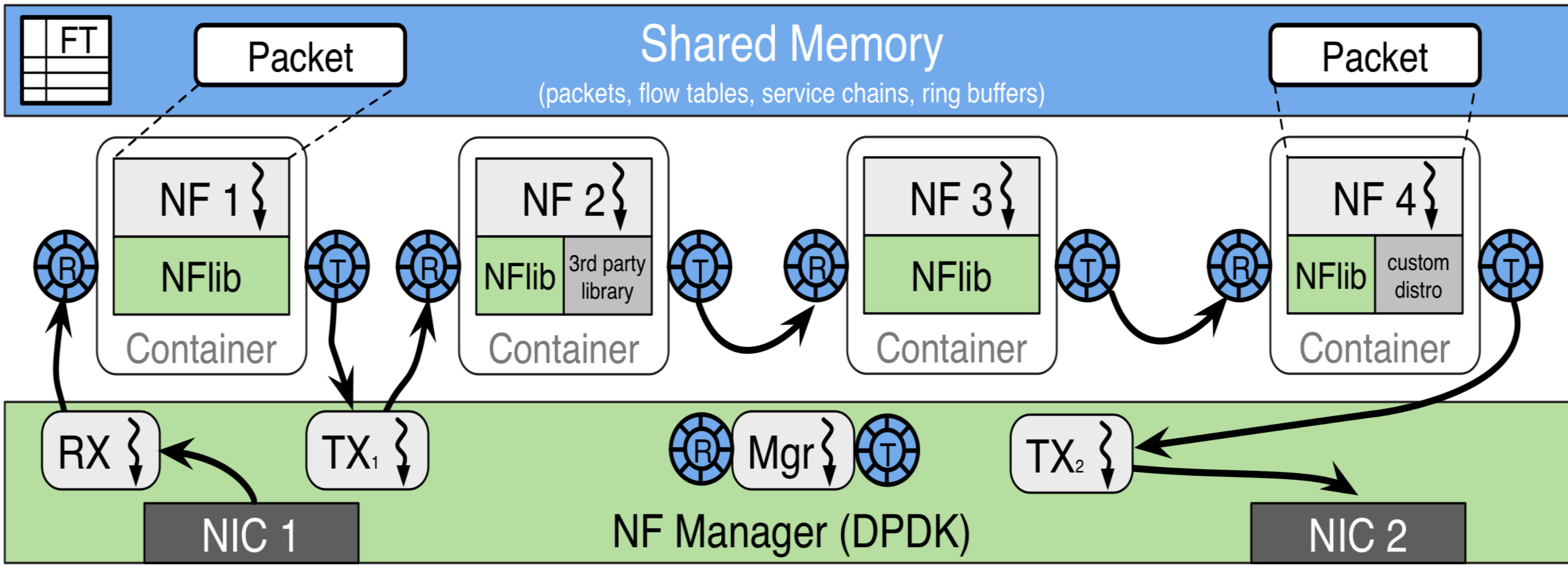
Summary

- Networks are changing – moving to a software base
  - SDN's centralized control
  - NFV's software based implementations
- OpenNetVM – a high performance NFV platform with containers; shared memory for zero-copy
- With proper NF scheduling and flow management (backpressure) we are able to provide scalability, fairness and effectively use available CPU resources
  - Packet processing by software platforms need to be rate and cost aware
    - Rate-and-cost proportional fairness important

Getting OpenNetVM

- Source code and NSF CloudLab images at

  http://sdnfv.github.io/


FT
Packet
Shared Memory
(packets, flow tables, service chains, ring buffers)
Packet
NF 1
NFlib
Container
NF 2
NFlib
3rd party library
Container
NF 3
NFlib
Container
NF 4
NFlib
custom distro
Container
R
T
RX
TX1
Mgr
TX2
NIC 1
NF Manager (DPDK)
NIC 2