Truong, Hong Linh; Rydzi, Filip

Benchmarking Blockchain Interactions in Mobile Edge Cloud Software Systems

Published in:
Benchmarking, Measuring, and Optimizing - Third BenchCouncil International Symposium, Bench 2020, Revised Selected Papers

DOI:
10.1007/978-3-030-71058-3_13

Published: 01/01/2021

Document Version
Peer reviewed version

Please cite the original version:
Truong, H. L., & Rydzi, F. (2021). Benchmarking Blockchain Interactions in Mobile Edge Cloud Software Systems. In F. Wolf, & W. Gao (Eds.), *Benchmarking, Measuring, and Optimizing - Third BenchCouncil International Symposium, Bench 2020, Revised Selected Papers* (pp. 213-231). (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Vol. 12614 LNCS). https://doi.org/10.1007/978-3-030-71058-3_13

Benchmarking Blockchain Interactions in Mobile Edge Cloud Software Systems

Hong-Linh Truong[1][0000−0003−1465−9722] and Filip Rydzi[2]

[1] Department of Computer Science, Aalto University, Finland
linh.truong@aalto.fi
[2] Independent, Slovakia
rydzi.filip@gmail.com

Abstract. As blockchain becomes an essential part of many software systems in the edge and cloud, the developer starts to treat blockchain features like commodity software components that can be integrated into edge and cloud software systems. For the developer it is quite challenging to determine, customize, and evaluate suitable blockchain features for software systems in the edge and cloud environments. In this paper, we conceptualize important blockchain interactions in mobile edge computing software systems (MECSS) and present generic techniques for evaluating these interactions. We determine different interaction patterns for different deployments of compute resources and networks. We abstract and represent application-level mobile edge computing (MEC) features and blockchain features to create MECSS deployment models to be coupled with testbed deployments for benchmarking application-level interactions within application contexts. Based on that, we develop a generic framework for building and executing benchmarks of application-level blockchain interactions within MECSS. We will demonstrate our framework for vehicle-to-everything communication scenarios with two main blockchain technologies, Hyperledger Fabric and Ethereum, using various types of compute resources in edge and cloud infrastructures.

Keywords: benchmark, testing, blockchain, edge computing, microservices

1 Introduction

To date, blockchain technologies have been widely exploited in different types of software systems [8, 38]. Applications and systems utilize blockchain as commodity features to support verifiable and open transactions. One of the challenges for the application developer is the question of which blockchain features should be integrated in which part of the developer's complex software systems. Especially, in our focus, mobile edge computing software systems (MECSS) [21, 31] have increasingly leveraged blockchain potentials due to application requirements [12, 18, 15, 39]. For the developer, combining blockchain features with MEC features is extremely challenging because the complexity and diversity of software services, system configurations and interactions.

The need for development tools for blockchain-based software is high [27] and we lack tools to support the design and evaluation of combined features from MEC and blockchain for different realistic deployments. Especially, our work is concerned with the application-level interactions in MEC that need blockchain features. Determining such interactions is time-consuming. Furthermore, software development knowledge for integrating edge computing with blockchain is hard to acquire, due to several challenges [6, 35, 14]. Therefore, generic methods and tools to enable testing of such interactions play an important role. Such tools and methods are different from current benchmarking and testing for particular blockchain systems, individual blockchain features, or for individual MEC resources, as we need to develop generic ways to evaluate *integration interaction patterns* in which blockchain-based data exchanges are carried within suitable MEC design models, reflecting application developer needs. Furthermore, key quality metrics and appropriate deployments, including underlying computing resources and application topologies, must be considered at the application level. As we also discuss in the related work (see Section 5), many works provide high-level recommendations of using blockchain features and blockchain systems, but they do not treat combined blockchain features with other features within contexts of interactions in MECSS.

This paper identifies such interactions and supports methods to evaluate the interactions in the development of MECSS with blockchain features. A MECSS includes various components deployed in MEC infrastructures that interact with each other. Without loss of generality and based on the development trend, we concentrate on MECSS components developed as microservices [24, 28]: we focus on interactions in such microservices, not low-level networks or infrastructures in MEC. We (i) treat blockchain features like software components that can be taken from existing blockchain libraries/frameworks, (ii) consider common integration of such features based on different needs, and then (iii) test such features for MECSS, especially for application-level interactions within MECSS. To help the developer to perform a systematic way of benchmarking interactions, we choose a known MECSS application domain – the vehicle-to-everything (V2X) – to explain our work. We will use scenarios in V2X to present our methods and software prototype. We first present an abstract view on MECSS and their blockchain features, key interactions in the interest of testing, and their software coupling from an application viewpoint (see Section 2). We then contribute a benchmark framework with features of automatic deployment and execution of benchmarks (see Section 3).

Our framework is designed for evaluating blockchain features based on a complex dependency of blockchain features, application-related operations, system topology and resources running blockchain features. Utilizing our framework, we carry out several experiments to present how our framework could provide insights into realistic deployments of MECSS for the developer (see Section 4). We illustrate our work with well-known blockchain systems, like Hyperledger Fabric and Ethereum, using edge and cloud infrastructures.

2 Aspects in Benchmarking Blockchain Features for MECSS

2.1 Mobile Edge Computing Software Systems

Several papers have conceptualized mobile edge computing and fog computing [7] with multiple layers. Practically, we focus on applications atop a common 3-level of MEC infrastructures, shown in Figure 1. In this 3-level abstract infrastructure model, a mobile component *mc* is movable and interacts with other mobile components, edge units, edge data centers and clouds. The level 1 edge $edge_{l1}$ indicates typical edge units/devices/systems which are very close to the mobile component, serving for a limited number of mobile components. The level 2 edge $edge_{l2}$ is deployed in edge data centers with reasonable capabilities, e.g. connected telco base stations. Finally, cloud-based data centers offer *cloud* services for *mc*, $edge_{l1}$ and $edge_{l2}$ components. Such infrastructures are common in many real deployments and scenarios in smart building, vehicle communication and smart factory. Mapping to the 5 layers of edge/fog/cloud in [21], our *mc* is at layer 5, $edge_{l1}$ is at layer 2, $edge_{l2}$ at layer 3 and *cloud* is at layers 2 and 1. In this paper, we will use the V2X communication scenarios [12, 17, 13, 23, 2, 16, 25] to demonstrate our work.

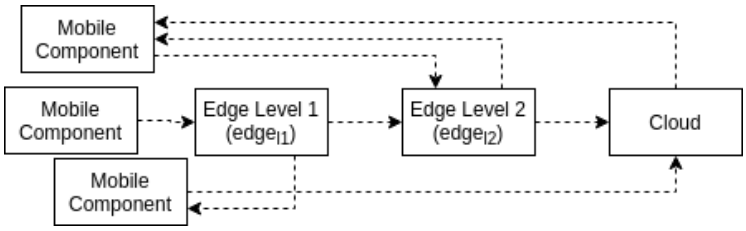

Fig. 1. Simplified view of three-level MEC with two layers of edges

Given the above-mentioned view on MEC infrastructures, we consider distributed, application-required components as microservices residing in different levels in the above-mentioned view. With the assumption that underlying networks and operating systems of MEC have different mechanisms to enable us to compose and deploy these microservices to build MECSS (e.g., via containerization), we will address the service developer concerns in application-level interactions utilizing blockchain features. MECSS have different application-level software features, covering typical MEC software features at the application level, such as sending and receiving messages from message brokers, blockchain features, like read/writing blockchain transactions and consensus/mining execution, and application-specific features, such as reporting events or performing data analytics. Such features will be implemented within microservices deployed in resources in the 3-level MEC infrastructural model. There are many types of

interactions, shown in Table 1, in existing MEC use cases that one can find in the literature and real-world applications (e.g., [12, 17, 13, 23, 2, 16, 25]).

For such application-level features working properly, many system services have to be deployed or available to offer infrastructure- and platform-level features, such as message brokers, blockchain nodes, message flow engines, etc.

Table 1. Example of key interactions that should be benchmarked

Interactions: Description	**Example of metrics**	**V2X data exchange scenarios**
mc–*mc*: a service in a *mc* interacts directly with a service in another *mc*	transaction acceptance rate, transaction acceptance time	vehicles directly exchange driving data [36].
mc–$edge_{l1}$–*mc*: a service in a *mc* interacts with a service in another *mc* via a service in $edge_{l1}$	transaction acceptance rate, transaction acceptance time, cost	vehicles send road warnings via a road side unit (RSU) [25, 17]
mc–$edge_{l2}$–*mc*: a service in a *mc* interacts with another service in another *mc* via a service in a $edge_{l2}$	transaction acceptance rate and synchronization state, transaction acceptance time, cost	exchanges of data between vehicles over a data broker deployed in an edge node in mobile base stations
mc–$edge_{l1}$–$edge_{l2}$–*mc*: a service in a *mc* interacts with another service in another *mc* via a service in a $edge_{l1}$ via a service in $edge_{l2}$	transaction acceptance rate and synchronization state, transaction acceptance time, cost	exchanges of data between vehicles via a RSU, which relays data to an edge data center
mc–$edge_{l2}$–*cloud*: a service in a *mc* interacts with a service in a *cloud* via a service in a $edge_{l2}$	infrastructure costs, transaction acceptance rate and synchronization state	vehicles invoke application services running in the edge and the cloud to sell traffic/environmental data
mc–$edge_{l1}$–$edge_{l2}$–*cloud*: a service in a *mc* utilizes various services in *cloud* via services in $edge_{l1}$and in $edge_{l2}$	infrastructure costs, transaction acceptance rate and synchronization state	vehicles invoke application services running in the edge and the cloud, e.g. payment for using fast lanes [2]

2.2 Application-required Features for Blockchain-based MECSS

To develop a blockchain-based MECSS, many common IoT, edge and cloud infrastructure- and platform-level features are needed. Let $MEC_F = \{mec_f\}$ represent such features. A feature $mec_f \in MEC_F$ is provided by a component. Thus, we can have $MEC_F = \{mosquitto\_mqtt\_broker, v2xpaas, kafka\}$ as an example where features are provided by the Eclipse Mosquitto MQTT broker[3], Apache Kafka[4] or a V2X Platform-as-a-Service [25]. These components are deployed and run as microservices, leading to the set of service instances $MEC_S = \{s(mec_f)_i\}$. With blockchain-based MECSS, such services will be integrated with blockchain features. From the application viewpoint, blockchain

[3] `https://mosquitto.org/`
[4] `https://kafka.apache.org/`

can be used as composite or individual features based on primitive operations, like *creating*, *signing*, *validating*, *mining*, and $verifying$ [10]. We work at the blockchain features level for different interactions of MECSS. Generally, we consider $BC_F = \{bcf_i\}$ as a set of application-specific blockchain features that should be benchmarked. For example, in the experiments of this paper we focus on BC_F= {*creator*, *consensus*, *all*} where *creator* is a composite feature including operations *creating*, *signing* and *submitting* a transaction, *consensus* is atomic feature reflecting the *mining* process, and *all* means all needed blockchain operators. To use blockchain we will need several blockchain nodes, $BC_N = \{bc_n\}$, offering bcf_i. For testing purposes, BC_F and concrete BC_N will be mapped into concrete blockchain implementations. For example, let $BC_N = \{standard_n, miner_n\}$, we can have $standard_n \in \{Client_{hyperledger}, PeerNodes_{hyperledger}, ETHNode_{eth}\}$ whereas $miner_n \in \{OrdererNode_{hyperledger}, MinerNode_{eth}\}$; these roles/nodes specified in Ethereum[5] and Hyperledger Fabric [5].

We define $MECSS = \{MEC_S, BC_N\}$. For a benchmark, we have a $MECSS$ deployment including MEC_S and BC_N executed in a testbed deployment of a set of containers or VMs across edge and cloud infrastructures. Given blockchain features and interactions, we need to consider main couplings between blockchain features and interactions. Such couplings are based on the design of blockchain-based software systems. For example, let C_{MEC}={*mc*,$edge_{l1}$,$edge_{l2}$,*cloud*} be the nodes in the previous discussed MEC model and let a set BC_F = {*creator*, *consensus*} be the blockchain features. In order to perform benchmark, we will need to have a specification of couplings (c,f), whereas $c \in C_{MEC}$ and $f \in BC_F$.

2.3 Benchmarked metrics

When developing blockchain features for MECSS, there are many metrics that the developer wants to consider [40, 37, 11, 32, 1]; such metrics are important and they have been well-documented in the literature. Table 2 shows examples of important metrics, which should be assessed. Besides these common metrics, the developer usually needs different types of metrics for different requirements. Thus, the developer should be able to customize possible metrics to be tested/measured. In our framework, we do not define such metrics but assume that benchmark functions (see Section 3) will implement suitable metrics defined in the literature. This way allows us to extend benchmark functions even for evaluating application-specific metrics, instead of evaluating features of specific blockchain frameworks without application interaction patterns.

2.4 MECSS under test and infrastructures

A benchmark for interactions in MECSS needs a suitable MEC infrastructure. Resources for a MECSS and their dependencies are also considered based on application requirements. Realistically, all components should be deployed in

[5] `https://docs.ethhub.io/using-ethereum/running-an-ethereum-node`

Table 2. Examples of importance metrics for interactions

Metrics	Description
Transaction (Tx) Acceptance Time	the ratio of accepted transactions to the total transactions submitted.
Synchronization State	the number of nodes, which lost their synchronization.
Transaction (Tx) Acceptance Rate	the number of the submitted transactions accepted by blockchain
Infrastructure Resource Utilization	typical CPU, memory, bandwidth of infrastructures
Cost	Cost for blockchain operations and for other related operations and infrastructures

the real systems suitable for different scenarios. However, practically, currently it is impossible to have the realistic infrastructure for MEC, especially for large-scale and complex topologies of MEC resources. Therefore, we apply symbiotic engineering principles for IoT Cloud systems by having real resources combined with emulation components:

- *mc*: emulated with multiple configurations using container with resource constraints or powerful machines.
- $edge_{l1}$: emulated by VMs/containers with reasonable capabilities, as it is resource-constrained in real deployments.
- $edge_{l2}$ and *cloud*: based on real systems using micro data centers or cloud resources.

The developer can use existing cloud and edge providers, e.g., Google Compute Engine[6], for running benchmarks.

2.5 Coupling software and infrastructure artefacts

Figure 2 presents a deployment topology model of artefacts and their configurations for benchmarks. The deployment model is used to define suitable deployments for specific application needs w.r.t. MECSS and blockchain-based interaction evaluation. Resources are defined by `ContainerConfiguration` and `NetworkConfiguration`. `BlockchainArtefact` and `ApplicationArtefact` are used to describe blockchain and application-specific features, respectively. The deployment topology model is crucial for understanding required software artefacts and corresponding configurations that will be deployed into a suitable testbed for evaluating interactions. For example, a `Node` will specify software artefacts to be deployed in a specific *mc*, $edge_{l1}$, $edge_{l2}$ or *cloud*, including blockchain software and other services, whereas `ContainerConfiguration` specifies information about the container environment hosting such software artefacts. `Nodes` are connected to create a topology of machines (a test infrastructure) through suitable network configuration (specified by `NetworkConfiguration`).

[6] https://cloud.google.com/compute/docs/instances/

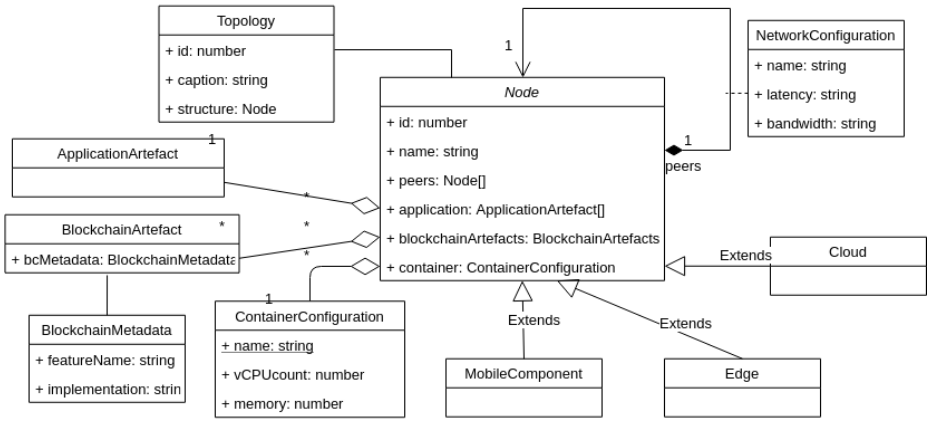

Fig. 2. Relationships between software and infrastructural artefacts

Based on this deployment model, we will define deployment configurations, explained in Section 3.3.

3 Framework for Benchmarks

3.1 Overview

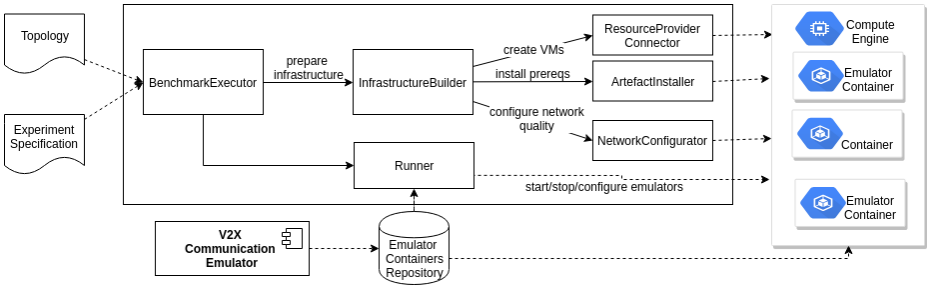

Fig. 3. Main components of the benchmark framework

Figure 3 outlines the architecture of our framework for developing and executing benchmarks for MECSS. Inputs for benchmarks are specified via (i) topology (see Section 2.5) and (ii) experiment specification. *BenchmarksExecutor* performs the deployment and execution of emulator containers in suitable infrastructures.

– **deployment and configuration**: *BenchmarkExecutor* uses *InfrastructureBuilder* to invoke *ResourceProviderConnector* to acquire virtual machines

(VMs). After acquiring VMs, *ArtefactDeployer* deploys application and blockchain artefacts according to the topology specification. Depending on the topology, a VM can host multiple container-based services running artefacts. *InfrastructureBuilder* utilizes *NetworkConfigurator* to set the configuration of network connections between nodes in the deployed infrastructure.

– **benchmark execution**: *BenchmarkExecutor* invokes *Runner* to start the dockerized component *InteractionEmulator* in each node in the deployed topology. *InteractionEmulator* is responsible for running the benchmark, which carries out blockchain operations. When all *InteractionEmulator* finish their running experiments, the framework will download the logs and results of the experiment from all nodes across the deployed topology. Finally, *Runner* will stop docker containers hosting *InteractionEmulator* and stop blockchain nodes and virtual machines.

InteractionEmulator includes all necessary features for emulating application-level blockchain interactions. This component basically emulates or implements interactions of exchanging data over blockchain. Thus, it emulates both data source and receiver in MECSS by connecting blockchain nodes and performing blockchain operations. The developer can select, customize and utilize suitable *InteractionEmulator* for application designs. During the benchmarks operations are monitored to collect data for analyzing metrics. *InteractionEmulator* is a customized component, in which the developer would emulate/test blockchain-based operations for suitable interactions using benchmark functions (see Section 3.2). Several *InteractionEmulator* can be managed in *EmulatorContainersRepository*. For example, in our experiment, a `V2XCommunicatorEmulator` is used (see Section 4) for benchmarking interactions in V2X. The use of *InteractionEmulator* allows us to add new functions for testing a different, new type of interactions.

Figure 4 shows some internal details to describe how the framework can be integrated with different blockchain implementations and underlying edge/-cloud infrastructures through a plugin architecture. For *InfrastructuralBuilder*, three different classes of plugins are used to deal with artefact deployment/undeployment, virtual machines provisioning and network configuration. For specific blockchain implementations, we will have to build suitable deployment/undeployment components, such as `HypFabTopologyDeployer` and `EthereumTopologyDeployer` for Hyperledger Fabric and Ethereum, respectively. This way will allow us to eventually extend the work to support other blockchain systems. For infrastructures, based on `ICloudVMService` we could build connectors to suitable infrastructure providers, enabling benchmarking and testing using different edge and cloud resources. Similar to artefacts and infrastructures deployment/undeployment, different plugins for running emulators performing benchmarks have to be developed for suitable blockchain features. For example, `RunnerHypFab` and `RunnerEthereum` are used for suitable emulators running in Hyperledger Fabric and in Ethereum, respectively. Using common interfaces and plugins design, we could extend and include new benchmark functions for other blockchain technologies.

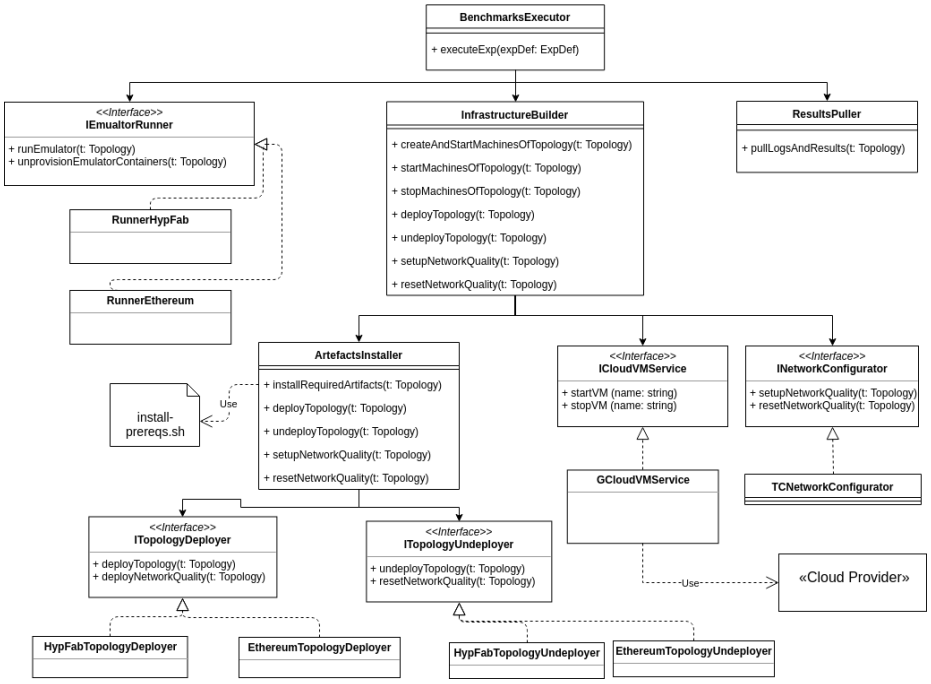

Fig. 4. Main classes for plugins in the framework

3.2 Benchmark Functions

One of the key features is to support the development of benchmark functions based on end-to-end interactions in the design of the developer. As discussed in the previous subsection, such benchmark functions are encapsulated into emulators which are executed as container-based microservices. In our approach we enable the developer to develop them via *InteractionEmulator* containers.

In our framework, the developer writes required functions based on *InteractionEmulator* templates, then the framework will take *InteractionEmulator* to run it. Key functions that the developer has to develop:

- emulated data and parameters for using data: data can be loaded, e.g., from files. Examples of parameters are sending/reading frequencies and number of records to be used.
- key benchmark functions: perform end-to-end test operations and report metrics

These functions are configured with existing features of the framework to build container-based emulators.

3.3 Benchmark Experiment Configuration

To allow the benchmarks of various types of interactions in different configurations, we enable the developer to define experiment specifications in YAML. The experiment specification includes information about the topology of infrastructural resources where blockchain artefacts will be deployed. `Node` to indicate infrastructural resources and blockchain artefacts. The experiment specification will also include other types of information, such as different possible deployments and blockchain features.

3.4 Prototype

In our current prototype, we use Google Cloud for infrastructural compute resources. We have used `geth` image[7] for Ethereum, and official images for peer, orderer, tools, certificate authority, `Kafka` and `Zookeeper` provided by Hyperledger Fabric[8]. *NetworkConfigurator* uses `tc`[9] to manipulate the traffic control. Concrete *InteractionEmulator* are implemented in typescript and `nodejs`, running in a `docker` container. In general, most of the components, like *InteractionEmulator* and testbed connectors can be extended to implement different types of interactions and use different types of resources. Our prototype is an open source available in GitHub at `https://github.com/rdsea/kalbi/tree/master/benchmark_framework`.

4 Experiments

4.1 Flexibility in Benchmark Configurations

One of the key requirements is the flexibility in configuring benchmarks that we illustrate in this section through examples of creating experiments through the parameterization of experiment configurations. For experiments, we use the V2X Communication scenarios whereas *mc* is a `vehicle`, $edge_{l1}$ is a `RSU`, $edge_{l2}$ is a resource in an edge data center, and *cloud* is a public cloud resource. We implement `V2XCommunicatorEmulator` for experiments. There are many combinations of MEC and blockchain features and deployment topologies, thus we have many MECSS deployments that we cannot present here all[10]. Table 3 shows examples of deployments; note that each types of interaction has different deployments of features. Table 4 gives dynamic configuration for testbed deployment. All machines are with Intel Xeon E5, Sandy Bridge 2.6GHz and Ubuntu 18.04. Network configurations for the experiments are varied, shown in Table 5. For each of deployments, we can select suitable configurations for emulating *mc*, $edge_{l1}$,

[7] `https://github.com/ethereum/go-ethereum`
[8] `https://github.com/hyperledger/fabric`
[9] `http://manpages.ubuntu.com/manpages/bionic/man8/tc.8.html`
[10] Detailed deployment configurations and logs as well as benchmark results can be found at: `https://github.com/rdsea/kalbi/tree/master/experiments`

$edge_{l2}$ or *cloud*. Furthermore, experiments are also configured with different blockchain implementations. Due to the flexibility of our framework, it is easy to create several experiments. For example, when focusing on a topology called *large scale* where we have max 48 *mc* and (i) for interaction *mc*–$edge_{l1}$–*mc*: one $edge_{l1}$, (ii) for interaction *mc*–$edge_{l2}$–*mc*: one $edge_{l2}$, and (iii)for interaction *mc*–$edge_{l1}$–$edge_{l2}$–*mc*: one $edge_{l2}$, two $edge_{l1}$, we carried out 180 experiments for Ethereum and 144 experiments for Hyperledger Fabric.

Table 3. Example of the deployment of blockchain features for interaction 4 – *mc*–$edge_{l1}$–$edge_{l2}$–*mc*

Blockchain features deployment				
Deployment ID	*mc*	$edge_{l1}$	$edge_{l2}$	*cloud*
0	*creator*	*consensus*	*creator*	-
1	*creator*	*all*	*creator*	-
2	*creator*	*creator*	*all*	-
3	*creator*	*creator*	*consensus*	-
4	*creator*	*consensus*	*consensus*	-
5	*creator*	*all*	*consensus*	-
6	*creator*	*consensus*	*all*	-
7	*creator*	*all*	*all*	-
8	*all*	*all*	*all*	-
9	*all*	*consensus*	*all*	-
10	*all*	*creator*	*all*	-
11	*all*	*creator*	*creator*	-
12	*all*	*consensus*	*creator*	-
13	*all*	*all*	*creator*	-
14	*all*	*creator*	*consensus*	-
15	*all*	*consensus*	*consensus*	-
16	*all*	*all*	*consensus*	-

Table 4. Example of VM configurations for running containers

Component	**CPU**	**RAM**	**Storage (GB)**
cloud	4vCPU	16 GB	60 SSD
$edge_{l2}$	4vCPU	16 GB	60 SSD
$edge_{l1}$	1vCPU	2 GB	16 SSD
mc (light)	1vCPU	2 GB	20 SSD
mc (medium)	2vCPU	4 GB	20 SSD
mc (big)	4vCPU	8 GB	20 SSD

Table 5. Example of used network configurations

Type	**Latency**	**Bandwidth**
3G	200ms	1000kbps
4G	100ms	10000kbps
5G	5ms	54mbps

4.2 Insights from Benchmarks for Developers

Table 6 presented a small set of results to illustrate the richness of our framework. Several results can be obtained through the framework by parameterizing

experiment specifications. Our goal is to illustrate the framework. In the following we will only focus on few aspects of how the developer could benefit from our methods and framework.

Table 6. Examples of best benchmarks results

Input			Result	Performance				Reliability		
				Transaction Acceptance Time				Transaction Acceptance Rate		
Interaction Id	Blockchain framework	Scale	Experiment ID	*Minimum Time*	*Maximum Time*	*Median Time*	*Average Time*	*Accepted count*	*Failed count*	Not-sync nodes count
2	Ethereum	small	24	240	6510	2316.5	2507.7	200	0	0
2	Ethereum	large	94	979	23013	5691.5	6617.78	4800	0	0
2	Hyperledger Fabric	small	188	2044	2072	2054	2054.45	200	0	0
2	Hyperledger Fabric	large	272	50	6956	124	487.18	4800	0	0
3	Ethereum	small	53	474	5120	1862	2153.24	200	0	0
3	Ethereum	large	125	429	16774	3698.5	4444.62	4800	0	0
3	Hyperledger Fabric	small	233	2038	2064	2049	2049.55	200	0	0
3	Hyperledger Fabric	large	302	49	5708	154	494.91	4800	0	0
4	Ethereum	small	61	199	4914	2151.5	2188.34	200	0	0
4	Ethereum	large	149	306	47711	5218.5	6529.1	4618	182	0
4	Hyperledger Fabric	small	260	2041	2063	2048.5	2050.21	200	0	0

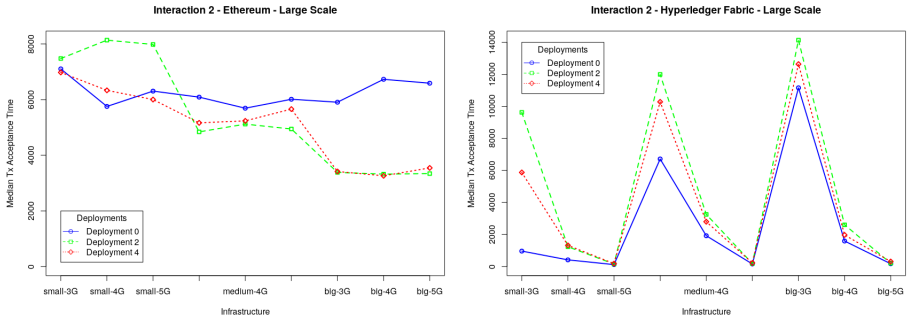

Fig. 5. Medians of transaction acceptance times for interaction 2 (mc–$edge_{l1}$–mc)

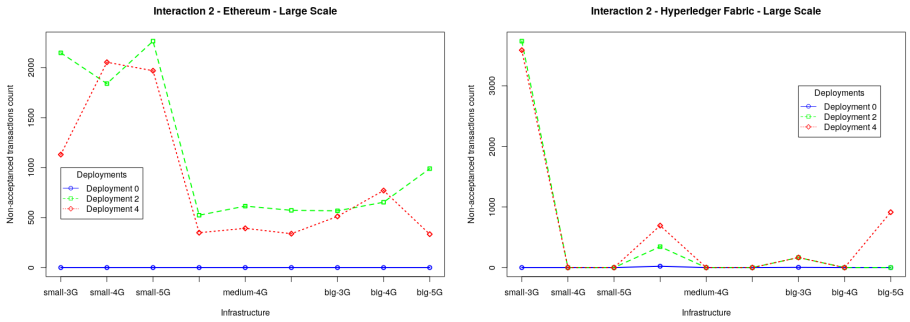

Fig. 6. Number of rejected transactions for interaction 2 (mc–$edge_{l1}$–mc)

Interaction 2 (mc–$edge_{l1}$–mc) Figures 5 and 6 depict a dependency between the infrastructure and performance and reliability respectively, among all

blockchain deployments for this interaction. For a large scale topology, the experiments didn't achieve 100% reliability, except deployment 0 (where *mc* is configured with *creator* and $edge_{l1}$with *all* blockchain features). However, the deployment 0 is centralized (consensus algorithm is running only in $edge_{l1}$(RSU)). For deployment 2 (*mc* with *all* and $edge_{l1}$ with *creator* blockchain features) and deployment 4 (*mc* and $edge_{l1}$ with *all* blockchain features) results is strongly dependent on the used infrastructure. However, for a better resource configuration for *mc*, we obtained a worse performance. It is possible that *mc* created more transactions when utilizing more powerful configuration, leading to longer synchronization time. Hyperledger Fabric has shown better results for the large scale (the best deployment is for experiment 272, not shown in the paper).

Considering that the goal of a developer is to find a single deployment and infrastructure configuration. Deriving from several benchmark results and the above observations, the developer might choose Hyperledger Fabric, deployment 0, small VM type for *mc* and 5G network, as the best results concerning reliability and performance. However, deployment 0 makes the network partially centralized, which violates the principles of blockchain.

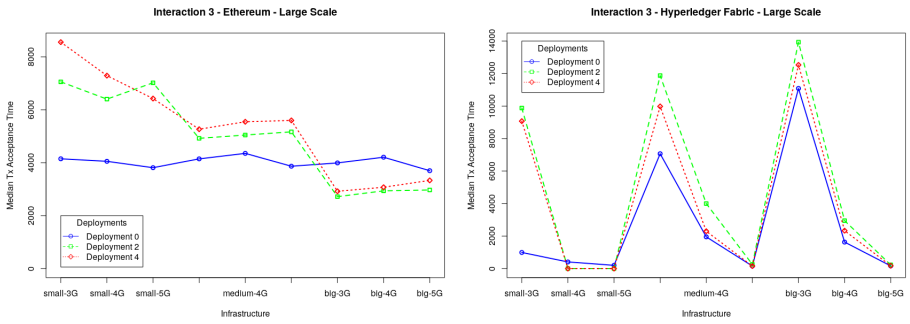

Fig. 7. Medians of transaction acceptance times for interaction 3 (mc–$edge_{l2}$–mc)

Interaction 3 (mc–$edge_{l2}$–mc) The performance depicted in Figure 7 follows similar patterns as in the previous interaction for both blockchain implementations and topology scales. In deployment 2 (*mc* with *all* and $edge_{l2}$ with *creator* blockchain features) we observed an increasing number of rejected transactions for the big machine types for *mc* (vehicles). For all other deployments in the large scale, we have similar patterns as in previous interactions.

From the benchmark, the developer can use deployment 4 (both *mc* and $edge_{l2}$ with *all* blockchain features) with big machine type for *mc* and 5G network. With deployment 0 (*mc* with *creator* and $edge_{l2}$ with *all* blockchain features), medium machine types for *mc* and 5G network we got the best results. From all benchmarks for this interaction, our framework could provide a hint to the developer that Hyperledger Fabric, deployment 0, medium machine type and 5G network is considered as the best deployment and configuration.

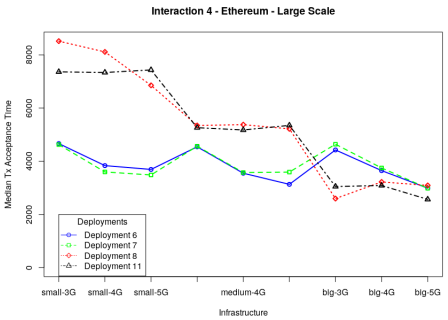

Fig. 8. Median of transaction acceptance times for interaction 4 (mc–$edge_{l1}$–$edge_{l2}$–mc)

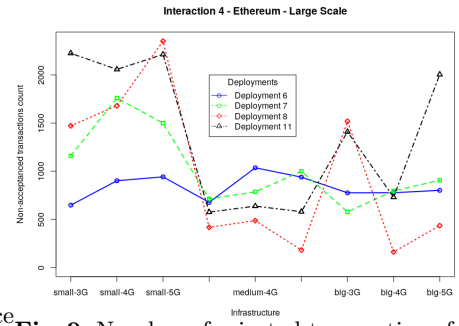

Fig. 9. Number of rejected transactions for interaction 4 (mc–$edge_{l1}$–$edge_{l2}$–mc)

Interaction 4 (mc–$edge_{l1}$–$edge_{l2}$–mc) Figure 8 and Figure 9 showed performance and reliability results, respectively, for this interaction with Ethereum (no Hyperledger Fabric experiments for this scale). Concerning the reliability, the results showed a dependency between infrastructure and number of rejected transactions. In this interaction we achieved the best results for experiment 149 (deployment 8, medium machine types for mc and 5G network).

Additional discussion We notice that experiments with different underlying blockchain systems show different performance. In our work, we did not investigate the underlying reasons for the difference. The reason is that it is not our goal to compare different blockchain internal structures and this comparison in MEC will also require intensive work on analyzing blockchain internals and their relationships with MEC middleware and applications. Some related works have studied intensively internal properties of Ethereum and Hyperledger Fabric [11, 20, 9]. Nevertheless, from the list of experiments, the best configuration from our experiments is based on parameters set in the experiment configuration and might not be the best for the requirements of MECSS designed by the developer (e.g., in V2X, requirements from safety regulation). This raises the importance of having benchmarks with different customization capabilities for the developer.

5 Related Work

There is no lack of survey papers about blockchain, edge computing and IoT, such as [4, 14, 26, 3, 33]. In MEC various papers have addressed different issues of blockchain and presented various scenarios [29, 35]. Unlike these works, we focus on benchmarking blockchain features with MECSS from software development viewpoint, especially we focus on application-level interactions in an end-to-end view. There are general blockchain benchmark tools[11] but they are

[11] e.g., `https://github.com/dsx-tech/blockchain-benchmarking` and `https://github.com/hyperledger/caliper`

focused on understanding specific blockchain systems in general view - not in application-level interactions and generic frameworks for blockchain-based interactions in connection with other relevant software services. Several papers have performed benchmarks and testing of blockchain and blockchain networks [11, 32, 34, 19]. Generally, they share key issues with our paper w.r.t. metrics and software components. However, our paper focuses on generic benchmarks of blockchain features within MECSS so we can provide a distinguished feature for the need of the developer. Note that in our work, we do not focus on definition of metrics to be benchmarked, like [1, 11, 32]. Instead, we provide utilities for developers to write core benchmark functions and we enable the execution and deployment of benchmarks. Another difference of our framework is the design of *InteractionEmulator* and connectors to allow the developer to create new configurations and benchmark functions to support new types of application-level interactions, which might exist in specific application designs.

The work [22] specifically develops blockchain for V2X and has also performed simulation and benchmark. However, it is a typical benchmark for systems developed for V2X. In our work, we abstract the MECSS to focus on application-level interactions carried out through different services, creating a generic framework for benchmarking blockchain interactions in MEC; we leverage V2X scenarios only for testing our framework.

6 Conclusions and Future Work

In this paper we present a framework for developing and executing benchmarks for blockchain-based mobile edge computing software systems. We have focused on important application-level interactions and metrics for realistic deployment of MECSS. Our framework is generic enough to cover various aspects of deployments and software features. Benchmarking functions for interactions can be defined in *InteractionEmulator* that can be extended, whereas the design of connectors for blockchain, common and infrastructural services allows us to deploy and integrate with different software systems and resources. The contribution is not about specific benchmark values for blockchain but the capabilities to perform benchmarks in a generic way for application specific interactions.

In our current work, we focus on improving the prototype and extending metrics as plugins as well as to support fine-grained couplings of MEC features and blockchain features for MECSS. We are also integrating the benchmark framework and benchmark data with a service for managing knowledge from benchmarks [30] to enable the reuse of knowledge from benchmarks for MECSS design and implementation.

Acknowledgment Filip Rydzi work was performed as a part of his study at TU Wien. Partial results of this paper were also reported in his master thesis. We thank Google Cloud Platform Research Credits Program for supporting computing resources.

References

1. Hyperledger blockchain performance metrics, https://www.hyperledger.org/wp-content/uploads/2018/10/HL_Whitepaper_Metrics_PDF_V1.01.pdf, last access: 26 May 2019
2. Vehicle-to-vehicle cooperation to marshal traffic, https://patents.google.com/patent/US9928746B1/en
3. Blockchain challenges and opportunities: A survey. Int. J. Web Grid Serv. **14**(4), 352–375 (Jan 2018), http://dl.acm.org/citation.cfm?id=3292946.3292948
4. Ali, M.S., Vecchio, M., Pincheira, M., Dolui, K., Antonelli, F., Rehmani, M.H.: Applications of blockchains in the internet of things: A comprehensive survey. IEEE Communications Surveys Tutorials pp. 1–1 (2018). https://doi.org/10.1109/COMST.2018.2886932
5. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolić, M., Cocco, S.W., Yellick, J.: Hyperledger fabric: A distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference. pp. 30:1–30:15. EuroSys '18, ACM, New York, NY, USA (2018)
6. Bagchi, S., Siddiqui, M.B., Wood, P., Zhang, H.: Dependability in edge computing. Commun. ACM **63**(1), 5866 (Dec 2019). https://doi.org/10.1145/3362068, https://doi.org/10.1145/3362068
7. Bittencourt, L., Immich, R., Sakellariou, R., Fonseca, N., Madeira, E., Curado, M., Villas, L., DaSilva, L., Lee, C., Rana, O.: The internet of things, fog and cloud continuum: Integration and challenges. Internet of Things **3-4**, 134 – 155 (2018)
8. Chakraborty, P., Shahriyar, R., Iqbal, A., Bosu, A.: Understanding the software development practices of blockchain projects: A survey. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 28:1–28:10. ESEM '18, ACM, New York, NY, USA (2018)
9. Chen, T., Li, Z., Zhu, Y., Chen, J., Luo, X., Lui, J.C.S., Lin, X., Zhang, X.: Understanding ethereum via graph analysis. ACM Trans. Internet Technol. **20**(2) (Apr 2020). https://doi.org/10.1145/3381036, https://doi.org/10.1145/3381036
10. Christidis, K., Devetsikiotis, M.: Blockchains and smart contracts for the internet of things. IEEE Access **4**, 2292–2303 (2016). https://doi.org/10.1109/ACCESS.2016.2566339
11. Dinh, T.T.A., Wang, J., Chen, G., Liu, R., Ooi, B.C., Tan, K.L.: Blockbench: A framework for analyzing private blockchains. In: Proceedings of the 2017 ACM International Conference on Management of Data. pp. 1085–1100. SIGMOD '17, ACM, New York, NY, USA (2017)
12. Dorri, A., Steger, M., Kanhere, S.S., Jurdak, R.: Blockchain: A distributed solution to automotive security and privacy. IEEE Communications Magazine **55**(12), 119–125 (Dec 2017). https://doi.org/10.1109/MCOM.2017.1700879
13. Faezipour, M., Nourani, M., Saeed, A., Addepalli, S.: Progress and challenges in intelligent vehicle area networks. Commun. ACM **55**(2), 90–100 (Feb 2012)
14. Ferrag, M.A., Derdour, M., Mukherjee, M., Derhab, A., Maglaras, L.A., Janicke, H.: Blockchain technologies for the internet of things: Research issues and challenges. IEEE Internet of Things Journal **6**(2), 2188–2204 (2019). https://doi.org/10.1109/JIOT.2018.2882794, https://doi.org/10.1109/JIOT.2018.2882794

15. Grewe, D., Wagner, M., Arumaithurai, M., Psaras, I., Kutscher, D.: Information-centric mobile edge computing for connected vehicle environments: Challenges and research directions. In: Proceedings of the Workshop on Mobile Edge Communications. pp. 7–12. MECOMM '17, ACM, New York, NY, USA (2017)
16. Guo, H., Meamari, E., Shen, C.: Blockchain-inspired event recording system for autonomous vehicles. CoRR **abs/1809.04732** (2018)
17. Harding, J., Powell, G., Yoon, R., Fikentscher, J., Doyle, C., Sad e, D., Lukuc, M., Simons, J., Wang, J., et al.: Vehicle-to-vehicle communications: readiness of v2v technology for application. Tech. rep., United States. National Highway Traffic Safety Administration (2014)
18. Kim, N.H., Kang, S.M., Hong, C.S.: Mobile charger billing system using lightweight blockchain. In: 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS). pp. 374–377 (Sep 2017). https://doi.org/10.1109/APNOMS.2017.8094151
19. Kim, S.K., Ma, Z., Murali, S., Mason, J., Miller, A., Bailey, M.: Measuring ethereum network peers. In: Proceedings of the Internet Measurement Conference 2018. pp. 91–104. IMC '18, ACM, New York, NY, USA (2018)
20. Lee, X.T., Khan, A., Sen Gupta, S., Ong, Y.H., Liu, X.: Measurements, analyses, and insights on the entire ethereum blockchain network. In: Proceedings of The Web Conference 2020. p. 155166. WWW 20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3366423.3380103, `https://doi.org/10.1145/3366423.3380103`
21. Li, C., Xue, Y., Wang, J., Zhang, W., Li, T.: Edge-oriented computing paradigms: A survey on architecture design and system management. ACM Comput. Surv. **51**(2), 39:1–39:34 (Apr 2018)
22. Li, L., Liu, J., Cheng, L., Qiu, S., Wang, W., Zhang, X., Zhang, Z.: Creditcoin: A privacy-preserving blockchain-based incentive announcement network for communications of smart vehicles. IEEE Transactions on Intelligent Transportation Systems **19**(7), 2204–2220 (July 2018). https://doi.org/10.1109/TITS.2017.2777990
23. Lu, N., Cheng, N., Zhang, N., Shen, X., Mark, J.W.: Connected vehicles: Solutions and challenges. IEEE Internet of Things Journal **1**(4), 289–299 (Aug 2014). https://doi.org/10.1109/JIOT.2014.2327587
24. Newman, S.: Building Microservices. OReilly Media, Inc., 1st edn. (2015)
25. Nokia: Vehicle-to-everything communication will transform the driving experience, `https://networks.nokia.com/products/vehicle-to-everything`, last access: 27 May 2019
26. Panarello, A., Tapas, N., Merlino, G., Longo, F., Puliafito, A.: Blockchain and iot integration: A systematic survey. Sensors **18**(8), 2575 (2018). https://doi.org/10.3390/s18082575, `https://doi.org/10.3390/s18082575`
27. Porru, S., Pinna, A., Marchesi, M., Tonelli, R.: Blockchain-oriented software engineering: Challenges and new directions. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). pp. 169–171 (2017)
28. Qu, Q., Xu, R., Nikouei, S.Y., Chen, Y.: An experimental study on microservices based edge computing platforms (2020)
29. Rahman, M.A., Hossain, M.S., Loukas, G., Hassanain, E., Rahman, S.S., Alhamid, M.F., Guizani, M.: Blockchain-based mobile edge computing framework for secure therapy applications. IEEE Access **6**, 72469–72478 (2018). https://doi.org/10.1109/ACCESS.2018.2881246
30. Rydzi, F., Truong, H.: Sharing blockchain performance knowledge for edge service development. In: 5th IEEE International Conference on Collaboration and

Internet Computing, CIC 2019, Los Angeles, CA, USA, December 12-14, 2019. pp. 20–29. IEEE (2019). https://doi.org/10.1109/CIC48465.2019.00012, https://doi.org/10.1109/CIC48465.2019.00012
31. Taleb, T., Dutta, S., Ksentini, A., Iqbal, M., Flinck, H.: Mobile edge computing potential in making cities smarter. IEEE Communications Magazine **55**(3), 38–43 (March 2017). https://doi.org/10.1109/MCOM.2017.1600249CM
32. Thakkar, P., Nathan, S., Vishwanathan, B.: Performance benchmarking and optimizing hyperledger fabric blockchain platform. CoRR **abs/1805.11390** (2018)
33. Viriyasitavat, W., Anuphaptrirong, T., Hoonsopon, D.: When blockchain meets internet of things: Characteristics, challenges, and business opportunities. Journal of Industrial Information Integration (2019)
34. Walker, M.A., Dubey, A., Laszka, A., Schmidt, D.C.: Platibart: A platform for transactive iot blockchain applications with repeatable testing. In: Proceedings of the 4th Workshop on Middleware and Applications for the Internet of Things. pp. 17–22. M4IoT '17, ACM, New York, NY, USA (2017)
35. Xiong, Z., Zhang, Y., Niyato, D., Wang, P., Han, Z.: When mobile blockchain meets edge computing. IEEE Communications Magazine **56**(8), 33–39 (August 2018). https://doi.org/10.1109/MCOM.2018.1701095
36. Xu, Q., Mak, T., Ko, J., Sengupta, R.: Vehicle-to-vehicle safety messaging in dsrc. In: Proceedings of the 1st ACM International Workshop on Vehicular Ad Hoc Networks. pp. 19–28. VANET '04, ACM, New York, NY, USA (2004)
37. Xu, X., Weber, I., Staples, M., Zhu, L., Bosch, J., Bass, L., Pautasso, C., Rimba, P.: A taxonomy of blockchain-based systems for architecture design. In: 2017 IEEE International Conference on Software Architecture (ICSA). pp. 243–252 (April 2017). https://doi.org/10.1109/ICSA.2017.33
38. Xu, X., Weber, I., Staples, M.: Architecture for Blockchain Applications. Springer (2019)
39. Zhang, L., Luo, M., Li, J., Au, M.H., Choo, K.K.R., Chen, T., Tian, S.: Blockchain based secure data sharing system for internet of vehicles: A position paper. Vehicular Communications **16**, 85 – 93 (2019)
40. Zheng, P., Zheng, Z., Luo, X., Chen, X., Liu, X.: A detailed and real-time performance monitoring framework for blockchain systems. In: 2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP). pp. 134–143 (2018)