Efficient Traffic Aware Power Management for Multicore Communications Processors

Muhammad Faisal Iqbal
University of Texas at Austin
faisaliqbal@utexas.edu

Lizy K. John
University of Texas at Austin
ljohn@ece.utexas.edu

ABSTRACT

Multicore communications processors have become the main computing element in Internet routers and mobile base stations due to their flexibility and high processing capability. These processors are designed and equipped with enough resources to handle peak traffic loads. But network traffic varies significantly over time and peak traffic is observed very rarely. This variation in amount of traffic gives us an opportunity to save power during the low traffic times. Existing power management schemes are either too conservative or are unaware of traffic demands. We present a predictive power management scheme for communications or network processors. We use a traffic and load predictor to pro-actively change the number of active cores. Predictive power management provides more power efficiency than reactive schemes because it reduces the lag between load changes and changes in power adaptations since adaptations can be applied before the load changes. The proposed scheme also uses Dynamic Voltage and Frequency Scaling (DVFS) to change the frequency of the active cores to adapt to variation in traffic during the prediction interval. We perform experiments on real network traces and show that the proposed traffic aware scheme can save up to 40% more power in communications processors as compared to traditional power management schemes.

Categories and Subject Descriptors

C.2.6 [**Internetworking**]: Routers; C.1.4 [**Processor Architectures**]: Parallel Architectures

General Terms

Performance, Design

Keywords

Power Management, Network Processors, P-States, C-States

1. INTRODUCTION

The Internet infrastructure contributes to about 2% of world's energy consumption [13, 49, 32]. This contribution is likely to increase in future with exponential growth in number of users and high bandwidth services. According to different studies, routers are major contributors of power in Internet infrastructure [16, 32]. The energy consumption in routers is reaching the limits of air cooling [16, 14]. For example, a fully configured Cisco CRS-1 router can consume up to one megawatt of power [14]. A typical router has a set of line cards and each line card has one or more network processors [8, 2, 42]. Multicore network processors have become the major computing element in routers due to their flexibility and high processing capability. FreeScale's P4080 [5], Intel IXP [12] and Tilera processors [10] are some examples of multicore processors being used in networking applications. Power consumption of a single line card can reach up to 500 Watts [2, 6]. Modern routers can have hundreds of line cards. For example, a CISCO CRS-1 router can house up to 1152 line cards in different chassis. These line cards are densely packed in routers. High power consumption can result in high temperature of parts and failure due to thermal stress. Such failures affect the reliability and availability of networks. This results in lower quality of service and increased expenditures in replacement parts. High power consumption of equipment leads to higher cooling costs and results in increased operational expenditure of the network. According to Erricson's vice president, "The cost of electricity over lifetime of network equipment is more than the cost of network equipment itself" [23]. With increasing traffic rate demands and computational complexity, the number and complexity of cores in network processors are on the rise resulting in more and more power consumption. Tight power budgets and dense integration requirements call for design of power efficient network processors.

The multicore packet processing systems are usually designed and provisioned with enough resources to satisfy peak traffic load. But network traffic varies with time and reaches the peak value for only a small portion of time. Figure 1 shows traffic observed over two days by CAIDA monitor [20] at Internet backbone in Chicago. There is a huge variation in packet rates and thus different processing requirements at different times of the day. Most of the time the traffic rate is below the maximum traffic and we do not need to run the processors at full capabilities. The low activity periods can be exploited to save power in network processors by running them in low power modes and/or by turning off some processing cores.

We propose a predictive power management scheme whereas previous schemes proposed for Network Processors [42, 41] are reactive in nature. Predictive power management provides more power efficiency than reactive schemes because it reduces the lag between load changes and changes in power adaptations since adaptations can be applied before the load changes. Power management policies used in general purpose processors are unaware of traffic

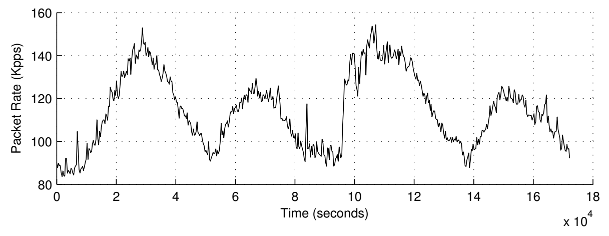


Figure 1: Variation in traffic arrival rates(Kilo Packets Per Sec) over 2 days at equinix-chicago Internet backbone

demands and provide sub-optimal results for network processors. In this paper we make following contributions:

- We propose a predictive power management scheme for communications processors which uses a low cost traffic and load predictor.
- The proposed scheme aims at reducing both active and idle power by utilizing P and C-states of the processor.
- We propose a new parameter called *traffic_factor* which combines traffic prediction and application processing requirements into a single parameter for efficiently predicting required number of active cores.
- We perform experiments on real network traces and show that the proposed scheme can save up to 40% more power as compared to traditional schemes (Section 5).

2. BACKGROUND

Modern processors are equipped with capabilities to save power during active periods (P-states) and during idle periods (C-states). P and C-states are part of an open industrial standard called Advanced Configuration and Power Interface (ACPI) [33]. ACPI was proposed by Intel, Microsoft and Toshiba to facilitate the development of Operating System based power management. Policies to manage P and C-states are usually implemented as Operating System modules. Almost all modern operating systems have such modules or governors. In this section we give short background of P and C-states. We also provide an overview of existing policies for C and P-states and explain why these policies may result in un-optimal power management in case of network processors.

2.1 Active Power Management Using P-states

P-states refer to the different performance states of the processor and provide choices for different power/performance points to adapt to dynamic processing requirements. P-states are an implementation of Dynamic Voltage and Frequency Scaling (DVFS) and are aimed at reducing dynamic power. Recall that dynamic power is given as

$$P_{dynamic} = k \times v^2 \times f \tag{1}$$

where k is a workload and processor dependent parameter determined by switching capacitance and activity of the processor. Dynamic power can be saved if we lower frequency and voltage. P-states define frequency and voltage levels of the processor so that during times of low processing requirements, frequency and voltage are lowered to save power and energy. P-states are named numerically form P_0 to P_N. P_0 is the highest performance state. Performance and power consumption reduces with increasing P-state numbers. Table 1 shows example P-states for a typical processor [18]. These P-states are similar to the P-states of AMD Opteron Processor [1].

P-state	Frequency	Voltage
P0	$F0$	$V0$
P1	$F0 \times 0.85$	V0 x 0.96
P2	$F0 \times 0.75$	V0 x 0.90
P3	$F0 \times 0.65$	V0 x 0.85
P4	$F0 \times 0.50$	V0 x 0.80

Table 1: Example P-states of a typical processor

2.2 Idle Power Management Using C-states

Processor's C-states represent the capability of processor to save power during idle periods. States are named numerically starting from C_0 to C_N, where C_0 represents the active state. As the C-state number increases, the power consumption of the processor decreases and wakeup latency increases. Designers employ different techniques to implement C-states. Low latency techniques include clock and fetch gating whereas high latency techniques include voltage scaling and power gating. Table 2 shows an example of C-states. The table shows only three C-states. Modern processors have a large number of C-states. For example, Intel Core 2 Duo has five C-states [50] and some processors even have up to eight C-states [3]. Wakeup latency of processors increases as we move to deeper sleep states. It only makes sense to enter a C-state if inactive time is equal or greater than break even time T_{BE}[17]. The break even time is composed of two terms: the total transition time (i.e., $T_{tr} = T_{enter} + T_{exit}$) and minimum time that has to be spent in that state to compensate for the additional power during transition. If power consumption during transition is less than or equal to on-state power (this is what we assume in this study) then $T_{BE} = T_{tr}$. This break even time is usually used as a threshold for transitioning into deeper states. For Table 2, the break even time will be 20 μs for C1 and 200 μs for C2. Also note that modern operating systems support a tick-less kernel i.e., idle CPUs do not have to respond to periodic ticks. These CPUs are allowed to remain idle and are woken up by interrupts when a new job arrives for them. We assume such a tick-less kernel in this study.

C-state	Response Latency	Relative Power
C0	0	100%
C1	10 uS	40%
C2	100 uS	5%

Table 2: Example C-states

2.3 Policies for C-state Management

2.3.1 Using Idle Time

Most implementations of C-state management use *Fixed Timeout* policy. For example, Ladder governor in Linux is used to implement C-state management [11, 9]. This governor uses elapsed idle time to predict the total duration of the current idle period. When a processor becomes idle a counter starts. This idle time counter is then compared with pre-defined thresholds. When the counter reaches $C1_{th}$ value, the system is forced into a sleep state C1. The counter continues counting until the processor is woken up by an external event. If the counter reaches $C2_{th}$, the system transitions to C2 and so on. The processor keeps transitioning to deeper C-states until it reaches the lowest power state or it is woken up by an external event. The CPU starts from C1 again when it becomes idle the next time.

As described in Section 2.2, the threshold values are decided based on break even times which are of the order of hundreds of microseconds. This scheme works well to exploit idle time in general purpose applications e.g., time waiting for user input or response from I/O subsystem where the waiting times are very high. But in case of NPs, the inter-packet arrival times viewed by multiple cores are usually smaller than these thresholds even if the number of active cores is more than the required to sustain a certain amount of traffic. In other words, this scheme might be too conservative and wastes a lot of power saving opportunity which could be exploited if we directly have information about traffic and processing demands. Furthermore, this is a reactive scheme and some power saving opportunity is lost in waiting from threshold times to elapse. In contrast, we present a predictive scheme which uses direct information about traffic to more efficiently manage power.

2.3.2 *Using Number of Idle Threads*

The scheme proposed by Luo et al. [42, 43] targets multicore NPs and is the closest related work to our proposal. This scheme monitors the number of idle threads in the thread queue during an interval. If the number of idle threads is more than the required number of cores for majority of the interval, it shuts down the additional threads and cores. Using the number of idle threads works fine if we assume that each processor runs at maximum frequency. But if each core is allowed to change its frequency, the number of idle threads does not effectively represent the load. For example, consider a situation where two processors are active and running at half of the maximum frequency. These processors will be utilized 100% of the time to handle a traffic which a single processor could handle at full speed. But the threads running on slow cores will never enter the thread queue and hence we will never be able to turn off any cores. Hence the number of idle threads does not represent processing requirements in this situation.

Furthermore, this scheme is also a reactive scheme and uses only clock gating. The transition overheads for clock gating are small so the reactive scheme works well. Since this scheme is targeted for 280 nm, leakage power is not a big issue and clock gating works fine. But in modern technologies, the power consumption is usually dominated by leakage power and hence it is important to utilize deep sleep states which have additional power savings even beyond DVFS and clock gating. These deep sleep states have high overheads in terms of transition delays and reactive scheme may result in opportunity loss for power saving. A predictive scheme is needed so that the lag between power adaptation and load changes is minimized.

2.4 Policies for P-State Management

2.4.1 *Using Processor Utilization*

P-state management policies are generally aimed at saving power during performance-insensitive phases of programs. For example, power can be saved during memory-bound phase of a program by reducing clock frequency. Many implementations of these policies use processor utilization to drive P-states. These policies try to maintain processor utilization within a certain range [18, 15, 30, 9]. Processor utilization or activity level represents the ratio of code execution time (active time) to wall clock time (active + idle time) i.e.,

$$utilization = \frac{Time_{active}}{(Time_{active} + Time_{idle})} \qquad (2)$$

Listing 1 shows an implementation of Linux "Ondemand" governor [11]. The algorithm monitors processor utilization for an interval and then makes a decision whether to increase or decrease the frequency.

```
 #define up_threshold 0.90

 for (each sampling interval){

   if (utilization > up_threshold)
      freq = max_freq;

   else
      freq = next_lower_freq;
}
```

Listing 1: Linux ondemand frequency Governor

Ondemand governor is the most aggressive governor in Linux implementations because it tries to settle to lowest frequency in case of zero load and will settle to the highest frequency at peak load. Another relevant governor is a conservative governor [11] which is similar to power management module found in Vista [15]. This governor tries to maintain the processor utilization within a range say 0.3 to 0.6.

The policy which uses CPU utilization does not factor in traffic demands and suffers from several pitfalls. First, CPU utilization is a function of mixture of events (e.g., performance of memory, I/O devices etc.) and does not directly indicate load requirements. Second, if this policy is too conservative, it will lose a lot of opportunities to save power and if it is too aggressive then it may result in performance degradation in terms of traffic loss.

2.4.2 *Minimizing Energy Per Instruction*

Herbert et al. proposed a greedy search method to minimize Energy Per Instruction (EPI) [30, 31] for CMPs. This method is an extension of the technique proposed by Magklis et al. [44]. The P-state controller attempts to operate at frequency level which minimizes EPI assuming EPI is a bath-tub shaped function of voltage and frequency levels. This algorithm assumes the availability of current sensors which help to approximate EPI. After each interval, the controller compares current EPI with the EPI of previous interval. If EPI is improved, the controller makes a move in the same direction as last one. If the EPI has increased, it is assumed that controller has overshot the optimal frequency level. It makes a transition in opposite direction as the last one and stay there for N = 5 intervals. After the holding period the controller continues exploration in a direction opposite to one which preceded the hold. This type of scheme is un-aware of traffic demands and may result in running at lower frequency than needed and may result in extra packet loss.

3. PREDICTIVE TRAFFIC AWARE POWER MANAGEMENT (PTM)

An efficient power management scheme for NP has to make the following decisions:

1. Predicting load for the next interval

2. Deciding required number of active cores N_{opt} for the predicted load

3. Deciding frequency f_i for each active core i.

In this section we provide details of our proposed PTM scheme and explain how PTM makes the three above mentioned decisions.

3.1 Prediction of Load

The computational requirements for NPs depend on both traffic arrival rate and the computation complexity of the applications. We propose a new parameter called *Traffic_factor* which combines both traffic rate and computational complexity to give a true estimation of load to be handled by the NP.

3.1.1 *Traffic Prediction*

PTM uses Double Exponential Smoothing Predictor (DES) for traffic prediction. A more complex predictor may result in greater accuracy in some situations but a low overhead predictor is desirable for energy efficiency. Recently, Iqbal et al. [34] studied many real network traces and have shown that Double Exponential Smoothing (DES) predictor is a low overhead predictor with accuracy comparable to complex predictors like Artificial Neural Network (ANN) or Wavelet transform based predictors. This makes DES very suitable for our application. This study uses DES predictor but designers can use the proposed scheme with any other predictor based on their requirements. We will give a brief introduction to DES predictor in this section and a short comparison of different predictors is presented in Section 5.2.

3.1.2 *Double Exponential Smoothing (DES) Predictor*

Exponential Smoothing assigns exponentially lower weights to older observations. Single exponential smoothing does not work well when there is a trend in data [4]. Trend means that the average value of the time series increases or decreases with time. However, Double Exponential Smoothing adds trend component for estimation and is considered more appropriate for data with trends. The equation for DES based prediction for a time series $X(t)$ is given as

$$X_{t+1} = S_t + b_t \tag{3}$$

where

$$S_t = \alpha X_t + (1-\alpha)(S_{t-1} + b_{t-1}) \tag{4}$$

and

$$b_t = \gamma(S_t - S_{t-1}) + (1-\gamma)b_{t-1} \tag{5}$$

S_t and b_t are smoothed value of stationary process and trend value respectively. S_t and b_t are added together to get the prediction for next interval. α defines the speed at which older values of S_t are damped. When α is close to 1, dampening is quick and when α is close to 0, dampening is slow. γ is similar smoothing constant for b_t. The values of α and γ are obtained using non-linear optimization techniques and are learned during the training phase of the predictor. Note that this is a very low cost predictor. It requires only four registers for storing α, γ, S_{t-1} and b_{t-1}. To make a prediction it requires six multiplications and four addition operations. This low overhead and reasonable accuracy makes it an appropriate predictor for the purpose of power management.

3.1.3 *Traffic Factor*

We propose a new parameter called *Traffic_Factor* which combines traffic rate and application's processing capability to give a true estimation of processing requirement. Traffic rate is the rate at which packets arrive at the input and is represented as Packets Per Second (PPS). We are naturally tempted to use this parameter directly to exploit traffic variability. But different applications have different processing requirements and hence can support different packet rates i.e., a complex application will require more resources to sustain a particular traffic rate when compared to a simple application. This means packet rate directly cannot be used for power management purposes. But if we can incorporate applications processing requirements with input packet rate, we can use this information to drive the power management scheme. If we know Cycles Per Instruction (CPI) and Instructions Per Packet (IPP), we can directly find Cycles Per Packet (CPP) i.e., $CPP = IPP \times CPI$. We define *Traffic_Factor* as

$$Traffic\_Factor = \frac{PPS_{predicted} \times CPP}{(max\_cpu\_MHZ \times num\_cores)} \tag{6}$$

where $PPS_{predicted}$ is the traffic predicted using DES predictor explained above. *Traffic Factor* incorporates both application performance requirements and traffic rate into a single parameter and is an excellent parameter for use in power management schemes. Note that CPP is independent of the frequency level i.e., cycles per packet will remain constant with changing frequency and hence this parameter does not suffer from the same limitations as the processor utilization used by previous schemes. Processor utilization is a direct function of frequency whereas CPP does not depend upon frequency if we assume that there are limited off-chip memory accesses. This assumption is valid in network processors since the packet processing applications are small and fit into caches and the packet queues are also implemented on on-chip memories [12]. Furthermore, we do not need any additional resources to measure this parameter. Many network processors like P4080 provide performance counters to measure PPS, IPP and CPI directly [7]. Table 3 lists the performance counters which can be used for measuring these parameters.

Metric	Performance Counters	Formula
Core Cycles	CE:Ref:1	CE:Ref:1
Ins. Count	CE:Ref:2	CE:Ref:2
Pkt. Count	SE:Ref:36	SE:Ref:36
CPI	CE:Ref:1 CE:Ref:2	CE:Ref:2/CE:Ref:1
IPP	SE:Ref:36 CE:Ref:2	CE:Ref:2/SE:Ref:36
PPS	SE:Ref:36,CE:Ref:1 CE:Ref:1	SE:Ref:36/A A=(CE:Ref:1/Freq)

Table 3: P4080 Counters to measure CPI, IPP and PPS

3.2 Deciding Number of Active Cores

Required number of active cores *P* can be directly calculated from the Traffic Factor i.e., $P = Traffic\_Factor \times total\_cores$ and number of active cores are adjusted as shown in Listing 2. The sampling interval used is 500 μS.

```
for (every sampling interval){
   p = traffic_factor * total_cores;

   if (p > active_cores )
      wakeupCores(p - active_cores);

   else if (p < active_cores)
      killCores(active_cores - p);
}
```

Listing 2: Algorithm to decide number of active cores

If *P* is less than the current number of active cores, we can shutdown all the extra cores. When a core goes into a sleep state, it first goes into C1, where it stays for 2 sampling intervals and then it goes to state C2. In our scheme, we have used C2 as the deepest sleep state. Note that the difference between ladder governor and our proposed scheme is different input parameters. Instead of using idle time, we are using *traffic_factor* to drive C-state management.

In order to wakeup cores, we look at the input queue length in addition to *traffic_factor* (See Listing 3). And if the size of the queue length reaches a certain threshold, we wakeup one of the sleeping cores.

3.3 Finding P-states of Active Cores

If the load predictor over-predicts, or there is variation in traffic during prediction interval, we can make use of DVFS to further save power at smaller timescales. At regular intervals (50 μS), we check the size of the input queue and based on size of the queue we decide whether to increase or decrease the power levels. When a packet arrives at the input interface, the interface control logic stores the packet in the input queue until it is picked and serviced by an available processor. Figure 3 shows a simplified model of a network processor. If the input queue is nearly empty most of the time, we have enough resources to handle the traffic load and if it is near full, it means we need more processing capability. Length of the input queue gives a direct indication of whether we need more resources or not. The algorithm for finding the appropriate P-state is shown in Listing 3. If the queue is nearly empty, we assume that we have excess processing capability and we go to a lower frequency level. When the queue starts to grow, it means that the current processing capability is lower than what is needed and we increase the frequency level.

```
int pstate[numcores];
for (every sampling interval){

  if (avg < low$_{th}$){
    core = findMin(pstate);
    pstate[core] = pstate[core]+1;
  }

  else if (avg < high$_{th}$){}

  else{
    core = findMax(pstate);
    if(core == -1)
       WakeUpCore(1);
    else
       pstate[core] = pstate[core]-1;
  }
}
```

Listing 3: Algorithm to find P-state values

Note that Listing 3 uses a global governor which makes decisions based on queue size instead of having a separate governor for each core. The array $pstate[numcores]$ holds the p-states of each core. Each core can have five possible P-states similar to Table 1. The function $findMin()$ and $findMax()$ return indices of fastest and slowest cores respectively. In case the queue size is less than the threshold value low_{th}, we lower the frequency of the fastest core and in case it is higher than the $high_{th}$ we increase the frequency of the slowest core. Also note that if frequency cannot be increased further, we increase the number of cores. This allows us to adjust to dynamic variation in traffic during the interval or under prediction and helps us avoid dropping any packets.

3.3.1 *Measuring Queue Length*

We use the average queue length during the sampling interval to make decision about choosing the appropriate P-states. We use a low pass filter to calculate the average queue size as proposed in the RED algorithm [24]. Thus short term increase in traffic which results from bursty traffic or transient network congestion does not affect the average queue length. The filter used is exponential smoothing filter and is given as

$$avg = \alpha \times qlength + (1 - \alpha) \times avg \qquad (7)$$

where qlength is the instantaneous size of the queue and α defines the speed at which older values are dampened. We use an α of 0.025 in this study. Many congestion control algorithms like RED [24], rely on occupancy of the input queue and modern network processors provide hardware support for these congestion control algorithms. P4080 processor provides a dedicated hardware (QMan) for queue management. QMan implements a variant of RED algorithm and keeps track of the input queue length. Thus queue length can be used for power management purposes and it does not require additional resources since it is already monitored for congestion control purposes. Even if it is not readily available in any processor, it is easy to add functionality in the interface logic to keep track of the queue length or simple dedicated hardware to do the purpose.

3.3.2 *Deciding Threshold Values*

The algorithms proposed in Listing 2 and 3 monitor length of the input queue and compare it with the predefined threshold values to decide the state of the processor. We allow the thresholds to change dynamically to adjust to changing traffic load and thus do not need to find a common value for all traffic rates and applications. The optimum value of $high_{th}$ depends on the wake up time for a core in deep sleep state. When qlength reaches this threshold value and all the active processors are running at highest frequency, we need to turn on additional cores in order to avoid dropping packets. Assuming the wakeup time of 200 μS, we need to find extra buffer space which is enough such that no additional packets are dropped before an additional core is on line. In our experiments we found the maximum packet rate to be 200 KPPS. Assuming this is the worst case increase in the packet rate, we need an additional buffer space for 40 packets before the core is on line. If maximum queue size is Q_{max}, we use $Q_{max} - 40$ as the value of $high_{th}$. We have used a Q_{max} value of 80 in our simulations. The value of low_{th} can be chosen from a wide range. Essentially, it should not be too close to $high_{th}$ to avoid $avg\_qlength$ to oscillate between low and high thresholds. We chose low_{th} to be such that $high_{th} = 4 \times low_{th}$. Essentially, P-state manager tries to keep input queue to be less than 50% full all the time. But if the traffic rate is higher than what P-states can manage, the queue length will increase more than $high_{th}$. When queue length exceeds this value we know that current number of processor are not enough to manage the traffic and an additional processor is woken up. If value of queue length reaches 95%, we know that we turned on a new processor too late and C_{th}, low_{th} and $high_{th}$ value are decreased by 10%. If qlength never reaches C_{th} value for 10 consecutive intervals, we increase all the thresholds by 10%.

4. EVALUATION METHODOLOGY

4.1 Simulation Infrastructure

Our simulation infrastructure is based on SCE simulator from UC Irvine [22] and follows host compiled simulation approach proposed in [27, 28]. This approach pairs a high level functional model with back annotation of statitically determined timing and power estimates in order to achieve fast and accurate simulation. At the highest level, the user application consists of a set of back annotated tasks that are controlled and interact with underlying OS model. Figure 2 shows the back annotation flow which is similar to the flow proposed in [28]. The packet processing applications are compiled

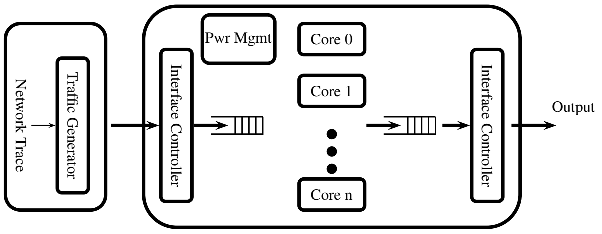


Figure 3: Simulation Model of Network Processor

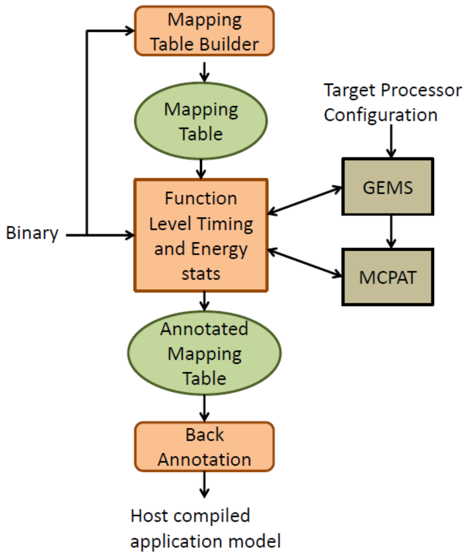

Mapping Table Builder
Mapping Table
Target Processor Configuration
GEMS
Binary
Function Level Timing and Energy stats
MCPAT
Annotated Mapping Table
Back Annotation
Host compiled application model

Figure 2: Back Annotation Flow[28]

for target platform and are simulated in GEMS full system simulators [45] to gather detailed performance statistics. GEMS simulator provides hooks to find statistics for particular piece of code using special unused opcodes (called magic instructions in GEMS terminology) from the target ISA. We add these magic instructions at the function boundaries in the source code of the application to get per function statistics. Performance statistics from GEMS are fed into MCPAT [40] Power simulator to get power statistics. Power is multiplied by time to get energy for each iteration of the function execution. We use average delay and energy consumption of all iterations of the function for back annotation. MCPAT also allows to get power numbers for different P-states. The final back annotated application has power and delay numbers for each function for each possible P-states. This back annotated application is then simulated on top of an abstract OS model. Figure 4 shows the high level picture of simulation model. The OS model is responsible for controlling the power states of the processor which we study in this work. A Hardware Abstraction Layer (HAL) consists of necessary I/O drivers and implements an interrupt handling mechanism. A HAL combined with an abstract Transaction Level Modeling (TLM) layer provides a high level processor model that interfaces with the TLM backplane. The complete simulation model is based on System Level Design Language (SLDL) simulation kernel. This kernel provides basic concurrency and event handling on the host machine. The SLDL kernel used in this study is based on SpecC

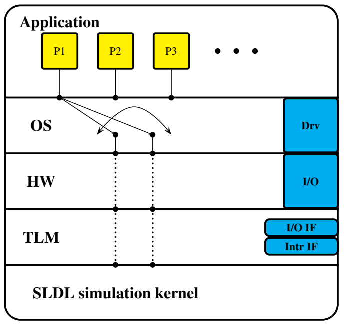


Figure 4: Host compiled multicore platform model [27]

[26] which is similar in design and philosophy to SystemC [29]. Each core in GEMS simulator is modeled after the processing core in P4080 network processor and we model a network processor with 32 processing cores in this study. The detailed configuration of the processing cores is listed in Table 4. For power modeling in MCPAT we use 45nm technology.

L1 Cache	*L2 Cache*	*DRAM*	*Pipeline*	*Branch Prediction*
16KB 2way	64KB 4 way	4 GB	10 stages 2-wide	YAGS/ 64entry BTB

Table 4: Configuration of the processing cores

Figure 3 shows the block diagram of our SpecC based simulation model. The traffic generator module replays the actual trace with accurate timing. Each core is an instance of back annotated application. Power Management module monitors different statistics and control the C and P-states of the cores. The power and delay numbers are looked up from the mapping table of each core based on its P and C-states.

4.2 Network Traces and Benchmarks

We chose seven packet processing applications from NPBench [39] and PacketBench [48]. Table 6 lists the application used for evaluation in this study. We use real network traces from CAIDA's *equinix-chicago* and *equinix-sanjose* monitors [20] as inputs to these benchmarks. We performed experiments with large set of traces but the results of two representative traces (one from each) are presented in this study. All other traces show similar behavior. CAIDA monitors capture the traffic traces on OC192 links. The traces in this data set are of one hour long duration and are captured in year 2011. The network traces are replayed with actual arrival times and packet sizes in the SpecC to model the real traffic scenario. We also present some results with synthetic traces to show the effectiveness of scheme at different traffic rates.

Network Trace	Description
equinix-chicago	One hour long traffic trace at Internet backbone in Chicago
equinix-sanjose	One hour long traffic trace at Internet backbone in Sanjose
Synthetic	One minute long traffic synthetically generated traffic traces at different rates

Table 5: Network Traces used in this study

Application	Description
FRAG	Packet fragmentation Algorithm
IPV4T	IPV4 routing based on trie
IPV4R	IPV4 routing based on radix tree structure
IPSEC	IP Security protocol
MPLS	Multi Protocol Layer Switching is forwarding technology based on short labels
SSLD	Secure Socket Layer dispatcher is an example of content based switching
WFQ	Queue scheduling algorithm to serve packets in order of their finish time

Table 6: Benchmark Applications used in this study

5. RESULTS

We implemented different power management policies for comparison purposes. Table 7 lists the policies under consideration. The C-state policy in Base scheme is similar to Linux ladder governor. The thresholds used are based on Table 2. The P-state management policy is based on Linux ondemand governor explained in Listing 1. Greedy scheme uses similar ladder governor for C-state management but the P-state manager is a greedy algorithm of Section 2.4.2 which tries to minimize EPI. The IdleT policy uses a scheme similar to one proposed by Luo et al. [42, 43] for managing number of active cores based on number of idle threads in the given interval. For fair comparison with PTM, we modified this scheme to go into deeper sleep state if it remain in the current state for two consecutive intervals. Note that the original scheme just made use of clock gating and did not utilize deeper sleep states. We further augmented this scheme to use ondemand governor for frequency management of individual cores. PTM is the proposed scheme based on traffic prediction.

Policy	C-state Policy	P-state Policy
Base	ladder	ondemand
Greedy	ladder	greedy EPI
IdleT	idle threads	ondemand
PTM	Traffic Factor based	Queue Length based

Table 7: Power Management Policies Implemented for Comparison

Figure 5(a) and Figure 5(b) show power savings for the equinix-sanjose and equinix-chicago traces respectively. We use Base policy as the baseline and results are presented relative to Base. Table 8 shows the absolute numbers for power consumption during the trace. We can see that the proposed PTM consistently beats the other policies on all applications. Major portion of the energy saving comes from having right number of active cores. Table 9 shows average number of active cores throughout the trace for different strategies. PTM scheme has the lowest average number of active cores on all benchmarks. *Traffic Factor* allows PTM to estimate minimum number of active cores as compared to other schemes which base their decisions on idle time. PTM changes number of cores pro actively and thus reduces the lag between load changes and power adaptation. Other schemes, being reactive in nature lose some power saving opportunities since the cores are turned off after some idle time has been elapsed. Base and Greedy schemes result in highest number of active cores. C-state management in Base and Greedy is too conservative in the sense that it waits for break even time to elapse before going to deep sleep state. Also note that number of active cores depends on the p-state management as well. For example, if the active cores are running at lower speed than needed, they will result in activating more cores than required. PTM calculates required number of active cores directly using traffic factor and results in minimum number of cores being active. For IdleT also, the number of active cores is a function of frequency of individual cores and results in higher number of cores than needed. In some situations, Greedy results in more number of cores than any other policy. The reason is that P-state policy in this scheme is unaware of traffic at all. It tries to minimize energy per instruction irrespective of the traffic rate. Thus it results in more cores since individual cores are running at lower frequency. Consider the situation presented in Figure 6. The input packet rate is such that it is required that a single core operates at power level P1 to sustain that traffic.

	equinix-sanjose				sanjose-chicago			
	Base	Greedy	IdleT	PTM	Base	Greedy	IdleT	PTM
FRAG	7.82	7.29	5.93	3.53	7.4	7.32	6.6	3.8
IPV4T	6.73	7.79	6.11	4.10	6.73	6.91	6.23	4.5
IPV4R	29.80	27.80	28.1	25.23	31.5	29.1	28.3	26.1
IPSEC	63.20	60.69	59.80	56.59	71.5	69.3	68.7	62.4
MPLS	45.03	46.6	45.2	40.8	55.5	52.2	51.1	46.3
SSLD	101.6	95.4	96.1	90.1	99.4	94.1	96.1	90.1
WFQ	15.94	16.73	16.02	12.1	15.8	16.3	14.7	11.1

Table 8: Power Consumption in Watts for different schemes

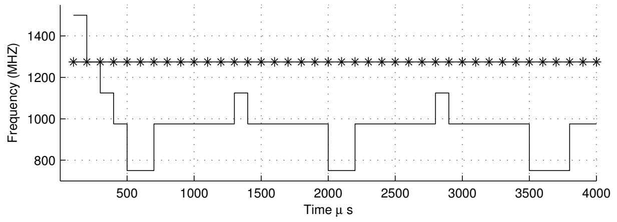
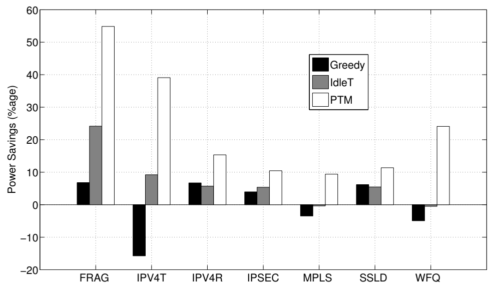


Figure 6: Greedy Algorithm to minimize EPI. Asterisks show optimum power level

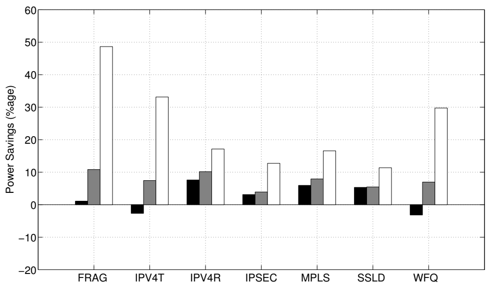
Figure 6 shows the response of greedy algorithm described in Section 2.4.2 to this input traffic. Since this greedy algorithm is unaware of the traffic requirement, it continues to move in the direction of lowering EPI and overshoots the required power level and operates at frequency lower than the required frequency for rest of the trace. This scheme will results in running 2 cores instead of 1 for the above situation and will result in more power consumption.

Application	Base	Greedy	IdleT	PTM
Frag	3.10	2.81	2.21	1.30
IPV4T	3.34	3.69	3.10	1.42
IPV4R	7.70	7.90	7.62	5.50
IPSEC	29.10	30.10	29.00	24.30
MPLS	13.81	15.72	13.81	10.31
SSLD	30.53	31.21	31.11	26.11

Table 9: Average Number of active cores



(a) equinix-sanjose



(b) equinix-chicago

Figure 5: Effectiveness of proposed methodology on two real traces.

5.1 Effectiveness of DVFS

Although most of the benefit comes from having right number of active cores, the ability of individual cores to change frequency also provide some power benefits. Figure 7 presents a comparison of proposed PTM scheme with and without the capability of DVFS. In the scheme without DVFS, the number of cores are controlled by the traffic aware scheme and there is no DVFS i.e., all the active cores run at full speed. Although most of the benefit comes from having right number of active cores but DVFS still has significant impact on performance. From the figure we see that in most cases DVFS improves the power consumption by 15% and as much as by 39% in case of FRAG. This benefit comes from the fact that if we over predicted the workload or there is variation in traffic during the prediction interval, then DVFS helps us at reducing power by lowering the frequencies of the cores. Figure 8 shows potential benefit of having the ability to change frequency and voltage of the cores in addition to adjusting the number of active cores. The plot is for a 16 core system running IP4R benchmark when traffic is varied from 0 to 100% which the given configuration can handle. The power consumption is plotted relative to the system which has ability to change number of active cores but does not have a per core DVFS. We see that there is a lot of potential power saving at low to medium traffic. At high traffic, obviously there is less room for power savings. The dotted line shows potential power savings if we have global DVFS, i.e., all cores change their frequency in unison and at any moment all cores are at the same frequency. Global DVFS provides a good tradeoff between design complexity and power savings since it decreases the design complexity and is able to exploit most of the power saving opportunities. Our P-state algorithm becomes even simpler if global DVFS is used instead of per core DVFS i.e., instead of using findMin() and findMax() function we can increase or decrease the frequencies of all cores together based on queue length.

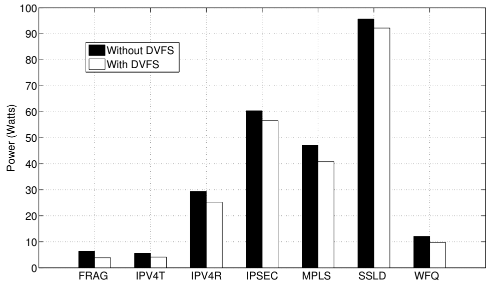


Figure 7: Comparison of proposed scheme with and without DVFS

5.2 Effectiveness of DES Prediction

A distinguishing feature of PTM is that the traffic_factor is based on DES predictor for traffic (Section 3.1). Figure 9 compares the accuracy of different predictors. The predictors under comparison are listed in Table 10.

Predictor	Description
LV	Last observed value is used as prediction for next interval
MA	Moving average of last 8 observations
AR	Auto Regression based prediction
ARMA	AutoRegressive Moving Average of order
ANN	Artificial Neural Network. (3 layers)
DES	Double Exponential Smoothing

Table 10: Predictors used for comparison

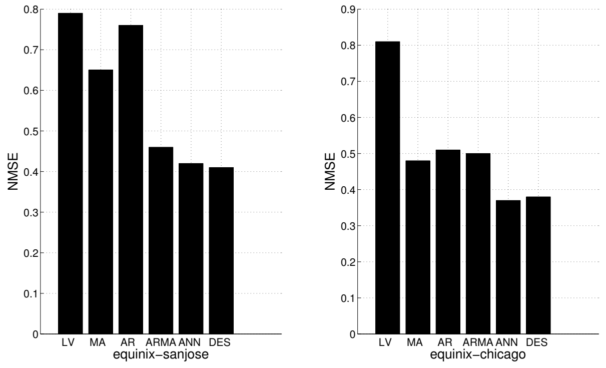


Figure 9: Accuracy Comparison of Predictors. Graphs show NMSE values (lower bar is better).

We use Normalized Mean Square Error ($NMSE$) to compare the performance of predictors. This metric is widely used for evaluating prediction performance. It is the ratio of mean square error to the variance of the series.

$$NMSE = \frac{1}{\sigma^2}\frac{1}{M}\sum_{t=1}^{M}(X_t - \hat{X}_t)^2 \tag{8}$$

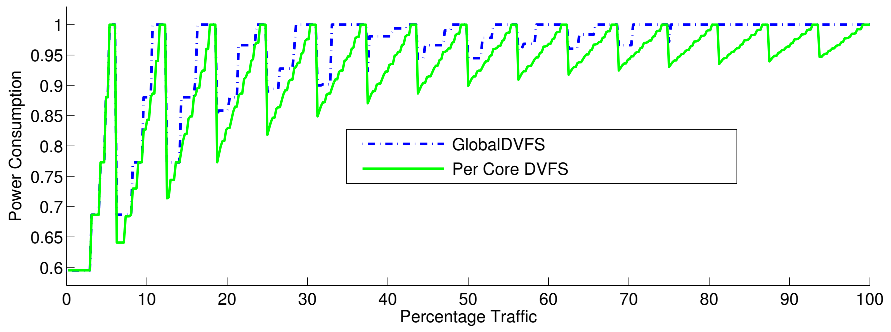


Figure 8: Benefit of having DVFS

where X_t is the actual value of traffic during interval t, $\hat{X}_t$ is the predicted value of X_t and M is the total number of predictions. σ^2 is the variance of time series during prediction. This metric compares the performance of the predictor with a trivial predictor (one which always predicts mean of the time series). In case of this trivial predictor (mean predictor) $NMSE = 1$. If $NMSE > 1$, this means that the predictor is worse than the trivial. $NMSE = 0$ in case of a perfect predictor.

From Figure 9, we can see that DES performs comparably well as compared to more complex ANN based predictor and it outperforms other predictors in the study by big margin. More details about different traffic prediction techniques can be found in [34]. Figure 10 shows power savings with DES predictor over LV predictor. The figure shows that a good predictor can result in more efficient management of power.

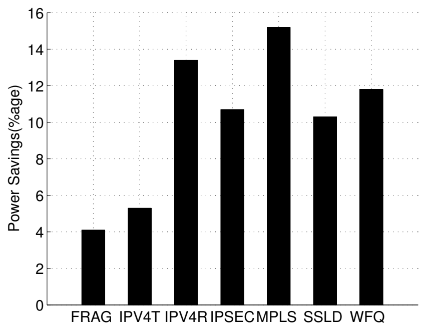


Figure 10: Power Saving when using DES Predictor compared to LV predictor. The trace used is equinix-snjose.

5.3 Packet Queue Behavior

Figure 11 shows the queue length values during a portion of the equinix-sanjose trace. We see that the filtered queue length effectively neglects the short term variation in traffic and is effective in preventing the system from oscillating between states. Also, the scheme is effective to adapt with increasing traffic i.e., if the queue length increases above the threshold values, the system is able to adapt its resources and brings back the queue length within desired limits.

5.4 Power Saving at Different Traffic Rates

Figure 12 shows comparison of different power management schemes at different traffic rates with synthetic traces. The proposed scheme adapts well at different traffic rates and is the best performer for all applications at all rates. It is important to note that PTM does not gain much benefit from prediction in these traces. These traces are of constant data rate and reactive schemes, which behave similar to LV predictor, are also able to accurately predict traffic. For expensive applications like SSLD and IPSEC, there is not much room for power saving since load is already high but PTM is able to get benefit around 6-10% even for those applications. In general greedy scheme runs individual cores at lower power levels but results in activating more number of cores. For IPSEC and MPLS Greedy scheme seems to perform similar to PTM at some data rates. Although power numbers are similar but Greedy scheme results in 11% packet loss while PTM scheme does not drop any packets.

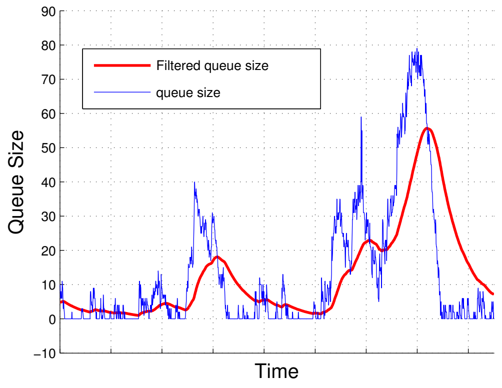


Figure 11: Queue Size and Filtered Queue Size

6. RELATED WORK

Most prior work on power management [18, 31, 44, 35] are not in communication processors domain, but we discuss the network related work in this section.

Need for power efficient Internet infrastructure has fueled studies on design of power efficient network processors. A modeling framework for network processors was presented by Crowley et al. [21]. Franklin et al. also developed an analytical model to explore design space for power efficient network processors [25]. Memick et al. proposed techniques of data filtering to reduce bus accesses [46]. Reduction in bus activity can help save significant power. Zane et al. [52] and Kaxiras et al. [36] propose power efficient TCAM structures to be used in packet processors. Wu et al. [51] investigate a runtime management system for NPs to exploit

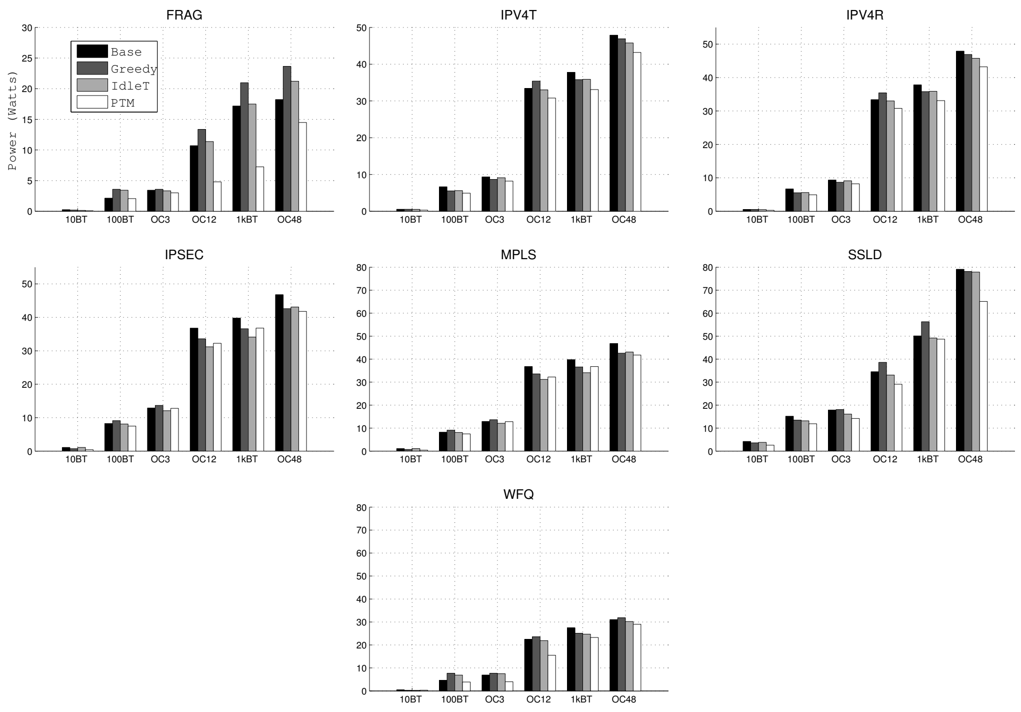


Figure 12: Power consumption comparison of different power management techniques at different Traffic Rates

variability in network traffic for efficient allocation of processing tasks to processing cores. The system is able to adapt to varying demands of traffic and balance utilization of all resources to maximize throughput. Kuang et al. [38] use DVFS for scheduling of pipelined networking applications. This scheme allocates frequencies to the pipeline stages statically and is not aimed at exploiting dynamic traffic variations. Kokku et al. [37] show that variability in traffic can be used for run time scheduling of tasks for conserving energy. Our proposed scheme is not related to scheduling and can complement the power efficient scheduling schemes. Another set of studies which can be used to complement our proposed scheme targets various parts of Internet infrastructure for power efficiency. Chiaraviglio et al. [19] present a scheme to turn off links and nodes during periods of low activity while still guaranteeing full connectivity. Nedevschi et al. [47] show that in addition to putting elements to sleep, link rate adaptation with varying traffic can help further to save power. Luo et al. [41] used processor idle time for DVFS in network processors. All cores are active all the time in this scheme. In another study, Luo et al. [42, 43] applied clock gating to change number of active cores in network processors. This scheme is explained in Section 2.3.2. In contrast to both the above scheme, we propose a predictive scheme to use traffic information directly to manage power in Network Processors and our proposed methodology makes use of both changing number of active cores and DVFS to save power and is simple and general enough to apply to any network processor.

7. CONCLUSIONS

We have proposed and evaluated a predictive power management scheme for Network Processors to exploit variation in traffic. This scheme is able to recognize optimum number of active cores and frequency level of each active core to sustain the input traffic. Traditional power management schemes either result in high power consumption or result in packet loss. Our proposed methodology does not have this weakness as it uses traffic information directly for power management purposes. We have shown that predictive power management schemes work well for packet processing. A simple predictor like DES is able to capture the trends in traffic behavior. Our scheme adapts to variation within the prediction interval using DVFS by monitoring the queue length. DVFS provides us the capability to adaptation at a finer time scale. Furthermore by using filtered queue length we minimize the impact of short term variation in traffic. Experiments on real network traces show that the proposed scheme can save up to 40% more power than traditional schemes.

8. ACKNOWLEDGEMENTS

We would like to thank Heather Hanson (IBM) for her many helpful comments. We would also like to thank anonymous reviewers for their useful suggestions to improve the text. This work was sponsored in part by Semiconductor Research Consortium (SRC) under Task ID 2155.001. The opinions and views expressed in this paper are those of the authors and not those of SRC.

9. REFERENCES

[1] AMD Opteron Processor Power and Thermal Datasheet. http://support.amd.com/us/Processor_TechDocs/30417.pdf.
[2] Cisco 7609-s router. http://www.cisco.com.
[3] Energy Efficient Platforms - Considerations for Applications Software and Services. http://download.intel.com/technology/pdf/Green_Hill_Software.pdf.

[4] NIST/SEMATECH e-handbook of statistical methods.

[5] The P4080 processor. http://www.frescale.com.

[6] Power consumption for MX960. http://www.juniper.net.

[7] PowerQuiccIII Monitors. http://www.freescale.com.

[8] Quidway NetEngine 5000e. http://www.huawei.com.

[9] Red hat enterprise linux 6 power management guide. http://docs.redhat.com/.

[10] The Tilera processor. http://www.tilera.com.

[11] Using the linux cpufreq subsystem for energy management. http://publib.boulder.ibm.com/infocenter/lnxinfo /v3r0m0/topic/liaai/cpufreq/liaai-cpufreq_pdf.pdf.

[12] Intel IXP hardware reference manual, January 2003.

[13] Global action plan report. http://www.globalactionplan.org.uk, 2006.

[14] System power challenges. http://www.slidefinder.net/c/ cisco_routing_research/seminar_august_29/1562106, 2006.

[15] Processor power management in windows vista and windows server 2008. http://www.microsoft.com, November 2007.

[16] J. Baliga, K. Hinton, and R. Tucker. Energy consumption of the internet. In *Optical Internet, 2007 and the 2007 32nd Australian Conference on Optical Fibre Technology. COIN-ACOFT 2007. Joint International Conference on*, pages 1 –3, june 2007.

[17] L. Benini, A. Bogliolo, and G. De Micheli. A survey of design techniques for system-level dynamic power management. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 8(3):299 –316, june 2000.

[18] W. L. Bircher and L. K. John. Analysis of dynamic power management on multi-core processors. ICS '08, NY, USA.

[19] L. Chiaraviglio, M. Mellia, and F. Neri. Reducing power consumption in backbone networks. In *In IEEE International Conference on Communications (ICC 09*, 2009.

[20] K. Claffy, D. Andersen, and P. Hick. The caida anonymized 2011 internet traces.

[21] P. Crowley and J. L. Baer. A modeling framework for network processor systems. In *Network Processor Workshop*, 2002.

[22] R. Dömer, A. Gerstlauer, J. Peng, D. Shin, L. Cai, H. Yu, S. Abdi, and D. D. Gajski. System-on-chip environment: a specc-based framework for heterogeneous mpsoc design. *EURASIP J. Embedded Syst.*, 2008, January 2008.

[23] U. Ewaldsson. Cut your network's electricity bill and carbon footprint. http://wwww.globaltelecomsbusiness.com /Article/2436697/Cut-your-networks-electricity-bill-and-carbon-footprint.html, February 2010.

[24] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *Networking, IEEE/ACM Transactions on*, 1(4):397 –413, aug 1993.

[25] M. A. Franklin and T. Wolf. Power considerations in network processor design. In *In Network Processor Workshop in conjunction with Ninth International Symposium on High Performance Computer Architecture (HPCA-9)*, 2003.

[26] D. D. Gajski, J. Zhu, R. Domer, A. Gerstlauer, and S. Zhao. *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers Boston, MA, 2000.

[27] A. Gerstlauer. Host-compiled simulation of multi-core platforms. In *Rapid System Prototyping (RSP)*, 2010.

[28] Gerstlauer, A. et al. Abstract system-level models for early performance and power exploration. In *ASPDAC*, 2012.

[29] T. Grotker. *System Design with SystemC*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[30] S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. ISLPED '07, pages 38–43, New York, NY, USA, 2007. ACM.

[31] S. Herbert and D. Marculescu. Variation-aware dynamic voltage/frequency scaling. In *HPCA*, Feb. 2009.

[32] K. Hinton, J. Baliga, M. Feng, R. Ayre, and R. Tucker. Power consumption and energy efficiency in the internet. *Network, IEEE*, 25(2):6 –12, march-april 2011.

[33] Intel, Microsoft, and Toshiba. Advanced configuration and power interface specifications. http://www.intel.com/iam/powermgm/specs.html, 1996.

[34] M. F. Iqbal and L. K. John. Power and performance analysis of network traffic prediction techniques. In *ISPASS*, 2012.

[35] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *MICRO-39*, dec. 2006.

[36] S. Kaxiras. Ipstash: A set-associative memory approach for efficient ip-lookup. *IEEE Infocom*, pages 992–1001, 2005.

[37] R. Kokku, T. L. Riché, A. Kunze, J. Mudigonda, J. Jason, and H. M. Vin. A case for run-time adaptation in packet processing systems. *SIGCOMM Comput. Commun. Rev.*, 34:107–112, January 2004.

[38] J. Kuang and L. Bhuyan. Optimizing throughput and latency under given power budget for network packet processing. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, pages 2901–2909, Piscataway, NJ, USA, 2010. IEEE Press.

[39] B. Lee and L. John. Npbench: a benchmark suite for control plane and data plane applications for network processors. In *Computer Design*, oct. 2003.

[40] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. MICRO 42, NY, USA, 2009.

[41] Y. Luo, J. Yang, L. Bhuyan, and L. Zhao. Nepsim: a network processor simulator with a power evaluation framework. 24:4 – 44, 2004.

[42] Y. Luo, J. Yu, J. Yang, and L. Bhuyan. Low power network processor design using clock gating. In *DAC*, pages 712–715, 2005.

[43] Y. Luo, J. Yu, J. Yang, and L. N. Bhuyan. Conserving network processor power consumption by exploiting traffic variability. *ACM Trans. Archit. Code Optim.*, 4(1), Mar. 2007.

[44] G. Magklis, P. Chaparro, J. Gonzalez, and A. Gonzalez. Independent front-end and back-end dynamic voltage scaling for a gals microarchitecture. In *ISLPED*, oct. 2006.

[45] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. CAN, 2005.

[46] G. Memik and W. H. Mangione-smith. Increasing power efficiency of multi-core network processors through data filtering. In *In Intl. Conf. on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 108–116. ACM Press, 2002.

[47] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall. Reducing network energy consumption via sleeping and rate-adaptation. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 323–336, Berkeley, CA, USA, 2008. USENIX Association.

[48] R. Ramaswamy and T. Wolf. PacketBench: A tool for workload characterization of network processing. In *Proc. of IEEE 6th WWC*, Austin, TX, Oct. 2003.

[49] B. Sanso and H. Mellah. On reliability, performance and internet power consumption. In *Design of Reliable Communication Networks, 2009. DRCN 2009. 7th International Workshop on*, pages 259 –264, oct. 2009.

[50] A. V. D. Ven. Absolute Power. http://software.intel.com/sites/oss/pdfs/absolutepower.pdf.

[51] Q. Wu and T. Wolf. On runtime management in multi-core packet processing systems. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, pages 69–78, New York, NY, USA, 2008. ACM.

[52] F. Zane, G. Narlikar, and A. Basu. Coolcams: Power-efficient tcams for forwarding engines. In *IN IEEE INFOCOM*, pages 42–52, 2003.