An Experimental Evaluation of the TCP Energy Consumption

Raffaele Bolla, *Member, IEEE*, Roberto Bruschi, *Senior Member, IEEE*,
Olga M. Jaramillo Ortiz, and Paolo Lago

***Abstract*—The objective of this paper is to determine how the incoming and outgoing traffic patterns provided by the TCP impact the energy efficiency of a networked device under realistic scenarios. We set up a complex test-bed that allowed us to perform a large number of measurements of energy- and network-performance indexes on a general-purpose computing system with power management capabilities, and the Linux operating system. The performed measurements gave us a chance not only 1) to provide a complete energy profiling of TCP connections, but also 2) to analyze the role of underlying network protocols in detail, and 3) to identify specific situations where current network protocols may trigger high inefficiencies in networked hosts. Based on the obtained results, we also conducted further experiments for evaluating techniques for reducing the energy consumption induced by TCP data transfers under different scenarios.**

***Index Terms*—Measurement, Power Management, TCP.**

I. Introduction

In recent years, several studies have identified energy efficiency as one of the key aspects that may constrain the evolution of the Internet technology and its wide adoption as public telecommunications infrastructure. Estimates and projections from telecom operators and third-parties [1] clearly suggest that the energy for supplying wire–line and wireless networking infrastructures is rapidly becoming no more sustainable, if no radical changes on Internet technology design will be undertaken. Moreover, this figure becomes even more impressive if we consider not only networking devices (e.g., switches, etc.) inside telecoms and home networks, but also the "networked" ones [2]. The number of Internet-connected devices is forecasted to explode to over 15 billion by 2015—twice the world's population [3]. Moreover, Cisco forecasts that by 2015 Internet traffic will get very close to the impressive threshold of one zettabyte a year, reaching approximately 966 exabytes a year.

Starting from this scenario, the sustainability of the Internet firmly and clearly relies on the efficiency of technologies and of protocols working at the network edge, and more precisely inside networked devices [4]. To build a more sustainable Internet, on one hand, networked devices should include advanced power management schemes, which may allow certain proportionality between their energy consumption and actual workload. On the other hand, the network protocol stack and its implementation (often realized at the software level) need to be optimized as much as possible for best fitting the dynamics and features of power management schemes. It is worth noting that even those optimization solutions that assure just only slight 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 the performance of the Internet traffic and, therefore, the device energy savings. In this respect, we decided to focus on the Transmission Control Protocol (TCP) since we firmly believe that its optimization could have a key role in the future green Internet. This decision arises from a double–faced consideration: *(i)* the TCP protocol is and will be present and widely used in networked devices (TCP still carries more than 80% of the Internet traffic [5]); *(ii)* TCP implements the flow and congestion control mechanisms, which directly manage the largest part of the packet–level traffic dynamics.

Packet–level dynamics are a critical aspect, especially because the processing of incoming and outgoing packets is a non–negligible part of networked device workload. Moreover, packet–level time scales roughly coincide with (or at least are very close to) the ones of hardware power management primitives [6]. Different traffic arrival patterns may trigger power management mechanisms at the hardware level, and result in quite different energy efficiency levels [7].

In this paper, our main objective is to evaluate experimentally which is the energy efficiency of a networked device when receiving or transmitting data through TCP connections, and to understand which are the key aspects and parameters that can play a significant role. At this purpose, we created specific probes and used sophisticated hardware and software instrumentation, which allowed us to set up a complex and complete test-bed for performing a large number of white–and black–box measurements on energy and network performance indexes on a general purpose computing system with power management capabilities, and the Linux operating system. The performed measurements allowed us to deeply understand how internal software and hardware dynamics can be affected by external TCP operating conditions, and how these dynamics can influence the energy consumption of the system. Moreover, software profiling has been performed in order to precisely understand the computational complexity of each operating system module

Manuscript received March 19, 2015; revised September 3, 2015; accepted September 3, 2015. Date of publication September 25, 2015; date of current version November 16, 2015.

R. Bolla, O. M. J. Ortiz, and P. Lago are with the Department of Electrical, Electronic, and Telecommunications Engineering, and Naval Architecture (DITEN), University of Genoa, 16145 Genoa, Italy (e-mail: raffaele.bolla@unige.it; olga.jaramillo@unige.it; paolo@tnt-lab.unige.it).

R. Bruschi is with the National Inter-University Consortium for Telecommunications (CNIT), Research Unit, University of Genoa, 16145 Genoa, Italy (e-mail: roberto.bruschi@cnit.it).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2015.2482038

and function involved in TCP transmission and reception operations in a wide spectrum of testing scenarios. This approach allowed us to identify specific network situations where TCP may trigger high inefficiencies in networked hosts, and to propose some early mitigation approaches that can be realized by enabling existing hardware offloading mechanisms. It is worth noting that the choice of using a general–purpose computing system (i.e., a workstation) does not limit the scope of the results and the analysis performed in this paper. On the contrary, our results can be rather generalized not only to servers and data–centers where a large part of CPUs and operating systems are well-known to be very close to the ones used in the selected workstation, but also to a large part of Android mobile devices [8], which share large share of software with Linux (especially in the power management and in the network side), and use CPUs with power management capabilities very similar to defined to the ACPI standard [9]. This paper is an extension of a preliminary work appeared in [10]. Despite a more comprehensive analysis and introduction to the problem with respect to this preliminary contribution, this paper provides an extended experimental analysis by considering multiple TCP connections and different TCP variants (i.e., Reno, Vegas, and Cubic). It also includes a discussion on methods that can be applied for reducing the energy consumption induced by TCP connections. The achieved results obtained in a wide spectrum of testing scenarios may constitute a solid basis for driving future research towards energy consumption and TCP modeling, more energy–efficient TCP optimizations and networked device architectures. The paper is organized as follows. Section II discusses related work. The power management primitives usually available in general-purpose PCs, and that are available in many consumer electronic devices are described in Section III. Section IV describes the test-bed and all related tools and components. The obtained results and their analysis are in Section V. The evaluation of the selected techniques to reduce the energy consumption of the networked device is discussed in Section VI. Finally, the conclusions and lessons learned are drawn in Section VII.

II. Related Work

Zorzi and Rao were the first to propose a study on the energy efficiency of the TCP protocol [11]. They analyzed the energy consumption of various TCP versions (Old Tahoe, Tahoe, Reno and New Reno). A similar work was conducted in [12], where the authors demonstrated that the energy/throughput tradeoffs of various TCP versions are similar. Other contributions analyze the computational energy cost of TCP [7]. The authors presented a detailed breakdown study of the energy cost of different TCP functions. Their experimental results showed that the TCP processing cost accounts for 15%, of which 20–30% is used for the TCP checksum. Techniques such as eliminating the user–to–kernel memory copy allowed the reduction of TCP processing cost by 20–30%. Conducting several simulation tests, researchers in [13] demonstrated that is possible to achieve energy savings and maintaining good network performances by choosing the right TCP version. In [14], the authors proposed a "Green TCP/IP" protocol extension, whereby it is possible to maintain "active" connections between clients and servers when client PCs are sleeping. In [15], the authors examined the impact of TCP acknowledgment, demonstrating significant energy savings can be achieved by reducing the ACK frequency. TCP Incast is one of the performance problems present in modern data center. The survey in [16] shows a detailed study of some solutions to mitigate this problem and to improve the energy efficiency of data centers. Some solutions have been proposed for energy and performance optimization of TCP. A rate adaptation scheme was introduced in [17], where, using a small buffer and determining the data rate based on queue length, authors increased the TCP throughput and obtained a reduction in energy consumption of 14–36%. In [18] the authors introduce a periodic early detection scheme, which stabilizes the performance of TCP network applications by reducing the buffer sizes. Other studies evaluated the impact of moving the TCP processing to a dedicated processor. In [19], the authors proposed to eliminate interrupts by simply polling the NIC. Similar studies were conducted in [20] where the authors showed greater performance when dedicating one processor to packet processing. The purpose of our work is in line with the concern of the research community of knowing the energy consumption of the TCP protocol, but from another point of view. Our interest is to understand how the different network conditions (different end–to–end bandwidth, end-to-end delays, and additional packet loss probabilities) affect the behavior of the TCP traffic triggering high inefficiencies in networked hosts. Our work also demonstrates that significant energy savings can be achieved applying some already present techniques such as Packet Coalescing and TCP Segmentation Offloading.

III. Anatomy of PC Power Management

In general–purpose computing systems (like PCs and servers) and similar devices (e.g., mobile devices like smartphones, tablets, etc. based on Android [8]), the power management system is usually designed across three layers, namely hardware, firmware, and software. 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, 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 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 [21] framework, which provides a standard interface between hardware–dependent power saving techniques and software layers. This standard interface completely hides the hardware (HW) internal techniques to reduce power consumption, which may differ depending on the processor, to the Operating System (OS) and Software (SW) applications.

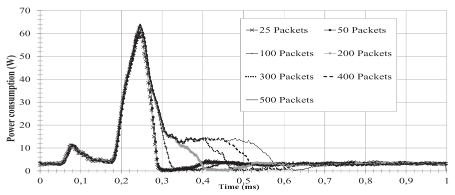


Fig. 1. Energy consumption of the CPU when receiving a burst of a variable number of packets. These results have been obtained with the ATX riser board in subsection IV-C.

Focusing on processors' power saving primitives, the ACPI standard introduces two main different power saving mechanisms, namely performance (P–) and power (C–) states, which can be independently employed and tuned for the largest part of today's processors. However, due to the effectiveness and the simplicity of usage, the power management of modern OSs tends mainly to rely on C-states. Regarding the C–states, the C_0 is the active power state where the CPU executes instructions, while C_1 through C_n states corresponds to sleeping or idle modes, where the processor consumes less power and dissipates less heat. As the sleeping state $(C_1, \ldots, C_n)$ becomes deeper, the transition between the active and the sleeping state (and vice versa) requires longer time and more energy [10]. 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, the Level 1 and 2 data caches 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. C–states are very effective primitives (more than P–states) since they allow quickly to shut off some internal CPU components, avoiding intrinsic energy wasting due to leakage current. However, when components are turned on, they need a no negligible additional start–up power. For instance, Fig. 1 shows the instantaneous power consumption of an Intel Core i5 processor when receiving IP packet bursts with different lengths. Here, the first packet is signaled to the CPU after approximately 50 s. Then, at about 250 s, we have a first HW start-up consumption due to the re–activation of CPU memory controllers, and then a huge spike of power due to the wake–up of CPU cores. Finally, the period where the CPU really processes the incoming packets is clearly visible, and changing according to the number of packets in the burst.

C–state transitions are mainly driven by the Operating System (OS) and device I/O operations. When the CPU finishes serving its job backlog, the scheduler of the OS may decide to enter a low power idle mode $(C_1, \ldots, C_n)$. In such states, the CPU can wake only by some scheduled activities, or by an interrupt coming from an I/O component (like a NIC). In this respect, it is worth noting that network traffic dynamics (and then the TCP itself) can heavily influence interrupt generation from I/O hardware, and then the effectiveness of C–state transitions. In the C_0 state, the ACPI allows the processor performance to be tuned by means of P–states. P–states modify the operating energy configuration by altering the working frequency and/or voltage, or throttling the clock of the processor. Thus, using P–states, a CPU can provide different power consumption and performance when active. Given issues in silicon electrical stability, the transition time between different P–states is generally very slow (more than 1 *ms*), and, consequently, their automatic tuning is performed within time scales much larger than network dynamics.

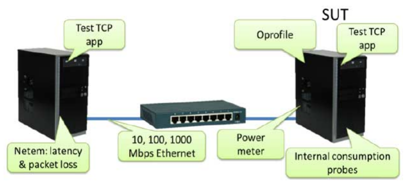

SUT
Test TCP app
Oprofile
Test TCP app
Netem: latency & packet loss
10, 100, 1000 Mbps Ethernet
Power meter
Internal consumption probes

Fig. 2. Experimental test-bed.

IV. The Test–Bed and the Measurement Methodology

Energy consumption and network performance of networked devices depend on many factors. For this reason, estimating the energy consumption due to the networking protocol stack, and especially of the TCP, is not a simple task. As first, the energy consumption of a device strictly depends on its hardware design, and on the available power management primitives. Moreover, the efficiency of the software layer (on both applications and the operating system) may have a significant role in workload dynamics and network performance. Under this line of reasoning, an experimental evaluation performed only with "black box" performance indexes can clearly results in a noisy data collection, whose interpretation and generalization may be hard, or even impossible. A complete and general understanding of TCP impact on energy consumption can be clearly achieved by mapping the measured aggregated/external performance to the real sources (e.g., at HW or SW levels, due to features of network protocols, etc.). This mapping can be accomplished only through a careful characterization of the internal dynamics of networked devices. This approach may certainly allow generalizing the obtained results also to other hardware architecture (e.g., smartphones and high-end servers), operating systems, etc. Thus, to collect the largest set of internal performance indexes we set up a benchmarking environment (depicted in Fig. 2). All the used hardware devices, software and measurement tools were selected to ease the collection "white-box" measures, and to provide enough flexibility for realizing different testing scenarios.

Five main HW platforms and measurement tools compose the test-bed:

- The *System Under Test* (SUT): a commercial off-the-shelf PC, based on the Intel i5 processor, and running the Debian Linux operating system. As described in details in the following subsection, the SUT has been equipped with some Software/Hardware (SW/HW) probes to perform white-box measurements (see Fig. 3).
- A high-end server: used for TCP traffic generation, network performance indexes collection, and network emulation.

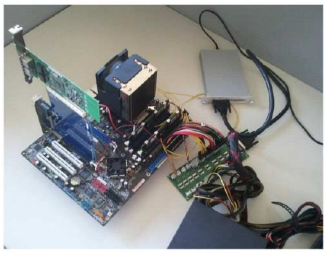


Fig. 3. The SUT equipped with PCI-e, DDR3 and ATX risers connected to the DAQ system.

- A Gigabit Ethernet switch: used to separate the Ethernet links between the SUT and the server, and to provide Ethernet link speed changes only at the server, without involving the SUT (that is kept connected at 1 Gbps).
- An AC watt-meter: to measure the entire energy consumption of the SUT.
- A multi-channel Data AcQuisition (DAQ): used to collect a high number of DC power consumption probes, placed inside the SUT as described in subsection IV-C.

The testing methodology is simple. Instead of using real world workloads, we have developed an application for generating (server) and receiving (client) TCP traffic. This choice gives us a more controlled test environment and helps us easily understand the energy and network performance from TCP/IP packet processing. Such application continuously generates a number of TCP connections (upon completion of TCP connection, a new one with the same features is started), which are used to move traffic unidirectionally (from the "sender" to the "receiver"). Some network emulation tools are used to realize different network scenarios, by reducing the transmission capacity between the sender and the receiver, to introduce packet loss probability, and to increase the delay between the hosts. At the same time, all the SW and HW probes collect samples on energy consumption and network performance. All the tests have been repeated 20 times, and their duration was fixed to 200 times the average connection lifetime.

The rest of this section is devoted to introducing the test-bed elements above mentioned into details.

A. The System Under Test

The measurements were performed on a Linux workstation equipped with an Intel i5-760 processor running at 2.68 GHz, with 4 physical cores and a maximum power consumption of 95 W [22]. The workstation is equipped with 2 GB of DDR3 RAM, and an Intel PRO Gigabit Ethernet adapter. The Operating System (OS) is the Linux Debian 5.0.6, and the kernel version is the "vanilla" 3.4, which supports symmetric multi-processor (SMP). To collect "clean" internal measurements and avoiding background dynamics from other SW services, the OS was configured with the essential SW packages to run the tests: no application strictly needed for our scopes was installed. All the configurations related to the networking stack and of the operative system have been maintained as suggested in the Debian default settings. For instance, the TCP receive window has been kept fixed to 8 KB. Only socket resource recycling option has been disabled, since, in the presence of cyclic TCP connection generations between the same pair of hosts (as in the next subsection), it can introduce not realistic behavior in the OS.

B. Traffic Generation and Network Emulation

As previously sketched, we implemented a simple and very light-weight testing application, named "Tcptest" [23], to generate and to receive TCP connections. With respect to other well–known applications for generating TCP flows (e.g., TTCP, iPerf among the others), Tcptest provides more flexible flow generation (especially in the presence of multiple simultaneous connections), avoiding overhead operations (e.g., disk access, etc.). Tcptest can be used both as client (receiver) and server (sender) of connections. Once a connection is established, Tcptest sends a desired number of segments at the maximum MTU. When the client successfully receives the last segment, the TCP connection is immediately closed. The segments of all the generated connections contain replicas of a same pre–allocated pattern. The amount of data sent, the number of simultaneous TCP connections, and other options are configurable through a simple command line interface. Tcptest is able to work in two modes: *(i)* One-shot: the chosen number of TCP connections is generated only once; *(ii)* Continuous: the chosen number of TCP connections is generated in a persistent and cyclic way. In both the modes above, Tcptest provides the average duration time of connections and other monitoring parameters made available by the Linux kernel. The continuous mode also provides the number of running and completed connections every second. For our purposes, except for some preliminary tests, we used Tcptest in continuous mode. Regarding the network emulation, our objective was to act on the main network performance indexes that can heavily influence the TCP behavior: the available bandwidth capacity, the end-to-end delay, the packet loss probability. To emulate different bandwidth capacities, we disabled the Ethernet auto–negotiation at the server NIC and made it working at 1 Gbps, 100 Mbps, and 10 Mbps. We can note how 10 Mbps is a very interesting bandwidth capacity because it roughly corresponds to the speed of common broadband residential accesses (e.g., ADSL), today provided by many Telecom operators. Regarding packet losses and end-to-end delay, we used the NetEm module [24], available in latest releases of the Netfilter framework in the Linux kernel. The NetEm module was activated on the server for both incoming and outgoing traffic.

C. Power Consumption Probes

In order to provide a complete characterization of the energy absorption dynamics, we decided to collect two main kinds of data: *(i)* the energy absorption of the entire SUT on the AC plug; *(ii)* the energy absorption of the most significant HW subcomponents of the SUT (e.g., CPU, RAM, PCI cards, etc.). The measures on the AC plug were directly collected using a Raritan Dominion PX-5297 wattmeter [25] while the measures on the internal subcomponents required a particular effort. In fact, PC

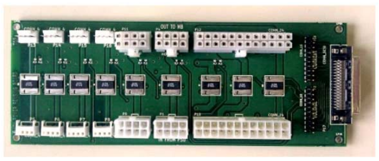


Fig. 4. The ATX riser board prototype specifically realized for this work. It can sense the current and the voltage of every rails in 24 and 8 ATX power connectors, and also the energy supplies of up to 4 fans.

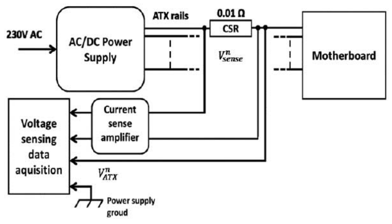


Fig. 5. High-level scheme of the riser board for ATX voltage rails.

HW does not usually include power probes, and the power supply to HW subcomponents is carried within multiple heterogeneous interfaces (e.g., different sockets for CPUs, DIMM and SO-DIMM slots for memories, PCI-e slots for network interface cards, etc.). To measure the absorption of HW subcomponents in an independent fashion, we used some special bus risers [26] [27] equipped with current and voltage measurement probes. Unfortunately, this approach cannot be applied to all the physical interfaces in a PC, due to issues related to the much-miniaturized physical connector dimensions, to the stability of electrical signals, etc. This last case applies for sure to CPU sockets. However, to measure the energy consumption of CPU, we exploited the ATX standard [28] that defines the internal power supply for general–purpose computing systems. ATX motherboards are supplied using a 24–pin connector, which includes among the others three main voltage rails at 3.3, 5 and 12 V, and 1 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 available on the motherboard to provide additional power to the CPU. The 12 V rail on the 6/8 pin connector is commonly devoted to providing isolated power supply to the CPU cores only. Other subcomponents of the CPU (e.g., cache, bus and DDR controllers, etc.) are supplied using the 12 V rail on the 24-pin connector [28]. Owing the above-mentioned ATX characteristics, we monitored the CPU consumption by sensing both the 12 V ATX rails. We designed and developed a riser board for ATX power connectors (see Fig. 4), 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. 5 shows the high–level architecture of such riser board. The current probes are realized with tiny ($10\,m\Omega$) Current Sensing Resistors (CSR). The voltage drop V^n_{sense} across its CSR (where n denote the n^{th} ATX voltage rail) is proportional to the incoming current. To reduce the noise and eventual cross–talk on the measured V^n_{sense} values, we decided to connect the CSRs to current–sense amplifiers closely. An external DAQ device finally measures the amplified CSR voltage drops and the ATX rail voltages. In our tests, we used An Agilent U2356A DAQ [29], which can sample 64 channels at 500 $K\ sample/s$ with a 16–bit resolution.

D. Software Probes

The internal SW measurements have been carried out by using a particular tool (called profiler). This software tool can trace the percentage of CPU utilization for each source-code function of any SW application, service, kernel activity, and module running on the PC. This powerful tool obviously permitted us deeply to analyze the role and the computational efficiency of all the SW components, and it gave the chance of breaking down the energy consumption per SW functionality. From a general point of view, many tools for SW profiling may perturb the system performance since they may require a relevant computational effort. We have experimentally verified that one of the best is Oprofile [30], an open source tool that realizes a continuous monitoring of system dynamics with a frequent and quite regular sampling of CPU HW registers. Oprofile allows the fine–grained evaluation of the CPU utilization of both Linux user- and kernel-space with a very low computational overhead. It is worth noting that Oprofile can collect the values of HW registers only during CPU activity periods: so that, when the CPU enters into low power idle modes (C–states—see Section III), no samples are collected. Thus, for evaluating the real CPU time utilization of SW functionalities, we also measured the CPU activity percentage.

E. Network Performance Probes

For evaluating TCP and network performance indexes, we enabled the well-known "tcpdump" packet sniffer on the server to collect entire traffic traces of performed testing sessions. Collected traffic traces have been analysed offline by means of the Linux "tcptrace" utility in order to determine the number of packet losses, out-of-order packets, TCP retransmission due to timeout and fast retransmit, etc.

V. Experimental Results

As previously sketched, TCP traffic is well known to provide different behaviors depending on parameters like end-to-end bandwidth, delay and packet loss probability. We decided to perform a number of test sessions under different simple network scenarios, which may represent some common cases in the today's Internet.

As shown in Table I, the selected scenarios provide different end–to–end bandwidth (10, 100, and 1000 Mbps), end–to–end delays, and "additional" packet loss probabilities.[1] In more detail, the 10 Mbps bandwidth bottlenecks can describe typical situations in today's residential broadband network accesses (e.g., DSL); the 100 Mbps and the 1 Gbps cases in enterprise networks and data centers. In order to be able deeply to analyze the experimental results, and precisely to decompose the consumption sources in such a complex system, the

[1] Additional to the ones induced by the TCP congestion control.

TABLE I
TESTING SCENARIOS

Testing Scenario	Bottleneck bandwidth [Mbps]	Additional packet loss probability [%]	Additional delay [ms]
a	1000	0	0
b	1000	0.5	0
c	1000	0	10
d	10	0	0
e	100	0	0
f	10	0	10
g	100	0	10

testing scenarios have to remain as simple and deterministic as possible, avoiding the introduction of stochastic "disturbs" on network parameters (e.g., on bandwidth and delay, etc.). To this end, we also disabled the power management policies provided by the Linux OS, which dynamically adjust the operating P–state of the CPU according to the average CPU utilization (usually monitored every $100-200\,ms$). In every testing scenario, we performed tests by changing the length and the number of TCP connections. The chosen length values correspond to 10 and 10000 segments of the maximum TCP transfer unit, equal to 1460 B, the number of TCP connections correspond to 1, 10 and 100 and by setting different P–states. However, due to the lack of room, we decided to report only the results obtained when the CPU is working at the maximum clock frequency. The results obtained with smaller working frequency differ from the ones here reported by a general increase of the CPU active time, but not significant energy savings. Each test has been repeated for assuring a result confidence of 3% on the average measured values. Several versions of the TCP protocol have been proposed for improving the performance in terms of throughput and stability of the congestion control mechanisms. In this respect, some tests were also performed for different TCP variants to evaluate the performance effects on the energy consumption. The selected TCP versions are Reno, Vegas and Cubic.[2] Moreover, we used the TCP connection for unidirectional data transfers, not only for the sake of simplicity but also because it is a standard approach in many applications. The rest of this section is organized as follows. A first breakdown of the HW energy consumption sources of the SUT is in subsection V.A. In subsection V.B, the analysis of TCP performance and energy consumption is discussed. In subsection V.C, measures regarding the dynamics and the role of the various SW elements are shown. Finally, subsection V.D goes further into the details of TCP profiling trying to break down the CPU utilization on a per-function basis.

A. Identify the Energy Proportional HW Components

The first measurements we obtained aim at evaluating the power consumptions of various SUT HW components, and their proportionality to the actual workload. The power consumption of these internal elements was measured both when the system is idle (turned on, but performing no operations), and

[2]The TCP congestion control available in the Linux kernel.

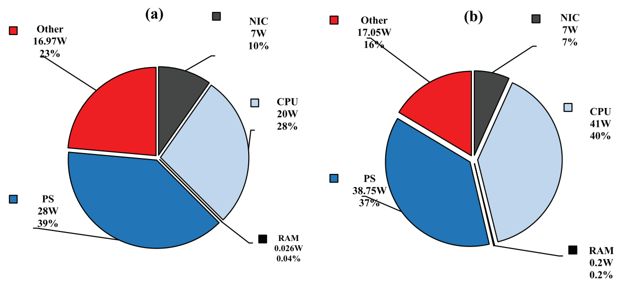


Fig. 6. Energy consumption of the different internal HW components of the SUT during idle (a) and active (b) periods (RAM: memory, NIC: network interface card; PS: AC/DC power supply; Other: all the other components in the SUT, from motherboard chips to disks).

when one core is active and performing operations at the maximum speed. The obtained results are in Fig. 6, and, as expected, it outlines how the CPU is the most energy-hungry component, exhibiting also an explicit dependency on the workload since it passes from $20\,W$ (28% of the overall consumption) in idle mode to $41\,W$ (40% of the overall consumption) when active. Except for the AC/DC power supply (whose efficiency depends on the electric load of internal SUT components), all the other components show negligible energy absorption variations (below $200\,mW$). For this reason, in following we will focus only on the energy consumption of the CPU, omitting the contributions of the other HW components.

B. The TCP Power Consumption and Performance

This subsection presents the network and energy performance measures of the TCP behavior in the testing scenarios in Table I, for all indicated TCP versions. Regarding network performance, the average lifetimes of the TCP connections are reported in Fig. 7, and the frequency of segment losses due to fast–retransmits and time–out expiration are in Fig. 8. From Fig. 7, we can note that the behavior of the lifetime of TCP connections is somewhat dependent on the TCP congestion control; this effect is well known to be due to the differences in the congestion window algorithm, which provide sensibly different performance in exploiting the capacity available at the network path [31]. We can observe that the average connection lifetime with Reno is higher than Vegas and Cubic because Reno suffers performance problems when multiple packets are dropped from a window of data. Moreover, Reno produces a periodic oscillation in the window size due to the constant updating of the window. This oscillation affects the round trip delay of the packets, resulting in an inefficient use of the available bandwidth. The Cubic provides better performance since it keeps the window growth rate independent of RTT, and, therefore, the throughput is maintained high also in the presence of large values of the bandwidth-delay product [32].

As one can expect, Fig. 7 highlights also how the lifetime of TCP connections heavily depends on the bottleneck bandwidth, the RTT, packet losses and the number of active TCP connections. In this respect, the addition of 0.5% packet loss probability causes a non-negligible increase of the connection lifetime (see the scenario *b* vs. the scenario *a*). Also the addition of the $10\,ms$ delay in the RTT introduces even more evident

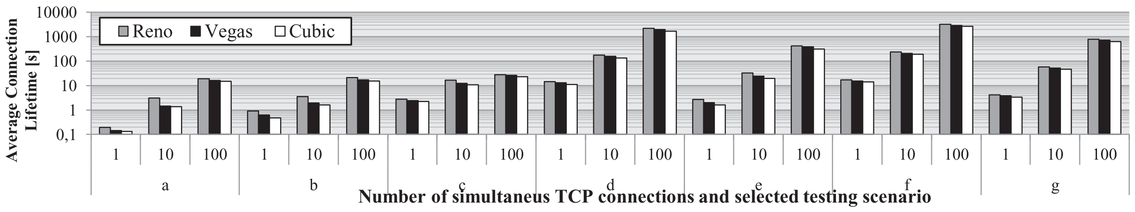


Fig. 7. Average lifetimes of multiple TCP connections with 10 Kpkt, when the SUT acts as “client” and “server” for Reno, Vegas and Cubic TCP versions.

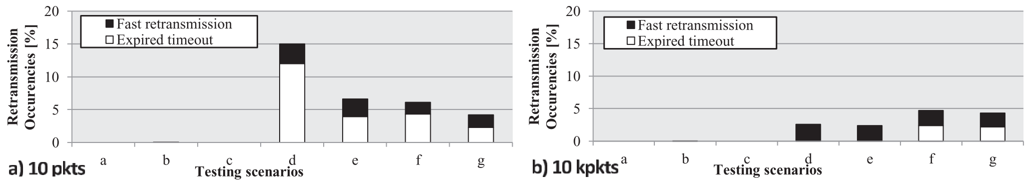


Fig. 8. Frequency of TCP fast retransmits and timeout expiration (the SUT is acting as “sender”) for the case with 10 packets (a) and the one with 10000 packets, both in the case of a single TCP connection.

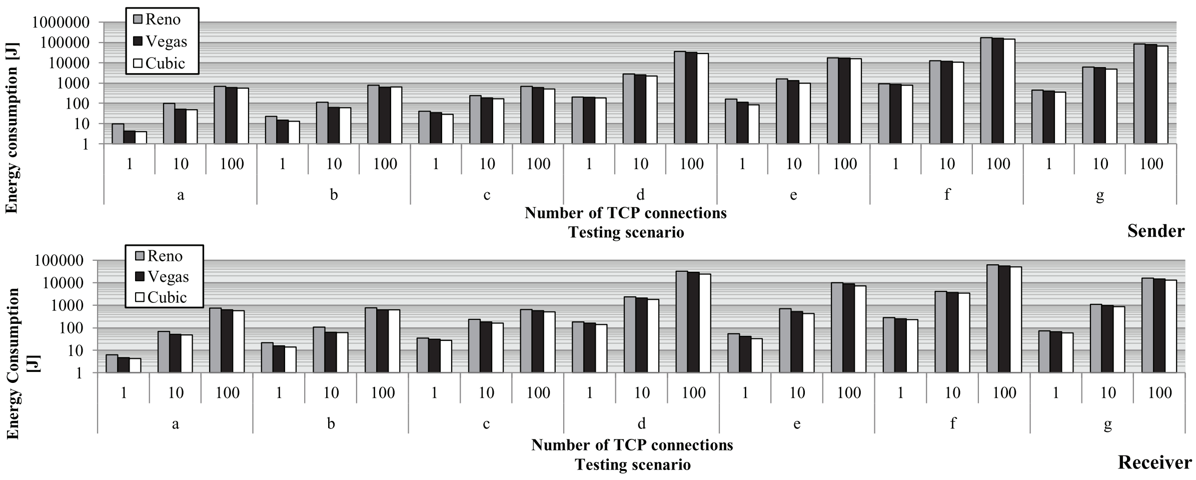


Fig. 9. Energy Consumption of TCP connections with 10000 pkt at the SUT when acting as “sender” and “receiver” for Reno, Vegas and Cubic TCP versions.

performance decays than in the previous case (see scenarios c, f, and g with respect to the a, d, and e ones). Moreover, the additional delay results to have an impact proportional to the bottleneck bandwidth (i.e., it is proportionally more evident for connection crossing the 1 Gbps bottleneck than in the 10 Mbps case). Finally, as one can expect, the bottleneck speed has also a key role in the TCP performance, due to both the increase of transmission times of segments and inefficiency of the congestion control as above described. As shown in Fig. 8 the number of retransmissions, in the case of a single TCP connection using the Cubic version, due to the reception of duplicate Acknowledgements (ACKs) or to timeout expirations, increases significantly in the 10 and 100 Mbps cases. This increase is a clear sign of the central role of TCP congestion control in the scenarios d, e, f, g. However, all the collected results are in line with many studies [32] on the TCP performance proposed in the past years.

Figs. 9 and 10 report the energy consumed by connections in all the considered testing scenarios for all indicated TCP versions. In more detail, we measured the average power consumption of the SUT during TCP tests (the Tcptest application has been made running in “continuous” mode—see subsection IV.B—and each measured value comes from averaging the results of a number test sessions, which has been dimensioned to assure a confidence interval of 3%). The obtained power consumption values were then multiplied for the average TCP connection lifetimes in Fig. 7 to get the energy absorbed by the SUT during the transmission/reception of TCP traffic. Then, Figs. 9 and 10 show the difference between such energy absorptions and the ones the SUT would have experienced while in idle mode for the same period. In other words, the values in these figures are the energy delta consumed by the SUT for sending or receiving data from TCP connections in the presence of different congestion control mechanisms. Analyzing the data in Figs. 9 and 10, we can outline how, in the presence of lower bottleneck speeds (cases d, e, f and g), the energy consumption increases in a significant way. This increase is due to the C–state transition overheads, as discussed in Section III. Given that, in the case of 10 Mbit sized bottleneck, the minimum packet inter–arrival time is approximately $1.2\,ms$, the CPU has enough time to *(i)* wake up from the low power idle mode (i.e., the C_3 one) upon packet reception, *(ii)* to thoroughly process the received

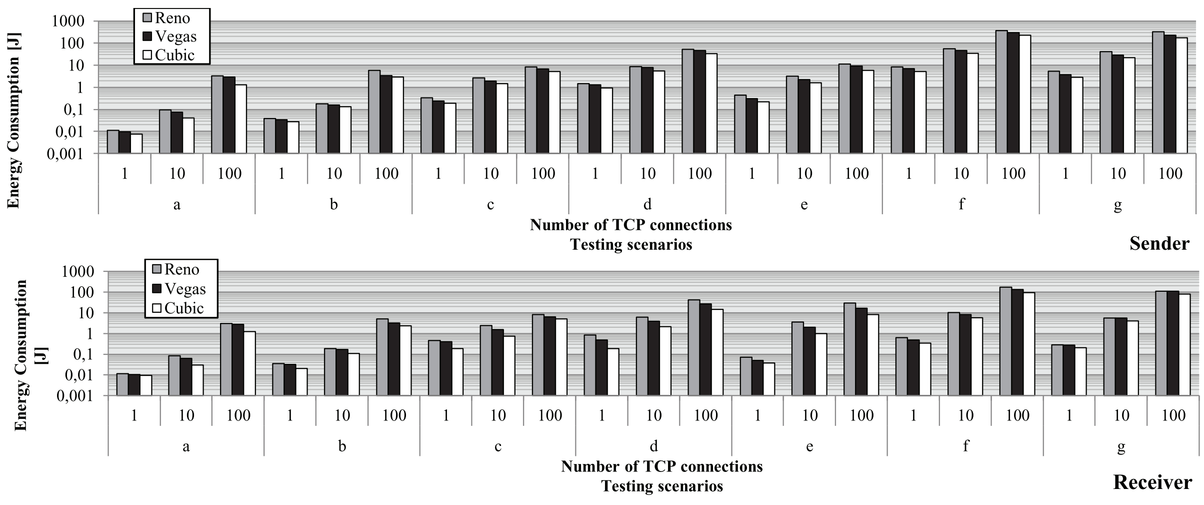


Fig. 10. Energy Consumption of TCP connections with 10 pkt at the SUT when acting as “sender” and “receiver” for Reno, Vegas and Cubic TCP versions.

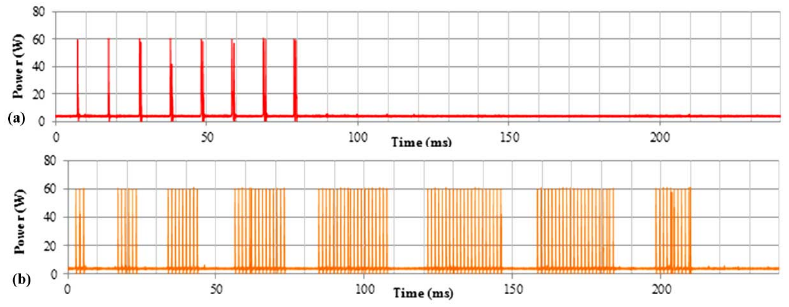


Fig. 11. Power consumption of the SUT CPU during a TCP connection in the testing scenarios c (a) and f (b). The SUT is acting as “receiver,” and the connection carries 100 pkt.

packet, and *(iii)* to return to the C_3 state before the next packet arrives. So, a wake–up power peak (see Fig. 2) is spent for every packet.

The RTT plays a similar role as well. In fact, TCP connections in testing scenarios c, f and g exhibit bigger energy requirements than in the cases a, d and e, respectively. The main cause of this behavior is due to the well–known bandwidth–delay product effect in the TCP. When this product is high (as in c, f and g), the TCP throughput becomes more “bursty,” in the sense that groups of segments are sent every RTT, and among the transmission of these groups there are some “silence” periods. Every time a new group of segments arrives, or it is sent, the CPU SUT has to move from the C_3 state to the C_0 one, causing the wake–up energy spike of Fig. 2. When the product is low (e.g., scenario a), the silence periods, and then the $C_3 - C_0$ transitions, are rarer. Moreover, Figs. 9 and 10 highlight how the TCP Cubic version obtains lower energy consumptions than Reno and Vegas ones. These lower consumptions are caused by higher efficiency level of the Cubic congestion control mechanism, which ensures high network utilization also during the fast recovery of lost packet.

All these dynamics and the impact of both bottleneck bandwidth and the RTT are evident in the transitory measurements in Fig. 11. As expected, in the testing scenario b, the average energy consumption is larger than in the case a: the loss of segments obviously causes the sender to wake–up an additional number of times in order to retransmit the lost packets, or to receive it separately from the original burst. Moreover, after a packet loss, also the congestion window is decreased, and, consequently, the TCP transmission may become more “bursty” (it may happen to have some additional silence periods between two groups of segments, given the decrease of the congestion window after loss events). Besides the considerations above, it is interesting noting that the energy consumption for sending data through a TCP connection appears to be higher than the one needed to receive them in all the considered scenarios. For the sake of simplicity, the following sections will report further results and tests with the Cubic version.

C. *Profiling the CPU SW Activities*

From the results in Section V.B, the energy consumption of the SUT clearly results to be dependent on two main aspects: *(i)* the number of transitions between CPU idle and active states, which can be caused by excessive inter–packet and inter–burst arrival times, and *(ii)* the CPU activity time. The latter certainly depends on the SW computational load, which directly influences the processing time of incoming and outgoing traffic. In this respect, Fig. 12 shows the average CPU uptime values for TCP Cubic connections (i.e., the time the CPU spends in the C_0 state). This parameter can be interpreted as an indirect estimation of SW computational complexity, since it represents how

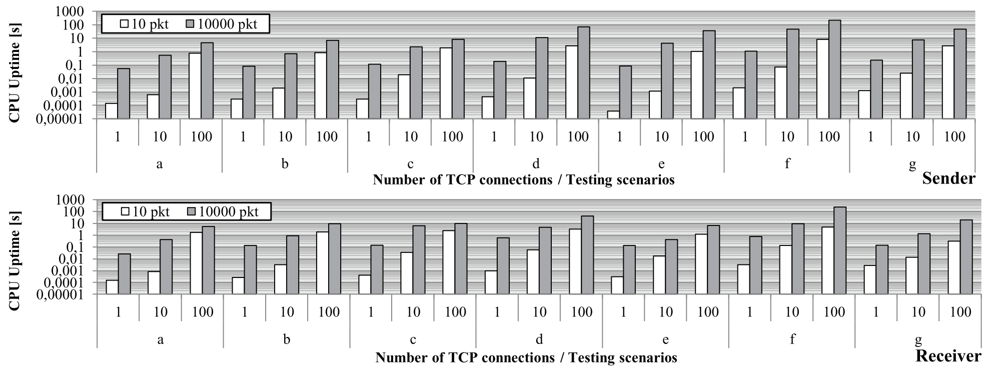


Fig. 12. Average value of CPU uptime during TCP connections with Cubic at the SUT when acting as "sender" and "receiver".

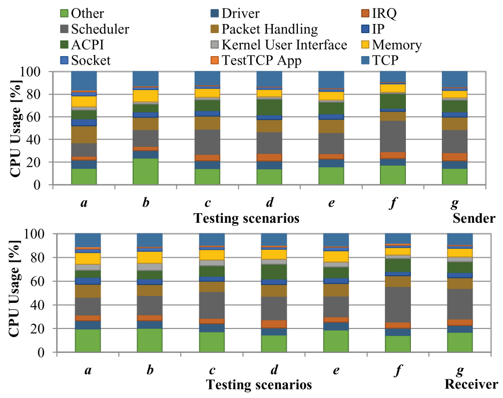


Fig. 13. Breakdown of the CPU usage of a TCP connection of 10 pkt.

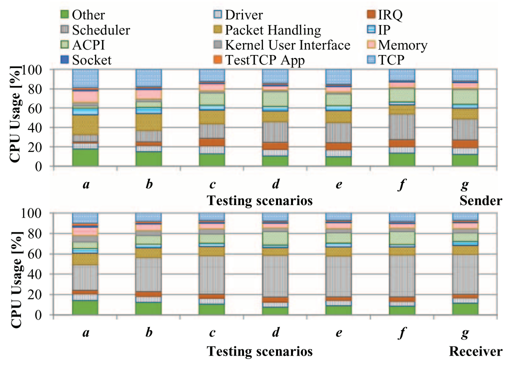


Fig. 14. Breakdown of the CPU usage of a TCP connection of 10 kpkt.

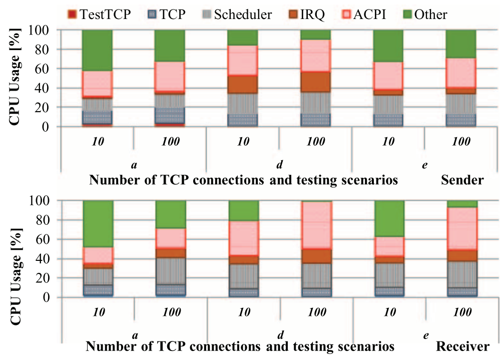


Fig. 15. Breakdown of the CPU usage of 10 and 100 TCP connections of 10 kpkt.

many CPU clock cycles are needed on a time unit to perform all the SW operations needed to send or to receive data through the TCP connection in the selected scenarios. As expected, the CPU uptime seems to be much lower in those scenarios where TCP performance level is higher: increasing the performance of the TCP transfer, also the CPU SW obviously behaves in a more efficient way (e.g., operations to recover losses are rare). However, when the bottleneck bandwidth decreases, or the RTT increases, the CPU uptime may enlarge of almost two orders of magnitude. Moreover, Fig. 12 clearly outlines how the TCP data reception operations appear to require more CPU uptime than the transmission ones. To understand insights of the computational complexity of SUT SW components (and then where a part of the energy consumption arises), we used Oprofile (see Sect. IV.D). Figs. 13–15 show a selection of the results obtained. To make these results as intuitive as possible, we grouped together the source code functions of the Linux OS into 12 categories, each one representing a different functional block of the SUT SW architecture. The selected categories are defined in Table II. Figs. 13 and 14 show the breakdown of the CPU usage of a TCP connection of 10 and 10000 pkts, respectively. The processing of TCP protocol account for 10%–17% of the CPU uptime. This value tends to be higher in sender mode, given the additional TCP processing to retransmit the lost segments (see the next-subsection). The Tcptest application does not significantly affect the CPU uptime (approximately 1–2%). The OS scheduler accounts for a significant share since it varies between 10% and 41%, which is somehow dependent on the HW interrupts (2–8%). In fact, the OS is wakened up by HW interrupts generated by the NIC upon traffic reception. Once woken up, the OS scheduler decides for the next job to be executed (i.e., serving incoming traffic). The scheduler is also recalled when the CPU completes its job backlog, and then is ready to enter into the low power idle state (executing ACPI functions—ranging between 5% and 15%). For this reason, scheduler, IRQ, and ACPI functions tend to increase significantly in all the testing scenarios with a high number of active-idle transitions. So, these three categories may be meant as sources of overhead in the energy consumption.

Within multiple connections, these values further increase (see Fig. 15). For simplicity, this figure shows the categories that require a greater amount of CPU cycles for their execution

TABLE II
FUNCTION CATEGORIES FOR THE ENTIRE SUT OS

Name	Description
Scheduler	the operating system scheduler
ACPI	the ACPI and idle-active transitions functions
Socket	management of the TCP "socket" interface
Driver	the hardware driver of the NIC
Packet Handling	kernel processes related to the reception and transmission of traffic
Kernel User Interface	functions needed to copy data from the kernel to the user-space and vice versa
Tcptest	the TCP testing application
IRQ	management of HW interrupts, mainly due to the NIC
IP	traffic processing at the IP layer
Memory	management of kernel memory
TCP	traffic processing at the TCP layer
Other	other spurious sources of CPU activity

TABLE III
FUNCTION CATEGORIES FOR THE TCP OPERATION

ID	Name	Description
A	*Timer_RTT*	RTT and timeout management
B	*Rx_data*	Processing of received segments
C	*Tx_data*	Processing of segments to be sent
D	*Options*	Processing of TCP options (e.g., SACK)
E	*Congestion_Control*	Calculation of congestion control window
F	*Flow_Control*	TCP flow control mechanism
G	*Checksum*	Checksumming of Rx and Tx segments
H	*Acknowledge*	Generation of the Acknowledgements
I	*Open_Close*	Connection starting and ending phases
J	*State_Machine*	TCP protocol state machine
K	*Retransmission*	Operations to retransmit lost packets

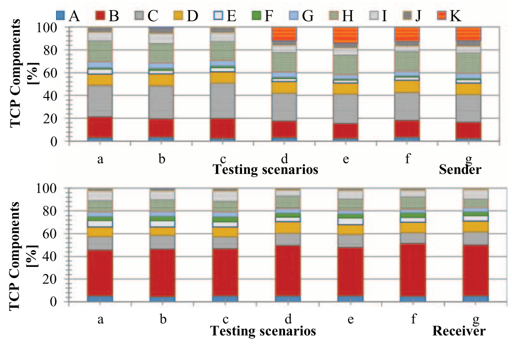


Fig. 16. Breakdown of the TCP process in a connection of 10 pkt.

in cases *a*, *d* and *e*. The ACPI category is the one that experienced a significant increase according to the number of active TCP connections, especially in the case d where the active–idle transitions are higher than in the other scenarios.

D. Breakdown of TCP Functionalities

To analyze the TCP behavior under the different network scenarios in more detail, we decided to refine further the Oprofile results reported in the previous section. The functions related to the TCP were broken down into the new categories of Table III, and the obtained results are in Figs. 16–18. From these figures, the difference between the sender and the receiver modes become manifest.

In the sender mode, the most time–consuming part is the Tx_data, which ranges 25% and 76%. This percentage depends on the amount of packets sent, the network scenario and on

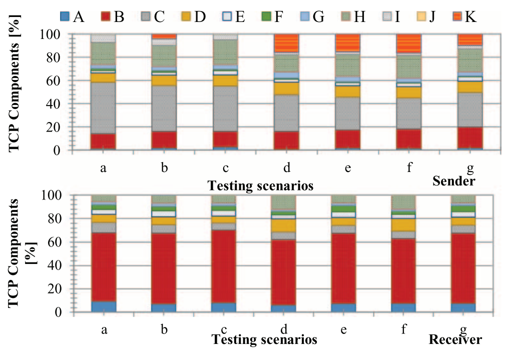


Fig. 17. Breakdown of the TCP process in a connection of 10 kpkt.

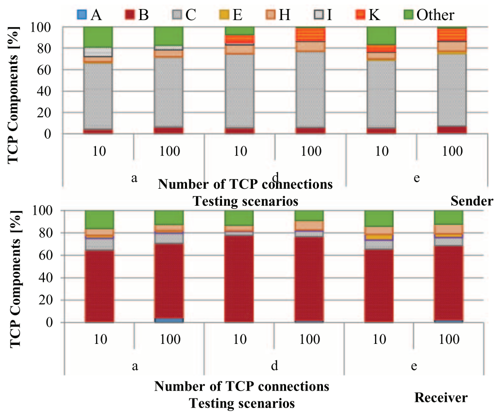


Fig. 18. Breakdown of the TCP process in 10 and 100 connections of 10 kpkt.

the number of active TCP connections. In the cases where the bottleneck speed is lower and/or the RTT is relatively high (*b*, $d-f$ and *g*), we can also note a significant influence of retransmissions (1–16%) due to fast re-transmissions and timeout expirations (see Fig. 8).

The ACK management on the sender is much more consuming that at the receiver (14–21%): in fact the sender must generate a valid ACK field for each sent segment, while the receiver exploits delayed ACKs [33] in a more efficient way (it is generated a single ACK message for a group of correctly received packets). On this respect, we can also note that the *Rx_data* and Acknowledge categories are proportionally dependent, and they require almost the same percentage of TCP CPU cycles in the sender mode. The *Open_Close* and the *State_Machine* categories do not represent a significant part of the overall power consumption of the TCP, since they account for less than 7%. The relative weight of the *Open_Close* and the *State_Machine* categories is obviously more considerable in the case of shorter TCP connections. The management of timeout counters and the estimation of RTT appear to be lightweight operations (less than 4%). When multiple packet losses occur, and the SACK recovering is applied, it consumes up to 9% of the TCP CPU uptime share. The *Congestion_Control* category does not affect the overall weight of the TCP in a significant way; it varies between 2–5%. On the receiver side, the most consuming part of the TCP processing is the *Rx_data*; it spends between 40% and 77% of CPU cycles due to TCP elaboration. The ACK processing is less consuming than on the sender, since this functionality accounts approximately for the same figure of *Tx_data*, 11% of the TCP process. They both represent the elements with a higher percentage of CPU use, after *Rx_data*. We can also underline how the *Flow_Control* mechanism only accounts for 5% of TCP operations. In the presence of multiple connections, these values are higher and depend on the link conditions (see Fig. 18). By increasing the number of TCP connections, the processing for receiving (*Rx_data*) and transmitting data (*Tx_data*) are the most consuming parts in the TCP processing.

VI. Methods To Reduce The TCP Energy Consumption

The experimental analysis of Section V outlines how TCP data transfers may have a significant impact on PC energy consumption. This impact seems to arise only in a minor part from the computational complexity of the SW implementations of the TCP or of related network protocols/functionalities, which appear to be limited. However, it is rather caused by the computation dynamics that network protocols and TCP trigger, and that can lead to relatively high energy inefficiencies. In this respect, it is worth recalling the results in Figs. 9 and 10, where the energy consumption of a TCP connection increases by even more than two orders of magnitude depending on the network scenario and packet-level dynamics. The power consumption measures in Fig. 11 and the software profiling results in Figs. 13–15 make evident that these inefficiencies mainly spring from the "de-burstification" of network traffic, which triggers the CPU to additional and unnecessary wake–ups and sleeping procedures. Given these considerations, we considered suitable and simple approaches to reduce the energy consumption induced by the TCP at both the sender and the receiver sides. The former approach (at the sender side) aims at reducing the number of CPU operations for creating headers of packets to be transmitted using an already existing offloading mechanism. The latter (at the receiver side) tries to reduce the host energy consumption by limiting the number of active–idle transitions at the CPU. Also in this case, we exploited an already existing tuning/offloading functionality at the network adapter. It is worth to underline that our primary objective in this respect is not to identify the optimal mechanism to minimize the TCP energy consumption, neither the optimal parameter configuration of the aforementioned techniques, but to provide a tangible proof that mechanisms already existing in modern network adapters and often underutilized can help in dramatically reduce the energy absorption.

A. The Sender Side

To save energy during transmission operations, we enabled "TCP Segmentation Offload" (TSO) [34] on the NIC host at the transmitter side. The TSO mechanism allows the OS to process multiple TCP segments as single "large" packet, so that the OS has only to process a single TCP/IP header instead of one header per segment. Then, at the physical layer, the TSO–enabled NIC divides the aggregated segments and produces a valid header for each (1500 B sized) packet to be transmitted.

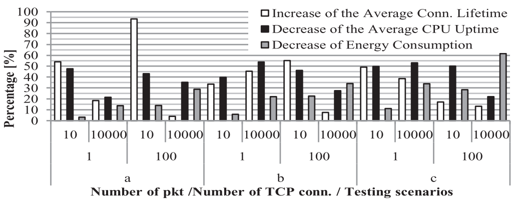


Fig. 19. Increase of the average connection lifetime, and decrease of CPU uptime and of energy consumption for 1 and 100 TCP connections in the scenarios a, b, and c when the TSO mechanism is enabled. Values are expressed as percentages with respect to the ones without the TSO mechanism enabled.

The advantage of this technique is to reduce the computational weight of TCP transmission operations. In other words, the OS can handle the data to be transmitted as a sort of "jumbo frame" (up to 64 KB), building a single TCP/IP header stack. Given that TSO works with high–speed interfaces, we carried out some tests in the *a*, *b* and *c* testing scenarios where the bandwidth of bottleneck link was set to 1 Gbps. It is well known from previous scientific works [35] that TSO has some evident limitations/drawbacks, partly because past implementations showed little or limited performance benefit. One of the most known disadvantages resides in the negative influence on RTT, resulting in larger connection lifetimes as demonstrated in Fig. 19. Notwithstanding such performance decays, Fig. 19 shows how the TSO mechanism allows sensibly reducing the CPU activity time (38% on the average with peaks exceeding 50%). Fig. 19 clearly shows that the final impact on energy consumption obviously depends on the balance between the increase of the connection lifetimes and the decrease in the CPU utilization. Where these parameters counterbalance themselves, the energy saving appears to be limited (from 3% to 11%). Where the CPU utilization decrease overcomes the increase of the average connection lifetimes, energy savings exceed 30% and may arrive even to 60% (as in the *c* test case with 100 parallel connections size with 10K packets). Generally speaking, the energy saving provided by the TSO could be further improved by increasing the maximum number of aggregated segments to be processed by the OS, however this approach could dramatically degrading performance as previously highlighted.

B. The Receiver Side

The results reported in Sect. V.B indicate that "slow" link capacities (of the order of some tens of Mbps) cause the increase of packet inter–arrival times. This effect produces a larger amount of wake-up power peaks because the host receives packets from the sender one at a time. To avoid these active-idle transitions, we enabled coalescing mechanisms already existing at the NIC [36]. In particular, we enabled the HW interrupt coalescing, configuring it to its default settings. The HW interrupt coalescing limits the rate of raised interrupts (IRQ) upon the reception of new packets by waiting a certain amount of time before generating an interrupt. Thus, instead of generating an interrupt for every incoming packet, interrupt coalescing adds a delay in the IRQ generation allowing the CPU to proccess groups multiple packets upon a single IRQ. This IRQ rate limitation somehow allows the

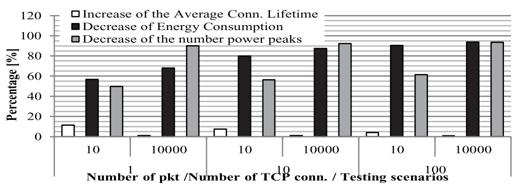


Fig. 20. Decrease/increase on the average connection lifetime, on the energy consumption, and on the number of power peaks in the testing scenario d when the hardware coalescing is enabled.

"re-burstification" of network traffic, and consequently reduces the possible number of CPU wake–up upon packet reception events. To evaluate the potential gain, we decided to report in Fig. 20 the results of some tests carried out in the scenario d. For better assessing the obtained results, the values in Fig. 20 have been expressed as increase/decrease percentage against the results of the testing scenario d without the hardware interrupt coalescing (see Sect. V.B). Fig. 20 outlines how close the performance provided by the packet coalescing approach is to the usual case with multiple TCP connections. The average lifetime of TCP connections increases by less of 4%. Moreover, it can be noted that HW interrupt coalescing provides better results in testing scenarios with higher levels of traffic. For instance, with 10 Kpkts sized connections the impact on the connection lifetime appears to be negligible ($< 1\%$). Regarding the energy aspects, Fig. 20 exhibits significant savings also exceeding 75%. These savings are related to the considerable reduction of the number of wake–up power peaks due to active–idle transitions as depicted in Fig. 20. Determining the optimal value of interrupt generation delay usually involves a trade–off between latency and energy efficiency. Fig. 20 makes evident that, when network traffic is low, delayed interrupts are unlikely to improve performance, in these cases, shorter interrupt delays are desirable to minimize the latency on each packet. Differently, when network traffic is high, larger interrupt delays are desirable to minimize interrupt-processing overhead and improve the energy efficiency of the system.

VII. Lesson Learned and Conclusions

In this paper, we conducted an extensive experimental evaluation to study the impact of TCP traffic on a networked device under realistic scenarios. From the experimental analysis in Section V, we observed that the energy consumption induced by TCP traffic can vary in a significant way depending on the network scenario. The measurements reported in Figs. 9 and 10 underline how the energy budget to transmit or to receive a single TCP segment can pass from few tens of μJ to approximately $15\,mJ$ (almost three orders of magnitude). This figures are impressive if compared to the Cisco forecast of 966 exabytes a year of Internet traffic in 2015 [3]. The obtained results helped us in understanding that this enourmous variation in energy efficiency levels depends only in minor part on the particular TCP version used by the network device, or on its software implementation. For instance, the TCP Cubic can improve energy efficiency only up to approximately 50% and 20% on the average with respect to the Reno version.

The achieved results made manifest that energy efficiency levels are mainly dependent on the number of transitions between CPU idle states and active state, which can be triggered by excessive inter–packet and inter–burst arrival times. In fact, the highest energy budgets per TCP segment have been measured especially in the presence of low bottleneck speeds (cases d, and f) and/or when the RTT is relatively high (cases f, and g). In these cases, we tried to decompose the activity time of the CPU by means of SW profiling tools in order to get a better insight of these inefficiencies. In the presence of cases d, f and g, this analysis outlined how, despite the additional energy spent for making the CPU entering and exiting frequently from the C_3 state, the CPU is used for a very large share of time by power and OS management functions. For instance, the aggregate CPU utilization of functions related to the OS scheduler, the ACPI and IRQs passes from approximately 30% of the case a to more than 60% in the case f.

In order to mitigate the effects shown above, we evaluated the impact of enabling the IRQ coalescing mechanisms already existing in NICs. The results underlined how these mechanisms can provide to TCP receivers energy savings spanning from 60% to 90% in the critical cases above mentioned (d, f and g). Regarding the sender side, enabling the well–known TSO mechanism at the NIC, we obtained limited energy reduction (in the order of 3–11%) in case of a single active connection, while in case of multiple active connections savings span from 30% to 60%. We strongly believe that these preliminary evaluations along with the detailed analysis here performed can help our community to design new mechanisms and optimizations for achieving much higher energy efficiency levels in TCP operations of networked devices.

References

[1] GESI (2008), *Smart 2020* [Online]. Available: http://www.smart2020.org/

[2] R. Bolla *et al.*, "Cutting the energy bills of Internet service providers and telecoms through power management: An impact analysis," *Comput. Netw.*, vol. 56, no. 10, pp. 2320–2342, 2012.

[3] Cisco White Paper, "Cisco Visual Networking Index: Forecast and methodology, 2014–2019," May 2015, [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/serviceprovider/ipngnipnextgenerationnetwork/white_paper_c11481360.html.

[4] A.-C. Orgerie *et al.*, "A survey on techniques for improving the energy efficiency of large-scale distributed systems," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 47:1–47:31, Mar. 2014.

[5] A. Finamore *et al.*, "Experiences of Internet traffic monitoring with tstat," *IEEE Netw.*, vol. 25, no. 3, pp. 8–14, May 2011.

[6] S. Nedevschi *et al.*, "Reducing network energy consumption via sleeping and rate-adaptation," in *Proc. 5th USENIX Symp. Netw. Syst. Des. Implement.*, 2008, pp. 323–336.

[7] B. Wang and S. Singh, "Computational energy cost of TCP," in *Proc. 23rd Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM'04)*, Mar. 2004, vol. 2, pp. 785–795.

[8] S. Datta *et al.*, "Android power management: Current and future trends," in *Proc. 1st IEEE Workshop Enabl. Technol. Smartphone Internet of Things (ETSIoT)*, Jun. 2012, pp. 48–53.

[9] V. Guittot, *Power Management of an Arm System* [Online]. Available: https://www.linaro.org/blog/power-management-of-an-arm-system/

[10] R. Bolla *et al.*, "The energy consumption of TCP," in *Proc. 4th Int. Conf. Future Energy Syst.*, 2013, pp. 203–212.

[11] M. Zorzi and R. Rao, "Is TCP energy efficient?" in *Proc. IEEE Int. Workshop Mobile Multimedia Commun. (MoMuC'99)*, Nov. 1999, pp. 198–201.

[12] V. Tsaoussidis *et al.*, "Energy/throughput tradeoffs of TCP error control strategies," in *Proc. 5th IEEE Symp. Comput. Commun. (ISCC'00)*, 2000, pp. 106–112.

[13] A. Sassu *et al.*, “TCP behavior over a greened network,” in *Proc. Sustain. Internet ICT for Sustain. (SustainIT’12)*, Oct. 2012, pp. 1–5.

[14] L. Irish and K. Christensen, “A ‘Green TCP/IP’ to reduce electricity consumed by computers,” in *Proc. IEEE Southeastcon*, Apr. 1998, pp. 302–305.

[15] P. Reviriego *et al.*, “On the impact of the TCP acknowledgement frequency on energy efficient ethernet performance,” in *Proc. IFIP TC 6th Int. Conf. Netw.*, 2011, pp. 265–272.

[16] Y. Zhang and N. Ansari, “On architecture design, congestion notification, TCP Incast and power consumption in data centers,” *IEEE Commun. Surv. Tuts.*, vol. 15, no. 1, pp. 39–64, Feb. 2013.

[17] Y. Hanay *et al.*, “Saving energy and improving TCP throughput with rate adaptation in ethernet,” in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2012, pp. 1249–1254.

[18] A. Francini, “Periodic early detection for improved TCP performance and energy efficiency,” *Comput. Netw.*, vol. 56, no. 13, pp. 3076–3086, Sep. 2012.

[19] G. Regnier *et al.*, “TCP onloading for data center servers,” *Computer*, vol. 37, no. 11, pp. 48–58, Nov. 2004.

[20] R. Westrelin *et al.*, “Studying network protocol offload with emulation: Approach and preliminary results,” in *Proc. 12th Annu. IEEE Symp. High Perform. Interconnects*, Aug. 2004, pp. 84–90.

[21] ACPI Standard Specification [Online]. Available: http://www.acpi.info/

[22] Intel Core i5 Processor [Online]. Available: http://download.intel.com/design/processor/datashts/322164.pdf

[23] TCP Test Application [Online]. Available: http://www.tnt.dist.unige.it/Tcptest/

[24] S. Hemminger, “Network emulation with NetEm,” in *Proc. 6th Australia’s Natl. Linux Conf. (LCA)*, Apr. 2005.

[25] Dominion PX-5297 [Online]. Available: http://www.raritan.com/px-5000/px-5297/tech-specs/

[26] DDR3 Riser Card [Online]. Available: http://www.adexelec.com/other.htm#DDR3

[27] PEX16LX PCI-e x16 tall extender [Online]. Available: http://www.adexelec.com/

[28] ATX 2.1 [Online]. Available: http://www.formfactors.org/developer/specs/atx2_1.pdf

[29] Agilent U2356A [Online]. Available: http://www.home.agilent.com/

[30] Oprofile [Online]. Available: http://oprofile.sourceforge.net/

[31] P. Sarolahti and A. Kuznetsov, “Congestion control in linux TCP,” in *Proc. USENIX Conf. FREENIX Track*, 2002, pp. 49–62.

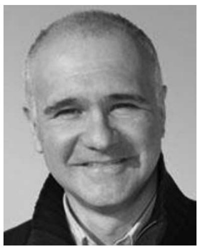
[32] S. Ha *et al.*, “Cubic: A new TCP-friendly high-speed TCP variant,” *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008.

[33] R. Brade, “Requirements for internet hosts-communication layers,” *IETF RFC*, vol. 1122, sec. 4.2.3.2, Oct. 1989.

[34] D. Freimuth *et al.*, “Server network scalability and TCP offload,” in *Proc. USENIX Annu. Tech. Conf.*, Apr. 2005, pp. 209–222.

[35] J. C. Mogul, “TCP offload is a dumb idea whose time has come,” in *Proc. 9th Conf. Hot Topics Oper. Syst.*, 2003, p. 5.

[36] M. Zec *et al.*, “Estimating the impact of interrupt coalescing delays on steady state TCP throughput,” in *Proc. 10th SoftCOM*, 2002.



Raffaele Bolla (M’91) was born in Savona, Italy, in 1963. He received the “Laurea” degree in electronic engineering from the University of Genoa, Genoa, Italy, in 1989, and the Ph.D. degree in telecommunications from the Department of Communications, Computer and Systems Science (DIST), University of Genoa, in 1994. From 1996 to 2004, he was a Researcher with DIST, University of Genoa, where since 2004, he has been an Associated Professor. He teaches courses in telecommunication networks and telematics for the degrees in electronic and communication engineering at the University of Genoa. He has authored or coauthored about 100 scientific publications in international journals and conference proceedings. He has been the Principal Investigator in many projects in the field of telecommunication networks. His research interests include resource allocation, call admission control and routing in multiservice IP networks, multiple access control, resource allocation and routing in both cellular, and ad hoc wireless network.

Roberto Bruschi (SM’09) received the M.Sc. degree in telecommunication engineering and the Ph.D. degree in electronic engineering from the University of Genoa, Genoa, Italy, in 2002 and 2006, respectively. Since 2009, he has been a Researcher with the National Inter-University Consortium for Telecommunications (CNIT), University of Genoa Research Unit, Genoa, Italy. He is the Technical Coordinator Assistant in the ECONET project, and the Principal Investigator in the GreenNet project. He has coauthored about 80 scientific papers in international journals, book chapters, and international conference proceedings. His research interests include green networking, software routers, and multiple-play networks. He has been a Technical Committee Member of many international conferences. He was the recipient of the the Best Paper Award at the Next-Generation Networking Symposium of the IEEE ICC conference in 2009, and the 3rd International Workshop on GreenCom (GreenCom 10) co-located with the IEEE GLOBECOM Conference in 2010.

Olga Maria Jaramillo Ortiz was born in Guayaquil, Ecuador, in 1983. She graduated in Electronic and Telecommunication Engineering from Escuela Superior Politecnica del Litoral (ESPOL), Guayaquil, Ecuador, in 2007. She is currently a Post-Doc with the Telecommunication Networks and Telematics Laboratory (TNT), DITEN Department, University of Genoa, Genoa, Italy. She is a member of CNIT, the Italian Inter-University Consortium for Telecommunications. Her research interests include power consumption measurements in wired networks environments and the green networking field in general.

Paolo Lago was born in Genoa, Italy, in 1985. He received the “Laurea” degree (equivalent to M.Sc.) in telecommunication engineering and the Ph.D. degree from the University of Genoa, Genoa, Italy, in 2010 and 2014, respectively. He is currently a Post-Doc with the Telecommunication Networks and Telematics Laboratory (TNT), DITEN Department, University of Genoa. He also collaborates with the National Inter-University Consortium for Telecommunications (CNIT). His research interests include green networking, software routers, and network virtualization.