Future Generation Computer Systems 113 (2020) 541–555

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

A self-integration testbed for decentralized socio-technical systems

Farzam Fanitabasi [a,*], Edward Gaere [a], Evangelos Pournaras [b]

[a] Professorship of Computational Social Science, ETH Zurich, Zurich, Switzerland
[b] School of Computing, University of Leeds, Leeds, UK

ARTICLE INFO

Article history:
Received 3 February 2020
Received in revised form 29 May 2020
Accepted 13 July 2020
Available online 18 July 2020

Keywords:
Internet of Things
Testbed architecture
Socio-technical system
Self-integration
Decentralized systems
Multi-agent system

ABSTRACT

The Internet of Things (IoT) comes along with new challenges for experimenting, testing, and operating decentralized socio-technical systems at large-scale. In such systems, autonomous agents interact locally with their users, and remotely with other agents to make intelligent collective choices. Via these interactions they self-regulate the consumption and production of distributed (common) resources, e.g., self-management of traffic flows and power demand in Smart Cities. While such complex systems are often deployed and operated using centralized computing infrastructures, the socio-technical nature of these decentralized systems requires new value-sensitive design paradigms; empowering trust, transparency, and alignment with citizens' social values, such as privacy preservation, autonomy, and fairness among citizens' choices. Currently, instruments and tools to study such systems and guide the prototyping process from simulation, to live deployment, and ultimately to a robust operation of a high Technology Readiness Level (TRL) are missing, or not practical in this distributed socio-technical context. This paper bridges this gap by introducing a novel testbed architecture for decentralized socio-technical systems running on IoT. This new architecture is designed for a seamless reusability of (i) application-independent decentralized services by an IoT application, and (ii) different IoT applications by the same decentralized service. This dual self-integration promises IoT applications that are simpler to prototype, and can interoperate with decentralized services during runtime to self-integrate more complex functionality, e.g., data analytics, distributed artificial intelligence. Additionally, such integration provides stronger validation of IoT applications, and improves resource utilization, as computational resources are shared, thus cutting down deployment and operational costs. Pressure and crash tests during continuous operations of several weeks, with more than 80K network joining and leaving of agents, 2.4M parameter changes, and 100M communicated messages, confirm the robustness and practicality of the testbed architecture. This work promises new pathways for managing the prototyping and deployment complexity of decentralized socio-technical systems running on IoT, whose complexity has so far hindered the adoption of value-sensitive self-management approaches in Smart Cities.

1. Introduction

The Internet of Things (IoT) radically transforms how complex socio-technical systems are designed, operated and managed. Smart Cities turn into organic ecosystems of ubiquitous sensors, autonomous vehicles, and personal pervasive devices that are massively interconnected and distributed [1–3]. New opportunities arise to control and manage socio-technical systems in real-time as the means to cope with uncertainties and continuous change [4–6]: self-improving socio-technical operations by seamlessly self-integrating decentralized services that measure, learn, optimize, and adapt [7–9], i.e. load-balancing transport or power networks to prevent traffic congestion and blackouts. However, the IoT complexity, heterogeneity, scale, infrastructural cost, and privacy concerns have so far limited the broader experimentation and research on designing such general-purpose services [2,10]. Nevertheless, decentralized systems exhibit properties that can empower values by design in a socio-technical context: (i) They can better preserve privacy by processing sensitive information locally and allowing informational self-determination [11,12]. (ii) They are more transparent against algorithmic nudging and manipulation, as data are not centrally located and users preserve their autonomy [13,14]. (ii) They can be designed to promote social welfare such as fairness [15,16]. Therefore, their adoption in socio-technical IoT applications of Smart Cities has a social and sustainability impact.

Prototyping and testing decentralized socio-technical systems that continuously change and adapt is a challenge. In particular, testing in real-world self-improving system integration (SISSY)

* Correspondence to: Stampfenbachstrasse 48, 8092 Zurich, Switzerland.
E-mail address: farzamf@ethz.ch (F. Fanitabasi).

https://doi.org/10.1016/j.future.2020.07.036

[4–6] as the means to cope with complex system dynamics as well as user and network uncertainties remains to a high extent an ad hoc process. This paper introduces a new IoT testbed architecture with a novel dual self-integration capability: (i) An IoT application integrates several application-independent and modular decentralized services to compose low-cost complex functionalities without changing the application implementation. (ii) A decentralized service is integrated into several IoT applications without changing the service implementation. This reusability is made possible by abstracting the software engineering complexity and interactions within two software agents under user's control: the *application agent*, and *service agent*.

Prototyping IoT applications and services to support, in particular, multiple self-integration scenarios, requires testing and refinements at multiple stages that start from simulations, move to live deployments, and ultimately to high Technical Readiness Level (TRL)[1] operations. Maintaining different implementations, or changing the code back and forth to validate new functionality is costly and complex [3,17]. Experience shows that such flexibility for decentralized multi-agent systems is extremely scarce [1,2]. Existing toolkits cannot serve in practice the self-integration scenarios envisioned. This barrier is overcome by introducing a prototyping toolkit that extends and improves earlier work [18]: The *Livepeer* toolkit. It provides support for IoT devices, i.e. software agent running on smart phones, a new efficient networking module, a new scalable logging infrastructure for system monitoring and analysis, as well as an improved design to limit earlier severe memory leaks and synchronization problems.

The testbed architecture with the two software agents is experimentally evaluated with real-world data under long-lasting pressure and crash tests over several weeks. The complex operations of two decentralized socio-technical services are integrated in Livepeer as a proof of concept: (i) I-EPOS (*Iterative Economic Planning and Optimized Sections*) [8], and (ii) DIAS (*Dynamic Intelligent Aggregation Service*) [9]. I-EPOS performs decentralized combinatorial optimization using learning agents with structured interactions. In contrast, DIAS performs real-time collective measurements over a dynamic unstructured network of agents, where agents can arbitrary join, leave, or fail, while their input data continuously change. Both services empower highly sophisticated IoT application scenarios, such as traffic flow optimization, power peak-shaving, load-balancing of bike sharing stations, participatory crowd-sensing of mobility, and traffic [8,9,16,19]. Results confirm the self-integration capability, and the performance benchmarks validate the robustness of the testbed architecture in scenarios of continuous change and adaptation, with more than 80,000 agents joining and leaving the network, 2.4 million parameter changes, and 100 million communicated messages. The findings of this paper provide new insights to communities, government bodies, system operators and utilities on how to manage, operate, and regulate complex socio-technical IoT infrastructures.

In summary, the contributions of this paper are as follows: (i) A conceptual testbed architecture that facilitates dual self-integration of various decentralized services by an IoT application, and different IoT applications by a decentralized service. (ii) The realization of the conceptual testbed architecture by abstracting the software engineering complexity in two software agents and their interactions in a generic communication protocol. (iii) An improved and extended distributed prototyping toolkit for decentralized socio-technical systems of TRL-6. (iv) Improvements of the I-EPOS software artifact [8] that transitions from simulations to live deployment with a demonstrated TRL-6 continuous operation. (v) A proof of concept based on the self-integration and experimental evaluation of two decentralized services for IoT applications under highly dynamic environments.

The rest of this paper is outlined as follows: Section 2 reviews relevant previous work. Section 3 introduces the testbed architecture, and the realization protocol. Section 4 illustrates the Livepeer toolkit, and Section 5 introduces the two studied services. Sections 6 and 7 illustrate the experimental methodology, and evaluations, respectively. Finally, Section 8 concludes this paper and outlines future work.

2. Related work

Self-adaptive frameworks have been earlier introduced to address the complexity, heterogeneity and uncertainties [39,40] of large-scale integrated networked systems such as pervasive/ubiquitous computing and IoT [41]. Such frameworks study adaptive service composition in dynamic environments at runtime, utilizing various techniques, such as context-aware computing [42,43], service re-selection heuristics [44] and parallel service execution [45,46]. However, these frameworks often do not address the self-integration of different physical devices at runtime [39].

In the field of IoT, experimental facilities and physical testbeds have been subject to extensive previous research and surveys [1,2,10,38,47,48]. Physical testbeds equip researchers with deployed and ready-to-use physical devices, simplifying the design and evaluation of novel IoT systems and services (e.g., network protocols, Big Data algorithms, city-wide IoT services) under realistic operational conditions [20–25]. One example is SmartSantander [21] with a city-wide scale (∼20,000 sensors). SmartSantander nodes act only as data sources and can be configured (centrally via a management plane) to run applications such as environmental monitoring. However, in physical testbeds, often the domain/project-specific requirements determine the design and technological aspects (i.e., communication protocols) of sensors, smart objects, and middleware. This limits their reusability in different domains and applications. To tackle such challenges, the *PaaS* (platform-as-a-service) model has been studied and utilized. The PaaS model leverages standard interfaces and interoperability measures, to provide researchers with tools to rapidly develop, execute, and manage IoT systems without the complexity of building and maintaining the infrastructure [3]. This enables the design and deployment of cross-application IoT platforms [3]. Xively[2] is an example in the context of distributed cloud-based applications with a centralized control plane, where different tasks are executed in separate platforms and devices. For instance, application-level functions can be executed in different virtual and real entities to reduce latency and bottlenecks. Nevertheless, PaaS approaches often neglect socio-technical requirements, such as data locality, privacy, autonomy, and decentralized control.

Agent-based computing has been used extensively to enable cooperative, decentralized, dynamic, and open IoT systems [38]. In such systems, agents autonomously interact and cooperate based on (typically) asynchronous message passing mechanisms to perform a task or a service. Shared communication standards facilitate agent interoperability and allow for incorporating heterogeneous resources. Examples of such systems include Lysis [3], which introduces a PaaS model with virtualized autonomous social agents, allowing for the deployment of fully distributed applications. ACOSO-Meth [17] introduces an agent-oriented architecture based on IoT smart objects, as well as a taxonomy for assessing system-level requirements and technological readiness

[1] https://www.nasa.gov/directorates/heo/scan/engineering/technology [Last accessed: May 2020].

[2] https://xively.com [Last accessed: May 2020]

Table 1
Comparison of related work.

Related Work	Paradigm	Abstraction	Socio-Technical System Design Considerations				Reusability	Interoperability
			Data Locality	Privacy	Autonomy	Decentralized Control		
FIT IoT-Lab [20]	PT	H	–	–	–	–	H	Te + Sy
SmartSantander [21]	PT	H	–	✓	–	–	H	Te + Sy
City of Things [22]	PT	H	–	–	–	–	H	Te
CityLab [23]	PT	H	–	–	–	–	H	Te
SmartCampus [24]	PT	H	–	–	–	–	H	Te + Sy
MakeSense [25]	PT	H	–	✓	–	–	H	Te
VICINITY [26]	PaaS	H + D	–	✓	–	–	H + D	Te + Sy + Se
IoTbed [27]	TaaS	H	–	–	–	–	H	Te + Sy
Xively [28]	PaaS	D	–	–	–	–	D	Sy
Lysis [3]	AO-PaaS	D (AO)	✓	✓	✓	✓	D	Sy + Se
AoT [29]	AO-Arch	D (AO)	–	–	✓	–	D	Te + Sy
SIoT [30]	AO-Arch	D (AO)	–	–	✓	✓	–	Se
iSapiens [31]	AO-Arch	D (AO)	–	–	✓	–	D	Te + Sy
BEMOSS [32]	AO-Arch	D (AO)	–	–	–	–	D	Te + Sy
UBIWARE [33]	AO-Arch	D (AO)	–	–	–	–	D	Te + Sy + Se
FIoT [34]	AO-Arch	D (AO)	–	–	✓	✓	D	Te + Sy
ACOSO-Meth [17]	AO-Arch	D (AO)	✓	–	✓	–	D + S	Sy + Se
VIVO [35]	Framework	D	✓	✓	–	–	D	Sy
iCore [36]	Framework	D (AO)	–	✓	✓	–	D + S	Te + Sy + Se
Fluidware [37]	Framework	D	✓	✓	–	✓	D	Sy + Se
Proposed	AO-Arch	D (AO) + S	✓	✓	✓	✓	D + S	Sy

Symbols: PT = Physical Testbed. PaaS = Platform-as-a-Service, TaaS = Testbed-as-a-Service, AO-Arch = Agent-Oriented Architecture. H = Hardware, D = Device, S = Service, Te = Technical, Sy = Syntactical, Se = Semantical.
The abstraction indicates the three possible levels of applied abstraction: H: Hardware abstraction by providing software routines to access the hardware via programming interfaces. D: Device abstraction by having virtualized counterparts for each IoT device at the system-level. The AO indicates whether virtual counterpart is an agent. Agents are networked software components that autonomously perform specific tasks on device/user behalf by interacting with other agents and with their environment [17]. S: Indicates service-level abstraction by providing common communication protocols for IoT services. Data locality, refers to local processing of data, and autonomy is the ability of the device to autonomously interact and actuate its function. Decentralized control indicate the existence/lack of central control entities at the service-level. Reusability refers to the ability to reuse the Hardwares (H), Devices (D), or Services (S) in different application scenarios. The interoperability illustrates the utilized communication paradigm. Technical refers to technological approaches (e.g., bluetooth), syntactical the shared message formats, and semantical the use of shared ontologies and knowledge representation [38].

of IoT systems. While agent-based approaches utilize device virtualization and address some socio-technical considerations, on the service-level they often suffer from lack of standard interfaces and interoperability. Thus, to reuse a specific service in different applications, the code and communication protocol should change. The proposed architecture in this paper utilizes service abstraction to enable the reusability of devices and services in various application domains.

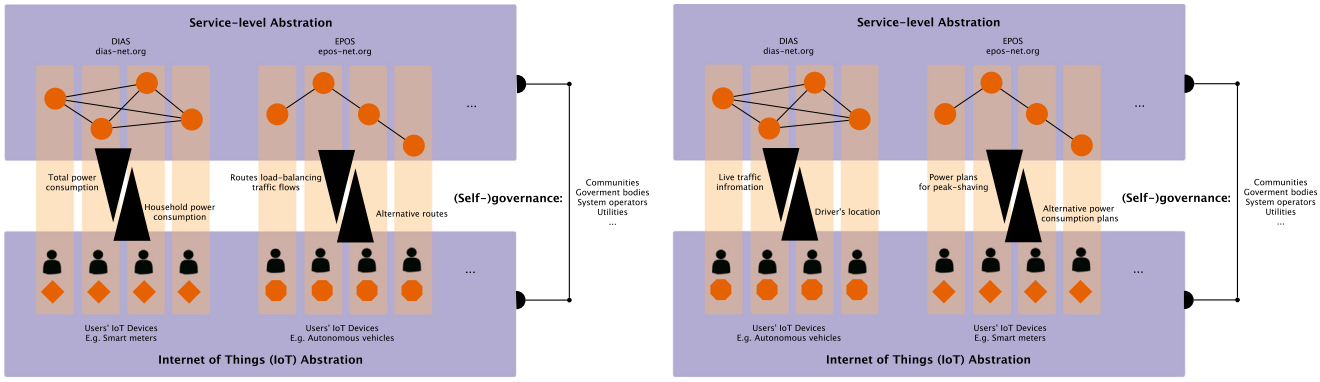
IoT systems are operated in increasingly dynamic and complex environments [37], where during system runtime, devices can fail, users might join/leave, communication across devices and agents becomes disrupted, system goals and requirement can vary, and new services are required. Not all such changes can be foreseen during the initial design phase. Often it is infeasible for a centralized controller to have knowledge of all such changes in a timely manner. Self-adaptive approaches, autonomic computing [49,50], and hierarchical self-aware decision-making [51] have been studied as means to handle such changes at runtime, with minimal human intervention [5,50,52,53]. To this end, the proposed testbed architecture facilitates the rapid prototyping and experimentation of IoT services that can handle dynamic environments (with high TRL) as well as autonomously initialize and include various devices and services during runtime. Table 1 illustrates a non-exhaustive comparison between relevant previous research, providing insights of the existing experimental IoT testbeds.[3]

3. A conceptual IoT testbed architecture for system self-integration

This paper introduces a conceptual testbed architecture, designed to enable seamless reusability and self-integration of (i) application-independent decentralized services by an IoT application, and (ii) different IoT applications by the same decentralized service. This architecture focuses on decentralized socio-technical IoT services with autonomous agents, without central authority to coordinate the agents and their actions. These agents interact locally with their users and remotely with each other to make intelligent collective choices, via which they can self-regulate the consumption and production of common resources. Hence, in this context a service is essentially a distributed software running on multiple agents. Examples of such services include monitoring services [54], real-time analytics [9], planning and coordination systems [8], learning techniques [55], and distributed control systems [7]. Fig. 1 illustrates the conceptual testbed architecture, and two examples of self-integrating different decentralized services with different IoT applications.

To enable the aforementioned reusability and self-integration, the proposed architecture utilizes two levels of abstraction: (i) IoT application level, and (ii) decentralized service level. At the IoT application level, this abstraction creates *application agents*: a piece of software lying on each user's IoT device, acting as the middleware for communication with the decentralized service. These IoT devices provide sensing and actuation capabilities. They are of different types (e.g., sensors, mobile phones) and geo-spatially distributed. At the decentralized service level, this abstraction creates *service agents, as one-to-one* counterparts for each application agent. Each service agent has the following tasks: (i) Receiving data from the corresponding application agent (IoT device). (ii) Executing the service by interacting and cooperating with other service agents. Finally, (iii) Providing the outcome of the service to the application agent (e.g. in the form of control commands). This dual abstraction creates a decoupling between the internal operations of the IoT application and the complex functionality of the decentralized service: The first abstraction level (application to services) facilitates the inclusion of heterogeneous devices, and their reusability of their applications in

[3] Note that the comparisons and distinction of the socio-technical considerations are based on the system design goals rather than subsequent third-party augmentations and applications.



(a) Monitoring power demand and optimizing traffic.

(b) Monitoring traffic and optimizing power demand.

Fig. 1. Conceptual testbed architecture and two examples of application scenarios by self-integrating different decentralized services with different IoT applications. Note how by switching the coupling of the two IoT applications with the two decentralized services, new application scenarios are seamlessly supported.

different services, while the second abstraction level (service to applications) simplifies the reusability of decentralized services in new applications, as the interfaces, communication logic, and protocols remain unaffected by changes in the IoT devices or applications. In production-ready systems, both the application and service agents can run on the same computational node to reduce latency and provide data locality, i.e. on a user's device such as a smartphone, or at two remote nodes, i.e. on a user's device, a cloud node, or a crowdsourced community server.[4]

The deployment and governance of the testbed depend on the target IoT application. The service agents are deployed and managed by the service operator, which can be a third-party mediator in sensing-as-a-service scenarios [47], or a community in case of participatory sensing applications [57]. Examples of third-party mediators include companies such as Waze,[5] Uber,[6] and Swiss Mobility,[7] while environmental monitoring [58], and urban sensing [59] are examples of participatory sensing applications deployed and managed by service communities. Furthermore, Smart City scenarios run by municipalities [60], Smart Grids [61], and smart supply chains [62] are examples where a central authority such as the municipality or the utility company governs the system. However, in participatory sensing scenarios, such as environmental monitoring [58], and urban sensing [59], the testbed can be self-governed by users and the service community (public good infrastructure).

3.1. Communication protocol & runtime cycle

A distributed protocol is designed for the communication and self-integration between the two abstracted levels. This generic communication protocol is application and service independent. It determines the communication logic and common interfaces between service and application agents. Table 2 illustrates the various system entities in the protocol according to Fig. 1a. Fig. 2 illustrates the protocol sequence diagram, and runtime cycle. The protocol is outlined as follows: (i) The service operator initializes the service agents, and the gateway. The gateway act as a bootstrapping proxy, connecting the service agents

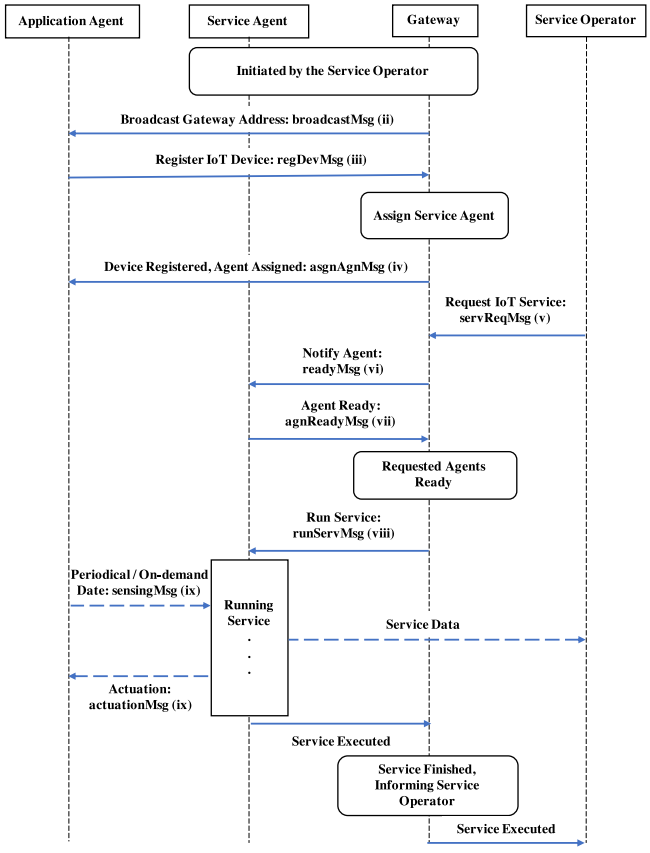


Fig. 2. The communication protocol that realizes the conceptual testbed architecture. The details of messages are illustrated in Section 3. This protocol treats the running services as blackboxes and create standard interface between different components of the testbed. Hence, different devices and services can be self-integrated at runtime.

to the corresponding application agent. The gateway is agnostic of data, the internal processes of IoT applications and decentralized services.[8] (ii) It is assumed that application agents know the public address of the gateway. This is possible via the `broadcastMsg:{GWAddr, servInfo}` by the gateway, where

[4] Such as the Diaspora [56] foundation: diasporafoundation.org [Last accessed: May 2020].

[5] https://www.waze.com [Last accessed: May 2020].

[6] https://www.uber.com [Last accessed: May 2020].

[7] https://www.mobility.ch [Last accessed: May 2020].

[8] In practice, the gateway does not need to be a separate entity, and can be incorporated in a service agent.

Table 2
System entities.

Entity	Explanation	Example (Fig. 1a)
IoT Application	Control logic	Monitoring power demand
IoT Device	Sensing and actuation	Smart meters
IoT Service	Autonomous general-purpose agents	Collective measurements
Gateway	System bootstrapping proxy for service agents	Software
Service Operator	Setting up service agents and the gateway	Communities, power utility

`GWAddr` is the gateway address, and `servInfo` indicates the service. (iii) To connect to the decentralized IoT service, each application agent contacts the service gateway, registers itself, and its corresponding IoT device, via the `regDevMsg:{devAddr, devInfo, servInfo}`, where `devAddr` is the application agent's address, and `devInfo` includes information such as device type, and location. (iv) In response, the gateway assigns a service agent to the IoT device and informs the application agent via the `asgnAgnMsg:{agnAddr}`, where `agnAddr` is the address of the assigned service agent. (v) The service operator submits a service request to the gateway, specifying its requested service, and execution metadata, via the `servReqMsg:{servInfo, servMD}`, where `servMD` contains the metadata required to execute the service, such as the number of service agents, number of devices, and their locations. (vi) The gateway receives the service request, and notifies the service agents, via the `readyMsg:{servInfo, servMD}`. (vii) The service agents validate the service information along with the associated metadata, and reply to the gateway, via the `agnReadyMsg:{agnAddr, servInfo}`. (viii) When all agents are notified and ready to run the service, the gateway sends the execute service command, via the `runServMsg:{servInfo}`. (ix) Each service agent requests/receives data from the application agent via the `sensingMsg:{servInfo, data}`, and submits control/actuation messages to the IoT device either periodically, on-demand, or at the end of the service execution via the `actuationMsg:{servInfo, actuation}`. (x) Finally, after the service is executed, the application agent, gateway, and the service operator are informed.

4. Prototyping decentralized IoT systems

To ensure efficient and reliable performance, continuous testing and refinements are needed, from the early stages of simulation, to live deployment, to long-lasting stable operation. Moreover, high costs and complexity are involved in the maintenance of different implementations when code is changed back and forth to validate new functionality and expand to new application domains. To address these challenges, this paper introduces the Livepeer toolkit. Livepeer is based on an improved version of the general-purpose prototyping toolkit Protopeer [18], now made highly robust and efficient for long-term operations. Livepeer comes with a high Technical Readiness Level (TRL-6) and its new modules include: (i) The redesigned and reengineered Protopeer node, providing core functionality such as communication protocols (e.g. TCP messaging), timers, and an execution environment for the service agents. (ii) Software clients, acting as application agents, for supporting IoT devices (i.e. smartphones). (iii) A networking module for efficient and reliable application-to-services and service-to-applications communication. (iv) A scalable monitoring infrastructure, integrated to each computational node, for application/service monitoring and analysis.

4.1. Livepeer: redesigning & reengineering protopeer

The Protopeer toolkit [18] is designed with the main goal of facilitating the rapid prototyping of P2P applications, and the transition from simulation to live environments. However, the transition from live environments to robust, long-lasting, TRL-6 operations is not trivial, as the latter often has a larger scale, higher realism regarding performance degradation by network partitions, and message losses. For instance, long-term operations require magnitude resources, incur a higher number of operations, and communicate a larger volumes messages. This results in the discovery of unforeseen system faults and deficiencies, such as sporadic message losses, deadlocks, synchronization, excessive thread counts, and memory leaks by evolving communication processes. To address the above challenges, this paper introduces a redesigned and reengineered version of Protopeer for highly efficient and robust long-lasting experimentation. In addition to several implementation-specific improvements, the redesigned Protopeer includes: (i) A redesigned communication and networking module, enabling multiple queues for message processing (Section 4.2), and (ii) a novel module for distributed monitoring and logging of services, events, and memory usage (Section 4.3). This new system has been tested extensively over several weeks under highly dynamic environments, with approximately 3000 node join/leaves, 150,000 runtime parameter changes, and over 2.1 million exchanged messages per day (Section 7.2).

Fig. 3 illustrates a summarized view of the internal architecture of a Livepeer node. There are two core concepts within each node: the peer, and the peerlet. The peer provides core functionality such as communication protocols (e.g., TCP messaging), and timers. It acts as the execution environment (container) for the peerlets. The peerlets, on the other hand, are independent modules that provide specific functionality and tasks. Typically a node consists of a single Livepeer peer and multiple peerlets that collectively fulfill the required functionality of the service. In addition to the service agent, two other examples of peerlets are the communication topology peerlet, and the monitoring peerlet. The communication topology peerlet determines the service network topology and the communication logic, for instance, a tree-topology [63] (utilized in I-EPOS, see Section 5.1), or the gossip-based peer sampling for P2P systems [64] (utilized in DIAS, see Section 5.2) The monitoring peerlet stores and submits logs from different modules in the peer to the monitoring infrastructure (Section 4.3). By default, this peerlet includes three different logging modes: (i) *Service logger*, which logs the specific service logs. (ii) *Event logger*, which provides event-based logging, and insights into execution sequence. (iii) *Memory logger*, which measures the total memory footprint of a peer in memory, including nested objects.

4.2. Communication platform

The proposed testbed utilizes a messaging protocol based on a fast and lightweight TCP/IP implementation using ZeroMQ for agent-to-agent communication. Each agent is instantiated with a

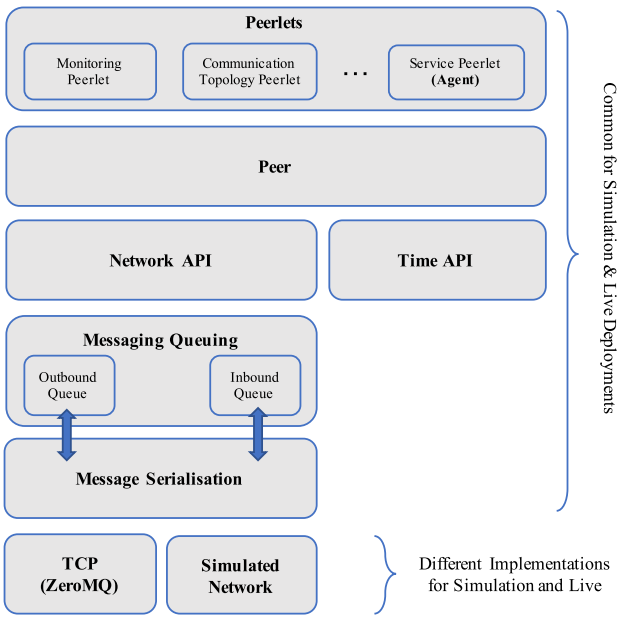


Fig. 3. Internal architecture and modules of the Livepeer node, a redesigned version of the Protopeer node. The modules are separated in two categories: deployment-dependent and deployment-independent. This design has the advantage to first test a system in a controlled simulated environment and then gradually move to a large-scale live deployment, while the deployment-dependent modules are the only ones that require change.

single PULL socket and multiple PUSH sockets. Thus, each agent can act as a sink and receives messages from any other agents in the network, whilst simultaneously sending messages to other agents through the PUSH sockets. This is achieved by implementing two independent messaging queues, one for inbound messages, and one for outbound messages. This separation allows the monitoring of queue size within each agent to regulate traffic flows.

4.3. Monitoring infrastructure

A major challenge in decentralized IoT services with autonomous agents is to log and monitor the service, both at the individual agent level, as well as system-wide [65]. Devices and agents can be geo-spatially distributed, and deployed over different computational clusters and networks [66]. While each agent can run autonomously and independently, most analytics require an aggregated view in real-time, whilst also allowing to drill-down and investigate the internal activities within a single agent. Providing such multi-granular views is even more challenging for large-scale systems [67]. Simple NFS (Networked File System) solutions may not provide the necessary throughput to sustain heavy logging from several agents [67]. Additionally, IoT devices and agents can be restricted in computational resources, hence, the logging and monitoring needs to be lightweight, simple to use, efficient, and with minimal impact on the real-time performance [68].

To address these challenges, this paper devises a monitoring infrastructure, comprised of the following components: (i) A single database, containing the logged data from agents. (ii) A single logging gateway, accessible to all agents, which receives the logs and commits them to the database. The main task of the logging gateway is to perform authentication, authorization, and connection pooling for the database. (iii) A single, lightweight peerlet on each agent, known as the monitoring peerlet, which collects the logs and submits them to the logging gateway.[9] This infrastructure is easy to integrate (by only adding the peerlet) in Livepeer nodes. It is distributed, and modular in design, and can be connected to various observability platforms and dashboards, such as Grafana[10] and Redash[11] for real-time visualization.

5. Studied services

This paper studies two live implementations of generic multi-purpose IoT services as proof-of-concept and use-cases of the proposed architecture; Namely, collective learning based on the I-EPOS system (Section 5.1), and decentralized collective measurements based on the DIAS system (Section 5.2). I-EPOS performs collective learning for multi-agent combinatorial problems [8], utilizing a structured network topology (tree-topology) with synchronous learning iterations and communication between agents. DIAS performs decentralized privacy-preserving data analytics, relying on local computations, peer-to-peer interactions, and hashed information [7,9]. It has an unstructured topology (P2P) for asynchronous communication [9]. Due to their decentralized socio-technical design, both of these services are very relevant to IoT applications. However, they are profoundly different in their operation, which challenges the flexibility and applicability of the proposed architecture.

5.1. Collective learning

The I-EPOS system [8] performs fully decentralized, self-organizing, privacy-preserving combinatorial optimization.[12] The I-EPOS agents (service agents) provide the learning service, and each has a set of local plans generated by the application agent. These plans can be alternative routes from an autonomous vehicle, or power consumption schedules from a smart appliance (e.g., smart washing machine). I-EPOS optimizes a system-wide goal, measured by a global cost function. This goal can be load-balancing traffic flows in a city [19], or peak-shaving power demand for Smart Grids [16]. The I-EPOS agents interact and cooperate with each other to select a plan that minimizes the global cost. The agents self-organize in a tree-topology [63] as a way of structuring their interactions. I-EPOS performs consecutive learning iterations, which includes two phases: the *bottom-up* (leaves to root) phase and *top-down* (root to leaves) phase. At each iteration t, agent u selects the plan $p^t_{u,s}$ to satisfy the following optimization objective:

$$p^t_{u,s} := \arg\min_{j=1}^{|P_u|} \Big((1-(\alpha+\beta)) \left(f_G(p^t_{u,j})\right) + \beta \left(f_L(p^t_{u,j})\right) + \alpha \left(f_U(p^t_{u,j})\right) \Big) \tag{1}$$

In the above equations, $|P_u|$ is the number of plans for agent u, $f_G(p^t_{u,j})$ is the global cost of selecting $p^t_{u,j}$, which can be the variance of traffic load across different routes (in case of traffic load-balancing). Each plan has a local cost, calculated by $f_L(p^t_{u,j})$, which can be the trip duration (in case of alternative routes), or user discomfort (in case of shifting power consumption). $f_U(p^t_{u,j})$ is the unfairness, calculated by the dispersion of the local cost of the selected plans over all agents, with lower values indicating

[9] The earlier version of the Protopeer toolkit saves logging objects locally that creates a discrepancy for post-processing and analysis as systems operate in the long run.

[10] https://grafana.com [Last accessed: May 2020]

[11] https://redash.io [Last accessed: May 2020]

[12] Available at http://epos-net.org [Last accessed: May 2020].

more equal distribution of the local cost across all agents. The α, β, and $1-(\alpha+\beta)$ parameters indicate the agents' preferences for unfairness, local cost, and global cost, respectively. For instance, an agent with $\alpha = 0,\ \beta = 1$ is known as a *selfish* agents, which prioritizes the minimization of its local cost, while another agent with $\alpha, \beta = 0$ is known as an *altruistic* agent, which minimizes the global cost. After the final iteration F is completed, $p^F_{u,s}$ is presented to the users' devices for execution. Further elaboration on I-EPOS is out of the scope of this paper and the interested reader is referred to previous work [8].

5.2. Decentralized collective measurements

The DIAS system [9] performs fully decentralized privacy-preserving data analytics for the Internet of Things.[13] Each application agent acts as a data supplier and consumer: Data suppliers are sensors that locally generate a stream of real-time privacy-sensitive data, while data consumers collect these data to compute information (e.g., summation, average, max/min, top-k.). For instance, data suppliers provide consumption data from residential smart meters, and data consumers receive the aggregated power consumption of the neighborhood, as well as status updates about the reliability of the Smart Grid [69]. This process is fully decentralized and privacy-preserving, as users' data is not shared with a central entity. Each pair of data supplier and consumer is connected to a DIAS agent (service agent). Each data supplier discovers data consumers in the service network, to which the local sensor data is sent. This discovery is performed via a fully decentralized gossiping protocol: the peer sampling service [70]. Data suppliers spread the local sensor data in the network periodically by pushing them to remote data consumers to maintain a high accuracy in the estimations of the aggregation functions. Each data consumer collects the input data for the computation of the aggregation functions. Finally, data suppliers receive the outcome of the performed aggregation. Each bilateral interaction between a data supplier and consumer is referred to as an aggregation session.

The raw data from data suppliers can be privacy-sensitive and rapidly varying over time. To tackle this challenge, DIAS utilizes data summarization by assigning the raw values from stream data to a selected state chosen from a limited number of k possible states. Additionally, DIAS addresses two other uncertainties: changes in the set of possible states, and agents leaving/failing/rejoining the network. The challenge here is to preserve the accuracy of DIAS estimations under these two dynamics. To address this, DIAS uses a distributed memory system based on Bloom filters [71] to track the history of the performed computations and when needed perform self-corrective actions. Further elaboration on DIAS is out of the scope of this paper and the interested reader is referred to previous work [7,9].

6. Experimental methodology & settings

The experiments in this paper are divided into two evaluation scenarios, both utilize the Livepeer toolkit, and follow the conceptual architecture, and communication protocol illustrated on Figs. 1 and 2, respectively. The first evaluation scenario (Section 6.1) studies the accuracy of the two services in live environments, to provide a performance benchmark for the testbed architecture and Livepeer toolkit given experimental realism [21]. The second evaluation scenario (Section 6.2) studies the efficiency and robustness of the services during long-lasting operation under dynamic and volatile environments.

[13] Available at http://dias-net.org [last accessed: May 2020].

Table 3
I-EPOS settings and parameters for the two evaluation scenarios. The profiles for evaluation scenario I are illustrated in Table 4. In evaluation scenario II, the number of agents range between [150, 250], and the α, β values range between [0, 1], given $\alpha + \beta = 1$.

Parameters	Value in evaluation scenario I	Value in evaluation scenario II
Performed experiments	100 per profile	Continuous: Intensity change every 8 h
Number of agents	50/100/300	[150, 250]
Dataset	EV dataset	EV dataset: 7-days ahead
Plans per agent	4	4
Plan dimensions	1440/4320/10,080	10,080
Number of iterations	50	50
Global cost function	MIN-VAR/RMSE	MIN-VAR/RMSE
Local cost function	Discomfort	Discomfort
Agent preference	$\alpha = 0, \quad \beta = 0/1$	$\alpha, \beta \in [0, 1], \ \alpha + \beta = 1$
Network topology	Balance binary tree	Balance binary tree

6.1. Evaluation scenario I: Comparing accuracy in non-volatile environments

Experiments under live environments, even without dynamic changes, can incur inaccuracies due to networking errors (e.g., packet losses), clock differences across machines, and system failures [10]. This evaluation scenario analyzes the accuracy of the two studied scenarios, and provides a benchmark comparison in live non-volatile environments, to study the validity of the testbed architecture and the Livepeer toolkit. For I-EPOS, this evaluation is made by comparing the simulation and live deployments of the service, and for DIAS the evaluation is made based on long-lasting operation with high experimental realism.

6.1.1. I-EPOS

The experimental settings and parameters for I-EPOS in evaluation scenario I are illustrated in Table 3. The utilized dataset contains charging plans for 2779 electric vehicles (EV) in three different planning horizons: 1, 3, and 7 days ahead [72]. In all cases, each EV has 4 alternative charging plan in the form of a vector, specifying the energy demand for each minute during the planning horizon. For 1-day-ahead plans, the length is 1440 (24 h $*$ 60 min), and for 3 and 7-days-ahead plans, the length is 4320 ($3d * 24$ h $*$ 60 min), and 10,080 ($7d * 24$ h $*$ 60 min), respectively. Two different global cost functions are applied to the total charging demand of the participating EVs, each addressing a different charging scenario: (i) Minimizing charging demand variance (MIN-VAR), and (ii) shifting charging times to night (MIN-RMSE).[14] Each plan also has a local cost, which is its discomfort calculated by the historical likelihood of using the EV while charging [72].[15] The performed experiments are based on the 12 profiles illustrated in Table 4. Each profile is tested 100 times, and overall there are 1200 experiments in the simulation, and 1200 in the real-world environment. To compare the performance between the simulation and live environments, the relative differences between the global cost and average local cost are calculated as:

$$\begin{aligned} &\textit{Relative global cost difference} : \frac{g^t_{s,i} - g^t_{l,i}}{g^t_{s,i}} \\ &\textit{Relative average local cost difference} : \frac{l^t_{s,i} - l^t_{l,i}}{l^t_{s,i}} \end{aligned} \tag{2}$$

[14] MIN-RMSE: Minimizing the root mean square error between the total charging demand of all EVs, and the steering signal set by the service operator to incentivize night charging. The steering signal is a vector of the same length as the charging plans, with the day-time charging target set to 0.

[15] Further elaboration on this dataset can be found in previous work [72].

Table 4
12 Profiles used for I-EPOS experiments scenario I.

Profiles	Scale	Agent preference	Planning horizon	Global cost function
1		$\alpha = 0,\ \beta = 0$	1 day ahead	MIN-VAR
2	Small	$\alpha = 0,\ \beta = 1$	1 day ahead	MIN-VAR
3	(50 agents)	$\alpha = 0,\ \beta = 0$	1 day ahead	RMSE
4		$\alpha = 0,\ \beta = 1$	1 day ahead	RMSE
5		$\alpha = 0,\ \beta = 0$	3 days ahead	MIN-VAR
6	Medium	$\alpha = 0,\ \beta = 1$	3 days ahead	MIN-VAR
7	(100 agents)	$\alpha = 0,\ \beta = 0$	3 days ahead	RMSE
8		$\alpha = 0,\ \beta = 1$	3 days ahead	RMSE
9		$\alpha = 0,\ \beta = 0$	7 days ahead	MIN-VAR
10	Large	$\alpha = 0,\ \beta = 1$	7 days ahead	MIN-VAR
11	(300 agents)	$\alpha = 0,\ \beta = 0$	7 days ahead	RMSE
12		$\alpha = 0,\ \beta = 1$	7 days ahead	RMSE

Table 5
Rate of change for dynamics in I-EPOS and DIAS live experiments.

Service/Parameters	Intensity/Rate		
	Low	Medium	High
I-EPOS			
Plan change	10%	20%	50%
α and β change	10%	20%	50%
Global cost function (System-wide)	10%	20%	50%
Agent join/leave	10%	20%	50%
DIAS			
Change of possible states	3h	2h	1h
Change of selected state	5′	2′	1′
Agent join/leave	10′	5′	2′

where $g^t_{s,i}$ and $g^t_{l,i}$ are the global costs of profile i at iteration t in simulation and live settings, respectively. Similarly, $l^t_{s,i}$ and $l^t_{l,i}$ are the average local costs of all agent in profile i at iteration t in simulation and live settings, respectively.

6.1.2. DIAS

These experiments are based on the GDELT (*Global Dataset of Events, Languages, and Tone*) platform.[16] GDELT monitors and captures print/broadcast/web-based global news media in real-time. Its data can be accessed via an API in 15-min intervals. This paper employs the DIAS-GDELT demonstrator [73].[17] It fetches GDELT news updates every 15 min, extracts the possible states, and sends them to the application agents. DIAS agents (service agents) are mapped to 28 application agents, each representing a country from GDELT. Each DIAS agent receives the number of news generated during the last 15 min from the application agent, disseminates them in the network, and receives the aggregated total number of news generated by the other agents.

6.2. Evaluation scenario II: Handling system dynamics

In this evaluation scenario, a set of continuous real-world experiments between 24/11–24/12/2020 are performed to study the performance of both services under complex and dynamic environments. Each day is divided into three 8-h time periods: low, medium, and high intensity, each imposing different rate of change for system dynamics.

6.2.1. I-EPOS

The experimental settings for this scenario are shown in Table 3. The I-EPOS service is initialized with 200 agents, and each agent is randomly assigned to one of the 2779 EVs from the EV dataset with 7-days-ahead planning horizon. During runtime, four dynamics are adopted, each corresponding to a change in system settings: (i) Agents joining/leaving, (ii) local plan change, (iii) α, β (weight) change, and (iv) global cost function change. The rate of change for each dynamic varies across the intensity periods (Table 5), with the high-intensity period incurring the highest number of changes. For example, in the low-intensity period at the end of each run[18] agents change their plans with 5% probability. The rate of change for α and β operates the same way, however, the change in the global cost function is applied system-wide. The effect of such dynamic changes on the performance of I-EPOS is studied using two metrics: (i) The latency indicates the variation of the I-EPOS execution time with varying dynamics, with respect to non-changing dynamics [74]. The execution time is defined as the time it take (in milliseconds) for I-EPOS to complete 50 iterations, plus applying changes enforced by the dynamics (e.g., agents join/leave, changes in plans, or α, β values). The latency is calculated as follows:

$$Latency := \frac{Varying\ Dynamics\ Execution\ Time}{Non\text{-}changing\ Dynamics\ Execution\ Time} \tag{3}$$

(ii) WAT, which indicates if the system spends excessive time adapting to dynamic changes with respect to performing service-related task. The WAT is calculated as follows:

$$WAT := \frac{Working\ time}{Adaptivity\ time} \tag{4}$$

where the working time concerns the time (in milliseconds) required to execute the 50 learning iterations of I-EPOS, while the adaptivity time concerns the time required to adapt to dynamic changes [74]. For instance, adapting to changes in the number of agents, which triggers the self-reorganization of the tree-topology.

6.2.2. DIAS

At the start of each day, DIAS is initialized with 20 agents. During runtime, three different dynamics are adopted, each corresponding to a change in the system settings: (i) Agents joining/leaving, (ii) change in the set of possible states, and (iii) change in the selected state. Every time an agent changes its set of possible states, it randomly selects 9 numbers between the current time and the next hour. For instance, the possible states for an agent at 14:00 (1400) is 9 numbers in the range of [1400, 1500]. The rate of changes for each dynamic varies across the intensity periods (Table 5), with the high-intensity period incurring the highest number of changes on the system. For example, in the low-intensity period, each DIAS agent changes its selected state every 5 min. The agent join/leave rate means that, in the high-intensity case, all agents leave the network every 2 min, and return 2 min later.

6.3. Deployment infrastructure

The deployment infrastructure of the testbed is as follows: one server with higher computational power for scaling, and a less powerful server for long-running experiments. Both servers provide 'bare metal' access, which is substantially faster than using virtual images. The larger machine has the following specifications: Intel Xeon hex-core 3.50 GHz 256 GB DDR3 RAM, 2 TB Raid 1 storage, Ubuntu 16.04. As for the smaller machine: Intel Core i7-6700 Quad-Core, 64 GB DDR4 RAM, 1 TB storage space, Ubuntu 16.04. Each service agent is implemented using

16 https://www.gdeltproject.org [Last accessed: May 2020].
17 http://dias-net.org/dias-gdelt-live/ [Last accessed: May 2020].
18 Each run refers to the completion of 50 learning iterations by I-EPOS.

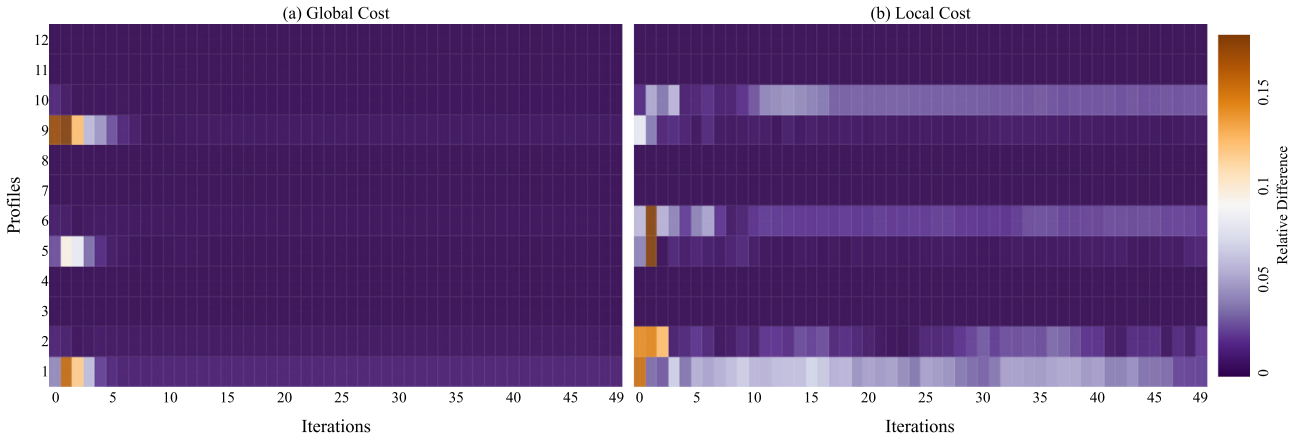


Fig. 4. Relative difference in global and local cost between the simulation and live settings, calculated based on Eq. (2). The value of each cell shows the mean across 100 repeated experiments for the given profile. The low values confirm that utilizing the conceptual architecture and the Livepeer toolkit, I-EPOS can transition from simulation to live with minimal introduced error.

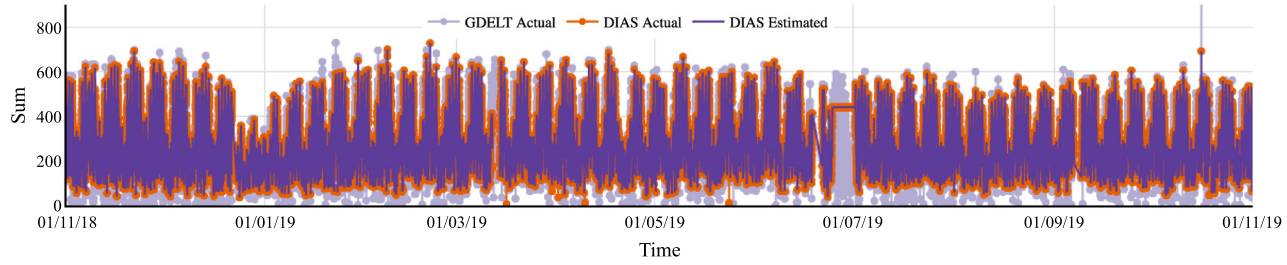


Fig. 5. (i) GDELT Actual: baseline values extracted from GDELT (i.e., total number of news items generated by the 28 countries) (ii) DIAS Actual: sum of selected states from each of the 28 DIAS agents, based on the set of possible states for each agent. (iii) DIAS Estimated: the estimated total number of news items by all countries, calculated by averaging the estimation of each agent. DIAS can accurately estimate the actual GDELT events in the long term.

the Livepeer toolkit (Section 4) as a separate JVM object.[19] The logging gateway is a single persistence daemon, that creates a single connection to the database (PostgreSQL 10.6) and has a predefined commit rate and queue size that can be adapted based on system scale. It listens to ZeroMQ messages with logging information sent by the agents, and commits the information to the database. Each agent notifies the logging gateway of the required relations, tables, and indices, in the form of SQL query templates created on the database. The communication between all agents is based on message passing, implemented based on the ZeroMQ library.

7. Experimental results

7.1. Evaluation scenario I: Comparing accuracy in non-volatile environments

This section illustrates the results of the experiments based on the methodology introduces in Section 6.1.

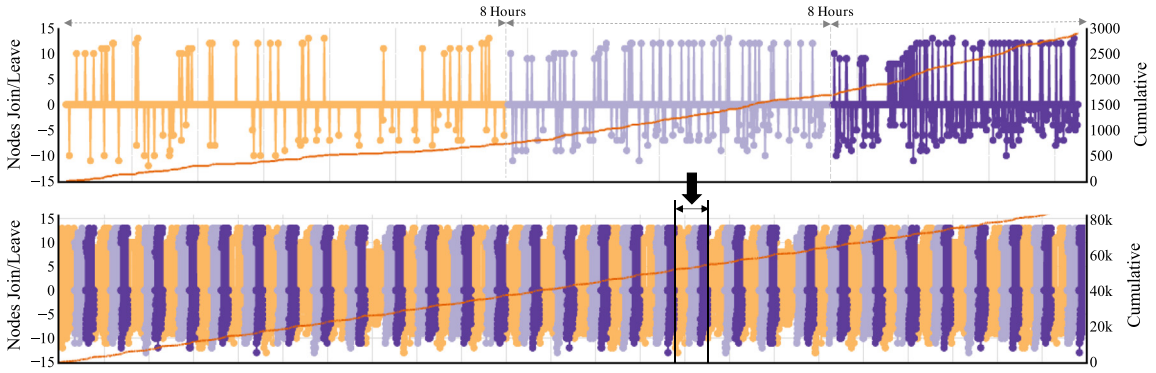
7.1.1. I-EPOS

Fig. 4 shows the relative difference in global and local cost between the simulation and live environments across the I-EPOS iterations. For each profile, the global cost is calculated based on the corresponding global cost function (MIN-VAR/MIN-RMSE) in Table 4, and the local cost is the average discomfort of agents' selected plans, calculated by the likelihood of using the EV while it is charging [72]. The value of each cell in Fig. 4 shows the mean across 100 repeated experiments for the given profile. For each experiment, I-EPOS initializes random trees and assigns the agents to the nodes. This random assignment generates small variations in the I-EPOS outcome [75]. However, the general trend across all profiles shows that as the number of learning iterations progresses to the final iteration (50), the global and average local costs of the simulation and live environments converge. After 10 learning iterations, the relative difference in global cost for all profiles is less than 0.02, and the highest difference in average local cost at the final iteration is 0.0256 related to Profile 10. This confirms that the I-EPOS transition from simulation to live based on the Livepeer toolkit, can be performed with minimal introduced inaccuracies.

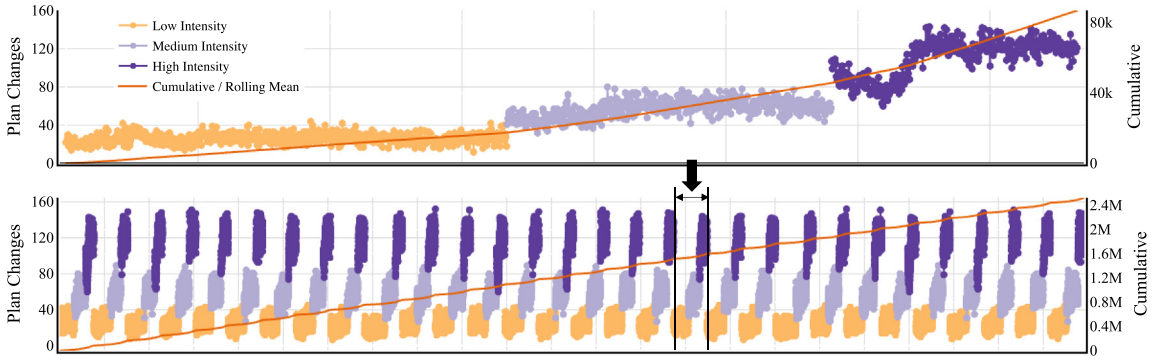
7.1.2. DIAS

Fig. 5 outlines three time-series based on the experimental methodology introduced in Section 6.1.2: (i) GDELT Actual: the raw baseline values extracted from GDELT, representing the total number of news items generated by the 28 countries. (ii) DIAS Actual: the sum of selected states from each of the 28 DIAS agents. Each selected state is the number of generated news items by the assigned country. The set of possible states is extracted by a sliding a window of 27 observations, uniformly sampling 9 values. (iii) DIAS Estimated: The estimated total number of news items by all countries (estimated DIAS actual), calculated by averaging the estimates of each agent. This estimation is what each DIAS agent calculates as the true value for the DIAS actual. The accuracy of this estimation is affected by various factors, such as the sampling pool size of data suppliers, convergence time, and the selected state changes. This experiment has been running since November 1st, 2018. As illustrated in Fig. 5, DIAS service

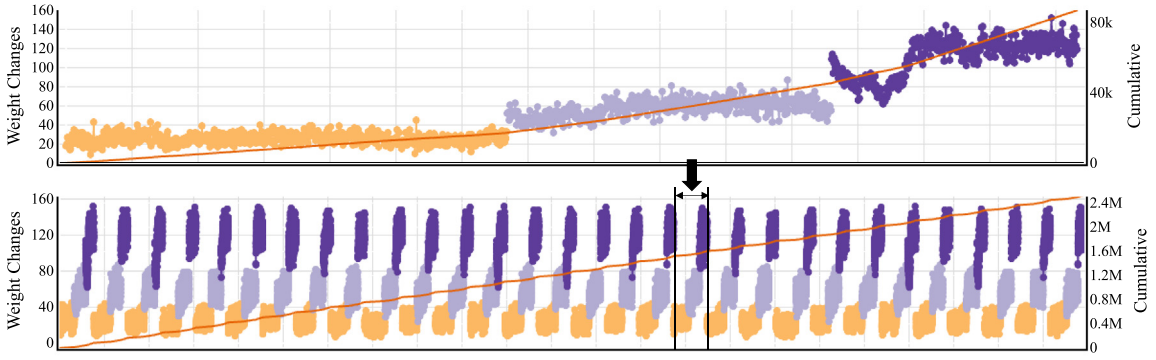
[19] By using this approach, in principal agents can run on a different machines and networks.



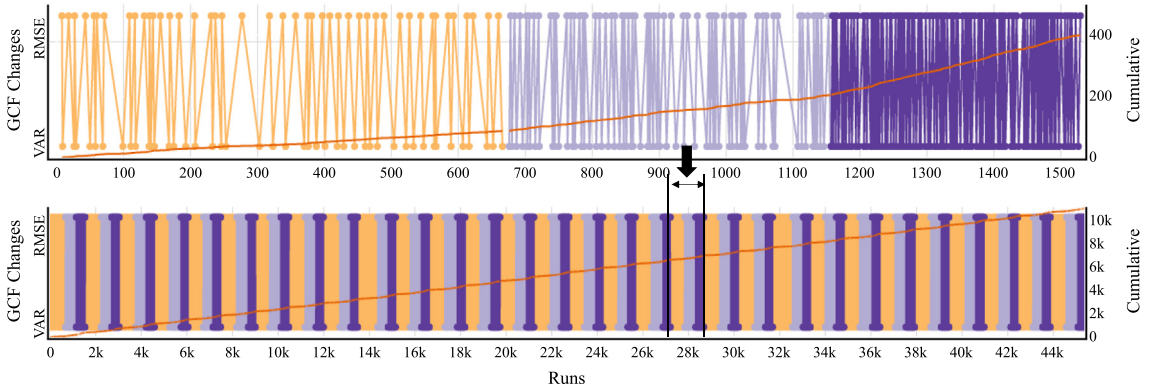
(a) Nodes Join/Leave - upper figure: snapshot from December 12th, 2019, and lower figure: 30-days live operations



(b) Plan Changes - upper figure: snapshot from December 12th, 2019, and lower figure: 30-days live operations

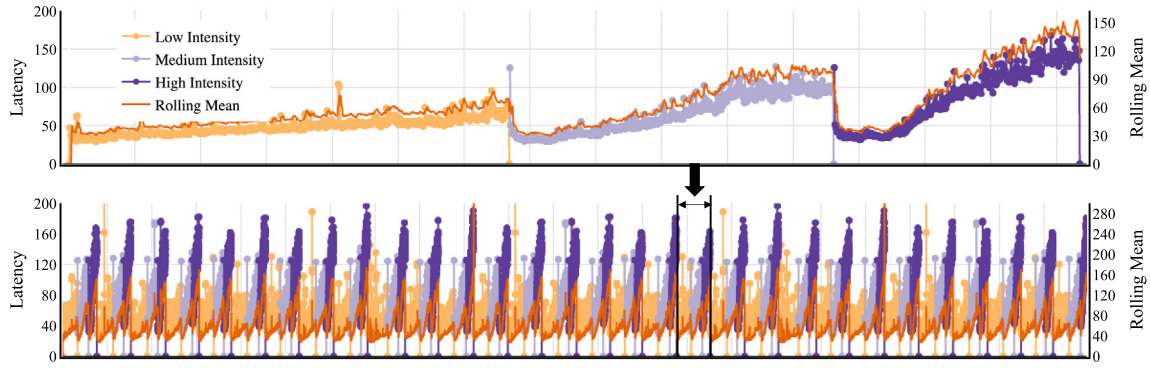


(c) Weight Changes - upper figure: snapshot from December 12th, 2019, and lower figure: 30-days live operations

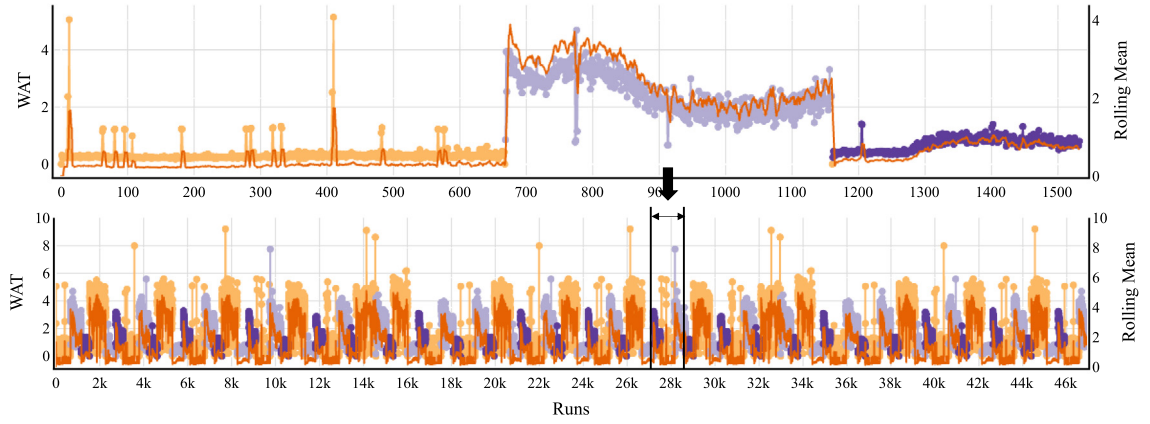


(d) GCF Changes - upper figure: snapshot from December 12th, 2019, and lower figure: 30-days live operations

Fig. 6. EPOS live operations between 24/11–24/12/2020. Each run refers to completion of 50 learning iterations by I-EPOS, and GCF denotes the changes in the global cost function, as a system-wide parameter. For each pair, the upper figure shows the snapshot of operations on December 12th, 2019, while the lower figure illustrates the operations over the month-long experiments.



(a) Latency - upper figure: snapshot from December 12th, 2019, and lower figure: 30-days live operations



(b) WAT - upper figure: snapshot from December 12th, 2019, and lower figure: 30-days live operations

Fig. 7. EPOS live operations between 24/11–24/12/2020, with snapshot on December 12th, 2019. Latency indicates the ratio between I-EPOS runtime given varying dynamics with respect to runtime with static non-changing dynamics (Eq. (3)). WAT indicates if the system is spending too much time adapting to dynamic changes rather than performing the I-EPOS learning iterations (Eq. (4)). Note that due to lower WAT (higher adaptivity time), I-EPOS manages to complete fewer runs in higher intensity period, during the same time-frame.

based on the Livepeer toolkit can perform long-running operation and accurate estimations of the GDELT baseline. It can rapidly adapt to sudden changes.

7.2. Evaluation scenario II: Handling system dynamics

This section illustrates results of the experiments based on the methodology introduced in Section 6.2.

7.2.1. I-EPOS

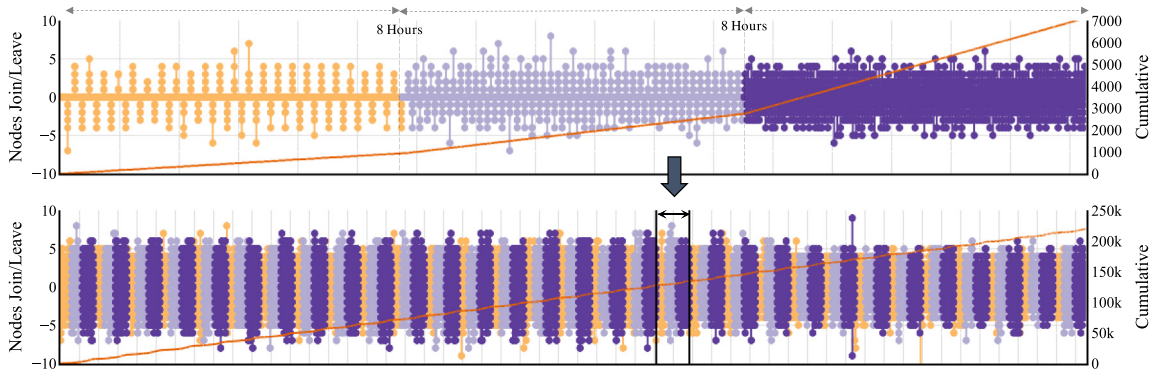
These experiments were continuously executed between 24/11–24/12/2019, with the intensity setting changing every 8 h. Figs. 6 and 7 illustrate a snapshot of I-EPOS live operation during December 12th, 2019, as well as the live operation during the month-long experiments. During a typical day (over the month-long period), I-EPOS live handles approximately 80.000 changes in agents' plans (2.4 million), 80.000 changes in α, β parameters (2.4 million), 3000 agents joining/leaving (80,000), as well as 400 changes in the global cost function (10,000). Fig. 7a shows the latency of the I-EPOS across different intensity periods. On average, the latency increases by 27% from low to medium, and 102% from medium to high. Fig. 7b shows the WAT in different intensity settings, where the average WAT is always higher than 1. Generally, if the ratio is less than one, the system is spending a lot of time adapting to changes [74]. The above experiments confirm that even under highly dynamic environments, I-EPOS completes its learning iterations without any crashes/failures, and delivers the learning outcome.

7.2.2. DIAS

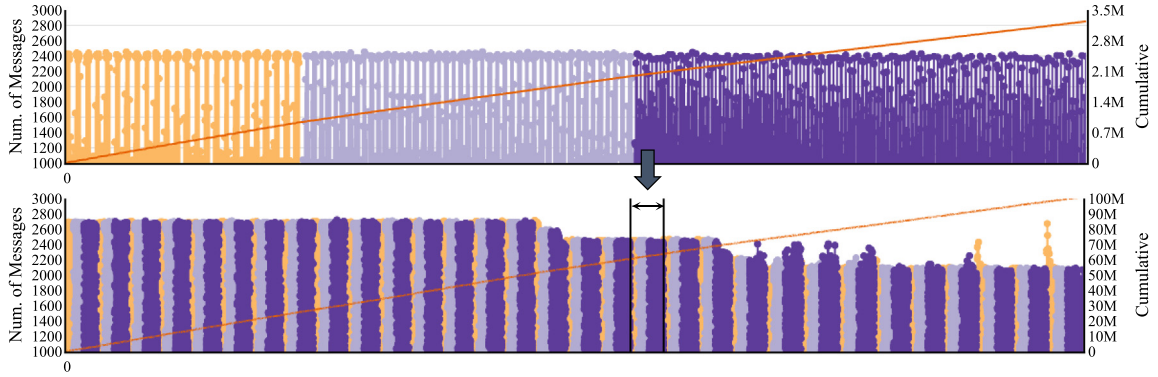
The experiments are continuously executed between 24/11–24/12/2019, with changing intensity settings every 8 h. Figs. 8a to d, illustrate a snapshot of DIAS live operation during December 12th, 2019, as well as the live operation during the month-long experiments. During a typical day (over the month-long period), DIAS handles approximately 4000 agent joins/leaves (250,000), 16,000 state changes (480,000), and 2 million exchanges of messages (100 million). The estimated sum of the selected states of all DIAS agents is shown in Fig. 8c. As shown, even under intense dynamic changes, the DIAS live still provides accurate estimations. Finally, Fig. 8d illustrates the overall DIAS error, calculated as the difference between true sum (raw data) and the estimated sum. This error is caused by various factors, such as summarization (raw values to the set of possible states), rapid state changes, agent joining/leaving, and convergence time. As shown, this error increases with the rise in intensity, however due to quick dissemination of state changes and convergence in the network, the rolling mean error is low.

8. Conclusion and future work

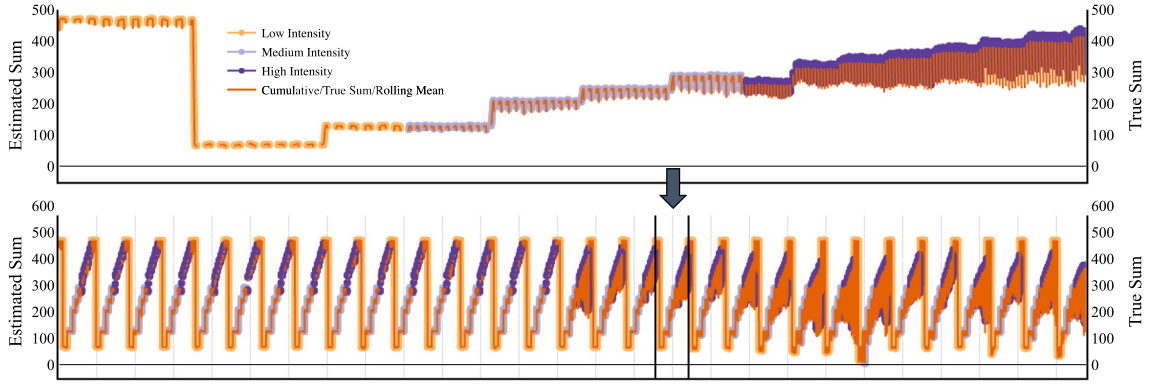
This paper introduces a novel IoT testbed architecture for decentralized socio-technical services and applications running on IoT. This architecture applies two layers of abstraction on both the IoT application (devices), and the decentralized services, enabling a dual self-integration capability: (i) an IoT application integrating several application-independent and modular decentralized services, and (ii) a decentralized service integrates to several IoT



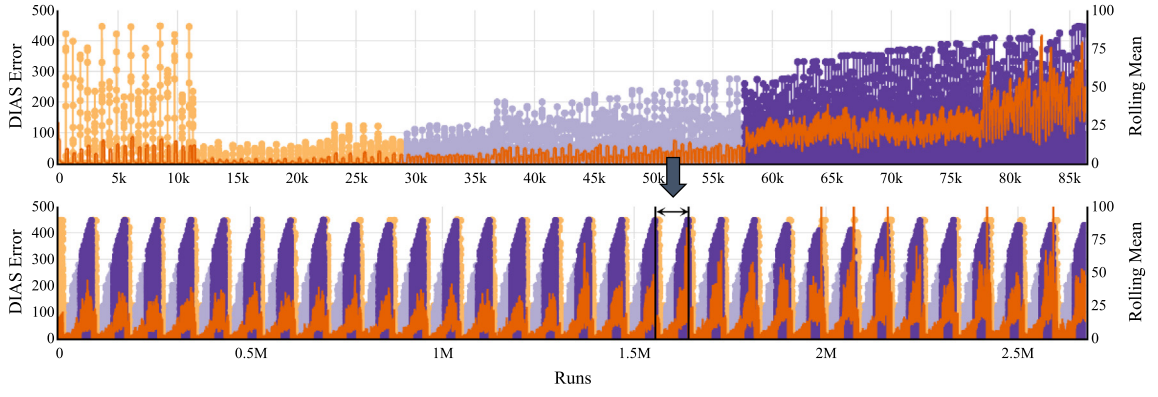
(a) Nodes Join/Leave - upper figure: snapshot from December 12th, 2019, and lower figure: 30-days live operations



(b) Number of Messages - upper figure: snapshot from December 12th, 2019, and lower figure: 30-days live operations



(c) Estimated Sum - upper figure: snapshot from December 12th, 2019, and lower figure: 30-days live operations



(d) DIAS Error- upper figure: snapshot from December 12th, 2019, and lower figure: 30-days live operations

Fig. 8. DIAS live operations between 24/11–24/12/2020. For each pair, the upper figure shows the snapshot of operations on December 12th, 2019, while the lower figure illustrates the operations over the month-long experiments. The initial burst in DIAS error (Fig. 8c) is due to the change in set of possible states at midnight. The estimated sum is calculated by averaging the estimation of each agent, indicating the value each DIAS agent estimates as the true sum of selected state for all DIAS agents. DIAS error is the difference between true sum (raw data) and the estimated sum. While this error increases with the rise in intensity, due to quick dissemination and convergence, the rolling mean remains low.

applications without, changing the implementation of the service. A distributed communication protocol is designed to realize and operationalize the conceptual architecture, providing common interfaces and the communication logic required for self-integration of applications and services at runtime. Additionally, this paper contributes the Livepeer toolkit, providing a general purpose IoT prototyping toolkit for rapid design and testing of decentralized socio-technical applications, as well as facilitating the transition from simulation to live environments. Experimental evaluations on two decentralized IoT services, performed under highly dynamic environments confirm the efficiency, and robustness of the testbed architecture.

This work promises new instruments for prototyping and developing decentralized socio-technical services running on IoT, and pathways to manage their complexity, which so far have hindered value-oriented self-management approaches in Smart Cities. Ultimately, the architecture and toolkit will be able to facilitate pilot tests in Smart City use-cases. Future research can address the inclusion of other decentralized services with different requirements and network structures, device/agent mobility, and semantic service composition. Lastly, further deployments in larger-scale infrastructures (e.g., PlanetLab[20]) can provide new insights about the applicability of the proposed architecture.

Artifacts & reusability

To facilitate the reusability of the testbed and Livepeer toolkit by the community, the code bases, protocols, and the documentations are made openly available: Simulation and live versions of I-EPOS[21], Simulation and live versions of DIAS[22], monitoring infrastructure[23] and its documentation[24], Livepeer[25] and its documentation[26], and IoT device/application agents[27] are all available for the community.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by the ERC Advanced Grant (324247) Momentum, and the Engineering Social Technologies for a Responsible Digital Future Project at ETH Zurich and TU Delft. Authors would also like to thank Renato Kunz for his contributions in development and implementation of the testbed.

References

[1] H. Arasteh, V. Hosseinnezhad, V. Loia, A. Tommasetti, O. Troisi, M. Shafie-Khah, P. Siano, Iot-based smart cities: a survey, in: 2016 IEEE 16th International Conference on Environment and Electrical Engineering, EEEIC, IEEE, 2016, pp. 1–6.

[2] M. Chernyshev, Z. Baig, O. Bello, S. Zeadally, Internet of Things (IoT): Research, simulators, and testbeds, IEEE Internet Things J. 5 (3) (2017) 1637–1647.

[3] R. Girau, S. Martis, L. Atzori, Lysis: A platform for IoT distributed applications over socially connected objects, IEEE Internet Things J. 4 (1) (2016) 40–51.

[4] K. Bellman, J. Botev, A. Diaconescu, L. Esterle, C. Gruhl, C. Landauer, P.R. Lewis, A. Stein, S. Tomforde, R.P. Würtz, Self-improving system integration-status and challenges after five years of SISSY, in: 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems, FAS* W, IEEE, 2018, pp. 160–167.

[5] K.L. Bellman, C. Gruhl, C. Landauer, S. Tomforde, Self-improving system integration-on a definition and characteristics of the challenge, in: 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems, FAS* W, IEEE, 2019, pp. 1–3.

[6] S. Tomforde, M. Goller, To adapt or not to adapt: A quantification technique for measuring an expected degree of self-adaptation, Computers 9 (1) (2020) 21.

[7] E. Pournaras, M. Yao, D. Helbing, Self-regulating supply–demand systems, Future Gener. Comput. Syst. 76 (2017) 73–91.

[8] E. Pournaras, P. Pilgerstorfer, T. Asikis, Decentralized collective learning for self-managed sharing economies, ACM Trans. Auton. Adapt. Syst. (TAAS) 13 (2) (2018) 10.

[9] E. Pournaras, J. Nikolic, A. Omerzel, D. Helbing, Engineering democratization in internet of things data analytics, in: 2017 IEEE 31st International Conference on Advanced Information Networking and Applications, AINA, IEEE, 2017, pp. 994–1003.

[10] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, T. Razafindralambo, A survey on facilities for experimental internet of things research, 2011.

[11] N. Zhang, W. Zhao, Distributed privacy preserving information sharing, in: Proceedings of the 31st International Conference on Very Large Data Bases, Citeseer, 2005, pp. 889–900.

[12] S. Mahboubi, R. Akbarinia, P. Valduriez, Privacy-preserving top-k query processing in distributed systems, in: Transactions on Large-Scale Data-and Knowledge-Centered Systems XLII, Springer, 2019, pp. 1–24.

[13] A. Preece, Asking 'Why'in AI: Explainability of intelligent systems–perspectives and challenges, Intell. Syst. Account. Financ. Manage. 25 (2) (2018) 63–72.

[14] A. Adadi, M. Berrada, Peeking inside the black-box: A survey on explainable artificial intelligence (XAI), IEEE Access 6 (2018) 52138–52160.

[15] Y. Huang, G. Poderi, S. Šćepanović, H. Hasselqvist, M. Warnier, F. Brazier, Embedding Internet-of-Things in large-scale socio-technical systems: A community-oriented design in future smart grids, in: The Internet of Things for Smart Urban Ecosystems, Springer, 2019, pp. 125–150.

[16] F. Fanitabasi, E. Pournaras, Appliance-level flexible scheduling for socio-technical smart grid optimisation, IEEE Access (2020).

[17] G. Fortino, W. Russo, C. Savaglio, W. Shen, M. Zhou, Agent-oriented cooperative smart objects: From IoT system design to implementation, IEEE Trans. Syst. Man Cybern.: Syst. 48 (11) (2017) 1939–1956.

[18] W. Galuba, K. Aberer, Z. Despotovic, W. Kellerer, ProtoPeer: a P2P toolkit bridging the gap between simulation and live deployement, in: Proceedings of the 2nd International Conference on Simulation Tools and Techniques, ICST (Institute for Computer Sciences, and Social-Informatics, 2009, p. 60.

[19] I. Gerostathopoulos, E. Pournaras, TRAPPed in traffic? A self-adaptive framework for decentralized traffic optimization, in: 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS, IEEE, 2019, pp. 32–38.

[20] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, et al., FIT IoT-LAB: A large scale open experimental IoT testbed, in: 2015 IEEE 2nd World Forum on Internet of Things, WF-IoT, IEEE, 2015, pp. 459–464.

[21] L. Sanchez, L. Muñoz, J.A. Galache, P. Sotres, J.R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis, et al., SmartSantander: IoT experimentation over a smart city testbed, Comput. Netw. 61 (2014) 217–238.

[22] S. Latre, P. Leroux, T. Coenen, B. Braem, P. Ballon, P. Demeester, City of things: An integrated and multi-technology testbed for IoT smart city experiments, in: 2016 IEEE International Smart Cities Conference, ISC2, IEEE, 2016, pp. 1–8.

[23] J. Struye, B. Braem, S. Latré, J. Marquez-Barja, The CityLab testbed—Large-scale multi-technology wireless experimentation in a city environment: Neural network-based interference prediction in a smart city, in: IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS, IEEE, 2018, pp. 529–534.

[24] M. Nati, A. Gluhak, H. Abangar, W. Headley, Smartcampus: A user-centric testbed for internet of things experimentation, in: 2013 16th International Symposium on Wireless Personal Multimedia Communications, WPMC, IEEE, 2013, pp. 1–6.

20 https://www.planet-lab.org/about
21 https://github.com/epournaras/EPOS
22 https://github.com/epournaras/DIAS-Development
23 https://github.com/epournaras/Livelog
24 https://github.com/epournaras/Livelog-Documentation
25 https://github.com/epournaras/Livepeer
26 https://github.com/epournaras/Livepeer-Documentation
27 https://github.com/epournaras/DIASClient

[25] J. Jiang, R. Pozza, N. Gilbert, K. Moessner, Makesense: An IoT testbed for social research of indoor activities, 2019, arXiv preprint arXiv:1908.03380.
[26] A. Cimmino, V. Oravec, F. Serena, P. Kostelnik, M. Poveda-Villalón, A. Tryferidis, R. García-Castro, S. Vanya, D. Tzovaras, C. Grimm, VICINITY: IoT semantic interoperability based on the web of things, in: 2019 15th International Conference on Distributed Computing in Sensor Systems, DCOSS, IEEE, 2019, pp. 241–247.
[27] M.M. Hossain, S. Al Noor, Y. Karim, R. Hasan, IoTbed: A generic architecture for testbed as a service for internet of things-based systems, in: ICIOT, 2017, pp. 42–49.
[28] N. Sinha, K.E. Pujitha, J.S.R. Alex, Xively based sensing and monitoring system for IoT, in: 2015 International Conference on Computer Communication and Informatics, ICCCI, IEEE, 2015, pp. 1–6.
[29] A.M. Mzahm, M.S. Ahmad, A.Y. Tang, Agents of Things (AoT): An intelligent operational concept of the Internet of Things (IoT), in: 2013 13th International Conference on Intellient Systems Design and Applications, IEEE, 2013, pp. 159–164.
[30] L. Atzori, A. Iera, G. Morabito, Siot: Giving a social structure to the internet of things, IEEE Commun. Lett. 15 (11) (2011) 1193–1195.
[31] F. Cicirelli, A. Guerrieri, G. Spezzano, A. Vinci, O. Briante, G. Ruggeri, iSapiens: A platform for social and pervasive smart environments, in: 2016 IEEE 3rd World Forum on Internet of Things, WF-IoT, IEEE, 2016, pp. 365–370.
[32] M. Pipattanasomporn, M. Kuzlu, W. Khamphanchai, A. Saha, K. Rathinavel, S. Rahman, BEMOSS: An agent platform to facilitate grid-interactive building operation with IoT devices, in: 2015 IEEE Innovative Smart Grid Technologies-Asia, ISGT ASIA, IEEE, 2015, pp. 1–6.
[33] V.-M. Scuturici, S. Surdu, Y. Gripay, J.-M. Petit, UbiWare: Web-based dynamic data & service management platform for AmI, in: Proceedings of the Posters and Demo Track, 2012, pp. 1–2.
[34] N.M. do Nascimento, C.J.P. de Lucena, Fiot: An agent-based framework for self-adaptive and self-organizing applications based on the internet of things, Inform. Sci. 378 (2017) 161–176.
[35] L. Luceri, F. Cardoso, M. Papandrea, S. Giordano, J. Buwaya, S. Kundig, C.M. Angelopoulos, J. Rolim, Z. Zhao, J.L. Carrera, et al., VIVO: A secure, privacy-preserving, and real-time crowd-sensing framework for the Internet of Things, Pervasive Mob. Comput. 49 (2018) 126–138.
[36] P. Vlacheas, R. Giaffreda, V. Stavroulaki, D. Kelaidonis, V. Foteinos, G. Poulios, P. Demestichas, A. Somov, A.R. Biswas, K. Moessner, Enabling smart cities through a cognitive management framework for the internet of things, IEEE Commun. Mag. 51 (6) (2013) 102–111.
[37] F. Zambonelli, M. Viroli, G. Fortino, B. Re, Towards adaptive flow programming for the IoT: The fluidware approach, in: 2019 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops, IEEE, 2019, pp. 549–554.
[38] C. Savaglio, M. Ganzha, M. Paprzycki, C. Bădică, M. Ivanović, G. Fortino, Agent-based Internet of Things: State-of-the-art and research challenges, Future Gener. Comput. Syst. 102 (2020) 1038–1053.
[39] P. Novoa-Hernández, C.C. Corona, D.A. Pelta, Self-adaptation in dynamic environments-a survey and open issues, Int. J. Bio-Inspir. Comput. 8 (1) (2016) 1–13.
[40] C. Krupitzer, F.M. Roth, S. VanSyckel, G. Schiele, C. Becker, A survey on engineering approaches for self-adaptive systems, Pervasive Mob. Comput. 17 (2015) 184–206.
[41] L. Baresi, L. Pasquale, Live goals for adaptive service compositions, in: Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, 2010, pp. 114–123.
[42] A. Urbieta, A. González-Beltrán, S.B. Mokhtar, M.A. Hossain, L. Capra, Adaptive and context-aware service composition for IoT-based smart cities, Future Gener. Comput. Syst. 76 (2017) 262–274.
[43] K. Geihs, M. Wagner, Context-awareness for self-adaptive applications in ubiquitous computing environments, in: International Conference on Context-Aware Systems and Applications, Springer, 2012, pp. 108–120.
[44] L. Barakat, S. Miles, M. Luck, Adaptive composition in dynamic service environments, Future Gener. Comput. Syst. 80 (2018) 215–228.
[45] Y. Brun, R. Desmarais, K. Geihs, M. Litoiu, A. Lopes, M. Shaw, M. Smit, A design space for self-adaptive systems, in: Software Engineering for Self-Adaptive Systems II, Springer, 2013, pp. 33–50.
[46] A. Ferscha, Collective adaptive systems, in: Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers, 2015, pp. 893–895.
[47] S.K. Y.R., H. Champa, An extensive review on sensing as a service paradigm in IoT: architecture, research challenges, lessons learned and future directions, Int. J. Appl. Eng. Res. 14 (6) (2019) 1220–1243.
[48] S. Sotiriadis, N. Bessis, E. Asimakopoulou, N. Mustafee, Towards simulating the internet of things, in: 2014 28th International Conference on Advanced Information Networking and Applications Workshops, IEEE, 2014, pp. 444–448.
[49] J.O. Kephart, D.M. Chess, The vision of autonomic computing, Computer 36 (1) (2003) 41–50.
[50] C. Landauer, K. Bellman, Designing cooperating self-improving systems, in: 2015 IEEE International Conference on Autonomic Computing, IEEE, 2015, pp. 273–278.
[51] A. Diaconescu, B. Porter, R. Rodrigues, E. Pournaras, Hierarchical self-awareness and authority for scalable self-integrating systems, in: 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems, FAS* W, IEEE, 2018, pp. 168–175.
[52] S. Tomforde, S. Rudolph, K. Bellman, R. Würtz, An organic computing perspective on self-improving system interweaving at runtime, in: 2016 IEEE International Conference on Autonomic Computing, ICAC, IEEE, 2016, pp. 276–284.
[53] C.M. Barnes, K. Bellman, J. Botev, A. Diaconescu, L. Esterle, C. Gruhl, C. Landauer, P.R. Lewis, P.R. Nelson, A. Stein, et al., CHARIOT-towards a continuous high-level adaptive runtime integration testbed, in: 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems, FAS* W, IEEE, 2019, pp. 52–55.
[54] S.D.T. Kelly, N.K. Suryadevara, S.C. Mukhopadhyay, Towards the implementation of IoT for environmental condition monitoring in homes, IEEE Sens. J. 13 (10) (2013) 3846–3853.
[55] U.S. Shanthamallu, A. Spanias, C. Tepedelenlioglu, M. Stanley, A brief survey of machine learning methods and their sensor and IoT applications, in: 2017 8th International Conference on Information, Intelligence, Systems & Applications, IISA, IEEE, 2017, pp. 1–8.
[56] A. Bielenberg, L. Helm, A. Gentilucci, D. Stefanescu, H. Zhang, The growth of diaspora-a decentralized online social network in the wild, in: 2012 Proceedings IEEE INFOCOM Workshops, IEEE, 2012, pp. 13–18.
[57] S. Tilak, Real-world deployments of participatory sensing applications: Current trends and future directions, ISRN Sens. Netw. 2013 (2013).
[58] J. Shah, B. Mishra, IoT enabled environmental monitoring system for smart cities, in: 2016 International Conference on Internet of Things and Applications, IOTA, IEEE, 2016, pp. 383–388.
[59] F. Calabrese, L. Ferrari, V.D. Blondel, Urban sensing using mobile phone network data: a survey of research, Acm Comput. Surv. (csur) 47 (2) (2015) 25.
[60] A. Medvedev, P. Fedchenkov, A. Zaslavsky, T. Anagnostopoulos, S. Khoruzhnikov, Waste management as an IoT-enabled service in smart cities, in: Internet of Things, Smart Spaces, and Next Generation Networks and Systems, Springer, 2015, pp. 104–115.
[61] S.E. Collier, The emerging enernet: Convergence of the smart grid with the internet of things, IEEE Ind. Appl. Mag. 23 (2) (2016) 12–16.
[62] L. Wu, X. Yue, A. Jin, D.C. Yen, Smart supply chain management: a review and implications for future research, Int. J. Logist. Manage. 27 (2) (2016) 395–417.
[63] E. Pournaras, Multi-level reconfigurable self-organization in overlay services, Ph.D. dissertation, TU Delft, Delft, The Netherlands, 2013.
[64] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, M. Van Steen, The peer sampling service: Experimental evaluation of unstructured gossip-based implementations, in: ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, Springer, 2004, pp. 79–98.
[65] Q. Wang, W.U. Hassan, A. Bates, C. Gunter, Fear and logging in the internet of things, in: Network and Distributed Systems Symposium, 2018.
[66] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, ACM, 2012, pp. 13–16.
[67] H. Cai, B. Xu, L. Jiang, A.V. Vasilakos, IoT-based big data storage systems in cloud computing: perspectives and challenges, IEEE Internet Things J. 4 (1) (2016) 75–87.
[68] D. Trihinas, G. Pallis, M. Dikaiakos, Low-cost adaptive monitoring techniques for the internet of things, IEEE Trans. Serv. Comput. (2018).
[69] E. Pournaras, M. Vasirani, R.E. Kooij, K. Aberer, Decentralized planning of energy demand for the management of robustness and discomfort, IEEE Trans. Ind. Inf. 10 (4) (2014) 2280–2289.
[70] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, M. Van Steen, Gossip-based peer sampling, ACM Trans. Comput. Syst. (TOCS) 25 (3) (2007) 8.
[71] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun. ACM 13 (7) (1970) 422–426.
[72] E. Pournaras, S. Jung, S. Yadhunathan, H. Zhang, X. Fang, Socio-technical smart grid optimization via decentralized charge control of electric vehicles, Appl. Soft Comput. 82 (2019) 105573.

[73] E. Pournaras, E. Gaere, R. Kunz, A.N. Ghulam, Democratizing data analytics: Crowd-sourcing decentralized collective measurements, in: 13th International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2019, IEEE, 2019.
[74] E. Kaddoum, C. Raibulet, J.-P. Georgé, G. Picard, M.-P. Gleizes, Criteria for the evaluation of self-* systems, in: Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, ACM, 2010, pp. 29–38.
[75] J. Nikolic, E. Pournaras, Structural self-adaptation for decentralized pervasive intelligence, in: 2019 22nd Euromicro Conference on Digital System Design, DSD, IEEE, 2019, pp. 562–571.

Farzam Fanitabasi is a Ph.D. candidate at the Chair of Computational Social Science in ETH Zurich. His background is in computer science and information systems, and holds M.Sc. in information technology engineering from Sharif University of Technology, Iran 2015. His research interests include: (i) combinatorial multi-objective optimization in multi-agent systems, (ii) collaborative decision making for demand response mechanisms in Smart Grids, and (iii) fog computing for geo-spatial and temporal hierarchical distributed environments.

Edward Gaere has over 20 years of experience in applied analytics, with a specialization in time-series analysis. With a team of researchers, he designed novel quantitative models with self-learning functionality, that have proved to be highly successful in the area of short-term electronic trading, bridging some of the theoretical gaps between dynamical models and static machine learning techniques. He is currently a Senior Data Analytics Manager at Sunrise Communications in Switzerland, working on data science related to customer experience, and is also a Ph.D. student at ETH Zurich in the field of machine learning and the automation of business decision taking in the industry.

Dr. Evangelos Pournaras is an Associate Professor at University of Leeds. He has more than 5 years research experience at ETH Zurich in Switzerland after having completed his Ph.D. at Delft University of Technology. Evangelos has been a visiting researcher at EPFL in Switzerland and IBM T.J. Watson Research Center in the USA. Evangelos won the Augmented Democracy Prize, the 1st prize at ETH Policy Challenge as well as 4 paper awards and honors. He has published more than 50 peer-reviewed papers in the area of distributed and intelligent social computing systems with expertise in Smart Cities and Smart Grids.