Future Generation Computer Systems 160 (2024) 76–91

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

LIGHT-HIDRA: Scalable and decentralized resource orchestration in Fog-IoT environments

Carlos Núñez-Gómez [a,*], Martijn de Vos [b], Jérémie Decouchant [c], Johan Pouwelse [c], Blanca Caminero [a], Carmen Carrión [a]

[a] University of Castilla-La Mancha (UCLM), Albacete, Spain
[b] École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
[c] Delft University of Technology, Delft, The Netherlands

ARTICLE INFO

Keywords:
Decentralized resource orchestration
Fog-IoT environments
Workload scheduling
Decentralized systems
Byzantine reliable broadcast

ABSTRACT

With the proliferation of Internet of Things (IoT) ecosystems, traditional resource orchestration mechanisms, executed on fog devices, encounter significant scalability, reliability and security challenges. To tackle these challenges, recent decentralized algorithms in Fog-IoT use Distributed Ledger Technologies to orchestrate resources and payments between peers. However, while distributed ledgers provide many desirable properties, their consensus mechanism introduces a performance bottleneck. This paper introduces LIGHT-HIDRA, a consensus-less and decentralized resource orchestration system for Fog-IoT environments. At its core, LIGHT-HIDRA uses Byzantine Reliable Broadcast (BRB) to coordinate actions without centralized control, therefore drastically reducing communication overhead and latency compared to consensus-based solutions. LIGHT-HIDRA coordinates the scheduling and execution of workloads, and securely manages the payments that peers receive for dedicating resources to workloads. LIGHT-HIDRA further increases performance and reduces overhead by grouping peers into distinct domains. We conduct an in-depth analysis of the protocol's security properties, investigating its efficiency and robustness in diverse situations. We evaluate the performance of LIGHT-HIDRA, highlighting its performance over HIDRA, a state-of-the-art baseline that uses smart contracts. Our experiments demonstrate that LIGHT-HIDRA reduces the bandwidth usage by up to 57x, the latency of workload offloading by up to 142x, and shows superior throughput compared to HIDRA.

1. Introduction

The explosive growth of Internet-of-Things (IoT) ecosystems has upsurged the volume of data generated by IoT devices such as cameras and sensors [1]. To process and analyze this massive influx of IoT data, the fog computing paradigm is increasingly being used [2]. The key idea behind fog computing is to process as much data as possible on (edge) devices themselves, close to where the data is produced, therefore avoiding latency induced by edge-cloud data communication. Resource allocation in fog computing has become a critical yet complex task, compounded by the need for efficient mechanisms capable of handling potentially many resource-constrained devices that might also be faulty, unresponsive, or malicious (Byzantine) [3,4]. While centralized orchestration solutions such as Kubernetes are common solutions, they must apply complex techniques to support high availability (and avoid single points of failure) and tackle trust issues, specially in multi-domain environments [5].

Decentralized resource orchestration in Fog-IoT settings has emerged as a promising alternative to centralized approaches [6–8]. In this setting, fog nodes coordinate the resource orchestration amongst themselves without the need for centralized coordination. An increasing amount of work in this domain leverage the capabilities of Distributed Ledger Technologies (DLT) and smart contracts which provides principles of autonomy, transparency, and resilience [9–12]. However, the need to reach consensus to ensure the integrity of distributed ledgers often becomes a significant performance bottleneck, introducing overhead and latency that undermine system efficiency [13,14].

To overcome this bottleneck, this work introduces LIGHT-HIDRA, a decentralized mechanism for resource orchestration in Fog-IoT environments based on Byzantine broadcast primitives. LIGHT-HIDRA is efficient, lightweight, and scalable, and operates in Fog-IoT environments in which devices may be resource-constrained, numerous and

* Corresponding author.
E-mail addresses: carlos.nunez@uclm.es (C. Núñez-Gómez), martijn.devos@epfl.ch (M. de Vos), j.decouchant@tudelft.nl (J. Decouchant), J.A.Pouwelse@tudelft.nl (J. Pouwelse), mariablanca.caminero@uclm.es (B. Caminero), carmen.carrion@uclm.es (C. Carrión).

https://doi.org/10.1016/j.future.2024.05.041
Received 10 August 2023; Received in revised form 26 March 2024; Accepted 19 May 2024
Available online 25 May 2024

malicious. Light-HIDRA circumvents the need for consensus, a common scalability barrier in DLT-based solutions. Instead, our mechanism relies on Byzantine Reliable Broadcast (BRB) [15] to synchronize and coordinate workloads among peers, as well as to manage payments that peers receive for the resources they dedicate to workloads. Relying on BRB guarantees the robustness and reliability of Light-HIDRA, while significantly reducing communication overhead and latency compared to consensus-based approaches. We show in this work that key components of resource orchestration, including workload scheduling, workload execution and payments, can be implemented based on BRB. Through our performance evaluation, we demonstrate that Light-HIDRA is not only a theoretical construct, but is also an efficient, robust, and practical solution for resource orchestration in Fog-IoT environments, in particular compared to consensus-based approaches.

In summary, the contributions of this work are four-fold:

1. We introduce Light-HIDRA, a novel, secure and scalable solution for decentralized resource orchestration in Fog-IoT environments. Light-HIDRA includes BRB-based sub-protocols for workload scheduling, execution, monitoring and payments (Sections 3 and 4).
2. We provide a security analysis of Light-HIDRA, highlighting its resilience against Byzantine attacks and system failures (Section 5).
3. We implement Light-HIDRA and open-source its implementation.
4. We compare the performance and scalability of Light-HIDRA against that of HIDRA, a state-of-the-art baseline in the field (Section 6). Our experiments show that Light-HIDRA significantly reduces bandwidth usage, latency and throughput.

2. Byzantine Reliable Broadcast (BRB)

Byzantine Reliable Broadcast (BRB) is a key distributed computing abstraction that provides reliable message dissemination even in the presence of a limited number of Byzantine faults. Such Byzantine faults refer to arbitrary and malicious behaviors exhibited by faulty peers in a distributed system. The BRB protocol has a sending peer that broadcasts a message to all other peers in the system and guarantees that if the sender is correct then all correct peers will eventually deliver its message. The fault tolerance of BRB ensures that the algorithm can cope with up to a certain threshold of faulty peers f, where the threshold depends on n, the number of total peers in the system. In particular, Bracha's BRB protocol assumes less than $\frac{n}{3}$ faulty peers in asynchronous networks [15–18]. The BRB primitive has already been applied by the scientific community in other works to avoid consensus. Recently, it has been shown that it can be used to coordinate cryptocurrency payments in a decentralized way [19], and multi-hops payments in payment channel networks [20]. In an earlier work, it was also used as an underpinning primitive for data replication between servers [21].

BRB emphasizes reliable message dissemination, ensuring totality and guaranteeing that all correct peers eventually agree on the broadcast messages they deliver (i.e., even when their sender is faulty). A BRB round involves several steps of message exchange (also illustrated in Fig. 1): (1) an initial *SEND* step in which the sender peer broadcasts a message to the other peers, (2) an all-to-all *ECHO* step that the correct peers employ to ensure consistency of the sender's message, and (3) an all-to-all *READY* step to indicate the willingness of the correct peers to deliver the sender's message, thus ensuring totality. Note that the BRB algorithm requires the collaboration of Byzantine quorums of participating peers ($2f + 1$ peers) to reach agreement and deal with Byzantine faults. Peers receiving a threshold of $2f + 1$ *ECHO* or *READY* messages will be enabled to deliver the sender's message. Moreover, BRB implements an amplification step for sending *READY* messages that is key to achieving the totality property: if a correct peer receives $f + 1$ *READY* messages but has not yet sent its *READY* message, then that peer will be enabled to send its *READY* message.

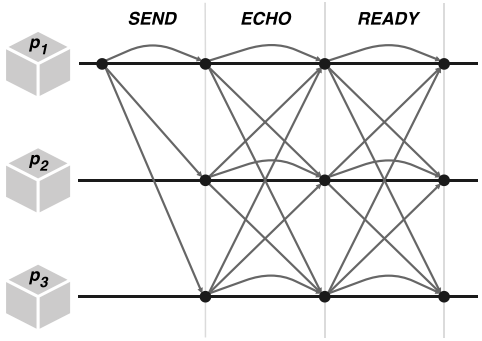


Fig. 1. Bracha's Byzantine Reliable Broadcast. The *ECHO* and *READY* steps are all-to-all communication phases that ensure the consistency and totality properties.

For a better understanding of the security properties to be achieved by Light-HIDRA, we summarize the properties enforced by a BRB algorithm [22]:

- **Validity.** If a correct peer broadcasts a message, then all correct peers eventually deliver it.
- **No duplication.** Correct peers deliver a message at most once.
- **Integrity.** If a correct peer delivers a message from a given sender, then the message was indeed sent by that sender.
- **Consistency.** All correct peers deliver the same message.
- **Totality.** If a message is delivered by any correct peer, then all correct peers eventually deliver the message.

3. System overview and threat model

We now introduce our system and threat models, and state the assumptions made in this work.

Architecture. Fig. 2 shows an overview of Light-HIDRA. We partition the system into multiple domains that work cooperatively on task execution, in line with other related works in the literature [10,19,23]. This partitioning (also called sharding) improves the performance of decentralized systems as agreement is often only required amongst small group of peers instead of network-wide [24]. Each peer is part of exactly one domain, its *parent domain*. Light-HIDRA considers a total of n peers participating in the system that are divided into m different domains. We refer to the set of participating peers as N and to the set of domains as M. Domains in Light-HIDRA are composed of peers related to a particular Fog-IoT environment, e.g., peers belonging to an inter-campus computing group or to a global P2P resource offloading marketplace. Peers first try to offload workloads to peers in their parent domain and then offload workloads to other domains if unsuccessful. Simultaneously, peers offer their own resources to others in the system. This conceptually turns each domain into a decentralized computing cluster in which peers offer finite amounts of resources to execute workloads.

Domain and peer bootstrapping. In its current form, Light-HIDRA targets controlled Fog-IoT environments where there is an administrative entity per domain. In those settings, a peer registration phase and a bootstrapping phase manage the participation of peers in the system domains. When a new peer p wants to join a domain, it must apply for registration in advance by submitting its network address, public key, and the maximum amount of resources it is willing to provide. When p completes the registration process, it joins a parent domain $M_p \in M$. The specific parent domain that a peer joins depends, for example, on geographical location or administrative domain (such as a building on a university campus). Peers must have a parent domain where they can offer their resources and can receive payments by executing workloads of other peers. All peer identities and their domain membership are published as predefined lists that can be accessed externally by any

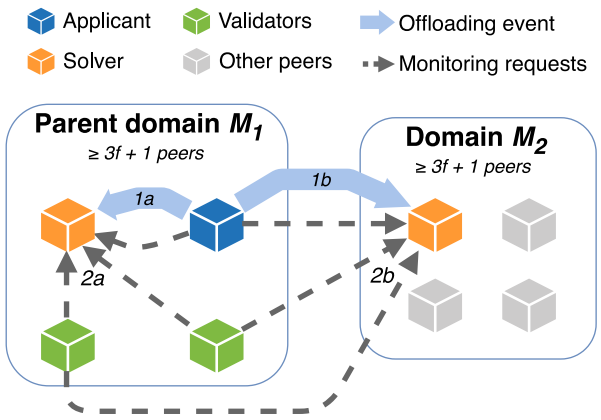


Fig. 2. Light-HIDRA's architecture, roles and interactions. Each peer is part of exactly one *parent domain* and we assume that each domain contains at least $3f+1$ peers and at most f Byzantine peers.

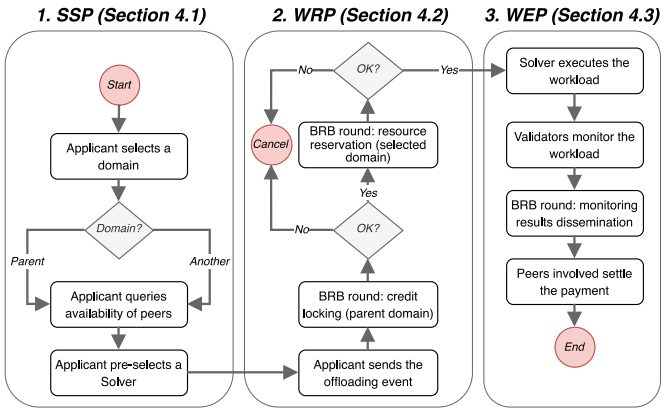


Fig. 3. Overview of the Light-HIDRA workflow, including the different phases of the protocol and sections where these are described.

potential user, e.g., via public repositories or websites. For the development of the prototype and testing of Light-HIDRA, we assume that all peers in the system know in advance this information about the other peers as this can easily be shared out of band [19].

Roles and interactions. Peers have the flexibility to assume distinct roles over time, including being an *Applicant*, *Solver* or *Validator*. Furthermore, peers can simultaneously undertake multiple roles based on the scheduling requests or *offloading events* they engage in. *Applicant* peers offload their workloads to other peers in the system. *Solver* peers execute the workloads created by Applicant peers. The selection of the Solver peer that will handle a particular workload is performed by Applicants themselves. This pre-selection process by itself does not guarantee that the workload will eventually be executed by the selected Solver: the Applicant's payment must first be confirmed and the Solver's resources must be reserved prior to execution. Finally, *Validator* peers monitor whether the workload is actually being executed by the Solver. Fig. 2 shows the roles and interactions of Light-HIDRA and outlines the protocol for workload scheduling: an Applicant (blue cube) sends an offloading event to a Solver (orange cube) either intra-domain (1a) or inter-domain (1b) depending on the local availability of candidate Solver peers. Once the payment is confirmed and the resources are reserved, peers in the Applicant domain become Validators (green cubes) and monitor the workload deployed on the local (2a) or remote (2b) Solver peer. Grey cubes illustrate peers that are not involved in the current offloading event.

Payments. Rewarding Solver peers for their work is an important part of resource orchestration [25,26]. In line with other work [27,28], Light-HIDRA integrates a payment system where Applicant peers pay credits to Solver peers for the work they perform. The amount of credits offered by an Applicant peer will depend on the execution time requested for its workload and on a price per unit of time (see Section 4.2 for more details). We assume that credits are assigned to peers by an external administrator, and that peers can convert their credits for some fiat currency using an out-of-band mechanism. Note that in order to fairly carry out offloading events for both Applicants and Solvers and to minimize risks, Light-HIDRA includes a partial payment system that allows Validators to lock and settle payments gradually according to the monitoring results obtained.

Threat model. Correct peers in domains strictly follow the protocol rules. Peers might also crash, and a limited number of peers per domain might behave maliciously. We assume the presence of adversaries that attempt to corrupt the system, for example, by allowing duplicated messages, compromising the integrity and consistency of the information exchanged, omitting some protocol rules, etc. More specifically, Light-HIDRA takes into account the existence of at most f faulty peers per domain, assuming that each domain in the system contains at least $3f+1$ peers. Therefore, an adversary cannot corrupt more than f peers per domain in order to maintain the protocol correctness [15]. In addition, Applicants and Solvers may act as faulty peers and try to cheat each other by sending inaccurate payments and executing workloads incorrectly.

Network model. Light-HIDRA adopts a partial synchronous network model in which the network begins as an asynchronous network (i.e., peers could delay message delivery for any finite amount of time), and later becomes a synchronous network after some special time, which is called the Global Stabilization Time (GST) [29]. Since the workload scheduling protocol is based on BRB, it can operate in asynchronous networks. However, some steps of the Light-HIDRA protocol rely on global timeouts or timestamps in the future for message delivery. For example, Applicants define for each offloading event a timestamp in the future before which the event must be validated to start the workload execution and monitoring. These special times are necessary since Light-HIDRA must avoid offloading events stuck due to unresponsive peers (a workload must eventually be executed). Thus, Applicants are able to retry the request against other peers or domains.

Target workloads. Applicant peers bundle the specifications of their workloads within offloading events, and Solver peers execute the workload according to the received information. Light-HIDRA focuses on the offloading of deterministic workloads that execute services during fixed amounts of time. During execution times, Applicants' domains divide Applicants' payments into different partial payments that are gradually settled according to the health of the service. Deployed services expose a network port to which both the end-users and the Validator peers connect. We limit the type of workloads supported to those workloads accessible via the HTTP protocol that expose services such as web pages or applications, REST APIs or file download repositories. Note that we intend to simplify the proposal and the development of the prototype by focusing on the HTTP protocol because of its simplicity and because it is widely used in Fog-IoT environments. Thus, Validators can analyze the monitoring responses according to the HTTP response status codes, Quality of Service (QoS) metrics such as latency or availability [30], or the requested content by comparing multiple monitoring responses. The latter requires extending the Light-HIDRA proposal by including a result verification mechanism as in [27].

4. Light-HIDRA Protocol

The Light-HIDRA protocol allows Applicant peers to schedule workloads on Solver peers, and has Validator peers that monitor the correct execution of the scheduled workloads. Light-HIDRA includes a decentralized payment mechanism where Applicant peers remunerate Solver

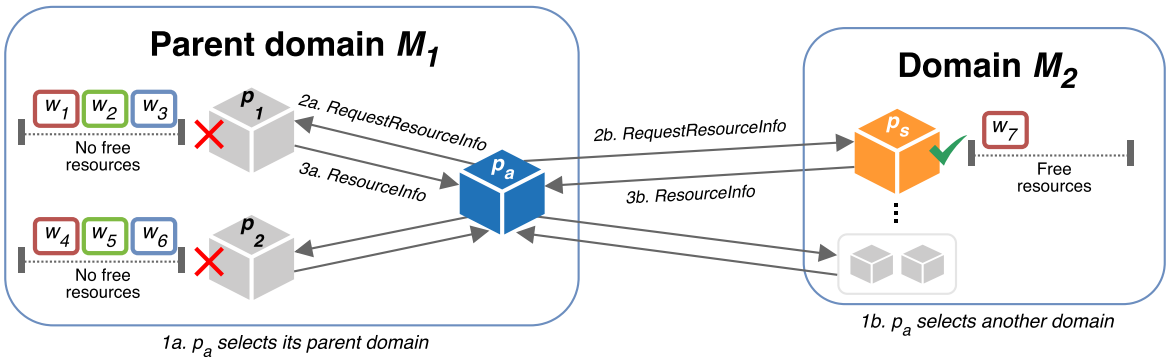


Fig. 4. SSP workflow. Step 1: p_a selects a domain (first tries M_1 and then M_2). Step 2: p_a sends *RequestResourceInfo* messages to the peers in M_1 (step 2a) or M_2 (step 2b). Step 3: peers in the selected domain reply to p_a with *ResourceInfo* messages.

Table 1
Notations used in LIGHT-HIDRA per phase.

Global symbols	
N	Set of all peers in the system
M	Set of domains composed of peer subsets of N
p	Regular peers in a domain
p_a	Applicant peers that send offloading events
p_s	Solver peers that receive offloading events and execute workloads
p_v	Validator peers that monitor workloads and settle partial payments
e	Offloading events
w	Workloads within offloading events
Solver Selection Phase (SSP)	
$r_{\max}$	Maximum amount of resources that peers offer to their parent domains
r_{free}	Current amount of free resources of Solver peers
$tout_{\text{ssp}}$	Timeout for the selection of Solver peers
Workload Reservation Phase (WRP)	
sn_e	Sequence number to identify offloading events per Applicant peer
t_{exec}	Total execution time requested by Applicant peers for their workloads
p_{ratio}	Payment per unit of time offered by Applicant peers to Solver peers for workload execution
ts_{start}	Timestamp in the future that specifies the start time of the monitoring process
d_e	Deposits offered by Applicant peers and locked by domains for the settlement of partial payments
sn_r	Sequence number to identify resource reservations per Solver peer
Workload Execution Phase (WEP)	
n_{ep}	Number of epochs in which the monitoring process is divided to send monitoring requests and settle partial payments
n_{mon}	Number of monitoring requests sent by Validator peers per epoch
f_{thr}	Threshold set by Validator peers to account for failed monitoring requests
ts_{end}	Timestamp in the future that specifies the end time of the monitoring process

peers for the execution of workloads. The protocol is divided into three phases: (1) the Solver Selection Phase (SSP) in which information on available peers and resources is exchanged in order to pre-select a suitable Solver to execute a workload; (2) the Workload Reservation Phase (WRP) that allows peers to disseminate offloading events, lock Applicants' payment offers and reserve resources for workload execution; and (3) the Workload Execution Phase (WEP) responsible for monitoring the workload and ensuring the settlement of the payments. In this section we detail the workflow for each phase. For readability and simplicity reasons, we summarize the LIGHT-HIDRA protocol and describe the notations we use in Fig. 3 and Table 1, respectively.

4.1. Solver selection phase (SSP)

Before sending an offloading event e, Applicant peers query information about other domains and peers to select a suitable Solver peer for the execution of a workload w. This selection is performed locally by Applicant peers and could take into account information about other peers such as the amount of free resources (e.g., CPU or memory), reputation, or geographical location. For presentation clarity and without loss of generality we assume the amount of free resources as the only deciding attribute for Solver peer selection, which is expressed by a single number. The steps involved in the SSP are shown in Fig. 4 and are detailed below:

1. **Domain selection.** An Applicant peer p_a iteratively queries different domains until it has pre-selected a candidate Solver peer that has sufficient resources for the workload. By default, p_a tries to send e and execute w on its parent domain first. If M_1 is not willing to handle e or does not have enough resources (example shown in Fig. 4), p_a can try other domains. Furthermore, in case of canceled offloading events, p_a can also retry the request through other domains. The process is repeated until it finds a Solver peer capable of executing w, or until it has exhausted all domains.
2. **Querying resource usage.** Peer p_a sends *RequestResourceInfo* messages in parallel to an arbitrary number of peers in the selected domain including details about the offloading event to be carried out. This information includes the payment details, requested resources, the workload type and its configuration. Thus, peers receiving *RequestResourceInfo* messages can validate the Applicant's proposal and accept or deny it according to their own criteria.
3. **Solver selection.** Peers reply to the *RequestResourceInfo* by p_a with a *ResourceInfo* message which includes their availability/willingness to execute w (*YES* or *NO*) and resource information such as their $r_{\max}$ and r_{free}. Note that each domain is responsible for managing the resources of its peers, so peers belonging to the same domain are aware of each other's resource information. Therefore, peers also include the resource information of the others in the *ResourceInfo* messages, allowing Applicants to collect and contrast this information from multiple sources. To delimit the selection time, Applicant peers set a local timeout $tout_{\text{ssp}}$ to wait for *ResourceInfo* messages. Once $tout_{\text{ssp}}$ expires, p_a compares the information received and selects a Solver peer p_s prioritizing replies on a first-come first-served basis as long as p_s has enough resources to execute w. At the end of the SSP, p_a has selected a candidate Solver peer, but the payment and resource reservation have not yet been confirmed.

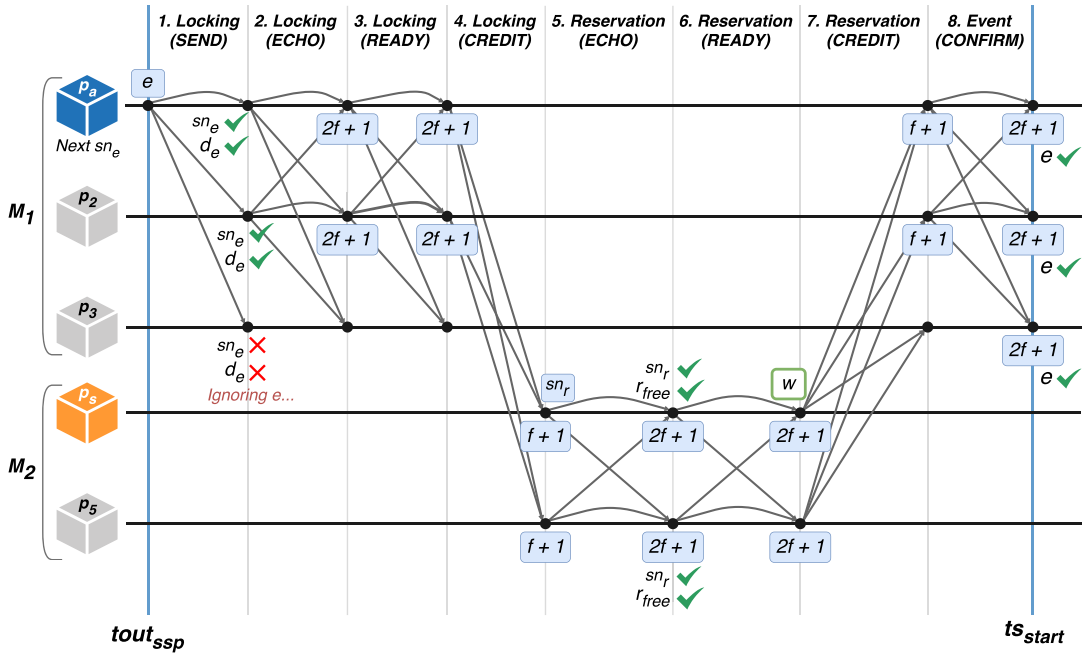


Fig. 5. WRP workflow during which Applicant peer p_a schedules a workload on a Solver peer p_s. Steps show the number of messages required from peers to continue the workflow.

4.2. Workload Reservation Phase (WRP)

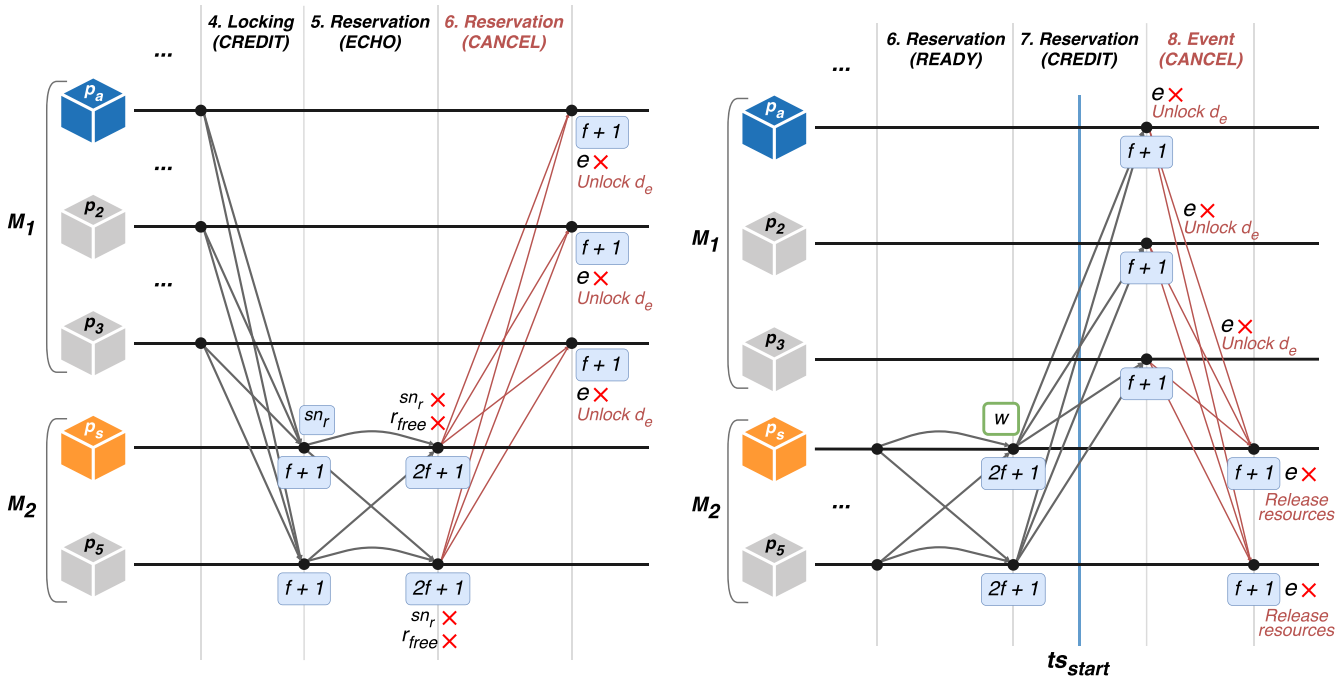
The Workload Reservation Phase (WRP) attempts to schedule a workload to the candidate peer p_s selected in the SSP. The WRP is designed around the BRB protocol that allows peers from a single domain or from multiple domains to register a workload offloading. This mechanism encourages peers to carry out offloading events through partial payments in order to minimize the fraud risk between Applicant and Solver peers. The WRP employs two BRB rounds to ensure offloading event dissemination among peers and domains, credit locking by Applicants' domains and resource reservation by Solvers' domains. Even if workloads are scheduled across domains, credit locking and payments are always handled by Applicants' domains, which ultimately certify the credit transfer to Solvers' domains. Note that the WRP conducts the credit locking to prove that the Applicant is willing to pay for the execution of w, but does not perform the payment itself. Payments are settled in the WEP described in Section 4.3.

Credit locking. Fig. 5 shows the WRP workflow divided by domains and message types, depicting the two BRB rounds, for credit locking and resource reservation, and a final step to confirm the offloading event. In this scenario, Applicant peer p_a in domain M_1 intends to schedule a workload on a Solver peer p_s in another domain M_2. We first detail the BRB round used for credit locking (steps 1-4):

1. Peer p_a sends a *Locking (SEND)* message including the offloading event e to all peers in its parent domain M_1. This message contains the pre-selected candidate p_s, the workload w to be executed and other parameters such as the requested execution time t_{exec}, the payment per unit of time p_{ratio} and a timestamp in the future ts_{start} before which e must be confirmed. Listing 1 shows all fields of an offloading event in JSON format. Offloading events also include a sequence number sn_e to track events/payments per Applicant, so an event can be globally identified by the tuple (p_a, sn_e). Correct peers increment sn_e by one after each event sending.
2. Each correct $p \in M_1$ receives e and checks that sn_e is the next sequence number of p_a. This check prevents Applicants from double-spending their credits by reusing sequence numbers since it does not allow receiving two different events/payments with the same sn_e. Peers also check that p_a has enough credits to cover the entire payment by calculating a deposit d_e as t_{exec} multiplied by p_{ratio}. If so, correct peers temporarily lock d_e credits from the Applicant's balance. If both conditions are satisfied, correct peers that received e broadcast a *Locking (ECHO)* message to M_1. Otherwise, peers ignore e.
3. Correct peers in M_1 wait for at least $2f + 1$ *Locking (ECHO)* valid messages that relate to e (a Byzantine quorum). Once this is achieved, each correct $p \in M_1$ broadcasts a *Locking (READY)* message to M_1 containing e. Alternatively, peers are allowed to send a *Locking (READY)* message to M_1 after receiving $f + 1$ *Locking (READY)* messages from other peers (i.e., while they wait for the $2f + 1$ *Locking (ECHO)* messages). This is known as the READY amplification step.

Listing 1: Content of an offloading event (in JSON format).

```
{
    "applicant": "7d7918afebac...",
    "sn_e": 0,
    "to": {
        "domain": "a6ab43a9113a...",
        "solver": "3bf7ad8b7bf0...",
    },
    "workload": {
        "image": "nginx",
        "resource_limit": 1024,
        "port": 80
    },
    "t_exec": {
        "value": 10,
        "unit": "h"
    },
    "p_ratio": {
        "value": 5,
        "unit": "h"
    },
    "ts_start": 1640995200,
    "signature": "6e367d727a46..."
}
```



(a) *Reservation (CANCEL)* step taken when M_2 determines that sn_r is wrong or p_s is not available to execute w. Peers in M_1 unlock d_e.

(b) *Event (CANCEL)* step taken when M_1 fails to confirm e in time. Peers in M_1 unlock d_e and peers in M_2 release resources.

Fig. 6. WRP workflow cancellation scenarios. Cancellation sent by Solver's domain and Applicant's domain, respectively.

4. Finally, correct peers that receive at least $2f+1$ *Locking (READY)* messages send a *Locking (CREDIT)* message including e to M_2 (the domain the Solver peer p_s belongs to). A *Locking (CREDIT)* message constitutes a quorum certificate that attests to M_2 of the offloading event and the payment offer made by p_a. Now, M_1 have to wait for M_2 to approve the resource reservation for w.

Resource reservation. In the SSP, Solver peers are only pre-selected, so an additional phase is needed to reserve the requested resources. This reservation phase is crucial since Solvers may receive concurrent offloading events trying to reserve the same resources. To overcome this, the reservation phase orders the reservation requests addressed to each Solver peer. Note that the peers in the Solvers' parent domain are responsible for reserving the requested resources. Fig. 5 illustrates the steps of the second BRB round (steps 5-7) for resource reservation, which are explained below:

5. Correct peers in M_2 wait for $f+1$ *Locking (CREDIT)* messages to deliver the offloading event e. Once the Solver peer p_s delivers e, it assigns its next reservation sequence number sn_r to e in order to reserve in M_2 the necessary resources to execute w. Then, p_s sends a *Reservation (ECHO)* message to the peers in M_2 including sn_r. To prevent concurrent events with the same sn_r (for example, initiated by a faulty p_s) from being executed, correct peers that receive sn_r send their *Reservation (ECHO)* messages to M_2 including a vote (*YES* or *NO*) that determines the next offloading event chosen to be executed. Thus, at most only one event will succeed and the others will be canceled as they fail to collect enough votes.
6. Each correct $p \in M_2$ waits for $2f+1$ *Reservation (ECHO)* messages including positive votes to guarantee the Byzantine quorum of the assignment (e, sn_r). Alternatively, peers can also receive negative votes. In case of receiving $f+1$ *Reservation (ECHO)* messages including negative votes, peers in M_2 will cancel the offloading event. Once the Byzantine quorum of the assignment is guaranteed, correct peers in M_2 check that: (1) sn_r is the next reservation sequence number of p_s, and (2) p_s has enough resources to execute w depending on the workload configuration included in e. If so, each correct $p \in M_2$ locally updates the free resources r_{free} of p_s and broadcasts a *Reservation (READY)* message to M_2. Note that this step also rely on the READY amplification step of the BRB protocol.
7. At the end of the reservation BRB round, correct peers that receive $2f+1$ *Reservation (READY)* messages send a *Reservation (CREDIT)* message back to the Applicant's domain M_1 as a quorum certificate that attests the resource reservation made in M_2 for p_s. Moreover, when p_s receives $2f+1$ *Reservation (READY)* messages, i.e., it achieves the Byzantine quorum, p_s can start the execution of w according to the configuration parameters specified in e.

After credits are locked and the resources are reserved, domains M_1 and M_2 have proved both the willingness of p_a to pay for the execution of w and the willingness of p_s to execute w. Fig. 5 shows a last message exchange (step 8) between the peers in M_1 to confirm to each other the reception of valid *Reservation (CREDIT)* messages and confirm that all correct peers agree on the same decision. In this last step, each correct $p \in M_1$ waits for the reception of $f+1$ *Reservation (CREDIT)* messages and then broadcasts an *Event (CONFIRM)* message to M_1. Thereafter, correct peers collecting at least $2f+1$ *Event (CONFIRM)* messages can consider e as confirmed. At this point, faulty or out-of-date peers receiving $2f+1$ *Event (CONFIRM)* messages can update their local state to the actual one. Note that Applicant peers set for each offloading event a timestamp in the future ts_{start} before which the event must be confirmed. This parameter is also key to the proper monitoring of the workload executed by p_s and the accounting of partial payments in the WEP phase described in Section 4.3.

Handling failures. So far we have assumed a WRP workflow in which credit locking and resource reservation rounds are successfully carried out. Since several requirements are checked during the WRP workflow, such as sequence numbers, Applicants' credit balances and Solvers' free resources, it is common that in deployed settings peers

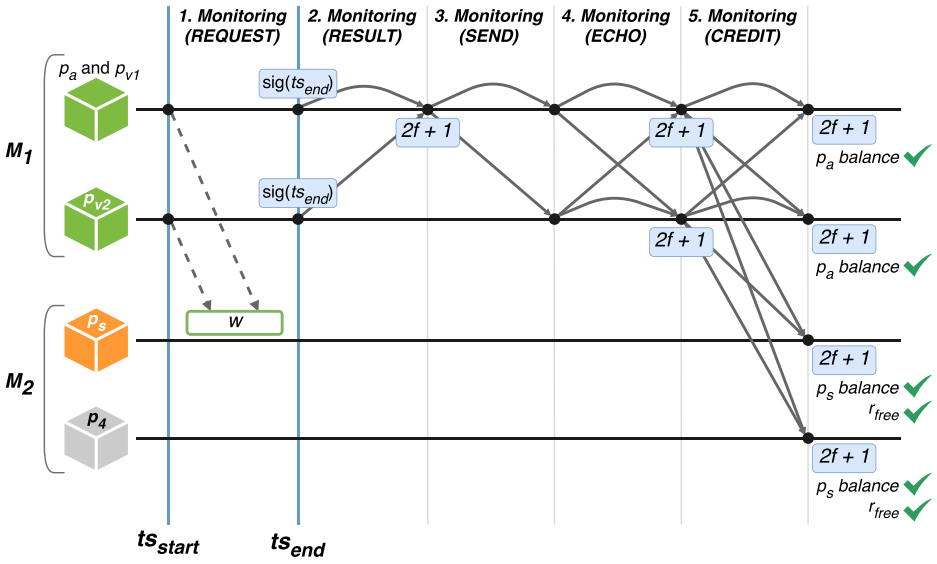


Fig. 7. WEP workflow during which Validator peers in M_1 monitor the workload w executed by p_s, select a shared ts_{end} and settle the Applicant's payment. Finally, peers in M_2 release the resources used by p_s to execute w.

participating in an offloading event do not reach an agreement. Therefore, our protocol needs to handle failures during workload scheduling. Fig. 6 shows additional steps that both M_1 and M_2 perform to cancel an offloading event in order to unlock deposits and release resources to be used in future events.

Fig. 6(a) shows the scenario in which, after M_1 locked the credits offered by p_a, the sn_r chosen by p_s for the resource reservation is wrong or peer p_s is not available to execute w. In this scenario, correct peers in M_2 broadcast a *Reservation (CANCEL)* message to M_1. Then, each correct $p \in M_1$ waits to collect $f + 1$ *Reservation (CANCEL)* messages to unlock d_e from the Applicant's balance. The second cancellation scenario takes place at the end of the WRP workflow.

Fig. 6(b) represents what happens if an offloading event is not confirmed in M_1 before reaching the timestamp ts_{start}, i.e., correct peers do not receive $2f + 1$ *Event (CONFIRM)* messages before ts_{start}. In this case, correct peers in M_1 first unlock d_e from the Applicant's balance and then send *Event (CANCEL)* messages to M_2. Finally, correct peers in M_2 wait for $f + 1$ of these messages to release the resources of p_s previously reserved for w. Furthermore, Solver p_s stops the execution of w.

4.3. Workload execution phase (WEP)

By executing the WRP protocol, peers disseminate offloading events, attest to payment offers and reserve resources for workload execution. The Workload Execution Phase (WEP) is responsible for ensuring the proper execution of workloads and settlement of payment offers. Since there are peers that could fail or act maliciously, we propose a workload monitoring mechanism that assembles Byzantine quorums on the state of executed workloads. The WEP employs a final BRB round to ensure the dissemination of the monitoring results used to determine the final payment sent to the Solver peer. Continuing with the last step of the WRP, we now detail the WEP phase, which is also shown in Fig. 7:

1. Once timestamp ts_{start} specified in e is reached, correct peers in M_1 start the monitoring process of w. Note that ts_{start} prevents Applicant and Solver peers from starting the monitoring process themselves, thus avoiding cheating (i.e., peers that notify only a subset of the peers in M_1 about the WRP completion). Now, each $p \in M_1$ that starts the monitoring process becomes a Validator peer p_v. Since all correct peers in M_1 receive e in the WRP, they already know the parameters t_{exec} and p_{ratio}, so each p_v can determine the number of partial payments and monitoring requests to be sent:

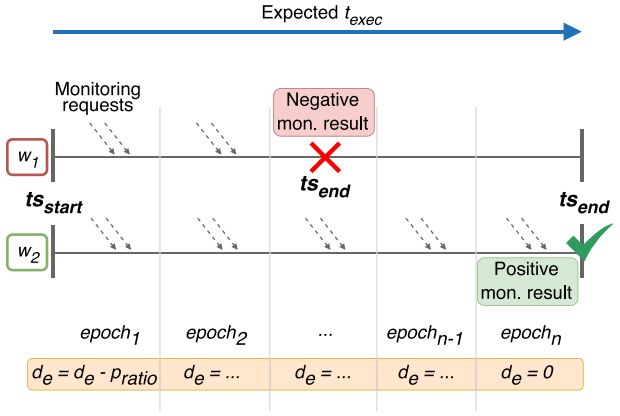


Fig. 8. WEP monitoring timeline representing both a workload with a negative monitoring result (w_1) and a workload with a positive monitoring result (w_2).

   - Partial payments are settled based on predefined time epochs, settling one partial payment per epoch. The number of epochs n_{ep} and their duration depends on the t_{exec} and p_{ratio} parameters. For example, if $t_{exec} = 10$ hours and $p_{ratio} = 5$ credits per hour, each p_v must settle ten partial payments of five credits, i.e., ten epochs settling p_{ratio} credits per epoch. For better understanding, Fig. 8 details the monitoring timeline of two workloads showing different epochs and partial payments.
   - Monitoring requests are sent at random times within epochs in order to prevent p_s from anticipating when w is monitored. Validator peers send n_{mon} *Monitoring (REQUEST)* messages per epoch to w. This parameter can be configured to perform monitoring with varying intensity.

2. Correct Validators count the number of monitoring requests that fail due to no response by the Solver peer, malformed responses or insufficient QoS levels (e.g., high latencies). In any case, if a Validator reaches a threshold f_{thr} of failed monitoring requests, it notifies p_a via a signed *Monitoring (RESULT)* message that includes its personal decision, namely a negative monitoring result. Positive monitoring results are only sent when t_{exec} expires without reaching f_{thr}. Just after sending a *Monitoring (RESULT)*

message, Validator peers stop sending monitoring requests. Note that all correct Validators start the monitoring process at the same time (ts_{start}), but it is also crucial for Validators to agree on a shared end timestamp ts_{end}. Timestamp ts_{end} depends on the monitoring result obtained by each Validator: the sum of t_{exec} to ts_{start} in case of a positive result, or an arbitrary time in case of a negative result. Regardless of the monitoring result, Validators include their own ts_{end} in *Monitoring (RESULT)* messages in order to later select a shared ts_{end} with which peers in M_1 can equally account for elapsed epochs and partial payments.

3. Applicant p_a waits for $2f + 1$ valid *Monitoring (RESULT)* messages. It does not matter what monitoring results p_a receives, i.e., positive, negative or a mixture, it only matters to receive $2f + 1$ valid monitoring results to eventually select a shared ts_{end}. Once received, p_a bundles the monitoring results and sends them via a *Monitoring (SEND)* message to the peers in M_1. This message starts the BRB round required to reach the Byzantine quorum on the monitoring results sent by p_a. Note that without this BRB round, a faulty p_a could send different sets of monitoring results, so peers would eventually select a different shared ts_{end}.
4. Correct peers in M_1 receive the *Monitoring (SEND)* message and check the signed monitoring results. If the *Monitoring (SEND)* message contains $2f + 1$ valid monitoring results from other peers, then correct peers broadcast a *Monitoring (ECHO)* message containing the monitoring results to M_1.
5. Peers receiving $2f + 1$ identical *Monitoring (ECHO)* messages can consider as valid the *Monitoring (SEND)* message sent by p_a. At this point, correct peers in M_1 send a *Monitoring (CREDIT)* message as quorum certificate including the monitoring results to M_1 and M_2. The partial payments settled from the deposit d_e depend on the number of epochs that have elapsed from ts_{start} to ts_{end}. To obtain ts_{end} consistently, correct peers receiving $2f+1$ *Monitoring (CREDIT)* messages select the $(f+1)$th smallest timestamp from the monitoring results. Finally, correct peers in M_1 and M_2 determine the number of partial payments to be settled and update the balances of p_a and p_s. In addition, correct peers in M_2 release the resources of p_s used by w.

5. Security analysis

In this section we evaluate the security of Light-HIDRA by discussing potential shortcomings and security properties required to carry out offloading events in the system. We also analyze the three phases of the protocol by identifying potential attack vectors concerning the different roles involved in the system during the execution of an offloading event. For each potential risk identified, we describe the security measures that mitigate it.

We first focus on identifying potential shortcomings of the Light-HIDRA proposal. In decentralized environments, addressing malicious actions that could degrade system trust, reliability and performance is of critical importance. Sybil attacks [31,32] pose a significant threat to the security of decentralized systems. In a Sybil attack, a single adversary creates multiple identities to attempt to compromise trust and gain undue influence on the system. Light-HIDRA mitigates such attacks by establishing an initial domain and peer bootstrapping process managed by an administrative entity. The management of identities free from administrative entities falls outside the scope of this paper. Consequently, we highlight an additional shortcoming related to the administrative overhead during this initial process. As stated in Section 3, we assume an administrative entity responsible for composing domains and publishing peer lists, a process that will entail extra steps and time for the end-user. We also identify other attack vectors in decentralized systems such as Denial of Service (DoS) attacks [33]. Although all messages received by peers during the different phases of Light-HIDRA are validated before being computed, it could lead to peer saturation if a very large number of repeated or malformed messages are received. External systems such as firewalls and network traffic monitors could prevent these attacks [34,35].

We continue to emphasize the BRB security properties and how the Light-HIDRA protocol leverages them. First, the *validity* property verifies that a message (an offloading event e, a reservation request or a monitoring result) sent by a correct peer is eventually delivered to every correct peer. The BRB primitive guarantees the *validity* property in Light-HIDRA since, assuming a correct sender, the correct peers in the system (or in a domain) that receive the message broadcast it again to the other peers via ECHO messages. Consequently, each correct peer receives at least $N - f$ ECHO messages, and $N - f \geq 2f + 1$, so correct peers are able to deliver the sender's message. Regarding the *no duplication* property, peers in Light-HIDRA implement a message tracking mechanism that uses sequence numbers, sender identities and message type identifiers. This mechanism allows correct peers to detect and discard duplicate messages. Light-HIDRA achieves the *integrity* and authenticity of exchanged messages by implementing authenticated links via digital signatures. This ensures that if a correct peer receives a message from a (supposedly) correct sender, the message has indeed been sent by that sender and has not been tampered. The next security property is *consistency*. This property is achieved by ECHO messages and Byzantine quorums, i.e., correct peers have to collect at least $2f+1$ ECHOs of the same message in order to validate and deliver it. Thus, all correct peers achieve a consistent view of the sender's message.

In addition to the aforementioned security properties, it is necessary to guarantee *totality* in order to achieve reliable message delivery in Light-HIDRA. This property ensures final agreement in such a way that correct peers only deliver a message if all other correct peers deliver that message. In Light-HIDRA, achieving the *totality* property is essential since all correct peers in the system or in a domain need to agree on whether an offloading event has succeeded based on if (1) the Applicant's credits have been locked, (2) the resources have been reserved in the Solver, or (3) the execution of the workload has been successful. To accomplish this, Light-HIDRA incorporates a final broadcast round of READY messages. Again, correct peers collecting at least a Byzantine quorum of $2f+1$ READY messages are able to deliver the sender's message.

5.1. SSP risks and measures

Applicant peers conduct the Solver Selection Phase to pre-select the domains and Solver peers which execute their workloads. To achieve this, Applicant peers iteratively query different domains and peers in the system until they receive a response. Below, we detail the potential risks in this phase and the security measures employed in Light-HIDRA to mitigate them:

- A potential risk is that the Applicant peer receives *ResourceInfo* messages with fake resource information from faulty peers. Note that the Applicant is responsible for collecting the same information from at least $f + 1$ peers belonging to the system or domain. If the Applicant does not receive the same information from $f + 1$ peers, it discards its current selection and searches for other candidate peers or domains.
- Another risk to consider is the selection of a faulty peer as the Solver peer. In the SSP this is not a problem because the Applicant peer can choose another peer or send a new *RequestResourceInfo* message to a different domain after canceling the offloading event in the WRP or obtaining a premature negative monitoring result in the WEP.
- An Applicant peer could provide details of the offloading event during this phase that differ from what it will actually send within e in the WRP. In this case, peers asked for pre-selection will reply incorrectly based on this information. However, both credit locking and resource reservation will be carried out using

the information sent within e in the first step of the WRP. If the Applicant ultimately does not have enough credits or the Solver does not have available resources, event e will be canceled. Additionally, potential Solvers can also compare the event details received in the SSP with those received in the second BRB round of the WRP. Thus, Solver peers are able to cancel requests from this kind of faulty Applicants.

- Limited availability of peers participating in the SSP could hinder the pre-selection phase. Note that the responsibility for pre-selection and sending the offloading event e lies with the Applicant peer. Therefore, during the SSP, the Applicant is the only peer that must remain available. If peers being queried do not reply, the Applicant will repeat the process with other peers and domains.

5.2. WRP risks and measures

Once a Solver peer has been pre-selected, the Applicant broadcasts the offloading event e to the peers in its parent domain and also to the Solver domain in case of an inter-domain request. The Workload Reservation Phase is the most critical phase of the Light-HIDRA protocol, since it must ensure a secure and fair credit locking and resource reservation among participating peers, avoiding any fraudulent action for individual profit. We now identify the potential attack vectors and discuss the security measures for this second phase of Light-HIDRA:

- One of the most significant risks to consider is the possibility of double spending by Applicant peers. Light-HIDRA is a trustless P2P system in which peers participate and collaborate according to the protocol rules. To send e and its related payment to a Solver peer, an Applicant must first be approved by the system or parent domain to which it belongs. This procedure is managed by the first BRB round of the WRP, which locks the payment until the resource reservation and execution of w occur. Therefore, Applicant peers are required to prove their balances and willingness to pay to all participating peers in an offloading event. Note that payments are managed and settled in a decentralized way by the Applicant's peers, making it not possible to reject a payment once sent (unless otherwise determined by the protocol) or to use the same credits for multiple payments (since credits are locked).
- Once the Applicant's payment is confirmed, it is also crucial to ensure that the Solver peer follows the protocol and makes a fair resource reservation. The resources in the system or domain are collectively managed by the peers within them, which prevents a Solver from conducting unfair actions on its own. To minimize the risk in the case of faulty Solvers, Applicant's payments are divided and settled in partial payments over time. Thus, Applicants can stop paying for the execution of workloads and recover part of the locked credits.
- Another potential risk is the sending of duplicated or simultaneous offloading events that compromise the consistency of the system, i.e., that produce different states in each peer (e.g., if two events arrive simultaneously in a different order at different peers). To avoid this, Light-HIDRA orders offloading events and resource reservations using sequence numbers. This ensures that peers receiving offloading events with a sn_e, and reservations including a sn_r, can execute them in an ordered way to preserve consistency regarding balances, locked credits and available resources.
- Peers in Light-HIDRA must mutually confirm the credit locking and resource reservation for a workload to be eventually executed. If one of the two BRB rounds of the WRP is not successfully completed, both the locked credits and reserved resources must be released for use in future offloading events. The WRP supports several additional cancellation messages (see Section 4.2 for more details) to allow participating domains and peers to report unsuccessful offloading events and release both credits and resources. Note that the timestamp ts_{start} is also crucial when canceling an offloading event, since it determines the maximum time the event has to complete the locking and reservation phases before starting the WEP and workload monitoring.
- Finally, we discuss whether the availability of participating peers in the WRP could hinder the locking and reservation phases. Once the Applicant peer sends e to its peers using a *Locking (SEND)* message, it is no longer required for the Applicant to remain available during the WRP. The credit locking will be carried out by other available peers. However, the selected Solver peer must be available to execute w before starting the WEP if it wants to succeed without obtaining a premature negative monitoring result.

5.3. WEP risks and measures

The Workload Execution Phase is responsible for monitoring the workload executed by the Solver peer in the WRP and, ultimately, settling the payment in both the Applicant and Solver domains. The final payment depends on the monitoring results obtained by Validator peers belonging to the Applicant's domain. The risks identified in the WEP and the measures implemented by Light-HIDRA to mitigate them are as follows:

- Once the WRP is completed, the Solver peer may not execute w or may not execute it on time. Applicant peers add a timestamp ts_{start} to each offloading event sent that determines the maximum time to perform the WRP and the WEP start time. If the Solver does not execute w or executes it out of time, the Validators will get a premature negative monitoring result that will terminate e at no cost or minimal cost for the Applicant. Therefore, Solver peers are responsible for the correct execution of w. Note also that the Solver domain could deny an offloading event in the WRP in case it receives a ts_{start} too small. This will depend on the configuration and requirements of the Solver domain and its peers.
- Applicant peers are responsible for collecting and broadcasting the monitoring results that include the ts_{end} timestamps. This is because if each Validator broadcasts its own ts_{end}, the other peers might receive different sets of monitoring results, resulting in the selection of a different shared ts_{end}. Therefore, potential risks could be that a faulty Applicant does not send the *Monitoring (SEND)* messages or sends different *Monitoring (SEND)* messages to each peer. If an Applicant does not broadcast the monitoring results, the Validator peers will settle the entire payment in favor of the Solver peer. Besides, sending different sets of monitoring results to each peer is not feasible from the point of view of a faulty Applicant since the WEP employs a third BRB round to safely broadcast the monitoring results.
- Applicants and Solvers perform special tasks in the WEP, i.e., the broadcasting of monitoring results and timestamps ts_{end} and the execution of workloads respectively, so it is crucial at this phase that both Applicant and Solver maintain a high level of availability. Note that it will be the responsibility of Applicants and Solvers to remain available to maximize their profits, since it is not feasible to harm other participating peers in case of low availability (Applicants and Solvers are the only ones interested in completing an offloading event).

6. Experimental evaluation

This section presents the implementation of Light-HIDRA and assesses its feasibility and performance, compared to the state-of-the-art in this field. We compare Light-HIDRA with the HIDRA scheduling system, which is our previous proposal for resource scheduling in Fog-IoT environments [6]. In particular, our experiments answer the following questions:

1. How does the bandwidth usage of Light-HIDRA and HIDRA scale with the network size (Section 6.2)?
2. What is the end-to-end latency of an offloading event in Light-HIDRA and HIDRA, and how does this latency change when increasing the number of events (Section 6.3)?
3. How does the CPU and memory usage of Light-HIDRA and HIDRA scale with the network size (Section 6.4)?

Considering the testbed chosen to carry out the experiments (detailed in Section 6.1), our main findings are as follows:

1. Light-HIDRA **reduces bandwidth usage by up to** 57× compared to HIDRA under the same workload.
2. Light-HIDRA **reduces the latency of offloading events by up to** 142× compared to HIDRA under the same workload.
3. Light-HIDRA demonstrates **superior throughput performance** compared to HIDRA, exhibiting a notably lower average event fulfill latency across all evaluated system loads.
4. Light-HIDRA **reduces CPU usage by up to** 10× **and memory usage by up to** 9.6× compared to HIDRA under the same workload.

Baseline. HIDRA schedules workloads in a distributed way leveraging blockchain technology and smart contracts. Each peer belonging to a HIDRA network executes a management client and a blockchain client to record and share information via blockchain transactions, thus allowing all nodes to know the current state of scheduling flows. Blockchain transactions allow HIDRA nodes to perform tasks such as node registration, sending offloading events and selecting Solver nodes that execute the scheduled workloads. We remark that in HIDRA the Solver selection process is carried out by voting of the nodes participating in the network. HIDRA involves four steps of information exchange via blockchain transactions. When an Applicant node sends an offloading event to the network, the other nodes reply by sending information needed for the selection of a Solver (e.g., current resource usages, node location, etc.). Then, HIDRA waits for the response from a minimum number of nodes, a parameter configurable in the network setup. Using the collected information, nodes deterministically select a Solver and publish their vote on the blockchain. The smart contract then waits for a minimum number of votes (also configurable in the setup) to notify the Solver node that it can execute the requested workload. Finally, the Solver node solves the offloading event once it finishes executing the workload. It should be noted that the security and performance of HIDRA will depend on the consensus mechanism used by the underlying blockchain.

Implementation. Light-HIDRA is developed in the Python 3 programming language and is based on the IPv8 peer-to-peer networking library.[1] This lightweight library enables authenticated data exchange among peers belonging to the same overlay network. Moreover, HIDRA is implemented in the Golang programming language and utilizes the Ethereum platform, the Geth client, and Solidity smart contracts for the processing of offloading events. Both proposals employ libraries for asynchronous event processing and multiprocessing. All software artifacts of both Light-HIDRA[2] and HIDRA[3] are available online.

6.1. Experimental setup

The main objective of our experiments is to compare both Light-HIDRA and HIDRA on different aspects. Our aim is to establish the scalability and performance benefits of Light-HIDRA as a decentralized and lightweight approach that sidesteps the need for a global consensus algorithm to maintain a shared state of payments and resource reservations in the system. To this end, our experiments measure key, non-functional attributes such as the bandwidth, latency and resource usage (CPU and memory usage) of offloading events sent over Light-HIDRA and HIDRA networks of differing sizes. An innovative part of Light-HIDRA is the partitioning of peers into distinct domains. To demonstrate the benefits of domains, our experiments consider two possible configurations of Light-HIDRA: (1) a configuration in which all peers in the network belong to the overall system, so there are no domains (or there is only one "virtual" domain) and offloading events are sent globally (intra-domain), and (2) a configuration that divides the network peers into different domains so offloading events are sent across domains (inter-domain).

Testbed. To carry out the experiments, different Light-HIDRA and HIDRA networks, composed of up to 400 peers, have been deployed on an Ubuntu 22.04.2 server with 64 Intel Xeon Gold 5218 CPUs and 504 GB of memory. The HIDRA networks deployed for the experiments are permissioned networks using a lightweight consensus algorithm such as Proof-of-Authority (PoA) and a zero block time (i.e., instant transactions). We configure the HIDRA networks to require the same number of correct peers as in Light-HIDRA, i.e., at least $2f + 1$ peers sending replies and votes. Note that Light-HIDRA experiments consider the worst case scenario regarding the f parameter (i.e., the largest possible f value given the number of peers in each experiment). During the execution of an offloading event, all involved peers send all protocol messages (regardless of whether they are correct or faulty messages) to all other peers in the network. Since we assume networks with at most f faulty peers, correct peers only make use of the first $2f + 1$ correct messages they receive. In order to ensure a fair comparison, the experiments also take into account several assumptions when executing offloading events and measuring the results. On the one hand, we omit network latencies between peers to avoid interference with the obtained results. Thus, the results solely relate to the execution of the Light-HIDRA and HIDRA scheduling protocols. On the other hand, the experiments do not include statistics related to network bootstrapping and peer discovery, and we assume peers know each other in advance when the experiment starts. For example, the results obtained for HIDRA do not include the overhead of initial on-chain transactions such as the smart contract deployment and the registration of each peer in the system.

Experiment workloads. We prepare synthetic workloads that can be handled by HIDRA and Light-HIDRA. Synthetic workloads are similar to real workloads and can be repeatedly applied to both systems, allowing us to reproduce specific scenarios for carrying out the experiments. To conduct a fair comparison, in each experiment both proposals execute synthetic workloads containing the same tasks. Such tasks are actually offloading events like the one shown in Listing 1. Offloading events are triggered by one or more randomly chosen Applicant peers at the beginning of each experiment. In the case of HIDRA, an offloading event will be disseminated across all nodes in the blockchain network, performing the transaction exchange needed to select a Solver peer and finally execute a service. In contrast, an offloading event in Light-HIDRA will go through the different phases of the protocol described in this paper, i.e., the pre-selection of the Solver, the credit locking, the resource reservation and the service monitoring process. Note that all messages or transactions produced during the execution of an offloading event in both systems are truly sent, received, validated and computed. In the experiments, we only simulate the execution of the services that would run on top of HIDRA and Light-HIDRA.

6.2. Bandwidth usage

We first evaluate the bandwidth generated when sending an offloading event in Light-HIDRA and HIDRA networks of different sizes.

Setup. We employ the tcpdump tool to measure the bandwidth usage from the start to the completion of one offloading event in both

[1] See https://github.com/Tribler/py-ipv8.
[2] See https://github.com/swarleynunez/HIDRA_IPv8
[3] See https://github.com/swarleynunez/HIDRA

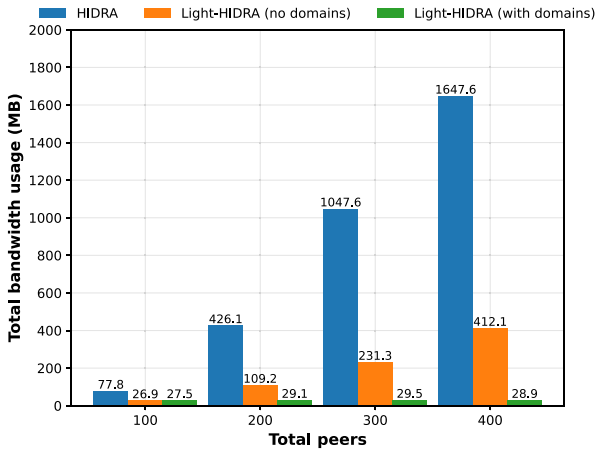


Fig. 9. Total bandwidth usage in the scheduling of an offloading event sent both on-chain (via HIDRA) and off-chain (via Light-HIDRA).

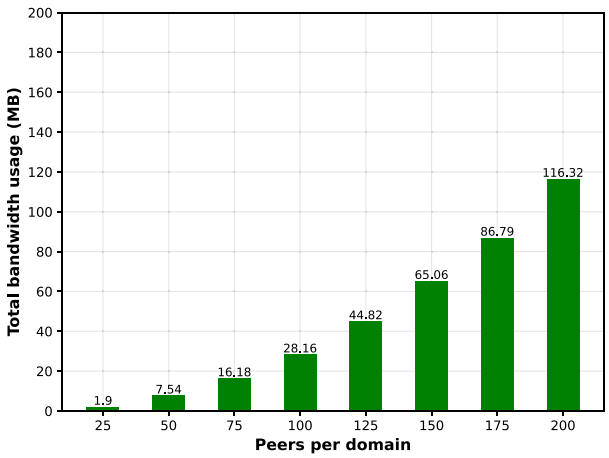


Fig. 10. Total bandwidth usage of an offloading event in Light-HIDRA using different domain sizes.

proposals. For each network type and size, a random peer belonging to that network sends the same event including the same workload.

Results. Fig. 9 displays the total bandwidth usage of HIDRA and Light-HIDRA, for different network sizes. We compare three settings: HIDRA, Light-HIDRA without domains and Light-HIDRA with domains. In the latter configuration we fix the domain size to 100 peers. Figure reveals that bandwidth usage increases as the network size grows. The graph also shows high bandwidth usage of HIDRA compared to Light-HIDRA. Even without dividing the network into domains, Light-HIDRA uses significantly less bandwidth than the baseline system. For example, in a network comprising 400 peers, HIDRA consumes a total of 1647.6 MB, while Light-HIDRA consumes at most 412.1 MB and 28.9 MB when using domains: a reduction of 75.0% and 98.2%, respectively. We also observe differences in bandwidth usage between the two configurations of Light-HIDRA: in the case of Light-HIDRA without domains, larger networks result in higher bandwidth usages, whereas in Light-HIDRA with domains the bandwidth remains constant. Note that the first measurement yields similar bandwidth results for both Light-HIDRA configurations, since the total number of peers is 100 and we fix the domain size to 100 peers for this experiment, so there can only be one domain.

Fig. 10 showcases the effect of using domains of different sizes on bandwidth usage, detailing the evolution of bandwidth usage with respect to the number of peers per domain. In this experiment, we deploy two domains for each domain size. Thus, for a domain size of 200 peers, we actually deploy a Light-HIDRA network with 400 peers. Comparing the obtained results with those from Fig. 9, the use of two domains with 200 peers in Light-HIDRA (400 peers in total) incurs 3.66 times less bandwidth than an HIDRA network with 200 peers, and 14.17 times less bandwidth than an analog HIDRA network of 400 peers. Finally, we note that Light-HIDRA networks with domains add a slight bandwidth overhead compared to Light-HIDRA networks without domains. This is because in inter-domain offloading events, the Applicant and Solver domains have to exchange additional data than in the case of intra-domain events. For example, peers in inter-domain events are required to send the *Monitoring (CREDIT)* messages to both peers in their parent domain and to peers in the Solver's domain (more details in Section 4.3), which duplicates the bandwidth for this type of message. In contrast, peers in intra-domain events only exchange *Monitoring (CREDIT)* messages with each other within their parent domain.

Conclusions. We demonstrated that Light-HIDRA incurs a significantly lower bandwidth consumption compared to HIDRA. The use of domains also reduces bandwidth usage. By dividing peers into domains, Light-HIDRA further reduces the overhead of offloading events for the same number of participating peers. Note that we assume networks with f faulty peers and at least $3f+1$ peers, so the larger a Light-HIDRA network is, the more peers will need to participate in an offloading event. Moreover, since the experiments consider the worst case scenario regarding the f parameter, i.e., all the f peers send messages, any value $f' \leq f$ will not lead to additional overhead in the results. For example, in the 100-peer Light-HIDRA network without domains in Fig. 9, both with $f' = f$ and with $f' = 0$ the bandwidth usage is about 26.9 MB, whereas if we assume a scenario with f silent peers that selfishly avoid participating in the protocol, the bandwidth usage is 19.14 MB (keeping the network size at 100 peers in all cases).

6.3. Latency and throughput of offloading events

We next quantify the sustainable throughput and end-to-end latencies of offloading events in HIDRA and Light-HIDRA. This latency is the time between the creation of an offloading event by an application peer and the moment the workload gets assigned to a Solver peer.

Setup. To conduct a fair comparison between both proposals, we only measure the time required by the scheduling protocols on their own to complete offloading events. We do not include in the results additional times such as the blockchain block time. This is because the block time will depend on the blockchain network on which HIDRA is executed and will always increase the fulfill latency of an offloading event (since we use instant transactions in the experiments). We also do not include in the results the configurable timeouts of Light-HIDRA (i.e., $tout_{\text{ssp}}$ and ts_{start}).

Results. Fig. 11 shows the event fulfill latencies of an offloading event in HIDRA and Light-HIDRA networks ranging from 100 to 400 peers, with domains composed of 100 peers. Fig. 11 shows high latency for HIDRA: it takes over four minutes to fulfill an offloading event in a network of 400 peers. We observe that the latency increases between the different HIDRA networks are significantly larger (superlinear) than the increases occurring in Light-HIDRA networks. In contrast, the event fulfill latencies in Light-HIDRA, both with and without domains, are significantly lower. In a network of 400 peers, Fig. 11 demonstrates a reduction 19.4× and 142× for Light-HIDRA without and with domains, respectively. Note that in Light-HIDRA, the steps that contribute the most to increase latencies are those with $O(N^2)$ complexity, i.e., steps that involve an all-to-all communication and have to guarantee the Byzantine quorum before the execution flow can continue (ECHO and READY messages in the different BRB rounds).

Our next experiments explore the throughput of HIDRA and Light-HIDRA with domains. We measure the throughput of offloading events for both systems in networks with 100 peers and domains composed of

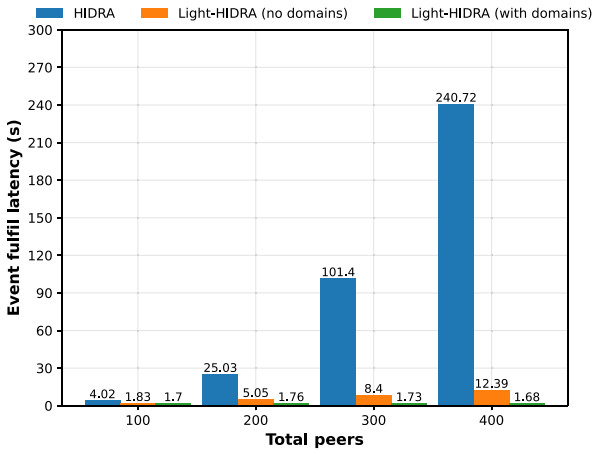


Fig. 11. Fulfill latency of an offloading event in HIDRA and Light-HIDRA networks of different sizes.

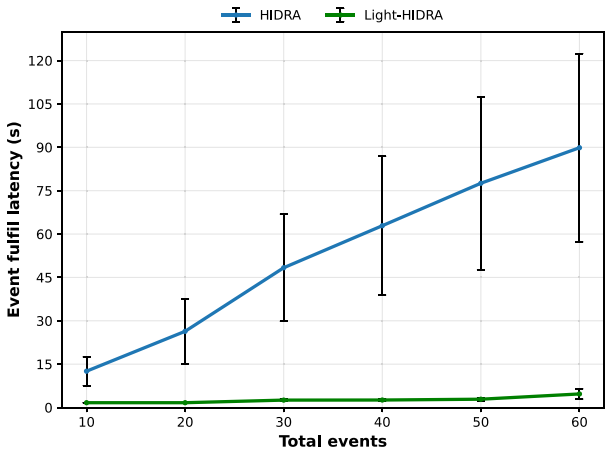


Fig. 12. Fulfill latency when sending multiple offloading events.

50 peers. We expose both systems to an increasing number of offloading events, and measure the increase in event fulfill latency. This is a common approach to measure the throughput of distributed algorithms [14]. Fig. 12 shows an increase of the fulfill latencies per offloading event as the number of simultaneous events sent increases for both systems. We also show the standard deviation in latency with error bars. Our first observation is that Light-HIDRA shows significantly lower average event fulfill latencies for all evaluated loads. For example, when sending 60 offloading events, HIDRA shows an average fulfill latency of 89.86 s, while Light-HIDRA achieves an average of 4.7 s per event, 19.1 × lower.

As the system load increases, Fig. 12 shows a notable increase of event fulfill latencies in HIDRA whereas Light-HIDRA is mostly indifferent to this increase. We do notice a small increase latency for Light-HIDRA when sending more than 50 simultaneous offloading events. Specifically, when increasing the load from 50 to 60 events, the average event fulfill latency of Light-HIDRA increases by 1.63 times. We also noticed that in some executions, HIDRA is unable to handle more than 60 simultaneous events and is unable to fulfill some of these events in these conditions.

Conclusions. Light-HIDRA shows significant reductions in the fulfill latency of offloading events compared to HIDRA, namely up to 142×. Light-HIDRA also demonstrates superior throughput performance compared to HIDRA, exhibiting a notably lower average event fulfill latency across all evaluated system loads. To explore the influence of the f parameter as is done with bandwidth usage, we evaluate the fulfill latency of an offloading event by assuming a scenario with f silent peers that selfishly avoid participating in the protocol. We observe similar fulfill latencies in experiments with and without silent peers. For example, in a 100-peer Light-HIDRA network without domains, we obtain a fulfill latency of 1.51 s for the scenario with f silent peers, compared to the 1.83 s shown in Fig. 11.

6.4. CPU and memory usage

Our final set of experiments measure the CPU and memory overhead of HIDRA and Light-HIDRA.

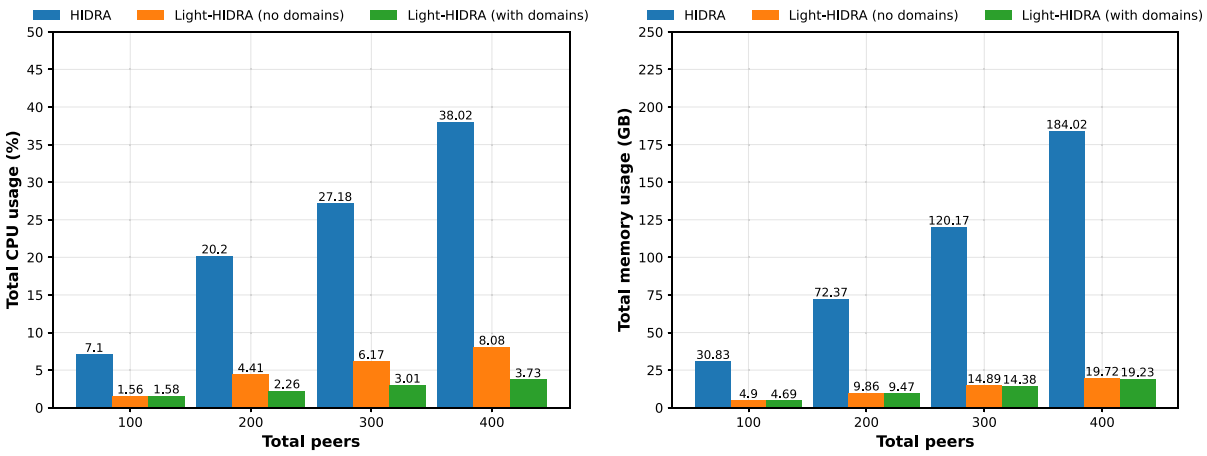
Setup. We measure the resource usage of the HIDRA and Light-HIDRA networks deployed for the experiments in Figs. 9 and 11. This helps us to assess the feasibility of running Light-HIDRA on lightweight, resource-constrained devices. To this end, we add some additional features to the management clients of HIDRA and Light-HIDRA in order to monitor the resource usage of each peer deployed in the experiments. Specifically, we employ the psutil library in both proposals to monitor peers during the lifecycle of an offloading event. Note that the overhead produced by the monitoring features is excluded from the obtained results.

Results. Fig. 13 highlights the CPU and memory usages of HIDRA and Light-HIDRA networks with varying sizes, when sending a single offloading event. Figure shows the overall CPU and memory usage of all peers belonging to the network deployed in each experiment. The values are displayed in relation to the total amount of resources of our testbed. The obtained results for CPU usages (Fig. 13(a)) and memory usages (Fig. 13(b)) demonstrate a high resource consumption by HIDRA in comparison with Light-HIDRA. Taking as example the measurement of a 400-peer network, HIDRA has a CPU consumption of 38.02% compared to 8.08% for Light-HIDRA without domains, which represents a 4.71× reduction in CPU usage. Another observation is the difference in CPU usage between the two Light-HIDRA configurations. Fig. 13(a) shows approximately double of CPU usage in Light-HIDRA networks without domains, except for the measurement with 100 peers, since there is only one domain. This is because all peers in networks without domains fully execute the three phases of the Light-HIDRA protocol, whereas in networks with domains the tasks are distributed among the Applicant and Solver domains.

Fig. 13(b) shows that Light-HIDRA significant reduces memory usage compared to HIDRA. In a 400-peer network, Light-HIDRA uses up to 9.6× less memory than HIDRA. We also observe that memory usage for the two configuration of Light-HIDRA is mostly the same, and domains do not impact memory usage.

It should be noted that the high CPU and memory usages of HIDRA are partially attributed to the use of its two software clients, i.e., the management client and the blockchain client (Geth), in comparison with Light-HIDRA, which only requires to execute the management client since it is an off-chain solution. We decided to add the resource usage of the underlying blockchain to the results as it is an essential component in HIDRA.

Conclusions. Light-HIDRA consumes significantly less CPU and memory resources compared to HIDRA, and therefore is more suitable for deployment on lightweight devices. Regarding the f parameter, any value $f' \leq f$ will not lead to additional overhead in the results as the experiments consider the worst case scenario. We also evaluate CPU and memory usages of an offloading event in a scenario with f silent peers that avoid participating in the protocol. In the case of the 100-peer Light-HIDRA network without domains in Figs. 13(a) and 13(b), we obtain a CPU usage of 1.4% and a memory usage of 4.88 GB for the scenario with f silent peers, results slightly lower to those obtained in the previous experiments (i.e., 1.56% and 4.9 GB, respectively).



(a) CPU usage.

(b) Memory usage.

Fig. 13. Total resource usage employed by HIDRA and Light-HIDRA networks with different number of peers for sending an offloading event.

7. Related work

In this section we emphasize prominent related work in the field of distributed resource orchestration and workload offloading in order to highlight the novelty of Light-HIDRA compared to other works in the literature. We summarize the most relevant and comparable works and their main features in Table 2.

Workload scheduling in Fog-IoT environments is an integral part of resource orchestration. According to the survey presented in [36], the vast majority of proposals for orchestration in fog environments adhere to a centralized architecture. This survey also identifies several goals addressed by orchestrators, including Service Level Agreement (SLA) guarantees, service cycle management, and more, with resource management being the most frequently researched goal.

Given that applications and services are often encapsulated as containers for easier deployment and management, a growing trend is to leverage Kubernetes-like container orchestrators for fog/edge resources. To this end, specialized versions of Kubernetes tailored to fit resource-scarce distributed nodes have been developed, such as K3s [37] or KubeEdge [38]. However, note that in all these cases, resource orchestration is still performed by a centralized server. Furthermore, the scalability of the Kubernetes' ecosystem is approached through the federation of different Kubernetes clusters, exemplified by Karmada [39], which again relies on a centralized control plane governing multiple Kubernetes clusters.

However, a centralized approach poses challenges such as potential single points of failure, increased latency and scalability limitations. To address these issues, researchers are exploring alternate solutions based on distributed resource orchestration. Due to the intrinsic decentralized nature of fog computing, its confluence with blockchain and other distributed technologies seems natural, not only in the application domain (in order to track and secure data from Fog-IoT environments), but also to achieve a truly distributed resource management. The EdgeChain framework [40] proposes a credit-based resource management system based on a permissioned blockchain and a currency system. The behavior of the IoT devices and rule enforcement is regulated by means of smart contracts. Another prominent work in the field of distributed resource management is MODiCuM [27]. The authors present an open resource outsourcing market based on blockchain technology and smart contracts in which resource owners and end users exchange computational resources and credits. To avoid misbehavior by system participants, MODiCuM models the exchange of resources as a game-theoretic model with the objective of assigning them fair rewards and penalties.

The authors of Light-HIDRA also proposed HIDRA, a fully distributed orchestrator designed for managing fog resources [6]. The primary focus of HIDRA was to effectively orchestrate workloads as containers within local fog clusters, harnessing the combined power of blockchain networks and lightweight virtualization technologies. In HIDRA, the workload scheduling tasks rely on a set of smart contracts. Consequently, the control plane of HIDRA is deliberately decentralized, avoiding the risks associated with single points of failure. By distributing control across all nodes in the cluster, the system's security is significantly enhanced, effectively protecting against potential DoS attacks and physical threats. More recently, S-HIDRA [10] has been developed to address Fog-IoT environments that are geographically broader and organized in domains composed of fog nodes/end devices. To further empower the decentralized orchestration of containerized services, S-HIDRA integrates Software Defined Networking (SDN) capabilities. This strategic fusion of SDN technology and the HIDRA orchestrator, along with S-HIDRA domains, enables dynamic and programmable management of network traffic, facilitating efficient and scalable operations across the fog computing landscape.

Regarding the most specific topic of workload offloading, the recent survey in [41] focuses on strategies for node selection in order to optimize QoS and reduce energy consumption. The proposals address the computing offloading among cloud, fog, and edge layers, not among peers as in the case of Light-HIDRA. The optimization-based techniques are the most preferred choices to improve the QoS and reduce energy. Most of those techniques can complement Light-HIDRA by providing more sophisticated criteria for solver selection (auto-selection in our case, since there is not a centralized controller). This survey also includes a few studies that harness the blockchain technology in different ways, such as using smart contracts at the application level for interaction among stakeholders [42], or securing the offloading process [43–46].

More precisely, Shi et al. propose a blockchain-enabled Vehicular Edge Computing (VEC) framework [43] aimed at boosting the reliability and efficiency of vehicle-to-vehicle task offloading. They also employ a Deep Reinforcement Learning (DRL)-based computation offloading scheme. This enables task vehicles to delegate a portion of their computation-intensive tasks to neighboring vehicles. To enhance security, the paper introduces an improved consensus algorithm based on Practical Byzantine Fault Tolerance (PBFT), along with an algorithm for selecting consensus nodes. The proposed scheme for VEC is validated through simulation, considering a fixed area with 50 base stations and up to 30 consensus nodes.

Sellami et al. combine blockchain and DRL to facilitate energy-aware task scheduling and offloading within an Internet of Things (IoT) network enabled by SDN [44]. They implement a centralized control plane, which includes a task scheduler within the Ryu SDN controller (i.e., offloading decisions are made off-chain in a centralized

Table 2
Key features of comparable approaches. Decentralized management (D.M.), scalability-focused (S.F.), workload validation (W.V.), underlying consensus/agreement mechanism (U.M.).

Ref.	Context	Main goal	D.M.	S.F.	W.V.	U.M.	Proposal validation	Evaluation etrics
[40]	Fog-IoT environments	Managing resources using smart contracts deployed on permissioned blockchains	✓	✗	✗	PoW	Experimental testbed	CPU, memory and disk usages, block and transaction delays, block sizes, request acceptance rates
[27]	Computation outsourcing	Decentralized market of computational resources based on blockchain and smart contracts	✓	✗	✓	PoA	Equilibrium analysis, proof of concept	CPU and memory usages, gas costs, function latencies
[6]	Fog computing	Resource orchestration based on blockchain and smart contracts	✓	✗	✗	PoA	Physical testbed	CPU, memory and disk usages, block sizes, power consumption
[10]	Fog computing	Resource orchestration based on blockchain and SDN; domain-based coordination	✓	✓	✗	PoA	Physical testbed	Request latency and availability
[43]	Vehicular edge computing	Secure and reliable vehicular task allocation system for computation offloading among smart contract-enabled vehicles	✓	✗	✗	PBFT	Simulation	Average delay of offloading tasks, average utility
[44]	5G IoT networks	SDN-enabled centralized energy-aware task offloading (off-chain); blockchain to communicate offloading decisions	✗	✗	✗	PoA	Emulation	Network latency, transaction throughput and energy consumption
[19]	Online payments	Decentralized and scalable off-chain payments	✓	✓	✗	BRB	Testbed (public cloud network)	Payment latencies and throughput
Our work	Fog computing	Scalable, decentralized and consensus-less resource orchestration in Fog-IoT environments	✓	✓	✓	BRB	Security analysis, testbed	Bandwidth, CPU and memory usages, latency and throughput of offloading events

point). The task assignment and offloading decisions are communicated securely through the blockchain, by means of a set of smart contracts on top of a PoA-based network. PoA is shown to outperform Proof-of-Work (PoW) and PBFT algorithms in terms of network latency, transaction throughput and energy consumption.

Our in-depth comparison with HIDRA, which also uses a PoA consensus, showcases the main benefit of Light-HIDRA to lower the overhead incurred by conventional blockchains. Existing distributed ledger technologies pose some performance and scalability issues that the scientific community is currently analyzing [47–49]. In the field of online payments, decentralized alternatives beyond those based on blockchain technology have been explored. Authors in [19] propose Astro, an off-chain decentralized payment system based on the BRB broadcast primitive that avoids the need for consensus to prevent double-spending. Astro employs a sharding scheme to partition nodes into different shards or domains. The goal of sharding is to improve scalability and avoid the requirement of a global system state shared by all nodes. Whereas Astro focuses its efforts on designing a payment system, we go a step further with Light-HIDRA by proposing a decentralized resource orchestration system that also implements payments in order to reward peers for the resources provided to the network.

8. Conclusion

This paper introduced Light-HIDRA, a novel and efficient approach for decentralized resource orchestration within IoT ecosystems. By building upon the Byzantine Reliable Broadcast algorithm for the orchestration of resources, Light-HIDRA bypasses the need for consensus, dramatically reducing communication overhead and latency compared to existing approaches that rely on distributed ledger technology. Our system achieves further scalability by grouping peers into distinct domains and reducing inter-domain communication.

Our comprehensive security analysis and experimental evaluation show Light-HIDRA's robustness and efficiency. Notably, our experimental results reveal the superior performance of Light-HIDRA over HIDRA, a state-of-the-art approach for decentralized resource orchestration. Specifically, Light-HIDRA reduces the bandwidth usage by up to 57 times and the latency of workload offloading by up to 142 times, whilst maintaining higher throughput. These findings establish Light-HIDRA as a promising solution for resource orchestration.

As future work, we plan to explore real-world use cases where we could apply the Light-HIDRA approach, such as inter-campus computing groups or resource offloading marketplaces. We also intend to study the use of Light-HIDRA in more open Fog-IoT environments. To achieve this, we will design a reputation system to effectively measure and quantify the behavior of both domains and peers, aiming to mitigate common trust attacks in open networks such as Sybil attacks.

CRediT authorship contribution statement

Carlos Núñez-Gómez: Conceptualization, Investigation, Methodology, Software, Visualization, Writing – original draft, Writing – review & editing. **Martijn de Vos:** Conceptualization, Methodology, Resources, Validation, Visualization, Writing – original draft, Writing – review & editing. **Jérémie Decouchant:** Conceptualization, Methodology, Resources, Validation, Visualization, Writing – review & editing. **Johan Pouwelse:** Funding acquisition. **Blanca Caminero:** Resources, Validation, Writing – original draft, Writing – review & editing. **Carmen Carrión:** Resources, Validation, Writing – review & editing.

Declaration of competing interest

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

Data availability

Data will be made available on request.

Acknowledgments

This work was supported by MCIN/AEI/10.13039/501100011033 and European Regional Development Fund (ERDF), "A way to make Europe" (ref. PID2021-123627OB-C52), and under 2023-GRIN-34056 grant funded by the Regional Government of Castilla-La Mancha for Consolidated Research Groups.

References

[1] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I.A.T. Hashem, A. Siddiqa, I. Yaqoob, Big IoT data analytics: architecture, opportunities, and open research challenges, IEEE Access 5 (2017) 5247–5261.
[2] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, 2012, pp. 13–16.
[3] B. Jamil, H. Ijaz, M. Shojafar, K. Munir, R. Buyya, Resource allocation and task scheduling in fog computing and internet of everything environments: A taxonomy, review, and future directions, ACM Comput. Surv. 54 (11s) (2022) 1–38.
[4] I.B. Lahmar, K. Boukadi, Resource allocation in fog computing: A systematic mapping study, in: 2020 Fifth International Conference on Fog and Mobile Edge Computing, FMEC, IEEE, 2020, pp. 86–93.
[5] C. Carrión, Kubernetes scheduling: Taxonomy, ongoing issues and challenges, ACM Comput. Surv. 55 (7) (2022) 1–37.
[6] C. Núñez-Gómez, B. Caminero, C. Carrión, HIDRA: A distributed blockchain-based architecture for fog/edge computing environments, IEEE Access 9 (2021) 75231–75251, http://dx.doi.org/10.1109/ACCESS.2021.3082197.
[7] S. Jošilo, G. Dán, Decentralized algorithm for randomized task allocation in fog computing systems, IEEE/ACM Trans. Netw. 27 (1) (2018) 85–97.
[8] Z.A. Mann, Decentralized application placement in fog computing, IEEE Trans. Parallel Distrib. Syst. 33 (12) (2022) 3262–3273.
[9] S. Tuli, R. Mahmud, S. Tuli, R. Buyya, Fogbus: A blockchain-based lightweight framework for edge and fog computing, J. Syst. Softw. 154 (2019) 22–36.
[10] C. Núñez-Gómez, C. Carrión, B. Caminero, F.M. Delicado, S-HIDRA: A blockchain and SDN domain-based architecture to orchestrate fog computing environments, Comput. Netw. 221 (2023) 109512, http://dx.doi.org/10.1016/j.comnet.2022.109512.
[11] Z. Xiong, S. Feng, W. Wang, D. Niyato, P. Wang, Z. Han, Cloud/fog computing resource management and pricing for blockchain networks, IEEE Internet Things J. 6 (3) (2018) 4585–4600.
[12] Y. Jiao, P. Wang, D. Niyato, K. Suankaewmanee, Auction mechanisms in cloud/fog computing resource allocation for public blockchain networks, IEEE Trans. Parallel Distrib. Syst. 30 (9) (2019) 1975–1989.
[13] P.W. Eklund, R. Beck, Factors that impact blockchain scalability, in: Proceedings of the 11th International Conference on Management of Digital Ecosystems, 2019, pp. 126–133.
[14] B. Nasrulin, M. De Vos, G. Ishmaev, J. Pouwelse, Gromit: Benchmarking the performance and scalability of blockchain systems, in: 2022 IEEE International Conference on Decentralized Applications and Infrastructures, DAPPS, IEEE, 2022, pp. 56–63.
[15] G. Bracha, Asynchronous Byzantine agreement protocols, Inform. and Comput. 75 (2) (1987) 130–143.
[16] S. Bonomi, J. Decouchant, G. Farina, V. Rahli, S. Tixeuil, Practical Byzantine reliable broadcast on partially connected networks, in: 2021 IEEE 41st International Conference on Distributed Computing Systems, ICDCS, IEEE, 2021, pp. 506–516.
[17] D. Kozhaya, J. Decouchant, P. Esteves-Verissimo, RT-ByzCast: Byzantine-resilient real-time reliable broadcast, IEEE Trans. Comput. 68 (3) (2018) 440–454.
[18] D. Kozhaya, J. Decouchant, V. Rahli, P. Esteves-Verissimo, Pistis: an event-triggered real-time byzantine-resilient protocol suite, IEEE Trans. Parallel Distrib. Syst. 32 (9) (2021) 2277–2290.
[19] D. Collins, R. Guerraoui, J. Komatovic, P. Kuznetsov, M. Monti, M. Pavlovic, Y.-A. Pignolet, D.-A. Seredinschi, A. Tonkikh, A. Xygkis, Online payments by merely broadcasting messages, in: 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, IEEE, 2020, pp. 26–38.
[20] O. Ersoy, J. Decouchant, S.P. Kumble, S. Roos, Syncpcn/psyncpcn: Payment channel networks without blockchain synchrony, in: Proceedings of the 4th ACM Conference on Advances in Financial Technologies, 2022, pp. 16–29.
[21] C. Cachin, J.A. Poritz, Secure intrusion-tolerant replication on the internet, in: Proceedings International Conference on Dependable Systems and Networks, IEEE, 2002, pp. 167–176.
[22] C. Cachin, R. Guerraoui, L. Rodrigues, Reliable broadcast, in: Introduction to Reliable and Secure Distributed Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 73–135, http://dx.doi.org/10.1007/978-3-642-15260-3_3.
[23] J. Wang, H. Wang, Monoxide: Scale out blockchains with asynchronous consensus zones, in: NSDI, Vol. 2019, 2019, pp. 95–112.
[24] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, B. Ford, Omniledger: A secure, scale-out, decentralized ledger via sharding, in: 2018 IEEE Symposium on Secur. Priv., SP, IEEE, 2018, pp. 583–598.
[25] J. Hu, K. Yang, K. Wang, K. Zhang, A blockchain-based reward mechanism for mobile crowdsensing, IEEE Trans. Comput. Soc. Syst. 7 (1) (2020) 178–191, http://dx.doi.org/10.1109/TCSS.2019.2956629.
[26] H. Wang, L. Wang, Z. Zhou, X. Tao, G. Pau, F. Arena, Blockchain-based resource allocation model in fog computing, Appl. Sci. 9 (24) (2019) http://dx.doi.org/10.3390/app9245538.
[27] S. Eisele, T. Eghtesad, N. Troutman, A. Laszka, A. Dubey, Mechanisms for outsourcing computation via a decentralized market, in: Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems, DEBS '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 61–72, http://dx.doi.org/10.1145/3401025.3401737.
[28] S.a. Jošilo, G. Dán, Selfish decentralized computation offloading for mobile cloud computing in dense wireless networks, IEEE Trans. Mob. Comput. 18 (1) (2019) 207–220, http://dx.doi.org/10.1109/TMC.2018.2829874.
[29] C. Dwork, N. Lynch, L. Stockmeyer, Consensus in the presence of partial synchrony, J. ACM 35 (2) (1988) 288–323.
[30] M. Haghi Kashani, A.M. Rahmani, N. Jafari Navimipour, Quality of service-aware approaches in fog computing, Int. J. Commun. Syst. 33 (8) (2020) e4340, http://dx.doi.org/10.1002/dac.4340.
[31] J.R. Douceur, The sybil attack, in: P. Druschel, F. Kaashoek, A. Rowstron (Eds.), Peer-to-Peer Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 251–260.
[32] R. John, J.P. Cherian, J.J. Kizhakkethottam, A survey of techniques to prevent sybil attacks, in: 2015 International Conference on Soft-Computing and Networks Security, ICSNS, 2015, pp. 1–6, http://dx.doi.org/10.1109/ICSNS.2015.7292385.
[33] N. Tripathi, N. Hubballi, Application layer denial-of-service attacks and defense mechanisms: A survey, ACM Comput. Surv. 54 (4) (2021) http://dx.doi.org/10.1145/3448291.
[34] A. Huseinović, S. Mrdović, K. Bicakci, S. Uludag, A survey of denial-of-service attacks and solutions in the smart grid, IEEE Access 8 (2020) 177447–177470, http://dx.doi.org/10.1109/ACCESS.2020.3026923.
[35] R. Singh, S. Tanwar, T.P. Sharma, Utilization of blockchain for mitigating the distributed denial of service attacks, Secur. Priv. 3 (3) (2020) e96, http://dx.doi.org/10.1002/spy2.96.
[36] B. Costa, J. Bachiega, L.R. de Carvalho, A.P.F. Araujo, Orchestration in fog computing: A comprehensive survey, ACM Comput. Surv. 55 (2) (2022) http://dx.doi.org/10.1145/3486221.
[37] K3s Project Authors, K3s: Lightweight Kubernetes, 2023, URL https://k3s.io.
[38] KubeEdge Project Authors, KubeEdge: Kubernetes native edge computing framework, 2023, URL https://kubeedge.io.
[39] Karmada Authors, Karmada: Open, multi-cloud, multi-cluster kubernetes orchestration, 2023, URL https://karmada.io.
[40] J. Pan, J. Wang, A. Hester, I. Alqerm, Y. Liu, Y. Zhao, EdgeChain: An edge-IoT framework and prototype based on blockchain and smart contracts, IEEE Internet Things J. 6 (3) (2019) 4719–4732, http://dx.doi.org/10.1109/JIOT.2018.2878154.
[41] Kanupriya, I. Chana, R.K. Goyal, Computation offloading techniques in edge computing: A systematic review based on energy, QoS and authentication, Concurr. Comput.: Pract. Exper. (2024) e8050, http://dx.doi.org/10.1002/cpe.8050.
[42] T. Nguyen, R. Katila, T.N. Gia, An advanced Internet-of-Drones System with Blockchain for improving quality of service of Search and Rescue: A feasibility study, Future Gener. Comput. Syst. 140 (2023) 36–52, http://dx.doi.org/10.1016/j.future.2022.10.002.
[43] J. Shi, J. Du, Y. Shen, J. Wang, J. Yuan, Z. Han, DRL-Based V2V Computation Offloading for Blockchain-Enabled Vehicular Networks, IEEE Trans. Mob. Comput. 22 (7) (2023) 3882–3897, http://dx.doi.org/10.1109/TMC.2022.3153346.
[44] B. Sellami, A. Hakiri, S. Ben Yahia, Deep reinforcement learning for energy-aware task offloading in join SDN-blockchain 5G massive IoT edge network, Future Gener. Comput. Syst. 137 (2022) 363–379, http://dx.doi.org/10.1016/j.future.2022.07.024.
[45] A. Samy, I.A. Elgendy, H. Yu, W. Zhang, H. Zhang, Secure task offloading in blockchain-enabled mobile edge computing with deep reinforcement learning, IEEE Trans. Netw. Serv. Manag. 19 (4) (2022) 4872–4887, http://dx.doi.org/10.1109/TNSM.2022.3190493.
[46] A. Heidari, M.A. Jabraeil Jamali, N. Jafari Navimipour, S. Akbarpour, Deep Q-learning technique for offloading offline/online computation in blockchain-enabled green IoT-edge scenarios, Appl. Sci. 12 (16) (2022) http://dx.doi.org/10.3390/app12168232.
[47] M.H. Nasir, J. Arshad, M.M. Khan, M. Fatima, K. Salah, R. Jayaraman, Scalable blockchains – A systematic review, Future Gener. Comput. Syst. 126 (2022) 136–162, http://dx.doi.org/10.1016/j.future.2021.07.035.
[48] D. Yang, C. Long, H. Xu, S. Peng, A review on scalability of blockchain, in: Proceedings of the 2020 the 2nd International Conference on Blockchain Technology, ICBCT '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1–6, http://dx.doi.org/10.1145/3390566.3391665.
[49] Q. Zhou, H. Huang, Z. Zheng, J. Bian, Solutions to scalability of blockchain: A survey, IEEE Access 8 (2020) 16440–16455, http://dx.doi.org/10.1109/ACCESS.2020.2967218.

Carlos Núñez-Gómez graduated from University of Castilla-La Mancha as a computer engineer. In 2019, he obtained a M.Sc. degree in information and communication security at Universitat Oberta de Catalunya. Subsequently in 2019, he started a Ph.D. degree in advanced computer technologies at UCLM. He is a member of the High-Performance Networks and Architectures Group (RAAP) at Albacete Research Institute of Informatics (I3A). His research interests include distributed systems, blockchain, information security and fog computing environments.

Martijn de Vos is a postdoctoral researcher at the Scalable Computing Systems Laboratory (SACS) at EPFL, Switzerland. His research focuses on developing lightweight decentralized solutions and decentralized machine learning.

Jérémie Decouchant is an Assistant Professor at the Distributed Systems section of the Software Technology department of TU Delft, the Netherlands. His research interests include resilient distributed computing, privacy-preserving systems, blockchain, and distributed machine learning.

Johan Pouwelse is an associate professor at the Distributed Systems section of the Software Technology department of TU Delft, specialized in large-scale cooperative systems. He is the founder of Tribler, a living laboratory and proving ground for next generation self-organizing systems research and ledger technology.

Blanca Caminero is a Full Professor in Computer Architecture and Technology at the Computing Systems Department at UCLM. She teaches networking related subjects at the School of Computer Science and Engineering in Albacete since 2000. She holds a Ph.D. Degree in Computer Science from the UCLM, and carries out her research in the High-Performance Networks and Architectures Group (RAAP) in the Albacete Research Institute of Informatics (I3A). Her current research interests are QoS support and efficient resource scheduling in distributed systems (Cloud, Fog, Edge, ...). She is a member of the IEEE.

Carmen Carrión is a Full Professor in Computer Architecture and Technology at the Computing Systems Department at The University of Castilla-La Mancha. She holds a Ph.D. Degree in Physics from The University of Cantabria, and her interests include Higher Education, resource management schemes, virtualization technologies and QoS in Fog-IoT frameworks. She is a member of IEEE.