Resource and NF Management Challenges

K. K. Ramakrishnan

and

Tim Wood

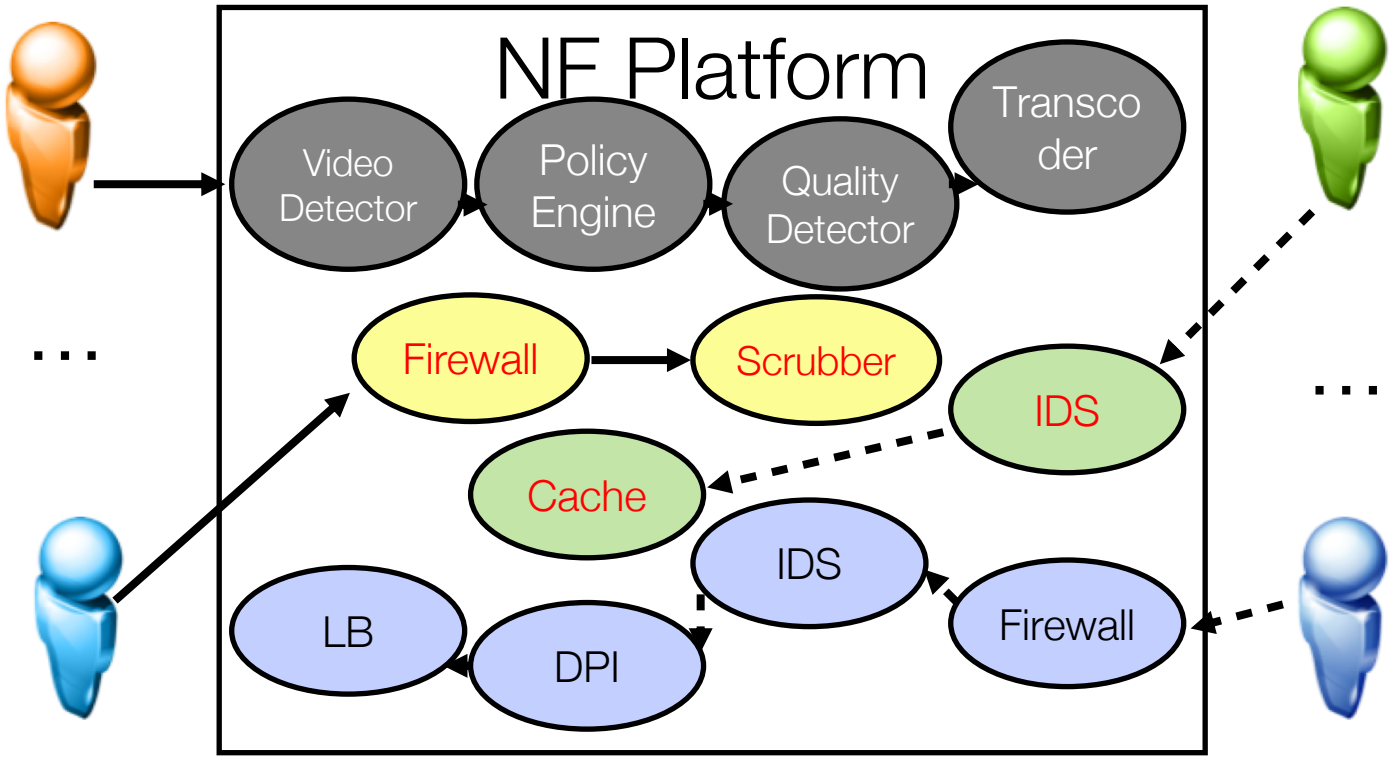
UNIVERSITY OF CALIFORNIA, RIVERSIDE

Per Flow Customization

(Flurries: ACM CoNext 2016)

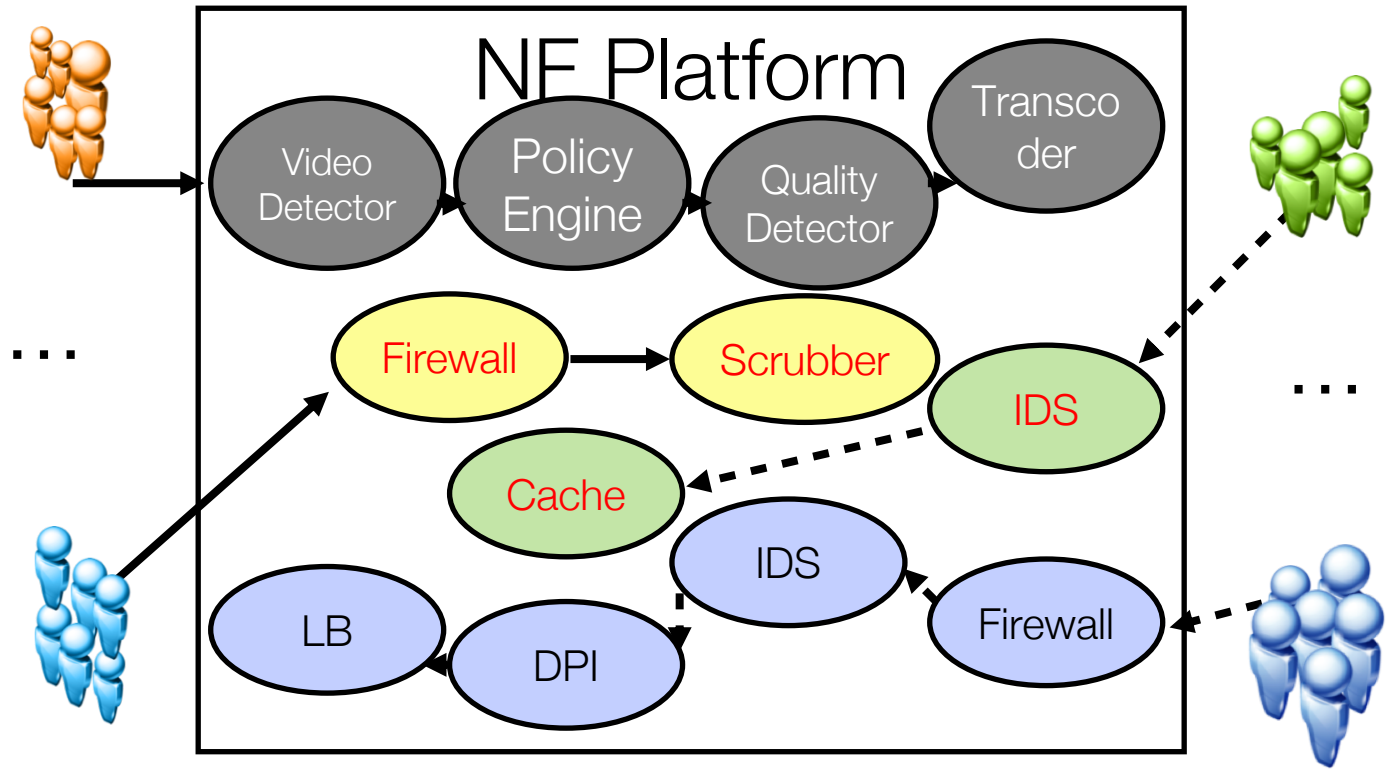
Service Diversity & Multiple Flows

- A typical NF platform may host NFs for many different service chains
- Each flow may need customized services



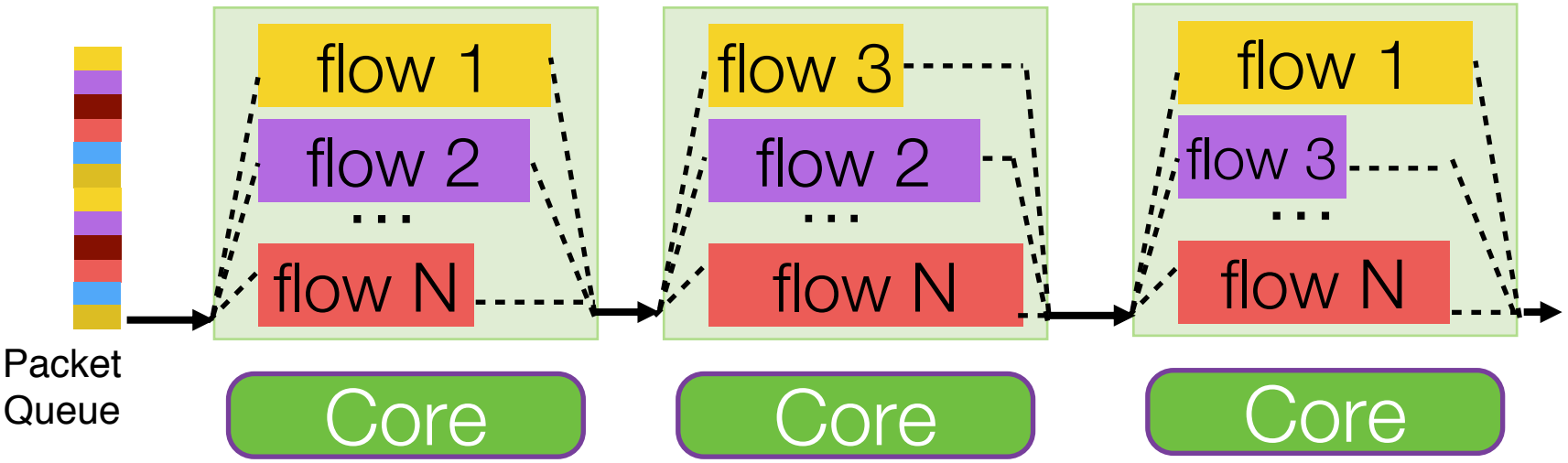
Service Diversity & Multi-Flows

- NF platforms host NFs for many different service chains
- Each flow may need customized services
- Many different flows, each with slightly different need



Monolithic NFs

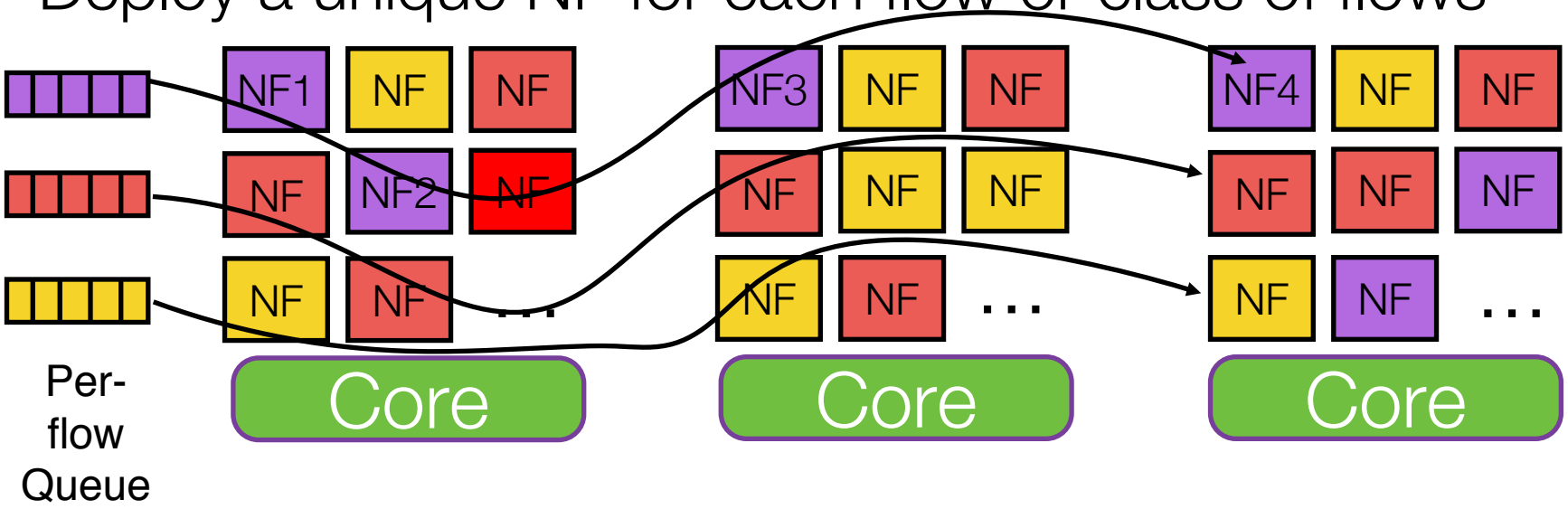
- Multiple flows have to go through an NF
  - Scheduling packets: complex, multiple flows share packet queues
  - NF must classify flows? Or NF manager?
  - Manage flow interference
  - Scalability: avoid restriction of 1 core per NF



Need a **high speed** platform which can **isolate** and process flows with **fine granularity** *and* **efficiently** use resources

Goal: Per-Flow NFs

- Make the flow the scheduling entity
- Deploy a unique NF for each flow or class of flows



Flurries

- A **scalable** platform for **unique, short-lived** NFs
- Published in ACM CoNext 2016 Conference
- **Run unique NFs per flow or per class of flows**
- **Benefits:**
  - **Flexible and customized flow processing**
  - **Flow-level performance management**

Flurries

. A **scalable** platform for **unique, short-lived** NFs

Flurries contributions/features:

- Hybrid polling and interrupts to efficiently run 1000s of NFs
- Flow director maps flows to NFs; NFlib recycles NFs
- Adaptive wakeup system and prioritized NF scheduling

. Challenges

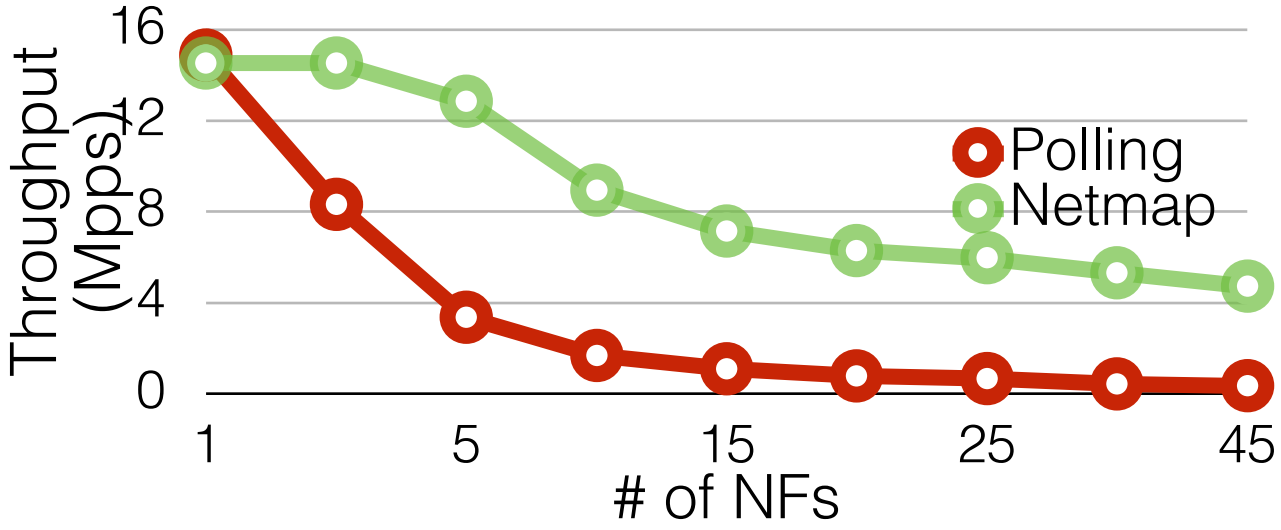
- How to move packets efficiently across service chains?
- How to run large numbers of NFs on a host?
- How to manage the mapping of flows to NFs?
- How to schedule NFs?

Current Data Plane Platforms

- Current data plane platforms:
  - Function based: DPDK pipeline, BESS [TR'15], VPP
  - VM or container-based: netmap [ATC'12], NetVM [NSDI'14], ClickOS [NSDI'14]
- Can not run many NFs per host

Poor isolation

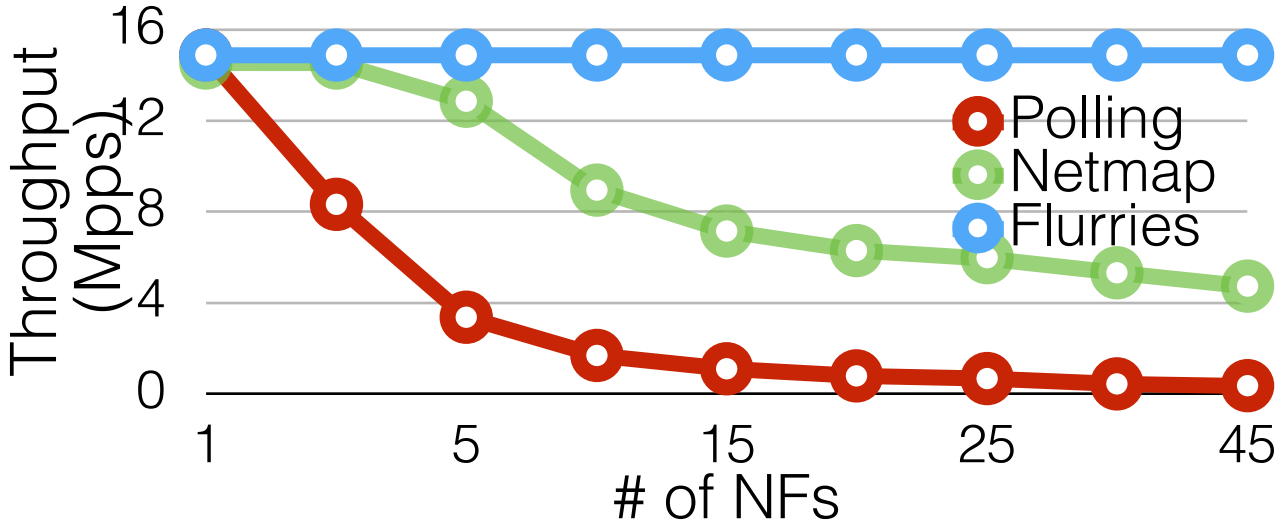
Poor performance or flexibility



Neither polling or interrupts can provide high scalability

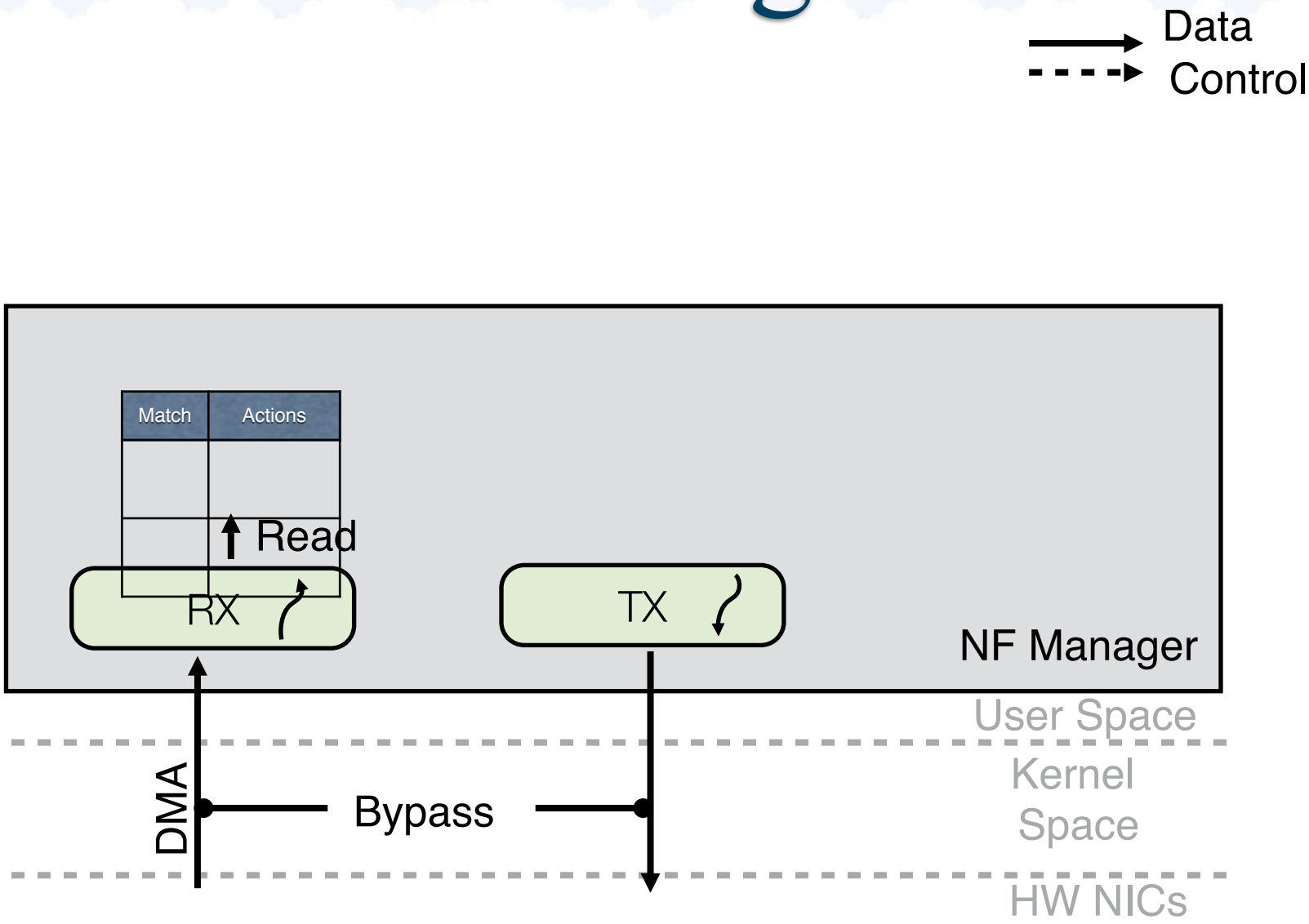
Current Data Plane Platforms

- Current data plane platforms:
  - Functions: DPDK pipeline, BESS [TR'15], VPP
  - VM or container-based: netmap [ATC'12], NetVM [NSDI'14], ClickOS [NSDI'14]
- Can not run many NFs per host

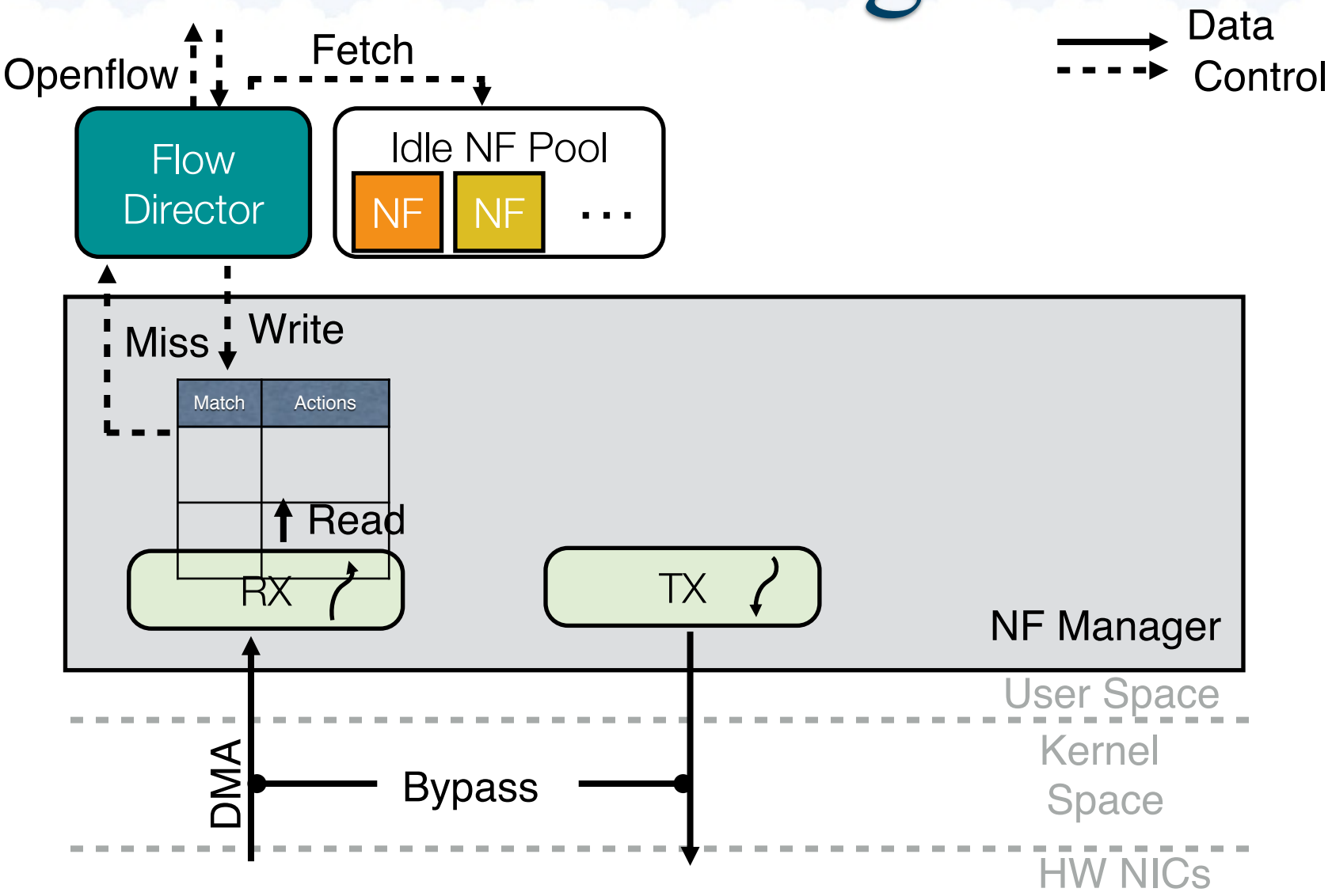


Goal: fine grained isolation, flexibility, and performance

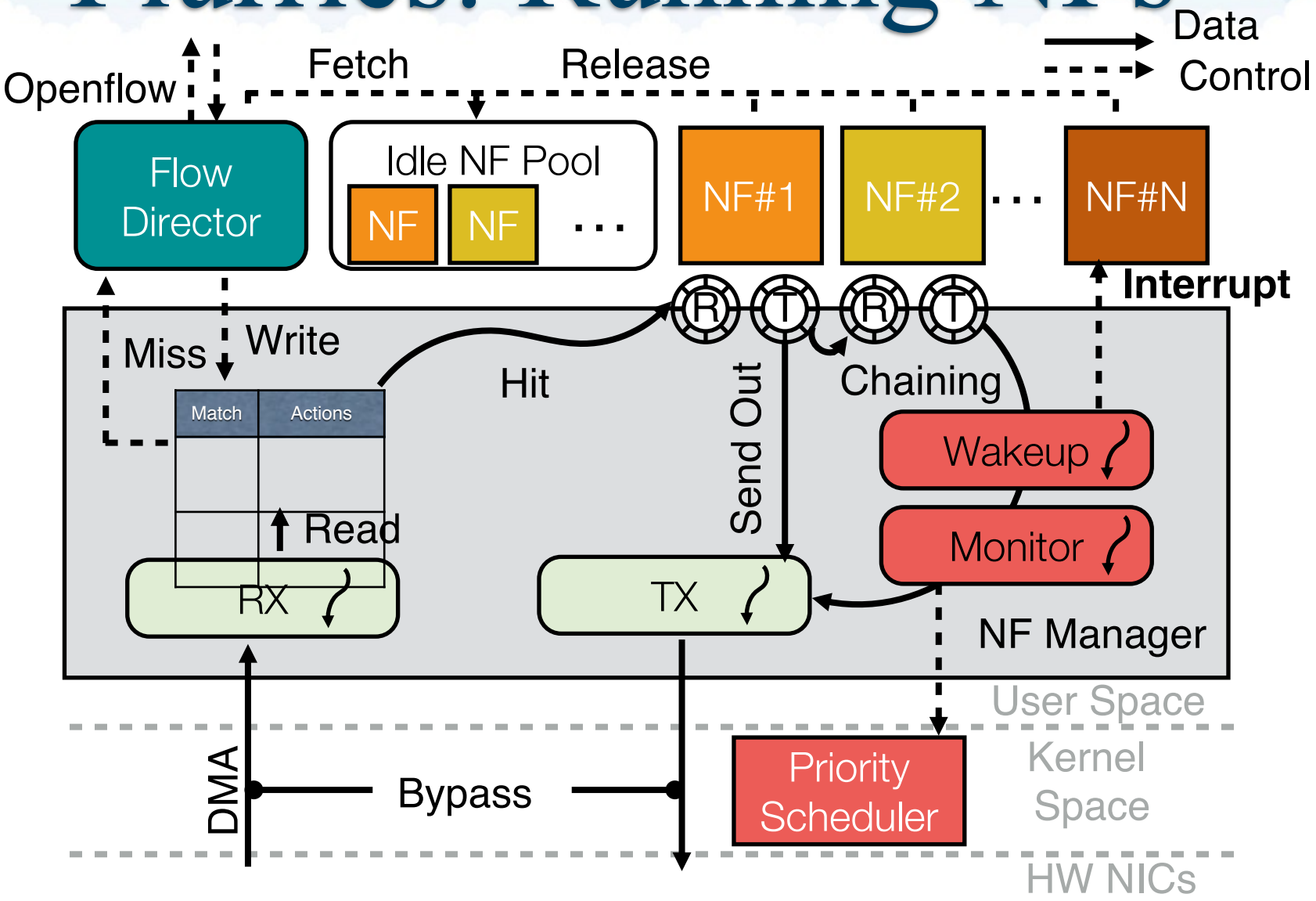
Flurries: Moving Packets



Flurries: Directing Flows

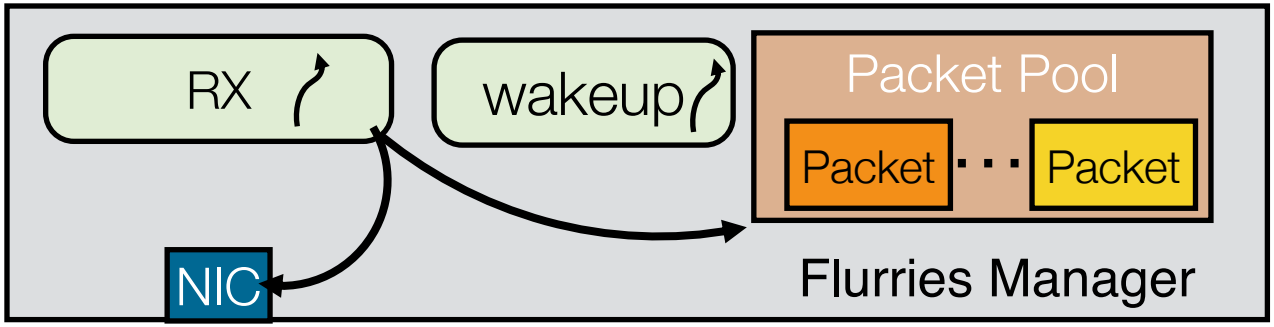


Flurries: Running NFs



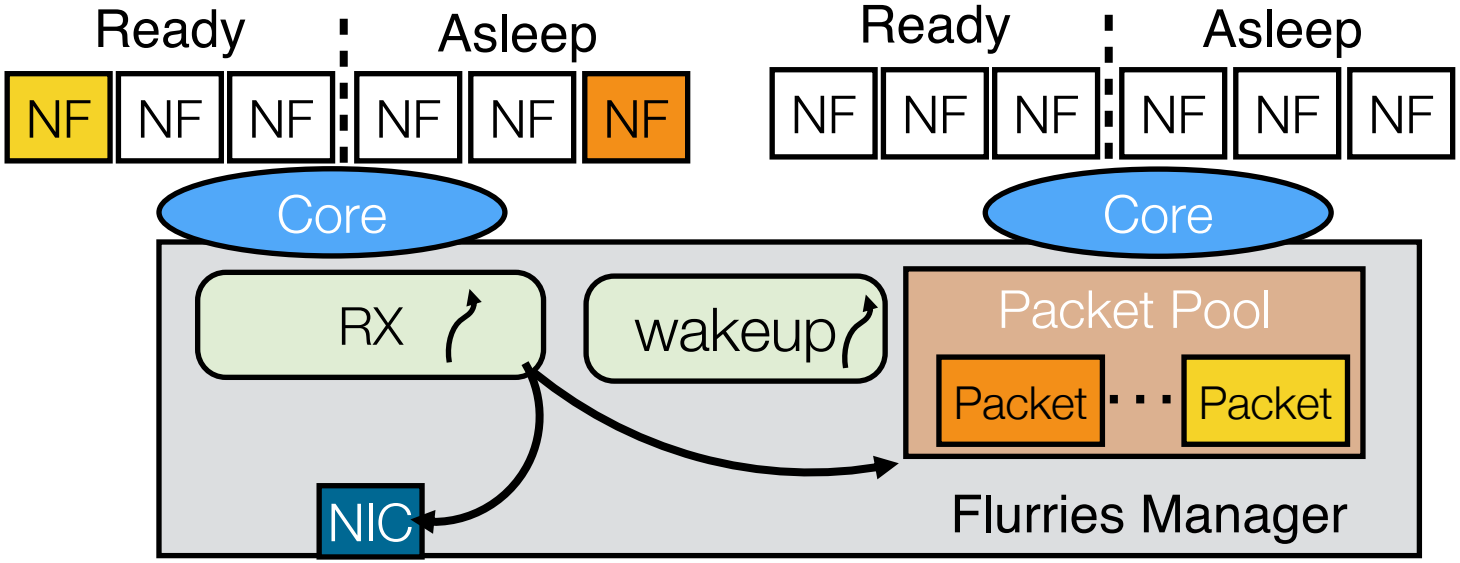
Hybrid Polling and Interrupt

- Manager's RX thread polls the NIC for packets
  - **Polling** retrieves packets with high throughput and low latency



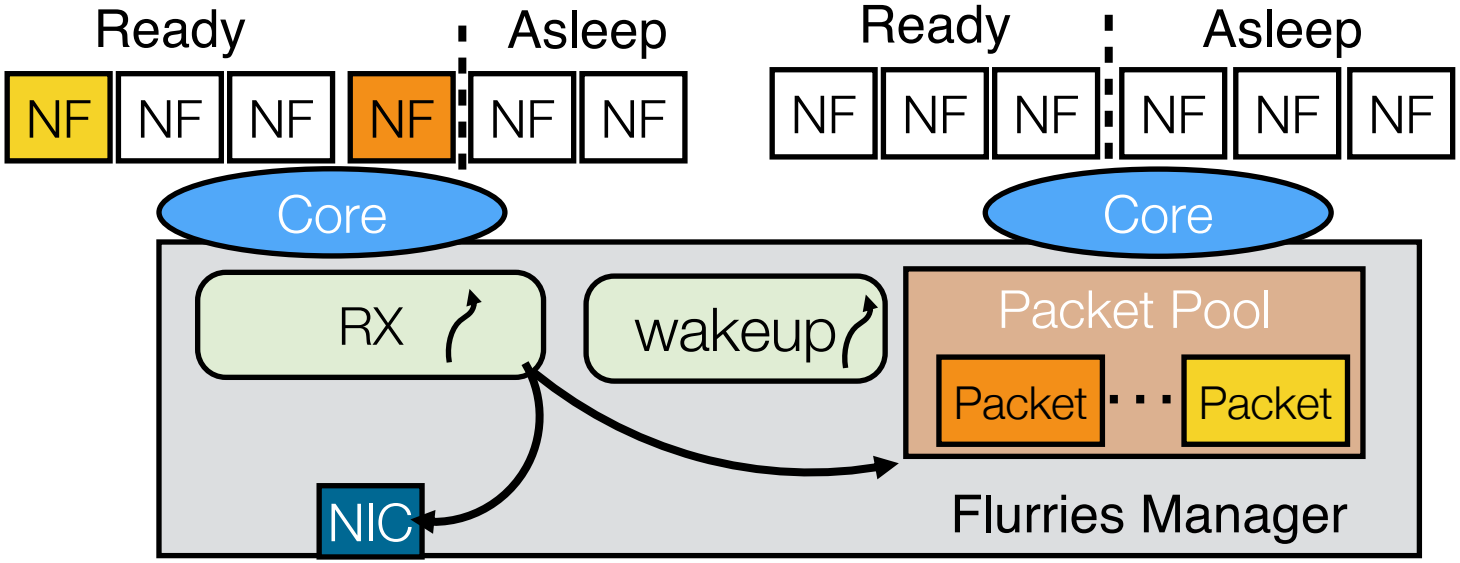
Hybrid Polling and Interrupt

- Manager's RX thread polls the NIC for packets
  - **Polling** retrieves packets with high throughput and low latency
- Wake up thread alerts NFs when there is a batch of packets ready
  - **Interrupts** let many NFs efficiently share a CPU core



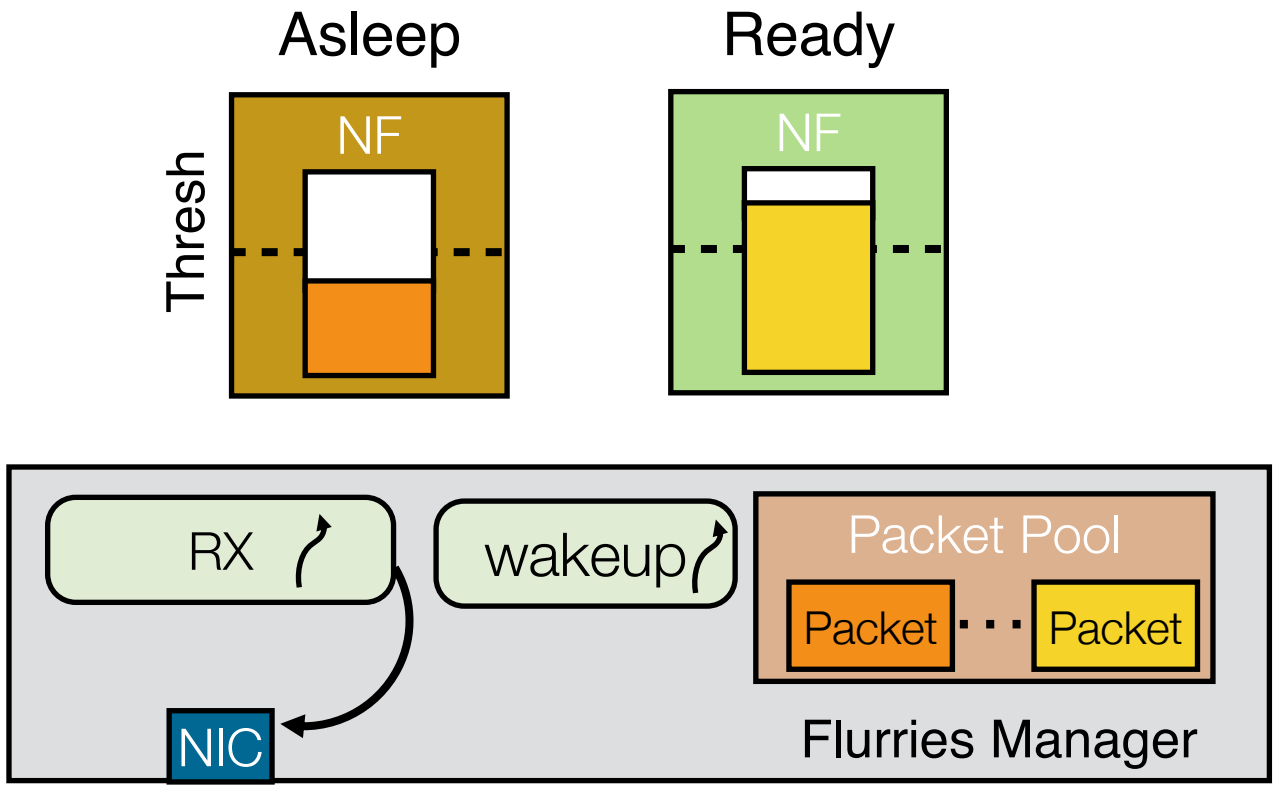
Hybrid Polling and Interrupt

- Manager's RX thread polls the NIC for packets
  - **Polling** retrieves packets with high throughput and low latency
- Wake up thread alerts NFs when there is a batch of packets ready
  - **Interrupts** let many NFs efficiently share a CPU core



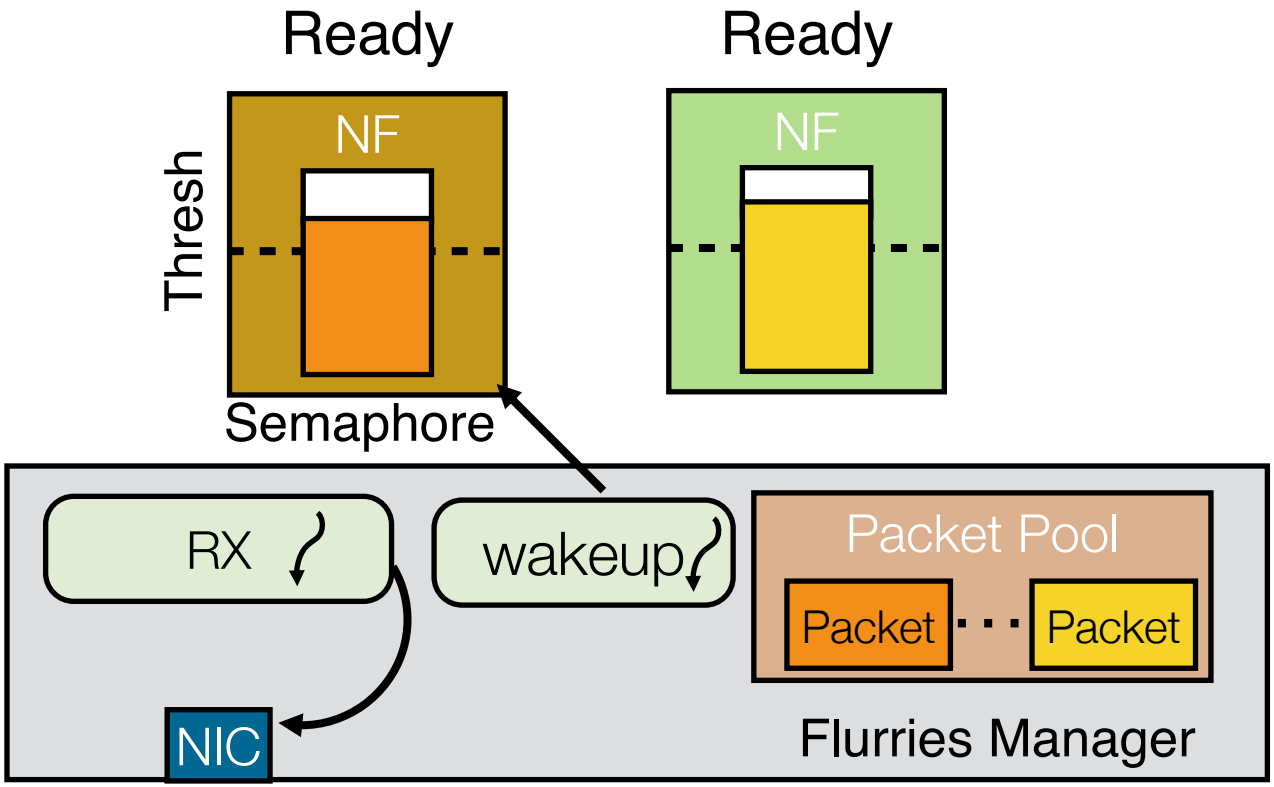
When to Interrupt?

. Wakeup thread checks the queue length of each NF



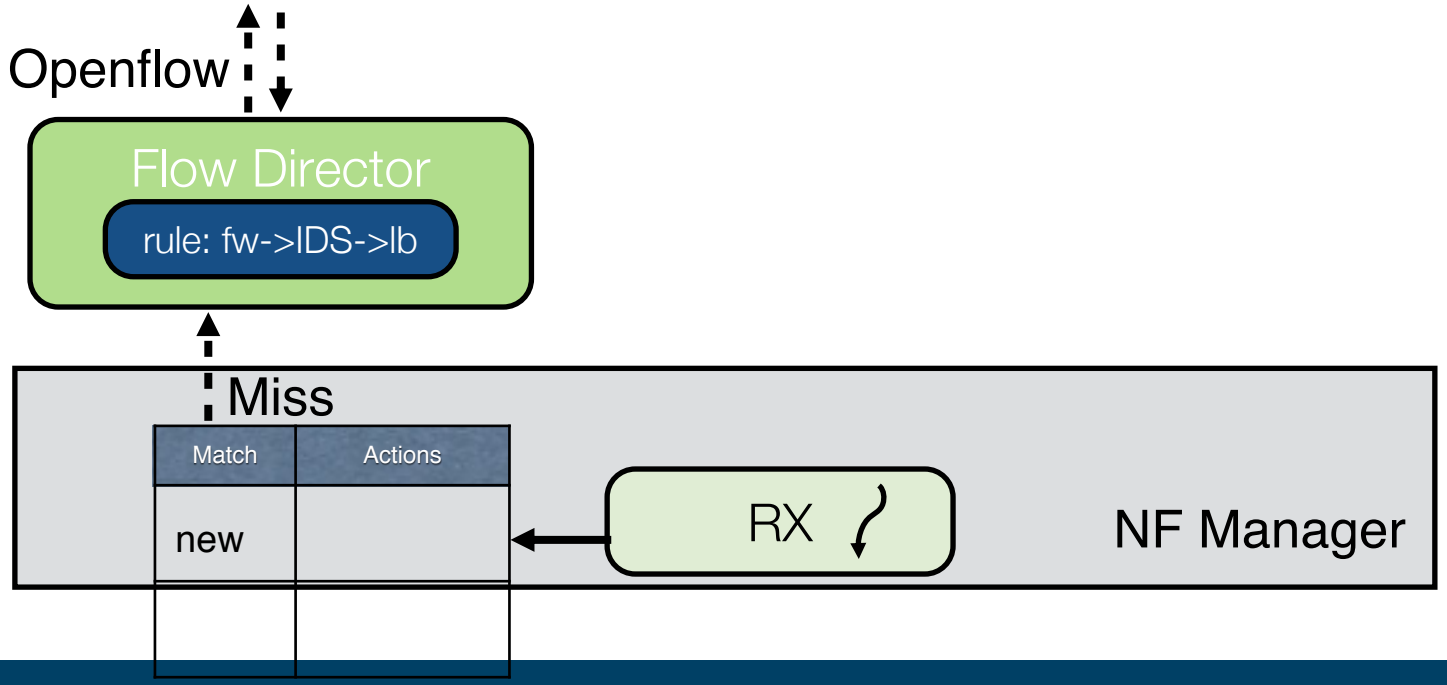
When to Interrupt?

- Wakeup thread checks the queue length of each NF
- Use semaphore to wake up NF to process packets
  - Set threshold carefully



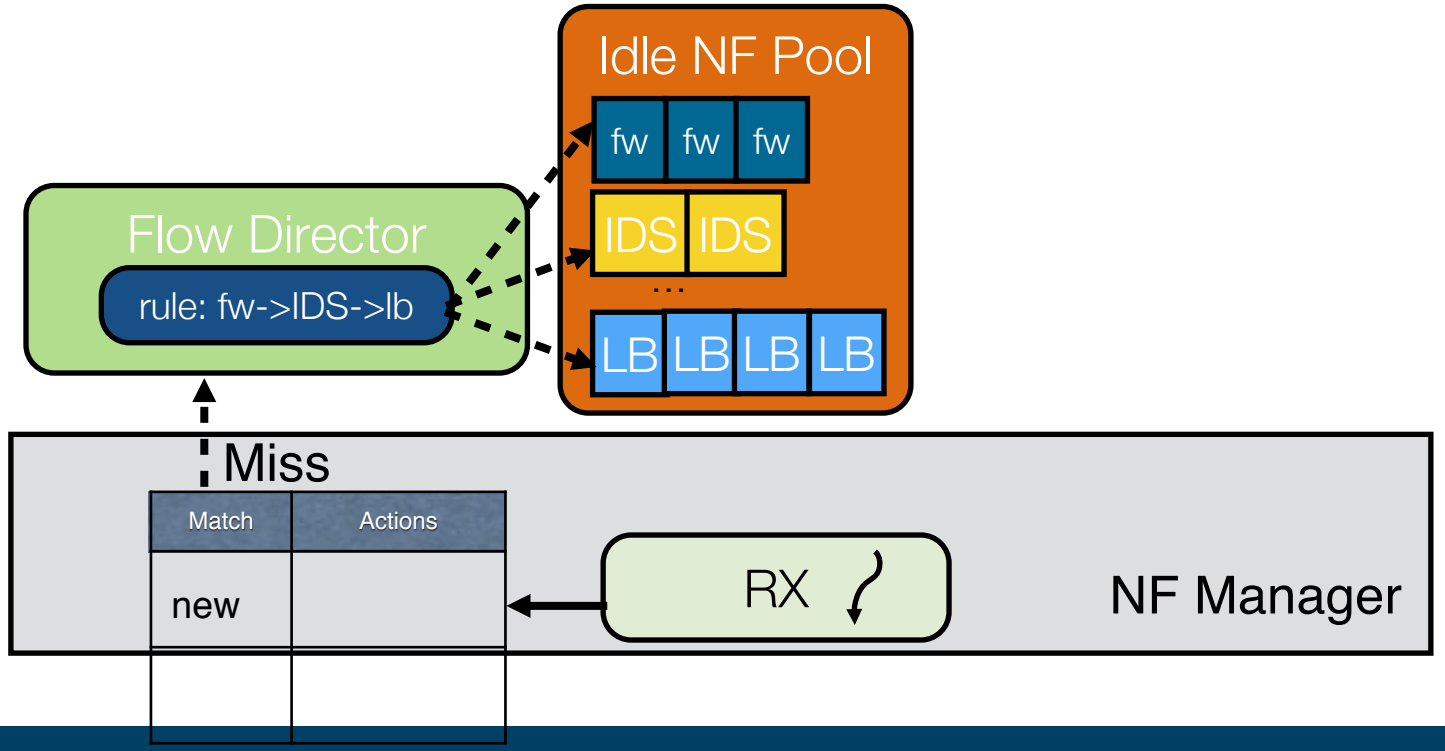
Mapping flows to NFs

- Flow director
  - Check an internal rule set or contact the controller



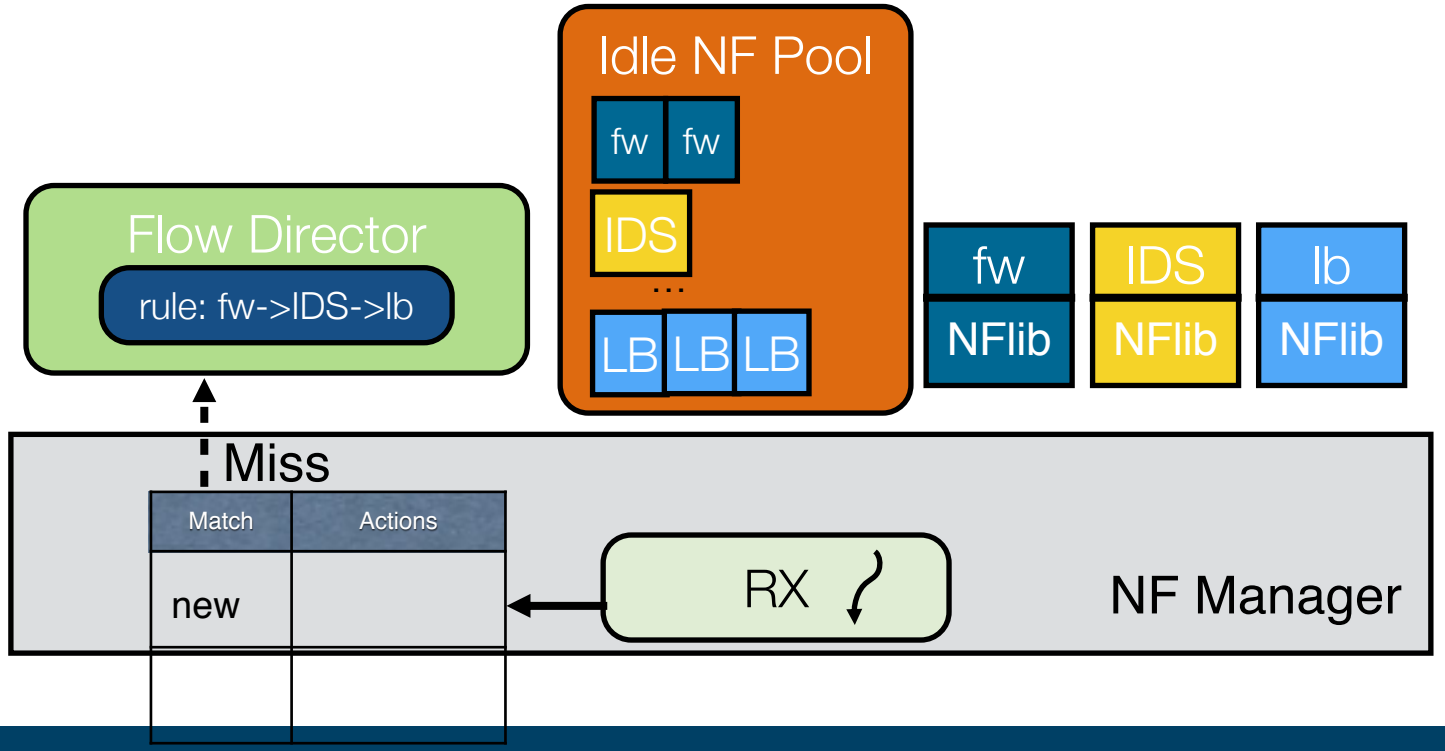
Mapping flows to NFs

. Flow director
  - Check an internal rule set or contact the controller
  - Request free NFs for new service chain



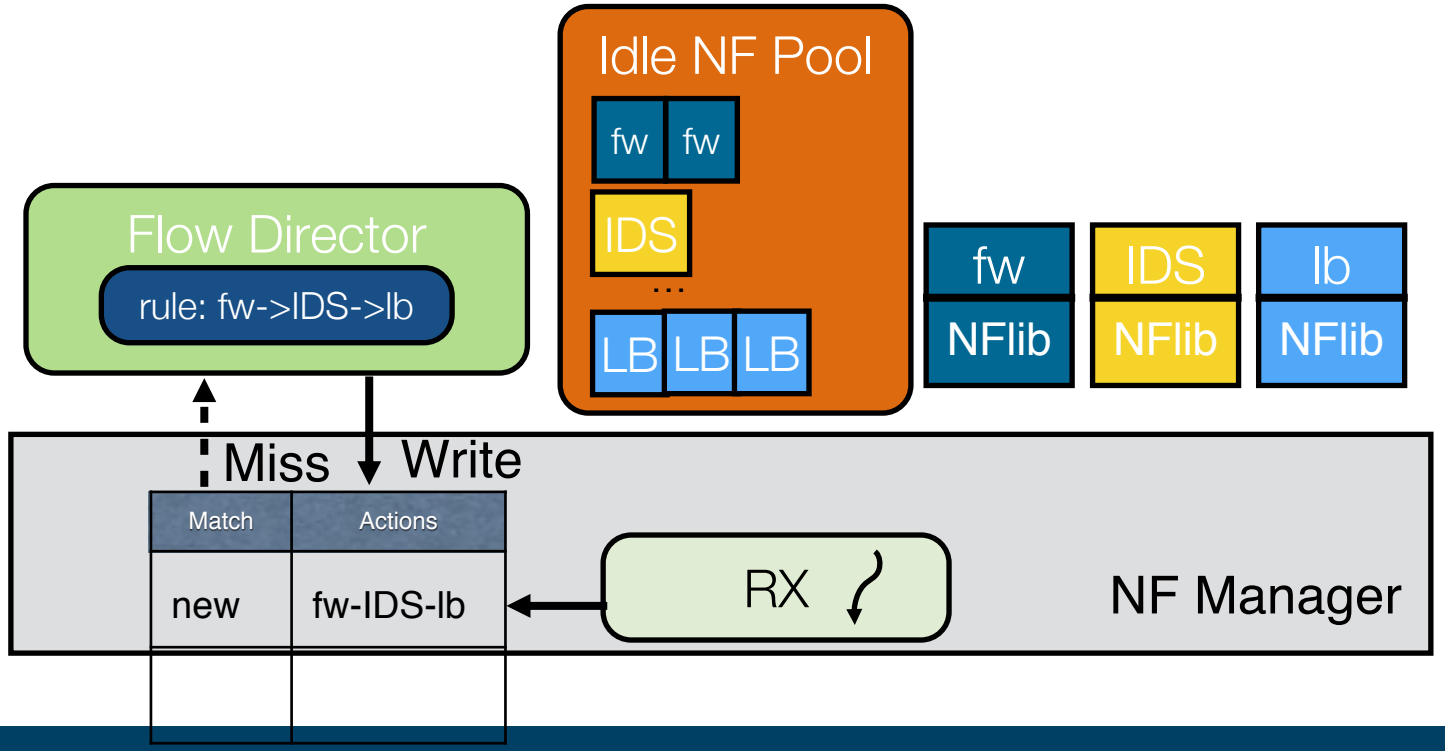
Mapping flows to NFs

. Flow director
  - Check an internal rule set or contact the controller
  - Request free NFs for new service chain



Mapping flows to NFs

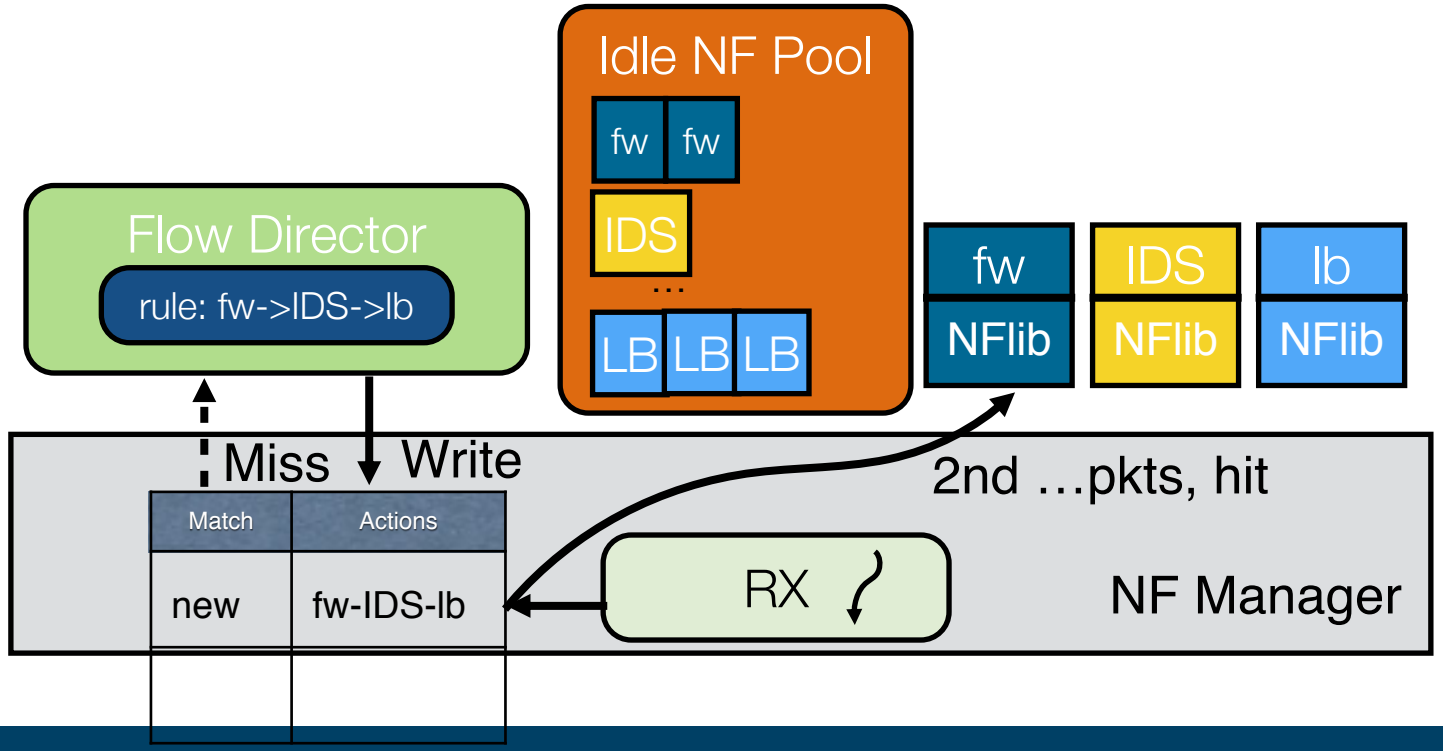
. Flow director
  - Check an internal rule set or contact the controller
  - Request free NFs for new service chain
  - Add new flow entry



Mapping flows to NFs

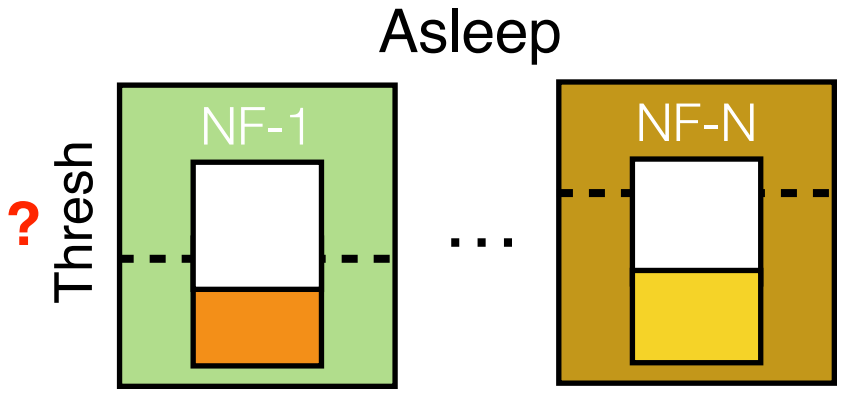
. Flow director

- Check an internal rule set or contact the controller
- Request free NFs for new service chain
- Add new flow entry
- Subsequent packets go directly to NF



Adaptive Wakeup System

- Goal: determine wakeup threshold to mitigate interference caused by throughput vs. latency-sensitive flows, while avoiding starvation



- Problem: Need to be careful about TCP setup and termination for all types of flows – need low latency

Adaptive Wakeup System

- Three-way handshake: wakeup threshold = 1
  - Minimize the overhead of flow director
  - Provide efficient flow startup

Adaptive Wakeup System

- Three-way handshake: wakeup threshold = 1
  - Minimize the overhead of flow director
  - Provide efficient flow startup
- Flow termination: wakeup threshold = 1
  - Release and recycle NF in time

Adaptive Wakeup System

- Three-way handshake: wakeup threshold = 1
  - Minimize the overhead of flow director
  - Provide efficient flow startup
- Flow termination: wakeup threshold = 1
  - Release and recycle NF in time
- running flow : AIMD + timer adaptively adjust threshold
  - Classify flows as latency or bandwidth sensitive
  - AIMD: Additive increase/multiplicative decrease
  - Balance latency and context switch overhead

AIMD + timer

$$thresh(t+1) = \begin{cases} thresh(t) + a, & \text{triggered by} \\ thresh(t) / b, & \text{triggered by} \end{cases}$$

AIMD + timer

$$thresh(t+1) = \begin{cases} thresh(t) + a, & \text{triggered by NF} \\ thresh(t) / b, & \text{triggered by timer} \end{cases}$$

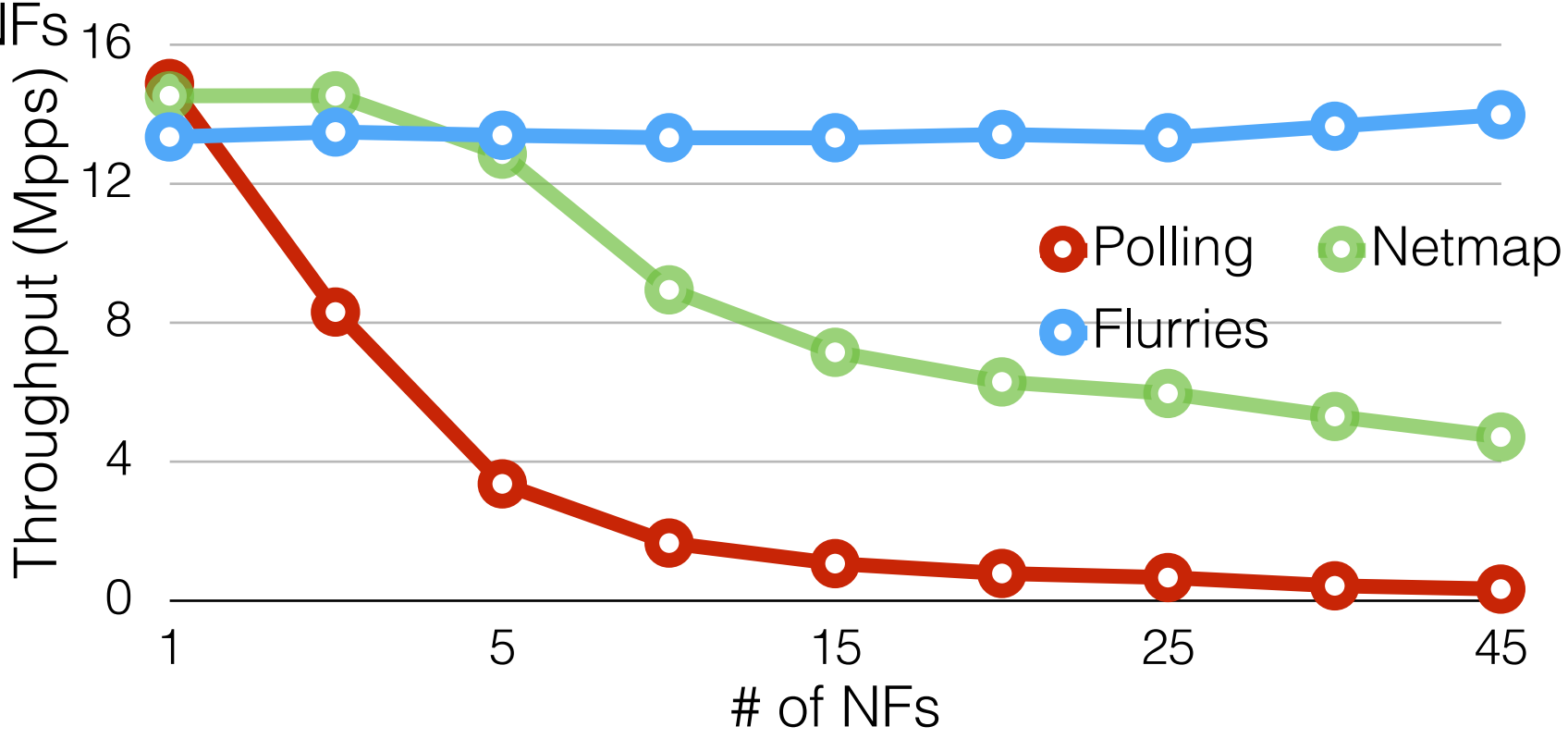
- Mitigate interference, at the same time avoid starvation

	Web Requests/s	Web Latency	Iperf Throughput
RSS	3378	11.8ms	9.3Gbps
Flurries	6650	6.0ms	8.8Gbps

- RSS (receive side scaling): demultiplex flows and balance them across NFs
  - All of flows have fixed and static wakeup threshold
  - Multiple flows may go through one NF

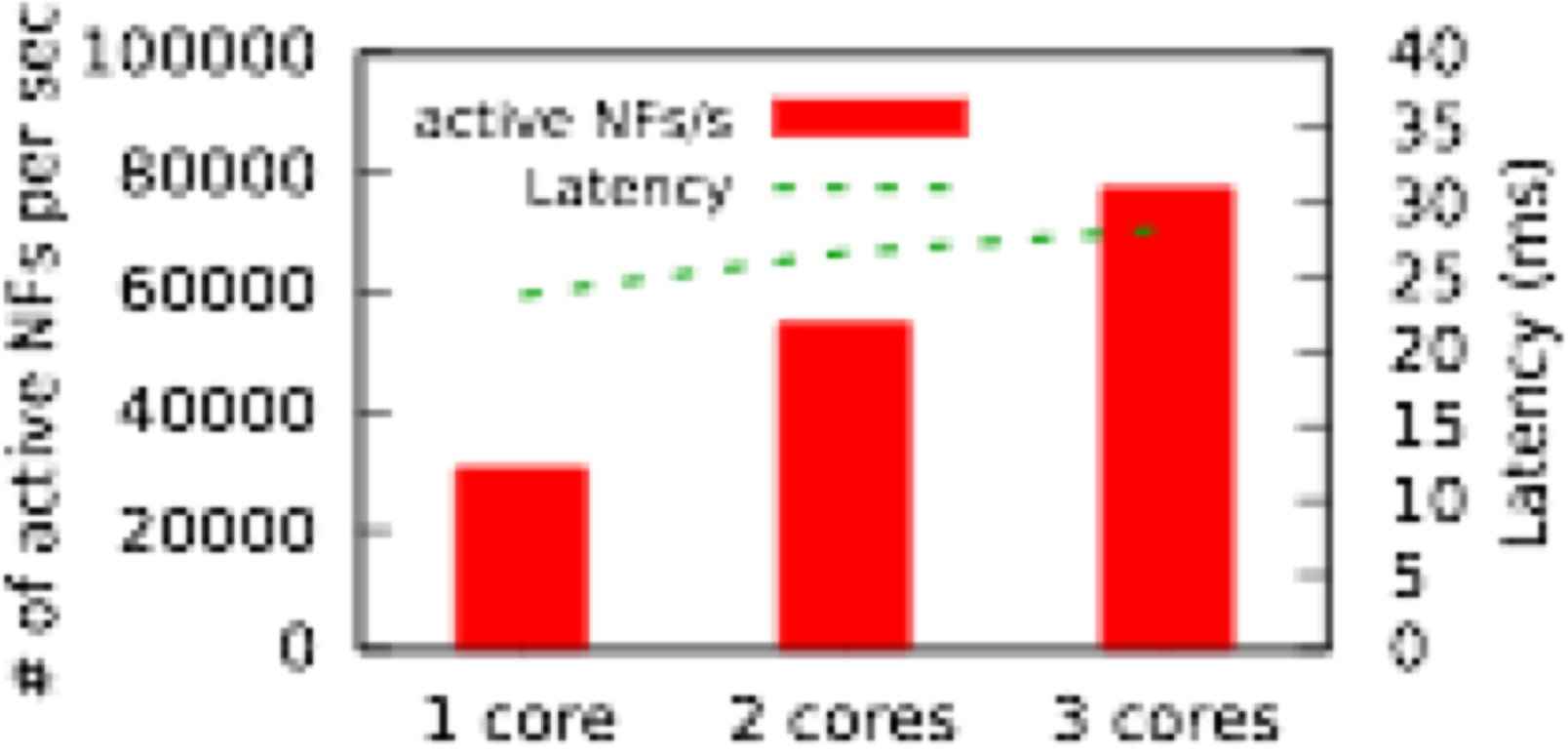
Flurries Performance: Benefit of Hybrid Polling & Interrupts

- Throughput drops as the number of NFs increases on the core for polling and netmap
- Flurries achieves good performance even with large number of NFs



Scale Out

- Run up to 80,000 NFs in a one second interval per host
- Achieve 30Gbps traffic rate and incur minimal added latency to web traffic

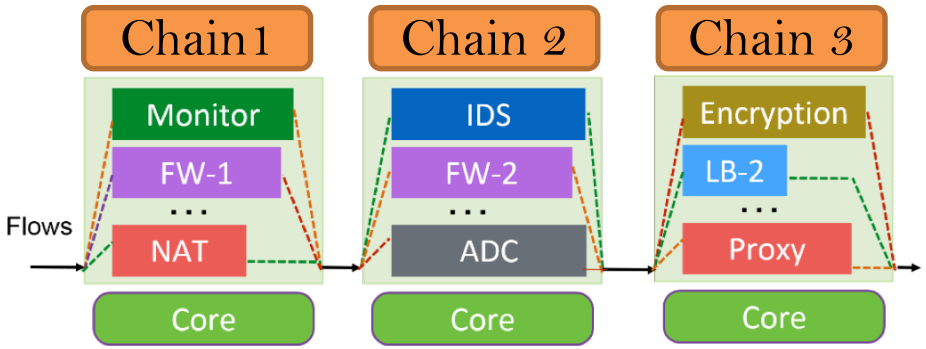


Scalability, Performance, Fairness: NF Scheduling and Backpressure

(ACM Sigcomm 2017)

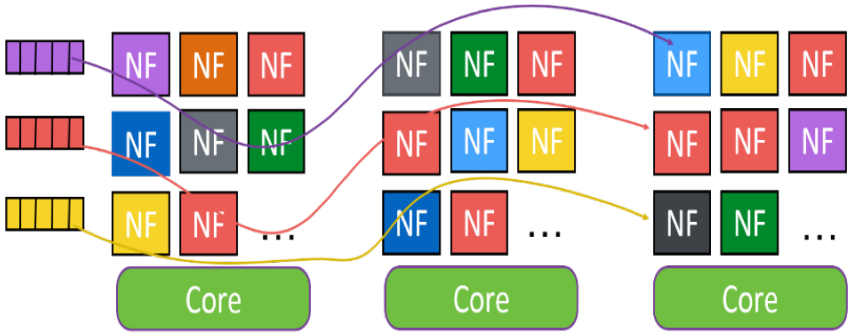
How to address performance and scalability for NFV platform?

- 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.

☑Scalability; ☒Performance?



How to Schedule the NFs to optimize the system utilization?

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

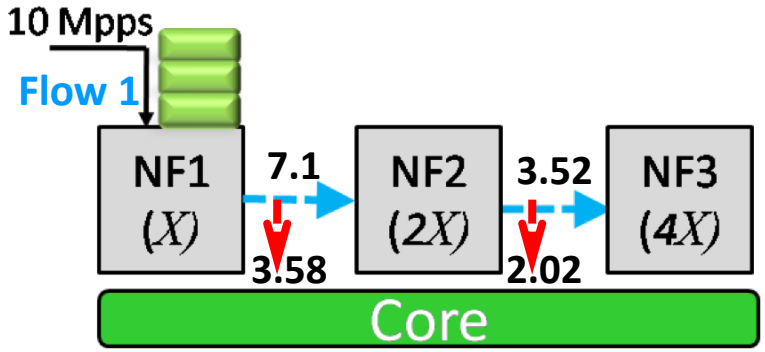
- We want:
  - High **throughput**
  - **Fairness** across NFs
  - Low **latency**
  - Low **context switch** overheads

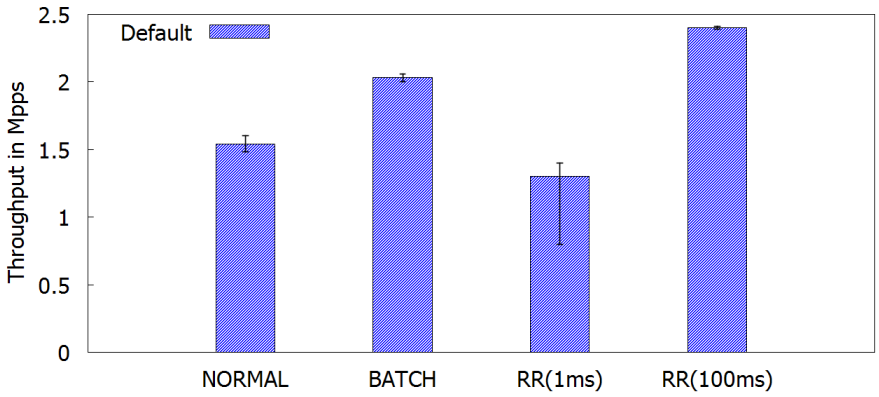
Do existing schedulers perform well?

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

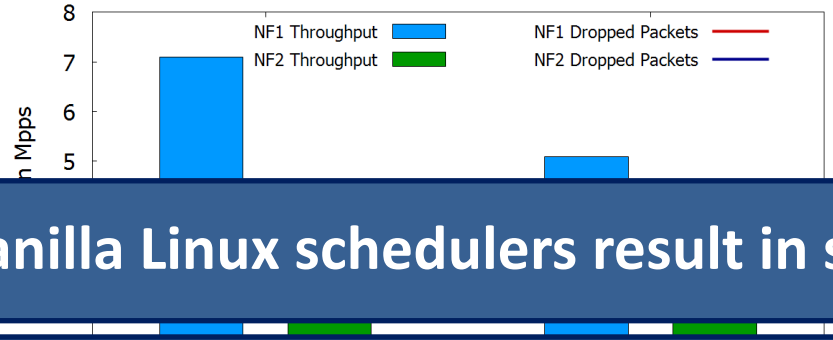
OS Scheduler Characterization (Chain)

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





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



CPU %	NORMAL	BATCH	RR (1ms)	RR (100ms)
NF1	34%	15%	18%	9%
Total	20K/s	2K/s	3K/s	1K/s

Vanilla Linux schedulers result in sub-optimal resource utilization.

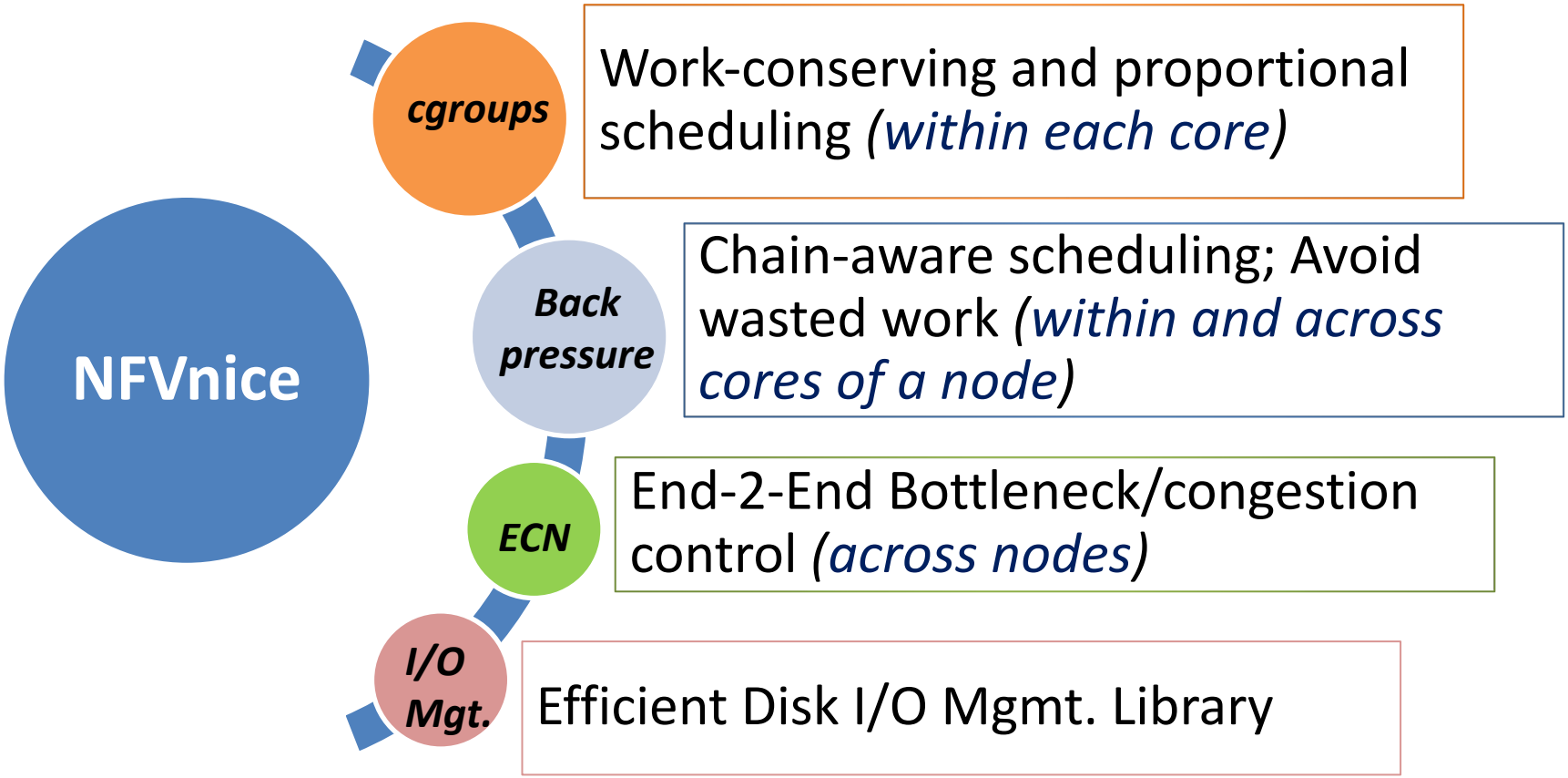
Need the schedulers to be Load, NF characteristic, & chain aware!

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 (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



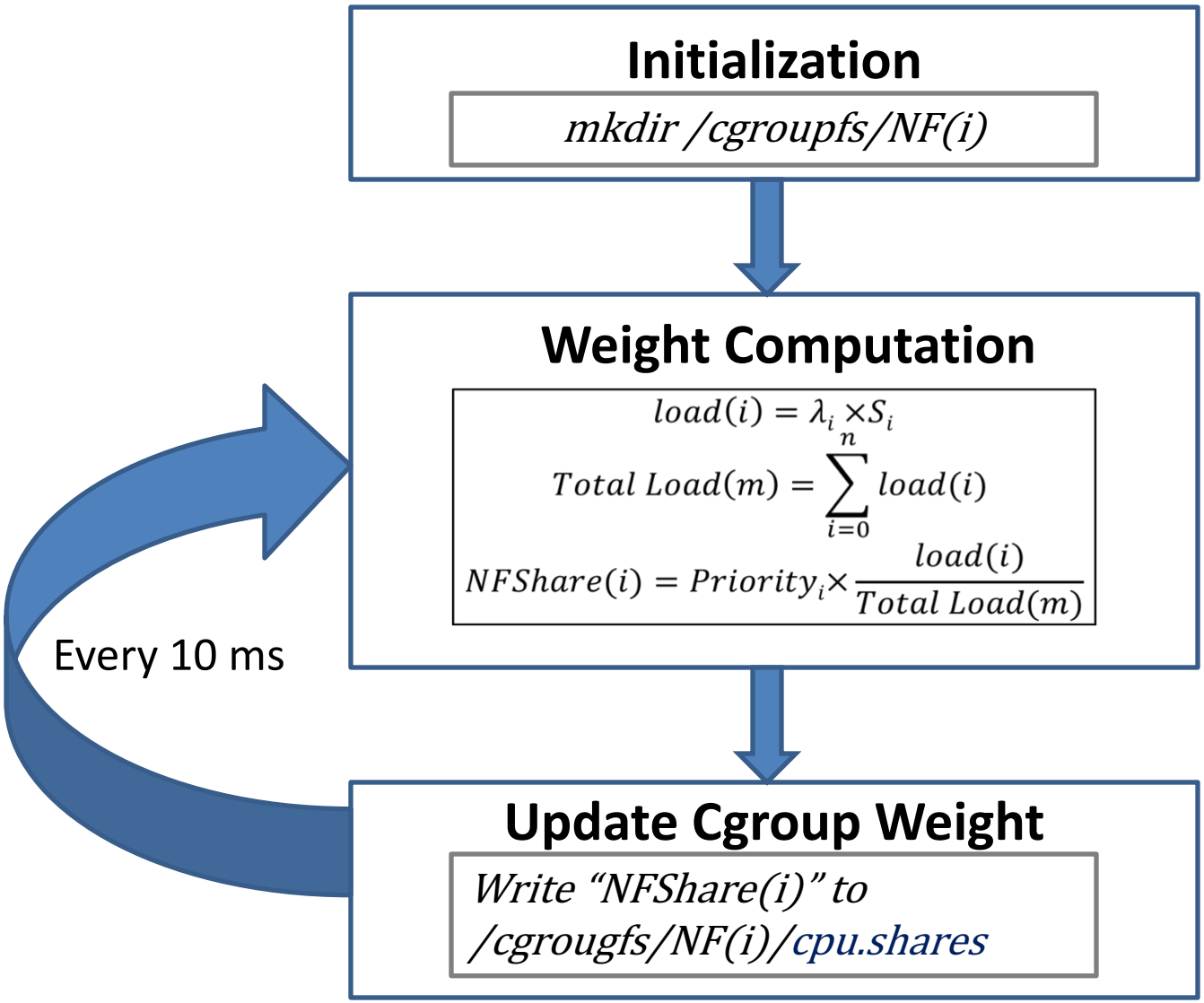
cgroups

Rate-Cost Proportional Fairness

- **What is Rate-Cost Proportional Fairness?**
  - Determines the NFs CPU share by accounting for the:
    - Rate: NF Load (Avg. packet arrival rate, instantaneous Queue length)
    - Compute Cost: 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 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

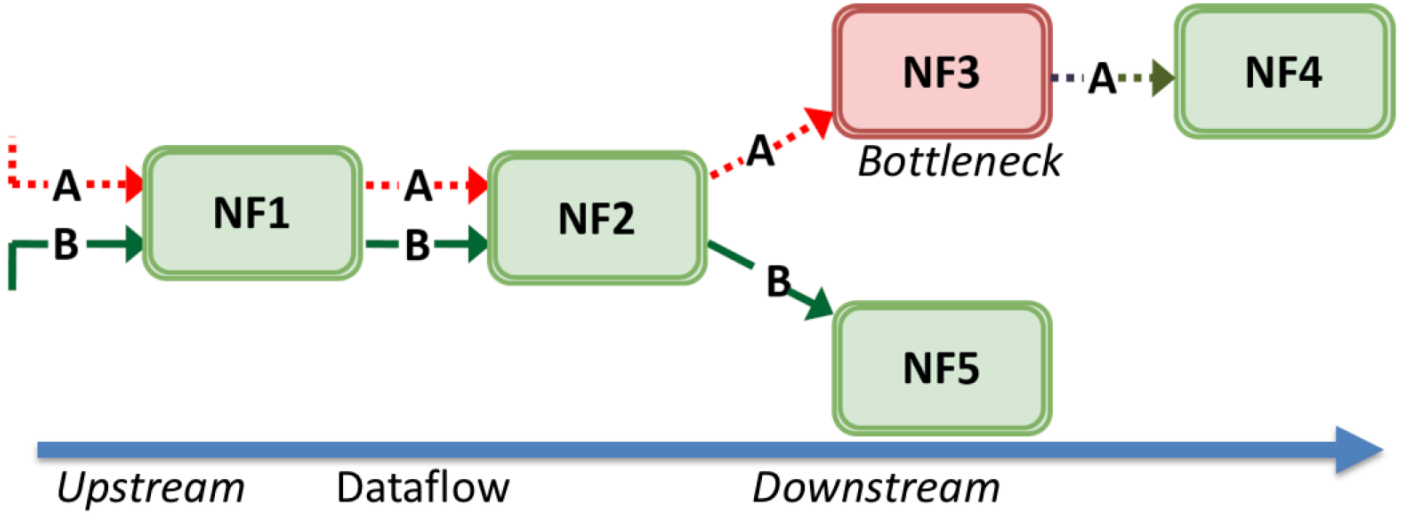
Rate-Cost Proportional Fairness



Back pressure

Backpressure in NF chains

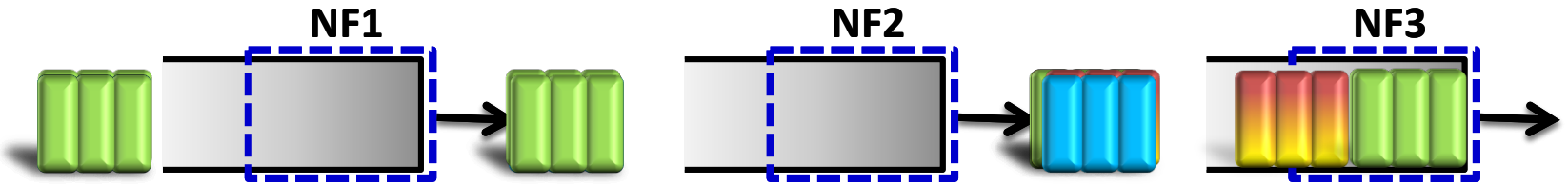
- Selective per chain backpressure marking.

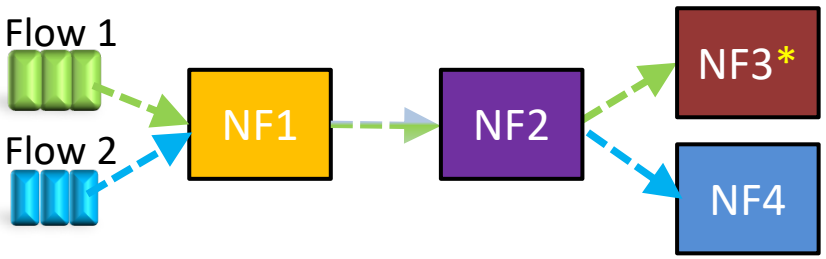


  - Only Flow “A” going through bottleneck NF (NF3) is back pressured and throttled at the upstream source NF1.
  - while Flow “B” is not affected.

Back pressure →

NFVnice Backpressure



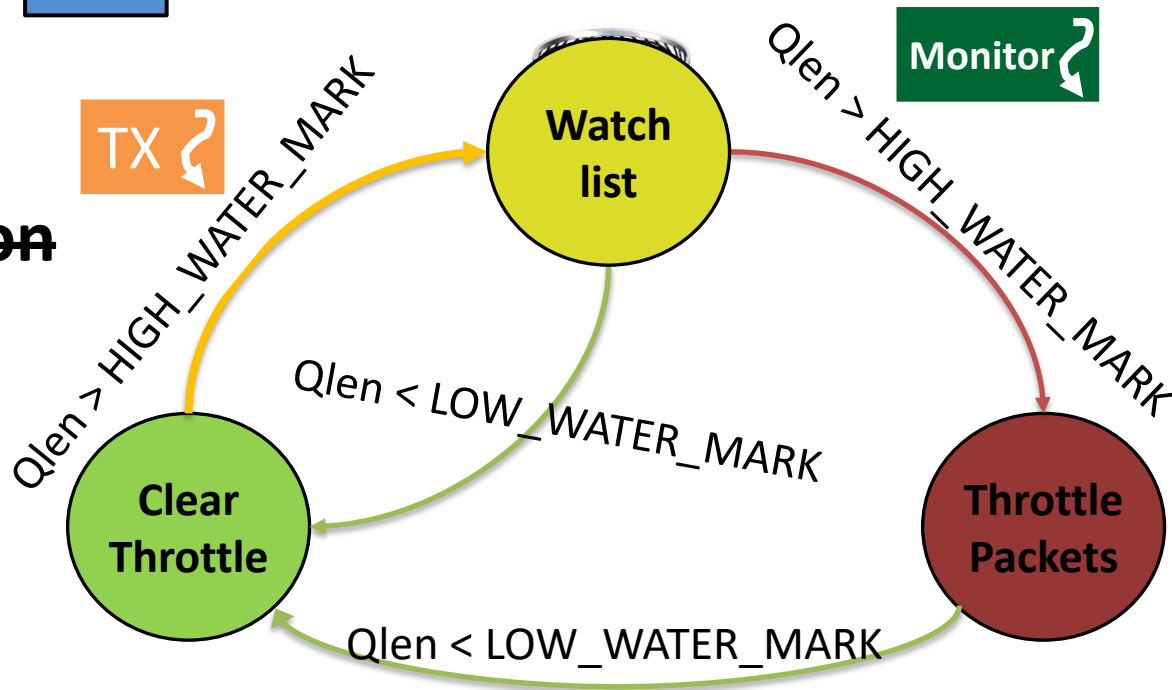


~~Lots of Wasted Work~~

~~Incurs Delay, No Isolation~~

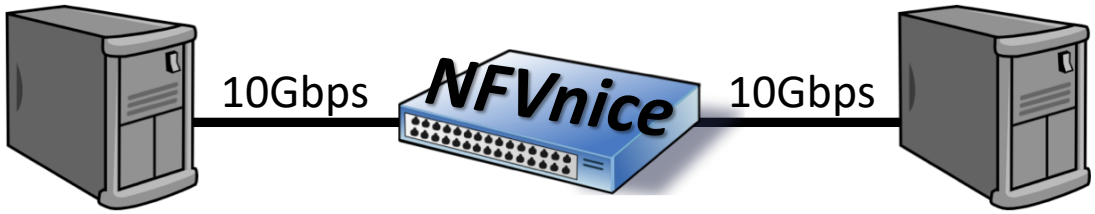
Reacts Instantaneously

Chain Isolation



Evaluation

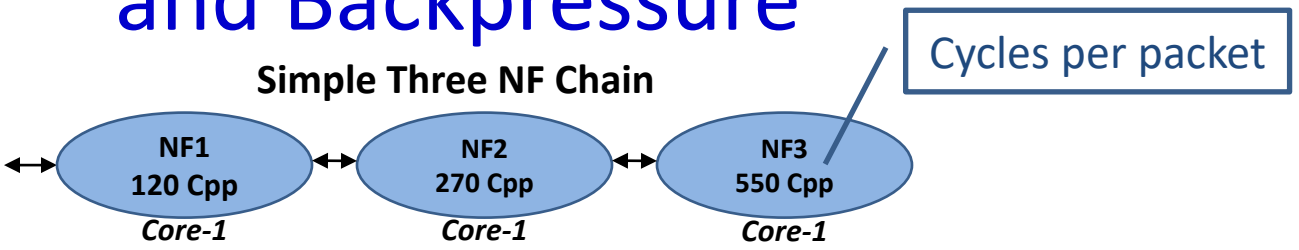
- Testbed:
  - Hardware: 3 Intel Xeon(R) CPU E5-2697, 28 cores @2.6Ghz servers, with dual port 10Gbps DPDK compatible NICs.
  - Software: Linux kernel 3.19.0-39-lowlatency profile.
  - NFVnice: built on top of OpenNetVM.

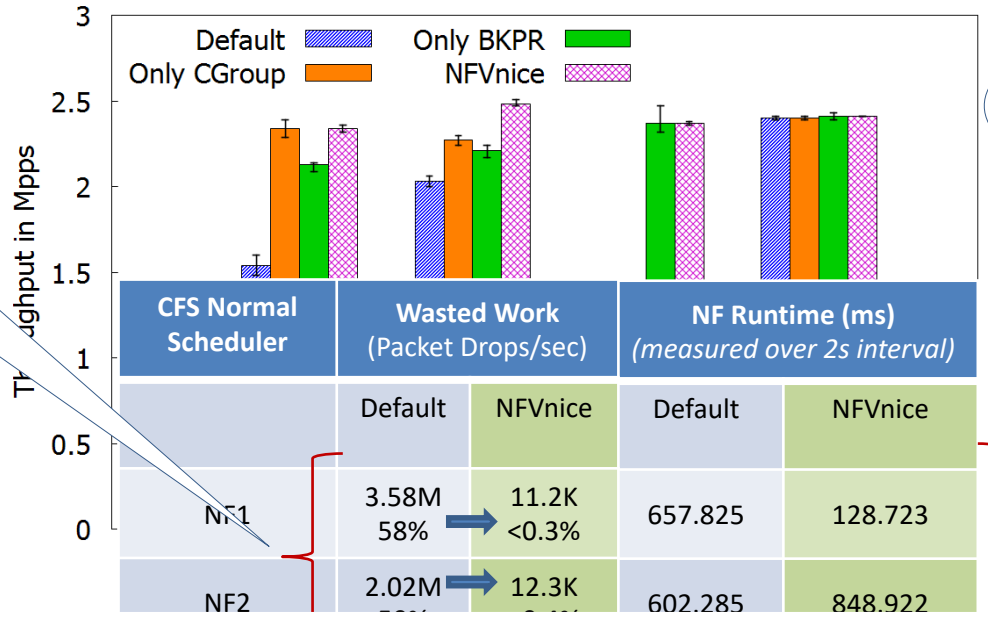


- Traffic:
  - Pktgen and Moongen: Line rate traffic (64 byte packets).
  - Iperf: TCP flows.

- 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





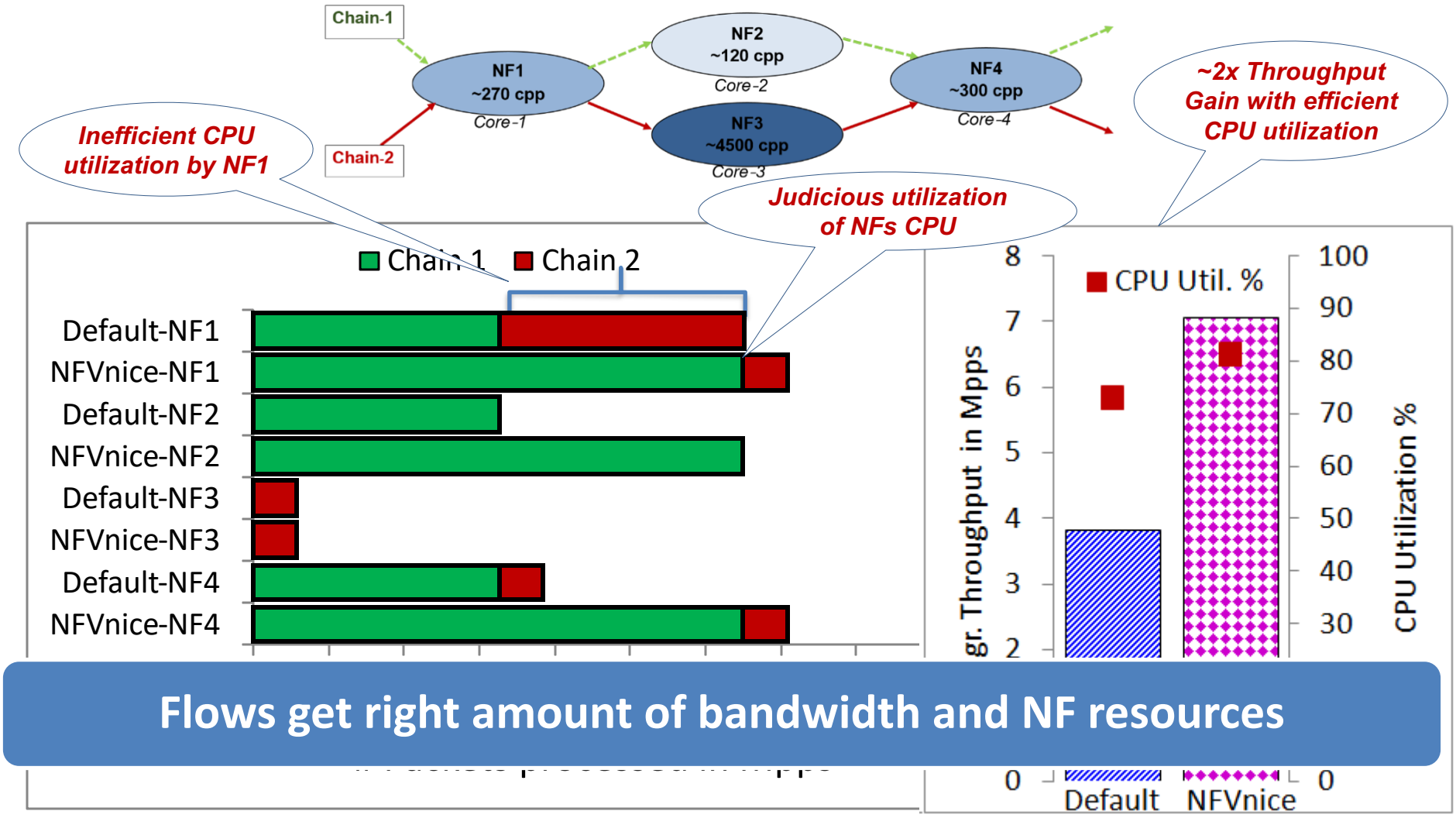
CFS Normal Scheduler	Wasted Work (Packet Drops/sec)		NF Runtime (ms) *(measured over 2s interval)*	
	Default	NFVnice	Default	NFVnice
NF1	3.58M 58%	11.2K <0.3%	657.825	128.723
NF2	2.02M	12.3K	602.285	848.922

Significant Reduction in Wasted Work!

CPU Allocation α **to Computation Cost**

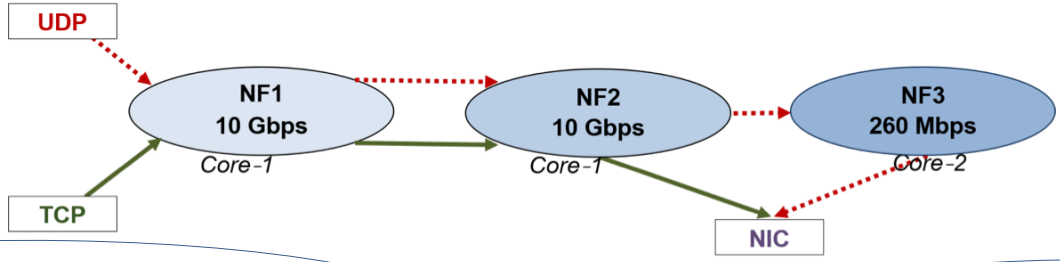
NFVnice improves throughput for all kernel schedulers.

Performance + Resource Utilization



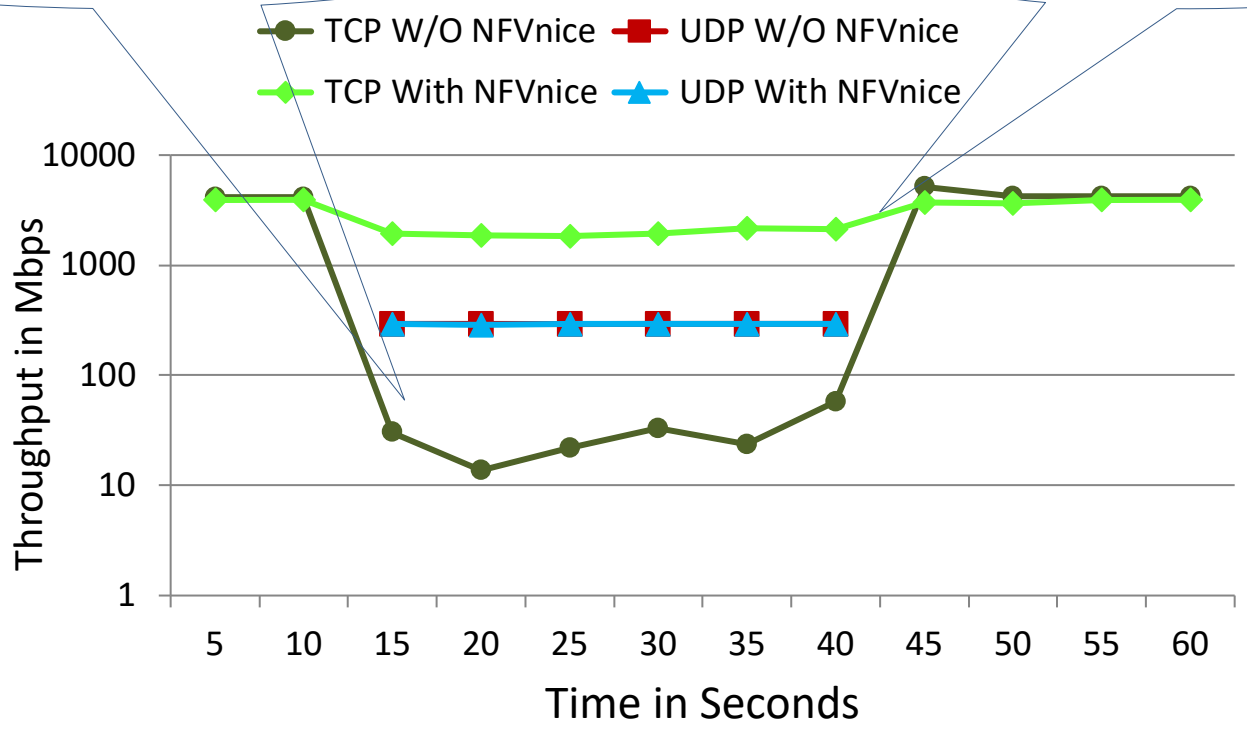
Flows get right amount of bandwidth and NF resources

TCP and UDP Isolation



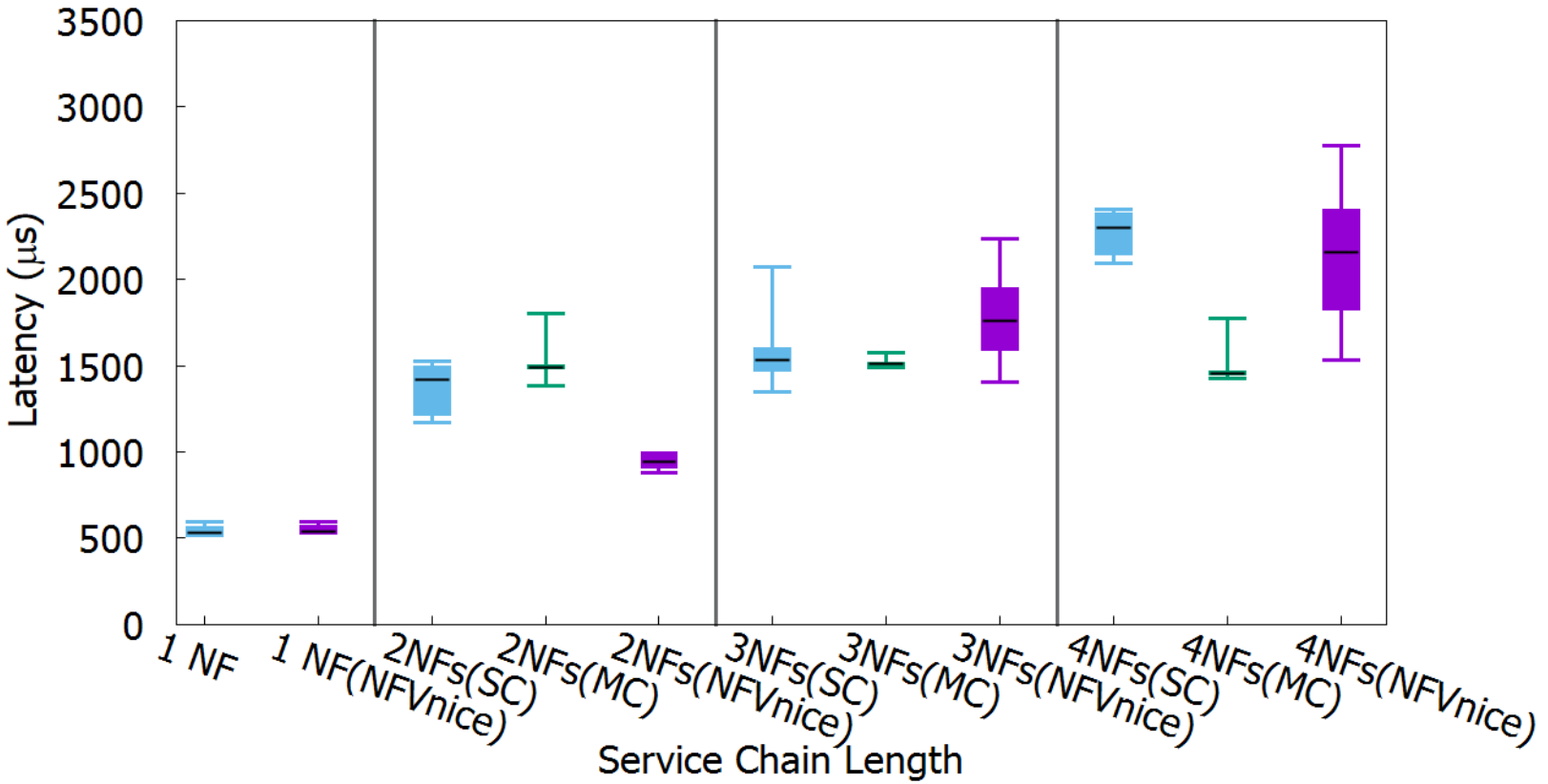
TCP affected by UDP flows!
Wastage of NF1,NF2 bandwidth

Effectively isolates UDP and TCP flows



LATENCY: SERVICE CHAINS MULTICORE VS NFVNICE

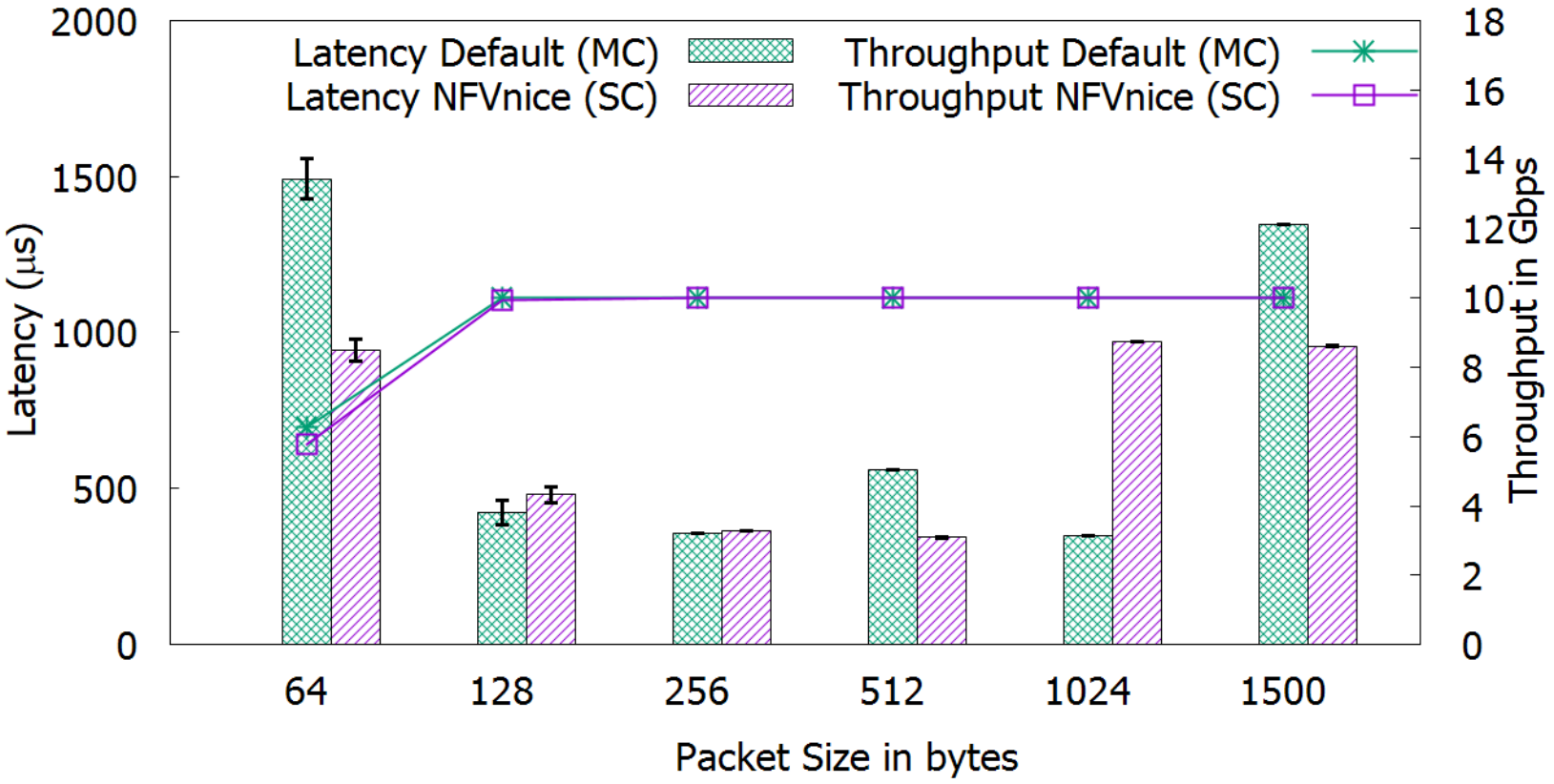
- We study the packet latency through OpenNetVM – comparing the case of having a single NF per core (called Multicore (MC)), a single core (shared across NFs with the default Linux CFS scheduler (SC) and NFVnice). 64 byte packet @ line rate.



- NFVnice doesn't introduce additional overhead. Small chains: NFVnice has lower latency because of caching benefits. Larger chains, when CPU processing becomes the bottleneck can take advantage of multiple cores

LATENCY AND THROUGHPUT FOR NFVNICE

- We study the packet latency through OpenNetVM –two NF in a chain (two cores(MC) vs. NFVnice). Varying packet size @ line rate.



Overhead of NFVnice is minimal compared to using the default OS scheduling.

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
  - Customization of services on a per-flow basis: `Flurries' – activate per-flow NFs and service chains

UCR

Getting OpenNetVM

- Source code and NSF CloudLab images at

  http://sdnfv.github.io/

UCR