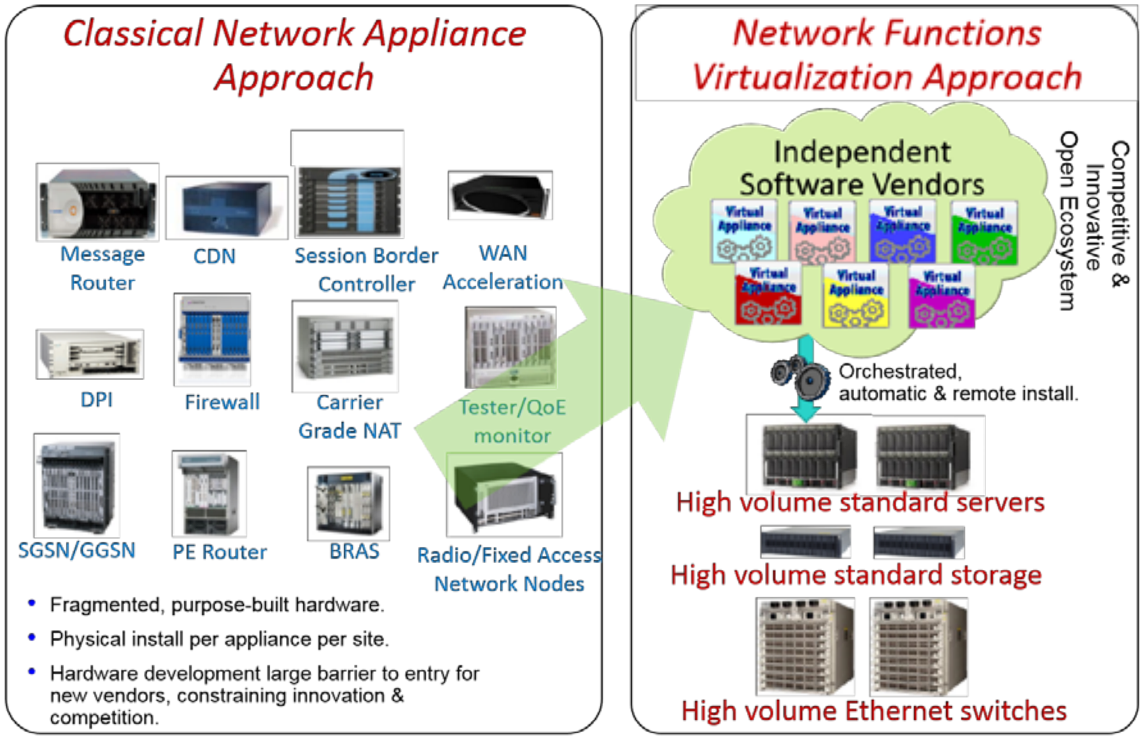
NFVnice: Dynamic Backpressure and Scheduling for NFV Service Chains

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

[1]University of Göttingen
[2]George Washington University
[3]IBM T J Watson Research Center
[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.

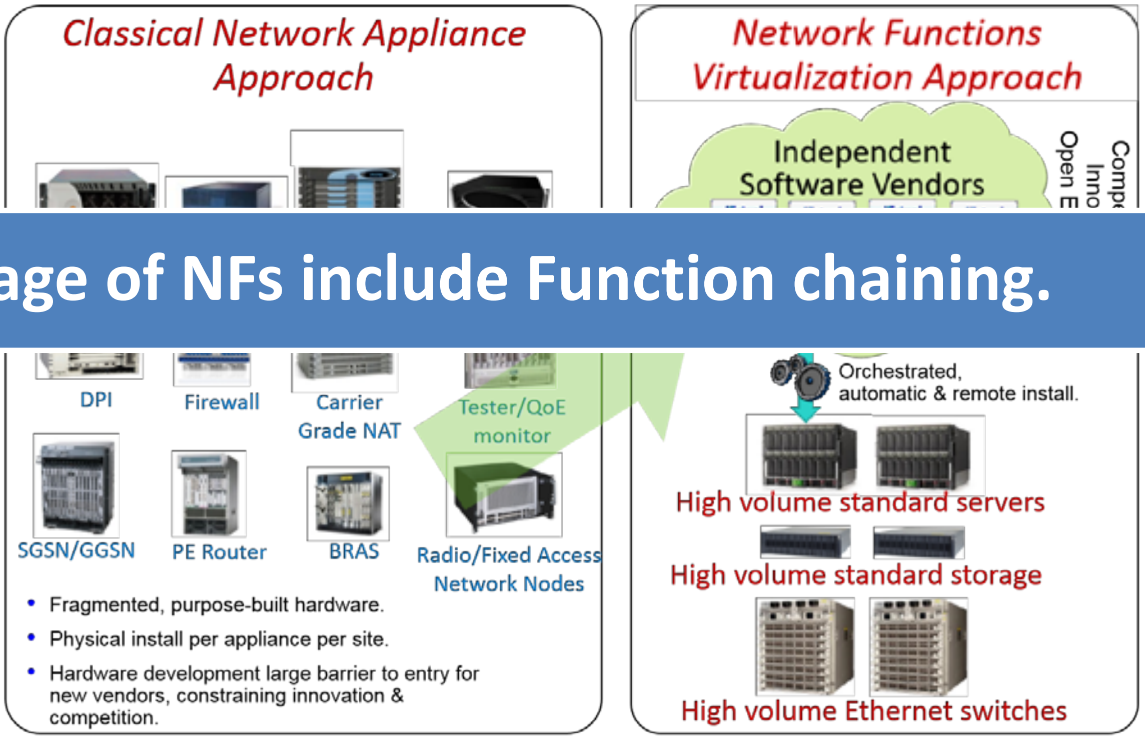

Classical Network Appliance Approach
Message Router
CDN
Session Border Controller
WAN Acceleration
DPI
Firewall
Carrier Grade NAT
Tester/QoE monitor
SGSN/GGSN
PE Router
BRAS
Radio/Fixed Access Network Nodes
Network Functions Virtualization Approach
Independent Software Vendors
Virtual Appliance
Competitive & Innovative Open Ecosystem
Orchestrated, automatic & remote install.
High volume standard servers
High volume standard storage
High volume Ethernet switches

- Fragmented, purpose-built hardware.
- Physical install per appliance per site.
- Hardware development large barrier to entry for new vendors, constraining innovation & competition.

*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.



Classical Network Appliance Approach
DPI
Firewall
Carrier Grade NAT
Tester/QoE monitor
SGSN/GGSN
PE Router
BRAS
Radio/Fixed Access Network Nodes
Network Functions Virtualization Approach
Independent Software Vendors
Orchestrated, automatic & remote install.
High volume standard servers
High volume standard storage
High volume Ethernet switches

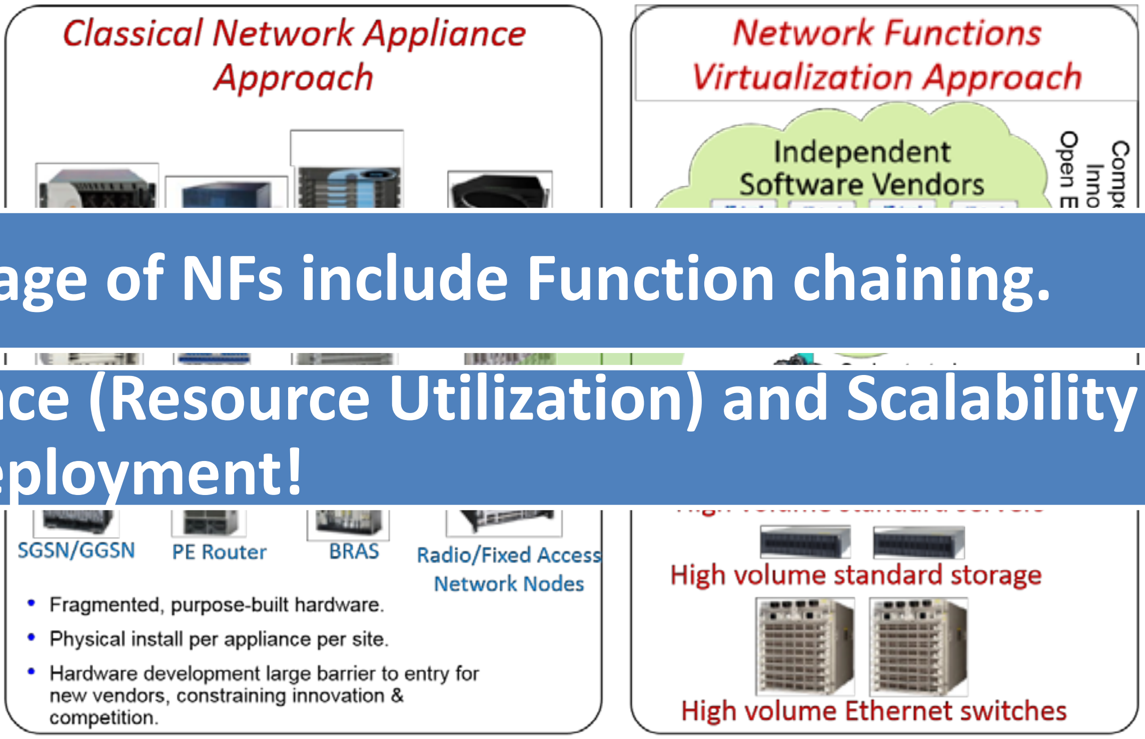
- Fragmented, purpose-built hardware.
- Physical install per appliance per site.
- Hardware development large barrier to entry for new vendors, constraining innovation & competition.

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.


Classical Network Appliance Approach
Network Functions Virtualization Approach
Independent Software Vendors
SGSN/GGSN
PE Router
BRAS
Radio/Fixed Access Network Nodes
High volume standard storage
High volume Ethernet switches

- Fragmented, purpose-built hardware.
- Physical install per appliance per site.
- Hardware development large barrier to entry for new vendors, constraining innovation & competition.

Typical usage of NFs include Function chaining.

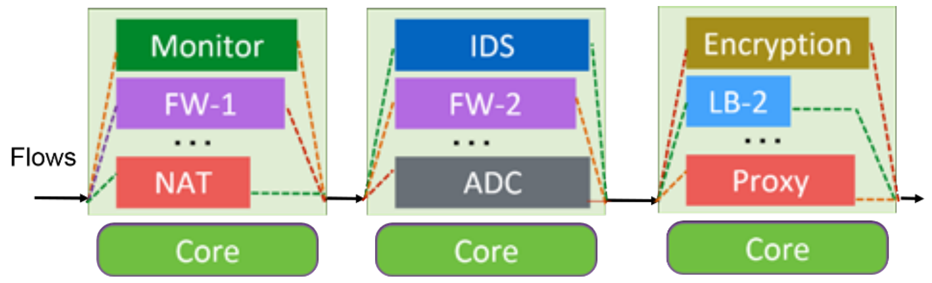
Performance (Resource Utilization) and Scalability are the key for NFV deployment!

*source: ETSI NFV June'13

How to address performance and scalability for NFV platform?

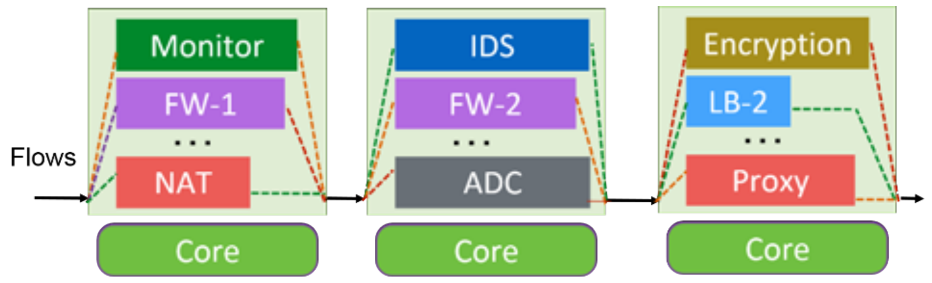
How to address performance and scalability for NFV platform?

- 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?

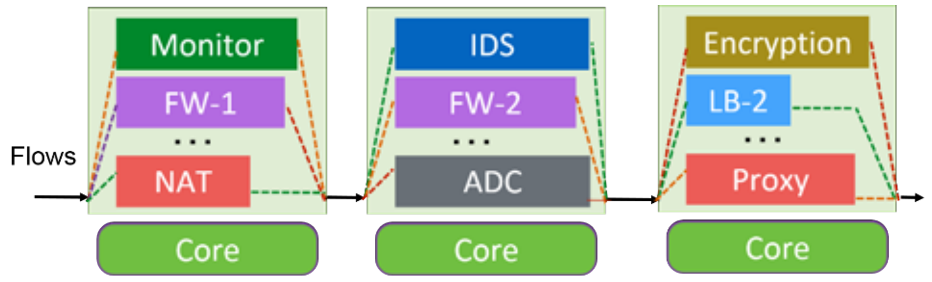
- Consolidation approaches
  - E2 [SOSP '15], NetBricks [OSDI'16]:
  - Consolidate NFs of a chain on single core.



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

How to address performance and scalability for NFV platform?

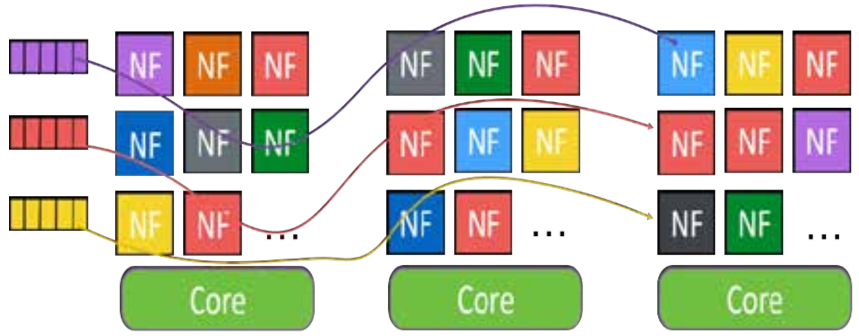
- Consolidation approaches
  - E2 [SOSP '15], NetBricks [OSDI'16]:
  - Consolidate NFs of a chain on single core.



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

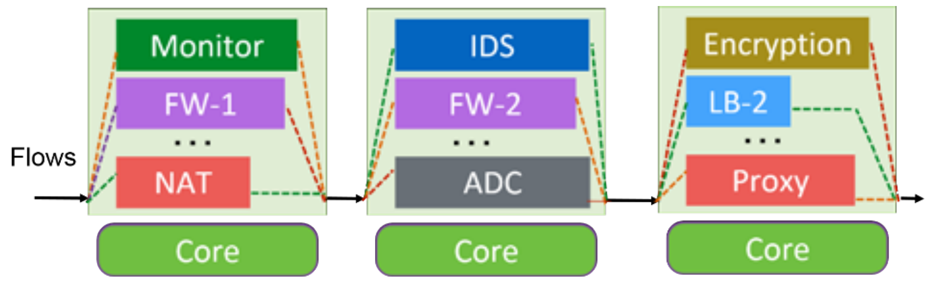
- Multiplexing approach:
  - Flurries [CoNext '16], ClickOS [NSDI'14]
  - Multiplex NFs on same core.

Performance?, Scalability



How to address performance and scalability for NFV platform?

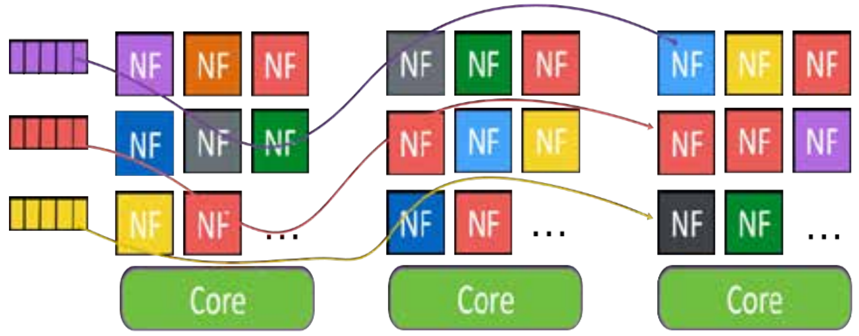
- Consolidation approaches
  - E2 [SOSP '15], NetBricks [OSDI'16]:
  - Consolidate NFs of a chain on single core.



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

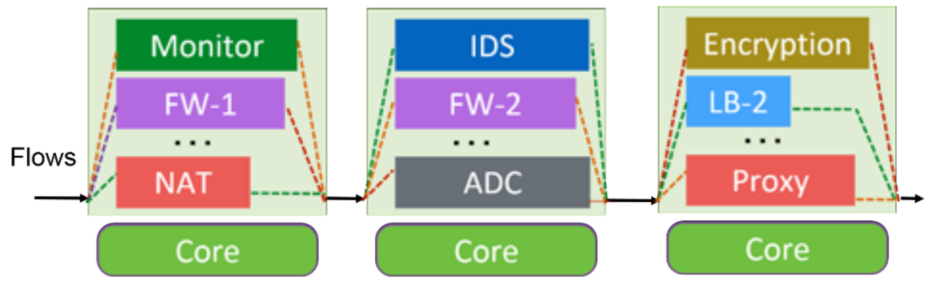
- Multiplexing approach:
  - Flurries [CoNext '16], ClickOS [NSDI'14]
  - Multiplex NFs on same core.

Performance?, Scalability



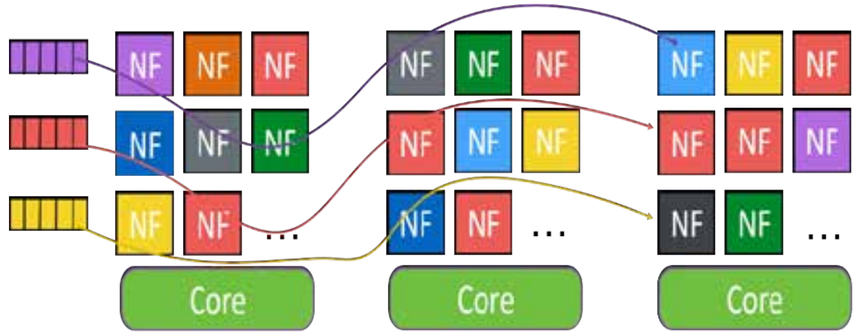
How to address performance and scalability for NFV platform?

- 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.

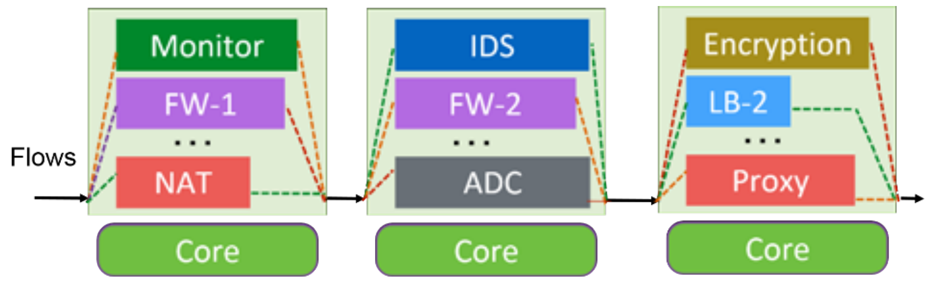


Performance?, Scalability

Then, how to Schedule the NFs to optimize the performance?

How to address performance and scalability for NFV platform?

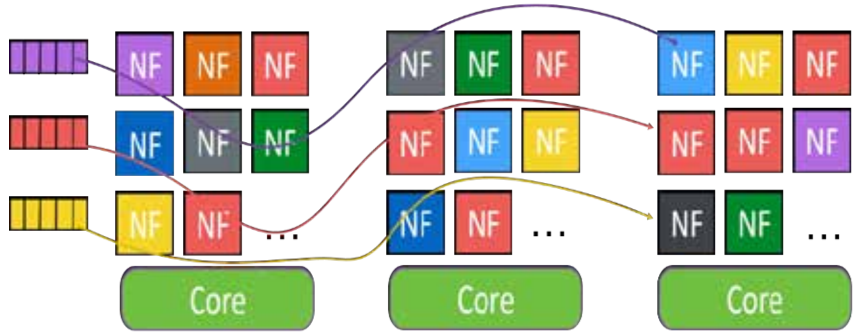
- Consolidation approaches
  - E2 [SOSP '15], NetBricks [OSDI'16]:
  - Consolidate NFs of a chain on single core.



- 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:

Do existing schedulers perform well?

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

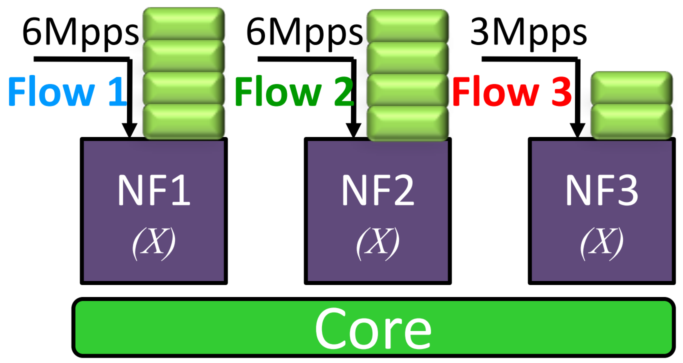
Real Time Scheduler
- Round Robin
- FIFO

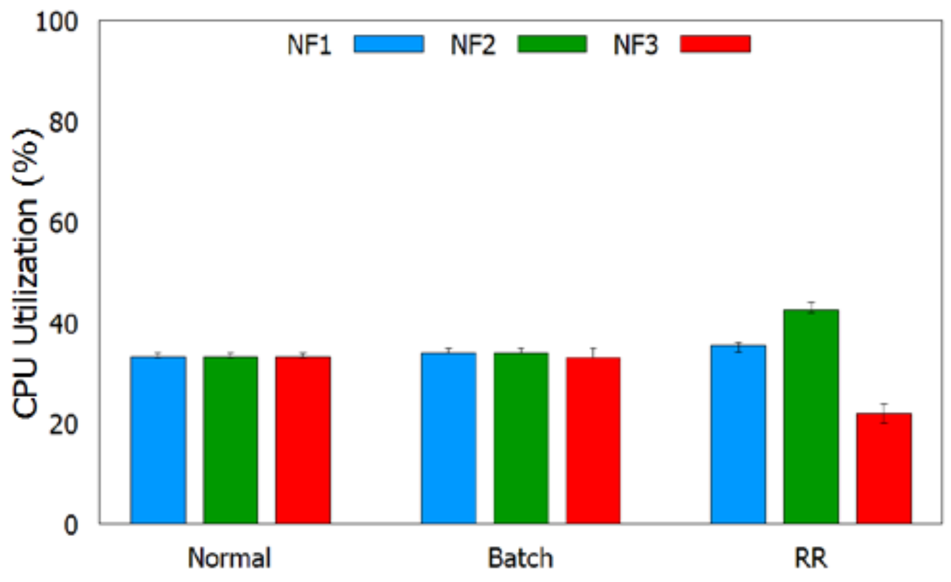
- Time slice
- Millisecond granularity

Do existing schedulers perform well?

OS Scheduler Characterization (Load)

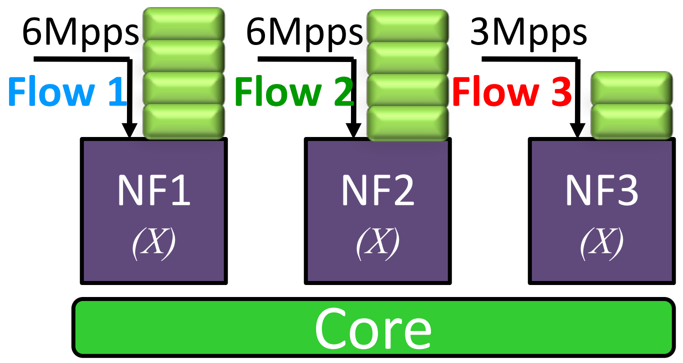
3 Homogeneous NFs running on a same core with offered load 2:2:1.

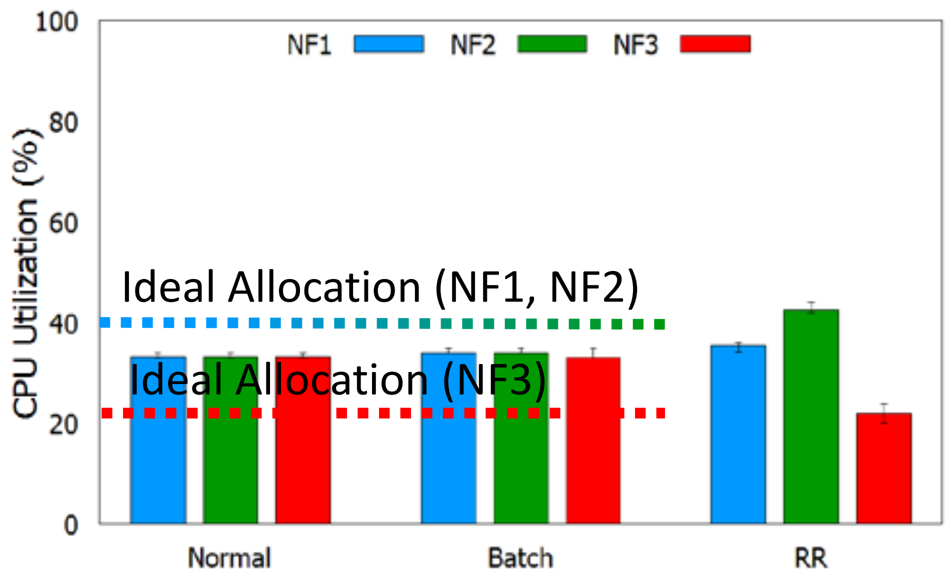




OS Scheduler Characterization (Load)

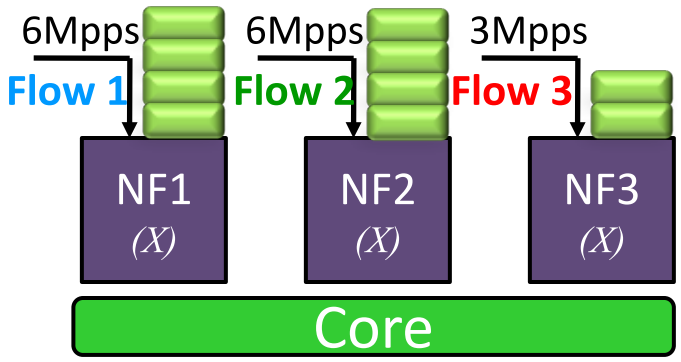
3 Homogeneous NFs running on a same core with offered load 2:2:1.

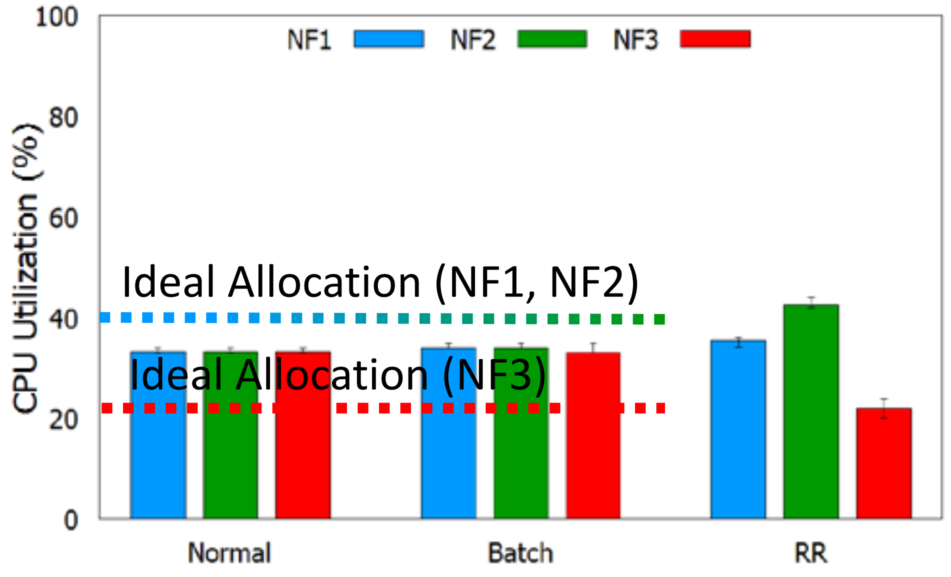


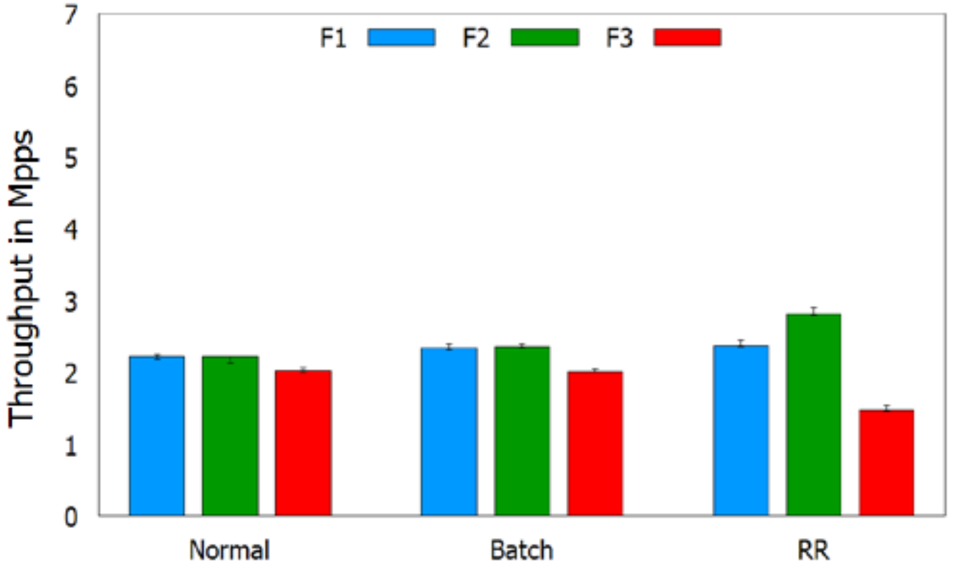


OS Scheduler Characterization (Load)

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

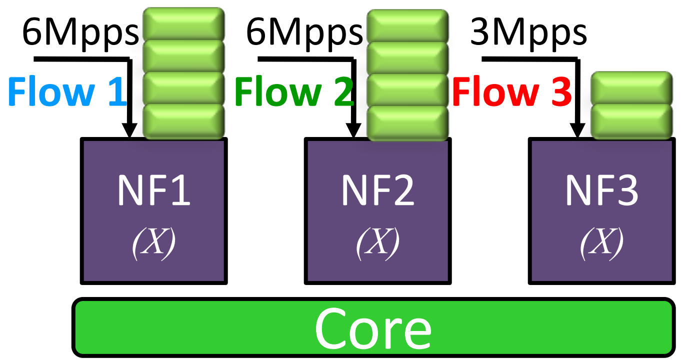


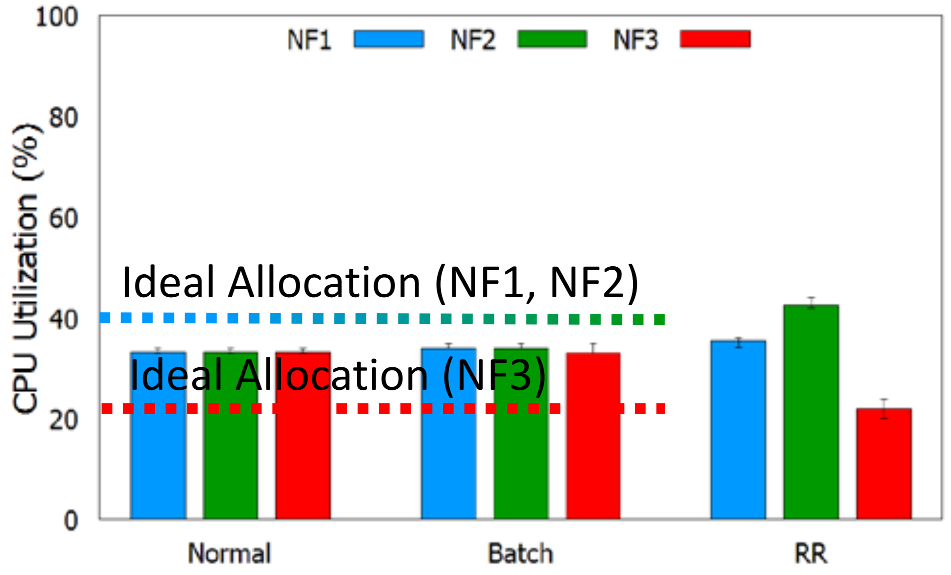


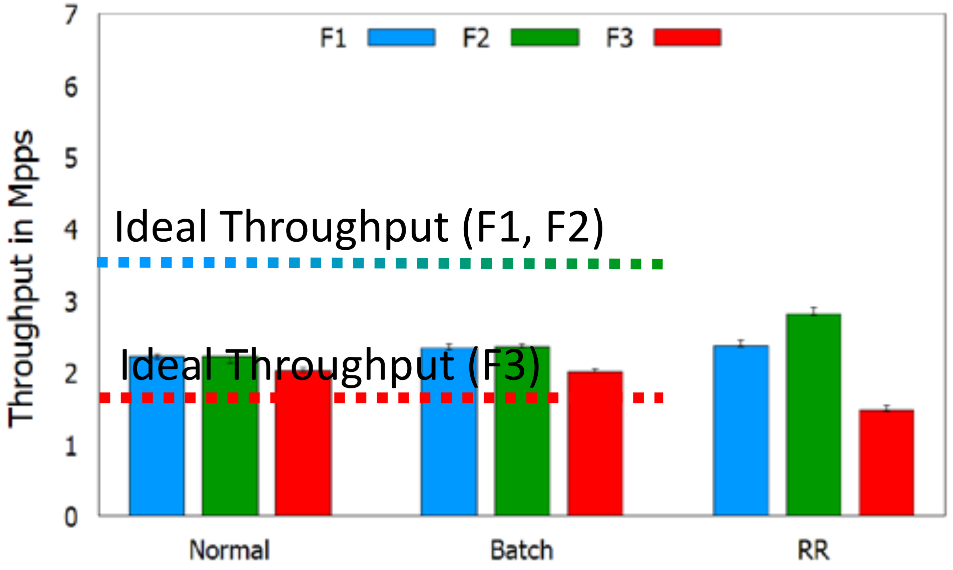


OS Scheduler Characterization (Load)

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

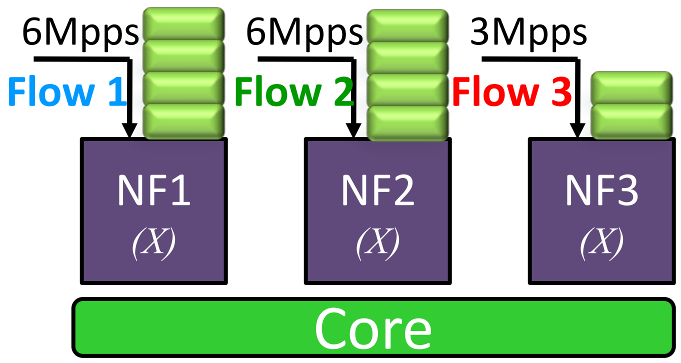




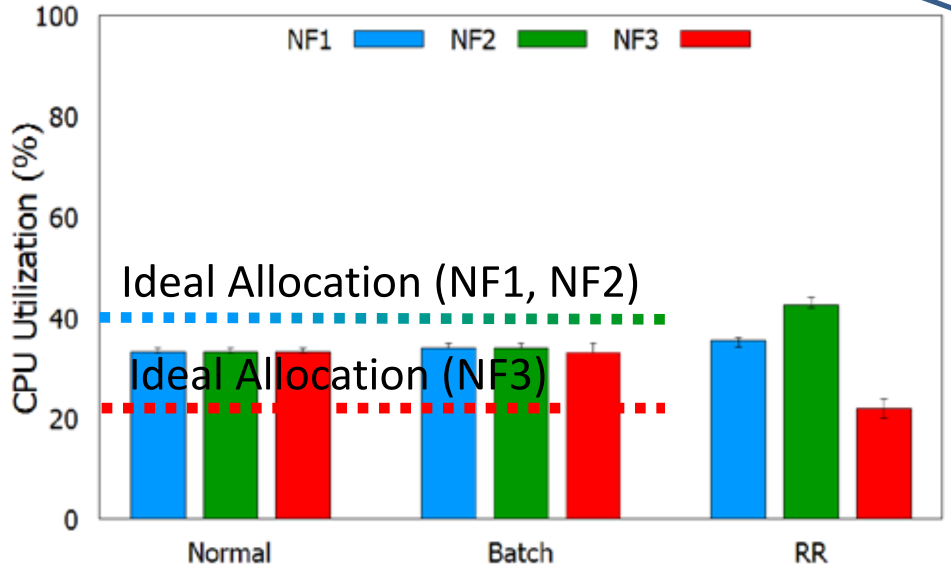


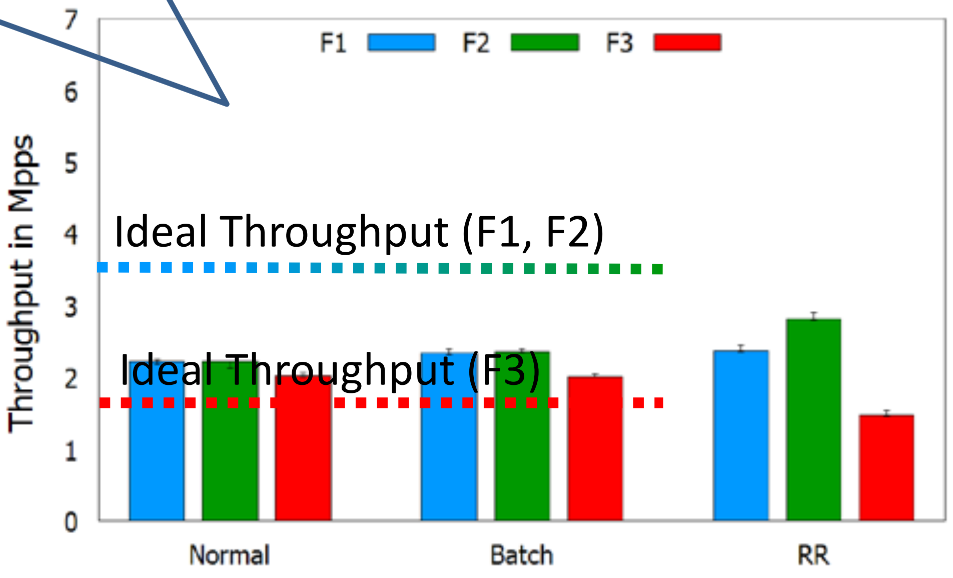
OS Scheduler Characterization (Load)

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



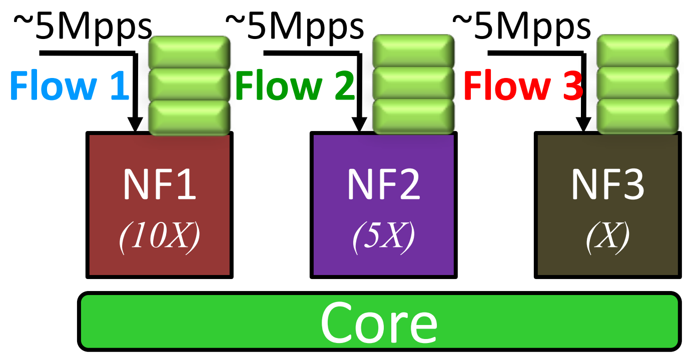
Schedulers fail to account NF load!

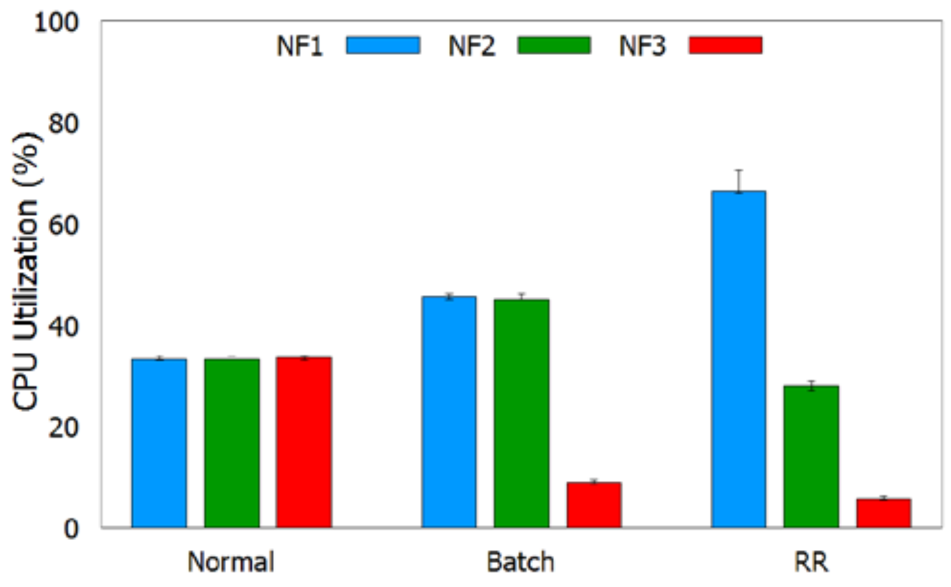




OS Scheduler Characterization (Cost)

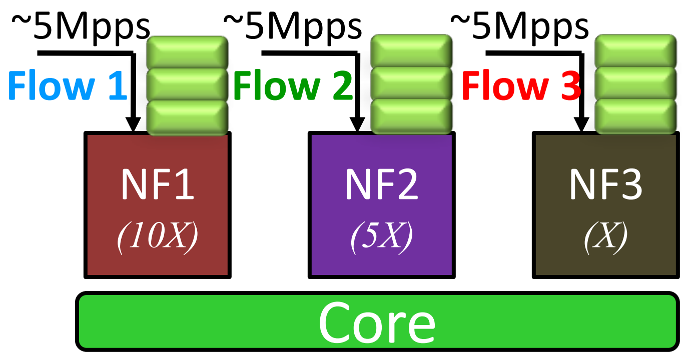
3 Heterogeneous NFs (per packet processing cost 10:5:1) with equal load

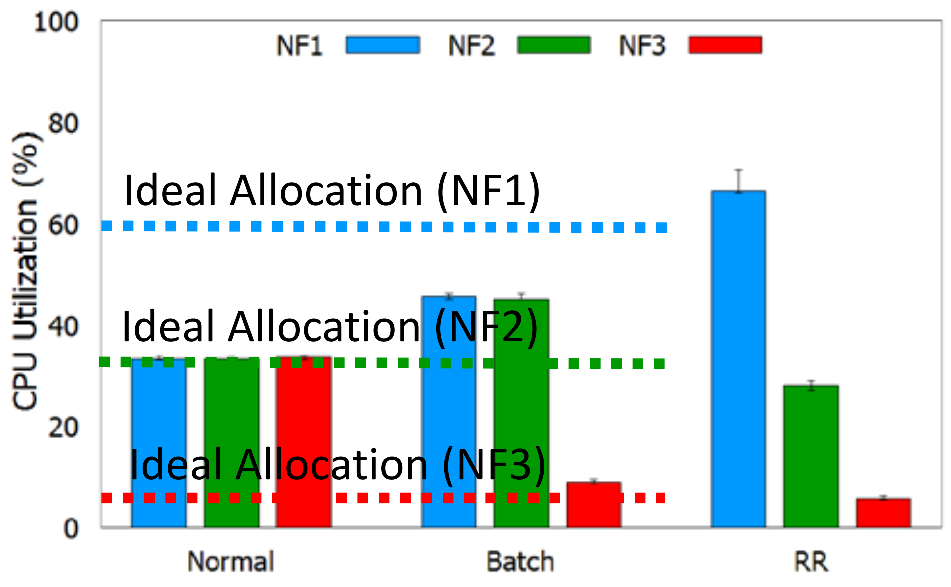




OS Scheduler Characterization (Cost)

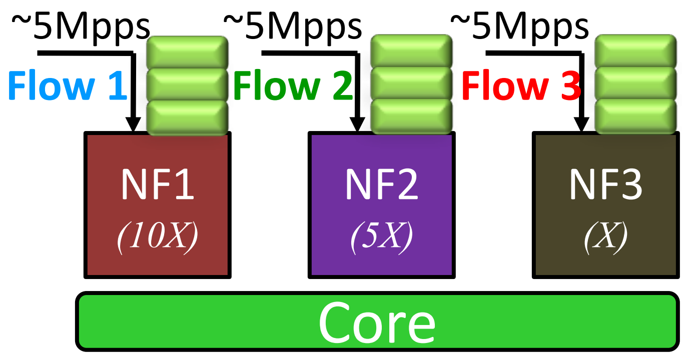
3 Heterogeneous NFs (per packet processing cost 10:5:1) with equal load

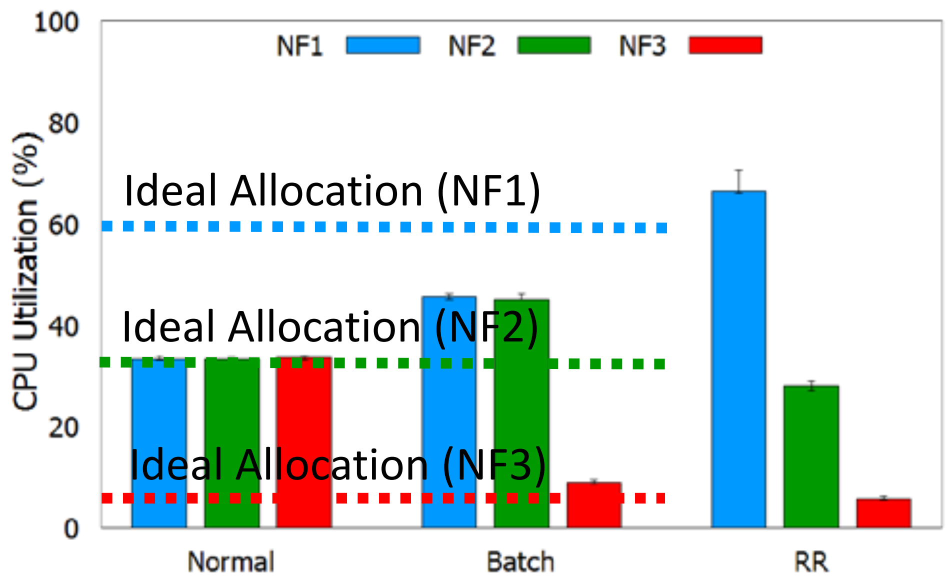


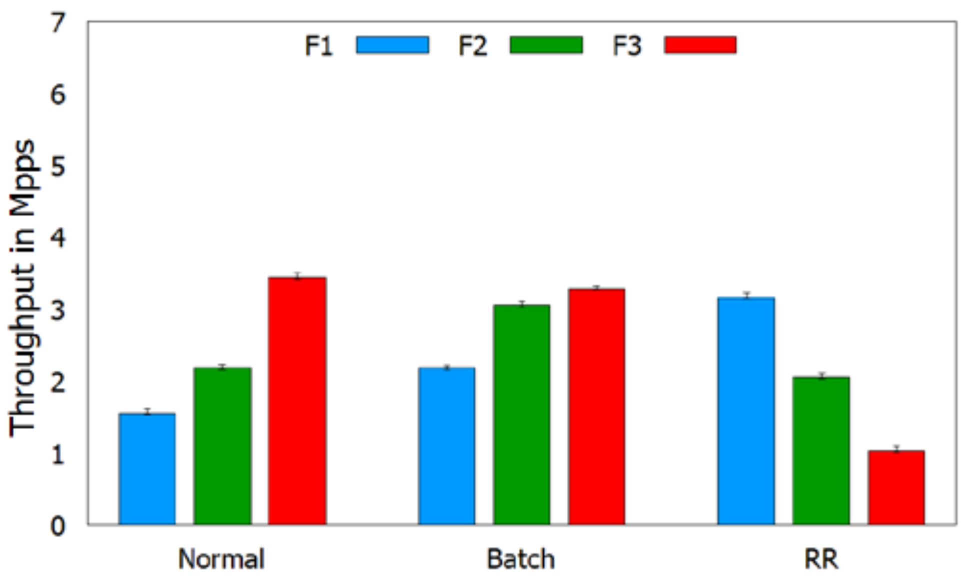


OS Scheduler Characterization (Cost)

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

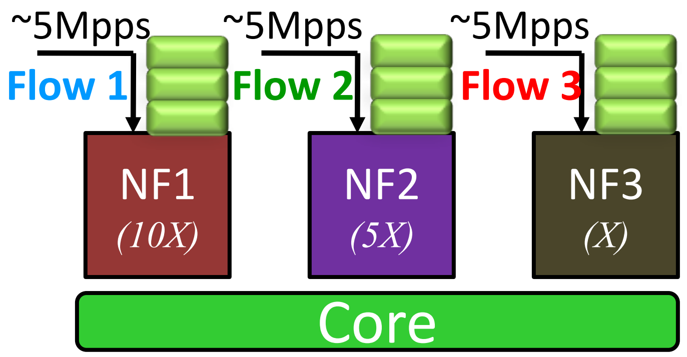


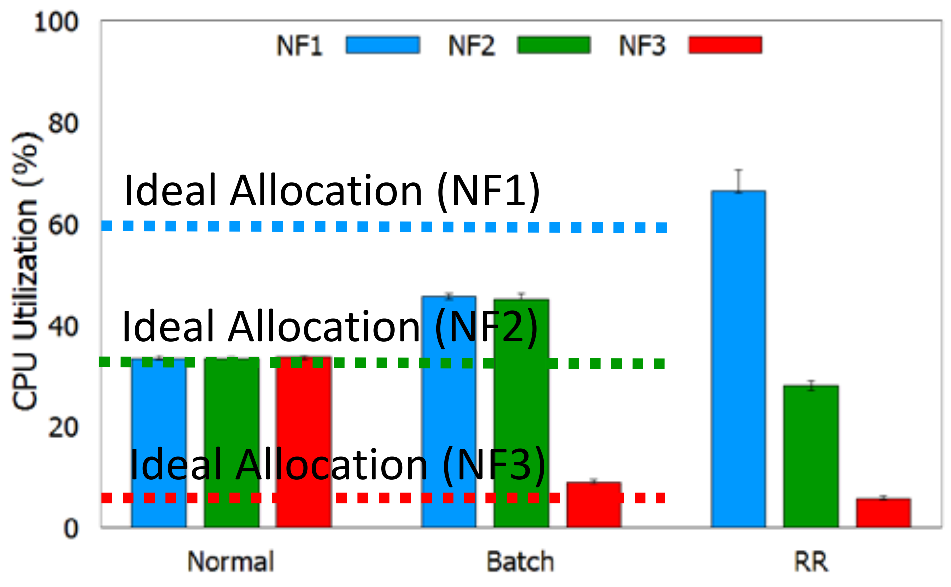


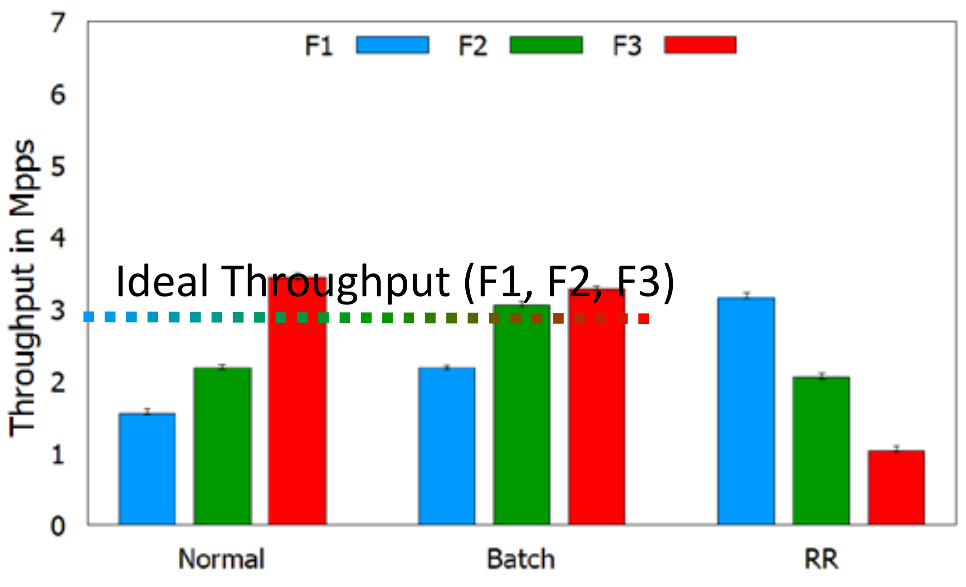


OS Scheduler Characterization (Cost)

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

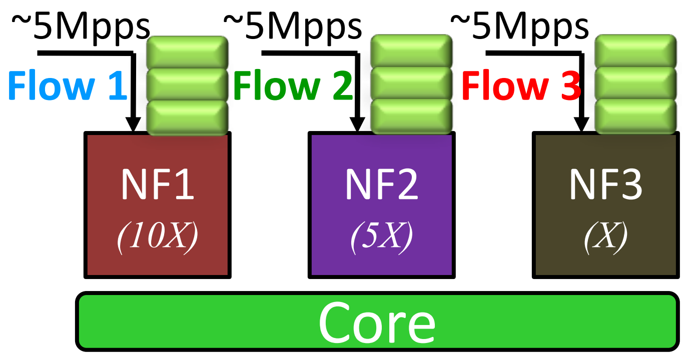




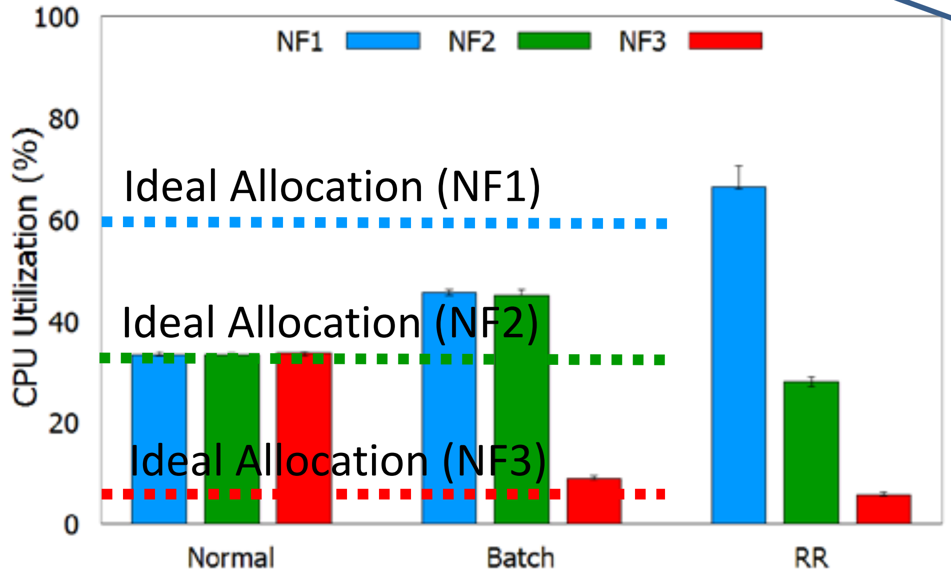


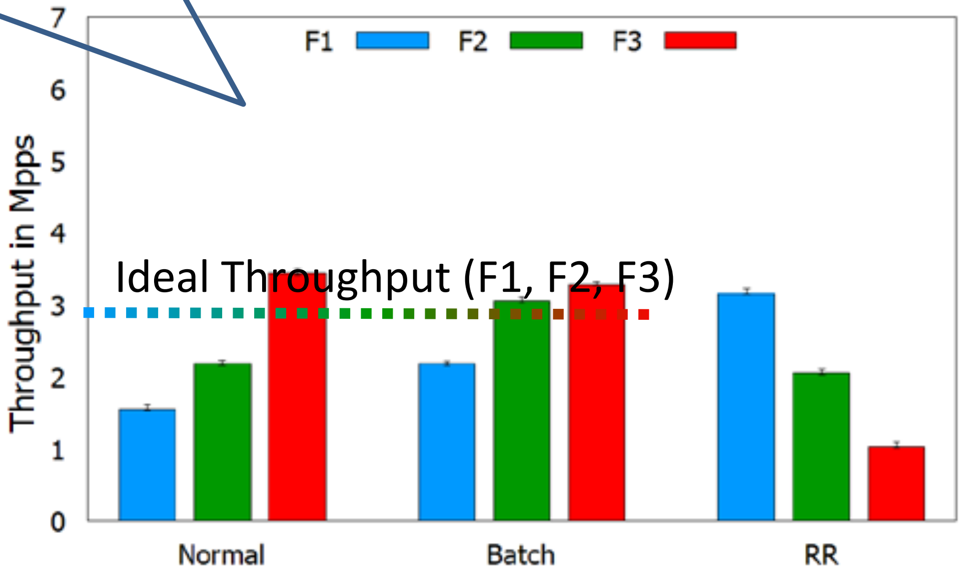
OS Scheduler Characterization (Cost)

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



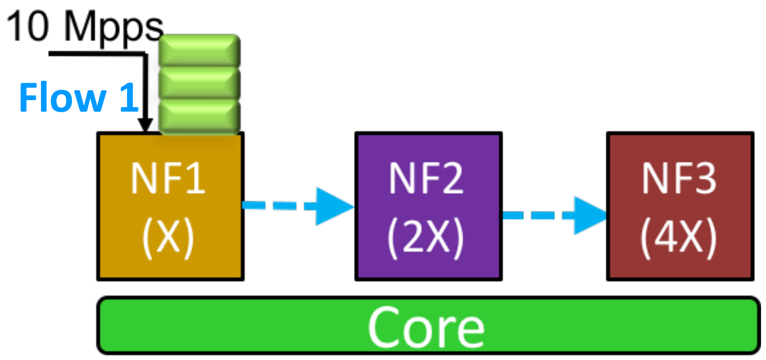
Schedulers fail to account NF Processing cost!





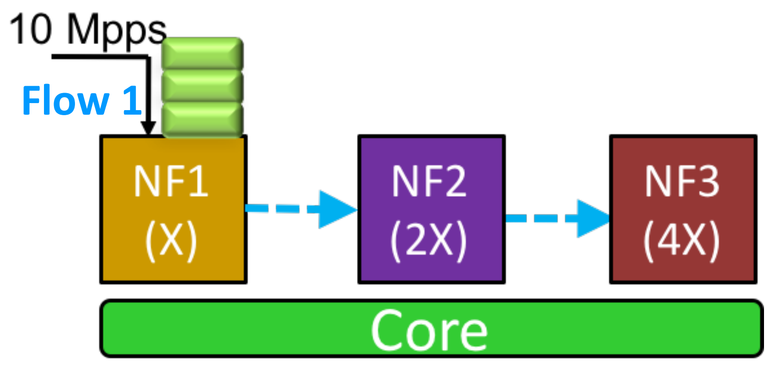
OS Scheduler Characterization (Chain)

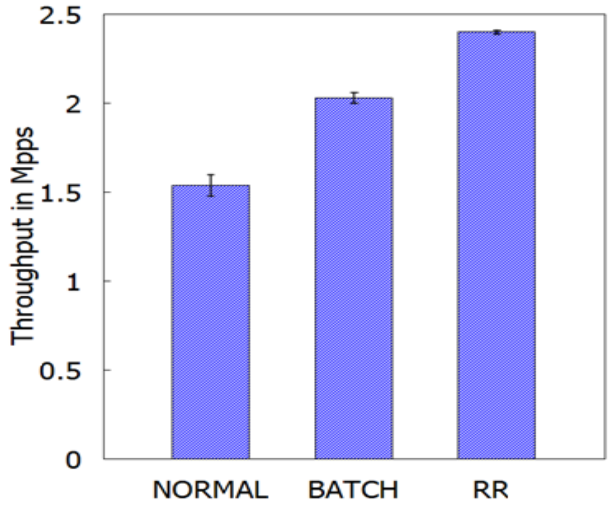
3 NF chain (all NFs running on same core):



OS Scheduler Characterization (Chain)

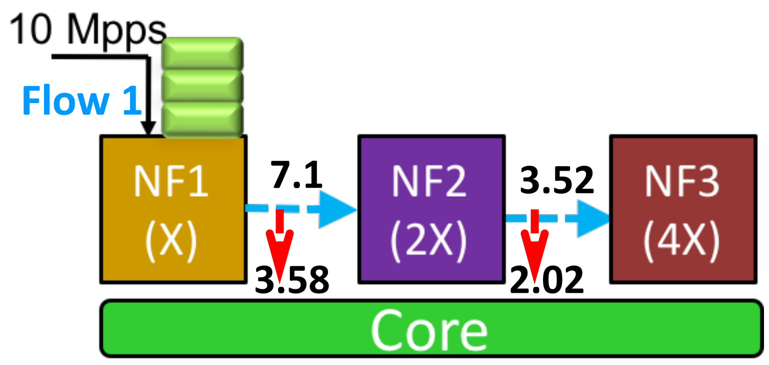
3 NF chain (all NFs running on same core):

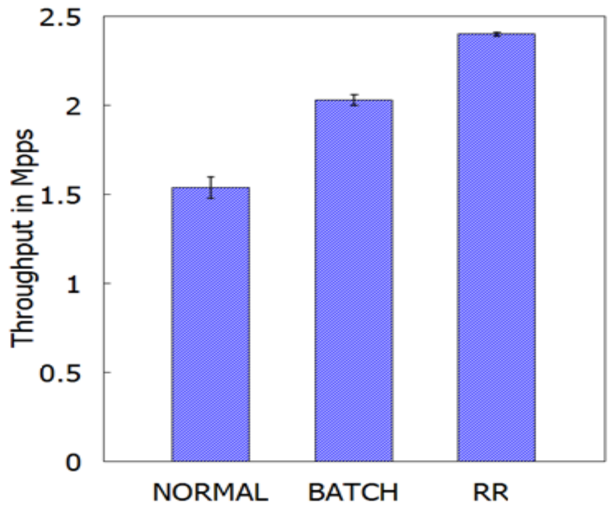


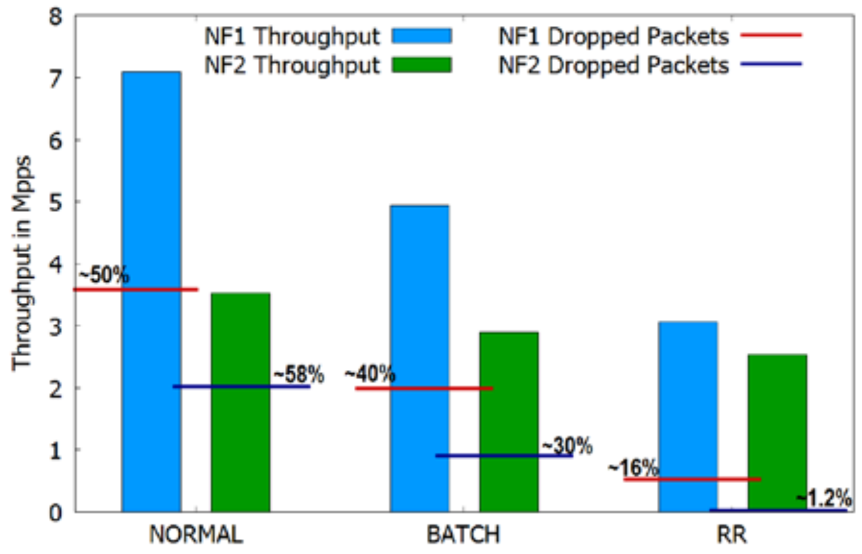


OS Scheduler Characterization (Chain)

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

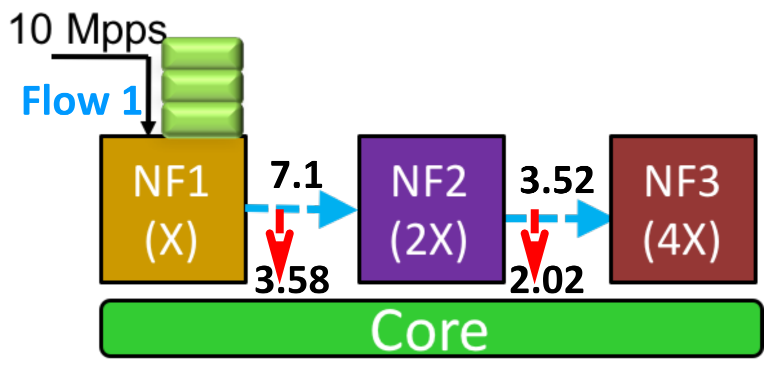


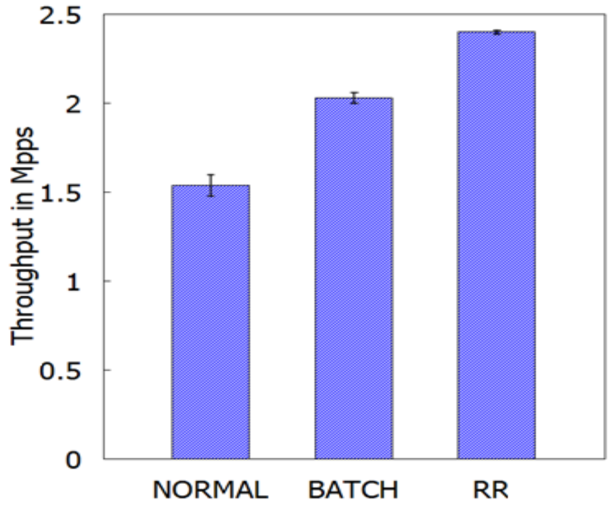


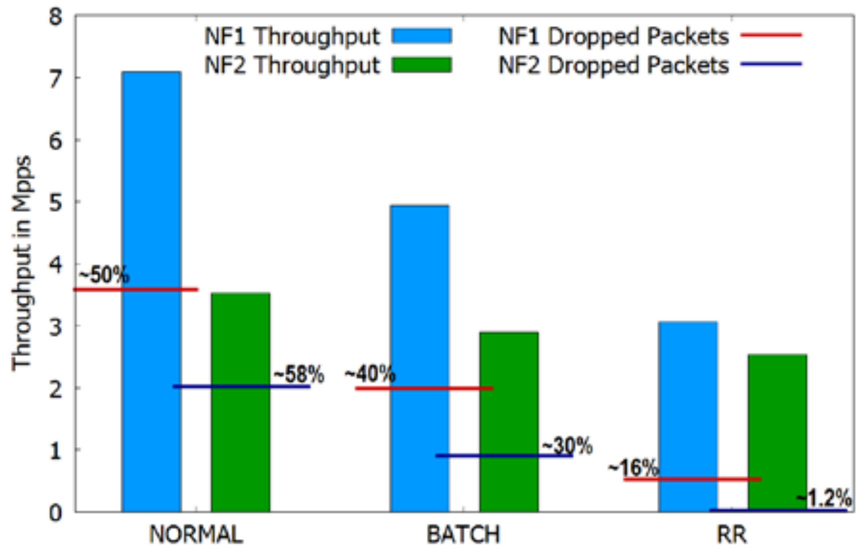


OS Scheduler Characterization (Chain)

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





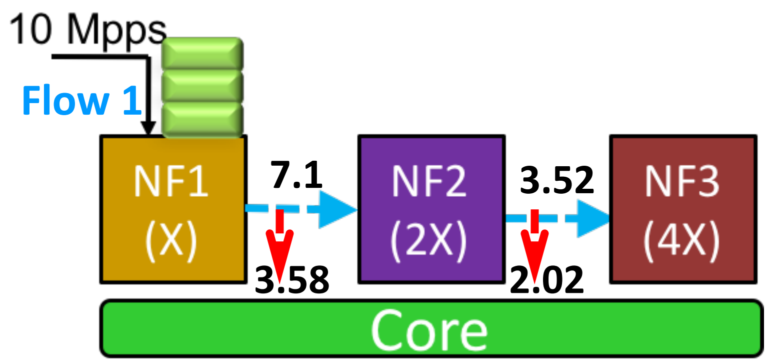


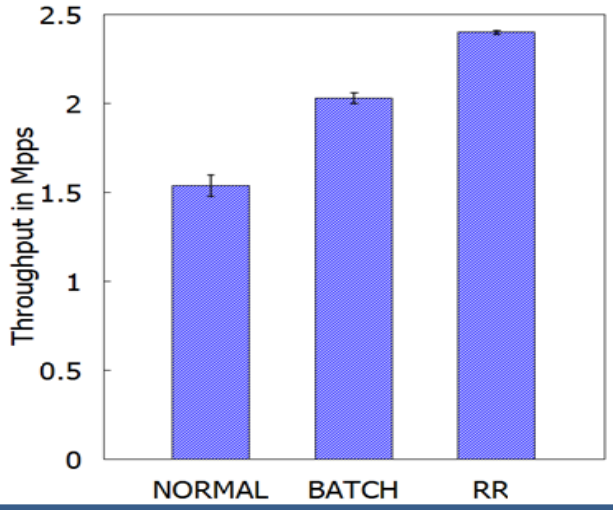
Ctx sw/s	NORMAL	BATCH	RR
Total	**20K/s**	**2K/s**	**1K/s**

CPU %	NORMAL	BATCH	RR
NF1	34%	15%	9%
NF2	34%	42%	37%
NF3	33%	43%	54%

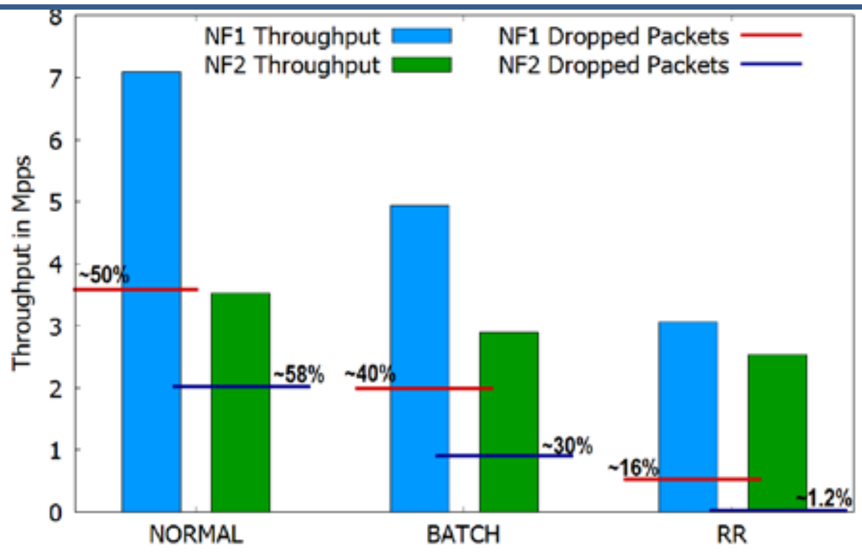
OS Scheduler Characterization (Chain)

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





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

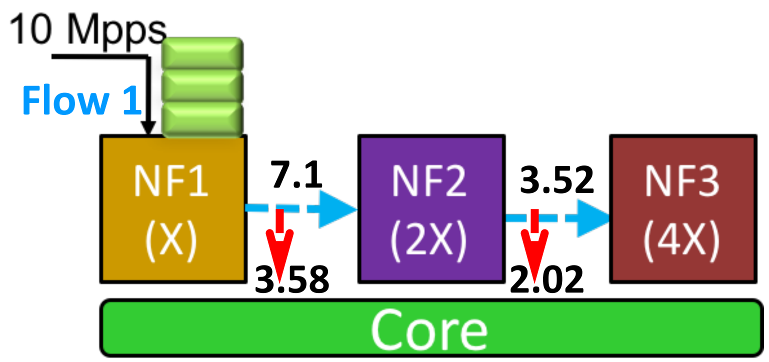


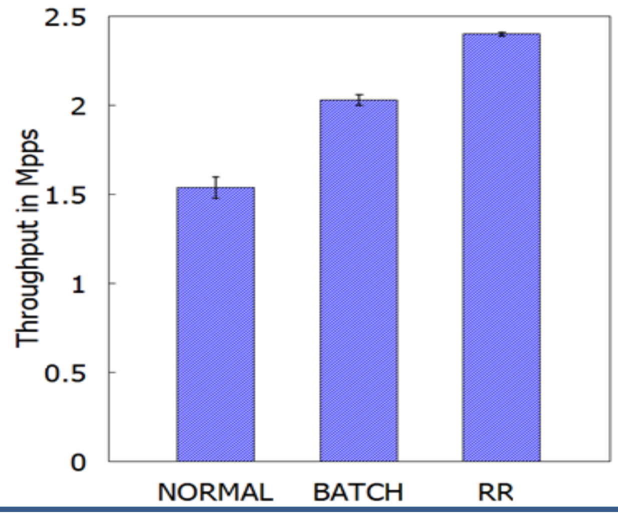
Ctx sw/s	NORMAL	BATCH	RR
Total	**20K/s**	**2K/s**	**1K/s**

CPU %	NORMAL	BATCH	RR
NF1	34%	15%	9%
NF2	34%	42%	37%
NF3	33%	43%	54%

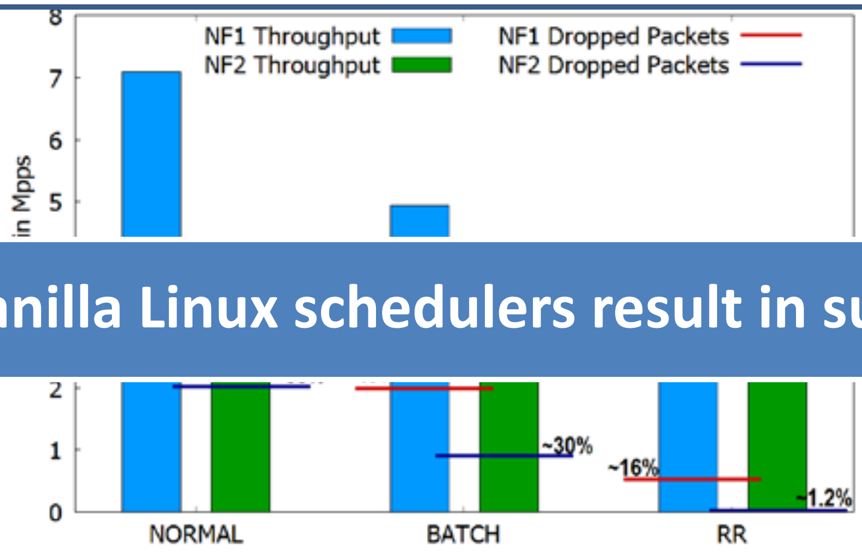
OS Scheduler Characterization (Chain)

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





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



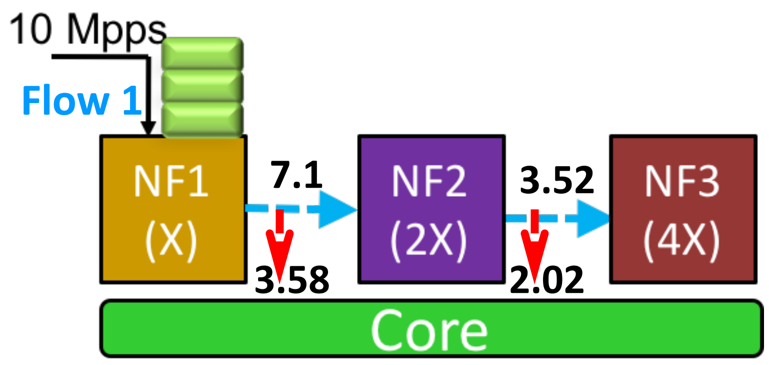
Ctx sw/s	NORMAL	BATCH	RR
Total	20K/s	2K/s	1K/s

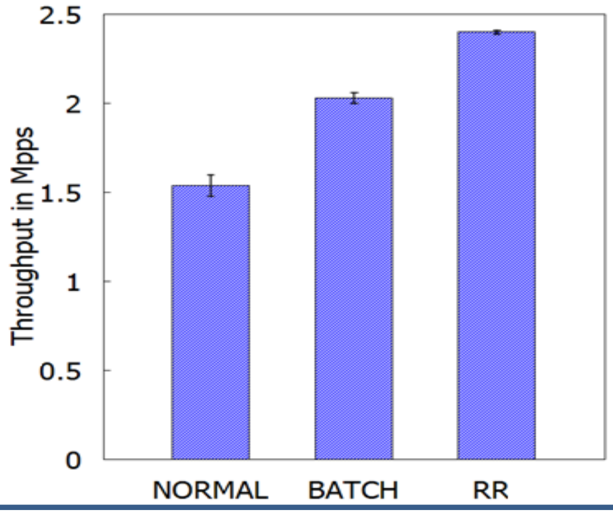
CPU %	NORMAL	BATCH	RR
NF2	34%	42%	37%
NF3	33%	43%	54%

Vanilla Linux schedulers result in sub-optimal resource utilization.

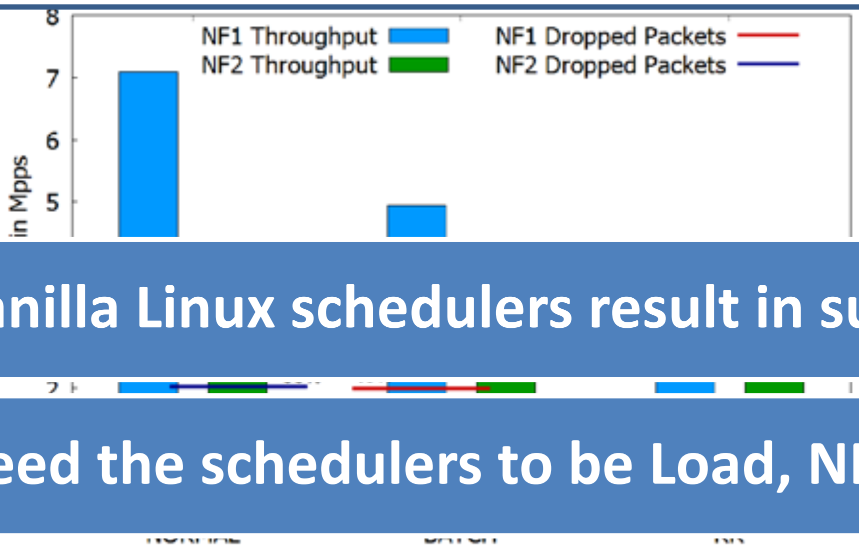
OS Scheduler Characterization (Chain)

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





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



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

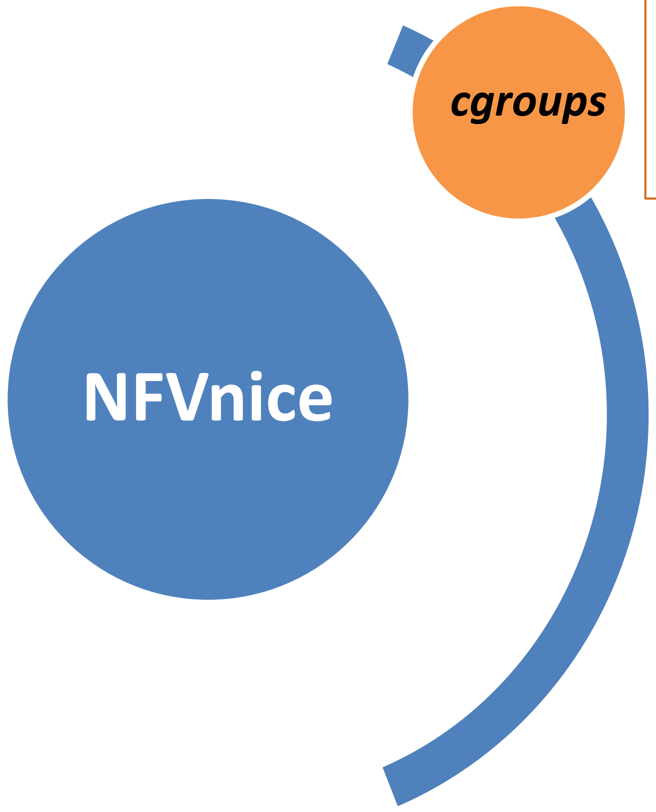
A user space control framework for scheduling NFV chains.

NFVnice

A user space control framework for scheduling NFV chains.

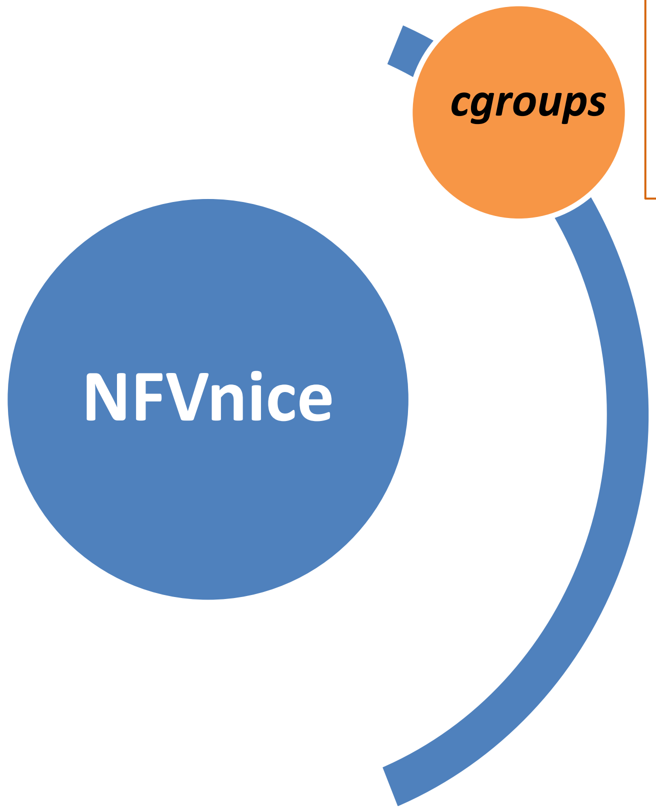
- NFVnice in a nutshell:
  - Complements the existing kernel task schedulers.
    - Integrates “Rate proportional scheduling” from Hardware schedulers.
    - Integrates “Cost Proportional scheduling” from software schedulers.
  - Built on OpenNetVM[HMBox’16, NSDI’14]: *A DPDK based NFV platform.*
    - 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



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.

cgroups

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.

cgroups

Rate-Cost Proportional Fairness

Initialization

mkdir /cgroupfs/NF(i)

cgroups

Rate-Cost Proportional Fairness

Initialization

mkdir /cgroupfs/NF(i)

Weight Computation

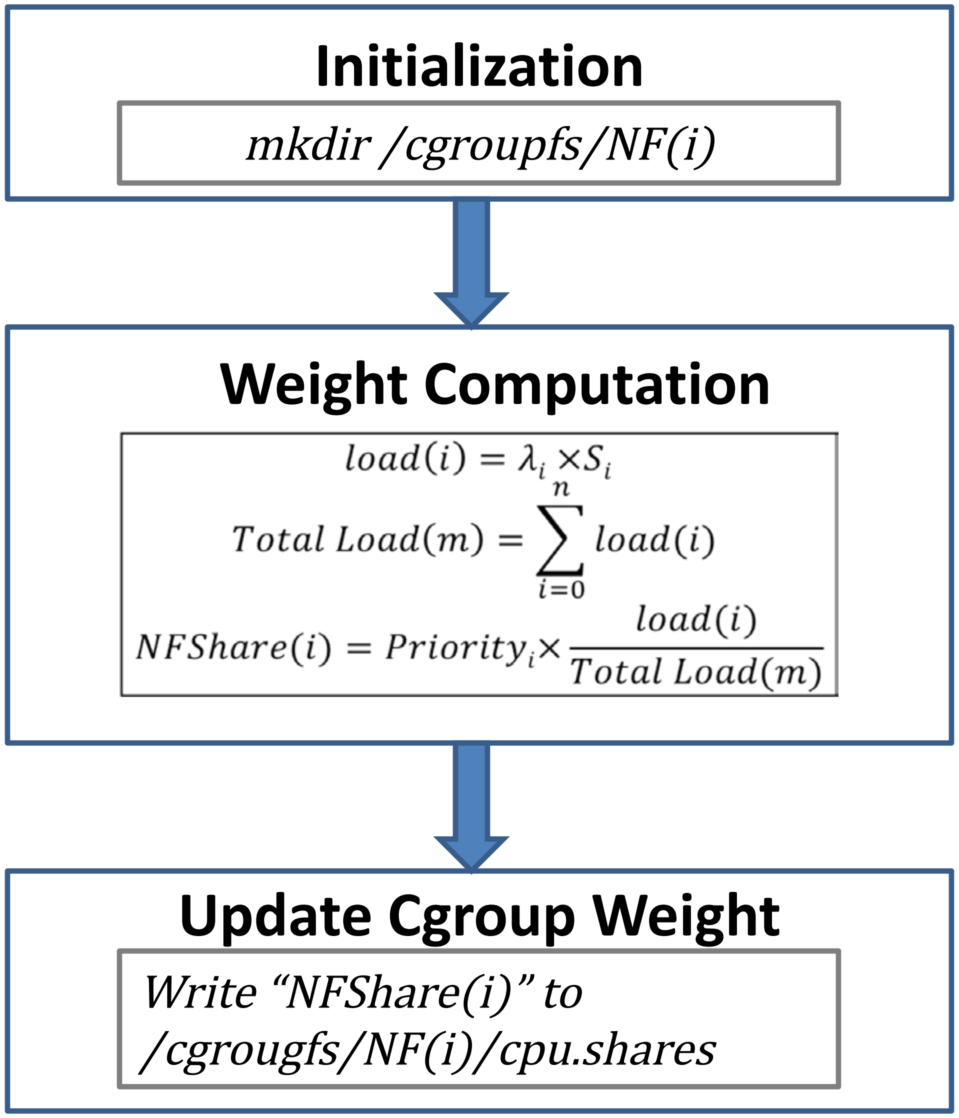
$$load(i) = \lambda_i \times S_i$$

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

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

cgroups

Rate-Cost Proportional Fairness



Initialization

mkdir /cgroupfs/NF(i)

Weight Computation

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

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

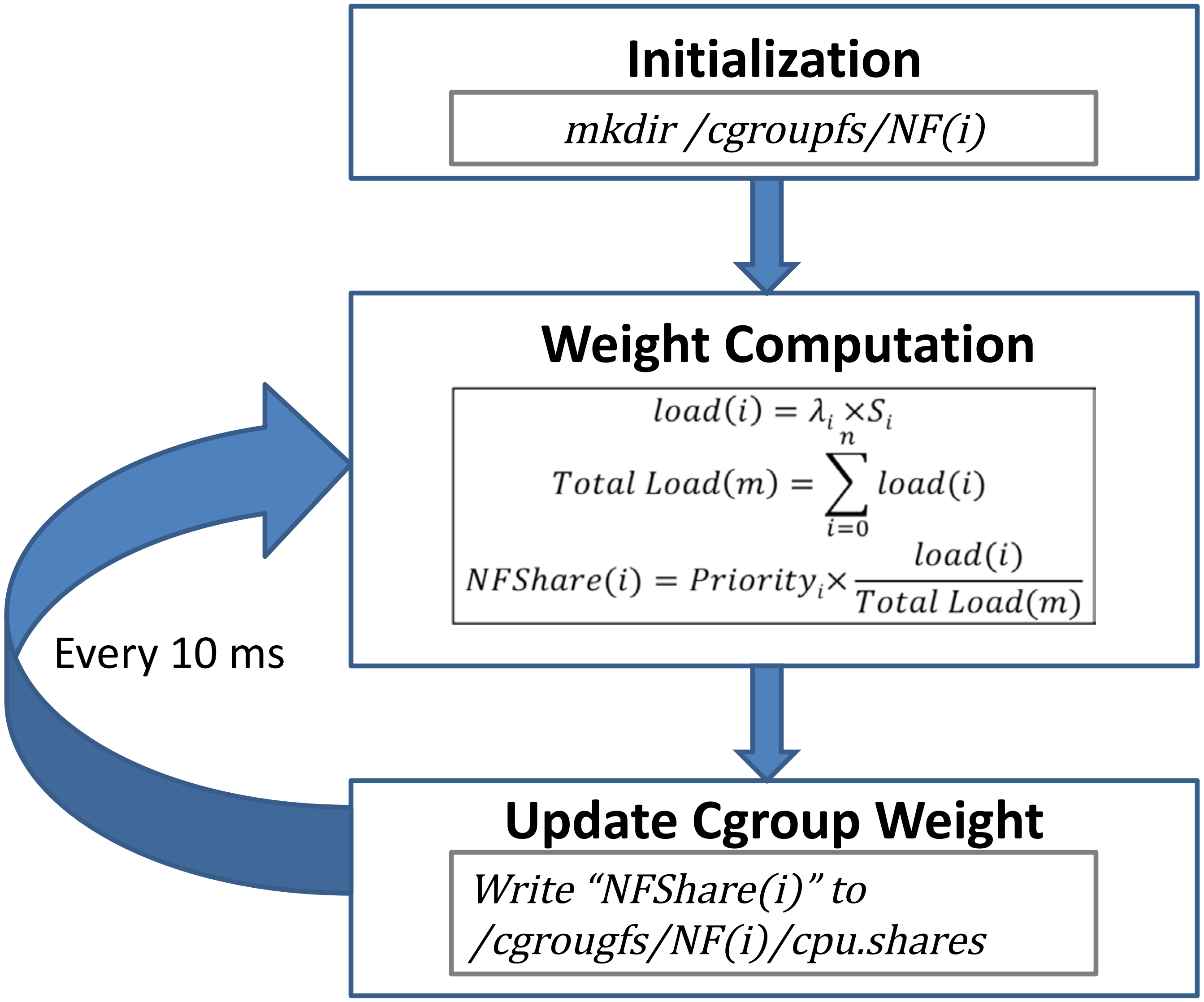
$$NFShare(i) = Priority_i \times \frac{load(i)}{Total\ Load(m)}$$

Update Cgroup Weight

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

cgroups

Rate-Cost Proportional Fairness



mkdir /cgroupfs/NF(i)

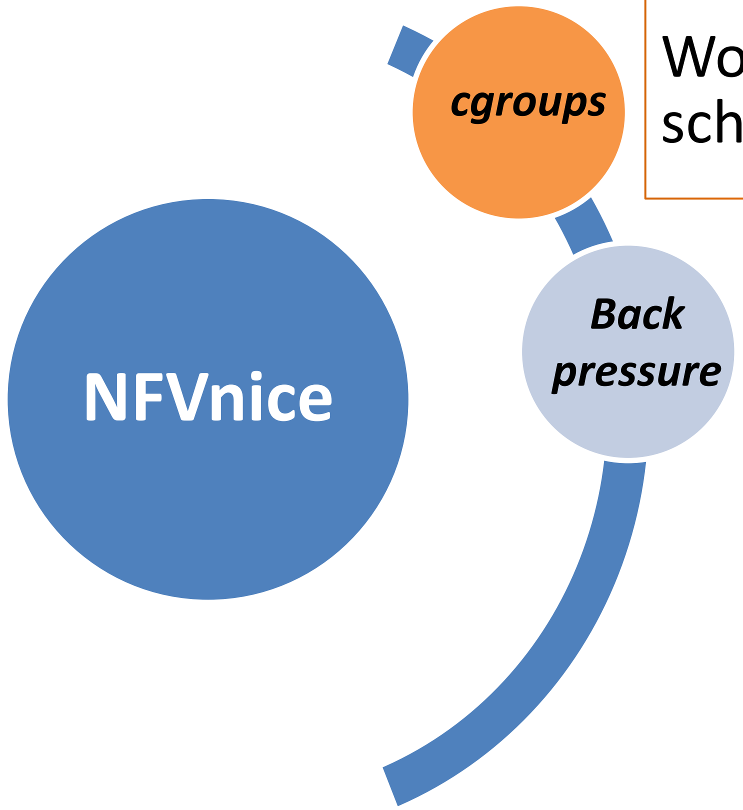
$$load(i) = \lambda_i \times S_i$$

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

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

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

NFVnice: Building Blocks



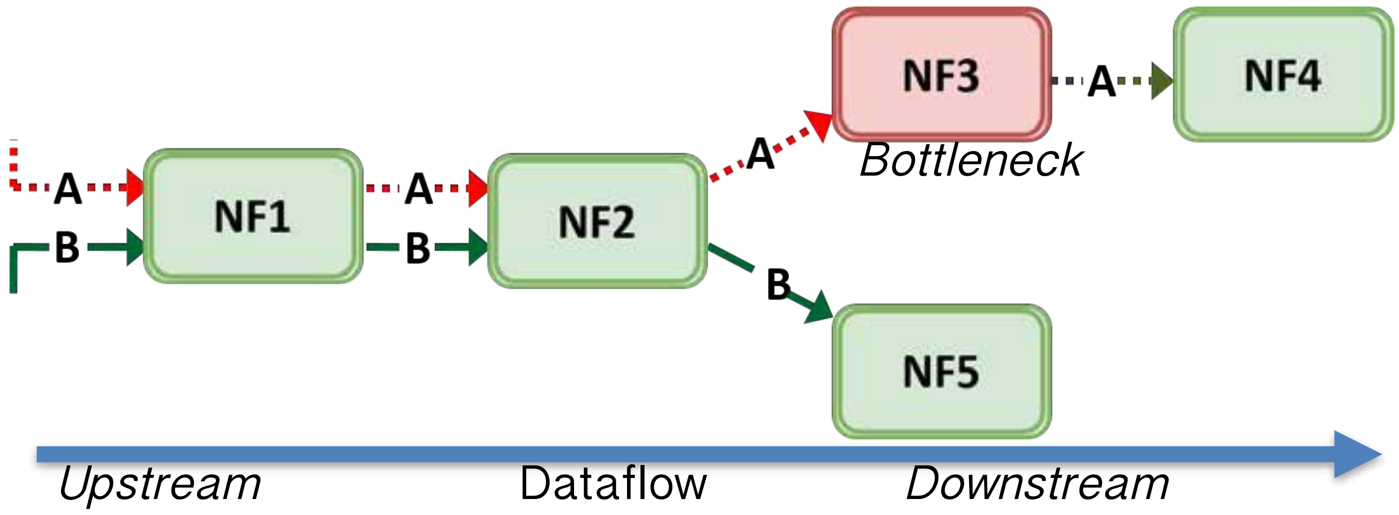
Work-conserving and proportional scheduling *(within each core)*

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

Back pressure

Backpressure in NF chains

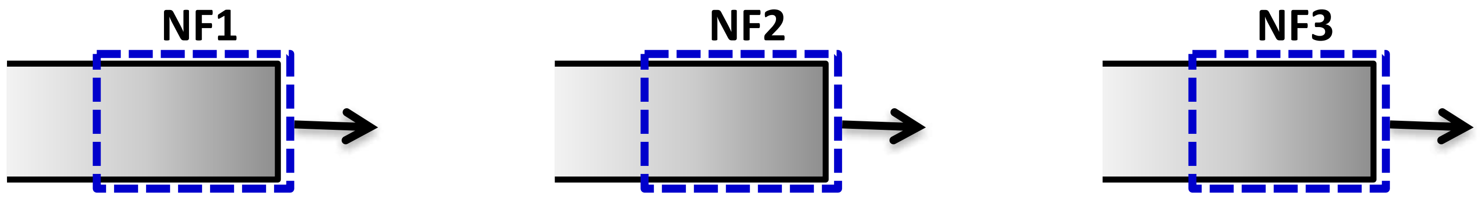
- Selective per chain backpressure marking.

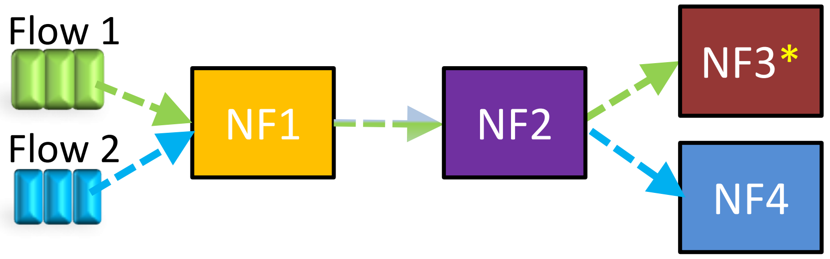


  - Only Flow “A” going through bottleneck NF (NF3) is back pressured and throttled at the upstream source NF1.
  - while Flow “B” is not affected.

Back pressure

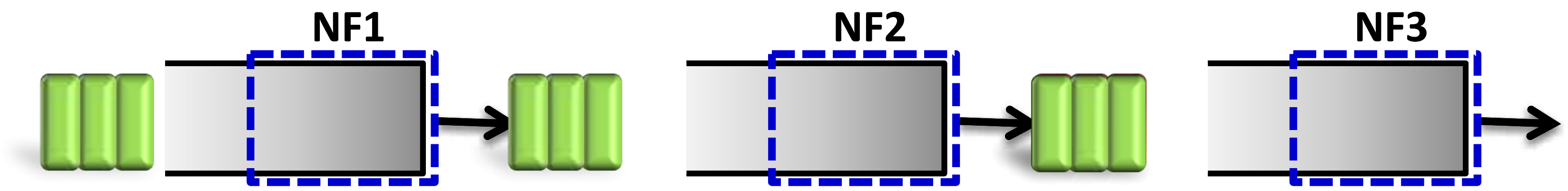
Scenario: No Backpressure

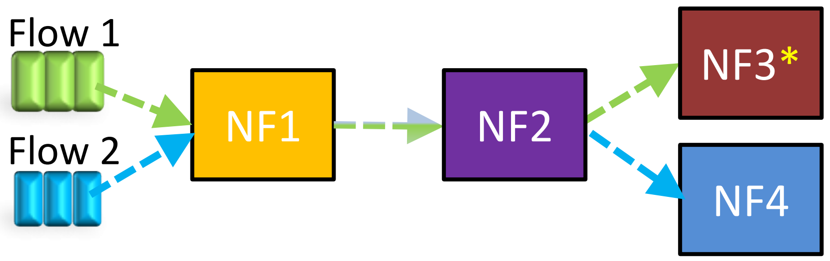




Back pressure

Scenario: No Backpressure

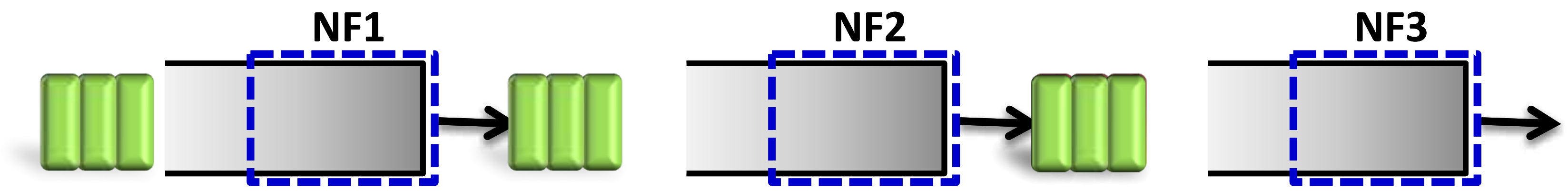


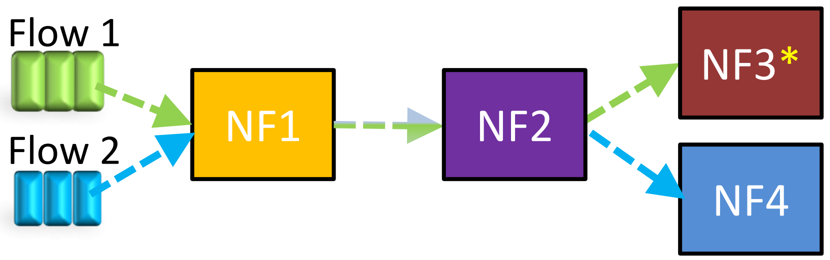


NF1 and NF2 contend for CPU, and steal the CPU cycles from NF3!

Back pressure

Scenario: No Backpressure



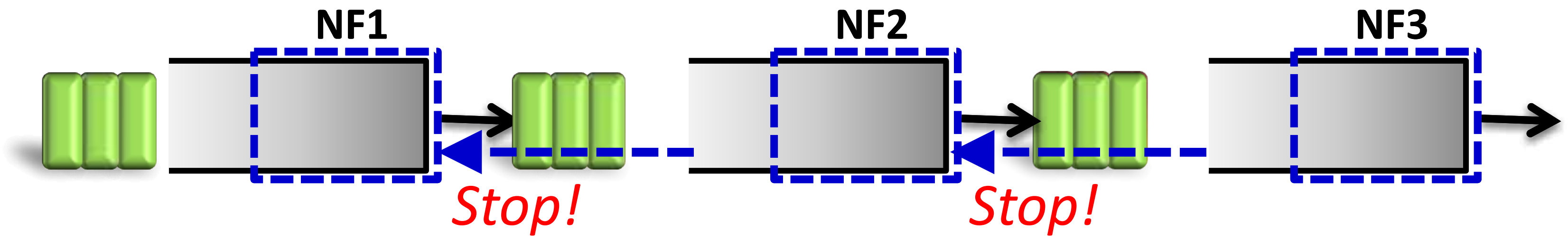


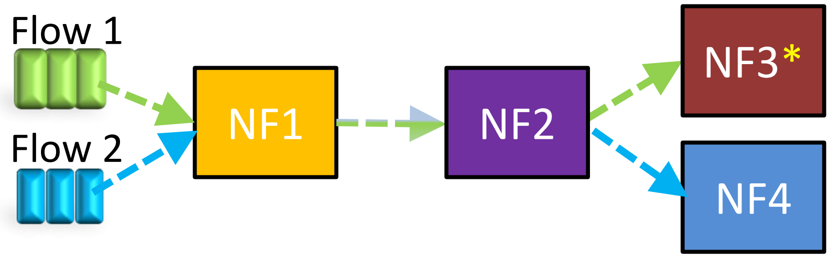
NF1 and NF2 contend for CPU, and steal the CPU cycles from NF3!

Lots of Wasted Work!!!

Back pressure

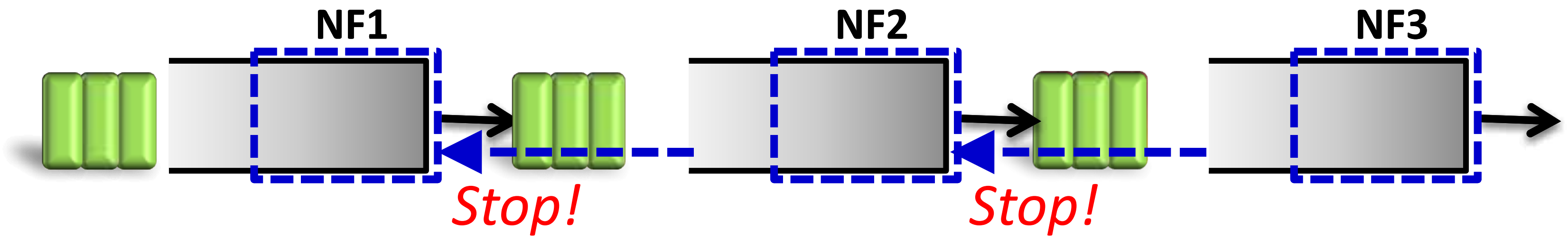
Traditional Backpressure

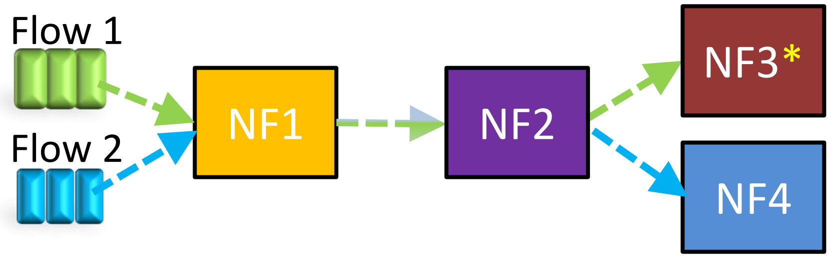




Back pressure

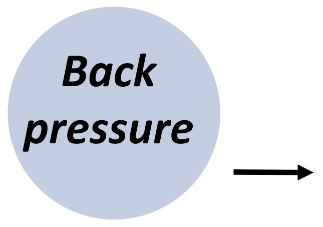
Traditional Backpressure



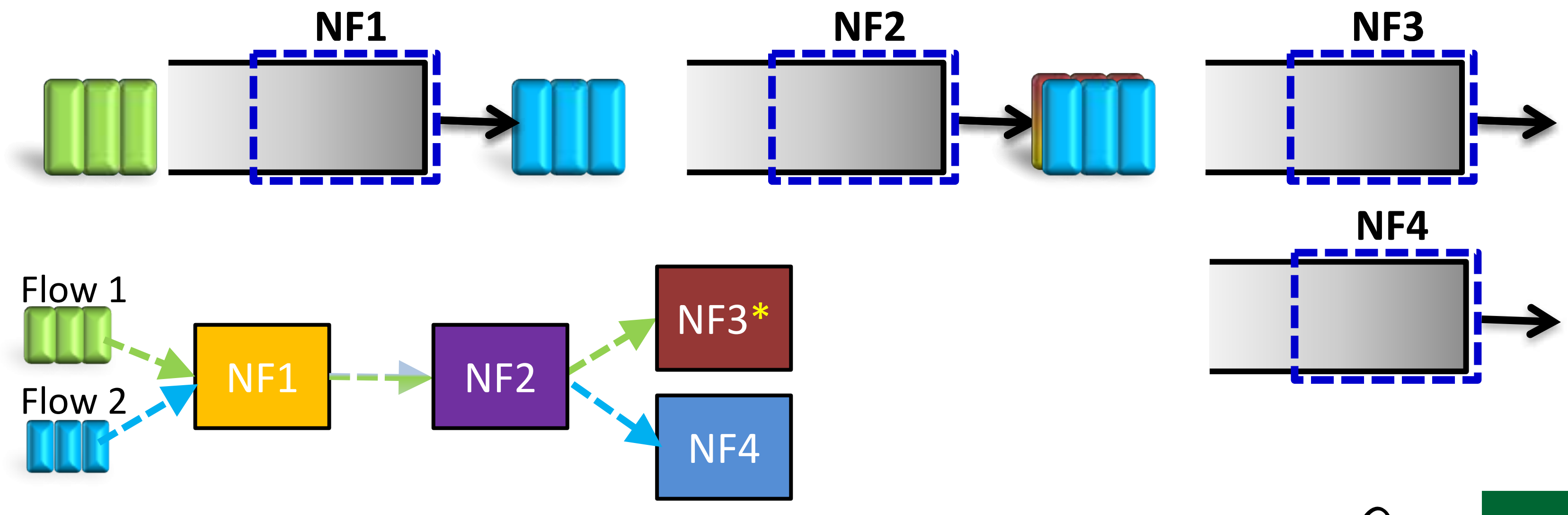


~~**Lots of Wasted Work**~~

Incurs feedback delay and offers no flow isolation.



NFVnice Backpressure

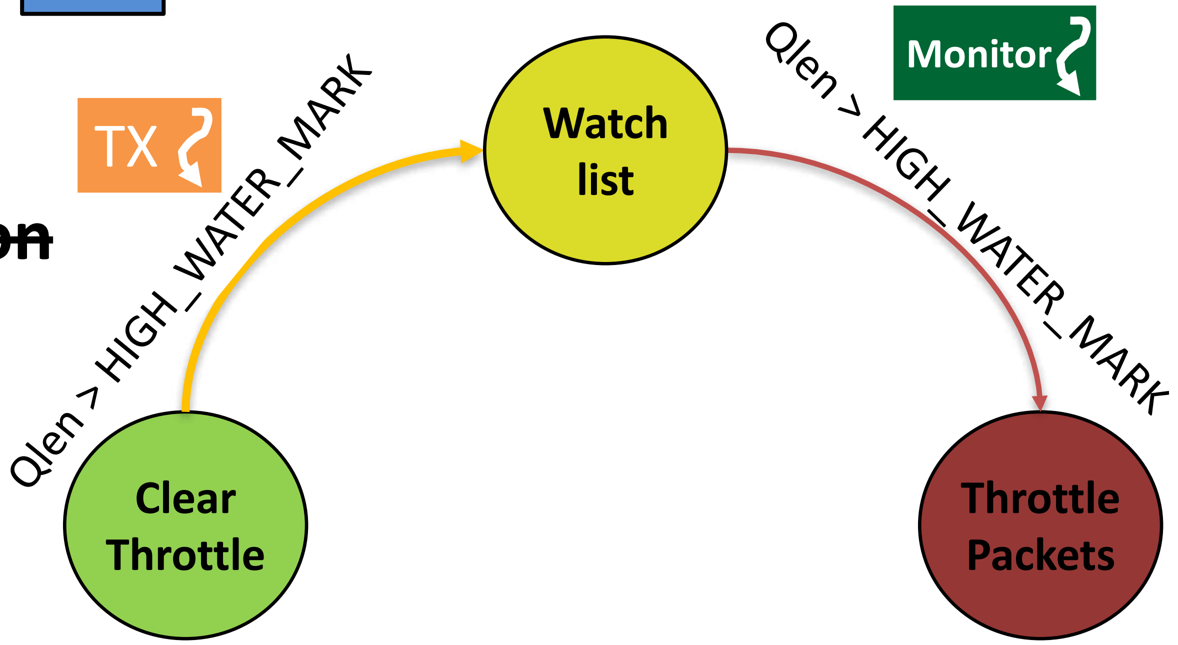


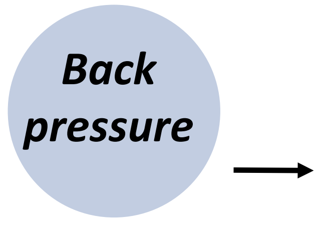
~~Lots of Wasted Work~~

~~Incurs Delay, No Isolation~~

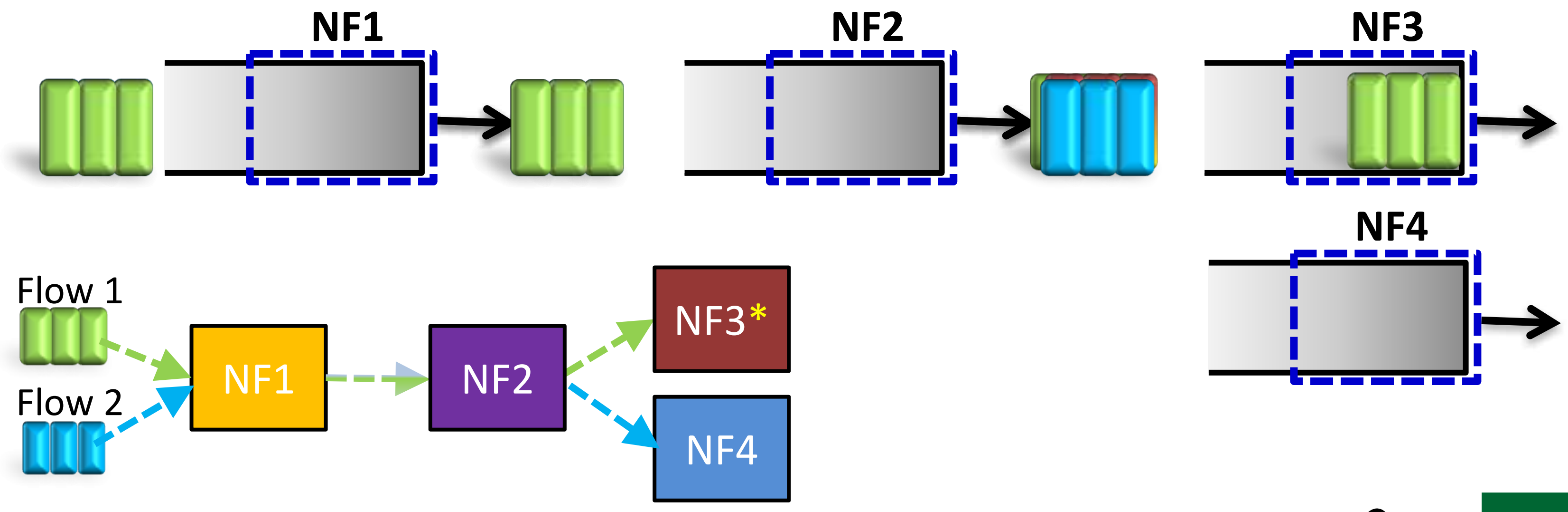
Reacts Instantaneously

Packet Throttling per Chain





NFVnice Backpressure

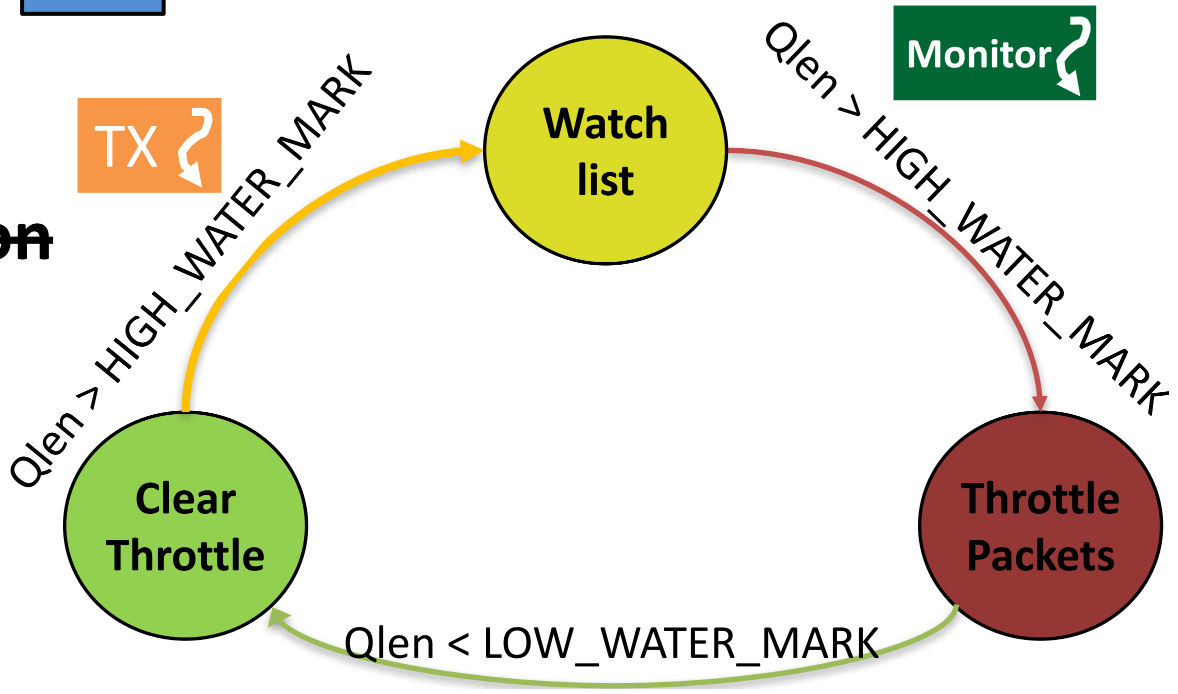


~~**Lots of Wasted Work**~~

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

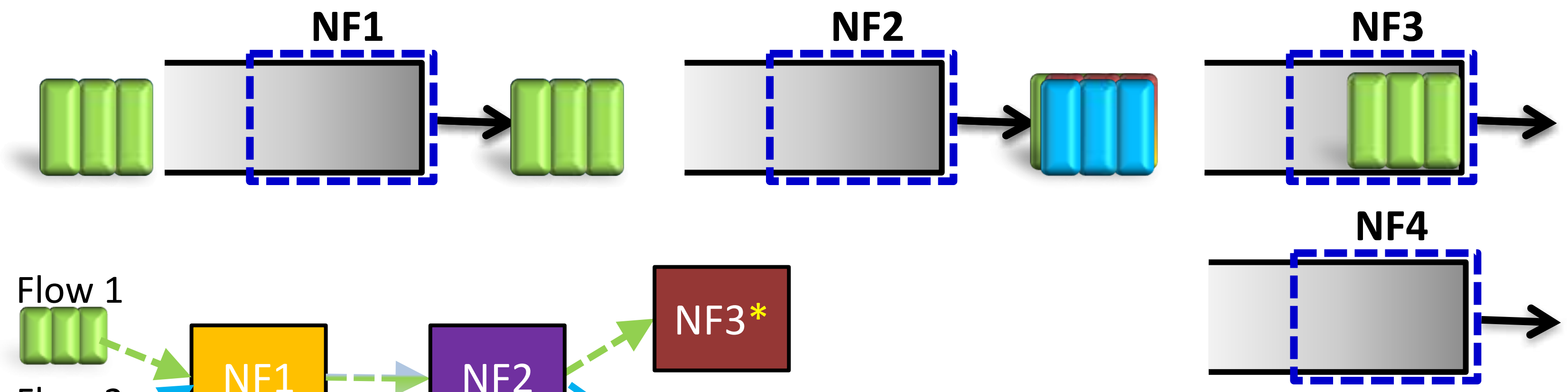
Reacts Instantaneously

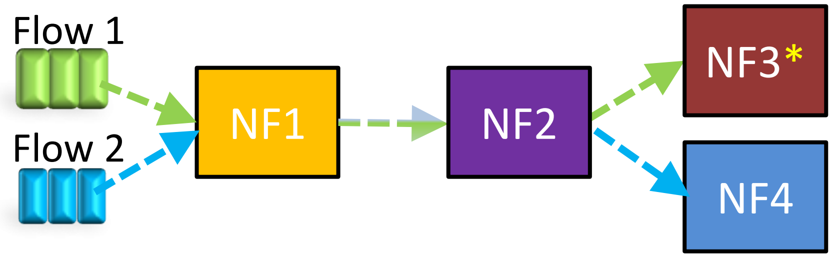
Packet Throttling per Chain



Back pressure →

NFVnice Backpressure



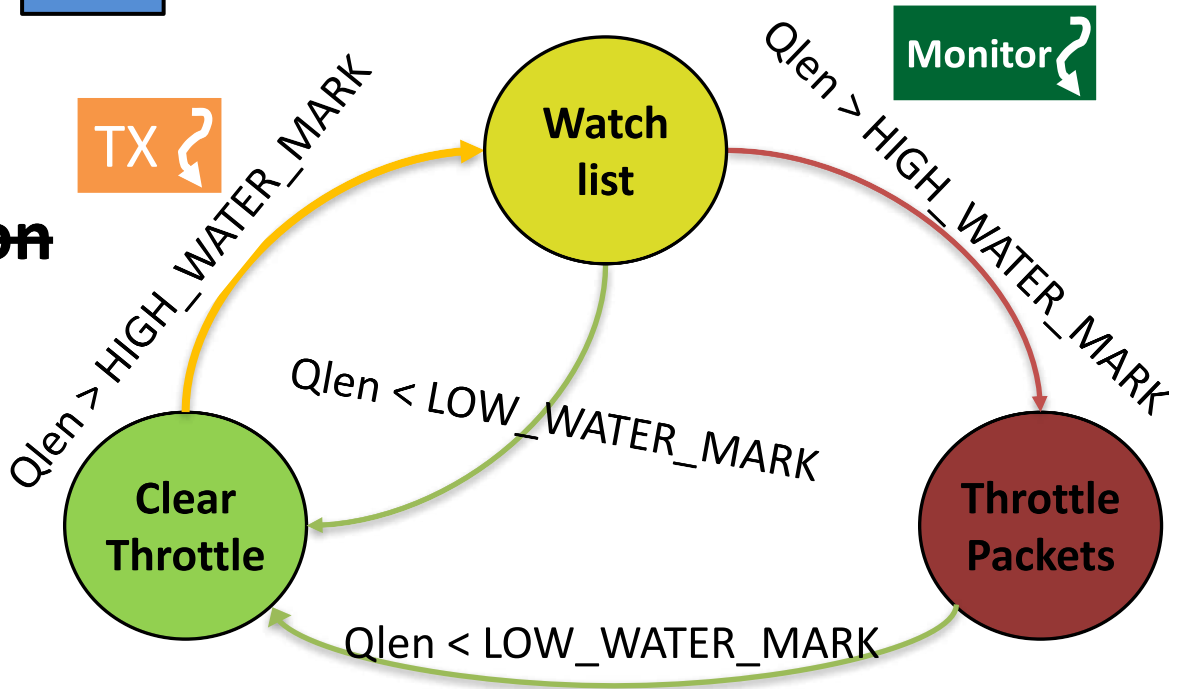


~~Lots of Wasted Work~~

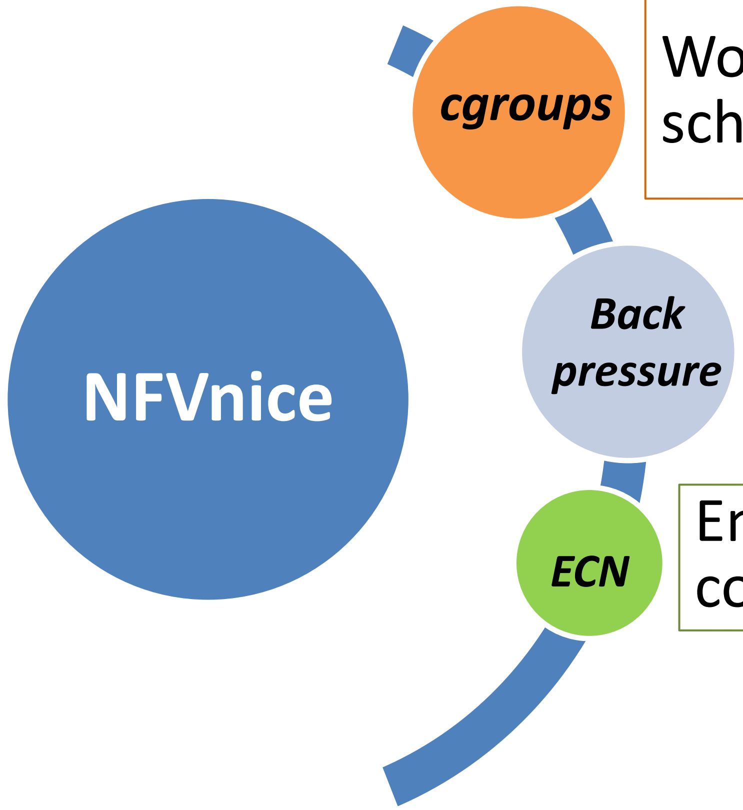
~~Incurs Delay, No Isolation~~

Reacts Instantaneously

Packet Throttling per Chain



NFVnice: Building Blocks



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

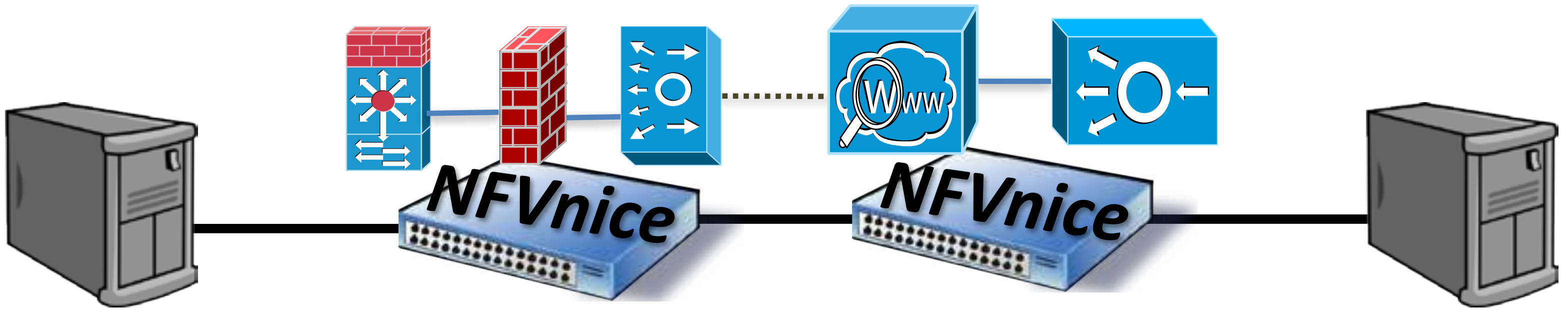
Chain-aware scheduling; Avoid wasted work *(within and across cores)*

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

ECN

ECN Marking

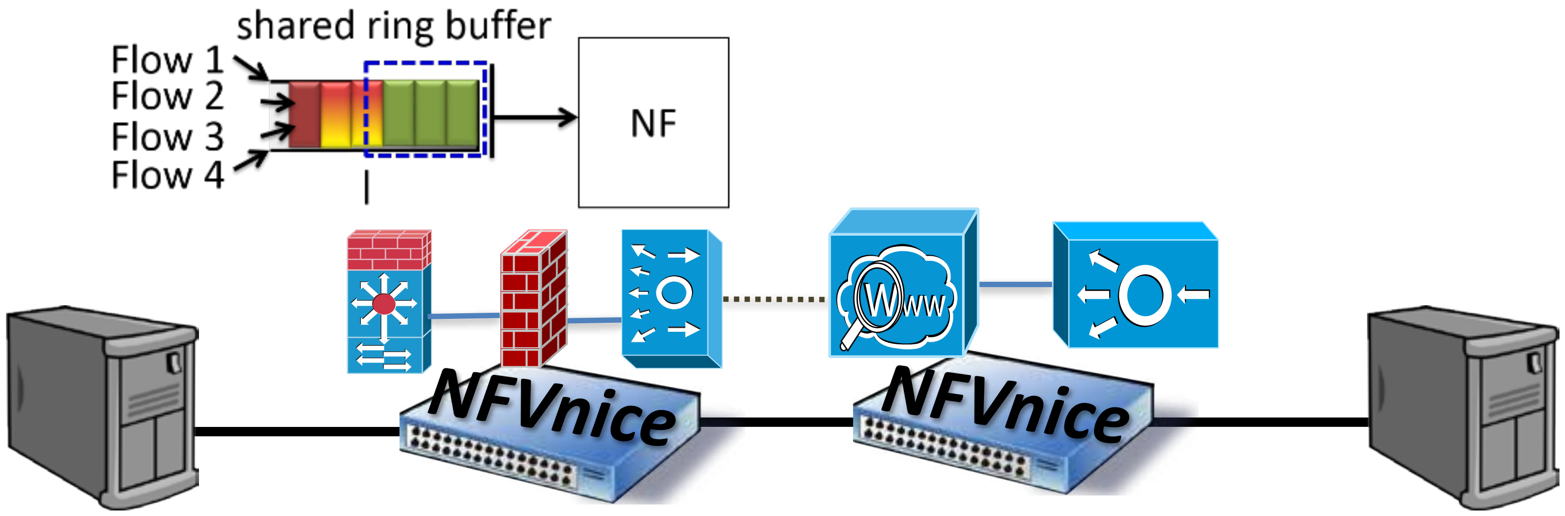
- ECN-aware NF Manager:
  - Per-NF ECN marking based on Active Queue Management (AQM) policies.



ECN

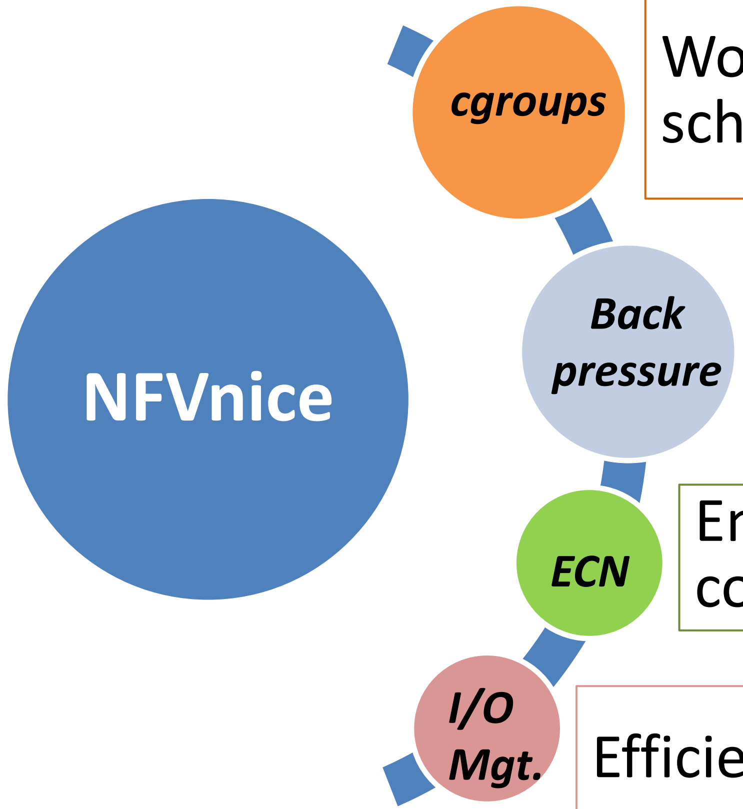
ECN Marking

- 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



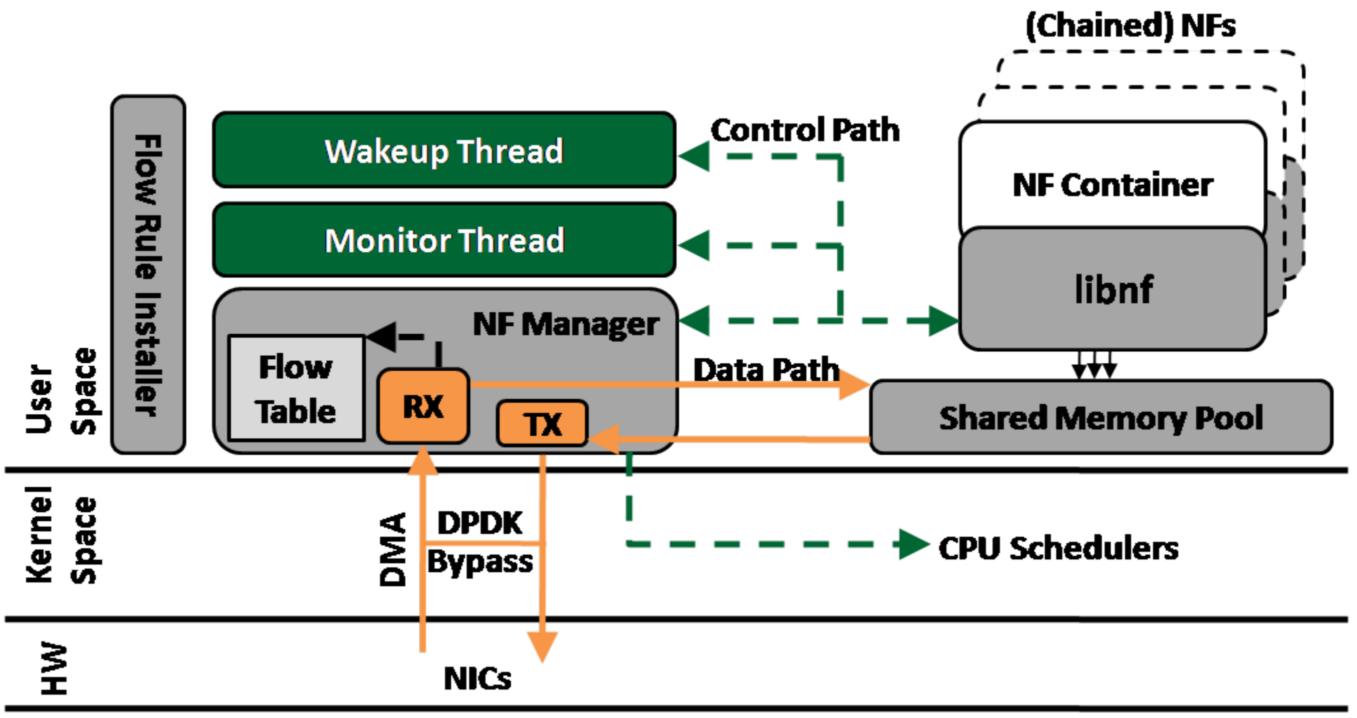
Work-conserving and proportional scheduling *(within each core)*

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

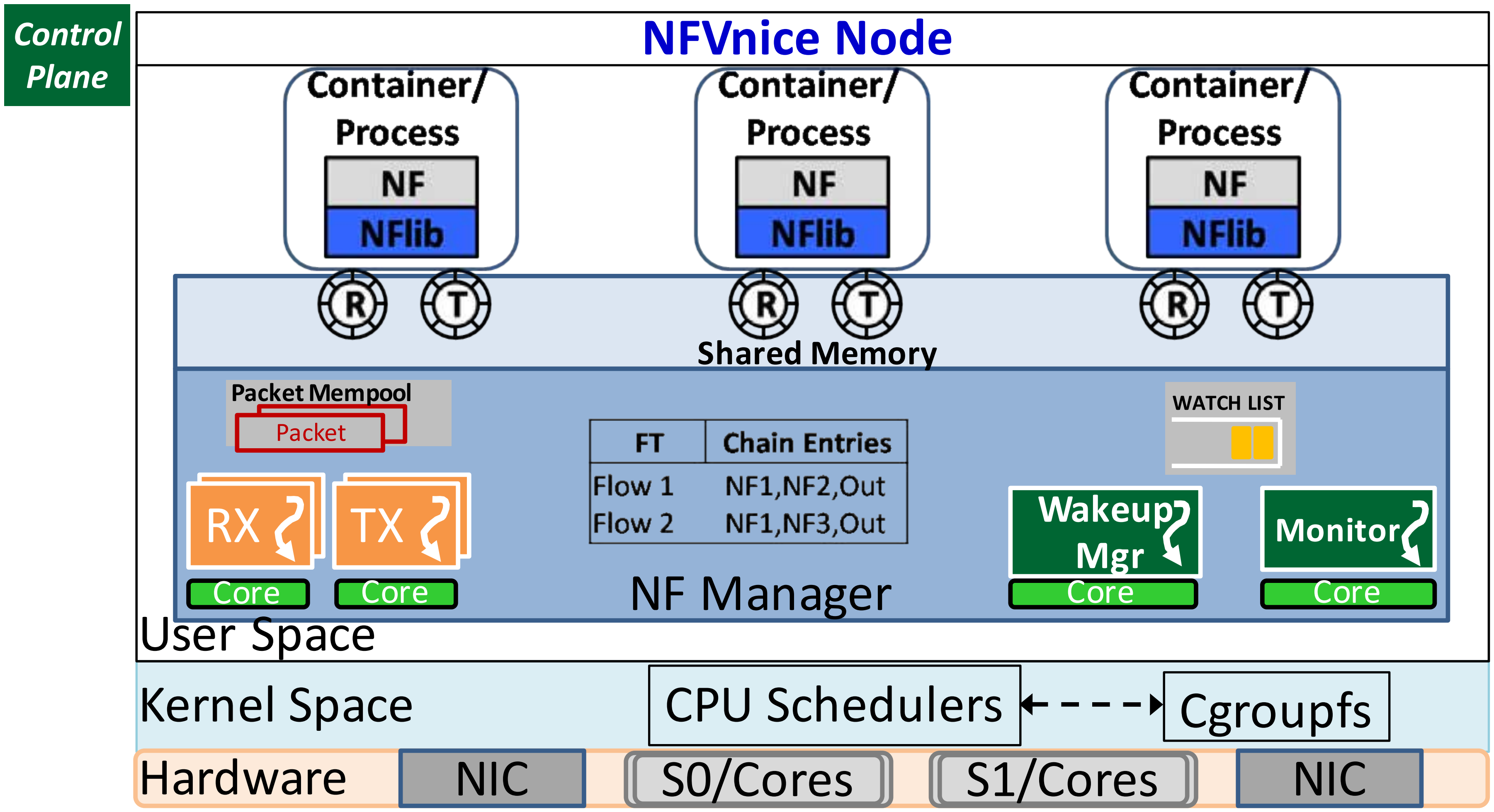
End-2-End Bottleneck/congestion control *(across nodes)*

Efficient Disk I/O Mgmt. Library

NFVnice: Implementation

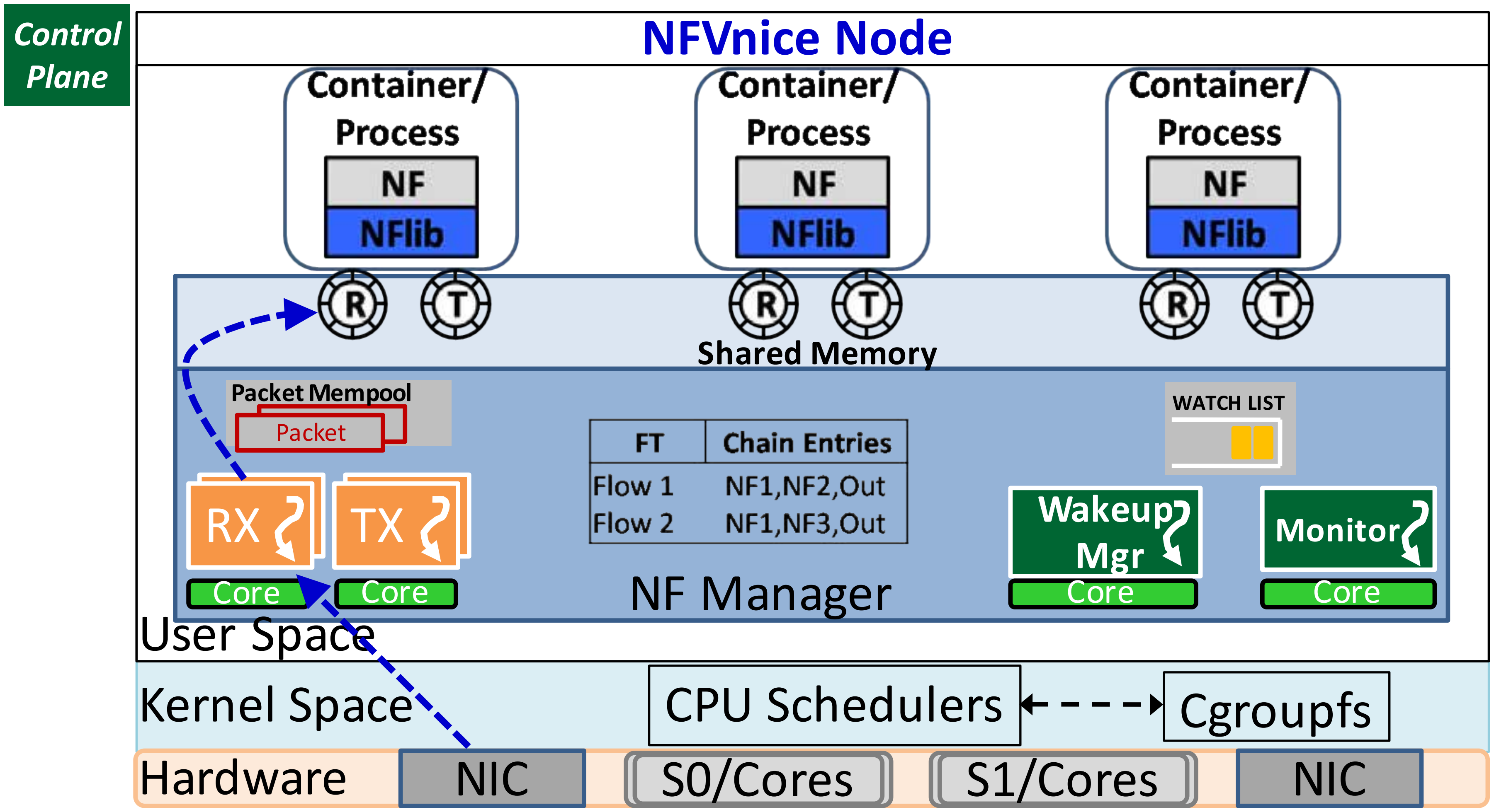


- OpenNetVM [HMBox’16, NSDI’14] makes use of DPDK to enable fast packet processing.
- NFVnice extends the **Data** and **Control** plane functionalities to facilitate efficient multiplexing and scheduling of NFs on same core.

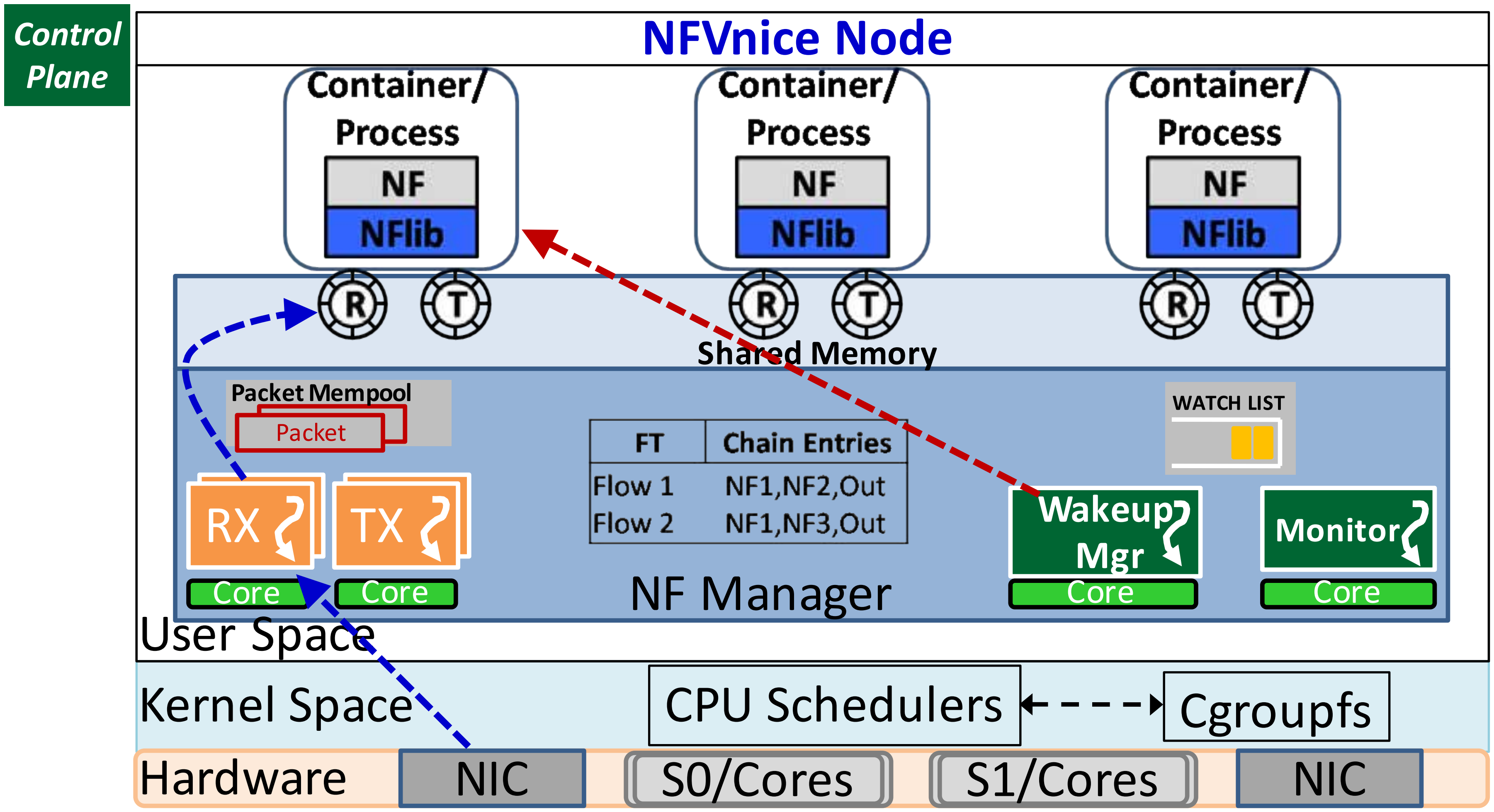


FT	Chain Entries
Flow 1	NF1,NF2,Out
Flow 2	NF1,NF3,Out

- Resource monitoring and control functions.

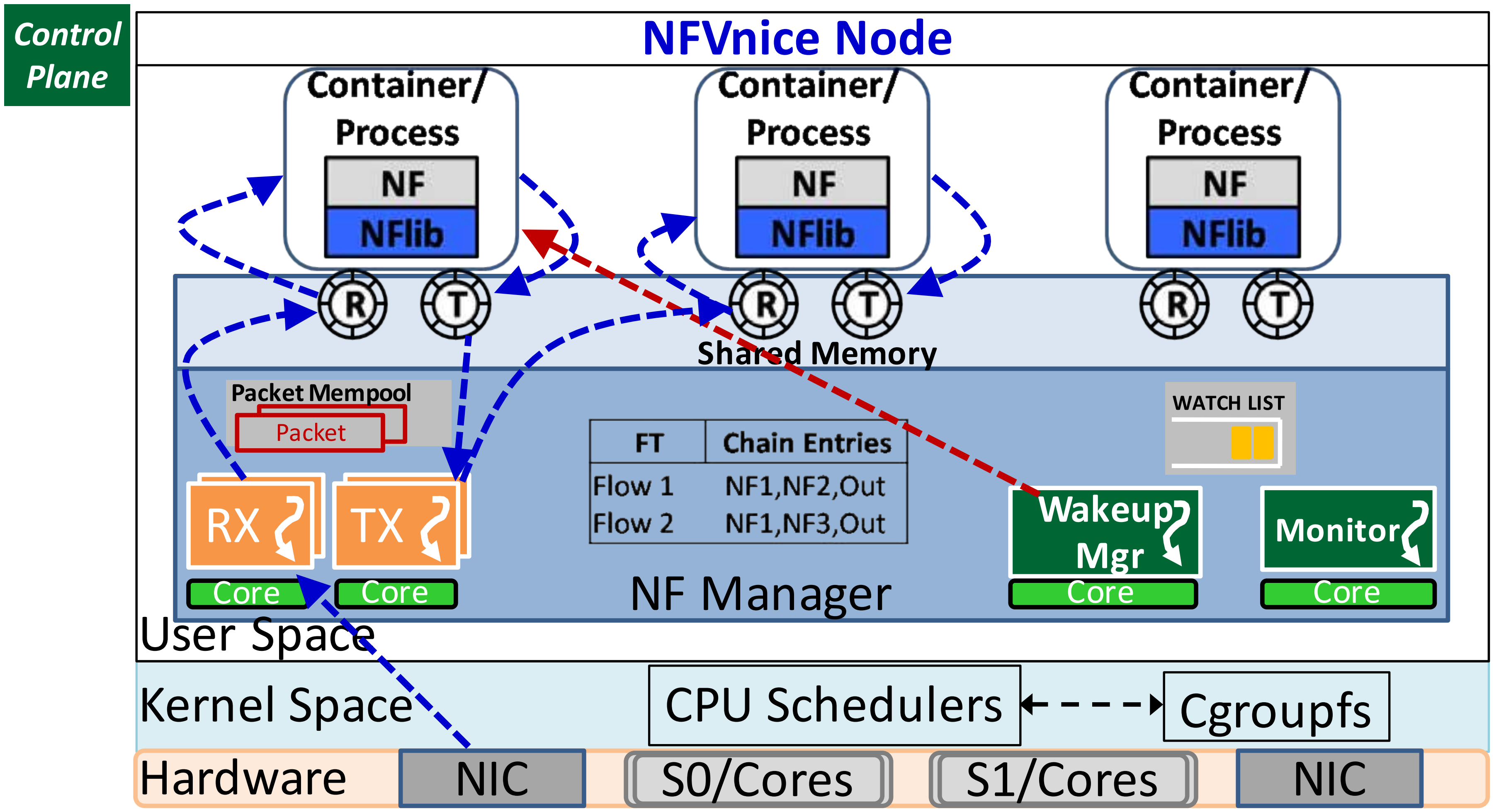


- Resource monitoring and control functions.
  - ***Wakeup Thread***
    - Wakeup notification to the NFs
    - Timer Management.

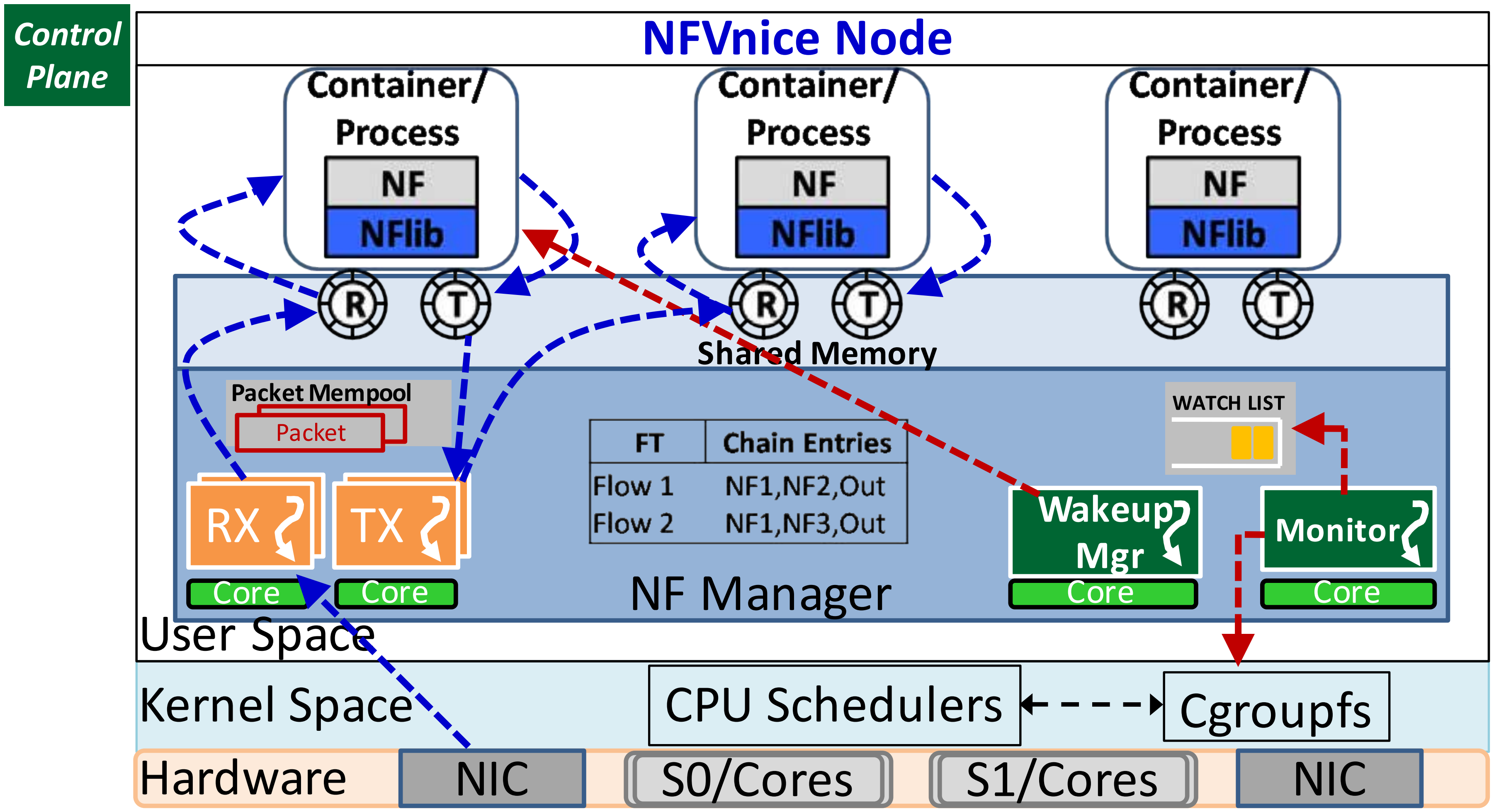


FT	Chain Entries
Flow 1	NF1,NF2,Out
Flow 2	NF1,NF3,Out

- Resource monitoring and control functions.
  - ***Wakeup Thread***
    - Wakeup notification to the NFs
    - Timer Management.



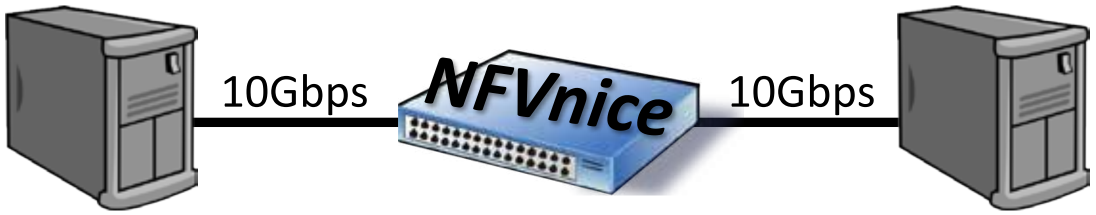
- Resource monitoring and control functions.
  - ***NF threads (libnf)***
    - Voluntary yield decisions.
    - Estimate per-packet processing cost.



- Resource monitoring and control functions.
  - ***Monitor Thread***
    - periodically (1*ms*) monitors NF load.
    - computes the cpu share for each core.
    - Tracks EWMA of NFs Rx queue length and mark ECN.

Evaluation

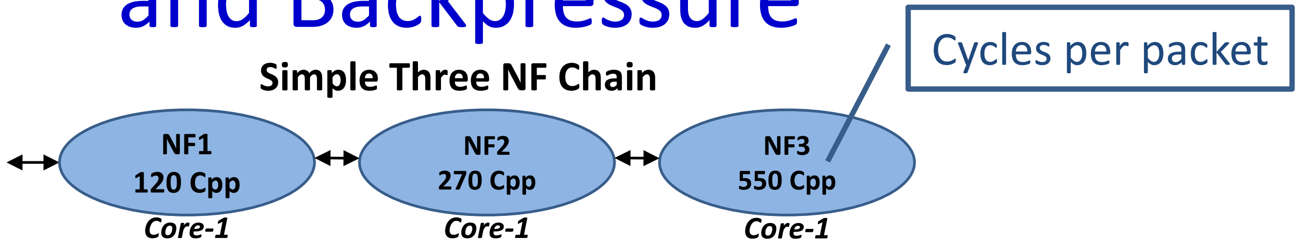
- Testbed:
  - Hardware: 3 Intel Xeon(R) CPU E5-2697, 28 cores @2.6Ghz servers, with dual port 10Gbps DPDK compatible NICs.
  - Software: Linux kernel 3.19.0-39-lowlatency profile.
  - NFVnice: built on top of OpenNetVM.

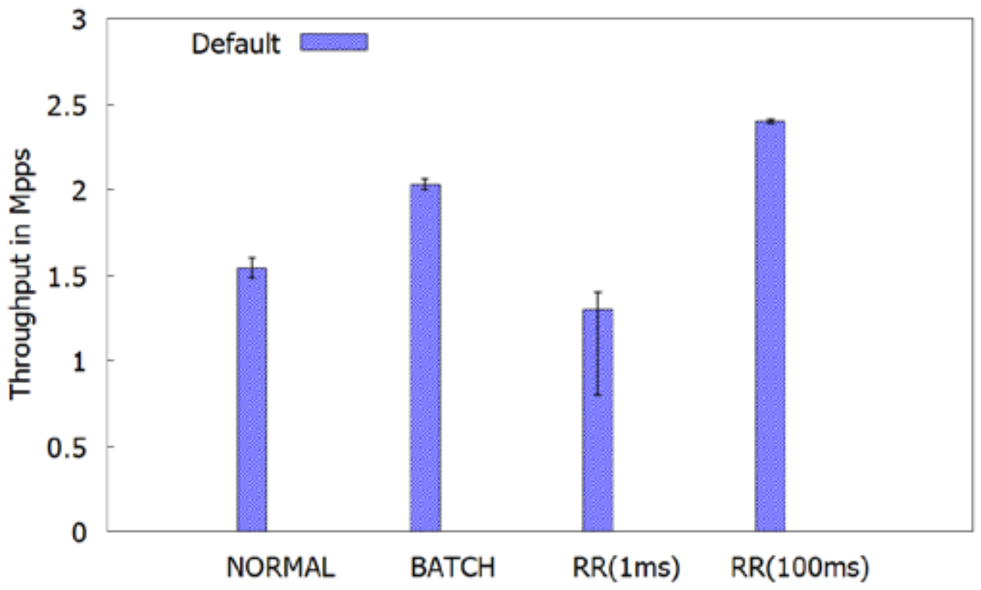


- Traffic:
  - Pktgen and Moongen: Line rate traffic (64 byte packets).
  - Iperf: TCP flows.

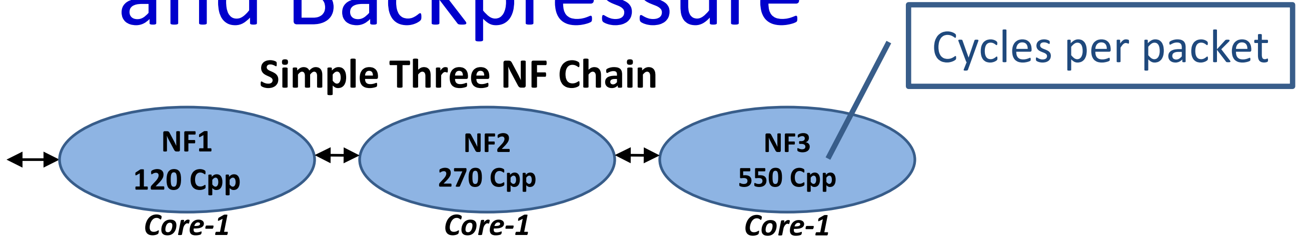
- Schemes compared:
  - Native Linux Schedulers with and w/o NFVnice.
  - Different NFs (varying computation costs) and chain configurations.

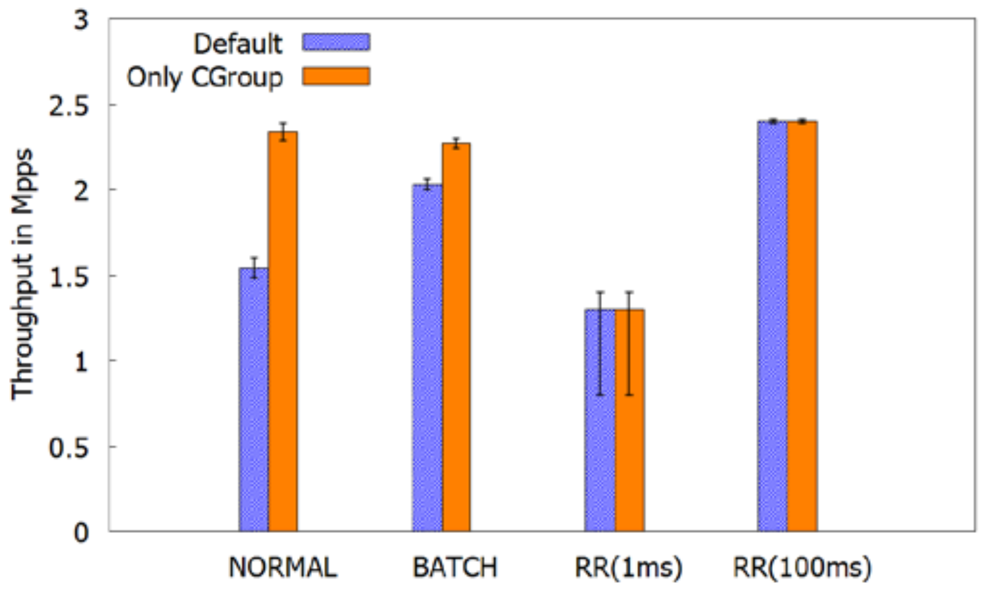
Performance: Impact of cgroup weights and Backpressure



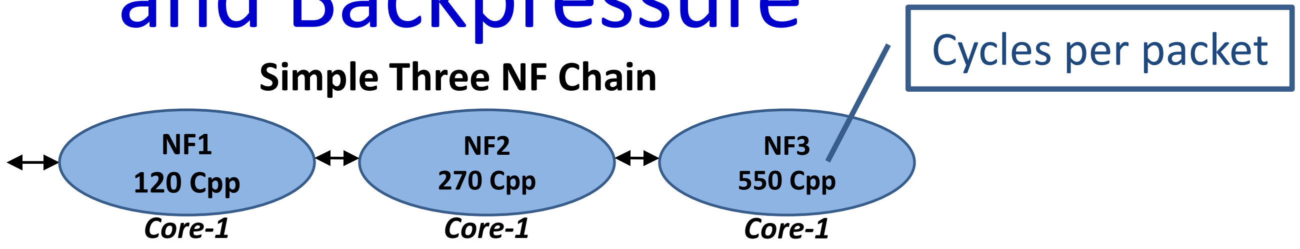


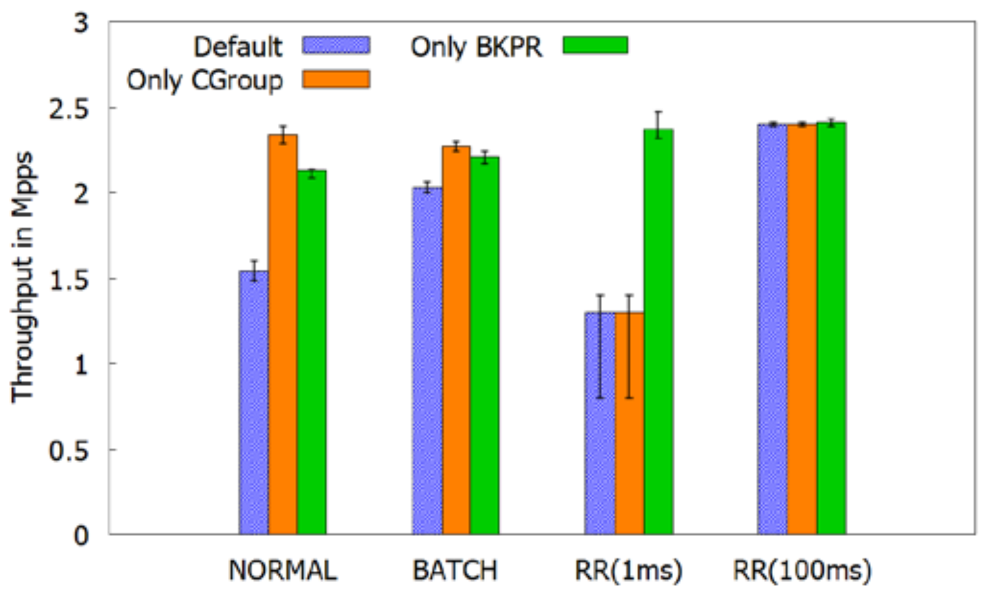
Performance: Impact of cgroup weights and Backpressure



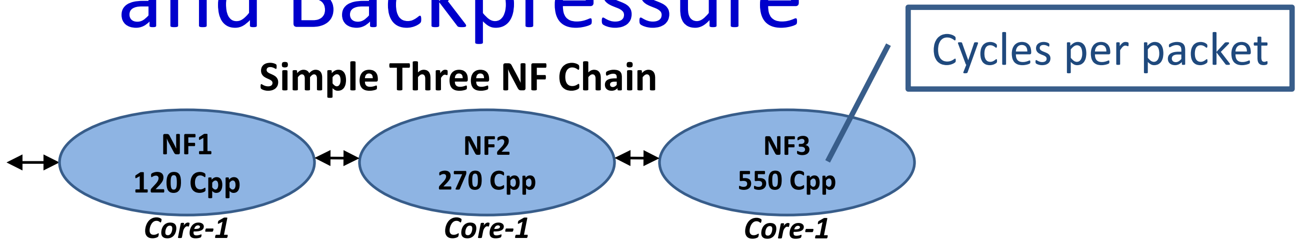


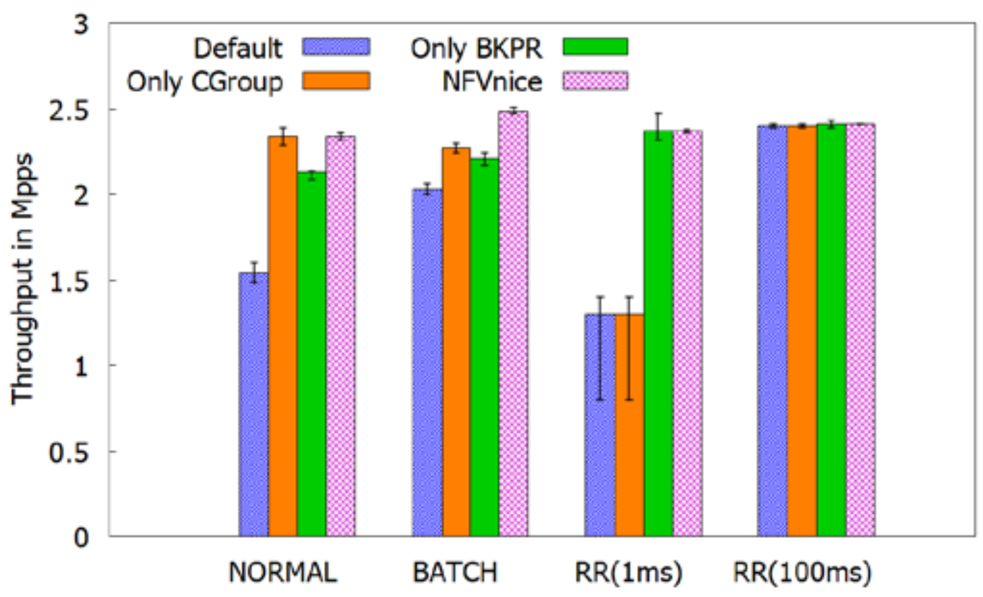
Performance: Impact of cgroup weights and Backpressure





Performance: Impact of cgroup weights and Backpressure

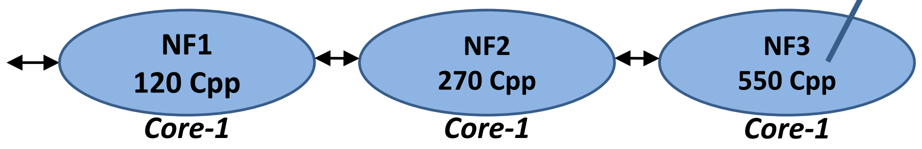


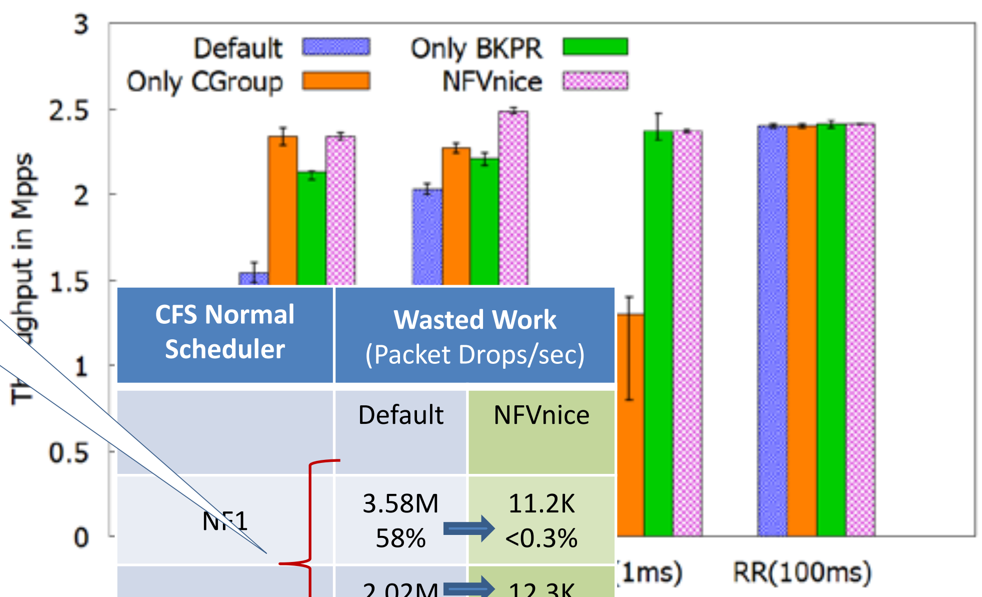


Performance: Impact of cgroup weights and Backpressure

Cycles per packet

Simple Three NF Chain

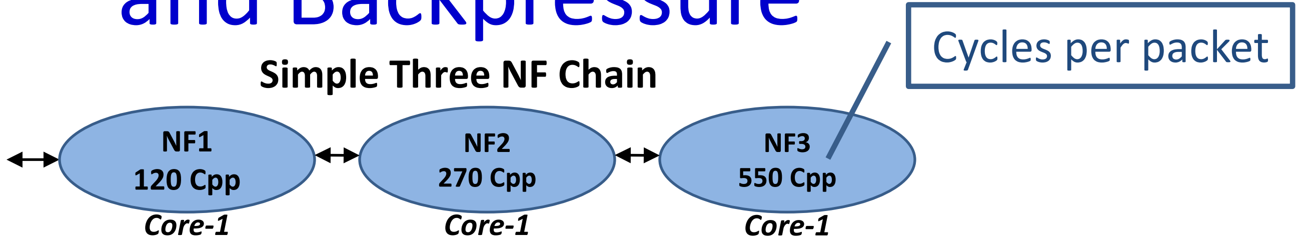


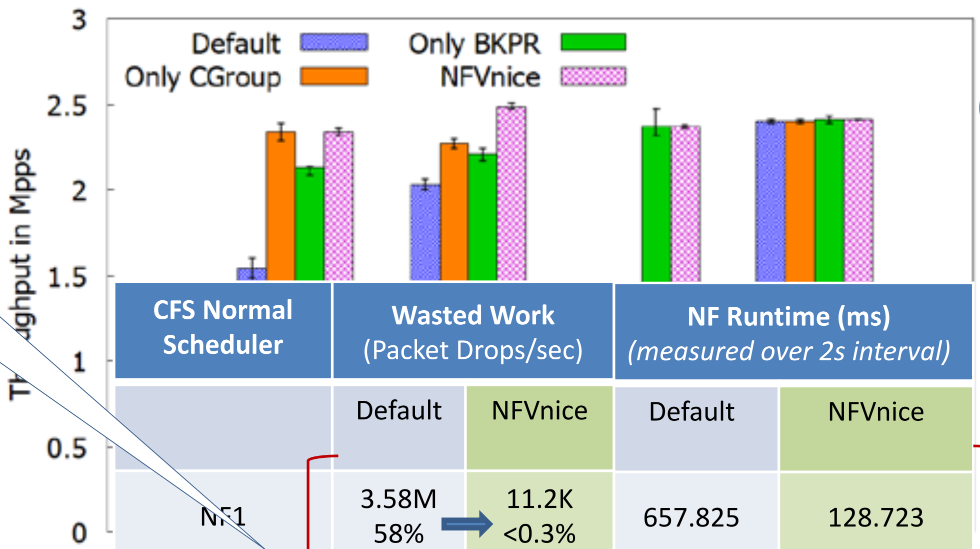


Significant Reduction in Wasted Work!

CFS Normal Scheduler	Wasted Work (Packet Drops/sec)	
	Default	NFVnice
NF1	3.58M 58%	11.2K <0.3%
NF2	2.02M 50%	12.3K <0.4%
NF3	-	-

Performance: Impact of cgroup weights and Backpressure





Significant Reduction in Wasted Work!

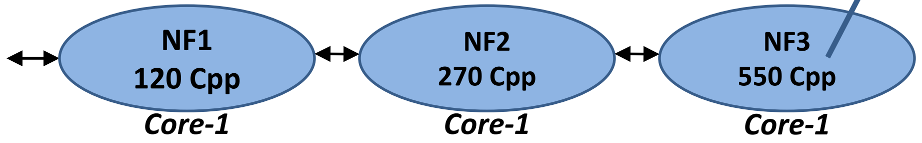
CFS Normal Scheduler	Wasted Work (Packet Drops/sec)		NF Runtime (ms) *(measured over 2s interval)*	
	Default	NFVnice	Default	NFVnice
NF1	3.58M 58%	11.2K <0.3%	657.825	128.723
NF2	2.02M 50%	12.3K <0.4%	602.285	848.922
NF3	-	-	623.797	1014.218

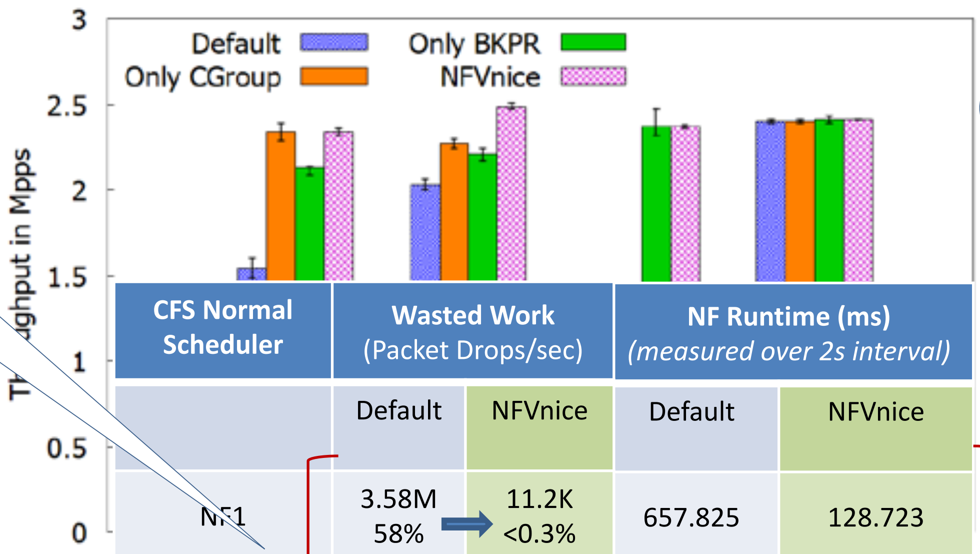
CPU Allocation α Computation Cost

Performance: Impact of cgroup weights and Backpressure

Cycles per packet

Simple Three NF Chain





Significant Reduction in Wasted Work!

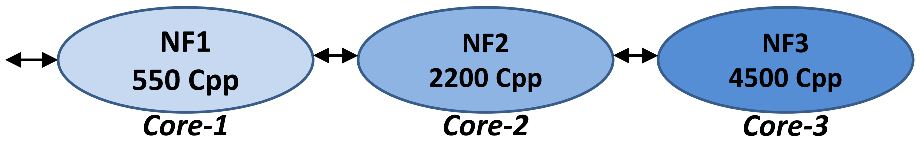
CFS Normal Scheduler	Wasted Work (Packet Drops/sec)		NF Runtime (ms) *(measured over 2s interval)*	
	Default	NFVnice	Default	NFVnice
NF1	3.58M 58%	11.2K <0.3%	657.825	128.723
NF2	2.02M	12.3K	602.285	848.922

CPU Allocation α Computation Cost

NFVnice improves throughput for all kernel schedulers.

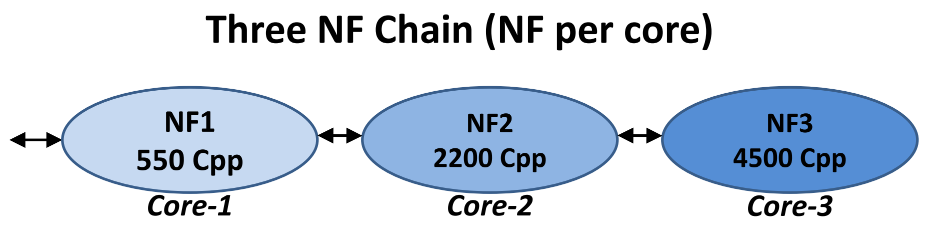
Efficient Resource (CPU) Utilization

Three NF Chain (NF per core)



	Default		
	Svc. Rate	Drop rate	CPU Util
NF1 (550cycles)	5.95Mpps	4.76Mpps 80%	100%
NF2 (2200cycles)	1.18Mpps	0.58Mpps 49%	100%
NF3 (4500cycles)	0.6Mpps	-	100%
Aggregate	0.6Mpps	-	

Efficient Resource (CPU) Utilization

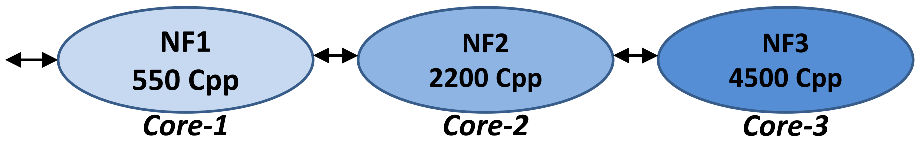


Lots of Wasted Work!
Burning CPU!!

	Default		
	Svc. Rate	Drop rate	CPU Util
NF1 (550cycles)	5.95Mpps	4.76Mpps 80%	100%
NF2 (2200cycles)	1.18Mpps	0.58Mpps 49%	100%
NF3 (4500cycles)	0.6Mpps	-	100%
Aggregate	0.6Mpps	-	

Efficient Resource (CPU) Utilization

Three NF Chain (NF per core)

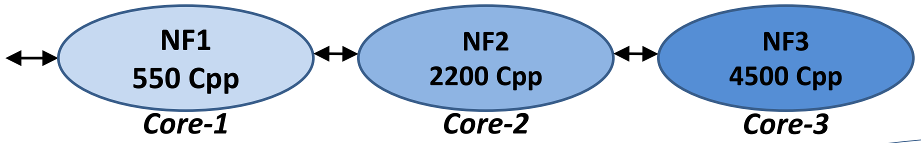


Lots of Wasted Work! Burning CPU!!

	Default			NFVnice		
	Svc. Rate	Drop rate	CPU Util	Svc. Rate	Drop rate	CPU Util
NF1 (550cycles)	5.95Mpps	4.76Mpps 80%	100%	0.82Mpps	0.15Mpps	**11%**
NF2 (2200cycles)	1.18Mpps	0.58Mpps 49%	100%	0.72Mpps	0.07Mpps	**64%**
NF3 (4500cycles)	0.6Mpps	-	100%	0.6Mpps	-	100%
Aggregate	0.6Mpps	-		0.6Mpps	-	

Efficient Resource (CPU) Utilization

Three NF Chain (NF per core)



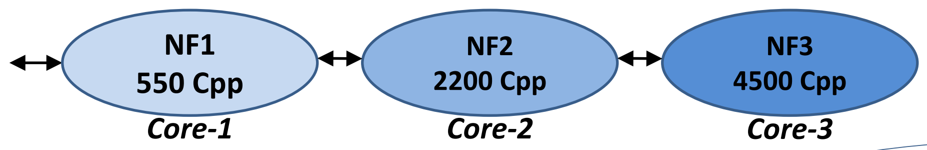
Lots of Wasted Work! Burning CPU!!

Avoid Wasted Work! Optimize CPU utilization.

	Default			NFVnice		
	Svc. Rate	Drop rate	CPU Util	Svc. Rate	Drop rate	CPU Util
NF1 (550cycles)	5.95Mpps	4.76Mpps 80%	100%	0.82Mpps	0.15Mpps	**11%**
NF2 (2200cycles)	1.18Mpps	0.58Mpps 49%	100%	0.72Mpps	0.07Mpps	**64%**
NF3 (4500cycles)	0.6Mpps	-	100%	0.6Mpps	-	100%
Aggregate	0.6Mpps	-		0.6Mpps	-	

Efficient Resource (CPU) Utilization

Three NF Chain (NF per core)



Lots of Wasted Work! Burning CPU!!

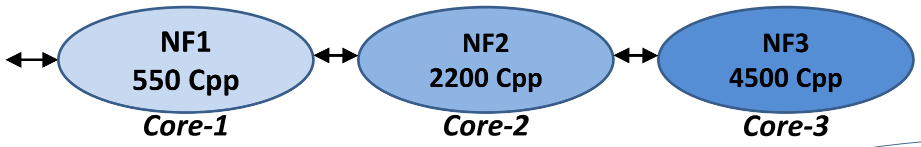
Achieves Same Throughput

Avoid Wasted Work! Optimize CPU utilization.

	Default			**NFVnice**		
	Svc. Rate	Drop rate	CPU Util	Svc. Rate	Drop rate	CPU Util
NF1 (550cycles)	5.95Mpps	4.76Mpps 80%	100%	[illegible]2Mpps	0.15Mpps	**11%**
NF2 (2200cycles)	1.18Mpps	0.58Mpps 49%	100%	0.7[illegible]Mpps	0.07Mpps	**64%**
NF3 (4500cycles)	0.6Mpps	-	100%	0.6Mpps	-	100%
Aggregate	0.6Mpps	-		0.6Mpps	-	

Efficient Resource (CPU) Utilization

Three NF Chain (NF per core)



Lots of Wasted Work! Burning CPU!!

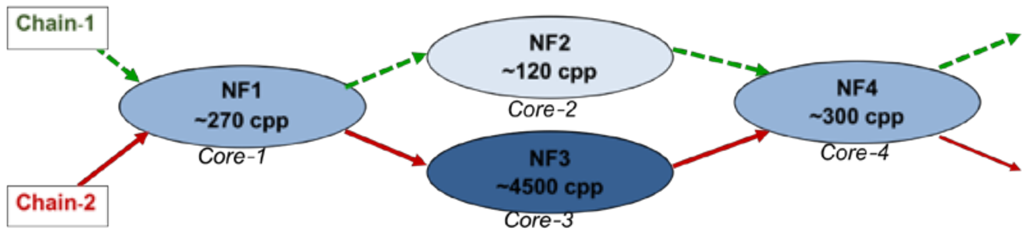
Achieves Same Throughput

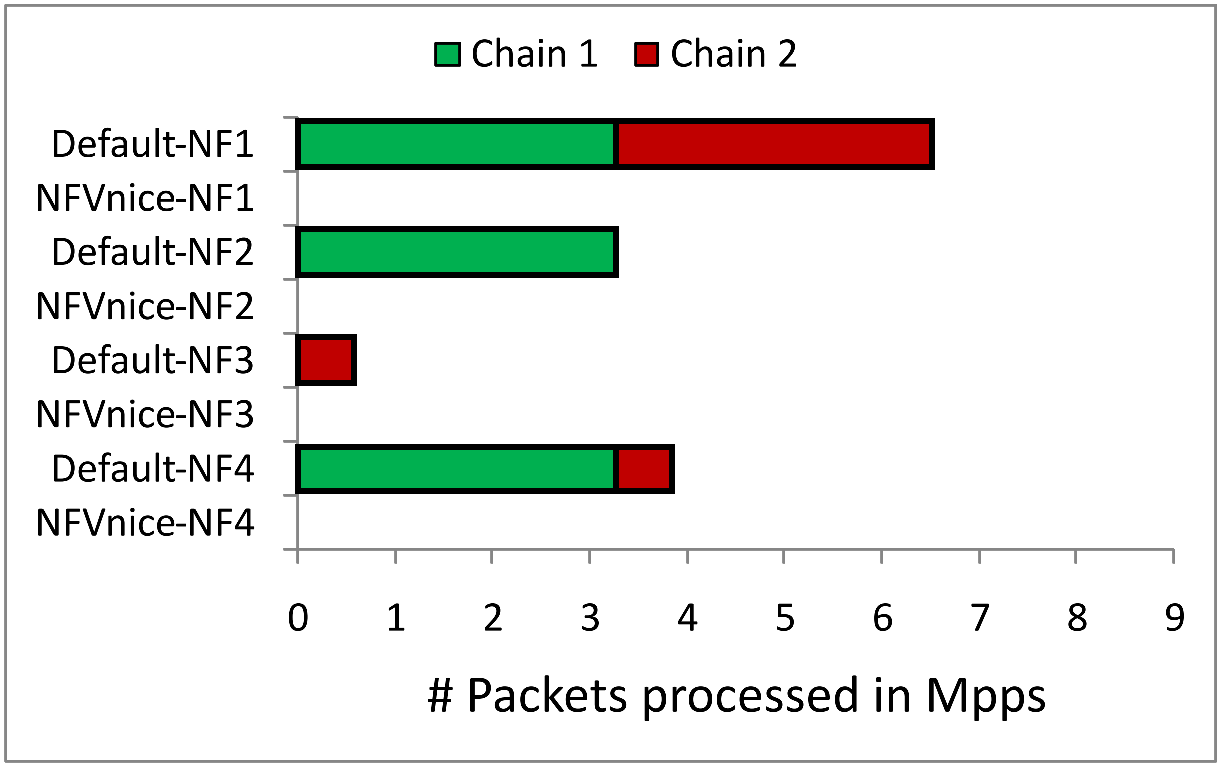
Avoid Wasted Work! Optimize CPU utilization.

	Default			NFVnice		
	Svc. Rate	Drop rate	CPU Util	Svc. Rate	Drop rate	CPU Util
NF1 (550cycles)	5.95Mpps	4.76Mpps 80%	100%	32Mpps	0.15Mpps	**11%**
NF2 (2200cycles)	1.18Mpps	0.58Mpps 49%	100%	0.72Mpps	0.07Mpps	**64%**
Aggregate	0.6Mpps	-		0.6Mpps	-	

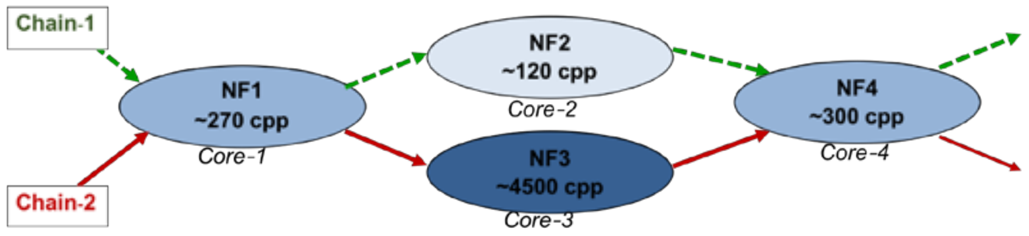
NFVnice avoids wasted work; provides better resource utilization

Performance + Resource Utilization

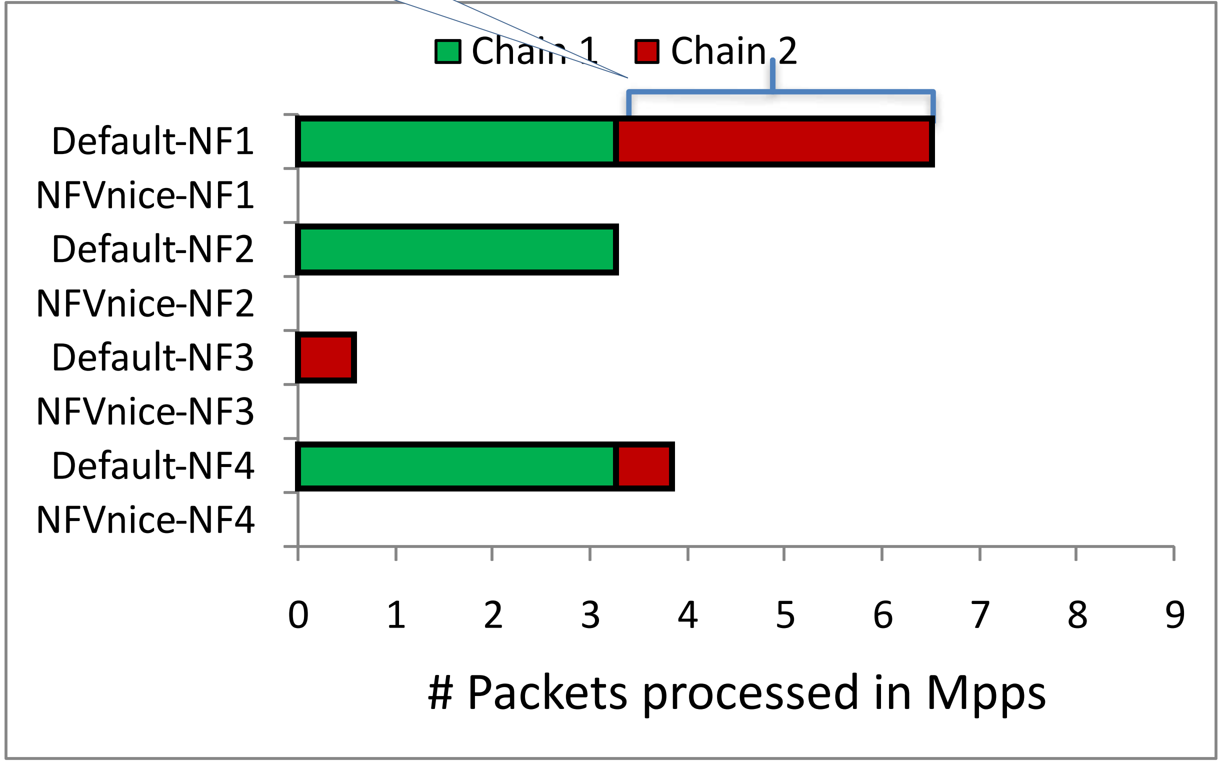




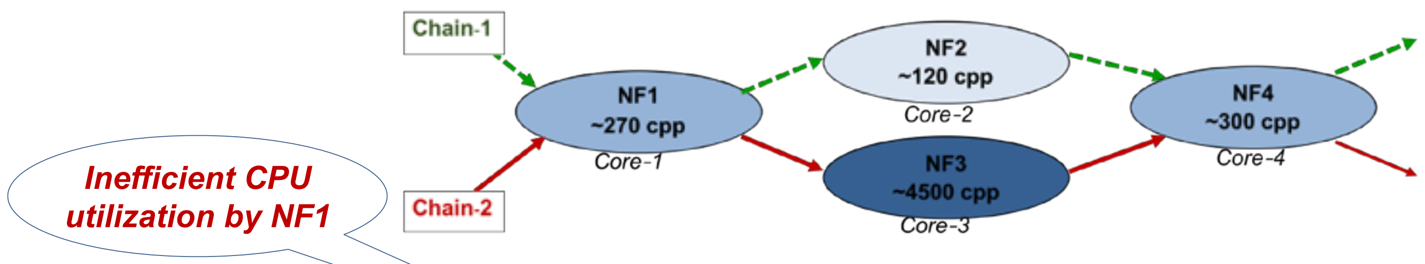
Performance + Resource Utilization

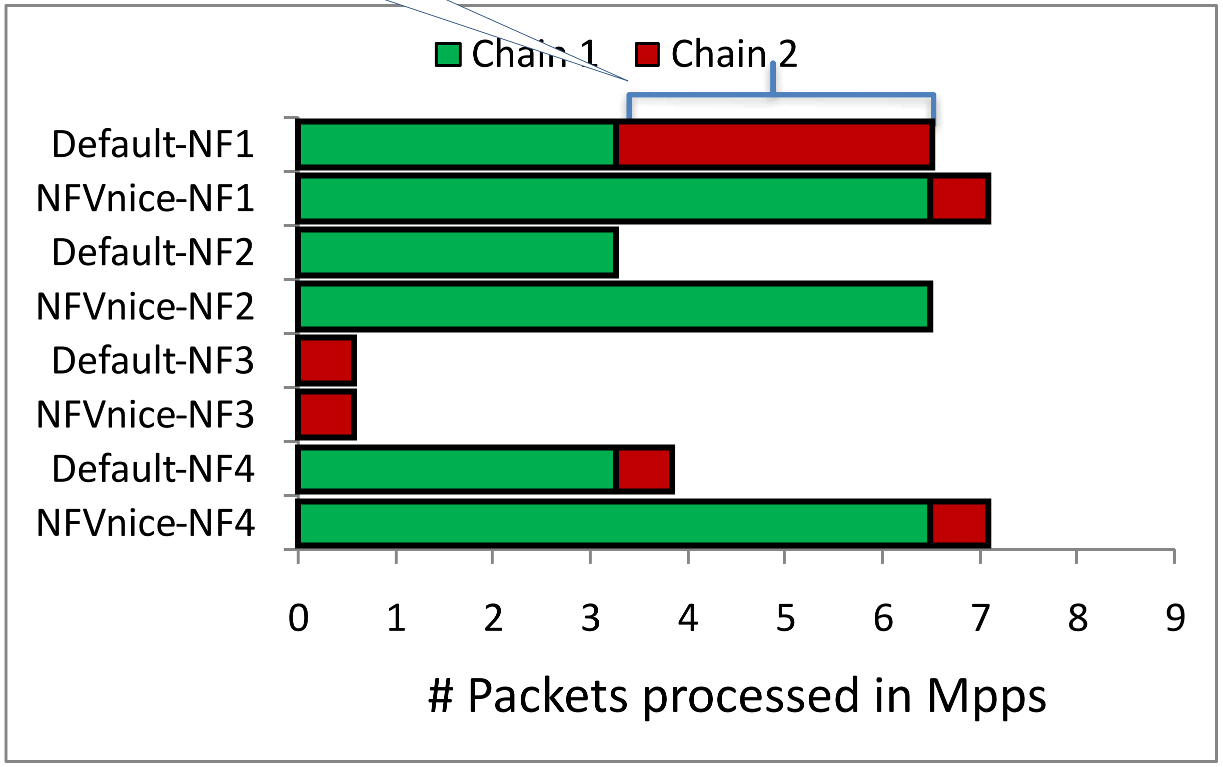


Inefficient CPU utilization by NF1

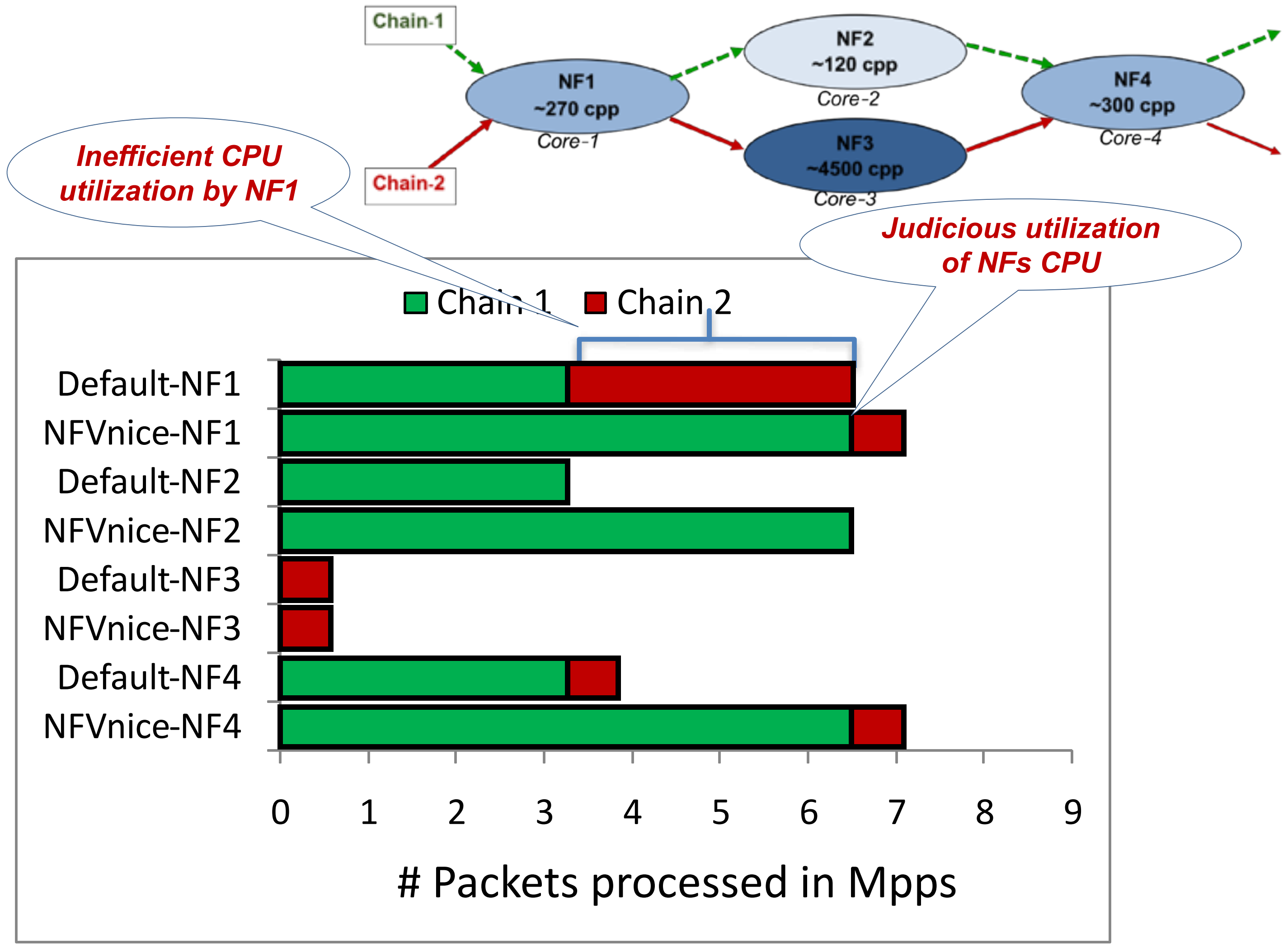


Performance + Resource Utilization

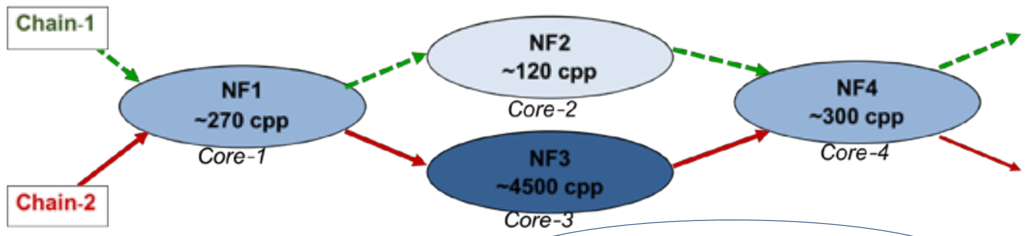




Performance + Resource Utilization

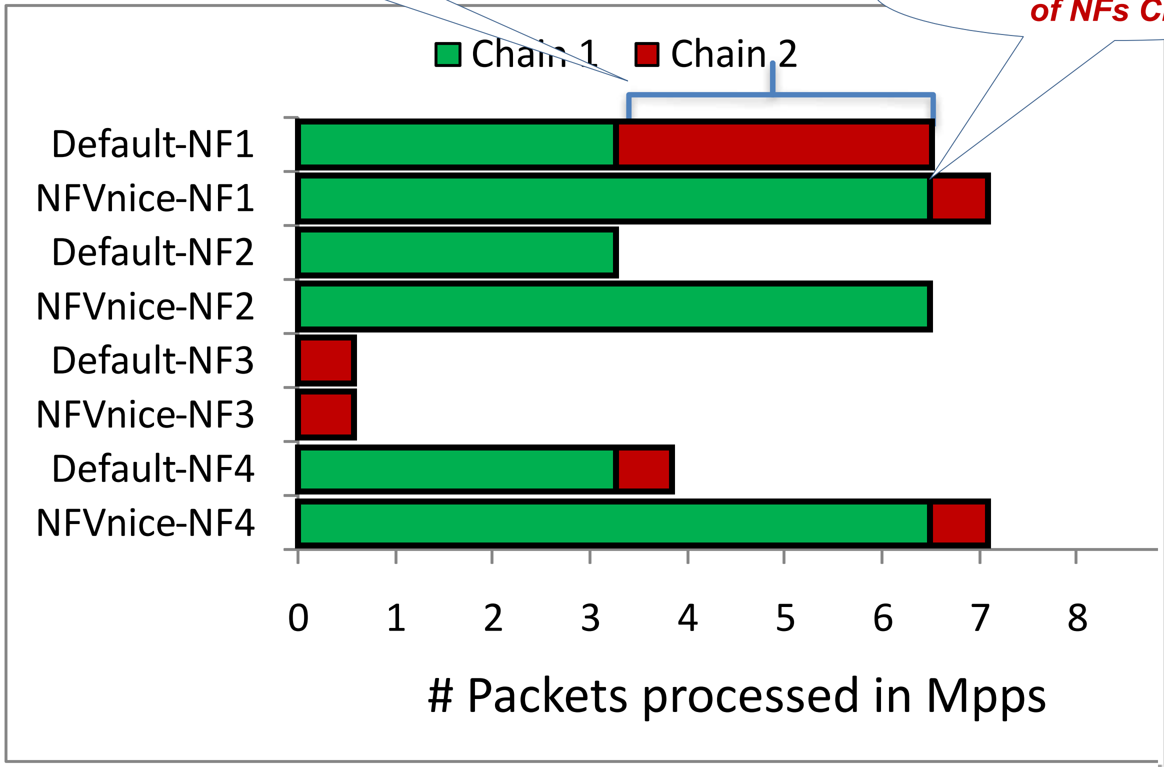


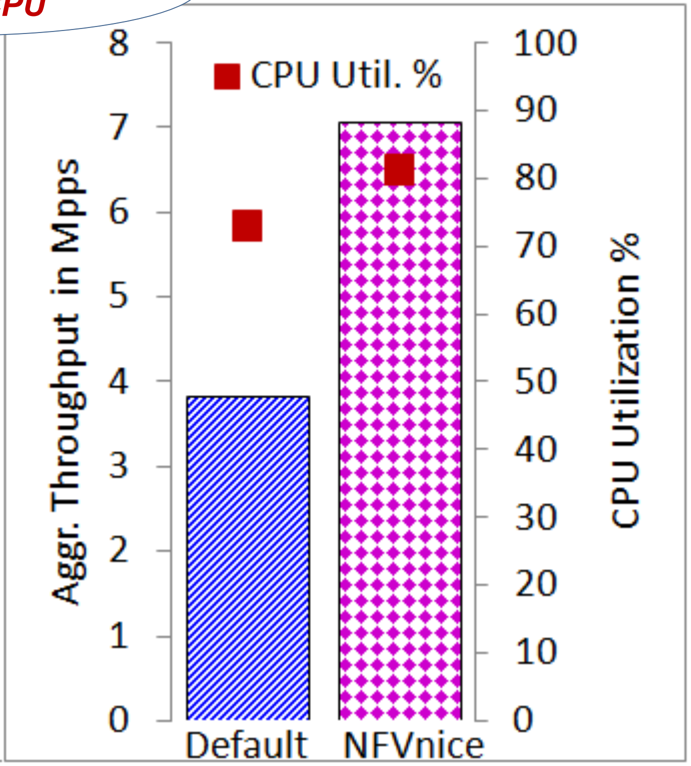
Performance + Resource Utilization



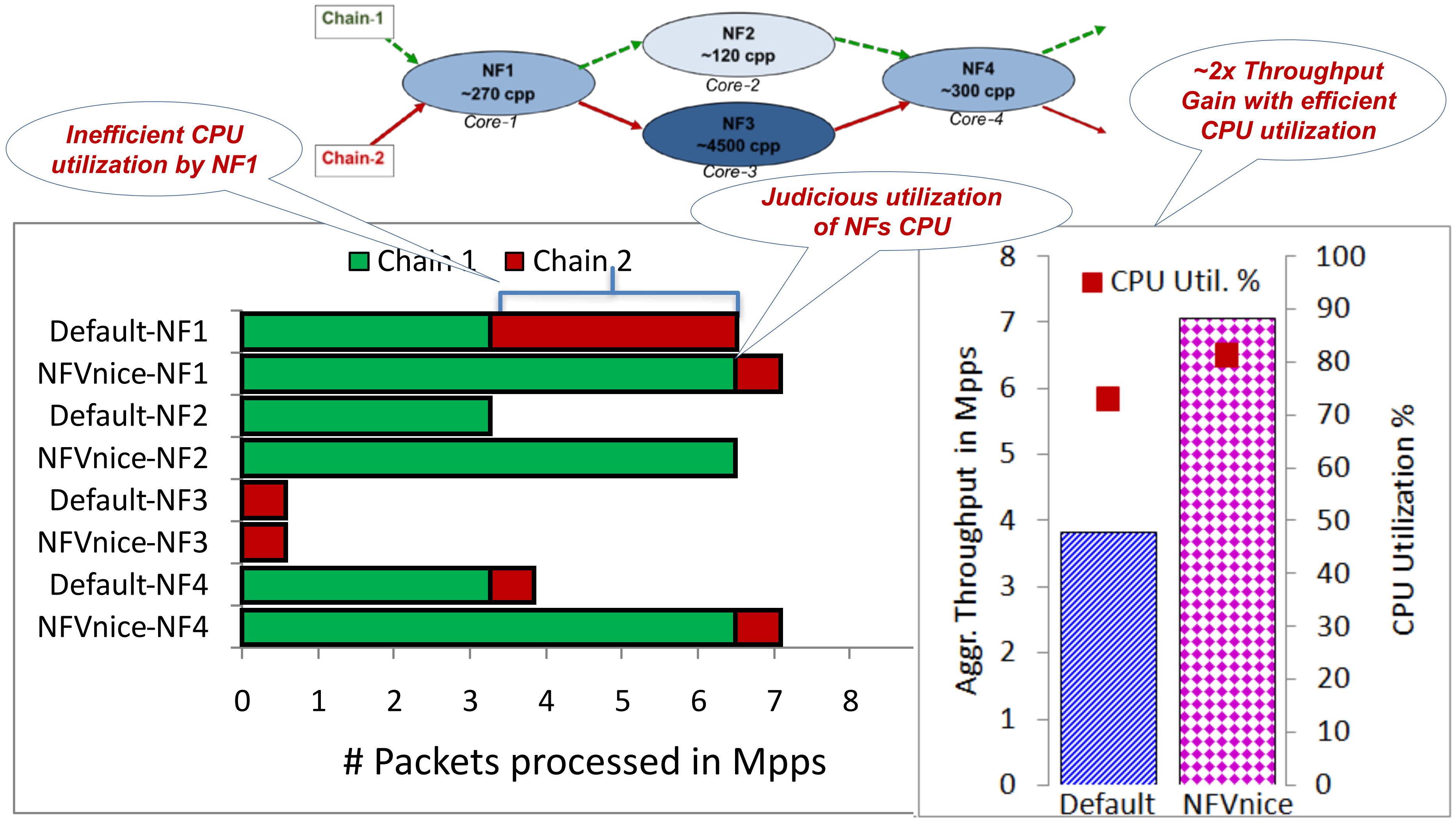
Inefficient CPU utilization by NF1

Judicious utilization of NFs CPU

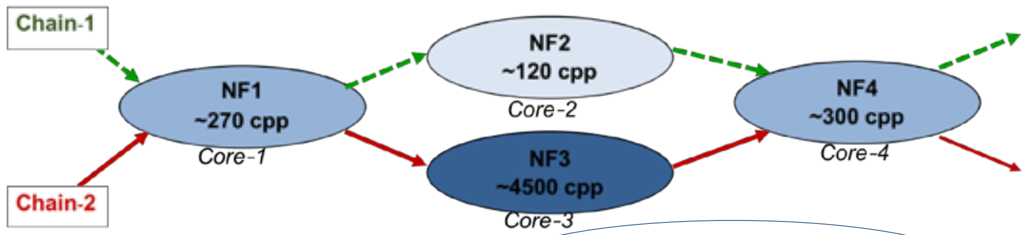




Performance + Resource Utilization



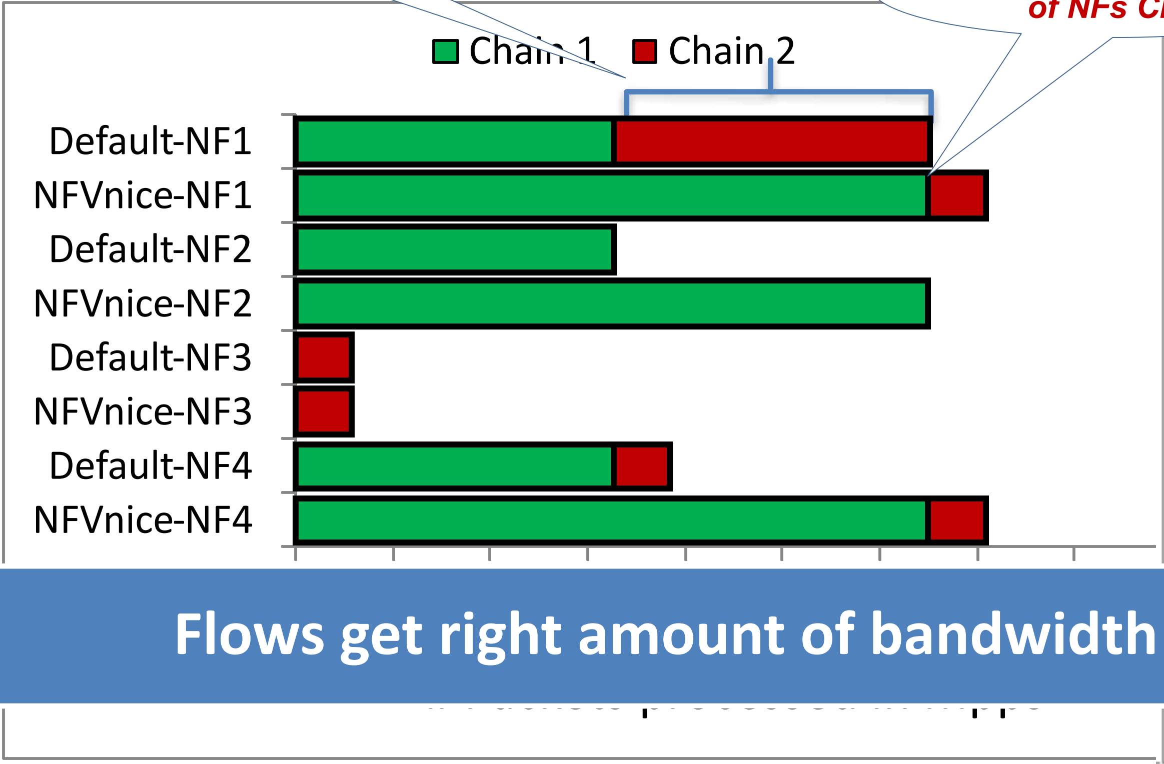
Performance + Resource Utilization

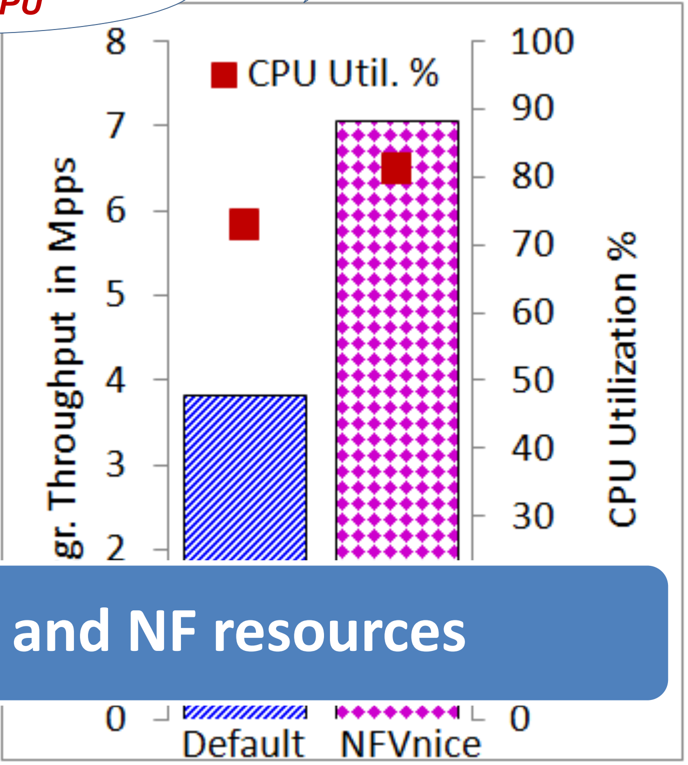


Inefficient CPU utilization by NF1

Judicious utilization of NFs CPU

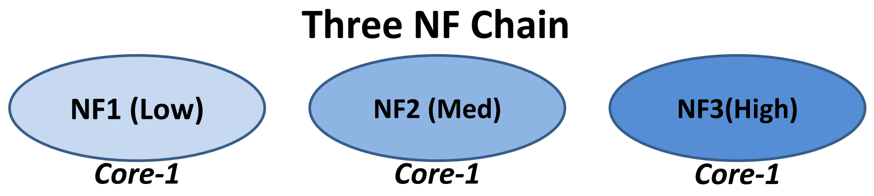
~2x Throughput Gain with efficient CPU utilization



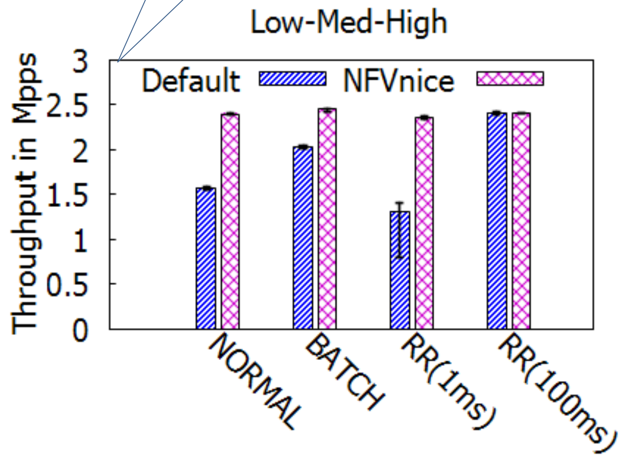


Flows get right amount of bandwidth and NF resources

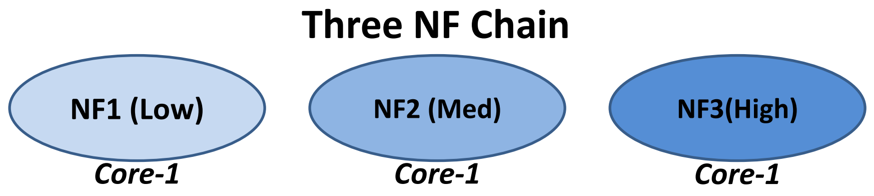
Robustness: Chain Diversity



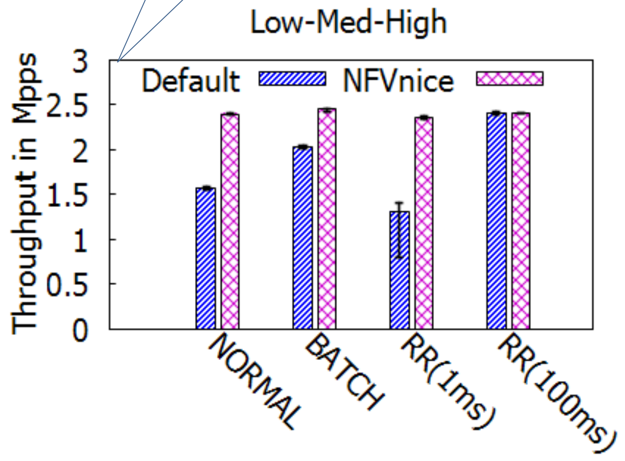
Robust across schedulers

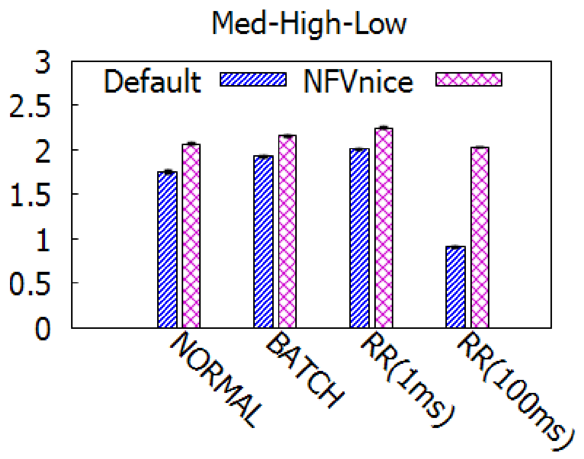


Robustness: Chain Diversity



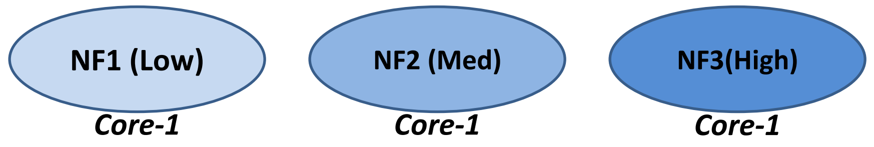
Robust across schedulers





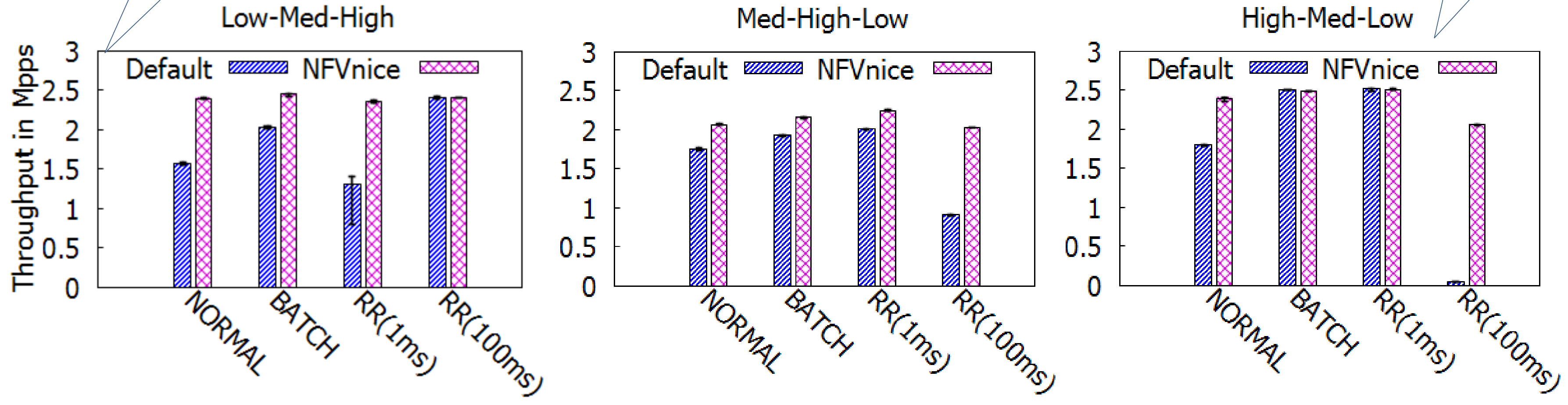
Robustness: Chain Diversity

Three NF Chain

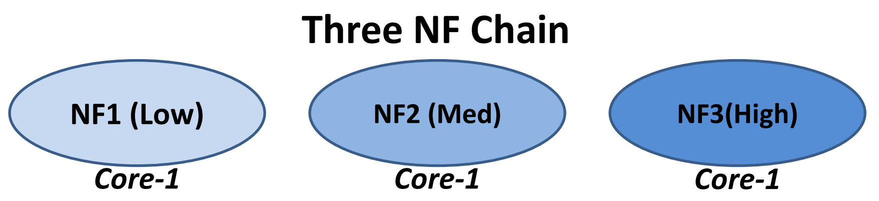


Robust across schedulers

Robust across chain diversity

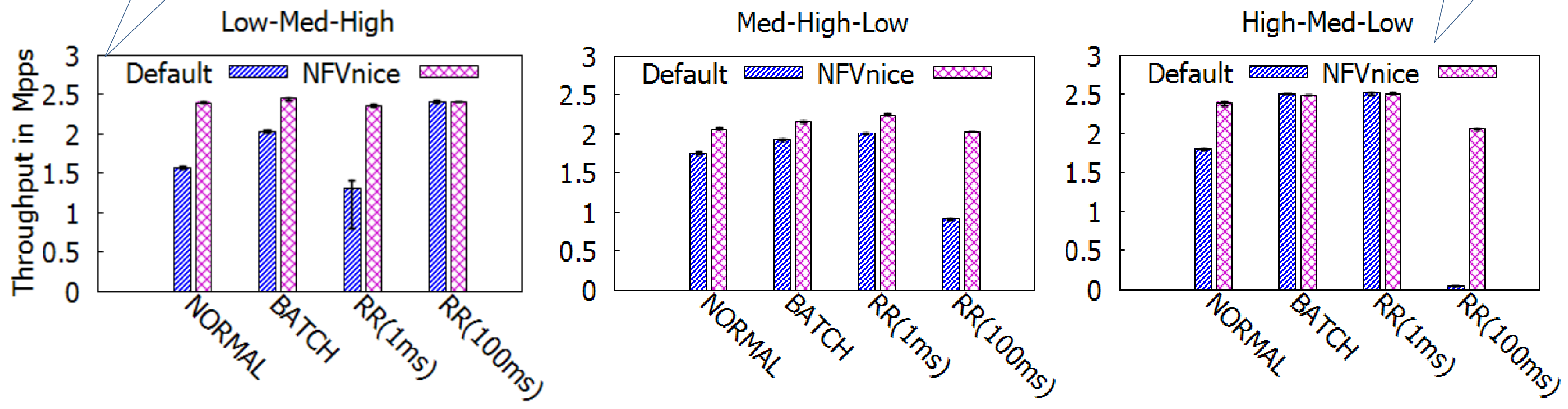


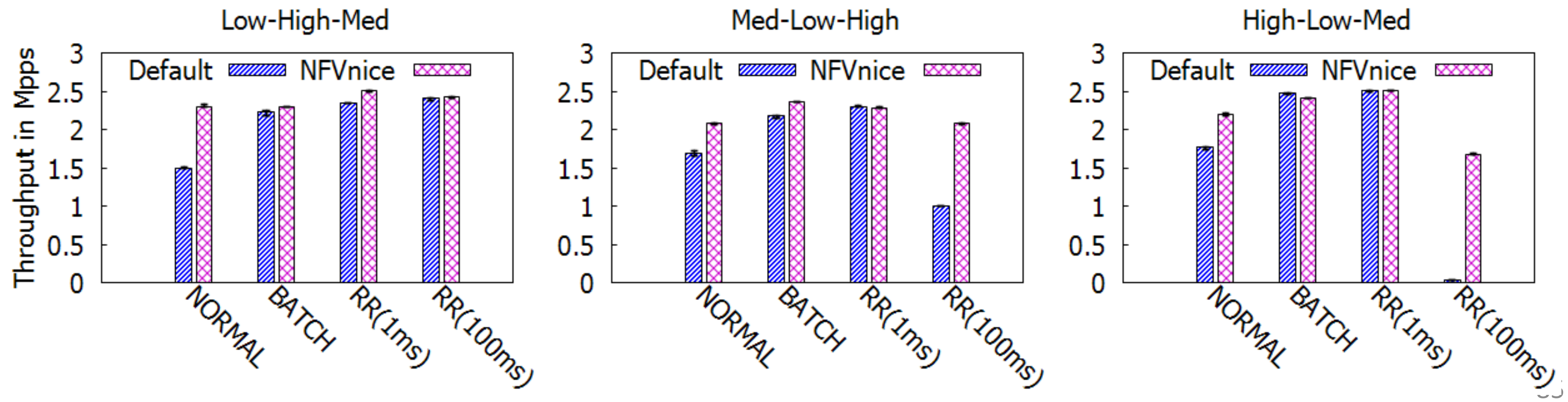
Robustness: Chain Diversity



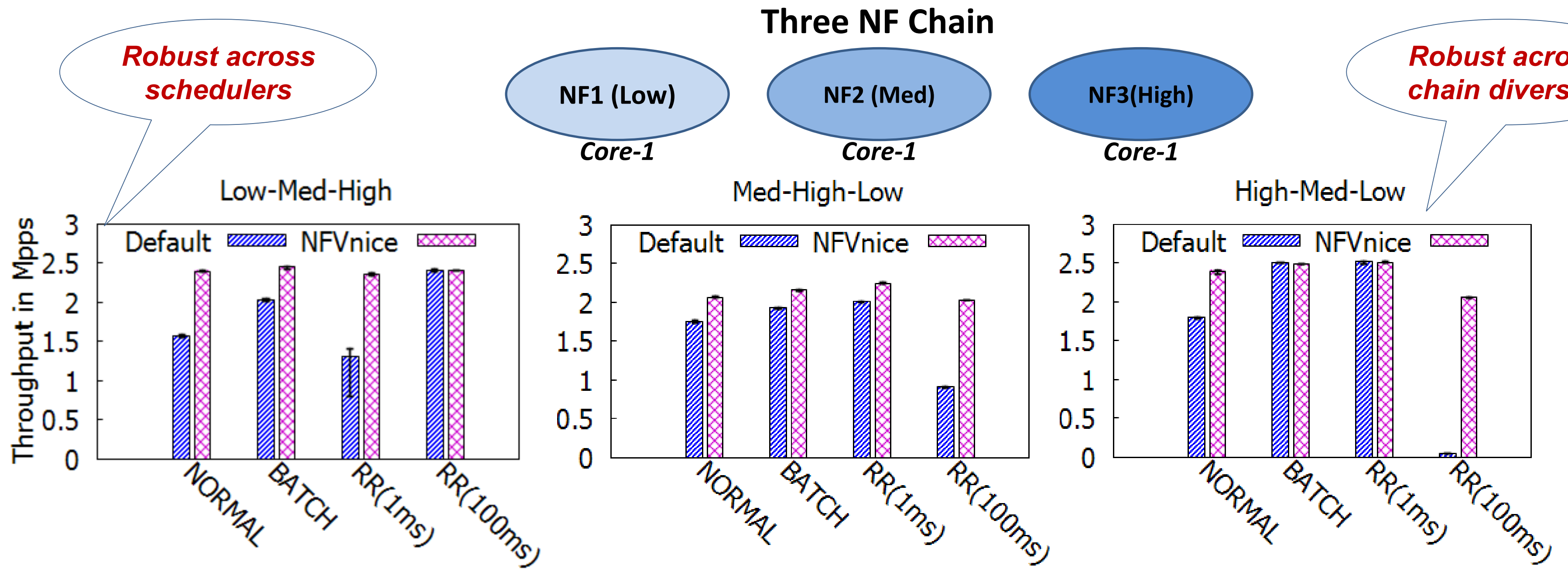
Robust across schedulers

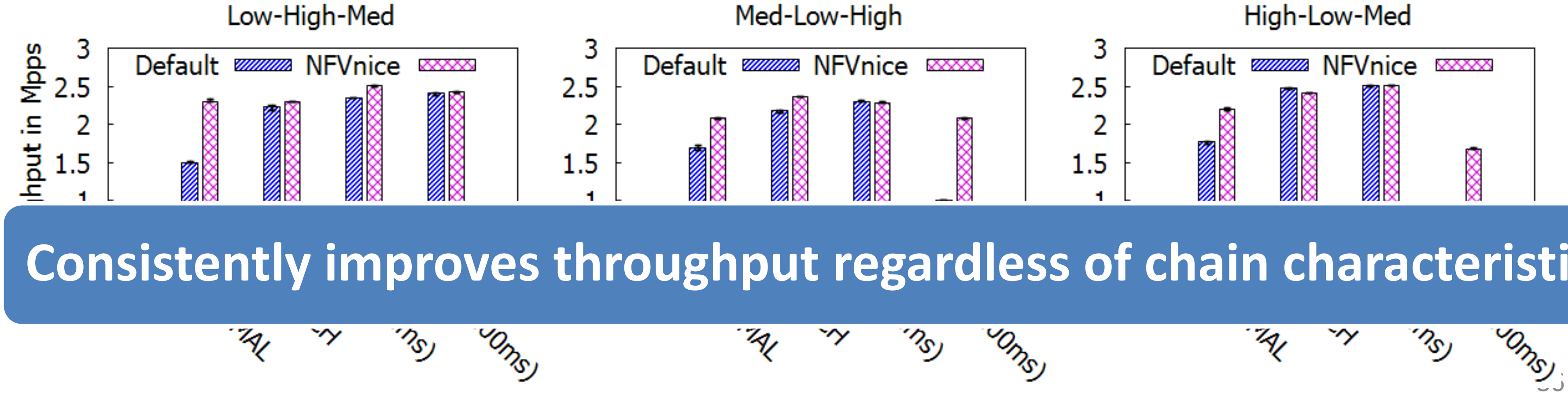
Robust across chain diversity





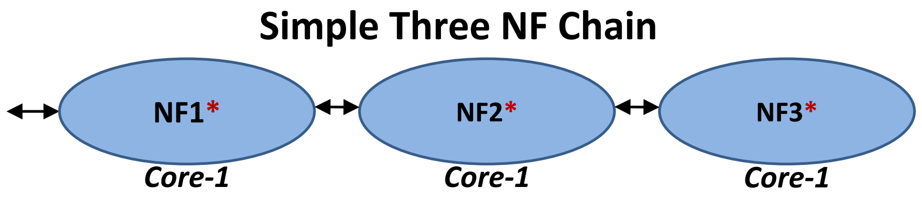
Robustness: Chain Diversity



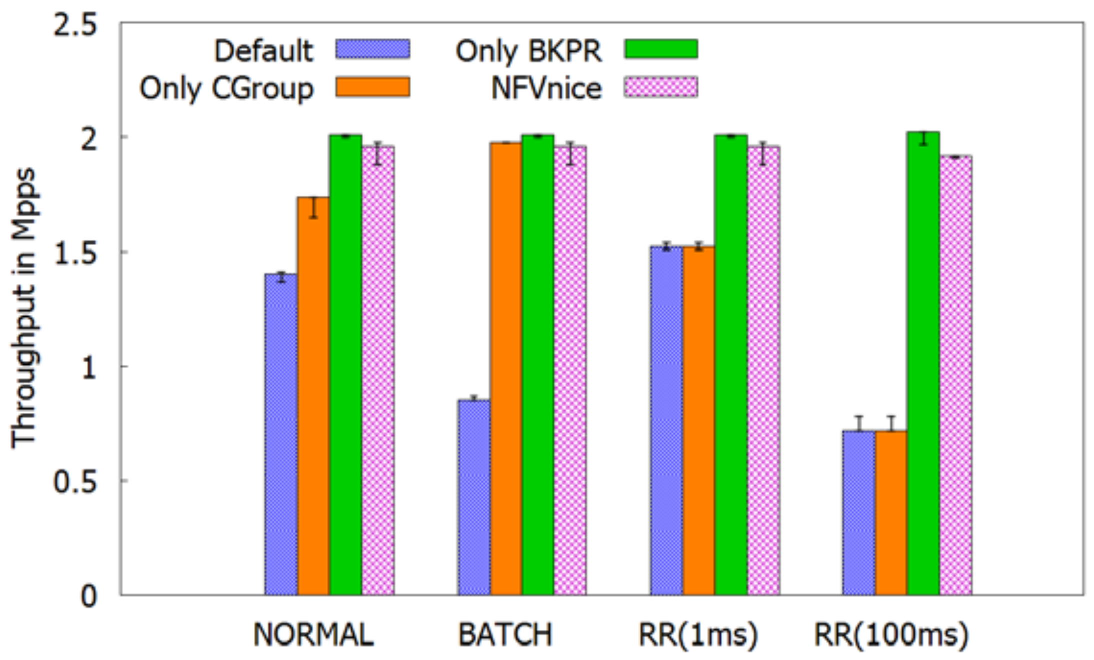


Consistently improves throughput regardless of chain characteristics

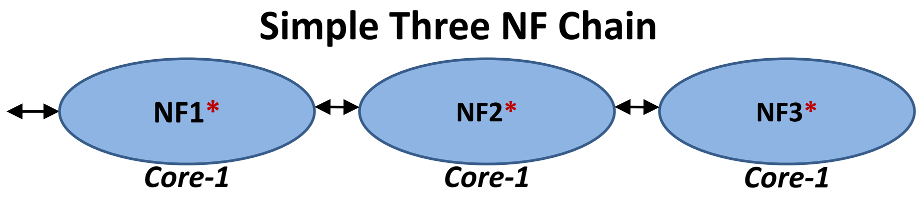
NF Processing cost variation



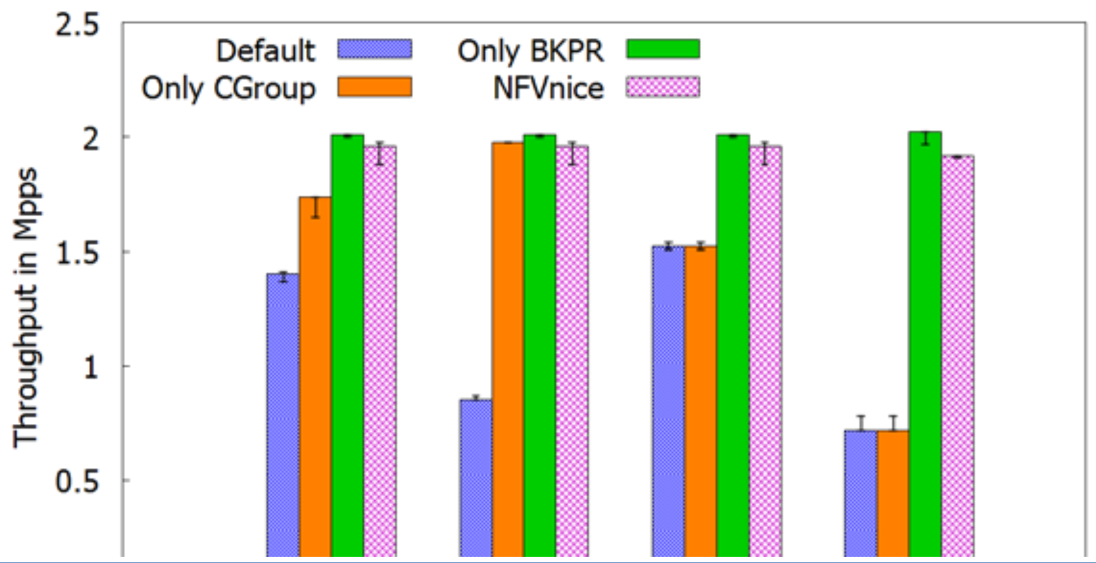
*Variable per packet processing cost [120 to 550 cpp]



NF Processing cost variation

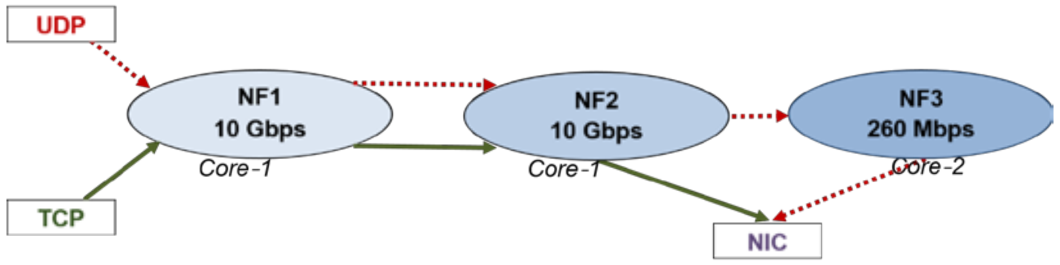


*Variable per packet processing cost [120 to 550 cpp]

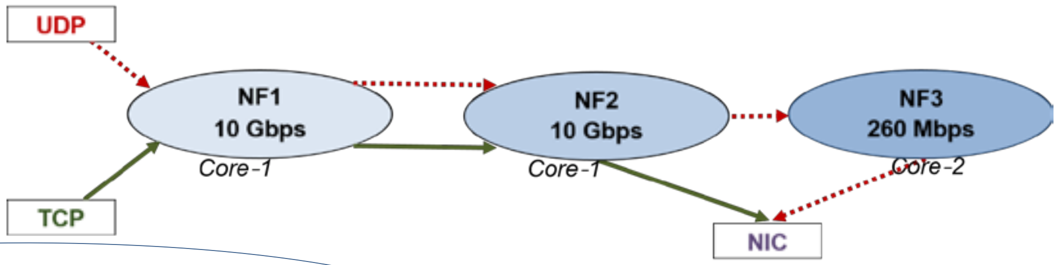


NFVnice is resilient to cost variations!

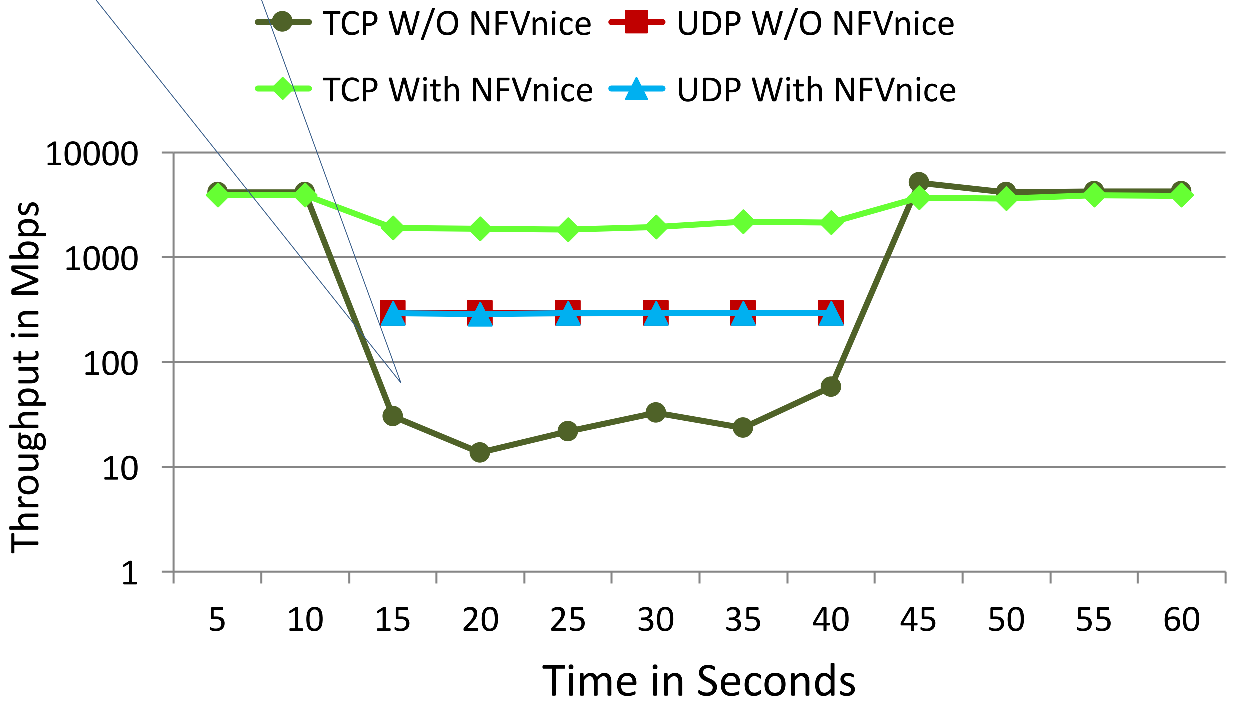
TCP and UDP Isolation



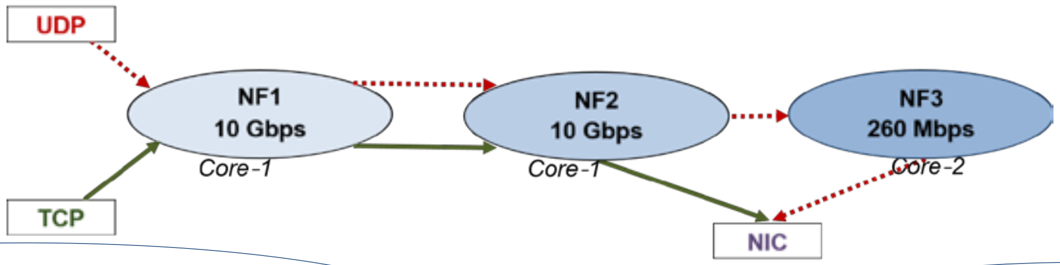
TCP and UDP Isolation



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

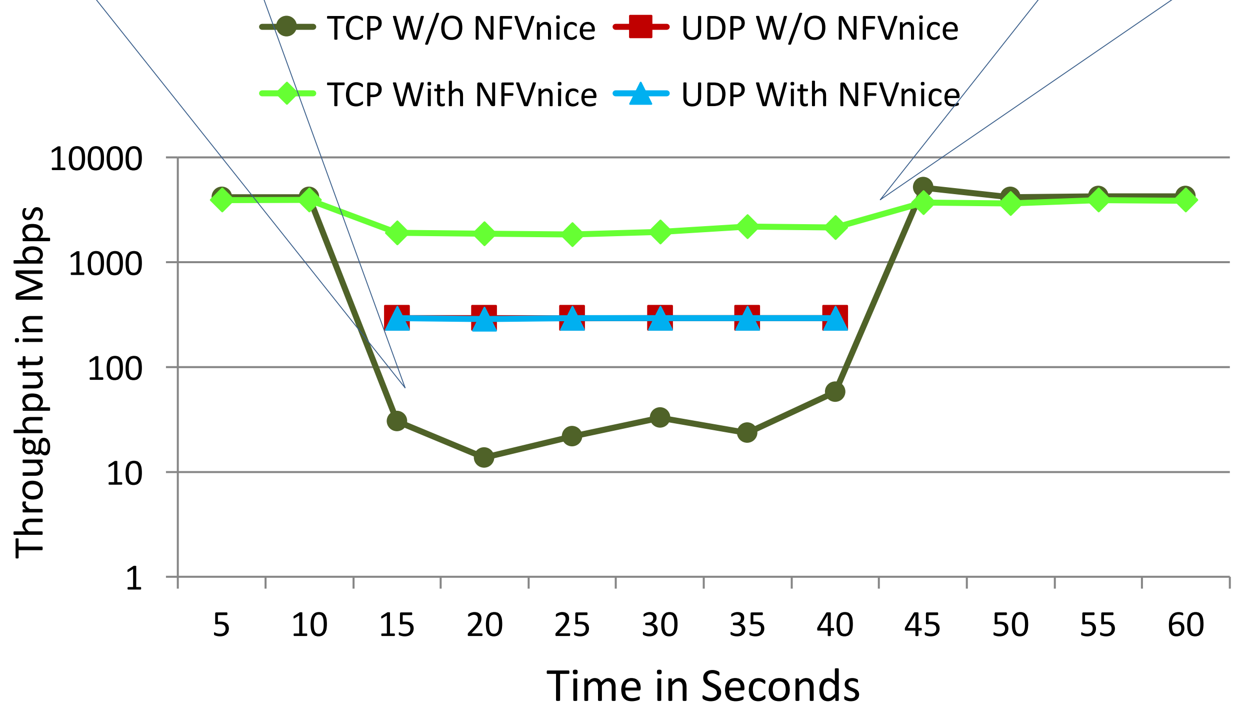


TCP and UDP Isolation



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

Effectively isolates UDP and TCP flows



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:
  - Get OpenNetVM: http://sdnfv.github.io/
  - 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