electronics

Article

Differentiated Security Requirements: An Exploration of Microservice Placement Algorithms in Internet of Vehicles

Xing Zhang [1,2], Jun Liang [1,*], Yuxi Lu [3], Peiying Zhang [3] and Yanxian Bi [4,*]

[1] Automotive Engineering Research Institute, Jiangsu University, Zhenjiang 212013, China; xzhang@sues.edu.cn
[2] School of Management, Shanghai University of Engineering Science, Shanghai 201620, China
[3] Qingdao Institute of Software, College of Computer Science and Technology, China University of Petroleum (East China), Qingdao 266580, China; s21070050@s.upc.edu.cn (Y.L.); zhangpeiying@upc.edu.cn (P.Z.)
[4] China Academy of Electronic and Information Technology, CETC Academy of Electronics and Information Technology Group Co., Ltd., Beijing 100041, China
* Correspondence: liangjun@ujs.edu.cn (J.L.); biyanxian@cetc.com.cn (Y.B.)

Abstract: In recent years, microservices, as an emerging technology in software development, have been favored by developers due to their lightweight and low-coupling features, and have been rapidly applied to the Internet of Things (IoT) and Internet of Vehicles (IoV), etc. Microservices deployed in each unit of the IoV use wireless links to transmit data, which exposes a larger attack surface, and it is precisely because of these features that the secure and efficient placement of microservices in the environment poses a serious challenge. Improving the security of all nodes in an IoV can significantly increase the service provider's operational costs and can create security resource redundancy issues. As the application of reinforcement learning matures, it is enabling faster convergence of algorithms by designing agents, and it performs well in large-scale data environments. Inspired by this, this paper firstly models the placement network and placement behavior abstractly and sets security constraints. The environment information is fully extracted, and an asynchronous reinforcement-learning-based algorithm is designed to improve the effect of microservice placement and reduce the security redundancy based on ensuring the security requirements of microservices. The experimental results show that the algorithm proposed in this paper has good results in terms of the fit of the security index with user requirements and request acceptance rate.

Keywords: microservices; reinforcement learning; security constraints; user requirement fit; request acceptance rate

check for updates

Citation: Zhang, X.; Liang, J.; Lu, Y.; Zhang, P.; Bi, Y. Differentiated Security Requirements: An Exploration of Microservice Placement Algorithms in Internet of Vehicles. *Electronics* **2024**, *13*, 1597. https://doi.org/10.3390/electronics13081597

Academic Editor: Hung-Yu Chien

Received: 27 March 2024
Revised: 18 April 2024
Accepted: 19 April 2024
Published: 22 April 2024

1. Introduction

The Internet of Vehicles (IoV) is a very promising technology [1], and in recent years, with the rapid development of the Internet and the IoV, the number of vehicle users has been growing, and the generation and processing of large amounts of data have become common phenomena [2]. Meanwhile, as the cost of a network channel's bandwidth and computing resources decreases, internet applications are becoming larger and larger, with too much coupling existing between functions and features for system changes and later maintenance [3,4]. Unlike traditional networks, bandwidth and computing resources are generally scarce in IoV environments, and the use of individual applications may face quality of service problems such as high latency and network congestion. Microservices are one solution to this problem because they have the feature of splitting a large application into multiple small and independent services [5,6], each running in an independent process. Microservices communicate with other services through open service interfaces to jointly fulfill the functions of the application [7]. Several studies have shown that microservice technology can be ported to the IoV by combining containerized microservice technology with the Kubernetes orchestration tool [8–10], as well as by designing orchestration algorithms to manage the whole process. By applying microservice technology within the IoV, the performance in terms of network flexibility and scalability can be improved.

Although microservice management can be implemented in the IoV, the large scale of microservices creates a complex chain of microservice calls as the number of microservices grows exponentially. The vehicle may not be able to handle the task when it reaches the IoV due to its limited computational resources [11]. In addition, network communication between microservices exposes a larger attack surface because communication between vehicles and between vehicles and fixed base stations on the road, servers, and other facilities uses wireless communication [12]. In the communication process, the number of data transfers increases, and the increased number of data transfers not only increases communication overheads but also poses challenges to the security of user data. Protecting personal privacy and sensitive information in this process has become a topic of great concern [13]. Existing security placement solutions are not as effective as traditional security defense techniques such as intrusion detection in the face of large, dynamically deployed microservices, and are more difficult to implement [14,15]. In addition, users have different needs concerning security levels; for example, the transmission of citizens' personal information and governmental data requires a high level of communication security, whereas behaviors such as the prediction of traffic flow in the city do not require a high level of security.

Figure 1 shows the network environment where microservices are placed in the Internet of Vehicles (IoV) as an example on the left side, and the result of the microservices' deployment on the right side. Figure 1 shows that information between vehicles and with fixed actors such as roadside base stations is transmitted via wireless signals, which makes it much easier for an attacker to intercept these data using specialized devices or software than wired communications that use physical links for data transmission, thereby stealing sensitive information or interfering with the normal operations of microservices [16]. In addition, when it comes to information-sensitive applications such as navigation, vehicles collect a large amount of important information such as roadside and geographic coordinates while driving, and if this service is placed in a node that does not meet the security standards, it will increase the likelihood of sensitive information leakage and attacks. Designing a deployment algorithm that meets the security expectations of microservices can greatly improve the security performance of the entire network. This allows the operator's entire service to provide services to users at a high level of security, and the user's security requirements are guaranteed at the same time, resulting in greater user satisfaction and promoting the development of a network society [17,18]. However, higher security necessitates greater costs, and with a large number of network nodes, these costs need to be borne by network operators and users, so it is not practical to upgrade every node where microservices can be placed to a highly reliable and secure node. Each microservice has different security requirements. For example, assume there are two services involving two high-security to-be-placed nodes and one low-security to-be-placed node, and the two microservices are the police linkage service and the road information reporting service. Compared to the police linkage service, the road information reporting service does not have high security requirements. Thus, the most suitable placement scenario is to place the police linkage service in the high-security node and the road condition information service in the low-security node. This requires reducing the likelihood of attacks on microservices and improving the security of the entire network, taking into account the constraints of the CPU and other resources of the nodes as well as the QoS. There are several deployment options. This study has chosen B, C, G, and F as the nodes where each of the four microservices are placed, and the red link is the actual deployment link. In Figure 1, the network topology used contains eight nodes and nine links, which is a relatively simplified network. However, in real-world network environments, the number of network nodes, the complexity of the links, and the volume of requests for microservices are usually far beyond these numbers. Currently, numerous scholars, such as He et al. [19–24], use traditional mathematical methods or heuristic algorithms to deal with the placement of microservices. Although these methods are effective in some cases, they tend to consume a lot of computational resources when dealing with large-scale, complex networks and may lead to obtaining only locally optimal solutions. In addition, other scholars, such as Li et al. [25–28], have begun to explore the use of deep learning techniques, but these studies tend to ignore post-deployment security issues. Considering the current practical challenges of microservice deployment, the trend of increasing network size in the future [29], and the research value of microservice-related problems, it is particularly important to develop an algorithm that considers both security and efficiency when handling microservice deployment [30]. This algorithm should not only provide solutions to existing problems but should also be able to adapt to future developments in the network environment to ensure long-term viability and effectiveness. This will ensure that operators can provide services to their subscribers at a highly secure level of performance, while at the same time fulfilling the subscribers' requirements for security, thereby enhancing subscriber satisfaction. This enhancement not only strengthens subscriber trust but also contributes to the overall development of a network society.

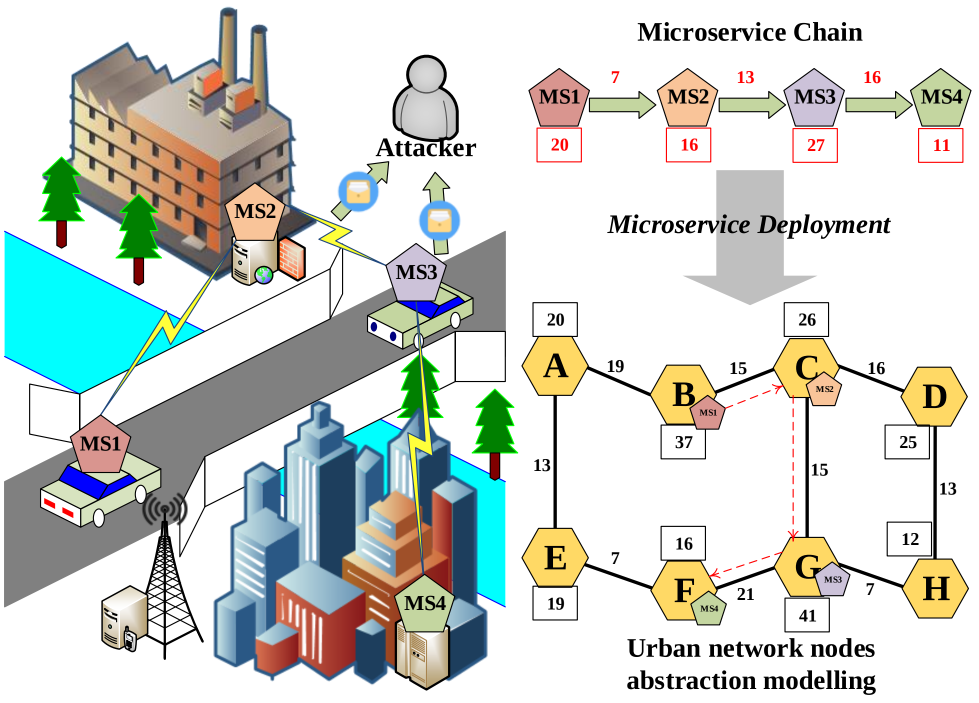

Figure 1. Microservice placement environment and basic placement methods.

The main contributions of this paper are as follows:

- This study proposes a microservice placement strategy based on differential security requirements, targeting environments like the IoV that heavily rely on wireless communication and are susceptible to network attacks. This study models the microservice placement process as a graph-embedding process. This study combines the node placement process with an asynchronous reinforcement learning algorithm by setting up an agent to make the algorithm converge quickly and to make it perform well in a network environment with a large number of microservices to be placed.
- Unlike traditional placement methods, the proposed algorithm dynamically handles the security parameters of each microservice placement node. The security index is set for the placement node to cope with the users' differentiated security requirements, ensuring that each microservice is placed in a node that meets its security requirements as much as possible to reduce the likelihood of cyber-attacks.

- In addition, this study also conducts simulation experiments based on the real network environment, and the algorithm proposed in this paper shows good results in the request acceptance rate and security demand satisfaction index in comparison with other algorithms.

The overall structure of this paper is as follows. The work related to the secure placement of microservices is presented from two perspectives in Section 2. Immediately after that, the proposed asynchrony-based differential microservice placement algorithm is described in detail in Section 3. Section 4 describes the environment setup for simulation experiments and analyzes the results in detail. Finally, in Section 5, this study is concluded and possible future research directions for microservice security placement are analyzed.

2. Related Work

This section introduces and analyzes research advances in recent years in the areas of microservice security and microservice placement, respectively.

2.1. Microservices Security

In the study by Gopal [31] and Hossain et al. [32], it is proposed that although microservices, as a lightweight architecture that can fully utilize the capabilities of edge nodes, can effectively address challenges such as poor scalability of monolithic programs, the complex invocation relationships that exist between microservices expose a larger attack surface. The placement, scheduling, and auto-scaling of microservices may have security threats, so it is crucial to develop lightweight cryptographic mechanisms or secure placement algorithms. Jin et al. [33] proposed a framework that can sense changes in the cloud environment and dynamically optimize for changes in mobile target defense. Jin et al. [33] formalized various attack scenarios to build a multidimensional attack model. The effectiveness of container technology and mobile target defense in a microservices architecture is accurately evaluated. Rishi et al. [34] attempt to detect attacks launched towards microservices and propose a Docker-based lightweight architecture that can reduce traffic overhead by deploying the solution at the edge for earlier detection and mitigation of attacks. Zdun et al. [35] proposed a model-based approach to determine compliance with safety standards by predicting whether the environment conforms to predefined metrics. Experimental results show the accuracy of this method. In contrast to setting security policies, Xu et al. [36] propose the idea of setting security components separately. They deployed an additional security gateway that supports authentication and authorization in the original gateway. The experimental results also show that the communication latency of microservices provided based on the secure gateway is lower. Ibrahim et al. [37] devise a set of security rules to constrain network function placement behavior using a mixed-integer linear programming optimization model, but the paper focuses more on latency minimization. Overall, there is relatively little existing research on the security aspects of microservice placement and a lack of algorithms for microservice placement based on users' differentiated security requirements.

2.2. Microservice Placement

For this type of node placement problem, the classical approach proposed by Cheng et al. [38] uses a Markov model to rank the network nodes according to their resources and topological properties. Experimental results also show that the method proposed by Cheng et al. yields some improvement in the evaluation metrics of long-term yield and acceptance rate. Similarly, Selimi et al. [39] design a low-complexity heuristic algorithm to solve the microservice placement problem. The information required for the microservice placement decision is derived from the node availability and network bandwidth dimensions of the network information. Experimental results show that the microservice placement algorithm proposed by Selimi et al. provides a significant improvement over the random placement method in terms of both customer response time and packet loss rate. However, the present placement method does not consider security in specific usage scenarios and other factors that affect placement. In contrast, Gu et al. [40] first proved that the microservice placement problem is an NP-hard problem, modeled the microservice placement problem in the form of integer linear programming, and then solved the microservice deployment problem by combining intra-service and inter-server sharing. Experiments show that the approach proposed by Gu et al. provides a large improvement in edge throughput rate. He et al. [19] define the service placement problem in microservice systems with complex dependencies and multiple instances as a fractional polynomial problem and transform it into a quadratic ratio and fractional problem to be solved by a greedy algorithm. The experimental results also show the computational speed advantage of the algorithm proposed by He et al. [19] Moreover, Li et al. [41] also use heuristic algorithms to establish a microservice layout optimization model to minimize the average transmission latency and load balancing and use genetic algorithms to effectively achieve low latency and load balancing. Liu et al. propose a multifactor evolutionary algorithm for the container placement problem in the microservices framework, setting multiple optimization objectives to cope with the container placement problem for a wide range of applications of different sizes in the heterogeneous clustering environment, solving the container placement problem in heterogeneous clustered environments. Su et al. [42] address the problem of microservice placement in satellite networks in a space–air–ground integrated network and propose an approach that uses an attention mechanism to efficiently capture attributes, such as computational resources, that affect microservice placement. Simulation experiments show that the proposed method of Su et al. performs well in terms of gain–overhead ratio and placement request acceptance rate.

In general, existing solutions to the node placement problem are mainly classified into several categories such as using traditional algorithms, heuristic algorithms, and deep learning methods. However, there is not much research on the microservice placement problem considering security. This paper is inspired by the above research and proposes a microservice placement strategy with differentiated security requirements.

3. Algorithm Description

In this section, the modeling scheme of the problem is first presented, and reasonable constraints and performance evaluation metrics are designed according to the network. The proposed algorithm is presented in detail immediately after based on the problem modeling.

3.1. Problem Definition

In this study, the set MSR is used to represent the set of service requests, and its expression is shown in Equation (1).

$$MSR = \{G_1^{ms}, G_2^{ms}, \cdots, G_x^{ms}\}, \tag{1}$$

where this study abstracts all microservices that complete a full service into a directed weighted graph G, as shown in Equation (2).

$$G^{ms} = \{N^{ms}, L^{ms}\}, \tag{2}$$

where N^{ms} represents the set of all microservices that complete this service and L^{ms} represents the set of open links invoked between microservices.

For each microservice node ($n^{ms} \in N^{ms}$), there is a $Cost_{CPU}(n^{ms})$ representing the CPU resources that need to be consumed to run this microservice. Similarly, for each link l^{ms}, there is a $Cost_{BW}(l^{ms})$ representing the bandwidth resources that need to be consumed by the two microservices to call through the open interface. Furthermore, this study defines the abstraction of the entity nodes where the microservices are placed as an undirected weighted graph G^{env}, as shown in Equation (3).

$$G^{env} = \{N^{env}, L^{env}\}, \tag{3}$$

where N^{env} represents the collection of physical nodes where the microservice is deployed and L^{env} represents the collection of communication link between the physical nodes. For any $n^{env} \in N^{env}$, there exists $Remain(n^{env})$, representing the remaining to compute and storage resources of that node; $Type_n^{env}$, representing the set of types of microservices that this node supports for deployment, and $Sec(n^{env})$, representing the security level of the node. Similarly, for a link linking physical nodes, there is $Band(l^{env})$, representing the remaining bandwidth of the link. Security requirements for the IoV include, but are not limited to, security requirements for remote wireless update packet authentication, geo-referenced security requirements when providing location information services, and data security requirements in data analytics services. Due to the number of services and the variety of security requirements in the IoV, it is not easy to quantify. Therefore, this study quantifies the security requirements of different microservices in the IoV in terms of security levels.

In a real network environment, every node is not exactly the same in terms of security due to hardware and technology differences. In other words, they have different levels of security. For example, high-security nodes usually integrate stronger firewall features or can support multiple encryption protocols such as WPA3, which can effectively block malicious traffic and potential threats. Deploying microservices in higher-security nodes can effectively reduce the likelihood of malicious attacks on microservices, thus reducing the probability of the entire application being affected. This study establishes a positive correlation between security index and security performance, where the higher the security index, the better the security. Similarly, microservices have different security-level needs. Microservices that have less impact on national and personal interests (e.g., commodity information management services) require lower placement node security levels than sensitive services (e.g., payment services).

Given that microservice requests eventually need to be deployed on physical nodes in the environment, the relationship between G_i^{ms} and G^{env} can be formalized as Equation (4).

$$G_i^{ms} \mapsto G^{env}, \forall G_i^{ms} \in MSR, \tag{4}$$

The deployment process is considered to be completed when each of the microservices contained in the service request and their corresponding links are successfully deployed in the physical network. In the early stages of microservice deployment, due to the relative abundance of available resources in the physical network, each service request arriving in the network has a variety of potential deployment locations to choose from when deploying the microservice and its links. Each deployment option will have an impact on the subsequent deployment process. This study uses two binary variables, Equations (5) and (6), to express whether the microservices and links in the request G_i^{ms} are successfully deployed to the corresponding physical nodes or links, respectively.

$$\Phi_j^i = \begin{cases} 1, & n_i^{ms} \text{ successfully deployed to } n_j^{env} \\ 0, & n_i^{ms} \text{ unsuccessfully deployed to } n_j^{env} \end{cases} \tag{5}$$

where n_i^{ms} represents the i-th microservice in this service request and n_j^{env} represents the j-th physical node in the physical network topology. A binary variable of 1 means that n_i^{ms} has been successfully deployed to node n_j^{env}, and conversely, a variable of 0 means the microservice n_i^{ms} has not been deployed to n_j^{env}. Similarly, the binary variable Ψ shown in Equation (6) follows this rule.

$$\Psi_{rs}^{pq} = \begin{cases} 1, & l_{pq}^{ms} \text{ successfully deployed to } l_{rs}^{env} \\ 0, & l_{pq}^{ms} \text{ unsuccessfully deployed to } l_{rs}^{env} \end{cases} \tag{6}$$

where l^{ms}_{pq} represents the link between microservice n^{ms}_{p} and n^{ms}_{q} and l^{env}_{rs} represents the link between physical node n^{env}_{r} and n^{env}_{s}. The formulae for calculating the remaining computing and storage resources are shown in Equations (7) and (8).

$$Band(l^{env}) = O_{bw}(l^{env}) - \sum_{i=1,j=1,l^{ms}\Uparrow l^{env}}^{|MSR|} Cost\left(l^{ms}_{ij}\right), \tag{7}$$

where $O_{bw}(l^{env})$ represents the bandwidth at the idle time when no microservices are placed on the physical link l^{env} and $|MSR|$ represents the number of microservice requests.

$$Remain(n^{env}) = O_{cpu}(n^{env}) - \sum_{i=1,n^{ms}\Uparrow n^{env}}^{|MSR|} Cost(n^{ms}_{i}), \tag{8}$$

where $O_{cpu}(n^{env})$ represents the total amount of computational and storage resources in the physical node when no microservices are deployed.

The constraints shown in Equations (9)–(13) also need to be satisfied during microservice placement. Firstly, to ensure the proper operation of the microservices, the remaining computational resources of the server must be greater than the computational resources required by the microservices, which is expressed as shown in Equation (9).

$$\begin{gathered}\text{Remain}\left(n^{env}_{p}\right) - \text{Cost}_{CPU}\left(n^{ms}_{q}\right) \geq 0, \\ n^{env}_{p} \in N^{env}, n^{ms}_{q} \in N^{ms}, if\ \Phi^{p}_{q} = 1\end{gathered} \tag{9}$$

Similarly, the remaining bandwidth of the communication links between servers must also meet the bandwidth consumed by the calls between microservices. Otherwise, network delays and packet loss will occur, which will affect the normal operation of the whole service. The expression is shown in Equation (10).

$$\begin{gathered}\text{Band}(l^{env}_{rs}) - \text{Cost}_{BW}\left(l^{ms}_{ij}\right) \geq 0, \\ l^{env}_{rs} \in L^{env}, l^{ms}_{ij} \in L^{ms}, if\ \Psi^{ij}_{rs} = 1\end{gathered} \tag{10}$$

In addition, the security index of each microservice placement request should be at least greater than or equal to the actual user requirement for the security index. The expression is shown in Equation (11).

$$\sum_{n^{ms}\Uparrow n^{env}} \text{Sec}(n^{env}_{i}) \geq \text{Sec}_{user} \tag{11}$$

Security-related information is self-contained attribute information in the network nodes, similar to other resource attributes such as CPU and bandwidth, which can be obtained directly from the network. Finally, each microservice can only complete its lifecycle in one physical node, and this expression is shown in Equation (12).

$$\sum_{j=1}^{|N^{env}|} \Phi^{i}_{j} = 1 \tag{12}$$

In contrast to nodes, the transmission of data emanating from a microservice over a physical link can take multiple hops to reach the destination node, and this expression is shown in Equation (13).

$$\sum_{i=1}^{|L^{env}|} \Psi^{pq}_{l^{env}_{i}} \geq 1 \tag{13}$$

Finally, consider the rapid development of container technologies such as Docker and container orchestration technologies such as Kubernetes. In the future, microservice architectures will not be affected by inherent placement requirements and will achieve smoother deployment and usage, so the inherent placement requirements for certain microservices are not constrained.

3.2. Evaluation Indicators

The placement acceptance rate of microservice requests is the most basic metric for evaluating the quality of placement services. The placement acceptance rate is directly proportional to the performance of the placement algorithm. Under a certain number of elements in the MSR of the microservice placement request set, if the placement acceptance rate is high, it means that the more applications are successfully deployed in the physical network, the stronger the stability of the whole microservice architecture, and the better the user experience. The expression for the placement acceptance rate is shown in Equation (14).

$$R_{\text{accept}} = \frac{|MSR_{\text{accepted}}|}{|MSR_{\text{arrived}}|} \times 100\%, \tag{14}$$

where $|MSR_{\text{accepted}}|$ is the total number of microservice placement requests arriving at the network and $|MSR_{\text{accepted}}|$ represents the number of microservices that were successfully placed. Under the premise of ensuring the acceptance rate, the security index of the whole link is made to meet the user's needs as far as possible. The security index after the final placement is completed will neither fall short of the user's security needs nor cause the risk of privacy leakage. At the same time, it will not be substantially higher than the user's security needs, causing unnecessary waste of security resources. Equation (15) is the calculation method for the evaluation index of user fit.

$$R_{\text{srfr}} = \frac{\text{Sec}_{user}}{\sum_{n^{ms} \Uparrow n^{env}} \text{Sec}(n_i^{env})} \times 100\% \tag{15}$$

where Sec_{user} is the security requirement and $\sum_{n^{ms} \Uparrow n^{env}} \text{Sec}(n_i^{env})$ is the sum of the security levels of the nodes occupied by the actual microservices prevented. As mentioned earlier, this paper tries to place services in nodes that meet their security requirements as much as possible, reducing the redundancy of security resources while safeguarding their basic security needs. To measure this effect, this paper proposes the evaluation metric of security fit. A higher security fit R means less waste of security resources and better results of the algorithm.

3.3. Algorithm Design

To achieve the optimization goals mentioned above, and to adapt to complex and dynamic microservice placement environments and constraints, as well as to reduce human intervention in placement, this study proposes an algorithm based on asynchronous reinforcement learning (RL) for secure microservice placement. This study uses Actor–Critic as the basic framework, but unlike the traditional Actor–Critic model, the asynchronous security algorithm proposed in this paper improves convergence by introducing multiple parallel intelligences. Each agent possesses its own parameters and policies, allowing them to interact and learn in diverse network environments. After a certain number of learning iterations, the gradient of the loss function in each agent is calculated and updated in the central neural network. The whole architecture is similar to a tree of public ancestors. Because each intelligence learns individually in its own environment, the empirical data collected are relatively independent and do not affect each other, thus increasing the diversity and richness of the samples. After updating to the established rounds, the model of the central network is the final model this study uses for making the selection of nodes for microservice placement.

The detailed architecture of the method is shown in Figure 2. The Actor and Critic are the two key parts of the algorithm. In the algorithm, the Actor is mainly used for the selection of microservice placement actions, and the Critic is used to evaluate the value generated by the microservice placement and provide feedback to the Actor to help it improve the placement strategy. In carrying out the above process, three core concepts are involved: state space, action space, and reward. Here, the precise definition of the state space is the key to implementing an effective reinforcement learning algorithm. This is because the Actor must select the appropriate action based on the current state and learn the optimal behavioral strategy in the process. In addition, the network environment changes (e.g., node computational resource availability) after the microservice deployment task is completed, which requires the algorithm to be able to perform state updates in real time. At the same time, the reward mechanism needs to give feedback to the Actor based on the change in state. The complexity of the network environment further increases the difficulty of obtaining the state of the network environment. In particular, the size of the state space directly determines the complexity of the problem, as the intelligence cannot understand the network environment directly. If the information extracted from the environment contains too much irrelevant data, it will reduce the efficiency of the algorithm. To solve this problem, this study extracts key information from the environment that affects the effectiveness of microservice deployment to form a feature matrix. Specifically, this study selects the following network features from the network environment to construct the feature matrix.

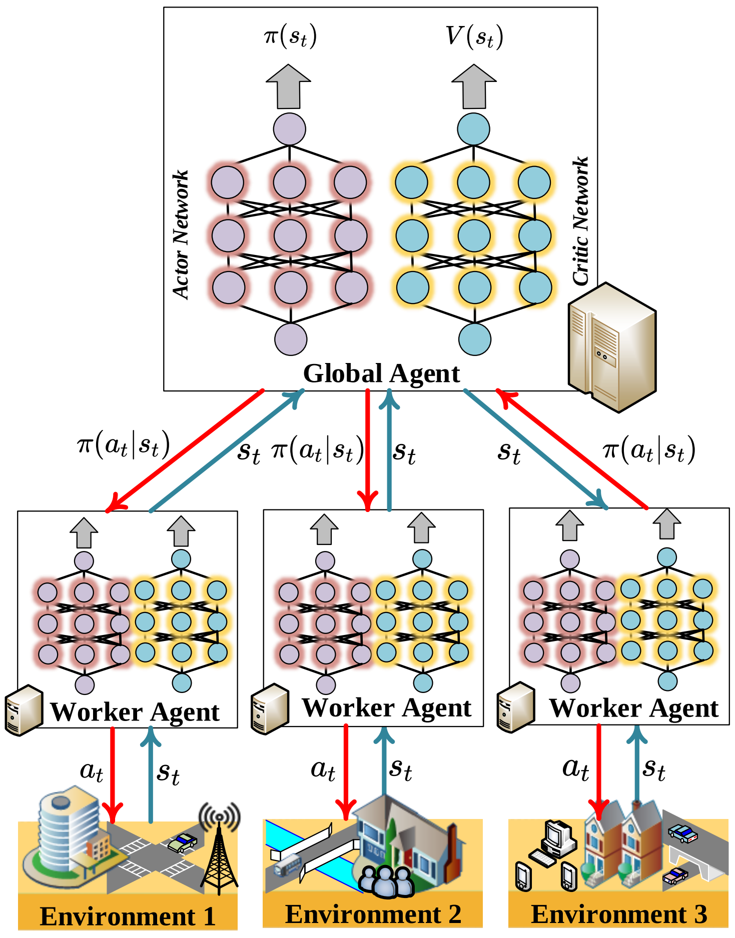

Figure 2. Microservice security placement network architecture.

- **Available resources of nodes:** If the amount of available resources of a physical node does not reach the resources consumed by a microservice's deployment, it cannot be deployed, so it is evident that the amount of available resources of a node is directly related to the success rate of the microservice deployment. Given this, this study includes available node resources as a key feature extracted from the network environment. Its calculation is shown in Equation (8).

- **Bandwidth resources of links:** The greater the available bandwidth resources of a link, the more data will be allowed to be transmitted simultaneously. If the available bandwidth of a link to which a physical node is connected is greater, the likelihood of a successful deployment of a microservice is greater. This study uses $Av_{BW}(n^{env})$ to denote this feature, which is computed as shown in Equation (16).

$$Av_{BW}(n^{env}) = \sum_{l^{env} \in L_{n^{env}}} Band(l^{env}) \tag{16}$$

- **Connectivity of nodes:** The more connected a physical node is to other nodes, the more optional the communication links between deployed microservices are, and the higher the probability is of implementing microservice placement using a multi-hop format. This study uses the degree $De(n^{env})$ of a physical node to denote the connectivity of a node.
- **Security level of nodes:** Deploying microservices in physical nodes that meet the security level is the only way to reduce the possibility of attacks and thefts of the sensitive information they carry, so the security level of the physical node is an important factor that affects the secure placement of microservices. This study denotes security level by $Sec(n^{env})$.

Based on this, the feature information of each physical node is aggregated into a feature matrix, as shown in Equation (17). During training, the intelligences simply extract this feature matrix directly from the network environment.

$$\mathbf{M} = \begin{bmatrix} F_{n_1^{env}} \\ F_{n_2^{env}} \\ \vdots \\ F_{n_{|N^{env}|}^{env}} \end{bmatrix} = \begin{bmatrix} Remain(n_1^{env}) & Av_{BW}(n_1^{env}) & De(n_1^{env}) & Sec(n_1^{env}) \\ Remain(n_2^{env}) & Av_{BW}(n_2^{env}) & De(n_2^{env}) & Sec(n_2^{env}) \\ \vdots & \vdots & \vdots & \vdots \\ Remain(n_{|N^{env}|}^{env}) & Av_{BW}(n_{|N^{env}|}^{env}) & De(n_{|N^{env}|}^{env}) & Sec(n_{|N^{env}|}^{env}) \end{bmatrix} \tag{17}$$

This study uses a dominance function to measure the superiority of each placement action relative to the average to help estimate the error of the value function and to aid in updating the parameters of the value function. The Q function is used to represent the expected value of the cumulative future rewards that intelligence will receive for executing its strategy, and it is expressed in Equation (18).

$$Q(s_t, a_t) = \sum_{i=0}^{\sigma-1} \gamma^i r_{t+i} + \gamma^\sigma V(s_{t+\sigma}) = r_t + \gamma V(s_{t+1}), \tag{18}$$

where s_t represents the state at step t and a_t represents the placement action chosen at state s_t. σ represents the offset of the time step used to calculate the cumulative reward, starting at the current time step t. The cumulative reward calculation takes into account the rewards at the next σ time steps. To achieve low latency updates and to improve the stability of the algorithm, σ takes the value of 1. γ is a discount factor between $[0,1]$ that weighs the importance of current and future rewards. The placement behavior of microservices is a holistic one and needs to be considered in the long run, so the discount factor in this paper takes the value of 0.9. r_{t+i} denotes the reward obtained at $t+i$ time steps. $V(s_{t+\sigma})$ denotes the expected value of the cumulative reward obtained in the future under the implementation of the microservice placement policy. This formula is subtracted from the value function $V(s_t)$ in the current state, which is determined in Equation (19), to obtain a dominance function that can measure the degree of superiority or inferiority of the action.

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{19}$$

Finally, the dominance function and the entropy $H(\pi(s_t))$ used to encourage exploration and increase the randomness of the strategy together form Equation (20) to enable strategy updating.

$$\theta = \nabla_{\theta'} \log \pi(a_t \mid s_t) A(s_t, a_t) + \beta \nabla_{\theta'} H(\pi(s_t)) \tag{20}$$

where $\nabla_{\theta'} \log \pi(a_t \mid s_t)$ is the strategy gradient, which is used to indicate how the parameters should be adjusted to increase the probability of taking a favorable action. $\nabla_{\theta'}$ is a gradient operator. The policy network uses a multilayer perceptron. An MLP is a feed-forward neural network that contains a series of linear transformations and nonlinear activation functions, components that are admittedly differentiable. $A(s_t, a_t)$ is the dominance function mentioned in Equation (19), which instructs which nodes will increase the probability of which microservices will be placed. $\beta \nabla_{\theta'} H(\pi(s_t))$ is the entropy regularization term, and $H(\pi(s_t))$ is the entropy of the policy in state s_t, which is used to encourage the policy to remain exploratory and prevent premature convergence to a locally optimal strategy.

The training pseudo-code for the algorithm is shown in Algorithm 1. The inputs to the algorithm include the microservice placement request, the physical network on which the microservice is placed, and the training parameters. Among them, the training parameters include a maximum number of training rounds of 100, a discount factor of 0.9, an update frequency of 10, a policy network learning rate of 0.005, and an evaluation network learning rate of 0.001. The two evaluation indicators introduced in Section 3.2 are used as a reward function to assist in training. At the end of the training, the output model contains weighted parameters that allow the network state to be mapped to a probability distribution of microservice placement actions. This study uses these model parameters to determine the probability of microservices' placements in each node and uses the Dijkstra algorithm to complete the placement of call links between microservices.

Algorithm 1: Training

```
1:  Input: training parameters, physical network, MSR;
2:  Output: network model;
3:  Initialization of optimizers, global networks, worker networks, training parameters;
4:  while ($epoch_{current} < epoch_{max}$) do
5:      reset $epoch\_reward$;
6:      while True do
7:          Select the placement action;
8:          Calculating the $state_{next}$, $reward_{now}$;
9:          $epoch\_reward$ += $reward_{now}$;
10:         Update work node parameters;
11:         if Arriving to update global agent rounds then
12:             Synchronizing global network parameters;
13:         end if
14:         if Placement complete then
15:             Save the best model under the current round;
16:         end if
17:         $epoch_{current} + +$;
18:         $state_{now} = state_{next}$;
19:     end while
20: end while
21: return model;
```

4. Experiments

Experiments were run on a computer with an Intel i7-11800H processor and 16GB of RAM on an NVIDIA RTX3060 graphics card with one 6G video memory. The algorithms were run in a Miniconda environment, and the code was written in Python 3.8. In order to reflect the effectiveness of the algorithm proposed in this paper, the simulation experiments are chosen to be carried out in the data from a real environment. This study uses the network topology data provided by SNDlib [43] for the experiments. SNDlib is a database for telecommunication network design created by the Warsaw University of Technology in 2005, which aims to provide researchers with data about real telecommunication networks to facilitate the optimization and operation of telecommunication networks and their operations. The Germany50 [43] dataset this study used is one of the real classical network topologies provided by SNDlib. The instance contains 50 nodes and 88 edges. Since it does not contain data on security per se, this study only uses its topology. The range of experimental parameter settings was generated based on the real parameters as well as the joint parameter settings of the studies [38,44,45]. This study sets the computational resources and link bandwidths of each physical node to conform to a normal distribution in the interval $[50, 100]$ with 1000 service microservice placement requests. Moreover, this study sets the microservice placement request arrival frequency to 5 per 100 time units. Each placement request contains microservices that obey a uniform distribution within $[2, 6]$. In addition, regarding the setting of security parameters and security requirements, this study sets the security index of each physical node to be uniformly distributed between $[10, 20]$. Specific experimental and training parameters are demonstrated in Table 1.

Table 1. Parameter settings.

Types	Description	Value
Training parameters	training rounds	100
	discount factor	0.9
	update frequency	10
	policy network learning rate	0.005
	evaluation network learning rate	0.001
	offset of the time step	0
Experimental parameters	number of nodes	50
	number of edges	88
	compute resource	$[50, 100]$
	link bandwidth	$[50, 100]$
	number of microservices included in a single microservice request	$[2, 6]$
	security level	$[10, 20]$
	security requirements of microservices	$[5, 20]$
	request arrival frequency	5
	service microservice placement requests	1000

This study documents in Figure 3 the changes in the request acceptance rate and the fit of the security requirements of the algorithm during the training process. In the pre-training phase, the algorithm is in an exploratory phase and underperforms on the two evaluation indicators because it has not learned a good microservice placement strategy. As the training proceeds, the algorithm improves its performance on the evaluation indicators based on the feedback of the reward changes. As the training proceeds, the changes in the evaluation indicators gradually slow down, and the algorithm gradually reaches a converged state. Thus, Figure 3 demonstrates that the performance of the algorithm proposed in this paper can be significantly improved with more training iterations and also demonstrates the convergence of the algorithm.

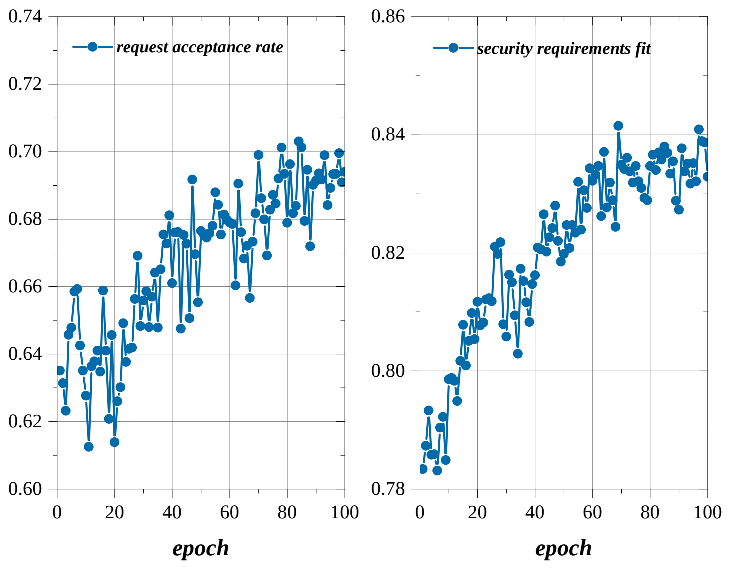

Figure 3. The changes in request acceptance rate and fit of security requirements during the training process.

Since the microservice placement process studied in this paper can be abstracted as a network embedding problem, three state-of-the-art algorithms in the field of network embedding, namely NodeRank, PSM, and CS-RL-VNE, are chosen as the comparison methods in order to prove the effectiveness of the proposed algorithms in this paper. The basic principle of the comparison algorithm is described as follows:

1. **PSM [44]:** This is a classical embedding algorithm and there is a large body of literature using it as a baseline algorithm. The basic idea of this algorithm is to construct a subset of deployable nodes for each of the microservices to be deployed using the greedy conceptm and then, for the available subset, select the node with the greatest available resources as the deployment node.
2. **NodeRank [38]:** The NodeRank method calculates the physical node rank based on Markov random wandering. This method uses the amount of resources of a node as a measure of its deployment priority, ranks the nodes based on the resource and topological attributes of the network nodes, and deploys the microservices in the nodes with the largest amount of remaining resources.
3. **CSS-RL-VNE [45]:** CSS-RL-VNE uses traditional RL methods as a means to improve the algorithm's performance. The important attribute features of the underlying network are extracted for training by involving the policy network; the deployment possibilities are ranked based on the training results, and the physical node with the highest probability of successful deployment is selected as the deployment node.

In addition, to ensure the fairness of the experiments, all the algorithms are run in network environments with the same parameters, and for link deployment between microservices, this study uniformly uses the breadth-first search algorithm. Figures 4 and 5 show the performances of the proposed algorithm on two key metrics: placement request acceptance rate and user security requirement fit, respectively.

As can be seen in Figure 4, algorithm generally outperforms the other two classical placement algorithms in terms of request acceptance rate. This is because this work includes consideration of microservice placement request acceptance rates in the reward during reinforcement learning training, and asynchronous-based reinforcement learning can interact extensively with the environment and learn towards a higher reward. In addition, to accommodate the other three comparison algorithms, this study only calculates the fit of successful placement requests.

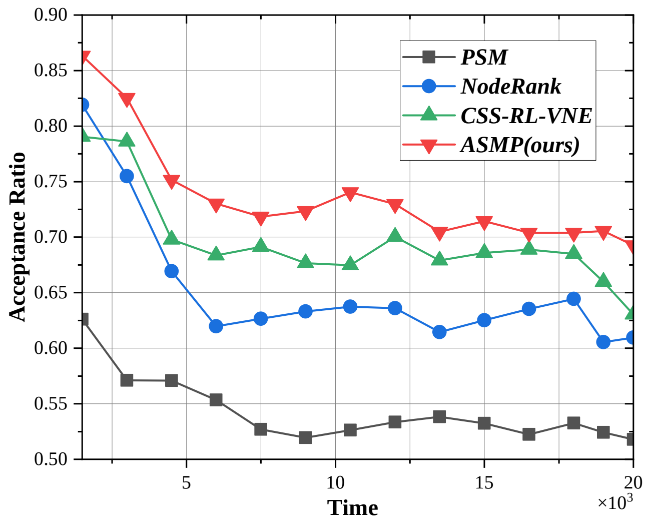

Figure 4. Comparison of request acceptance rates.

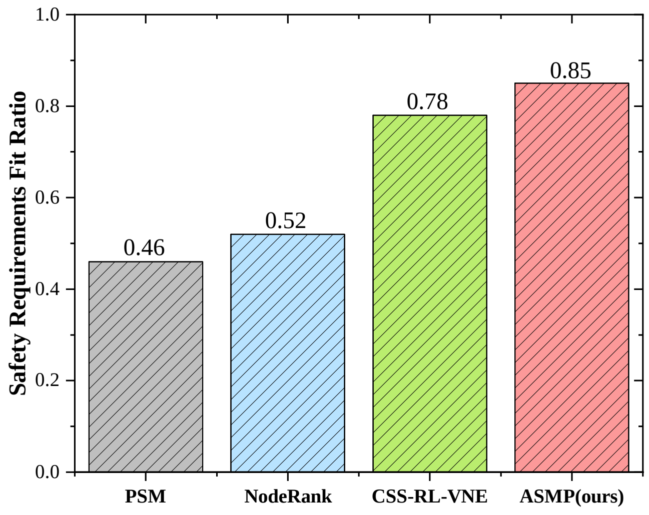

Figure 5. Comparison of fit to security requirements.

As can be seen in the right-hand side of Figure 5, the proposed algorithm also performs better than the other three algorithms in terms of the fit to the user's security needs, thanks to the security considerations this study has included in the algorithm. Among the comparison algorithms, PSM and NodeRank show very low levels of fit because they do not take security into account. The reason why the CSS-RL-VNE algorithm performs better than PSM and NodeRank is because it takes security into account. However, compared to the ASMP algorithm proposed in this paper, it does not consider the fit to the user's security requirements, which leads to excess performance even if the minimum security requirement is met. For example, if a microservice only needs to be deployed into a node with security level 3 to satisfy its minimum security requirement, the CSS-RL-VNE algorithm may place it into a physical node with security level 8 to satisfy its security requirement. In this process, there is an excess of resources. Unlike the CSS-RL-VNE algorithm, the proposed algorithm is designed with fit to user security requirement in mind. The algorithm proposed in this paper achieves a security requirement fit of 0.85 in the Gemmery50 network topology. This means that on the basis of satisfying the basic security requirements of the microservices, the chosen security level does not exceed the microservices' needs too much and thus reduces the redundancy of security resources. As shown in the experimental results, there are two main reasons why the algorithm does not fully achieve a 100% fit to user security requirements. First, since the primary goal is to ensure the normal deployment and operation of the application, the algorithm will first meet the basic security requirements and resource requirements to ensure placement with the greatest success rate and then will consider the degree of fit to security requirements, which inevitably will in the microservices being deployed in a node that exceeds the security requirements. Secondly, the distribution of security levels in the Gemmery50 nodes is not regular, and it is difficult to make the sum of the number of nodes in each security level exactly equal to the security requirements of the applications. Moreover, normal development may involve business expansion and security level upgrading, so leaving an appropriate security margin can reduce the costs of future maintenance.

Comprehensive experimental results show that the proposed algorithm has 5.9% and 8.97% better performance over the latest algorithms in terms of request acceptance rate and fit to user's security requirements, respectively, and that it ensures security without wasting security resources.

5. Conclusions

Compared to traditional monolithic applications, microservice architecture still faces many challenges, although it can be used to split complex applications into small and autonomous sub-modules, providing greater flexibility and scalability for business-level expansion. On the one hand, the splitting of services and the increase in network size causes an increase in placement complexity. On the other hand, after splitting the services, the number of data transfers and the exposure time increases, bringing about great security risks. Ensuring high security for all nodes requires greater cost overheads. This paper reduces the operational costs and reduces the redundancy of security resources for service providers using an ingenious approach. This study designs an asynchronous reinforcement learning algorithm for differentiated security requirements, which uses the parallel training of multiple intelligences that interact with the microservice placement environment in parallel, and introduces a policy network for the selection of placement nodes, which ensures the acceptance rate of the placement requests and reduces the redundancy of the security resources based on the basic security requirements of the microservices. The experimental results show the effectiveness of this proposed algorithm.

In future studies, researchers should try to consider different scenarios and larger-scale network environments. In addition, future research could also consider starting from the intruder's point of view, and constructing a multi-layered and all-round security protection system for aspects such as the transport layer encryption, authentication, and security mechanism of the API Gateway covered by the microservice security protocols. Finally, subsequent research can also consider the security of the link into the algorithm as well.

Author Contributions: Conceptualization, X.Z. and J.L.; methodology, Y.L.; software, P.Z.; validation, Y.L. and P.Z.; formal analysis, X.Z.; investigation, Y.B.; resources, J.L.; data curation, Y.L.; writing—original draft preparation, J.L. and Y.L.; writing—review and editing, J.L.; visualization, Y.B.; supervision, J.L.; project administration, X.Z.; funding acquisition, P.Z. and Y.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work is partially supported by the China University Industry-University-Research Innovation Funding under Grant 2021FNB02001 and is partially supported by the Natural Science Foundation of Shandong Province under Grants ZR2023LZH017 and ZR2022LZH015.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy.

Conflicts of Interest: Author Yanxian Bi was employed by the company China Academy of Electronic and Information Technology, CETC Academy of Electronics and Information Technology Group Co., Ltd. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Wu, Q.; Wang, W.; Fan, P.; Fan, Q.; Wang, J.; Letaief, K.B. URLLC-Awared Resource Allocation for Heterogeneous Vehicular Edge Computing. *IEEE Trans. Veh. Technol.* **2024**, 1–16. [CrossRef]
2. Qiong, W.; Shuai, S.; Ziyang, W.; Qiang, F.; Pingyi, F.; Cui, Z. Towards V2I Age-Aware Fairness Access: A DQN Based Intelligent Vehicular Node Training and Test Method. *Chin. J. Electron.* **2023**, *32*, 1230–1244. [CrossRef]
3. Zhang, P.; Gan, P.; Kumar, N.; Hsu, C.H.; Shen, S.; Li, S. RKD-VNE: Virtual network embedding algorithm assisted by resource knowledge description and deep reinforcement learning in IIoT scenario. *Future Gener. Comput. Syst.* **2022**, *135*, 426–437. [CrossRef]
4. Pallewatta, S.; Kostakos, V.; Buyya, R. Placement of Microservices-Based IoT Applications in Fog Computing: A Taxonomy and Future Directions. *ACM Comput. Surv.* 2023, *just accepted*. [CrossRef]
5. Zhang, P.; Pang, X.; Kumar, N.; Aujla, G.S.; Cao, H. A Reliable Data-Transmission Mechanism Using Blockchain in Edge Computing Scenarios. *IEEE Internet Things J.* **2022**, *9*, 14228–14236. [CrossRef]
6. Pallewatta, S.; Kostakos, V.; Buyya, R. QoS-aware placement of microservices-based IoT applications in Fog computing environments. *Future Gener. Comput. Syst.* **2022**, *131*, 121–136. [CrossRef]
7. Zeb, S.; Rathore, M.A.; Hassan, S.A.; Raza, S.; Dev, K.; Fortino, G. Toward AI-Enabled NextG Networks with Edge Intelligence-Assisted Microservice Orchestration. *IEEE Wirel. Commun.* **2023**, *30*, 148–156. [CrossRef]
8. Wang, Y.; Tian, Y.; Hei, X.; Zhu, L.; Ji, W. A Novel IoV Block-Streaming Service Awareness and Trusted Verification Scheme in 6G. *IEEE Trans. Veh. Technol.* **2021**, *70*, 5197–5210. [CrossRef]
9. Alvarenga, L.D.C.; Sousa, P.; Costa, A. Allocation and migration of microservices in SDN-based vehicular fog networks. In Proceedings of the 2022 17th Iberian Conference on Information Systems and Technologies (CISTI), Madrid, Spain, 22–25 June 2022; pp. 1–4. [CrossRef]
10. Dong, L.; Gao, H.; Wu, W.; Gong, Q.; Dechasa, N.C.; Liu, Y. Dependence-Aware Edge Intelligent Function Offloading for 6G-Based IoV. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 2265–2274. [CrossRef]
11. Wu, Q.; Wang, S.; Ge, H.; Fan, P.; Fan, Q.; Letaief, K.B. Delay-Sensitive Task Offloading in Vehicular Fog Computing-Assisted Platoons. *IEEE Trans. Netw. Serv. Manag.* **2023**, *21*, 2012–2026. [CrossRef]
12. Wang, L.; Deng, X.; Gui, J.; Chen, X.; Wan, S. Microservice-Oriented Service Placement for Mobile Edge Computing in Sustainable Internet of Vehicles. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 10012–10026. [CrossRef]
13. Wang, C.; Jiang, C.; Wang, J.; Shen, S.; Guo, S.; Zhang, P. Blockchain-Aided Network Resource Orchestration in Intelligent Internet of Things. *IEEE Internet Things J.* **2023**, *10*, 6151–6163. [CrossRef]
14. Zhang, P.; Wang, Y.; Aujla, G.S.; Jindal, A.; Al-Otaibi, Y.D. A Blockchain-Based Authentication Scheme and Secure Architecture for IoT-Enabled Maritime Transportation Systems. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 2322–2331. [CrossRef]
15. Wang, Y.; Zhao, C.; Yang, S.; Ren, X.; Wang, L.; Zhao, P.; Yang, X. MPCSM: Microservice Placement for Edge-Cloud Collaborative Smart Manufacturing. *IEEE Trans. Ind. Informatics* **2021**, *17*, 5898–5908. [CrossRef]
16. Siddiqui, H.; Khendek, F.; Toeroe, M. Microservices based architectures for IoT systems-State-of-the-art review. *Internet Things* **2023**, *23*, 100854. [CrossRef]
17. Ray, K.; Banerjee, A.; Narendra, N.C. Proactive Microservice Placement and Migration for Mobile Edge Computing. In Proceedings of the 2020 IEEE/ACM Symposium on Edge Computing (SEC), San Jose, CA, USA, 12–14 November 2020; pp. 28–41. [CrossRef]
18. Zdun, U.; Queval, P.J.; Simhandl, G.; Scandariato, R.; Chakravarty, S.; Jelic, M.; Jovanovic, A. Microservice Security Metrics for Secure Communication, Identity Management, and Observability. *ACM Trans. Softw. Eng. Methodol.* **2023**, *32*, 1–34. [CrossRef]
19. He, X.; Tu, Z.; Wagner, M.; Xu, X.; Wang, Z. Online Deployment Algorithms for Microservice Systems with Complex Dependencies. *IEEE Trans. Cloud Comput.* **2023**, *11*, 1746–1763. [CrossRef]
20. Bahreini, T.; Grosu, D. Efficient Algorithms for Multi-Component Application Placement in Mobile Edge Computing. *IEEE Trans. Cloud Comput.* **2022**, *10*, 2550–2563. [CrossRef]
21. Varasteh, A.; Hofmann, S.; Deric, N.; He, M.; Schupke, D.; Kellerer, W.; Machuca, C.M. Mobility-Aware Joint Service Placement and Routing in Space-Air-Ground Integrated Networks. In Proceedings of the ICC 2019–2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; pp. 1–7. [CrossRef]
22. Skarlat, O.; Nardelli, M.; Schulte, S.; Dustdar, S. Towards QoS-Aware Fog Service Placement. In Proceedings of the 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), Madrid, Spain, 14–15 May 2017; pp. 89–96. [CrossRef]
23. Chen, F.; Zhou, J.; Xia, X.; Jin, H.; He, Q. Optimal Application Deployment in Mobile Edge Computing Environment. In Proceedings of the 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), Beijing, China, 19–23 October 2020; pp. 184–192. [CrossRef]
24. Yu, Y.; Yang, J.; Guo, C.; Zheng, H.; He, J. Joint optimization of service request routing and instance placement in the microservice system. *J. Netw. Comput. Appl.* **2019**, *147*, 102441. [CrossRef]
25. Fu, K.; Zhang, W.; Chen, Q.; Zeng, D.; Peng, X.; Zheng, W.; Guo, M. QoS-Aware and Resource Efficient Microservice Deployment in Cloud-Edge Continuum. In Proceedings of the 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Portland, OR, USA, 17–21 May 2021; pp. 932–941. [CrossRef]
26. Chen, L.; Xu, Y.; Lu, Z.; Wu, J.; Gai, K.; Hung, P.C.K.; Qiu, M. IoT Microservice Deployment in Edge-Cloud Hybrid Environment Using Reinforcement Learning. *IEEE Internet Things J.* **2021**, *8*, 12610–12622. [CrossRef]

27. Lv, W.; Wang, Q.; Yang, P.; Ding, Y.; Yi, B.; Wang, Z.; Lin, C. Microservice Deployment in Edge Computing Based on Deep Q Learning. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 2968–2978. [CrossRef]
28. Wu, Q.; Zhao, Y.; Fan, Q.; Fan, P.; Wang, J.; Zhang, C. Mobility-Aware Cooperative Caching in Vehicular Edge Computing Based on Asynchronous Federated and Deep Reinforcement Learning. *IEEE J. Sel. Top. Signal Process.* **2023**, *17*, 66–81. [CrossRef]
29. Ding, Z.; Wang, S.; Jiang, C. Kubernetes-Oriented Microservice Placement with Dynamic Resource Allocation. *IEEE Trans. Cloud Comput.* **2023**, *11*, 1777–1793. [CrossRef]
30. Wu, C.; Peng, Q.; Xia, Y.; Jin, Y.; Hu, Z. Towards cost-effective and robust AI microservice deployment in edge computing environments. *Future Gener. Comput. Syst.* **2023**, *141*, 129–142. [CrossRef]
31. Gopal, H.; Song, G.; Zhu, T. Security, Privacy and Challenges in Microservices Architecture and Cloud Computing-Survey. *arXiv* **2022**, arXiv:2212.14422.
32. Hossain, M.D.; Sultana, T.; Akhter, S.; Hossain, M.I.; Thu, N.T.; Huynh, L.N.; Lee, G.W.; Huh, E.N. The role of microservice approach in edge computing: Opportunities, challenges, and research directions. *ICT Express* **2023**, *9*, 1162–1182. [CrossRef]
33. Jin, H.; Li, Z.; Zou, D.; Yuan, B. DSEOM: A Framework for Dynamic Security Evaluation and Optimization of MTD in Container-Based Cloud. *IEEE Trans. Dependable Secur. Comput.* **2021**, *18*, 1125–1136. [CrossRef]
34. Sairam, R.; Bhunia, S.S.; Thangavelu, V.; Gurusamy, M. NETRA: Enhancing IoT Security Using NFV-Based Edge Traffic Analysis. *IEEE Sens. J.* **2019**, *19*, 4660–4671. [CrossRef]
35. Zdun, U.; Queval, P.J.; Simhandl, G.; Scandariato, R.; Chakravarty, S.; Jelić, M.; Jovanović, A. Detection Strategies for Microservice Security Tactics. *IEEE Trans. Dependable Secur. Comput.* **2023**, 1–17. [CrossRef]
36. Jin, W.; Xu, R.; You, T.; Hong, Y.G.; Kim, D. Secure Edge Computing Management Based on Independent Microservices Providers for Gateway-Centric IoT Networks. *IEEE Access* **2020**, *8*, 187975–187990. [CrossRef]
37. Tamim, I.; Jammal, M.; Hawilo, H.; Shami, A. Introducing Virtual Security Functions into Latency-aware Placement for NFV Applications. In Proceedings of the ICC 2020–2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; pp. 1–7. [CrossRef]
38. Cheng, X.; Su, S.; Zhang, Z.; Wang, H.; Yang, F.; Luo, Y.; Wang, J. Virtual Network Embedding through Topology-Aware Node Ranking. *SIGCOMM Comput. Commun. Rev.* **2011**, *41*, 38–47. [CrossRef]
39. Selimi, M.; Cerdà-Alabern, L.; Sánchez-Artigas, M.; Freitag, F.; Veiga, L. Practical Service Placement Approach for Microservices Architecture. In Proceedings of the 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), Madrid, Spain, 14–17 May 2017; pp. 401–410. [CrossRef]
40. Gu, L.; Chen, Z.; Xu, H.; Zeng, D.; Li, B.; Jin, H. Layer-aware Collaborative Microservice Deployment toward Maximal Edge Throughput. In Proceedings of the IEEE INFOCOM 2022—IEEE Conference on Computer Communications, London, UK, 2–5 May 2022; pp. 71–79. [CrossRef]
41. Li, H.; Tang, B.; Xu, W.; Guo, F.; Zhang, X. Application Deployment in Mobile Edge Computing Environment Based on Microservice Chain. In Proceedings of the 2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Hangzhou, China, 4–6 May 2022; pp. 531–536. [CrossRef]
42. Su, X.; Tolba, A.; Lu, Y.; Tan, L.; Wang, J.; Zhang, P. An Attention Mechanism-Based Microservice Placement Scheme for On-Star Edge Computing Nodes. *IEEE Access* **2023**, *11*, 114341–114351. [CrossRef]
43. Orlowski, S.; Pióro, M.; Tomaszewski, A.; Wessäly, R. SNDlib 1.0–Survivable Network Design Library. *Networks* **2010**, *55*, 276–286. [CrossRef]
44. Yu, M.; Yi, Y.; Rexford, J.; Chiang, M. Rethinking virtual network embedding: Substrate support for path splitting and migration. *SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 17–29. [CrossRef]
45. Zhang, P.; Wang, C.; Jiang, C.; Kumar, N.; Lu, Q. Resource Management and Security Scheme of ICPSs and IoT Based on VNE Algorithm. *IEEE Internet Things J.* **2022**, *9*, 22071–22080. [CrossRef]