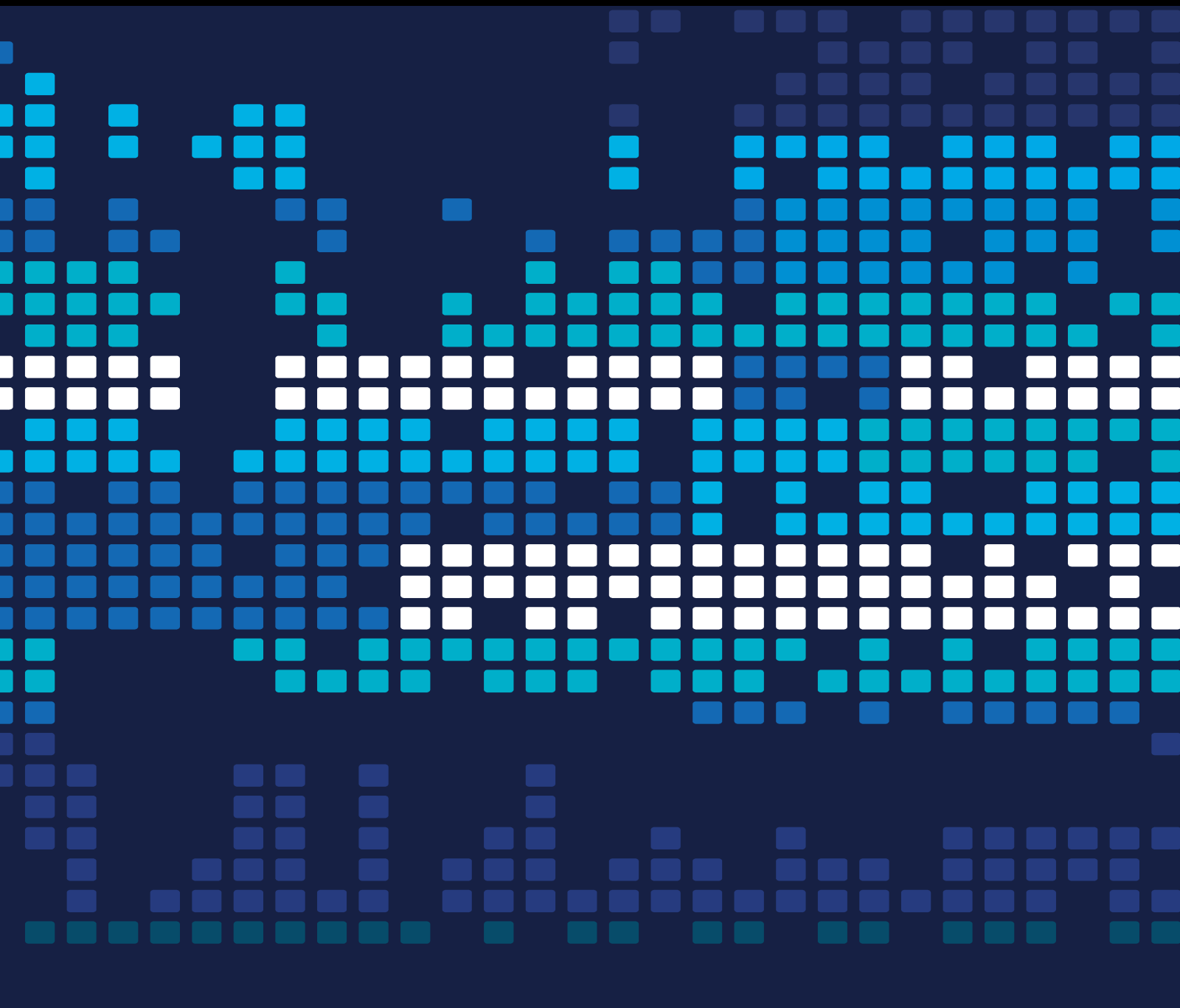


# Scientific Programming in the Fog and Edge Computing Era

Lead Guest Editor: Daniele D'Agostino

Guest Editors: Ivan Merelli and Daniele Cesini





---

# **Scientific Programming in the Fog and Edge Computing Era**

Scientific Programming

---

## **Scientific Programming in the Fog and Edge Computing Era**

Lead Guest Editor: Daniele D'Agostino

Guest Editors: Ivan Merelli and Daniele Cesini




---

Copyright © 2021 Hindawi Limited. All rights reserved.

This is a special issue published in "Scientific Programming." All articles are open access articles distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

# Chief Editor

Emiliano Tramontana , Italy




## Academic Editors

Marco Aldinucci , Italy  
Daniela Briola, Italy  
Debo Cheng , Australia  
Ferruccio Damiani , Italy  
Sergio Di Martino , Italy  
Sheng Du , China  
Basilio B. Fragueta , Spain  
Jianping Gou , China  
Jiwei Huang , China  
Sadiq Hussain , India  
Shujuan Jiang , China  
Oscar Karnalim, Indonesia  
José E. Labra, Spain  
Maurizio Leotta , Italy  
Zhihan Liu , China  
Piotr Luszczek, USA  
Tomàs Margalef , Spain  
Cristian Mateos , Argentina  
Zahid Mehmood , Pakistan  
Roberto Natella , Italy  
Diego Oliva, Mexico  
Antonio J. Peña , Spain  
Danilo Pianini , Italy  
Jiangbo Qian , China  
David Ruano-Ordás , Spain  
Željko Stević , Bosnia and Herzegovina  
Kangkang Sun , China  
Zhiri Tang , Hong Kong  
Autilia Vitiello , Italy  
Pengwei Wang , China  
Jan Weglarz, Poland  
Hong Wenxing , China  
Dongpo Xu , China  
Tolga Zaman, Turkey

## Contents

---

**Hardware and Software Solutions for Energy-Efficient Computing in Scientific Programming**

Daniele D'Agostino , Ivan Merelli , Marco Aldinucci , and Daniele Cesini 


Review Article (9 pages), Article ID 5514284, Volume 2021 (2021)

**MCAF: Developing an Annotation-Based Offloading Framework for Mobile Cloud Computing**

Yilian Zhou, Ligang He , Bin Wang, Yi Su, and Hao Chen






Research Article (9 pages), Article ID 5304612, Volume 2020 (2020)

**Deep Recurrent Neural Networks for Edge Monitoring of Personal Risk and Warning Situations**

Emanuele Torti , Mirto Musci, Federico Guareschi, Francesco Leporati, and Marco Piastra

Research Article (10 pages), Article ID 9135196, Volume 2019 (2019)

**A Practical Approach to Protect IoT Devices against Attacks and Compile Security Incident Datasets**

Bruno Cruz , Silvana Gómez-Meire , David Ruano-Ordás , Helge Janicke, Iryna Yevseyeva , and Jose R. Méndez 

Research Article (11 pages), Article ID 9067512, Volume 2019 (2019)

## Review Article

# Hardware and Software Solutions for Energy-Efficient Computing in Scientific Programming

**Daniele D'Agostino** <sup>1</sup>, **Ivan Merelli** <sup>2</sup>, **Marco Aldinucci** <sup>3</sup> and **Daniele Cesini** <sup>4</sup>

<sup>1</sup>*CNR-IEIIT, Genoa, Italy*

<sup>2</sup>*CNR-ITB, Segrate (MI), Italy*

<sup>3</sup>*University of Turin, Turin, Italy*

<sup>4</sup>*CNAF-Italian Institute for Nuclear Physics, Bologna, Italy*

Correspondence should be addressed to Daniele D'Agostino; [daniele.dagostino@ge.imati.cnr.it](mailto:daniele.dagostino@ge.imati.cnr.it)

Received 25 January 2021; Accepted 28 May 2021; Published 9 June 2021

Academic Editor: Cristian Mateos

Copyright © 2021 Daniele D'Agostino et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Energy consumption is one of the major issues in today's computer science, and an increasing number of scientific communities are interested in evaluating the tradeoff between time-to-solution and energy-to-solution. Despite, in the last two decades, computing which revolved around centralized computing infrastructures, such as supercomputing and data centers, the wide adoption of the Internet of Things (IoT) paradigm is currently inverting this trend due to the huge amount of data it generates, pushing computing power back to places where the data are generated—the so-called fog/edge computing. This shift towards a decentralized model requires an equivalent change in the software engineering paradigms, development environments, hardware tools, languages, and computation models for scientific programming because the local computational capabilities are typically limited and require a careful evaluation of power consumption. This paper aims to present how these concepts can be actually implemented in scientific software by presenting the state of the art of powerful, less power-hungry processors from one side and energy-aware tools and techniques from the other one.

## 1. Introduction

Information and communication technologies (ICT) play a fundamental role in supporting human activities for the global economic, social, and environmentally sustainable developments [1]. However, energy consumption is one of the most relevant issues for present computing platforms, and this trend is expected to continue in the foreseeable future. This implies that the electricity bill increasingly dominates costs related to the running of applications and the consequent environmental pollution [2].

This situation is evident for high-performance computing (HPC) infrastructures, where the sum of the energy bills over a supercomputer's lifetime is comparable to the

acquisition cost and represents one of the most relevant elements of the total cost of ownership [3]. This is because energy is used not only for computation but also for cooling, communication, storage, and display [4].

The focus of performance-at-any-cost computer operations has led to the emergence of supercomputers that consume vast amounts of electrical power and produce so much heat in that extended cooling facilities must be constructed to ensure proper performance. The consequence is that, in the context of deploying an exascale system, the simple scaling of current technologies would result in a supercomputer with a power consumption of 100 MW, while a limit of 20 MW has been estimated as the maximum acceptable limit [5]. The attention to the flop-per-watt

performance has been demonstrated by the introduction, in 2007, of the Green500 List [6] that ranks the top 500 supercomputers by energy efficiency [7].

The same problem also arises in general-purpose data centers: in the US, such infrastructures consumed about 70 billion kWh in 2014, representing 1.8% of total US electricity consumption, as reported in [8]. Some projections estimate for 2020 an electricity demand that varies by about 135 billion kWh, depending on the adoption rate of efficiency measures [9].

This scenario must be combined because in the past two decades, computing has been focused around centralized (and possibly complex [10]) infrastructures, but the wider diffusion of cyber-physical systems (CPSs) is currently inverting this trend, pushing computing power back to where data are generated. In both cases, the energy consumption of telecommunication networks is very relevant [11]. A striking example of the trend is the Internet of Things (IoT) paradigm, by which millions of devices generate a huge amount of data that are pre-elaborated locally before being integrated remotely in a data analytics context. Nevertheless, also considering science, the diffusion of powerful data acquisition devices boosted the diffusion of pre-elaboration computational architectures, such as in bioinformatics [12, 13].

While HPC is a well-specific market sector, the so-called “embedded HPC” is an emerging topic [14] to develop and employ microservers/highly parallel embedded computing systems in the CPS. Therefore, the adoption of energy-efficient systems represents a crucial aspect considering the characteristics of fog/edge computing environments [15].

We can formulate the problem as the need to assess a satisfactory tradeoff between time-to-solution and energy-to-solution. This problem has been faced with different approaches, which can be summarised as follows: vendors work on power-efficient processor architectures and software developers on how to use them. However, to reach exascale computing, an effective solution is possible only by properly managing all layers of the system, from the software stack to the cooling system [16] passing by less power-hungry CPUs. This can be achieved by reducing the energy consumed in the total system via both power-efficient software and hardware integrated solutions [17, 18].

Energy efficiency is a key design challenge for modern computing systems for many years. Even more now, the Big Data paradigm requires addressing both issues related to the efficient processing of such an enormous amount of data and how to achieve this goal in a green way, i.e., considering issues related to sustainability and environmental concerns [19].

Therefore, many papers proposing novel techniques for managing power aspects and presenting real-world experiences, together with surveys and overviews, have been published. A critical analysis on how to greening the whole life cycle of big data systems is presented in [20]. On a more technical perspective, Czarnul et al. [21] focused on the available methods and tools allowing proper configuration,

management, and simulation of HPC systems for energy-aware processing. An overview of application performance analysis tools, including the energetic profiling of an application and auto-tuning tools for energy saving, has been presented in [22]. The usage of low-power System-on-Chip (SoC) architectures for scientific (and industrial) applications is discussed in [23], intending to assess the tradeoff among time-to-solution, energy-to-solution, and economic aspects for both scientific and commercial purposes they can achieve in comparison to traditional server-grade architectures adopted in present infrastructures.

However, an issue is represented by the fact that nearly all the existing surveys focus on only one of the two main strategies, i.e.,

- (i) The development and usage of new energy-efficient CPUs and SoCs
- (ii) The use of software tools and frameworks for reducing the power consumption of software using an existing CPU

Moreover, as recognized by most of these papers, this is a rapidly evolving research field where new results are continuously presented. For example, at the time of writing, the following five European research projects and initiatives are ongoing:

- (i) Mont-Blanc 2020, European scalable, modular, and power-efficient HPC processor
- (ii) HiPEAC, High Performance and Embedded Architecture and Compilation
- (iii) LEGaTO, Low-Energy Toolset for Heterogeneous Computing
- (iv) SDK4ED, Software Development toolKit for Energy optimization and technical Debt elimination
- (v) TeamPlay, Time, Energy and security Analysis for Multi/Many-core heterogeneous PLATforms

This is because the European Commission has been aware since at least 2010 that the ICT sector is responsible for carbon emissions which are rapidly growing and should be kept to a minimum and therefore is supporting the development of more energy-efficient computing technologies.

Therefore, this work’s main goal is to present the most relevant available solutions for users interested in improving the energy consumption of scientific software focusing on computation. This is achieved by investigating the availability and performance of current hardware devices and software tools for scientific applications.

This means that the aspects related to energy efficiency in communications are not considered here. Interested readers can rely on [24, 25].

The structure of the paper is as follows: Section 2 presents hardware techniques and solutions for achieving energy-savvy processing, Section 3 discusses tools and methodologies for supporting developers in producing energy-aware software, while the last section concludes the paper.



## 2. Energy-Efficient Architectures

*2.1. General-Purpose Techniques.* Firstly, let us review the techniques that exploit hardware characteristics to reduce energy consumption. Most of the present architectures, in fact, implement energy-saving techniques. They are based on the use of low-level electronic characteristics to run no faster than necessary at a voltage no higher than acceptable. They are

- (i) Dynamic frequency scaling (DFS)
- (ii) Dynamic voltage scaling (DVS)
- (iii) Dynamic voltage and frequency scaling (DVFS)
- (iv) Near-threshold voltage (NTV)
- (v) Dynamic power management (DPM)

Dynamic frequency (DFS) or voltage (DVS) scaling allows to modulate the power consumption processor and memory [26], scaling the clock frequency of one or both subsystems according to the execution of memory- or compute-bound application kernels [27].

For example, voltage reduction has to be considered for the heterogeneous accelerators equipping current systems also because the efficient reduction of the total power can be achieved with different voltage reduction levels for each available chip [28].

Very often, voltage and frequency ranges are fully interdependent, i.e., a change in clock frequency does imply changes in the supply voltage, and vice versa: in these cases, the technique is called dynamic voltage and frequency scaling (DVFS) [29]. Specific hardware mechanisms can implement DVFS with minimal software and operating system involvement or through enabling software.

For example, DVFS is implemented in the Linux kernel with the CPUfreq subsystem [30, 31]. The original implementation of kernel 2.6 has been designed to be used when no real-time tasks are executed. However, it is possible to relax this constraint [32].

More recently, other projects focused on near-threshold voltage (NTV) computing [33], making the processors work at even lower voltages. Since this may lead to computation errors, appropriate checks and recomputation have to be added to algorithms in this case.

On the contrary, the Intel Turbo Boost technology opportunistically allows the processor to run faster than the nominal frequency if the CPU is operating below the defined power and temperature limits to speed up compute-intensive applications [34]. In detail, as explained in [35], “the thermal design power (TDP) represents the maximum amount of power the cooling system in a computer requires to dissipate. This is the power budget under which the system needs to operate. Nevertheless, this is not the same as the maximum power the processor can consume. The processor can consume more than the TDP for a short time without it being thermally significant.” More details on this and the hardware power controller called Running Average Power Limit (RAPL) introduced with the Sandy Bridge architecture are provided in [36]. A similar solution, the

NVIDIA Management Library (NVML), has been provided for NVIDIA GPUs [37, 38].

The Advanced Configuration and Power Interface specification has been developed since 1996 to provide the possibility to manage these aspects via software, e.g., at the operative system level. For example, ACPI defines up to 16 active states, named P0–P15, associated with a set of power/performance/latency characteristics [39]. In P0, the process runs at the maximum power and frequency level, while these values are decreased from P1 till maximum supported Pi [40].

*2.2. Commercial-Off-the-Shelf Low-Power Devices.* The energy-efficient architectures range from many-core architectures, such as the Graphics Processing Unit (GPU) to System on Chip (SoC), to Systems-on-Chip (SoCs). GPUs feature a high performance-per-watt ratio. At the time of writing this paper, the most powerful GPU devices, AMD MI100 and NVIDIA A100, presented, respectively, a peak performance of 38.33 gigaflops per watt (GFlops/W) and 24.25 GFlops/W considering 64 bit floating-point operations, with a power consumption of, respectively, 300 and 260 watt. It is, therefore, clear that GPUs aim at one side at energy efficiency, but they require careful programming and optimization to provide high computing performance.

The increasingly adopted class of low-power processors, often called System-on-Chip (SoC), originally designed for the embedded and mobile market, represents an attractive solution for scientific and industrial applications given their increasing computing performance coupled with relatively low cost and low electrical power demand.

SoC hardware platforms typically embed in the same die low-power multicore processors possibly combined with a GPU and all the circuitry needed for several I/O devices. For the case of off-the-shelf SoCs, various limitations may arise, such as 32 bit-only architectures, small CPU caches, small RAM sizes, high latency interconnections, and unavailability of ECC memory.

However, some solutions are progressively reducing the performance gap with high-end processors, with the added value of keeping a competitive edge on costs, reducing their carbon footprint, and preserving the environment. For these reasons, in this paper, we disregard devices such as Arduino or Raspberry Pi devices that, even if considered for compute-intensive applications [41], are mainly used for equipping IoT systems [42, 43] without significant, local preprocessing of data.

Fugaku represents the most important example of the adoption of SoCs for HPC—the first supercomputer in the TOP500 list of November 2020 and the most recent at the time of writing this paper—which is equipped with Fujitsu’s 48-core A64FX SoC, providing a comparable performance-per-watt value with respect to GPU-based systems [44].

In the corresponding Green500 List, we can see that Fugaku appears in position 10 with a value of 15.418 GFlops/W, while NVIDIA DGX SuperPOD, the most energy-savvy system which is equipped with NVIDIA A100 GPUs,

provides 26.195 GFlops/W but is ranked only at position 170 in the TOP500. A more interesting comparison is between Fugaku and Selene, again a supercomputer equipped with A100 GPUs: this last appears in position 5 in both lists, with a value of 23.983 GFlops/W but providing only 63,460 TFlops/s with respect to 442,010 TFlops/s provided by Fugaku.

As for most HPC architectures, the question remains this [45]: do the raw numbers related to performance per second and watt correspond to achievable performance figures for most of the scientific applications and, in particular, for the application I am interested into?

This was the goal of the Computing On SOC Architecture (COSA) project [46, 47], an initiative funded by the Italian Institute for Nuclear Physics (INFN) between 2015 and 2018. In particular, the COSA project focused on assessing the energy consumption behavior of a wide set of state-of-the-art architectures using benchmarks and software widely used in many scientific applications.

In particular, an in-depth comparison of the performance of x86-based SoCs (i.e., Pentium N3700 and J4205, Avoton C2750, Xeon D1540, and Atom C3958) and low-power GPUs (i.e., Jetson TK1 and TX1) for state-of-the-art high-end solutions (i.e., Xeon E5-2683 and Tesla K20) is discussed in [23] with two benchmarks, represented by the widely used, computationally intensive N-body algorithm and the use of a deep learning approach applied to a classification problem, together with the real-world application taken from the field of molecular biology.

Although comparing high-end commercial/HPC servers with motherboards based on low-power SoC taken from the mobile and embedded world can be considered unfair, the results assess that the use of low-power architectures represents a feasible choice in terms of tradeoff among time-to-solution, energy-to-solution, and economic aspects.

The authors also discuss the economic aspects in [15, 48] by showing how a proper placement of the computational services considering edge and fog's composition cloud infrastructures is the key factor for achieving the best tradeoff between costs, performance, and power consumption.

Regarding the usage of SoCs based on ARM instruction set architectures (ISAs) or FPGAs, a quantitative evaluation is presented, for example, in [49], again using the N-body algorithm. Both these devices have been exploited in the ExaNoDe project to build a prototype of computing element for exascale [50].

However, it is to note that the porting of the code on these architectures is a bit more complex because the development and tuning tools have not yet reached the maturity level, ease of use, and does not provide the wide set of functionalities as those provided for free by Intel or NVIDIA [51].

*2.3. HPC Low-Power Devices.* If we move from off-the-shelf products to the design of new solutions for joining high performance and energy efficiency, one of the most important references is represented by the Mont-Blanc project, started in 2011. Its goal is to foster the development of a low-

power European processor for Exascale, with a target of 50 GFlops/W at the processor level. This project is part of the European Processor Initiative, a Framework Partnership Agreement to develop the European skills in the design and exploitation of such processors.

Also, this project, together with ExaNoDe [52], is part of a wider group of EU-funded projects (e.g., ExaNeSt [53] focused on interconnection and storage and Ecoscale [54] focused on the heterogeneous architecture and, in particular, on the use of FPGAs), pursuing a strategic vision for economical, low-power approaches.

Also, the Mont-Blanc projects consider the use of ARM instruction set architectures (ISAs), such as the ThunderX processor family [55], and quantitative evaluations about different energy-performance tradeoffs achievable when designing an architecture based on mobile market technologies have been presented [56].

Heterogeneity seems to represent the most promising way, e.g., by integrating CPUs (X86 or ARM), GPUs, and FPGA in a single platform [57]. Also, the great efforts in developing unified programming models and API supporting all these heterogeneous hardware architectures such as OpenCL, SYCL, and oneAPI [58] demonstrate this trend.

### 3. Tools for Energy-Efficient Computing

In the previous section, we saw that power and energy consumption had become the driving metrics for computing hardware design and the most interesting CPUs. However, the advances in hardware efficiency must be followed by energy-aware algorithms, appropriate choice and allocation of specific hardware to applications, and adequate management techniques.

One of the most complete and interesting introductions to the problem was presented by Prof. Gallaghers [59] in summer school "ICT-Energy: Energy consumption in future ICT device" organized in 2016 within the context of the ICT-Energy European project [60].

The key concept is that energy is consumed by hardware, but this occurs under the control of software. Normal high-level languages (e.g., C++ and Java) hide the hardware characteristics, but the key aspect is that there could be many differences in the same high-level code (e.g., C++) machine instruction programs with different energy consumption figures. To this extent, an interesting tool is represented by Compiler Explorer [61], an open-source web application for interactive compiler code generation observation based on Node.js [62]. It shows the assembly output of the compiled code with different compilers and compiler versions to extract valuable information as, for example, for evaluating the power consumption.

Therefore, energy saving has to start at the software level to be propagated to the hardware level. Techniques for saving energy with power-aware hardware management or power capping [63] described in the previous section can represent a valuable complement. However, a key aspect, neglected by nearly all programmers, is their active engagement to inspect where a program wastes energy and, therefore, experiment with different designs. This is

obviously coupled with the fact that results have to be produced within an acceptable deadline [64], an aspect often disregarded approaching the energy efficiency problem.

**3.1. Profiling Tools.** The first step for achieving energy-efficient behavior is to investigate software behavior using information gathered as a program executes (i.e., profiling it) or simulating this through a performance model.

One of the most used tools for profiling is the Performance API (PAPI) analysis library [65]. PAPI is platform independent and provides developers with an interface and methodology for gathering performance-related data made available by hardware. The basic principle is to allow developers to see the relation between the software performance and processor events. As regards the power consumption, PAPI has been extended to measure and report energy and power values also on complex architectures [66].

Also, the PowerPack framework [67] provides a set of tools for analyzing the energetic performance. Unlike PAPI, the measurements are gathered on a separate machine in order to limit probe effects.

The scalable performance measurement infrastructure for parallel codes (Score-P) [68] has been extended for collecting information from technologies such as the aforementioned Intel RAPL.

Extrac is a tool relying on PAPI that allows collecting its countermetrics (including power and thermal data) for parallel programs [37]. Paraver effectively supports the analysis of such information, a visual data browser developed at the Barcelona Supercomputing Center as the previous one [69].

The Energy-Aware COmputing Framework (EACOF) has been designed to allow developers to profile their code for energy consumption [70]. In particular, it allows profiling codes in order to know exactly where energy is being used. Moreover, it allows applications to adapt at runtime based on current energy consumption. As an example application, the authors proposed a video player that may intelligently adapt based on energy consumption readings to ensure a video will complete before the battery runs out. The framework is available on GitHub [71], but no updates have been published since 2015.

In general, many tools such as these two have been presented in the literature. It is worth citing EProf [72], having the main feature to support fine-grained attributions of energy consumption to a particular function/software segment. However, in most cases, they are not actively maintained at the end of the projects where they have been developed, and software becomes difficult—if not impossible—to find and run.

A similar fate occurred for the Multiple Metrics Modeling Infrastructure (MuMMI) [73] project, focused on integrating existing tools such as PAPI and PowerPack for facilitating measurement, modeling, and prediction of software for multicore systems.

**3.2. Dynamic Tuning.** Some tools aim to achieve energy-saving figures automatically. In detail, many of them have been proposed, e.g., [74, 75], but, as stated before, not actively maintained. Here, we present just two of them because they are not part of wider and integrated solutions, which are discussed below.

The Global Extensible Open Power Manager (GEOPM) is a framework for exploring power and energy optimizations targeting high-performance computing [76]. One of the most interesting features is the possibility to dynamically coordinate hardware settings across all compute nodes used by an application in response to the application's behavior and requests from the resource manager. For example, it is possible to optimize MPI applications to improve energy efficiency or reduce the effects of work imbalance, system jitter, and manufacturing variation through built-in or user-defined control algorithms. The framework is available on GitHub [77].

The COUNTDOWN Slack library [78] allows identifying and automatically reducing power consumption during communication and synchronization primitives [79]. The library faces the problem of power wasting in communication and synchronization operations because of the adopted blocking mechanisms [80]: for example, nearly all MPI implementations use a busy-waiting mechanism. This library, on the contrary, is able to run a processor in a low-power mode, resulting in lower power consumption with limited or no impact on the execution time [81].

**3.3. Integrated Solutions.** The Runtime Exploitation of Application Dynamism for Energy-efficient eXascale computing (READEX) project has been funded by the European Union's Horizon 2020 research program between 2015 and 2018 to develop a tool-aided methodology for dynamic auto-tuning for performance and energy efficiency [82]. The tool suite was released in 2018, and it is available via GitHub [83].

The methodology is based on instrumenting an application with Score-P. This can be performed in an automatic way by compiling it with Score-P. Then, the dynamism of the application is detected and analyzed in order to identify the significant regions that will be managed with the project tuning methodology at runtime.

The key advantage of this suite is that it can be exploited by any developer even if she/he is unaware of the READEX methodology, with the result of increasing the energy efficiency of her/his application. It has been estimated that the application of the READEX tool suite to a nearly complex application can take several days [84], mainly for compiling the application with Score-P.

The Low-Energy Toolset for Heterogeneous Computing (LEGaTO) project has been funded by the European Union's Horizon 2020 research program between 2017 and 2020 to design and develop a software toolchain for energy-efficiency computing on heterogeneous hardware, i.e., a system equipped with CPUs, GPUs, and FPGA [57, 85].

The toolchain was released in 2020, and it is available via GitHub [86]. It is composed by several software components integrated to achieve a consistent programming environment across heterogeneous hardware platforms.

The hearth of the toolchain is represented by OmpSs [87], an extension to OpenMP developed at the Barcelona Supercomputing Center for supporting the asynchronous parallelism on heterogeneous resources as multicore CPUs, GPUs, and FPGAs.

An application in the OmpSs programming model is composed of one or more tasks with possible data dependency flow among some of them. The runtime environment analyses the resulting graph and produces a correct and possibly concurrent order of task execution. Several compiler and runtime systems (e.g., Nanos6, XiTAO [88], and Mercurium) support the process and manage all the energy efficiency, security, and fault-tolerance aspects [89].

Three use cases have been defined in healthcare, IoT for Smart Homes and Cities, and machine learning because they have different requirements in terms of energy efficiency, fault tolerance, and security. Results have been published in Deliverable 5.4 [90].

The Software Development toolKit for Energy optimization and technical Debt elimination (SDK4ED) project has been funded by the European Union's Horizon 2020 research program between 2018 and 2020 to minimize the cost, the development time, and the complexity of low-energy software development processes by designing a methodological approach and a software toolchain [91].

The SDK4ED platform [92] consists of five toolboxes: Technical Debt Management, Energy Optimization, Dependability Optimization, Forecaster, and Decision Support. They are implemented following the microservice paradigm as Docker images containing the specific web service.

Focusing on the Energy toolbox, it analyses projects available in an online repository (e.g., GitHub) on the machine running the Docker container with regard to its energy efficiency. This means it finds the energy hotspots, estimates the energy consumption through static or dynamic analysis [93, 94], and inspects possible solutions by suggesting specific code refactoring. This is a valuable approach, in particular, for software reusing [95].

The project ended at the end of 2020. Therefore, at the time of writing, not all the details and the code are available.

The Time, Energy, and security Analysis for Multi/Many-core heterogeneous PLAtforms (TeamPlay) project has been funded by the European Union's Horizon 2020 research program since 2018 to design and develop new techniques for producing highly parallel software for low-energy systems, such as IoT devices and CPS [96].

The idea is to develop a set of tools for allowing programmers to reason about time, energy, and security at the program source level. The idea is to design new language constructs to manage these extrafunctional properties as first-class citizens of the source code and express contracts in the source code that are machine-checkable by an underlying proof system.

The project is ongoing; therefore, at the time of writing, little information and software components were available.

## 4. Conclusions

Energy consumption is increasingly becoming one of the most relevant issues concerning the computing platforms for scientific applications and workloads.

As stated in [97], the huge level of energy consumption of ICT systems is probably due to the fact that nobody really cared for a long time, but today, things are changing because of economic reasons and also because our way of thinking has changed.

In this paper, we presented state-of-the-art solutions, both hardware and software, and methodological approaches for pursuing energy efficiency in scientific software to provide interested readers an updated introduction to the topic. The conclusion we can derive is that there are an increasing number of projects focusing on these topics, and some interesting SoC-based solutions are available. From the software side, instead, the situation is not satisfactory because tools are sometimes difficult to be found, not integrated, and, very often, disappear after the end of the project that developed them. What is actually needed is the definition of a common methodology and a coordination effort of groups acting in this field comparable with that of the Virtual Institute-High-Productivity Supercomputing (VI-HPS) [98], having in mind the tradeoff among time-to-solution, energy-to-solution, and usability of the proposed tools.

## Data Availability

No data were used to support this study.

## Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

## References

- [1] J. Wu, S. Guo, H. Huang, W. Liu, and Y. Xiang, "Information and communications technologies for sustainable development goals: state-of-the-art, needs and perspectives," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2389–2406, 2018.
- [2] C. Magazzino, D. Porrini, G. Fusco, and N. Schneider, "Investigating the link among ict, electricity consumption, air pollution, and economic growth in eu countries," *Energy Sources, Part B: Economics, Planning, and Policy*, pp. 1–23, 2021.
- [3] M. Heikkurinen, S. Cohen, F. Karagiannis, K. Iqbal, S. Andreozzi, and M. Michelotto, "Answering the cost assessment scaling challenge: modelling the annual cost of European computing services for research," *Journal of Grid Computing*, vol. 13, no. 1, pp. 71–94, 2015.
- [4] G. Fagas, L. Gammaitoni, J. P. Gallagher, and D. Paul, *ICT-Energy Concepts for Energy Efficiency and Sustainability*, BoD-Books on Demand, Norderstedt, Germany, 2017, <https://www.intechopen.com/books/ict-energy-concepts-for-energy-efficiency-and-sustainability>.
- [5] P. Kogge, S. Borkar, D. Campbell et al., "Exascale computing study: technology challenges in achieving exascale systems,"

- Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Technical Representative*, vol. 15, no. 1, 2008.
- [6] 2021, <https://www.top500.org/green500/>.
- [7] T. R. Scogland, C. P. Steffen, T. Wilde et al., “A power-measurement methodology for large-scale, high-performance computing,” in *Proceedings of the 5th ACM/SPEC international Conference on Performance Engineering*, pp. 149–159, Dublin, Ireland, March 2014.
- [8] A. Shehabi, S. Smith, D. Sartor et al., *United States Data Center Energy Usage Report*, Lawrence Berkeley National Laboratory (LBNL), Berkeley, CA, USA, 2016.
- [9] A. Shehabi, S. Smith, E. Masanet, and J. G. Koomey, “Data center growth in the United States: decoupling the demand for services from electricity use,” *Environmental Research Letters*, vol. 13, no. 12, p. 124030, 2018.
- [10] D. D’Agostino, A. Clematis, A. Galizia et al., “The DRIHM project: a flexible approach to integrate HPC, grid and cloud resources for hydro-meteorological research,” in *Proceedings of the SC’14: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 536–546, IEEE, New Orleans, LA, USA, November 2014.
- [11] J. Wu, S. Rangan, and H. Zhang, *Green Communications: Theoretical Fundamentals, Algorithms, and Applications*, CRC Press, Boca Raton, FL, USA, 2016.
- [12] F. Chiappori, I. Merelli, L. Milanese, and A. Marabotti, “Static and dynamic interactions between GALK enzyme and known inhibitors: guidelines to design new drugs for galactosemic patients,” *European Journal of Medicinal Chemistry*, vol. 63, pp. 423–434, 2013.
- [13] D. Corrada, F. Viti, I. Merelli, C. Battaglia, and L. Milanese, “myMIR: a genome-wide microRNA targets identification and annotation tool,” *Briefings in Bioinformatics*, vol. 12, no. 6, pp. 588–600, 2011.
- [14] J. M. P. Cardoso, J. G. F. Coutinho, and P. C. Diniz, “High-performance embedded computing,” in *Embedded Computing for High Performance*, J. M. Cardoso, J. G. F. Coutinho, and P. C. Diniz, Eds., pp. 17–56, Morgan Kaufmann, Boston, MA, USA, 2017, <http://www.sciencedirect.com/science/article/pii/B9780128041895000028>.
- [15] D. D’Agostino, L. Morganti, E. Corni, D. Cesini, and I. Merelli, “Combining edge and cloud computing for low-power, cost-effective metagenomics analysis,” *Future Generation Computer Systems*, vol. 90, pp. 79–85, 2019.
- [16] C. Conficoni, A. Bartolini, A. Tilli, L. Benini, and G. Tecchiolli, “Energy-aware cooling for hot-water cooled supercomputers,” in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1353–1358, IEEE, Grenoble, France, March 2015.
- [17] G. Fagas, J. P. Gallagher, G. Luca, and D. J. Paul, “Energy challenges for ICT,” in *ICT—Energy Concepts for Energy Efficiency and Sustainability* IntechOpen, London, UK, 2017.
- [18] M. Capra, R. Peloso, G. Masera, M. R. Roch, and M. Martina, “Edge computing: a survey on the hardware requirements in the internet of things world,” *Future Internet*, vol. 11, no. 4, p. 100, 2019.
- [19] J. Wu, S. Guo, J. Li, and D. Zeng, “Big data meet green challenges: greening big data,” *IEEE Systems Journal*, vol. 10, no. 3, pp. 873–887, 2016.
- [20] J. Wu, S. Guo, J. Li, and D. Zeng, “Big data meet green challenges: big data toward green applications,” *IEEE Systems Journal*, vol. 10, no. 3, pp. 888–900, 2016.
- [21] P. Czarnul, J. Proficz, and A. Krzywaniak, “Energy-aware high-performance computing: survey of state-of-the-art tools, techniques, and environments,” *Scientific Programming*, vol. 2019, p. 8348791, 2019.
- [22] O. Vysocky, L. Riha, and A. Bartolini, “Overview of application instrumentation for performance analysis and tuning,” in *Parallel Processing and Applied Mathematics*, R. Wyrzykowski, Ed., Springer International Publishing, Cham, Switzerland, pp. 159–168, 2020.
- [23] D. D’Agostino, A. Quarati, A. Clematis et al., “Soc-based computing infrastructures for scientific applications and commercial services: performance and economic evaluations,” *Future Generation Computer Systems*, vol. 96, pp. 11–22, 2019.
- [24] Y. Li, T. Jiang, K. Luo, and S. Mao, “Green heterogeneous cloud radio access networks: potential techniques, performance trade-offs, and challenges,” *IEEE Communications Magazine*, vol. 55, no. 11, pp. 33–39, 2017.
- [25] X. Cao, L. Liu, Y. Cheng, and X. Shen, “Towards energy-efficient wireless networking in the big data era: a survey,” *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 303–332, 2018.
- [26] J. Murray, P. Wettin, P. P. Pande, and B. Shirazi, “Dynamic voltage and frequency scaling,” in *Sustainable Wireless Network-on-Chip Architectures*, J. Murray, Ed., Morgan Kaufmann, Boston, MA, USA, pp. 79–105, 2016.
- [27] D. Horak, L. Riha, R. Sojka, J. Kruzik, and M. Beseda, “Energy consumption optimization of the total-FETI solver and BLAS routines by changing the CPU frequency,” in *Proceedings of the 2016 International Conference on High Performance Computing Simulation (HPCS)*, pp. 1031–1032, Innsbruck, Austria, July 2016.
- [28] G. Papadimitriou, A. Chatzidimitriou, D. Gizopoulos et al., “Exceeding conservative limits: a consolidated analysis on modern hardware margins,” *IEEE Transactions on Device and Materials Reliability*, vol. 20, 2020.
- [29] E. Calore, A. Gabbana, S. F. Schifano, and R. Tripicciono, “Evaluation of DVFS techniques on modern HPC processors and accelerators for energy-aware applications,” *Concurrency and Computation: Practice and Experience*, vol. 29, no. 12, p. e4143, 2017.
- [30] 2021 <https://www.kernel.org/doc/html/v4.14/admin-guide/pm/cpufreq.html>.
- [31] V. Spiliopoulos, S. Kaxiras, and G. Keramidas, “Green governors: a framework for continuously adaptive DVFS,” in *Proceedings of the 2011 International Green Computing Conference and Workshops*, pp. 1–8, IEEE, Orlando, FL, USA, July 2011.
- [32] C. Scordino, L. Abeni, and J. Lelli, “Real-time and energy efficiency in Linux,” *ACM SIGAPP Applied Computing Review*, vol. 18, no. 4, pp. 18–30, 2019.
- [33] S. Catalán, J. R. Herrero, E. S. Quintana-Ortí, and R. Rodríguez-Sánchez, “Energy balance between voltage-frequency scaling and resilience for linear algebra routines on low-power multicore architectures,” *Parallel Computing*, vol. 73, pp. 28–39, 2018.
- [34] D. Lo and C. Kozyrakis, “Dynamic management of turbo-mode in modern multi-core chips,” in *Proceedings of the 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 603–613, IEEE, Orlando, FL, USA, February 2014.
- [35] S. Pandruvada, *Running Average Power Limit*, 01 STAFF, Intel Open Source.org, Santa Clara, CA, USA, 2014, <https://01.org/blogs/2014/running-average-power-limit--rapl>.
- [36] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan, “Power-management architecture of the intel

- microarchitecture code-named sandy bridge,” *IEEE Micro*, vol. 32, no. 2, pp. 20–27, 2012.
- [37] F. Mantovani and E. Calore, “Performance and power analysis of HPC workloads on heterogenous multi-node clusters,” *Journal of Low Power Electronics and Applications*, vol. 8, no. 2, p. 13, 2018.
- [38] K. Kasichayanula, D. Terpstra, P. Luszczek et al., “Power aware computing on GPUs,” in *Proceedings of the 2012 Symposium on Application Accelerators in High Performance Computing*, pp. 64–73, IEEE, Argonne, IL, USA, July 2012.
- [39] C. Lefurgy, K. Rajamani, F. Rawson, W. Felner, M. Kistler, and T. W. Keller, “Energy management for commercial servers,” *Computer*, vol. 36, no. 12, pp. 39–48, 2003.
- [40] I. Ratković, N. Bežanić, C. S. Ūnsal, A. Cristal, and V. Milutinović, “An overview of architecture-level power-and energy-efficient design techniques,” *Advances in Computers*, vol. 98, pp. 1–57, 2015.
- [41] P. M. M. Pereira, P. Domingues, N. M. M. Rodrigues, G. Falcao, and S. M. M. Faria, “Assessing the performance and energy usage of multi-CPU’s, multi-core and many-core systems: the MMP image encoder case study,” *International Journal of Distributed and Parallel Systems*, vol. 7, no. 5, pp. 1–20, 2016.
- [42] D. R. Patnaik Patnaikuni, “A comparative study of arduino, raspberry pi and esp8266 as iot development board,” *International Journal of Advanced Research in Computer Science*, vol. 8, no. 5, 2017.
- [43] P. B. Otte and D. Djukanovic, “A cost effective and reliable environment monitoring system for HPC applications,” *CoRR*, vol. abs/1802, p. 00724, 2018.
- [44] R. Okazaki, T. Tabata, S. Sakashita et al., “Supercomputer Fugaku Cpu A64fx realizing high performance, high-density packaging, and low power consumption,” *Fujitsu Technical Review*, <https://www.fujitsu.com/global/documents/about/resources/publications/technicalreview/2020-03/article03.pdf>, 2020.
- [45] E. Danovaro, A. Clematis, A. Galizia, G. Ripepi, A. Quarati, and D. D’Agostino, “Heterogeneous architectures for computational intensive applications: a cost-effectiveness analysis,” *Journal of Computational and Applied Mathematics*, vol. 270, pp. 63–77, 2014.
- [46] <http://www.cosa-project.it/>.
- [47] D. Cesini, E. Corni, A. Falabella et al., “Power-efficient computing: experiences from the cosa project,” *Scientific Programming*, vol. 2017, p. 7206595, 2017.
- [48] I. Merelli, L. Morganti, E. Corni et al., “Low-power portable devices for metagenomics analysis: fog computing makes bioinformatics ready for the internet of things,” *Future Generation Computer Systems*, vol. 88, pp. 467–478, 2018.
- [49] D. Goz, G. Ieronymakis, V. Papaefstathiou et al., “Performance and energy footprint assessment of FPGAs and GPUs on HPC systems using astrophysics application,” *Computation*, vol. 8, no. 2, p. 34, 2020.
- [50] P. Y. Martinez, Y. Beilliard, M. Godard et al., “Exanode: combined integration of chipllets on active interposer with bare dice in a multi-chip-module for heterogeneous and scalable high performance compute nodes,” *2020 IEEE Symposium on VLSI Technology*, pp. 1–2, 2020.
- [51] A. Armejach, “Porting the mont-blanc 2020 applications to teh arm isa and SVE,” Tech. Rep. D3.5, MONT-BLANC Project, Ile-de-France, France, 2020.
- [52] A. Rigo, C. Pinto, K. Pouget et al., “Paving the way towards a highly energy-efficient and highly integrated compute node for the exascale revolution: the exanode approach,” in *Proceedings of the 2017 Euromicro Conference on Digital System Design (DSD)*, pp. 486–493, IEEE, Vienna, Austria, September 2017.
- [53] M. Katevenis, N. Chrysos, M. Marazakis et al., “The exanest project: interconnects, storage, and packaging for exascale systems,” in *Proceedings of the 2016 Euromicro Conference on Digital System Design (DSD)*, pp. 60–67, IEEE, Limassol, Cyprus, September 2016.
- [54] I. Mavroidis, I. Papaefstathiou, L. Lavagno et al., “Ecoscale: reconfigurable computing and runtime system for future exascale systems,” in *Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 696–701, IEEE, Dresden, Germany, March 2016.
- [55] A. Armejach, M. Casas, and M. Moretó, “Design trade-offs for emerging HPC processors based on mobile market technology,” *The Journal of Supercomputing*, vol. 75, no. 9, pp. 5717–5740, 2019.
- [56] A. Adria, C. Marc, and M. Miquel, “Design trade-offs for emerging HPC processors based on mobile market technology,” *The Journal of Supercomputing*, vol. 75, no. 9, pp. 5717–5740, 2019.
- [57] B. Salami, K. Parasyris, A. Cristal et al., “Legato: low-energy, secure, and resilient toolset for heterogeneous computing,” in *Proceedings of the 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 169–174, IEEE, Grenoble, France, March 2020.
- [58] 2021, <https://khr.io/tr>.
- [59] 2021, <https://www.nipslab.org/files/summerschool-aalborg-jpg-part1.pdf>.
- [60] K. Eder and J. Gallagher, “Energy-aware software engineering,” *ICT-Energy Concepts for Energy Efficiency and Sustainability*, pp. 103–127, 2017.
- [61] 2021, <https://repo.hca.bsc.es/epic/>.
- [62] M. Godbolt, “Optimizations in c++ compilers,” *Queue*, vol. 17, no. 5, pp. 69–100, 2019.
- [63] C. Jin, B. R. De Supinski, D. Abramson et al., “A survey on software methods to improve the energy efficiency of parallel computing,” *The International Journal of High Performance Computing Applications*, vol. 31, no. 6, pp. 517–549, 2017.
- [64] A. Quarati, A. Clematis, and D. D’Agostino, “Delivering cloud services with QOS requirements: business opportunities, architectural solutions and energy-saving aspects,” *Future Generation Computer Systems*, vol. 55, pp. 403–427, 2016.
- [65] D. Terpstra, H. Jagode, H. You, and J. Dongarra, “Collecting performance data with PAPI-C,” in *Tools for High Performance Computing 2009*, pp. 157–173, Springer, Berlin, Germany, 2010.
- [66] H. McCraw, J. Ralph, A. Danalis, and J. Dongarra, “Power monitoring with PAPI for extreme scale architectures and dataflow-based programming models,” in *Proceedings of the 2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 385–391, IEEE, Madrid, Spain, September 2014.
- [67] R. Ge, X. Feng, S. Song et al., “Powerpack: energy profiling and analysis of high-performance systems and applications,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 5, pp. 658–671, 2009.
- [68] A. Knüpfer, C. Rössel, D. A. Mey et al., “Score-p: a joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir,” in *Tools for High Performance Computing 2011*, pp. 79–91, Springer, Berlin, Germany, 2012.
- [69] A. Munera, S. Royuela, G. Llort et al., “Experiences on the characterization of parallel applications in embedded systems

- with extrae/paraver,” in *Proceedings of the 49th International Conference on Parallel Processing-ICPP*, pp. 1–11, Edmonton, Canada, August 2020.
- [70] H. Field, G. Anderson, and K. Eder, “Eacof: a framework for providing energy transparency to enable energy-aware software development,” in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pp. 1194–1199, Gyeongju, Republic of Korea, March 2014.
- [71] 2021 <https://github.com/eacof/eacof>.
- [72] S. Schubert, D. Kostic, W. Zwaenepoel, and K. G. Shin, “Profiling software for energy consumption,” in *Proceedings of the 2012 IEEE International Conference on Green Computing and Communications*, pp. 515–522, IEEE, Besancon, France, November 2012.
- [73] X. Wu, C. Lively, V. Taylor et al., “Mummi: multiple metrics modeling infrastructure,” in *Proceedings of the 2013 14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pp. 289–295, IEEE, Honolulu, HI, USA, July 2013.
- [74] B. Rountree, D. K. Lowenthal, B. R. De Supinski et al., “Adagio: making DVS practical for complex HPC applications,” in *Proceedings of the 23rd International Conference on Supercomputing, ICS '09*, Yorktown Heights, NY, USA, June 2009.
- [75] A. Marathe, P. E. Bailey, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. De Supinski, “A run-time system for power-constrained HPC applications,” in *Lecture Notes in Computer Science*, vol. 9137, pp. 394–408, Springer, Berlin, Germany, 2015.
- [76] J. Eastep, S. Sylvester, C. Cantalupo et al., “Global extensible open power manager: a vehicle for HPC community collaboration on co-designed energy management solutions,” in *High Performance Computing*, J. M. Kunkel, Ed., Springer International Publishing, Cham, Switzerland, pp. 394–412, 2017.
- [77] 2021, <https://github.com/geopm/geopm>.
- [78] 2021, <https://github.com/EEESlab/countdown>.
- [79] D. Cesarini, A. Bartolini, A. Borghesi, C. Cavazzoni, M. Luisier, and L. Benini, “Countdown slack: a run-time library to reduce energy footprint in large-scale MPI applications,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 11, pp. 2696–2709, 2020.
- [80] M. Torquati, D. De Sensi, G. Mencagli, M. Aldinucci, and M. Danelutto, “Power-aware pipelining with automatic concurrency control,” *Concurrency and Computation: Practice and Experience*, vol. 31, no. 5, p. e4652, 2019.
- [81] D. Cesarini, A. Bartolini, P. Bonfà et al., “Countdown: a run-time library for performance-neutral energy saving in MPI applications,” *IEEE Transactions on Computers*, vol. 70, 2020.
- [82] J. Schuchart, M. Gerndt, P. G. Kjeldsberg et al., “The readex formalism for automatic tuning for energy efficiency,” *Computing*, vol. 99, no. 8, pp. 727–745, 2017.
- [83] 2021 <https://github.com/readex-eu>.
- [84] L. Riha, *D5.3: Evaluation of the READEX Tool Suite Using the READEX Test-Suite*, READEX Project, Naples, FL, USA, 2018, <https://www.readex.eu/wp-content/uploads/2018/11/D5.3.pdf>.
- [85] D. Gizopoulos, G. Papadimitriou, A. Chatzidimitriou et al., “Modern hardware margins: CPUs, GPUs, FPGAs recent system-level studies,” in *Proceedings of the 2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 129–134, IEEE, Rhodes, Greece, July 2019.
- [86] 2021, <https://github.com/legato-project>.
- [87] A. Duran, E. Ayguadé, R. M. Badia et al., “Ompss: a proposal for programming heterogeneous multi-core architectures,” *Parallel Processing Letters*, vol. 21, no. 2, pp. 173–193, 2011.
- [88] M. Pericàs, “Elastic places: an adaptive resource manager for scalable and portable performance,” *ACM Transactions on Architecture and Code Optimization*, vol. 15, no. 2, 2018.
- [89] K. Givaki, B. Salami, R. Hojabr et al., “On the resilience of deep learning for reduced-voltage FPGAs,” in *Proceedings of the 2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pp. 110–117, IEEE, Västerås, Sweden, March 2020.
- [90] 2021 <https://legato-project.eu/sites/default/files/uploaded/d5.4.pdf>.
- [91] L. Papadopoulos, C. Marantos, G. Digkas et al., “Interrelations between software quality metrics, performance and energy consumption in embedded applications,” in *Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems, SCOPES '18*, Sankt Goar, Germany, May 2018.
- [92] 2021, <https://gitlab.seis.iti.gr/sdk4ed-wiki/wiki-home/wikis/home>.
- [93] M. Axling, *D3.2 Suitable Monitor Indicators for Energy Consumption*, SDK4ED ProjectCentre of Research & Technology – Hellas (CERTH), Marousi, Greece, 2019, <https://drive.google.com/file/d/1zkX71Efl2ybfWTzPNUPKTHhb145-DzuH/view>.
- [94] D. Tsoukalas, *D3.4 Forecasting Methods for TD/Energy/Dependability*, SDK4ED ProjectCentre of Research & Technology – Hellas (CERTH), Marousi, Greece, 2019, <https://drive.google.com/file/d/1DVhM9JvSD3LsSVXIE9SfBrBXWobHMSXT/view>.
- [95] N. Nikolaidis, G. Digkas, A. Ampatzoglou, and A. Chatzigeorgiou, “Reusing code from stackoverflow: the effect on technical debt,” in *Proceedings of the 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA'19)*, IEEE, Kallithea, Greece, August 2019.
- [96] A. M. Coutinho Demetrios, D. De Sensi, A. F. Lorenzon et al., “Performance and energy trade-offs for parallel applications on heterogeneous multi-processing systems,” *Energies*, vol. 13, no. 9, p. 2409, 2020.
- [97] S. D’Elia, “Powering up: energy and computing,” *HiPEAC Info*, vol. 59, 2020.
- [98] 2021, <https://www.vi-hps.org/about/about.html>.

## Research Article

# MCAF: Developing an Annotation-Based Offloading Framework for Mobile Cloud Computing

Yilian Zhou,<sup>1</sup> Ligang He ,<sup>2</sup> Bin Wang,<sup>3</sup> Yi Su,<sup>2</sup> and Hao Chen<sup>1</sup>

<sup>1</sup>College of Computer Science and Electronic Engineering, Hunan University, Changsha, China

<sup>2</sup>Department of Computer Science, University of Warwick, Coventry, UK

<sup>3</sup>ZTE Corporation, Shenzhen, China

Correspondence should be addressed to Ligang He; [ligang.he@warwick.ac.uk](mailto:ligang.he@warwick.ac.uk)

Received 29 November 2019; Revised 24 February 2020; Accepted 7 May 2020; Published 16 July 2020

Academic Editor: Daniele D'Agostino

Copyright © 2020 Yilian Zhou et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Offloading computation from mobile to remote cloud servers is a promising way to reduce energy consumption and improve the performance of mobile applications. However, a great challenge arises as automatic integration of powerful computing resources in remote cloud infrastructure and the portability of mobile devices. In this paper, we develop a Java annotation-based offloading framework, called MCAF, for android mobile devices. This framework is designed and committed to simplifying the development of android applications enabled with the offload capability. All the developers need to do is to import the SDK library of our MCAF and annotate the computation-intensive methods. MCAF can automatically extract the annotated source code and generate the code that will be run in the Cloud. Moreover, the codes of making the offloading decisions are automatically inserted into the original source code. We also conducted the real experiments to show the applicability of our MCAF.

## 1. Introduction

Advances in the portability and capability of mobile devices such as smartphones, together with increasingly faster and widespread wireless networks, a large number of mobile APPs such as shopping, gaming, information management, and so on have been made. Smartphones have changed human lives and become indispensable gadgets in modern society. Despite the fast increase in hardware performance of mobile devices, it is still limited comparing with their desktop counterparts and cannot meet the ever-increasing demand from end-users and APP developers (e.g., CPU, storage, and battery life). The limited hardware resources impede further improvement of Quality of Services (QoS) [1] for users and also further expansion of mobile APP categories. Integrating mobile devices with powerful Cloud platform [2, 3] to provide the Mobile Cloud Computing (MCC) offers a promising solution to overcoming the challenge [4, 5].

MCC is able to save the energy consumption of mobile devices and/or improve the running performance of the

mobile applications by offloading part of executions from mobile devices to cloud servers. Although various offloading frameworks have been developed, a big obstacle to hinder the wide adoption of these offloading frameworks is that it relies heavily on the programmers to develop the mobile applications in a manner designated in the offloading framework. In this work, we aim to design and develop an offloading framework which simplifies the development of mobile applications. Our offloading framework, called MCAF, makes use of Java annotation, which is different from the existing ones in the following aspects.

In all existing offloading frameworks, the programmers either have to annotate the source code or write the source code in a specific manner, and the framework developers need to develop the bespoke compiler to compile the source code embedded with the offloading ability. To use MCAF, there is no need to develop a new compiler, and only the existing Java compiler is needed to realize the offloading process.

Moreover, most existing frameworks such as Cuckoo [6] require the programmers to implement the code to be run on



the Cloud. In contrast, MCAF aims to minimize the effort of conducting the offload. The programmers who employ MCAF do not need to implement the code running on the Cloud but write the source code as normal for the mobile device and add the Java annotations that specify the code offloading. MCAF automatically extracts from the source code the part that is to be run on the Cloud according to the Java annotation. The extracted code is then encapsulated automatically as a stand-alone Java program and is offloaded to the Cloud when the conditions are met.

Another major difference between MCAF and most existing frameworks is that MCAF offloads the task at the granularity of the method level. MCAF extracts the source code of a method that is to be offloaded and then wraps up the method as a stand-alone program.

Overall, MCAF aims to reduce the efforts of the framework developers and the programmers in realizing the offloading capability in mobile Apps. All that the users of MCAF have to do is to import the MCAF SDK library and add annotation for computation-intensive methods. In this paper, we present the design and implementation of our offloading framework. In order to showcase the applicability of our approach, we also developed two mobile applications for real-life scenario by applying our offloading framework. Experiments have also been conducted to evaluate the effectiveness of our framework.

Note that in practice confidentiality and the integrity of the transmitted data should be ensured. There are the existing measures in the literature [7, 8] to address the security issues in the offloading framework. For example, in the work presented in [7], the data are encrypted and signed before transmitting, and are then decrypted in the Cloud. The encryption and decryption costs are taken into account when making the offloading decisions. Ensuring security is not the focus of this paper. MCAF can make use of the existing security measure in the literature (such as the one presented in [4]) to ensure confidentiality and the integrity.

The remainder of this paper is organized as follows. In Section 2, we provide the background of Java annotation and the related work. In Section 3, we present the design of MCAF and also a case study to show the exact workings of MCAF. MCAF is evaluated in Section 4. This paper is concluded in Section 5.

## 2. Background

*2.1. Annotation.* Annotation is an important feature in Java since it relieves Java developers from the pain of cumbersome configurations [9]. It shifts the responsibility of writing the boilerplate Java code from the programmer to the compiler. Annotations were first introduced in Java 5.0 and more advanced features are gradually supported in later Java versions.

Annotation is a form of metadata and can be used to describe any information about the elements (packages, classes, methods, fields, arguments, variables, etc.) in a program [10]. Annotations do not directly affect the program semantics. However, since annotation does affect the way in which the programs are treated by tools and libraries,

they can in turn affect the semantics of the running program. Annotations can be read from source files, class files statically, or be read reflectively at runtime. For example, annotations, such as `Deprecated`, `Override`, or `NotNull`, can be used to describe the constraints or the usages of the elements in a method during the compilation.

Although typically an application programmer never have to define an annotation type, it is not hard to do so. The declaration of an annotation type is similar to that of a normal interface. An at-sign (@) precedes the keyword “interface.” The declaration of a method will also define the elements of the annotation type. The declaration of a method must not have any parameters and can have default values. Return types are restricted to primitives, `String`, `Class`, `enum`, `annotation`, and arrays of these types. The following is an exemplar declaration of an annotation type. Once an annotation type is defined, program developers can use it to annotate source code. In general, annotations will be processed during the compilation. The command-line utility APT, an annotation processing tool, is used to find and execute the annotation processors based on the annotations inserted in the source files.

*2.2. Related Work.* Many researchers believe that combining mobile computing with clouds is a promising solution to overcome the battery limitation of smartphones and extend the performance of smartphones [11]. Indeed, much recent work has focused on building the frameworks that enable the offloading of mobile computations to the cloud [12, 13].

Cuervo et al. proposed an offloading system called MAUI, which can enable fine-grained energy-aware code offloading from smartphone to the remote server [14]. In MAUI, a method that is possible to be offloaded is declared as a “remotable” method. MAUI then models the offloading problem as an integer programming problem and finds the optimal offloading decision by solving the integer program. Microsoft.NET Common Language Runtime (CRL) is the programming language used by MAUI. However, the drawback of MAUI is that it needs the developers to modify the source code run on the local mobile and implement the source code run in the Cloud.

Compared with MAUI, CloneCloud [15, 16] goes one step further and does not ask the programmer to label (e.g., declare) the offloadable methods. CloneCloud automatically identifies the offloading costs by analyzing the source code both statically and dynamically at runtime. It then runs an optimizer to partition the tasks between mobile devices and the Cloud.

MAUI and CloneCloud are the systems for offloading parts of the existing program running on the mobile device to the Cloud. Cuckoo [6, 17] is the application development framework. Cuckoo can be used by programmers to develop the mobile Apps that have the offloading ability. Cuckoo provides a simple programming model and allows a single interface with a local and a remote (on the Cloud) implementation. The Apps developed using Cuckoo will decide at runtime automatically whether the local or the remote implementation is invoked for an interface.

DPSF [18] is another offloading framework based on Java. It first analyses the call-graph of the application offline and determines which classes can be offloaded. It then partitions all Java classes into two parts during the compilation. The classes in one part are run locally in the mobile phone while the other will be offloaded to run on the server. The entire application is deployed in both local mobile and remote server. The application starts running in local mobile. When it runs to a method in a class that is offloaded, it makes use of the RMI (Remote Method Invocation) mechanism in Java to invoke the method in the Java application deployed in the remote server. Although DPSF also makes use of the existing ability in Java to realize offloading, there are the following differences from our MCAF.

First, DPSF is essentially a static offloading framework because it, whose classes are offloaded, has to be determined at the programming stage through the offline profiling, and then the source code has to be adjusted to fulfil the offloading. DPSF is dynamic only in the sense that when it begins to run the application but detects that the network connection is poor, it will run the original Java application. Once the original application or the application with the offloading ability starts running, it will run to completion. There is no other mechanism in DPSF that makes the offloading decisions for individual classes during the application execution. In MCAF, although we annotate which methods can be offloaded at the programming stage, the offloading decisions can be made for a particular method. If the offloading condition is met, the offloading is then carried out for that method.

Second, DPSF offloads the workload at the granularity of the class level, while it is at the method level in MCAF.

Third, when using DPSF, the programmer has to change the source code, for example, changing the modifier of non-private fields to private and generating the public getter/setter methods for them, generating a proxy class for each offloaded class, and so on. In MCAF, the programmer only annotates the methods through Java annotation scheme, but does not change the original source code, which reduces the programmer's effort and is less error-prone during the development stage. The source code for running the offloaded methods in the Cloud server is automatically generated by MCAF, not written by the programmer. If the annotation is ignored, the application behaviour remains unchanged.

### 3. MCAF

**3.1. MCAF Modules.** The Mobile Cloud Annotation Framework (MCAF) consists of the following modules: (1) Annotation Handler: extracting all information related to the annotated methods; (2) Cloud Proxy: realizing the communications between the local mobile device and the Cloud; (3) Offloading Decider: implementing the offloading strategies; and (4) Code Rewriter: automatically wrapping up the source codes of the extracted method as stand-alone Java programs and compiling them as classes files.

**3.1.1. Annotation Handler.** APT is the existing annotation processing tool. In MCAF, APT is used to identify which

method is annotated and save the annotation information. The saved annotation information contains the accessing modifiers, the return type, and the parameters type of the annotated methods. However, the parameter values and the method code are not included. One of the responsibilities of Annotation Handler is to complement the annotation information saved by APT. Annotation Handler extracts the segment of the source code in each annotated method. Specifically, a regular matching expression is constructed in Annotation Handler, which is used to search the annotation information saved by APT and to match the part of source code which implements the annotated methods. The extracted code segment will be packaged as the stand-alone Java program and compiled to the class files by the Code Rewriter module. The class files will be transferred to and run on the Cloud according to the offloading decision made by Offloading Decider.

**3.1.2. Offloading Decider.** After the processing of Annotation Handler, all annotated methods are identified, and the annotation information and the source file of these methods are saved. When an annotated method is invoked, the Offloading Decider module calculates the offloading cost of this method. If it is beneficial to offload, the bytecode of this method, which is obtained by the Code Rewriter module, is retrieved and transferred to the cloud by the Cloud Proxy module. If the Offloading Decider decides not to offload, the method will be executed in the local mobile device.

Note that the offloading strategy is not the focus of this paper. MCAF is a framework to realize the offloading ability. The developer can plug any existing offloading strategy into the Offloading Decider. The Offloading Decider interfaces with other components in MCAF through its output, which is either True or False. If the output is true, the method in question will be offloaded. Otherwise, the method will be run locally. Algorithm 1 in Section 3.2 presents an example of this.

**3.1.3. Cloud Proxy.** The Cloud Proxy module is responsible for offloading-related communications between local mobile device and the Cloud. Firstly, when the Offloading Decider decides to offload an annotated method, Cloud Proxy transfers the bytecode and the arguments of the method to the Cloud. Second, the execution results of the offloaded method are returned to the local mobile device and passed to the invoked method.

**3.1.4. Code Rewriter.** Code Rewriter generates two types of source code (class). After Annotation Handler extracts the code segment of the annotated methods, Code Rewriter generates the stand-alone source code for each annotated method. The source code is then compiled to a Java class, whose name is the name of the annotated method appended by a word "Class." For example, if the name of the annotated method is "Add," the name of the new class is "AddClass." The generated classes may be transferred to and run on the cloud after the offloading decision is made. In the class, a member function is defined which includes all instructions of the annotated method. By invoking the member function of the newly generated class on

```

(1) public class A_ extends A {
(2)     int Add(int x, int y) {
(3)         isAddClassNeedUpload = Decider.getClassUploadStatus();
(4)         if (isAddClassNeedUpload) {
(5)             isAddClassRemoteExists = Decider.getRemoteExistsStatus();
(6)             if (!isAddClassRemoteExists) {
(7)                 ClientProxy.uploadClass("xx/xxx/AddClass.Class");
(8)                 return ClientProxy.getResult("Add", x, y);
(9)             }
(10)        }
        return x + y;
    }
}

```

ALGORITHM 1: Generating a new class “A\_” that extends from class “A” and then overwriting the “Add” method by inserting the offloading instructions.

the cloud, we will obtain the same results of the annotated method as it runs on the local smartphone. We call this type of class the *Cloud execution class*.

The second type of source code (class) generated by Code Rewriter is run in the local smartphone. In this source code, the new class inherits from the class where the annotated method is located. The name of this new class is the name of the inherited class appended by an underscore symbol “\_.” For example, if the inherited class is “A,” the name of the new class is “A\_.” Code Rewriter overrides the new class “A\_” by automatically inserting the code which makes the offload decisions and the code that interacts with the Cloud, such as uploading the “AddClass” generated above, and its input parameters to the Cloud and receiving the return results from the Cloud. If the decision made by the added decision making code is false (i.e., do not offload), the original instructions of the annotated method will be run in the local smartphone. We call this new class the *policy and local execution class*.

3.2. *A Case Study of MCAF.* In this subsection, we present an example to illustrate the workings of MCAF.

We first define a new annotation named “Upload” in Algorithm 2.

*Target* and *Retention* are two predefined annotation types in Java. The *Target* type with the value of *ElementType.METHOD* in line 1 is used to specify that the annotation is applied at the method level. The *Retention* type with the value of *RetentionPolicy.CLASS* in line 2 is used to specify that the added annotation is retained by the compiler at the compile time (but is ignored by the Java Virtual Machine). Line 3 defines a new annotation type called *Upload*. Defining a new annotation type is similar as defining an interface in Java except that the keyword *interface* is preceded by the sign @. In the body of the *Upload* annotation type, three annotation type elements are declared: *type*, *module*, and *valueType*. The *type* element is used to identify the *Upload* annotation. The *module* element is used to specify the module in which the annotated method is located (assume the annotated method is in the “app” module). *valueType* is the data type of the input parameters of the annotated method. We developed an Annotation.jar package. The “Upload” annotation type is defined in

Annotation.jar. Annotation.jar also implements the functionality of the Code Rewriter component.

After the Upload annotation type is defined, we can use it to annotate a method. In Algorithm 3, a class with the name of “A” is defined, which includes an “Add” method. The “Add” method takes two input parameters with the integer type, *x* and *y*. Before the “Add” method, the newly defined “Upload” annotation type is inserted, in which the *valueType* element takes the actual types (i.e., integer) of the two input parameters, *x* and *y*. The other two elements, “type” and “module,” take the default values.

When the code in Algorithm 3 is compiled, the compiler realizes that the “Add” method is annotated by “Upload.” Two types of source code (two classes) will be generated.

On one hand, the compiler will invoke the Code Rewriter functionality implemented in Annotation.jar to generate a new class named “AddClass” as shown in Algorithm 4. If the offload decider decides to offload the “Add” method, the “AddClass” will be uploaded to the Cloud for execution.

On the other hand, the compiler will also invoke the Code Rewriter functionality to generate another new class named “A\_” which extends from class “A” (the “Add” method is located in class “A”). Then the “Add” method is overwritten by inserting a set of instructions shown in Algorithm 1 (from line 4 to line 10) in the “Add” method. In Algorithm 1, line 3 calls the offload decider to determine whether the “Add” method should be uploaded. If the output of the Offload Decider is True (checked in line 4), the method needs to be offloaded and lines 5–8 will be run. Lines 5 and 6 check whether the “Add” method has been offloaded (the App may be executed repetitively). If it has been offloaded, it means that the code of the “Add” method exists in the Cloud and there is no need to upload the code of the method again. If the method has not been offloaded before, it calls the Cloud Proxy to upload the “AddClass” (line 7), which contains the code of the “Add” method that will be run in the Cloud, and then wait for the Cloud to return the result of running the “Add” method (line 8). If the out of the Offload Decision is False, the calculation is performed locally (line 10).

When the Cloud server receives the “AddClass.Class” uploaded by the smartphone, it invokes the “Add” method in the AddClass.Class through the Java reflection mechanism. The segment of code that run the “Add” method in the Cloud

```

(1) @Target(ElementType.METHOD)
(2) @Retention(RetentionPolicy.CLASS)
(3) public @interface Upload {
(4)     String type() default "Upload";
(5)     String module() default "app";
(6)     String[] valueType() default {};
(7) }

```

ALGORITHM 2: Defining a new annotation type named "Upload."

```

public class A extends MainActivity {
    @Upload(valueType = {"int", "int"})
    int Add(int x, int y) {
        return x + y;
    }
}

```

ALGORITHM 3: Annotating the "Add" method in the "A" class.

```

package org.cloud.annotations.MainActivity;
public class AddClass {
    public int Add(int x, int y) {
        return x + y;
    }
}

```

ALGORITHM 4: Generating the "AddClass" to be run on the Cloud.

```

(1) Object[] parameters = new Object[]{x, y};
(2) Class[] parametersType = new Class[]{int.class, int.class};
(3) Object[] parameters = (Object[]) entityBean.parametersValue.toArray();
(4) Class[] parametersType = entityBean.parametersType.toArray(new Class[entityBean.parametersType.size()]);
(5) Class<?> cl = Class.forName("org.demo.data.AddClass");
(6) Method method = cl.getDeclaredMethod("add", parametersType);
(7) Object object = cl.newInstance();
(8) ret = (int) method.invoke(object, parameters);

```

ALGORITHM 5: The Cloud server executes the "Add" method in the AddClass.Class through the Java reflection scheme.

is shown in Algorithm 5, in which  $x$  and  $y$  will take the values uploaded by the smartphone.

**3.3. System Architecture.** In this section, we present the system architecture for implementing the Android application with the offload capability. First, we introduce what type of methods can be annotated using our scheme. Android applications run their bytecode on the Dalvik Virtual Machine (Dalvik VM). There are some differences between Dalvik VM and Java Virtual Machine (JVM). Dalvik VM has optimized the execution of the android code. For example, UI graphic, camera, and sensors cannot be executed on JVM because their execution depends on the Android Runtime Environment. The methods which can be annotated must

meet the following conditions: (1) the annotated methods can only contain the pure java code. Note that the percentage of the code that is offloaded will not be significantly reduced due to this condition since the methods that are most likely to be offloaded are computation-intensive ones and all arithmetic operations can be programed in pure Java code. (2) The annotated methods should not use the global variables. If the global variables have to be used, they need to be transferred to the Cloud server together with other method parameters. (3) The annotated methods have to be public methods.

**3.3.1. Build Process.** Figure 1 shows the process of building an Android application using our framework from the source code to the apk file. The only thing that the developers



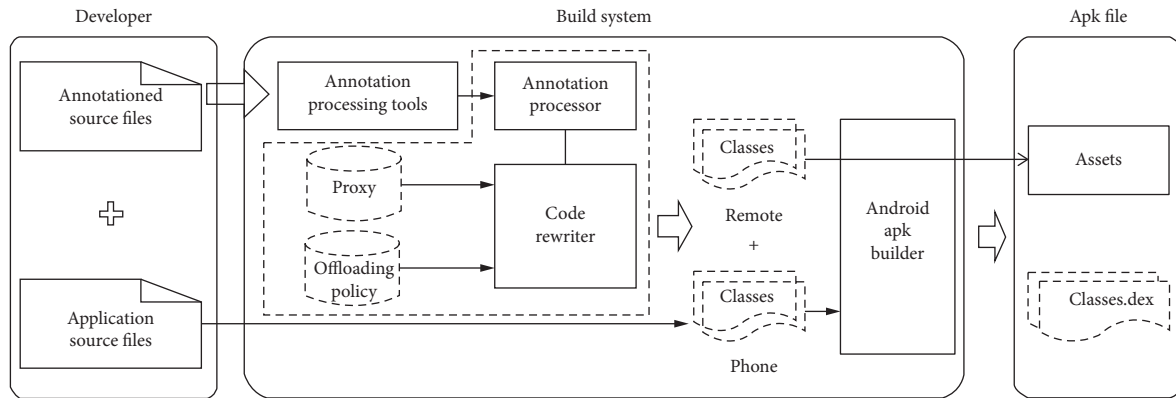


FIGURE 1: The build process of an android application with MCAF.

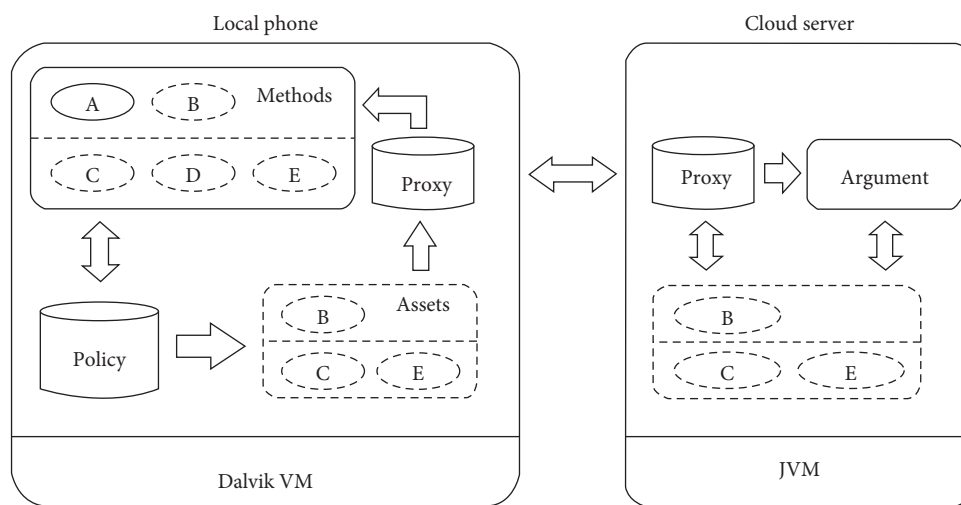


FIGURE 2: The runtime architecture of android application generated by the framework.

need to do is to add the annotation, which divides the source files into two categories: annotation source files and the original source files. In this process, Annotation Process Tools (APT) can find all annotated methods in annotated source files. Then, it generates the *Cloud execution* classes that can be executed on the Cloud and the classes that run on the local smartphone through the framework SDK.

**3.3.2. Runtime Architecture.** Figure 2 shows the runtime architecture of the application. In the smartphone, the runtime architecture is mainly divided into four parts. The first part is Methods, which contains the methods that will be executed when the android application runs. In Figure 2, there are two classes in the methods part, which are divided by the dotted line. Methods A and B are in the same class, while methods C, D, and E are in the other. In addition, we use the dotted ellipse to represent the annotated methods, which are methods B, C, and E. The second part is Policy, which makes the offload decisions for the annotated methods. The third part is assets; it is a resource folder and contains the source code of the annotated methods, which can be uploaded and executed on the Cloud. The last part is Proxy; it is responsible for communication with the cloud. In

the Cloud server, the module argument is used to receive the arguments from local smartphone when an annotated method is executed in the Cloud.

In the Cloud server, the argument module is responsible for managing the transmission of the parameters between the smartphone and the server. When an annotated method is invoked in the local phone during runtime, the method will wrap the parameter values into the serialized file stream, which is transmitted to the server through the network and is deserialized to the parameters value on the server. By using the serialization, we can transmit the complex type parameters easily in addition to basic types of data. On the Cloud server, we allocate the memory space to store the executable files uploaded from the smartphone. These files are invoked through the Java reflection mechanism. The required parameters during the execution are provided by the argument module. When the Cloud server obtains the execution results, we will use the serialization mechanism to pass the results to the smartphone.

## 4. Evaluation

The aim of developing the offload framework is to save the energy consumption of the smartphone where the mobile

TABLE 1: The execution time of the matrix computation application.

Power $n$	Offloading (ms)		Local phone (ms)	Speedup (local_run/offloading_run)
	Transmission	Cloud		
20	237	61	6912	23.19
40	232	77	13742	44.47
60	232	102	20610	61.71
80	214	128	27468	80.32
100	210	152	34551	95.44

TABLE 2: The execution time of the Image Segmentation application.

Times	Offloading (ms)		Local phone (ms)	Speedup (phone/offloading)
	Transmission	Remote		
2	969	20	3727	3.85
4	984	27	7485	7.61
6	1003	29	11201	11.17
8	1065	43	15022	14.11
10	1067	93	18710	17.54

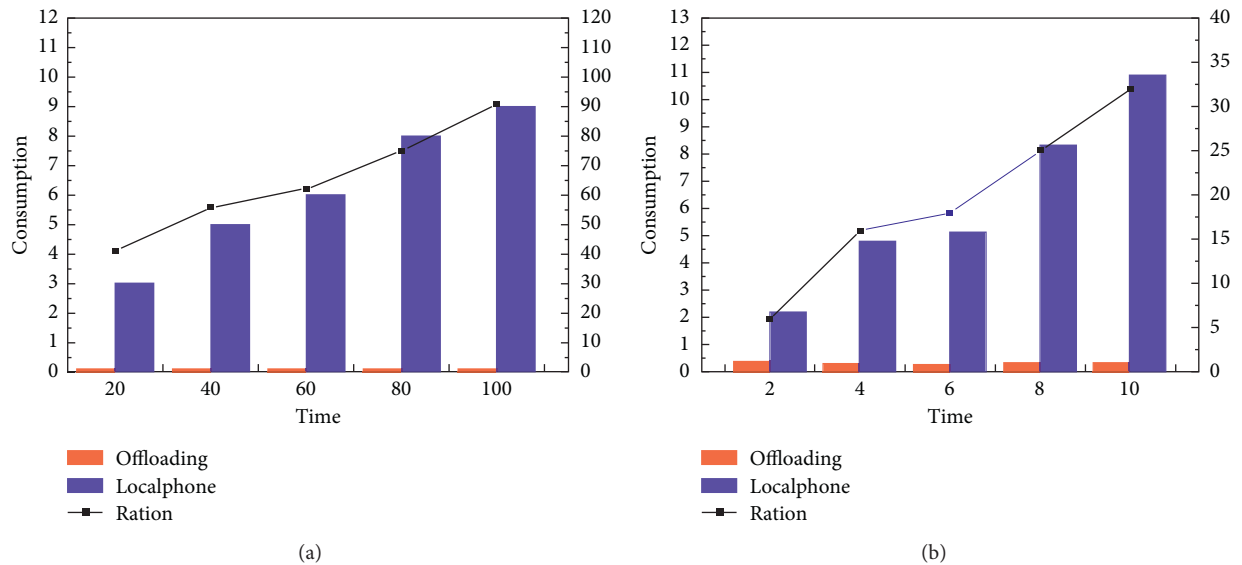


FIGURE 3: (a) The energy consumption of the matrix computation application; the  $x$ -axis is the number of times the matrix multiplication is performed (i.e., the power  $n$ ). (b) The energy consumption of the image segmentation application; the  $x$ -axis is the number of times the image segmentation is performed.

App is running while maintaining the performance of the App. In this section, we evaluate the performance and power consumption of our MCAF. We conducted real experiments on a smartphone, which is Vivo X20 with 1.8 GHz CPU, octa core processors, 4 GB RAM and Android OS 7.1.1. We used a desktop as a Cloud server, which runs ubuntu 14.04 with a 3.40 GHz CPU and 16 GB RAM. The Cloud server and smartphone are interconnected by wifi.

We implemented two applications and ran them either on the local phone or on the cloud server through offloading. The first application is to perform matrix computation, which is a computational intensive application. In this application, we generated the square matrix with the size of  $100 \times 100$ . The

elements of the matrix are random numbers between  $-10$  and  $10$ . The application computes the  $n$ -th Power of the matrix. First, we ran the application on the smartphone. Then, we ran the application by offloading it to the Cloud server. We also measured the power consumption of the smartphone and the execution time of the application in both cases. How to design smart offloading strategies is not the focus of this work. In the experiments, the application is always offloaded when the wireless connection is available.

**4.1. Execution Time.** Table 1 lists the execution times of the application when running in the smartphone and when

offloading. The offloading execution time consists of 2 parts, including the communication time of uploading the code and the parameters to the Cloud and the execution time on the Cloud server. As the table shows, the communication time is relatively stable. This is because the data transmitted and the network speed during the offloading process change little. The execution time on the Cloud server increases gradually as the power  $n$  (i.e., the number of matrix multiplication) increases. We also calculate the speedup of running through offloading over running in the smartphone. As the computation increases, the speedup increases too. The results show that our offload framework can significantly improve the running performance of the application.

The second application we used in the experiments is image segmentation, which partitions a digital image into multiple segments. It is also a computation-intensive application. The execution times of the application in local phone and in the Cloud server are shown in Table 2. The input size of the test image is  $300 \times 300$  pixel. As the table shows, the application incurs more communication time compared to the matrix computation application. The results still show good speedup when the application is offloaded.

**4.2. Energy Consumption.** To evaluate the energy consumption of these applications, we used software called Trepro Profiler [19]. It is an on-target power and performance profiling application for mobile devices. In the Trepro Profiler, the power measurement for a single app can be achieved with a feature called Show Deltas. Figures 3(a) and 3(b) show the energy consumption of these two applications when they are executed locally and through offloading. It can be observed from the figure that offloading offers significant energy saving. The amount of energy saved increases as the computation size increases.

## 5. Conclusion

This paper presents a Java annotation-based offload framework called MCAF for mobile cloud computing. MCAF is used to upload the annotated source code to the Cloud server at the granularity of methods. By using MCAF, developers do not need to implement the Cloud-side service, which can be generated automatically by MCAF during the compilation. The codes of making the offloading decisions are also automatically inserted. All that the developers need to do is to import the SDK library of our MCAF and annotate the computation-intensive methods. We conducted the real experiments to showcase the applicability of MCAF.

It is possible to extend this work to allow the offloading of the Android-related methods (i.e., the methods that are not programmed in pure Java code, but call the functions in Android-related libraries). In order to realize the offloading of android-related code, the following work should be conducted: (i) extracting the offloaded method and generating a stand-alone apk that encapsulates the offloaded method; (ii) setting up a VM in the Cloud and deploy the Dalvik VM environment in the VM; (iii) developing a

functional component in the Cloud VM to enable the communication between the Dalvik VM and the mobile. The development of the above framework is expected to involve much more engineering work than deploying a standard Java VM as in the current MCAF. We will carry out careful investigations as to whether it is worth the effort. After all, only computation-intensive tasks are most likely to be offloaded, which can be coded in pure Java code.

MCAF is originally designed for offloading the workload from smartphones to the Cloud server. We plan to explore the application of MCAF in fog-edge computing. In fog-edge computing, data processing is moved to edge devices, which are closer to where the data are generated compared with Cloud computing. As long as the edge devices are more powerful than smartphones, there is no reason why MCAF cannot be used to achieve the offloading of the workload from smartphones to edge devices in fog-edge computing.

## Data Availability

The data supporting the results of this work are available upon request from the corresponding author.

## Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

## Acknowledgments

This research was partially funded by National Natural Science Foundation of China under Grants 61972137 and 61772183 and the Worldwide Byte Security Co. Ltd.

## References

- [1] J. Mei, K. Li, and K. Li, "Customer-satisfaction-aware optimal multiserver configuration for profit maximization in cloud computing," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 1, pp. 17–29, 2017.
- [2] K. Li, C. Liu, K. Li, and A. Y. Zomaya, "A framework of price bidding configurations for resource usage in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 8, pp. 2168–2181, 2016.
- [3] C. Liu, K. Li, C. Xu, and K. Li, "Strategy configurations of multiple users competition for cloud service reservation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 508–520, 2016.
- [4] H. Qi and A. Gani, "Research on mobile cloud computing: review, trend and perspectives," in *Proceedings of the 2nd IEEE International Conference on Digital Information and Communication Technology and Its Applications (DICTAP)*, pp. 195–202, IEEE, Bangkok, Thailand, May 2012.
- [5] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *Wireless Communications and Mobile Computing*, vol. 13, no. 18, pp. 1587–1611, 2013.
- [6] C. A. I. Long, K. K. H. Kunasekaran, V. Ramakrishnan et al., "Task offloading to the cloud by using cuckoo model for minimizing energy cost," in *Proceedings of the 2016 International Conference on Information Engineering, Management and Security (ICIEMS 2016)*, Tirupur, India, 2016.

- [7] B. Huang, Y. Li, Z. Li et al., "Security and cost-aware computation offloading via deep reinforcement learning in mobile edge computing," *Wireless Communications and Mobile Computing*, vol. 2019, Article ID 3816237, 20 pages, 2019.
- [8] V. Viduto, C. Maple, W. Huang, and D. López-Peréz, "A novel risk assessment and optimisation model for a multi-objective network security countermeasure selection problem," *Decision Support Systems*, vol. 53, no. 3, pp. 599–610, 2012.
- [9] W. Cazzola and E. Vacchi, "@Java: bringing a richer annotation model to Java," *Computer Languages, Systems & Structures*, vol. 40, no. 1, pp. 2–18, 2014.
- [10] <http://tutorials.jenkov.com/java/annotations.html>.
- [11] J. Liu, K. Li, D. Zhu, J. Han, and K. Li, "Minimizing cost of scheduling tasks on heterogeneous multicore embedded systems," *ACM Transactions on Embedded Computing Systems*, vol. 16, no. 2, p. 36, 2016.
- [12] Y. Wang, I.-R. Chen, and D.-C. Wang, "A survey of mobile cloud computing applications: perspectives and challenges," *Wireless Personal Communications*, vol. 80, no. 4, pp. 1607–1623, 2015.
- [13] M. A. Khan, "A survey of computation offloading strategies for performance improvement of applications running on mobile devices," *Journal of Network & Computer Applications*, vol. 56, pp. 28–40, 2015.
- [14] E. Cuervo, A. Balasubramanian, D. Cho et al., "MAUI: making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ACM, San Francisco, CA, USA, pp. 49–62, June 2010.
- [15] B. G. Chun, S. Ihm, P. Maniatis et al., "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the 6th Conference on Computer Systems*, ACM, Salzburg, Austria, pp. 301–314, April 2011.
- [16] L. P. R. Mali and L. G. D. Naik, "A review on distributed application processing framework-clone cloud," *International Journal of Engineering and Computer Science*, vol. 4, no. 1, 2015.
- [17] R. Kemp, N. Palmer, T. Kielmann et al., "Cuckoo: a computation offloading framework for smartphones," in *Proceedings of the 2010 International Conference on Mobile Computing, Applications, and Services*, pp. 59–79, Los Angeles, CA, USA, October 2010.
- [18] D. Kong, T. Qi, T. Yang et al., "A dynamic computation offloading framework for Android," in *Proceedings of the 5th IEEE International Conference on Broadband Network & Multimedia Technology*, pp. 134–138, IEEE, Guilin, China, November 2013.
- [19] <https://developer.samsung.com/game/trepn>.



## Research Article

# Deep Recurrent Neural Networks for Edge Monitoring of Personal Risk and Warning Situations

**Emanuele Torti** , **Mirto Musci**, **Federico Guareschi**, **Francesco Leporati**,  
and **Marco Piastra**

*Department of Electrical, Computer and Biomedical Engineering, University of Pavia, Pavia I-27100, Italy*

Correspondence should be addressed to Emanuele Torti; [emanuele.torti@unipv.it](mailto:emanuele.torti@unipv.it)

Received 1 September 2019; Accepted 5 November 2019; Published 13 December 2019

Guest Editor: Daniele D'Agostino

Copyright © 2019 Emanuele Torti et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Accidental falls are the main cause of fatal and nonfatal injuries, which typically lead to hospital admissions among elderly people. A wearable system capable of detecting unintentional falls and sending remote notifications will clearly improve the quality of the life of such subjects and also helps to reduce public health costs. In this paper, we describe an edge computing wearable system based on deep learning techniques. In particular, we give special attention to the description of the classification and communication modules, which have been developed by keeping in mind the limits in terms of computational power, memory occupancy, and power consumption of the designed wearable device. The system thus developed is capable of classifying 3D-accelerometer signals in real-time and to issue remote alerts while keeping power consumption low and improving on the present state-of-the-art solutions in the literature.

## 1. Introduction

Nowadays, unintentional falls are among the principal causes of fatal injuries. Moreover, they are the most common cause of hospitalization after nonfatal traumas. A study conducted by the World Health Organization [1] highlights that 25% of people aged over 65 years old fall every year, with an increment to 32%–42% if considering only over 70. Furthermore, even when the falls lead to less severe injuries, the associated discomfort significantly reduces the quality of life. It is worth noting that not only elderly people are affected by unintentional falls, every person with some kind of fragility related, for example, to postoperative conditions, disability, or any other disease affecting mobility are part of similar statistics. In addition, it is well known that the majority of unintentional falls happens in the home environment.

These facts highlight the importance of an automated system capable of detecting unintentional falls sending remote notifications so that timely help can be given. Among the different approaches described in the literature, embedded wearable devices are emerging as the best choice for

this kind of systems. This is mainly due to their low intrusiveness, reduced power consumption, and cost effectiveness. Moreover, the recent innovations on the microcontroller units (MCUs), which are typically used for wearable devices, provide the necessary computational power that enables them to perform complex computations directly on the MCU. This allows to implement more complex methods than in the past on-board wearable devices, thus effectively enabling a very specific kind of edge computing. Among these methods for fall detection, deep learning techniques recently showed to be a very promising approach [2, 3].

In this paper, we describe an embedded framework based on edge computing and on machine learning for fall detection on-board wearable devices. In particular, we extend our system proposed in [4] with a novel strategy to improve the overall system performance together with the design of a suitable Bluetooth Low Energy (BLE) protocol for an effective minimization of data transmission.

It is worth highlighting that the adopted methodology for this personal monitoring system based on deep learning methods is very general and could be reused also for

different applications beyond the reported one. As an example, deep learning methods on embedded devices can be used in different fields, such as automotive [5], security and surveillance [6], augmented reality [7], and healthcare [8].

The paper is organized as follows. In Section 2, the state of the art of both wearable devices and fall detection through machine/deep learning techniques is described. Then, the proposed system is presented and the classification and communication modules are described in detail, highlighting the reasons for the different design choices. These modules are then evaluated from different points of view, including computational complexity, power consumption, and memory occupancy in Section 3, evaluating how these modules impact on the performance of the overall system. Section 4 gives some final remarks and possible future research lines.

## 2. Materials and Methods

*2.1. State of the Art.* In the fall detection literature, two main trends can be identified: one is based on ambient monitoring, while the other harnesses wearable or portable devices. Ambient monitoring is typically based on cameras mounted in rooms. The main issues of this approach are the expensiveness, the high power consumption, and the intrusiveness of the system in terms of privacy. Portable and wearable devices do not suffer from these limitations. In the latter category, two main approaches can be found: one is typically based on smartphones [9–12] and another on several kinds of wearable devices with specific hardware [13–16]. As fall detection devices, smartphones suffer from several limitations, which affect the overall performance [17]. First, the sensors are shared and managed by the operating system in a pre-emptive fashion. In a smartphone, in fact, several applications are simultaneously executed and this implies that all sensors are shared among all the applications which need sensory data, some of which run at a high priority. Therefore, it is not possible in practice to achieve the guarantee of a fixed sampling frequency and this is a critical issue especially for artificial intelligence methods. Last but not the least, in general, the typical duration of the battery is not compatible with continuous monitoring during the entire day. On the contrary, wearable devices are designed for one specific task only, they have direct access which guarantees a fixed sampling frequency; therefore, battery duration can be made to be longer than with smartphone-based solutions.

A thorough analysis of the existing wearable system solutions for fall detection highlights two main research tracks. In the first track, we find on-board elaboration of sensory readings; typically, the acquired sensors data are filtered and then processed through techniques based on a fixed threshold or via other statistical methods, and then the results are sent to a remote device.

Cola et al. [13] proposed a head-worn device containing an accelerometer and a barometer integrated with a TI MSP430 MCU. Jung et al. [14] described a system including a three-axial accelerometer, an MCU, and a Bluetooth module. These components are attached to a jacket and are

connected to each other via stretchable conductive nylon. Nyan et al. [15] proposed a system using accelerometers and gyroscopes. In this work, sensors are connected via Zigbee transceivers to a board based on an Intel PXA255 processor, where the actual processing takes place. In another system [18], a custom processor based on FPGA technology elaborates the data acquired from the accelerometers.

It is worth noting that this kind of elaboration is simple and does not require high-computational capabilities. On the contrary, the accuracy of fixed threshold-based and statistical methods is not so high and is outperformed by artificial intelligence techniques.

On the second track, several works adopted more sophisticated methods, but by abandoning the on-board processing in favor of remote elaboration. In these approaches, the wearable device acquires data from the sensors and then sends them to a workstation that performs the elaboration. A relevant example is the SHIMMER (Sensing Health with Intelligence, Modularity, Mobility, and Experimental Reusability) integrated sensors platform [19]. In [20], the data acquired by the SHIMMER 3D accelerometers is sent via Bluetooth to a remote workstation, which performs the classification through a Support Vector Machine (SVM) classifier. Authors also compared the SVM with K-Nearest Neighbours (KNN) and complex trees. A similar study is conducted in [3], where gyroscopes and accelerometers data are processed by machine learning approaches.

Apart from the SHIMMER, other Commercial Off-the-Shelf (COTS) devices are emerging. In particular, the SensorTile board produced by STMicroelectronics is attracting the researchers because of its computing and memory capability together with low power consumption. The core of this device is the STM32L476JGY MCU with a maximum clock frequency of 80 MHz. The board is also equipped with two three-axial accelerometers, a magnetometer, a barometer, and a gyroscope. It also integrates a Bluetooth 4.1 transceiver, which is a popular protocol for IoT applications.

The mounted MCU is an ARM Cortex M4 core, equipped with a Floating-Point Unit (FPU) which is fully compliant with the IEEE single-precision floating-point standard. The board is also equipped with 1 MB of flash memory and 128 KB of SRAM. This device has been successfully employed in human activity recognition [21] and fall detection [4]. In the above perspective, the fall detection system presented in [4] performs data classification through deep learning methods elaborated on the device. This means that it outperforms the other devices in terms of accuracy because it adopts deep learning methods and reduces power consumption since the elaboration is performed on board, without the need for continuous data transfers. However, this system can be further optimized both from the point of view of computational complexity and power consumption. In this paper, we present a significant improvement of our previous work described in [4]; in particular, the system has been enriched with a pre-elaboration step which prevents the MCU from classifying data that does not contain relevant information. Moreover, a communication protocol based on the BLE standard has been designed in order to

minimize the communications between the SensorTile board and a remote host.

**2.2. Deep Learning.** Deep learning methods are currently the state-of-the-art approach for many computer vision and signal processing problems. In particular, to process time series signals (i.e., data acquired by sensors over time), Recurrent Neural Networks (RNNs) are considered the best solution [22]. Such networks are a specific kind of artificial neural network, in which part of the output is fed back as input.

Generally speaking, an RNN can be described by

$$y^{(t)} = wg(Wx^{(t)} + Uh^{(t-1)} + b) + c, \quad (1)$$

where  $x^{(t)}$  and  $y^{(t)}$  are the input and the output at the time  $t$ , respectively,  $W$ ,  $U$ ,  $w$ ,  $b$ , and  $c$  are the network parameters, and  $g$  denotes a nonlinear function. The term  $h^{(t)}$  indicates the hidden state, which is defined as follows:

$$h^{(t)} = g(Wx^{(t)} + Uh^{(t-1)} + b) + c. \quad (2)$$

The drawback of this kind of network is the training phase, which is very hard to perform both in theory and in practice. The standard training method for RNNs is *temporal unfolding*, where each training input sequence of predefined length is fed as input to the unfolded network. This technique is shown in Figure 1.

To analyze time series (e.g., accelerometers acquisitions over time) the input data stream is scanned through a sliding window of a suitable size, which must match the predefined level of unfolding for training.

Once the training is completed, the obtained RNN can be used to analyze an input stream by sliding a window of size  $\omega\omega$  over the input stream, resetting and rerunning the RNN for each input window. This process is known as *inference* and is used to recognize specific patterns in the input. To reduce the computational cost of inference, the input window  $\omega_i$  is typically slid at interval  $s \gg 1$  of constant length called *strides*. All the concepts related to the sliding window technique are shown in Figure 2.

Long Short-Term Memory (LSTM) [23, 24] cells are a particular kind of RNN which are able to detect and reproduce long-term temporal dependencies. They feature the capability to learn how to forget and filter part of their hidden state during the inference. The main advantage is that those networks are easier to train because they do not suffer from the so-called *vanishing gradient* problem. On the contrary, they are more complex than standard RNNs from the computational point of view. The behavior of an LSTM cell is described by the following equations:

$$c_{in}^{(t)} = \tanh(W_{xc}x^{(t)} + W_{hc}h^{(t-1)} + b_c), \quad (3)$$

$$i^{(t)} = \text{sigmoid}(W_{xi}x^{(t)} + W_{hi}h^{(t-1)} + b_i), \quad (4)$$

$$o^{(t)} = \text{sigmoid}(W_{xo}x^{(t)} + W_{ho}h^{(t-1)} + b_o + b_{\text{forget}}), \quad (5)$$

$$f^{(t)} = \text{sigmoid}(W_{xf}x^{(t)} + W_{hf}h^{(t-1)} + b_f), \quad (6)$$

$$c^{(t)} = f^{(t)}c^{(t-1)} + i^{(t)}c_{in}^{(t)}, \quad (7)$$

$$h^{(t)} = o^{(t)}\tanh(c^{(t)}), \quad (8)$$

where  $W \in \mathbb{R}^{LS \times LS}$ ,  $x^{(t)}$ ,  $h^{(t)}$ ,  $c_{in}^{(t)}$ ,  $i^{(t)}$ ,  $o^{(t)}$ ,  $f^{(t)}$ ,  $c^{(t)}$ ,  $b \in \mathbb{R}^{LS}$ .  $LS$  is a hyperparameter, called the *LSTM size*, and is defined upfront by design as constant among all cells.

Figure 3 shows the typical structure of an LSTM cell, where  $x^{(t)}$  and  $h^{(t-1)}$  are given as input for computing equations (3)–(6), indicated as circles in the figure. The small circles with a point inside indicate the element-wise multiplications needed for preparing the inputs for the evaluation of equations (7) and (8). The result of equation (8) is the output of the cell.

**2.3. Fall Detection with Deep Learning for Embedded System.** Different deep learning methods have been successfully used for fall detection [2, 25–28]. Analyzing those systems, we see that all these methods rely either on models with a huge number of parameters or on remote communication. In terms of potential criticalities, in the former approach, the set of binary parameters (usually called the *model* of a network) can easily become too heavy to be elaborated in real-time on a wearable device, while in the latter approach, if we consider a 24/7 monitoring, intensive data communication might well drain the battery charge too quickly [29].

The implementation of deep learning methods on an embedded system is a topic of current interest, as witnessed by the development of TensorFlow Lite [30]. This software is a reduced version of the complete TensorFlow software framework and can be executed on mobile and embedded devices. However, this software is still in an early development stage and at present has some limitations: first, only some MCUs are currently supported and, among the low-power microcontrollers, only the ARM Cortex M3 MCU is currently supported. Moreover, the part of the complete TensorFlow framework which is implemented is at present insufficient to implement LSTMs. To the best of the authors' knowledge, RNNs have been successfully deployed to an ARM Cortex M4 MCU only in our previous work [4], where we described a runtime inference module based on an RNN network. In this work, the communication module plus other energy-saving provisions were not discussed and, as we will see here, such improvements can be significantly beneficial to the whole embedded module.

**2.4. Architecture of the Proposed Edge Computing System.** Overall, the proposed software system can be divided into two main components: the first component is devoted to the offline training of the LSTM network, while the second relates to the real-time classification of sensor readings and the communication to a gateway of relevant events. The training of a deep network directly onboard with a typical MCU like the one used in this project is unfeasible due to

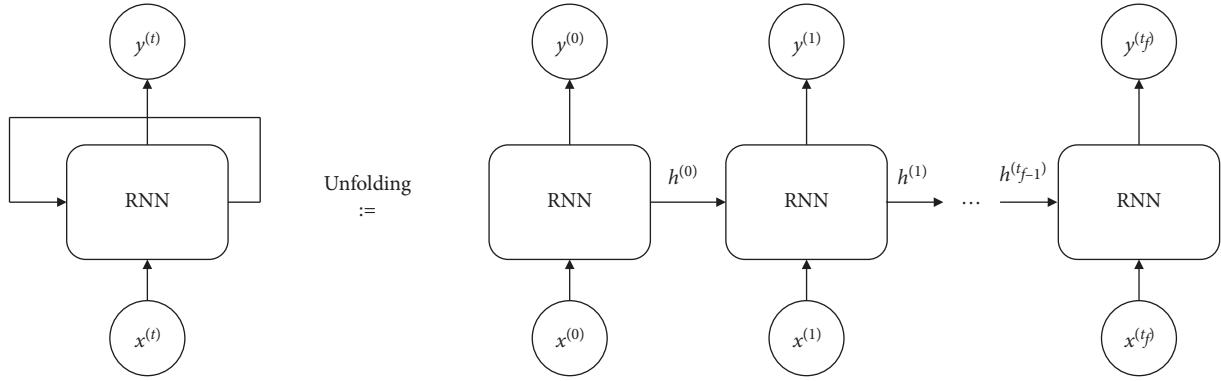


FIGURE 1: A single RNN cell is unfolded for the training.

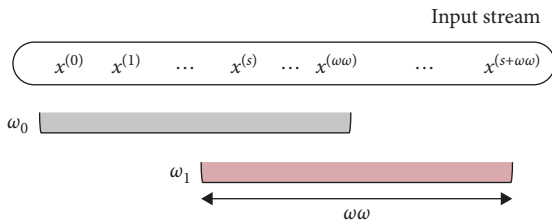


FIGURE 2: An example of sliding window  $\omega_i$  with width  $\omega\omega$  and stride  $s < \omega\omega$ .

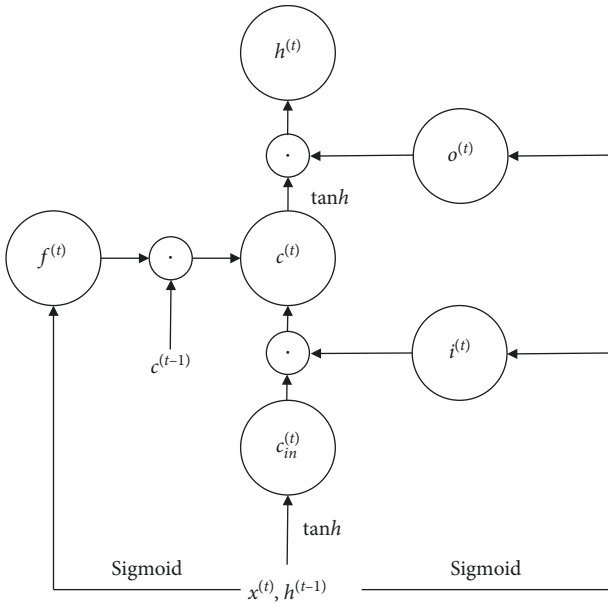


FIGURE 3: The structure of an LSTM cell.

memory and computational power constraints. Thus, network training must be performed off-board, on a workstation. In the proposed system, the training is performed on a Dell 5810 workstation using TensorFlow 1.8. The training set was collected during an extensive campaign conducted at the University of Pavia in which over 40 volunteers participated in recording sensory data while performing simulated activities and falls, according to a predefined protocol of 17 standard maneuvers. Each recording is associated to a

video sequence that describes the performed activity, and annotations are added subsequently by identifying and marking the actual time intervals in which specific events took place [31]. Once trained, the model can be deployed on the MCU.

In the literature, there are several strategies to optimize both memory occupancy and power consumption. One of the most popular techniques is integer quantization. This method requires the definition of a range of values for parameters and variables and uses 8-bit integer encoding for converting back and forth floating-point values into such range. The gain in memory occupancy is clear since every parameter or variable has a footprint four times lower than adopting floating point. However, from the computational point of view, quantization raises also some critical issues. In particular, the LSTM cell requires both linear and nonlinear operations. If the precision loss in linear operations is negligible, the nonlinear operations suffer from a substantial inaccuracy compared to the floating-point counterparts. In addition, depending on the approach adopted, quantization may require several bidirectional conversions between integers and floats, thus making the overall gain in terms of computational efficiency to be clearly assessed.

Another possible strategy which is very popular in custom hardware architectures is the fixed-point representation. In this technique, the data are represented using a subset of the bits of a word for the integer part and the remaining bits for the decimal part. In this way, the basic mathematical operations can be performed using only the integer arithmetic unit of the MCU. However, according to the experiments performed in our laboratory, the precision is again a critical issue because, also in this case, the nonlinear operations suffer from significant precision loss. We also evaluated a hybrid approach when the linear operations are performed in fixed-point format and the nonlinear functions adopt the floating-point format. In this case, a conversion between the two different numeric representations is mandatory. This leads to a significant loss in computational time, which does not allow to achieve the real-time constraint. Moreover, this solution performs even worse than a pure floating-point inference because the adopted MCU is equipped with a floating-point unit, which can perform basic arithmetic operations in only one clock cycle. Therefore, for those reasons, we adopted the single-



precision floating-point representation in our runtime module.

Once the LSTM network is deployed on the SensorTile device, the wearable system is ready to act as a wearable, intelligent fall detector. The device should be worn by the monitored subject and, at this point, it begins to acquire sensory data and to classify events. If the classifier detects a fall or a warning situation, the device sends via BLE a message to the gateway, which then forwards it to the cloud for the notification of the alert to the service actors that are designated to intervene.

Both the classifier module and the BLE protocol developed in this work are described in detail in the following sections.

The general architecture of the fall detection system is depicted in Figure 4.

The LSTM training is performed on a workstation using TensorFlow, and the model is then deployed on the MCU by uploading a custom firmware. Events classification is performed online and in real-time and, when dangerous situations are detected, the device issues a BLE message to a gateway that forwards this information to a remote monitor through the cloud.

**2.5. Inference Runtime Module on the MCU.** As a basis for our classifier module, we adopted the same network architecture described in [4]. Such an architecture includes two LSTM layers, two fully connected layers, and a *softmax* layer. The inner dimension of the LSTM cell is 32. In [4], this network architecture performs the classifying inference in real-time by considering a 1 second window width.

In the work described here, we have improved the overall classifying module by introducing a procedure for the preliminary detection of operating conditions, in particular, whether the device has been worn by the monitored subject, which engages the full classifying network only when sensor readings are compatible with the occurrence of a significant event. We also modified the initialization phase to perform some self-diagnostics and signal to the gateway of hardware malfunctions, if any. In particular, the system can detect hardware malfunctions related to the sensors, i.e., if the chip is not responding to the MCU requests. The flow chart of this extended classifying module is shown in Figure 5.

The first step is the initialization of the accelerometers, the BLE module, and the inference module. The latter module is the most interesting one since it is not a standard software routine. It manages all the memory allocations for the network variables and initializes the weights matrices by loading the values from the flash memory.

During the accelerometer initialization, the routine checks if there are hardware malfunctions and, in this case, issues a BLE message to the gateway.

When all the initializations have been performed correctly, the main loop begins.

At every second, a window containing the accelerometer readings is ready to be processed. The first step is to compute the variance of the accelerations. The ARM Cortex M family can exploit the ARM CMSIS library which includes several

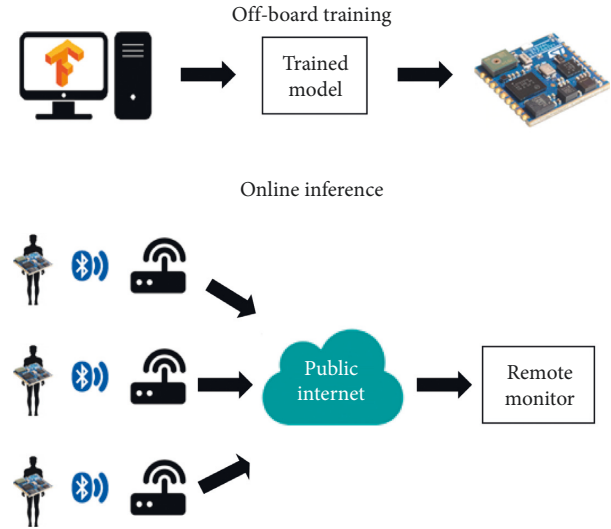


FIGURE 4: The architecture of the proposed system.

floating-point routines among which there is the variance computation [32]. However, the variance is computed there using the standard formula, which is not optimized. For this reason, we implemented the variance computation according to the Welford online algorithm:

$$M_{2,n} = \sum_{i=1}^n (x_i - \bar{x}_n)^2,$$

$$M_{2,n} = M_{2,n-1} + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n), \quad (9)$$

$$s_n^2 = \frac{M_{2,n}}{n-1},$$

where  $x_i$  is the  $i$ th sample and  $\bar{x}_n$  is the mean after  $n$  samples. This algorithm computes the variance inspecting each sample only once, avoiding to loop over the data to compute the mean of the sample window. The obtained variance is compared with two different thresholds. As said before, the first comparison is used to know if the device is worn by the monitored subject (*wear threshold*), while the latter detects if the dynamics of the signal is compatible with the occurrence of interesting events (*classify threshold*). These two thresholds have been experimentally estimated by recording the accelerometer data in different conditions. For the wear threshold, the device has been put on a flat surface in different positions, in order to record the accelerometer outputs at different orientations. After that, we evaluated the accelerometer readings when the device is worn by people performing daily activities such as walking, standing up, and sitting down.

If the variance is lower than the wear threshold, the device sends a BLE message to the gateway in order to signal that the device is not worn by the monitored subject. Otherwise, the second threshold is considered. If the dynamics of the reading is low, a more sophisticated and expensive classification is not necessary, since the occurrence of any relevant events can be simply ruled out. Otherwise, when the sensory dynamics are higher, readings are passed as input to the classification module, which acts as

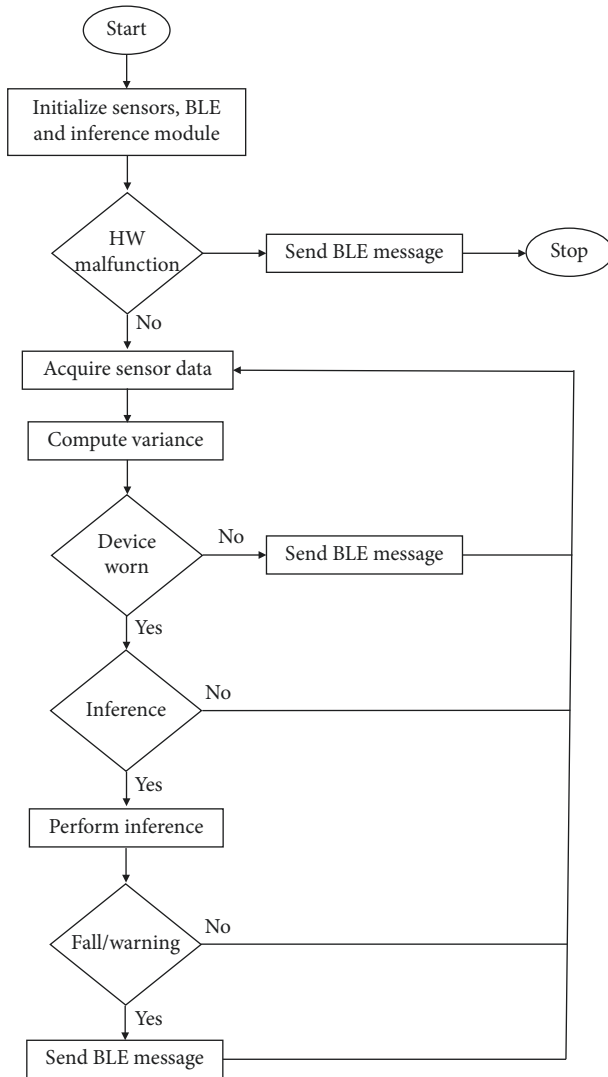


FIGURE 5: The flow chart of the classification module implemented on the ARM Cortex M4 MCU.

described in [4]. When the classification module detects in turn either a fall or a warning situation, the BLE module sends an alert message to the gateway, which notifies the service actors that are designed to intervene.

The firmware of the wearable device repeats these operations every second until the user switches off the system.

**2.6. Bluetooth Low Energy Protocol.** The communication between the wearable device and the gateway relies on the BLE protocol. This protocol is designed to transmit data only occasionally. Figure 6 shows the BLE protocol stack [33].

As can be seen from Figure 6, a generic BLE application is made up of three main components: the Application, the Host, and the Controller [34]. The Application is the highest level and contains all the logic and data handling.

The Host consists of the following layers:

- (i) Logical Link Control and Adaptation Protocol (L2CAP): it encapsulates data into BLE packets and

manages data fragmentation and recombination tasks.

- (ii) Attribute Protocol (ATT): it is a simple client/server protocol based on attributes presented by a device. A client requests data from a server, and the server then sends data to its clients.
- (iii) Generic Attribute Profile (GATT): it adds a data model and hierarchy defining how data are organized and exchanged between different applications.
- (iv) Generic Access Profile (GAP): it controls advertising and connections and specifies how devices perform control procedures such as device discovery, connection, and security levels.
- (v) Security Manager Protocol (SMP).

The Controller includes the following layers:

- (i) Link Layer (LL): it is in charge of establishing connections and filters out advertising packets depending on the Bluetooth address or based on the data itself
- (ii) Physical Layer (PHY): it contains the circuitry to modulate and demodulate analog signals and to convert them into digital signals

The GATT is the most important component to develop in order to design an effective protocol. It is organized in *Services*, each one containing one or more *Characteristics*. The BLE standard defines *Services* for the most common and general task of an application, such as the *Battery Service*, which includes the *Battery Level* characteristic, containing the charge percentage of the battery. Besides the default services, custom services and characteristics can be added to the GATT. This is of crucial importance for the fall detection system since the actual BLE standard does not include a service related to this task. For this reason, we defined a custom service with specific characteristics. Figure 7 shows a diagram of the GATT services included in our wearable device.

It is worth noticing that, in our wearable system, the BLE standard *Battery Service* coexists with the custom *Fall Service* we defined. The *Fall Probability* and *Warning Probability* characteristics are represented as unsigned 8 bit integers since their value ranges from 0 to 100. Also, the *Status* is an unsigned 8 bit integer. In this case, the least significant bit is used to signal if the device is worn or not, while the bit in position 1 is used for alerting in the case of hardware malfunctions. The *Wear Threshold*, *Classify Threshold*, and *Alert Threshold* are characteristics that can be written by the gateway in order to change the value of their respective variables used as a threshold in the inference module. These characteristics are represented as single-precision floating-point numbers. The *Postural Monitoring* is a long characteristic, which can have a maximum size of 512 bytes, according to the BLE standard. It is used to transmit 10 seconds of recording to the gateway, in order to monitor the accelerometer values which are related to the status of the monitored subject after a fall. Even if the BLE standard defines the maximum size of a long characteristic as

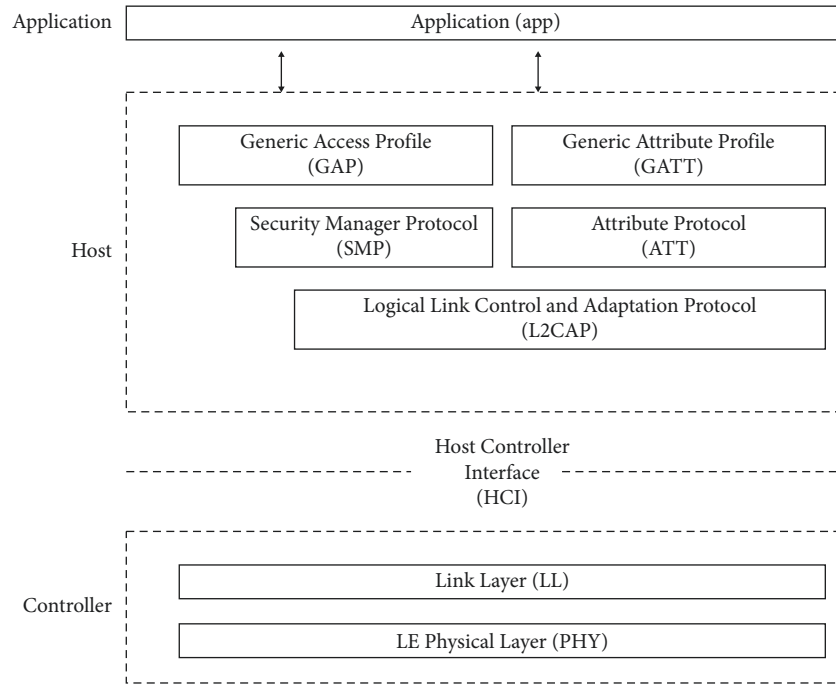


FIGURE 6: The BLE stack.

512 bytes, in the SensorTile this limit has been set to 256 bytes by the vendor. The 3D-accelerometer data are sampled at a frequency of 100 Hz; therefore, the total amount of data needed for the postural monitoring is 1200 bytes. Since this value exceeds the size limit of the device, we decided to under-sample the acquired data by a factor of 5, achieving a data size to transmit of 240 bytes, which is compatible with the restriction of our device.

### 3. Results and Discussion

In an offline computational validation, the embedded component has been tested against the TensorFlow results in order to ensure that the classification module produces the correct outputs. We tested the wearable system classification module by feeding different prerecorded signals sequences to the network as inputs. The results of the embedded classification module differed from those obtained with TensorFlow of about  $10^{-7}$ , which is a negligible error considering that the outputs are probability values ranging from 0 to 1. The classification achieves an accuracy of 90%. The total number of tracks acquired during the campaign performed at the University of Pavia is 18032. About 80% of those tracks have been used to train the system, while the remaining 20% as test set. It is worth noticing that the size of this database is much bigger than the size of other databases in the literature.

To evaluate the impact of the proposed classification module compared to the one described in [4], it is necessary to estimate the computational complexity of the Welford online algorithm, used for the variance estimation, which is the main modification to the runtime module, whereas the comparison between the variance and the threshold can be neglected since their computational weight is not

comparable to the whole network complexity. The Welford online algorithm requires  $30\omega\omega + 5$  FLOPS, which is significantly lower than the number of FLOPS needed by the data classification complexity reported in [4]. This consideration is confirmed by the experimental data, which highlighted that the differences in the elaboration times are negligible. On the other hand, the gain in computational time is evident when considering that the classification is performed only for certain signal windows. This means that also the power consumption decreases since fewer operations are needed and the MCU can be put into *sleep* mode for a longer time than performing the classification.

In addition, if we consider memory occupancy, the impact of the proposed module is negligible compared to the work in [4]. Indeed, the Welford algorithm and the BLE module require only some scalar variables, except for the postural monitoring, which needs a  $3\omega\omega$  long array of float elements, which is significantly smaller than the 82 kB taken by the network parameters.

Considering the BLE communication module, it uses the very low-power BLE single-mode network processor integrated in the SensorTile board. This network processor has a current consumption of about  $1.7 \mu\text{A}$  when the module is active but not transmitting to the gateway. On the other hand, the maximum drained current is about 8.2 mA, which is higher than the current consumption of the MCU when performing the classification (about 5 mA [4]). This is a critical issue because it limits the quantity of data that can be transmitted without draining the battery charge too fast. In fact, continuous data transmission will nearly reduce to one-third of the battery charge duration, since the absorbed current diminishes from 5 mA to about 13 mA. For this reason, the proposed communication protocol has been designed in order to transmit only alerts related to particular

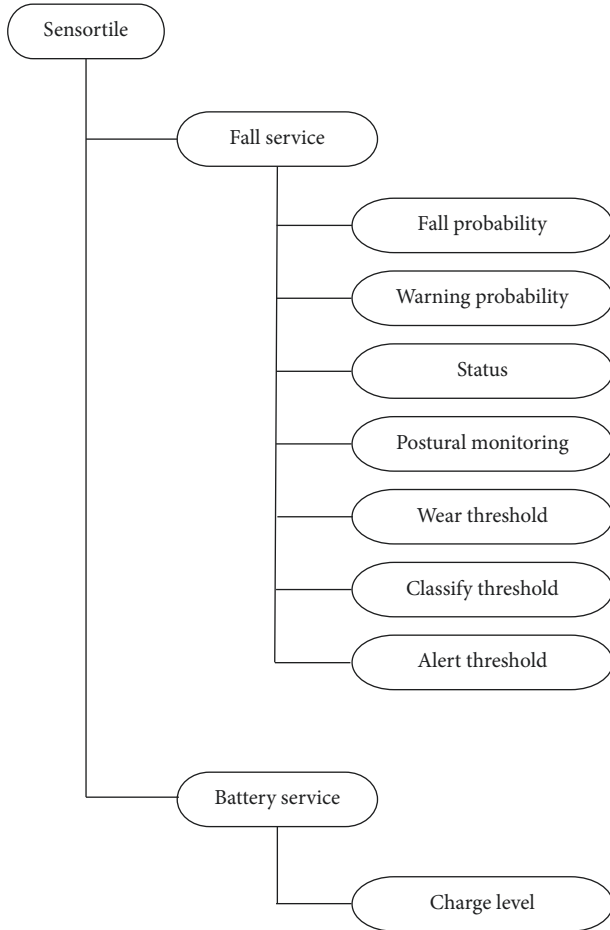


FIGURE 7: The GATT services and characteristics of our wearable system.

events. Moreover, the number of packets to transmit and receive to/from the gateway is minimal. When considering unsigned 8-bit integers, these data can fit in a single BLE packet, while float data require two BLE packets. The main limit of the proposed protocol is represented by the Postural Monitoring long characteristic, which requires 12 packets to be transmitted to the gateway. For this reason, the Postural Monitoring should be performed only when it is strictly necessary, in order to preserve the battery charge.

It is worth noticing that the transmitted data are related to events that are infrequent, therefore, the BLE radio is inactive for most of the time, keeping the BLU module current consumption negligible.

The improvements with respect to the work presented in [4] are summarized in Table 1.

Table 1 clearly shows that the proposed system improves and outperforms our previous work, both from the computational and power consumption point of view. The table shows both the best case and the worst case analysis of the proposed solution. The first is related to all those situations when the variance is below the inference threshold and therefore it is possible to save computational power by avoiding to evaluate the whole LSTM network. The latter is about data that should be analyzed by the LSTM network,

TABLE 1: Comparison between [4] and this work.

System	Processing time (ms)	Memory occupancy	Power consumption
[4]	342	82 kB	5 mA for 342 ms
This work best case	0.264	83.2 kB	5 mA for 0.264 ms and 1.7 $\mu$ A for BLE
This work worst case	$\sim$ 342	83.2 kB	5 mA for 342 ms and 8.2 mA for BLE

and a BLE message is sent to the remote host. It is worth noticing that the best case is the more frequent one, since the variance threshold has been estimated in order to avoid to perform inference on data related to normal daily living activities. Therefore, it is possible to say that the proposed system is operating in the best case conditions for the majority of the time, allowing a significant gain in power consumption. Moreover, this modification requires a negligible memory occupancy increase, as it can be seen from Table 1. Finally, this system is capable of communication with a remote host, through a BLE protocol that has been designed in order to minimize data transfers (again to reduce the impact on the power consumption).

#### 4. Conclusions

In this paper, we described the development of an edge-computing wearable device for personal monitoring exploiting deep learning methods, capable of detecting unintentional falls. In particular, we discussed the optimization of the real-time classification module embedded on the wearable device, together with the developed strategies to avoid unnecessary computations and to reduce power consumption.

Those strategies have been developed after analyzing the data collected during an extensive campaign conducted at the University of Pavia that allowed to carry out one of the biggest databases that can be found in the literature. In particular, the two thresholds used to avoid unnecessary computations have been defined after a careful analysis of this data.

We also described the developed BLE protocol, in order to minimize the communications between device and gateway, enabling a suitable alerting when specific events happen, without affecting the battery charge duration.

These results clearly improve on and further complete our previous system described in [4], enriching it with significant extensions. To the best of the authors' knowledge, this is the first edge computing wearable system for fall detection including such deep learning techniques and with the level of performance obtained.

Future works will be focused on integrating data from different kinds of sensors (i.e., barometers and/or gyroscopes) in order to improve the classification accuracy.

Moreover, the developed prototype could include different hardware architectures, with better potential support for the quantization of both network parameters and nonlinear operation processing. This is a further investigation line to explore for our research.



## Data Availability

The labeled Sisfall dataset used to support the findings of this study are available from the corresponding author upon request.

## Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

## Acknowledgments

This work received financial support from Regione Lombardia under the “Intelligent Personal Health Safety Domotic Monitoring” (IPHSDM) project (ID: 379273).

## References

- [1] WHO, *WHO Global Report on Falls Prevention in Older Age*, World Health Organization, Geneva, Switzerland, 2008.
- [2] F. J. Ordóñez and D. Roggen, “Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition,” *Sensors*, vol. 16, no. 1, p. 115, 2016.
- [3] M. Hemmatpour, R. Ferrero, B. Montrucchio, and M. Rebaudengo, *A Neural Network Model Based on Co-occurrence Matrix for Fall Prediction*, pp. 241–248, Springer, Cham, Switzerland, 2017.
- [4] E. Torti, A. Fontanella, M. Musci et al., “Embedded real-time fall detection with deep learning on wearable devices,” in *Proceedings of the 21st Euromicro Conference on Digital System Design, DSD 2018*, pp. 405–412, Prague, Czech Republic, August 2018.
- [5] F. Falcini, G. Lami, and A. M. Costanza, “Deep learning in automotive software,” *IEEE Software*, vol. 34, no. 3, pp. 56–63, 2017.
- [6] G. Sreenu and M. A. Saleem Durai, “Intelligent video surveillance: a review through deep learning techniques for crowd analysis,” *Journal of Big Data*, vol. 6, no. 1, 2019.
- [7] O. Akgul, H. I. Penekli, and Y. Genc, “Applying deep learning in augmented reality tracking,” in *Proceedings of the 12th International Conference on Signal Image Technology and Internet-Based Systems, SITIS 2016*, pp. 47–54, Naples, Italy, December 2017.
- [8] P. Teikari, R. P. Najjar, L. Schmetterer, and D. Milea, “Embedded deep learning in ophthalmology: making ophthalmic imaging smarter,” *Therapeutic Advances in Ophthalmology*, vol. 11, Article ID 251584141982717, , 2019.
- [9] S. Biswas, T. Bhattacharya, and R. Saha, “On fall detection using smartphone sensors,” in *Proceedings of the 2018 International Conference on Wireless Communications, Signal Processing and Networking, WiSPNET 2018*, pp. 1–4, Chennai, India, March 2018.
- [10] Y. Lee, H. Yeh, K. H. Kim, and O. Choi, “A real-time fall detection system based on the acceleration sensor of smartphone,” *International Journal of Engineering Business Management*, vol. 10, Article ID 184797901775066, , 2018.
- [11] J. A. Santoyo-Ramón, E. Casilari, and J. M. Cano-García, “Analysis of a smartphone-based architecture with multiple mobility sensors for fall detection with supervised learning,” *Sensors*, vol. 18, no. 4, p. 1155, 2018.
- [12] Y. W. Hsu, K. H. Chen, J. J. Yang, and F. S. Jaw, “Smartphone-based fall detection algorithm using feature extraction,” in *Proceedings of the 2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics, CISP-BMEI 2016*, pp. 1535–1540, Datong, China, October 2017.
- [13] G. Cola, M. Avvenuti, P. Piazza, and A. Vecchio, *Fall Detection Using a Head-Worn Barometer*, Springer, Cham, Switzerland, 2017.
- [14] S. Jung, S. Hong, J. Kim et al., “Wearable fall detector using integrated sensors and energy devices,” *Scientific Reports*, vol. 5, no. 1, p. 17081, 2015.
- [15] M. N. Nyan, F. E. H. Tay, and E. Murugasu, “A wearable system for pre-impact fall detection,” *Journal of Biomechanics*, vol. 41, no. 16, pp. 3475–3481, 2008.
- [16] D. Giuffrida, G. Benetti, D. De Martini, and T. Facchinetti, “Fall detection with supervised machine learning using wearable sensors,” in *Proceedings of the International Conference on Industrial Informatics*, Espoo, Finland, July 2019, In press.
- [17] R. Igual, C. Medrano, and I. Plaza, “Challenges, issues and trends in fall detection systems,” *BioMedical Engineering OnLine*, vol. 12, no. 1, p. 66, 2013.
- [18] S. Abdelhedi, M. Baklouti, R. Bourguiba, and J. Mouine, “Design and implementation of a fall detection system on a Zynq board,” in *Proceedings of the 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pp. 1–7, Agadir, Morocco, December 2016.
- [19] A. Burns, B. R. Greene, M. J. McGrath et al., “SHIMMER—a wireless sensor platform for noninvasive biomedical research,” *IEEE Sensors Journal*, vol. 10, no. 9, pp. 1527–1534, 2010.
- [20] F. Hossain, M. L. Ali, M. Z. Islam, and H. Mustafa, “A direction-sensitive fall detection system using single 3D accelerometer and learning classifier,” in *Proceedings of the 2016 International Conference on Medical Engineering, Health Informatics and Technology (MediTec)*, pp. 1–6, Dhaka, Bangladesh, December 2016.
- [21] A. Nicosia, D. Pau, D. Giacalone, E. Plebani, A. Bosco, and A. Iacchetti, “Efficient light harvesting for accurate neural classification of human activities,” in *Proceedings of the 2018 IEEE International Conference on Consumer Electronics, ICCE 2018*, pp. 1–4, 2018.
- [22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, USA, 2016.
- [23] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [24] F. A. Gers, “Learning to forget: continual prediction with LSTM,” in *Proceedings of the 9th International Conference on Artificial Neural Networks: ICANN’99*, pp. 850–855, Edinburgh, UK, September 1999.
- [25] T. Mauldin, M. Canby, V. Metsis, A. Ngu, and C. Rivera, “SmartFall: a smartwatch-based fall detection system using deep learning,” *Sensors*, vol. 18, no. 10, p. 3363, 2018.
- [26] A. Nait Aicha, G. Englebienne, K. van Schooten, M. Pijnappels, and B. Kröse, “Deep learning to predict falls in older adults based on daily-life trunk accelerometry,” *Sensors*, vol. 18, no. 5, p. 1654, 2018.
- [27] R. Rajagopalan, I. Litvan, T.-P. Jung, R. Rajagopalan, I. Litvan, and T.-P. Jung, “Fall prediction and prevention systems: recent trends, challenges, and future research directions,” *Sensors*, vol. 17, no. 11, p. 2509, 2017.
- [28] B. Hu, P. C. Dixon, J. V. Jacobs, J. T. Dennerlein, and J. M. Schiffman, “Machine learning algorithms based on signals from a single wearable inertial sensor can detect surface- and age-related differences in walking,” *Journal of Biomechanics*, vol. 71, pp. 37–42, 2018.

- [29] T. Nguyen Gia, V. K. Sarker, I. Tcareno et al., “Energy efficient wearable sensor node for IoT-based fall detection systems,” *Microprocessors and Microsystems*, vol. 56, pp. 34–46, Feb. 2018.
- [30] TensorFlow lite, 2019, <https://tensorflow.org/lite/>.
- [31] M. Musci, D. De Martini, N. Blago, T. Facchinetti, and M. Piastra, “Online fall detection using recurrent neural networks,” 2018, <https://arxiv.org/abs/1804.04976>.
- [32] ARM CMSIS Library, 2019, <https://developer.arm.com/embedded/cmsis>.
- [33] L. Leonardi, G. Patti, and L. Lo Bello, “Multi-hop real-time communications over Bluetooth low energy industrial wireless mesh networks,” *IEEE Access*, vol. 6, pp. 26505–26519, 2018.
- [34] K. Townsend, *Getting Started with Bluetooth Low Energy*, O’Reilly, Newton, MA, USA, 2014.

## Research Article

# A Practical Approach to Protect IoT Devices against Attacks and Compile Security Incident Datasets

**Bruno Cruz** <sup>1</sup>, **Silvana Gómez-Meire** <sup>1</sup>, **David Ruano-Ordás** <sup>1,2,3,4</sup>, **Helge Janicke**,<sup>3,4</sup>  
**Iryna Yevseyeva** <sup>3,4</sup> and **Jose R. Méndez** <sup>1,2</sup>

<sup>1</sup>Department of Computer Science, University of Vigo, ESEI—Escuela Superior de Ingeniería Informática, Edificio Politécnico, Campus Universitario As Lagoas s/n, 32004 Ourense, Spain

<sup>2</sup>SING Research Group, Galicia Sur Health Research Institute (IIS Galicia Sur), SERGAS-UVIGO, Vigo, Spain

<sup>3</sup>Cyber Technology Institute, School of Computer Science and Informatics, De Montfort University, Gateway House 5.33, The Gateway, LE1 9BH Leicester, UK

<sup>4</sup>Faculty of Computing, Engineering & Media (CEM), De Montfort University, Leicester, UK

Correspondence should be addressed to Jose R. Méndez; [moncho.mendez@uvigo.es](mailto:moncho.mendez@uvigo.es)

Received 27 April 2019; Revised 26 June 2019; Accepted 7 July 2019; Published 29 July 2019

Guest Editor: Daniele D'Agostino

Copyright © 2019 Bruno Cruz et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The Internet of Things (IoT) introduced the opportunity of remotely manipulating home appliances (such as heating systems, ovens, blinds, etc.) using computers and mobile devices. This idea fascinated people and originated a boom of IoT devices together with an increasing demand that was difficult to support. Many manufacturers quickly created hundreds of devices implementing functionalities but neglected some critical issues pertaining to device security. This oversight gave rise to the current situation where thousands of devices remain unpatched having many security issues that manufacturers cannot address after the devices have been produced and deployed. This article presents our novel research protecting IOT devices using Berkeley Packet Filters (BPFs) and evaluates our findings with the aid of our Filter.tlk tool, which is able to facilitate the development of BPF expressions that can be executed by GNU/Linux systems with a low impact on network packet throughput.

## 1. Introduction and Motivation

The evolution of Internet and communication networks from their emergence in the sixties to today has enabled a revolution in the way people and businesses interact. People today communicate worldwide using mobile devices, which have a reliable broadband (4G) Internet connection. Despite these great advances, Aceto et al. [1] note that network outages are still a challenge to solve because they are frequent, hard to fix, expensive, and, in particular, poorly understood by users. Whilst there exists a variety of problems surrounding network availability (Aceto et al. [1]), this study presents a proposal to avoid or at least minimize the effects of problems caused by software attacks through networks, including worms [2, 3] and remote attacks to exploit server vulnerabilities [4].

Patching avoids the compromise of target systems through, e.g., malware and vulnerability exploits. However, the growth of Internet of Things (IoT) applications running on devices, that frequently do not support patching, using Internet and TCP/IP networks for communication purposes, limits this possibility. Moreover, software upgrades are not always immediately available when the vulnerability is discovered, as patch development and distribution depend on developer circumstances. The existence of proactive defense mechanisms [5] capable of mitigating risks associated with unpatched components within an otherwise trusted TCP/IP network would be very valuable. In this study, we take advantage of the firewall support of operating systems to develop a highly efficient mechanism to detect and bring together information about malicious network traffic.

Firewalling support has become an essential feature of modern operating systems. The use of firewalls is one of the easiest mechanisms to manage network defense. However, its effectiveness is clearly limited to protect IoT devices against malware and vulnerability exploiting [6]. In the well-known Linux operating system, firewall capabilities have been provided primarily through packet filtering technology and have evolved from a netfilter ipfw system port (included in Linux kernel 1.1) to netfilter/iptables (included in Linux 2.4 kernel series). This evolution entailed the introduction of significant innovations such as the tracking of TCP connections or the possibility of altering packets in transit (mangle table). Despite the popularity of these filters, netfilter/iptables firewalling subsystem will be replaced in order to speed up the filtering process and increase the information achieved for each packet to filter (such as payload information).

Wireshark capture filters [7] are defined by using libpcap filter language. Filter examples that are designed to detect some worms and exploits are available in Wireshark Wiki [8] showing the power of this filter syntax. The syntax of capture filters is commonly known as Berkeley Packet Filter (BPF) and is supported in the kernel of most UNIX-like operating systems. This syntax is also implemented by libpcap/WiNpcap to be used at the user level in tools such as Wireshark. BPF [9] was first introduced in 1990 as a tool for capturing and filtering network packets that matched specific rules. BPF support was included in Linux kernel by implementing a small virtual machine that runs compiled BPF programs injected from user-space [10]. Later, a BPF Just-In-Time (JIT) compiler was added to speed up the performance of the execution of bytecodes. Currently, BPF can be loaded for its execution into kernel with different tools to execute different tasks, such as system monitoring (through using perf tool), network traffic control and quality of service (through tc tool), and packet filtering (through ip link tool included in iproute2 suite or iptables).

Due to its flexibility, BPF has been used by important technology companies such as Google, Facebook, Cloudflare, and Netflix to address network security issues, load-balancing, traffic-filtering, and monitoring [11–14]. A comparison of the filtering performance achieved by a BPF-based filter (BPFilter), iptables, and nftables has also been provided in other studies [11, 15] showing that BPFilter runs up to 5 times faster than iptables. This scenario led to the consideration of BPF as a reliable candidate to replace iptables (and nftables) as the kernel firewall subsystem for Linux [11]. However, despite the fact that BPF syntax is more powerful than that offered by current Linux firewalls, BPFilter only takes advantage of the BPF virtual machine to speed up rules created by older tools. Majkowski [16, 17] demonstrated how to take advantage of BPF in conjunction with iptables to filter packages and define new chains. These works allowed system administrators to take advantage of the rich syntax and efficient execution of BPF expressions to filter packages in real environments and protect IoT devices against malware and vulnerability exploiting.

We developed Filter.tk to work in conjunction with these tools. Filter.tk is a framework to complete the full lifecycle (creation, debugging, and testing) of BPF iptables-compliant pattern design for mitigating both worm and exploit attacks. The development of patterns will be useful for the future creation of a BPF rules database usable in the form of well-known community collaboration products such as Ansible Galaxy [18, 19] or DockerHub [20], where users can share BPF to protect IoT devices, computers, and software against worm and exploit attacks. Additionally, the information about harmful network packets can be uploaded to centralized repository for research purposes. Particularly, this data, if compiled worldwide, could allow the identification of security threats and help in the identification of new offensive packet patterns.

The remainder of this paper is structured as follows: Section 2 introduces the state of the art in well-known worms, security vulnerabilities, and IoT security. Section 3 introduces our proposal to address both the protection of devices against security vulnerabilities (and hence, worm attacks exploiting those vulnerabilities), and the compilation of security incident datasets in the context of IoT while Section 4 is centered on discovering the utility of the toolset through case studies. Finally, Section 5 presents the main conclusions of the work and outlines the directions for future research.

## 2. State of Art

During the early 2000s, Internet worms became very popular due to the effects of well-known worms such as Code Red (versions 1 and 2), Nimda, SQL Slammer, or Blaster. Some of these worms are compiled in the work of Qing and Wen [2]. However, due to the increased awareness of users and developers about the importance of security, this kind of malicious software is solely spread in P2P networks and operates in a passive form [21]. Instead of performing an active search to infect computers, passive worms require human intervention, i.e., by downloading an infected file from a P2P network, to replicate themselves. Despite the propagation of passive worms in P2P networks mainly connected with the illegal downloading of software and multimedia materials, the dissemination of these Internet worms and their mitigation has been fairly well discussed in previous studies [21–29]. The detection of new vulnerabilities allowing remote exploitation is a very active area as evidenced by the latest exploits published in exploit-db [4]. Although the existence of vulnerabilities allowing the execution of remote commands could provide a mechanism for the dissemination of worms, the quick response of software development teams to provide security patches discourages malware developers from designing new worms. With this in mind, the goal of this work is to mitigate attacks exploiting software vulnerabilities, with a special interest in those targeting an IoT device.

Many IoT applications and devices have become available for smart home automation. Querying “remote” “hardware” exploits in exploit-db and other similar databases resulted in a number of exploitable vulnerabilities in



well-known products (such as intelligent TVs, cameras, etc.). This shows that IoT developers have been prioritizing the development and creation of functionalities for most demanding users while frequently neglecting security considerations.

A few works have addressed issues in IoT security, such as the use of block-chain communications [30–33]. These usually refer to security issues pertaining to confidentiality, integrity, and availability in the communications between IoT devices and IoT. The work of Ammar et al. [34] provides a critical review of eight well-known IoT frameworks with special emphasis on security issues (analyzing models and approaches provided for ensuring security and privacy, pros and cons of each framework in terms of fulfilling the security requirements and meeting the standard guidelines, and identifying design flaws). Wood and Stankovic [35, 36] provide studies about network issues. Particularly, the former work is centered in security-related issues about IoT communication protocols whilst the later analyzes denial of service threats in IoT environments.

Wack et al. [37] review the risk of platform software/firmware vulnerabilities that enable the reception of malicious attacks. To the best of our knowledge, there is no research work focused on the prevention, management, and response to vulnerability exploiting and worm attacks in IoT. Closing this gap, we studied how to take advantage of firewalling schemes to implement these protections.

**2.1. OS Firewalling Support.** Common OS firewalls, such as those that can be implemented through GNU/Linux kernel firewalling subsystem, are usually implemented as packet filters [37, 38], which consist of a default policy for packets and a sequence of rules that define the actions performed on packets when they satisfy certain conditions. Specifically, each firewalling rule contains a triggering condition, usually a simple condition or the logical AND of simple conditions, together with an action to execute when the rule is triggered. Triggering conditions are defined over the second, third, and fourth TCP/IP layers. The support for stateful inspection of connections is available for kernel versions 2.4 and above [39].

The first GNU/Linux firewall generation was included on 1.1 kernel through an implementation of ipfw functionalities contributed by Alan Cox [40]. The ipfwadm user-space tool was used to configure the ipfw services offered by the kernel [41]. These kernels allowed defining three different firewall filters to handle (i) input packets (*-I* ipfwadm argument), (ii) output packets (*-O*), and (iii) forwarded packets (*-F*, used in conjunction with *ip\_forwarding* feature). Accept, deny (discard the packet), and reject actions were used either for rules (*-a* command parameter) or as default policy (*-p*). In order to create and design the trigger condition of each rule, the system administrator can test for the protocol (TCP, UDP, ICMP, or IP), the port (for TCP and UDP) or the ICMP type. Logging is supported through the *-o* modifier.

The support for ipfw was replaced by ipchains in the 2.2 version of the kernel [42]. One of the most important

changes in the ipchains scheme was the introduction of chains to help reduce the computational cost and facilitate its design [43]. As opposed to the others, ipchains firewalls include INPUT, OUTPUT, and FORWARD chains, which bring together filtering rules applied to packets where the current computer is the destination, the origin, or a router for the packet, respectively. Each firewall chain is composed of a ruleset and a default policy. The default policy is applied to packets that do not match any rule. The existence of default policies allows defining firewalls using two different schemes: (i) accept all except those packets explicitly denied or (ii) deny all except those packets explicitly accepted. Of these, the latter is advisable for security reasons.

New functionalities offered by ipchains with regard to ipfw were quite limited, so it was quickly replaced by iptables (in Linux 2.4 series) [44]. Iptables/netfilter included the table concept to bring together chains with similarities. Iptables included the firewall tables filter, nat, and mangle. The first one included three chains to support the filtering of input, output, and forwarded connections (INPUT, OUTPUT, and FORWARD respectively). The NAT table is composed of PREROUTING and POSTROUTING chains to add rules to change destination or source addresses, respectively. To this end, rules included in these chains could only use DNAT, SNAT, REDIRECT, or MASQUERADE actions. Finally, MANGLE table allows marking packages for further processing and modifying some parameters of packets including TOS or TTL. These actions could be executed by using MARK, TOS, and TTL actions. Finally, iptables brought the stateful packet inspection to Linux firewalls making it possible to determine whether a packet belongs to an established TCP connection (*-m state--state=ESTABLISHED*) or, conversely, is connected with other previous packets (*-m state--state=RELATED*).

Iptables have been widely used to implement packet filters on Linux for many years [45]. However, some limitations of iptables, such as the existence of a unique action for a rule or the complexity of the syntax, led to the creation of other filtering frameworks. Hence, nftables emerged as an iptables replacement on kernel version 3.13 (2013) [44]. Nftables included a completely new and fresh syntax that avoided the need to use hyphens and the uppercase/lowercase flags. The use of nftables allows tables and chains to be created with specific names and associated with hooks, thus avoiding the strict tables/chains structure defined by iptables.

Despite the new functionalities of nftables, most Linux users continue using the old iptables framework, in part due to the numerous changes in the syntax which hindered adoption. Some translation utilities were introduced to aid in the migration from iptables to nftables [46]. In addition, nftables work as a sequential filter whereby every packet is matched one by one against a list of rules. The speed of checking the rules is quite limited (up to three times slower than using BPF) [11], which led to the emergence of bpfILTER as a new Linux firewalling subsystem [47] able to outperform the speed of previous

filtering alternatives [11]. Bpfilter has been added experimentally to Linux kernel 3.18, now allowing nftables and iptables rules to be executed by Linux kernel as BPF.

Standard definitions within BPF only allow current packet filtering firewall [48] schemes to analyze some information from packet headers (such as IP and MAC addresses, ports, TCP flags, ICMP types, etc.) and packet state. Additional new features would improve the firewalling performance, such as analyzing the payload of the packet or the information about application layer protocols. These features are frequently included in deep packet inspection techniques [49, 50] but are often too slow to be included in standard firewalls. In order to provide a deep description of packets on the firewall layer and quickly evaluate them, the use of BPF language together with the BPF virtual machine subsystem included in the current versions of Linux kernel seems to be an elegant solution, especially as BPF had been used before to accomplish similar difficult network tasks with low computational effort [11].

Despite its performance and low computational costs, developing a BPF-based firewall able to exploit full packet data (headers and payload) is a hard task that would require both the existence of tools to aid in the development of conditions and packet datasets. Caploader [51] and Wireshark/tcpdump [7], which can also be integrated with NDPI [52], are capable of loading and analyzing packets included in PCAP files and check whether a BPF expression match them. These tools can be successfully executed with large collections of packets, such as that shared by Netresec [53]. However, the design of BPF filters is not easy and should be simplified to impact on real-world firewall applications. Similarly, the evaluation of BPF filters should be automated to improve performance. Both the simplification and automation have been addressed by our Filter.tlk toolset and are the main contribution of this work. Filter.lk's functionality and its practical use are described in the next section.

### 3. Filter.tlk

This section provides a comprehensive description of the design architecture of Filter.tlk [54] tool and documents the process of creating customized filters to classify network traffic according to the content of the packets.

Filter.tlk comprises three different utilities to aid in the creation of BPF filters: (i) an interface to design BPF filtering conditions, (ii) a Wireshark LUA plugin to automate the testing of BPF filters with PCAP packet datasets, and (iii) a script to easily compile BPF filters and create iptables rules. Figure 1 shows the different components included in Filter.tlk and their use in a real environment.

As we can deduce from Figure 1, the design of a firewall rule with Filter.tlk comprises three stages that are made with different tools included in the package. We begin by taking advantage of the BDAT (BPF design aid tool) to design a filtering condition to detect a certain kind of packet. The designed BPF filters can then be tested

with different packet sets (a set of packets matching the pcap filter and others mismatching the pcap filter) using BPF Testing tool (BTT). BTT is able to easily assess the quality of an input filter by using different datasets. Once BPF rules have been tested, they can be easily transformed into iptables rules using IPTables Rule Builder (IPTRB) script.

BDAT is responsible for creating BPF filters as conditions defined from transport and network layers (see Figure 1). By using BDAT through a simple graphical interface, we can create a Boolean BPF filter evaluating expressions related to network or transport headers and payloads (UDP, TCP, IP, ICMP). In order to create header conditions, users must select the field of the header on which they want to establish the condition that the filter must fulfill. Once the condition has been defined, the user can continue adding new conditions for the same filter or create a new one. Once all filters are defined, they can be exported to a file for testing in BTT (BPF Testing Tool). As an example, Figure 2 shows how administrators can easily incorporate a condition about a HTTP POST request by specifying conditions about TCP payload.

As depicted in Figure 2(a), we selected the first four octets from TCP payload for comparison purposes. BDAT allows selecting one, two, or four octets from each word (32 bits) to check the condition. The next step of the wizard (see Figure 2(b)) allows to easily define the value using Hexadecimal, ASCII, or Decimal notations. In order to compare any octet from payload (and options/padding), the offset value (highlighted in red in Figure 2(a)) can be edited to the desired value. Please note that the designed condition is provided as example and should be complemented with a "header length" value of five to ensure the absence of options/padding field. In the next step of the wizard, the current BPF condition rule is added to the whole BPF filter, allowing the generation of filters comprising multiple tests.

BTT is a plugin for Wireshark that applies a filter or set of filters over a pcap file. As a result, we obtain information for each applied filter about the number of packages analyzed, accepted, and rejected. A set of pcaps with the packets accepted by the filter grouped by the destination IP is also provided for debugging purposes. In order to detect errors, each rule should be tested using a pcap database containing only the packets that should be captured (ensuring a result of 0 rejected is achieved) and another one containing normal network traffic (guaranteeing a result of 0 accepted is achieved).

Finally, IPTRB (IPTables Rule Builder) script can transform the BPF uncompiled filters into full featured IPTables rules. The process is guided by an intuitive libncurses-based graphical user interface that allows customizing the generated rule. The rule can be generated for filtering and/or harmful packet logging purposes. A scheduled task (i.e., crontab) could be periodically executed (for instance once a day) to upload the compiled information (logs) to a centralized repository for its further analysis.

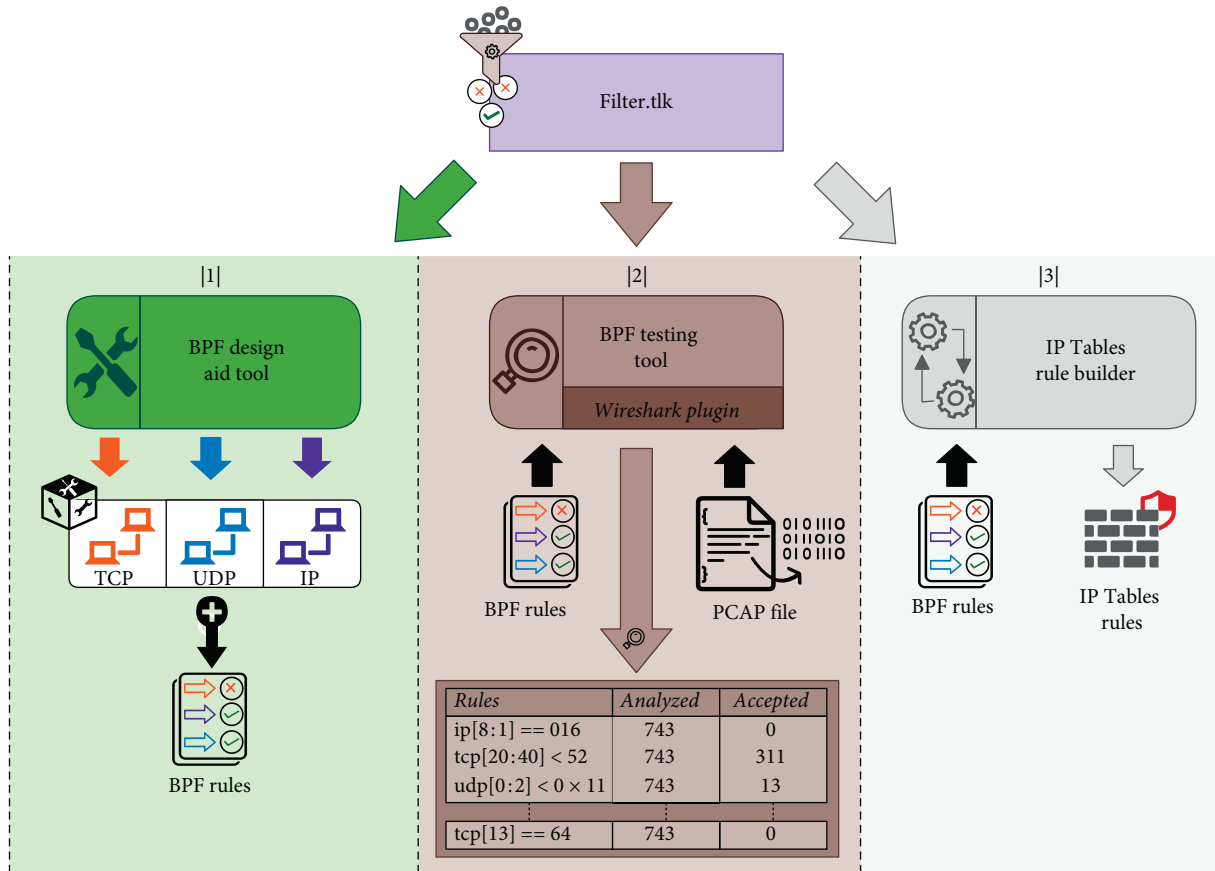


FIGURE 1: Filter.tlk architecture.

3.1. *Filter.tlk Implementation.* This subsection provides a brief description of the most relevant implementation issues for the development of each tool included in Filter.tlk.

BDAT was designed as a Java standalone application, which can easily be executed using any Java Virtual Machine implementation. The interface was designed using JFC/Swing library [55].

BTT is a Wireshark plugin that was implemented using the Lua programming language [56], which is supported by Wireshark for the development of new functionalities, such as the creation of dissectors or listeners [57]. The dissectors are intended to analyze part of the data of a packet, while the listeners are used to count the number of occurrences of an event; for example, the number of packets matching a filter. In this study, we used Lua language to implement a Wireshark listener to evaluate filters and count packets fitting the target BPF condition (accepted) or not (rejected).

IPTRB is a bash script that combines the use of dialog command [58] to provide an easy-to-use intuitive graphical user interface. Moreover, the compilation of BPF rules into bytecode is done by using tcpdump [59] functionalities.

Finally, we used Ansible [19] (a well-known IT Automation tool) to automate the installation of Filter.tlk in all supported platforms. Ansible is a popular IT automation tool whose main features are (i) avoiding the need of scripts and/or custom code to deploy and update applications and

(ii) replacing agents on remote systems by standard SSH tools. The installation script was provided for Debian-based GNU Linux distributions.

3.2. *Deployment of Generated Filters.* To take advantage of expressions (iptables rules and BPF) generated using our BPF framework, we consider two different scenarios: (i) IoT devices using a GNU Linux-based software/firmware (ii) and other IoT devices with no BPF/iptables support. In the first scenario, iptables rules can be directly integrated into the firmware to protect them against malicious attacks. We are working on the development of a service to share BPFs together with a tool able to automatically download and upgrade BPFs for different IoT devices.

Although GNU/Linux is present in some devices, there are many appliances running other operating systems where the execution of BPF is not possible. Taking this into account, we are working on the design of a small bridge router (brouter) [60] device running GNU/Linux and ebtables. A brouter is a device that is able to transparently forward all traffic between two ethernet interfaces and allows the inclusion of filtering rules for network interfaces. This solution would be applicable for IoT devices connected to the network through an ethernet connection.

The main weakness of using BPF filters to protect devices against attacks is that we are unable to protect 802.11-based (WLAN, wireless local area network) IoT devices that do not run GNU/Linux.

Offsets	Octet	0	1	2	3
Octet	Bit	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	
0	0	Source Port		Destination Port	
4	32	Sequence Number			
8	64	Acknowledgment Number			
12	96	Header Length	Reserved 0000	NS	RESERVED
16	128	Checksum		Urgent Pointer	
<b>Options &amp; Padding &amp; Payload</b>					
20	160	First Octet	Second Octet	Third Octet	Fourth Octet

(a)

Enter the conditions that the filter must meet.

Pattern rule: `tcp[20:4]`

Condition: ==

Value:  HEX  ASCII  DEC

POST

0x504F5354

(b)

FIGURE 2: BPF rule definition process. (a) Select TCP payload octets. (b) Establish values for octets.

Next section presents a comprehensive practical example describing the process of using the Filter.tlk tool to design a filter capable of detecting and filtering two important vulnerabilities recently discovered on well-known IoT devices.

#### 4. Experiments

In this section, we test the filter designed to detect and filter attacks using two vulnerabilities in two well-known IoT devices that allow the remote execution of arbitrary

commands: (i) LG Supersign TVs, and (ii) ASUS ADSL Router DSL-N12E\_C1. The next subsection shows the work environment prepared in order to generate high-quality BPF patterns. Moreover, subsection 4.2 presents the experimental protocol and results of our case studies while subsection 4.3 measures the impact of the use of these filters in the performance of IoT devices. Finally, subsection 4.4 shows how to compile and take advantage of the information gathered by IoT devices using Filter.tlk for scientific purposes.



*4.1. Configuring the Working Environment.* In order to generate high-quality BPF expressions that describe the pattern of vulnerability exploitation, the use of a large packet database for testing purposes is advisable. Fortunately, many publicly available packet datasets can be freely downloaded from the Internet. Table 1 compiles a list of useful datasets.

From the datasets included in Table 1 and other sources, we built up a group of packets that would be used to ensure that inoffensive network requests are not captured by the designed BPF expression.

We decided to study and generate BPF filters for two vulnerabilities of well-known IoT devices. In order to determine the quality of the BPF expressions created using a BTT Lua script, we used as negative samples the conjunction of all packets from sources introduced in Table 1 and other legitimate packet sets compiled by us. In order to aggregate all negative samples in a single pcap file, we combined all sources using the mergecap [64] tool provided by Wireshark.

*4.2. Executing the Experiment.* Recently, a vulnerability allowing remote execution of arbitrary commands appeared on LG SuperSign TVs (CVE-2018-17173) [65, 66]. These smart TVs include a CMS running on the top of LG webOS 3.3 (a Linux-based OS). The discovered vulnerability allows remote code execution (by achieving a reverse shell connection) by taking advantage of the URL used to see thumbnails of the user images. We used the exploit versions to generate a pcap file capturing the attacks. The Filter.tk comprises a three-stage operation. The first step designs the filtering condition to detect the packets, and the second step tests the BPF filter with a set of packets captured in a pcap filter and with a set of normal packets. The third step converts the BPF into iptables rules. The generated BPF expression is shown in Table 2. These BPF expressions could be directly included in LG webOS to protect the TV.

The second analysis is about a remote code execution vulnerability in ASUS DSL-N12E\_C1 router, specifically in firmware version 1.1.2.3\_345 (CVE-2018-15887) [67]. This vulnerability has been classified as critical because it allows the execution of arbitrary code using an unknown function of the file “Main\_Analysis\_Content.asp.” A remote attacker can then access the router as a privileged user via telnet application and run OS commands. Again, we take advantage of our framework to generate BPF expressions to filter this type of attack. The generated BPF is shown in Table 3.

As shown in Tables 2 and 3, an iptables command can be easily generated from a BPF expression to drop and log packets that match it. Although iptables can only check expressions in the header of network packets, BPF expressions make it possible to examine information included in both packet headers and payload in order to find any potential exploitation of vulnerabilities. The second generated iptables rule allows for storing security information that can be uploaded to a centralized repository for its further

TABLE 1: Publicly available datasets.

Dataset	Number of packets	Number of files	Format	Short description
Contagiodump [61]	988898	1154	pcap zipped	Collect malicious and exploit pcaps from various public resources (2013–2015)
Malware traffic analysis [62]	2445211	1291	pcap zipped	Malicious network traffic (2013–2018)
GTISK PANDA malrec [63]	100201	373	pcap	Malware samples run in PANDA (2018)

analysis. In next section, we evaluate the performance impact on the IoT devices when using these filters.

*4.3. Impact on Filter Throughput.* We assessed the impact of using BPF filters on IoT devices in order to determine if they could be successfully used to protect IoT devices against network vulnerability exploitation. To perform this analysis, we used an Apache web server installed on a Raspberry Pi 2 Model B [68].

We leveraged the functionalities of Apache HTTP server benchmarking [69] and GNU parallel [70] tools to evaluate the impact of using BPF firewalls in IoT hardware. Using these tools, we benchmarked the execution of two parallel tests making 10000 HTTP requests distributed in 10 threads, with 1000 requests per thread. The average of measurements made for parallelized tests is provided as result. For comparison purposes, we used the generated BPF expressions for the two case studies shown before. Table 4 compares the performance between the absence of attack protections and the usage of two BPF filtering rules.

The results compiled in Table 4 show that the performance impact when using BPF filters is quite limited and will not severely affect the overall operation of IoT devices. We analyzed the impact of progressively adding BPF rules to the filter by adding up to 100 new rules and measured the transfer rate after each BPF expression was added (see Figure 3). As long as the performance is highly influenced by the presence of additional traffic in network and other processes consuming CPU, we plotted a trend line to observe the degradation.

As can be seen from Figure 3, the throughput degradation is close to zero when using up to 50 (nonfitting) BPF rules. However, the inclusion of more than 50 rules clearly damages the performance of GNU/Linux firewalling system and would require the usage of additional iptables speedup strategies, such as the creation of additional chains [71] and counters-based optimizations [72].

One of the most interesting features of Filter.tk framework is the compilation of information about worldwide IoT security incidents. The information gathered could be successfully analyzed using machine learning

TABLE 2: BPF expression to mitigate CVE-2018-17173 vulnerability.

BPF expression	ip[2:2] > 0x008A and ip[9] == 0x06 and tcp[2:2] == 0x2378 and tcp[32] == 0x47 and tcp[77:4] == 0x3d253237 and tcp[81:4] == 0x2532302d and tcp[85] == 0x3b			
	BPF assembler code			Bytecode
(000) ldh	[12]			22
(001) jeq	#0x800	jt 2	jf 21	40 0 0 12
(002) ldh	[16]			21 0 19 2048
(003) jgt	#0x8a	jt 4	jf 21	40 0 0 16
(004) ldb	[23]			37 0 17 138
(005) jeq	#0x6	jt 6	jf 21	48 0 0 23
(006) jeq	#0x6	jt 7	jf 21	21 0 15 6
(007) ldh	[20]			21 0 14 6
(008) jset	#0x1fff	jt 21	jf 9	40 0 0 20
(009) ldx	4 * ([14]&0xf)			69 12 0 8191
(010) ldh	[x + 16]			177 0 0 14
(011) jeq	#0x2378	jt 12	jf 21	72 0 0 16
(012) ldb	[x + 46]			21 0 9 9080
(013) jeq	#0x47	jt 14	jf 21	80 0 0 46
(014) ld	[x + 91]			21 0 7 71
(015) jeq	#0x3d253237	jt 16	jf 21	64 0 0 91
(016) ld	[x + 95]			21 0 5 1025847863
(017) jeq	#0x2532302d	jt 18	jf 21	64 0 0 95
(018) ldb	[x + 99]			21 0 3 624046125
(019) jeq	#0x3b	jt 20	jf 21	80 0 0 99
(020) ret	#262144			21 0 1 59
(021) ret	#0			6 0 0 262144 6 0 0 0

*Iptables commands*

```
iptables -t filter -A INPUT -m bpf --bytecode "22,40 0 0 12,21 0 19 2048,40 0 0 16,37 0 17 138,48 0 0 23,21 0 15 6,21 0 14 6,40 0 0 20,69 12 0 8191,177 0 0 14,72 0 0 16,21 0 9 9080,80 0 0 46,21 0 7 71,64 0 0 91,21 0 5 1025847863,64 0 0 95,21 0 3 624046125,80 0 0 99,21 0 1 59,6 0 0 262144,6 0 0 0" -j DROP
iptables -t filter -A INPUT -m bpf --bytecode "22,40 0 0 12,21 0 19 2048,40 0 0 16,37 0 17 138,48 0 0 23,21 0 15 6,21 0 14 6,40 0 0 20,69 12 0 8191,177 0 0 14,72 0 0 16,21 0 9 9080,80 0 0 46,21 0 7 71,64 0 0 91,21 0 5 1025847863,64 0 0 95,21 0 3 624046125,80 0 0 99,21 0 1 59,6 0 0 262144,6 0 0 0" -j LOG --log-prefix "Filter.tlk"
```

techniques to provide worthwhile knowledge about (i) the origin of the threat, (ii) better patterns for traffic-filtering, or (iii) the threat scale. Studying information about systems from which the attack is performed, we can successfully identify worms exploiting a certain vulnerability, the presence of an individual hacker, and the execution of Distributed Denial of Service attacks or botnets.

Offering IoT users a product to protect their devices against attacks, whilst at the same time achieving information about dangerous offensive network packets targeting IoT products, will replicate a threat response model undertaken by traditional antivirus products. This knowledge allows the identification of better and perhaps simpler BPF patterns that can be used for network intrusion detection.

## 5. Conclusions and Future Work

In this paper, we have introduced an easy-to-use framework designed to aid in the development of fast firewalls based on using BPF, which can be executed by using standard firewall capabilities included in the Linux kernel (IPTables/netfilter). These firewalls have been specifically conceived to protect IoT devices against the exploitation of remote vulnerabilities. Since

the use of BPF bytecode can drastically speed up the execution of firewalls, we designed a collection of tools to facilitate the inclusion of BPF into firewalling rules. An experiment was carried out for the application of the introduced toolset.

Since BPF is one of the most efficient forms of filtering traffic, it provides a reliable solution for filtering in the context of IoT. Although, the use of specific BPF filters allows using payload information included in packets, it can only be directly implemented in devices using a GNU/Linux-based firmware. We are currently working on the design of specific hardware to overcome the limitations of nonGNU/Linux ethernet IoT devices and on the development of a package manager to automatically download BPF filtering strings and configure the firewall.

One of the most relevant functionalities of this scheme is the ability to easily build a dataset with the security incidents occurred in worldwide IoT devices, such as VizSec [73]. Future work will include mechanisms for analyzing them to achieve valuable security knowledge. We consider evolutionary computation as a candidate method for automatic filter generation through packet captures and consider DPI (deep packet inspection [52]) to be a reliable way of simplifying filtering conditions, since it allows access to application layer information to define matching

TABLE 3: BPF expression to mitigate CVE-2018-15887 vulnerability.

BPF expression	ip[2:2] > 0x0174 and ip[9] == 0x06 and tcp[2:2] == 0x0050 and tcp[32] == 0x47 and tcp[326:4] == 0x3d253630		
	BPF assembler code		Bytecode
(000) ldh	[12]		18
(001) jeq	#0x800	jt 2	40 0 0 12
(002) ldh	[16]		21 0 15 2048
(003) jgt	#0x174	jt 4	40 0 0 16
(004) ldb	[23]		37 0 13 372
(005) jeq	#0x6	jt 6	48 0 0 23
(006) jeq	#0x6	jt 7	21 0 11 6
(007) ldh	[20]		21 0 10 6
(008) jset	#0x1fff	jt 17	40 0 0 20
(009) ldx	4 * ([14]&0xf)		69 8 0 8191
(010) ldh	[x + 16]		177 0 0 14
(011) jeq	#0x50	jt 12	72 0 0 16
(012) ldb	[x + 46]		21 0 5 80
(013) jeq	#0x47	jt 14	80 0 0 46
(014) ld	[x + 340]		21 0 3 71
(015) jeq	#0x3d253630	jt 16	64 0 0 340
(016) ret	#262144		21 0 1 1025848880
(017) ret	#0		6 0 0 262144
			6 0 0 0

*Iptables commands*

```
iptables -t filter -A INPUT -m bpf --bytecode "18,40 0 0 12,21 0 15 2048,40 0 0 16,37 0 13 372,48 0 0 23,21 0 11 6,21 0 10 6,40 0 0 20,69 8 0 8191,177 0 0 14,72 0 0 16,21 0 5 80,80 0 0 46,21 0 3 71,64 0 0 340,21 0 1 1025848880,6 0 0 262144,6 0 0 0" -j DROP
iptables -t filter -A INPUT -m bpf --bytecode "18,40 0 0 12,21 0 15 2048,40 0 0 16,37 0 13 372,48 0 0 23,21 0 11 6,21 0 10 6,40 0 0 20,69 8 0 8191,177 0 0 14,72 0 0 16,21 0 5 80,80 0 0 46,21 0 3 71,64 0 0 340,21 0 1 1025848880,6 0 0 262144,6 0 0 0" -j LOG --log-prefix "Filter.tlk"
```

TABLE 4: Performance impact when using BPF filters.

	No protection	With BPF (2 rules)
HTML transferred (bytes)	107010000	107010000
Concurrent time per request (ms)	1.9345	1.937
Time per request (ms)	19.3465	19.367
Time taken for tests (seconds)	19.3465	19.367
Total transferred (bytes)	109750000	109750000
Transfer rate (Kbytes/sec)	5539.905	5534.1

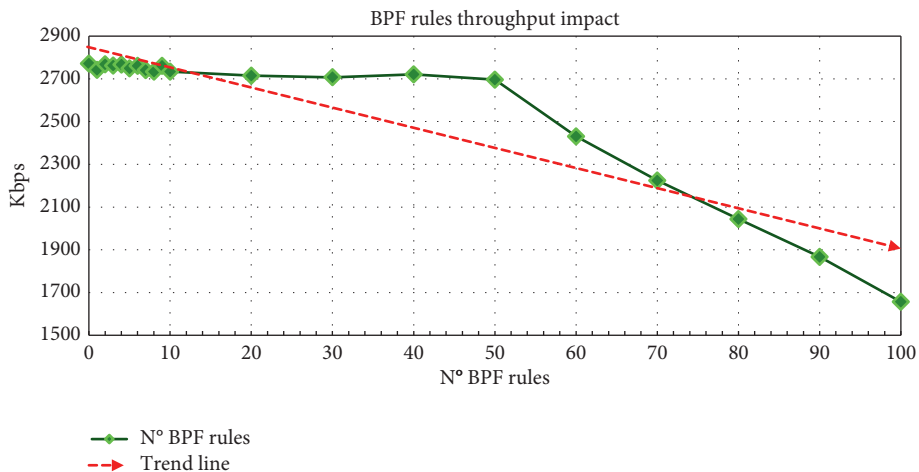


FIGURE 3: Analysis of the impact of BPF rules in throughput.

expressions. While DPI expressions cannot be directly included in BPF filters, we believe that they could be automatically transformed into simple BPF expressions to simplify the generation of BPF filters.

## Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Acknowledgments

D. Ruano-Ordás was supported by a postdoctoral fellowship from Xunta de Galicia (ED481B 2017/018). Additionally, this work was funded by the project Semantic Knowledge Integration for Content-Based Spam Filtering (TIN2017-84658-C2-1-R) from the Spanish Ministry of Economy, Industry and Competitiveness (SMEIC), State Research Agency (SRA), and the European Regional Development Fund (ERDF) and by the Consellería de Educación, Universidades e Formación Profesional (Xunta de Galicia) under the scope of the strategic funding of ED431C2018/55-GRC Competitive Reference Group. SING group thanks CITI (Centro de Investigación, Transferencia e Innovación) from University of Vigo for hosting its IT infrastructure.

## References

- [1] G. Aceto, A. Botta, P. Marchetta, V. Persico, and A. Pescapé, "A comprehensive survey on internet outages," *Journal of Network and Computer Applications*, vol. 113, pp. 36–63, 2018.
- [2] S. Qing and W. Wen, "A survey and trends on Internet worms," *Computers & Security*, vol. 24, no. 4, pp. 334–346, 2005.
- [3] D. E. Hiebeler, A. Audibert, E. Strubell, and I. J. Michaud, "An epidemiological model of internet worms with hierarchical dispersal and spatial clustering of hosts," *Journal of Theoretical Biology*, vol. 418, pp. 8–15, 2017.
- [4] Offensive Security, Exploit Database, 2009.
- [5] M. Ge, J. B. Hong, S. E. Yusuf, and D. S. Kim, "Proactive defense mechanisms for the software-defined Internet of things with non-patchable vulnerabilities," *Future Generation Computer Systems*, vol. 78, pp. 568–582, 2018.
- [6] R. K. Deka, K. P. Kalita, D. K. Bhattacharya, and J. K. Kalita, "Network defense: approaches, methods and techniques," *Journal of Network and Computer Applications*, vol. 57, pp. 71–84, 2015.
- [7] G. Combs, Wireshark Go Deep, 1998.
- [8] Wireshark, Wireshark CaptureFilters, 2016.
- [9] S. McCanne and V. Jacobson, "The BSD packet filter: a new architecture for user-level packet capture," in *Proceedings of the USENIX Winter 1993 Conference*, p. 2, San Diego, CA, USA, January 1993.
- [10] P. Anand, "An intro to using eBPF to filter packets in the Linux kernel," 2017, <http://www.OpenSource.com>.
- [11] T. Graf, "Why is the kernel community replacing iptables with BPF?," November 2018, <https://cilium.io/blog/2018/04/17/why-is-the-kernel-community-replacing-iptables/>.
- [12] Ł. Makowski and P. Grosso, "Evaluation of virtualization and traffic filtering methods for container networks," *Future Generation Computer Systems*, vol. 93, pp. 345–357, 2019.
- [13] G. Bertin, "XDP in practice: integrating XDP into our DDoS mitigation," in *Proceedings of the Technical Conference on Linux Networking*, p. 5, Seoul, South Korea, November 2017.
- [14] M. Yuhara, B. N. Bershad, C. Maeda, and J. E. B. Moss, "Efficient packet demultiplexing for multiple endpoints and large messages," in *Proceedings of the 1994 Winter USENIX Conference*, pp. 153–165, San Francisco, CA, USA, January 1994.
- [15] D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak, and G. Carle, "Performance implications of packet filtering with Linux eBPF," in *Proceedings of the 2018 30th International Teletraffic Congress (ITC 30)*, pp. 209–217, Vienna, Austria, September 2018.
- [16] M. Majkowski, BPF—the Forgotten Bytecode, 2014.
- [17] M. Majkowski, Introducing BPF Tools, 2014.
- [18] Read Hat, *Ansible Galaxy*, Read Hat, Inc., Raleigh, NC, USA, 2018.
- [19] M. DeHaan, *Ansible is Simple IT Automation*, Read Hat, Inc., Raleigh, NC, USA, 2012.
- [20] Docker Inc., *Docker Hub*, Docker, Inc., San Francisco, CA, USA, 2016.
- [21] M. A. Rguibi and N. Moussa, "Hybrid trust model for worm mitigation in P2P networks," *Journal of Information Security and Applications*, vol. 43, pp. 21–36, 2018.
- [22] F. Wang, Y. Zhang, and J. Ma, "Defending passive worms in unstructured P2P networks based on healthy file dissemination," *Computers & Security*, vol. 28, no. 7, pp. 628–636, 2009.
- [23] T. Chen, X.-S. Zhang, H. Li, X.-D. Li, and Y. Wu, "Fast quarantining of proactive worms in unstructured P2P networks," *Journal of Network and Computer Applications*, vol. 34, no. 5, pp. 1648–1659, 2011.
- [24] C.-S. Feng, J. Yang, Z.-G. Qin, D. Yuan, and H.-R. Cheng, "Modeling and analysis of passive worm propagation in the P2P file-sharing network," *Simulation Modelling Practice and Theory*, vol. 51, pp. 87–99, 2015.
- [25] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The Eigentrust algorithm for reputation management in P2P networks," in *Proceedings of the Twelfth International Conference on World Wide Web*, p. 640, Budapest, Hungary, May 2003.
- [26] L. Xiong and L. Liu, "PeerTrust: supporting reputation-based trust for peer-to-peer electronic communities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 7, pp. 843–857, 2004.
- [27] R. Zhou and K. Hwang, "PowerTrust: a robust and scalable reputation system for trusted peer-to-peer computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 4, pp. 460–473, 2007.
- [28] I. Stirling, W. Calvert, and C. Spencer, "Evidence of stereotyped underwater vocalizations of male Atlantic walrus (*Odobenus rosmarus rosmarus*)," *Canadian Journal of Zoology*, vol. 65, no. 9, pp. 2311–2321, 1987.
- [29] L. Cai and R. Rojas-Cessa, "Mitigation of malware proliferation in P2P networks using double-layer dynamic trust (DDT) management scheme," in *Proceedings of the IEEE Sarnoff Symposium*, pp. 1–5, Princeton, NJ, USA, April 2009.
- [30] D. Minoli and B. Occhiogrosso, "Blockchain mechanisms for IoT security," *Internet of Things*, vol. 1–2, pp. 1–13, 2018.



- [31] M. A. Khan and K. Salah, "IoT security: review, blockchain solutions, and open challenges," *Future Generation Computer Systems*, vol. 82, pp. 395–411, 2018.
- [32] Y. Qian, Y. Jiang, J. Chen et al., "Towards decentralized IoT security enhancement: a blockchain approach," *Computers & Electrical Engineering*, vol. 72, pp. 266–273, 2018.
- [33] I. Makhdoom, M. Abolhasan, H. Abbas, and W. Ni, "Blockchain's adoption in IoT: the challenges, and a way forward," *Journal of Network and Computer Applications*, vol. 125, pp. 251–279, 2019.
- [34] M. Ammar, G. Russello, and B. Crispo, "Internet of things: a survey on the security of IoT frameworks," *Journal of Information Security and Applications*, vol. 38, pp. 8–27, 2018.
- [35] A. K. Das, S. Zeadally, and D. He, "Taxonomy and analysis of security protocols for Internet of things," *Future Generation Computer Systems*, vol. 89, pp. 110–125, 2018.
- [36] A. D. Wood and J. A. Stankovic, "Denial of service in sensor networks," *Computer*, vol. 35, no. 10, pp. 54–62, 2002.
- [37] J. P. Wack, K. Cutler, and J. Pole, *Guidelines on Firewalls and Firewall Policy*, Booz Allen Hamilton Inc, McLean, VA, USA, 2002.
- [38] K. A. Scarfone and P. Hoffman, *Guidelines on Firewalls and Firewall Policy*, National Institute of Standards and Technology, Gaithersburg, MD, USA, 2009.
- [39] R. P. Hinglaspure and B. R. Burghate, "Analysis of packet filtering technology in computer network security," *International Journal of Computer Science and Mobile Computing*, vol. 3, no. 4, pp. 1302–1927, 2014.
- [40] O. Kirch and T. Dawson, *Linux Network Administrator's Guide*, O'Reilly Media, Newton, MA, USA, 2nd edition, 2000.
- [41] J. Vos and W. Konijnenberg, "IPFWADM: Linux firewall facilities for kernel-level packet filtering," in *Proceedings of the NLUUG Spring Conference*, Amsterdam, Netherlands, May 1996.
- [42] R. Russell, *Linux IPCHAINS-HOWTO*, 2000.
- [43] J. Stanger and P. T. Lane, "Chapter 9—implementing a firewall with ipchains and iptables," in *Hack Proofing Linux*, pp. 445–506, Syngress, Burlington, MA, USA, 2001.
- [44] P. Russell, *Netfilter: Firewalling, NAT, and Packet Managing for Linux*, 2000.
- [45] B. Sharma and K. Bajaj, "Packet filtering using IP tables in Linux," *International Journal of Computer Science Issues*, vol. 8, pp. 320–325, 2011.
- [46] A. Alemayhu, "Moving from iptables to nftables," 2018, <https://nftables.org>.
- [47] J. Corbet, "BPF comes to firewalls," 2018, <https://lwn.net/>.
- [48] K. Ingham and S. Forrest, "A history and survey of network firewalls," Technical Report 2002-37, University of New Mexico, Albuquerque, NM, USA, 2002.
- [49] R. Antonello, S. Fernandes, C. Kamienski et al., "Deep packet inspection tools and techniques in commodity platforms: challenges and trends," *Journal of Network and Computer Applications*, vol. 35, no. 6, pp. 1863–1878, 2012.
- [50] T. Bujlow, V. Carela-Español, and P. Barlet-Ros, "Independent comparison of popular DPI tools for traffic classification," *Computer Networks*, vol. 76, pp. 75–89, 2015.
- [51] Netresec A. B., *CapLoader*, 2018.
- [52] Ntop Team, *Say Hello to nDPI 2.0*, 2017.
- [53] Netresec, "Publicly available PCAP files," 2018, <http://www.netresec.com>.
- [54] B. Cruz, D. Ruano-Ordás, and J. R. Mendez, *FilterTLK*, 2019.
- [55] Oracle, *Trail: Creating a GUI with JFC/Swing*, Oracle, Redwood City, CA, USA, 2017.
- [56] R. Ierusalimsky, W. Celes, and L. H. de Figueiredo, *The Programming Language Lua*, Lua.Org, Brazil, ISBN: 8590379868, 4th edition, 2016.
- [57] H. Kaplan, *Lua Dissectors*, 2015.
- [58] W. Shotts, *LinuxCommand: Dialog*, No Starch Press, San Francisco, CA, USA, 2000.
- [59] The Tcpdump Group, *Tcpdump and Libpcap*, 2010.
- [60] Netfilter Project, *Ebtables*, <http://ebtables.netfilter.org>.
- [61] M. Parkour, *Contagio—Malware Dump*, 2014.
- [62] B. Duncan, "A Source for Pcap files and Malware Samples," *Malware-Traffic-Analysis*, 2018.
- [63] B. Dolan-Gavitt, *GTISK PANDA Malrec—PCAP Files from Malware Samples Run in PANDA*, 2018.
- [64] S. Renfo and B. Guyton, *MergeCap: merges two or more capture files into one* 2018, <https://www.wireshark.org/docs/man-pages/mergcap.html>.
- [65] A. Fanjul, *LG SuperSign RCE*, 2018.
- [66] Common Vulnerabilities and Exposures, CVE-2018-17173, 2018, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-17173>.
- [67] Common Vulnerabilities and Exposures, CVE-2018-15887, 2018, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-15887>.
- [68] Raspberry P. I., *Raspberry Pi 2 Model B*, 2018.
- [69] Apache Software Foundation, *Apache HTTP Server Benchmarking Tool: Apache HTTP Server Version 2.4*, Apache Software Foundation, Forest Hill, MD, USA, 2018.
- [70] O. Tange, *GNU Parallel* 2018, 2018.
- [71] L. Zhao, A. Shimae, and H. Nagamochi, "Linear-tree rule structure for firewall optimization," in *Proceedings of the Sixth {IASTED} International Conference on Communications, Internet, and Information Technology*, pp. 67–72, Banff, Canada, July 2007.
- [72] L. Defert, *Iptables-optimize*, 2014.
- [73] VizSec Group, "VizSec ciber security datasets," in *Proceedings of the IEEE Symposium on Visualization for Cyber Security*, Berlin, Germany, June 2019, <https://vizsec.org/>.