HAT: Heterogeneous Adaptive Throttling for On-Chip Networks

Kevin Kai-Wei Chang
Rachata Ausavarungnirun
Chris Fallin
Onur Mutlu

Carnegie Mellon University SAFARI

Executive Summary

Problem: Packets contend in on-chip networks (NoCs), causing congestion, thus reducing system performance

Approach: Source throttling (temporarily delaying packet injections) to reduce congestion

1) Which applications to throttle?

Observation: Throttling **network-intensive** applications leads to higher system performance

➔Key idea 1: Application-aware source throttling

2) How much to throttle?

Observation: There is no single **throttling rate** that works well for every application workload

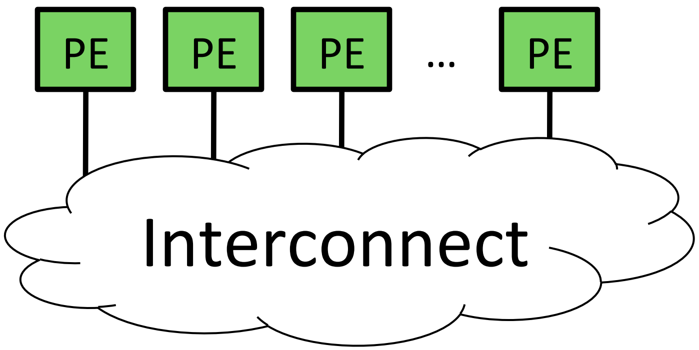
➔Key idea 2: Dynamic throttling rate adjustment

Result: Improves both system performance and energy efficiency over state-of-the-art source throttling policies

Outline

- **Background and Motivation**
- Mechanism
- Comparison Points
- Results
- Conclusions

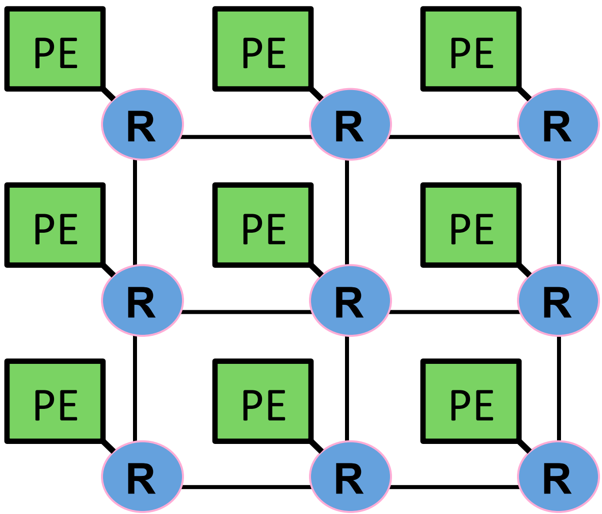
On-Chip Networks



- Connect **cores, caches, memory controllers, etc.**
  - Buses and crossbars are not scalable

Processing Element
(Cores, L2 Banks, Memory Controllers, etc)

On-Chip Networks

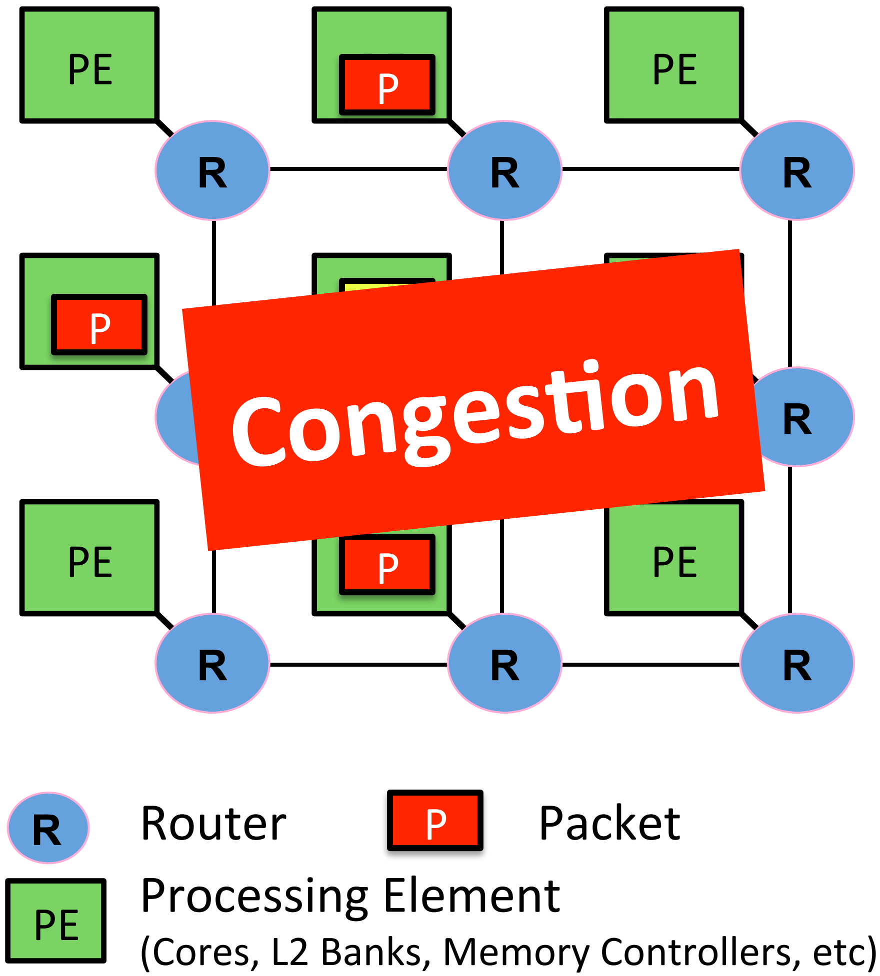


- Connect **cores, caches, memory controllers, etc**
  - Buses and crossbars are not scalable
- **Packet switched**
- **2D mesh:** Most commonly used topology
- Primarily serve **cache misses** and **memory requests**

Router

Processing Element
(Cores, L2 Banks, Memory Controllers, etc)

Network Congestion Reduces Performance



Limited shared resources (buffers and links)

- due to **design constraints**: Power, chip area, and timing

Network congestion:
↓system performance

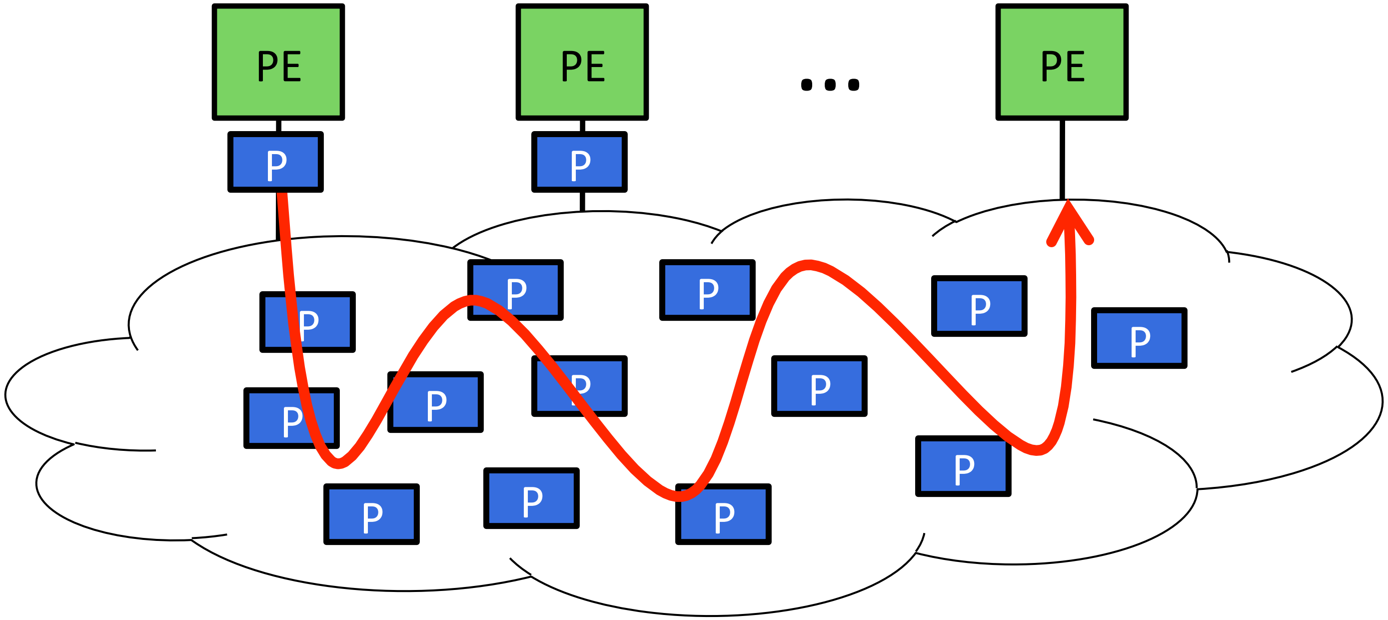
Goal

- **Improve system performance in a highly congested network**

- **Observation**: Reducing **network load** (number of packets in the network) decreases network congestion, hence improves system performance

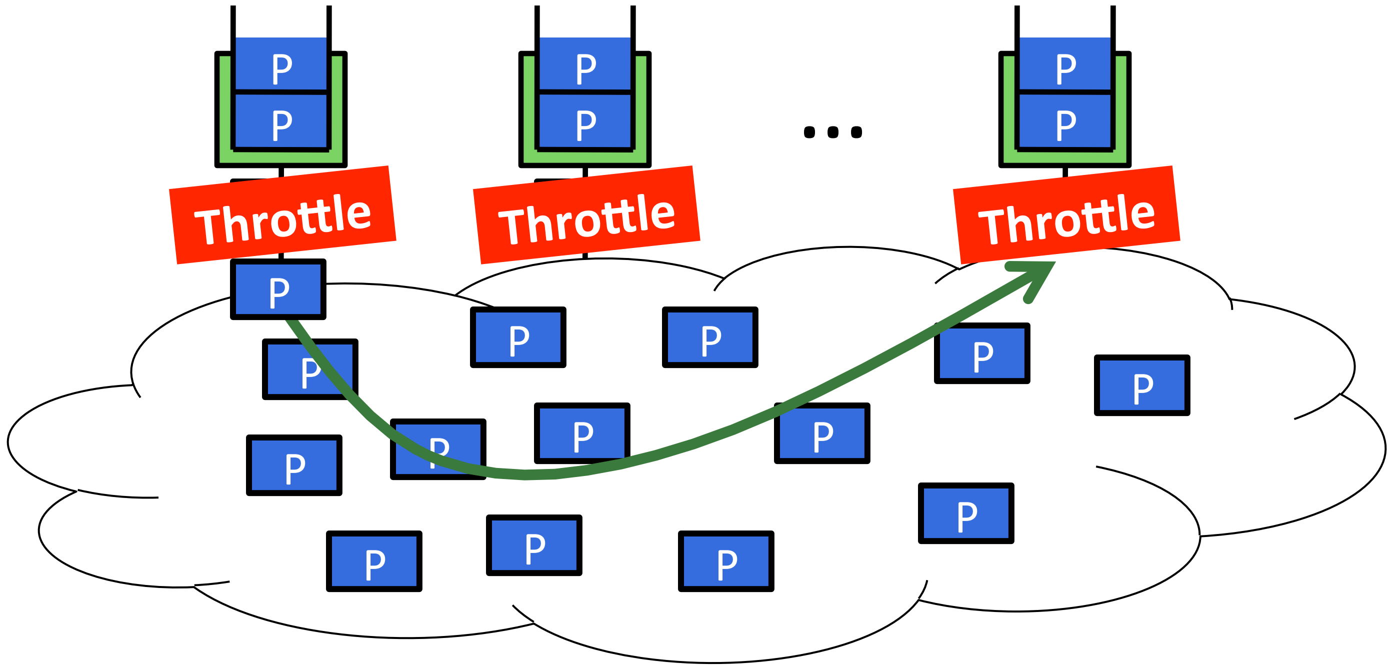
- **Approach:** Source throttling (temporarily delaying new traffic injection) to reduce network load

Source Throttling



Long network latency when the network is congested

Source Throttling



- Throttling makes some packets wait longer to inject
- Average network throughput increases, hence higher system performance

Network P Packet

PE Processing Element
(Cores, L2 Banks, Memory Controllers, etc)

Key Questions of Source Throttling

- Every cycle when a node has a packet to inject, source throttling blocks the packet with probability **P**
  - We call **P** "**throttling rate**" (ranges from 0 to 1)
- **Throttling rate** can be set independently on every node

<u>Two key questions</u>:

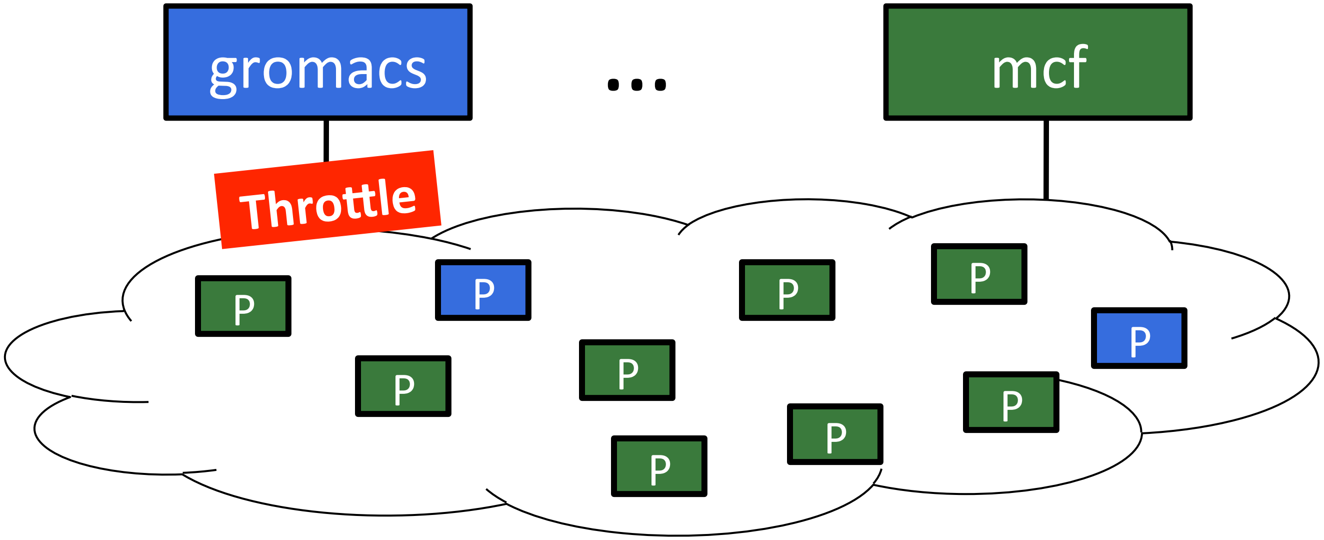
1. Which applications to throttle?
2. How much to throttle?

Naïve mechanism: Throttle every single node with a constant throttling rate

Key Observation #1

Throttling network-intensive applications leads to higher system performance

Configuration: 16-node system, 4x4 mesh network,
8 gromacs (network-non-intensive), and 8 mcf (network-intensive)

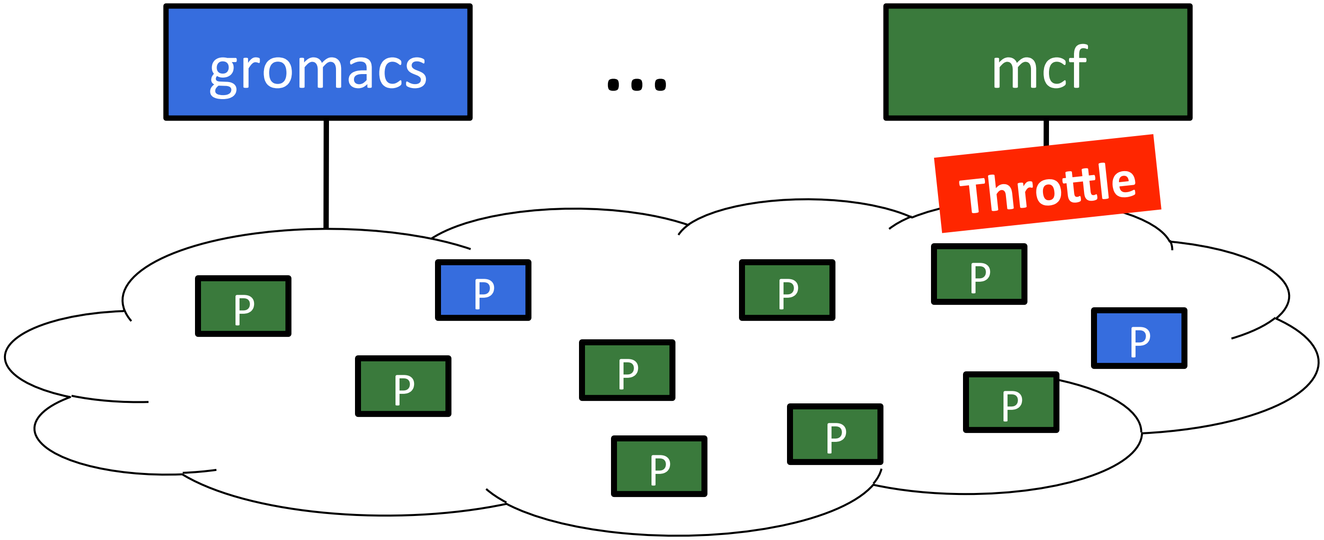


Throttling gromacs decreases system performance by **2%** due to minimal network load reduction

Key Observation #1

Throttling network-intensive applications leads to higher system performance

Configuration: 16-node system, 4x4 mesh network,
8 gromacs (network-non-intensive), and 8 mcf (network-intensive)

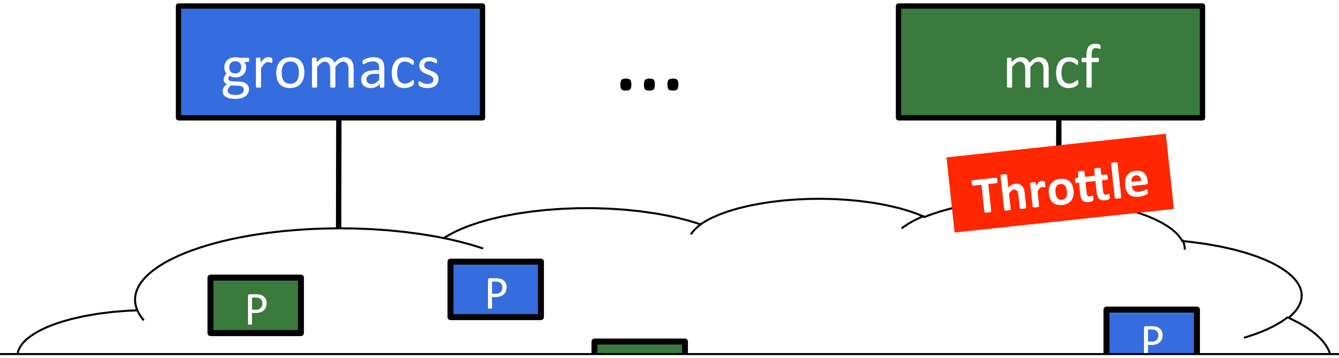


Throttling mcf increases system performance by **9%** (gromacs: **+14%** mcf: **+5%**) due to reduced congestion

Key Observation #1

Throttling network-intensive applications leads to higher system performance

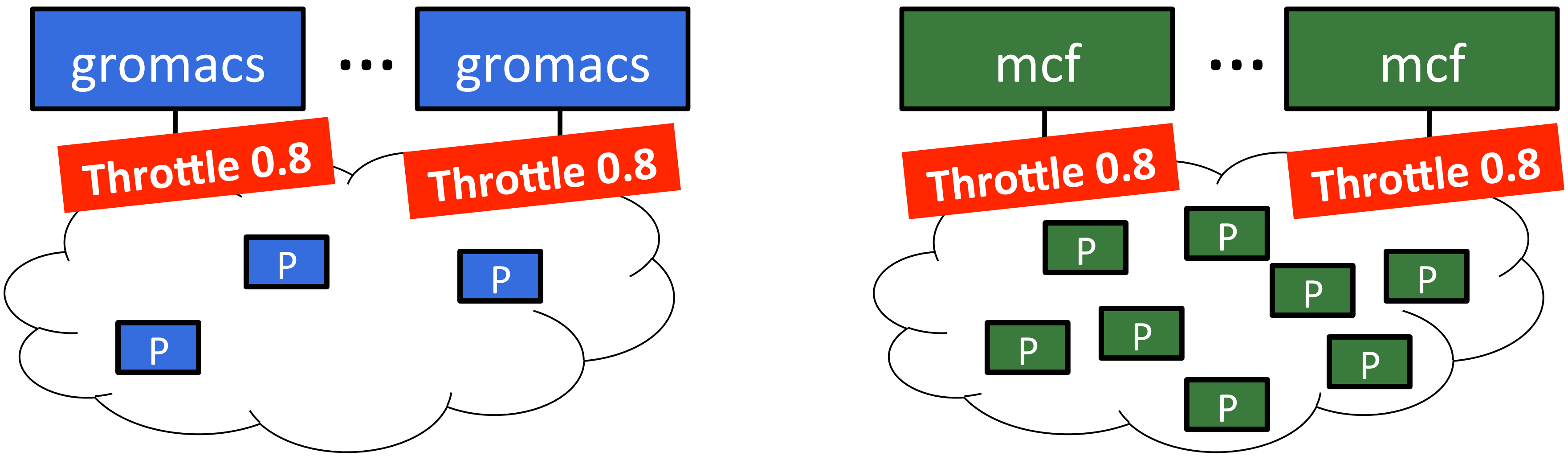
Configuration: 16-node system, 4x4 mesh network,
8 gromacs (network-non-intensive), and 8 mcf (network-intensive)



- Throttling network-intensive applications reduces congestion
- Benefits both network-non-intensive and network-intensive applications

Key Observation #2

There is no single **throttling rate** that works well for every application workload



Network runs best at or below a certain network load

Dynamically adjust throttling rate to avoid overload and under-utilization

Outline

- Background and Motivation
- **Mechanism**
- Comparison Points
- Results
- Conclusions

Heterogeneous Adaptive Throttling (HAT)

1. **Application-aware throttling**:
   Throttle **network-intensive** applications that interfere with **network-non-intensive** applications

2. **Network-load-aware throttling rate adjustment**:
   Dynamically adjust throttling rate to adapt to different workloads and program phases

Heterogeneous Adaptive Throttling (HAT)

1. **Application-aware throttling**:
   Throttle **network-intensive** applications that interfere with **network-non-intensive** applications

2. **Network-load-aware throttling rate adjustment**:
   Dynamically adjust throttling rate to adapt to different workloads and program phases

Application-Aware Throttling

1. **Measure applications' network intensity**

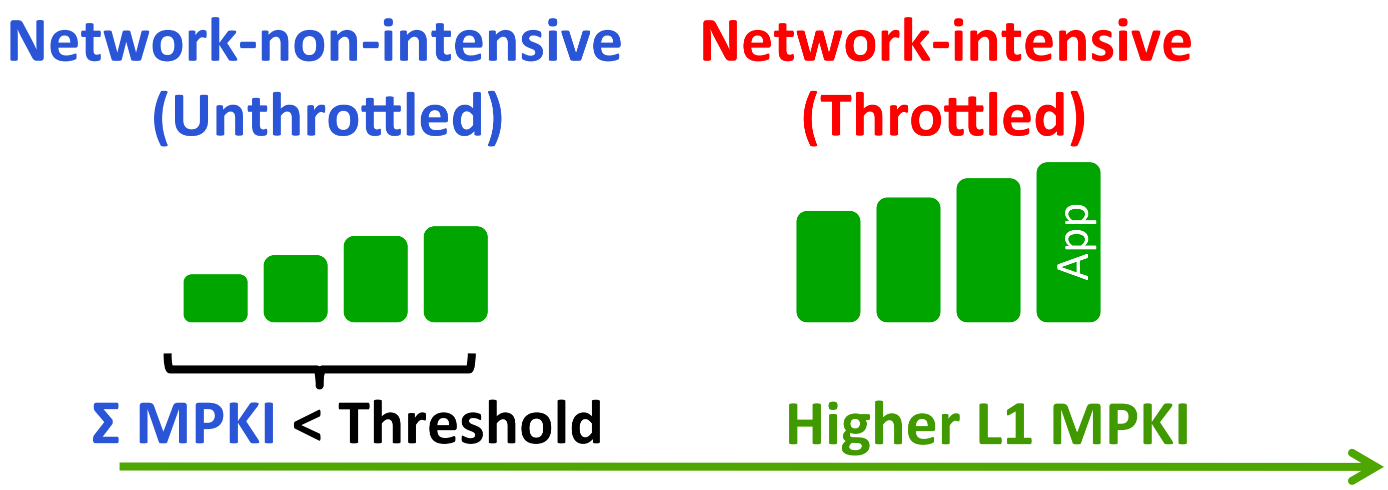
Use L1 MPKI (misses per thousand instructions) to estimate network intensity

2. **Throttle network-intensive applications**

How to select unthrottled applications?

- Leaving too many applications unthrottled overloads the network

→ Select unthrottled applications so that their total network intensity is less than the total network capacity



Heterogeneous Adaptive Throttling (HAT)

1. **Application-aware throttling**:
   Throttle **network-intensive** applications that interfere with **network-non-intensive** applications

2. **Network-load-aware throttling rate adjustment**:
   Dynamically adjust throttling rate to adapt to different workloads and program phases

Dynamic Throttling Rate Adjustment

- Different workloads require different throttling rates to avoid overloading the network
- But, **network load** (fraction of occupied buffers/ links) is an accurate indicator of congestion
- **Key idea**: Measure current network load and **dynamically adjust** throttling rate based on load

```
if network load > target:
    Increase throttling rate
else:
    Decrease throttling rate
```

If network is congested, throttle more

If network is not congested, avoid unnecessary throttling

Heterogeneous Adaptive Throttling (HAT)

1. **Application-aware throttling**:
   Throttle **network-intensive** applications that interfere with **network-non-intensive** applications

2. **Network-load-aware throttling rate adjustment**:
   Dynamically adjust throttling rate to adapt to different workloads and program phases

Epoch-Based Operation

- **Application classification** and **throttling rate adjustment** are expensive if done every cycle
- **Solution:** recompute at epoch granularity

<u>During epoch</u>:

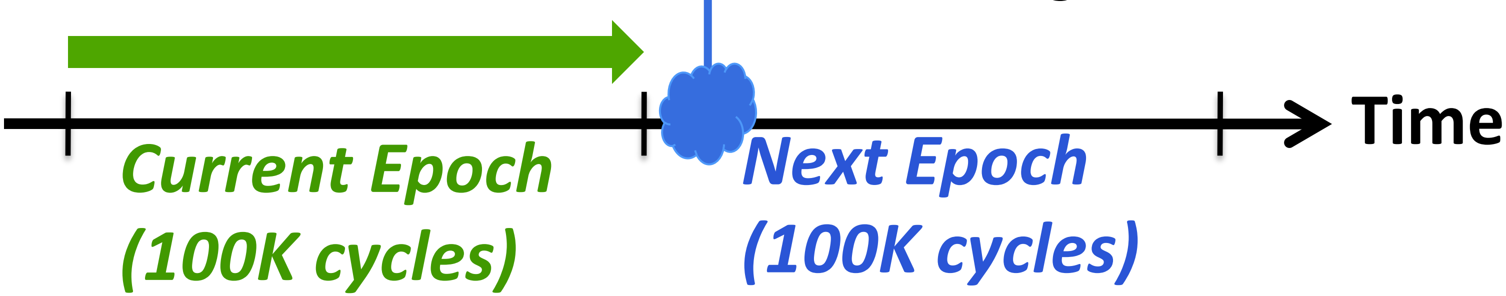
Every node:

1) Measure **L1 MPKI**
2) Measure **network load**

<u>Beginning of epoch</u>:

All nodes send measured info to a **central controller**, which:

1) Classifies applications
2) Adjusts throttling rate
3) Sends new classification and throttling rate to each node



Putting It Together: Key Contributions

1. Application-aware throttling

– Throttle network-intensive applications based on applications' network intensities

2. Network-load-aware throttling rate adjustment

– Dynamically adjust throttling rate based on network load to avoid overloading the network

HAT is the first work to combine application-aware throttling and network-load-aware rate adjustment

Outline

- Background and Motivation
- Mechanism
- **Comparison Points**
- Results
- Conclusions

Comparison Points

- **Source throttling for bufferless NoCs**
  [Nychis+ Hotnets'10, SIGCOMM'12]
  - Throttle network-intensive applications when other applications cannot inject
  - Does not take network load into account
  - We call this "Heterogeneous Throttling"

- **Source throttling for buffered networks**
  [Thottethodi+ HPCA'01]
  - Throttle every application when the network load exceeds a dynamically tuned threshold
  - Not application-aware
  - Fully blocks packet injections while throttling
  - We call this "Self-Tuned Throttling"

Outline

- Background and Motivation
- Mechanism
- Comparison Points
- **Results**
- Conclusions

Methodology

- **Chip Multiprocessor Simulator**
  - **64-node** multi-core systems with a **2D-mesh topology**
  - Closed-loop core/cache/NoC cycle-level model
  - 64KB L1, perfect L2 (always hits to stress NoC)

- **Router Designs**
  - **Virtual-channel buffered** router**:** 4 VCs, 4 flits/VC [Dally+ IEEE TPDS'92]
    - **Input buffers** to hold contending packets
  - **Bufferless deflection** router: **BLESS** [Moscibroda+ ISCA'09]
    - **Misroute (deflect)** contending packets

Methodology

- **Workloads**
  - 60 multi-core workloads of SPEC CPU2006 benchmarks
  - 4 network-intensive workload categories based on the network intensity of applications (**L**ow/**M**edium/**H**igh)

- **Metrics**

  System performance: $Weighted\ Speedup = \sum_i \frac{IPC_i^{shared}}{IPC_i^{alone}}$

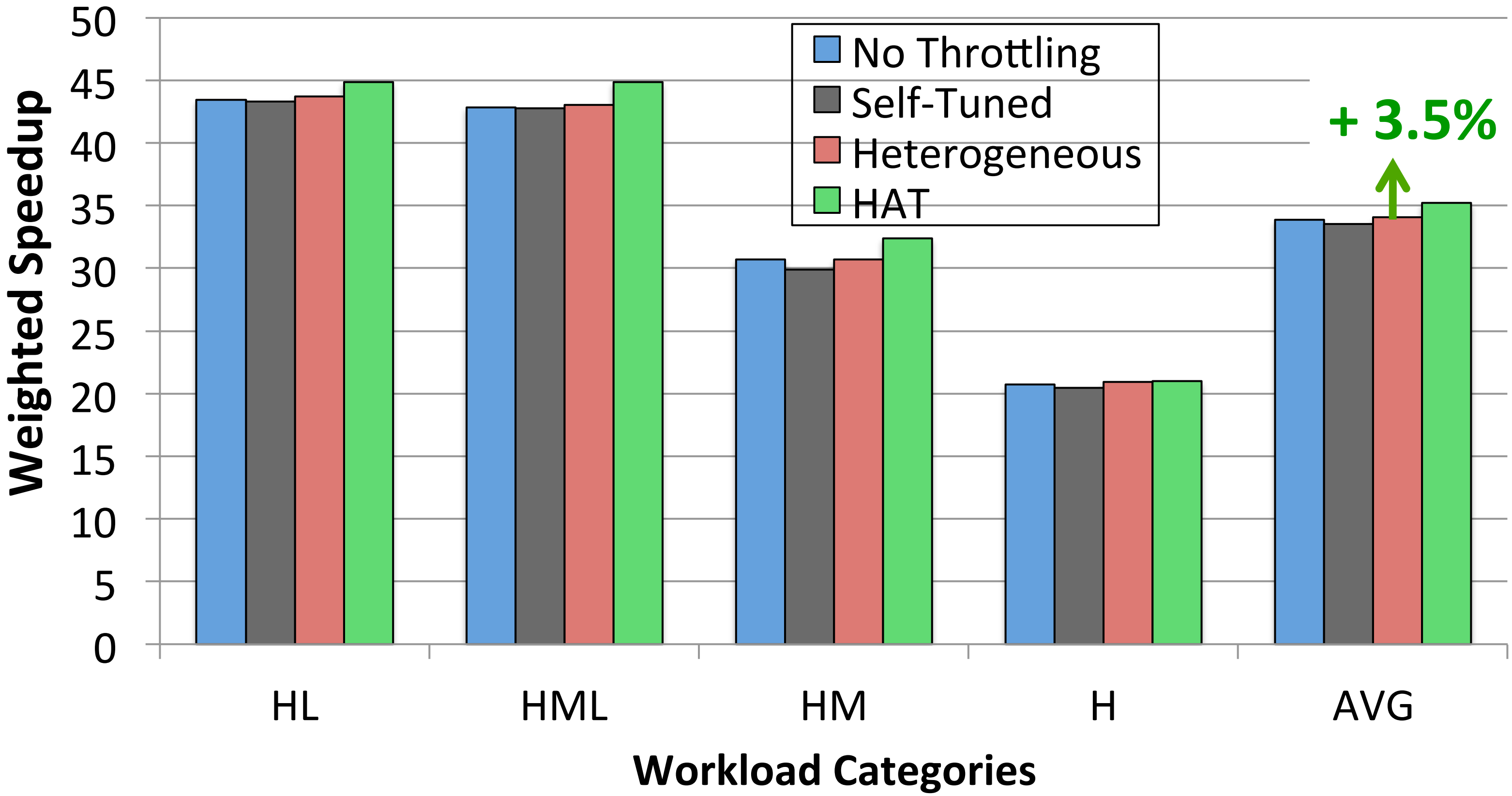
  Fairness: $Maximum\ Slowdown = \max_i \frac{IPC_i^{alone}}{IPC_i^{shared}}$

  Energy efficiency: $PerfPerWatt = \frac{WeightedSpeedup}{Power}$

Performance: Bufferless NoC (BLESS)

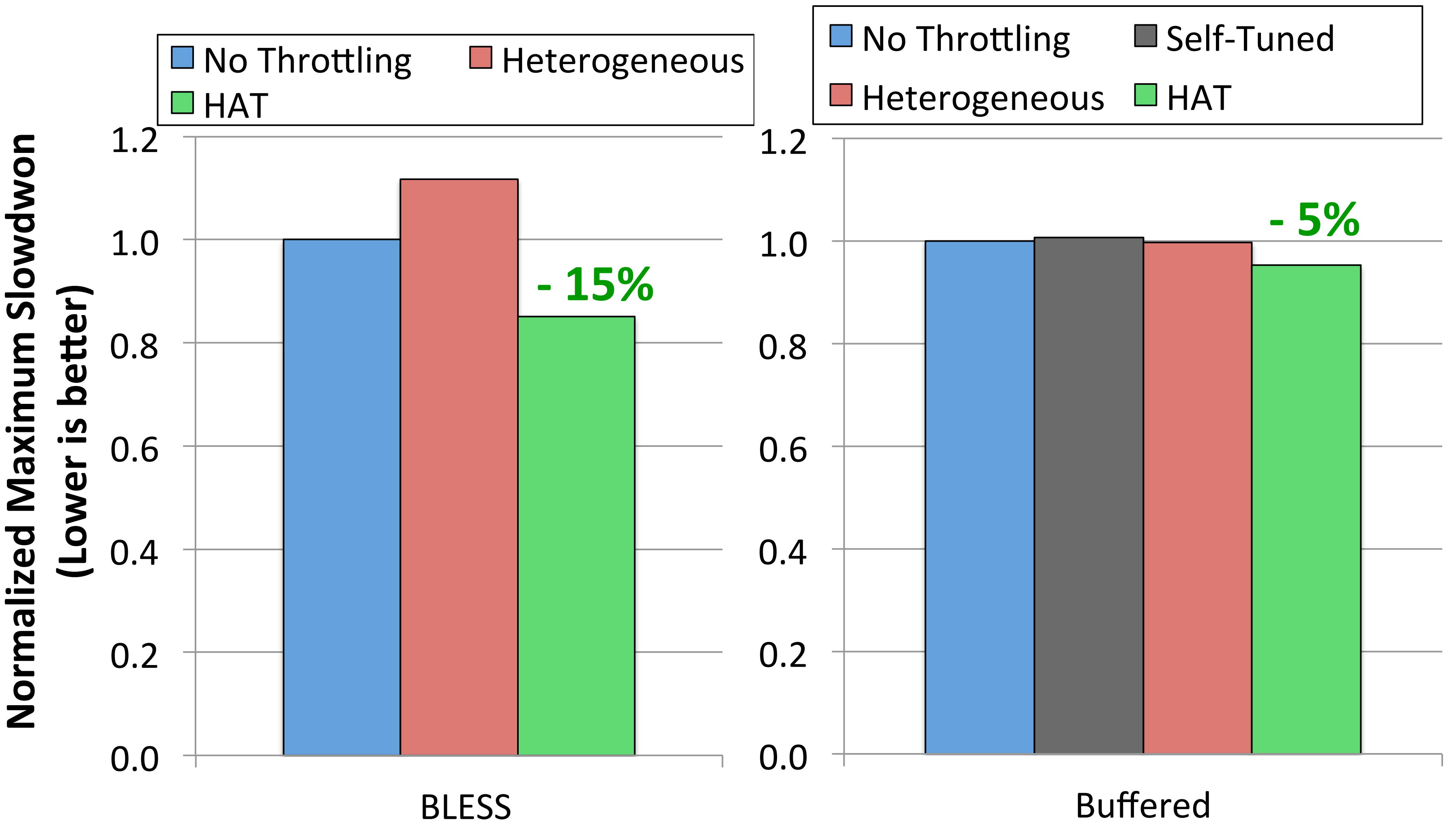
1. **HAT** provides better performance improvement than state-of-the-art throttling approaches
2. Highest improvement on **heterogeneous** workloads
   - **L** and **M** are more **sensitive** to network latency

Performance: Buffered NoC



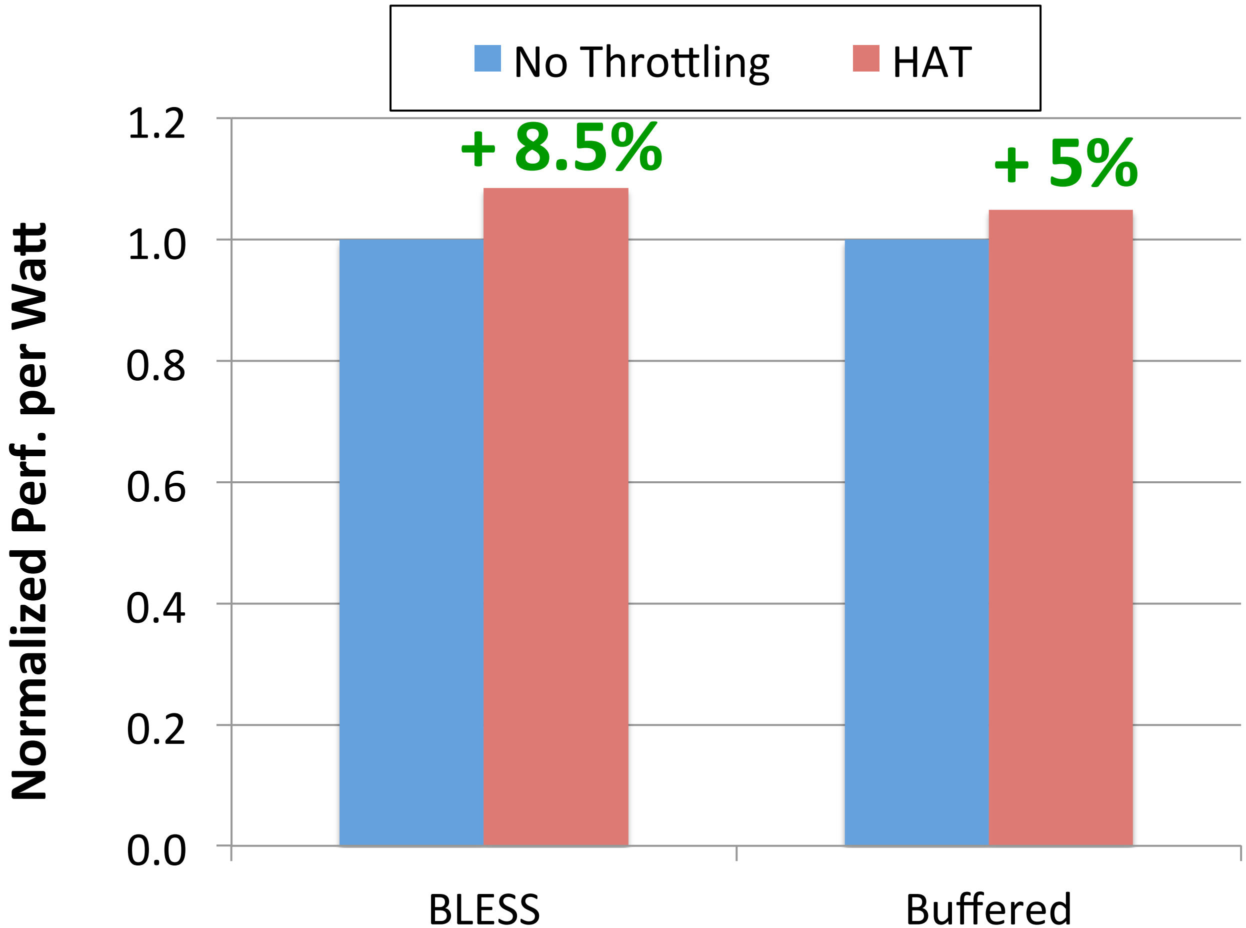
HAT provides better performance improvement than prior approaches

Application Fairness



HAT provides better fairness than prior works

Network Energy Efficiency



HAT increases energy efficiency by reducing network load by 4.7% (BLESS) or 0.5% (buffered)

Other Results in Paper

- Performance on **CHIPPER** [Fallin+ HPCA'11]
  - **HAT** improves system performance
- Performance on **multithreaded** workloads
  - **HAT** is not designed for multithreaded workloads, but it slightly improves system performance
- Parameter sensitivity sweep of **HAT**
  - **HAT** provides consistent system performance improvement on different network sizes

Conclusions

Problem: Packets contend in on-chip networks (NoCs), causing congestion, thus reducing system performance

Approach: Source throttling (temporarily delaying packet injections) to reduce congestion

1) Which applications to throttle?

Observation: Throttling **network-intensive** applications leads to higher system performance

➔Key idea 1: Application-aware source throttling

2) How much to throttle?

Observation: There is no single **throttling rate** that works well for every application workload

➔Key idea 2: Dynamic throttling rate adjustment

Result: Improves both system performance and energy efficiency over state-of-the-art source throttling policies

HAT: Heterogeneous Adaptive Throttling for On-Chip Networks

Kevin Kai-Wei Chang
Rachata Ausavarungnirun
Chris Fallin
Onur Mutlu

Carnegie Mellon University SAFARI

Throttling Rate Steps

Algorithm 1 HAT: Application Classification Algorithm

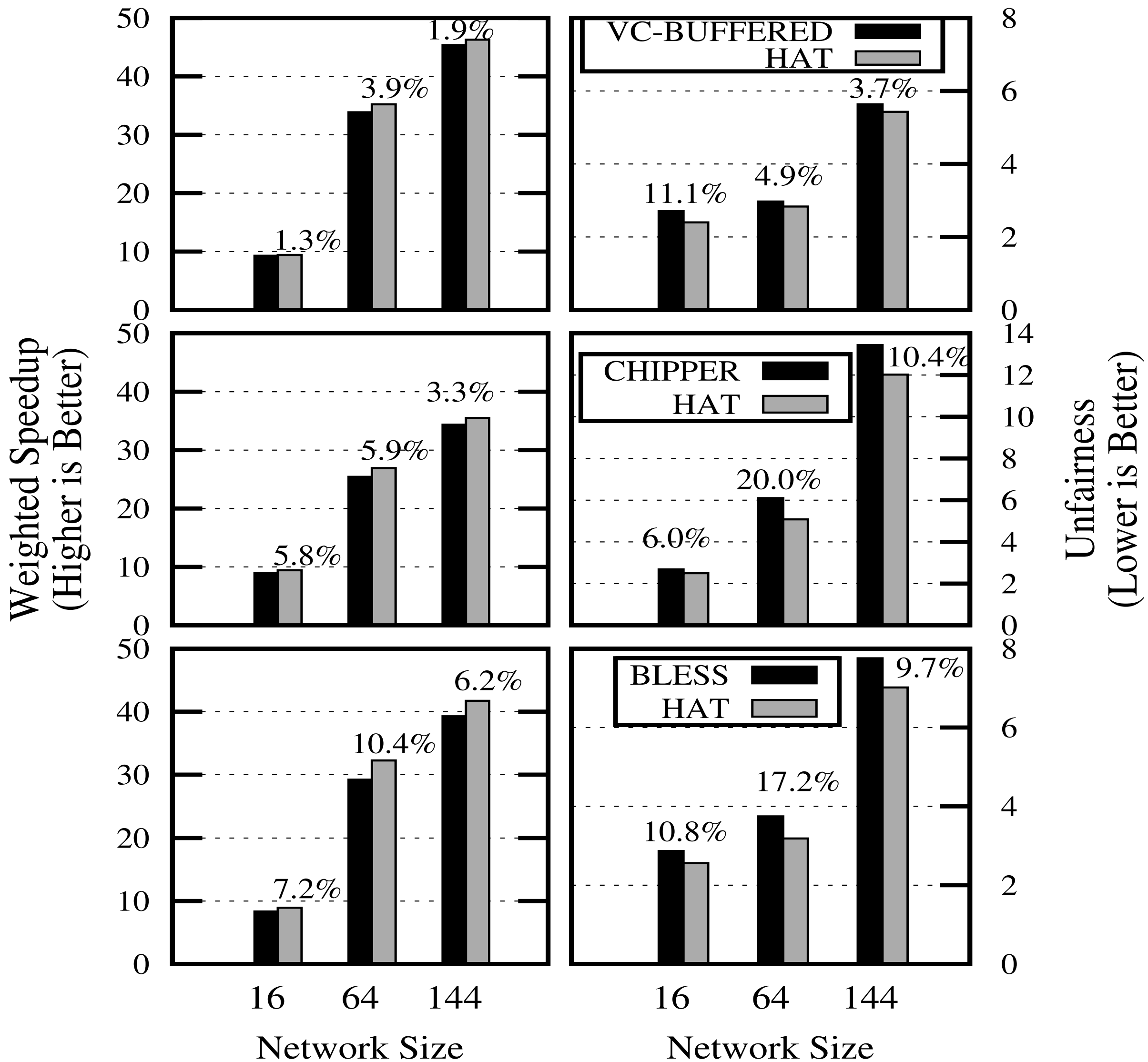
---

at the beginning of each epoch:
empty the groups
sort N applications by MPKI measurements $MPKI_i$
for sorted application i in N **do**
  if total MPKI of network-non-intensive group $+MPKI_i \leq NonIntensiveCap$ **then**
    Place application i into the network-non-intensive group
  else
    Place application i into the network-intensive group
  end if
end for

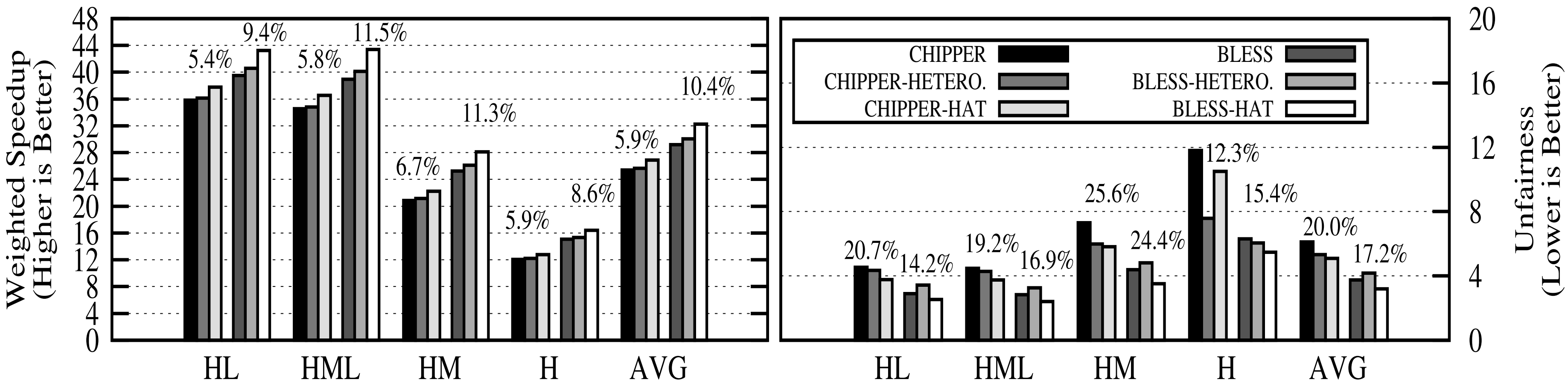
Current Throttling Rate	Throttling Rate Step
0% – 70%	10%
70% – 90%	2%
90% – 94%	1%

Table II. Throttling rate adjustment used in each epoch.

Network Sizes



Performance on CHIPPER/BLESS



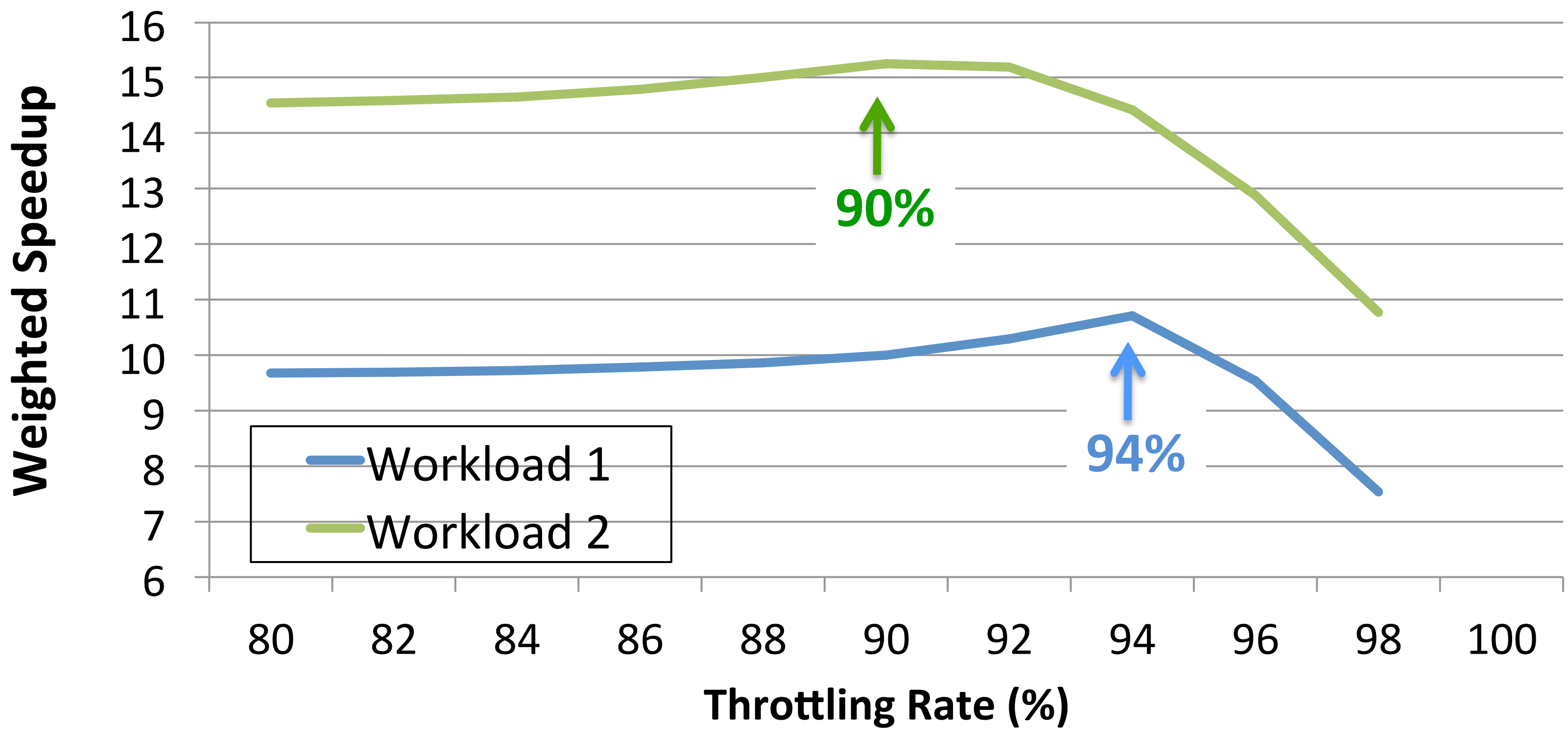
Injection rate (number of injected packets / cycle):
+8.5% (**BLESS**) or +4.8% (**CHIPPER**)

Multithreaded Workloads

Benchmark	fft	luc	lun	cholesky
VC-Buffered	0.1%	0.0%	7.5%	-0.1%
BLESS	0.1%	0.0%	4.2%	0.0%
CHIPPER	0.1%	-0.1%	1.0%	-0.1%

Table V. Execution time reduction of HAT on multithreaded workloads.

Motivation



16
15
14
13
12
11
10
9
8
7
6
Weighted Speedup
90%
94%
Workload 1
Workload 2
80
82
84
86
88
90
92
94
96
98
100
Throttling Rate (%)

Sensitivity to Other Parameters

- **<u>Network load target</u>**: WS peaks at b/w 50% and 65% network utilization
  - Drops more than 10% beyond that
- **<u>Epoch length</u>**: WS varies by less than 1% b/w 20K and 1M cycles
- **<u>Low-intensity workloads</u>**: HAT does not impact system performance
- **<u>Unthrottled network intensity threshold</u>**:

	0	50	100	150	200	250
Δ WS	8.5%	10.1%	10.6%	10.8%	10.3%	9.6%
Δ Unfairness	-11%	-9.5%	-7.1%	-5.0%	-2.5%	-2.1%

Table VI. Sensitivity of HAT improvements to $NonIntensiveCap$.

Implementation

- **L1 MPKI**: One L1 miss hardware counter and one instruction hardware counter
- **Network load**: One hardware counter to monitor the number of flits routed to neighboring nodes
- **Computation**: Done at one central CPU node
  - At most several thousand cycles (a few percent overhead for 100K-cycle epoch)