Cluster Computing (2019) 22:661–677
https://doi.org/10.1007/s10586-018-2874-8

SD-CPS: software-defined cyber-physical systems. Taming the challenges of CPS with workflows at the edge

Pradeeban Kathiravelu[1,2,3] · **Peter Van Roy**[3] · **Luís Veiga**[2]

Received: 29 January 2018 / Revised: 25 June 2018 / Accepted: 23 November 2018 / Published online: 28 November 2018

Abstract
A cyber-physical system (CPS) is a smart mechanical environment, developed by an amalgamation of computation, networking, and physical dimensions. Each CPS consists of a network of devices, often limited in computing, storage, or bandwidth resources. Moreover, the frequent small-scale communications between the various counterparts of CPS require data and computation of CPS to be deployed close to each other, with the ability to support micro-executions. Due to these operational requirements, CPS faces several inherent challenges, uncommon to a traditional computational environment. In this paper, we describe software-defined cyber-physical systems (*SD-CPS*), a CPS framework built by extending and adapting the design principles of software-defined networking (SDN) into CPS. We realize the support for CPS operation as a workflow of microservices, possibly in continuous or cyclic execution. *SD-CPS* coordinates each CPS execution step, performed by a microservice, through an extended SDN controller architecture. By creating, placing, deploying, migrating, and managing the computation processes of CPS as service workflows at the edge, *SD-CPS* orchestrates the entire lifecycle of the CPS effectively and efficiently. *SD-CPS* thus addresses the general challenges of CPS, concerning modeling, development, performance, management, communication and coordination, scalability, and fault-tolerance, through its software-defined approach. Our evaluations highlight the efficiency of the *SD-CPS* framework and the scalability of its SDN controller to manage the complex CPS environments.

Keywords Cyber-physical system (CPS) · Software-defined networking (SDN) · Message-oriented middleware (MOM) · Software-defined systems (SDS)

1 Introduction

A cyber-physical system (CPS) comprises numerous sensors or sensor-based devices that collect different types of data from various access points. It is composed of autonomous systems with frequent communication among them [29]. Smart homes [42], Smart grids [24], smart cities [20, 44], mobile ad-hoc networks (MANETs) [37], and vehicular ad-hoc networks (VANETs) [64] are a few systems that are currently built as CPS. CPS often have execution nodes in an edge network. They utilize the computing resources in the edge nodes as surrogates for the workload execution from their resource-constrained devices [46]. Such use of CPS eliminates the resource limitations in CPS execution. Thus, adoption and capabilities of CPS continues to grow with the use of edge.

1.1 Common challenges faced by CPS

CPS faces several challenges in design and performance, due to its scale and variety in its devices [28]:

(Ch_1) Unpredictability of the execution environments [27]

✉ Pradeeban Kathiravelu
pradeeban.kathiravelu@tecnico.ulisboa.pt

Peter Van Roy
peter.vanroy@uclouvain.be

Luís Veiga
luis.veiga@inesc-id.pt

[1] Emory University School of Medicine, Atlanta, USA

[2] INESC-ID Lisboa/Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal

[3] Université catholique de Louvain, Louvain-la-Neuve, Belgium

(Ch_2) Orchestrating the communication and coordination within the CPS [47].
(Ch_3) Security, distributed fault-tolerance, and recovery upon system and network failures [9].
(Ch_4) Decision making in the large-scale geo-distributed execution environments [53].
(Ch_5) Modeling and designing the complex CPS environments [14].
(Ch_6) Management and orchestration of the intelligent agents [8].

Deploying the workloads at the edge can offer a high-performant execution for the latency-sensitive CPS applications [45]. However, the current cloud and edge environments often do not favor a seamless deployment and smooth frequent migrations of the workloads between the execution environments, which are essential for CPS. This limits the number of edge nodes that can participate in the CPS execution. Furthermore, the diversity of the edge nodes as well as CPS has created a management challenge in the execution of CPS workloads.

1.2 Network softwarization and CPS

Software-defined networking (SDN) offers programmability and management capabilities, to data networks, typically within cloud networks, but also extended to wide area network scenarios such as content distribution networks (CDNs) [63]. Initiatives such as network service orchestration (NSO) [7] and lifecycle service orchestration (LSO) [41] aim at addressing the shortcoming in resource orchestration for workflows in the network environments. They unify the network services through standardization, to improve the interoperability of services. The state-of-the-art indicates the potential to leverage SDN and the network softwarization efforts to manage the CPS environments more effectively. SDN has been proposed to improve the resilience of multi-networks in CPS [49], secure the CPS networks through SDN-assisted emulations [4], and enhance the resilience [17] of CPS. These early research efforts highlight the potential of SDN and how its global network awareness and controllability can be extended to model and manage CPS. However, despite these promising early results on exploiting SDN to manage CPS efficiently, currently there is no complete framework for leveraging network softwarization to mitigate the identified challenges of CPS.

A framework to support workload execution at the edge with capabilities to support frequent communication and workload migration is essential for the successful execution of CPS workloads at the edge. We propose executing the CPS workloads as web service workflows can offer unified deployment and execution, as web services are developed following standards. However, for the successful execution of the CPS workflows at the edge, we need to build an orchestration framework to manage the services and resources in the wide area network, as well as the CPS environment.

1.3 Motivation

Given the above premises, we aim at addressing the following research questions in this paper:

(RQ_1) Can SDN or a more encompassing approach inspired by SDN help mitigate the identified challenges of CPS that hinder its wide-scale adoption?
(RQ_2) Can we seamlessly scale such an approach beyond data centers, to a wide area network, for modeling and executing CPS?
(RQ_3) Can we leverage the edge resources for a distributed execution of CPS workloads through a unified control, without additional overheads?
(RQ_4) Can we generalize the CPS execution as service workflows at the edge, to support interoperable execution across wide area networks such as edge and cloud-assisted overlay networks?

1.4 Contributions

The goal of this paper is to answer the identified research questions to mitigate the major challenges faced by CPS. We presented an early design of our software-defined approach for CPS in our previous work [26]. The main contributions of this paper are:

1. A generic framework for building and executing CPS through service workflows at the edge, to support interoperable execution of the workloads in multi-domain networks (RQ_3, RQ_4 and Ch_2). Presented in Sect. 3.1.
2. Software-defined cyber-physical systems (*SD-CPS*), a scalable and distributed software-defined system (SDS), to coordinate the heterogeneous CPS devices by extending SDN with message-oriented middleware (MOM) protocols [11] for wide area networks (RQ_1, RQ_2, and Ch_4). Presented in Sect. 3.2.
3. Incorporating user policies and resource requirements for efficient CPS resource allocation and migration. Thus, offering efficient control of the network resources for the user applications, despite the unpredictability of the CPS networks that are shared across several users (Ch_1). Presented in Sect. 3.3
4. A reusable execution model for the CPS workflows to modeling, incremental development, and seamless

execution in a sandbox and production environments. Thus, managing the execution of CPS in the physical and cyberspaces effectively with minimal duplicate effort (Ch_5 and Ch_6). Presented in Sect. 3.2.

5. Resilient and agile CPS execution by offloading the CPS workload as service composition workflows at the edge, by exploiting a logically centralized control of the edge resources (Ch_3). Presented in Sect. 3.4.

The complexity of CPS increases due to both volume and variety of its components. The edge offers a compromise on bandwidth usage between on-device or on-premise computation and computation in the cloud. Thus, it has been proposed for specific smart environments such as a smart campus [35] or connected cars [38]. *SD-CPS* mitigates the complexity of the computation and limitation of the resources by decoupling and decomposing the execution of CPS into workflows of microservices and offloading the workflows to edge environments. However, discovering the resource availability and deploying the workflow as service invocations at the nodes need to be performed effectively to reap the benefits of the edge. We design a controller deployment as the core of *SD-CPS* to perform the overall coordination and manage the "cyber" of the CPS and orchestrate the CPS elements. Along with its software-defined approach, *SD-CPS* executes CPS in a programmable and predictable manner, inherently addressing many of the operational challenges of CPS. We implemented and evaluated a prototype to assess the performance of the *SD-CPS* approach. The quantitative evaluations highlight the efficiency, success rate, and scalability of CPS execution modeling and resource allocation through edge workflows.

1.5 Paper organization

SD-CPS is built as a generic software-defined approach applicable for the CPS environments. However, we explain the motivation behind *SD-CPS* using MANETs and VANETs as a specific use case scenario, in Sect. 2. Section 3 presents the *SD-CPS* solution architecture. Section 4 presents the *SD-CPS* implementation. We further narrow down our focus to VANETs in the prototype evaluations (presented in Sect. 5) for a more representative outcomes for the specific network scenario. Section 6 further elaborates how *SD-CPS* builds upon the state-of-the-art, and qualitatively assesses it against the other research work addressing the challenges faced by CPS. Finally, Sect. 7 concludes the paper with the current state of the research and future research directions.

2 MANETs and VANETs: a case for *SD-CPS*

Complex computations at the mechanical or physical devices of CPS often require resources beyond what is available physically on the device, such as a smart vehicle or a mobile terminal. Hence, workloads heavy in computing or memory are delegated to the *cloud-based cyberspace*, instead of executing them (often only as firmware) in the smart terminals of the CPS. In this section, we will look into a few common characteristics of CPS that motivate the case for an orchestration framework in a wide area network, by discussing MANETs and VANETs as potential cases for CPS.

MANETs are comprised of mobile devices, that can function independently as autonomous systems as well as sensors, while also communicating among themselves. A typical example of a MANET device, a smart mobile phone, can sense its environment, including (i) background noise level through its audio sensor (microphone), (ii) light/vision through its camera, and (iii) motion through its motion/shock detectors. In addition to the sensing capabilities, a smart mobile device also has computing and memory resources (that can be leveraged to detect, analyze, and monitor user activity such as standing, walking, running, and cycling), though in a limited capacity compared to the traditional computation devices.

VANETs include autonomous automotive systems such as networks composed of self-driving vehicles [3] or smart vehicles [57]. A self-driving vehicle depends heavily on the contextual information sensed by itself as well as those shared by other smart vehicles, due to the absence of an experienced human driver. Smart vehicles collaborate and coordinate with one another, to share information such as current traffic, and dynamically decide their traveling path by analyzing the accumulated dynamic data, in real-time. While VANETs can be considered a subset of MANETs, by its nature regarding its velocity, VANETs have a higher degree of dynamism than the other MANETs such as mobile terminals. VANETs are expanding their scope beyond the road traffic, also to consider the air traffic monitoring and aircraft control [51]. These latest advancements have widened the research challenges associated with the CPS.

An automobile such as a connected car [19] or an aircraft can pass through various control stations, while on the move. Therefore its proximity to sensors (including satellites for the air traffic control) or computing nodes (for connected cars) in the VANET differs with time. Thus, the current updated location of a smart device or terminal needs to be considered when some data is routed towards it. Research identifies networking and message exchanging problems caused by the dynamism in VANETs [64].

Remarkably, VANET CPS needs to route various messages from adjacent sensors or other vehicles towards a moving vehicle considering its current geographic location and ensure that an ongoing flow is served following the previous route to avoid data loss or inconsistency.

Metadata in the controller should be dynamically updated to reflect the changes in the cost of each path in a dynamic CPS network environment such as MANETs, VANETs, and robotics systems. SDN has been extended for VANETs to coordinate the data scheduling efficiently in a centralized manner [34]. Previous research has proposed using cloud-assisted execution for CPS applications such as context-aware vehicles [62]. The availability of resources in their proximity to the execution has led to considering smart devices to utilize the edge instead of cloud resources [59]. While VANET communications are typically mouse flows (data flows caused by frequent small-scale interactions), flows such as backup of vehicular data to the edge cloud are heavy in bandwidth requirements and data volume, and are elephant flows. Such elephant flows need to be rerouted with minimal overhead if the previously chosen path is not the best-fit anymore, due to the mobility of the vehicle. These are complex traffic engineering problems that are addressed by advances in SDN research [1]. However, the scale and variety of CPS require further research beyond the intra-data center traffic engineering.

3 Solution architecture

SD-CPS aims at mitigating the design and operational challenges of CPS through its software-defined approach. In this Section, we will look into the solution architecture of *SD-CPS*.

3.1 CPS execution as service workflows

SD-CPS consists of multiple *tenants*, the users who configure and deploy their application workload as workflows, abiding by their respective policies. Each tenant defines service level objectives (SLOs) for her workflows, enabling differentiated execution based on tenant-defined policies. The multi-tenant execution of *SD-CPS* virtually associates each workflow with its tenant at the edge nodes despite sharing the execution space with other tenants.

SD-CPS designs a CPS execution as a workflow composed of web services. These services would typically be running iteratively or periodically, thus continuously carrying out monitoring, assessment, decision, and actuation activities. Consider a traffic analysis workflow w belonging to a VANET CPS. It is composed of various services, including (i) data sensing at each of the vehicular sensor devices, (ii) data matching from various adjacent vehicles for data correction to minimize noises, (iii) data integration at an edge node, (iv) analytics based on current and past data, (v) traffic prediction on each route, and (vi) receiving or sending personalized information from the edge cloud back to the vehicle. Equation 1 illustrates a generalized form of such a CPS workflow, composed of various services.

$$\forall x \in \mathbb{Z}^+; w = s^1 \circ s^2 \circ \cdots \circ s^x. \quad (1)$$

The CPS data flow goes through various intermediary nodes, from the sensors that collect data as input devices, to the actuators that perform actions based on the contextual data. These links between the nodes form the execution paths of *SD-CPS* workflows. Decomposition of the workload into smaller deployable services (e.g., containerized microservices) at the edge helps increase the perceived path redundancy of the workflow. *SD-CPS* exploits an extended controller deployment to manage the metadata that governs the alternative execution paths for each workflow of a CPS execution. The controller is physically decentralized, yet logically centralized. Thus, it controls the service workflows globally, yet in a decentralized manner. The workflows can have different replication levels for each service, based on the user policies indicating the importance of each of them. We aim for increased resilience, load balancing, and congestion control among the underlying network paths with the path redundancy and the global awareness of the controller on their existence.

Figure 1 models a wireframe of the underlying system of CPS with data flow between two smart devices, with multiple potential paths. The origin and destination nodes are respectively the start and the end nodes of a communication initiated by a distributed computation. In a data center network, these nodes are hosts or servers, while the intermediate nodes are traditionally switches that connect the underlying network. However, due to the heterogeneous nature of CPS, origin or destination can be smart mobile devices/terminals or virtual execution spaces in the controller, while intermediate or destination nodes can be surrogate nodes such as edge servers or switches in a data center network. In a mobile CPS or a MANET, these end nodes can be dynamic with an ever-changing geographic location, such as automobile devices, moving robots, or smartphones on the move.

With the dynamic traffic of network flows, a few service or network nodes and links may become congested. Moreover, some nodes may be prone to failures. The *SD-CPS* controller identifies the congested, malfunctioning, or malicious nodes and links (that are highlighted and differentiated as unhealthy in Fig. 1 for the ease of reference) through its controller, by monitoring the responsiveness of the nodes. Thus the controller alters execution paths

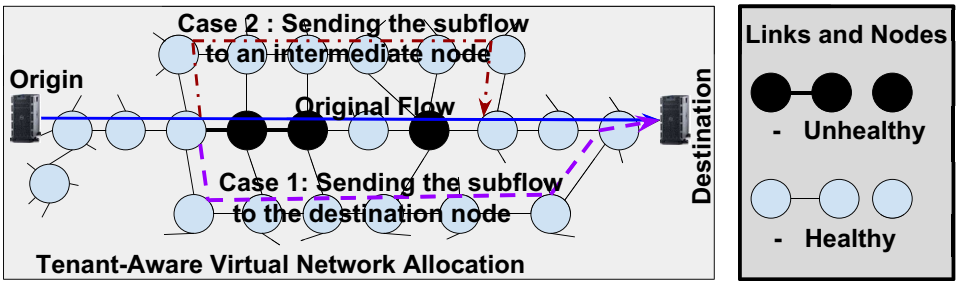


Fig. 1 Executions as service workflows with alternative execution paths

towards a healthy alternative dynamically. It creates subflows by diverting or cloning parts of the flows and sends them towards a node known as the clone destination. For mission-critical CPS such as ICU medical monitoring systems, the critical services need to execute correctly and timely. In such workflows, when an intermediary is identified as failed or slow, the controller enforces partial redundancy in the data flows to ensure correctness and end-to-end delivery. By the extension of this, the controller aims for a load balanced execution environment at the edge among the microservice deployments.

SD-CPS defines a *clone destination* where it reconstructs the flow from the subflows it created. The clone destination can be as same as the original destination of the flow, or an intermediary node in the flow path. In case 1 identified in Fig. 1, the clone destination is the same as the original destination of the CPS workflow. While this scenario may lead to a higher level of redundancy due to the presence of duplicate or replicated subflows, it avoids the need to manipulate the network flows at the intermediary nodes. However, case 2 has a clone destination that differs from the original. Here the cloned subflow is sent towards an intermediate node (on the original path connecting the origin and destination), in an alternative path. The flow is recomposed afterward. The case 2 approach minimizes unnecessary redundancy when it is possible to recompose the flow at the clone destination or an intermediate node. When such a recomposition of flows (i.e., making flows reconverge) is impossible at an intermediate node due to the technical difficulties, or due to the nature of the congestion or network failure itself, the flow is eventually recomposed when it reaches the destination node as in the case 1.

3.2 *SD-CPS* coordination

The controller is the element with the highest processing power in the *SD-CPS* ecosystem. It manages the communication and coordination across all the entities, including the CPS, humans, and the applications. It controls the inter-domain CPS workflows at the edge through subscriptions and messages. Rather than having a single centralized controller deployment (including the distributed controller architectures or hierarchies that are still managed by a single entity, thus effectively an administratively centralized one), *SD-CPS* proposes a federated controller deployment. An *SD-CPS* federated controller deployment comprises numerous controllers from different domains or organizations at the edge. Each controller has protected access to the other controllers of the federated controller deployment, or a *Controller Farm* [25].

The CPS workload needs to be defined in a format that supports migration and interoperability between multiple edge nodes, due to the small-scale and heterogeneity of these nodes. Therefore, *SD-CPS* represents the workloads as a decomposable chain, a microservice workflow, to achieve a seamless execution across the execution environments. The workload is offloaded to the edge as web service invocations if the resource capacity is limited in the CPS firmware. With a unified view, *SD-CPS* chooses the workflows to be executed in its cyberspace or at the edge, based on the resource availabilities. Thus, *SD-CPS* develops a seamless, unified approach to modeling a CPS in cyberspace and executing it in the physical environment, consisting of the mechanical devices (physical space) as well as the edge nodes as the surrogates or extended cyberspace.

CPS applications are often first modeled as software simulations before being built as device firmware or deployed into the physical devices of CPS. *SD-CPS* performs a simulation through placeholders that can be controlled by its controller, in the same way the controller coordinates the modeled physical systems. The *SD-CPS* controller consists of the *Modeling Sandbox*, a controlled space to execute the CPS models. This controlled execution of the same code in the cyberspace significantly reduces the unpredictability of CPS executions. Figure 2 represents how the systems are modeled in the sandbox environment of the *SD-CPS* controller. The counterparts of the CPS physical space are modeled in the cyberspace as *virtual intelligent agents*. The interactions among the CPS counterparts in the physical space are mapped between the Virtual Intelligent Agents to reflect the communications in the cyberspace and to model the workflows better. Such a mapping offers latency-aware resource provisioning by

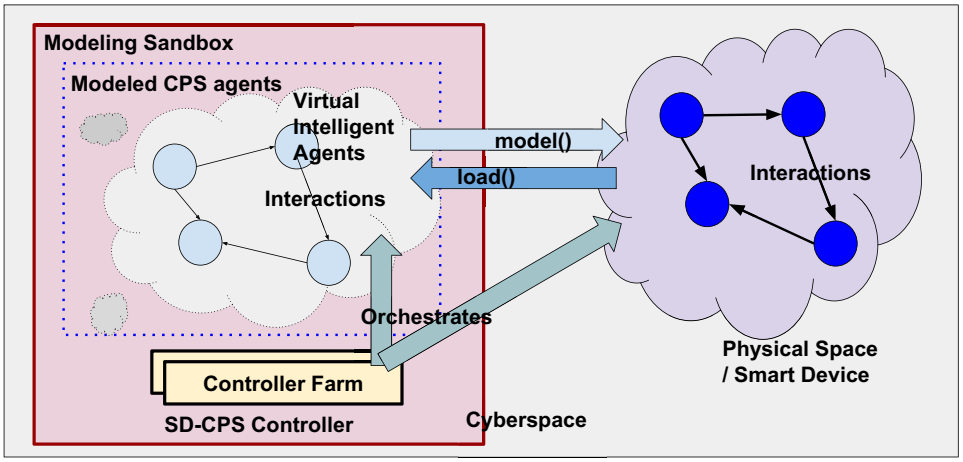


Fig. 2 CPS design and development with *SD-CPS* approach

understanding the interaction between the service workflows of *SD-CPS* entities.

The Controller Farm orchestrates both the physical systems and their simulated counterparts in the cyberspace. By simulating the API that connects the physical space into the cyberspace, *SD-CPS* creates a one-to-one mapping between the simulated Virtual Intelligent Agents and interdependent components of the physical system. The interactions are modeled and closely monitored in the Modeling Sandbox before the decisions are loaded into the physical space.

Thus, the Modeling Sandbox seamlessly models the executions in the cyberspace as simulations and emulations and then load the changes to the physical execution environment. The simulation functions as a virtual proxy for the designed system that it simulates—including its actors, such as sensors, actuators, and other physical and mechanical components. *SD-CPS* thus builds once and executes the same code in the controller's Modeling Sandbox or the physical space, reusing the same single development effort.

The *SD-CPS* modeling approach minimizes the code duplication by executing the real code from the controller, instead of having a simulation or model running custom code, thus independent of the actual execution. The *SD-CPS* controller, devised as an extension to OpenDaylight, is developed in Java, a high-level language. Therefore, it enables deployment of custom applications as controller plugins to alter or reprogram the behavior of CPS. We simulate the execution environment and the CPS workload through Java objects inside the SDN controller. The physical system loads the decisions from the cyberspace. The multi-tenant execution space of *SD-CPS* supports parallel modeling of multiple CPS. The microservice-based *SD-CPS* execution further avoids repeated computation efforts by caching the previously completed service outcomes.

3.3 Resource allocation

SD-CPS deploys the CPS workflows as services across the edge, aiming to satisfy the CPS policies while maximizing the overall resource utilization of the edge nodes. *SD-CPS* leverages the node utilization and health statistical data retrieved from the edge nodes as messages, to estimate the utilization of edge nodes. This health check of the edge nodes ensures that the resource requirements of the services are met, and the available nodes are sufficiently utilized with minimal idling nodes that are connected to the edge.

SD-CPS thus identifies, with the deployment of a service $\overline{s}$ of a CPS workflow $\overline{w}$, how much resources a node n or a link l_n that connects to the node will have in excess. The underutilization of a resource r of a node n (with $\overline{s}$ in n), and the underutilization of bandwidth of a link l_n (with $\overline{s}$ in l_n) are respectively defined by δ_{r_n} and $\delta_{b_{ln}}$ in Eq. 2. $(\sum_{s\in S_n} r_s)$ represents the total resource consumption by the other service workloads in the node n. Similarly, $(\sum_{s\in S_l} b_s)$ denotes the consumption of the link resources (such as bandwidth) by the existing service workloads sharing the link l. Complete details on the notations are listed in Table 1.

$$\begin{aligned} \delta_{r_n} &= r_n - \left(\left(\sum_{s\in S_n} r_s\right) + r_{\overline{s}}\right); \\ \delta_{b_{ln}} &= b_{l_n} - \left(\left(\sum_{s\in S_l} b_s\right) + b_{\overline{s}}\right) \end{aligned} \tag{2}$$

SD-CPS normalizes the variables using feature scaling as in Eq. 3, to give equal weight to all the variables considered (including δ_{r_n}, $\delta_{b_{ln}}$, t_{l_n}, and i_p). The max(X) and min(X) indicate the potential edge node with the maximum and minimum value for the variable X, respectively.

Table 1 Notation of the *SD-CPS* representation

$\mathcal{N}$	The set of nodes (including the cyberspace and the edge nodes)
$\mathcal{L}$	The set of links that connect the CPS to $\forall n \in \mathcal{N}$
R	The set of computing resources (CPU, memory, etc.) in a node
$P_{\mathcal{N}}$	The set of static properties of a node (availability, up time, etc.)
$P_{\mathcal{L}}$	The set of static properties of a link (multitenancy, QoS guarantees, etc.)
S_n	The set of services concurrently deployed in a node n
S_l	The set of services concurrently sharing a link l
X	The set of variables defining the utility value, Δ
l_n	A link towards the node n from the current CPS workload
i_p	Value of a static property p
r_n	Maximum resource capacity of a node
b_{l_n}	Maximum bandwidth allocated to a link l_n
t_{l_n}	Latency of a link l_n
r_s	Maximum resource consumption by a service s
b_s	Maximum bandwidth allocated to a service s

$$\forall x \in X, \hat{x} = \frac{x - min(X)}{max(X) - min(X)} \tag{3}$$

Equation 4 defines a compound utility value Δ_n for each n. Tenants define a coefficient for each of the resources, for their workflows to input the relative importance of each of node or link properties. The coefficients c_r, c_b, c_t, and c_p are respectively defined for (i) each resource (such as memory and CPU) availability in the nodes, (ii) the bandwidth availability in the link towards the node, (iii) latency to reach the edge from the CPS workload, and (iv) other static properties. *SD-CPS* chooses the nodes with the maximum utility value as the execution space for the services of CPS workflow.

$$\forall n \in \mathcal{N}, \forall l_n \in \mathcal{L}, \{c_r, c_b\} \subset \mathbb{Z}^+, \{c_t, c_p\} \subset \mathbb{N}, P = P_{\mathcal{N}} \cup P_{\mathcal{L}};$$
$$\Delta_n = \begin{cases} \sum_{r \in R}(c_r \cdot \hat{\delta}_{r_n}) + c_b \cdot \hat{\delta}_{b_{l_n}} + (\frac{c_t}{\hat{t}_{l_n}}) + \sum_{p \in P}(c_p \cdot \hat{i}_p), & \left(\prod_{r \in R} \delta_{r_n}\right) \times \delta_{b_{l_n}} > 0 \\ -\infty, & \left(\prod_{r \in R} \delta_{r_n}\right) \times \delta_{b_{l_n}} \leq 0 \end{cases} \tag{4}$$

The inherent properties (such as the quality of service (QoS) guarantees and uptime) of the nodes and the links $P_{\mathcal{N}}$ and $P_{\mathcal{L}}$ are static and do not change frequently (except in the case of failures, which edge providers are supposed to minimize, often abiding by the service level agreements (SLAs) [56]). However, the node properties such as current available memory and CPU are dynamic and change with time, based on the other service executions. Similarly, the bandwidth of a link is shared with various flows, and is limited by a maximum capacity b_{l_n}. b_{l_n} is shared by multiple tenant workflows, and each tenant workflow is throttled in bandwidth allocation for fair use of the bandwidth by various tenants. Thus, utilizing a particular link l_n for $\overline{s}$ depends on the existing virtual tenant network allocations.

SD-CPS aims to deploy the services in the nodes that satisfy the minimal resource requirements, that is also connected by a link to the CPS physical space with the necessary bandwidth. The utility value for the nodes that do not satisfy the minimal resource requirements for a workflow is set to negative infinity ($-\infty$) to avoid choosing these nodes for the CPS workflow deployment.

When multiple suitable edge nodes are available (the nodes represented with a utility value other than $-\infty$), *SD-CPS* chooses the one with the maximum utility value. Thus, *SD-CPS* identifies and uses the edge nodes that offer the best quality of experience (QoE) to the user, abiding by the user-defined policies. *SD-CPS* gives appropriate weights (as provided by the tenant) to each of the resource requirement while satisfying the minimum requirements for the workflow deployment at the nodes.

The resource allocation approach based on Δ is implemented as the core allocation algorithm and incorporated into the core SDN controller of *SD-CPS*, as an extension. The controller, based on its contextual information on the nodes and the links, ensures that the resource availabilities in the node and the link that connects the node can support the maximum resource requirements of the service workload. The controller then deploys the services of the workflow, fulfilling their resource requirements while aiming to satisfy the tenant policy in allocating the resources to the workflows. The execution node location for each type of service can be cached in the CPS devices, thus invoking controller only once before the start of the workflow. Subsequent services of the same nature can execute on the already chosen node, without reverting to the controller for the resource allocation, unless controller notifies of failure, congestion, or a change in the status of the current execution nodes.

3.4 *SD-CPS* controller

Figure 3 depicts the solution architecture of the *SD-CPS* controller. The software-defined sensor networks are built on top of the Controller Farm, representing each mobile terminal or a smart device as a sensor or an actuator. Tenant-Aware Virtual Network Allocation of *SD-CPS* virtually allocates the bandwidth among the *SD-CPS* tenants who share the execution space.

SD-CPS control plane can communicate with the underlying network through implementations of SDN southbound protocols such as OpenFlow [39] and interact with the devices that are non-compliant with OpenFlow through a similar approach inspired by OpenFlow. Various other southbound protocols can be incorporated into the *SD-CPS* controller to communicate with the devices, to support the systems that are not SDN-native. The incorporation of multiple protocols ensures that while *SD-CPS* has SDN at its core, it is not limited to software-defined networks with SDN switches that are still far from widespread in CPS settings (outside data centers).

An *SD-CPS* Controller Farm is a loosely connected federated deployment of controllers, without a static hierarchy or topology. Each controller in the Controller Farm communicates with other controllers in a wide area network flexibly through messages. The messages are managed through subscriptions, to orchestrate the CPS devices in the edge network. We extend OpenDaylight controller with advanced message queuing protocol (AMQP) [60] messaging integration, such that multiple stand-alone controllers can communicate and collaborate through messages brokered by ActiveMQ broker, without actually sharing the single global view of the network. The *SD-CPS* controller persists the messages in a queue for a higher level of parallel messages, to handle the scale efficiently. When the controller is overloaded, or the memory of the server that runs the controller is overutilized due to a higher number of messages, the messages are persisted to an instance of KahaDB file-based data store in the local filesystem. The stored messages can later be retrieved when necessary.

A new CPS domain can enter a multi-domain edge environment by using its controller to connect to the Controller Farm. The controllers communicate with each other to collaborate in a protected and managed manner, by passing messages, leveraging the public or dedicated/private connectivity between the domains. The MOM supports inter-domain communication and sharing the health statistics of the edge nodes to the relevant CPS. Local network topology data that is relevant to the other controllers, including information on the data tree, event notifications, and remote procedure calls (RPCs) are shared with other controllers, based on their subscriptions. Thus *SD-CPS* disseminates crucial information on its network topology and service health statistics. By leveraging SDN extended with MOM, the Controller Farm aims to provide a seamless scaling with the problem size.

4 Prototype implementation

The *SD-CPS* controller is built by extending an SDN controller. While any SDN controller can work with the architecture, we chose OpenDaylight as it offers a modular architecture based on Apache Karaf [43] OSGi runtime. We used OpenDaylight Beryllium [40] as the default core SDN controller, as its architecture makes it easy to extend it with MOM middleware. We used Oracle Java 9.0.4 as

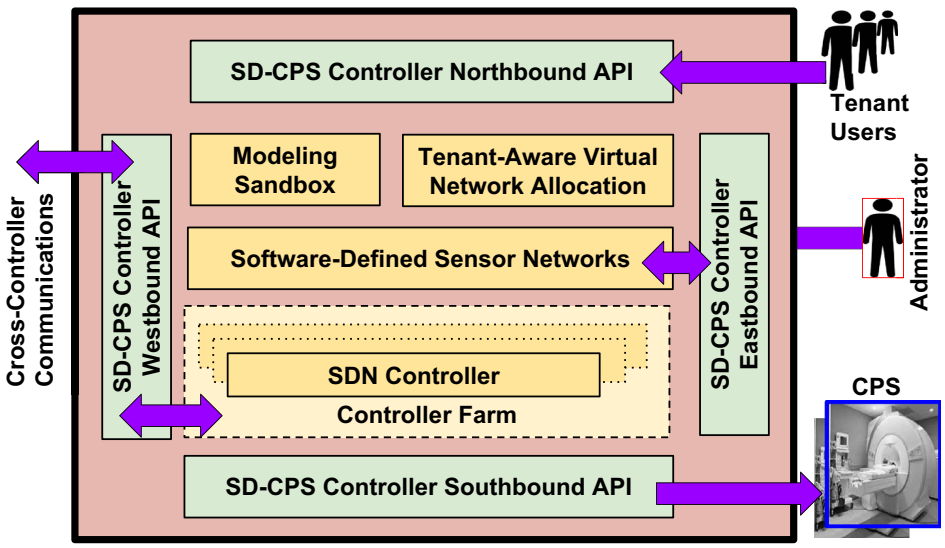


Fig. 3 *SD-CPS* controller architecture

the programming language and runtime, and ActiveMQ 5.15.2 [54] as the message broker for MOM protocols. We implemented the resource allocation algorithms as an extension to OpenDaylight.

SD-CPS controller exposes its functionalities to its users through its APIs following the nomenclature and style of the SDN controller APIs [23]. The APIs consist of the implementation of different integration protocols, and connection points for the extended distributed controller deployment for CPS. The Westbound API enables inter-control communication among the controllers in *SD-CPS*, as well as inter-domain communications across multiple *SD-CPS* controller deployments, through their westbound. The Eastbound API is leveraged by the *SD-CPS* system administrators to configure and manage the controller deployment.

The *SD-CPS* Controller Northbound API consists of the typical SDN northbound protocols including REST and MOM protocols such as AMQP or MQTT (formerly, message queue telemetry transport) [36] for the tenant processes to interact with the controller. As MOM protocols are long researched for use with networks of wireless sensors and actuators [10, 21], extending SDN with MOM increases its applicability, in addition to scalability. The Southbound API consists of implementations of SDN southbound protocols and MOM protocols for the communication with the physical devices. Based on the defined policies, rules, and the tenant inputs from northbound, controller determines the actions. The actions are propagated back to the data plane consisting of the SDN switches and physical devices through the southbound. Thus, the southbound API handles the communication, coordination, and integration of the network data plane consisting of the CPS devices with the control plane.

Controller Farm supports orchestrating CPS by controlling larger networks beyond data center scale, without binding to static network topologies in the wide area network. On the other hand, a fixed or static hierarchy of controllers would require a central controller deployment in one or a selected set of domains, thus forcing the controllers of other domains to accept a single controller as the core or primary authority (thus creating a controller hierarchy or levels of control). Opening up controllers for a higher level controller from outside domain could also make the network topologies of each CPS open for compromises from outside, making the crucial information on network topologies vulnerable to curious entities or onlookers from outside the domain or organization. Thus, a centralized approach would limit the applicability of the solution and is not feasible in a multi-domain edge environment with multiple CPS and third-party data center and service providers. Due to these reasons, we chose a Controller Farm instead of an inter-domain hierarchy of controllers.

Figure 4 shows the software-defined sensor networks enabled by *SD-CPS* as its communication medium. The model-driven service abstraction layer (MD-SAL) of OpenDaylight [40] enables a straightforward extension of the controller with more bindings. OpenDaylight controller by default consists of REST northbound bindings, through which the data tree, remote procedure calls (RPCs) [55], and notifications are exposed. Messaging4Transport is an OpenDaylight bundle that we developed to expose the OpenDaylight data tree as messages in a messaging protocol. While we used AMQP as the default messaging protocol, other messaging protocols can be implemented following the same design. The data tree of OpenDaylight MD-SAL is leveraged to store the dynamic context data sensed by the sensors of the various appliances. Each of the appliances subscribes to the relevant topics while publishing the relevant information sensed by their sensor, to be written to the data tree.

Integrating the sensor networks into the SDN controller enables quicker response to the dynamic changes in the context information. The SDN controller is often a supercomputer or a cluster of high-end servers. Hence, *SD-CPS* ensures efficient communication between its counterparts by leveraging its Software-Defined Sensor Networks approach and storing the dynamic data inside the data tree of the controller.

5 Evaluation

We deployed the *SD-CPS* controller on a server of AMD A10-8700P Radeon R6, 10 Compute Cores 4C+6G × 4 processor, 8 GB of memory, and 1 TB hard disk. We assess the performance of *SD-CPS* in a scenario of CPS execution at the edge as service workflows, and its efficiency in composing and managing CPS, through simulations and microbenchmarks.

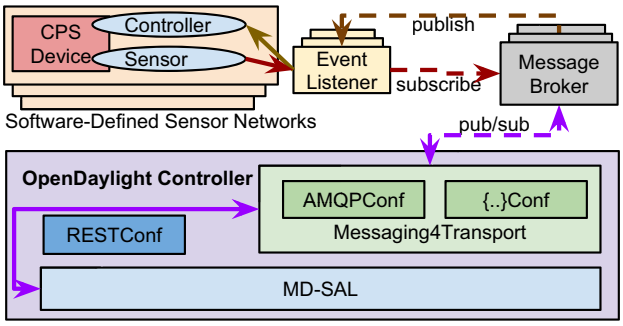


Fig. 4 Network layer—higher level view

5.1 CPS execution modeling

We deploy a prototypical VANET CPS with vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I) and infrastructure-to-vehicle (I2V) communications [15]. We model the traffic data monitoring workflow with *SD-CPS*, as a composition of several parallel instances of a few services. We first design an execution environment with several nodes, falling into different categories, and then develop a complex workflow involving V2I/I2V communication services.

5.1.1 Execution nodes at the edge

We modeled the execution environment with four categories of nodes: N1: 10,000 Embedded mobile systems in the connected cars, N2: 1000 edge cloud nodes, N3: 100 servers, and N4: 10 larger servers. Due to its pervasiveness and proximity to the vehicles, N1 has the least latency to most of the devices, while N4 has the highest of the CPU and memory with often poor latency from the vehicles compared to N1. Figure 5 depicts how the resource availability of each node, belonging to the 4 different categories, is feature scaled (as illustrated by Eq. 3 in Sect. 3) by the *SD-CPS* controller. The nodes are depicted with background processes, and each node can vary slightly within a range of its resource capabilities (as not all nodes will typically be strictly identical).

5.1.2 CPS workflows

Having designed our VANET CPS execution environment with *SD-CPS*, we model a vehicular monitoring workflow W. The workflow W is a composition of various services. Service S1 fetches and transforms personalized sensor information to the grid. Due to quick data transfer needs, S1 has a necessity for minimal latency. Throughput, CPU, and uptime are not crucial for its execution. Service S2 performs traffic data cleaning and integration at an aggregator node. It needs a minimal latency and also high memory for quick in-memory computations. S2 aims to output integrated data while avoiding outliers that may indicate malicious, dirty, or bogus data collected from S1. Service S3 performs data analysis for traffic congestion control with contextual data. It has moderate latency and memory requirements. Service S4 conducts data crunching with past and historical data for more data science computations, including traffic prediction. Latency is the least important among the resources for it, while it requires high bandwidth, CPU, and memory. Service S5 performs personalization of alerts and notifications to the vehicles. Characteristics and requirements of S5 are similar to those of S1.

W is modeled as $W = 1000S1 \circ 100S2 \circ 10S3 \circ 2S4 \circ 1000S5$, with parallel execution of multiple service instances of S1, S2, S3, S4, and S5 composing the workflow. 1000 of S1 service instances send their output to 100 of S2 service instances, which further have their outcome forwarded to 10 of S3 instances, which in turn have their output sent to 2 of S4 instances. Finally, the relevant results of S4 are sent to the 1000 instances of S5 for the final processing.

Figure 6 illustrates the resource requirements of the services in the workflow. A workload is modeled by a set of tuples, *<resource_requirement, resource_utility_function_weight>*, in defining the utility value Δ. In workflow W, all the coefficients for the resources are set to 1, for ease of representation. However, different values for each coefficient could be utilized for particular scenarios to indicate a different utility value compared to the other vehicles in the CPS. For example, workflows belonging to an ambulance or law enforcement authorities could be given higher priority, and such coefficients give tenants more control in choosing their execution nodes when multiple alternative nodes offer possible deployments to the services.

The negative normalized values for certain resources indicate that even the minimum available resources in a node is sufficient for the service. For example, S1, S2, and S5 have negative normalized utility value for uptime, indicating that all the node types can serve them, concerning uptime. On the other hand, the highly positive

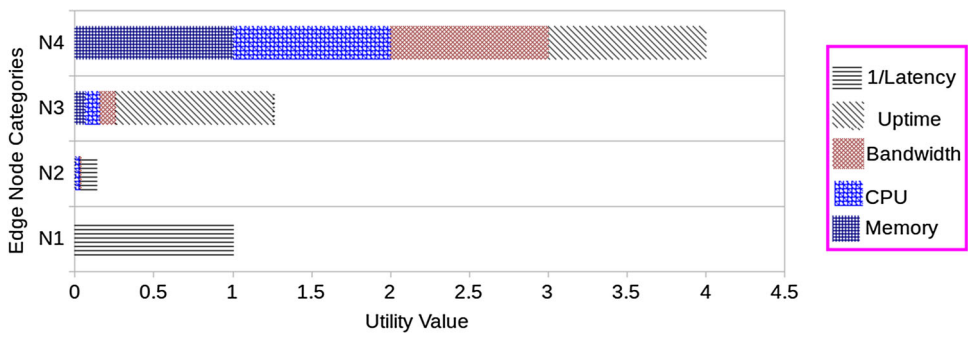


Fig. 5 Properties of the nodes (normalized)

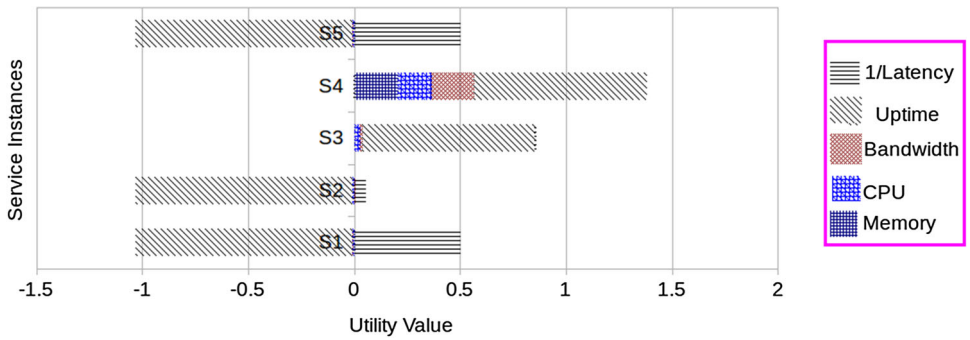


Fig. 6 Resource requirements (normalized) of the services

value of uptime for S3 and S4 show their demand for high uptime. Similarly, Fig. 6 also highlights the demand for minimal latency (or a high $\frac{1}{latency}$) for S1 and S5.

5.2 Resource allocation efficiency

We deploy 1 million instances of workflow W across several edge nodes (as described in Sect. 5.1) over the course of 1 h. The edge nodes and the workflows are coordinated by the *SD-CPS* Controller Farm. With the CPS and its workloads modeled, we evaluate the resource allocation efficiency of *SD-CPS*, with simulations and microbenchmarks. We aim to assess the efficiency in resource utilization and confirm that *SD-CPS* workflow-based approach minimizes idling nodes. Figure 7 illustrates the percentage of services hosted in each node types over a parallel deployment of the workflows during a time frame of an hour. Around 98% of the services were observed to be deployed in the node with the highest utility value. The deviation for the remaining 2% of services accounts for the full utilization of specific type of nodes at times, thus aiming to maximize the utilization of all the node types.

5.2.1 Resource utilization

Next, we evaluated the efficiency of resource utilization as well as the effectiveness of *SD-CPS* in minimizing the idling edge nodes. We assessed the performance of *SD-CPS* allocating services with parallel execution of 1 million workflows, over a timespan of an hour. Figure 8 illustrates the average resource utilization (%) across each node category. The resource utilization was always around 90%, with more variations in resource allocations to N4 nodes. We believe that having just 10 of the N4 nodes, each with abundant resources, yet with less bandwidth to manage the more frequently occurring services (S1 and S5) must have contributed to the relatively frequent drops in resource allocation of N4. The nodes of the other categories remained more stable in their resource allocation.

Figure 9 demonstrates the percentage of idling nodes during the execution of the workflows. Except during the start and the end of the workflows (fixed to have a time-frame of 1 h), there was near-zero percentage of idling nodes. The results highlight the efficiency of the resource allocation approach of *SD-CPS*, with minimal idling nodes and high resource utilization (when the requests for resources are high to utilize the abundant resources).

5.3 *SD-CPS* controller performance

We evaluated the performance, resilience, and scalability of the *SD-CPS* controller in coordinating the CPS components.

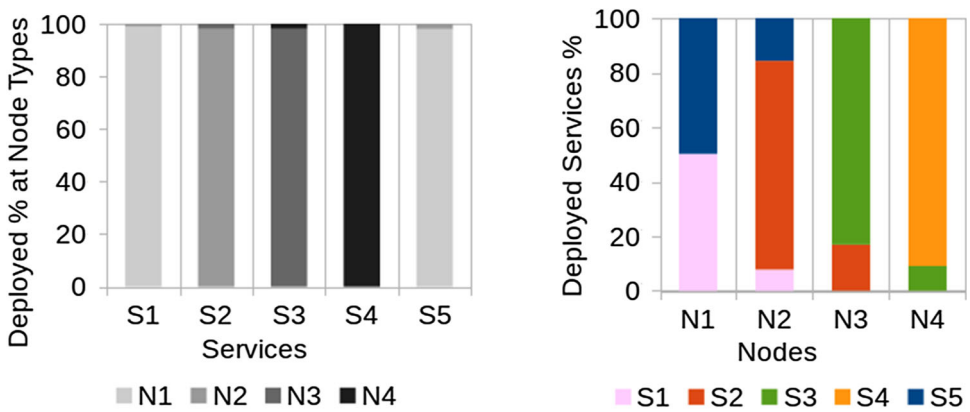


Fig. 7 Service deployment over the nodes

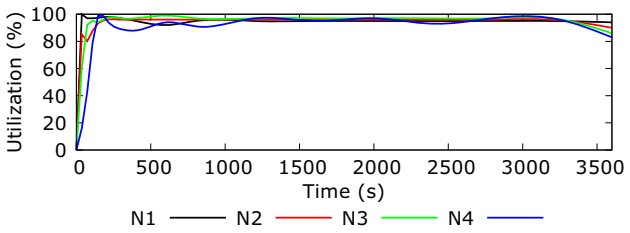


Fig. 8 Average resource utilization with a parallel execution of 1 million workflows

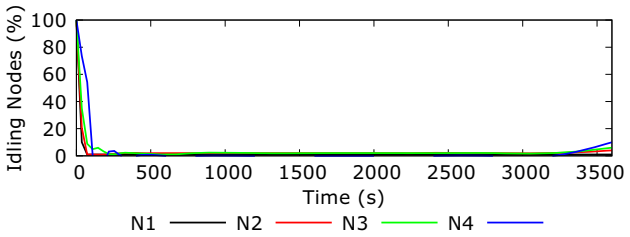


Fig. 9 Idling nodes during a parallel execution of 1 million workflows

5.3.1 Execution environment

We built a simple CPS with several sensors and several actuators, connected to a single *SD-CPS* controller deployment. We modeled the sensors as publishers sending data as messages to the controller to disseminate to the relevant destination nodes based on topics appropriately, and the actuators as those subscribed to receive the messages.

5.3.2 Success rate

First, we measured the performance of the *SD-CPS* controller through its success ratio against the number of messages it receives from the sensors. Through this, we aim to observe how many messages can be processed and delivered per second with a high success ratio with a single controller deployment of *SD-CPS*. Figure 10 illustrates the observed success rate of the message processing of a stand-alone deployment of the controller, with an increasing number of parallel messages.

The success rate remained high around 100%, always above 99.5% regardless of the number of parallel messages the controller handles. For up to 10,000 parallel messages, the success rate remained at 100%, and only slightly reduced down to 99.5% for a more substantial number of messages: up to 10 million parallel messages. Even the reported 0.5% of failures indicates just the failed attempt at the first attempt; *SD-CPS* resends the failed message, based on the configurations (whether to retry or ignore in case of a message processing or delivery failure). The evaluations indicated that the success rate of *SD-CPS* is mostly resilient regardless of the increased number of messages, except for a massive amount of messages such as 100 million parallel messages, which makes the server that hosts the controller run out of memory, due to a large number of messages in the queue in-memory.

5.3.3 Throughput

We then measured throughput of an *SD-CPS* controller as the number of messages entirely processed by the controller, arriving from the sensors to be forwarded towards a relevant receiver. We observed how the throughput varies with the number of concurrent and parallel messages at the controller. Figure 11 illustrates the throughput or the typical message processing rate of a single *SD-CPS* controller against that of the number of parallel messages that the controller processes. By effectively handling the parallel messages, the *SD-CPS* controller processes 5000 messages/s in a concurrency of 10 million messages.

5.3.4 Processing time

Finally, Fig. 12 illustrates the total time taken to process the complete set of messages at an *SD-CPS* controller, against the varying number of messages. The *SD-CPS* controller scaled linearly regarding processing time with the number of parallel messages. It processes 10 million messages in 40 min.

The results presented are for a single stand-alone deployment of the *SD-CPS* controller (the throughput or the number of messages processed in a given time can be

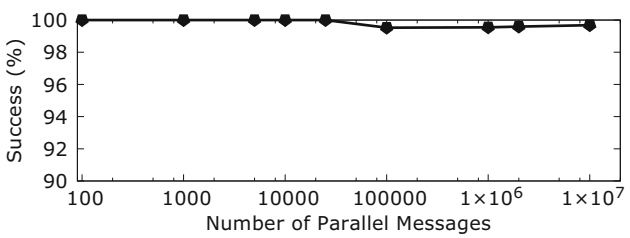


Fig. 10 Success rate of an *SD-CPS* controller vs. number of messages processed in parallel

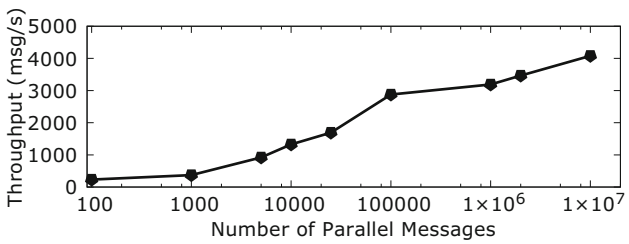


Fig. 11 Throughput of an *SD-CPS* controller versus number of messages processed in parallel

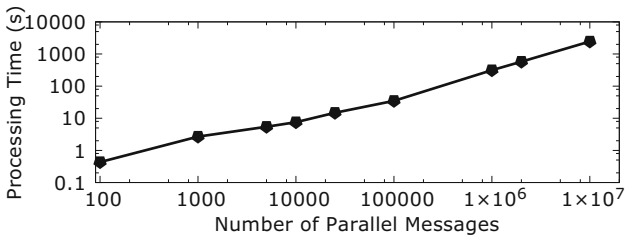


Fig. 12 Number of messages processed and the total processing time

increased by a clustered controller deployment in each controller domain). Nonetheless, *SD-CPS* is designed to operate with a federated controller deployment. With a decentralized, federated deployment, *SD-CPS* scales horizontally to cover a wider area with more CPS devices and improved latency. Therefore, the *SD-CPS* controller can further be scaled to handle more concurrent messages in each controller domain.

6 Related work

In this section, we will look into a few related works to *SD-CPS*.

6.1 SDN for distributed systems

Albatross [31] is a membership service that addresses the uncertainty of distributed systems. Albatross tends to be ten times more efficient than the previous membership services by exploiting the standard interface offered by SDN to monitor and configure network elements. It addresses common network failures, while avoiding interfering with working processes and machines, and maintaining a quick response time and high overall availability. The challenges such as split-brain scenarios and violations in availability and consistency that are addressed by Albatross are relevant for CPS too. However, while CPS is a distributed system, it has its peculiar challenges uncommon in a typical distributed system, starting from building the CPS to operating the CPS, due to its diverse nature in implementation and devices. While Albatross aims to address the uncertainty and difficulties in ensuring reachability of distributed systems, *SD-CPS* seeks to mitigate a list of challenges inherent to CPS by innovating a logically centralized control plane and management architecture. Albatross approach narrows itself to data centers equipped with software-defined networks, leaving wide area networks as future work. *SD-CPS* is designed exclusively for CPS, to scale up from the edge to the Internet scale.

6.2 Smart environments and CPS

Software-defined environment (SDE) [16, 32] focuses on factors such as, (i) resource abstraction based on capability, (ii) workload abstraction and definition based on policies, goals, and business/mission objectives, (iii) workload orchestration, and (iv) continuous mapping and optimization of workload and the available resources. SDN controller and physical and virtual SDN switches remain the heart of SDE. The control of computing, network, and storage is built atop a virtualized network. By leveraging Network Function Virtualization (NFV) [5], middlebox actions typically handled by hardware middleboxes such as firewalls and load balancers are replaced by relevant software components in an SDE.

Software-defined buildings (SDB) [13] envision a building operating system (BOS) which functions as a sandbox environment for heterogeneous device firmware to run as applications atop it. The BOS spans across multiple buildings in a campus, rather than confining itself to a single building. The scale of SDB and SDE can be increased through collaboration and coordination of the controllers of the buildings or environments. While SDB and SDE architectures can be extended for CPS, they cannot cater for CPS by themselves due to the variety and heterogeneity in the design and requirements of CPS. The increased dynamics and mobility of CPS compared to the environments controlled by SDB and SDE further hinder adopting them for CPS. *SD-CPS* caters to the resource provisioning, execution, and migration challenges of CPS, by offering a unified workflow-based approach at the edge, regardless of the specifics of domain, realization, or deployment of a CPS.

Hierarchy of orchestrators has been proposed to control multiple domains and services, consisting of an Overarching Orchestrator (OO) and a set of Domain Orchestrators (DOs). Each DO controls a specific technological domain by coordinating with the respective controller. The OO sits atop the multiple domains, orchestrating the DOs while ensuring end-to-end service orchestration in SDN and cloud environments [6]. The 5G Exchange (5GEx) builds upon SDN and NFV for a multi-domain orchestrator (MdO) that spans across multiple domains, technologies, and operators, for an automated and high-performance network service provisioning [52]. SDCPS [18] is an architecture with a hierarchy of multi-domain SDN controllers for CPS, similar to MdO. Both SDCPS and *SD-CPS* aims at bringing SDN to CPS. However, unlike the SDCPS and MdO approaches, *SD-CPS* does not require a hierarchy of SDN controllers, thus natively supports inter-domain edge environments with multiple service providers.

6.3 Software-defined internet of things (SDIoT)

SDIoT [22] proposes a software-defined architecture for IoT devices by handling the security [2], storage [12], and network aspects in a software-defined approach. SDIoT proposes an IoT controller composed of controllers for software-defined networking, storage, security, and others. This controller operates as an orchestrating middleware between the data-as-a-service layer consists of end-user applications, and the physical layer consists of the database pool and sensor networks. Software-Defined Industrial Internet of Things [61] extends SDIoT to industrial internet of things (IIoT), to offer more flexible networks to Industry 4.0. As CPS is a core enabler of Industry 4.0 [30], adopting SDN for IIoT is highly related to *SD-CPS* approach, though unlike *SD-CPS* this work is focused more on safety, reliability, and standardization aspects.

Various research and enterprise use cases are proposed and built, including SDIoT for smart urban sensing [33], and end-to-end service network orchestration [58]. Multinetwork INformation Architecture (MINA) self-observing and adaptive middleware [50] has been extended with a layered SDN controller to implement a controller architecture for IoT [48]. While sharing similarities with IoT, CPS is set to address a broader set of problems with more focus on ground issues on interoperability of cyber and physical dimensions in a CPS. While *SD-CPS* is inspired by the SDIoT approaches, it is built as an entirely new approach, targetting CPS. Hence, *SD-CPS* differs in scope and implementation to those of SDIoT, though they share similar motivation and can benefit from each other.

7 Conclusion

Wide-spread adoption of CPS is hindered by several challenges it has to tackle regarding building, operating, and maintaining the CPS. Recent research works have proposed approaches to address a few of these challenges. In this paper, we presented *SD-CPS*, a framework that aims to mitigate the common challenges faced by CPS, through a standard software-defined approach for the design and operation of CPS. We built *SD-CPS* as an extended architecture inspired by SDN, to orchestrate and manage the CPS workflows with unified control. *SD-CPS* leverages the edge resources in offloading the CPS workload as service workflows. We observe that the execution of CPS as a workflow of microservices can enable resource allocation and migration with higher performance, supporting the user policies and intents effectively.

We believe that combined with the SDN adoption in wide area networks, and the increasing reach of edge nodes, the *SD-CPS* approach will lead the next generation CPS. We propose as future work, to incorporate the *SD-CPS* approach with implementations of various networking and integration protocols for multiple use case scenarios. Furthermore, the *SD-CPS* deployments should be tested against baseline implementations of various CPS for their efficiency in addressing the identified challenges of CPS.

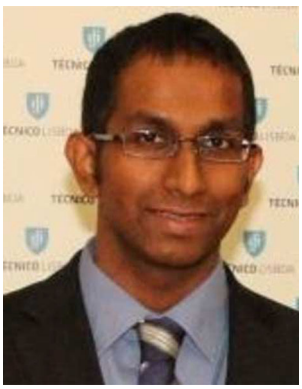
Acknowledgements This work was supported by national funds through Fundação para a Ciência e a Tecnologia with reference UID/CEC/50021/2013 and a Ph.D. grant offered by the Erasmus Mundus Joint Doctorate in Distributed Computing (EMJD-DC) under grant agreement 2012-0030.

References

1. Akyildiz, I.F., Lee, A., Wang, P., Luo, M., Chou, W.: A roadmap for traffic engineering in sdn-openflow networks. Comput. Netw. **71**, 1–30 (2014)
2. Al-Ayyoub, M., Jararweh, Y., Benkhelifa, E., Vouk, M., Rindos, A., et al.: Sdsecurity: a software defined security experimental framework. In: 2015 IEEE International Conference on Communication Workshop (ICCW), pp. 1871–1876. IEEE (2015)
3. Alheeti, K.M.A., Gruebler, A., McDonald-Maier, K.D., Fernando, A.: Prediction of dos attacks in external communication for self-driving vehicles using a fuzzy petri net model. In: 2016 IEEE International Conference on Consumer Electronics (ICCE), pp. 502–503. IEEE (2016)
4. Antonioli, D., Tippenhauer, N.O.: MiniCPS: a toolkit for security research on CPS networks. In: Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy, pp. 91–100. ACM (2015)
5. Batalle, J., Riera, J.F., Escalona, E., Garcia-Espin, J.A.: On the implementation of NFV over an OpenFlow infrastructure: routing function virtualization. In: 2013 IEEE SDN for Future Networks and Services (SDN4FNS), pp. 1–6. IEEE (2013)
6. Bonafiglia, R., Castellano, G., Cerrato, I., Risso, F.: End-to-end service orchestration across SDN and cloud computing domains. In: 2017 IEEE Conference on Network Softwarization (NetSoft), pp. 1–6. IEEE (2017)
7. Boyd, R.: Network Service Orchestration enabled by Tail-f. https://blogs.cisco.com/cin/tail-f (2015)
8. Calvaresi, D., Marinoni, M., Sturm, A., Schumacher, M., Buttazzo, G.: The challenge of real-time multi-agent systems for enabling IOT and CPS. In: Proceedings of the International Conference on Web Intelligence, pp. 356–364. ACM (2017)
9. Cardenas, A., Amin, S., Sinopoli, B., Giani, A., Perrig, A., Sastry, S.: Challenges for securing cyber physical systems. In: Workshop on Future Directions in Cyber-physical Systems Security, p. 5 (2009)
10. Collina, M., Corazza, G.E., Vanelli-Coralli, A.: Introducing the QEST broker: scaling the IoT by bridging MQTT and REST. In: 2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications-(PIMRC), pp. 36–41. IEEE (2012)
11. Curry, E.: Message-oriented middleware. In: Middleware for Communications, pp. 1–28 (2004)
12. Darabseh, A., Al-Ayyoub, M., Jararweh, Y., Benkhelifa, E., Vouk, M., Rindos, A.: SDStorage: a software defined storage experimental framework. In: 2015 IEEE International Conference on Cloud Engineering (IC2E), pp. 341–346. IEEE (2015)
13. Dawson-Haggerty, S., Ortiz, J., Trager, J., Culler, D., Katz, R.H.: Energy savings and the "software-defined" building. IEEE Des. Test Comput. **29**(4), 56–57 (2012)

14. Derler, P., Lee, E.A., Vincentelli, A.S.: Modeling cyber-physical systems. Proc. IEEE **100**(1), 13–28 (2012)
15. Dey, K.C., Rayamajhi, A., Chowdhury, M., Bhavsar, P., Martin, J.: Vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication in a heterogeneous wireless network-performance evaluation. Transp. Res. C **68**, 168–184 (2016)
16. Dixon, C., Olshefski, D., Jain, V., DeCusatis, C., Felter, W., Carter, J., Banikazemi, M., Mann, V., Tracey, J.M., Recio, R.: Software defined networking to support the software defined environment. IBM J. Res. Dev. **58**(2/3), 3:1–3:14 (2014)
17. Dong, X., Lin, H., Tan, R., Iyer, R.K., Kalbarczyk, Z.: Software-defined networking for smart grid resilience: opportunities and challenges. In: Proceedings of the 1st ACM Workshop on Cyber-Physical System Security, pp. 61–68. ACM (2015)
18. Freris, N.M., et al.: A software defined architecture for cyber-physical systems. In: 2017 Fourth International Conference on Software Defined Systems (SDS), pp. 54–60. IEEE (2017)
19. Glancy, D.J.: Autonomous and automated and connected cars-oh my: first generation autonomous cars in the legal ecosystem. Minn. J. Law Sci. Technol. **16**, 619 (2015)
20. Gurgen, L., Gunalp, O., Benazzouz, Y., Gallissot, M.: Self-aware cyber-physical systems and applications in smart buildings and cities. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 1149–1154. EDA Consortium (2013)
21. Hunkeler, U., Truong, H.L., Stanford-Clark, A.: MQTT-S—a publish/subscribe protocol for wireless sensor networks. In: 3rd International Conference on Communication Systems Software and Middleware and Workshops, 2008 (COMSWARE 2008), pp. 791–798. IEEE (2008)
22. Jararweh, Y., Al-Ayyoub, M., Benkhelifa, E., Vouk, M., Rindos, A.: SDIoT: a software defined based internet of things framework. J. Ambient Intell. Humanized Comput. **6**(4), 453–461 (2015)
23. Jarschel, M., Zinner, T., Hoßfeld, T., Tran-Gia, P., Kellerer, W.: Interfaces, attributes, and use cases: a compass for SDN. IEEE Commun. Mag. **52**(6), 210–217 (2014)
24. Karnouskos, S.: Cyber-physical systems in the smartgrid. In: 2011 9th IEEE International Conference on Industrial Informatics (INDIN), pp. 20–23. IEEE (2011)
25. Kathiravelu, P., Veiga, L.: CHIEF: controller farm for clouds of software-defined community networks. In: 2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW), pp. 1–6. IEEE (2016)
26. Kathiravelu, P., Veiga, L.: SD-CPS: taming the challenges of cyber-physical systems with a software-defined approach. In: 2017 Fourth International Conference on Software Defined Systems (SDS), pp. 6–13. IEEE (2017)
27. Lee, E.A.: Computing Foundations and Practice for Cyber-physical Systems: A Preliminary Report. Technical Report UCB/EECS-2007-72. University of California, Berkeley (2007)
28. Lee, E.A.: Cyber physical systems: design challenges. In: 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), pp. 363–369. IEEE (2008)
29. Lee, E.A.: The past, present and future of cyber-physical systems: a focus on models. Sensors **15**(3), 4837–4869 (2015)
30. Lee, J., Bagheri, B., Kao, H.A.: A cyber-physical systems architecture for industry 4.0-based manufacturing systems. Manuf. Lett. **3**, 18–23 (2015)
31. Leners, J.B., Gupta, T., Aguilera, M.K., Walfish, M.: Taming uncertainty in distributed systems with help from the network. In: Proceedings of the Tenth European Conference on Computer Systems, p. 9. ACM (2015)
32. Li, C.S., Brech, B., Crowder, S., Dias, D., Franke, H., Hogstrom, M., Lindquist, D., Pacifici, G., Pappe, S., Rajaraman, B.: Software defined environments: an introduction. IBM J. Res. Dev. **58**(2/3), 1:1–1:11 (2014)
33. Liu, J., Li, Y., Chen, M., Dong, W., Jin, D.: Software-defined internet of things for smart urban sensing. IEEE Commun. Mag. **53**(9), 55–63 (2015)
34. Liu, K., Ng, J.K., Lee, V.C., Son, S.H., Stojmenovic, I.: Cooperative data scheduling in hybrid vehicular ad hoc networks: VANET as a software defined network. IEEE/ACM Trans. Netw. **24**(3), 1759–1773 (2016)
35. Liu, Y., Shou, G., Hu, Y., Guo, Z., Li, H., Seah, H.S.: Towards a smart campus: innovative applications with wicloud platform based on mobile edge computing. In: 2017 12th International Conference on Computer Science and Education (ICCSE), pp. 133–138. IEEE (2017)
36. Locke, D.: MQ Telemetry Transport (MQTT) v3.1 Protocol Specification. IBM developerWorks Technical Library. http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html (2010)
37. Macker, J.: Mobile ad hoc networking (manet): routing protocol performance issues and evaluation considerations (1999)
38. Mahmood, A., Casetti, C., Chiasserini, C.F., Giaccone, P., Harri, J.: Mobility-aware edge caching for connected cars. In: 2016 12th Annual Conference on Wireless On-Demand Network Systems and Services (WONS), pp. 1–8. IEEE (2016)
39. McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J.: OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (2008)
40. Medved, J., Varga, R., Tkacik, A., Gray, K.: Opendaylight: towards a model-driven SDN controller architecture. In: Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014 (2014)
41. MEF: Lifecycle Service Orchestration|Third Network. https://www.mef.net/third-network/lifecycle-service-orchestration (2017)
42. Munir, S., Stankovic, J.A.: Depsys: dependency aware integration of cyber-physical systems for smart homes. In: 2014 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS), pp. 127–138. IEEE (2014)
43. Nierbeck, A., Goodyear, J., Edstrom, J., Kesler, H.: Apache Karaf Cookbook. Packt Publishing Ltd., Birmingham (2014)
44. Pacheco, J., Tunc, C., Hariri, S.: Design and evaluation of resilient infrastructures systems for smart cities. In: 2016 IEEE International Smart Cities Conference (ISC2), pp. 1–6. IEEE (2016)
45. Pasha, M., Khan, K.U.R.: Architecture and channel aware task offloading in opportunistic vehicular edge networks. IJSSST **18**(4), 15.1–15.7 (2015)
46. Patel, P., Ali, M.I., Sheth, A.: On using the intelligent edge for IoT analytics. IEEE Intell. Syst. **32**(5), 64–69 (2017)
47. Persson, M., Håkansson, A.: A communication protocol for different communication technologies in cyber-physical systems. Procedia Comput. Sci. **60**, 1697–1706 (2015)
48. Qin, Z., Denker, G., Giannelli, C., Bellavista, P., Venkatasubramanian, N.: A software defined networking architecture for the internet-of-things. In: 2014 IEEE Network Operations and Management Symposium (NOMS), pp. 1–9. IEEE (2014)
49. Qin, Z., Do, N., Denker, G., Venkatasubramanian, N.: Software-defined cyber-physical multinetworks. In: 2014 International Conference on Computing, Networking and Communications (ICNC), pp. 322–326. IEEE (2014)
50. Qin, Z., Iannario, L., Giannelli, C., Bellavista, P., Denker, G., Venkatasubramanian, N.: Mina: a reflective middleware for managing dynamic multinetwork environments. In: 2014 IEEE Network Operations and Management Symposium (NOMS), pp. 1–4. IEEE (2014)

51. Sampigethaya, K., Poovendran, R.: Cyber-physical system framework for future aircraft and air traffic control. In: 2012 IEEE Aerospace Conference, pp. 1–9. IEEE (2012)
52. Sgambelluri, A., Tusa, F., Gharbaoui, M., Maini, E., Toka, L., Perez, J., Paolucci, F., Martini, B., Poe, W., Hernandes, J.M., et al.: Orchestration of network services across multiple operators: the 5G exchange prototype. In: 2017 European Conference on Networks and Communications (EuCNC), pp. 1–5. IEEE (2017)
53. Siozios, K., Soudris, D., Kosmatopoulos, E.: Cyber-Physical Systems: Decision Making Mechanisms and Applications. River Publishers, Delft (2017)
54. Snyder, B., Bosnanac, D., Davies, R.: ActiveMQ in Action, vol. 47. Manning, Shelter Island (2011)
55. Srinivasan, R.: RPC: Remote procedure call protocol specification version 2 (1995)
56. Stantchev, V., Schröpfer, C.: Negotiating and enforcing QoS and SLAs in grid and cloud computing. In: International Conference on Grid and Pervasive Computing, pp. 25–35. Springer (2009)
57. Vegni, A.M., Biagi, M., Cusani, R.: Smart vehicles, technologies and main applications in vehicular ad hoc networks. In: Vehicular Technologies-Deployment and Applications. InTech (2013)
58. Vilalta, R., Mayoral, A., Pubill, D., Casellas, R., Martínez, R., Serra, J., Verikoukis, C., Muñoz, R.: End-to-end SDN orchestration of IoT services using an SDN/NFV-enabled edge node. In: Optical Fiber Communication Conference, W2A-42. Optical Society of America (2016)
59. Villari, M., Fazio, M., Dustdar, S., Rana, O., Chen, L., Ranjan, R.: Software defined membrane: policy-driven edge and internet of things security. IEEE Cloud Comput. **4**(4), 92–99 (2017)
60. Vinoski, S.: Advanced message queuing protocol. IEEE Internet Comput. **10**(6), 87 (2006)
61. Wan, J., Tang, S., Shu, Z., Li, D., Wang, S., Imran, M., Vasilakos, A.V.: Software-defined industrial internet of things in the context of industry 4.0. IEEE Sens. J. **16**(20), 7373–7380 (2016)
62. Wan, J., Zhang, D., Zhao, S., Yang, L., Lloret, J.: Context-aware vehicular cyber-physical systems with cloud support: architecture, challenges, and solutions. IEEE Commun. Mag. **52**(8), 106–113 (2014)
63. Wichtlhuber, M., Reinecke, R., Hausheer, D.: An sdn-based cdn/isp collaboration architecture for managing high-volume flows. IEEE Trans. Netw. Serv. Manag. **12**(1), 48–60 (2015)
64. Yousefi, S., Mousavi, M.S., Fathy, M.: Vehicular ad hoc networks (vanets): challenges and perspectives. In: 2006 6th International Conference on ITS Telecommunications Proceedings, pp. 761–766. IEEE (2006)



Pradeeban Kathiravelu is a Ph.D. Researcher at INESC-ID Lisboa / Instituto Superior Técnico, Universidade de Lisboa, Portugal, and Université catholique de Louvain, Belgium. He is a Fellow of Erasmus Mundus Joint Degree in Distributed Computing (EMJD-DC), researching Software-Defined Service Compositions at the Edge. He holds a Master of Science degree, Erasmus Mundus European Master in Distributed Computing (EMDC), from Instituto Superior Técnico, Portugal, and KTH Royal Institute of Technology, Sweden. He also holds a Bachelor of the Science of Engineering (Hons) degree, majoring Computer Science & Engineering, with a first class from the University of Moratuwa, Sri Lanka. His research interests include Software-Defined Networking (SDN), Distributed Systems, Cloud Computing, Web Services, Big-Data in Biomedical Informatics, and Data mining. He is highly interested in free and open source software development, and is an active participant in the Google Summer of Code (GSoC) program since 2009, as a student and as a mentor.

Peter Van Roy is professor in the ICTEAM Institute at the Université catholique de Louvain since 1996, where he heads the Programming Languages and Distributed Computing Research Group. He has an Engineering degree from the Vrije Universiteit Brussel (1983), a Ph.D. in Computer Science from UC Berkeley (1990), and a Habilitation à Diriger des Recherches from the Université Paris Diderot (1996). He coordinated the European project SELFMAN on self management of large-scale systems and was partner in the projects MANCOOSI, EVERGROW, PEPITO, and CoreGRID. He is a developer of the Mozart Programming System and coauthor of a well-known textbook on computer programming. He teaches two MOOCS on computer programming in the edX platform. He is currently partner in the SyncFree project (syncfree.lip6.fr), which is building systems for large-scale computation without synchronization.

Luís Veiga is a (tenured) Associate Professor in the Computer Science and Engineering Department at IST/UTL. He has taught courses on Middleware for Distributed Internet Applications, Virtual Execution Environments and Cloud Computing and Virtualization. He is a Senior Researcher at INESC-ID, and Group Manager for the Distributed Systems Group. He has been an active participant in government and industry funded R&D projects and he is leading two research projects financed by FCT (Portuguese Science Foundation) on P2P cycle-sharing systems, and as local coordinator of two others in distributed virtual machines and multicore component programming. He coordinates locally the FP7 CloudForEurope project, participates in FP7 Timbus project on digital preservation and virtualization. His research interests include distributed systems, replication, virtualization technology and deployment, distributed garbage collection, middleware for mobility support, grid and peer-to-peer computing. He has 90 scientific publications (Best-paper at Middleware 2007, Best-Paper Award Runner Up at IEEE CloudCom 2013, and Best-Paper candidate nominee at IEEE CloudCom 2014) peer-reviewed scientific communications in workshops, conferences, book chapters, edited books, and journals since 2000. He was one of the General Chairs of ACM Middleware 2011, Track-Chair for CloudCom 2013 and Euro-Par 2014, and he has previously served in international conferences as member of program committee, proceedings editor (ACM Middleware 2012, EuroSys 2007, ACM PPPJ 2007 and 2008, and MobMid/M-MPAC Workshop at ACM Middleware 2008, 2009, and 2010) and as reviewer. He is a member of the

Editorial Board of the Journal of Internet Services and Applicaitons (JISA), Springer, and International Journal of Big Data Intelligence (Inderscience). He is a member of the ACM Middleware Steering Committee. He was three times "Excellence in Teaching" IST mention recipient (2008, 2012, 2014), Best Senior Researcher at INESC-ID (2014) nominated (2013), Best Young Researcher at INESC-ID (2012) nominated (2010, 2011). He is a member of IEEE, ACM and EuroSys. He has served terms on CS Department Council, CS-PhD Coordination Commission, and University Assembly. He is a member of the Scientific Board of the MSc in Communication Networks and the Erasmus Mundus Master in Distributed Computing. He is a chair of the IEEE Computer Society Chapter, IEEE Section Portugal for 2014–2016.