LEDGER
ledgerjournal.org

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2022.230

RESEARCH ARTICLE

HLF-Kubed: Blockchain-Based Resource Monitoring for Edge Clusters

Achilleas Tzenetopoulos,[*] Dimosthenis Masouros, Nikolaos Kapsoulis,[†] Antonios Litke,[‡] Dimitrios Soudris, Theodora Varvarigou[§]

Abstract. In the past several years, there has been an increased usage of smart, always-connected devices at the edge of the network, which provide real-time contextual information with low overhead to optimize processes and improve how companies and individuals interact, work, and live. The efficient management of this huge pool of devices requires runtime monitoring to identify potential performance bottlenecks and physical defects. Typical solutions, where monitoring data are aggregated in a centralized manner, soon become inefficient, as they are unable to handle the increased load and become single points of failure. In addition, the resource-constrained nature of edge devices calls for low-overhead monitoring systems. In this paper, we propose *HLF-Kubed*, a blockchain-based, highly available framework for monitoring edge devices, leveraging distributed ledger technology. HLF-Kubed builds upon Kubernetes container orchestrator and HyperLedger Fabric frameworks and implements a smart contract through an external chaincode for resource usage storing and querying. Our experimental results show that our proposed setup forms a low-overhead monitoring solution, with an average of 448 MB of memory and 6.8% CPU usage, while introducing 1.1s end-to-end latency for store operation and 0.6s for ledger querying respectively.

1. Introduction

As the years progress, users' and applications' demand for more computational power is increasing rapidly. This considerable increment in terms of computational requirements is apparent mainly for two reasons. First, the rise of the "cloud-native" computing approach, where applications are pushed and executed on the cloud, combined with the need of enterprises to extract deeper knowledge and insights from collected data, have increased the computational demands imposed by end-users. Second, the rapid growth of the Internet-of-Things (IoT) domain has also increased demands. The use of network-connected devices everywhere, from smart homes and cities,[1,2] to manufacturing,[3] agriculture,[4] and others, leads to the generation of an enormous amount of data which has to be processed and analyzed to extract valuable information.

The need to serve all this huge computational demand, as well as the real-time requirements

[*] 0x6730CCF240EBe438b3a45C81A3a7640F4d7896C8

[†] A. Tzenetopoulos (atzenetopoulos@microlab.ntua.gr), D. Masouros (dmasouros@microlab.ntua.gr), and N. Kapsoulis (nkapsoulis@mail.ntua.gr) are PhD candidates in the Department of Electrical and Computer Engineering, NTUA, Athens, Greece.

[‡] A. Litke, PhD (litke@mail.ntua.gr) is a Senior Researcher in the Department of Electrical and Computer Engineering, NTUA.

[§] D. Soudris (dsoudris@microlab.ntua.gr) and T. Varvarigou (dora@telecom.ntua.gr) are Professors in the Department of Electrical and Computer Engineering, NTUA.

and strict Quality-of-Service (QoS) demands imposed by IoT applications, has widened the horizon for a new computing paradigm known as *edge computing*.[5] Edge computing brings computation to the edge of the network, closer to where data is gathered, thus reducing the long-distance communication latency and the high bandwidth required to offload computation to the cloud. Although edge computing seeks to push computation closer to end-users, a number of first-generation edge/IoT deployments did not reach their full potential because they could not realize all of the benefits of the cloud. Even though applications, software and systems were hosted in the cloud, connectivity was still being provided through "old-economy" cellular connections, while at the same time, the constraints of low-end devices in terms of computational capabilities and attainable power consumption did not allow the entire execution to be performed at the edge.

However, the rise of 5G technology and the rethinking of edge computing have led to a mutual, intelligent effort between edge and cloud, where IoT applications are backed up by back-end services running on the cloud.[6,7] Inevitably, this cooperation between edge and cloud leads to extreme-scale distributed deployments, where the efficient management of such infrastructures requires an automated way of controlling and handling them. From the cloud point of view, Kubernetes forms the de-facto standard to deploy and manage containerized applications.[8,9] Generalizing to the edge, containers have an equally profound effect due to the flexibility and scalability they offer. In fact, lightweight containers and orchestrators have already been developed by both academia and industry, realizing the edge-cloud computing continuum.[10–13]

Even though the edge-cloud architecture delivers a huge pool of available computing resources, it also introduces new challenges, *i.e.*, how to efficiently and effectively monitor and manage the underlying infrastructure. Runtime orchestration of containers in edge-cloud architectures, which may include thousands of nodes, requires scalable monitoring solutions able to deliver information regarding the availability, performance and operation of the underlying system in a flexible and timely manner. However, the centralized nature of current monitoring solutions becomes an obstacle, due to the inability to handle and distribute the monitoring load, thus becoming a performance bottleneck and single point of failure.[14,15]

The latest iteration of distributed ledger technology (DLT), known as a *blockchain*, is a promising technology that can be leveraged to provide a monitoring solution for large-scale IoT infrastructures. Specifically, a blockchain is a shared, immutable ledger, based on a peer-to-peer topology, that facilitates the process of recording transactions and tracking assets in a network. Also, it allows anyone on the network to examine other entries in near real-time. Contrary to permissionless (or public) blockchains (*e.g.*, Bitcoin,[16] Ethereum[17]), permissioned (or private) blockchains provide a way to secure interactions among a group of entities with a common goal. On a public blockchain, the sequential execution of all transactions by all peers—due to the untrustworthy network members—limits performance. However, permissioned blockchains are more flexible and lightweight. In the context of edge computing, blockchains can be utilized as a distributed monitor-and-store solution, where IoT devices share their status as blockchain transactions on the network. Resource monitoring is a field that can be explored as a distributed ledger application. Resource usage and micro-architectural events can be exploited by malicious users to infer the type of workload executed on the machine. Therefore such resource information should be transferred to trusted members of a network, by leveraging DLT. In this case, DLT-based monitoring also adds value because of the way a shared ledger provides a single "truth"

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2022.230

regarding resource utilization. The indestructible tracking of monitored resources can also be beneficial for both the resource providers and the users, allowing the former to be aware of the resource usage of their hardware, and the latter to evaluate the hardware they are utilizing and get billed for. In this concept, similar initiatives, *i.e.*, the Pledger project, have recognized the need for cross-layer knowledge between the service providers and the users, and explore the feasibility of such distributed ledger applications on the edge.[18]

In this paper, we propose a distributed, blockchain-based resource monitoring system for low-power IoT devices. Our work relies on the **H**yper**L**edger **F**abric framework deployed on top of a lightweight **Kube**rnetes IoT cluster (HLF-Kubed).[8,19] Employing Kubernetes features for deployment automation and availability, HLF-Kubed installs a permissioned blockchain on a resource-constrained edge cluster. In order to use distributed ledger technology, we have implemented a smart contract through an external chaincode for resource usage storing and querying. Each edge node reports its resource usage metrics, *i.e.*, memory, CPU, network utilization, and stores them into the distributed ledger. Since the entire blockchain functionality is supported exclusively by low-power, resource constrained edge nodes, the end-to-end *execute-order-validate* transaction flow is managed solely by them. We measure the impact of each added layer in the installed technology stack on Raspberry Pi 3b+ and evaluate the overhead in terms of latency and memory footprint of our distributed monitoring solution. HLF-Kubed monitoring operates at 455MB RAM usage in the worst case and an average of 6.8% CPU utilization, while introducing 1.1s end-to-end latency for store operation and 0.6s for ledger querying.

The rest of this article is organized as follows. Section 2 provides an in-depth discussion of related work. Section 3 describes the background of the two main frameworks used in this work, *i.e.* K3S, a lightweight Kubernetes container orchestrator, and HyperLedger Fabric, a distributed operating system for permissioned blockchains.[9,12,19] Section 4 presents HLF-Kubed, our proposed blockchain-based monitoring framework that operates over Kubernetes IoT distributions. Finally, Sections 5 and 6 examine our experimental results and conclude the paper, providing future directions, respectively.

2. Related Work

Lately, much research has been conducted by both academia and industry, targeting the fields of container orchestration/performance optimization and system monitoring in IoT and cloud infrastructures, as well as the employment of blockchain approaches to provide distributed system architectures for edge computing environments.

2.1. Container Orchestration & Resource Management—Container orchestration and performance optimization of workloads running on the cloud/edge has been in the center of attention of many research groups. The rise of "cloud-native" platforms, such as Kubernetes, that facilitate the deployment of applications on lightweight containers and expand their capacity to dynamically scale resources, have provided grounds for the study of their impact on the performance of applications.[8,12] A portion of prior scientific works propose novel Kubernetes schedulers to optimize the placement of incoming containerized applications on the underlying cluster,[20–23] either by minimizing the overall energy utilization of cloud/edge nodes,[20] or by attempting to reduce the interference between co-located workloads either on the system or the network level.[21–23] Moreover, except for Kubernetes-based resource management approaches, other

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2022.230

scientific papers propose custom solutions able to efficiently place workloads on edge nodes. In Amit Samanta and Jianhua Tang's "Dyme," the authors propose a dynamic microservice scheduling scheme for mobile edge computing, aiming to reduce energy consumption while providing fair Quality-of-Service (QoS) among all devices.[24] In addition, Kaiya Zhang and Nancy Samaan present a mechanism for optimal offloading decisions, by formulating tasks as a constrained Markov decision process on battery constrained IoT devices.[25]

2.2. System & Application Monitoring—Monitoring and exploitation of the performance state of the underlying nodes of a cluster is essential to provide deeper insights regarding the available resources, thus driving better scheduling decisions. To this end, much research has been conducted regarding monitoring approaches, targeting both system performance and security concerns.[26] Moreover, monitoring of cloud infrastructures and containerized workloads has been tackled by prior scientific works and commercial products—*e.g.*, Prometheus—showing the potential benefits of fine-grained monitoring.[14,27,28] Moving towards the edge computing paradigm, novel monitoring systems are emerging, which provide lightweight solutions able to be deployed at resource-constrained edge devices.[29,30] Taghavi *et al.* employ Game Theory to formulate a federated cloud scenario and ensure providers' SLA compliance,[31] leveraging the Ethereum network.[17] In "IoTCop," Seshadri *et al.* utilize HyperLedger Fabric to monitor network traffic between edge devices and to isolate malicious devices that do not comply with the security policies.[32] However, in this work, we evaluate the viability of a blockchain-based monitoring system for communicating resource usage between lightweight devices on the far edge.

2.3. Blockchain-Enabled Edge Computing—Blockchains have gained a lot of popularity lately, due to their decentralized nature, which provides greater transparency between different peers, as well as increased efficiency and enhanced security. These are usually permissionless blockchains, used for permissionless cryptocurrencies (*e.g.*, Bitcoin, Ethereum),[16,17] but *permissioned* blockchains can also be leveraged as distributed database solutions for certain applications.[33,34] Specifically, in the context of Internet-of-Things and edge computing, a great amount of research has been conducted on how the convergence of these two paradigms, *i.e.*, permissioned and permissionless, can enable the construction of secure and scalable critical infrastructures,[35] whereas other scientific approaches examine the adoption of blockchain technology in certain areas, such as smart cities and industrial facilities.[36–38] Moreover, permissioned blockchains systems have also been developed to enable distributed collaboration among applications.[39–42] In "CAPER," Amiri *et al.* examine using permissioned blockchain systems to manage the internal and cross-application transactions for distributed applications,[39] while in "SharPer," and "SEPAR," Amiri *et al.* propose scalable- and privacy-oriented permissioned blockchain systems respectively.[40,41] Finally, in their "Implementation of Smart Contracts for Blockchain Based IoT Applications," Papadodimas *et al.* develop a decentralized application for sharing IoT sensors' data, by exposing them as smart contracts between IoT nodes.[42] Even though the latter is close to our work, in this paper, we propose and explore the efficacy of HyperLedger Fabric on top of Kubernetes for edge monitoring, clearly extending the aforementioned approach.

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2022.230

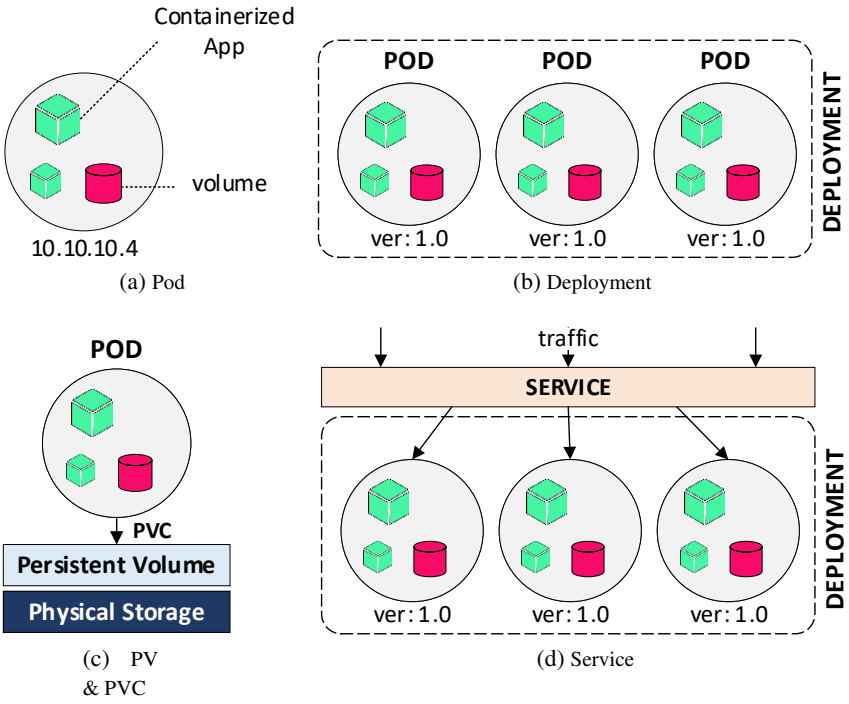


Fig. 1. Kubernetes core deployment components

3. Kubernetes & HLF Background

This section provides a brief background regarding the underlying architecture and terminology of the adopted frameworks for container orchestration and permissioned blockchains, *i.e.*, Kubernetes and HyperLedger Fabric.

3.1. Kubernetes Container Orchestrator—Kubernetes (also abbreviated as K8S) is an open-source container-orchestration system for automating computer application deployment, scaling, and management.[8] It defines a set of building blocks, which collectively provide mechanisms that deploy, maintain, and scale applications based on CPU, memory or custom metrics. Kubernetes provides a plethora of high-level abstractions that ease the deployment and management of services in the cluster. Below, we analyze the basic deployment units of Kubernetes, which are further used in Section 4 as the basic building blocks for our proposed framework.

Pod/Deployment/Service: A `pod` (Fig. 1a) is a group of one or more containers, with shared storage/network, and a specification on how to run the containers. A pod's contents are always co-located and co-scheduled, and run in a shared context. A pod models an application-specific "logical host" and it contains one or more application containers which are relatively tightly coupled. Containers within a pod share an IP address and port space, and can find each other via `localhost`. Applications within a pod also have access to shared volumes. A `deployment` includes several similar pods in a way that someone can refer to it, instead of each pod separately. Deployments facilitate pod scaling and mapping in a group of pods with similar properties. Finally, a `service` is an abstract way to expose an application running on a set of pods as a network service. Using labels, developers can map a service object to one or more pods and enable binding of an IP address to a name.

Persistent Volume: As mentioned before, containers may fail and be restarted during their

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2022.230

lifetime. By default containers do not persist their data, leading to data losses in cases of container failures/restarts. The `persistent volume` (PV - Fig. 1c) is a piece of storage in the cluster that has been provisioned by an administrator. Pods can request a part or the whole PV by applying a `persistent volume claim` (PVC). Claims can request specific size and access modes (*e.g.*, read/write or read-only).

3.2. Blockchains & HyperLedger Fabric—A blockchain can be defined as a set of blocks, shared between peers over a distributed network with complementary or conflicting interests.[43] The peers execute a consensus protocol (*e.g.*, Paxos, Raft) to validate incoming transactions and append them in an immutable database existing among different members, called ledger.

Blockchains can be roughly divided into two categories, *permissioned* and *permissionless*, which differ in the consensus. Bitcoin, the most popular blockchain network, is based on `Proof of Work` (PoW). PoW is a heavy computational task that justifies the honest majority assumption through *incentive alignment*. Thus, lately, peers of such permissionless blockchain networks need to consume large amounts of energy, using high-end specialized hardware, *e.g.*, GPUs, ASICs.

Permissioned blockchains, on the other hand, are composed of a set of identified participants. Such blockchains consist of a group of entities that share common goals, *e.g.*, commercial transactions, and can agree to trust a centralized, third-party credentialer (in this case the Membership Service Provider (see below)), even if they do not fully trust each other. HyperLedger Fabric (HLF) is an open-source implementation of a permissioned, distributed ledger technology. HLF introduces the three-phase *execute-order-validate* architecture.[19] In HLF, a distributed application is implemented on a `smart contract` called chaincode. HLF can also be considered as a distributed operating system for permissioned blockchains. It consists of the following components:

Peer: A peer executes (endorses) smart contracts, and maintains an immutable append-only ledger for each channel it participates in. Smart contracts and ledgers are used to encapsulate the shared processes and information in a network, respectively. Every peer maintains a persistent, immutable ledger. It stores both the current value of the attributes of the objects, and the history of transactions that resulted in the current values.

Membership Service Provider (MSP): The MSP maintains the identities of every node in the blockchain network and is responsible for issuing node credentials that are used for authentication and authorization.

Channel: A channel is a private `subnet` of the HyperLedger Fabric blockchain between two or more organizations. Each chaincode invocation and transaction execution takes place on a channel, where each party must be authenticated and authorized with an identity given by MSPs.

Ledger: The ledger component is a shared database among the peers participating in the same channel. HLF uses LevelDB as the state database, which is embedded in the peer process and stores chaincode data as key-value pairs.[44] The Ledger is an immutable, append-only set of blocks, synchronized among the channel's members. After being executed by the endorsing peers and ordered by an external organization, transactions are validated by the peers before updating the ledger.

Ordering Service Nodes (OSNs): When a client collects a predefined number of endorsements, it assembles a transaction and submits it to the ordering service. During the ordering phase of the HyperLedger Fabric implementation architecture, OSNs batch the incoming transactions and form blocks. A block gets dispatched as soon as any of the three conditions are met: (a)

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2022.230

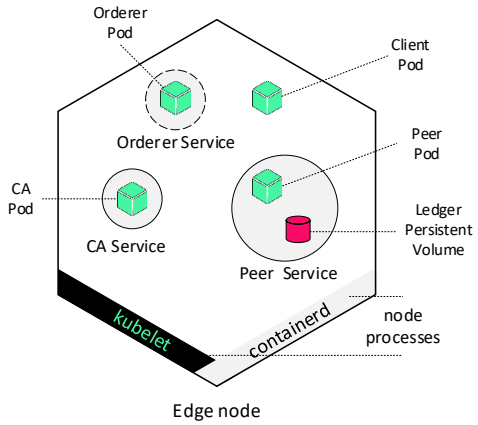


Fig. 2. HLF components Kubernetes deployment

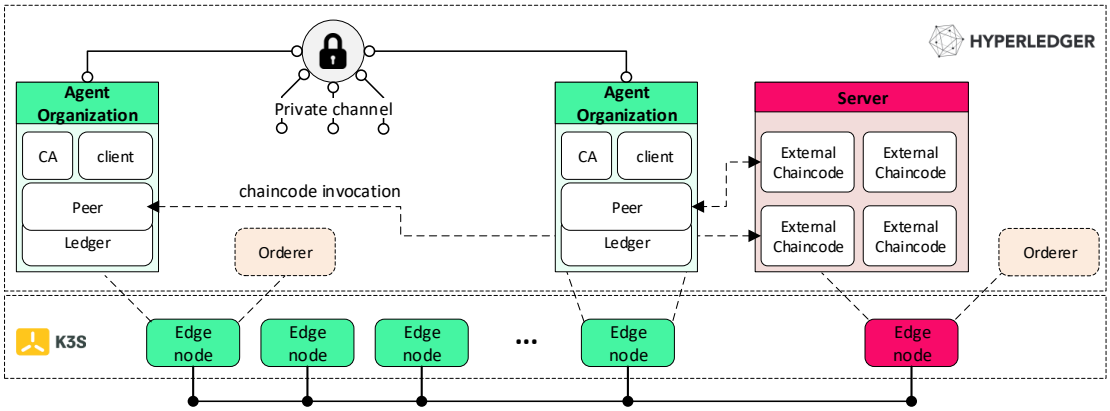


Fig. 3. HLF-Kubed architecture overview

the maximal number of transactions has been reached, (b) the block has reached a maximal size (bytes); or (c) the batch timeout since the injection of the first transaction has elapsed. According to these policies, the tuning of the aforementioned parameters may incur variability in performance metrics such as latency and throughput. Finally, OSNs ensure that the delivered blocks on one channel are ordered.

4. HLF-Kubed: A Blockchain-Based Resource Monitoring Framework

In this section, we describe the implementation of our framework setup, which is designed to comply with resource-constrained environments and provide a fault-tolerant, scalable and highly available, blockchain-based resource monitoring system. The name of HLF-Kubed was inspired by the combination of HyperLedger Fabric and Kubernetes.

4.1. ***Kube****rnetes-oriente****d*** ***H****yper****l****edger* ***F****abric*—Recently, containerization technologies have started to be used at the edge as well. More and more devices have become powerful enough to be able to support container runtimes, *e.g.*, Docker, which provides an additional layer of abstraction between the user and the host. On top of that, lightweight container orchestrators have also been designed in order to serve resource-constrained devices found at the edge, *e.g.*, KubeEdge and K3s.[11,12]

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2022.230

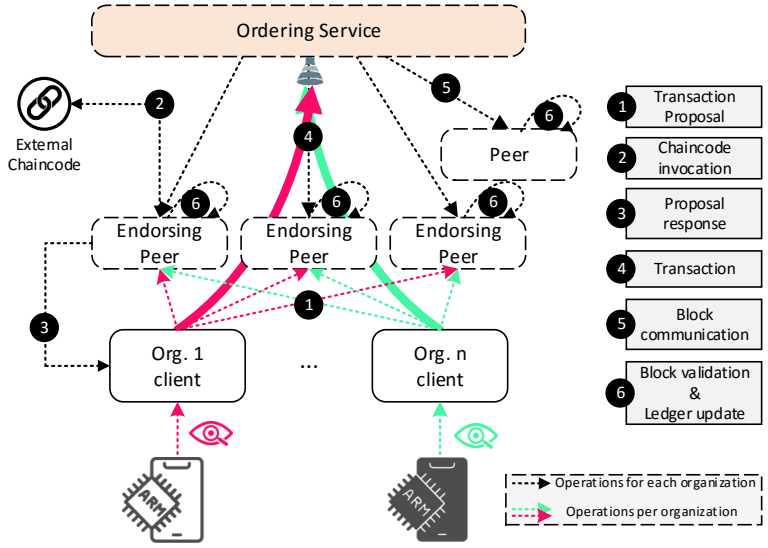


Fig. 4. HLF-Kubed monitoring system flow

For the purposes of the current work, we utilize K3s light-weight Kubernetes. K3s is based on the official Kubernetes source code, specifically designed for IoT/edge nodes. Compared to the official, production-grade Kubernetes distribution, which is intended to provide increased availability, K3s is packaged as a single <40MB binary.

HLF-Kubed uses HyperLedger Fabric v2.2. The implemented permissioned blockchain network consists of organizations which utilize resource-constrained devices for HyperLedger Fabric related operations. In correspondence to the HLF framework described in Section 3, each organization consists of various components. These components—*e.g.* peer, MSP—belong to a specific party. As MSP, we use the standalone certification authority provided by Fabric, called `fabric-ca`. Edge nodes are initially running k3s agent processes, such as the containerd service. For the purposes of our implementation, we applied some short modifications to the HyperLedger Fabric source code, and cross-compiled each component for *armv8* (ARM64) architecture. As it is illustrated in Figure 2, after being containerized, each one of the aforementioned components was encapsulated into a pod. In edge-computing environments, where devices are less stable compared to high-end data-center systems and exposed to physical environment conditions (*e.g.*, physical damage, high temperature), scheduled applications need to be easily and rapidly replaced. In this setup, the K3s controller captures and manages such node failures, by utilizing the `Deployment` objects as described in Section 3. The deployment controller ensures availability since it monitors the object's desired state; thus, in case of any component failure, (*e.g.*, due to resource starvation), the pod will automatically be re-created. Regarding the persistent, immutable ledger, we utilize a persistent volume (PV), which is attached to the `peer` pod using a persistent volume claim (PVC). The circles in Fig. 2 depict the configured Kubernetes *Service* feature. With Service, deployments' pods, labeled with unique key-value identifiers, are accessible by developers at application-level, using a single DNS name. Since container orchestration is managed centrally by K3s server, we place every component associated with a specific organization into a separate edge-node using the node affinity feature.

Except for the main HyperLedger Fabric-participating organizations, we also deploy ordering

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2022.230

service nodes belonging to an external organization. To achieve consensus on the strict ordering of transactions between ordering service nodes, we utilize Raft.[45] Raft is the recommended consensus implementation in HyperLedger Fabric v2.x. It is a crash fault tolerant (CFT) ordering service based on an implementation of Raft protocol in etcd. In Raft, nodes following a *leader/-follower* model, elect a leader to order the incoming transactions and its decisions are replicated to the followers. Spreading the ordering service nodes across different devices, racks or even locations is a factor in the establishment of a high availability strategy for an ordering service. Therefore, using pod `anti-affinity`, we spread them across every edge-node in the cluster.

Figure 3 illustrates an overview of the HLF-Kubed architecture. All trusted organizations (edge nodes) are members of a channel designated for monitoring purposes. Thus, peers of different organizations publish their {CPU, memory, network} resource usage to the distributed ledger. Therefore, information regarding resource contention of every network participant is distributed across the peers, through the shared, appended blocks at their ledger. Chaincodes are executed externally, preferably on a remote device (even in cloud data-centers), other than the one being monitored.

The main idea of this work is a lightweight, decentralized, IoT monitoring service, enabled by the permissioned blockchain technology's enhanced consensus protocols, HyperLedger Fabric's efficient architecture implementation, and Kubernetes's high availability, ease of deployment, and application management features. As illustrated in Figure 4, devices belonging to different organizations ❶ report their resource usage metrics in the form of a transaction proposal towards the endorsing peers. Endorsing peers ❷ simulate the proposal by executing the external chaincode and return the proposal response. When a client collects enough endorsements, it ❸ assembles a transaction and ❹ submits it to the ordering service. An ordering service groups, batches and ❺ sends transactions to the peers for ❻ block validation and ledger update.

4.2. External Chaincode—Traditionally, application chaincode, also referred to as a *smart contract*, runs in a separate process within a Docker container environment. While this technique isolates the chaincodes from the peer environment, it adds increased complexity in an already container-based setup. In this paper, peer, fabric-ca and the other HyperLedger Fabric nodes are deployed as pods. To this end, each node is running on top of a container runtime; thus any chaincode execution would invoke a Docker in Docker (DinD) operation.

In this context, we employ the *external chaincode* feature supported by Fabric since v2.0. Following the cloud-native, highly modular, *everything as a service* paradigm of Kubernetes, we deploy the application chaincode externally. Authorized peers remotely invoke the chaincode associated with the organization they belong to, through a Kubernetes service endpoint. Therefore, the application logic of our solution is implemented on an external chaincode, hosted isolated within a pod on an arbitrary node-member of the Kubernetes cluster.

We developed our chaincode as a proof of concept in Golang. The developer/user is able to retrieve resource usage metrics from each organization participating in the channel. Moreover, they can also query any individual device's allocated ($getDevAllocated(id)$) and available ($getDevAvailable(id)$) resources by inserting its unique identifier.

4.3. Monitoring—On five-second intervals, client nodes monitor the average CPU utilization, memory (RAM) allocation, and incoming and outgoing network bandwidth of the respective device. The volume of the extracted data of the examined scenario is low, *i.e.*, about 25 bytes per reporting device. However, in cases with a greater data footprint, we can consider using an

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2022.230

off-chain store, *i.e.*, InterPlanetary File System (IPFS), in order to address the unbounded data growth.[46] Subsequently, the client signs and sends a transaction proposal to the endorsing peers for simulating the execution against the endorser's local blockchain state. The client collects endorsements back until they satisfy the endorsement policy, creates the transaction and passes it to the ordering service nodes. Orderers using Raft, order, group and broadcast transactions to the peers belonging to the channel for approval. We execute a monitoring script in the client pod which is placed in a different device for each organization.

The simple algorithm of this script is described in Algorithm 1. We get CPU, memory and network bandwidth utilization. For CPU and memory we calculate the average between the measured values of the current and the previous iteration. Regarding the network bandwidth, since the value retrieved is accumulated, we calculate the difference between the previous and the current measurement divided by the time passed in seconds. Then, we invoke the external chaincode passing the device ID, network device name, and the measured resources' values as parameters. Finally, until the five-second interval time has passed, the system sleeps.

Algorithm 1: Device Resource Usage Monitoring

$cpu_o, mem_o, netwIn_o, netwOut_o \leftarrow getValues()$;
$start \leftarrow date_{ms}()$;
initialization;
while *true* **do**
  $cpu_n, mem_n, netwIn_n, netwOut_n \leftarrow getValues()$;
  $runtime \leftarrow getTimePassed_s$;
  $avg\_cpu \leftarrow \frac{(cpu_n + cpu_o)}{2}$;
  $avg\_mem \leftarrow \frac{(mem_n + mem_o)}{2}$;
  $avg\_netwOut \leftarrow \frac{(netwout_n - netwout_o)}{runtime}$;
  $avg\_netwIn \leftarrow \frac{(netwout_n - netwout_o)}{runtime}$;
  $ccInvoke(device\_name, avg\_cpu, avg\_mem, avg\_netwOut, avg\_netwIn)$;
  $end \leftarrow date_{ms}()$;
  $cpu_o, mem_o, ... \leftarrow cpu_n, mem_n, ...$;
  $sleeptime \leftarrow end - start$;
  $start \leftarrow end$;
  $sleep(sleeptime)$;
end

5. Experimental Evaluation

We evaluate the efficacy of our solution on a cluster that consists of three Raspberry Pi 3b+ devices. Raspberry Pi 3b+ includes a quad-core Cortex A53 ARM processor and 1GB RAM. Further specifications are described in Table 1. On top of them, we installed K3s, with a single server node and two agents. K3s includes and defaults to containerd, an industry-standard container runtime.[47] This cluster is the base infrastructure environment for our experiments.

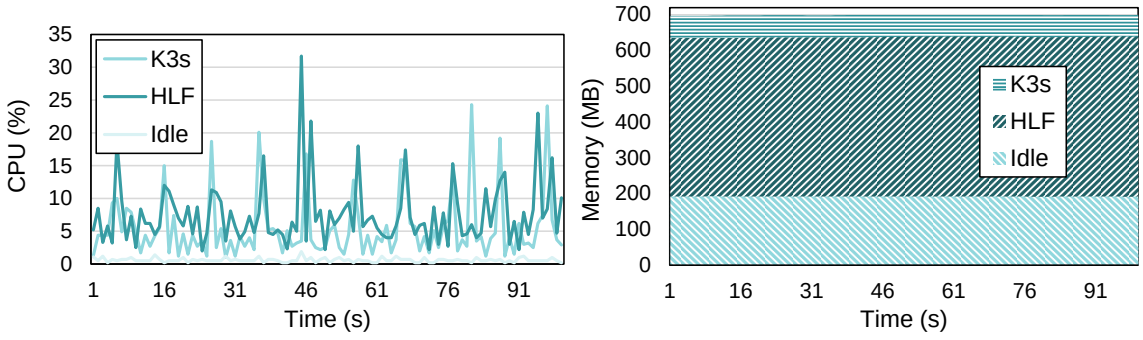
5.1. Technology Stack Impact—Resource-constrained devices, found at the edge, are most of the times unable to host cloud-native technologies. In our first experiment, we monitor CPU

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2022.230

Table 1. Hosts specifications

Architecture	ARM v8 64bit SoC	**L1 Cache**	32KB
Processor Model	Cortex-A53	**L2 Cache**	512KB
Cores	4	**Networking**	Gigabit Ethernet (via USB channel)
Frequency	1.4GHz	**Dimensions**	82mm x 56mm x 19.5mm, 50g
Memory	1GB LPDDR2 SDRAM	**Operating System (kernel)**	Linux raspberrypi 5.4.79-v8+

and memory utilization in different layers of the proposed technology stack. Utilizing the UNIX `top` utility, we extract CPU and Memory usage at one-second intervals. Figure 5 depicts K3s server's (5a and 5b) and agent's (5c and 5d) resource utilization on idle, K3s, and HyperLedger Fabric setup respectively.



(a) Server CPU Utilization

(b) Server Memory Utilization

(c) Agent CPU Utilization

(d) Agent Memory Utilization

Fig. 5. Resource utilization of server and agent nodes over our step-by-step technology stack installation

Server: K3s server, after the K3s installation, introduces increased utilization in both CPU and memory resources. Since the cluster has been set up exclusively on edge devices, the resource-hungry server node is limited by its 1GB of available RAM. Nonetheless, lightweight Kubernetes is successfully installed. Additionally, the contribution of the server to the HyperLedger Fabric hosting is insignificant; thus no additional overhead is added due to the permissioned blockchain. Figures 5a and 5b illustrate the overhead of K3s to the server's utilization. While the Raspberry Pi 3b+ is running idle, the RAM and CPU utilization numbers are 191MB and 0.6%, respectively. After installing K3s, those values escalate to 698MB and 5.4%.

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2022.230

Agents: On the other hand, the resource utilization in K3s agents varies. Figures 5c and 5d illustrate the lower impact Kubernetes installation has on device resources. K3s increases memory utilization to 298MB and CPU usage to 1.9%; however, HyperLedger Fabric components—*i.e.*, peer, certificate authority, ordering service and client—in contrast to K3s server, grant an additional value of 118MB of RAM. CPU utilization increases to 3.6% as well.

A conclusion derived from the previous metrics is that just 400MB of memory were sufficient for every node agent to install Kubernetes and HyperLedger Fabric, proving this setup a viable option for embedding permissioned blockchain technology on edge devices. On top of that, since the resources for hosting the server which orchestrates containers—*i.e.*, K3s and HLF components—were sufficient, the solution can be applied exclusively on edge devices without the need for remote management.

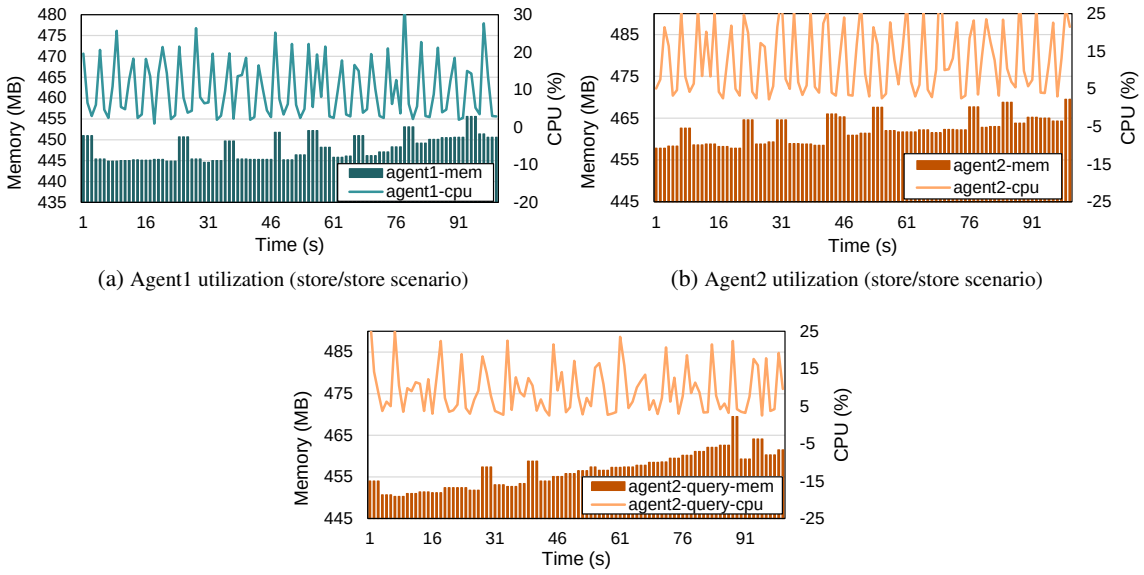
5.2. Blockchain-Based Monitoring—Finally, we measure the resource usage of agent devices during the execution of the monitoring (Section 4). Similarly to the previous subsection, we utilize the `top` utility to monitor CPU and Memory usage considering two different scenarios.

In the first scenario both organizations, using the monitoring script described in Algorithm 1, measure and store CPU, memory utilization, and incoming and outgoing network bandwidth usage to the distributed ledger. The end-to-end execution time for a) external chaincode invocation, b) execution of the transaction proposal by the endorsers, c) the ordering phase, d) the validation phase, and e) a ledger update for every channel member, is 1.1 seconds. In our case, since a majority endorsing policy is required, both peer organizations should accept any incoming invocation request, in order for it to be considered as a valid transaction. In our setup, we place external chaincode randomly; therefore processes are not balanced between the two agent nodes. For that reason, we measure usage metrics for both of the agent devices. Figure 6a illustrates the CPU and memory utilization of Agent1. The spikes in the memory utilization depict the store operations. Agent2 accordingly, in Figure 6b, presents higher memory and CPU utilization. This may be due to the external chaincode service that was co-scheduled with Fabric components in the same device. Hence, given the fact that external chaincode is placed remotely, HLF-Kubed extracts and stores resource usage metrics in every channel member device's ledger with a 448MB average and 456MB worst-case memory usage respectively, utilizing only 6.8% time of CPU usage.

In the second use-case scenario, we monitor the device's resource utilization when Agent2 queries the shared ledger. This scenario describes the case in which the role of a participating member in the blockchain network is restricted to accessing the ledger for information retrieval. More precisely, while Agent1 keeps monitoring and storing as before, Agent2 queries resource usage metrics from the ledger. The average time delay for querying the shared ledger was 0.6 seconds. CPU and memory usage traces of Agent2, during resource usage retrieval in an interval of 5 seconds, are illustrated in Figure 6c. As expected, in this scenario, we observe lower memory and CPU utilization compared to the same agent's behaviour in the first scenario (Fig. 6b), in which the store operation induced additional overhead.

In both of the deployed scenarios, an increase in the utilized memory is illustrated (Fig. 6). This behaviour is justified by the presence of an embedded database in the peer, utilized by the HyperLedger Fabric framework. This database caches the latest blocks inserted in the ledger, based on a user-specified amount of memory for low latency retrieval. Thus, the user can tune this value with the amount of memory used for caching, depending on the desired window in the

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2022.230



(a) Agent1 utilization (store/store scenario)

(b) Agent2 utilization (store/store scenario)

(c) Agent2 utilization (store/query)

Fig. 6. Resource utilization of agent nodes under two use-case scenarios

ledger history that needs to be accessed with low latency.

6. Conclusion & Future Work

In this paper, we presented HLF-Kubed, a blockchain-based monitoring tool for resource-constrained edge clusters. HLF-Kubed acts as a proof of concept, showing that permissioned blockchains can be implemented on low-power edge devices, leveraging lightweight Kubernetes for HyperLedger Fabric's components orchestration. We measured our solution's impact on system resources, and observed a maximum of 455MB RAM and 6.8% CPU usage. However, in networks with *majority* as consensus policy, device scaling will result to reduction of average resource usage, since the percentage of the endorsing peers remains stable.

From the above, it is evident that hosting autonomous, permissioned blockchains, solely on a resource-constrained environment, is a viable solution. The proposed setup offers simplicity through seamless addition and removal of peers, and fault-tolerance, leveraging the recovery mechanisms of Kubernetes. On the other hand, sharing the current state of devices among a network of trusted peers is a desired functionality in edge computing environments. In the examined proof of concept, edge devices, unburdened from the need of any external coordinator, store their current resource usage to the shared ledger.

Regarding the described use-case, future works could exploit the distributed shared ledger for resource usage awareness across every device on a network for scheduling or resource management purposes and compare it with traditional, decentralized monitoring tools, *e.g.*, Prometheus.[14] Also, the inherent design characteristics of a blockchain network, such as data integrity from the consensus protocol, provide trust for resource availability amongst the different parties of the network. From an infrastructure perspective, HLF-Kubed can also be extended to host heterogeneous devices with diverse characteristics and to examine the functionality of such a decentralized network at large scale. Finally, in this concept of scalability evaluation, additional

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2022.230

research could be conducted on applying such solutions on remote, federated Kubernetes clusters in order to further harness the potential of decentralized peer-to-peer networks.

Author Contributions

All authors contributed equally to this work.

Conflict of Interest

The authors have affirmed they have no conflicts of interest as described in Ledger's Conflict of Interest Policy.

Notes and References

[1] Khan, L. U., Yaqoob, I., Tran, N. H., Kazmi, S. A., Dang, T. N., Hong, C. S. "Edge-Computing-Enabled Smart Cities: A Comprehensive Survey." *IEEE Internet of Things Journal* **7.10** 10200–10232 (2020) https://doi.org/10.1109/JIOT.2020.2987070.

[2] Mocrii, D., Chen, Y., Musilek, P. "IoT-Based Smart Homes: A Review of System Architecture, Software, Communications, Privacy and Security." *Internet of Things* **1** 81–98 (2018) https://doi.org/10.1016/j.iot.2018.08.009.

[3] Chen, B., Wan, J., Celesti, A., Li, D., Abbas, H., Zhang, Q. "Edge Computing in IoT-Based Manufacturing." *IEEE Communications Magazine* **56.9** 103–109 (2018) https://doi.org/10.1109/MCOM.2018.1701231.

[4] Somov, A., *et al.* "Pervasive Agriculture: IoT-Enabled Greenhouse for Plant Growth Control." *IEEE Pervasive Computing* **17.4** 65–75 (2018) https://doi.ieeecomputersociety.org/10.1109/MPRV.2018.2873849.

[5] Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L. "Edge computing: Vision and Challenges." *IEEE internet of things journal* **3.5** 637–646 (2016) https://doi.org/10.1109/JIOT.2016.2579198.

[6] No Author. "The Future of Computing: Intelligent Cloud and Intelligent Edge." *Microsoft* (accessed 9 March 2022) https://azure.microsoft.com/en-us/overview/future-of-cloud.

[7] Wang, J., Pan, J., Esposito, F., Calyam, P., Yang, Z., Mohapatra, P. "Edge Cloud Offloading Algorithms: Issues, Methods, and Perspectives." *ACM Computing Surveys (CSUR)* **52.1** 1–23 (2019) https://doi.org/10.1145/3284387.

[8] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., Wilkes, J. "Borg, Omega, and Kubernetes: Lessons Learned from Three Container-Management Systems Over a Decade." *Queue* **14.1** 70–93 (2016) https://doi.org/10.1145/2898442.2898444.

[9] Brewer, E. A. "Kubernetes and the Path to Cloud Native." In *Proceedings of the Sixth ACM Symposium on Cloud Computing* 167–167 (2015) https://doi.org/10.1145/2806777.2809955.

[10] Park, M., Bhardwaj, K., Gavrilovska, A. "Toward lighter containers for the edge." In *3rd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 20)* (2020) https://www.usenix.org/conference/hotedge20/presentation/park.

[11] Xiong, Y., Sun, Y., Xing, L., Huang, Y. "Extend Cloud to Edge with KubeEdge." In *2018 IEEE/ACM Symposium on Edge Computing (SEC)* IEEE 373–377 (2018) https://doi.org/10.1109/SEC.2018.00048.

[12] No Author. "Lightweight Kubernetes." *K3s* (accessed 9 March 2022) https://k3s.io/.

[13] No Author. "MicroK8s: High Availability K8s." *Canonical* (accessed 9 March 2022) https://microk8s.io.

[14] No Author. "Prometheus." *Prometheus* (accessed 9 March 2022) https://prometheus.io.

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2022.230

[15] Taherizadeh, S., Jones, A. C., Taylor, I., Zhao, Z., Stankovski, V. "Monitoring Self-Adaptive Applications Within Edge Computing Frameworks: A State-of-the-Art Review." *Journal of Systems and Software* **136** 19–38 (2018) https://doi.org/10.1016/j.jss.2017.10.033.

[16] Nakamoto, S. "Bitcoin: A Peer-to-Peer Electronic Cash System." (2008) (accessed 9 March 2022) https://bitcoin.org/bitcoin.pdf.

[17] Wood, G. "Ethereum: A Secure Decentralised Generalised Transaction Ledger, Berlin Version 4b05e0d — 2022-03-09." (2014, 2022) (accessed 9 March 2022) https://ethereum.github.io/yellowpaper/paper.pdf.

[18] "Pledger Project." *Pledger* (accessed 9 March 2022) http://www.pledger-project.eu.

[19] Androulaki, E., *et al.* "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains." In *EuroSys '18: Proceedings of the Thirteenth EuroSys Conference* 1–15 (2018) https://doi.org/10.1145/3190508.3190538.

[20] Kaur, K., Garg, S., Kaddoum, G., Ahmed, S. H., Atiquzzaman, M. "KEIDS: Kubernetes-Based Energy and Interference Driven Scheduler for Industrial IoT in Edge-Cloud Ecosystem." *IEEE Internet of Things Journal* **7.5** 4228–4237 (2019) https://doi.org/10.1109/JIOT.2019.2939534.

[21] Tzenetopoulos, A., Masouros, D., Xydis, S., Soudris, D. "Interference-Aware Orchestration in Kubernetes." In H. Jagode, H. Anzt, G. Juckeland, H. Ltaief (Eds.), *High Performance Computing: ISC High Performance 2020 International Workshops, Frankfurt, Germany, June 21–25, 2020, Revised Selected Papers* Cham: Springer International Publishing 321–330 (2020) https://doi.org/10.1007/978-3-030-59851-8_21.

[22] Verreydt, S., Beni, E. H., Truyen, E., Lagaisse, B., Joosen, W. "Leveraging Kubernetes for Adaptive and Cost-Efficient Resource Management." In *WOC '19: Proceedings of the 5th International Workshop on Container Technologies and Container Clouds* 37–42 (2019) https://doi.org/10.1145/3366615.3368357.

[23] Santos, J., Wauters, T., Volckaert, B., De Turck, F. "Towards Network-Aware Resource Provisioning in Kubernetes for Fog Computing Applications." In *2019 IEEE Conference on Network Softwarization (NetSoft)* IEEE 351–359 (2019) https://doi.org/10.1109/NETSOFT.2019.8806671.

[24] Samanta, A., Tang, J. "Dyme: Dynamic Microservice Scheduling in Edge Computing Enabled IoT." *IEEE Internet of Things Journal* **7.7** 6164–6174 (2020) https://doi.org/10.1109/JIOT.2020.2981958.

[25] Zhang, K., Samaan, N. "Optimized Look-Ahead Offloading Decisions Using Deep Reinforcement Learning for Battery Constrained Mobile IoT Devices." In *2020 IEEE International Conference on Smart Cloud (SmartCloud)* IEEE 181–186 (2020) https://doi.org/10.1109/SmartCloud49737.2020.00042.

[26] Bauman, E., Ayoade, G., Lin, Z. "A Survey on Hypervisor-Based Monitoring: Approaches, Applications, and Evolutions." *ACM Computing Surveys (CSUR)* **48.1** 1–33 (2015) https://doi.org/10.1145/2775111.

[27] Masouros, D., Xydis, S., Soudris, D. "Rusty: Runtime Interference-Aware Predictive Monitoring for Modern Multi-Tenant Systems." *IEEE Transactions on Parallel and Distributed Systems* **32.1** 184–198 (2020) https://doi.org/10.1109/TPDS.2020.3013948.

[28] Brondolin, R., Santambrogio, M. D. "A Black-Box Monitoring Approach to Measure Microservices Runtime Performance." *ACM Transactions on Architecture and Code Optimization (TACO)* **17.4** 1–26 (2020) https://doi.org/10.1145/3418899.

[29] Souza, A., Cacho, N., Noor, A., Jayaraman, P. P., Romanovsky, A., Ranjan, R. "Osmotic Monitoring of Microservices Between the Edge and Cloud." In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)* IEEE 758–765 (2018) https://doi.org/10.1109/HPCC/SmartCity/DSS.2018.00129.

[30] Großmann, M., Klug, C. "Monitoring Container Services at the Network Edge." In *2017 29th International Teletraffic Congress (ITC 29)* IEEE 130–133 (2017) https://doi.org/10.23919/ITC.2017.8064348.

[31] Taghavi, M., Bentahar, J., Otrok, H., Bakhtiyari, K. "A Blockchain-Based Model for Cloud Service Quality Monitoring." *IEEE Transactions on Services Computing* **13.2** 276–288 (2019) https://doi.org/10.1109/TSC.2019.2948010.

ISSN 2379-5980 (online)
DOI 10.5195/LEDGER.2022.230

[32] Seshadri, S. S., *et al.* "IoTCop: A Blockchain-Based Monitoring Framework for Detection and Isolation of Malicious Devices in Internet-of-Things Systems." *IEEE Internet of Things Journal* **8.5** 3346–3359 (2020) https://doi.org/10.1109/JIOT.2020.3022033.

[33] Belotti, M., Božić, N., Pujolle, G., Secci, S. "A Vademecum on Blockchain Technologies: When, Which, and How." *IEEE Communications Surveys & Tutorials* **21.4** 3796–3838 (2019) https://doi.org/10.1109/COMST.2019.2928178.

[34] Monrat, A. A., Schelén, O., Andersson, K. "A Survey of Blockchain from the Perspectives of Applications, Challenges, and Opportunities." *IEEE Access* **7** 117134–117151 (2019) https://doi.org/10.1109/ACCESS.2019.2936094.

[35] Wu, Y., Dai, H.-N., Wang, H. "Convergence of Blockchain and Edge Computing for Secure and Scalable IIoT Critical Infrastructures in Industry 4.0." *IEEE Internet of Things Journal* **8.4** 2300–2317 (2020) https://doi.org/10.1109/JIOT.2020.3025916.

[36] Xie, J., *et al.* "A Survey of Blockchain Technology Applied to Smart Cities: Research Issues and Challenges." *IEEE Communications Surveys & Tutorials* **21.3** 2794–2830 (2019) https://doi.org/10.1109/COMST.2019.2899617.

[37] Li, M., Hu, D., Lal, C., Conti, M., Zhang, Z. "Blockchain-Enabled Secure Energy Trading with Verifiable Fairness in Industrial Internet of Things." *IEEE Transactions on Industrial Informatics* **16.10** 6564–6574 (2020) https://doi.org/10.1109/TII.2020.2974537.

[38] Liang, W., Tang, M., Long, J., Peng, X., Xu, J., Li, K.-C. "A Secure FaBric Blockchain-Based Data Transmission Technique for Industrial Internet-of-Things." *IEEE Transactions on Industrial Informatics* **15.6** 3582–3592 (2019) https://doi.org/10.1109/TII.2019.2907092.

[39] Amiri, M. J., Agrawal, D., Abbadi, A. E. "CAPER: A Cross-Application Permissioned Blockchain." *Proceedings of the VLDB Endowment* **12.11** 1385–1398 (2019) https://doi.org/10.14778/3342263.3342275.

[40] Amiri, M. J., Agrawal, D., El Abbadi, A. "SharPer: Sharding Permissioned Blockchains Over Network Clusters." In *SIGMOD/PODS '21: Proceedings of the 2021 International Conference on Management of Data* 76–88 (2021) https://doi.org/10.1145/3448016.3452807.

[41] Amiri, M. J., Duguépéroux, J., Allard, T., Agrawal, D., El Abbadi, A. "SEPAR: A Privacy-Preserving Blockchain-Based System for Regulating Multi-Platform Crowdworking Environments." *arXiv* (2020) (accessed 9 March 2022) https://doi.org/10.48550/arXiv.2005.01038.

[42] Papadodimas, G., Palaiokrasas, G., Litke, A., Varvarigou, T. "Implementation of Smart Contracts for Blockchain Based IoT Applications." In *2018 9th International Conference on the Network of the Future (NOF)* IEEE 60–67 (2018) https://doi.org/10.1109/NOF.2018.8597718.

[43] Kolb, J., AbdelBaky, M., Katz, R. H., Culler, D. E. "Core Concepts, Challenges, and Future Directions in Blockchain: A Centralized Tutorial." *ACM Computing Surveys (CSUR)* **53.1** 1–39 (2020) https://doi.org/10.1145/3366370.

[44] Costan, V., *et al.* "LevelDB: Fast Key-Value Storage Library." *GitHub* (accessed 9 March 2022) https://github.com/google/leveldb.

[45] Ongaro, D., Ousterhout, J. "In Search of an Understandable Consensus Algorithm." In *USENIX ATC '14: Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference* 305–320 (2014) https://dl.acm.org/doi/10.5555/2643634.2643666.

[46] No Author. "IPFS." *IPFS* (accessed 9 March 2022) https://ipfs.io.

[47] containerd Authors "containerd - An Industry-Standard Container Runtime with an Emphasis on Simplicity, Robustness, and Portability." *continerd* https://containerd.io.

SSN 2379-5980 (online)
DOI 10.5195/LEDGER.2022.230