Decentralized Routing for In-Network Computing Cooperating with Distributed Ledger Technologies

沒辦法用centralized controller來分配resource, 提出用distributed ledger來分享reputation

***Abstract*—Distributed and decentralized in-network computing is a key technology for building scalable, high-performance, and trustable future applications. However, an effective decentralized architecture of in-network computing has not yet been discussed in the literature because making a resource assignment decision without a centralized controller is a significant challenge. In this study, we propose a Decentralized Routing Framework for In-Network Computing (DRINC) that cooperates with distributed ledger technologies to share trustworthy reputations for distributed computing resources and to make deterministic routing decisions in a decentralized manner. We also propose an algorithm called Maximal Satisfactory Routing (MSR) for DRINC that determines the optimal routing of the process requests to maximize the satisfactory score, which represents the degree of fulfillment of requirements in the quality of the data process. The satisfactory score is calculated by mutual evaluations of the multiple computing resources stored in the distributed ledger. We evaluated our proposals by simulations assuming machine learning applications in a distributed edge computing environment and confirmed that the DRINC with MSR can process the stream data with high accuracy, minimizing the response time by forwarding requests to appropriate computing resources.**

***Index Terms*—distributed ledger technology, in-network computing, stream data process**

I. Introduction

Recent advances in technologies for in-network computing (or computing in the network [1]) and edge computing [2]–[4] have made it possible to deploy data processing functions to the distributed computing resources located close to end-user terminals or inside distributed network equipment, instead of deploying to cloud servers. Applications can benefit from a low-latency response and high computing performance in an easier, more flexible, and scalable manner. Additionally, the collaboration of multiple servers distributed in the network enables the integration of computing resources to complete large-scale computing tasks.

In such an in-network computing model, there are several trade-offs regarding the selection of computing resources to process tasks, such as analytic tasks using machine learning. In general, although the in-network computing resources (e.g., edge servers) have small network latency/overhead to reach from an end device, they have less CPU power/memory than servers in the cloud. Therefore, there is a trade-off between the network latency/overhead and process latency. There is also a trade-off between network latency and the quality of the process results. For example, in machine-learning applications, many datasets are composed and trained using a variety of models in a cloud server. Thus, the accuracy of the prediction in an edge server is lower than that in a cloud server. Users can satisfy the accuracy of the given results only when the edge server installs the appropriate prediction model and trained datasets work properly.

Several recent studies have investigated strategies for assigning in-network computing resources by considering such trade-offs to achieve high-performance services [5]–[7]. However, most studies assume a centralized controller to manage resource assignments. Although this simplifies the system construction, it inevitably brings a single point of failure. A centralized controller can potentially be used as a source of control to apply surveillance or censorship, or it can lead to the abuse of trust. However, because multi-server collaboration involves multiple servers from different organizations, decentralizing the controller faces difficulty in integrating biased information, security, and trust.

To address such issues, distributed ledger technology (DLT) [8]–[10] has attracted considerable attention as a promising direction for building a decentralized network. DLT can effectively provide unbiased and consistent information with multiparty consensus. Several studies have utilized DLT to give security, privacy, and consensus to in-network computing services in a decentralized manner [11]–[14]. However, since making a resource assignment decision without a centralized controller is a technically challenging issue, there has been little research on solving the trade-offs of in-network computing that have an affinity for DLT.

In this paper, we propose a trustable decentralized in-network data processing framework called Decentralized Routing Framework for In-Network Computing (DRINC). DRINC provides a high-quality data processing service that solves the trade-offs in computing resource assignments through cooperation with DLT. We focus on *stream data processing*, which processes an unbounded stream of continuous data in real or near real-time. Stream data processing is the basic function of many applications such as IoT sensor data processing, data analytics, and dynamic content delivery.

Our contributions can be summarized as follows:

- We describe the novel *Decentralized Routing Framework for In-Network Computing (DRINC)* and show its basic design. DRINC stores the trustworthy and unbiased ledger states of the distributed computing resources in DLT. The ledger states represent the reputation of computing resources. The states are updated by mutual evaluation reports from multiple computing resources. The requests for stream data processing are assigned and

routed to the appropriate computing resources according to the ledger states.
- We define a metric called *satisfactory score* to represent the expected degree of fulfillment of requirements in the quality of the stream data process. The satisfactory score is calculated using the reports in the DRINC. We also propose a Maximal Satisfactory Routing (MSR) algorithm to route the requests to the optimal computing resources that maximize the satisfactory score deterministically.
- We evaluate the DRINC with MSR by simulations assuming typical machine learning applications with skewed models in an edge computing environment.

The remainder of this paper is organized as follows. In Section II, we present related work. In Section III, we present a system design for DRINC, including an overview of the applied DLT. We then describe the details of the MSR algorithm in Section IV. We present the definitions of the score functions for applying MSR to machine learning applications in Section V. In Section VI, we present the evaluation results of simulations that apply machine learning applications. Finally, in Section VII, we present our conclusions.

II. Related Work

Several studies have focused on the optimal assignment of resources in in-network computing or edge computing environments to solve the tradeoffs [5]–[7]. However, they assumed a centralized controller for decision-making. In addition, most of them assumed that the expected performance in computing resources was known beforehand in a centralized controller, which is an impractical assumption.

Some decentralized algorithms have been proposed for computation offloading in edge computing. Pham et. al [15] proposed a decentralized computation offloading and resource allocation algorithm to minimize the system-wide computation overhead based on matching theory. Wang et. al. [16] proposed a distributed computation offloading algorithm using multi-agent imitation learning. Chen and Wang [17] proposed a decentralized computation offloading framework based on a deep deterministic policy-gradient algorithm. These algorithms aim to minimize the computation offloading overhead and reduce task completion time. A decentralized decision is made to execute a task locally or offload it to a single server or either of the multiple edge servers. Nevertheless, none of the existing studies discussed decisions on solving trade-offs in process executions, including the quality of the process result. In addition, none of these studies considered the integration of DLT.

Studies on DLT-based data processing have also been reported in the literature. Careen and Dutta [18] proposed a reputation-based routing in mobile ad hoc networks (MANET) using blockchain to maintain the correct routing in order to prevent malicious routing and mining by adversaries. Aliyu et. al. [14] proposed a federated learning system using blockchain. The blockchain was used to reduce the risk of model poisoning. Abdellati et al. proposed Medge-Chain [19], which is a blockchain-based medical data processing framework. Medge-Chain includes an optimized blockchain configuration for minimal latency and computational cost, considering the priority of the medical data. BDLP [12] is a data life-cycle management framework for an information-centric network (ICN) [20], [21]. BDLP defines procedures to protect data in ICN from the misbehavior of a malicious data process using blockchain technology. These approaches incorporate DLT or blockchain technology to prevent attacks by malicious users or to provide incentive mechanisms. Although the procedures used in these approaches can be applied to generic data processing, utilizing the information stored in DLT for resource assignments or routings in decentralized computation has not been discussed.

BlockTC [13] is a blockchain-based architecture that implements a multiplex mutual trust networking and collaborative routing. BlockTC provides a routing verification scheme that prevents the disclosure of private information in each domain while collaborating with multi-domain edge computing servers. Although BlockTC aims to provide trusted routing and privacy protection, the assignment of computing resources was out of scope.

Overall, none of the existing studies utilized DLT to make optimal resource assignment decisions that provide trustable services.

III. Design of Decentralized Routing Framework for In-network Computing

We propose a trustable decentralized routing framework for in-network computing, called DRINC.

The principal features in DRINC that differentiate it from existing frameworks are the DLT function that maintains the unbiased states of the distributed computing resources and the routing function that routes the stream data process requests according to the current states in the DLT network. This section describes the entities and procedures of DRINC.

A. Distributed Ledger Technology

The following is a brief explanation of the DLT applied to DRINC.

DLT maintains a distributed ledger, which is a distributed record of transactions with consensus among a network of distributed computers (DLT nodes). DLT enables the operation of highly available and append-only data in an untrustworthy environment. The transactions are duplicated across all DLT nodes in the network, such that every DLT node has an identical copy of all transaction records. The protocols in DLT ensure that all DLT nodes have an identical copy of the distributed ledger records, with the consensus of the majority of DLT nodes. There is no centralized authority for the records. The ledger stores the current state of the objects and the history of the transactions that led to the current state. The history is immutable and thus regarded as a tamper-proof audit trail. A DLT node may be either a full-node or a partial-node. A full-node stores complete ledger records locally, which means that the full-node downloads all transactions generated in the DLT network. A partial-node

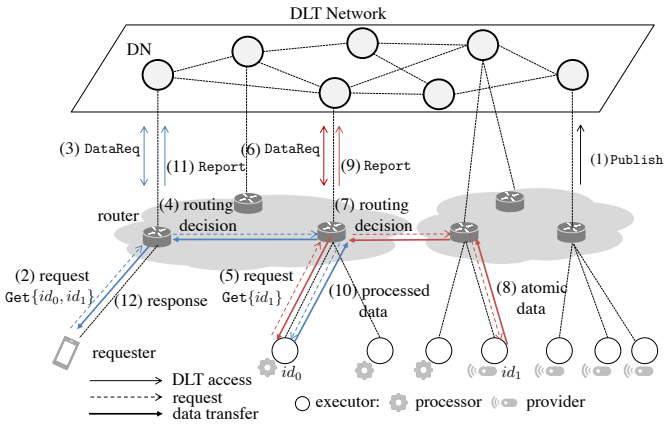


Fig. 1. The system model of DRINC

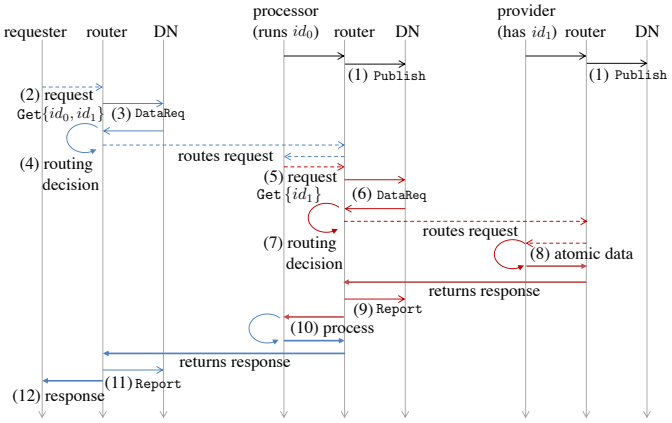


Fig. 2. The sequence of stream data process $\{id_0, id_1\}$

does not have all records locally and connects to the full-node to receive transactions that are necessary and relevant to the operations executed on the partial-node. DRINC does not rely on a specific DLT implementation and can work with permissionless blockchain (e.g., Bitcoin [22] and Ethereum [23]), permissioned blockchain (e.g., Hyperledger Fabric [24] and R3 Corda [25]), and other technologies [26].

B. System Model

Consider an in-network data processing system. In DRINC, the entities involved in the stream data processing are as follows.

- **Requester**: an entity that requests a stream data process.
- **Provider**: an entity that publishes and provides atomic data (non-processed raw data). A provider typically corresponds to a sensor. The content of the provided atomic data changes depending on time.
- **Processor**: an entity that publishes one or more functions to process data. The processor acts as a requester that issues requests to retrieve its input. The input may be atomic data or an output of another processor. The processor also provides the output of the process as content. A processor may have the ability to execute multiple types of data process functions.
- **Router**: an entity that associates with the above entities with authentication. The router determines the assignments of the requests continuously issued for the stream data process from requesters to the entities associated with itself or other routers. The router routes the request to the assigned entity. The router executes the DLT procedures in the DRINC on behalf of the associated entities.
- **DLT node (DN)**: an entity that serves as a node in the DLT network to store the states of the entities and corresponding transactions. Each router is associated with one DN. One DN may accommodate multiple routers. There may be DNs that are not associated with routers. DNs participate in the DLT network and run the DLT protocols such as propagating transactions, generating blocks, and making consensus.

Note that a physical host can run multiple entities (e.g., a host may have the roles of processor, router, and DN). Hereafter, we refer to the entities that provide content (i.e., providers and processors) as *executors*.

In each stream data process, requests are issued and processed periodically until the end of the stream (either the lifetime of the stream expires or the requester explicitly stops the stream). Each request is routed to one of the executors in the in-network data processing system. The network transport (data plane) of DRINC delivers requests to the entity and retrieves the content data with a specified identifier. There are several options for implementing network transport, such as TCP/IP, QUIC [27], and HTTP(S). Applying other promising approaches, such as information-centric networking (ICN) [20], [21] and key-based peer-to-peer routing protocols [28], [29], may improve the performance and reliability of data retrieval.

C. Procedures in DRINC Framework

We defined the procedures of DRINC that ensure the trust of in-network stream data processing. The principal procedures of DRINC are: 1) Registration, 2) Publish, 3) Data Request, and 4) Report.

1) Registration: For the registration procedure, a requester, provider, or processor discovers and connects to a router to participate in the system. The router authenticates the associated entities. Each entity in DRINC has its own identity (with public and private keys).

2) Publish: Once a new executor (provider or processor) is associated with a router, the router executes a `Publish` procedure for the contents on the DN. The `Publish` procedure registers the identifiers of the executor and content information in the DN so that the executor is discoverable in the system.

3) Data Request: When a requester requests a stream data process, the request is sent to the router. The router invokes a data request procedure (`DataReq`) on the DN. The router decides the remote executor to forward the request according to the returned information in DataReq. The returned information

is derived from current states of executors in DLT which varies depending on the routing algorithm. If the requester does not pass the validations by the DN, for example, the requester is not authorized to access the content, data access permission is not given, and the request is denied.

4) Report: The router invokes a `Report` procedure on the DN that includes an evaluation of the executor that provided the received content. The report procedure creates an unbiased reputation for the executor on the distributed ledger through mutual evaluation of the entities. The content provided by the executor is evaluated by an *evaluator*. The evaluator can be any entity in DRINC. This report alters the state of the executor, which may affect the access permission of the executor as a requester or the routing decision in a router.

Fig. 1 illustrates the system model of DRINC. The requester at the bottom left requests $\{id_0, id_1\}$ which corresponds to stream data that applys a function id_0 to atomic data id_1. Fig. 2 shows the corresponding message exchange sequence. To obtain the processed data of $\{id_0, id_1\}$, a `Get` request is forwarded to a processor that publishes function id_0. The processor then sends another `Get` request to id_1 for the input of id_0. The request is forwarded to a provider that publishes the atomic data id_1. `DataReq` and `Report` procedures are invoked accordingly during the stream data process.

IV. Maximal Satisfactory Routing

We propose a routing algorithm called *maximal satisfactory routing (MSR)* as a distributed routing algorithm in DRINC. MSR determines the routing of requests according to the *satisfactory score* defined for each application domain. A satisfactory score corresponds to a metric that represents the quality of service of the stream data process, calculated by the mutual evaluations.

A. Node and Network Configurations

Table I summarizes the notations used in the remainder of this paper.

Consider a distributed in-network computing environment consisting of a set of computing resources in the network. MSR assumes that each computing resource is partitioned into multiple virtual nodes. A virtual node corresponds to a process that runs on a physical host. The virtual node may be a virtual machine or a container allocated to dedicated CPUs and/or GPUs in a physical host. One executor in DRINC corresponds to one virtual node. Virtual nodes in the same physical host are homogeneous. In other words, virtual nodes in the same physical host exhibit the same performance.

We denote the set of virtual nodes in the system by V and the host of the virtual node $v \in V$ as $h(v)$. In DRINC, each virtual node has a set of *properties*. The property value of $v \in V$ represents the quality of the content provided by v such as the response time, frame rate, resolution, and accuracy.

In DRINC, a requester requests a stream data process. One requester issues one request. The set of requests in the system is denoted as R. MSR determines the routing of request $r \in R$ to the virtual node $v \in V$ that corresponds to the executor publishing the requested content. We denote the content that corresponds to the output of executor x for request y as $c(x, y)$.

TABLE I
Notations

Notation	Meaning
V	the set of virtual nodes in the system.
R	the set of requests in the system.
P	the set of properties used to score virtual nodes.
E	the set of evaluators for calculating the score.
$score(x, y)$	satisfactory score of virtual node x for request y.
$score_p(x, y)$	property score of property p. x is the target virtual node and y is the request.
$f_p(x, y)$	score function of property p. x, y are the contents provided by the executor and evaluator.
$c(x, y)$	content provided by executor x for request y.
$h(x)$	host of virtual node x.
I_{route}	the interval to update the routing decision.
I_{eval}	the interval to evaluate and report.
I_{score}	the interval to retain scores.

In MSR, the state of the executor in DLT corresponds to the *property scores*. The property score represents the expected degree of fulfillment of the requirement for a property value in the content when the request is assigned to a virtual node. The router in DRINC calculates the property score for each stream data request and virtual node. Property scores are stored and shared as the current state of the virtual node.

The property scores are determined by mutual evaluation of the virtual nodes using the report procedure in DRINC. For mutual evaluation, MSR allocates a virtual node dedicated to evaluations in each physical host. These virtual nodes act as evaluators. Because the virtual nodes in the same physical host are homogeneous, the properties of the content obtained as the output of an executor have the same value as that of an evaluator.

B. Problem Formulation

Let us now show the formal definition of MSR. We define a satisfactory score as the routing metric. A satisfactory score is calculated using the set of property scores of a virtual node. The property score is the accumulated score of the individual scores reported by the `Report` procedures.

MSR calculates the property score of an executor by using the contents provided by the executor and an evaluator for the same request. Consider property p in virtual node v for requrest r. The individual score of p in v is calculated by using the value of p in $c(v, r)$ and $c(e, r)$, where e is the evaluator. The individual score of p is calculated using the *score function* f_p, which is defined depending on the property. The individual score of p is represented by $f_p(c(v, r), c(e, r))$. The return value of the score function represents a degree of fulfillment of the requirement by a requester. The accumulated score is the average value of the individual scores. We denote

the accumulated score of property p in v for r as $score_p(v, r)$, which is represented as

$$score_p(v, r) = \frac{1}{|E|} \sum_{e \in E} f_p(c(v, r), c(e, r)) \quad (1)$$

where E denotes a set of evaluators. The range of scores is normalized to $[0, 1]$. The score of a property equals 0 or 1 if the property has a minimum or maximum quality, respectively.

We define the satisfactory score of request r in v using the property scores for the properties in a virtual node.

Definition 1 (satisfactory score)**.** *The satisfactory score of content provided by virtual node v for request r is denoted by $score(v, r)$, which is calculated by*

$$score(v, r) = \sum_{p \in P} w_p \; score_p(v, r) \quad (2)$$

where P denotes the set of properties and w_p is a weight parameter that satisfies $\sum_{p \in P} w_p = 1$.

The objective of MSR is to find a bijection from R to V that maximizes the total satisfactory score in the system. All virtual nodes in V are candidates for routing request r to obtain the content. A virtual node that does not publish the requested content has a satisfactory score of -1. In this case, the request is not routed to the virtual node.

We consider a set of binary decision variables $X_{vr} \forall v \in V, r \in R$ where $X_{vr} = 1$ when request r is routed to virtual node v. The problem is formulated as follows.

$$Maximize: \quad \frac{1}{|R|} \sum_{v \in V, r \in R} score(v, r) \quad (3a)$$

$$s.t.: \quad \sum_{v \in V} X_{vr} \leq 1, \; \forall r \in R \quad (3b)$$

$$\sum_{r \in R} X_{vr} = 1, \; \forall v \in V \quad (3c)$$

$$X_{vr} \in \{0, 1\}, \; \forall r \in R, v \in V \quad (3d)$$

Constraint (3b) indicates that each request can be served by one virtual node, or the request is dropped. Constraint (3c) indicates that only one request is assigned to a virtual node. This problem can be viewed as an *assignment problem*, which is known to be solved in a polynomial time. The well-known Hungarian Algorithm [30] solves the problem with complexity $O(|V|^3)$ assuming $|R| < |V|$.

C. The Algorithm of MSR

Fig. 3 illustrates an overview of the algorithm.

Algorithm 1 shows the pseudocode of the routing procedure in MSR. The pseudocode shows how the `Get` request r is handled by router n. As an input, the requester is given as s. The virtual node to forward to is given as v if the routing is already determined.

In the initial stage, the requester sends `Get` request r with $v = nil$ to the associated router. r may include the requirements for property values, such as response time thresholds

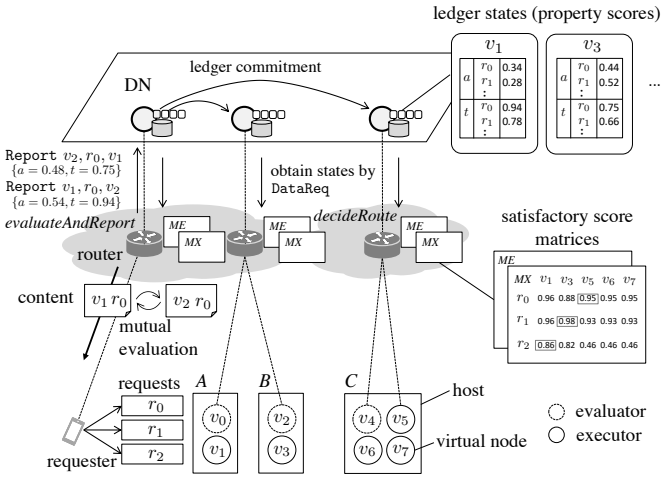


Fig. 3. The overview of MSR

Algorithm 1: the routing in MSR

```
1  //  v : virtual node to forward a request (nil if undecided)
2  //  r : the request
3  upon request ⟨Get, v, r⟩ on n
4      t ← nil, e ← nil
5      if v = nil then
6          t ← n.last[r]
7          if (t = nil) ∨ (I_route expired for r) then
8              v, e ← decideRoute(n.DN, r)
9      if v.router ≠ n then
10         // v is associated with another router
11         t ← v.router
12     else
13         t ← v
14     q ← copy r, q.sender ← n
15     x ← send request ⟨Get, v, q⟩ to t
16     send response⟨x⟩ to r.sender
17     n.last[r] ← t
18     if (e ≠ nil) ∧ (I_eval expired for r) then
19         evaluateAndReport(n, x, e)
```

to be considered satisfactory from the requester's perspective. Lines 5-8 describe the routing decision process. The routing decision is updated if the previous routing decision for the same stream data process does not exist or I_{route} is expired for r (line 7). I_{route} is the predefined time interval to update the routing decisions. The router decides the routing for r by invoking the *decideRoute* function (described in Algorithm 2) cooperating with the associated DN. The routing decision process can be executed asynchronously. The function decideRoute returns the executor and the evaluator to forward the request to. The details are presented in IV-D.

After the routing decision is made, r is forwarded to the virtual node or router to reach v in lines 9 to 15. The obtained content is returned to the requester in line 16. Line 17 stores the current routing decisions for the next stream data request. Lines 18 and 19 invoke the *evaluateAndReport* function (described in Algorithm 3). The evaluateAndReport function determines the property scores through mutual evaluation of

entities and sends reports. The details are presented in IV-E. In the stream data process, requests are generated repeatedly and continuously to apply a function to the data from the data source. Because the evaluation result is not expected to change significantly within a short period of time, evaluateAndReport is invoked when the I_{eval} is expired for r, which reduces the system overhead for the evaluations. I_{eval} is the predefined time interval used to evaluate the received content and report the evaluation result.

D. Making Routing Decision

Algorithm 2 shows the pseudocode of the *decideRoute* function. The decideRoute function decides the route by obtaining the current states in d, which is the associated DN of the router. The router invokes `DataReq` procedure of DRINC to access the current states and extracts V and R (line 4). V is a set of executors that can provide content for request r. R is the set of stream data process requests. $v \in V$ includes the property scores accumulated using the scores in reports.

The decideRoute function determines the routing for r by solving Eq. (3a). The routing decisions of r for the executor and evaluator differ. To determine the executor and evaluator, the function generates matrices *MX* and *ME*, respectively. These matrices are initialized in lines 5 and 6. Lines 7–17 generate $|R| \times |V|$ real-valued matrics, in which each element is calculated using Eq. (2). Through the loops, $ME[x][y]$ and $MX[x][y]$ store $score(x, y)$, where x is the executor and y is the request. The satisfactory score is -1 if the requester does not have access permission for the executor (lines 16 and 17). The satisfactory score is set to -1 in *ME* if r has been routed to a virtual node within the interval I_{score} (lines 11 and 12). I_{score} is the predefined time interval to refresh scores. This avoids the request being routed to the same evaluator repeatedly.

The router solves the assignment problem using *MX* and *ME* by Eq. (3a) in line 19. Lines 20–24 determine the executor and evaluator and assign them to v and e, respectively. e is the evaluator corresponding to the executor assigned by *ME* in the same host.

The routing decision agrees in all routers, as long as the current state of the virtual nodes is the same in the DNs, which is ensured by DLT protocol. Thus, the routing decision is decentralized. Although the difference in ledger commitment delay in DLT may lead to a temporarily different current state, the state is gradually consistent.

E. Mutual Evaluations and Reports

Algorithm 3 shows the pseudocode of the *evaluateAndReport* function. When the router associated with the requester receives content x from an executor, the router calculates the property scores of the properties in P for the executor by applying the score functions, and reports the property scores to the associated DN.

The router obtains the content y from evaluator e that corresponds to the same request as content x (lines 5–12). The router calculates individual property scores of the executor and evaluator by applying x and y to the score functions for all properties. Thus, the virtual nodes are mutually evaluated in the router (lines 15–17). The router sends the calculated scores to the DN as a report (lines 18 and 19). The DN updates the state of entities that have the same host (the same environment) as the target entity of the report. In Fig. 3, when the state of v_2 is updated, that of v_3 is also updated because $h(v_2) = h(v_3) = B$. The state of the entity in DN corresponds to the accumulated property score. A set of individual scores is also stored to DN in order to calculate Eq. (1).

Algorithm 2: The procedure to decide routes

```
1  // d : the DLT node
2  // r : the request
3  function decideRoutes(d, r)
4      V, R ← obtain the current states from d by DataReq
5      MX ← initialize matrics with −1
6      ME ← initialize matrics with −1
7      foreach x ∈ V do
8          foreach y ∈ R do
9              if (x can access input in r) ∧ (y.sender can access x)
                 then
10                 MX[x][y] ← score(x, y)
11                 if h(x) has been routed within I_score for y then
12                     ME[x][y] ← −1
13                 else
14                     ME[x][y] ← score(x, y)
15             else
16                 MX[x][y] ← −1
17                 ME[x][y] ← −1
18     v ← nil, e ← nil
19     AX ← Solve(MX), AE ← Solve(ME)
20     if MX[v][r] ≠ −1 then
21         v ← AX[r]
22     if ME[v][r] ≠ −1 then
23         x ← AE[r]
24         e ← the evaluator in h(x)
25     return v, e
```

V. Scoring Properties in Machine Learning Applications

The properties used in MSR and their score functions differ depending on the application. Thus, score functions need to be defined for each application. In this section, we define the score functions for the properties in a machine learning application, which is a typical use case in which MSR is particularly effective. We consider the case where request r is issued by the requester and routed to executer v, and the retrieved content is evaluated by evaluator e. Thus, $c(v, r)$ and $c(e, r)$ are applied to the score functions.

We explain *response time* and *accuracy* as the properties of the content. We denote the response time property by t and the accuracy property by a. We denote the property values of a and t in content x as $p_a(x)$ and $p_t(x)$, respectively.

A. Scoring Response Time

The response time to retrieve content is a common property value in the stream data processes. The response time is

Algorithm 3: the procedure to evaluate and report

```
1  //  n : the router
2  //  x : content to be measured
3  //  e : measuring entity (evaluator)
4  function evaluateAndReport(n, x, e)
5      r ← make a request to obtain x
6      r.sender ← n
7      if e.router ≠ n then
8          // v is associated with another router
9          t ← e.router
10     else
11         t ← e
12     y ← send request ⟨Get, e, r⟩ to t
13     X ← empty map
14     Y ← empty map
15     foreach p ∈ P do
16         X[p] ← f_p(x, y)
17         Y[p] ← f_p(y, x)
18     send⟨Report, v, r, e, X⟩
19     send⟨Report, e, r, v, Y⟩
```

measured on the router in DRINC. Response time T_{resp} corresponds to the sum of the latency values: $T_{resp} = T_{net} + T_{queue} + T_{proc}$, where T_{net} is the network latency (including propagation delay and data transfer delay), T_{queue} is the queueing delay on the virtual node caused by the access contentions, and T_{proc} is the process latency to provide the content on the executor. If the executor is a processor, the process latency includes the retrieval latency in obtaining the input. Note that we assume that the overhead to determine the routing in the router is negligible.

A request to the evaluator may be queued because of the contentions caused by the limited number of evaluators. However, such contentions are not expected to occur in the executors if a request is properly assigned. Thus we exclude the queueing delay when calculating the expected response time property of the evaluator, that is, $T_{resp} = T_{net} + T_{proc}$. Because T_{queue} and T_{proc} need to be declared by the executors, these property values can be falsified by an adversarial executor. The router can deny the property values by checking whether or not $T_{queue} + T_{proc}$ is approximately equal to $T_{resp} - T_{net}$. T_{net} can be estimated using network tools such as the ping command. The router can set the property score of an adversarial executor to -1.

We assume that the request includes the value of the maximum response time that satisfies the application requirement, which is denoted as T_{req}. If the response time does not satisfy the requirement, the property score is zero, and if not, it is greater than zero. The value can range between 0 and 1, reflecting the degree of deviation from the requirement. Thus, we define the score function for response time property t as follows:

$$f_t(c(v,r), c(e,r)) = \begin{cases} \frac{T_{req} - p_t(c(v,r))}{T_{req}} & p_t(c(v,r)) < T_{req} \\ 0 & otherwise. \end{cases}$$

Note that the evaluator e is not used in the score function, because the response time can be measured solely by the content provided by v.

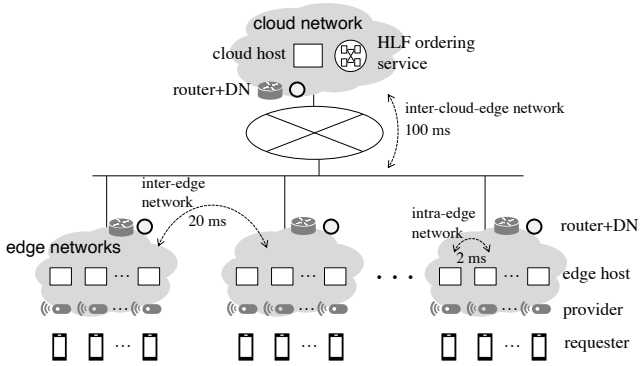


Fig. 4. The edge computing environment used in the simulation

B. Scoring Accuracy

Prediction accuracy is an important measure of quality in machine learning applications. Accuracy is the probability of a prediction being equal to the ground truth. We define the score function of the accuracy property a as follows:

$$f_a(c(v,r), c(e,r)) = p_a(c(v,r))\ p_a(c(e,r)).$$

This corresponds to the probability that both predictions in contents provided by v and evaluator e are equal to the ground truth.

The content from a processor may include multiple predictions. For example, in image object detection using machine learning, there can be multiple predicted objects in an image frame. We define the score function for the accuracy property for multiple predictions as follows:

$$f_a(c(v,r), c(e,r)) = \frac{\sum_{i \in A(c(e,r))} p_i(c(v,r)) p_i(c(e,r))}{|A(c(e,r))|}$$

where $A(x)$ denotes the set of accuracy properties corresponding to the multiple predictions in content x. The definition of f_a corresponds to the average probability that both the predictions in contents provided by v and e are equal to the ground truth for all predictions predicted in the content provided by e.

In both cases, by collecting scores from multiple evaluators, the property score obtains a higher value by Eq. (1) when the prediction in v is accurate.

VI. Evaluation by Simulations

A. Simulation setting

We implemented a discrete event simulator for DRINC. Table II lists the common simulation settings.

The simulation used a typical two-tier edge computing network, as illustrated in Fig. 4, containing edge networks and a cloud data center network. We assumed that the edge hosts were uniformly distributed in the edge networks; that is, the same number of edge hosts existed in each edge network.

In the basic setting, we set the number of edge networks to 5 and the number of edge hosts to 30.

The process latency and network latency parameters were determined according to the measurement results in a real testbed [31]. We used the measurement results on a server with GPU (NVIDIA Tesla V100) as the cloud host and Raspberry Pi 4 devices as the edge hosts. The number of virtual nodes in one host (slots per host) was 2 and 50 in the edge host and cloud host, respectively. One of the virtual nodes worked as an evaluator in each host. The average network latency between hosts in the same network, in different edge networks, and in the edge and cloud networks was set to 2 ms, 20 ms, and 100 ms, respectively. The latency varied according to a Gaussian distribution.

We used Hyperledger Fabric (HLF) as the DLT implementation. The ledger commitment latency followed the gamma distribution modeled in [32]. We used the parameters in the case where the average transactions per second was 10 tps and the block-generation timeout was 1.75 s. The average ledger commitment latency without the network latency was 1.614 s. The ordering service of HLF, which is a core HLF function for determining the order of transactions, was assumed to be located in the cloud network.

The results are the averages of 100 simulations with different random seeds. Each simulation included a 30 s warm-up phase so that the system could collect the expected response time property in the virtual nodes.

We used three baseline algorithms: *cloud, edgeward*, and *random*. The *cloud* algorithm routes all requests to the cloud server. The *edgeward* algorithm routes the request randomly to processors in the same edge network as the requester. The *random* algorithm routes requests to randomly selected virtual nodes from the edge and cloud servers. These baseline algorithms route requests to the hosts regardless of the number of virtual nodes.

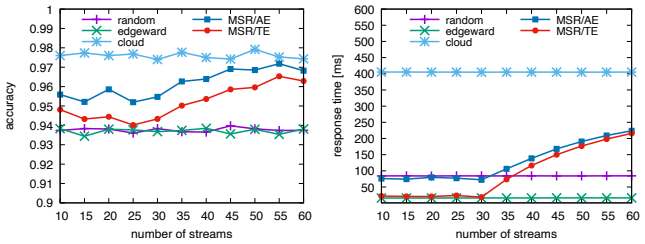
For the properties of MSR, we used $P = \{a, t\}$ where a is the accuracy and t is the response time. The requested response time (T_{req}) was set to 500 ms as a common parameter setting. The score functions defined in Section V were applied. We used two types of weight parameter settings for P. The first emphasized accuracy ($w_a = 0.9$ and $w_t = 0.1$), while the other emphasized response time ($w_a = 0.1$ and $w_t = 0.9$). In the following, we denote the former as MSR with accuracy emphasis (MSR/AE) and the latter as MSR with response time emphasis (MSR/TE).

To observe the impact of changes in the number of streams, the number of streams was varied from 10 to 60. The number of requesters and the provider were set the same as the number of streams (the number of requesters and the provider also changed from 10 to 60).

We conducted simulations using two scenarios; handwriting image classification (MNIST) and Image Object Detection (IOD).

TABLE II
SIMULATION SETTINGS

Parameters	Value
number of edge networks	5
number of edge servers	30
slots per host (edge, cloud)	2, 50
avg. network latency (intra-edge, inter-edge, inter-edge-cloud)	2, 20, 100 [ms]
network bandwidth	10 [Mbps]
avg. ledger commitment latency	1614 [ms]
stream data interval	1 [sec]
stream lifetime	100 [sec]
request response time (T_{req})	500 [ms]
I_{route}	5000 [ms]
I_{eval}	5000 [ms]
I_{score}	100 [sec]



(a) avg. accuracy (b) avg. response time

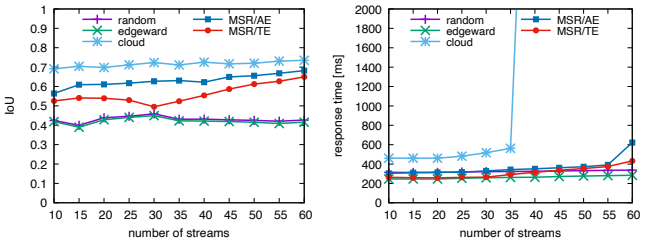
Fig. 5. MNIST scenario

B. MNIST scenario

The MNIST scenario aims to evaluate the basic performance of MSR. This scenario uses a machine learning application that classifies handwritten digit images provided by data sources using the MNIST dataset [33]. MNIST requires a small computational load, and thus, overload rarely occurs in computing resources. We applied Keras [34] as the deep learning library. We created five skewed training datasets and one non-skewed training dataset with 10,000 images picked from the MNIST dataset. The five skewed training sets included those with a Zipf distribution with top entries 0, 2, 4, 6, and 8. The model trained using these skewed datasets has better accuracy for limited digits, whereas the model trained using the non-skewed dataset has high accuracy for all digits. Each data source provided an image as a stream of data randomly picked from 10,000 test images.

The accuracy in the MNIST scenario is defined as the number of correctly classified images divided by the total number of images used in the streams. There was no significant difference in the processing time between a cloud server and an edge server because the MNIST process requires a small calculation overhead (less than 1 ms). Therefore, network latency dominated the response time.

Fig. 5 (a) and (b) respectively show the results of accuracy and response time in the MNIST scenario.



(a) avg. IoU (b) avg. response time

Fig. 6. IOD scenario

In the baseline algorithms (cloud, edgeward, and random), the accuracy results were constant regardless of the number of streams. The small fractions that occurred in the accuracy were caused by the randomness of the selection of data sources.

As expected, the cloud algorithm exhibited the highest accuracy. The accuracy in MNIST was approximately 0.98, which was the best performance in Keras using the MNIST test dataset. However, the cloud algorithm required the longest response time. The response time was approximately 400 ms because the process required a request/response for the MNIST process and data retrieval for its input.

The edgeward and random algorithms showed similar accuracies, which were much smaller than that of the cloud. The accuracy was approximately 0.94, approximately 0.04 smaller than that of the cloud because the edge processors with the skewed models were selected randomly. The response times in the edge and random algorithms were approximately 22 ms and 88 ms, respectively, which reflect the response times of intra-edge and inter-edge/cloud network latency. Although these algorithms assign without considering contentions in virtual nodes, the queueing time is small because MNIST is not a computation-bounded process.

MSR/AE achieved the best accuracy in algorithms other than the cloud algorithm regardless of the number of streams. The accuracy was approximately 0.95–0.96 when the number of streams was less than 30. This is because processors with an appropriate model in edge networks were selected. The response time of MSR/AE is longer than that of the edgeward algorithm because MSR/AE forwards requests to the processors in different edge networks from the requester to achieve better accuracy. The accuracy gradually increased to 0.97, as the number of streams increased. Simultaneously, the average response time also increased to approximately 230 ms as the number of streams increased. This is because when the number of streams reaches 30, the slots of virtual nodes in edge hosts are occupied, and the rest of the stream requests are forwarded to the cloud.

As expected, MSR/TE showed a smaller response time and lower accuracy than MSR/AE. MSR/TE exhibited slightly better accuracy than the edgeward and random methods. The response time was similar to that of the edgeward algorithm when the number of streams was less than 30. This is because MSR/TE tends to forward requests to processors in the same edge network, as with the edgeward algorithm. Both the accuracy and the response time in MSR/TE increased when the number of streams was more than 30, for the same reason as MSR/AE.

C. IOD scenario

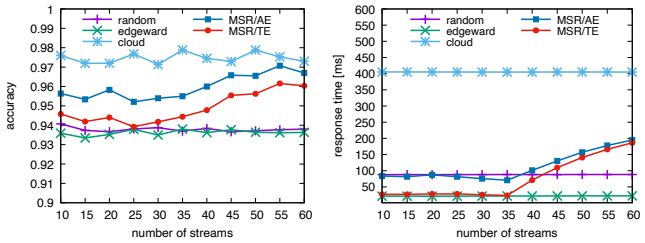
The IOD scenario is a more analytic application scenario that predicts objects in the image frames of distributed image data sources such as live streaming videos and surveillance cameras.

We used the results of YOLOv3 [35], a mature deep learning-based object detection library. We applied two types of data sources that provided images from the MS COCO dataset [36] and the PASCAL VOC dataset [37] (hereinafter, COCO and VOC). We selected 1,000 images from each dataset. In the simulation, half of the providers (data sources) provided images from COCO and the other half from VOC. Each data source provided an image as a stream randomly selected from COCO or VOC. The selected image was provided during the stream. We used the results of YOLOv3 with the COCO model, which was trained with the COCO dataset, for the processor in the cloud host. The COCO model achieves high accuracy in both the COCO and VOC datasets. We also used the results of YOLOv3 with the tiny-COCO and tiny-VOC models, which were trained with a reduced dataset, for the processors in edge networks. The tiny-COCO model and tiny-VOC model show better accuracy in the COCO dataset and VOC dataset, respectively. The process latencies in YOLOv3 in the edge host and cloud host were 234 ms and 56 ms, respectively.

As an accuracy index in the IOD scenario, we used the Intersection over Union (IoU) of detected objects in an image, which is a commonly used measure of accuracy in image analysis [38]. The IoU measure gives the similarity between the predicted and ground-truth regions for an object in the image. In the simulation, we defined the accuracy as the average IoU.

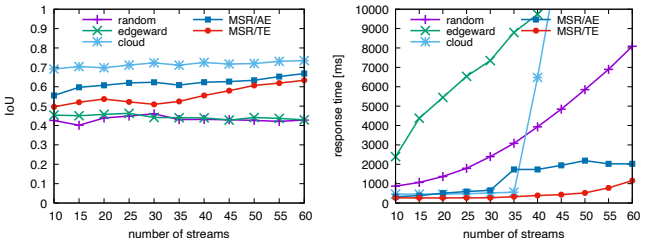
Fig. 6 (a) and (b) respectively show the results of IoU and response time in the IOD scenario.

The trend in IoU is the same as that of the accuracy in the MNIST scenario. The IoU in the cloud algorithm is approximately 0.7, which is the highest IoU among all the algorithms. Because the IOD scenario is a computation-bounded process, the process is saturated when the number of streams in the cloud algorithm is greater than 35. The response time was less than 600 ms when the number of streams was 35, but it increased to more than 10 s when the number of streams was 40. The IoUs in the random and edgeward algorithms were almost the same. In these algorithms, IoU is approximately 0.4, which is much smaller than that in the cloud algorithm. The latency in the random algorithm was slightly larger than that in the edgeward algorithm. The latency in the random algorithm was approximately 310 ms, while that in the edgeward algorithm was approximately 250 ms. This is because the random algorithm sometimes randomly selects the processor in the cloud.



(a) accuracy (b) response time

Fig. 7. MNIST scenario (heterogeneous)



(a) IoU (b) response time

Fig. 8. IOD scenario (heterogeneous)

MSR/AE and MSR/TE showed much better IoU than random and edgeward algorithms. The IoUs in MSR/AE and MSR/TE were approximately 0.6 and 0.5, respectively, when the number of streams was less than 30. The IoU increased when the number of streams exceeded 30 because the processors in the edge networks were occupied and the remaining requests were forwarded to the processor in the cloud.

As expected, the response in MSR/AE was slightly longer than that in MSR/TE. MSR/AE and MSR/TE showed similar response times to the random and edgeward algorithms, respectively. The reason is the same as in the MNIST scenario: MSR/AE forwards some requests to processors in different edge networks, and MSR/TE tends to forward requests to processors in the same edge network. The latency in MSR/AE increased when the number of streams was 60 because of the overcommitment of the CPU resources for the virtual nodes in the cloud host.

D. Impact of the heterogeneity

To see the impact of heterogeneity in the edge computing environment, we conducted simulations that included additional small devices, which have 10% of the computing power of the edge host in the previous setting. In a heterogeneous setting, we added one small device to each edge network.

The settings of the MNIST and IOD processors were the same as those of the processors in the edge hosts. The processors in the small device used the same model as those in the edge hosts. The small device had two virtual nodes, which also used the same settings as the edge host.

Fig. 7 and Fig. 8 show the results. The accuracy and IoU were similar to those in previous settings. A notable difference was the response time of each algorithm. In the MNIST scenario, the response time did not increase until the number of streams exceeded 35. This is because even the small devices could process the MNIST classification in a short period of time. The response time in the random and edgeward algorithms in the IOD scenario increased because of the long latency and saturations in the small devices. The amount of increase in the response time was more pronounced in the edgeward algorithm than in the random algorithm. This is because the random algorithm randomly selects the processor in the cloud host. The response time in MSR/AE increased when the number of streams exceeded 35 because the requests were forwarded to the processes in the small devices. On the other hand, the response time in MSR/TE did not increase because it avoided routing requests to the small devices to maintain a small response time.

Overall, we confirmed that the MSR forwards requests appropriately according to the weight parameters, even when the computing power is heterogeneous.

VII. Conclusion

In this study, we proposed a novel in-network processing framework DRINC that manages distributed computing resources and makes routing decisions in a decentralized manner. We also proposed a routing algorithm MSR on DRINC to achieve maximal satisfactory scores of the process result in the entire system. The simulation results showed that the DRINC framework with MSR can solve the trade-off between accuracy and response time in machine learning applications using mutual evaluations.

To the best of our knowledge, the DRINC with MSR is the first framework described in the literature that cooperates with DLT to make deterministic routing decisions for decentralized in-network computing. We believe that DRINC is a valuable component for future decentralized applications.

One issue that needs to be addressed is security to cope with attacks by Byzantine adversaries aiming to degrade the trustworthiness of the system. Countermeasures for such attacks will be addressed in future work. For example, excluding unnatural reports compared with the overall tendency may prevent an adversarial attack in which a colluded router and executor illegally report high property values.

Another issue that should be addressed is scalability. MSR shares the current states of all related entities as a distributed ledger. Applying efficient data sharing schemes, such as distributed hash tables (e.g., [39], [40]) and scalable broadcasts (e.g, [41]), for maintaining the current states may improve the performance in cases where a large number of computing resources are distributed over a wide area.

Our future work also includes the effective implementation of DRINC, such as an efficient discovery method for routers located physically close to the executors and requesters, lightweight data sharing by applying differential data exchanges, and utilizing in-network caches for faster response times to obtain the processed content.

REFERENCES

[1] Computing in the Network (COIN). [Online]. Available: https://trac.ietf.org/trac/irtf/wiki/coin

[2] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On Multi-access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[3] H. Tanaka, M. Yoshida, K. Mori, and N. Takahashi, "Multi-access Edge Computing: A Survey," *Journal of Information Processing*, vol. 26, pp. 87–97, 2018.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[5] S. Boumerdassi, A. Ceselli, S. Secci *et al.*, "Complexity-Performance Trade-offs in Robust Access Point Clustering for Edge Computing," in *2021 17th International Conference on the Design of Reliable Communication Networks (DRCN)*. IEEE, 2021, pp. 1–8.

[6] M. Hosseinzadeh, A. Wachal, H. Khamfroush, and D. E. Lucani, "Optimal Accuracy-Time Trade-Off for Deep Learning Services in Edge Computing Systems," in *ICC 2021-IEEE International Conference on Communications*. IEEE, 2021, pp. 1–6.

[7] H. Huang, K. Peng, and X. Xu, "Collaborative Computation Offloading for Smart Cities in Mobile Edge Computing," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 2020, pp. 176–183.

[8] A. Deshpande, K. Stewart, L. Lepetit, and S. Gunashekar, "Distributed Ledger Technologies/Blockchain: Challenges, opportunities and the prospects for standards," *Overview report The British Standards Institution (BSI)*, vol. 40, p. 40, 2017.

[9] N. El Ioini and C. Pahl, "A Review of Distributed Ledger Technologies," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2018, pp. 277–288.

[10] G. Suciu, C. Nădrag, C. Istrate, A. Vulpe, M.-C. Ditu, and O. Subea, "Comparative Analysis of Distributed Ledger Technologies," in *2018 Global Wireless Summit (GWS)*. IEEE, 2018, pp. 370–373.

[11] L. Cui, S. Yang, Z. Chen, Y. Pan, Z. Ming, and M. Xu, "A Decentralized and Trusted Edge Computing Platform for Internet of Things," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 3910–3922, 2019.

[12] R. Li and H. Asaeda, "A Blockchain-Based Data Life Cycle Protection Framework for Information-Centric Networks," *IEEE Communications Magazine*, vol. 57, no. 6, pp. 20–25, 2019.

[13] H. Yang, Y. Liang, J. Yuan, Q. Yao, A. Yu, and J. Zhang, "Distributed Blockchain-Based Trusted Multidomain Collaboration for Mobile Edge Computing in 5G and Beyond," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 11, pp. 7094–7104, 2020.

[14] I. Aliyu, M. C. Feliciano, S. Van Engelenburg, D. O. Kim, and C. G. Lim, "A Blockchain-Based Federated Forest for SDN-enabled In-Vehicle Network Intrusion Detection System," *IEEE Access*, vol. 9, pp. 102 593–102 608, 2021.

[15] Q.-V. Pham, T. Leanh, N. H. Tran, B. J. Park, and C. S. Hong, "Decentralized Computation Offloading and Resource Allocation for Mobile-Edge Computing: A Matching Game Approach," *IEEE Access*, vol. 6, pp. 75 868–75 885, 2018.

[16] X. Wang, Z. Ning, and S. Guo, "Multi-Agent Imitation Learning for Pervasive Edge Computing: A Decentralized Computation Offloading Algorithm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 411–425, 2020.

[17] Z. Chen and X. Wang, "Decentralized Computation Offloading for Multi-User Mobile Edge Computing: A Deep Reinforcement Learning Approach," *EURASIP Journal on Wireless Communications and Networking*, vol. 2020, no. 1, pp. 1–21, 2020.

[18] M. A. A. Careem and A. Dutta, "Reputation Based Routing in MANET Using Blockchain," in *2020 International Conference on Communication Systems & NETworkS (COMSNETS)*. IEEE, 2020, pp. 1–6.

[19] A. A. Abdellatif, L. Samara, A. Mohamed, A. Erbad, C. F. Chiasserini, M. Guizani, M. D. O'Connor, and J. Laughton, "Medge-chain: Leveraging Edge Computing and Blockchain for Efficient Medical Data Exchange," *IEEE Internet of Things Journal*, vol. 8, no. 21, pp. 15 762–15 775, 2021.

[20] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, 2009, pp. 1–12.

[21] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," in *ACM SIGCOMM Comput. Commun. Rev.*, 2014, p. 66–73.

[22] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *Decentralized Business Review*, p. 21260, 2008.

[23] G. Wood *et al.*, "Ethereum: A Secure Decentralised Generalised Transaction Ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[24] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[25] R. G. Brown, J. Carlyle, I. Grigg, and M. Hearn, "Corda: An Introduction," *R3 CEV, August*, vol. 1, no. 15, p. 14, 2016.

[26] S. Popov and Q. Lu, "IOTA: Feeless and Free," *IEEE Blockchain Technical Briefs*, 2019.

[27] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar *et al.*, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," in *Proceedings of the conference of the ACM special interest group on data communication*, 2017, pp. 183–196.

[28] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica, "Towards a Common API for Structured Peer-to-Peer Overlays," in *International Workshop on Peer-To-Peer Systems*. Springer, 2003, pp. 33–44.

[29] I. Baumgart and S. Mies, "S/Kademlia: A Practicable Approach Towards Secure Key-Based Routing," in *2007 International Conference on Parallel and Distributed Systems*. IEEE, 2007, pp. 1–8.

[30] H. W. Kuhn, "The Hungarian Method for the Assignment Problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[31] (deducted for the double-blind policy).

[32] S. Lee, M. Kim, J. Lee, R.-H. Hsu, M.-S. Kim, and T. Q. S. Quek, "Facing to Latency of Hyperledger Fabric for Blockchain-enabled IoT: Modeling and Analysis," 2021. [Online]. Available: https://arxiv.org/abs/2102.09166

[33] The MNIST Database of Handwritten Digits. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[34] N. Ketkar, "Introduction to Keras," in *Deep learning with Python*. Springer, 2017, pp. 97–111.

[35] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[36] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[37] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes (VOC) Challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[38] P. Jaccard, "Étude comparative de la distribution florale dans une portion des Alpes et des Jura," *Bull Soc Vaudoise Sci Nat*, vol. 37, pp. 547–579, 1901.

[39] R. Norvill, B. B. Fiz Pontiveros, R. State, and A. Cullen, "IPFS for Reduction of Chain Size in Ethereum," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 1121–1128.

[40] Y. Hassanzadeh-Nazarabadi, A. Küpçü, and Ö. Özkasap, "LightChain: Scalable DHT-based Blockchain," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 10, pp. 2582–2593, 2021.

[41] E. Rohrer and F. Tschorsch, "KadCast: A Structured Approach to Broadcast in Blockchain Networks," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 199–213.