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)

\[ \text{Idle Power vs. Exit Latency tradeoff} \]

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

- Single Processor
- Dual Processor
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 $n \times k$ cores at frequency $f/n$

```
16
1
2
3
4
5
6
0.02
0.04
0.06
0.08
0.1
0.12
```

Energy Efficiency (Mpkts/J)

2$k$ cores
$f=1.6$ GHz

$k$ cores
$f=3.2$ GHz

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?

<table>
<thead>
<tr>
<th>C-State</th>
<th>System Power (12 cores)</th>
<th>Exit Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>133 W</td>
<td>&lt; 1 us</td>
</tr>
<tr>
<td>C3</td>
<td>120 W</td>
<td>~ 60 us</td>
</tr>
<tr>
<td>C6</td>
<td>115 W</td>
<td>~ 87 us</td>
</tr>
</tbody>
</table>

Best for inactive Cores
Multicore case – Sleep states

How to operate inactive and underutilized cores?

<table>
<thead>
<tr>
<th>C-State</th>
<th>System Power (12 cores)</th>
<th>Exit Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>133 W</td>
<td>&lt; 1 us</td>
</tr>
<tr>
<td>C3</td>
<td>120 W</td>
<td>~ 60 us</td>
</tr>
<tr>
<td>C6</td>
<td>115 W</td>
<td>~ 87 us</td>
</tr>
</tbody>
</table>

Let underutilized cores take quick and light naps (C1)
Outline

Power consumption breakdown

Power-saving algorithms guidelines

Online algorithm implementation

Performance Evaluation
Power envelope

- **Offered Load (Mpps)**
- **System Power (W)**

- 1 core
- 12 cores

>>> Load increase >>>
Power envelope

[Graph showing the relationship between offered load (Mpps) and system power (W) with multiple lines indicating different numbers of cores, with a note on activating more cores and a load increase.]
Power envelope

![Graph showing system power consumption with varying offered load and core activation. The graph illustrates the relationship between offered load (Mpps) and system power (W) for different core counts.](image-url)
Power envelope

- 1 core
- 12 cores

<<< Load decrease >>>
Power envelope

System Power (W) vs Offered Load (Mpps)

- 1 core
- 12 cores

Decrease frequency

<<< Load decrease >>>
Power envelope

- Offered Load (Mpps)
- System Power (W)

1 core

12 cores

Turn OFF cores

<< Load decrease <<

Decrease frequency

<<< Load decrease <<<
Implementation

packets

RSS hash

7 lsb

redirect table

queue number

demux

RX queue 1

C1

RX queue 2

C2

RX queue n

Cn

Router
Implementation
Implementation

Controller

packets

RSS hash

7 lsb

redirect table

queue number

demux

RX queue 1

RX queue 2

RX queue n

Router

C1

C2

Cn
Implementation

Controller

packets

RSS hash

redirect table

7 lsb

queue number

demux

RX queue 1

RX queue 2

RX queue n

C1

C2

Cn

Router
Implementation

Controller

packets

RSS hash

redirect table

1 2 1
2 1 2
1 2 1
2 1 2
1

7 lsb

queue number

demux

RX queue 1

RX queue 2

RX queue n

C1

C2

Cn

Router
Implementation

Controller

packets

RSS hash

7 lsb

redirect table

queue number

demux

RX queue 1

RX queue 2

RX queue n

Router

C1

C2

Cn

active cores

idle cores (C6)
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%

Time (s)
System Power (W)
Latency / Loss / Reordering

Latency

\[ \gg \sim 10\mu s \text{ increase on average compared to polling} \]

No Packet Loss

No Reordering

\[ \gg \text{could happen when waking up a queue} \]
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