Distributed computational model for shared processing on Cyber-Physical System environments

Higinio Mora , José Francisco Colom , David Gil , Antonio Jimeno Morenilla

PII: S0140-3664(17)30042-7
DOI: 10.1016/j.comcom.2017.07.009
Reference: COMCOM 5535

To appear in: *Computer Communications*

Received date: 11 January 2017
Revised date: 26 May 2017
Accepted date: 19 July 2017

Please cite this article as: Higinio Mora , José Francisco Colom , David Gil , Antonio Jimeno Morenilla , Distributed computational model for shared processing on Cyber-Physical System environments, *Computer Communications* (2017), doi: 10.1016/j.comcom.2017.07.009

Highlights

- A computation model to distribute the application workload over the CPS is proposed.
- The scheduling method dynamically shares the tasks among the computing nodes of the CPS.
- This approach facilitates the networking and integration of heterogeneous computer devices.

Distributed computational model for shared processing on Cyber-Physical System environments

Higinio Mora[a,*], José Francisco Colom[a], David Gil[b], Antonio Jimeno Morenilla[c]

[a]Specialized Processor Architecture Laboratory, Department of Computer Science Technology and Computation, University of Alicante, 03690 Alicante, Spain
[b]Lucentia Research Group, Department of Computer Science Technology and Computation, University of Alicante, 03690 Alicante, Spain
[c]UniCAD Research Group, Department of Computer Science Technology and Computation, University of Alicante, 03690 Alicante, Spain
hmora@ua.es; fjcolom@dtic.ua.es; dgil@dtic.ua.es; jimeno@dtic.ua.es

*** Corresponding Author:** Dr. Higinio Mora, hmora@ua.es
Specialized Processor Architecture Laboratory, Department of Computer Science Technology and Computation, University of Alicante, 03690 Alicante, Spain
Tel. +34 96 590 3400 - Fax +34 96 590 3464

A B S T R A C T

Cyber-Physical Systems typically consist of a combination of mobile devices, embedded systems and computers to monitor, sense, and actuate with the surrounding real world. These computing elements are usually wireless, interconnected to share data and interact with each other, with the server part and also with cloud computing services. In such a heterogeneous environment, new applications arise to meet ever-increasing needs and these are an important challenge to the processing capabilities of devices. For example, automatic driving systems, manufacturing environments, smart city management, etc. To meet the requirements of said application contexts, the system can create computing processes to distribute the workload over the network and/or a cloud computing server. Multiple options arise in relation to what network nodes should support the execution of the processes. This paper focuses on this problem by introducing a distributed computational model to dynamically share these tasks among the computing nodes and considering the inherent variability of the context in these environments. Our novel approach promotes the integration of the computing resources, with externally supplied cloud services, to fulfill modern application requirements. A prototype implementation for the proposed model has been built and an application example has been designed to validate the proposal in a real working environment.

Keywords.- Cyber-Physical Systems, Internet of Things, Mobile computing, Modeling, Distributed computation

1. Introduction

The expansion of embedded systems into new application areas such as healthcare, automotive, robotics, home automation or smart cities has led to the development of the Internet of Things (IoT). This new paradigm consists in connecting any device with actuating, sensing, and computation capabilities [1, 2]. The growing presence of wireless communication technologies, such as wireless local area network (Wi-Fi), Long-Term Evolution communications (LTE) and Radio-frequency Identification (RFID), allows for the connection of devices to internet and remote monitoring and management through cloud applications. Ubiquitous possibilities enabled by IoT offers the ability to measure, infer and understand environmental indicators in said applications areas.

One step further is the introduction of more intelligent and interactive operations under the architecture of the IoT paradigm, resulting in Cyber-Physical Systems (CPS) [3]. These elements are networked to monitor, sense, and actuate physical elements in the real world and to work together as a system. The proliferation of connected devices in a communicating-actuating network creates smart environments where sensors and actuators blend seamlessly with the environment around us, and the information is shared across platforms [4].

Computing nodes in a CPS environment can be very heterogeneous in what concerns computing power and other capabilities. They are generally network-enabled small-sized computers ranging from nodes, with advanced sensing/actuating capabilities but very limited processing and storage resources, to powerful multi-core technologies with high storage capacity. In addition, other computers can cooperate in the network: servers, desktop and laptop computers (full-sized, not embedded computers), smartphones, and others.

In such a heterogeneous environment, as the one exposed by CPS, there are multiple options to consider when deciding which network nodes should support the execution of the processing tasks: sensor and actuators nodes can support additional processing with current technologies; local servers, desktop and mobile computers could also provide part of their resources for distributed system goals; and, furthermore, cloud computing services can be hired for additional computing power.

To be able to select the adequate CPS nodes, different factors need to be taken into account. In terms of network efficiency, it is preferable to bring the computing work near to data sources and users. Performance is another key issue: simple processing at a high data transfer flow rate is generally better performed by local devices (sensor or actuator), whenever possible; on the other hand, intensive processing at a low data transfer flow rate can be effectively done by powerful remote computers. Physical constraints are also an important aspect that needs to be addressed: some system components are able to integrate data from different sources, so sending the data flow through the network becomes mandatory. Other relevant factors include energy consumption in battery-powered devices [5], QoS (Quality of Service) requirements [6], available bandwidth [7], and monetary cost of the cloud services, among others.

The selection of the network node to be used for each distributed task could be done at the design stage. However, in a CPS, the high variability of the context generally recommends making decisions during execution time. Firstly, the flow rates of data transfers can be highly variable in time, depending on the input nature: for example, night video can be compressed more than daylight video using some compression techniques. Secondly, computing nodes can dynamically join or leave the network: incorporation of new sensors or actuators can dramatically increase data processing, but servers with free resources can alleviate the possible overload produced by this increase. The free resources provided by those servers for distributed systems can also be variable, depending on other tasks that must be carried out by the server. Last but not least, the available network bandwidth can vary depending on active network applications, usable technologies, or geographical location.

In general, there is an interesting open issue for efficient resource planning in the design of infrastructures

for CPS applications where there are heterogeneity of the environment and available computational resources. In this kind of execution contexts, the first step is to find the feasible options for scheduling based on the dynamic nature of the system. In addition, the computers may be doing their own work, apart from the tasks of the distributed application. Among the feasible options, some scheduling decisions may be more suitable than others depending on several aspects related with the overall application performance and/or user preferences. In addition, the cloud computing paradigm adds further elements to properly perform this scheduling process.

Advances in this area will allow for the proper sharing of computational cost of application processing among the devices networked, and the design of systems able to deploy advanced applications for providing additional added-value services to users.

This research tackles this problem. The **objective** of this work is to properly distribute the processing workload among the available computing nodes in a CPS taking into account the dynamic nature of the context observed in these environments and the desired preferences at the design stage. The **main contribution** of this work is the proposal of a formal model that will be able to quantify the available resources of each node and the computing power required by the tasks of the CPS application. On this foundation, it will be possible to model CPS environments, working on heterogeneous devices, and to design a method to find the suitable distribution of tasks. In addition, the overall design of CPS applications can be undertaken by following a framework which combines this formal model with a convenient application characterization.

This approach favors the integration of heterogeneous computer devices and even of external cloud services to meet the application requirements. Moreover, this integration is achieved in a flexible way: utility computing is used only when required depending on the CPS context, and sensor computing resources are preserved for their intended demanding tasks. In other words, different activities are distributed throughout the network, taking the best of both worlds: local execution with own computing resources and remotely supplied cloud services. The proposed solution brings a novel contribution by providing a coarse-grained data flow framework to successfully design a distributed application in dynamic CPS environments.

The rest of the paper is organized as follows. First, a review of related research areas and their relevant solutions is presented. Next, the model is described by formally defining the problem and providing a conceptual view of the solution. Section 4 describes a deployment method of the model by introducing a framework. Section 5 presents a study case: an example of a distributed application where the proposed approach is suitable. The experimental design and tests are made, taking the study case as reference, and the feasibility of the solution is verified. Finally, relevant conclusions and future directions of this research are outlined.

2. Related work

The following subsections discuss the state-of-the-art of the aspects related to this research. A final subsection is added, which outlines the contributions to previous work.

2.1 Distributed systems

Distributed computing is a field of computer science studying distributed systems. The components of a distributed system are located on networked computers, communicating and coordinating their actions by means of exchanging messages to meet a common goal. The development of these systems has madured through internet and cloud services. One of the most important research areas has to do with the so-called ubiquitous computing [8, 9], leading to various related concepts that emphasize different aspects of this type of computing: pervasive networking [10], pervasive computing [11], edge computing [12], and collective computing [13], among others.

One of the pillars of ubiquitous computing starts with the development of the mobile computing, which has to do with the use of portable devices equipped with one or more wireless interfaces and the exchange of data among them (mobile networking). This paradigm manages large amounts of data [14] and introduces more information dependent on the position (location-sensitive) in computing, resulting in a set of systems context-aware [15]. The evolution and progressive specialization of the devices (for example, towards the wearable computing), incorporating sensing and actuating capabilities, has led to the wireless sensor networks [16]. When the emphasis is done on the objects that embed these devices, more recent concepts arise, such as Internet of Things (IoT) (emphasis on global connectivity [1]) and cyber-physical systems (CPS) (attention to integration with physical processes [17]).

Applications of CPS have a great potential to improve citizens' quality of life. CPS will be operating in an environment with distributed elements, with the result that these applications are distributed and parallel in nature due to the wide variety of physical and cyber interactions involved. Extensive research is being done in distributed architectures for specific domains. For example, recent proposals can be found for robotics [18], industrial [19], smart driving [20], e-health [21] or smart cities [22-24] among many others.

The main challenges of these systems are those related to modelling and design tasks, interconnection and interoperability of heterogeneous devices, security issues, QoS, scheduling and control of the system [17, 25, 26]. There are a lot of research focused on all these topics. On this last matter, the problem is defined as resource provisioning in highly distributed systems.

2.2. *Distributed processing*

The scheduling problem attracts many research interest in the development of current CPS environments

since the proper distribution of the processing plays an essential role in the leveraging of the resources of CPS environments. An effective scheduling process enhances the whole performance of the system in handling advanced applications which overcome the capabilities of the individual nodes.

Scheduling and control problems in highly heterogeneous CPS may be considered as a family of NP-complete optimization problems in general cases as well as in constrained cases. In these systems, an arbitrary number of threads can be scheduled depending on the physical attributes of the system. In addition, there are usually frequent restrictions imposed by the application needs, such as QoS, latency or power consumption [25, 27].

Since CPS are a kind networked systems, the starting point to address this problem could be found in the set of methods for scheduling tasks on distributed computing systems. In strongly-coupled systems, the processing elements are in high speed networks and they are inter-connected through a network topology. Usually, this network has a very high bandwidth and the time cost due to communication delay is negligible. There are recent scheduling proposals for this kind systems both for similar [28] and for heterogeneous [29] computing components. A particular case is the multicore processing system where the cores are built in the same integrated circuit and they have similar [30] or different features [31]. In order to include the unpredictability and extra delay provided by the network, other approaches for distributing the computation in heterogeneous cluster systems can be applied. The objective for task scheduling depends on system requirements such as energy cost, schedule length, throughput, economic cost, etc. [32-34]. Advanced job scheduling systems such as '*Slurm Workload Manager*' (http://slurm.schedmd.com/) could also be considered to perform the scheduling work.

However, the CPSs are highly distributed architectures and they can be considered as very weakly-coupled computer systems where the heterogeneity of connected devices as well as their dynamic behavior make it very difficult to apply the previous methods. Further research is needed to adapt the job-based scheduling methods to embedded devices and types of applications involved in the CPS environments to fully utilize the nodes and to achieve performance improvements. As a result, there are a recent number of solutions proposed for task scheduling in CPS environments. Table 1 summarizes the recent works on this topic and remarks the main contributions and ideas for implementing them.

Table 1. Distributing processing recent contributions.

Work	Main contributions
Mixed scheduling [27]	Combines distributed scheduling algorithms with the game theory ideas.
TAOPN [35]	Time-constrained aspect-oriented Petri net to model and scheduling tasks.
Payload-Size and Deadline-Aware [36]	Traffic-sensitive real-time scheduling algorithm.
Multi-layered scheduling [37]	Multi-layered scheduling scheme for time and control critical tasks.
Adaptive Dynamic Scheduling for automotive CPS [38]	Adapt. dynamic scheduling to adjust the exec. on different criticality levels.
High performance real-time scheduling [39]	Changing the system's criticality to achieve fair scheduling of functions whose criticality levels are larger than or equal to the system's criticality
Distributed data traffic scheduling [40]	Scheduling strategy considering transmission time delays.
Modas [41]	Distribute agent algorithm to achieve multi-QoS requirements.
Effective scheduling for CPS society [42]	Scheduling strategy considering physical systems society factors.
Data Traffic Scheduling [43]	Traffic scheduling by system dynamics modelling
Crowdsourcing in Cyber-Physical Systems [44]	Cross-layer optimization framework to solve a finite-queue-aware CPS service maximization problem by stochastic methods.
New task modeling [45]	New periodic, fault-tolerant CPS task model.
Flat Semi-Dormant Multi-Controllers [46]	Considers an arbitrated networked control systems and a wakeup mechanism on the communication system
Comprehensive resource scheduling strategy [47]	Dynamic multi-priority scheduling at node network and at comp. center.

Most of the previous works are focused on real-time [27, 35-39, 42, 45] and other performance criteria [40, 41, 43, 44, 46, 47]. As they corroborate, the standard scheduling algorithms for distributed systems cannot satisfy the requirements in CPS environments. There are network physical factors involved that cause migration delay time between servicing node to serviced node. In addition, the independent functioning of the nodes which could have their own workload makes those schemes difficult to apply in practice. The current proposals for CPS environments provide different contributions and aim to include additional methods to overcome these issues such as different criticality level of the tasks [37-39, 47], mathematical techniques [27, 35, 44], agents [41], and control mechanisms [46]. Other works deal with network and physical dynamic issues [36, 40, 42, 43].

2.3. Cloud-Aided Distributed processing

Cloud computing refers to both the applications delivered as services over the Internet and the hardware and software in the data centers that provide those services. The data center hardware and software is what we will call a cloud, and the service being sold is utility computing.

Cloud computing can also be used to extend the limited computational resources available in the CPS and other mobile devices. In this way, Mobile Cloud Computing (MCC) trend is especially relevant. MCC is an emerging distributed computing paradigm that aims to augment the resources of mobile devices by leveraging the resources and services of remote cloud [48]. The most

common uses of this paradigm are primarily targeted to extend the battery life of mobile platforms [49], without considering the versatility that the remote computers can provide to extend the computing power of devices. Recent approaches already consider the increase in performance as one of the most important contributions of MCC to the mobile computing [50, 51]. The two main changes can be summarized as (i) the homogenization of devices' computing capabilities since they can run applications regardless of their native hardware and (ii) the overcome of the limitations of nodes in execution of advanced applications.

In this way, MCC leverages applications requiring integration of data coming from geographically distant computers and it is a key paradigm for developing and support IoT and CPS modern applications. Therefore, advanced applications can take advantage of both the central cloud and the CPS nodes to finish the computing tasks in the system [44]. This scheme can be generalized to move the workload to other dedicated processing centers near where the data are acquired. These scenarios correspond with the deployment of *Cloudlets*, *Mobile Edge Computing* and even *Fog Computing* infrastructures [12]. All these architectures perform the computation by sharing the application tasks among the local devices and other platform, where this latter can be a variety of platforms at several network layers.

2.3 Findings

After reviewing the previous work, some findings can be drawn to justify and summarize our contributions to the state-of-the-art:

- One of the key aspects of the evolution of CPS systems has been to expose the acquired data and computing capability of the devices to the outside. Networking these devices allows designing advanced application and achieving more ambitious goals.
- How to enable the CPS nodes to efficiently collaborate to accomplish more computing tasks is a very challenging problem and an important issue to improve the CPS applications.
- Conventional task scheduling schemes in embedded real-time systems are unable to satisfy performance requirements of CPS due to its task diversity and system heterogeneity. There is much research work still to be done in this area.
- Recent proposals try to overcome the performance drawbacks by means approaching the cloud computing resources to the nodes of the CPS that are going to consume it. An interesting research line arise in relation to CPS and cloud computing integration.

The scheduling mechanism based on sharing the processing between distributed devices of the CPS and cloud computing resources seems to be a promising way to increase the capabilities of the system and to achieve greater overall performance. The efficiently dynamic allocation of tasks is a very important and difficult topic on CPS environments. In this way, the knowledge of potential applications and the resource requirements for each task can open ways for developing new methods for enabling this collaboration and joint computation in an online and distributed fashion. This research work develops this idea by proposing a computational model to distribute the processing along the whole system and meet the applications' requirements. The key novelty of this model lies in considering the computing capabilities, features and current workload of all devices to perform the scheduling of the tasks.

3. Distributed computational model

A model of computation is defined in this section based on data flow processing between tasks that run in a parallel and distributed way. This model aims to leverage the computing infrastructure for distributing the application tasks. The model introduces a formal method for modelling CPS environments and enables to define the efficient distribution of tasks according to some objective function. The notation introduced is used to define the targeted applications in a precise way, and the main components on which the proposed solution is based.

To illustrate the basic idea, Fig. 1 draws an example of the overall system in different time instants. This example corresponds to a simplified version of a smart lighting application for smart cities. In this application, the street urban lamps are switch on or off depending on the presence of humans or vehicles in the street [23]. The application decomposition and the specific tasks described by the example have been taken from the state-of-the-art research works on human detection from images and videos [52]. This example application is made of seven tasks in a pipeline and three computing platforms: a smart sensor, a mobile device and a cluster server.

The figure shows three possible distribution of the tasks for this application which configure system states where the tasks are principally made by one of the platforms. In Fig. 1.a, the smart sensor is responsible for the initial tasks from capturing data (i.e. samples, frames) and preprocessing them by extracting the relevant features. Next, another device of the CPS environment made a middle task and then, the processed data is sent to the cloud for completing the work. The behaviour can be common in environments consisting of embedded smart cameras capable of executing some image processing methods and extract meaning from streaming video [53]. Fig. 1.b shows the scenario where the sensor device captures the data, and the majority of the tasks are made by another device of the CPS. Last, the outputs produced are sent to the cloud server for processing the predictive analysis. Finally, in the scenario depicted in Fig. 1.c, the devices of the CPS make little work of the application, and the most part is performed by the cloud platform. This scheme is mandatory in CPS systems where the acquisition devices do not have enough processing power to run complex algorithms [54].

The transition between the possible scenarios must be done according to available resources and the processing

costs, so that, the costly tasks are preferable to be processed on the resource-rich platforms (cloud systems in the example). Thus, if the devices of the CPS are not under intensive processor use, they can compute the application workload (Fig. 1.a and Fig. 1.b); in contrast, when they are heavily loaded with other user activities, the data are sent to the server for further processing (Fig. 1.c).

In general, data flow switching between computers can be decided for each task according to a variety of criteria, not only the availability of computing resources. For example, for some mission critical real-time modes (as the ones described in [55]), keeping the data flow inside the embedded device will be preferred, as running without external resources could be more easily predictable in terms of performance. However, loading tasks into a server, if available, will be chosen when the possibility of correlating streams from several hosts is a desirable aspect. In addition, overall system resilience can be achieved as a secondary effect, due to the multiplicity of computers supporting tasks and the capacity to switch between them in a dynamic way.

The presented work contributes with a formal model that provides some advantages. First, it allows characterizing a set of applications with common features shown by CPS: those applications are made of several digital signal processing tasks sharing data flows, with the possibility to be run on a heterogeneous subset of computers, including intelligent sensors, cloud services and other computers. Second, the formal model provides a framework to define resource utilization, that will be used for a double purpose: represent the state of each CPS node in terms of resource utilization (computer load) and establish a way to specify the impact of the different application tasks on the CPS node (tasks requirements). Third, it leverages an architecture that flexibly hosts different solutions for system monitoring and scheduling, allowing versatile adaptation to the application characteristics.

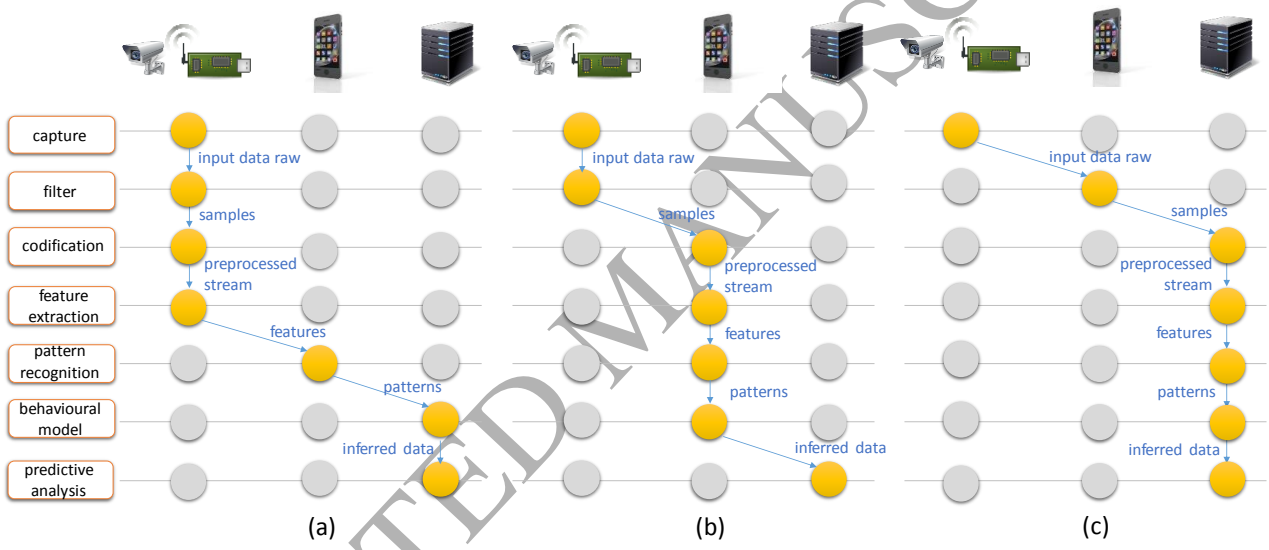

Fig. 1. Behavioral model built from a smart sensor, mobile phone and cloud. (a) Workload mainly processed on smart sensor; (b) Workload mainly processed on mobile device; (c) Workload mainly processed on the cloud

3.1. Target applications

The CPS applications targeted by the proposed model are characterized by the following features: (a) the application gets input data from the physical surrounding that need to be processed; (b) the work to be done can be decomposed into a set of individual tasks or processes, sharing data flows through the network and therefore, running in a parallel and distributed way; and (c) the results of the processing are translated into a set of actions that are performed by storage or actuator nodes.

In other words, those distributed applications can be represented by a directed graph $A = \{T, F\}$ where:

- T is the vertex set and represents the set of tasks required for data capturing, processing, storing and actuating.
- F is the edge set and represents the data flows exchanged between tasks.

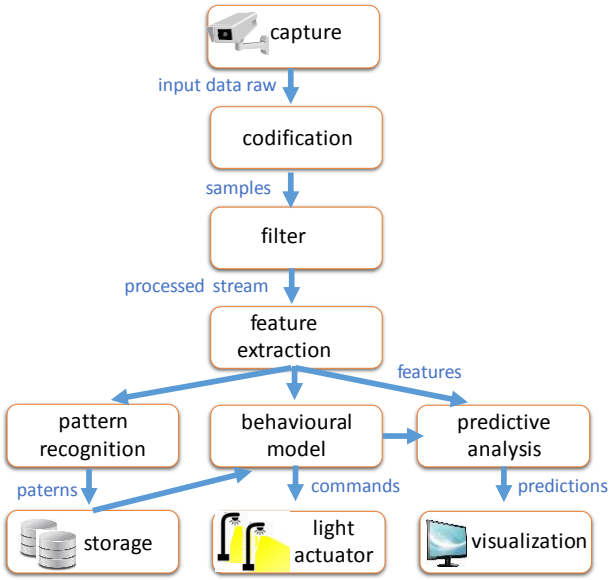

Fig. 2. Task and data flow decomposition for an example distributed system

The data flow diagram shown in Fig. 2 depicts the previous distributed application modelled according to this principle. The data from presence sensors are analyzed in accordance with a specified operating method and it shows some common operations in the area of data processing from digital signals in a possible CPS context.

The task and data flow decomposition represented by the directed graph A is designed by means an application partitioning method and the desirable granularity unit. This granularity unit determines the size unit of the application that can be offloaded to other platforms of the network [56]. A coarse-grained allows a high-level of abstraction and simpler mechanisms but increases the need for communicating application details and synchronization. In contrast, a fine-grained offers more opportunities for offloading but it needs much more scheduling work. The desirable granularity unit for outsourcing should be as small as possible to provide the highest flexibility, but small sizes imply higher management cost. In this way, the granularity unit for the distributed model could be variable depending on the features of the target platform for offloading. That is, little parts of the application can be outsourced for fast execution on surrounding platforms of the CPS and other intensive parts for offloading to external specialized platforms. The optimal partitioning is an NP-complete problem [51], in order to avoid time-consuming in automatic analysis of the code, the construction of the graph A for each application may be made in the design stage. The proposed characterization for CPS environments is based on a coarse-grained data flow approach which is a natural paradigm for describing digital signal processing applications for concurrent implementation on parallel hardware [57].

3.2. Resource specification

Following with the domain characterization, we define C as a set of computer elements where different instances of tasks in T can be potentially run. The C set is formed by the computing platforms of the CPS and the accessible cloud resources. This is the operating network where the system can be deployed.

This model could handle a dynamic computing power by including at design time the kind of potential devices that can exists at any time in the network. Let Λ be the different types of CPS nodes. For example: sensing nodes, wearables, mobile phones, laptops, server clusters, computer systems, etc. In this way, each device of C is defined as a type of Λ. Fig. 3 shows an example of the different types of nodes, coming from the deployed infrastructure, the available devices and the hired cloud resources.

To distribute resources in an efficient way, the resource utilization of the platforms must be properly characterized. Let Ω be the set of all possible required parameters of the CPS applications.

$$\Omega = \{p_1, p_2, \dots\} \quad (1)$$

where each p_i represents a performance parameter such as 'transfer rate', 'processor load', 'memory use', 'battery consumption', etc.

Next, the proposed model introduces L as a vector domain of the subset of the n relevant features for quantifying the required resources of a specific application 'App'.

$$L_{App} \equiv \Omega^n \quad (2)$$

Thus, the list of parameters of L_{App} depends on the 'App' application requirements. Therefore, the list can be different for each application. This approach provides flexibility to face application contexts with different requirements and opens several possibilities for sharing the workload.

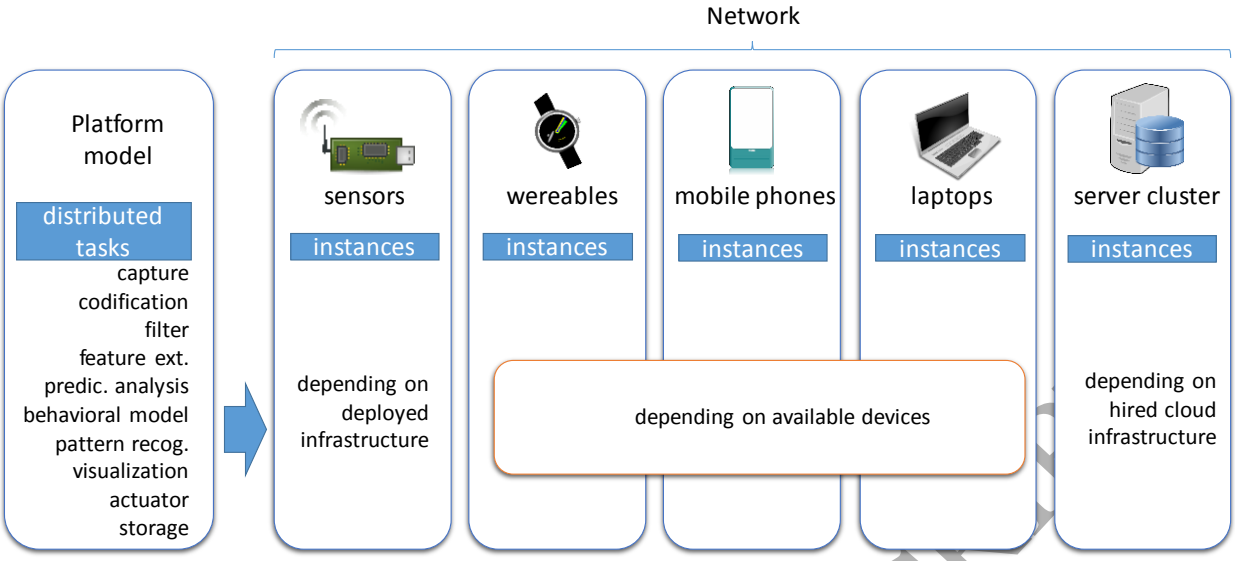

Fig. 3. Different sources of computing resources for the distributed system

The list of features is defined previously according to the CPS application needs. These are generally referred to computing resources, but other kind of components or user experience factors as energy consumption could be considered for practical convenience. For example, if some application A requires a specific capture device, present only in some computers, a component 'Capture device' can be added. In this way, the intention is that all the necessary requirements are included in L from the beginning. The next example shows the L vector defined as a vector with four components, with the following semantics:

$$L_{App} = \text{Memory use} \times \text{Transfer rate} \times \text{Processor load} \times \text{Battery consumption}$$

Once the L vector has been modelled by selecting the key resources for the target application, the resource utilization of the available CPS nodes can be represented as a vector of n-components. The $l_{c,time}$ vector defined by expression (3) quantifies its load in a time instant, and therefore its ability to run tasks in T.

$$\forall\, c \in C,\ l_{c,time} = \langle p_1, p_2, \ldots, p_n \rangle \in L \quad (3)$$
where $p_j \in \mathbb{R} \cup \{0\}$

The utilization of each component p_j is expressed as a non-negative relative value according its availability for running the tasks in T. The value '0' means maximum resource availability and the values equal or greater than '1' means that this resource cannot be accessed. In this way, the computer load is formally quantified as a tuple describing the fraction of relevant resources currently in use. Obviously, these values are different for each time instant, depending on the different activities in which the device is involved and the CPS context. For multi-application scenarios where several requirements and L-vectors exist, at each instant, the devices construct the union of the vectors for all distributed applications. Each application takes into account only its corresponding subset of features.

By characterizing the current computer load (current state) with relative values, the computer shows its ability to run the tasks in a homogeneous and comparable way, from the point of view of the different features modelled by L. For example, a dedicated server for the CPS application will generally show low values for 'Processor load', excepting situations near overload. However, a user device such as a smartphone will show high values for this feature if it is busy just with some user activity, showing its inability to run heavy tasks (in this case, from the 'Processor load' point of view) from T.

Of course, a suitable method is required to estimate the relative value to each component of L. These methods must be light processes to avoid, wherever possible, interfering with the device operation and, in addition, they should be compatible for the devices of each kind of node in Λ. The next paragraphs describe some examples for the previous resources of L.

The 'Memory use' component value can be computed just by dividing the amount of memory assigned to processes by the total amount of memory available. Generally, this information can be provided by the operating system.

The method to quantify the 'Transfer rate' component will take into account the characteristics of the interconnecting network. The current transfer rate can be known by each device, just by monitoring the network interface. However, in order to get a relative value, the total bandwidth available for the device is required. In a CPS where computing nodes are mobile and connected through standard wireless networks, knowing the total free bandwidth is not a trivial issue. First, in those standard networks, the bandwidth is shared among the interconnected devices, and second, the position of the mobile devices affects to the available bandwidth. For those cases, one feasible approach is to consider an estimation of the total bandwidth based on the average

values retrieved on periodic checking performed by the devices.

To properly set the component 'Processor load' one approach is to consider the average number of processes that are in a runnable state during the last minute. In the case of a multiprocessor computer with two cores, could reasonably accept up to two processes at a time, from the processor load point of view. A '1' value for the 'Processor load' component in this device means that it is supporting the maximum processor load (two processes average), and a new task requiring more processor load should not be accepted. Therefore, a good 'Processor load' quantification can be obtained by dividing the average number of processes in a runnable state during the last minute, by the number of cores, and taking the minimum of 1 and this quotient. From this approach can be deducted the processor utilization rate regardless number of cores.

Finally, the 'Battery consumption' component can be set in different ways. One possibility is to consider the battery discharge rate (percent of charge consumed per time unit), which ranges from a minimum value when the device is idle to a maximum value when the device is using all the battery consuming resources in an intensive way. In this case, the 'Battery consumption' component can be computed by dividing the current discharge rate by the maximum discharge rate, previously subtracting the minimum discharge rate from numerator and denominator.

Since the key features for the application are set from the beginning, several types of L could be defined to cover different types of applications running on the CPS environment. In this way, multi-application scenarios can be defined. The variability of the applications is not expected to be very high in a controlled CPS environment. For example, in a smart city, the types of distributed applications can be restricted to a small set according to the service provided. So that, a new task cannot arrive demanding new features in L.

In addition, operating conditions could be included implicitly in the formulation to make the scheduler work according to efficiency criteria and follow scheduling preferences about the computing resource consumption of the devices. For example, if some device's resource need to be reserved for private use only, its component of L vector can be set to '1'.

Once the L components are set and the obtaining methods for each of them are established, the effect in a device when a new task arrives needs to be defined by the model. As general case, when a new task is run on a node of a CPS, this computer will experiment an increase of its load described in terms of the components of L. For each computer in C and each task in T, this load increase must be estimated taking into account the task requirements. These requirements are modelled using a matrix R, where each element $r_{t,c} \in L$ quantify the requirements (load increase) of a task $t \in T$ in a device $c \in C$.

The vector $r_{t,c}$ effectively models the feasibility of a device (c) for a given task (t). Its values can be obtained empirically by testing each task on each device. However, in many cases, some values can be deduced from the results got in some reference computer c_0 for each kind of device in Λ. As a trivial example, if a task t_1 holds $r_{t1,c0} = 0.5$ for the Memory use feature, in a computer c_0 with 1GB, it will reasonably hold $r_{t1,c2} = 0.25$ for an equivalent computer c_2 with 2GB. Other computing resources will imply other calculations, and in some cases, empiric test will be required, as in the case of the processor load in a computer equipped with powerful GPU. The device type classification of Λ will help in order to assume a pre-estimated $r_{t,c}$ for those computers belonging to a defined profile or device class.

To illustrate how the task requirements are modelled, the next example considers a simple CPS made of a single-core intelligent sensor with a camera (c_1) and a laptop with N cores (c_2) but without a proper camera. The sensor device allows up to P processes using the capture interface. For simplicity, the cores of c_1 and c_2 are assumed equivalent. The application is made of two tasks: t_1, which is an image capture task, and t_2, which implements a heavy image processing algorithm. For the application convenience L is defined as 'L = Processor load × Camera availability'. As a result of a test, it has been found that t_1 increment the processor load (one core) by 10%, and t_2 produce an increase of 50% in the processor load (one core). In this example, the tasks requirements can be defined by the following R matrix:

$$R = \begin{pmatrix} (0.1, 1/P) & (0.1/N, 1) \\ (0.5, 1/P) & (0.5/N, 0) \end{pmatrix}$$

From the content of the R matrix, it can be deduced that c_1 is feasible for t_1, and c_1 and c_2 are feasible for t_2. As another example, considering 'L = Processor load', a laptop computer with a powerful GPU will generally show a low $r_{t,c}$ for tasks t requiring heavy image processing; in other words, this quantifies the degree in which the computer with GPU is feasible for image processing services.

The definition or R allows to configure the amount of resources dedicated to the CPS tasks by a device. For example, setting a value of 1 for the 'Memory use' component when a task requires the 50% of the total memory, configures a device to dedicate only the 50% of its memory to the CPS tasks.

3.3. Proposed solution design

The proposed solution consists in managing the application's tasks and its data flow along the computation nodes of the network according to the available resources, the processing needs, and the user preferences. This management system is made of two main components: (1) special proxy local processes running on each computation node of the network, we call them switches, and (2), a system controller which maintains a view of the overall system, and offers several framework services as discovering new nodes, monitoring tasks, and planning the source and target computer for each required data flow. These works are made in a centralized way in order to minimize the management communication costs and saving shared resources.

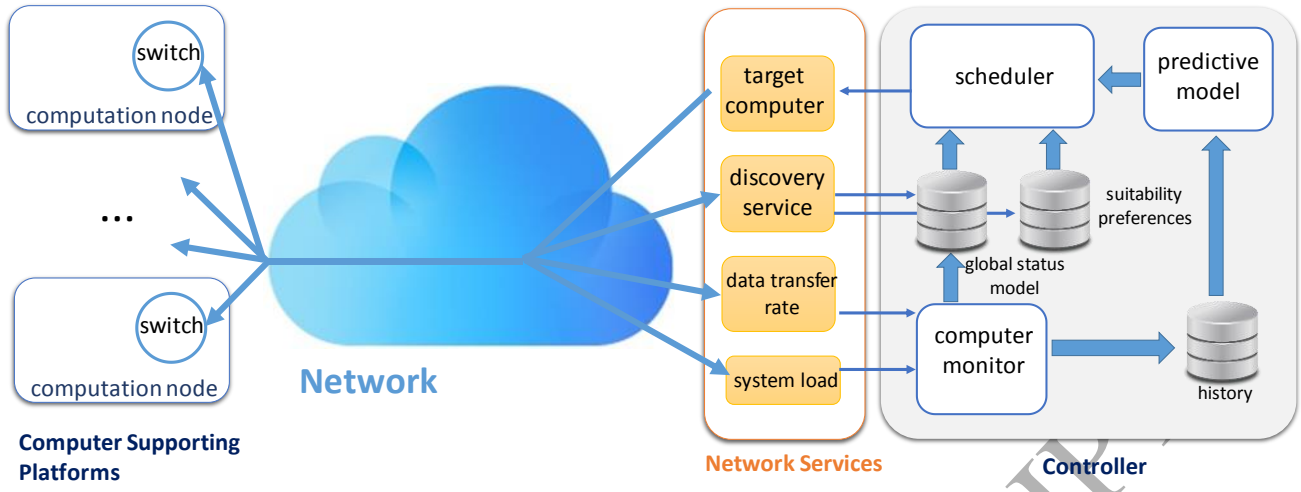

Fig. 4. Diagram of the proposed solution and the elements of the management system.

The system controller can be deployed in the cloud or in another computer of the CPS with enough resources and close to the rest of the nodes.

Fig. 4 shows a high-level view of the proposed solution. The main elements of this component are described as follows:

3.3.1. Computer monitor and discovering service

The CPS system is an open environment where new devices can appear. To manage this dynamic operation, a discovery service is added to conduct the following key capabilities: register a new device and unregister it when it does not available.

When a new device arrives to the CPS, the discovery device service register it in the system and retrieves its hardware inventory by using the SSH (Linux, UNIX) or SMB or WMI (Windows) protocol. To handle different performance features, a new table has been added to the system to keep the base performance load increase for each type of device. Thus, a matrix K is defined where each element $k_{t,c} \in L$ quantify the requirements of a task $t \in T$ in a type of device $c \in A$. The values in $k_{t,c}$ model the performance load increase for a base device representative of each type of node. For example, the base performance load increase for the 'memory use' parameter can be calculated in MB; the base performance load increase for the 'processor load' parameter can be calculated in 1Ghz/core; etc. Then, when a new device arrives to the environment, a new column in the R matrix to store the specific load increase for that device is created by combining the base requirements of K with the hardware features of the device.

The computer monitor feeds the global status model with the relevant parameters, $l_{c,time} \in L$, from all computers $c \in C$ each time instant. Different strategies can be applied here, taking into account the asynchronous nature of networks and the type of node. One possibility is to make all nodes to respond to a periodic request from the controller component. Secondly, the periodic request can be customized for each type of node. In this case, the last l_c received is used for scheduling. Other solution is to share a global clock signal by using, for example, Network Time Protocol (NTP). Broadcast communication should be used where possible.

The computer monitor also integrates computer unavailability when it happens without unregistering, their resources are required by their intended priority function or any other user defined restriction become in place.

Some parameters may be subject to short abnormal fluctuations due to performance peaks. For example, a smart camera has a consumption peak when takes a frame. If the performance parameters are required at this time, the resource utilization of this platform does not adequately report the normal consumption state of the device. A suitable procedure to avoid outlier values on performance parameters is to store the data vectors $l_{c,time}$ in a history database. This information is used as an input for a feasibility predictive model, so the scheduling can be performed not only based on the result of direct measures, but also on mined knowledge.

3.3.2. Predictive model

Following the notation introduced in Subsection 3.2, now it is time to predict the load of each computer from C when running a specific task of T. For this reason, an internal binary operator is defined in L and it is represented using the symbols $\oplus$ and $\sum^{\oplus}$ for the accumulation. The operation specifies the procedure for adding the task requirements to the current device load, allowing to deduce the load of the device if a task of the application is run on it. The formal definition of this operator is expressed by equation (4).

$$\oplus: L^2 \longrightarrow L \tag{4}$$

This operator calculates the device load predicted if a task t is assigned to a computer c. As shown in expression (5), the inputs of this operator are the resource utilization vector L of the computer ($l_{c,time}$) and the increase of load if the computer run this task ($r_{t,c}$). The output of this operator is the resource utilization of the computer for the next cycle when the task t is assigned to computer c.

$$l_{c,\,time'} = l_{c,time} \oplus r_{t,c} \quad (5)$$

where $l_{c,\,time'}$ is the load predicted for the next cycle if the task t is assigned to computer c.

The implementation of $\oplus$ will depend on the nature of the L component considered. In most of the cases, this operator is just an addition of the current load and the increased load in executing the new task. For example, the 'memory use' is the most obvious example of this calculation. In other cases, the predictive calculation considers additional factors to increase the accuracy for the prediction. This is the case of the 'Transfer rate' parameter where the available bandwidth of a wireless network may experience shifts in each instant because of the effect of mobility among other reasons.

Let us take the following case as a combined example of L with different calculation methods: L = Transfer rate × Memory use. Now, let $l = (l_1, l_2)$, $r = (r_1, r_2)$, where $l \in L$ be the current load of a computer, and $r \in L$ be the requirements of a task in this computer. Then,

$$l \oplus r = \left(\frac{B_{time} \cdot l_1 + B \cdot r_1}{B_{time}}, l_2 + r_2\right)$$

where B is the total bandwidth available when the r_1 was determined, and B_{time} is the total bandwidth available in the current time.

The quality of the prediction made by this operator lies in the accuracy of the requirements modelled by R. In addition, it can be improved by integrating history information through advanced machine learning techniques that allow to estimate the future performance from past behaviour; this approach is more appropriate for components that can be greatly influenced by external factors, as total free bandwidth in a shared medium network. In general, a sophisticated implementation for $\oplus$ will require more information.

3.3.3. Scheduling

The main element in the system controller, as shown in Fig. 4, is the scheduler component. It uses the model and information provided by the remaining controller components in order to decide the target computer for each required data flow.

Two functions, feasibility and a suitability, are defined to perform the scheduling of the tasks. In first place, the feasibility function models the real possibility of a device to work with a given load and points out the overload cases. The expressions (6) and (7) define this function.

$$\text{feasibility: } C \times T \rightarrow \{\text{True, False}\} \quad (6)$$

That is, for a given $c \in C$, $t \in T$, at each time instant the feasibility function obtains:

$$\text{feasibility}(c,t)_{\text{time}} = \begin{cases} \text{True, if } \forall p_j \in \langle l_{c,\text{time}} \oplus r_{t,c} \rangle .\, p_j \leq 1 \\ \text{False, otherwise} \end{cases} \quad (7)$$

This information is used for the scheduler module to decide if a computer node can process a task. Then, a first approximation of the scheduling problem can be stated by finding a correspondence function as:

$$\text{schedule: } T \rightarrow C \quad (8)$$

satisfying the following expression (for each given time instant):

$$\forall\, t \in T, c \in C.\, \text{feasibility}\,(c,t)_{\text{time}} \quad (9)$$

The CPS application could include some execution requirements to the system such as time constraints or optimum use of resources. To meet this kind of conditions, the scheduling process becomes a NP-complete problem which it can only be resolved by heuristic or search algorithms [58]. Instead, this work is focused on providing agile and effective solutions for the general case, where the most important issue is that each task of the application can be processed by some device of the system. In this way, the scheduler module proposes a possible scheduling order. When a new task arrives, the work just consists in checking the feasibility function for the nearest device to the data source for this task. The cost of this operation mode is linear with the list of application tasks T.

Secondly, a step forward in efficient scheduling can be set by considering configuration preferences in processing the tasks. This approach provides criteria for scheduling by means some suitability information modelled by the function described in expression (10).

$$\text{suitability: } \Lambda \text{ x } T \rightarrow [0,1] \subset \mathbb{R} \quad (10)$$

This function quantifies, in an increasing preference-scale, the preference for processing a task $t \in T$ in a type of device $d \in \Lambda$ of the CPS environment. Then, instead of just a feasibility function, the scheduling method can find the correspondence that maximizes the value of the following expression:

$$\sum_{\forall c \in C, t \in T} \text{suitability}(c,t) \quad (11)$$

The suitability information is stored using a new matrix S, where each element $s_{t,c}$ represents the value for suitability(c, t). Thus, the values in $s_{t,c}$ effectively model the suitability of a device for a given task, and then, although there are several feasible devices to perform a task, the more suitable will be selected in first place. Now, the scheduling method checks the feasibility function for the available devices in a decreasing suitability order. The cost of this operation mode is linear with the product of the available devices of the system C by the application tasks T. An improved design consists in sorting each row of the S matrix to provide the list of suitable devices for each task in decreasing order.

Continuing with the example above —the CPS which consists of an intelligent sensor (c_1) and a laptop (c_2)—, the task t_2 can be processed both in c_1 and in c_2. The matrix S

defines the suitability of performing the processing for each task and computer. For example:

$$S = \begin{pmatrix} 0.9 & 0 \\ 0.2 & 1 \end{pmatrix}$$

shows that the task t_1 is better processed in c_1 while the task t_2 is better processed in c_2.

The data of the matrix S can be set by the application designer according to the CPS properties, the configuration of the devices and the user preferences. In addition, these data can be updated dynamically according to the evolution of the vector set L. That is, when the computer load of a device raises, its preference for scheduling may decrease. For example, if the remaining battery is low for a device, the preference level of this device would fall.

The matrix S drives the scheduling method of this architecture. For each task, the devices can be ordered by preference rate and then the feasibility of each combination can be checked. In this way, the feasible combination with best rate will be selected in first place. Note that this method is designed specifically for a CPS or IoT environments. Thus, it focuses on the 'things' and not on the overall aspects of the application.

This approach considers the dynamic nature of the CPS environments because the things can change its operation conditions and preferences, and eventually, appear/disappear in/from the system. When a new device goes in/out the system, the matrices R and S are expanded/contracted with the data. The feasibility and suitability functions work with the available devices and the new data for distributing the application workload.

In this point, other strategies and policies focused on the application could be addressed, mainly by adapting the successful results from previous and future research. As a result of our previous research on distributed and mobile systems, a proposal combining imprecise computing strategies with cloud computing is introduced in [59], and it can be used for the scheduler component.

3.3.4. Data flow control

All task instances get their input data flow from either a switch process or from sensor/acquiring devices. The output data flow is also sent to a switch process, or used directly for performing a proper action (using storage or actuator devices). Each switch process directs the data flow to the corresponding target task instance, located in the same or different computer.

Generally, the decision is taken by the controller component, which maintains a representation of the overall system and runs the scheduling algorithm. If the controller is not available (for example, because of network or host failure), the switch process runs a fallback procedure, using only local information.

The control flow of a switch process is shown in Fig. 5. One interesting point in the figure is the value of the N threshold and the size of the data block. In general, the size of the data block will be variable and it can be specified in different ways. For example, for a video capture process, one block can include a fixed number of frames, or the number of frames captured during a fixed time period (discontinuous capture makes the difference). The higher the block size and N threshold, the lower the use of resources by the management system, and therefore, better efficiency can be achieved.

By contrast, decreasing these values improves the accuracy of the system controller, as it allows to maintain a more accurate picture of the global status, and therefore, more appropriate decisions can be taken. The proposed operation model needs the configuration and the collaboration of the computational elements, as well as the deployed applications in the CPS. In the next section, the framework for designing the systems is described.

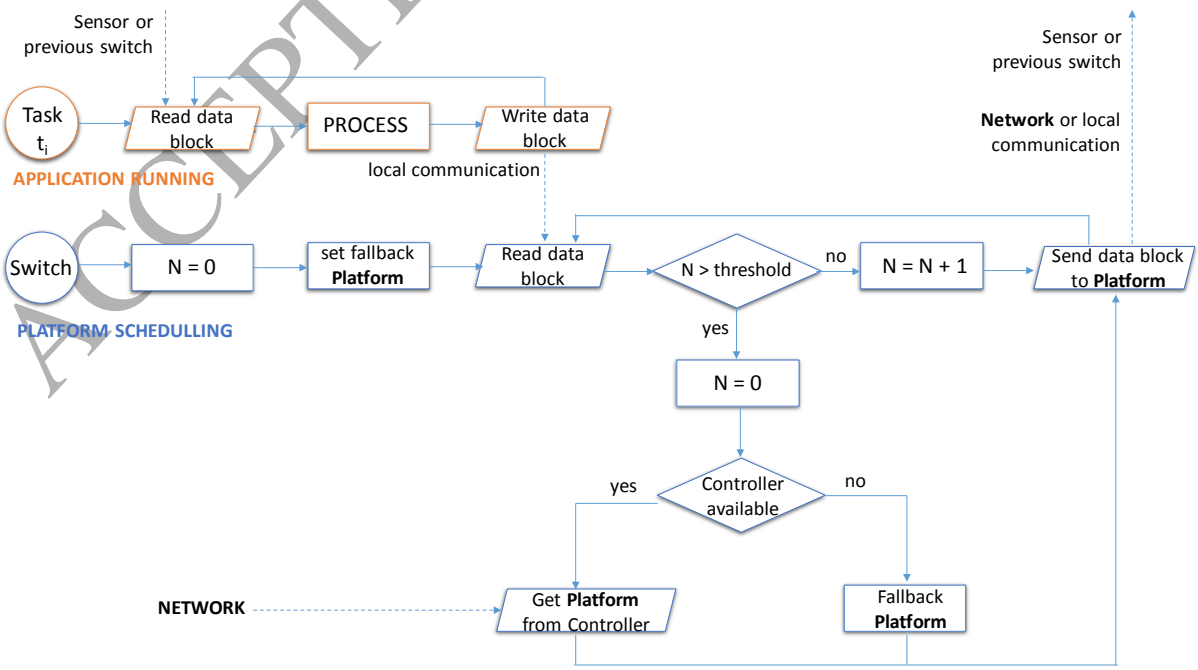

Fig. 5. Flow diagram showing the control data flow of a switch process

By contrast, decreasing these values improves the accuracy of the system controller, as it allows to maintain a more accurate picture of the global status, and therefore, more appropriate decisions can be taken. The proposed operation model needs the configuration and the collaboration of the computational elements, as well as the deployed applications in the CPS. In the next section, the framework for designing the systems is described.

4. Distributed application framework

The design of applications for CPS under the proposed model consist of three main steps: (a) task and data flow decomposition; (b) resource planning and (c) deployment and empirical adjustments. Fig. 6 shows a general overview with the inputs and outputs of each design stage.

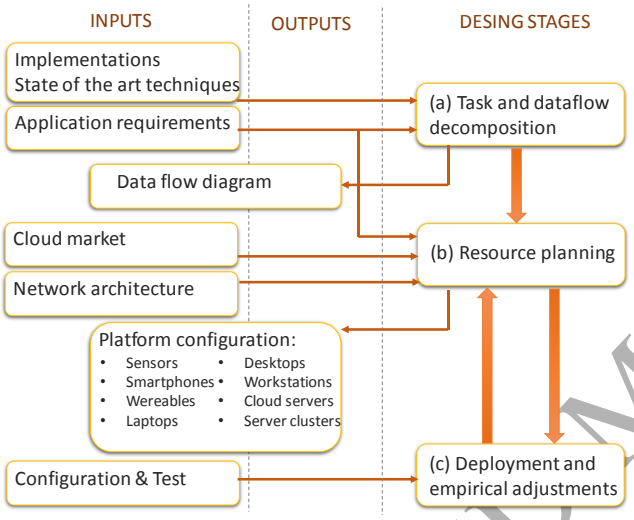

Fig. 6. Distributed application design scheme

4.1. Task and data flow decomposition

There are different ways in which the overall activity of the application can be decomposed in different tasks, sharing data flows, in order to build A = {T, F}. This decomposition depends mostly on the application area. For example, some common examples of distributed tasks in predictive analysis applications are data capture, feature vector extraction, building predictive models, pattern matching, data flow correlation, raising alerts, logging, etc. [60]. In addition, the availability of already implemented components can also be considered; it could be desirable to reuse existing software (legacy, open source, etc.) to implement some application subproblems; this software would implement a distributed task of the application.

Generally, the tasks get input data from the output of other tasks. However, initial tasks get the input directly from capture devices. Similarly, final tasks use their output to feed a physical action (for example, switching-on a lamp or raising an alarm) or just to send data to storage devices for logging and further analysis. This model allows to combine several strategies or implementations in order to put a more robust solution in place. For example, well-known, extensively tested techniques can be complemented with experimental tools working in parallel.

4.2. Resource planning

In the distributed system, each task can be run on a different computer $c \in C$, with different capabilities. In order to determine the most suitable computer where to run each task, a number of factors must be taken into account (components for the domain L): computing resources available in user devices, battery consumption in mobile devices, network bandwidth availability and latency, real-time requirements for some tasks, capture and data flow integration imperatives, etc. Most of these factors change over time; for example, the computing resources available in a specific computer or the free network bandwidth.

For each computer, the fraction of its resources which can be dedicated, as a maximum, to the tasks in T, must be quantified; in other words, a proper semantics must be defined for the values of the different components in L. Therefore, $l_{c,i}$ will show the fraction of the resources available for the application in a time instant i. In addition, the requirements of the application must be defined in terms of r_{tc} (requirements for each task t in each computer c). As explained in Subsection 3.2.2, this can be done by combining empiric test and deductive results.

4.3. Framework deployment and empirical adjustments

The implementation of the different tasks of the distributed application must be properly deployed on the required computers, along with the modules and services of the proposed model. For each computer or computer profile, the resources defined in the previous section must be properly configured. After that, the performance of the key computers under each profile should be carefully measured. It is necessary to check that the computer has always enough resources for meeting the requirements for its intended functions. In addition, the distributed application must offer results as expected, switching data flows between computers according to defined conditions. The resource planning step should be reviewed and adjusted until the overall system performs as expected; proper system modelling and simulation can help in the successful completion of this step.

5. Case study: distributed intrusion detection system

A case study of a distributed application is presented in this section. The presented application is suitable for the proposed model and designed according to the described framework. The aim of this case is to show an example in a real domain where some advantages from the proposed ideas can be obtained. The design of a *Distributed Intrusion Detection System* (DIDS) by following the proposed method, and further deployment using the framework, is approached.

Security is a big concern in IoT and CPS [61]. One of the main approaches to information security and cyber security has been the development and deployment of *Intrusion Detection Systems* (IDS), also in the context of IoT and particularly for CPS [62]. An IDS dynamically monitors the actions taken in a given environment, and decides whether these actions are symptomatic of an attack or constitute a legitimate use of the environment [63]. Since the initial proposal of this approach, a lot of intrusion detection tools, techniques, projects, and products have been developed. Data mining and machine learning have been at the core of many of these results. Nowadays, there is also an increasing interest in IDS topic as shown by the recent advances in anomaly detection [64], wireless sensor networks [65], CPS [66], smart grids [67], among many others.

5.1. Task and data flow decomposition

The proposed DIDS is designed by selecting and combining a number of existing IDS solutions. The first stage is the task and data flow decomposition. Fig. 7 shows the data flow diagram for a possible DIDS.

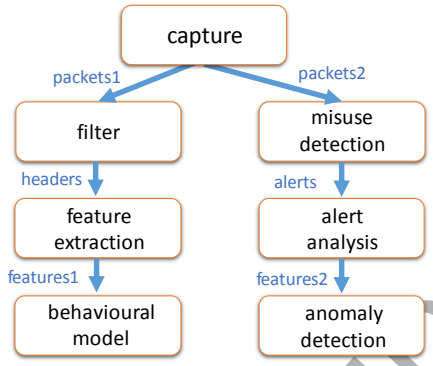

Fig. 7. Possible task and data flow decomposition for a generic DIDS

As can be seen, the capture task generates a packet flow, feeding two different IDS approaches. On one hand, misuse detection and alert analysis is applied; this will be implemented by standard well-proven tools. On the other hand, anomaly detection is performed; state-of-the-art techniques will be put in place for evaluation and further research. Anomaly detection involves some complex processes that can be run in parallel in a distributed way: filtering (extracting headers from packets), extracting features and building a stored behavioural model to be used for further anomaly detection. Table 2 describe details of the application tasks and data flows.

Table 2. Possible task and data flow decomposition for a generic DIDS

T (tasks)	**F (flows)**
Capture	packets1 (capture; filter)
filter	packets2 (capture; misuse detection)
feature extraction	headers (filter; feature extraction)
anomaly detection	features1 (feature extraction; behavioural model)
behavioural model	features2 (feature extraction; anomaly detection)
misuse detection	alerts (misuse detection; alert analysis)
alert analysis	

The DIDS described in Fig. 7 has been implemented by using some state-of-the-art tools and techniques. The objective is not to build a full operative IDS, but to illustrate the proposed method and test the feasibility of the proposed framework. Table 3 summarizes the tools deployed and relevant projects referenced for that purpose.

Table 3: Tools and projects on which the experimental DIDS is based

Task	**Tool/Project**
Capture	Tcpdump/Libpcap
Filter	TShark (part of the Wireshark® network analyser)
Feature extraction	MINDS (Minnesota INtrusion Detection System)
Behavioural model	Snort.AD
Anomaly detection	Snort.AD
Misuse detection	Snort®
Alert analysis	Hadoop®

Tcpdump with the corresponding *Libpcap* is a traditional option commonly used for capturing network traffic [68]. Our IDS runs *Tcpdump* to capture all incoming and outgoing traffic going through the network interface, with the exception of the data produced by the IDS tasks: otherwise, IDS would process data generated by itself, what may be considered useless and time consuming in most of the possible environments. The captured traffic will be further processed in two different ways: anomaly detection and misuse detection [69].

In 'anomaly detection', *TShark* is used to extract headers from network traffic in a proper format to be further processed for feature extraction. *TShark* is a command line interface in the *Wireshark suite* (https://www.wireshark.org/). The features are extracted from packet headers using an *R script*. The behavioral model and anomaly detection is built as a variation of the one provided in *Snort.AD* project (http://anomalydetection.info/), working in a standalone way (not integrated with *Snort*).

In 'misuse detection', a *Snort* sensor supplies a powerful list of rules to match network traffic against them. One typical drawback of misuse detection in general and *Snort* in particular, is the generation of many false positives that should be properly reviewed by an expert. Some of the solutions proposed in the literature include further alert analysis, using cloud big data clusters. For this reason, a *Hadoop* cluster for correlating alert messages has finally been deployed.

5.2. Resource planning

The described method can be used to properly distribute IDS tasks, considering security requirements and variable availability of computing resources. The presented approach favors the integration of limited computing resources in CPS, with externally supplied cloud services, in order to meet IDS requirements.

A simple CPS consisting of a wearable and a mobile phone device is considered in this case study. This type of device is currently very popular and common among citizens. They generally incorporate a lot of sensing features and most of the time they are idle. Therefore, they encompass the ideal target devices for distributed CPS applications and they should be used for supporting them. In addition, a cloud server is added to the framework to perform part of the processing cost. The wearable device and the mobile roles will be accomplished by low-cost, single-core, credit-card sized computers: Raspberry Pi, with an ARMv6-compatible processor and 512MB of main memory. For the server a desktop computer, with an Intel® Core™ i5 CPU and 6GB of main memory, will be employed. The computers are interconnected in a standard wireless local area network.

For pointing out the feasibility of the approach, we will take into consideration only the three tasks shown in Fig. 8. In addition, we will just consider transfer rate and processor load as the relevant performance parameters.

$T = \{\text{filter}(t_1), \text{feature extraction}(t_2), \text{anomaly detection}(t_3)\}$
$F = \{\text{packets}(t_1, t_2), \text{features}(t_2, t_3)\}$
$C = \{\text{wearable}(c_1); \text{mobile}(c_2); \text{cloud}(c_3)\}$
$L = \text{Transfer rate} \times \text{Processor load}$

The requirements of the tasks for each computer are stated by the matrix R, where the element $r_{t,c} \in L$ specifies the required transfer rate and processor load for task t in the computer c.

$$R = \begin{pmatrix} (0.5, 0.2) & (0.5, 0.2) & (0.5, 1) \\ (0.3, 0.5) & (0.3, 0.5) & (0.3, 0.1) \\ (0.1, 1) & (0.1, 1) & (0.1, 0.2) \end{pmatrix}$$

5.2.1. Task requirements

In order to determine the values for the components of each $r_{t,c}$ of the matrix R, empiric test has been conducted. For example, the transfer rate depends on the task and the maximum transfer rate allowed by the network (bandwidth). As all the computers in the experimental environments share the same network, the same estimation is shared by all computers, but is different for each task.

Regarding to the processor load, it has been observed that when the c_2 computer (mobile) runs t_2, (feature extraction), the processor load (average number of processes in a runnable state) increases in 0.5 processes; as it is a single-core computer (only one process average should be accepted as a maximum), we estimate 0.5 as the requirement for the feature extraction task in the mobile device, in relation to processor load. In contrast, the feature extraction task does not heavily increase the load of the multi-core processor in the cloud server (only 0.1 point is estimated).

Some tasks are directly discarded for certain computers. In those cases, we estimate 1, so the task will never be assigned to those computers. For example, the task 3 (*anomaly detection*) will never be run on the wearable or mobile devices, as we consider that they are always too much expensive from the computing resources point of view.

However, other tasks can be computed in several types of computer. For example, the task 2 (feature extraction) could be processed in the three computers, because these systems could have resources enough for it. The decision on where to process this task could come from the user preferences or other aspects of the distributed architecture. For example, the wearable (c_1) owner wants to minimize the utilization of his/her device for saving battery; then, he/she configures it for low utilization. In addition, the mobile device (c_2) is a better option over outsourcing the work to the cloud, because of the derived cost savings.

These conditions are particular for each CPS case, and they configure the scheduler work. The information needed is coded in the suitability matrix S, which specifies the preferences for processing the tasks. In this way, the suitability information is included into the model. For example, let the next matrix S be an example of the suitability data for the case study application.

$$S = \begin{pmatrix} 0.1 & 0.9 & 0.5 \\ 0.1 & 0.9 & 0.5 \\ 0 & 0 & 1 \end{pmatrix}$$

As shown, the mobile computer (c_2) is preferable to the other computers for processing the tasks 1 and 2. In addition, for task 3, the only suitable platform is the cloud (c_3). From these data, the distributed architecture of the CPS can decide among the feasible options for performing the scheduling work.

5.3. Deployment and empirical adjustments

A prototype of the proposed framework is provided in this subsection in order to perform some tests and adjust framework parameters for showing the viability of our approach. A minimalist implementation of the prototype has been made in order to focus on how the model works and not to interfere with the normal execution of the devices.

The implementation is based mostly on *Bash scripting*, and *Secure Shell (SSH)* as a communication mechanism for all services. To begin with, a script has been built for each task in the experimental DIDS. These scripts take input and output file names as arguments, and these file names are used inside the script to properly feed the tools in Table 3. Additionally, a prototype for the two main components of the framework must be provided: the controller mechanism and one switch process per task instance.

The initial startup of the computing nodes is done through a main program. This program reads a textual representation of A = {T, F}, C, and the R matrix. From this input, it runs the corresponding task instances by calling the task scripts, and also a switch process per task instance. The communication between task instances and the corresponding switch process is made through standard Unix FIFO streams, as shown in Fig. 8, due to its simple implementation and interoperability.

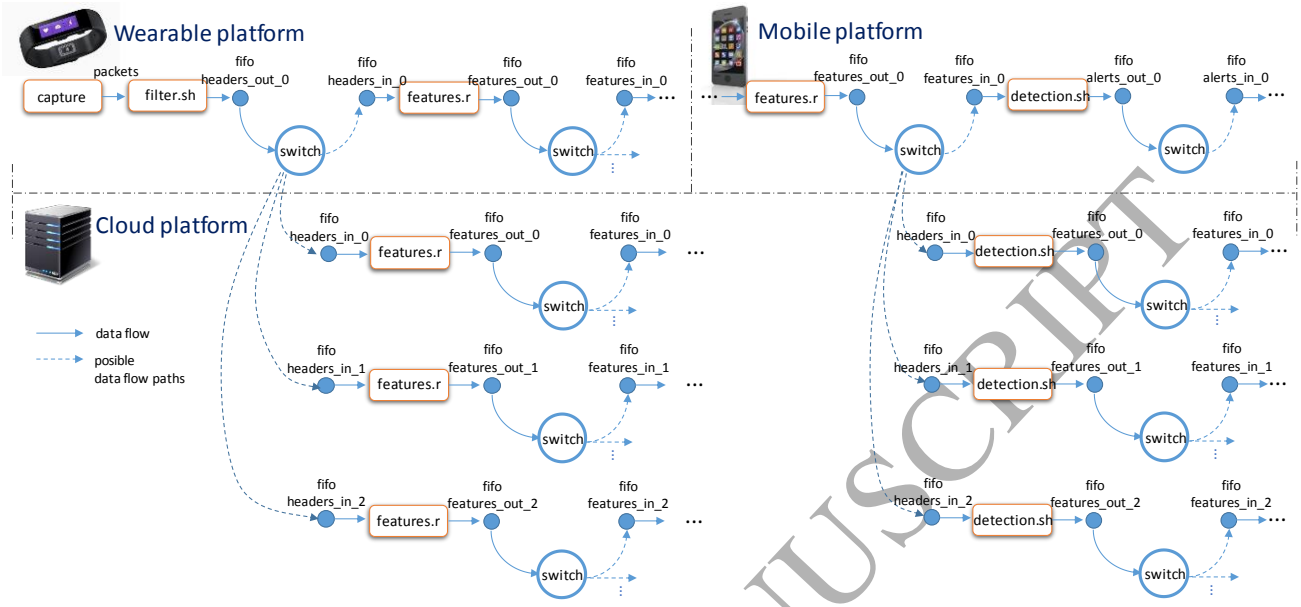

Fig. 8. Processes created in three computer profiles for a simple DIDS

The figure shows a schema of the processes created in the three example computer profiles, for a part of the simplified version of the experimental DIDS. The prototyped switch process periodically contacts the controller to find out the target computer for the corresponding data flow, following the algorithm in Fig. 5. This is done by using SSH to remotely call a command in the controlling computer. If the controlling computer is not available (it does not respond), the fallback decision is taken, namely local transfer of the flow. The data transfer is done by using *dd* along with *ssh* common commands.

The prototyped controller is made of a set of simplified processes that roughly implement those represented in Fig. 4. New computers are added by providing a script which, remotely invoked, registers the corresponding $r_{t,c}$ according to the computer profile. The prototyped computer monitor finds out current data transfer rate and system load average by remotely running *vnstat* and top/uptime commands, respectively on the target computer; the results are added into a history file. The system load average as calculated by top/uptime is the average number of processes that are using the CPU, waiting to use the CPU, or waiting for some I/O access.

A variable data block size is chosen, corresponding to all the data captured during 60 seconds. This may seem a lot of time, as it poses a severe limitation: decisions made by the framework scheduler can only be taken once a minute. The reason for such a long period is the time required by the implementation of the feature extraction process in our single-core wearable test computer: it adds a constant time of 40 seconds to process any data block, including empty data blocks. However, this should not be a limitation in a real environment: the feature extraction process can be heavily optimized (those additional seconds are not spent when running feature extraction in other platforms), and nowadays is common to find multicore architectures also in mobile sensors. As we have defined such a long time for a variable size block, we set the threshold value to 0 (revisit Fig. 5), so scheduling decisions are taken every single block.

5.4. Simulation results & discussion

Fig. 9 depicts a representative example of the simulation results. The three upper diagrams show the 'load average' of the wearable, mobile and cloud computer respectively. The bottom diagram represents the 'data transfer rate' along the network. The average is taken over the last minute. As can be seen, the devices are usually running user applications such as music, video reproduction, and incoming calls (colored areas in the two upper diagrams corresponding to c_1 and c_2 computers). At the beginning of the test, the tasks are launched on the wearable device (c_1) to be processed. Other scenarios can be designed with similar results. When the framework is disabled, the wearable device cannot take over the tasks of the DIDS application. Over the minutes 17 and 27 the computer load overcomes the capacity of the device. This situation may result in mp3 reproduction errors or in malfunction of the tasks. When the framework is enabled, according to R and S matrices, the tasks are sent to the other computers. The tasks 1 and 2 are initially sent to be processed on the computer c_2 and the task 3 is sent to the cloud computer node (c_3). However, over the minute 21 an mp4 reproduction starts in computer c_2. After that, the system load raises until reaching 1 system load average.

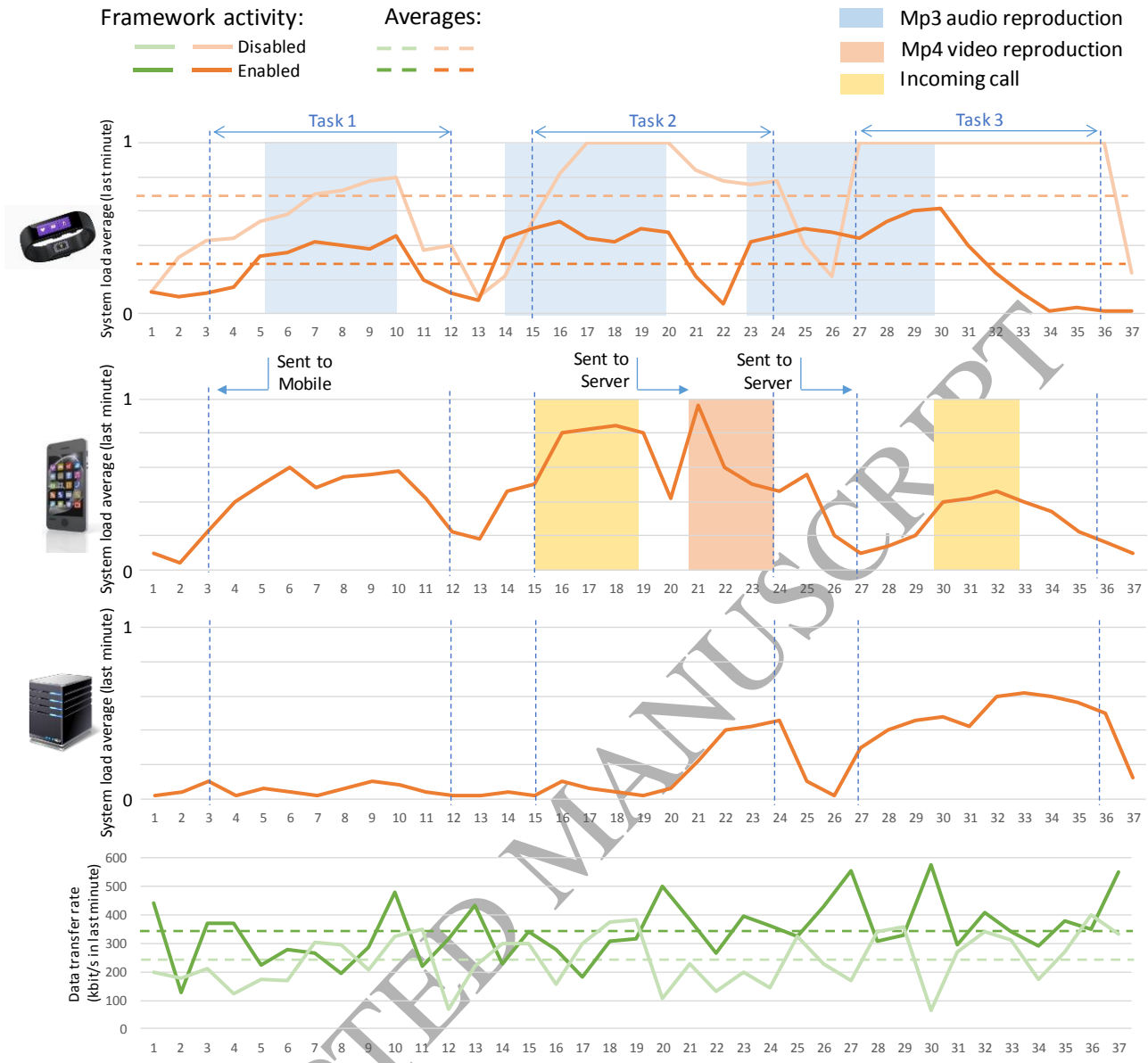

Fig. 9. System load average and data transfer rate

At this point, the framework scheduler instructs the switch process to send packet headers to the server computer, where the corresponding server task will extract the features. Consequently, the feature extraction task is not running anymore on this device, and the system load average goes decreasing. This decrease in load allows the video stream to be properly played.

We have conducted the same experiment several times, and despite the random nature of the user behaviour simulation, the essential result remains the same: successful switching of the packet header flows, freeing processor time of the devices c_1 and c_2 when required by user tasks. The simulation results in Fig. 9 show a reduction in the load average due to the transfer of part of the processing to the server node. The computing cost produced by the switch and the controller processes are assumable, while the processing along the network devices can be shared if necessary. As long as we properly extend L with relevant components, this model can be also applied to offloading strategies in mobile computers supplied with batteries.

Sending packet headers to an external server implies bandwidth use. This is reflected in the lower diagram of Fig. 9. Obviously the traffic flow is increased by a fraction of the analysed flow (the traffic between DIDS tasks is not captured for analysis). In our test environment, this increase in the data transfer rate can be perfectly assumed, since our wireless network support a bandwidth of several megabytes per second. However, in a real environment, complex scheduling policies must be put in place, taking into account other factors as variable free bandwidth in wireless networks with shared transmission medium, energy consumption and user preferences, among others. Those complex strategies fit into the L definition and the $\oplus$ operator for prediction.

The experiments show that the provided formal model can be adapted to a great set of applications to be distributed over a number of heterogeneous nodes found in

CPS. By properly defining L, the scheduling process will take decisions based on relevant performance metrics; by properly defining R, the suitability of the devices for the different tasks is modelled, and this information can be used for optimal or suboptimal decision computation. By establishing a procedure $\oplus$, the result of simple or sophisticated prediction models can be incorporated to the design, in order to improve the goodness of the scheduling results. Finally, by suitably configuring S, the scheduler can take into consideration the preferences and particular operating conditions of the available devices.

6. Conclusions

In this work, we have designed a novel resource definition framework and a method that allows convenient distribution of the application tasks on CPS environments. Thse contexts are mainly characterized by the diversity and dynamic availability of the computing elements involved. The proposal takes into account feasibility and suitability aspects such as the configuration preferences, variable availability of computing resources in CPS devices, personal and enterprise computers, and additional capabilities coming from cloud services. In addition, the framework supports solutions based on multiple technologies and approaches, combining well-known effective techniques with the latest research results. As a secondary effect, the framework also provides failure tolerance by supporting multiple instances of the different tasks required for the overall distributed application.

This approach offers an application independent solution for integrating computing resources in a flexible way and combining the scheduling possibilities for sharing the processing cost among the CPS nodes: cloud resources are used only when necessary, minimizing utility computing costs and security problems but preserving local resources when those are required for critical processes.

The experiments conducted provide a proof-of-concept prototype of the model and show the feasibility of the method for distributing the application tasks in a CPS environment.

For future research work, further effort must be invested in building a proper predictive model of the available resources, addressed to provide valuable information for increasing the effectiveness of the scheduler component. This a very important challenge for avoiding overload scenarios and properly leveraging the deployed infrastructure.

Another future work line has to do with the network performance analysis for obtaining accurate response times and available bandwidth for sharing the dataflows and tasks in CPS environments.

In addition, for the proposed further research, the experimental design should be completed, integrating other relevant factors, as main memory usage, storage requirements and energy consumption.

References

[1] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of Things (IoT): A vision, architectural elements, and future directions, Future Generation Computer Systems 29 (7) (2013) 1645-1660. doi: 10.1016/j.future.2013.01.010.

[2] D. Gil, A. Ferrández, H. Mora, J. Peral, Internet of Things: A Review of Surveys Based on Context Aware Intelligent Services. Sensors 16 (2016), 1069. doi: 10.3390/s16071069.

[3] J. Wan, M. Chen, F. Xia, D. Li, K. Zhou, From machine-to-machine communications towards cyber-physical systems, Computer Science and Information Systems 10 (3) (2013) 1105-1128. doi: 10.2298/CSIS120326018W.

[4] Y. Chen, G.M. Lee, L. Shu, N. Crespi, Industrial Internet of Things-Based Collaborative Sensing Intelligence: Framework and Research Challenges. Sensors 16 (2016), 215. doi: 10.3390/s16020215.

[5] S. T. Kouyoumdjieva, G. Karlsson, Energy-aware opportunistic mobile data offloading under full and limited cooperation, Computer Communications, 84 (2016), 84-95. doi: 10.1016/j.comcom.2016.02.008.

[6] K.-J. Park, J. Kim, H. Lim, Y. Eun, Robust path diversity for network quality of service in cyber-physical systems, IEEE Transactions on Industrial Informatics 10 (4) (2014) 2204-2215. doi: 10.1109/TII.2014.2351753.

[7] N. Nasser, R. Miller, A. Esmailpour, A.-E. M. Taha, T. Bejaoui, Optimized bandwidth allocation in broadband wireless access networks, Wireless Communications & Mobile Computing 15 (17) (2015) 2111-2124. doi: 10.1002/wcm.2479.

[8] G. D. Abowd, E. D. Mynatt, Charting past, present, and future research in ubiquitous computing, ACM Transactions on Computer-Human Interaction 7 (1) (2000) 29-58. doi: 10.1145/344949.344988.

[9] C. Keller, T. Schlegel, Model based and service oriented interaction for ubiquitous environments ACM International Joint Conference on Pervasive and Ubiquitous Computing, 2016, 429-434, doi: 10.1145/2968219.2971358.

[10] F. Xiao, G. Ge, L. Sun, R. Wang, An energy-efficient data gathering method based on compressive sensing for pervasive sensor networks, Pervasive and Mobile Computing, (2017) doi: j.pmcj.2017.02.005.

[11] M. R. Ogiela, L. Barolli, New paradigms for information and services management in grid and pervasive computing, Future Generation Computer Systems, 67 (2017) 227-229. doi: 10.1016/j.future.2016.10.011.

[12] M. Satyanarayanan, The Emergence of Edge Computing, Computer, 50 (1) (2017) 30 – 39, doi: 10.1109/mc.2017.9.

[13] G. D. Abowd, Beyond Weiser: From Ubiquitous to Collective Computing, Computer, 49 (1) (2016) 17 – 23. doi: 10.1109/mc.2016.22.

[14] J. Lanza et al., Managing Large Amounts of Data Generated by a Smart City Internet of Things Deployment, International Journal on Semantic Web and Information Systems, 12(4), (2016) 1-21. doi: 10.4018/ijswis.2016100102.

[15] T. A. Cherfia, F. Belala, K. Barkaoui, A bigraph-based framework for specification and analysis of context-aware systems, International Journal of Critical Computer-Based Systems, 6 (4) (2017) 322-342. doi: 10.1504/ijccbs.2016.081808.

[16] S. Abdollahzadeh, N. J. Navimipour, Deployment strategies in the wireless sensor network: A comprehensive review, Computer Communications, 91–92, (2016), 1-16. doi: 10.1016/j.comcom.2016.06.003.

[17] F. Hu, Y. Lu, A. V. Vasilakos, Q. Hao, R. Ma, Y. Patil, T. Zhang, J. Lu, X. Li, N. N. Xiong, Robust cyber-physical systems: Concept, models, and implementation, Future Generation Computer Systems, 56 (2016), 449–475. doi: 10.1016/j.future.2015.06.006.

[18] H. Tang, L. Li, N. Xiao, Smooth Sensor Motion Planning for Robotic Cyber Physical Social Sensing (CPSS). Sensors 17(2) (2017), 393. doi: 10.3390/s17020393.

[19] C. Chen, J. Yan, N. Lu, Y. Wang, X. Yang, X. Guan, Ubiquitous monitoring for industrial cyber-physical systems over relay-assisted wireless sensor networks, IEEE Transactions on Emerging Topics in Computing 3 (3) (2015) 352-362. doi: 10.1109/TETC.2014.2386615.

[20] D. B. Rawat, S. Reddy, N. Sharma, B. B. Bista, S. Shetty, Cloud-assisted GPS-driven dynamic spectrum access in cognitive radio vehicular networks for transportation cyber physical systems, IEEE Wireless Communications and Networking Conference, 2015, 1942-1947. doi: 10.1109/WCNC.2015.7127765.

[21] L. Y. Mano et al., Exploiting IoT technologies for enhancing Health Smart Homes through patient identification and emotion recognition, Computer Communications, 89–90, (2016), 178-190. doi: 10.1016/j.comcom.2016.03.010.

[22] H. Mora Mora, V. Gilart Iglesias, D. Gil, A. Sirvent Llamas, A computational architecture based on RFID sensors for traceability in smart cities, Sensors 15 (6) (2015) 13591-13626. doi: 10.3390/s150613591.

[23] L. R. Adrian, L. Ribickis, Intelligent power management device for street lighting control incorporating long range static and non-static hybrid infrared detection system, in: 16th European Conference on Power Electronics and Applications (EPE-ECCE Europe), 2014, 1-5. doi: 10.1109/EPE.2014.6911008.

[24] V. Gilart Iglesias, H. Mora, R. Prez del Hoyo, C. García Mayor, A computational method based on radio frequency technologies for the analysis of accessibility of disabled people in sustainable cities, Sustainability 7 (11) (2015) 14935-14963. doi: 10.3390/su71114935.

[25] P. J. Mosterman, J. Zander, Cyber-physical systems challenges: a needs analysis for collaborating embedded software systems, Software & Systems Modeling, 15 (1) (2016), 5-16. doi: 10.1007/s10270-015-0469-x.

[26] E.A. Lee, The Past, Present and Future of Cyber-Physical Systems: A Focus on Models. Sensors, 15(3) (2015), 4837-4869. doi: 10.3390/s150304837.

[27] B. Shen, X. Zhou, M. Kim, Mixed scheduling with heterogeneous delay constraints in cyber-physical systems, Future Generation Computer Systems, 61 (2016), 108–117. doi: 10.1016/j.future.2015.10.021.

[28] D. Dereniowski, W. Kubiak, Shared multi-processor scheduling, European Journal of Operational Research, 261 (2017) 503–514. doi: 10.1016/j.ejor.2017.03.002.

[29] B. Andersson, G. Raravi, Real-time scheduling with resource sharing on heterogeneous multiprocessors, Real-Time Systems, 50 (2), (2014) 270–314. doi: 10.1007/s11241-013-9195-z.

[30] Z. Zhang, J. M. Chang, A Cool Scheduler for Multi-Core Systems Exploiting Program Phases, IEEE Transactions on Computers, 63 (5) (2014), 1061 – 1073. doi: 10.1109/tc.2012.283.

[31] T. M. Birhanu, Z. Li, H. Sekiya, N. Komuro, Y.-J. Choi, Efficient Thread Mapping for Heterogeneous Multicore IoT Systems, Mobile Information Systems, 2017 (2017), ID 3021565, doi: 10.1155/2017/3021565.

[32] N. Kumar, D.P. Vidyarthi, An Energy Aware Cost Effective Scheduling Framework for Heterogeneous Cluster System, Future Generation Computer Systems, (71) (2017), 73-88. doi: 10.1016/j.future.2017.01.015.

[33] H. Kanemitsu, M. Hanada, H. Nakazato, Clustering-Based Task Scheduling in a Large Number of Heterogeneous Processors, IEEE Transactions on Parallel and Distributed Systems, (27) (11) (2016) 3144 – 3157. doi: 10.1109/tpds.2016.2526682.

[34] M. Mäsker, L. Nagel, A. Brinkmann, F. Lotfifar, M. Johnson, Smart Grid-aware scheduling in data centres, Computer Communications 96 (2016) 73–85. doi: 10.1016/j.comcom.2016.04.021.

[35] Z. Qian, H. Yu, A TAOPN Approach to Modeling and Scheduling Cyber-Physical Systems, International Conference on Information Science and Applications (ICISA), 2013. doi: 10.1109/icisa.2013.6579475.

[36] M. Haferkamp, B. Sliwa, C. Ide, C. Wietfeld, Payload-Size and Deadline-Aware scheduling for time-critical Cyber Physical Systems, Wireless Days, (2017) doi: 10.1109/wd.2017.7918106.

[37] R. Schneider, D. Goswami, A. Masrur, M. Becker, S. Chakraborty, Multi-layered scheduling of mixed-criticality cyber-physical systems, Journal of Systems Architecture 59 (2013) 1215-1230. doi: 10.1016/j.sysarc.2013.09.003.

[38] G. Xie, G. Zeng, Z. Li, R. Li, K. Li, Adaptive Dynamic Scheduling on Multi-functional Mixed-Criticality Automotive Cyber-Physical Systems, IEEE Transactions on Vehicular Technology, 99 (2017). doi: 10.1109/tvt.2017.2674302.

[39] G. Xie, G. Zeng, L Liu, R. Li, K. Li, High performance real-time scheduling of multiple mixed-criticality functions in heterogeneous distributed embedded systems, Journal of Systems Architecture, 70, (2016) 3–14. doi: 10.1016/j.sysarc.2016.04.008.

[40] C. Qu, W. Chen, J. Bin Song, H. Li, Distributed data traffic scheduling with awareness of dynamics state in cyber physical systems with application in smart grid, IEEE Transactions on Smart Grid 6 (6) (2015), 2895-2905. doi: 10.1109/tsg.2015.2399247.

[41] L. Songxi, W. Qinghua, W. Han, F. Yuanliang, P. Hui, Z. Gonglin, P. Haibo, Traffic scheduling with sustainable Cyber Physical Systems applying in smart grid, International Green and Sustainable Computing Conference, 2016, doi: 10.1109/igcc.2016.7892587.

[42] S. Park, J-H. Kim, G. Fox, Effective real-time scheduling algorithm for cyber physical systems society, Future Generation Computer Systems, 32 (2014) 253–259. doi: 10.1016/j.future.2013.10.003.

[43] H. Li, Z. Han, A. D. Dimitrovski, Z. Zhang, Data Traffic Scheduling for Cyber Physical Systems With Application in Voltage Control of Distributed Generations: A Hybrid System Framework, IEEE Systems Journal, 8 (2) (2014) 542 – 552. doi: 10.1109/jsyst.2013.2260915.

[44] M. Li, P. Li, Crowdsourcing in Cyber-Physical Systems: Stochastic Optimization With Strong Stability, IEEE Transactions on Emerging Topics in Computing, 1(2) (2013), 218 – 231. doi: 10.1109/tetc.2013.2273358.

[45] J. Lee, K. G. Shin, Development and use of a new task model for cyber-physical systems: A real-time scheduling perspective, The Journal of Systems and Software 126 (2017) 45–56. doi: 10.1016/j.jss.2017.01.004.

[46] H. Gong, R. Li, J. An, W. Chen, K. Li, Scheduling Algorithms of Flat Semi-Dormant Multi-Controllers for a Cyber-Physical System, IEEE Transactions on Industrial Informatics, (99) (2017). doi: 10.1109/tii.2017.2690939.

[47] C. Liu, L. Zhang, D. Zhang, Task Scheduling in Cyber-Physical Systems, Intl. Conf. on Ubiquitous Intelligence and Computing, 2014. doi: 10.1109/uic-atc-scalcom.2014.97.

[48] P. Nawrocki, W. Reszelewski, Resource usage optimization in Mobile Cloud Computing, Computer Communications, 99 (2017) 1-12. doi: 10.1016/j.comcom.2016.12.009.

[49] A. Saarinen et al, SmartDiet: offloading popular apps to save energy, ACM Sigcomm Computer Communication Review, 42 (4), (2012) 297-298. doi: 10.1145/2342356.2342418.

[50] K. Akherfi, M. Gerndt, H. Harroud, Mobile cloud computing for computation offloading: Issues and challenges, Applied Computing and Informatics, (2016), doi: 10.1016/j.aci.2016.11.002.

[51] Khan M. A., A survey of computation offloading strategies for performance improvement of applications running on mobile devices, Journal of Network and Computer Applications, 56 (2015) 28–40. doi: 10.1016/j.jnca.2015.05.018.

[52] D. T. Nguyen, W. Li, P. O. Ogunbona, Human detection from images and videos: A survey, Pattern Recognition, 51 (2016), 148–175, 2016. doi: 10.1016/j.patcog.2015.08.027.

[53] W. Wolf, B. Ozer, T. Lv, Smart cameras as embedded systems, Computer, 35 (2002), 48 – 53. doi: 10.1109/MC.2002.1033027.

[54] M.A. Alsmirat, I. Obaidat, Y. Jararweh, M. Al-Saleh, A security framework for cloud-based video surveillance system, Multimed Tools Appl (2017). doi:10.1007/s11042-017-4488-1.

[55] K. Kang, M.-Y. Nam, L. Sha, Model-based analysis of wireless system architectures for real-time applications, IEEE Transactions on Mobile Computing 12 (2) (2013) 219-232. doi: 10.1109/TMC.2011.260.

[56] J. Liu, E. Ahmed, M. Shiraz, A. Gani, R. Buyya, A. Qureshi, Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions, Journal of Network and Computer Applications, 48 (2015) 99–117. doi: 10.1016/j.jnca.2014.09.009.

[57] L. A. J. Marzulo, T. A. O. Alves, F. M. G. Franca, V. S. Costa, Couillard: Parallel programming via coarse-grained data-flow compilation, Parallel Computing 40 (10) (2014) 661-680. doi: 10.1016/j.parco.2014.10.002.

[58] Y.-S. Chen, H. C. Liao, T.-H. Tsai, Online real-time task scheduling in heterogeneous multicore system-on-a-chip, IEEE Transactions

on Parallel and Distributed Systems, 24 (1), (2013) 118–130. doi: 10.1109/tpds.2012.114

[59] H. Mora, D. Gil, J. F. Colom, M. T. Signes, Flexible framework for real-time embedded systems based on mobile cloud computing paradigm, Mobile Information Systems 2015 (2015) id. 652462. doi: 10.1155/2015/652462.

[60] JF Colom, H Mora, D Gil, MT Signes-Pont, Collaborative building of behavioural models based on internet of things, Computers & Electrical Engineering, 58 (2017). doi: 10.1016/j.compeleceng.2016.08.019.

[61] J. Guo, I.-R. Chen, J.J.P. Tsai, A survey of trust computation models for service management in internet of things systems, Computer Communications, 97, (2017), 1-14. doi: 10.1016/j.comcom.2016.10.012.

[62] C. Zimmer, B. Bhat, F. Mueller, S. Mohan, Time-based intrusion detection in cyber-physical systems, in: 1st ACM/IEEE International Conference on Cyber-Physical Systems, 2010, pp. 109-118. doi: 10.1145/1795194.1795210.

[63] W. El-Hajj, M. Al-Tamimi, F. Aloul, Real traffic logs creation for testing intrusion detection systems, Wireless Communications & Mobile Computing 15 (14) (2015) 1851-1864. doi: 10.1002/wcm.2471.

[64] R. Singh, H. Kumar, R. K. Singla, An intrusion detection system using network traffic profiling and online sequential extreme learning machine, Expert Systems with Applications 42 (22) (2015) 8609-8624. doi: 10.1016/j.eswa.2015.07.015.

[65] M. Riecker, S. Biedermann, R. El Bansarkhani, M. Hollick, Lightweight energy consumption-based intrusion detection system for wireless sensor networks, International Journal of Information Security 14 (2) (2015) 155-167. doi: 10.1007/s10207-014-0241-1.

[66] R. Mitchell, I.-R. Chen, A survey of intrusion detection techniques for cyber-physical systems, ACM Computing Surveys 46 (4) (2014). doi: 10.1145/2542049.

[67] M. A. Faisal, Z. Aung, J. R. Williams, A. Sanchez, Data-stream-based intrusion detection system for advanced metering infrastructure in smart grid: A feasibility study, IEEE Systems Journal 9 (1) (2015) 31-44. doi: 10.1109/JSYST.2013.2294120.

[68] V-H. Tran, Q. De Coninck, B. Hesmans, R. Sadre, O. Bonaventure, Observing real Multipath TCP traffic, Computer Communications, 94, (2016) 114-122. doi: 10.1016/j.comcom.2016.01.014.

[69] D. S. Punithavathani, K. Sujatha, J. M. Jain, Surveillance of anomaly and misuse in critical networks to counter insider threats using computational intelligence, Cluster Computing-the Journal of Networks Software Tools and Applications 18 (1) (2015) 435-451. doi: 10.1007/s10586-014-0403-y.