Computer Communications 50 (2014) 95–109

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Modeling power management in networked devices

Roberto Bruschi [a,*], Paolo Lago [b], Alfio Lombardo [c], Giovanni Schembra [c]

[a] National Inter-University Consortium for Telecommunications, Italy
[b] Department of Electrical, Electronic and Telecommunication Engineering, and Naval Architecture (DITEN), University of Genoa, Italy
[c] Dipartimento di Ingegneria Elettrica, Elettronica e Informatica (DIEEI), University of Catania, Italy

ARTICLE INFO

Article history:
Available online 5 April 2014

Keywords:
Power management
Networked devices
Markov models
Quality of service

ABSTRACT

This paper focuses on the power management of state-of-the-art networked devices (like common PCs, servers, set-top boxes, etc.) to evaluate their behavior, model their internal dynamics and possible sources of inefficiency, and optimize their performance and energy efficiency. To this purpose, we started from an experimental characterization of the power management schemes in common device platforms based on commercial off-the-shelf hardware and open-source software (i.e., common PCs/servers devices running the Linux operating system). The characterization allowed us to formalize an analytical model able to accurately capture the power management dynamics at hardware (at ACPI level and beyond) and software levels (Linux Governors). Finally, the proposed model has been applied to analyze the efficiency of networked devices according to various configurations of internal parameters and incoming workload. Thanks to its intrinsic accuracy and the representation of different fine-grained details, the model is able to provide precious information on the possible sources of inefficiency, and on how to act on policy parameters to optimize the system behavior.

1. Introduction

In the last few years, a number of studies have identified the eco-sustainability as one of the key aspects that may potentially constraint the Internet technology evolution, and its wide-adoption as public telecommunication infrastructure. Despite obvious environmental reasons, this interest springs from heavy and critical economical needs [1,2].

For supplying networking infrastructures and large-scale datacenters, telecom and service operators have so impressive and continuously increasing power requirements that appear to be among the major direct energy consumers in their nations. Telecom operators and third-parties estimated that such energy requirements will rapidly become no more sustainable, if no radical changes in Internet technology design will be undertaken [1]. Moreover, this projections becomes even more impressive if we consider not only "networking" devices (e.g., routers, switches, etc.) inside telecom and home networks, but also the "networked" ones (like common PCs, servers, set-top boxes, smartphones, etc.) [2–4], both in the user homes and in datacenters. In fact, consumer electronics are becoming smarter and even more "connected" through Internet technologies. Despite the proliferation of tablets and mobile phones, many other customer premises equipment are going to massively appear in the user homes (e.g., VoIP phones, set-top boxes, smart household appliances, etc.) [5].

At the same time, the largest part of data used by such devices, and related elaboration/processing, is moved inside powerful datacenters. It is a clear attempt to leverage "smart" devices from complex operations and high-end (energy-hungry) hardware requirements, while at the same time providing end-users with access to applications and data independently from the device and its location. To this purpose, applications are even more designed to be split between end-user and datacenter devices, and connected through the network. This application design trend is obviously causing a significant increase of network traffic, as well as the usage of network hardware and software components inside networked devices. Cisco forecasts that by 2015 Internet traffic will get very close to the impressive threshold of 1 zettabytes a year [5].

Starting from this scenario, it can be argued that the sustainability of the Internet strongly relies on the efficiency of network technologies working at the edges (customer homes and datacenters), and also inside networked devices [6].

To this goal, hardware platforms for networked devices should include advanced power management schemes, proving a certain proportionality between their energy consumption and actual

* Corresponding author. Tel.: +39 010 353 2057; fax: +39 010 353 2154.
E-mail addresses: roberto.bruschi@cnit.it (R. Bruschi), paolo@reti.dist.unige.it (P. Lago), lombardo@dieei.unict.it (A. Lombardo), schembra@dieei.unict.it (G. Schembra).

http://dx.doi.org/10.1016/j.comcom.2014.03.027
0140-3664/

workload. At the same time, the software (and especially the operating system) of such devices needs to be as optimized as possible to be aware of drawbacks of power management schemes, and to meet processing dynamics that are largely triggered by network traffic coming from the network.

It is worth noting that any optimization solutions, that even assure only very small power savings, may have a disruptive impact given the high density of networked hosts. However, the adoption of these power management capabilities in network devices affects both the performance of software applications, and of the Internet traffic, as well as the device energy savings [7].

In this paper, we deal with the efficiency of power management of state-of-the-art networked devices to deeply understand and evaluate their internal dynamics and possible sources of inefficiency. To this purpose, we started by focusing on the experimental characterization of common device platforms based on commercial off-the-shelf hardware and open-source software (i.e., common PCs/servers devices running the Linux operating system). This choice has been mainly driven by two main reasons: (*i*) commercial off-the-shelf hardware can well represent a large segment of technologies applied in networked devices, and already includes advanced power management mechanisms; (*ii*) the Linux operating system is open-source and allows us knowing/customizing every detail of the running software. Moreover, the selected software and hardware platforms are so widespread in consumer electronics, that the largest part of results and considerations can be easily extended in other scenarios (e.g., like in Android smart phones [8]).

The obtained measures allowed us to formalize a Markov based analytical model which is able to accurately capture the power management dynamics at hardware and software levels. In more detail, at the hardware level, ACPI power management primitives have been considered along with their fine-grained features and drawbacks (e.g., delay times in switching clock frequencies and in waking up hardware components, wake-up power consumption peaks, etc.). At the software level, we focused on the power management support of the Linux operating system. The orchestration policy of power management primitives and related operations (i.e., usage monitoring of hardware resources), both included in software modules also known as "Governors," has been accurately modeled.

Then, the proposed model has been applied to analyze the efficiency of the entire system according to various configurations of internal parameters and incoming workload. Thanks to its intrinsic accuracy and the representation of different fine-grained details, the model is useful to evaluate the system behavior in a number of incoming workload scenarios and power management configurations. The model output can provide a large set of precious information, which could be hardly obtained through experimentations. For instance, the model can be adopted to drive the parameter optimization of the Linux Governors, and to design and refine the performance of packet coalescing algorithms. Indeed, coalescing is a common approach at packet [9–13] and task scheduling levels [14–16] for improving processing performance. Despite of the different application scenarios, coalescing techniques are often designed on simple common idea to provide more efficient batch elaboration by sensibly reducing computational overhead. Recently, these kind of techniques have been successfully applied for boosting energy efficiency in network hardware (e.g., IEEE 802.3az interfaces [12]) and in networked devices (e.g., smartphones and tablets [13,16]).

The remainder of this paper is organized as follows. Section 2 introduces the anatomy of the power management subsystem in Linux devices based on commercial off-the-shelf hardware. Section 3 introduces how hardware and software components of the power management system interact among themselves at a fine-grained detail, with a particular reference to the case in which the PC workload is mainly driven by network traffic. Experimental results on the efficiency and dynamics of power management schemes are also shown in Section 3. The formalization of the proposed model is in Section 4, while the derivation of model-related performance indexes is in Section 5. Section 6 reports the performance evaluation results according to Governor parameters and workload levels. The conclusions are drawn in Section 7. Finally, an appendix section includes the description of the tools and instrumentation used to collect the experimental results.

2. The power management anatomy in linux PCs

In off-the-shelf PCs and similar devices (e.g., smart phones, tablets, etc.), the power management system is usually designed across three layers, namely hardware, firmware and software (see Fig. 1). At the *hardware* level, physical components (e.g., CPU, video cards, etc.) include specific mechanisms and solutions for dynamically modulating energy consumption and performance. These mechanisms are usually based on well-known hardware techniques for power management [18], like Dynamic Frequency Scaling (DFS), Dynamic Voltage and Frequency Scaling (DVFS), clock and supply voltage gating, etc. These mechanisms are usually implemented as power management "primitives" that can be enabled, disabled or configured by the software/firmware levels, which owns enough information to decide the most suitable trade-off between performance and energy consumption to meet application and user requirements.

The *firmware* level (i.e., the PC BIOS – Basic Input–Output System) includes the ACPI (Advanced Configuration and Power Interface) framework [19], which provides a standard interface between hardware-dependent power saving techniques and software layers.

Focusing on processors' power saving primitives, ACPI introduces two main types of abstraction for power management mechanisms, namely performance and idle states (*P*- and *C*-states), respectively. P- and C-states offer the possibility to the software level to neglect the specific implementation features and peculiarities of power management techniques at hardware level, and to expose them by means of intuitive functional abstractions.

In more detail, regarding C-states, C_0 is the active power state where the CPU executes instructions, while C_1 through C_n states correspond to low power idle modes, where the processor consumes less power and dissipates less heat. As the idle state becomes deeper ($C_1 \rightarrow C_n$), the transition between the active and an idle state (and vice versa) requires longer time and more energy [20]. In more detail, when a CPU is in the C_1 state, clock signals of the core are disabled. When the CPU is in the C_3 state, in addition to the core clocks turned off, the contents of Level 1 instruction cache as well as the Level 1 and 2 data caches of the processors are powered off. It is worth noting that transition times for moving to and from the C_3 state are significantly longer than in state C_1, since the CPU caches need to be flushed and restored from higher (and slower) memory units [23].

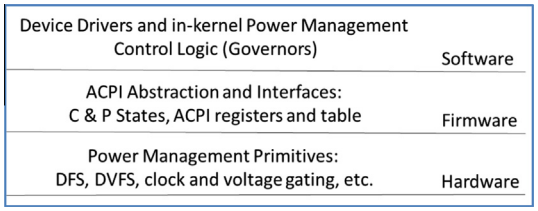


Fig. 1. Architecture and main components of the power management system in off-the-shelf PCs.

In the C_0 state, ACPI allows the performance of the processor's core to be tuned through P-state transitions. P-states allow modifying the operating energy point of a core by altering the working frequency and/or voltage. Thus, by using P-states, CPU can consume different amounts of power while providing different processing performance at the C_0 state. At a given P-state, the CPU can transit to higher C-states in idle conditions. In general, the higher the index of P- and C-states, the less will be both the power consumed and the heat dissipated.

At the *software* level, and with special reference to the Linux operating system, these power management primitives are dynamically orchestrated by specific control logics, named "*Governors*" [21,22]. Governors are hosted by the *cpuidle* and *cpufreq* software infrastructures that are devoted to control and manage C- and P-states, respectively.

The *cpuidle* infrastructure supports two Governors, namely *Ladder* and *Menu*, the former takes a step-wise approach by progressively setting deeper C-states depending on the time spent in the last idle period; the latter allows the CPU/Core to jump to the lowest possible idle state that does not significantly affect performance. In detail, the user and/or a device driver can request the Governor the maximum wake-up latency.

Regarding P-states, five Governors can be applied in the *cpufreq* subsystem: *Performance*, *Powersave*, *Userspace*, *Conservative* and *Ondemand*. The *Performance* and the *Powersave* policies are static and select, respectively, the highest and the lowest frequency available. The *Userspace* Governor allows the user or a user space application to set the desired working frequency. The *Conservative* Governor selects the CPU speed depending on the current utilization. In detail, every 200 ms the CPU utilization is monitored, if the utilization is above a certain threshold (called *up_threshold*) this Governor will step up the frequency to the next higher available frequency; if the utilization is below the *down_threshold,* the Governor will step down the frequency to the next lower available frequency.

The *Ondemand* Governor (OG) was introduced into Linux kernel 2.6.9 and has undergone several changes over time. In detail, we will consider the latest version available in the 3.6.8 Linux kernel. This Governor is similar to the *Conservative* one, since it monitors the CPU utilization but uses only one threshold (σ): if the utilization is above σ the clock frequency jumps to the maximum one, otherwise the Governor directly sets the minimum frequency available. The default value of the σ parameter is 85%, but it can be easily tuned by system administrators. The value of such threshold has clearly an important impact on CPU energy efficiency and processing performance. In fact, if σ is too small, the OG policy will set the maximum frequency even when the CPU is under-utilized, providing poor energy gain. On the contrary, a too large value of σ will make the CPU working with the minimum frequency even for medium/high workloads, considerably increasing its execution/processing times. The model proposed in this paper can support designers to choose this threshold, finding the best tradeoff between energy saving and system performance.

Fig. 2 summarizes the CPU power management architecture in a Linux PC.

In our case study we considered the system configured to work with the *Ondemand* Governor enabled to tune the clock frequency (in terms of P-states), and by leading the *Menu* Governor to throttle between the C_0 and the C_3 states, but every power management configuration can be evaluated through our model. We chose to focus on this power management configuration since it is the most widely used, and the default setting in the largest part of Linux distributions.

3. Power management dynamics

In order to accurately understand the power management of networked devices, a number of low-level/fine-grained details should be carefully considered. In particular, it is worth noting that the workload coming from intensive network traffic stays at a time scale very close to power management dynamics, differently from the workload triggered by direct interactions of users. Thus, in order to obtain an accurate evaluation of the Linux power management system in such a scenario, we need to take into account fine-grained behavior over time, and possible inefficiencies of both hardware implementation of C- and P-states in off-the-shelf CPUs, and control logic in operating systems.

To this purpose, this section is organized as follows. Subsection 3.1 introduces a set of experimental results, obtained with the aim of identifying and evaluating the main inefficiencies and drawbacks of power management schemes implemented at processor hardware level. Starting from this analysis and from the description of the Linux power management support in Section 2, Section 3.2 will summarize and formalize the dynamic

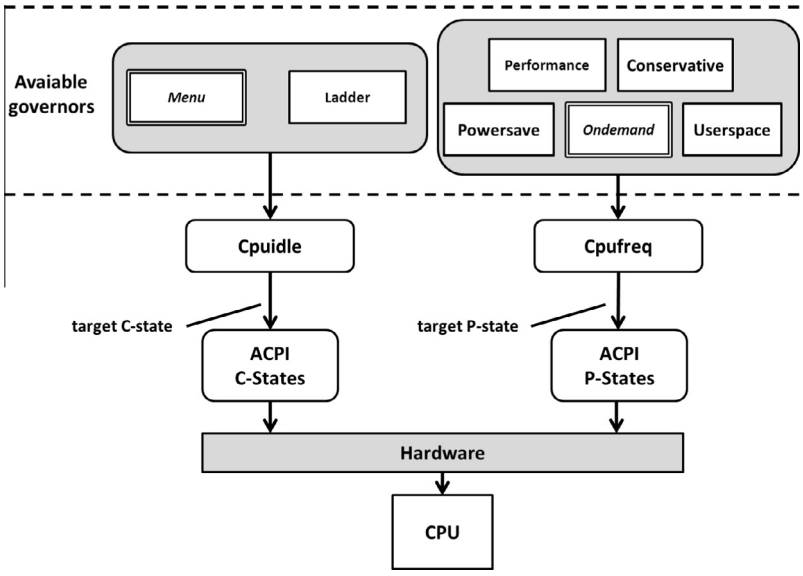


Fig. 2. The CPU power management architecture in Linux OS.

power profile of a CPU running the Linux operating system. In this respect, relevant details coming from hardware behavior, from Linux Governors and from applicative workload will be jointly considered to obtain a complete profiling of the CPU power management behavior.

3.1. Power management drawbacks and inefficiencies at hardware level

In order to identify possible sources of inefficiency and fine-grained details on the behavior of power management mechanisms available on off-the-shelf CPU, we performed a large number of experimental measurements by using the tools introduced in the Appendix. However, for the sake of simplicity and readability, we decided to report in this section only a limited set of the obtained results. The choice was driven with the aim of giving a useful insight about power management dynamics (especially at the hardware level) by avoiding dispersing the reader focus on too much details and testing scenarios. Additional details on this kind of measurements results can be found in [24] and [25], where only preliminary experimental results on the CPU energy consumptions dynamics are reported.

All the reported results have been obtained with an Intel I5 processor running a Linux operating system (Ubuntu distribution version 12.1 with a 3.6.8 kernel). As previously said, we left the default power management configuration of the operating system that includes the *Ondemand* and *Menu* Governors. The *Ondemand* Governor switches the CPU clock frequency between the maximum and the minimum values (2.67 and 1.60 GHz, respectively) available from the CPU hardware. The Menu Governor is configured to directly enter the CPU into the C_3 state upon it moves to the idle condition.

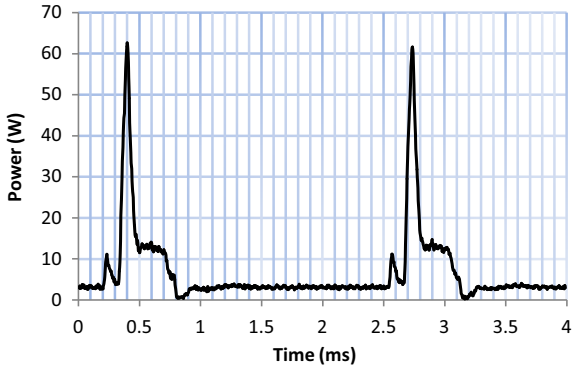


Fig. 3. CPU power consumption during the reception process of two IP bursts at 2.67 GHz and C3 idle state.

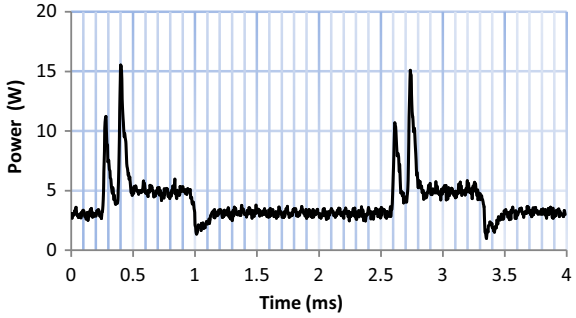


Fig. 4. CPU power consumption during the reception process of two IP bursts at 1.60 GHz and C_3 idle state.

Figs. 3 and 4 report the instantaneous power consumption of the I5 CPU, when working at 2.67 GHz and 1.60 GHz, which correspond to P_0 and P_8 states, respectively. In both the cases, the CPU receives (and processes) two bursts of 500 packets from the network interface (the first one at approximately 0.25 ms, and the second one at 2.6 ms).

The above figures present a similar behavior: a very low power consumption (3 W) during CPU idleness in C_3 state, and a considerable start-up power, which happens at the burst reception instant, needed to move from C_3 to the active state C_0. In more detail, during the wake-up procedure (from 0.25 ms to 0.5 ms and from 2.6 ms to 2.8 ms with reference to Figs. 3 and 4), two different peaks in power consumption can be identified. The former (and smaller one) is independent on the selected P-state, and is due to the wake-up and restoring operations of CPU instruction- and data-caches (see Section II). The latter clearly depends on the selected P-state (it achieves a maximum value of 62 W with the 2.67 GHz frequency, and only 15.5 W in case of 1.60 GHz), and is caused by the CPU core wake-up. This frequency dependency is partially due to the fact that the Core I5 includes DVFS mechanisms to implement P-States [23], which sensibly limit the maximum power adsorption by the CPU when working at lower frequencies.

It is also worth noting that, despite the different energy absorption, the transition time to leave C_3 appears to be constant and independent on the P-state selection.

Upon completion of the hardware wake-up phase, the core starts processing the incoming traffic at the selected P-state/clock frequency. It can be observed how not only the power consumption is influenced by the selected P-state (i.e., 13.3 W in case of 2.67 GHz clock frequency, and 4.7 W when the 1.67 GHz frequency is applied), but also elaboration time exhibits a clear dependency on the working frequency. In fact, the elaboration phases, starting at 0.5 ms and 2.8 ms in both Figs. 3 and 4, last for approximately 0.2 ms in the case of 2.67 GHz clock frequency, and 0.4 ms in the other considered case. It is important to underline how these processing times are in the same order of magnitude of C-state wake-up transitions.

Then after all the received packets have been processed, as the CPU becomes idle, it re-enters the C_3 state. During this period, the core and the caches are switched off, lowering the instantaneous energy consumption of the whole CPU.

In order to complete the characterization of power management behavior and related inefficiencies, we performed further tests also in case of transitions between P-states. It is worth recalling that these transitions are driven by the Governor logics hosted by the cpufreq infrastructure (see Section 2).

Fig. 5 reports one example of the instantaneous energy consumption profile during the transitions from the 1.60 GHz to the

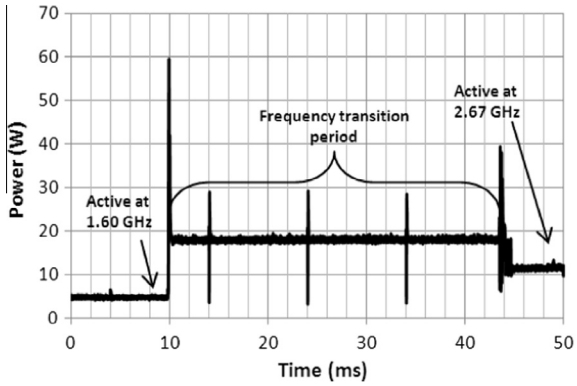


Fig. 5. CPU power consumption during the frequency transition from 1.60 GHz to 2.67 GHz.

2.67 GHz clock frequency for the above cited Intel Core I5. Many other tests, also obtained with other processor platforms, exhibited the same behavior.

The results in Fig. 5 outline how also the P-state transitions require an important amount of energy and time to be completed. In more detail, the CPU is fully active (in C_0) at the lowest clock frequency, and 10 ms after the measurement starts, the Ondemand Governor decides to select the P-state with better performance (i.e., the one corresponding to the 2.67 GHz clock frequency). When the selection is applied, the CPU stops working for approximately 35 ms (from 10 to 45 ms with reference to Fig. 5), exhibiting a relative high power consumption level of 18 W.

Moreover, at the right and left bounds of such transition period (i.e., at 10 ms and 45 ms), we can identify two power spikes, whose values depend on the working frequency. During the transition period, we can also see three couples of opposite spikes, which do not affect the average power consumption and can be neglected.

3.2. Characterizing and understanding the dynamic power profiles

The main objective of the present work is to accurately represent the power consumption profile over the time of the processor by also taking transitions among C- and P-states and related inefficiencies into account. The main reasons for this approach are given from the not negligible energy consumption due to state transitions, especially when the CPU workload is mainly triggered by network traffic.

Considering the measurement results and analysis in Subsection 3.1, we broke the possible transitory profile of a generic ACPI-compliant CPU (working with the *Ondemand* and *Menu* Governors as introduced above) in a set of 15 descriptive states "Sx", as shown in Figs. 6 and 7. In Table 1 such descriptive states have been associated with the related CPU power consumption and sojourn times as obtained by the experimental results. Moreover, we considered that the system workload is driven and triggered exclusively by incoming network traffic (this is a usual case for servers and firewalls).

With particular reference to Figs. 6 and 7, the S0 state represents the CPU while idle, (i.e., in the C_3 state). The CPU exits from S0 upon the reception of one or more packets or a scheduled power management event, and pass through the S1, S2 and S3(P_i) states, which togheter represent the wake-up transitions from C_3 (and, in more details, the two consumption peaks discussed in Subsection 3.1, and the time interval between them). The argument P_i represents the P-state/CPU frequency currently selected. We recall that the *Ondemand* Governor switches only between the maximum frequency (i.e., P_0, and corresponding to 2.67 GHz frequency value for the above cited Core I5 processor) and the minimum one (represented with the P_N ACPI state, where N is the maximum number of P-states available in the processor, and corresponding to 1.60 GHz for the Core I5). Thus, we indicate with $i = 0$ the cases

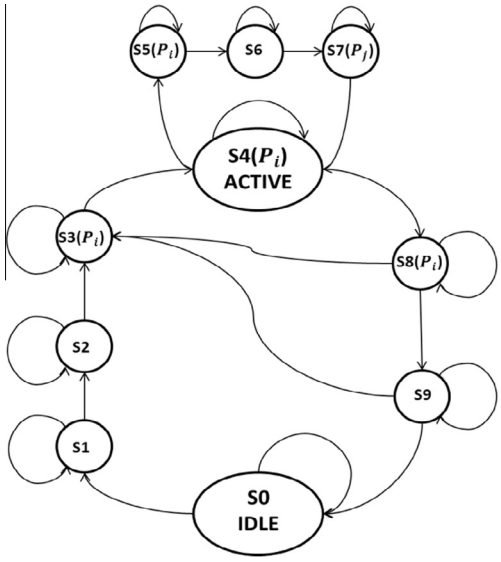


Fig. 7. State diagram of the CPU.

Table 1
Estimated values of power and duration of the system state.

State index	Power (Watt)	Sojourn Times (μs)
S0	3	–
S1	6.7	100
S2	4.1	24
S3(0)	38	100
S3(N)	9	100
S4(0)	13.3	–
S4(N)	4.7	–
S5(0)	21	196
S5(N)	35	196
S6	18	33300
S7(0)	20	80
S7(N)	20	80
S8(0)	6	80
S8(N)	3	20
S9	1.5	140

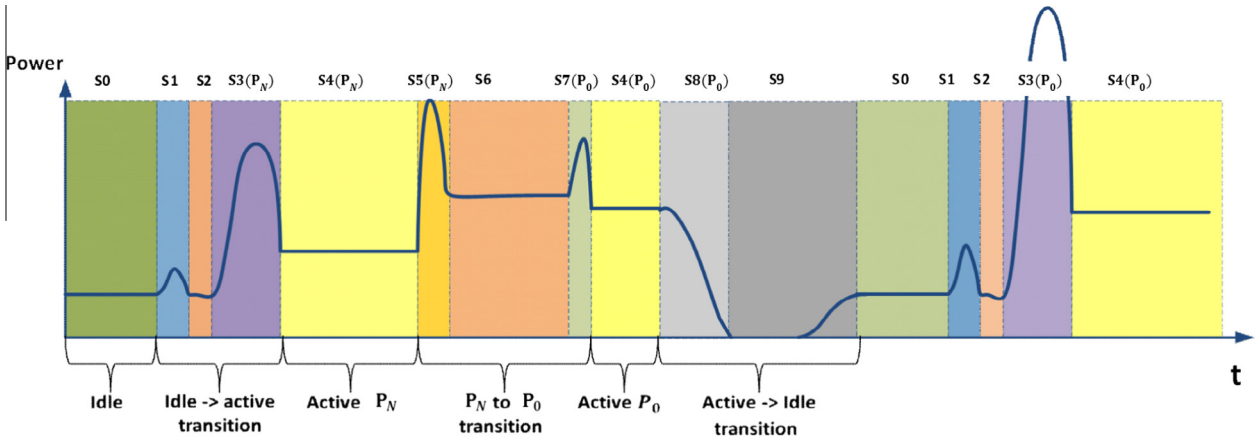


Fig. 6. Example of processor power state profile.

where the P-state is P_0, and with $i = N$ the ones where the P-state is P_N.

When a packet is received, it is backlogged in the network interface buffer, waiting for CPU elaboration. After exiting from the S3(P_i) state, the CPU becomes active at the selected frequency P_i, and starts processing the information contained in the received packets buffered at the network interface. This state is represeted by S4(P_i). The system can leave the S3(P_i) state only in two cases: the backlog of software jobs requiring processing is empty (and, then, the CPU is ready to re-enter the idle mode), or the Ondemand Governor decides to change the selected P-state (i.e., from P_0 to P_N or vice versa).

Regarding the P-state switching, the *Ondemand* Governor periodically monitors the CPU utilization. By default, the monitoring process observes the CPU utilization every 200 ms, and decides to switch the operating frequency if the measured utilization exceeds/is lower than the configured threshold. Obviously, monitoring events and *P*-state switching can occur only during CPU activity periods. So, monitoring state switching can require a CPU wake-up. If the *Ondemand* Governor decides for a P-state switching, the system moves through the S5(P_i), S6 and S7(P_j) states, with the j parameter representing the new operating frequency. Such states represent the same transient measured in Fig. 5 (i.e., the two peaks on the boundaries and the flat power consumption period). After, the system enters the active S4(P_j) state.

When the processing backlog is emptied and the CPU is ready to return to the idle mode, the system passes through the S8(P_i) and S9 to re-enter in low power idle (S0). It is worth noting that S8(P_i) and S9 are preemptible: if a new packet arrives to the system during these states, the transition towards *C*3 is aborted and the system immediately moves to the wake-up procedure (see Fig. 7).

Table 1 reports the estimated values of power and duration of all the system states showed in Figs. 6 and 7.

4. The proposed model

The proposed model is aimed at representing the time behavior of a networked device (based on off-the-shelf hardware and Linux) in terms of performance and energy consumption. Its main target is to provide a tool for evaluating and optimizing the power management configuration as, for instance, finding the parameter settings of existing Linux operating system Governors that best suits to a certain hardware and software setup of the networked device. To this purpose, the model explicitly considers fine-grained dynamics of energy consumption at the CPU, and the performance that the system may provide under the desired governor settings and computational loads.

More specifically, Subsection 4.1 introduces some important concepts that will be at the base of the model, which will be presented in Section 4.2.

4.1. Base modelling concepts

As outlined in Sections 2 and 3, the internal software/hardware architecture, the organization, the operating dynamics, and the power-management schemes make the representation of networked device behavior extremely complex, given also the strict interactions between CPU processing/utilization and power management dynamics.

As highlighted from the experimental results presented in Section 3, in order to provide results enough accurate to be used in real-world applications, the model is designed to capture any performance drawbacks that the adoption of power management schemes and mechanisms can cause. Among other aspects, the proposed model is able to represent consumption peaks during CPU wake-ups or unavailability of the CPUs during clock frequency switching.

At software level, the model includes the logic of the "Ondemand" Governor, but can be easily extended to other existing Governors or even new power management policies.

The core part of the proposed framework is the state machine shown in Fig. 7, which is associated to the CPU power management dynamics (as reported in Fig. 6). In addition to this core part, the model also includes other modules aimed at representing:

- the CPU processing in terms of both software activities/computational load that have to be carried out before returning in sleep mode;
- the CPU processing speed when a certain CPU clock frequency is selected;
- the sample collection of CPU utilization performed by the Governor, the criteria used to switch frequency, and the CPU unavailability period during the switching periods;
- the generation process of new software activities to be performed by the CPU.

In this respect, for the sake of easiness, we consider that the networked device workload arises only from network Input/Output[1] (I/O). In other words, the proposed model assumes that the networked device is triggered by packets incoming from the network. The model does not explicitly consider which kind of operations are executed on the incoming packets, but rather the workload that the received packets cause on the CPU. In more detail, the model binds to each received packet a computational load, the following referred to as "job", which can be meant as the number of clock cycles required to accomplish the processing of the received information.

This modeling approach allows us to flexibly represent a huge set of scenarios where different kinds of network applications/services may reside on the considered device. In principle, different applications/services could be fully characterized by defining different traffic patterns and per-packet computational loads (i.e., with different "job lengths").

Finally, another important consideration we have to make before presenting the model regards the relationship among operating system task scheduling policies, network I/O operations, and CPU power management. Since network I/O operations in Linux are realized in high-priority pre-emptive software activities (i.e., mainly hardware and software interrupt routines), the impact of CPU task scheduling policies of the operating system is almost negligible and it does not affect power consumption metrics. In fact, in the considered scenario, task scheduling might only change the elaboration order of software activities of user-space applications/services. Since the CPU is maintained active until all the software activities are performed, task scheduling does not have any impact on the CPU sleeping dynamics.

4.2. Markov model

In this section we define a discrete-time model of the behavior of the whole system described so far. In order to capture both first- and second-order statistics of the system, we use a multidimensional Markov model.

[1] It is worth noting that the model can be easily extended to the case where other "non-network driven" software applications run on the considered device. However, the model in its current form is able to represent with a good accuracy the behavior of the majority of networked devices (like servers, home-gateways, etc.), since many of their applications/services are mostly triggered by network traffic, rather than on direct human interactions (e.g., packet reception events are usually much more frequent than typing inputs on a keyboard).

As regards the input traffic, we model it as a Markov Modulated Bernoulli Process (MMBP), but any Markov model defined in the literature can be applied (see for example [26,27]). The choice of an MMBP model is due to its ability to capture both first- and second-order statistics of the traffic, demonstrated in [28] to be sufficient to represent the input traffic in performance evaluation of queueing systems. According to its definition, we characterize it with the transition probability matrix of the underlying Markov chain, $P^{(S)}$, and the array $\underline{\gamma}^{(S)}$, containing the packet arrival probability for each state of the underlying Markov chain.

Let us now define the model of the whole networked device. To this purpose we use a 6-dimensional model whose stateis $S^{(\Sigma)}(n) = (S^{(S)}(n), S^{(X)}(n), S^{(C)}(n), S^{(P)}(n), S^{(U)}(n), S^{(Q)}(n))$, where:

- $S^{(S)}(n) \in \Im^{(S)}$ is the state of the underlying Markov chain of the MMBP process representing the input traffic at the generic slot n;
- $S^{(X)}(n) \in \{0,1,2\}$ is the utilization monitoring process of the Governor at the generic slot n: if $S^{(X)}(n)$ becomes equal to 1 when the processor is active (state S4), at the same slot the Governor compares the processor utilization percentage with the utilization threshold to decide whether to change the clock frequency or not; on the contrary, if the processor is not active at the instant scheduled for the monitoring, $S^{(X)}(n)$ temporally moves to the state 2, waiting for the processor activation; when the processor enters the active state, the monitoring job is enqueued at the head of the processor queue and the utilization monitoring process returns to the state 0, waiting for the next utilization monitoring event;
- $S^{(C)}(n) \in \Im^{(C)}$ is the processor clock frequency process. It can assume all the frequency values allowed by the processor;
- $S^{(P)}(n) \in [0,9]$ is the processor state process at the generic slot n, as described in Section III.B, and specifically in Fig. 7;
- $S^{(U)}(n) \in \Im^{(U)}$ is the processor utilization process; in order to capture its behavior with the model, the space state $\Im^{(U)}$ is a set of quantized values ranging from 0% to 100%;
- $S^{(Q)}(n) \in \{0,1,\ldots,K\}$ represents the processor queue state at the generic slot n; $K-1$ is the maximum number of jobs that can be buffered in the queue, while one job is processed by the processor.

Let Δ be the slot duration. In the Numerical Results section we will assume it as equal to 4 μs, in such a way that all the values listed in Table 1 result multiple of the slot duration.

Now, in order to define the model time diagram, we consider two generic states: $s_{\Sigma 1} = (s_{S1}, s_{X1}, s_{C1}, s_{P1}, s_{U1}, s_{Q1})$ in the slot n, and $s_{\Sigma 2} = (s_{S2}, s_{X2}, s_{C2}, s_{P2}, s_{U2}, s_{Q2})$ in the slot $n+1$. Fig. 8 represents the time diagram of the generic slot. We assume the following event sequence during each slot:

(1) The first process that changes its value is the traffic state;
(2) then according to a timer, the Governor decides whether monitoring the utilization process or not; accordingly, the process $S^{(X)}(n)$ is updated;
(3) then the clock frequency process $S^{(C)}(n)$ is updated according to the Governor decision on the basis of the utilization process value at the slot n, i.e., s_{U1}, that has been monitored only if $s_{X2} = 1$;
(4) then the processor clock frequency process $S^{(P)}(n)$ is updated assuming the frequency value imposed by the Governor, if any;
(5) then the utilization process $S^{(U)}(n)$ is updated according to the state of the processor during the whole slot $n+1$;
(6) then according to the new state of the traffic and the work done by the processor during the whole slot, the process $S^{(Q)}(n)$ is updated;
(7) finally, at the end of the slot $n+1$, the system state variables are observed.

Now we can define the generic element of the state transition probability matrix as follows:

$$\begin{aligned} P^{(\Sigma)}_{[s_{\Sigma 1}, s_{\Sigma 2}]} &= \mathrm{Prob}\{S^{(\Sigma)}(n+1) = s_{\Sigma 2} | S^{(\Sigma)}(n) = s_{\Sigma 1}\} \\ &= P^{(S)}_{[s_{S1}, s_{S2}]} \cdot P^{(X)}_{[s_{X1}, s_{X2}]}(s_{P1}) \cdot P^{(C)}_{[s_{C1}, s_{C2}]}(s_{X1}, s_{X2}, s_{U1}) \\ &\quad \cdot P^{(P)}_{[s_{P1}, s_{P2}]}(s_{Q1}, s_{C1}, s_{C2}, s_{X2}) \cdot P^{(U)}_{[s_{U1}, s_{U2}]}(s_{P2}) \\ &\quad \cdot P^{(Q)}_{[s_{Q1}, s_{Q2}]}(s_{S2}, s_{C2}, s_{P2}, s_{X2}) \end{aligned} \quad (1)$$

where:

- $P^{(S)}$ is the transition probability matrix of the underlying Markov chain of the input traffic.
- $P^{(X)}(s_{P1})$ is the transition probability matrix of the utilization monitoring state; since monitoring occurs with an average frequency of F_S times per second, the matrix $P^{(X)}(s_{P1})$ is:

$$P^{(X)}(s_{P1}) = \begin{bmatrix} 1 - F_S\Delta & F_S\Delta & 0 \\ I_{s_{P1}=4} & 0 & 1 - I_{s_{P1}=4} \\ I_{s_{P1}=4} & F_S\Delta(1 - I_{s_{P1}=4}) & (1 - F_S\Delta)(1 - I_{s_{P1}=4}) \end{bmatrix} \quad (2)$$

where $I_{s_{P1}=4}$ is the indicator function of the active state of the processor: it is equal to 1 only if the processor state is 4, while it is 0 otherwise. The Markov diagram of this process is shown in Fig. 9.

- $P^{(C)}_{[s_{C1}, s_{C2}]}(s_{X1}, s_{X2}, s_{U1})$ is the generic element of the transition probability matrix of the clock frequency process. As described in Section III.B, this process works according to the processor utilization at the last non-processed monitoring slot (this situation is represented by the variable $S^{(X)}(n)$ that changes its state moving from a state in {1, 2} to the state 0). In a slot in which

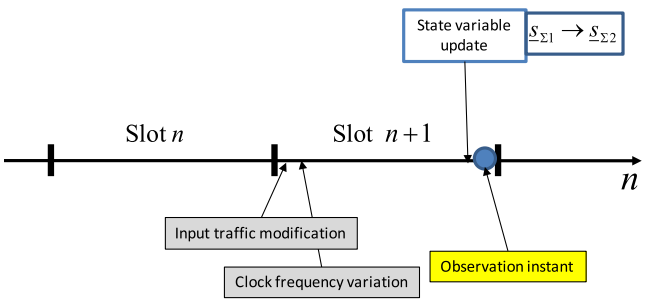


Fig. 8. System diagram.

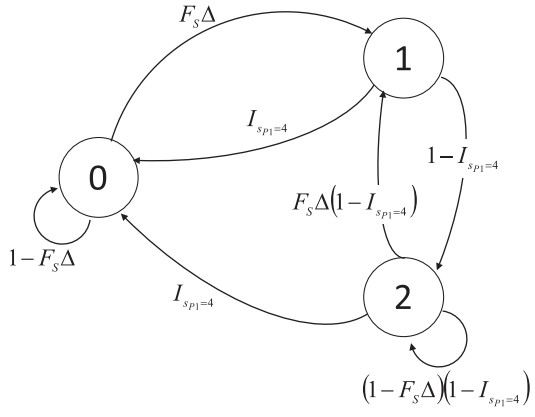


Fig. 9. State diagram of the monitoring process $S^{(X)}(n)$.

a monitoring event is processed ($s_{X1} \in \{1,2\}$ and $s_{X2} = 0$), the clock frequency is set to high ($s_{C2} = 1$) or low ($s_{C2} = 2$), according to the fact that the utilization is $s_{U1} \geqslant \sigma$, or $s_{U1} < \sigma$, respectively, and independently of the current value of the clock frequency. Therefore, the generic element of the transition probability matrix of this process is:

$$P^{(C)}_{[s_{C1},s_{C2}]}(s_{X1},s_{X2},s_{U1}) = \begin{cases} 1 & s_{C1}=s_{C2},\ s_{X1}=s_{X2}=0 \text{ or} \\ & s_{C1}=s_{C2},\ s_{X2}\neq 0 \text{ or} \\ & s_{C2}=1,\ s_{X1}\in\{1,2\},\ s_{X2}=0,\ s_{U1}\geqslant\sigma \text{ or} \\ & s_{C2}=2,\ s_{X1}\in\{1,2\},\ s_{X2}=0,\ s_{U1}<\sigma \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

- $P^{(P)}(s_{Q2}, s_{C1}, s_{C2}, s_{X2})$ is the transition probability matrix of the processor state; its Markov chain diagram follows the state diagram shown in Fig. 7. According to the processor behavior described in Section III.B, its generic element is defined as follows:

$$P^{(P)}_{[s_{P1},s_{P2}]}(s_{Q1},s_{C1},s_{C2},s_{X2}) = \begin{cases} 1 & \text{if } s_{P1}=0,\ s_{P2}=1,\ (s_{Q1}>0 \text{ or } s_{X2}\neq 0) \\ 1 & \text{if } s_{P1}=0,\ s_{P2}=0,\ (s_{Q1}=0 \text{ and } s_{X2}=0) \\ 1-\Delta/\overline{T}^{(s_{C2})}_{s_{P1}} & \text{if } s_{P1}\in\{1,2,3,5,6\},\ s_{P2}=s_{P1} \\ \Delta/\overline{T}^{(s_{C2})}_{s_{P1}} & \text{if } s_{P1}\in\{1,2,3,5,6\},\ s_{P2}=s_{P1}+1 \\ 1 & \text{if } s_{P1}=4,\ s_{P2}=8,\ s_{Q1}=0 \\ 1 & \text{if } s_{P1}=4,\ s_{P2}=5,\ s_{Q1}>0,\ s_{C2}\neq s_{C1} \\ 1 & \text{if } s_{P1}=4,\ s_{P2}=4,\ s_{Q1}>0,\ s_{C2}=s_{C1} \\ 1-\Delta/\overline{T}^{(s_{C2})}_{s_{P1}} & \text{if } s_{P1}=7,\ s_{P2}=s_{P1} \\ \Delta/\overline{T}^{(s_{C2})}_{s_{P1}} & \text{if } s_{P1}=7,\ s_{P2}=4 \\ 1 & \text{if } s_{P1}\in\{8,9\},\ s_{P2}=3,\ (s_{Q1}>0 \text{ or } s_{X2}\neq 0) \\ 1-\Delta/\overline{T}^{(s_{C2})}_{s_{P1}} & \text{if } s_{P1}\in\{8,9\},\ s_{P2}=s_{P1}, \\ & (s_{Q1}=0 \text{ and } s_{X2}=0) \\ \Delta/\overline{T}^{(s_{C2})}_{s_{P1}} & \text{if } s_{P1}\in\{8,9\},\ s_{P2}=\mathrm{mod}_{10}\{s_{P1}+1\}, \\ & (s_{Q1}=0 \text{ and } s_{X2}=0) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where $\overline{T}_{s_{P1}}$ is the mean duration of the state s_{P1}, as listed in Table 1. The symbol $\mathrm{mod}_7\{x\}$ is the x-modulo-7 operator.

- $P^{(U)}(s_{P2})$ is the transition probability matrix of the utilization process. In order to capture the behavior of the Governor in considering the past utilization history, we model the evolution of this process with an exponentially weighted moving average (EWMA) filter, that is:

$$\hat{U}_{n+1} = \varsigma U_{n+1} + (1-\varsigma)\hat{U}_n \tag{5}$$

where $\hat{U}_n$ and $\hat{U}_{n+1}$ are the average utilization at the slot n and $n+1$, respectively, while U_{n+1} is the instantaneous utilization at the slot $n+1$. The parameter ς can be timely chosen to decide the EWMA time constant. The instantaneous utilization at the slot $n+1$ can be calculated as follows:

$$U_{n+1} = \begin{cases} 1 & s_{P2}=4 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Since the value of $\hat{U}_{n+1}$ calculated in (5) may not belong to the state space $S^{(U)}(n) \in \Im^{(U)}$, we approximate the new utilization value with one of the two most adjacent states belonging to $\Im^{(U)}$. Let us indicate the most adjacent state with a value greater than $\hat{U}_{n+1}$ as $\lceil \hat{U}_{n+1} \rceil$, and the most adjacent state with a value lower than $\hat{U}_{n+1}$ as $\lfloor \hat{U}_{n+1} \rfloor$. The new average utilization state s_{U2} will be either $s_{U2} = \lceil \hat{U}_{n+1} \rceil$ or $s_{U2} = \lfloor \hat{U}_{n+1} \rfloor$ with a probability dependent on the distance between the average utilization value calculated as in (6) and the value associated to the adjacent states $\lceil \hat{U}_{n+1} \rceil$ and $\lfloor \hat{U}_{n+1} \rfloor$. More specifically:

$$S^{(U)}(n+1) = \begin{cases} \lceil \hat{U}_{n+1} \rceil & \text{with prob } :(\hat{U}_{n+1} - \lfloor \hat{U}_{n+1} \rfloor)/(\lceil \hat{U}_{n+1} \rceil - \lfloor \hat{U}_{n+1} \rfloor) \\ \lfloor \hat{U}_{n+1} \rfloor & \text{with prob } :(\lceil \hat{U}_{n+1} \rceil - \hat{U}_{n+1})/(\lceil \hat{U}_{n+1} \rceil - \lfloor \hat{U}_{n+1} \rfloor) \end{cases} \tag{7}$$

- $P^{(Q)}(s_{S2}, s_{C2}, s_{P2}, s_{X2})$ is the transition probability matrix of the processor queue state. Assuming that, as already said, within the same slot arrivals occur after departures, its generic element is defined as follows:

$$P^{(Q)}_{[s_{Q1},s_{Q2}]}(s_{S2},s_{C2},s_{P2},s_{X2}) = \sum_{\alpha=0}^{2} P(Arr=\alpha|s_{S2},s_{P2},s_{X2}) \cdot \begin{cases} 1 & \text{if } s_{Q1}=0, s_{Q2}=\alpha \\ 1-C(s_{C2}) & \text{if } s_{Q1}\in[1,K-2],\ s_{Q2}=s_{Q1}+\alpha \\ C(s_{C2}) & \text{if } s_{Q1}\in[1,K-2],\ s_{Q2}=s_{Q1}-1+\alpha \\ 1-C(s_{C2}) & \text{if } s_{Q1}=K-1,\ s_{Q2}=s_{Q1}+\alpha,\ \alpha\leqslant 1 \\ C(s_{C2}) & \text{if } s_{Q1}=K-1,\ s_{Q2}=s_{Q1}-1+\alpha,\ \alpha\leqslant 1 \\ 1 & \text{if } s_{Q1}=K-1,\ s_{Q2}=K,\ \alpha=2 \\ 1-C(s_{C2}) & \text{if } s_{Q1}=K,\ s_{Q2}=K-1,\ \alpha=0 \\ C(s_{C2}) & \text{if } s_{Q1}=K,\ s_{Q2}=K,\ \alpha=0 \\ 1 & \text{if } s_{Q1}=K,\ s_{Q2}=K,\ \alpha\geqslant 1 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

where $C(s_{C2})$ is the probability that one job is served by the processor in one slot when the clock frequency is s_{C2}. It can be derived by the processor capacity $\vartheta(s_{C2})$ as $C(s_{C2}) = \Delta \cdot \vartheta(s_{C2})$; the term $P^{(A)}(Arr=\alpha|s_{S2},s_{P2},s_{X2})$ is the probability that the number of jobs that arrive to the queue is α when the system state is such that $S^{(S)}(n+1) = s_{S2}$, $S^{(P)}(n+1) = s_{P2}$ and $S^{(X)}(n+1) = s_{X2}$. It can be calculated as follows:

$$\begin{aligned} P(Arr=2|s_{S2},s_{P2},s_{X2}) &= \begin{cases} \underline{\gamma}^{(S)}_{[s_{S2}]} & \text{if } s_{P2}=4 \text{ and } s_{X2}\neq 0 \\ 0 & \text{otherwise} \end{cases} \\ P(Arr=1|s_{S2},s_{P2},s_{X2}) &= \begin{cases} \underline{\gamma}^{(S)}_{[s_{S2}]} & \text{if NOT}\{s_{P2}=4 \text{ and } s_{X2}\neq 0\} \\ 1-\underline{\gamma}^{(S)}_{[s_{S2}]} & \text{otherwise} \end{cases} \\ P(Arr=0|s_{S2},s_{P2},s_{X2}) &= \begin{cases} 1-\underline{\gamma}^{(S)}_{[s_{S2}]} & \text{if NOT}\{s_{P2}=4 \text{ and } s_{X2}\neq 0\} \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{9}$$

where $\underline{\gamma}^{(S)}$ is the packet arrival probability array characterizing the MMBP traffic process.

Now, from the matrix $P^{(\Sigma)}$ we can derive the system steady-state probability array $\underline{\pi}^{(\Sigma)}$ by solving the following linear equation system:

$$\begin{cases} \underline{\pi}^{(\Sigma)} P^{(\Sigma)} = \underline{\pi}^{(\Sigma)} \\ \underline{\pi}^{(\Sigma)} \cdot \underline{1}^T = 1 \end{cases} \tag{10}$$

where $\underline{1}^T$ is a column array with all the elements equal to one. The generic element $\pi^{(\Sigma)}_{[s_\Sigma]}$ of the system steady-state probability array is the steady-state probability of the state $\underline{s}_\Sigma = (s_S, s_X, s_C, s_P, s_U, s_Q)$.

5. Performance parameter derivation

Let us now derive the main quality of service (QoS) system parameters by using the model proposed in the previous section.

The mean number of jobs in the system, i.e., in the queue and in service, can be calculated as follows:

$$E\{N\} = \sum_{s_Q=0}^{K} \underline{\pi}^{(Q)}_{[s_Q]} \tag{11}$$

where $\underline{\pi}^{(Q)}$ is the marginal steady-state probability array of the processor queue state. It can be derived from the system probability array as follows:

$$\underline{\pi}^{(Q)}_{[s_Q]} = \sum_{s_S \in \Im^{(S)}} \sum_{s_X \in \{0,1,2\}} \sum_{s_C \in \Im^{(C)}} \sum_{s_P=0}^{9} \sum_{s_U \in \Im^{(U)}} \underline{\pi}^{(\Sigma)}_{[s_S,s_X,s_C,s_P,s_U,s_Q]} \tag{12}$$

The mean value of arrived jobs per slot, $\overline{V}$, can be calculated as the sum of the mean number of jobs per slot relative to packet arrivals, $E\{Arr^{(Pack)}_{(s_{S1},s_{S2})}\}$, and the mean number of jobs per slot enqueued by the utilization monitoring process, $E\{Arr^{(Mon)}_{(s_{P1},s_{X1})}\}$. Therefore we have:

$$\overline{V} = E\{Arr(s_{S1}, s_{S2}, s_{P1}, s_{X1})\} = E\{Arr^{(Pack)}_{(s_{S1},s_{S2})}\} + E\left\{Arr^{(Mon)}_{(s_{P1},s_{X1})}\right\} \tag{13}$$

where:

$$E\left\{Arr^{(Pack)}_{(s_S)}\right\} = \sum_{\forall s_S} \underline{p}^{(S)}_{[s_S]} \cdot \underline{\pi}^{(S)}_{[s_S]} \tag{14}$$

$$E\{Arr^{(Mon)}_{(s_{P1},s_{X1})}\} = \\ = \sum_{s_S \in \Im^{(S)}} \sum_{s_Q=0}^{K} \sum_{s_{X1} \in \{1,2\}} \sum_{s_C \in \Im^{(C)}} \sum_{s_{P1} \in \{4\}} \sum_{s_U \in \Im^{(U)}} \underline{\pi}^{(\Sigma)}_{[s_S,s_Q,s_{X1},s_C,s_{P1},s_U]} \tag{15}$$

Now, starting from the mean number of jobs in the system calculated in (11) and the mean arrival rate calculated in (13), and applying the Little law, we can derive the mean sojourn time of a job in the system:

$$E\{T^{(\Sigma)}\} = \frac{E\{N\}}{\overline{V}} \tag{16}$$

Another important parameter representing the Governor contribution to energy saving is the marginal steady-state probability of the clock frequency process:

$$\underline{\pi}^{(C)}_{[s_C]} = \sum_{s_S \in \Im^{(S)}} \sum_{s_X \in \{0,1,2\}} \sum_{s_P=0}^{9} \sum_{s_U \in \Im^{(U)}} \sum_{s_Q=0}^{K} \underline{\pi}^{(\Sigma)}_{[s_S,s_X,s_C,s_P,s_U,s_Q]} \tag{17}$$

Let us now calculate the mean utilization of the processor:

$$E\{U\} = \sum_{s_U \in \Im^{(U)}} s_U \cdot \underline{\pi}^{(U)}_{[s_U]} \tag{18}$$

where $\underline{\pi}^{(U)}$ is the marginal steady-state probability array of the utilization process. It can be derived from the system probability array as follows:

$$\underline{\pi}^{(U)}_{[s_U]} = \sum_{s_S \in \Im^{(S)}} \sum_{s_X \in \{0,1,2\}} \sum_{s_C \in \Im^{(C)}} \sum_{s_P=0}^{9} \sum_{s_Q=0}^{K} \underline{\pi}^{(\Sigma)}_{[s_S,s_X,s_C,s_P,s_U,s_Q]} \tag{19}$$

Another important set of performance parameters is constituted by:

- the probability of the Active and Idle states:
$$P_{Active} = \pi^{(P)}_{[4]} \quad \text{and} \quad P_{Idle} = \pi^{(P)}_{[0]} \tag{20}$$
- the CPU transient states needed to move between Idle and Active (states S1, S2, S3, S8 and S9):
$$P_{I \leftrightarrow A} = \sum_{s_P \in \{1,2,3,8,9\}} \pi^{(P)}_{[s_P]} \tag{21}$$
- the probability of the CPU transient states to change the clock frequency (states S5, S6 and S7):
$$P_{FC} = \sum_{s_P=5}^{7} \pi^{(P)}_{[s_P]} \tag{22}$$

Finally, we can derive the mean power consumption:

$$\overline{C} = \sum_{s_S \in \Im^{(S)}} \sum_{s_X \in \{0,1,2\}} \sum_{s_C \in \Im^{(C)}} \sum_{s_P=0}^{9} \sum_{s_U \in \Im^{(U)}} \sum_{s_Q=0}^{K} \underline{\chi}^{(s_C)}_{[s_P]} \cdot \underline{\pi}^{(\Sigma)}_{[s_S,s_X,s_C,s_P,s_U,s_Q]} \tag{23}$$

where $\underline{\chi}^{(s_C)}_{[s_P]}$ is the power consumption associated to the processor state s_P when the clock frequency is set to s_C, as listed in Table 1.

Then, in order to quantify the gain achieved by applying the *Menu* and *Ondemand* Governors, we define the power gain as follows:

$$\Gamma^{(TOT)}_{MAX} = \frac{C^{(TOT)}_{MAX} - \overline{C}}{C^{(TOT)}_{MAX}} \cdot 100\% \tag{24}$$

where $C^{(TOT)}_{MAX}$ is the power consumed when the processor state is active working at the maximum clock frequency, i.e., $C^{(TOT)}_{MAX} = \chi^{(1)}_{[4]}$.

6. Numerical results

This section provides an overview of the numerical results obtained by an implementation of the proposed model with Matlab. By using different traffic patterns, we analyze the system performance according to different values of the OG policy threshold: $\overline{\sigma} = \{0.2, 0.5, 0.75, 0.80, 0.85, 0.90, 0.95\}$.

In order to better understand the efficiency of power management software schemes and hardware primitives, we decided to compare the obtained results with the case of a CPU without C-state optimization enabled (i.e., the CPU always stays in the C_0 state). This approach gives us the possibility of highlighting the impact of idle-active transitions on the system performance. The Markov chain of the system used as term of comparison is a subset of the proposed model (see Fig. 7), and is shown in Fig. 10.

For each considered scenario, we obtained the steady-state probabilities of the whole system by solving (10). The obtained values are consequently used to estimate the performance indexes introduced in Section 5.

According to the measurements presented in Section III, we assumed a CPU that can be switched between the maximum and the minimum P-states (i.e., clock frequencies). When the minimum P-state is applied, the average CPU processing time for a job is 40 μs. In case of the maximum P-state, the average processing time becomes 20 μs. These values were derived from the experimental measurements shown in Section 3.

Recalling that the slot duration is fixed to 4 μs (see Section 4) and by using the processing times defined above, the probability that a job is served by the processor during a single slot, according to (8), is 0.20 when the CPU works at the maximum clock frequency. Instead, when the CPU works at the minimum frequency, this probability decreases to 0.10.

The remainder of this section is organized as follows. Subsection 6.1 introduces the set of traffic patterns we applied in our analysis. Subsection 6.2 presents the performance evaluation of the system.

6.1. Traffic patterns

The model for networked devices proposed in this paper is very general regarding the traffic characterization. In this section, in

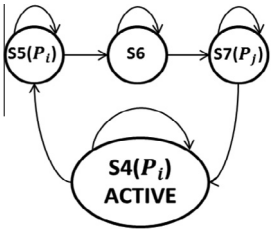


Fig. 10. Markov chain of the processor power state without C-state optimization enabled.

order to evaluate the impact of the burstiness and the average value of the input traffic on the system performance, we will consider the MMBP model described in Fig. 8, constituted by two OFF states, and two ON states: during OFF states no traffic packets arrive to the system, while during ON states the processor queue is loaded by traffic packets. Each arrived packet constitutes a job for the processor, and therefore a new job is enqueued in the processor queue at each packet arrival. More OFF and ON states can be used to capture different arrival timescales. In the case study here analyzed, we will characterize the traffic model shown in Fig. 11 with the following transition probability matrix:

$$P^{(S)} = \begin{bmatrix} p^{(S)}_{[00]} & 0 & p^{(S)}_{[02]} & p^{(S)}_{[03]} \\ 0 & p^{(S)}_{[11]} & p^{(S)}_{[12]} & p^{(S)}_{[13]} \\ p^{(S)}_{[20]} & p^{(S)}_{[21]} & p^{(S)}_{[22]} & 0 \\ p^{(S)}_{[30]} & p^{(S)}_{[31]} & 0 & p^{(S)}_{[33]} \end{bmatrix} \tag{25}$$

and the following emission probability array:

$$\underline{\gamma}^{(S)} = [0 \quad 0 \quad \Gamma^{(S)}_{ON1} \quad \Gamma^{(S)}_{ON2}] \tag{26}$$

where $\Gamma^{(S)}_{ON1}$ and $\Gamma^{(S)}_{ON2}$ are the packet arrival probabilities for the two considered ON states.However, let us stress that any traffic trace can be considered since, solving an inverse eigenvalue problem as in [29] and [30], we can derive the relative MMBP model with the same first- and second-order statistics of the given trace. In order to evaluate the performance of the networked device under different traffic conditions, we use four traffic patterns namely: High Traffic_0 (HT0), High Traffic_1 (HT1), Low Traffic_0 (LT0), and Low Traffic_1 (LT1). The considered traffic patterns are sketched in Fig. 12. The high traffic sources (Fig. 12 a and b) are characterized by an average packet rate of about 40kpps, while the low traffic sources (Fig. 12 c and d) have an average packet rate of 25kpps. All the sources are characterized by $\overline{T}_{ON} = 40\ \mu s$ $\overline{T}_{ON} = 40\ \mu s$ in both the ON1 and ON2 states.

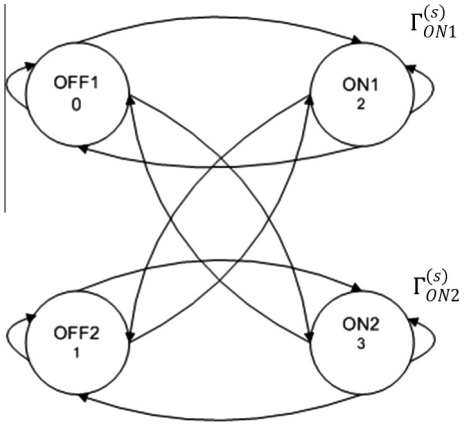


Fig. 11. State diagram of the traffic process.

The average packet rates during the ON1 and ON2 states are 250 kpps and 75 kpps, respectively. Therefore, the packet emission probability per slot in the ON1 and ON2 states are $\Gamma^{(s)}_{ON1} = 1$ and $\Gamma^{(s)}_{ON2} = 0.3$, respectively.

HT1 and LT1 are characterized by an average sojourn time longer in the OFF2 state, and shorter in OFF1, than in the case of HT0 and LT0. However, these sojourn times were set in such a way that HT1 presents the same packet emission rate of HT0, and LT1 of LT0.

Table 2 shows the values of $\overline{T}_{OFF1}$ and $\overline{T}_{OFF2}$ for all the considered traffic patterns. As we can note, although the HT0 and HT1 traffic patterns present the same mean emission value, the packet emission in HT0 is more uniform in time (see Fig. 12). The same can be said for the low traffic volume patterns: the packet emission in LT0 is more uniform than in LT1.

From the above parameters, the traffic matrix in (26) becomes:

$$\begin{bmatrix} p_{00} = \frac{(1-\Delta)}{T_{OFF1}} & 0 & p_{02} = (1-p_{00})P_{02} & p_{03} = (1-p_{00})P_{03} \\ 0 & p_{11} = \frac{(1-\Delta)}{T_{OFF2}} & p_{12} = (1-p_{11})P_{12} & p_{13} = (1-p_{11})P_{13} \\ p_{20} = (1-p_{22})P_{20} & p_{21} = (1-p_{22})P_{21} & p_{22} = \frac{(1-\Delta)}{T_{ON}} & 0 \\ p_{30} = (1-p_{33})P_{30} & p_{31} = (1-p_{33})P_{31} & 0 & p_{33} = \frac{(1-\Delta)}{T_{ON}} \end{bmatrix} \tag{27}$$

where the indices 0, 1, 2 and 3 refer to the states OFF1, OFF2, ON1 and ON2 according to Fig. 11. The factors P_{ij} represent the probability to enter the state j provided that the state i has been left. We set these factors as follows:

- $P_{02} = 1 - P_{03} = 40\%$
- $P_{12} = 1 - P_{13} = 39.3\%$
- $P_{20} = 1 - P_{21} = 90\%$
- $P_{30} = 1 - P_{31} = 90\%$

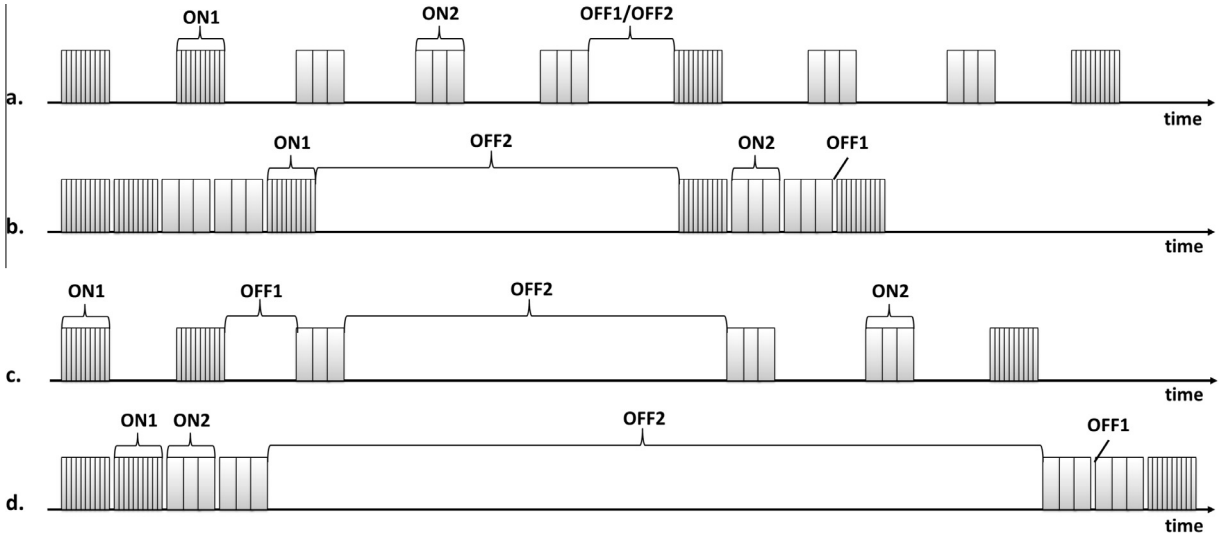


Fig. 12. Traffic patterns of the used sources: (a) HT0, (b) HT1, (c) LT0, and (d) LT1.

Table 2
Off state durations of the considered traffic patterns.

HT0	$\overline{T}_{OFF1} = \overline{T}_{OFF2} = 100us$
HT1	$\overline{T}_{OFF1} = 5us,\ \overline{T}_{OFF2} = 1000us$
LT0	$\overline{T}_{OFF1} = 100us,\ \overline{T}_{OFF2} = 1000us$
LT1	$\overline{T}_{OFF1} = 5us,\ \overline{T}_{OFF2} = 1850us$

Fig. 13 shows the steady-state probabilities for the considered traffic patterns. Because of the higher packet rate, the ON1 and ON2 probabilities are larger for the high traffic sources (i. e. HT0 and HT1), The OFF2 probability is larger for HT1 and LT1 sources, since they have longer silence periods with respect to the HT0 and LH0 sources.

6.2. Performance evaluation

Fig. 14 shows the probability of the CPU S0 state (idle/C3) for all the considered traffic patterns, and for all the considered OG policy threshold $\overline{\sigma}$. Given that the traffic burstiness level in the HT1 case is higher than in the HT0 (but the same packet emission rate between the two cases), the average time spent in S0 in the HT1 case is larger than in the HT0 one. This is caused by the fact that the increased traffic burstiness level helps in reducing the number of idle-active and active-idle transitions (and, consequently, the time spent in S1-S3 and in S8-S9). Similar considerations can be done for the LH1 and LH0 cases.

In fact, as shown in Fig. 15, most of the time in the HT0 and LT0 cases is spent in idle-active/active-idle transitions (S1,S2,S3, S8,S9). Moreover, for $\sigma = 0.85$, 0.90 or 0.95, the probabilities of the idle state (S0) and of the transient states (S1, S2, S3, S8, S9) decrease. This effect is due to the fact that increasing the threshold σ makes more probable to work at the minimum frequency. This effect is confirmed by Figs. 16 and 17, which show the steady-state probability of working at the minimum clock frequency for a low power idle capable CPU and no-capable low power idle CPU, respectively. As it can be observed, in both the cases, when the threshold σ is less than 0.85, the probability of working at the minimum frequency is almost 0. This behavior is caused by the probability of working at a certain clock frequency, which strictly depends on the monitored processor utilization level with respect to σ. In our case study, the CPU utilization estimated by the EWMA predictor results to be close to 0.85 for all the considered traffic patterns. Thus, for $\sigma < 0.85$ the governor decides to work at the maximum clock frequency for most of the time while, when σ exceeds this value, the governor starts to switch between the minimum and the maximum frequencies. As σ becomes closer to 0.95, the time spent by the CPU at the minimum frequency increases and causes the stretching of job service times and, consequently, an increase of the S4 steady-state probability.

Regarding the energy efficiency, Fig. 18 shows the power consumption gain calculated as in (24). Regarding the case with the HT0 traffic pattern and the lowest σ values, it can be observed that the power gain (with respect to the consumption in S4 of the maximum P-state) is negative. This effect derives from the additional power consumption due to the too frequent idle-active transitions.

When the threshold is set to larger values than in the previous cases (0.85, 0.90 and 0.95), the power gain for the HT0 ranges between 7% and 13%. On the one hand, this gain is due to the reduction of approximately 10% of idle-active transitions (see Fig. 15).

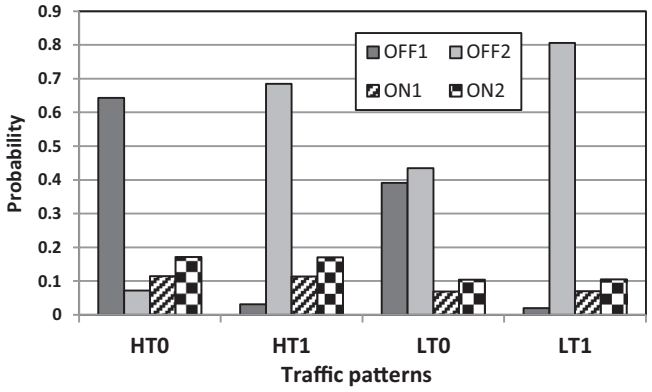


Fig. 13. Steady-state probabilities of the different traffic sources.

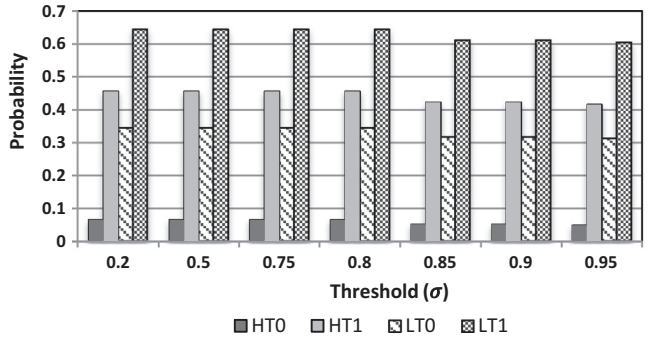


Fig. 14. Idle CPU state probability (S0).

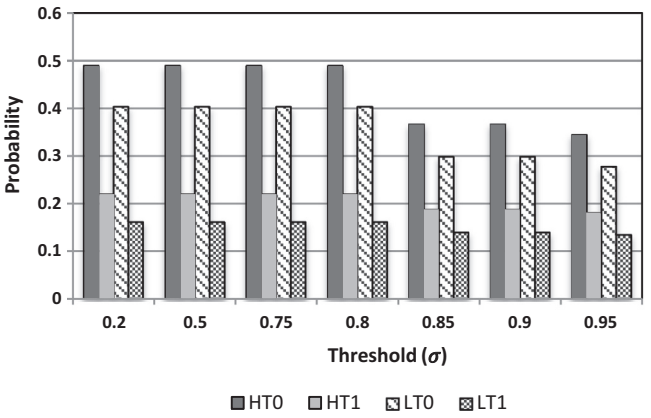


Fig. 15. Probability to be in the CPU transient states: S1, S2, S3, S8, S9.

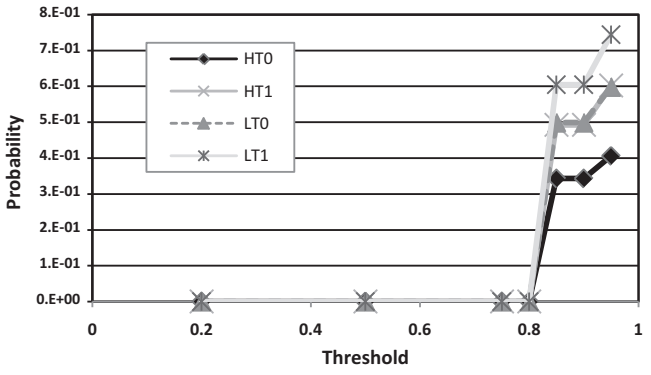


Fig. 16. Marginal steady-state probability of the minimum clock frequency for a low power idle capable CPU.

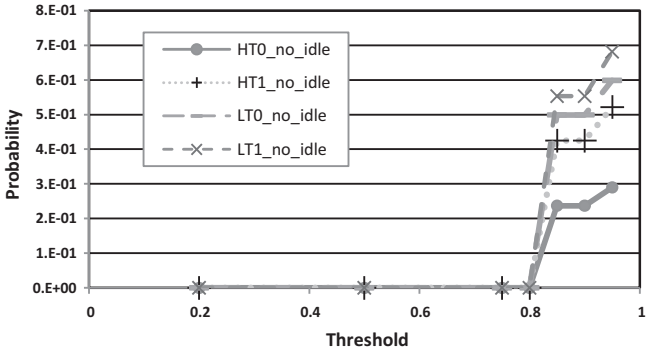


Fig. 17. Marginal steady-state probability of the minimum clock frequency for a no capable low power idle CPU.

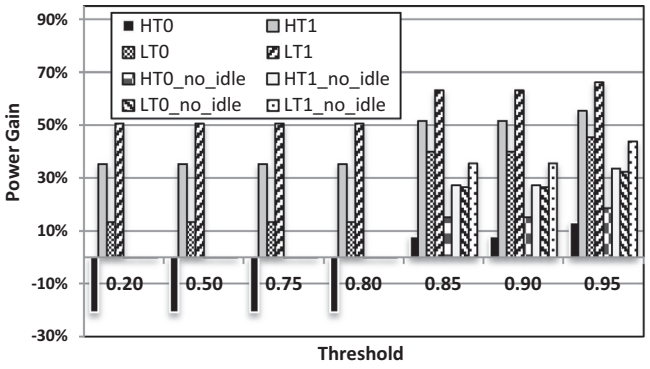


Fig. 18. Power consumption gain.

On the other hand, as shown in Fig. 16, the CPU works for 35%–40% of the time at the minimum clock frequency, reducing in this way the processing power.

As far as the HT1 traffic pattern is concerned, the increased burstiness level of incoming traffic allows the system performing a number of idle-active transitions lower than in HT0. The resulting figures exhibit a gain of approximately 35% also for the lowest considered values of σ, and in the range of 50–55% for the other threshold values (σ = 0.85,0.90 and 0.95). Similar considerations can be done for the LT0 and LT1 cases. Moreover, it is worth noting that, despite of the increased volume of incoming traffic, the HT1 traffic pattern allows the system to work even more efficiently than in the LH0 case, thanks to the much lower idle-active transition rate.

In order to have a deeper insight of the overhead and drawbacks caused by C-state optimizations, Figs. 18–20 compare the results in the cases in which transitions to C_3 are enabled or disabled (i.e., the system in Fig. 10 used as term of comparison for the original model). In these figures, the results related to the system with transitions to C_3 disabled are referred by putting "no_idle" as suffix. As regards power consumption (see Fig. 19), it remains constant at 13.3 W (which corresponds to the S4 energy consumption at the maximum P-state) for values of σ that span from 0.2 to 0.8, since neither frequency nor idle-active transitions occur; as a consequence the power gain is very close to 0% (see Fig. 18).

In case of C_3 transitions disabled, it can be noted that when higher values of the threshold are applied, the system exhibits a more efficient behavior during the processing of the HT1 and the LT1 patterns than the HT0 and LT0 patterns, similarly to case with low power idle enabled. This effect is caused by the fact that the HT1 and LT1 traffic patterns cause more bursty packet arrivals, and, consequently, more bursty jobs elaborations at the CPU. The

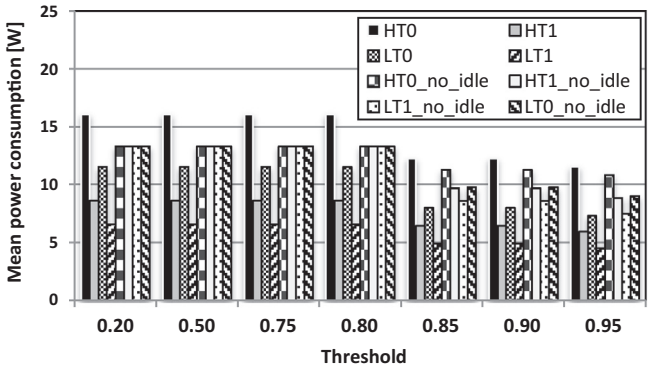
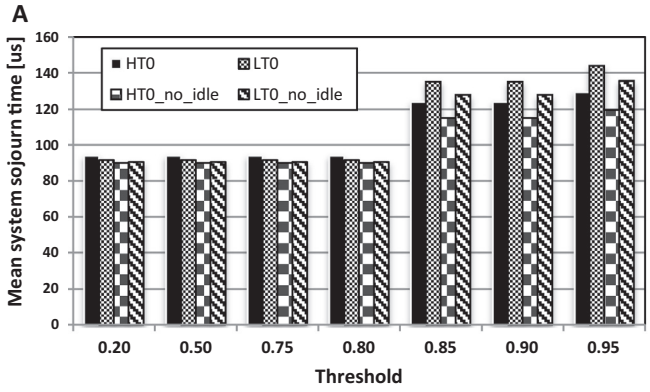
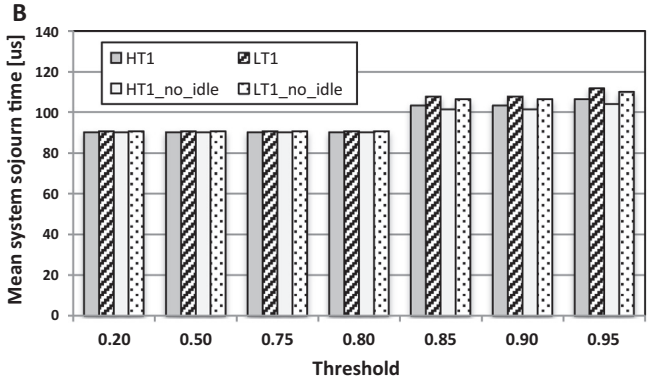


Fig. 19. Average power consumption.



(a) HT0 and LT0 patterns.



(b) HT1 and LT1 patterns.

Fig. 20. Mean system sojourn time of the jobs.

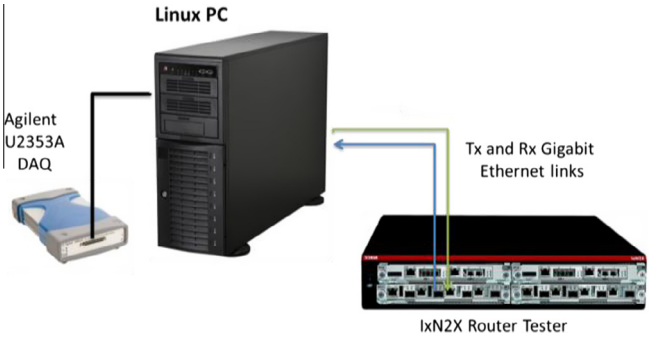


Fig. 21. Experimental test-bed used for the measurements.

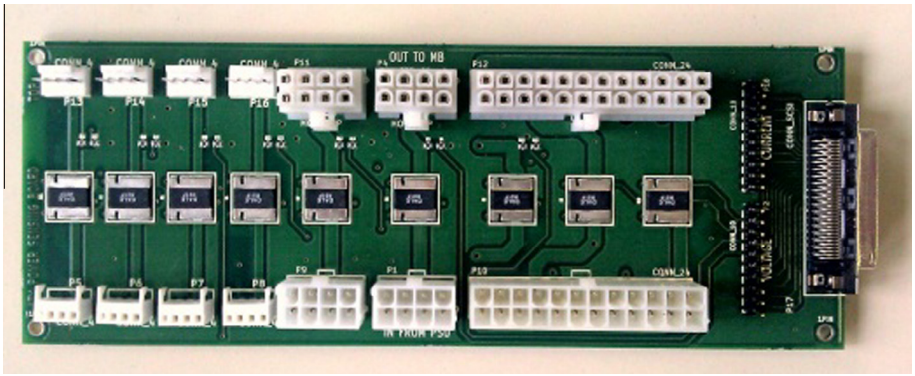


Fig. 22. The ATX riser board prototype specifically realized for this work. It can sense the current and the voltage of every rails in 24 and 8 ATX power connectors, and also the energy supplies of up to 4 fans.

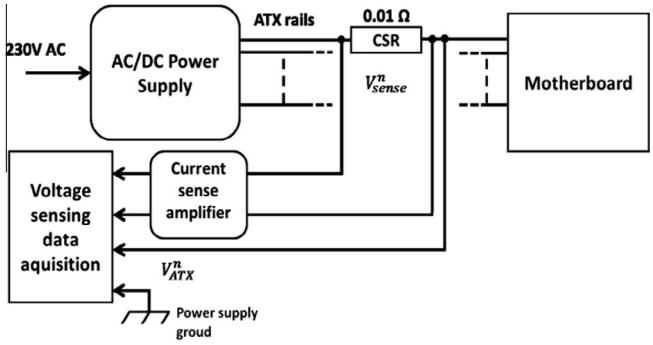


Fig. 23. High-level scheme of the riser board for ATX voltage rails.

result of these traffic patterns consists of a lower frequency of P-state transitions.

Disabling the C-state optimizations can improve the overall energy efficiency level of the system when the traffic patterns are "uniform" in the time (i.e., HT0 traffic pattern). For example, for the highest values of σ, the HT0_no_idle power gains range between 15%–18%, against 7%–13% when C_3 transitions are enabled. While, when σ is fixed at the lowest values, disabling C_3 transitions allows avoiding negative gains due to the energy consumption needed during idle-active transitions.

In order to evaluate the QoS provided by the system for the different configurations and workload patterns, we calculated the mean sojourn time of a job in the system as in (16).

Figs. 20.a and b show the mean sojourn time. By comparing these figures, we can observe that, for the highest values of σ and in the presence of the HT1 and the LT1 traffic patterns, the system provides better performance than in HT0 and LT0 (see Figs. 20.a and b). Moreover, besides the lower workload than in HT0 and HT1, in the LT0 and LT1 cases the average time spent by jobs in the system is longer (see Fig. 21).

The reason of this apparently strange behavior resides in the fact that the probability to work at the minimum clock frequency is higher in the LT0 and LT1 cases (see Fig. 16). For the highest values of the threshold, the system without transitions to the C_3 state provides slightly better performance with respect to case with the C_3 enabled, since there is no overhead in service delay times due to idle-active transitions.

Summarizing, as outlined by the results presented above, traffic patterns have a heavy impact on both energy efficiency and QoS level provided by networked devices. In detail, in presence of traffic with no bursty profiles (HT0 and LT0), a significant part of the energy consumed is wasted in idle-active transitions. In this scenarios, the model suggests that possible performance improvements can be obtained by:

- increasing the threshold σ of the Ondemand Governor, or
- disabling C-state optimization.

In general, the obtained results exhibit that setting $\sigma \geqslant 85\%$ provides good energy gains (between 13% and 66% depending on traffic pattern and workload). On the other hand, these gains come at the price of increasing the average elaboration time.

As one can expect, the largest energy gain have been obtained in presence of bursty traffic patterns, since, as previously discussed, C-states are better exploited than in non-bursty traffic scenarios. In this respect, achieved results suggest that decreasing the value of the σ parameter, on the one hand, does not significantly impact on the overall energy consumption (up to approximately 10% – see Fig. 18). On the other hand, it can play a role (by decreasing its value) to make networked device providing higher QoS levels (up to approximately 20% – see Fig. 20.b).

7. Conclusions

This paper focused on the power management of state-of-the-art networked devices (like common PCs, servers, set-top boxes, etc.) to evaluate their behavior, to model their internal dynamics and possible sources of inefficiency, and to optimize their performance and energy-efficiency.

To this purpose, we started by experimentally characterizing the power management schemes in common device platforms based on commercial off-the-shelf hardware and open-source software (i.e., common PCs/servers devices running the Linux operating system). The characterization allowed to formalize an analytical model able to accurately capture the power management dynamics at hardware (at ACPI level and beyond) and software levels (Linux governors).

The proposed model has been applied to analyze the efficiency of networked devices according to various configurations of internal parameters and incoming workload patterns.

Thanks to its intrinsic accuracy and the representation of different fine-grained details, the model provided precious information on the possible sources of inefficiency, and on how to act on policy parameters to optimize the system behavior. For instance, the model provides a good insight on the impact of different configurations of the *Ondemand* Governor (in terms of σ values) when different traffic patterns are used.

The model also outlined that, in particular but not improbable conditions, a not suitable configuration of the power management system may lead also to increase the energy consumption (energy

gain = −15% of the absorption when the CPU is fully active at the maximum frequency). On the other side, a suitable configuration of the system may lead to considerable energy gain (up to 60%).

A final consideration regards the model applicability during the normal running of a networked device. If we want to apply the achieved results at run time, the model can be solved off-line for many sets of traffic statistics (probability density function and autocorrelation function), in order to create a database that allows the networked device to decide its best configuration in real-time if the current traffic is present in the database. If the statistics of the current traffic are not present in the database, the networked device can temporarily use the configuration calculated with the statistics that are the closest to the measured ones, until in few minutes the correct configuration is derived. This new configuration is added to the database to be used also in the future, so limiting such sub-optimal behavior only to the first working period of the device.

Acknowledgments

This work was supported by the European Commission under the 7th Framework Programme (FP7) ECONET (low Energy Consumption NETworks) project, Grant Agreement no. 258454, the FIRB project Futuro in Ricerca Greening the Network (GreenNet), and the Programma Operativo Nazionale Ricerca & Competitività 2007-2013 within the project PON04a2_E – SINERGREEN – RES NOVAE –Smart Energy Master per il governo energetico del territorio.

Appendix A. Measurement tools and methodologies, and experimental test-bed

In order to characterize C-states and P-states capabilities on a Linux PC, we decided to setup an experimental test-bed, which involves in three main elements: the System Under Test (SUT – a Linux PC), the power measurement instruments, and the IP network with Ixia N2X router tester, useful to generate synthetic traffic flows for stressing the various components of the system.

Regarding the power measurement tools, since PC hardware does not usually include power probes to independently measure the power absorption of CPU, we exploited the ATX standard [30], which defines the internal power supply for general purpose computing systems.

ATX motherboards are supplied by means of a 24 pin connector, which includes among the others three main voltage rails at 3.3, 5 and 12 V, and a ground rail. The 12 V rail supplies many components of the motherboard and part of the CPU. An additional 6/8 pin ATX connector, carrying a second 12 V rail, is generally available on the motherboard to provide additional power to the CPU. In more detail, the 12 V rail on the 6/8 pin connector is commonly devoted to provide isolated power supply to the CPU Cores only. Other subcomponents of the CPU (e.g., cache, bus and DDR controllers and other internal control units) are supplied by means of the 12 V rail on the 24 pin connector.

Owing the above mentioned standard characteristics of PC supplies, we decided to monitor the CPU energy consumption by sensing the 6 pin 12 V ATX rails. To this purpose, we designed and developed a riser board for ATX power connectors (see Fig. 22), which allows putting some current and voltage probes on the available supply rails (mainly the two 12 V rails, and the 5 and 3.3 V ones). Fig. 23 shows the high-level architecture implemented by the ATX riser board. The current probes are realized by means of very small (10 mΩ) precise Current Sensing Resistors (CSR). The voltage drop V^n_{sense} across CSRs (where n denote the nth ATX voltage rail) is proportional to the incoming current. To reduce the noise and eventual cross-talk level on the measured V^n_{sense} values, we decided to closely connect the CSRs to some current-sense amplifiers. The amplified CSR voltage drops and the ATX rail voltages are finally measured by an external Data AcQuisition (DAQ) device. In our tests, we used An Agilent U2353A multifunction Data Acquisition (DAQ), which can simultaneously sample 4 channels at 250 kHz with a 16 bit resolution.

References

[1] GESI, "Smart 2020," report, DOI=http://www.smart2020.org/_assets/files/02_Smart2020Report.pdf.
[2] R. Bolla, R. Bruschi, A. Carrega, F. Davoli, D. Suino, C. Vassilakis, A. Zafeiropoulos, Cutting the energy bills of internet service providers and telecoms through power management: an impact analysis, Comput. Networks 56 (10) (July 2012) 2320–2342. Elsevier.
[3] K. Christensen, B. Nordman, R. Brown, Power management in networked devices, IEEE Comput. 37 (8) (Aug. 2004) 91–93.
[4] R. Bolla, R. Bruschi, K. Christensen, F. Cucchietti, F. Davoli, S. Singhf, The potential impact of green technologies in next generation wireline networks – is there room for energy savings optimization?, IEEE Commun Mag. 49 (8) (Aug. 2011) 80–86.
[5] Cisco Visual Networking Index: Forecast and Methodology, 2010–2015.
[6] R. Bolla, R. Bruschi, F. Davoli, F. Cucchietti, Energy efficiency in the future internet: a survey of existing approaches and trends in energy-aware fixed network infrastructures, IEEE Commun. Surv. Tutorials 13 (2) (2011) 223–244. 2nd Qr.
[7] C. Panarello, M. Ajmone Marsan, A. Lombardo, M. Mellia, M. Meo, G. Schembra, On the Intertwining between Capacity Scaling and TCP Congestion Control, in: Proceedings of the 3rd International Conference on Energy-Efficient Computing and Networking, e-Energy'12, Madrid, Spain, May 2012.
[8] S.K. Datta, C. Bonnet, N. Nikaein, Android power management: current and future trends, in: Proceedings of the 1st IEEE Workshop on Enabling Technologies for Smartphone and Internet of Things (ETSIoT), 2012, Seoul, Korea, June 2012, pp. 48–53.
[9] S. Makineni, R. Iyer, P. Sarangam, D. Newell, L. Zhao, R. Illikkal, J. Moses, Receive side coalescing for accelerating TCP/IP processing, in: Proceedings of High Performance Computing (HiPC 2006), Lecture Notes in Computer Science, Springer, vol. 4297, 2006, pp 289–300.
[10] A. Menon, W. Zwaenepoel, Optimizing TCP receive performance, in: Proceedings of USENIX 2008 Annual Technical Conference (ATC 2008), Boston, MA, USA, June 2008, pp. 85–98.
[11] W. Wu, P. Demar, M. Crawford, Sorting reordered packets with interrupt coalescing, Comput. Networks 53 (15) (Oct. 2009) 2646–2662. Elsevier.
[12] K. Christensen, P. Reviriego, B. Nordman, M. Bennett, M. Mostowfi, J.A. Maestro, IEEE 802.3az: the road to energy efficient Ethernet, IEEE Commun. Mag. 48 (11) (Nov. 2010) 50–56.
[13] R. Wang, J. Tsai, C. Maciocco, T.C. Tai, J. Wu, Reducing power consumption for mobile platforms via adaptive traffic coalescing, IEEE J. Sel. Areas Commun. 29 (8) (Sept. 2011) 1618–1629.
[14] M. Wilcox, I'll do it later: softirqs, tasklets, bottom halves, task queues, work queues and timers, in: Proceedings of the 2003 Linux Conference Australia (LCA 2003), Jan. 2003, Perth, Australia, <http://mirror.linux.org.au/linux.conf.au/2003/papers/Matthew_Wilcox/Abstract.html>.
[15] M.T. Jones, Deferrable functions, kernel tasklets, and work queues, in: Kernel APIs, Part 2, IBM DeveloperWorks, Mar. 2010, <http://www.ibm.com/developerworks/library/l-tasklets/l-tasklets-pdf.pdf>.
[16] G. Monroy, Wake-ups effect on idle power for intel's moorestown MID and smartphone platform, in: Proceedings of CELF Embedded Linux Conference, San Francisco, CA, USA, Apr. 2010, <http://elinux.org/images/0/07/Effect_of_wakeups_on_Moorestown_power.pdf>.
[17] S. Henzler, Power Management of Digital Circuits in Deep Sub-Micron CMOS Technologies, vol. 25, Springer Series in Adv. Microel., 2007, <http://www.springer.com/engineering/circuits+%26+systems/book/978-1-4020-5080-0>.
[18] ACPI Advanced Configuration & Power InterfaceWebsite, http://www.acpi.info/.
[19] Energy-Efficient Platforms – Considerations for Application Software and Services, Intel Whitepaper March 2011, <http://download.intel.com/technology/pdf/Green_Hill_Software.pdf>.
[20] Venkatesh Pallipadi, Alexey Starikovskiy, The Ondemand Governor IntelOpen Source Technology Center, in: Linux Symposium 2006.
[21] Venkatesh Pallipadi, Shaohua Li, Adam Belay, cpuidle—Do nothing, efficiently..., Intel Open Source Technology Center, in: Linux Symposium 2007.
[22] Intel® Core™ i5 Processor Datasheet, http://download.intel.com/design/processor/datashts/322164.pdf.
[23] R. Bolla, R. Bruschi, P. Lago, The hidden cost of network low power idle, in: Proceedings of the 2013 IEEE International Conference on Communications (IEEE ICC 2013), Budapest, Hungary, June 2013.
[24] R. Bolla, R. Bruschi, O.M. Jaramillo Ortiz, P. Lago, The energy consumption of TCP, in: Proceedings of 3rd ACM/IEEE International Conference on Future Energy Systems (e-Energy 2013), Berkeley, CA, USA, May 2013.

[25] P. Salvador, A. Pacheco, R. Valadas, Modeling IP traffic: joint characterization of packet arrivals and packet sizes using BMAPs, Comput. Networking 44 (Feb. 2004) 335–352.
[26] A. Klemm, C. Lindemann, M. Lohmann, Modeling IP traffic using the batch markovian arrival process, Perform. Eval. 54 (Oct. 2003) 149–173.
[27] Yonghwan Kim, San-qi Li, Timescale of interest in traffic measurement for link bandwidth allocation design, in: Proceedings of IEEE INFOCOM '96, IEEE, San Francisco, CA, USA, March 24–28, 1996, pp. 738–748.
[28] San-qi Li, C.-L. Hwang, Queue response to input correlation functions: discrete spectral analysis, IEEE/ACM Trans. Networking 1 (5) (1995).
[29] A. Lombardo, G. Morabito, G. Schembra, An accurate and treatable markov model of mpeg-video traffic, in: Proceedings of IEEE Infocom '98. San Francisco, CA, USA, March 29 – April 2, 1998.
[30] The ATX specification 2.1, http://www.formfactors.org/developer/ specs/atx2_1.pdf.