Building a Power-Proportional Software Router

Luca Niccolini, Gianluca Iannaccone, Sylvia Ratnasamy, Jaideep Chandrashekar, Luigi Rizzo

Motivation

Networking devices

- Provisioned for peak load
- Underutilized on average
  - ~5% in enterprise networks
  - 30-40% for ISPs
  - 5X variability in ADSL networks
- Highly inefficient at low load
  - 80-90% with no traffic

Large deployments of network appliances (x86 based)

- WAN optimizer, Firewall …
- Approximately 2 appliances for 3 routers in enterprises [Sekar – HotNets’11]

Challenge

How to build an **energy-efficient** ***software*** router?

Can adapt dynamically to the incoming rate

Consumes power in proportion to the incoming rate

Still achieves peak packet forwarding performance

Our solution:

Reduce energy by up to 50%,
Latency increase of 10us

HW/SW Platform

General Purpose x86 servers

Linux + Click modular router (kernel mode)

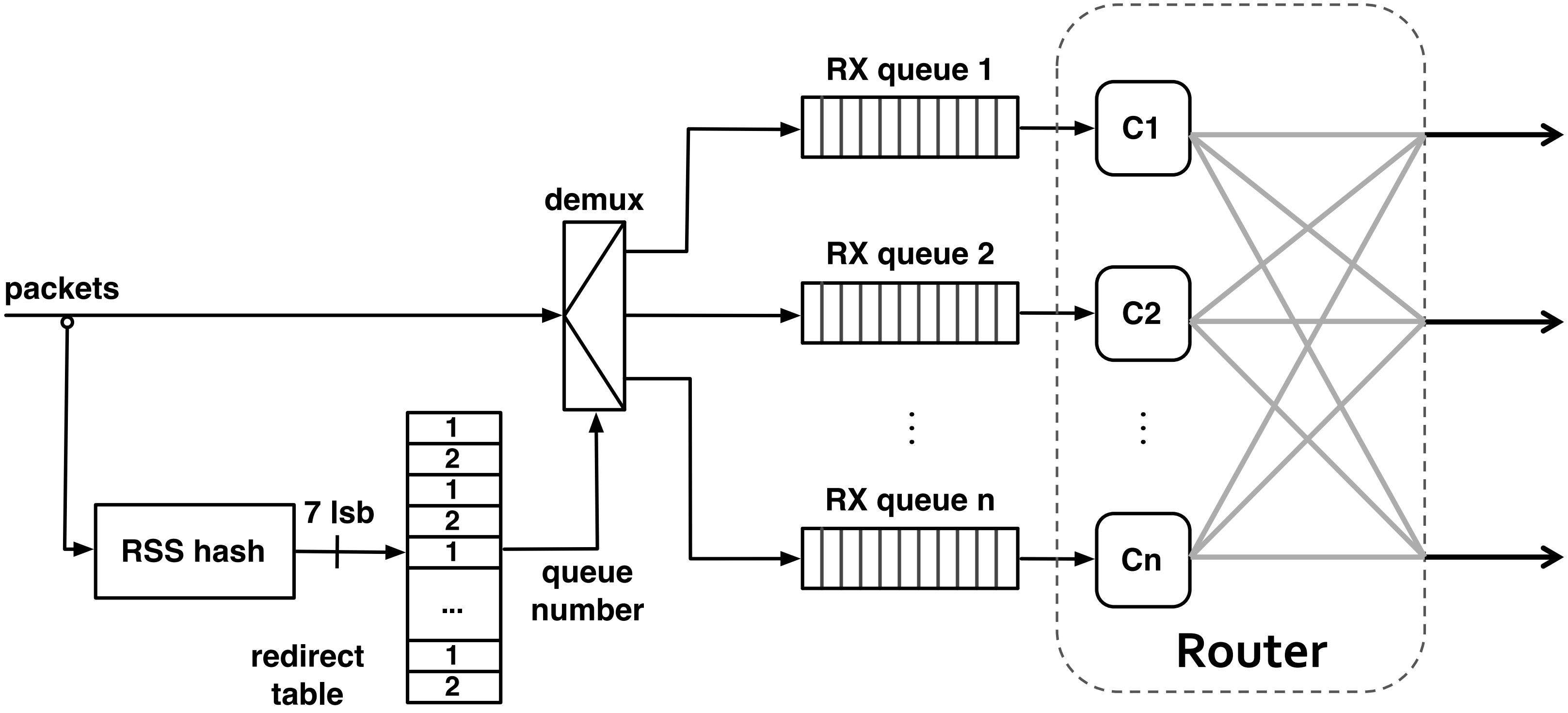
10Gbps network

- ≫ Fast enough
    - Routebricks, PacketShader, Netmap

Open Platform

- ≫ Can use OS primitives for low-power

Multiqueue operation



Traffic is split among multiple HW queues

Receive Side Scaling

- Each queue is managed by one core (no contention)
- How many queues/cores to use?

Primitives for low power

Sleep States / C-States

- *Co* – Active, executing instructions
- *C1* – Active, not executing instructions (clock-gated)

  …
- *Cn* – Deepest Sleep State (power-gated)

≫ *Idle Power* vs. *Exit Latency* tradeoff

DVFS / P-States

- *P0* – Max Operating Frequency
- *P1*, *P2*, *P3* …
- *Pn* – Min Operating Frequency

Outline

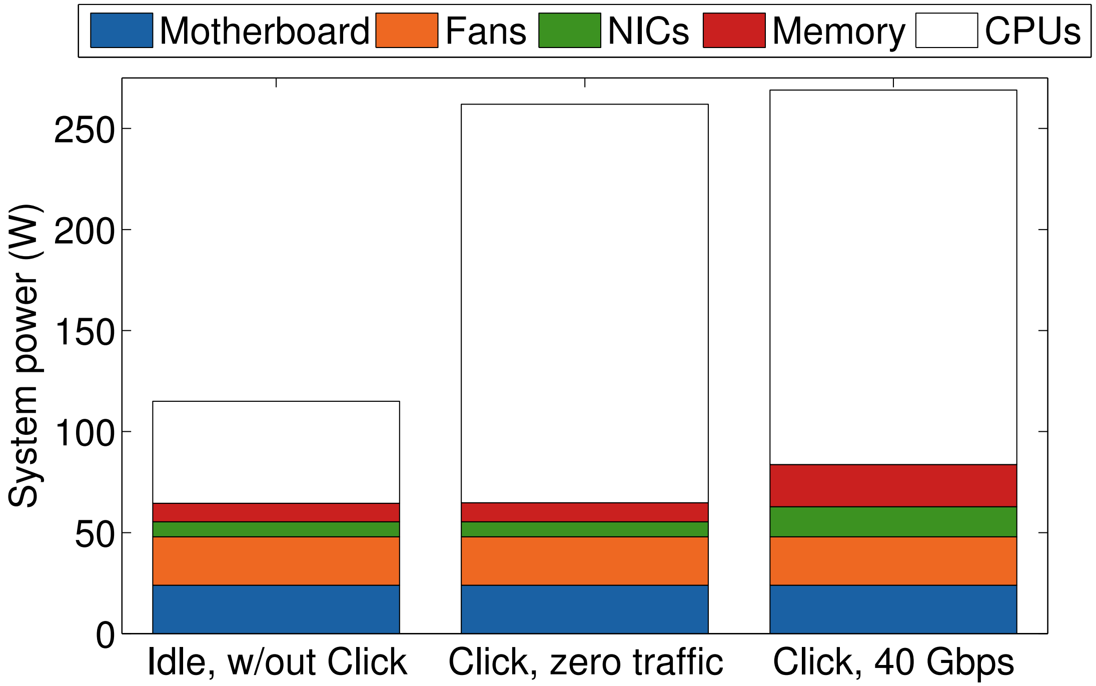
Power consumption breakdown

Power-saving algorithms guidelines

Online algorithm implementation

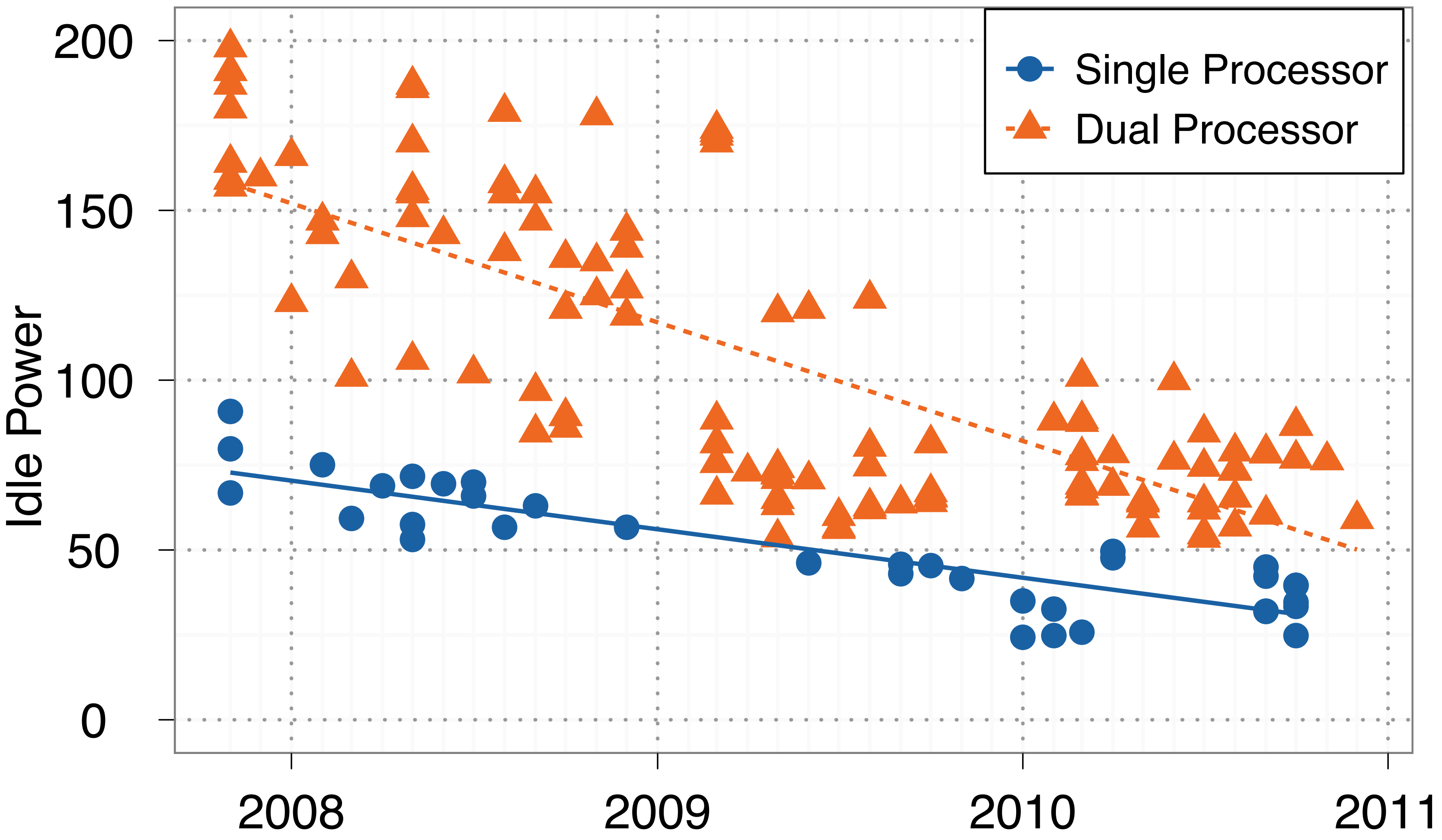
Performance Evaluation

Power Consumption Breakdown



- High IDLE power
- Memory, NICs contribute little
- CPUs are the most power-hungry components, with a high dynamic range

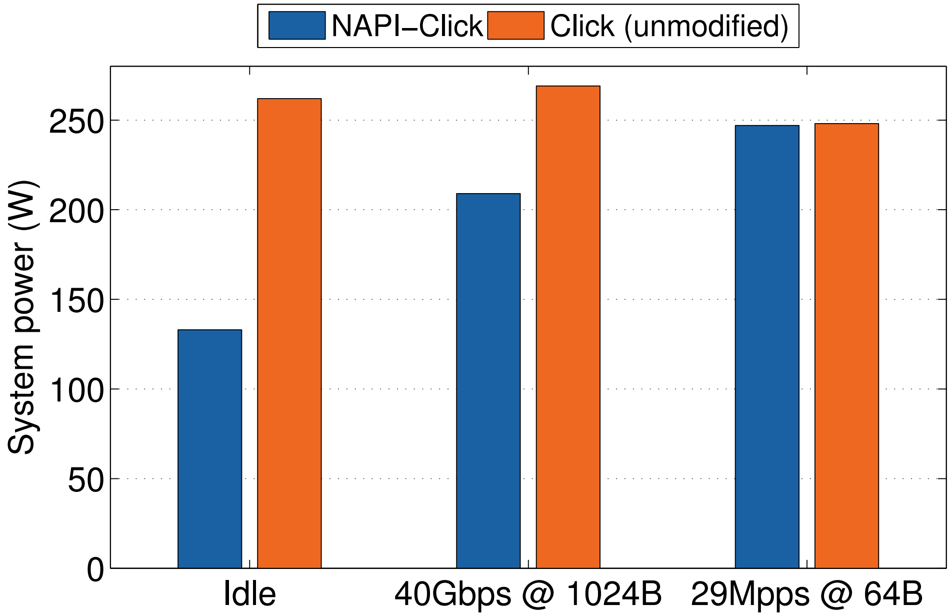
System Idle Power Trend

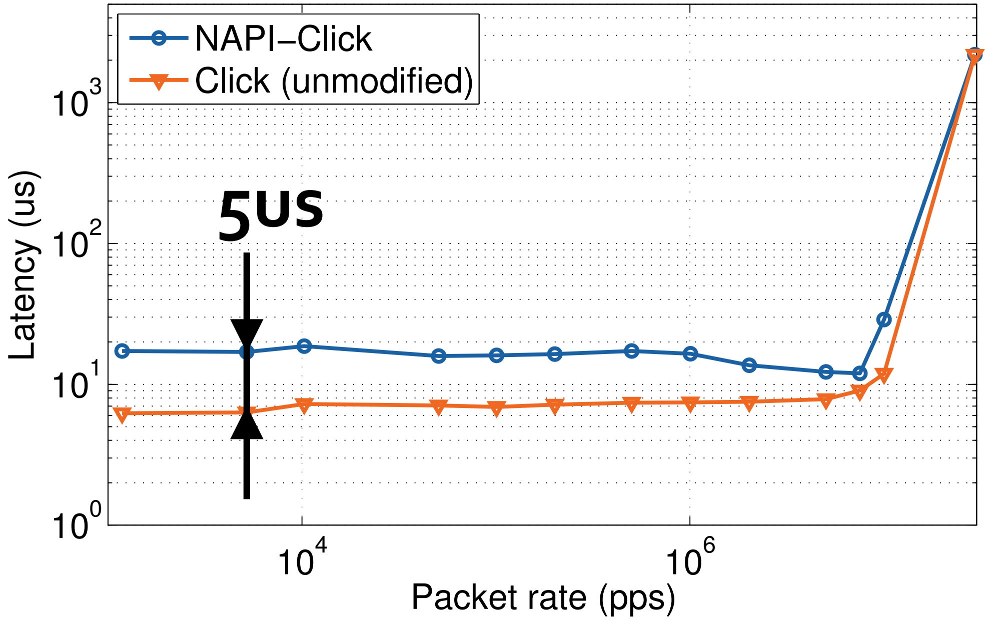


SPECPower data

Addressing SW inefficiency with NAPI

Enables power savings





introduced a modest increase in latency

Outline

Power consumption breakdown

Power-saving algorithms guidelines

Online algorithm implementation

Performance Evaluation

Power Saving algorithms Design space

1. How many cores to allocate?

2. At what frequency should they run?

3. Which sleep states to use?
   - Active – underutilized cores
   - Inactive cores

Power Saving algorithms
Design space

Single Core

- *Race-to-idle*
  - Process packets as fast as possible
  - Maximize sleep time

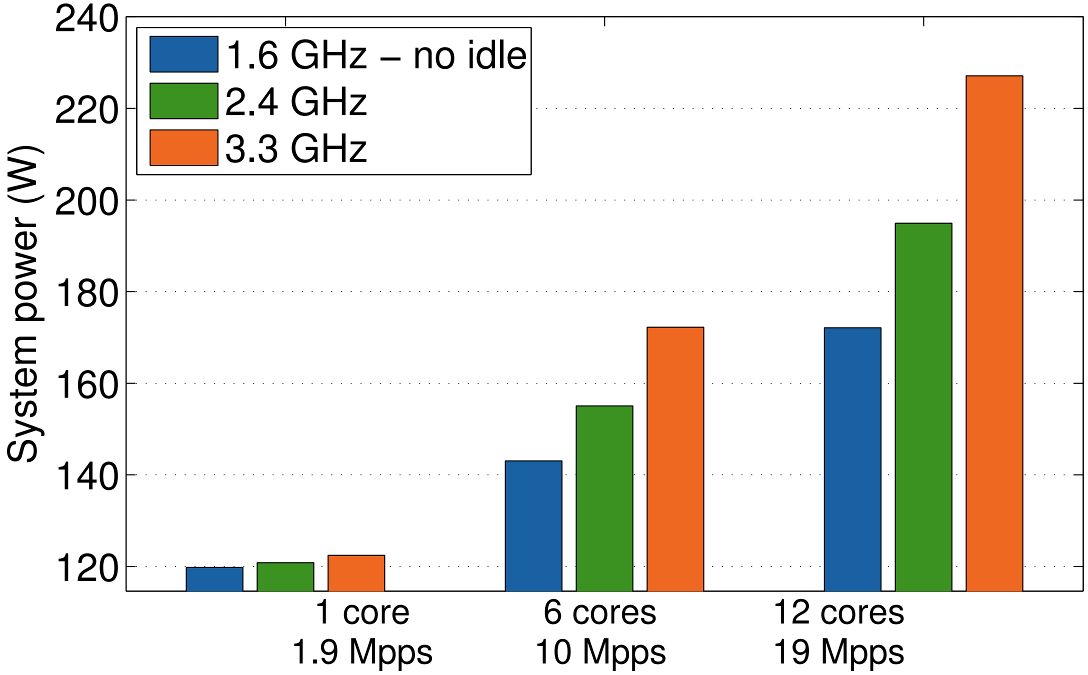
- *Just-in-time*
  - Process packets as slow as we can
  - Never sleep

Multi Core

- #cores vs. operating frequency tradeoff

Single Core Case

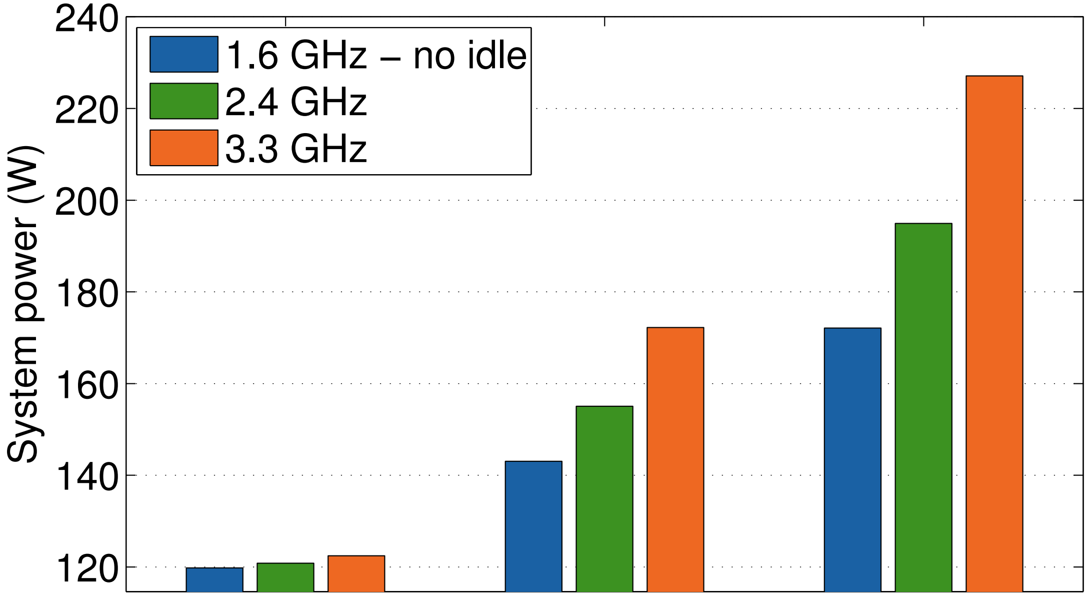
- *Just-in-time vs Race-to-idle*
  - I/O bound workload
  - Doubling the frequency does not halves the time
  - Race-to-idle drawbacks (idle power, exit-latency)



Single Core Case

≫ *Just-in-time vs Race-to-idle*

- I/O bound workload
- Doubling the frequency does not halves the time
- Race-to-idle drawbacks (idle power, exit-latency)

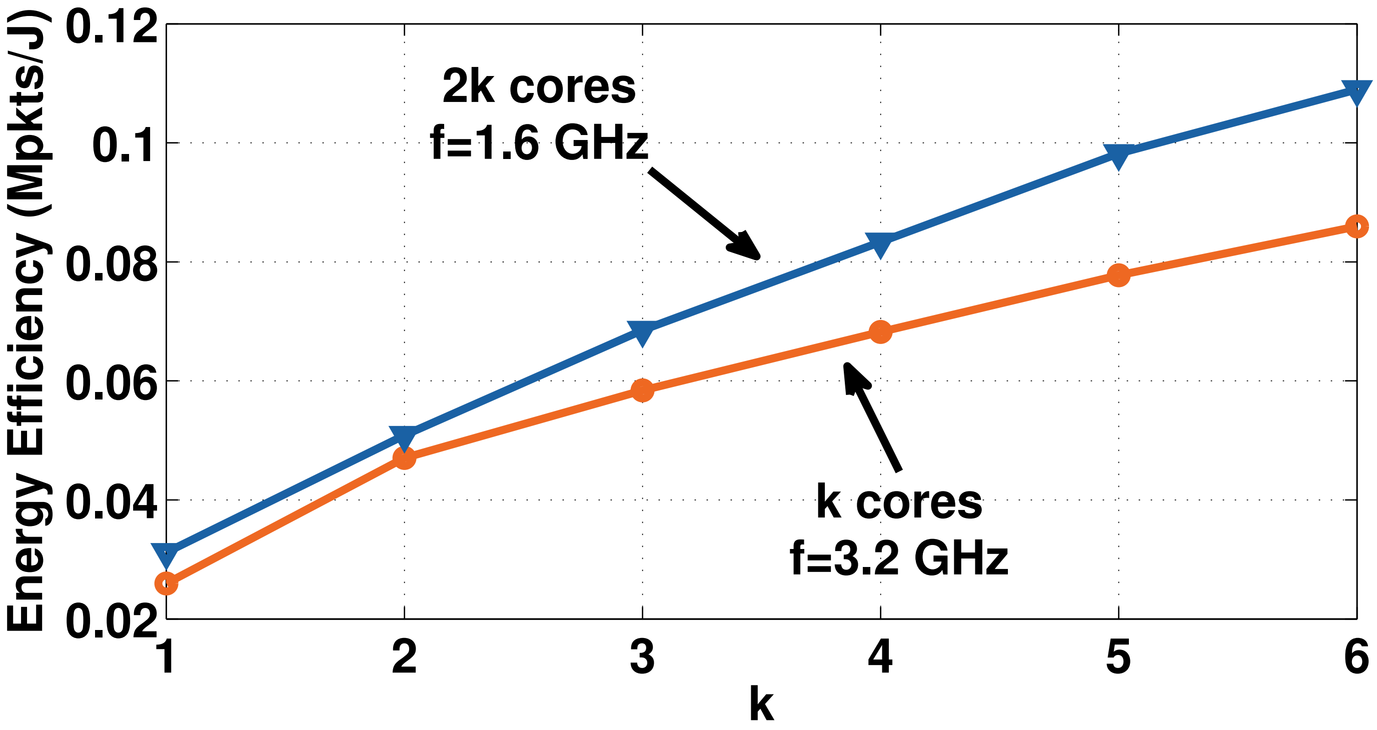


Run at the minimum frequency that keep up with the incoming rate

Multicore case - # cores

Use k cores at frequency f

Use n*k cores at frequency f/n



Higher
is better

Multicore case - # cores

Why not run all the cores all the time?

- ≫ Limited number of frequency levels available
- ≫ The lower frequency level is typically high
  - Half the maximum in our case (1.6GHz – 3.3Ghz)

Multicore case - # cores

Why not run all the cores all the time?

- Limited number of frequency levels available

- The lower frequency level is typically high
  - Half the maximum in our case (1.6GHz – 3.3Ghz)

Run the maximum number of cores that can be kept fully utilized

Multicore case – Sleep states

How to operate inactive and underutilized cores?

C-State	System Power (12 cores)	Exit Latency
C1	133 W	< 1 us
C3	120 W	~ 60 us
C6	115 W	~ 87 us

→ **Best for inactive Cores**

Multicore case – Sleep states

How to operate inactive and underutilized cores?

C-State	System Power (12 cores)	Exit Latency
C1	133 W	< 1 us
C3	120 W	~ 60 us
C6	115 W	~ 87 us

Best for inactive Cores

Let underutilized cores take quick and light naps (C1)

Outline

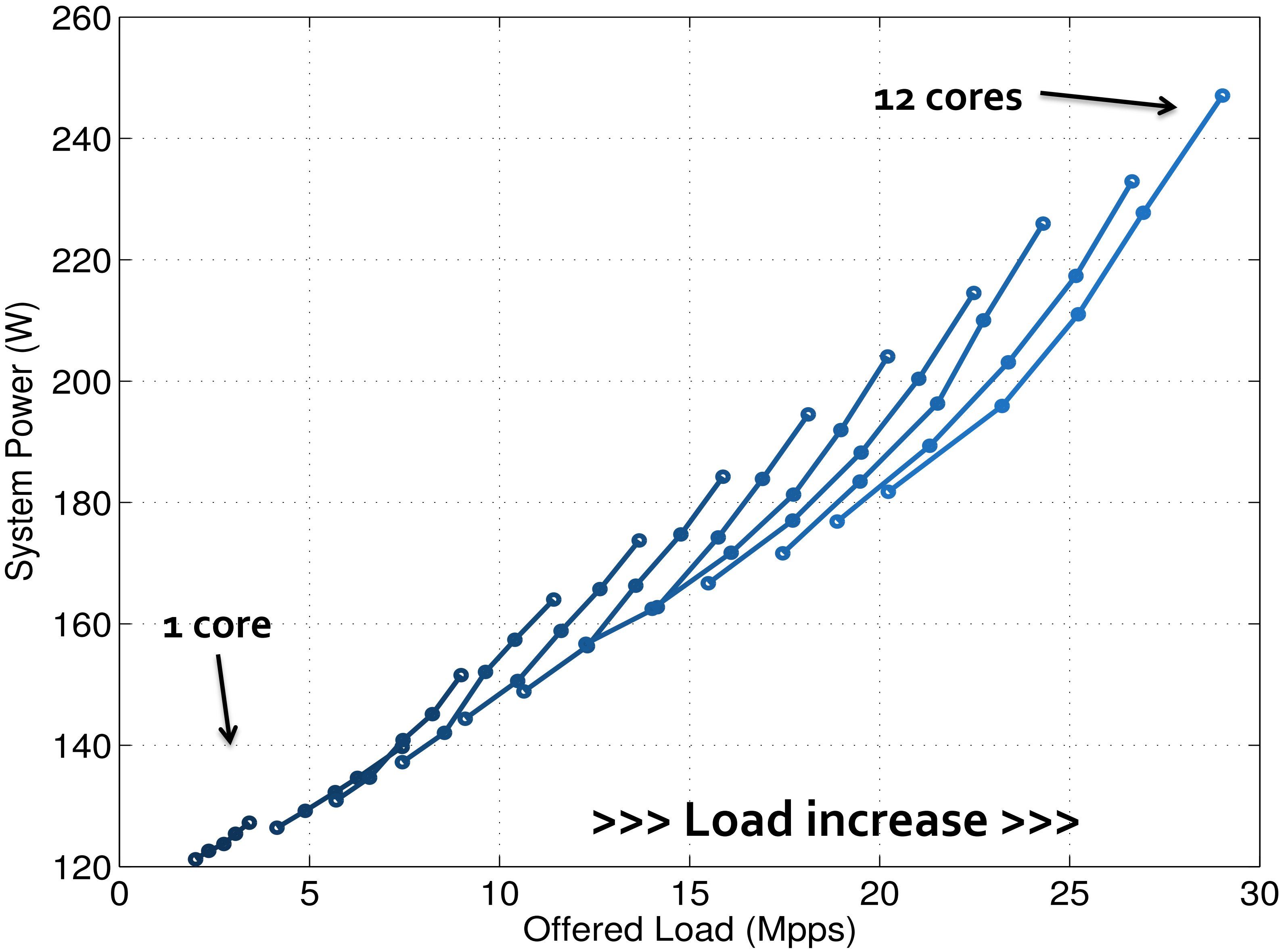
Power consumption breakdown

Power-saving algorithms guidelines

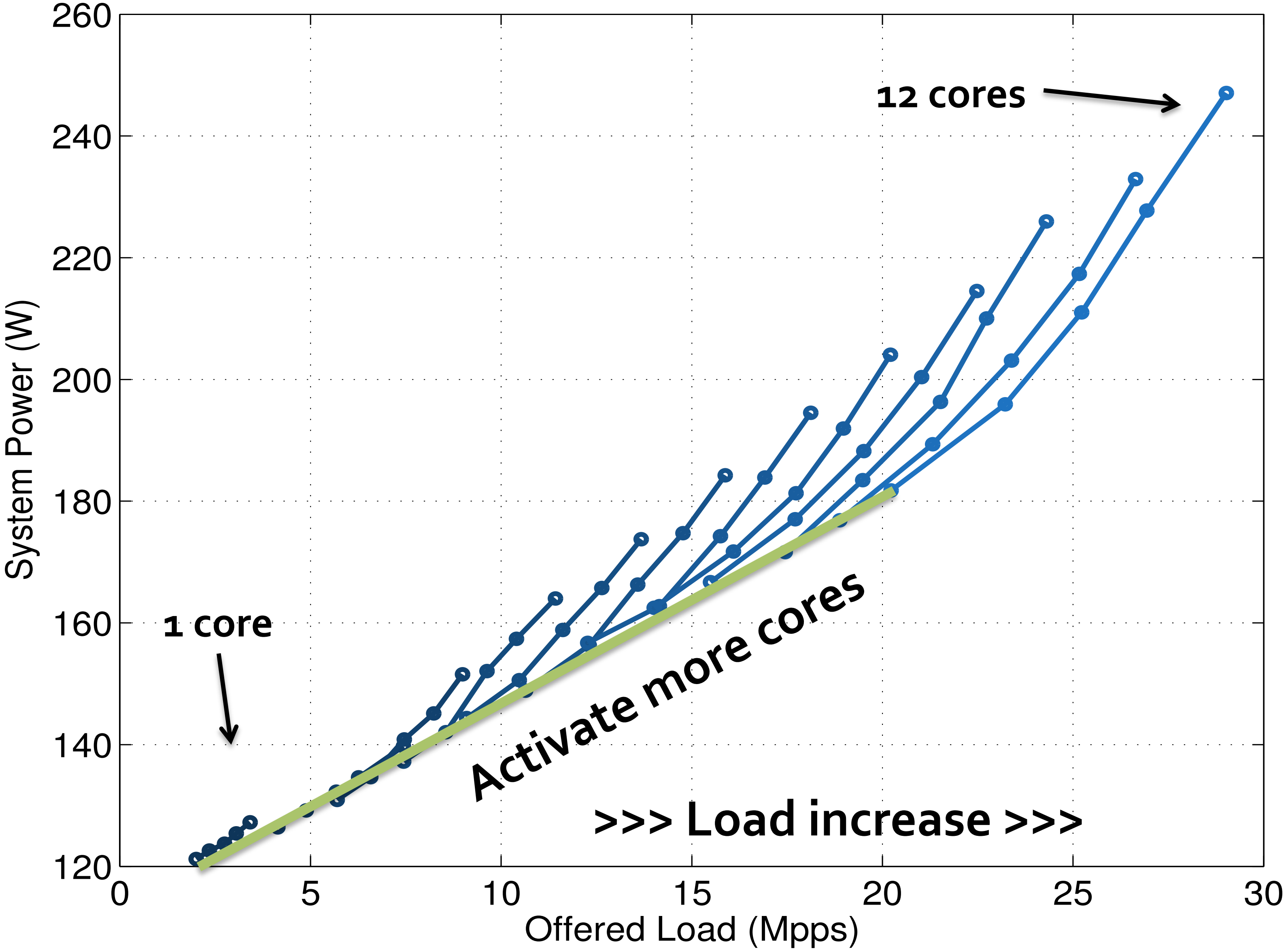
Online algorithm implementation

Performance Evaluation

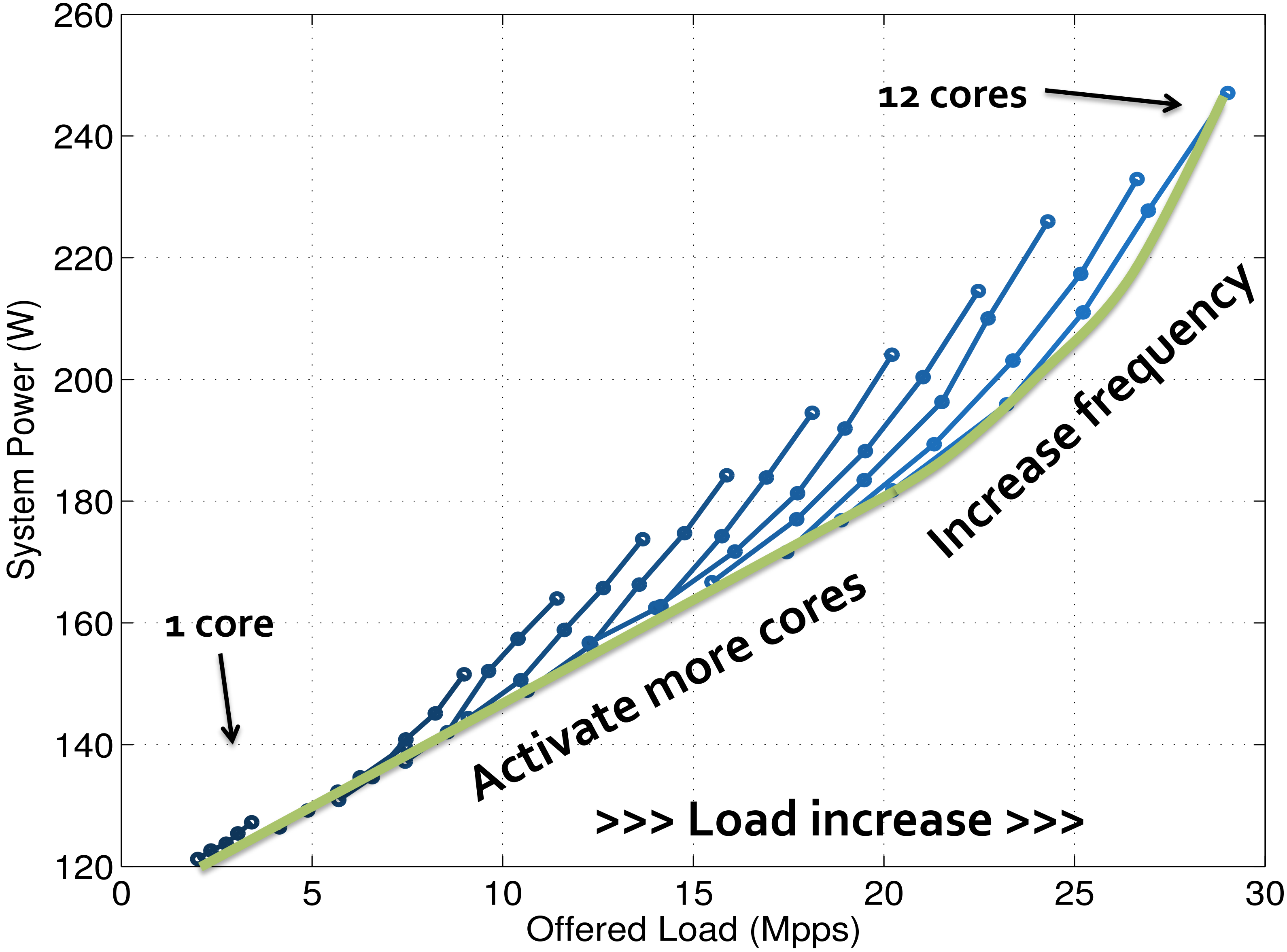
Power envelope



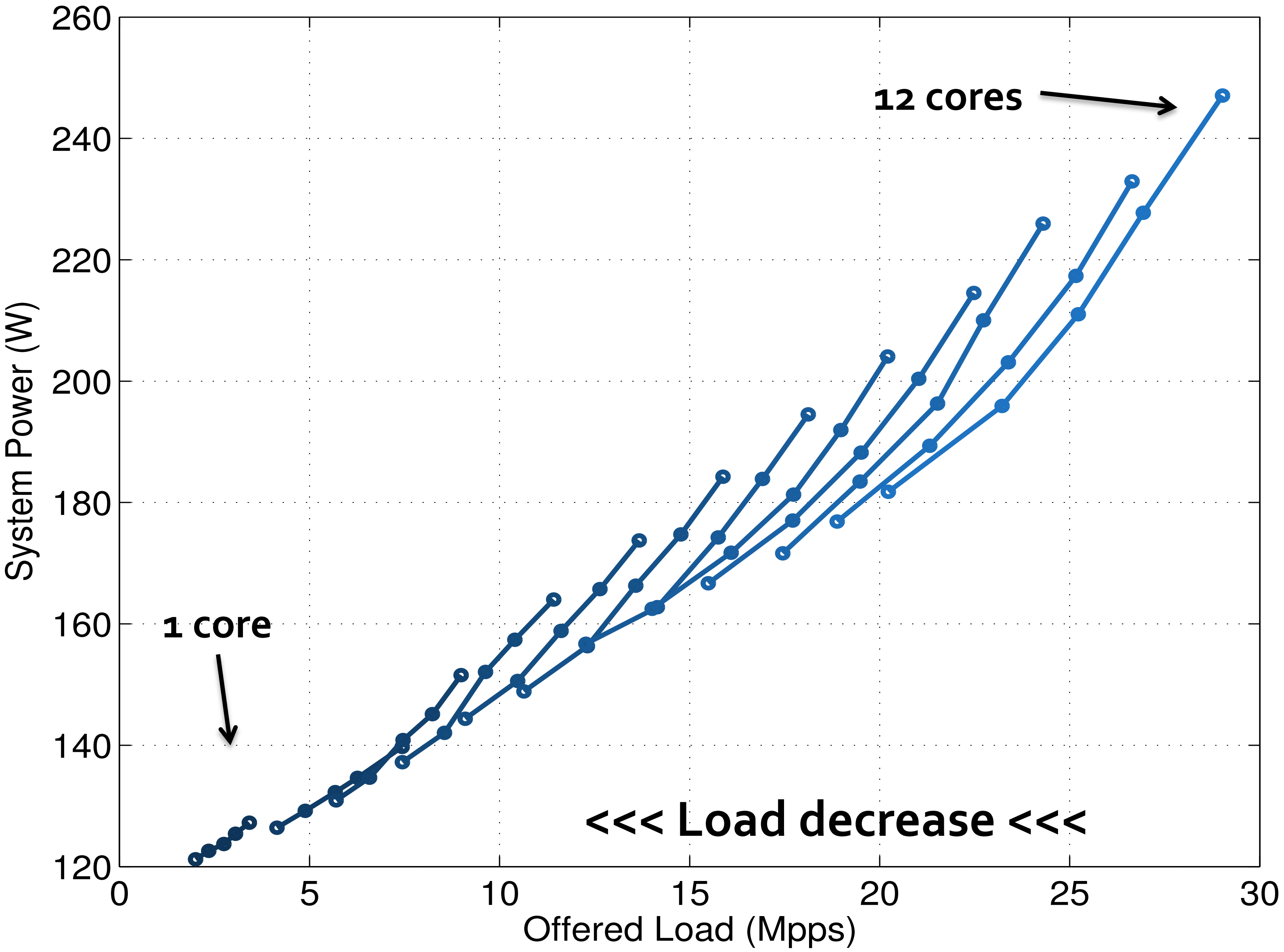
Power envelope



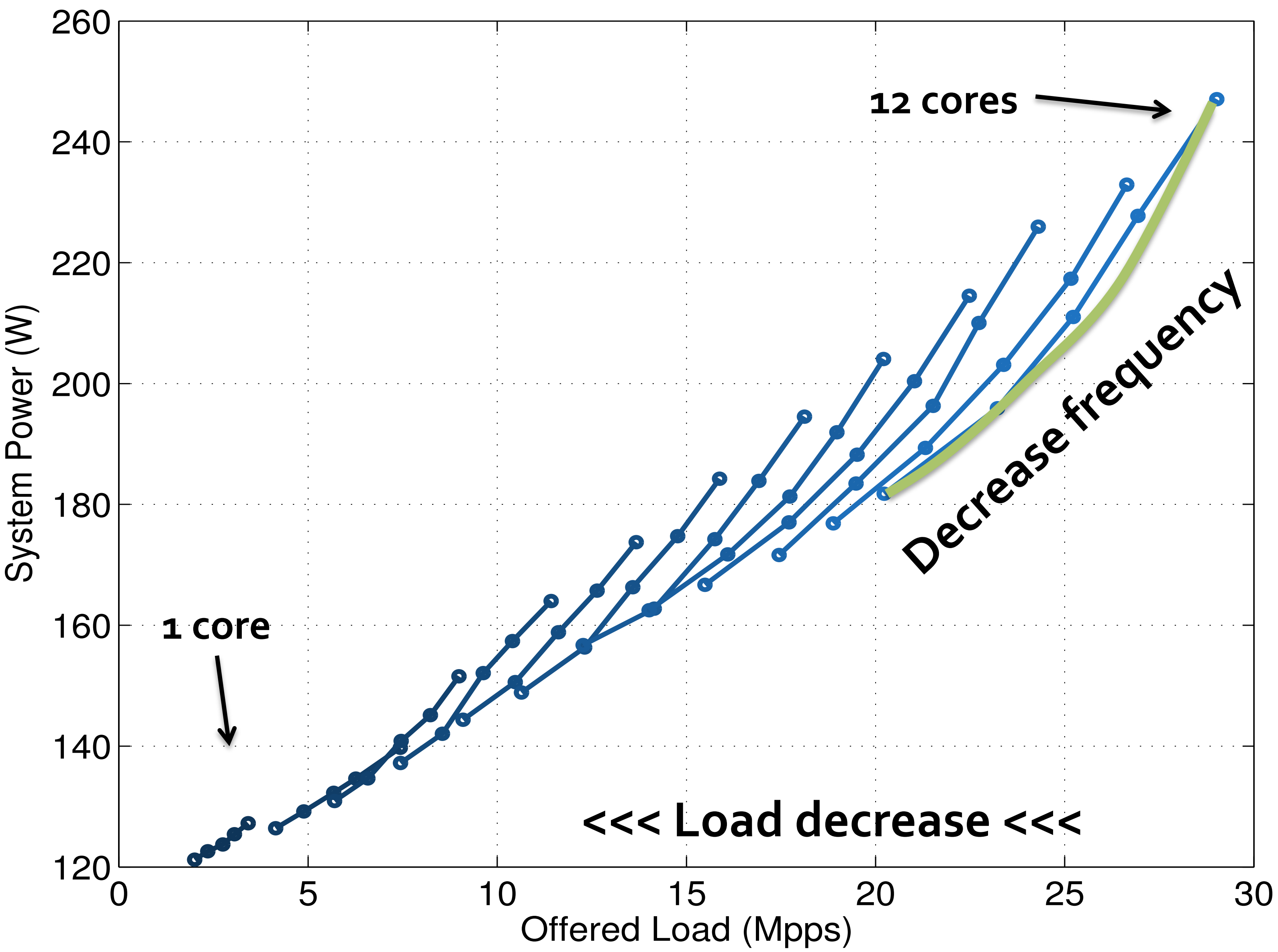
Power envelope



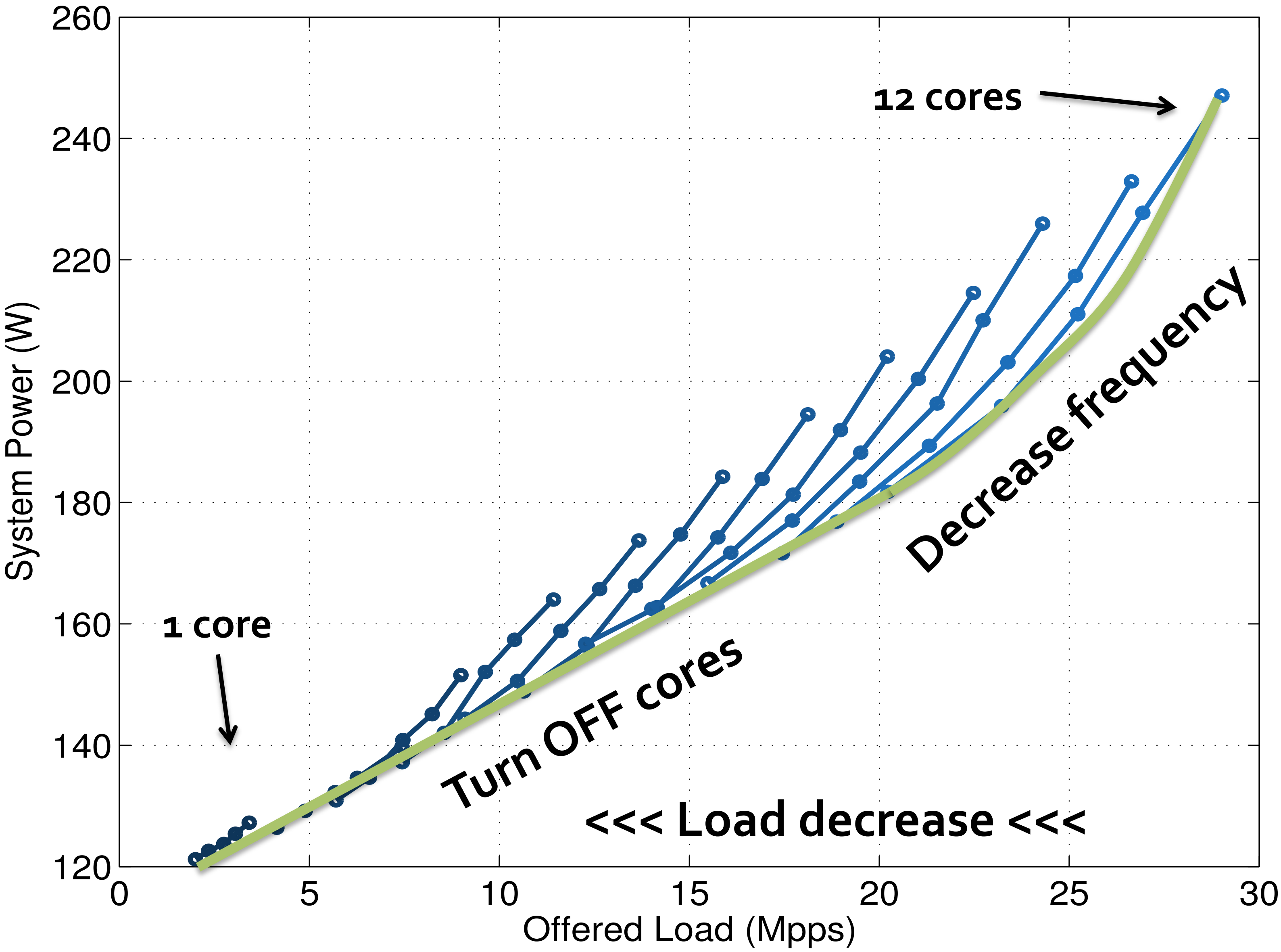
Power envelope



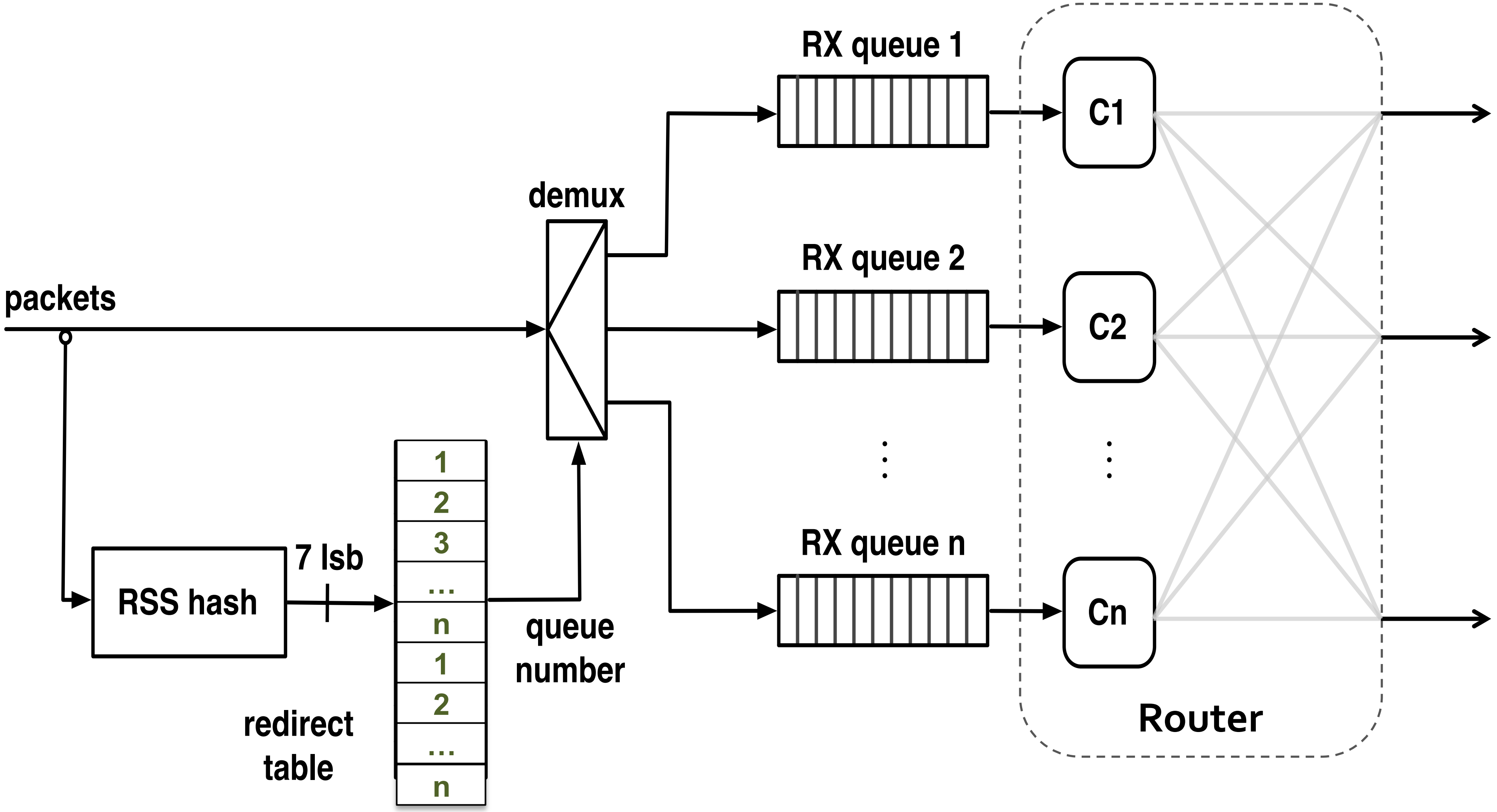
Power envelope



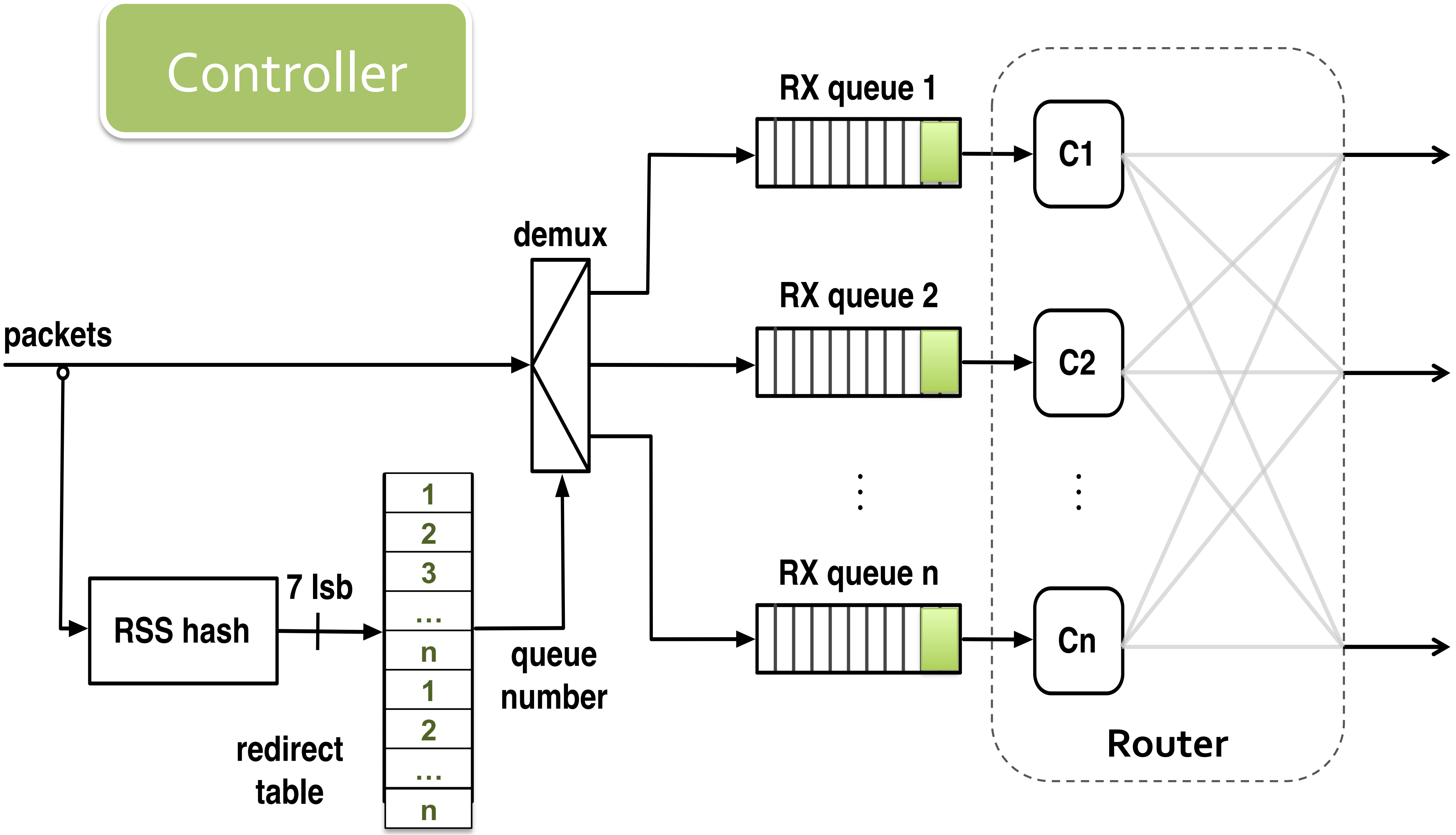
Power envelope



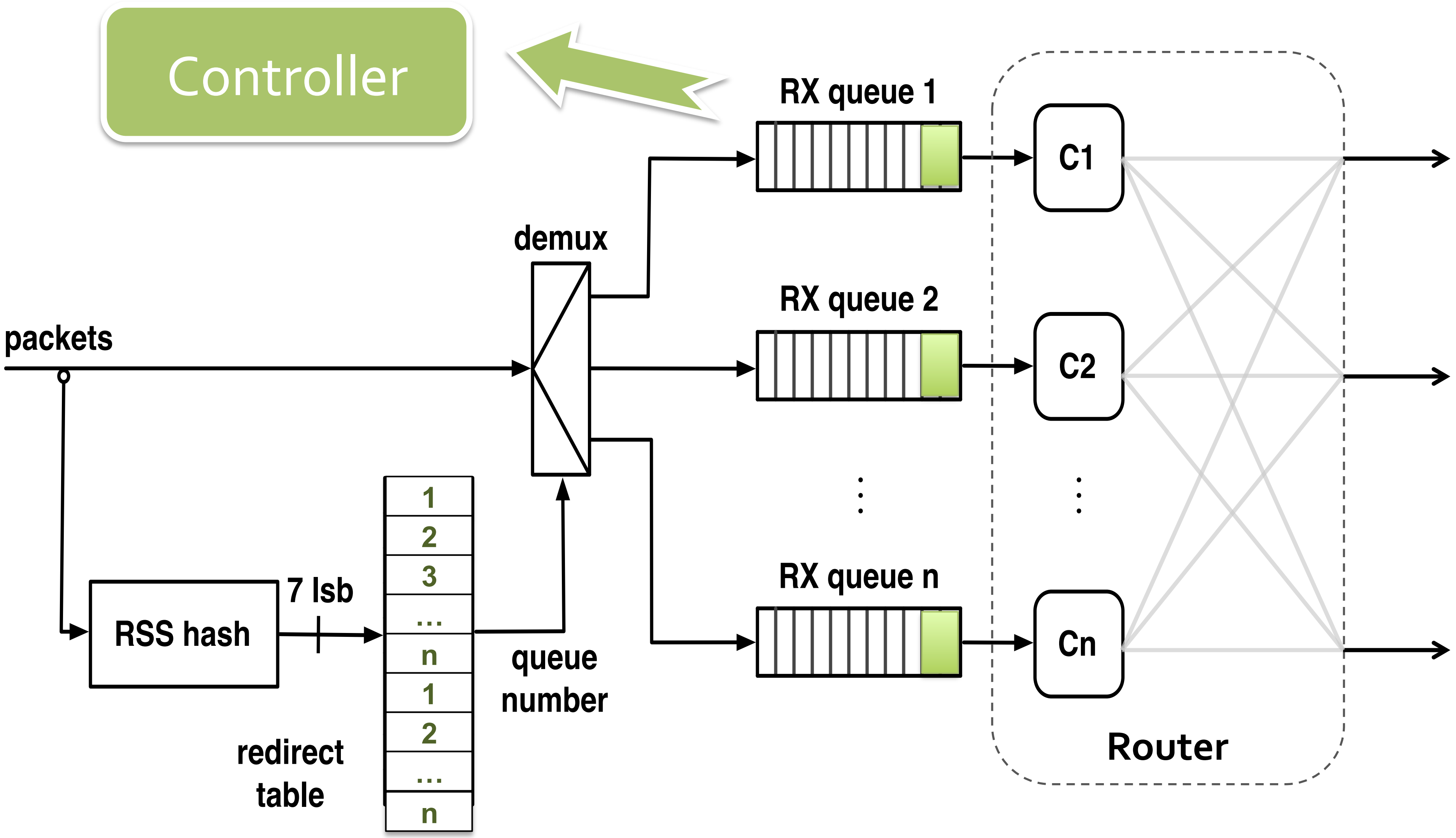
Implementation



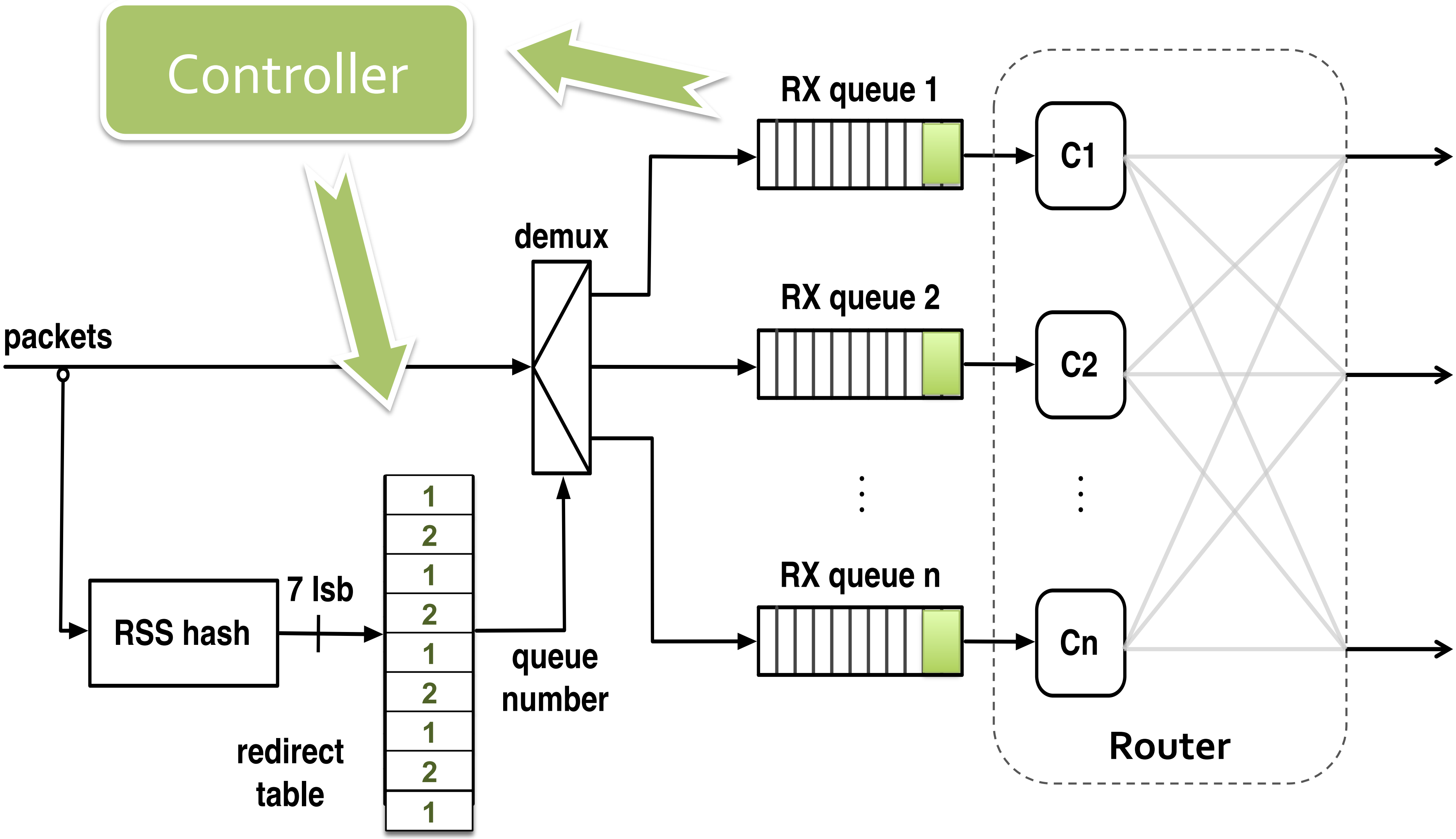
Implementation



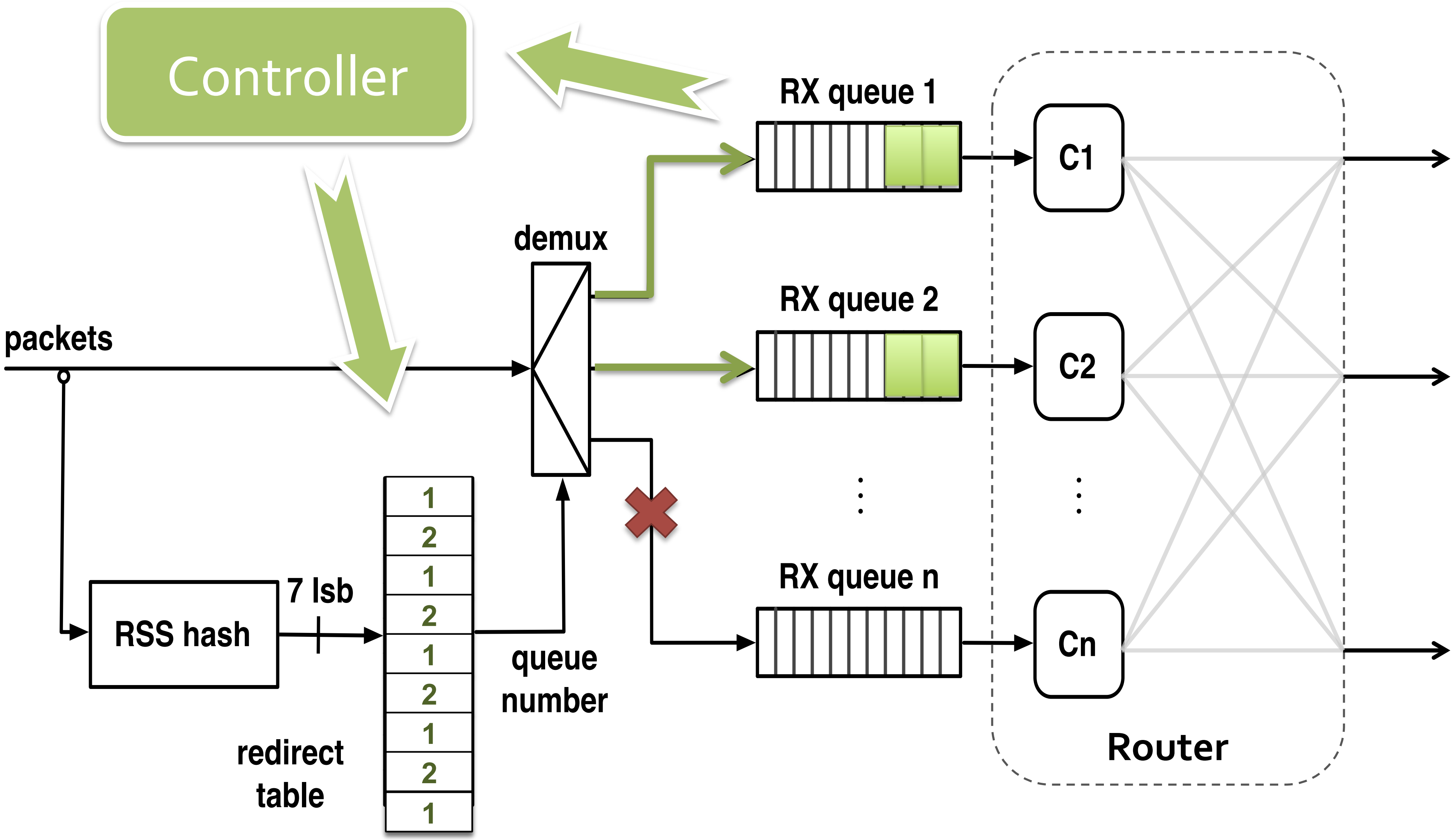
Implementation



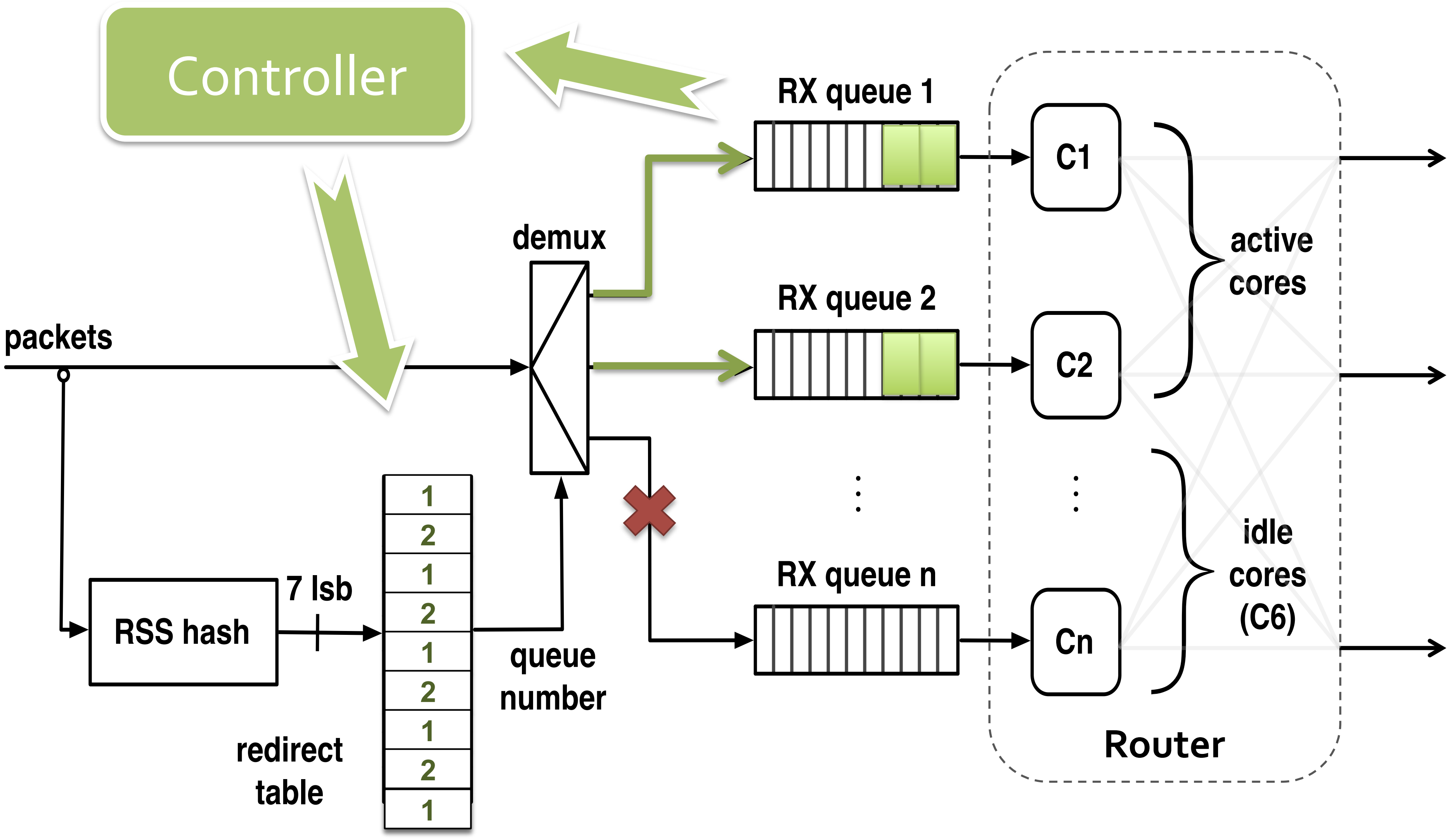
Implementation



Implementation



Implementation



Outline

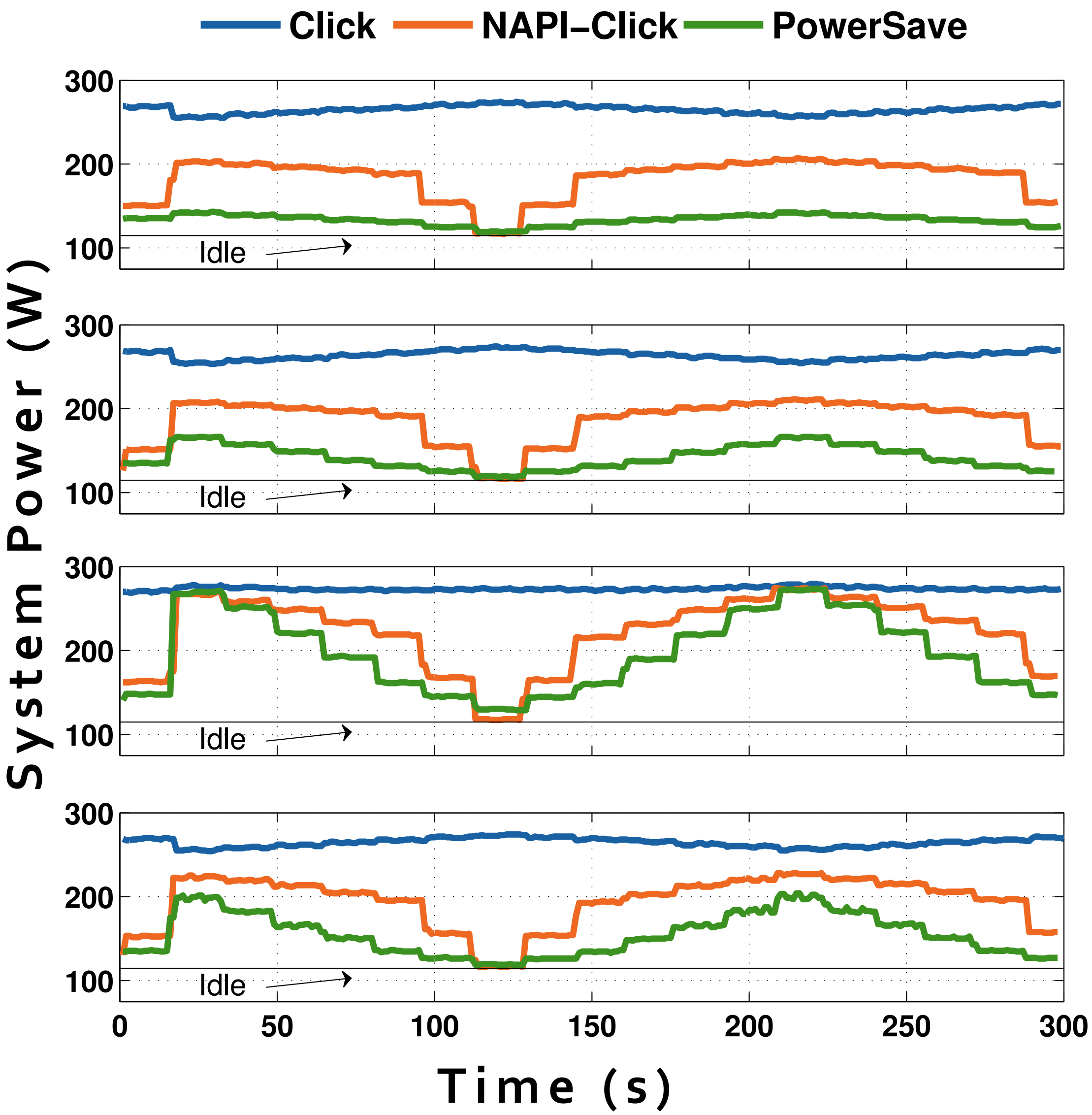
Power consumption breakdown

Power-saving algorithms guidelines

Online algorithm implementation

Performance Evaluation

Power consumption

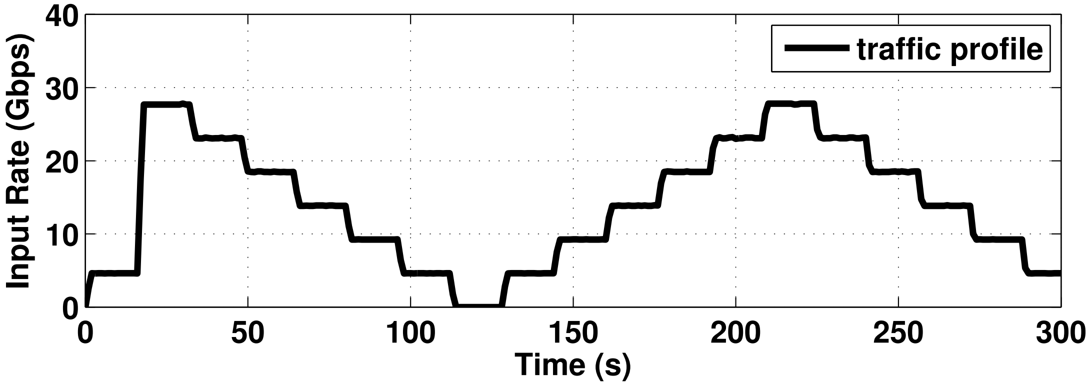


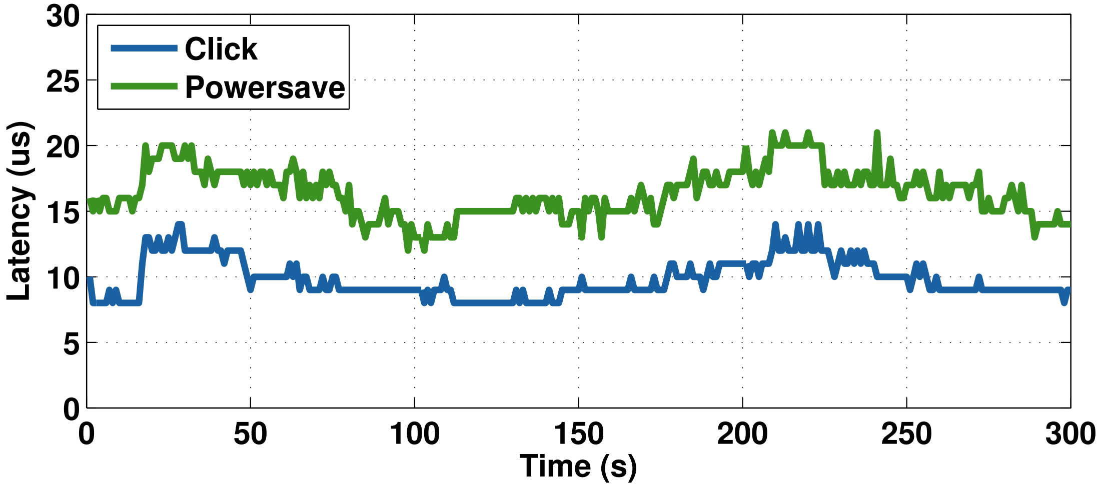
	Savings, compared to NAPI-Click
IPv4 Routing	28%
NetFlow	24%
IPSEC	12%
WAN Optimizer	20%

Latency / Loss / Reordering

Latency

≫ ~10µs increase on average compared to polling





No Packet Loss

No Reordering

≫ could happen when waking up a queue

Conclusion

Algorithm guidelines

- Run the smallest number of cores at the minimum frequency
- Increase number of cores before increasing the frequency

≫ Make the best use of power-hungry resources

On-Line algorithm implementation

≫ Monitor queue length and react quickly

- Make sure that queues can absorb traffic during sleep states transitions

Up to 50% savings are possible

≫ Depending on the application