Hindawi
Mobile Information Systems
Volume 2020, Article ID 8820507, 17 pages
https://doi.org/10.1155/2020/8820507

*Research Article*

# Falcon: A Blockchain-Based Edge Service Migration Framework in MEC

**Xiangjun Zhang**,[1,2,3] **Weiguo Wu**,[1,2,3] **Shiyuan Yang**,[1] **and Xiong Wang**[1]

[1]*School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China*
[2]*National High Performance Computing Center (Xi'an), Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China*
[3]*New Computer Research Institute, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China*

Correspondence should be addressed to Weiguo Wu; wgwu@mail.xjtu.edu.cn

Received 29 June 2020; Revised 2 September 2020; Accepted 24 September 2020; Published 16 October 2020

Academic Editor: Prosanta Gope

Driven by advanced 5G cellular systems, mobile edge computing (MEC) has emerged as a promising technology that can meet the energy efficiency and latency requirements of IoT applications. Edge service migration in the MEC environment plays an important role in ensuring user service quality and enhancing terminal computing capabilities. Application services on the edge side should be migrated from different edge servers to edge nodes closer to users, so that services follow users and ensure high-quality services. In addition, during the migration process, edge services face security challenges in an edge network environment without centralized management. To tackle this challenge, this paper innovatively proposes a blockchain-based security edge service migration framework, Falcon, which uses mobile agents different from VM and container as edge service carriers, making migration more flexible. Furthermore, we considered the dependencies between agents and designed a service migration algorithm to maximize the migration benefits and obtain better service quality. In order to ensure the migration of edge services in a safe and reliable environment, Falcon maintains an immutable alliance chain among multiple edge clouds. Finally, the experimental results show that "Falcon" has lower energy consumption and higher service quality.

## 1. Introduction

With the rapid development of mobile Internet and Internet of Things technologies, various new services have been continuously emerging, making mobile communication traffic experience explosive growth in the past few years. Mobile applications using cloud computing technology are becoming more and more popular, such as VR, real-time video, and 3D games [1]. This kind of emerging application brings great convenience to people's lives, but with the gradual complexity and diversification of services, higher requirements for network bandwidth and delay have been put forward [2]. Moreover, according to IDC forecasts, by 2020, the total amount of global data will be greater than 40 zettabytes (ZB), and 45% of the data generated by the Internet of Things will be processed at the edge of the network [3, 4]. The concept of mobile edge computing (MEC) has recently become a promising technology to solve these challenges. It pushes mobile computing, network control, and storage to the edge of the network (such as base stations and access points) to facilitate mobile applications with limited resources. Implement compute-intensive and delay-sensitive applications on the device. Its core idea is to bring computing closer to users, distribute small servers or data centers that carry cloud applications across the entire network, and directly connect to entities at the edge of the network (such as micro data centers and cloudlet). Due to its proximity to the terminal equipment, MEC not only reduces the processing pressure of the cloud center but also saves the cost of high end-to-cloud bandwidth and reduces the network response delay of end-to-end edge nodes, so it can provide a higher quality of service than traditional cloud platforms [5]. MEC is considered to be an important part of the future 5G mobile network architecture. Its main purpose is to achieve efficient and seamless integration of cloud computing functions with mobile networks, and for all stakeholders (mobile operators, service providers, and mobile users), provide convenience [6]. MEC is expected to significantly reduce latency and mobile energy consumption and solve the key challenges of realizing the 5G vision.

However, one of the key challenges facing MEC is how to dynamically migrate services. Due to the limited range of a single edge cloud and the frequent movement of terminal devices (such as smart cars and personal mobile terminals) to different geographical areas, the quality of edge cloud services has declined sharply, and even service is interrupted which makes it difficult to guarantee service continuity [7]. As shown in Figure 1, the terminal user requests the services provided by edge cloud A at position 1 (such as augmented reality and Internet of Vehicles applications). When the terminal user moves to position 2, because position 2 is far away from the service radius that edge cloud A can provide, the service quality drops sharply, and even the service is interrupted. In order to ensure service quality, services need to be migrated to an edge cloud near the current user location (such as edge cloud D or edge cloud E). On the one hand, when the terminal user moves, ideally the services on the edge server should also be migrated to a new nearby server in real time. Therefore, effective dynamic service migration is essential for the normal provision of edge services in edge computing environments [8]. On the other hand, edge services are located in different edge clouds, and edge servers may belong to different participants, such as telecommunications operators, Internet companies, and home users. Therefore, there is no centralized management for different heterogeneous hardware [9, 10]. The dynamic, open, and collaborative nature of the MEC network environment makes network security issues more and more complex. Therefore, the issue of trust in service migration is also very important.

To solve this problem, Chen et al. in [11] proposed an edge cognitive computing (ECC) paradigm that combines edge computing and cognitive computing. They have established a dynamic migration platform for edge services based on mobile user behavior cognition, which can predict user behavior and better guide service migration based on traffic data and network resource environment. Literature [12] is dedicated to determining the migration strategy and communication strategy through the reinforcement learning model, so as to solve the problem of edge service migration. However, most of the abovementioned research on edge computing uses virtual machines (VMs) as the carrier of migration. The overlay of VMs can be tens or hundreds of megabytes. For delay-sensitive applications, the total migration time is relatively long [13]. Furthermore, the VM overlay is not easy to maintain which limits the application in industry and academia. In addition, these studies did not consider the migration trust of edge services in different edge clouds. Edge computing environments cause security risks when data are sent to untrusted edge servers. The complexity of the mechanism and the huge amount of calculation make it difficult to overcome.

In this paper, we innovatively propose an edge service migration framework based on the blockchain named Falcon. Worth noting, Falcon uses a more flexible and portable mobile agent as the carrier of edge services and uses blockchain technology to jointly maintain a trusted alliance chain to solve the trust problem of service migration in multiple edge clouds. Secondly, seek the migration strategy with the greatest benefit through the migration decision algorithm and guarantee higher user service quality. Finally, we deployed this framework in a computing cluster to evaluate its performance. The evaluation results show that the framework ensures the service quality of end users while ensuring the security of service migration. The contribution of this work is summarized as follows:

(i) We propose a mobile agent-based edge service migration framework, Falcon, and design a migration decision algorithm to seek the migration path with the least migration cost. This framework makes the service migration more flexible and portable, with low migration cost, and ensures better service quality.

(ii) In multiple heterogeneous edge environments lacking a trusted third party, we built a consortium chain network of multiple edge nodes and improved the PBFT consensus algorithm of Ethereum. There is no need to perform three stages in the checkpoint phase. The submission process significantly reduces the amount of network communication. This ensures the safety and performance of Falcon during the service migration process.

(iii) We implemented a blockchain-based mobile agent dynamic service migration prototype platform and evaluated it through experiments. The results show that the proposed migration framework course meets the needs of user edge service migration, and the improved blockchain network throughput is excellent. Compared with the traditional PBFT algorithm, the performance of Falcon is encouraging.

The rest of this paper is organized as follows. Section 2 summarizes the related work. In Section 3, we describe in detail the mobile agent-based edge service migration architecture. Section 4 shows the mechanism of using blockchain technology to solve the trust problem of edge service migration. Section 5 makes a formal evaluation of the performance of the framework. The experimental results and performance evaluation of the agent-based dynamic service migration platform are discussed in Section 6. Finally, Section 7 concludes this paper.

## 2. Related Work

Edge service migration is an important mechanism to ensure user experience, and it has received extensive attention from industry and academia in recent years. It has been widely used to study how to overcome network latency and reduce service interruption time in edge networks. There are already some literature studies [14, 15] that have analyzed and summarized the service migration methods.

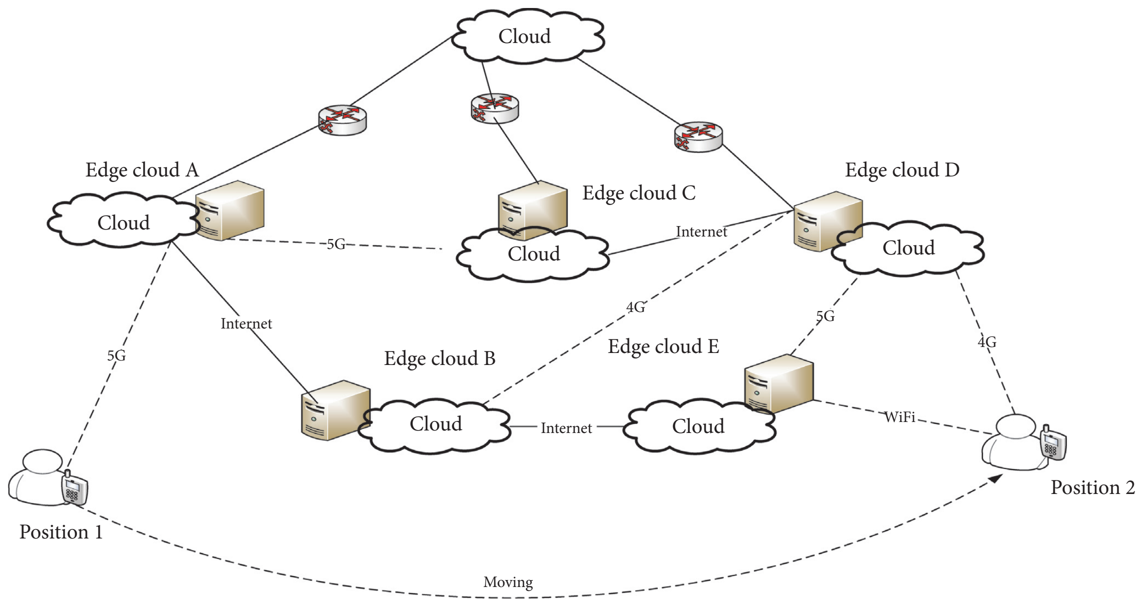


FIGURE 1: Service migration scenario in MEC.

Earlier research on edge services migration focused on the migration of virtual machines (VMs) which improve QoS by optimizing the migration time to reduce service downtime. Lu et al. [16] combined remote loading and redirection to speed up service migration. By tracking historical access patterns, a load request list is generated to locate the service application for booting, and then core code is automatically prefetched and cached. It enables running VMs to switch data access to merged image files, reducing edge service migration time. Sun et al. [17] proposed an improved serial migration strategy and introduced a post-copy migration scheme into serial migration. The queuing model is established to quantify the performance indicators, and mathematical analysis is used to evaluate the performance of the model. It improves the hybrid migration strategy of serial migration strategy and parallel migration strategy and improves resource utilization and QoS. The method proposed in [18] introduced low-latency access to data or edge services by introducing a distributed cloud layer composed of cloudlets ("small clouds" with low computing power) between users and the cloud. This method uses VM to realize the common architecture for service migration. Nevertheless, the size of the virtual machine overlay can be tens of megabytes or hundreds of megabytes depending on the application, which is relatively long for latency-sensitive applications and it is unacceptable.

Some researchers use docker container migration for service migration. For example, Ma et al. [19] conducted an in-depth study on the characteristics of docker container's hierarchical storage and proposed a migration method that uses a hierarchical storage system to reduce data synchronization overhead. But based on the checkpoint CRIU technology [20], all data in the storage layer and the entire file system are packaged and transmitted, which may worsen the condition of the edge network, especially in the case of a bad network environment, and even cause service interruption. In addition, considering the security of edge services, Tian et al. [21] proposed an authentication framework to solve the security and privacy protection in Internet of Drones (IoD), by adopting a lightweight online/offline signature design, and designed a MEC-based predictive authentication method to minimize potential authentication costs and guarantee privacy protection of the drone's identity, location, and flying routes. Gope and Sikdar [22] considered the physical security of UAVs and proposed an effective privacy-aware authentication key agreement scheme for UAV interconnection. This solution does not need to store any key in the device to provide the required security features. These studies have solved the privacy and security of IoD in MEC. However, edge service migration is unique. Since the edge node that provides the service may be an unauthorized malicious node, there is a risk that the service is migrated to an untrusted server or user data are intercepted. The current research on the security of service migration is mostly based on information hiding and encryption technology, but as the business becomes more and more complex, the problem of the throughput of these traditional methods gradually appears.

Agent-based edge service migration is a relatively new field and needs to be studied systematically. Compared with VMs and containers, mobile agents (MA), as a program that replaces other programs to perform certain tasks, can choose when and where to move autonomously from one host to another in a complex network system [23]. When a move is triggered, the mobile agent can suspend its running process as required, then move to another place on the network to restart or continue its execution, and finally, return the result and message. Aglets mobile agents can migrate and communicate autonomously in any Java environment, so they occupy less platform resources, while VMs and containers tend to save most resources in migration services [24]. In the network, mobile agent is very suitable as the carrier of edge service migration because of its characteristics of moving between different hosts. Moreover, because agent-based service migration not only provides program runtime environment but also uses autonomous agent-based application partitioning, which reduces the management burden of edge servers and mobile terminals [25]. On the contrary, in the process of service migration based on VMs and containers, these management burdens rely heavily on the support of the underlying virtualization technology, and the agent greatly reduces this management burden. Finally, edge servers have limited bandwidth, unstable network connections, storage, and processing capabilities, and running mobile agents that take up few resources on them will be more conducive to migration services [26].

## 3. Mobile Agent System Architecture

In this section, we first discuss the internal details of mobile agents. And take IBM Aglets as an example to analyze the advantages of mobile agent as an edge service migration carrier [27]. Second, we introduce the system's service collaboration mechanism.

*3.1. Migration between Mobile Agents.* In a dynamic heterogeneous MEC environment (such as VMs, containers, or even physical machines), using mobile agents for service migration can achieve better performance and better adaptability to changing network conditions. For example, agents can migrate between VMs, containers, and physical machines as long as the Java runtime environment is configured [28, 29]. This software execution environment has software and hardware support for agent operations. These resources can be a network computer, a grid node, or a cloud infrastructure in the edge network. Each agent is executed in a self-contained virtualization container. In Figure 2, the interaction process between mobile agents is shown. Each mobile agent system consists of two parts: mobile agent (MA) and mobile agent environment (MAE). MA is a software entity running in MAE, which can be migrated between different MAEs and interact with local services or resources to complete tasks. In our framework Falcon, MA is composed of user agent (UA) and service agent (SA). The IBM Aglets agent provides different event handling functions at different points in its life cycle. A running aglets can call the clone method to clone itself, or call the deactive method to store it on a binary storage medium [30]. Similarly, when the agent is successfully created and the onDispatching method is called, the aglets agent will be dispatched to another host to continue running according to the different parameters passed [31]. When moving, the system serializes the information carried by the agent into a standard format. Once the agent arrives at another host, it will call the onArrival method to rerun at the destination.

*3.2. Agent-Based Edge Service Migration Mechanism.* In the Falcon framework, edge services are provided by a group of cooperative SAs. The mobile agent runs on one platform, but can be moved to another platform. As shown in Figure 3, the agent-based service migration diagram, in addition to the service agent, there are four key service components: event decision server (EDS), resource management server (RMS), data management server (DMS), and trust management server (TMS).

Function of EDS is to collect system-wide events and functional notifications and pass them to the mobile agent to decide whether the agent needs to be migrated. The agent can register for network monitoring and security events, such as calculating the network condition by analyzing the network latency RTT and packet loss rate PacketLoss parameters of the current edge server. The agent can also prevent damage to the platform on which the agent is currently running through intrusion detection of the security event system. In addition, it can register to receive new platforms or announcement services for new platforms, thereby having better QoS. In these cases, EDS will decide to migrate to a new platform to avoid further losses if the current platform has been damaged. We will discuss the decision migration mechanism in detail in Section 3.3. EDS is essentially the trigger source for agent migration, enabling agents to better respond to changes in the edge network environment and provide users with better QoS.

Data manager server (DMS) and authorization management server (trust manager server, TMS) can, respectively, persistently store data (including service status data and user data) and secure mutual trust data between mobile agents migrated to destination hosts. Mobile agent saves the necessary service status and data in the database before migration and restores the status and progress of the job after migration to a trusted destination host. The migration destination edge node will maintain an alliance chain among multiple edge clouds to establish mutual trust, so as to ensure the secure migration of edge services among multiple edge cloud nodes.

Resource management server (RMS) is responsible for providing the required resources to agents and platforms. On the one hand, each agent functionally requires a set of resources (such as software modules, data storage, user interface, and various other functions), and the agent also has minimum requirements for each resource. On the other hand, the platform running the agent needs to have the ability to provide a certain number of resources, thereby providing a set of resources to support the function of the agent operation. Each platform can publish resources based on the QoS provided by the RMS, and an agent queries the resources and determines whether a platform can support the necessary resource conditions necessary for its operation. If the current platform cannot provide the agent with the appropriate resources, the agent must query the RMS to determine a viable platform to which it can migrate.

*3.3. Service Migration Decision Algorithm Based on Mobile Agent.* In this section, we will discuss in detail the migration decision algorithm based on mobile agents. As we discussed earlier, EDS issues a mobile agent migration notice, which involves factors such as the overload of the edge server

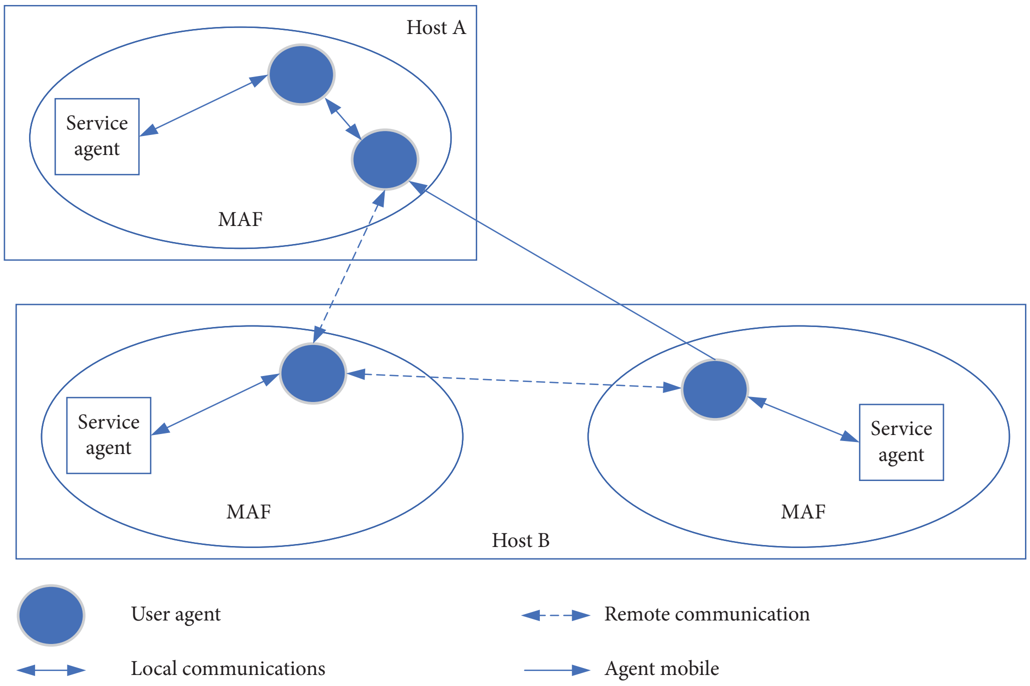


Figure 2: Agent interaction diagram.

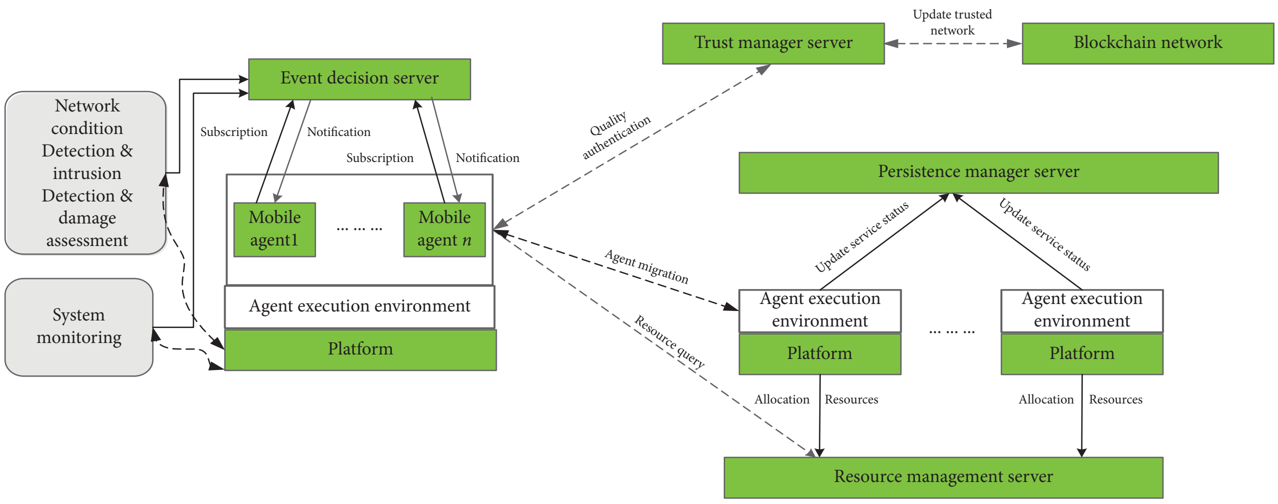


Figure 3: Framework diagram of agent-based service migration.

currently running the agent or the deterioration of the service quality due to the movement of the end user's location [32]. The reasons for the migration also include the following: the current platform is severely damaged or the required services cannot be provided. EDS must decide which possible platforms to migrate to. In general, the target node to which the task chooses to migrate mainly considers the following factors: network carrying capacity, processing capacity, memory space, migration experience, and subsequent node information of the current node information. But these factors are complicated, and there is a certain relationship between each factor.

In our system model, an information collection module is deployed on each edge server node to collect information on network dynamic load ($\alpha$), CPU computing power ($\beta$), memory space ($\lambda$), and migration experience ($\gamma$). The edge nodes and links between the nodes form an undirected connection graph $G\langle V, E\rangle$, where $V$ represents each edge node and $E$ is all the set of weighted links between the nodes, and each link has a migration right $Q$. Therefore, combining the above factors, the migration weight $Q$ is expressed by the following equation:

$$Q = \begin{cases} \alpha * s1 + \beta * s2 + \lambda * s3 + \gamma * s4, \\ s1 + s2 + s3 + s4 = 1\,(0 \leq s1, s2, s3, s4 \leq 1). \end{cases} \quad (1)$$

$Q$ is the migration weight. The larger the value, the more likely the task of the current node is to move to the

```
Input: (1) $S$, Migration alternative plan (2) $R$, constraint rule $R$ for agent group operation (including dependency, mutual exclusion, and atomic relations);
Output: Migration_flag, An indication of whether the migration was successful
(1)  HashMap < String, List < String ≫ decision BooleanMigration_flag = false
(2)  if $S$.length $\le 0$ then
(3)    return false;
(4)  end if
(5)  for each $S_i$ in decisionTmp do
(6)    Step 1. Perform a protocol check on alternative migration decisions and define a temporary migration plan WP from each platform of the migration decision $S_i$. Initialize WP $= \phi$
(7)    Step 2. Agent $A_i$ $(1 \le i \le m)$ sends a migration request to each destination host in the platform set of migration decision $S_i$, do
(8)      Step 2.1. Select a platform $P_i' \in S_i$ and add $P_i'$ to WP
(9)      WP = WP $\cup \{P_i'\}$
(10)   Step 3.
(11)   for each Constraint rules $r \in R$ do
(12)     Step 3.1. If $r$ is a dependency rule, an agent Ag depends on at least one agent in the $A'$ set, do
(13)       Step 3.1.1. If $\forall \mathrm{Ag}' \in A'$, Ag and Ag$'$ the corresponding platforms are different in WP, do
(14)         Reset WP $= \phi$;
(15) ;         return to step 2 and decide on the next temporary migration plan
(16)     Step 3.2. If $r$ is the atomic rule, all agents must be migrated to the same platform, do
(17)       Step 3.2.1. If $\forall P, P' \in \mathrm{WP} \wedge P \ne P'$, do
(18)         Reset WP $= \phi$;
(19) ;         return to step 2 and decide on the next temporary migration plan
(20)     Step 3.3. If $r$ is an exclusivity rule, that is, two or more agents cannot migrate to the same platform, do
(21)       Step 3.3.1. If $\forall P, P' \in \mathrm{WP} \wedge P = P'$, do
(22)         Reset WP $= \phi$;
(23) ;         Return to step 2 and decide on the next temporary migration plan
(24)   Step 4. If the temporary migration plan WP meets all of the constraints rule $R$, do
(25)     Step 4.1. If $\forall P' \in WP \wedge P' \in S_i$, do
(26)       Step 4.1.1. Add platform $P_i$ to agent $A_i$'s migration destination host
(27)         dispatch(Node$_i$.Ip);
(28) ;        Migration_flag = true
(29)   Step 5. Else, do
(30)     Step 5.1. If there were more resources, do
(31)
(32)   if SumStep < Step$_T$ then
(33)     sumStep++;
(34)     Return to step 1, update the weights between nodes, and recalculate the migration factor;
(35)   else
(36)       Migration_flag = false;
(37)     end if
(38)  end for
(39)    Return Migration_flag
(40) end for
```

Algorithm 1: Constraint protocol checking algorithm.

neighboring node. Among them, $s1$, $s2$, $s3$, and $s4$ represent the weight coefficients corresponding to the four indicators $(\alpha, \beta, \lambda, \gamma)$. Due to the complex edge network environment, unstable network bandwidth and edge node failures frequently occur. In order to improve the adaptability of the migration process in the mobile cloud computing environment, we take the task weights and the network conditions of the nodes into consideration and calculate more general migration weight parameters. $\mathrm{distNode}_{i,j}$ represents the distance between nodes, it is closely related to the network status value $\mathrm{netStat}_{i,j}$, and $\mathrm{netStat}_{i,j}$ is the communication status value between $\mathrm{Phys}_i$ and $\mathrm{Phys}_j$. We use the round-trip time (RTT) and packet loss rate (PacketLoss) of data packets to reflect the size of this value, which affects the reliability of data transmission. The node distance is expressed by the following formula:

$$\mathrm{DistNode}_{i,j} = \frac{\mathrm{band}_{\mathrm{aver}}}{\mathrm{band}_i + \mathrm{band}_j} * \mathrm{netStat}_{i,j}, \quad (2)$$

where $\mathrm{band}_{\mathrm{aver}}$ is the average bandwidth of the physical node. We define the migration weight factor as shown in the following equation:

$$\mu_{i,j} = \begin{cases} \dfrac{\sum_{k=1}^{m} w_k + w}{\mathrm{DistNode}_{i,j}}, & w \geq W_T, \\ \\ \dfrac{\sum_{k=1}^{m} w_k - w}{\mathrm{DistNode}_{i,j}}, & w < W_T. \end{cases} \tag{3}$$

In formula (3), $\mu_{i,j}$ represents the weight update factor of the global task node between $\mathrm{node}_i$ and $\mathrm{node}_j$ and $\mathrm{sum}_{k=1}^{m} w_k$ represents the sum of the accumulated task weights from the initial node to the current node through the migration path. $w$ represents the task weight of the current node $\mathrm{node}_i$, and $\mathrm{DistNode}_{i,j}$ represents the cloud node distance between $\mathrm{node}_i$ and $\mathrm{node}_j$. The formula of redefinition of migration weight after improvement is as follows:

$$Q = \begin{cases} (\alpha * s1 + \beta * s2 + \lambda * s3 + \gamma * s4) * \mu, \\ s1 + s2 + s3 + s4 = 1\,(0 \leq s1, s2, s3, s4 \leq 1). \end{cases} \tag{4}$$

When multiple mobile agents perform a certain task collaboratively, there are often cooperation or functional dependencies. For example, one agent may depend on other agents in function. Due to the sharing of public information resources, collaborative processing of information, etc., some agents may need to run on the same execution environment. In other cases, due to the separation of duties and functions and exclusion, some agents may have mutually exclusive relationships. Therefore, they cannot migrate to the same information host. Taking these factors into consideration, we designed a constraint rule checking algorithm (Algorithm 1) to make the migration of a group of agents under different constraint rules, so as to determine the migration of each mobile agent to the corresponding edge node. We set a set of constraint rules $R$ for the groups of agents that make up the edge service. Three types of constraint rules are specified in $R$:

(1) Atomic rules: a group of agents must migrate to the same platform.

(2) Dependency rule: an agent is functionally dependent on at least one of the other agents. Therefore, it must be migrated to the platform where the agent it depends on.

(3) Exclusion rule: two or more agents cannot be migrated to the same platform.

In the Falcon migration system, we deploy a perception module at each node, calculate and update the collected information to the node task weight, then calculate the distance between the cloud nodes, and update the weight of the link edge in $G$ through formula (4). If the current edge node has a directly connected edge node, namely, there is a directly connected edge in $G$, the destination host with the largest weight of the edge is added to the temporary migration plan. If there is no direct edge, use the Dijkstra shortest path algorithm in the undirected graph to obtain the migration path from the node to the destination node and put it into the temporary migration plan. In Figure 4, the edge node 1 has received the migration notification and migrated to the edge node 4. However, there is currently no directly connected edge between the two nodes and no consensus has been established. At this moment, we use the shortest path algorithm to find the path with the greatest benefit (the path $\mathrm{node1} \longrightarrow \mathrm{node5} \longrightarrow \mathrm{node4}$ is the shortest path). The nodes on this path are all safe nodes that have been verified by mutual trust. If the migration weight of the node exceeds the migration threshold $Q_T$, the node will move to the adjacent maximum weight node. If the migration weight of a node does not exceed the migration threshold $Q_T$, the leading node of the migrating node is returned, and the currently visited node is identified, then the leading node is a valid node. Calculate the migration weights of all unvisited nodes in $G$, so continue to iterate until it migrates to the final destination address. Among the multiple optional destination hosts in the migration, according to the three types of constraint rules between a group of agents, the migration plan that does not meet the operating rules is removed from the temporary migration plan. In order to prevent the node from returning to the front node multiple times, a jump threshold $\mathrm{Step}_T$ is set in the algorithm. If the value returned by the same node exceeds the jump threshold, the migration task ends. The service migration decision algorithm is shown in Algorithm 2.

## 4. Blockchain-Based Edge Service Migration

In this section, we focus on the features of Falcon's blockchain-based security service migration. First, Section 4.1 introduces the security issues of mobile agents and edge services and describes the huge potential of blockchain in solving the security issues of edge services. In Section 4.2, we present blockchain technology and compare the characteristics of different types of blockchains. Next, we introduced the problem description of blockchain technology to solve service migration in Section 4.3 and made a detailed description of the system architecture. In this chapter, we show how to use the blockchain network to ensure that edge service migration is complete in the environment of mutual trust and security and introduce the role of each layer. Finally, in Section 4.4, we described the specific details of the blockchain after improving the consensus algorithm.

*4.1. Blockchain-Based Edge Services Security Migration Practice.* In the case of mobile agent security, Falcon should basically support confidentiality, integrity, identity verification, access control, and nonrepudiation to protect the mobile agent from the aforementioned illegal attacks. Mobile agents are intelligent, autonomous, and adaptive. Especially, the mobile agent has mobility, and its mobility is the most significant feature of the mobile agent paradigm [33, 34]. Therefore, the secure migration of mobile agents is considered one of the most important security issues. If an anonymous mobile agent is warned illegally during the migration process, it will cause many serious problems. Therefore, in the mobile agent environment, the integrity of the mobile agent should be guaranteed

```
Input: $G, s_1, s_2, s_3, s_4$, Request$_i$, Current system iTH $(0 < i < m)$ migration request, $m$ represents the number of mobile agents that make up the current service.
Output: $S_j$, Migration decision for each agent $A_j$ to migrate to a platform $P_j$ $(1 \le j \le n)$, where $n$ represents the number of platforms available.
(1)  $A$ = ReadBlockchainData($i, G$), Read the block chain data to obtain the connection relation matrix $A$ between nodes;
(2)  ; $\lambda \leftarrow$ Request.experience, Boolean checkStata = false;
(3)  for $j = 1$; $j \le n$; $j$++ do
(4)      Updating task weight $W_i$;
(5)      netStat$_{i,j}$ = getNetworkCondation(RTT, PacketLoss, $j$)
     Get the network condition of node $j$;
(6)      DistNode$_{i,j}$ = (band$_{aver}$/ (band$_i$ + band$_j$)) $*$ netStat$_{i,j}$;
(7)      if $w \ge W_T$ then
(8)          $\mu_{i,j} = ((\sum_{i=1}^{m} W_k + w)/\text{DistNode}_{i,j})$;
(9)      else
(10)         $\mu_{i,j} = ((\sum_{i=1}^{m} W_k - w)/\text{DistNode}_{i,j})$;
(11)     end if
(12)     $\aleph$ = getNetworkCondation(RTT, PacketLoss);
(13)     $\beta$ = getCpuLoadRatio();
(14)     $\gamma$ = getMemoryLoadRatio();
(15)     $Q_{i,j} = (\alpha * s_1 + \beta * s_2 + \lambda * s_3 + \gamma * s_4) * \mu_{i,j}$;
(16)     $G = A * Q_{i,j}$;
(17) end for
(18) While $j = 1$; $j \le G$.length; $j$++ do
(19)     for $j = 1$; $j \le G$.length; $j$++ do
(20)         if $Q_{i,j} > Q_{i,j+1} > Q_T > \cdots > Q_{i,j+k}$ then
(21)             if $G[i][j]! = 0 \wedge G[i][j] < Q_T$ then
(22)                 $S_i$.add ($A_i$, hostname ($j$));
(23)             else
(24)                 Path = Dijkstra($G, i, j$);
(25)                 $S_i$.add($A_i$, Path);
(26)             end if
(27)         end if
(28)     end for
(29)     heckStata = CheckAcl(si);
(30)     if checkStata is true then
(31)         Retun 1;
(32)     else
(33)         return 0;
(34)     end if
(35) end While
```

Algorithm 2: Service migration decision algorithm.

first. In addition to the security of mobile agent migration, the security and credibility between the hosts of the edge service migration are also critical. Edge services are located in different edge clouds. Edge servers may belong to different participants, such as telecom operators, Internet companies, and home users. Therefore, there is no centralized management for different heterogeneous hardware, and it is difficult to solve the trust problem in service migration. As a result, data are sent to untrusted edge servers, which pose a security risk.

Blockchain technology is very suitable to solve the trust problem of service migration in the MEC environment due to its inherent distribution and immutability. The blockchain is an ever-growing list of records, called blocks, formed by linking these blocks one after another protected by cryptography. It is inherently resistant to data modification [35, 36]. Once the interactive information is recorded, the data in any given block cannot be retrospectively modified without changing all subsequent blocks and the network majority colluding. Therefore, the blockchain can be used as an open and distributed operating system to achieve decentralized consensus. In the Falcon framework, it can effectively record the interactions (for example, transactions) between two individuals or agents in a verifiable and permanent way.

*4.2. Blockchain.* Blockchain is essentially a decentralized database. As the underlying technology of Bitcoin, it is a series of data blocks associated with cryptographic methods [37, 38]. Each data block contains a batch of Bitcoin network transaction information to verify the validity of the information (anticounterfeiting) and generate the next block [39–41]. Blockchain technology is also a new application

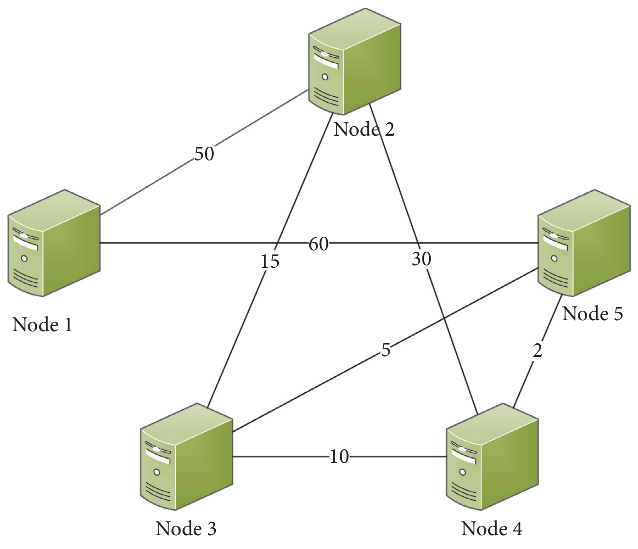


Figure 4: Task migration diagram.

model of computer technologies such as distributed data storage, point-to-point transmission, consensus mechanisms, and encryption algorithms [42].

*4.3. System Overview.* We improved the blockchain consensus algorithm and introduced it into Falcon as a security system verification mechanism. Multiple edge clouds jointly maintain a consortium chain network and develop a contract algorithm. When an unfamiliar edge network joins the blockchain network, the verification algorithm will be executed. As shown in Figure 5, the system framework of the entire trusted network is divided into three layers, namely, the audit layer, the edge control layer, and the user layer.

*4.3.1. Public Audit Service Layer.* The first layer is the public audit service layer. In this layer, all edge clouds in different geographic locations jointly maintain a tamper-proof database to store and share the interactive information of different edge clouds and mobile user information. Each edge cloud has equal rights in data management and user management. In the Falcon framework, we adopted a consortium blockchain to establish a data sharing scheme. The distributed database records public verification information of different organizations, and each member can verify the integrity of the data without requiring a trusted third party. Therefore, the whole system contains three entities: agents, users, and different edge cloud providers.

(1) As a separate organization, each edge cloud provider provides data sharing services for edge nodes. These nodes can store and manage the interactive data between edge nodes. In our system, these institutions can be edge servers, routers, switches, integrated access devices (IADs), and other devices close to the terminal. In addition, each institution is considered to be a blockchain member in alliance blockchain network. In our framework, each institution is a semitrusted party. Nodes in an organization are trusted, in other words, all nodes in an edge network trust each other, but devices from other organizations are not trusted. This characteristic of the alliance chain is well in line with the situation where there is no centralized management of different heterogeneous hardware in the edge computing environment.

(2) The mobile user is the owner of the data. They belong to different agencies and want to share data through these agencies.

(3) Nodes are controlled by blockchain members. They run several different algorithms to maintain a common ledger that records all verification information.

*4.3.2. Edge Controller Layer.* The second layer is the edge control layer. This layer is composed of multiple edge clouds and edge nodes, providing end users with diversified edge services (such as real-time data processing in public safety, intelligent networked vehicles and autonomous driving, VR, industrial Internet of things, smart home, and other applications) and related data storage and management services. As shown in Figure 5, five edge clouds Org1, Org2, Org3, Org4, and Org5 distributed in different geographical locations have established mutual trust connections with each other, and each edge cloud provides different edge services for end users. A more important function is that all members jointly maintain a blockchain database that provides public audit services for end users at the user level. Finally, they manage the migration scheduling on all edge servers and clusters on the WAN. The edge service scheduler at this layer is responsible for scheduling offloading services across edge servers or clusters. The scheduling parameters include but are not limited to the following 4 items: (1) the physical location of the edge server; (2) performance data collection of edge servers and service migration decisions; (3) bandwidth and latency perceived by the terminal user; and (4) the authentication service is used to verify the identity of the edge server and the terminal user.

*4.3.3. User Layer.* The third layer is the end user. End users are Linux mobile devices running mobile applications on Android, iOS, Windows, or other platforms. Users register at this layer and upload the edge services and data they share and use. The migration process of edge services is transparent to end users, and mobile devices can use WiFi or LTE to access edge nodes or edge controllers.

*4.3.4. Security Attributes.* The Falcon framework meets the following security features:

(1) Confidentiality of data: shared data are not visible to blockchain nodes. In our solution, the blockchain node maintains a blockchain database that records shared data verification information. Nodes should

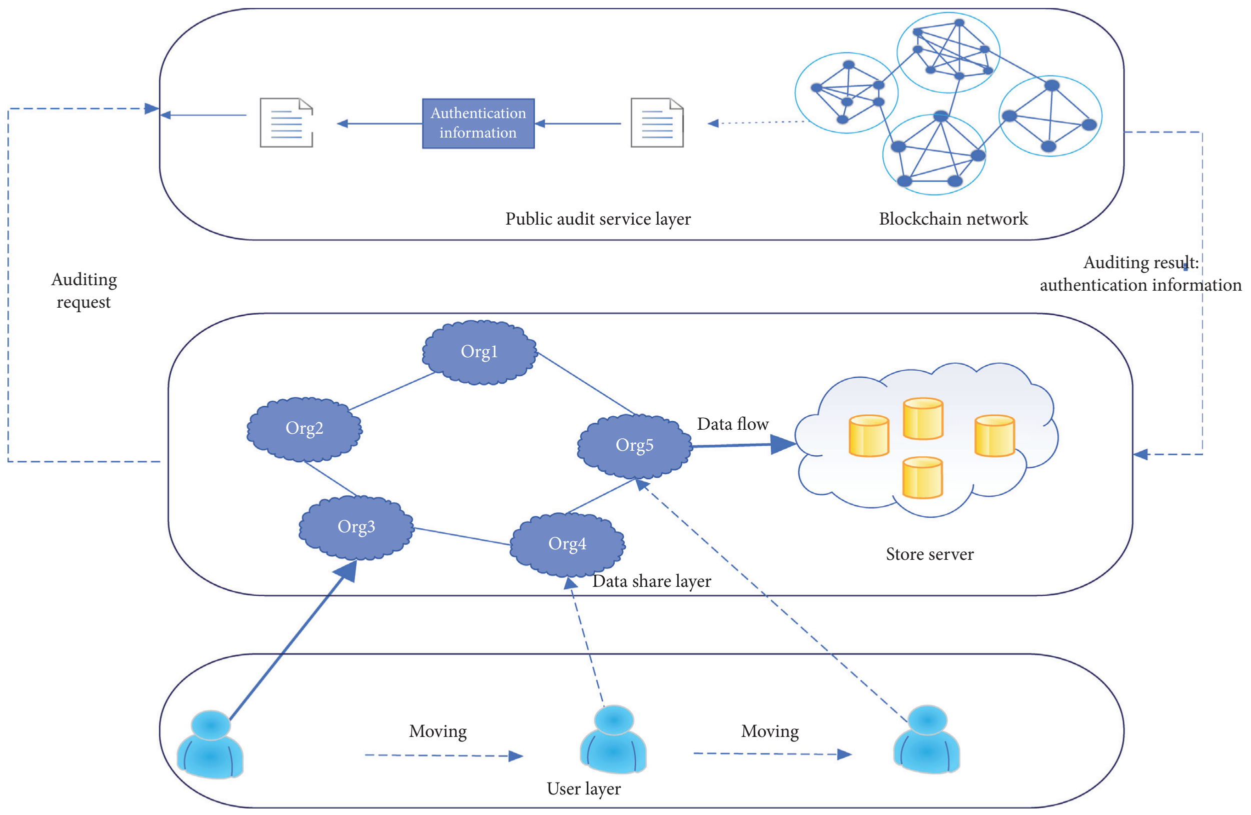


Figure 5: Blockchain-based service migration framework.

verify the correctness of the shared data, but they should not get real data.

(2) Anonymity: data owners want to share data without disclosing their true identities because the data may contain personal privacy information. To protect user privacy, data should be shared anonymously.

(3) Traceability: in some cases, when the edge service is maliciously attacked and tampered with, the manager at the edge control layer can track the true identity of the malicious user.

(4) Publicly verifiable: all users can verify the integrity of the shared data without the need for a trusted third party.

*4.4. PBFT Algorithm.* In order to prevent malicious nodes from attacking during the service migration process in the MEC environment, and to avoid blind trust in the integrity assurance claimed by cloud operators, the blockchain network adopted by our proposed Falcon framework is based on the alliance chain. However, the traditional consortium chain mostly uses the PBFT algorithm as the basic consensus algorithm [43, 44], which uses a three-stage submission process to be divided into preprepare, prepare, and commit. As shown in Figure 6, the consensus process of the PBFT algorithm [45] in a view is mainly divided into the following steps:

(1) The client initiates a consensus request to the master node of the consensus network

(2) When the master node receives the request from the client, it broadcasts the request to other replica nodes through the network

(3) All replica nodes need to perform a Prepare and Commit process between each other

(4) All replicas need to execute the request and reply back to the client

(5) Because the maximum number of Byzantine nodes in the network is $f$, the client needs to wait for $f+1$ different nodes to return the same result before confirming that the entire network reaches a consensus

*4.5. Analysis and Improvement of PBFT Algorithm.* From a detailed analysis of the PBFT process, it can be seen that the existing PBFT algorithm has the following deficiencies:

(1) The client only sends requests to the master node. If there are too many request messages, the master node may be overloaded and the availability will be reduced, which is not suitable for the P2P network environment of the blockchain.

(2) Although the checkpoint protocol solves the recovery of certificate information to reduce memory overhead. However, the certificate is cleared after regular negotiation through the network-wide consistency process. This process is carried out in a

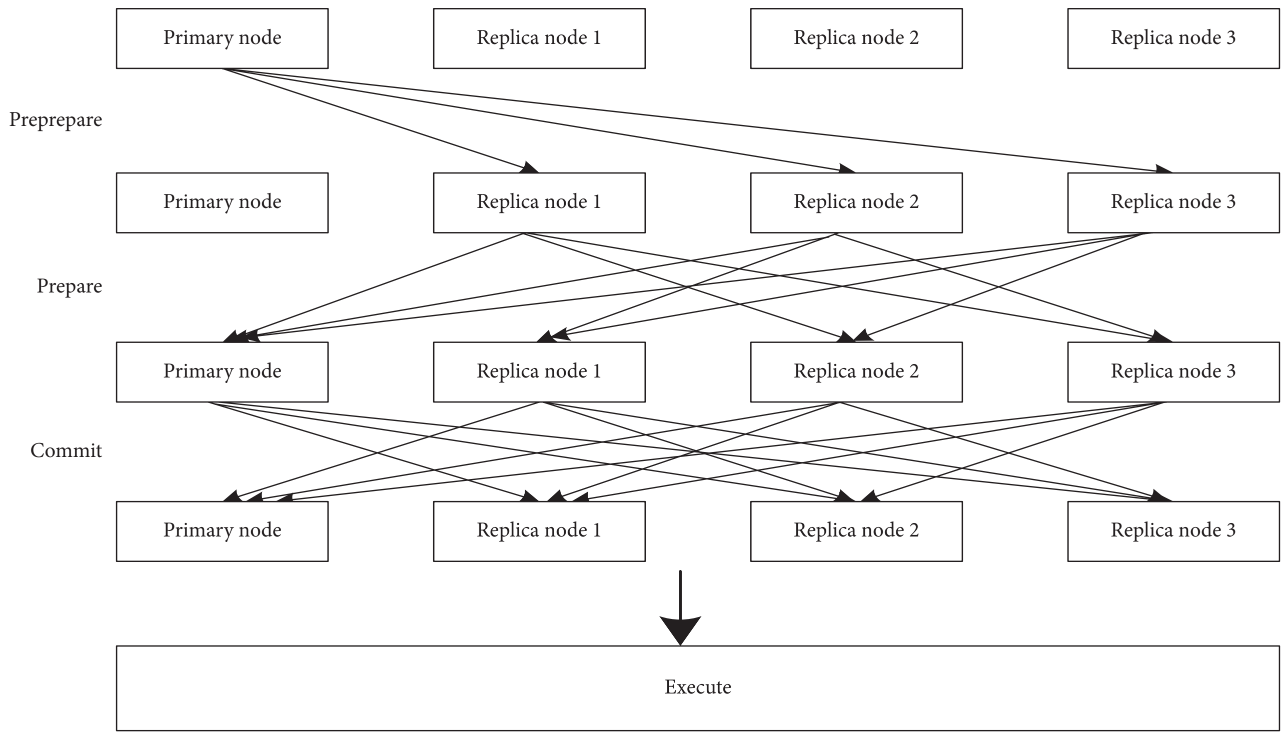


Figure 6: PBFT three-phase submission process.

three-stage submission process, which will cause a lot of unnecessary communication waste.

(3) The PBFT algorithm does not have a relatively complete data backup and synchronization process. As time goes by, the data between different nodes are relatively large.

To sum up the three reasons, the three-phase commit protocol of PBFT requires a relatively large network transmission and communication overhead, which needs to be further optimized. We use the improved PBFT algorithm (IPBF algorithm). Make the following changes:

(1) Modification of the checkpoint protocol: this paper clears the certificate based on the timestamp of the best block in the blockchain. The improvement we propose is to clear the block from the timestamp of the best block in the blockchain. The blockchain is connected in the form of a linked list according to the generation time of the block, so a certificate before a block timestamp has been verified, and the relevant status of the node has been broadcast and can be cleared. We can listen on the block addition event. When a block is added to the blockchain, the certificate before the block timestamp in the node is cleared. This clearing process does not require nodes to communicate with each other, and it can also ensure that the certificates are cleared in time, thereby reducing communication overhead. Fundamentally, it improves block throughput.

(2) Change the client's single point submission request to the master node, and broadcast the signed transaction data to the entire network. This way is more suitable for P2P environment.

(3) Add data synchronization and verification process, and perform data synchronization after the master node election is completed. During synchronization, the slave node verifies the synchronization data. If the verification is passed, the new master node is officially recognized and the next consensus process begins.

## 5. Experiment and Result Analysis

In this section, we introduce experiments and result analysis. In the designed simulation experiment, an edge service migration experiment was performed on the Falcon framework to evaluate the effectiveness of the migration algorithm and further compare it with the service migration based on the docker container. Secondly, the improved blockchain is used to compare throughput by sending batch transactions and creating accounts and compare the experimental results with Ethereum, which has not improved the BPFT consensus algorithm.

*5.1. Experimental Settings.* This research is to simulate the migration framework in part of the server clusters in the China National High Performance Computing Center (Xi'an). The cluster server configuration information is shown in Table 1. The cluster includes three types of server nodes with different computing resource configurations to simulate devices with different node performance in the edge network. Among them, the type A server has the highest CPU and memory resource configuration, the type B server has the lowest CPU and memory resource configuration, and the type C server has medium CPU and memory resource configuration. We use three A and B servers and two C servers.

*5.2. Experimental Design.* We implemented a Falcon prototype system to simulate the migration of edge services composed of a group of mobile agents from one host to another. The framework uses IBM Aglet (Aglet, 2016) as a mobile agent development platform. Aglet is a mobile agent program written in pure Java language for building mobile agent applications. A mobile agent is an executable program that runs in a specific environment. After creation, it can be deployed, migrated, executed, and finally destroyed after the task is completed. Technically, the aglet mobile agent can be executed on any information host that supports the Java virtual machine. The aglets server (called Tahiti) [22] is installed on the remote host, and it provides the aglet execution environment for any mobile agent to reach the local machine. As mentioned in the model in the previous section, Falcon built an event decision management program (EDS), resource management program (RMS), and a data management service program (DMS), and finally , we have developed a decentralized blockchain trust management service, which has a trust authorization mechanism. Aglets can communicate with other aglets, and different agents use the message interaction method provided by the agent to communicate.

In the Falcon framework, the service migration decision is made by EDS, and a group of services is migrated through the migration decision algorithm. As shown in Table 2, 8 mobile agents A1, A2, . . ., . . ., A8 are created in a service group, and each agent belongs to a different type. These agents are assigned to corresponding information hosts that meet the requirements to provide users with edge services. We assume that these 8 agents are initially executed on the same host. Their constraint rules include the following:

(1) A1, A2, and A3 are cooperative in information processing and must be migrated to the same information host
(2) The function of A4 depends on A5 and A6
(3) A7 and A8 are atomic operations due to security considerations and run separately (data management library service, database connection provided)

*5.3. Edge Service Migration Time Comparison.* In order to compare the efficiency of agent-based service migration time, we compare it with the current advanced docker container migration technology. Docker container migration is based on the user-level process checkpoint and recovery tool CRIU (Checkpoint/Restore In Userspace) [20]. We use CRIU to checkpoint the process, freeze the running container process, and then transfer the status data to the destination host for recovery. CRIU also provides a structure called action script, which allows any script to be executed before the container is selected and thawed. Therefore, we design a callback method in the action script to restore the container state. In the test equipment list in Table 1, we have 8 servers under the Type A server list, and the docker container running on each server contains video analysis and face recognition programs. The video parsing and face recognition programs were compared through the two methods of mobile agent and docker container migration, respectively. By using Wondershaper, different bandwidths for testing were adjusted to simulate different network conditions of edge services. The experimental use case is as follows:

Table 1: Laboratory equipment list.

| Server type | Type A | Type B | Type C |
|---|---|---|---|
| Number | 3 | 3 | 2 |
| CPU | 2 ∗ Xeon E5430 2.66 GHz | 2 ∗ Xeon E5310 1.6 GHz | 2 ∗ Xeon E5410 2.33 GHz |
| Core | 8 | 8 | 8 |
| Memory (GB) | 16 | 8 | 8 |
| Storage | 2 ∗ 146 GB SCSI disk | 146 GB SCSI disk | |
| Network | 1 ∗ 100 Mbps, 1 ∗ 1000 Mbps ethernet | | |

(1) Face recognition: we designed a face recognition service as an edge service, using OpenCV [46] to identify images. This service can identify the facial attributes (expressions, ages, genders, etc.) of people on pictures by calling api. Since end users are mobile users, dynamic changes in mobile computing resources are one of the factors that affect user QoS. In addition, when the user moves, the quality of service and network conditions will change, and the communication process needs to maintain ultra-high reliability.
(2) Video streaming: considering different user needs, user mobility, and dynamic network environment, we have prepared video decoding services. Due to the different video definitions under different resolutions, the requirements for decoding are also different. We use three different resolution edge services, namely, high, medium, and low. When the user moves, the edge device node judges whether to perform task migration according to the service quality, and the solution of task migration. For example, when the user moves to another edge node and is not sure of long-term or short-term stay, the low-resolution video decoding task can be migrated first. In the case of long-term user stays, high-resolution services can be provided to avoid untimely migration and waste of resources.

Figure 7 shows the comparison of the decoding time of video files of different sizes in 3 resolutions. It can be seen that it takes more time to decode video with high resolution and high quality of service. This is because we provide video services with different service qualities for users in different

Table 2: Agent distribution.

| Server type | Type A | Type B | Type C |
|---|---|---|---|
| Number | 3 | 3 | 2 |
| Agents | A1, A2, A3 | A4, A5, A6 | A7, A8 |

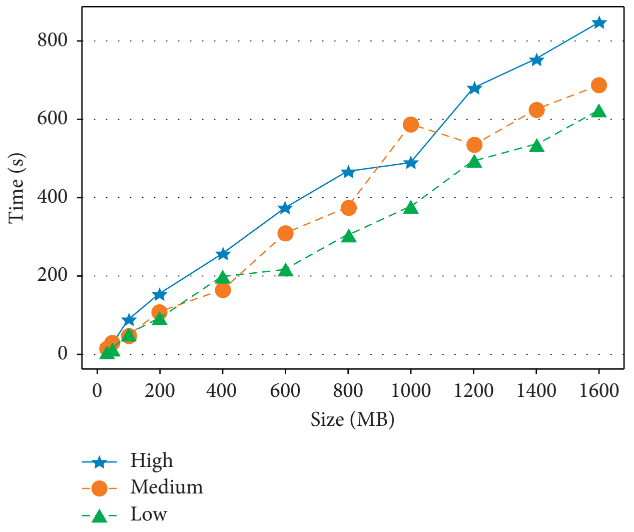


Figure 7: Video decoding time comparison under different service qualities.

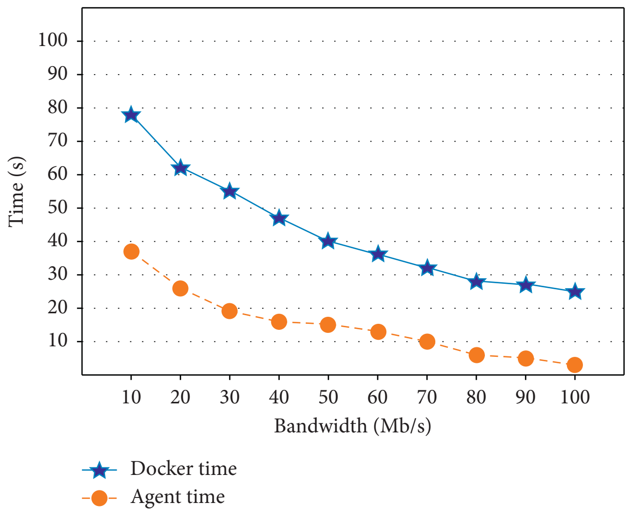


Figure 8: Comparison of migration time.

geographical network environments in edge services. Figure 8 shows the comparison of migration time between docker container and agent-based facial recognition in edge services. Due to its lightweight and its own mobile characteristics, mobile agents are more suitable as carriers for edge service migration. By comparison, it is found that agent-based service migration has a shorter migration time than traditional container migration. And when the bandwidth reaches a certain limit, the migration time basically stabilizes.

*5.4. Energy Consumption Comparison.* Figures 9(a) and 9(b) compares the CPU and memory usage during service migration based on mobile agent and service migration based on docker container. It can be seen that mobile agents are more efficient than docker migration and consume less computing and memory resources. Figure 10 shows a face recognition result seen at the target node after migration. The results show that the agent-based edge service migration adopted by Falcon can make decisions based on system conditions and migrate a group of services to the new edge server, maintaining a high QoS for users.

*5.5. Security Service Migration Simulation.* To test Falcon's mutual trust establishment and smart contract functions during the migration of multiple edge cloud services, we used the Ethereum platform to build a secure service migration framework. Ethereum is the best development platform for blockchain. For better performance, this article uses Truffle to test the Ethereum framework and uses Solidity language to write smart contracts to achieve trust verification of multiple edge clouds. Finally, run smart contracts on the blockchain. The process of building a smart contract in a security framework is shown in Figure 11.

We have introduced a traceability and verification module for the joining and leaving process of the edge cloud in the blockchain system of Falcon, which is compiled and deployed to the blockchain network. Then use TestRPC to simulate access to the Ethereum environment in the Eclipse platform. Finally, the various functions of the smart contract were debugged under the truffle framework to ensure the security of service migration under multiple edge clouds. Next, we create 1000 accounts and count the time spent by sending transaction requests to test the throughput of the modified blockchain. Specific steps are as follow:

(1) Ethereum uses the admin.addPeer (enode) method to connect each edge cloud to form a consortium chain. After starting Ethereum on each node, create an account and address.

(2) We use nodejs scripts to send http requests to the blockchain network for account creation and transaction operations. The instructions can contain parameters such as different transaction quantities and observe the resource consumption of the system under different conditions and the TPS performance of the blockchain. TPS value (transaction per second) can be expressed as follows:

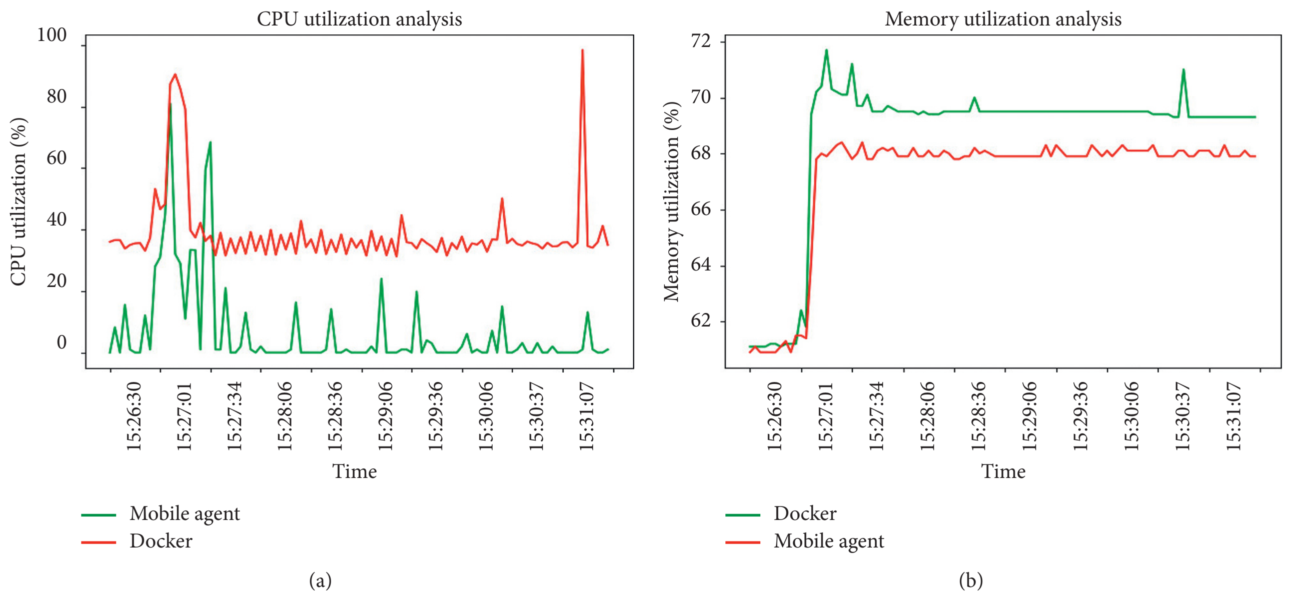
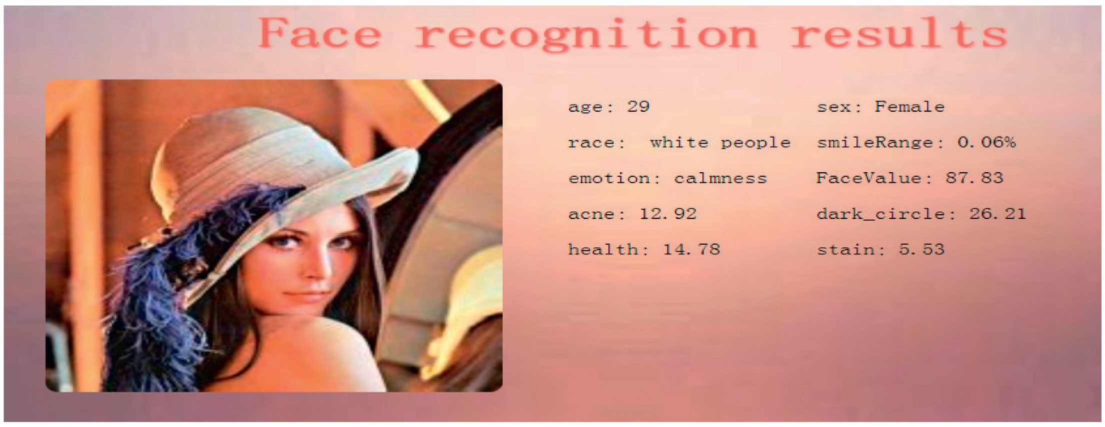


FIGURE 9: Comparison of energy consumption between CPU and memory. (a) Comparison of CPU usage. (b) Comparison of memory usage.



```
age: 29              sex: Female
race:  white people  smileRange: 0.06%
emotion: calmness    FaceValue: 87.83
acne: 12.92          dark_circle: 26.21
health: 14.78        stain: 5.53
```

FIGURE 10: Face recognition results.

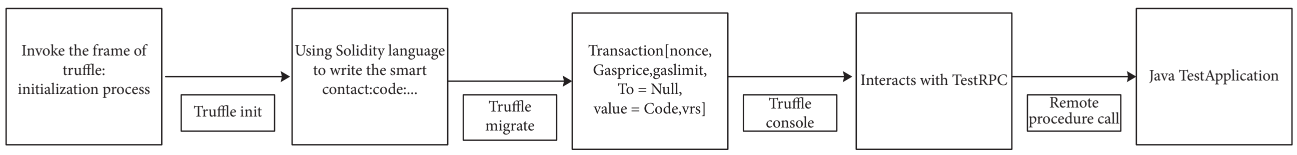
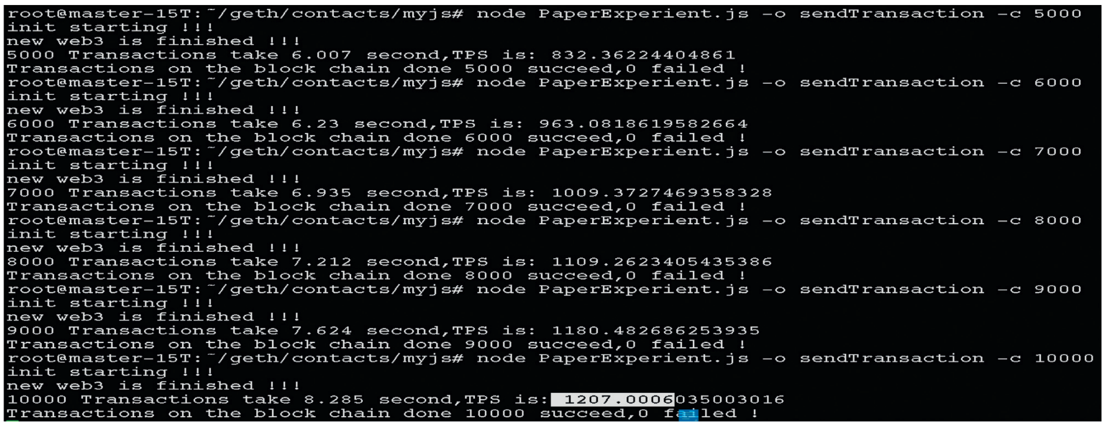

Invoke the frame of truffle: initialization process
Truffle init
Using Solidity language to write the smart contact:code:...
Truffle migrate
Transaction[nonce, Gasprice,gaslimit, To = Null, value = Code,vrs]
Truffle console
Interacts with TestRPC
Remote procedure call
Java TestApplication

FIGURE 11: Construction of a smart contract.

$$\text{TPS} = \frac{\text{Transactions}_{\Delta_t}}{\Delta_t}. \qquad (5)$$

$\text{Transactions}_{\Delta_t}$ is the number of transactions processed by the system within the block time and $\Delta_t$ is the block time. It can be seen from formula (5) that the throughput increases as the capacity of each block increases, but as the block capacity increases, the consensus time and network load will also increase. When it reaches a certain level, the throughput will decrease. As shown in Figure 12, when the number of transactions is 7000–8000, the throughput decreases. However, the overall throughput of the blockchain with the original



```
root@master-15T:~/geth/contacts/myjs# node PaperExperient.js -o sendTransaction -c 5000
init starting !!!
new web3 is finished !!!
5000 Transactions take 6.007 second,TPS is: 832.36224404861
Transactions on the block chain done 5000 succeed,0 failed !
root@master-15T:~/geth/contacts/myjs# node PaperExperient.js -o sendTransaction -c 6000
init starting !!!
new web3 is finished !!!
6000 Transactions take 6.23 second,TPS is: 963.0818619582664
Transactions on the block chain done 6000 succeed,0 failed !
root@master-15T:~/geth/contacts/myjs# node PaperExperient.js -o sendTransaction -c 7000
init starting !!!
new web3 is finished !!!
7000 Transactions take 6.935 second,TPS is: 1009.3727469358328
Transactions on the block chain done 7000 succeed,0 failed !
root@master-15T:~/geth/contacts/myjs# node PaperExperient.js -o sendTransaction -c 8000
init starting !!!
new web3 is finished !!!
8000 Transactions take 7.212 second,TPS is: 1109.2623405435386
Transactions on the block chain done 8000 succeed,0 failed !
root@master-15T:~/geth/contacts/myjs# node PaperExperient.js -o sendTransaction -c 9000
init starting !!!
new web3 is finished !!!
9000 Transactions take 7.624 second,TPS is: 1180.482686253935
Transactions on the block chain done 9000 succeed,0 failed !
root@master-15T:~/geth/contacts/myjs# node PaperExperient.js -o sendTransaction -c 10000
init starting !!!
new web3 is finished !!!
10000 Transactions take 8.285 second,TPS is: 1207.0006035003016
Transactions on the block chain done 10000 succeed,0 failed !
```

Figure 12: Throughput test.

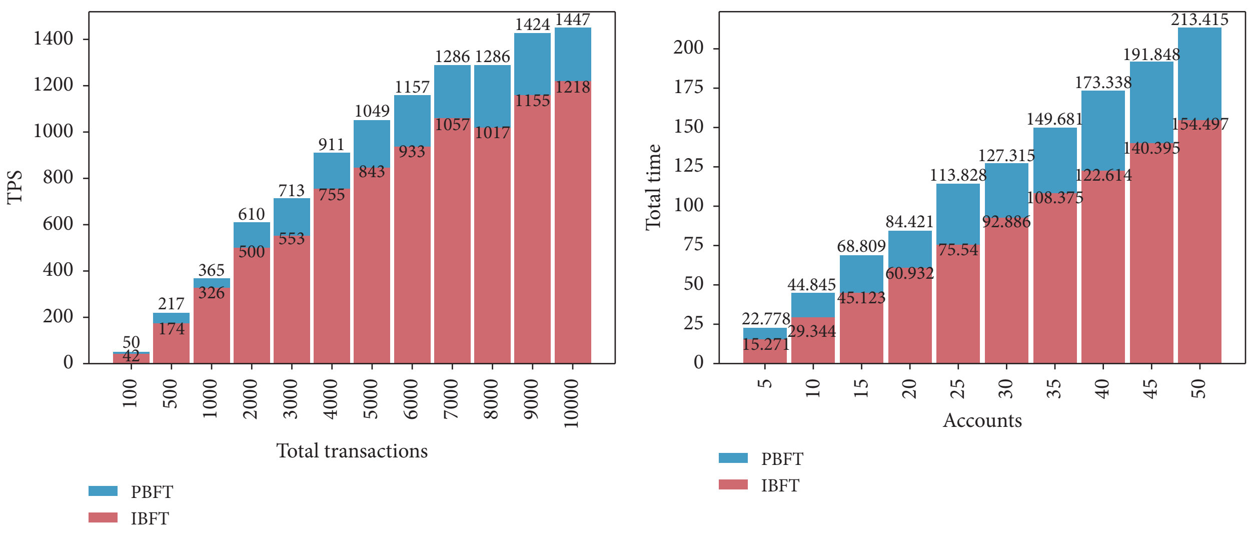


Figure 13: Falcon framework performance comparison. (a) Comparison of throughput of sending transactions. (b) Comparison of throughput of account creation.

unimproved consensus algorithm has been greatly improved, which can satisfy the mutual trust operation between multiple edge networks.

(3) Due to the performance of the machine, the data obtained by only one test have a certain degree of randomness and uncertainty. Therefore, this experiment gradually increases the number of transaction requests in the http request in the test script, performs transaction processing under 1000–10000 cases, and takes the average of the two test results as the TPS performance in this case to find the average. Compare the TPS value. The test process is shown in Figure 12.

Figures 13(a) and 13(b) compare the throughput and time consumption of sending transactions and creating accounts between the EPBFT blockchain system after Falcon's improved consensus algorithm and the unimproved system. The comparison results show that the blockchain system after the improved consensus algorithm has a higher throughput, which is 23% higher than the average throughput of the alliance chain without the improved consensus algorithm. The main reason is that we have optimized the process of clearing certificates and submitting methods after regular PBFT negotiation. The EPBFT algorithm clears the certificate based on the timestamp of the optimal block in the blockchain, which significantly improves throughput. Obviously, Falcon uses blockchain technology in an edge cloud environment that lacks centralized management of different heterogeneous hardware, which can make the migration of edge services in a dynamic, open, and collaborative MEC network environment more secure and reliable. The Falcon framework we proposed is safer in the absence of a trusted third-party environment and is more suitable for practical applications. It provides a feasible solution for other problems such as computing offloading and edge caching in the future edge computing environment.

## 6. Conclusion

In this work, we first proposed a blockchain-based edge service migration framework Falcon, which uses a group of cooperative mobile agents as the service migration carrier to make the service edge service migration more flexible, and designed a service migration decision-making algorithm, aiming to minimize the cost and better service quality for user service migration. Secondly, with regard to the security of service migration, we present that blockchain technology establishes mutual trust, and multiple edge clouds jointly maintain a consortium chain network to avoid trust and security between edge devices without unified management, providing reliability for edge service migration. Finally, through theoretical analysis and experimental simulation, Falcon is more efficient and has higher service quality than service migration based on docker containers and has great potential as a method for next-generation service migration.

In future work, we are interested in modeling the relationship between Falcon's work performance, resource consumption, and parameter configuration. Furthermore, Falcon currently does not support any artificial intelligence technology to control operations in different infrastructures, thereby improving the flexibility of the system. The addition of artificial intelligence technology may be an important contribution of Falcon.

## Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Acknowledgments

This work was supported in part by the National Key Research and Development Program of China under Grant no. 2017YFB0203003, the National Natural Science Foundation of China under Grant nos. 91630206 and 61672423, and the Science and Technology Programs of Henan Provincial Department of Transportation no. 2019J-2-5.

## References

[1] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing," *IEEE Transactions on Mobile Computing*, vol. 18, no. 2, pp. 319–333, 2019.

[2] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5G networks: new paradigms, scenarios, and challenges," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 54–61, 2017.

[3] S. S. Gill, S. Tuli, M. Xu et al., "Transformative effects of IoT, blockchain and artificial intelligence on cloud computing: evolution, vision, trends and open challenges," *Internet of Things*, vol. 8, Article ID 100118, 2019.

[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] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.

[6] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: the communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[7] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[8] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms," *ACM Computing Surveys*, vol. 52, no. 5, pp. 1–37, 2019.

[9] H. Li, G. Shou, Y. Hu, and Z. Guo, "Mobile edge computing: progress and challenges," in *Proceedings of the 2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pp. 83-84, Oxford, UK, March 2016.

[10] M. Mukherjee, R. Matam, L. Shu et al., "Security and privacy in fog computing: challenges," *IEEE Access*, vol. 5, pp. 19293–19304, 2017.

[11] M. Chen, W. Li, G. Fortino, Y. Hao, L. Hu, and I. Humar, "A dynamic service migration mechanism in edge cognitive computing," *ACM Transactions on Internet Technology*, vol. 19, no. 2, pp. 30:1–30:15, 2019.

[12] Z. Gao, Q. Jiao, K. Xiao, Q. Wang, Z. Mo, and Y. Yang, "Deep reinforcement learning based service migration strategy for edge computing," in *Proceedings of the 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp. 116–1165, San Francisco, CA, USA, April 2019.

[13] V. De Nitto Personè and V. Grassi, "Architectural issues for self-adaptive service migration management in mobile edge computing scenarios," in *Proceedings of the 2019 IEEE International Conference on Edge Computing (EDGE)*, pp. 27–29, Milan, Italy, July 2019.

[14] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: a survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2018.

[15] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *IEEE Wireless Communications*, vol. 25, no. 1, pp. 140–147, 2018.

[16] W. Lu, X. Meng, and G. Guo, "Fast service migration method based on virtual machine technology for MEC," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4344–4354, 2019.

[17] G. Sun, D. Liao, V. Anand, D. Zhao, and H. Yu, "A new technique for efficient live migration of multiple virtual machines," *Future Generation Computer Systems*, vol. 55, pp. 74–86, 2016.

[18] L. F. Bittencourt, M. M. Lopes, I. Petri, and O. F. Rana, "Towards virtual machine migration in fog computing," in *Proceedings of the 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pp. 1–8, Krakow, Poland, November 2015.

[19] L. Ma, S. Yi, N. Carter, and Q. Li, "Efficient live migration of edge services leveraging container layered storage," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 2020–2033, 2019.

[20] Criu, 2017, https://criu.org/Maintext{_}Page.

[21] Y. Tian, J. Yuan, and H. Song, "Efficient privacy-preserving authentication framework for edge-assisted internet of drones," *Journal of Information Security and Applications*, vol. 48, Article ID 102354, 2019.

[22] P. Gope and B. Sikdar, "An efficient privacy-preserving authenticated key agreement scheme for edge-assisted internet of drones," *IEEE Transactions on Vehicular Technology*, p. 1, 2020.

[23] S. Alami-Kamouri, G. Orhanou, and S. Elhajji, "Overview of mobile agents and security," in *Proceedings of the 2016 International Conference on Engineering MIS (ICEMIS)*, pp. 1–5, Agadir, Morocco, September 2016.

[24] R. J. C. D. F. S. Iqbal, *Aglets User Manual*, http://aglets.sourceforge.net.Akuma, 2016.

[25] Y. Zuo and J. Liu, "A reputation-based model for mobile agent migration for information search and retrieval," *International Journal of Information Management*, vol. 37, no. 5, pp. 357–366, 2017.

[26] M. G. R. Alam, Y. K. Tun, and C. S. Hong, "Multi-agent and reinforcement learning based code offloading in mobile fog," in *Proceedings of the 2016 International Conference on Information Networking (ICOIN)*, pp. 285–290, Kota Kinabalu, Malaysia, January 2016.

[27] S. Alami-Kamouri, G. Orhanou, and S. Elhajji, "Mobile agent service model for smart ambulance," in *Cloud Infrastructures, Services, and IoT Systems for Smart Cities*, A. Longo, M. Zappatore, M. Villari et al., Eds., pp. 105–111, Springer International Publishing, Cham, Switzerland, 2018.

[28] Y. Zuo and J. Liu, "Mobile agent-based service migration," in *Proceedings of the 2015 12th International Conference on Information Technology—New Generations (ITNG)*, pp. 8–13, IEEE Computer Society, Las Vegas, NV, USA, April 2015.

[29] W. Jiang, Y. Wang, Y. Jiang, J. Chen, Y. Xu, and L. Tan, "Research on mobile internet mobile agent system dynamic trust model for cloud computing," *China Communications*, vol. 16, no. 7, pp. 174–194, 2019.

[30] P. Desai and N. Jayakumar, "A survey on mobile agents," *International Journal for Research in Applied Science and Engineering Technology*, vol. 5, pp. 2915–2919, 2017.

[31] H. Niu and Y. Liu, "A mobile agent-based task seamless migration model for mobile cloud computing," in *Proceedings of the 2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA)*, pp. 241–246, Ottawa, Canada, September 2014.

[32] A. Belghiat, E. Kerkouche, A. Chaoui, and M. Beldjehem, "Mobile agent-based software systems modeling approaches: a comparative study," *Journal of Computing and Information Technology*, vol. 24, no. 2, pp. 149–163, 2016.

[33] R. Wang, Y. Cao, A. Noor, T. A. Alamoudi, and R. Nour, "Agent-enabled task offloading in UAV-aided mobile edge computing," *Computer Communications*, vol. 149, pp. 324–331, 2020.

[34] C. Yang, X. Chen, and Y. Xiang, "Blockchain-based publicly verifiable data deletion scheme for cloud storage," *Journal of Network and Computer Applications*, vol. 103, pp. 185–193, 2018.

[35] H. Huang, X. Chen, and J. Wang, "Blockchain-based multiple groups data sharing with anonymity and traceability," *Science China Information Sciences*, vol. 63, no. 3, Article ID 130101, 2019.

[36] R. Zhang, R. Xue, and L. Liu, "Security and privacy on blockchain," *ACM Computing Surveys*, vol. 52, no. 3, pp. 51:1–51:34, 2019.

[37] H. Huang, K.-C. Li, and X. Chen, "Blockchain-based fair three-party contract signing protocol for fog computing," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 10, Article ID e4469, 2018.

[38] E. Gaetani, L. Aniello, R. Baldoni, F. Lombardi, A. Margheri, and V. Sassone, "Blockchain-based database to ensure data integrity in cloud computing environments," in *Proceedings of the ITASEC*, Venice, Italy, January 2017.

[39] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "Integration of blockchain and cloud of things: architecture, applications and challenges," *IEEE Communications Surveys and Tutorials*, 2019.

[40] S. Huh, S. Cho, and S. Kim, "Managing IoT devices using blockchain platform," in *Proceedings of the 2017 19th International Conference on Advanced Communication Technology (ICACT)*, pp. 464–467, Bongpyeong, South Korea, February 2017.

[41] M. A. Ferrag, M. Derdour, M. Mukherjee, A. Derhab, L. Maglaras, and H. Janicke, "Blockchain technologies for the internet of things: research issues and challenges," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2188–2204, 2019.

[42] A. Singh, R. M. Parizi, Q. Zhang, K. R. Choo, and A. Dehghantanha, "Blockchain smart contracts formalization: approaches and challenges to address vulnerabilities," *Computers & Security*, vol. 88, Article ID 101654, 2020.

[43] K. Zheng, Y. Liu, C. Dai, Y. Duan, and X. Huang, "Model checking PBFT consensus mechanism in healthcare blockchain network," in *Proceedings of the 2018 9th International Conference on Information Technology in Medicine and Education (ITME)*, pp. 877–881, Hangzhou, China, October 2018.

[44] H. Sukhwani, J. M. Martínez, X. Chang, K. S. Trivedi, and A. Rindos, "Performance modeling of PBFT consensus process for permissioned blockchain network (hyperledger fabric)," in *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, pp. 253–255, Hong Kong, China, September 2017.

[45] S. Sasirekha and S. Swamynathan, "Cluster-chain mobile agent routing algorithm for efficient data aggregation in wireless sensor network," *Journal of Communications and Networks*, vol. 19, no. 4, pp. 392–401, 2017.

[46] Opencv User Manual, 2018 https://docs.opencv.org/4.2.0/d9/df8/tutorialtext{_}root.html.