Chapter 8
Energy-Efficient Parallel Packet Forwarding

Weirong Jiang and Viktor K. Prasanna

Abstract. As the Internet traffic continues growing rapidly, parallel packet forwarding becomes a necessity in Internet infrastructure to meet the throughput requirement. On the other hand, energy/ power consumption has been a critical challenge for Internet infrastructure. It has been shown that two thirds of power dissipation inside a core router is due to packet forwarding. This chapter studies the problem of energy-efficient parallel packet forwarding in Internet Infrastructure. According to whether the data structure is shared or duplicated among multiple engines, two types of parallel packet forwarding systems are discussed. For the system with shared data structure, we study how to partition the data structure and map onto multiple engines, so that the worst-case energy/ power consumption is minimized. For the system with duplicated data structure, we formulate as an optimization problem how to distribute traffic load onto multiple engines to minimize the overall power consumption while satisfying the throughput demand.

8.1 Introduction

The Internet infrastructure consists of routers and switches to achieve interconnectivity. Its primary function is to forward packets, where the header information (such as the destination IP address) extracted from each packet is looked up in the forwarding table inside the routers/ switches. With the network traffic growing rapidly, the throughput requirement becomes difficult to meet by using a single packet forwarding engine. For example, current backbone link rates have been pushed beyond OC-768 (40 Gbps) rate, which requires a throughput of 125 million packets per

Weirong Jiang
Juniper Networks Inc., Sunnvyale, California, USA
e-mail: weirongj@acm.org

Viktor K. Prasanna
University of Southern California, Los Angeles, California, USA
e-mail: prasanna@usc.edu

second (MPPS) for minimum size (40 bytes) packets. Employing multiple packet forwarding engines has been a standard in today's routers/switches. As depicted in Figure 8.1, in such a system, multiple packet forwarding engines process the network traffic in parallel. For each incoming packet, the dispatcher will decide which forwarding engine will be assigned to process the packet. When the traffic assigned to an engine is higher than the processing capacity of the engine, a queue is needed to buffer the packets.

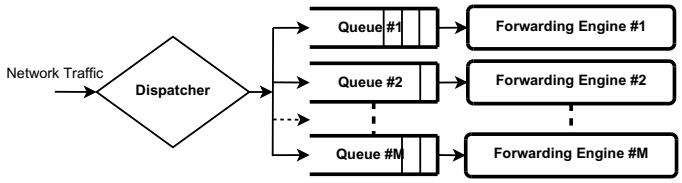

Fig. 8.1 Conceptual model of a parallel packet forwarding system.

8.1.1 Challenges

Most research in parallel packet forwarding is on balancing the traffic among these engines to achieve high throughput [8, 4, 29]. Little of them take power or energy consumption into account. On the other hand, as routers achieve aggregate throughputs of trillions of bits per second, power consumption has become an increasingly critical concern in backbone router design [30, 10]. Some recent investigations [23, 7] show that power dissipation has become the major limiting factor for next generation routers and predicts that expensive liquid cooling may be needed in future routers. As shown in Figure 8.2, the capacity of backbone routers used to double every 18 months until 5 years ago. Today, terabit routers with 10–15 kW are at the limit due to the power density. As a result, a thirty-fold shortfall in capacity by 2015 is foreseen as compared to the historical trend for single rack routers [23]. Recent analysis by researchers from Bell labs [23] reveals that, almost two thirds of power dissipation inside a core router is due to packet forwarding engines.

8.1.2 Related Work

Various techniques including data structure, hardware, system or network-level optimization [10, 13, 12, 7, 26] have been proposed to reduce the power consumption of routers and/or switches.

In [22] clock gating is used to turn off the clock of unneeded processing engines of multi-core network processors to save dynamic power when there is a low traffic workload. A finer-grained clock gating scheme is proposed in [12] to lower the

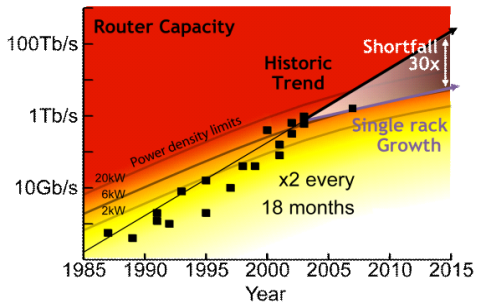

Fig. 8.2 Router capacity limited by power [23].

dynamic power consumption of pipelined forwarding engines. In [18] the more aggressive approach of turning off these processing engines is used to reduce both dynamic and static power consumption. Dynamic frequency and voltage scaling are used in [16] and [25], respectively, to reduce the power consumption of the processing engines.

Chabarek et al. [7] enumerate the power demands of two widely used Cisco routers. The authors further use mixed integer optimization techniques to determine the optimal configuration at each router in their sample network for a given traffic matrix. Nedevschi et al. [26] assume that the underlying hardware in network equipment supports sleeping and dynamic voltage and frequency scaling. The authors propose to shape the traffic into small bursts at edge routers to facilitate sleeping and rate adaptation.

Though reducing the power consumption of the Internet infrastructure has been a topic of significant interest, little of the existing work focuses on the *parallel* packet forwarding systems.

8.1.3 Organization

The rest of this chapter is organized as follows. Section 8.2 introduces the background on parallel packet forwarding systems which can be classified into two types: *heterogeneous* and *homogeneous* systems, based on whether the data structure is shared or duplicated among multiple engines. Section 8.3 considers the heterogeneous system and discusses the solution to partitioning and mapping the data structure onto the multiple pipelined packet forwarding engines to achieve energy efficiency. Section 8.4 focuses on the homogeneous system and develops a theoretical framework to minimize the overall power consumption while satisfying the throughput demand. Section 8.5 summarizes this chapter.

8.2 Background

8.2.1 Data Structure for Packet Forwarding

The entries in the forwarding table are specified using prefixes. The kernel of packet forwarding is IP lookup i.e. longest prefix matching. The most common data structure for IP lookup is some form of trie [28]. A trie is a binary tree, where a prefix is represented by a node. The value of the prefix corresponds to the path from the root of the tree to the node representing the prefix. The branching decisions are made based on the consecutive bits in the prefix. A trie is called a uni-bit trie if only one bit at a time is used to make branching decisions. Figure 8.3 (b) shows the uni-bit trie for the prefix entries in Figure 8.3 (a).

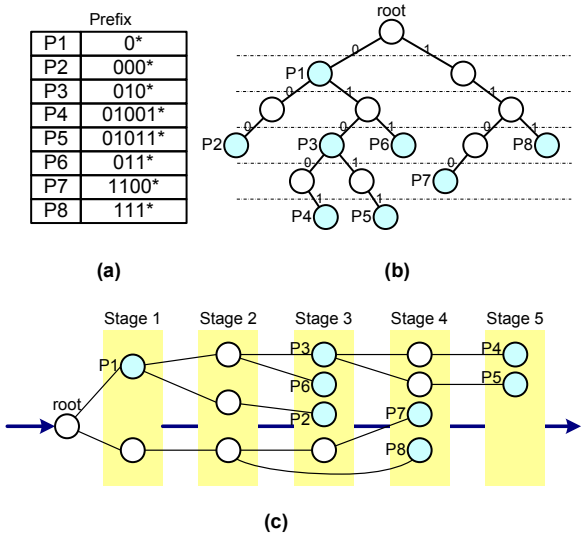

Fig. 8.3 (a) Prefix entries; (b) Uni-bit trie; (c) Fine-grained trie-to-pipeline mapping.

Given a uni-bit trie, IP lookup is performed by traversing the trie according to the bits in the IP address. The information about the longest prefix matched so far, which is updated when meeting a node containing a prefix, is carried along the traversal. Thus, when a leaf is reached, the longest matched prefix along the traversed path is returned. The time to look up a uni-bit trie is equal to the prefix length.

8.2.2 Pipelined Packet Forwarding Engines

Pipelining can dramatically improve the throughput of trie-based solutions. A straightforward way to pipeline a trie is to assign each trie level to a separate stage.

Each stage contains a block of static random addressable memory (SRAM) that stores the trie nodes. As a result, a lookup request can be issued every clock cycle [3]. But such a simple scheme results in unbalanced memory distribution across the pipeline stages. In an unbalanced pipeline, the global clock rate is determined by the access time to the "fattest" stage. Since it is unclear at hardware design time which stage will be the "fattest", we must allocate memory with the maximum size for each stage. Such an over-provisioning results in memory wastage and excessive power consumption [2]. Some proposed solutions [2, 19] balance the memory distribution across stages at the cost of lowering the throughput. In [14], a fine-grained node-to-stage mapping scheme is proposed for linear pipeline architectures. It is based on the heuristic that allows any two nodes on the same level of a trie to be mapped onto different stages of a pipeline. The heuristic is enabled by storing in each node the distance value to its child nodes. When a packet is passed through the pipeline, the distance value is decremented by 1 when it goes through each stage. Only when the distance value becomes 0, the child node's address is used to access the memory in that stage. Balanced memory distribution across pipeline stages is achieved, while a high throughput of one packet per clock cycle is sustained. Figure 8.3 (c) shows the mapping result for the uni-bit trie in Figure 8.3 (b) using the fine-grained mapping scheme.

8.2.3 Parallel Packet Forwarding

A parallel packet forwarding system consists of multiple engines. The number of engines is denoted P. According to whether the data structure is shared or duplicated among the multiple engines, parallel packet forwarding systems can be classified into two types. When the data structure is shared, each engine contains a different subset of the forwarding table. The full forwarding table is partitioned into P subsets and mapped onto the P engines. Any incoming packet is mapped to only one subset, and thus processed by the engine containing that subset. We call such a system to be *heterogeneous*. Essential to the heterogeneous system is how to perform the partitioning and mapping, which guides the construction of the dispatcher. When the data structure is duplicated, each engine contains the same forwarding table. An incoming packet can go through any of these engines. We call such a system to be *homogeneous*. Essential to the homogeneous system is the dispatcher that distributes incoming traffic load to the multiple forwarding engines. The basic goal of the dispatcher is to fully utilize the capacity of the multiple engines to maximize the overall throughput.

8.3 Energy-Efficient Multi-pipeline Architecture

8.3.1 Motivation

Ternary Content Addressable Memories (TCAMs), where a single clock cycle is sufficient to perform one IP lookup, are popular in today's routers. However,

TCAMs do not scale well in terms of clock rate, power consumption, or chip density [14]. Taylor [30] estimates that the power consumption per bit of TCAMs is on the order of 3 micro-Watts, which is 150 times more than for Static Random Access Memories (SRAMs). On the other hand, SRAM-based pipeline architectures have been developed as a promising alternative to TCAMs for high performance packet forwarding engines in next generation routers [3, 14, 6]. However, most SRAM-based solutions still suffer from high power consumption [32], which comes mainly from two sources. First, the size of the memories to store the search structure is large. Second, the number of accesses to these large memories (i.e. the pipeline depth), is large. The overall power consumption for one IP lookup can be expressed as in Equation 8.1:

$$Power_{overall} = \sum_{i=1}^{H} [P_m(S_i, \cdots) + P_l(i)] \tag{8.1}$$

Here, H denotes the number of memory accesses, i.e. the pipeline depth, $P_m(.)$ the function of the power dissipation of a memory access (which usually has a positive correlation with the memory size), S_i the size of the ith memory being accessed, and $P_l(i)$ the power consumption of the logic associated with the ith memory access. Since the logic dissipates much less power than the memory in these architectures [19, 12], our main focus is on reducing the power consumption of the memory accesses. Note that the power consumption of a single memory is affected by many other factors, such as the fabrication technology and sub-bank organization, which are beyond the scope of this paper. According to Equation 8.1, to reduce the worst-case power consumption, we should bound the number of memory accesses, as well as minimize the memory size for each access.

To reduce the memory size, we exploit chip-level parallelism by partitioning the forwarding table and allowing IP lookup be performed on only one of the partitions. We map the partitioned forwarding table onto multiple SRAM-based pipelines to ensure each pipeline requires the same amount of memory. We propose a two-phase partitioning scheme, including an effective method called *height-bounded split*, to partition a routing trie into several height-bounded subtries. As a result, all the IPs traverse the subtries in a bounded number of memory accesses. The partitioning scheme is enabled by using small TCAMs as part of the index table in the dispatcher.

8.3.2 Algorithms

First, we define the following terms.

Definition 1. The *depth* of a trie node is the directed distance from the trie node to the trie root. The depth of a trie refers to the maximum depth of all trie leaves.

Definition 2. The *height* of a trie node is the maximum directed distance from the trie node to a leaf node. The height of a trie refers to the height of the root. In fact, the depth of a trie is equal to its height.

Definition 3. The *size* of a trie is the number of nodes in the trie.

In the worst case, the number of memory accesses needed for an IP lookup is equal to the trie height. We propose a holistic scheme to (1) partition a full routing trie into many height-bounded subtries, and (2) map those subtries onto multiple pipelines so that each pipeline contains equal numbers of trie nodes. By these means, each IP lookup is completed through a bounded number of accesses on small-size memories, so that power efficiency is achieved according to Equation 8.1.

Our scheme consists of two phases: prefix expansion and height-bounded split, as illustrated in Figure 8.4 (a) and (b).

8.3.2.1 Prefix Expansion

Several initial bits are used as the index to partition the trie into many disjoint subtries. The number of initial bits to be used is called the *initial stride*, denoted I. A larger I can result in more subtries of smaller height, which can help balance the memory distribution among pipelines as well as across stages when mapping subtries to pipelines. However, a large I can result in size (prefix) expansion, where the sum of the sizes (number of prefixes) of all subtries is larger than the size (number of prefixes) of the original trie. In core routers, the majority of prefixes has lengths between 16 and 24 [11]. $I < 16$ does not result in much size (prefix) expansion.

8.3.2.2 Height-Bounded Split

After prefix expansion, we obtain $K = 2^I$ subtries, whose heights can vary from 1 to $32 - I$. We must partition these subtries further, to reduce the height of the resultant subtries, with minimum overhead. Meanwhile, we must map those subtries onto multiple pipelines to achieve a balanced memory allocation among those pipelines. In other words, given D pipelines (denoted P_i, $i = 1, \cdots, D$) and K subtries each of which has W_i nodes ($i = 1, 2, \cdots, K$), we must ensure that each pipeline (except the last pipeline) contains $B_P = \left\lceil \frac{\sum_{i=1}^{K} W_i}{D} \right\rceil$ nodes.

The above problem is a variant of packing problem [17], and is NP-hard. We develop a heuristic that allows a subtrie to be split and mapped onto different pipelines. First, we sort those subtries obtained from prefix expansion, in decreasing order of size. Then we traverse those subtries one by one. Each subtrie is traversed in the post-order. Once the height bound, denoted B_H, is reached, a new subtrie is split. After the number of nodes mapped onto a pipeline exceeds the size bound of the pipeline, B_P, we map the rest of nodes onto the next pipeline. Algorithm 1 shows a recursive implementation of the height-bounded split, where $size(P_i)$ denotes the number of nodes mapped onto the i th pipeline.

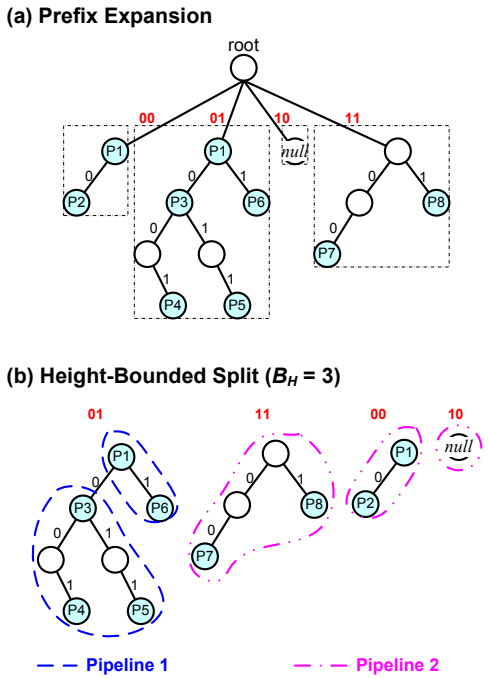

Fig. 8.4 Partition the trie shown in Figure 8.3 (b) and map onto 2 pipelines. The value of the height bound is $B_H = 3$.

8.3.3 Implementation Issues

After height-bounded splitting, the resulting subtries may be rooted at different depths of the original trie. For the subtries rooted at the depth of I, we use an index SRAM (called SRAM_A). For the rest of the subtries, we need an index TCAM and an index SRAM (called SRAM_B). The TCAM stores the prefixes to represent the subtries. The two index SRAMs store the information associated with each subtrie: (1) the mapped pipeline ID, (2) the ID of the stage where the subtrie's root is stored, and (3) the address of the subtrie's root in that stage. Figure 8.5 shows the index table used to achieve the mapping shown in Figure 8.4.

The index table resides in the dispatcher. An arriving input IP searches the index SRAM_A and the index TCAM in parallel. I initial bits of the input IP are used to index the SRAM_A. Meanwhile, the entire input IP searches the index TCAM and obtains the subtrie ID corresponding to the longest matched prefix. Then, the IP uses the subtrie ID to index the SRAM_B to retrieve the associated information. The result obtained from the SRAM_B has a higher priority than that from the SRAM_A. The number of entries in the SRAM_A is 2^I, while the number of entries in the index TCAM and SRAM_B is at most 2^{32-B_H}.

Algorithm 1. Height-bounded split: $HBS(n,i)$

Require: n: a node;
Require: i: the ID of the pipeline for n to be mapped on.
Ensure: i: the ID of the pipeline for the next node to be mapped on.

```
if n == null then
    Return i.
end if
i = HBS(n.left_child, i)
i = HBS(n.right_child, i)
if size(P_i) < B_P then
    Map n onto P_i.
else
    Map n onto P_{i+1}.
end if
if n.height >= B_H then
    Mark n as a subtrie root.
end if
Return i + 1.
```

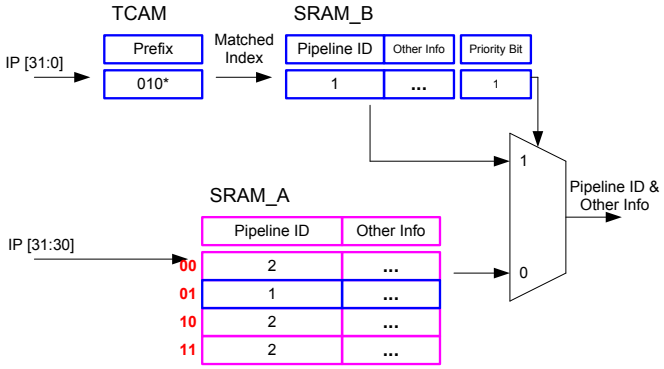

Fig. 8.5 Index table for the mapping shown in Figure 8.4

8.3.4 Performance Evaluation

We conducted simulation experiments on the 8 real-life backbone routing tables collected from [27] on 11/30/2007. Their information is listed in Table 8.1. For ASIC implementation, we used CACTI 5.3 [5] and an accurate TCAM model [1] to evaluate the performance of SRAMs and TCAMs, respectively. The number of pipelines was $D = 4$, and the initial stride for prefix expansion was $I = 12$.

Table 8.1 Representative Routing Tables

Routing table	Location	# of prefixes
RIPE-NCC	Amsterdam, Netherlands	243474
LINX	London, UK	240797
SFINX	Paris, France	238089
NYIIX	New York, USA	238836
DE-CIX	Frankfurt, Germany	243732
MSK-IX	Moscow, Russia	238461
PAIX	Palo Alto, USA	243731
PTTMetro-SP	Sao Paulo, Brazil	243242

8.3.4.1 Selection of Height Bound

How to set an appropriate height bound is an issue worth discussing. Using a small height bound can reduce the number of memory accesses for IP lookup, but it may also result in a large number of subtries, which requires a large table to index those subtires. A subtrie whose height is h may be split into $O(2^{h-B_H})$ subtries, whose heights are bounded by B_H.

We conducted experiments with various values of the height bound to evaluate such a trade-off. As Figure 8.6 shows, a smaller height bound resulted in a larger number of TCAM entries in the index table. The architecture achieved the lowest power consumption[1] when the value of the height bound was 16. In the following experiments, the height bound was $B_H = 16$, and the pipeline depth was $H = B_H + 1 = 17$.

8.3.4.2 Architecture Performance

We mapped the 8 routing tables onto a 4-pipeline 17-stage architecture. As Figure 8.7 shows, our partitioning and mapping scheme achieved a balanced memory allocation among the 4 pipelines. To perform the node-to-stage mapping, we used the same scheme as [14], to achieve balanced node distribution across stages within each pipeline. The results are shown in Figure 8.8. The first several stages of a pipeline could not be balanced, since there were few nodes at the top levels of any subtrie.

According to Figure 8.8, each stage contained fewer than 16K nodes. Thus 14 address bits were enough to index a node in the local memory of a stage. The pipeline depth was 17, and the first stage was dedicated for the subtrie roots. Thus we needed 4 bits to specify the distance for each node. Assuming each node needed 15 bits as the pointer to next-hop information, the total memory needed to store 243732 prefixes from the largest routing table DE-CIX in this architecture was $(13+4+15) \times 2^{14} \times 17 \times 4 \approx 34$ Mb $= 4.25$ MB, where each stage needed 64 KB of memory. According to CACTI 5.3 [5], a 64 KB SRAM using 65 nm technology needed 0.8017 ns to access and dissipated 0.0235 nJ power. For DE-CIX, there were 986 entries in the index TCAM. According to the TCAM model [1], a 986-row 18-bit TCAM using 65 nm technology needed 1.4653 ns to access and

[1] The measurement of the power consumption was similar as in Section 8.3.4.2.

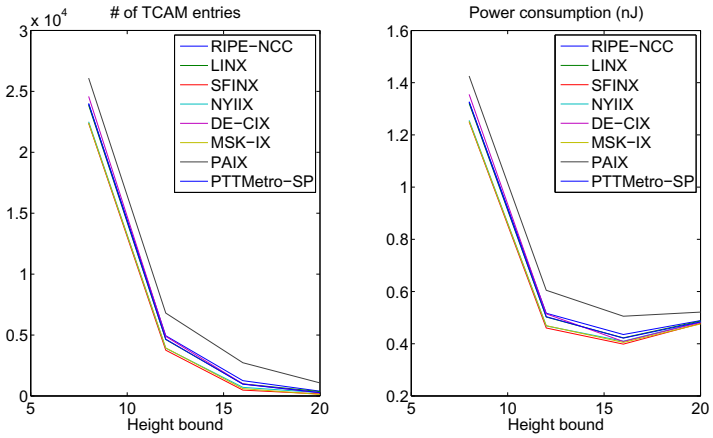

Fig. 8.6 Impact of height bound

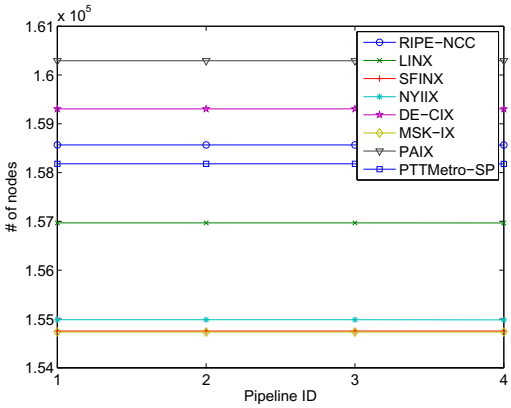

Fig. 8.7 Node distribution over 4 pipelines.

dissipated 0.0325 nJ power. The overall clock rate could achieve $\frac{1}{0.8017} = 1.25$ GHz, by using two copies of the index TCAM working in parallel. The throughput was thus 400 Gbps (i.e. 10× OC-768 rate) for minimum size (40 bytes) packets. The energy consumption for one IP lookup was $0.0235 \times 16 + 0.0325 \times 2 = 0.441$ nJ [2].

The results for the 8 routing tables are compared with the 4-partition CoolCAM [32] and IPStash [15] in Table 8.2. The CoolCAM results did not include the power dissipation of the index TCAM. The IPStash results were normalized using the best-case reduction ratio given in [15]. Our architecture achieved up to 7-fold and 3-fold reductions in power consumption over CoolCAMs and IPStash, respectively.

[2] The two index SRAMs, one with $2^{10} \times 10$ bits = 1.25 KB and the other with $2^{12} \times 18$ bits = 9 KB, were so small that their contribution to the overall performance could be ignored.

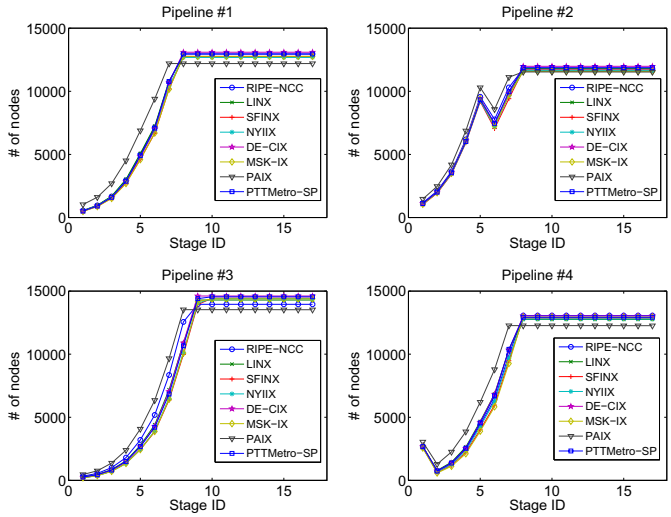

Fig. 8.8 Node distribution over 17 stages in each pipeline.

Table 8.2 Comparison on energy consumption (nJ) of different solutions

Routing table	RIPE-NCC	LINX	SFINX	NYIIX	DE-CIX	MSK-IX	PAIX	PTTMetro-SP
4-partition CoolCAM [32]	2.8881	2.8571	2.8246	2.8337	2.8920	2.8286	2.8920	2.8856
IPStash [15] (normalized)	1.1553	1.1428	1.1298	1.1335	1.1568	1.1314	1.1568	1.1543
4-pipeline arch ($B_H = 16$)	0.4353	0.4227	0.3986	0.4095	0.4410	0.4050	0.5055	0.4215

8.4 Power-Aware Load Distribution

In this section, we focus on the parallel packet forwarding system consisting of homogeneous engines. We develop a theoretical framework by modeling the power dissipation of a single engine as the function of the traffic load assigned to that engine. Then the power-aware load distribution in parallel forwarding becomes an optimization problem to minimize the overall power consumption while meeting the throughput requirement. We consider two types of power functions, which are different in terms of whether supporting sleep mode or not.

8.4.1 Optimization Framework

8.4.1.1 Problem Definition

The problem of distributing traffic load onto multiple forwarding engines to minimize the overall power consumption while satisfying the throughput requirement can be defined as follows.

$$\min \quad \sum_{i=1}^{M} P_i(x_i) \tag{8.2}$$

$$\text{s.t.} \quad \sum_{i=1}^{M} x_i = T \tag{8.3}$$

$$0 \le x_i \le B_i,\ i = 1,\ldots,M \tag{8.4}$$

In the above definition, M denotes the total number of forwarding engines and T the total throughput requirement. For the ith forwarding engine, $i = 1,\ldots,M$, x_i denotes its traffic load, $P_i(.)$ its power function with respect to its traffic load and B_i its throughput upper bound (i.e. the bandwidth). The bandwidth of each engine is lower than that of the output link attached to the engine.

8.4.1.2 Power Function of Each Engine

To solve the above optimization problem, we should first understand the power function of each engine. The most common are the following two types of power functions.

Sleep-disabled

According to [31, 24], the power consumption of most of today's network devices can be modeled as a linear function with respect to the traffic load, as shown in (8.5):

$$P(x) = ax + b \tag{8.5}$$

where x denotes the traffic load. a is the coefficient for dynamic power consumption while b the static power consumption. This type of power function assumes that the forwarding engine does not have the sleep mode. As a result, even when there is no traffic to be processed, the forwarding engine still dissipates (static) power.

Sleep-enabled

Recent work [26] proposes that next generation network devices should support sleep mode. With sleep mode enabled, the forwarding engine will not dissipate any power when there is no traffic load. Thus we have following power function for such kinds of forwarding engines.

$$P(x) = (ax+b)I_x = ax + bI_x = \begin{cases} 0 & x = 0 \\ ax+b & x > 0 \end{cases} \tag{8.6}$$

where I_x is a unit step function:

$$I_x = \begin{cases} 0 & x = 0 \\ 1 & x > 0 \end{cases} \tag{8.7}$$

8.4.1.3 Specific Case: Property-Homogeneous Engines

We start the discussion from a specific (and simple) case where all the forwarding engines have the same properties including the power function and the bandwidth. In other words, $\forall i,\ i = 1,2,\ldots,M,\ a_i = a,\ b_i = b,\ B_i = B$, where a, b and B are constants. We call these engines to be *property-homogeneous*.

1. Considering sleep-disabled forwarding engines, the objective function (1) will be

$$\sum_{i=1}^{M} P_i(x_i) = \sum_{i=1}^{M} [ax_i + b] = a\sum_{i=1}^{M} x_i + Mb = aT + Mb \tag{8.8}$$

   As a, b, T and M are given constants, there is no optimization to be done to minimize the overall power consumption.
2. Considering sleep-enabled forwarding engines, the objective function (1) will be

$$\sum_{i=1}^{M} P_i(x_i) = \sum_{i=1}^{M} [ax_i + bI_{x_i}] = a\sum_{i=1}^{M} x_i + b\sum_{i=1}^{M} I_{x_i} = aT + b\sum_{i=1}^{M} I_{x_i} \tag{8.9}$$

   Thus the optimal solution is to minimize the number of active forwarding engines while satisfying the throughput requirement. Since all the fowarding engines have the same bandwidth, the optimal solution for meeting throughput of T is to turn on $\lceil \frac{T}{B} \rceil$ forwarding engines while keeping the rest of forwarding engines to sleep. Then the overall power consumption is minimized to be: $\min \sum_{i=1}^{M} P_i(x_i) = aT + b\lceil \frac{T}{B} \rceil$.

8.4.1.4 General Case: Property-Heterogeneous Engines

In most cases, the properties of forwarding engines differ from each other. In other words, $\exists i, j,\ i, j = 1,2,\ldots,M,\ a_i \neq a_j,\ b_i \neq b_j$, and $B_i \neq B_j$. We call these engines to be *property-heterogeneous*.

1. Considering sleep-disabled forwarding engines, the objective function (1) will be

$$\sum_{i=1}^{M} P_i(x_i) = \sum_{i=1}^{M} [a_i x_i + b_i] = \sum_{i=1}^{M} a_i x_i + \sum_{i=1}^{M} b_i \tag{8.10}$$

   Then the optimization problem becomes a linear programming (LP) problem which can be solved in polynomial time. Note that $\sum_{i=1}^{M} b_i$ is constant, which will not affect the decision on traffic load distribution. Hence $\sum_{i=1}^{M} b_i$ can be omitted from (8.10) when solving the linear programming (LP) problem.
2. Considering sleep-enabled forwarding engines, the objective function (1) will be

$$\sum_{i=1}^{M} P_i(x_i) = \sum_{i=1}^{M} [a_i x_i + b_i I_{x_i}] \tag{8.11}$$

The optimization problem then becomes a non-linear programming problem which is hard to be solved. We convert it into a mixed integer programming (MIP) problem[3] by introducing a penalty parameter K to remove the unit step function. Then we have:

$$\min \quad \sum_{i=1}^{M} [a_i x_i + b_i y_i] \tag{8.12}$$

$$\text{s.t.} \quad \sum_{i=1}^{M} x_i = T \tag{8.13}$$

$$0 \leq x_i \leq B_i,\ i = 1, \ldots, M \tag{8.14}$$

$$y_i <= K * x_i,\ i = 1, \ldots, M \tag{8.15}$$

$$y_i >= x_i / K,\ i = 1, \ldots, M \tag{8.16}$$

$$y_i \in \{0, 1\} \tag{8.17}$$

We can prove that when we set $K >> \max_{i=1}^{M} B_i$, the above problem is identical to (8.11) with constraints ~~(2)(3)~~. A simple proof is as follows. If $x_i = 0$, then y_i must be 0; otherwise, i.e. $x_i > 0$, then y_i must be 1. Hence $y_i = I_{x_i}$. $i = 1, 2, \ldots, M$.

8.4.2 Implementation Issues

While we focus mainly on theoretical analysis, here we discuss briefly the issues in system design and implementation. The kernel of the dispatcher is the linear programming (LP) / mixed integer programming (MIP) solver. To obtain the parameters used in the optimization framework, we need following components in the dispatcher system.

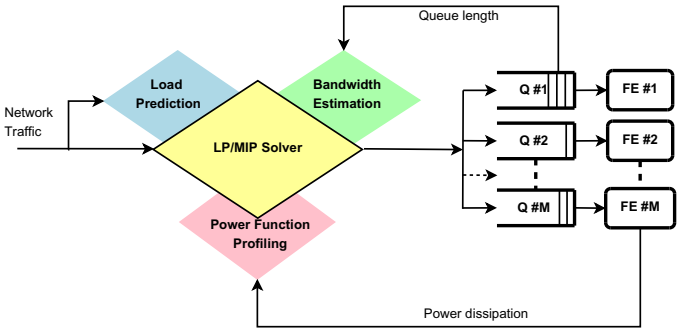

Fig. 8.9 Example of the dispatcher system design.

[3] Although a mixed integer programming (MIP) problem is *NP-hard*, there exist various computation-efficient MIP solvers.

- **Load predication** to predict T in real time.
- **Bandwidth estimation** to estimate B_i ($i = 1,2,\ldots,M$) for each forwarding engine if their values are unknown or varying.
- **Power function profiling** to retrieve the parameters a_i, b_i ($i = 1,2,\ldots,M$) for each forwarding engine if these parameters cannot be pre-determined.

Then we have the overall architecture as shown in Figure 8.9, where 'Q' and 'FE' are the abbreviations of 'Queue' and 'Forwarding Engine', respectively.

8.4.2.1 Load Predication

The optimal solution is only possible when we can predict future traffic load. Various load prediction techniques have been proposed in literature [9, 21]. One of the accurate prediction algorithms is Auto-Regressive Moving Average (ARMA) adopted in [21]. We use ARMA for load prediction in our experiments.

8.4.2.2 Bandwidth Estimation

In most cases, the bandwidth of each forwarding engine is known apriori. But in case the forwarding bandwidth of some forwarding engine is unknown or varying, we need to perform real-time estimation using other information. For example, we can monitor the queue length of a forwarding engine. Since the dispatcher keeps track of the traffic load distributed to that forwarding engine, we can infer the forwarding bandwidth based on the queue length of that engine.

8.4.2.3 Power Function Profiling

Usually we can pre-determine the power function of each forwarding engine. If we want to model the power function of a forwarding engine on the fly, we need the real-time information of the power dissipation of the engine. Since the dispatcher contains the traffic load records, we can profile the power functions based on power and load information.

8.4.3 Experimental Results

To evaluate how much power/ energy reduction can be achieved by using the proposed optimization framework, we conducted simulation experiments using real-life traffic traces from LBNL/ICSI Enterprise Tracing Project [20]. We downloaded 25 trace files collected on the same day (2005/01/07) and concatenated them into one trace. Its statistics is shown in Table 8.3 where throughput is measured in terms of the number of packets per second (PPS). Figure 8.10 depicts the traffic rate variation during the entire period.

Table 8.3 Statistics of the 18-hour traffic trace

Trace	Date	# of packets	Duration	Max. throughput	Min. throughput
LBNL/ICSI	20050107	26325056	17 hours 33 minutes	11083 PPS	0 PPS

We consider the general case where the system consists of four parallel forwarding engines whose parameters are different from each other. The system configuration for the simulation is summarized in Table 8.4. The parameters were set to comply with the power models of network devices observed in [31, 24]. Among the four forwarding engines, the third one (FE #3) represents a high-end engine and is the most power-hungry, while the fourth one (FE #4) represents a low-end engine with the least power consumption.

In following experiments, we consider two scenarios: sleep-disabled and sleep-enabled forwarding engines, respectively. We compared the performance achieved using our optimal (**power-aware**) solution with that using the traditional (**power-unaware**) load balancing -based parallel forwarding scheme.

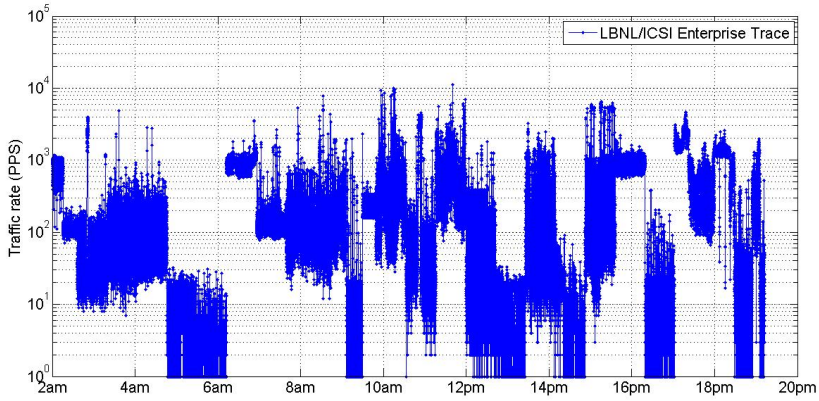

Fig. 8.10 Traffic rate variation of the trace.

Table 8.4 Summary of simulation configurations

Parameter	FE #1	FE #2	FE #3	FE #4
a	0.1	0.05	0.2	0.02
b	300	200	500	100
B	3000	1500	6000	600

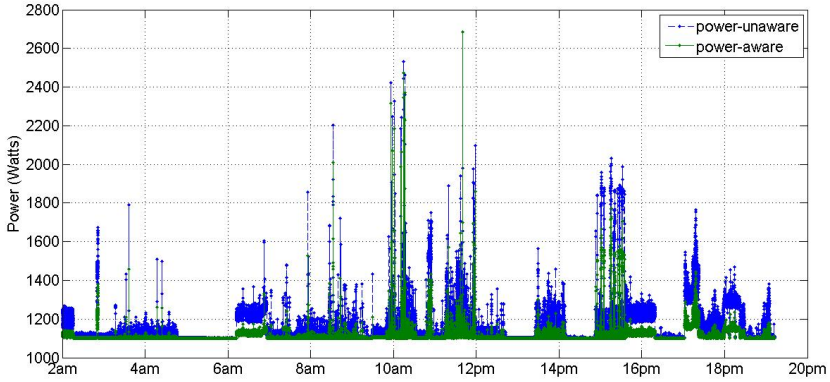

Fig. 8.11 Power consumption of power-aware versus power-unware parallel forwarding schemes (with sleep-disabled forwarding engines).

8.4.3.1 With Sleep-Disabled Engines

Figure 8.11 shows that our power-aware scheme achieved lower power consumption than the traditional power-unaware scheme, especially when the traffic load is low. Since the static power consumption cannot be reduced in sleep-disabled forwarding engines, the overall energy reduction was marginal: our solution achieved 3% lower energy consumption than the traditional scheme. However, if we consider the dynamic part only, our solution achieved **4.1**-fold reduction in energy consumption than the traditional scheme.

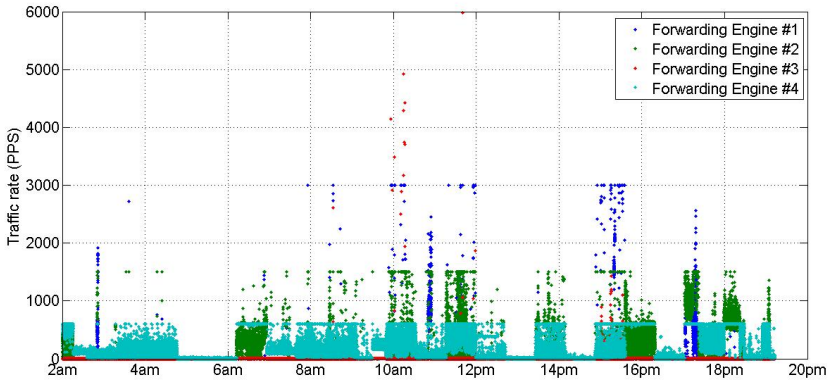

Fig. 8.12 Load distribution on 4 sleep-disabled forwarding engines with power-aware parallel forwarding.

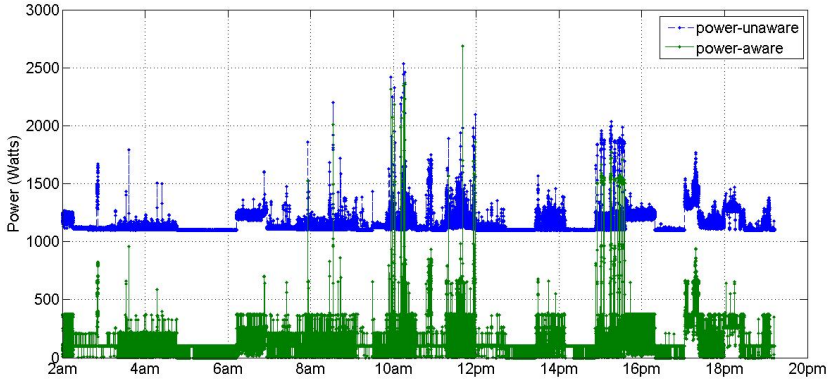

Fig. 8.13 Power consumption of power-aware versus power-unware parallel forwarding schemes (with sleep-enabled forwarding engines).

Figure 8.12 shows the load distribution on the 4 forwarding engines. We can see that, the forwarding engine with the lowest dynamic power consumption usually reached its throughput upper bound (i.e. the bandwidth). The forwarding engines with lower dynamic power consumption tended to receive higher volume of traffic than those with higher dynamic power consumption. As a result, the short-term load distribution among the forwarding engines was not balanced. But the overall throughput requirement was still met.

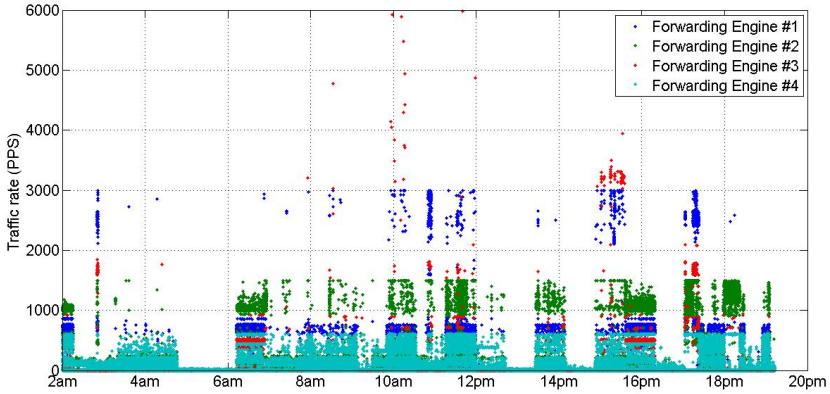

Fig. 8.14 Load distribution on 4 sleep-enabled forwarding engines with power-aware parallel forwarding.

8.4.3.2 With Sleep-Enabled Engines

When forwarding engines have the option to sleep, our power-aware parallel forwarding scheme can achieve significant power/ energy reduction. As shown in Figure 8.13, our power-aware scheme achieved much lower power consumption than the traditional power-unaware scheme. The fundamental reason is that the traditional parallel forwarding scheme does not exploit the sleep mode of forwarding engines to reduce the static power consumption. Our power-aware parallel forwarding scheme turned on only portion of forwarding engines based on the optimization results. As a result, our solution achieved **9.13**-fold reduction in energy (and average power) consumption than the traditional power-unaware scheme.

Figure 8.14 shows the load distribution on the 4 forwarding engines. In most situations, the forwarding engines with lower power consumption tended to receive higher volume of traffic than those with higher power consumption, which is similar as Figure 8.12. When the traffic rate was low, the engine with high power consumption tended to sleep. But when the traffic rate was high, there were some situations where it was better to turn on one engine with high dynamic power consumption than multiple engines with low dynamic power consumption. Thus we observed different load distribution between Figure 8.14 and Figure 8.12. Again, though the short-term load distribution among the forwarding engines was not balanced, the overall throughput requirement was still met.

8.5 Summary

Energy/power consumption has become a major concern in the design of next generation network infrastructure. Reducing the energy/power consumption of packet forwarding systems has been a topic of significant interest [10,7,26]. Most of the existing work focuses on either single forwarding engines or the system- and network-level optimizations. This chapter represents one of the first study on energy/power-aware parallel forwarding in routers/ switches.

First, we considered the system where the forwarding table is partitioned and mapped onto multiple engines. We identified the reason for the high power consumption of SRAM-based pipelined packet forwarding engines to be the unbounded number of accesses on large size memories. A two-phase scheme was proposed to partition a trie into many height-bounded subtries which were then mapped onto multiple pipelines. The partitioning and mapping scheme was carefully designed to balance the memory requirement among the pipelines. The scheme was enabled by combining small TCAM into the index table that worked as the dispatcher. Our experiments using real-life backbone routing tables showed that the proposed 4-pipeline architecture achieved up to 7-fold and 3-fold reductions in energy consumption over the state-of-the-art TCAM-based and SRAM-based solutions, respectively, while sustaining a throughput of 400 Gbps (i.e. $10\times$ OC-768 rate) for minimum size (40 bytes) packets.

Second, we considered the system where each engine contains a copy of the full forwarding table. Given that the power dissipation of each engine can be modeled as

a function of traffic load going through that engine, we formulated the optimization problem that minimizes the overall power consumption while satisfying the throughput demand. We discussed two types of power functions, in terms of whether they support sleep mode or not. We solved the two problems via linear programming (LP) and mixed integer programming (MIP), respectively. Our simulation using a 18-hour real-life traffic trace showed that our solution achieved up to **9.13**-fold reduction in energy (and average power) consumption compared with the traditional parallel forwarding scheme based on load balancing. We also discussed the system design issues and identified the challenges for real implementation.

Many other issues remain open. For example, we did not consider the power consumption of the queues / buffers in this chapter. Some applications require the parallel forwarding system preserve intra-flow packet order, which makes the problem more difficult. We hope our initial work can motivate more follow-up research in this area. We also believe the ideas proposed in this chapter can be applied to other parallel computing systems with high power density, such as data centers, server farms, or clusters.

References

1. Agrawal, B., Sherwood, T.: Ternary CAM power and delay model: Extensions and uses. IEEE Trans. VLSI Syst. 16(5), 554–564 (2008), doi:10.1109/TVLSI.2008.917538
2. Baboescu, F., Tullsen, D.M., Rosu, G., Singh, S.: A tree based router search engine architecture with single port memories. In: Proc. ISCA, pp. 123–133 (2005)
3. Basu, A., Narlikar, G.: Fast incremental updates for pipelined forwarding engines. IEEE/ACM Trans. Netw. 13(3), 690–703 (2005)
4. Bux, W., Denzel, W.E., Engbersen, T., Herkersdorf, A., Luijten, R.P.: Technologies and building blocks for fast packet forwarding. IEEE Communications 39(1), 70–77 (2001)
5. CACTI 5.3: `http://quid.hpl.hp.com:9081/cacti/`
6. Carli, L.D., Pan, Y., Kumar, A., Estan, C., Sankaralingam, K.: Flexible lookup modules for rapid deployment of new protocols in high-speed routers. In: Proc. SIGCOMM (2009)
7. Chabarek, J., Sommers, J., Barford, P., Estan, C., Tsiang, D., Wright, S.: Power awareness in network design and routing. In: Proc. INFOCOM, pp. 457–465 (2008)
8. Chen, B., Morris, R.: Flexible control of parallelism in a multiprocessor pc router. In: Proc. of the General Track: 2001 USENIX Annual Technical Conference, pp. 333–346 (2001)
9. Chen, G., He, W., Liu, J., Nath, S., Rigas, L., Xiao, L., Zhao, F.: Energy-aware server provisioning and load dispatching for connection-intensive internet services. In: Proc. NSDI, pp. 337–350 (2008)
10. Gupta, M., Singh, S.: Greening of the Internet. In: Proc. SIGCOMM, pp. 19–26 (2003)
11. Jiang, W., Prasanna, V.K.: Towards green routers: Depth-bounded multi-pipeline architecture for power-efficient IP lookup. In: Proc. IPCCC, pp. 185–192 (2008)
12. Jiang, W., Prasanna, V.K.: Reducing dynamic power dissipation in pipelined forwarding engines. In: Proc. ICCD (2009)
13. Jiang, W., Prasanna, V.K.: Architecture-aware data structure optimization for power-efficient ip lookup. In: Proc. HPSR (2010)

14. Jiang, W., Wang, Q., Prasanna, V.K.: Beyond TCAMs: An SRAM-based parallel multi-pipeline architecture for terabit IP lookup. In: Proc. INFOCOM, pp. 1786–1794 (2008)
15. Kaxiras, S., Keramidas, G.: IPStash: a set-associative memory approach for efficient IP-lookup. In: INFOCOM, pp. 992–1001 (2005)
16. Kennedy, A., Wang, X., Liu, Z., Liu, B.: Low power architecture for high speed packet classification. In: Proc. ANCS, pp. 131–140 (2008)
17. Kleinberg, J., Tardos, E.: Algorithm Design. Addison-Wesley Longman Publishing Co., Inc, Amsterdam (2005)
18. Kokku, R., Shevade, U.B., Shah, N.S., Dahlin, M., Vin, H.M.: Energy-Efficient Packet Processing (2004), http://www.cs.utexas.edu/users/rkoku/RESEARCH/energy-tech.pdf
19. Kumar, S., Becchi, M., Crowley, P., Turner, J.: CAMP: fast and efficient IP lookup architecture. In: Proc. ANCS, pp. 51–60 (2006)
20. LBNL/ICSI Enterprise Tracing Project: http://www.icir.org/enterprise-tracing/download.html
21. Le, K., Bianchini, R., Martonosi, M., Nguyen, T.D.: Cost- and Energy-Aware Load Distribution Across Data Centers. In: Proc. HotPower (2009)
22. Luo, Y., Yu, J., Yang, J., Bhuyan, L.N.: Conserving network processor power consumption by exploiting traffic variability. ACM Trans. Archit. Code Optim. 4(1), 4 (2007)
23. Lyons, A.M., Neilson, D.T., Salamon, T.R.: Energy efficient strategies for high density telecom applications. Princeton University, Supelec, Ecole Centrale Paris and Alcatel-Lucent Bell Labs Workshop on Information, Energy and Environment (2008)
24. Mahadevan, P., Sharma, P., Banerjee, S., Ranganathan, P.: A power benchmarking framework for network devices. In: Proc. Networking, pp. 795–808 (2009)
25. Mandviwalla, M., Tzeng, N.F.: Energy-efficient scheme for multiprocessor-based router linecards. In: Proc. SAINT (2006)
26. Nedevschi, S., Popa, L., Iannaccone, G., Ratnasamy, S., Wetherall, D.: Reducing network energy consumption via sleeping and rate-adaptation. In: Proc. NSDI, pp. 323–336 (2008)
27. RIS Raw Data: http://data.ris.ripe.net
28. Ruiz-Sanchez, M.A., Biersack, E.W., Dabbous, W.: Survey and taxonomy of IP address lookup algorithms. IEEE Network 15(2), 8–23 (2001)
29. Shi, W., MacGregor, M.H., Gburzynski, P.: Load balancing for parallel forwarding. IEEE/ACM Trans. Netw. 13(4), 790–801 (2005)
30. Taylor, D.E.: Survey and taxonomy of packet classification techniques. ACM Comput. Surv. 37(3), 238–275 (2005)
31. Valancius, V., Laoutaris, N., Massoulié, L., Diot, C., Rodriguez, P.: Greening the internet with nano data centers. In: Proc. CoNEXT, pp. 37–48 (2009)
32. Zane, F., Narlikar, G.J., Basu, A.: CoolCAMs: Power-efficient TCAMs for forwarding engines. In: Proc. INFOCOM (2003)