Received 15 February 2023, accepted 10 March 2023, date of publication 21 March 2023, date of current version 27 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3260147

SURVEY

A Survey on Microservices Trust Models for Open Systems

ZHONGYI LU[1], DECLAN T. DELANEY[2], AND DAVID LILLIS[1], (Senior Member, IEEE)
[1]School of Computer Science, University College Dublin, Dublin 4, D04 V1W8 Ireland
[2]School of Electrical and Electronic Engineering, University College Dublin, Dublin 4, D04 V1W8 Ireland

Corresponding author: David Lillis (david.lillis@ucd.ie)

This research is funded under the Science Foundation Ireland (SFI) Strategic Partnerships Programme (16/SPP/3296) and is co-funded by Origin Enterprises plc.

ABSTRACT The microservices architecture (MSA) is a form of distributed systems architecture that has been widely adopted in large-scale software systems in recent years. As with other distributed system architectures, one of the challenges that MSA faces is establishing trust between the microservices, particularly in the context of open systems. The boundaries of open systems are unlimited and unknown, which means that they can be applied to any use case. Microservices can leave or join an open system arbitrarily, without restriction as to ownership or origin, and MSA systems can scale extensively. The organisation of microservices (in terms of the roles they play and the communication links they utilise) can also change in response to changes in the environment in which the system is situated. The management of trust within MSAs is of great importance as the concept of trust is critical to microservices communication, and the operation of an open MSA system is highly reliant on communication between these fine-grained microservices. Thus, a trust model should also be able to manage trust in an open environment. Current trust management solutions, however, are often domain-specific and many are not specifically tailored towards the open system model. This motivates research on trust management in the context of open MSA systems. In this paper, we examine existing microservices trust models, identify the limitations of these models in the context of the principles of open microservices systems, propose a set of qualities for open microservices trust models that emerge from these limitations, and assess selected microservices trust models using the proposed qualities.

INDEX TERMS Microservices, trust management, service oriented architecture, open systems.

I. INTRODUCTION

As the complexity of enterprise applications has grown significantly, monolithic systems have long been considered unfit for modern systems. Service-Oriented Architecture (SOA), was introduced to address industrial demands for large enterprise systems [1] and has gained popularity over the past two decades [2]. SOA is a software architecture that decomposes systems into sets of reusable software components: services, to support software user requirements [3]. Web services, which are applications accessible to other applications over the Web [4], have become a way to implement SOA [3]. It has been identified, however, that Web services cannot support all the principles of SOA [2]. In 2014, an extended notion of SOA, the Microservices Architecture (MSA), was introduced [5], and has gained popularity in the construction of enterprise applications ever since.

The main strategy of microservices is decomposing large monolithic applications into many small components that communicate with each other using lightweight mechanisms, often HTTP resource APIs [6]. Large companies like Netflix, the Guardian, Uber, Etsy, and Amazon use MSA to deliver their services [7].

The importance of trust management has been stressed in previous research. To protect MSA systems from attacks coming from malicious microservices, only trusted microservices can be incorporated into the system [8].

The associate editor coordinating the review of this manuscript and approving it for publication was Claudio Agostino Ardagna.

Many competing definitions of "trust" have been advanced in the literature. In this paper, we combine the definitions of trust given in [9] and [10]. Dasgupta [9] defined a trustworthy system as as one that "will do what it says and only what it says". Truong et al. [10] defined trust as "a belief of a truster in a trustee that the trustee will provide or accomplish a trust goal as truster's expectation within a specific context for a specific period of time". Therefore, for this research, trust is ***A belief of a truster in a trustee that the trustee will provide or accomplish the services that it says it will provide and meet the expectations of the truster within a specific context for a specific period of time***.

As microservices should provide open descriptions of their APIs, GUIs and communication message formats [11], some companies and researchers have focused on the use of MSA to build open systems [12], [13]. Open systems exist in contrast to closed systems. Closed systems are often designed for a specific task [14]. On the contrary, open systems can react to environments that are changing continuously, unlimitedly, and unanticipatedly [15]: this could be the components inside the systems, the organisation of components, the services offered by the system, the use cases of the systems, etc. Inside open microservices systems, microservice instances can be designed and implemented in different styles, be self-controlled, and may join or leave the system casually without restriction as to ownership or origin [16].

Trust models can protect systems from attacks and threats that will make the system vulnerable and endangered [17]. Current trust management solutions, however, are often business-specific and focus on either defending against specific attacks or identifying and/or isolating malicious nodes [18], and as such can be considered as assuming a closed system. Questions remain about their ability to cope with the open-world settings and the highly dynamic environment of open microservices systems. Previous research, such as [18], has discussed the weaknesses of centralised-controlled microservices trust models in open environments where one controller is responsible for managing trust for all the microservices in the system: i) the system will be compromised if there is only one controller that manages trust for the system and that controller itself becomes compromised; ii) it is difficult to choose a single controller that is approved by all the components; and iii) the arbitrary joining and leaving of microservices brings difficulties to the system management that the controller needs to perform. This type of analysis motivates our work on trust management in open microservices systems to answer a number of research questions. To begin with, to what extent can existing microservices trust models, usually designed for a specific area, fully support trust management within open systems? If they cannot, what are the desirable qualities for effective open microservices trust models? Finally, what is a solution that can fully support the trust management of open microservices systems?

The primary contributions of this paper are:

- Reviewing the state of the art in microservices trust management;
- Proposing a novel classification for microservices trust models;
- Identifying the limitations of existing microservices trust models under the setting of open environments;
- Proposing a set of qualities that microservices trust models should have, in order to be applicable to open systems; and
- Examining the characteristics of existing microservices trust models using the proposed qualities.

The rest of the paper is organised as follows: Section II gives a literature review on microservices trust management. Section III identifies limitations of existing trust models when being applied within open systems. The desirable qualities that open microservices trust models require is proposed in Section IV. The proposed qualities are then applied to the selected existing trust models in Section V. Finally, the conclusion and future work are stated in Section VI.

II. LITERATURE REVIEW

There has been a long-standing trend in the software engineering industry to decompose large-scale systems into smaller components in order to promote ease of development and ease of deployment. Widespread industry adoption of Object Oriented Programming is a clear example of this trend. Other approaches have been proposed to greater or lesser degrees of adoption, for instance Agent Oriented Programming [19], Aspect Oriented Programming [20], Web Services [21], Component-Based Systems [22] and combinations of these [23], [24].

In recent times, Service Oriented Architectures (SOA), specifically those based on Microservices Architectures (MSA) have come to dominate the landscape for modern large-scale software systems. The following sections outline these architectural approaches, along with the consequent considerations in the context of open systems, and the implications for trust management.

A. SERVICE-ORIENTED ARCHITECTURE

Service Oriented Architecture (SOA) is a paradigm for organising and encapsulating pieces of functionality as individual services, making them available over a network to be accessed through defined interfaces, and integrating them into business solutions [25]. According to the definition given by Erl [26], services inside an application should: i) share the same contract design standard; ii) be loosely coupled; iii) only share essential information; iv) be reusable; v) be autonomous; vi) be stateless; vii) be discoverable; and viii) be composable.

Historically, one of the most common ways to implement SOA applications has been Web Services. According to the definition given by W3C Web Services Architecture Working Group in 2004, "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards'' [27].

B. MICROSERVICES ARCHITECTURE

The Microservices Architecture (MSA) represents the state of the art in current large-scale software development [28]. It is an extended notion of Service-Oriented Architectures (SOA) [5]. Microservice instances should be fine-grained, and should be ''single-responsibility units'' [6]. In academia, there has not yet been a consensus as to the tenets of microservices. There are some tenets, however, that most researchers tend to agree on, such as bounded context, small size, service independence, scalability, isolated state, elasticity, autonomy, and loose coupling [5], [6], [29], [30], [31].

C. PRINCIPLES OF OPEN MICROSERVICES SYSTEMS

Open systems can be described as systems that run in open environments. The term ''open'' implies that [16]:

- The components of the system (i.e., microservice instances) should be autonomous. In other words, the instances should function under their own control and governance.
- The components of the system can be heterogeneous, which means that the instances within the system can be designed, constructed, or implemented in different manners and languages.
- The environment can be dynamic: microservices can behave arbitrarily, and they may join or leave the system randomly.

Therefore, compared to microservices systems that are built for a specific area with finite components and are less dynamic, building open microservices systems requires additional due diligence in terms of autonomy, interoperability, scalability, elasticity, loose coupling, service independence, isolated states, granularity, and robustness.

D. MICROSERVICES TRUST MODELS

Microservices security is vital, especially when many IT companies are delivering their businesses with microservices [32]. The establishment of trust between individual microservices is one of the most critical security problems [10]. Inside a microservices system, if a microservice was attacked and controlled by a malicious actor, it would not only compromise itself but also have the potential to cause the entire system to fail. Therefore, microservices systems need a trust model to monitor the connection and establish trust between individual microservices in order to limit the potential for trust-related attacks [33] and system damage [29].

The MSA is an extension of SOA and has become a preferred solution for enterprises compared to SOAs based on Web Services [2]. Web Services, however, have a longer history, and the literature related to trust in Web Services architectures is consequently richer. It is therefore instructive to learn from previous Web Services trust models. In this section, some Web Services trust models will also be introduced, with a view to analysing the extent to which their characteristics are applicable to MSAs.

As microservices trust management is a relatively new area, there has yet to be a systematic method for trust model classification. However, categorising trust models into different classes facilitates researchers to know more about the commonalities and differences between them. This also facilitates the examination of the performance of existing trust models in open environments. A contribution of this paper, therefore, is to propose four categories within which existing microservices trust models may be grouped: Zero-trust-based, Socio-based, Composition-based, and Control-based. These are defined as follows:

- **Zero-trust-based**: Zero-trust-based models assume that microservices inside the systems are always hostile. They are designed based on zero-trust networks, in which all participants are similarly assumed to be hostile. Both external and internal threats exist on the network at all times; trust in a network cannot be decided merely by network locality; all the actors (including devices, users, and microservices) and their every action must be authenticated and authorised; and policies must be dynamic and calculated using multiple sources of data [34].
- **Socio-based**: Socio-based trust models regard a MSA system as a society, and each microservice within the system is a member of that society. The trust score of a microservice is a reflection of how other microservices inside the system think about this microservice based on their own behaviour.
- **Composition-based**: Composition-based trust models see a microservice as a member of a ''solution'': a group of microservices that form an overall service that is provided to trusters to satisfy their requests. After a solution has handled the request from the truster, it will be rated. Based on a decomposition algorithm that the trust model uses, each microservice inside the solution will be assigned a trust score according to the overall trust score of the solution.
- **Control-based**: Control-based trust models manage trust authorisation or trust-based microservices selection using the same logic as control models. To prevent a microservice from being abused by other (compromised) microservices, a microservices system needs to take control of inter-service access or the traffic (information flow) within the system. The approaches used to take over control of access or traffic are called control models [35]. Control models facilitate building trust in a way that is decentralised and secure.

The following sections discuss specific examples of trust models that fall under these four identified categories.

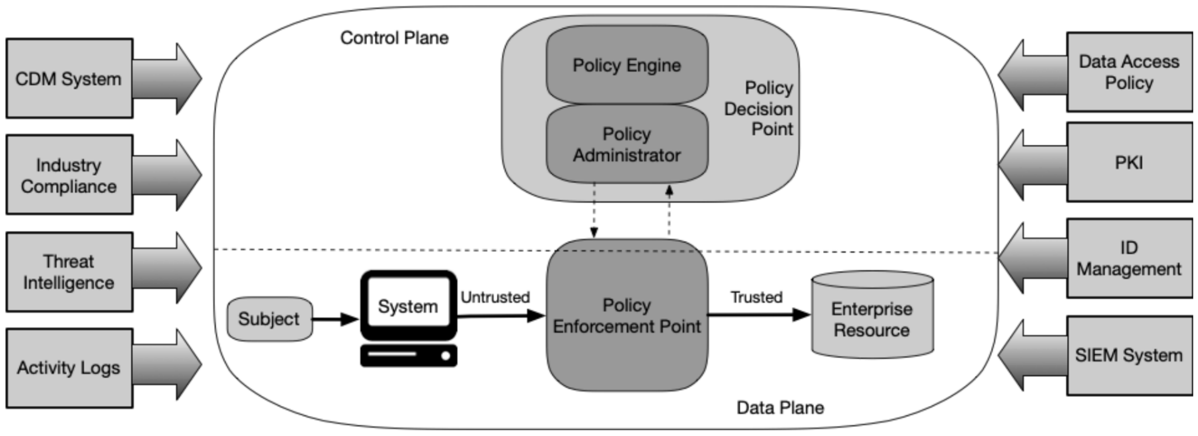

FIGURE 1. **A simple structure of zero-trust architecture (from [37]).**

1) ZERO-TRUST-BASED

In the context of MSA, adopting zero-trust models means that each request must be authenticated and verified at the entry point. All service-to-service interactions must be monitored to support trust authorisation [36].

Figure 1 illustrates a simple structure of a zero-trust architecture. This architecture contains five main components: Subject, Resource, Policy Decision Point, Policy Enforcement Point and Supplement.

In this structure, a Subject is a user or microservice requesting access to enterprise resources. A Resource, as in the context of MSA, can be either a piece of data or another microservice. A Policy Decision Point is responsible for deciding whether trust can be established between a Subject and the Resource it requests. This consists of two minor components: Policy Engine and Policy Administrator, which are responsible for decision-making and communication management. When a subject intends to access a resource, it will send its request to the Policy Enforcement Point. This then forwards the request to the Policy Decision Point. After the decision is made, it then issues a command back the to Policy Enforcement Point to establish or refute the trust between the subject and the resource. The Policy Decision Point makes its decisions based on the Supplement. The Supplement helps to provide useful information to the Policy Engine for more accurate and correct decisions and to achieve higher system security.

DeCusatis et al. [38] described a steganographic overlay zero-trust approach, which was a combination of transport access control (TAC) and first-packet authentication. Every network session was authenticated independently at the transport layer before any further actions. Explicit trust was established by authenticating network identity tokens on the first packet of a TCP connection and applying a security policy. The adoption of gateways assured that any response to unauthorised packets would be blocked, and only trustworthy users could get access to the protected resources.

Zaheer et al. [39] proposed eZTrust, a network-independent perimeterisation approach for microservices. Perimeterisation is an approach based on building a firewall for the system [40]. eZTrust shifted perimeterisation targets from network endpoints to workload identities that are defined as a set of authentic contexts tied to the microservice workload. Authentic contexts are trusted contexts that are collected from trusted infrastructure or trusted software packages (e.g., that have been digitally signed or sourced from official repositories). Inside eZTrust, there is a central coordinator (orchestrator) that listens to all the events emitted by any local transaction between microservices and triggers the next local transaction in a different microservice based on the incoming event [41]. When the orchestrator receives a packet, it will decode the sender-side contexts and perimeterise them to the recipient-side context to authorise the interaction in a purely network-independent fashion.

In any discussion of MSA technologies, it is important to consider their use in industry. Companies tend to use technologies such as Google Kubernetes Engine (GKE)[1] and Istio[2] to develop their MSA systems. To the extent to which industry has implemented trust models in their systems, they have tended to adopt a zero trust approach, based on the principles outlined above.

GKE provides a managed environment to deploy, manage, and scale containerised applications using Google infrastructure. Istio is a Service Mesh, consisting of two components: the data plane and the control plane. The data plane is the communication between services. Proxies are deployed to track and control the communication between microservices. The control plane takes the desired configuration and dynamically programs the proxy servers, updating them as the rules or the environment change. Istio offers an automatic implementation of Mutual Transport Layer Security (mTLS) between microservices, which allows connections to

[1]https://kubernetes.io/
[2]https://istio.io/

be encrypted and the services to be mutually authenticated. This can be used to enforce authorisation policies centrally managed by the Control Plane. Thus, Istio can be used as a platform upon which to build a Zero Trust Architecture [42].

Commercial service providers, such as Hashicorp,[3] Cyolo,[4] Forcepoint,[5] etc. also provide zero-trust solutions.

Industry standard technologies have also been adopted to realise zero-trust models in the academic sphere. Rodigari et al. [43] performed experiments whereby a zero trust implementation based on Istio was successfully deployed in various Kubernetes platforms, with a view to minimising latency and system resources. Scoppetta [42] upgraded a zero-trust-based system implemented with Spring Boot and Microsoft Azure with Istio. The Istio-based deployment maintained zero trust but provided greater visibility of traffic than the previous system that had been based on JSON Web Tokens.

Zero-trust models are adaptive: microservices and the users of the systems are continuously producing new information. The contexts that the systems are in and the structure of the systems can also be changing. Zero-trust models need to adjust and adapt automatically to fit the changing contexts or the influences of new information [44], e.g., the features that microservices demand. This can ensure that the trust between each microservice is up to date, which can support the dynamism of open MSA systems. However, the authentication process is often resource-intensive and will be costly, especially for open systems.

2) SOCIO-BASED

Socio-based trust models can be subdivided into two subtypes: blockchain-based and reputation-based.

a: BLOCKCHAIN-BASED

Blockchain has been incorporated into some trust models. Trust is only established between peers. Service Level Agreements (SLA) and/or Smart Contracts (SC) serve as the foundation for the expected service that the truster should receive. SLA is a type of performance-based contract. The desired performance of the trust is enforced by a determined service level [45]. The performance of the trust often refers to the expected Quality-of-Service (QoS) standards, which will further be discussed in Section III-H1 [46]. SC are computer protocols with a set of commitments that can digitally facilitate the dissemination, verification, and execution of contracts among the participants [47], [48]. These are then used to determine whether trust should be established between two microservices, combined with the actual performance of services.

Figure 2 shows an abstract architecture of blockchain-based trust models. Usually, a system using a blockchain-based trust model contains at least four layers: Application Layer, Blockchain Layer, Decision Layer, and Resource Layer. The Application Layer is the interface that receives requests from users who wish to interact with the system. The Blockchain Layer is in charge of verifying the SLA/SC of microservices. A blockchain layer allows knowledge that was gained through previous interaction between microservices to be used to evaluate the trust between microservices. The data collected in the Blockchain Layer will then be sent to the Decision Layer, where the system makes a determination as to the trustworthiness of each microservice. The Resource Layer that manages all the resources that are involved in the system (e.g., infrastructure, microservices, databases, etc.). The design of each layer may vary between different trust models.

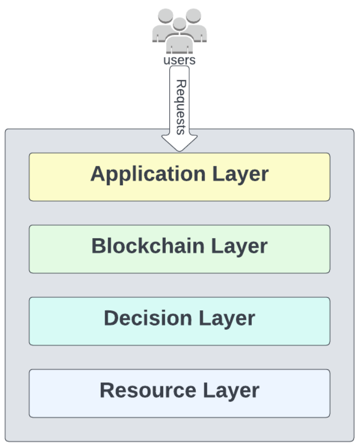

FIGURE 2. An abstract architecture of blockchain-based trust models.

Kochovski et al. [48] presented a blockchain-based trust management architecture for the DECENTER Fog Computing Platform. The system comprises four layers: Application Layer, Blockchain Layer, Edge-to-Cloud Orchestration Layer, and Decision-Making Layer. The Application Layer is the entry point for users who wish to interact with the system. The Blockchain Layer is built on Ethereum. It contains two important components: Smart Contracts and Smart Oracle. In this model, transactions can be assessed by the Smart Oracle in advance before assessing it on-blockchain. The Edge-to-Cloud Orchestration Layer contains infrastructures for the deployment of containerised microservices and data, Quality of Service (QoS) monitoring of the infrastructures and Internet of Things devices. The data received on this layer is passed to the Decision-Making Layer. The Decision-Making Layer is responsible for trustworthiness evaluation and trustee selection. It uses a Markov probabilistic decision-making method to rank the infrastructures. The infrastructure with the highest ranking will be considered a trusted infrastructure and recommended to the user.

Ruan et al. [49] introduced a multi-domain heterogeneous resource trust management architecture. This model adopts a consortium blockchain network for trusted resource sharing and transaction, while using virtual technology to mitigate the heterogeneity of multi-domain resources, with a differential

[3]https://www.hashicorp.com/solutions/zero-trust-security
[4]https://cyolo.io/zero-trust-ot/
[5]https://www.forcepoint.com/product/ztna-zero-trust-network-access

evolution algorithm to support resource scheduling. In this model, the trustworthiness of microservices was verified at the heterogeneous edge resource integration layer of the blockchain. When a new microservice joins the system, it is virtualised as node. The node needs to upload its registration information to the blockchain, including its name, location and the resources that it is willing to share. After receiving the registration request from the virtual node, the smart contract will verify the information. Verified nodes will then be considered to be trusted, and registered in the blockchain.

Blockchain-based trust models have also been developed with Web services in mind, including: Web Services Trust Ontology [50], a collaborative filtering algorithm called Trust-Based Service Recommendation [51], a collaborative Service Level Agreement and Reputation-based Trust Management solution for federated cloud environments [52], and a trust evaluation model of Web Services based on small-world networks [53].

The adoption of SC/SLA in Blockchain-based models provides the insight that in microservices systems, the trust of one microservice can be shared with all microservices. In other words, trust can be considered to be universal.

b: REPUTATION-BASED

Reputation-based models compute trust values for microservices based on the past behaviours of services and the ratings that other microservices have previously given to them.

Azarmi et al. [54] provided a solution for end-to-end security auditing in SOA (Figure 3). The proposed security architecture introduced two new components called taint analysis (a module that monitors the runtime activity of services and inspects the information flow between services) and a trust broker (a trusted third party responsible for maintaining a list of certified services, evaluating the trust level of a given service, and evaluating the appropriateness of service invocations). Microservices are labelled as "certified", "trusted", or "untrusted" according to their trust score. Microservices with the same trust label will be allocated to the same domain. The trust broker is used to return the trust level of the microservices that are selected by the user and evaluate the trust level of microservices using feedback from the taint analysis module. The taint analysis module monitors the activity of services during runtime, tracks traffic, and identifies violations. The record in the taint analysis module is passed back to the trust broker after each interaction to calculate the trust of the service.

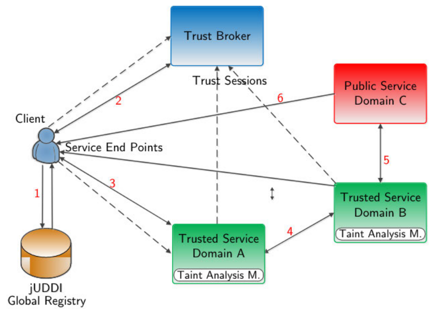

FIGURE 3. Architecture proposed by Azarmi et al. [54].

Kravari and Bassiliades [55] introduced a social agent-based trust model for the Internet of Things (IoT), adopting microservices architectures called StoRM. StoRM indicates three main types of IoT characters: entities, services, and devices. A Multi-agent system (MAS) approach was used to implement the microservices as part of a MSA. The trust between the agents is calculated based on the truster's ratings of a trustee in terms of response time, validity, correctness, cooperation, QoS, and availability. Also, LOCATOR, a locating rating mechanism that makes use of social graphs and P2P networks, was adopted to encourage agents to make trust recommendations. The more recommendations that an agent makes, the heavier the weight its ratings will be assigned when calculating an agent's trustworthiness. StoRM also took into account the dynamism of ratings; thus, it would discard outdated ratings.

In the Web Services domain, Reputation-based trust models include: a model to manage trust and reputation using Web Service Modelling Ontology in a P2P environment [56], REGRET [57], a Bayesian network trust and reputation model [58], a QoS-aware reputation-based trust model that leverages the correlation information among various QoS metrics [59], Total Trust Evaluator Framework [60], a TOPSIS (Technique for Order Preference by Similarity to an Ideal Solution) evaluation scheme for cloud service trustworthiness combining objective and subjective aspects [61], RATEWeb [62], and TRUSS [63].

Reputation-based trust models emphasise the social characteristics of open MSA systems. However, this also gives malicious actors more opportunities to attack the system or other microservices.

3) COMPOSITION-BASED

Composition-based trust models rate the trust score of the entire service that the system provides to the truster and then uses a decomposition strategy to compute the trust score of each microservice inside the overall service.

Adewuyi et al. [64] proposed SC-TRUST, which is a trust model designed especially for service compositions in the SOA-based IoT context. Instead of calculating the trust value of each microservice individually, SC-TRUST calculates their trusts using composition, aggregation, and decomposition. After receiving the request and trust criteria preference from the user (service requester), underlying services are assessed on those trust criteria, such as accuracy and response time, and partial trust scores are assigned. A composed solution is then formed according to the requirements of the request. The sum of the partial trust score should be higher than the threshold set by the user. The threshold is the minimum partial trust score that the user considers to constitute a trustworthy solution. After this interaction, the consumer rates the solution, and the overall trust score is decomposed into the trust score of each microservice using a weighted algorithm.

The approach of Azarmi [65] is that a microservice's trust value should inherit from all the other services that have direct or indirect interaction with it. To this end, two trust algorithms were introduced: Coarse-Grained Global Algorithm and Graph-Reduction Global Algorithm. Both algorithms calculate trust using graph-based composite trust schemes.

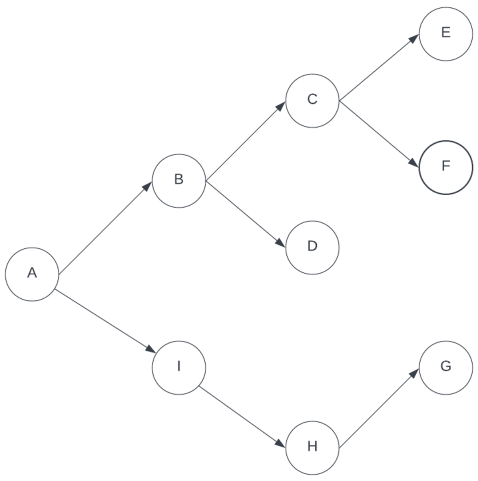

FIGURE 4. A simple graph-based composite trust architecture.

Figure 4 illustrates a simple service composition graph between microservices. To calculate a microservice's trust value using the Coarse-Grained Global Algorithm, the algorithm uses the microservice as the entry node and searches for all the other nodes (services) that can be retrieved through the arcs to create the service composition graph. For example, to compute the trust of microservice B, B is the entry point of the graph. Then the trust scores of all the nodes that can be retrieved with the arcs will be included to compute the overall trust of B (i.e., C, D, E, and F). After getting the trust values of all the relevant nodes within the graph, it calculates the trust value of the microservice based on predefined strategies. For the Graph-Reduction Global Algorithm, instead of capturing the structure of the service graph, it is based on graph abstraction. It replaces a basic topology in a graph by a node (service) with an equivalent trust value and weight, and gradually turns a composite service graph into a single node with a trust value. The calculation of the trust value of a node is the same as in the first algorithm.

Composition-based Web Services trust models include a trust evaluation method for collaborations of data-intensive services [66].

Composition-based trust models can be used to calculate trust in closed MSA systems. However, the components inside open systems can be different at any time, and the organisation of components is not fixed. Using composition to infer trust values for microservices may not be up-to-date as the organisation of components can be changed immediately after a trust value is inferred, and the form of a service graph or a trust solution may vary.

4) CONTROL-BASED

Control-based models compute or establish trust between microservices with access models or traffic control models.

Pasomsup and Limpiyakorn [67] presented a design of the HT-RBAC (Hierarchical Control Role-Based Access Control) model. The design was based on RBAC (Role-Based Access Control) models. Inside the system, each entity is assigned with a role. These roles have various hierarchies. Roles with higher hierarchies have more access to the data. Thus, in HT-RBAC, microservices with different roles have different weights when computing trust. This model also incorporates a Security Manager into the model to help authenticate, authorise, and identify a user's access control. Moreover, the flow of trust in each service is bound to the user to avoid Cross-Site Request Forgery (CSRF) or Cross-Site Scripting (XSS) attacks.

Skandylas et al. [18] proposed AT-DIFC, a DIFC (Decentralised Information Flow Control) model extended with trust and adaptation capabilities to enforce security in open distributed systems. They proposed and formalised a DIFC Model that used trust between principals to enforce information flow policies in microservices systems. They also introduced the concept of adaptive trust architectures that could be dynamically adapted to meet different stakeholder goals by breaking down the request into a group of contexts and comparing them with the contexts where microservices could be trusted (trust context). The trust of each microservice is based on the context that it is in, the number of messages that have been successfully exchanged in its actions, and its conformance with security policies. Trust contexts can be split and merged, so that the computation of trust can adapt to different requests.

Control-based models heavily rely on the control models that they are built upon. This makes the choice of control models crucial and becomes more difficult in open environments because the traffic of data will be heavier and the number of actors inside the system will be higher. The control models must be strong enough to take control of the data exchanged during all the interactions and resist the challenges that they will encounter due to the potentially unlimited expansion of open systems.

III. MICROSERVICES TRUST MODELS IN THE CONTEXT OF OPEN ENVIRONMENTS

Since existing microservices trust models tend to be domain-specific or attack-specific, it is important to examine their compatibility with open systems.

As discussed in Section II-C, open microservices systems have greater requirements in terms of autonomy, interoperability, scalability, elasticity, loose coupling, service independence, isolated state, granularity, and robustness when compared to more closed systems. Thus it is essential that trust models should also take these principles into account.

After identifying the microservices trust models mentioned in Section II-D, in this section, we examine these models, bearing in mind the characteristics of open systems, and identify a number of scenarios of note where difficulties could arise from deploying these trust models in open environments.

A. TRUST BOOTSTRAPPING

Trust bootstrapping helps establish an initial trust value for newcomers, which is essential in the context of an open system where microservices are free to join or leave the system at any time. A comprehensive trust bootstrapping approach enables the newcomer microservice to be integrated into the system and recommended to trusters. Some works have mentioned assigning an initial value to the newcomer microservice but have not provided a systematic approach to trust initialisation. Skandylas et al. [18] mentioned that the initial trust of the newcomer could be decided based on its context, but did not give any details on how to initiate trust with contexts. Azarmi [65] mentioned that a newcomer microservice should be assigned a value that is between 0 and 1 as its initial trust but did not introduce how to assign such a value.

B. EFFECT OF MISSING MICROSERVICES

In some socio-based and composition-based trust models, a truster service i uses trust-related knowledge from other microservices to infer its trust towards a microservice j, with which it has not previously interacted. This is described as a "trust chain" [18], [55], [65]. However, if any microservices in the chain are missing (for example, as a result of operational failures or the actions of the developers), then it becomes a question of whether the inferred trust i has to j is reliable or not. Another situation arises with models that use the Markov probabilistic decision-making method to decide the recommendation [48]. The total number of possible solutions is one of the parameters for trust evaluation. For open systems, microservices join and leave arbitrarily, so the total number of possible solutions is constantly changing. Since one of the parameters in the calculation is never stable, the result (i.e., the trust score) will also vary, which undermines the ability of the trust score to stabilise and converge. Another situation arises where other microservices leave the system immediately after the trust score of microservice i is calculated. The total number of possible solutions will be affected as the options decrease, resulting in the trust score for i potentially not reflecting its actual trustworthiness.

C. TRUST MANIPULATION

In systems that manage trust using ratings and recommendations, microservices can manipulate the trust value of entities by controlling their ratings or making false recommendations. For example, in StoRM, microservices can make recommendations to truster microservices [55]. The system assumes that all recommendations are based on good intentions. Whereas that assumption may hold in many closed microservices-based systems, it does not hold in open systems where malicious microservices can then manipulate trust by making false recommendations. This can be done either individually or via coordinated groups, which may collude to recommend a malicious microservice in order to artificially build its reputation and cause other microservices to trust it. Service-composition-graph-based models (e.g., [65]) also face this issue. As it is a graph-based trust model and the trust of the truster relies on its trustees, the truster microservice can give all its trustees high ratings to improve its own trust score.

D. COMPENSATED TRUST

Trust compensation can commonly be seen in composition-based models as the computation of trust needs to go through the composition and decomposition processes. In [64], when the truster gives ratings, it rates the entire solution that it has offered instead of rating individual microservices separately. The trust score of each microservice is computed using a weighted decomposition algorithm based on the overall trust score given to the solution that it contributes to. Although the decomposition algorithm is weighted, there is still a chance for a microservice to be assigned a trust score that is higher than its actual trustworthiness because other microservices within the solution are very trustworthy, and vice versa. In [65], because of the graph-based algorithm, the trust scores of a microservice i's previous trustees play an important role in the computation of its own trust. Therefore, there is a chance that i itself is not trustworthy, but because its trustees have high trust scores, i can still have an artificially high trust score.

E. EFFECT OF SINGLE FAILURES

As with any complex system, failures are detrimental to microservices systems, and so failure is one of the most direct metrics to justify whether a microservice is trustworthy or not. Sometimes a microservice might fail due to temporary factors (e.g., functional error, network outage, or misunderstanding). Thus, an argument can be made that microservices should not necessarily be considered untrustworthy just because of a single failure. Not all trust models exhibit this point. In [38], microservices will be considered as "untrustworthy" for single failures during verification. The trust computation algorithm in [65] will decrease the trust score of microservices significantly after any untrustworthy behaviour.

F. DYNAMIC TRUST

Trust should be dynamic [17], which means the trust score of a microservice should only reflect its trustworthiness over a certain period of time. For example, StoRM [55] introduced a discarding algorithm for the truster when calculating the trust score, which ensures that the trust computation can proceed with the most promising (possible trustworthiness) and more recent ratings.

G. MICROSERVICES LOCK-IN

Most microservices trust models recommend the microservice with the highest trust score to the truster. In an open environment, the system might receive the same request for trusters many times and always recommend the same microservice that provides a service that matches the truster's requirements, while having the highest trust score. This not only raises the risk of overloading the trusted service but also stops trustworthy microservices whose trust scores are not the highest from being recommended and invoked, and consequently building up trust.

H. RELIANCE ON SUBJECTIVE RATINGS

Some implementations use ratings as the basis to compute trust [18], [67]. Ratings are given by trusters: they can be subjective or even manipulative. To avoid this kind of subjectivity, many trust models incorporate some objective metrics into trust computation, for example, Quality of Service (QoS). Although QoS is not commonly used in microservices trust models, many Web Services trust models use QoS to compute trust more objectively [50], [51], [52], [56], [58], [59], [61], [62], [63], [68].

1) USING QoS in TRUST MODELS

Quality of Service (QoS) refers to the performance of a web service in certain aspects [69]. It can typically be monitored using third-party tools. Many trust models include QoS when calculating the trust score. QoS has been discussed extensively in the literature and has been seen as the major criterion for selecting web services to invoke [70]. Different trust models utilise different QoS metrics and different methods of QoS monitoring. Tables 1 and 2 list the QoS metrics that have been adopted in selected microservices trust models and web services trust models respectively.

Combining the QoS metrics given by W3C [71] and the most frequently-included QoS metrics from Tables 1 and 2, we identified the following QoS metrics that are promising for use within a trust framework for microservices-based open systems: availability, latency, cost, throughput, and reliability.

These metrics are important for the trusters, or the stakeholders, of the systems. Although some of them do not have direct connections with trust computation, they are good perspectives from which to evaluate the performance of microservices and can indicate some clues as to the probability of the service's ability to be sufficiently reliable so as to be trusted by the truster in the future:

- Availability represents the level to which a microservice is in an operable and committable state [55]. Availability is computed based on the time that a microservice is up and ready [59]. It can be calculated by dividing the time a service is available by the total service time. Availability ensures that trusters can find the trustees. Microservices cannot provide any services to the trusters unless they are available, which is part of the definition of trust.
- Latency represents the time within which trusters can have their requests satisfied. If a microservice has high latency, there is the possibility that it would become overloaded or that it is experiencing some kind of error, which would affect its ability to provide the expected service during a given time interval.
- Cost is the resource usage during service execution time, such as CPU utilisation, storage, etc. It is important for the stakeholder, as this is closely related to the resources that they will spend on the system. It is also important for the maintenance of the system. Cost is used to represent the consumption of resources; it does not affect the behaviour of microservices directly, but in the long term, it has the potential to impact latency if the load on the service increases in the future.
- Throughput is the number of requests that the microservice instance has served in a certain period of time. It can be calculated by dividing the number of service requests served by the total service time. This can be used to refer to the number of requests that a microservice can complete during a certain time interval. If a microservice has low throughput, it could be a sign of overload or not being able to provide services and meet requirements within a certain time period. This can affect the trust between microservices. Besides, microservices with higher throughput tend to be more scalable and fault-tolerant [72], which facilitates the management of open systems.
- Reliability is the ratio of successful responses to all responses that a service instance provides. It ensures that microservices can always provide their required functions under stated conditions for a specified time interval [61]. Reliable microservices will provide the correct services as their descriptions indicate and can meet the requests of the trusters. This is strongly related to our definition of trust.

IV. QUALITIES OF TRUST MODELS FOR OPEN MICROSERVICE SYSTEMS

In Section II-C we have discussed some principles that will need particular attention when building open microservices systems, and in Section III we identified the limitations that existing microservices trust models have in the context of open environments. Based on this, we now propose some qualities that microservices trust models designed for open microservices systems should exhibit, as follows:

- **Have a comprehensive process for trust bootstrapping**: The trust model should have a complete process for how the trust value for newcomer microservices that join the system can be initialised.
- **Be resilient to missing microservices**: For open systems, it is very common for microservice instances to leave the system (in addition to microservices who are missing due to failure). Missing microservices can have an impact on the actual trustworthiness of microservices or trust score. The trust model should not be affected

TABLE 1. Microservices trust models that consider QoS.

Paper	QoS Metrics
[55]	Packet loss, bit rate, transmission delay, availability, and failure probability
[65]	Cost and delay
[48]	Availability, response time, and throughput

TABLE 2. Web services trust models that incorporate QoS.

Paper	QoS Metrics
[56]	Throughput, latency, execution time, and transaction time
[58]	Response time, availability, and cost
[50]	Precision, accuracy, timeliness, and security
[51]	Response time, throughput, price, and availability
[59]	Response time, availability, throughput, reliability, latency, and cost
[52]	Network latency, CPU utilisation, availability, and response time
[61]	Response time, throughput, reliability, scalability, availability, accuracy, and interoperability
[68]	Response time, cost, and throughput
[62]	Response time, invocation fee, availability, accessibility, reliability, and etc.
[63]	QoS are included in SLA agreements and may vary for different systems

if a microservice instance leaves the system, and the trust value of each microservice should reflect its actual trustworthiness.

- **Be resistant to trust manipulation**: Open systems have a higher risk of being exposed to trust manipulation as microservice instances can join the system arbitrarily, with no guarantees as to ownership or benevolence. Therefore, the trust model should be resistant to trust manipulation: this could be a penalty scheme, a monitoring mechanism, or other measures.
- **Compute the trust value of each microservice individually**: Because of the dynamism that open systems have, microservices can join or leave the system freely. It should be possible to compute a trust value for a microservice independently of others, so that its trust value will still reflect its actual trust if any other microservices in the trust chain or composition leave the system. Trust chains or compositions can be attributed an independent trust score that captures their overall trustworthiness, but as these can be created and broken dynamically, this should be distinguished from trust scores of individual services.
- **Allow failures to some extent**: Failures are always considered detrimental to the operation of systems. Despite this, if a microservice failed to provide service on one occasion, it does not necessarily mean it is completely untrustworthy; it could be a runtime or environmental error that will not be repeated. The trust model should give microservices with poor performance several chances to be invoked before labelling them as "untrusted".
- **Use trust score to reflect recent trustworthiness**: Open systems are dynamic, and so is the trust value of each microservice. In order to bias the system towards more recent interactions, the weight associated with each record should decay over time when computing the trust of microservices. This reduces the impact of historical failures or successes on the microservice's trustworthiness far into the future.
- **Give all microservices chances to perform**: It is very common for trusters to pick the candidate trustee with the highest trust value and continue to pick the same one for future interactions. This can potentially lead to an overload of the microservice with the highest trust value, without using other trustworthy services of a similar nature whose computed trust value is insignificantly lower. Another issue is that if the microservice with the highest trust score leaves the system, the truster will need to select a new trustee. But since it has little information on the other candidates (due to a dearth of interactions with other trustees that have not been selected), it is difficult to decide which microservice should be the new trustee. Giving all candidates a chance to be selected allows the truster to have experience with all of them, and can speed up the process if the candidate with the highest trust score leaves the system, since other microservices also accumulate a history of interaction from which trust can be more reliably computed.
- **Use objective metrics (e.g., QoS) in trust calculation**: The computation of trust should be incorporate objective measures, and not only rely on subjective assessments from potentially-unreliable microservices. As we have discussed in Section III-H1, QoS has been widely adopted in many web services trust models to provide an objective perspective for trust computation. Therefore, adopting QoS as an aspect of trust computation can ensure that the trust score is more objective compared to purely relying on ratings given by trusters.

V. EXAMINING TRUST MODELS WITH THE QUALITIES OF OPEN MICROSERVICES TRUST MODELS

Having identified the desirable qualities of microservices trust models for open systems in the previous section, we next examine the degree to which the trust models that were

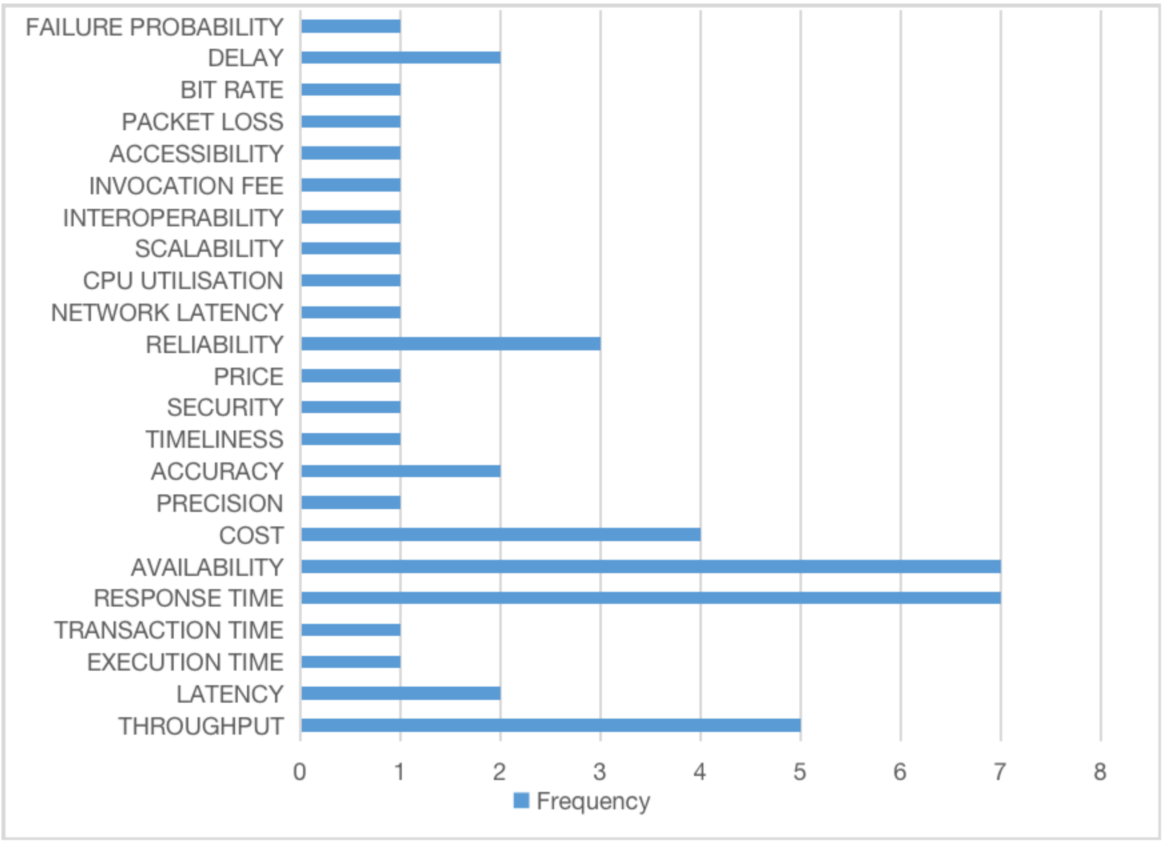

FIGURE 5. QoS metrics adopted in selected trust models.

TABLE 3. Mapping selected microservices trust models to the qualities that are desired for open systems.

	Zero-Trust-based		Socio-based				Composition-based		Control-based	
Qualities	[38]	[39]	[55]	[48]	[49]	[54]	[64]	[65]	[67]	[18]
Comprehensive process for trust bootstrapping	✓	✓	✓	✓	✓		✓			
Resilient to missing microservices	✓	✓				✓				
Resistant to trust manipulation	✓	✓		✓	✓		✓		✓	✓
Calculate the trust value of each microservice individually						✓				
Allow failures to some extent		✓	✓	✓	✓	✓	✓			✓
Reflect recent trustworthiness	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Give all microservices chances to perform						✓				
Use QoS into trust calculation			✓	✓	✓	✓	✓	✓		✓

studied in Section II-D exhibit each of these qualities. The results of this examination are summarised in Table 3. For each quality, a checkmark (✓) denotes that the relevant model exhibits that quality. The box is left empty otherwise.

Quality 1: Trust Bootstrapping Some of the trust models studied lack a comprehensive approach to handle newcomer microservices and bootstrap trust.

Zero-Trust-based models [38], [39] can be said to have a "bootstrapping process" as the theory of Zero-Trust is to distrust all microservices by default. Verification, authentication, and authorisation are needed before the establishment of any interaction between trusters and trustees.

For both blockchain-based models [48], [49], the information of the newcomer microservice is verified before registration using a smart contract. Registered microservices are considered as "trusted" and can be requested by other microservices. Thus, the trust can also be considered bootstrapped.

Some trust models used a neutral value as the initial trust of the newcomer microservice. Kravari and Bassiliades [55] set the range of trust values to $[-1,1]$, where -1 represents not trustworthy at all and 1 represents completely trustworthy. The initial trust value of the newcomer microservice is 0, which is "trust neutral". Adewuyi et al. [64] set the range of trust values to $[0,1]$, where 0 means completely untrustworthy and 1 means completely trustworthy. The initial trust value is 0.5.

Although the concept of giving newcomer microservices initial values of trust was mentioned in [18], [54], [65], and [67], none of them outline a detailed process for setting the initial trust.

Quality 2: Missing Microservices Less than half of the selected trust models are resilient to missing microservices.

Zero-Trust-based models [38], [39] are resistant as they never trust any microservice instances and have to repeat the authorisation and verification process before every interaction. The trustworthiness of other microservices will not be affected if any other microservices are no longer in the system.

The trust value of services is computed based on their QoS and reputation in [54]. When making recommendations, the trust model provides a list of all the microservices that meet the requirements, with the selection of specific microservices to invoke being left to the users. Thus, it will not be affected by missing microservices.

For the other model types, some use the trust chain (or trust path) to calculate trust values [55], [64], [65], [67]. If any node within the chain is missing, the chain will be broken, and the trust value based on the broken chain can no longer reflect the actual trust. Kochovski et al. [48] use the total number of solutions as one of the parameters for trust calculation. Therefore, if microservices leave the system immediately after calculation, the total number of solutions will be affected, and then the actual trustworthiness will differ from the calculated trust score. Skandylas et al. [18] use "trust context" to establish trust. The "trust context" is designed based on the strong trust between microservices provided by the stakeholders. If the microservices are missing, the trust context will no longer reflect the actual context, which will affect the establishment of trust. The relationships between microservices are considered part of the description of microservices in [49], which means that microservices are not fully independent from each other.

Quality 3: Trust Manipulation More than half of the selected trust models include measures to be resistant to trust manipulation [18], [38], [39], [48], [49], [64], [67]. Zero-Trust-based trust models [38], [39] manage this by verifying the information of trustees (e.g., the service that they provide, their APIs, etc.) before every interaction. Kochovski et al. [48] use smart contracts in the Blockchain Layer. Each smart contract instance communicates with external services through registered APIs with unique API keys. This helps to protect the system from malicious smart contracts. Smart contract verification was also used in [49], ensuring that the microservices in the system can perform as advertised. The entities within the system are assigned different roles in [67]. A Security Manager is specially designed to control the microservices and trust authentication. A minimum number of messages that are received or sent by microservices was set as a threshold in [18], before which the direct trust is presumed to be untrusted. This helps to guard against the situation where a malicious entity may act non-maliciously during its initial interactions, in order to build up its trustworthiness, but then begin acting maliciously at a later time. Direct trust is given higher weights in [64]. Thus, it helps trust values be updated reliably and consistently and can offset the influence of trust manipulation.

StoRM [55] cannot satisfy this quality as it allows trusters to make recommendations to others. Also, when trusters give ratings to microservices, they can also indicate a level of confidence. Malicious trusters can use this mechanism to make their ratings appear very confident and recommend untrustworthy microservices to others. Graph-based trust models introduced in [65]) are not resistant to trust manipulation as the trust value of each microservice is based on the trust value of its trustees. Malicious actors can potentially manipulate trust by manipulating the trust value of appropriate trustees. A mechanism against trust manipulation is not mentioned in [54].

Quality 4: Individual Trust The majority of the selected trust models do not calculate trust for microservices individually. Among these models, some use trust chains to compute trust value, while others do not quantify trust. The trust of microservices is calculated individually in [54], without using any underlying trust dependency. The trustworthiness of each microservice is evaluated based on its reputation (the feedback that trusters have given within a particular time interval).

The Zero-Trust-based trust models [38], [39] do not use the concept of a "trust value", and thus do not attempt to calculate an individual score for each microservice. The same condition is found in [49]. The trustworthiness of each microservice inside the system is decided by SC verification. Verified microservices will be treated as "trusted". Microservices can submit their status to update the verification.

Some trust models used trust path to calculate trust [48], [55], [64], [65], [67], which means that the trust score of a microservice reflects not only its own trustworthiness but also the trustworthiness of microservices that it depends on. The trust of a microservice is computed with both "inner" trust and "outer" trust in [18]. Inner trust is the reflection of the trust between the microservices that are in the same trust context as the target microservice, whereas outer trust refers to microservices in different trust contexts. Thus, the trust scores cannot purely illustrate the trustworthiness of microservices themselves. Although a Security Manager is adopted in [67] to authenticate, authorise, and identify before every session, the trust chains between microservices are also considered as a factor in determining trust.

Quality 5: Allow Failure Several trust models allow microservices to fail to some extent.

In [55], if a trustee that is stored in the truster's blocklist is recommended by a trusted recommender twice, then the trustee is trusted again. This is to avoid characterising the trustee as permanently "blocklisted" due to temporary misbehaviour (e.g., functional error or misunderstanding).

In [49], microservices are containerised by docker and work as nodes to respond to requests. An incentive algorithm is adopted to reward microservices whose descriptions in smart contracts match their actual performance. If the microservice does what it says it will do, it will be rewarded with a credit value 1, which can be used to invoke other microservices. If the microservice cannot provide the service it says, then its credit value is calculated as the ratio of the actual amount of resources provided by the node to the amount of resources it registered in the system. This should be a value less than 1. In this case, deception shows no benefits; hence, as a rational microservice, there is no reason to engage in such false behaviour. This gives microservices chances to perform after a single failure while punishing them gradually if they continue to fail.

Microservices of different trust levels are returned to the user in [54]. Therefore, the opportunity exists for users to choose to give an untrusted microservice a chance to perform. This possibility does not arise in models where recommendations yield only the most-trusted relevant microservice.

The number of correct messages that are successfully exchanged is considered a factor in trust computation in [18], [48], and [64], which shows that the system can tolerate a few rounds of failures until the impact of failures on the trust score becomes significant. In a similar way, Zaheer et al. [39] allow N context detection and/or microservices look up failure before dropping a packet. Therefore, single failures are allowed.

In contrast, some trust models have strong policies to guard against even single failures. To avoid compromising the system, trust will not be established between microservices in [38] if any failure occurs during verification. Similarly, Azarmi [65] states that trust should be dynamic. Whenever a microservice shows any untrustworthy behaviour, its trust score will be decreased significantly.

The main focus of [67] was the authorisation of trust. They did not introduce their policy against fault tolerance. Thus, the tolerance towards single failures was not mentioned.

Quality 6: Recent Trustworthiness All of the reviewed trust models can reflect the recent trustworthiness of the microservice [18], [38], [39], [48], [49], [54], [55], [64], [65], [67].

As Zero-Trust-based models [38], [39] establish trust every time before a new interaction, they verify the trustworthiness based on the real-time information of the trustee candidates.

For blockchain-based models, the Edge-to-Cloud Orchestration Layer in [48] is designed to monitor microservices continuously and collect up-to-date information. Collected information will then be passed to the Decision-Making Layer as the basis to compute trust and determine the optimal solution. Therefore, all data relating to trust is updated continuously. Microservices can submit their information to the blockchain from time to time in [49]. This allows the verification to be conducted using recent information.

Among reputation-based models, a discarding algorithm was introduced in [55] when calculating trust score, which can proceed with the most promising and more recent ratings. A similar algorithm was adopted in [54], where recent feedback has higher weight than past feedback. In [65], the trust value of microservices will be changed after every interaction to reflect their recent trustworthiness.

Control-based models both satisfy the recency quality: Trust is established before each session in [67]. This assures that the trust is verified using recent information. In [64], which is composition-based, trust scores are stored and used to guide future interactions; however, their importance declines over time.

The model proposed by Skandylas et al. [18] shows deficiency in this quality. Although the basis of trust is the ratio of successfully exchanged models to total messages, the number of successful exchanges will be treated as 0 before certain number of times. This would affect the accuracy of microservices that recently joined the system.

Quality 7: Chance to Perform Only one of the trust models considered gave all microservices with various trust scores a chance of being invoked by leaving the option to the trusters: In [54], all microservices capable of meeting the requests of trusters will be returned. Trusters will then select the trustees, which could be in any trust level. Some models recommend the microservice with the highest score [18], [48], [49], [55], [64], [65]. For the remaining models [38], [39], [67], the main purpose was to introduce the process of establishing trust between microservices. They do not contain a recommendation metric.

Quality 8: Objective Metrics More than half of the selected trust models included QoS as part of the calculation of trust.

Response time, validity, correctness, cooperation, and availability are the QoS metrics included in [55]. No specific QoS attributes were identified in [48], but the model addressed a variety of non-functional requirements, including Quality of Service (QoS), availability, privacy, and security requirements. The idea of using QoS as one of the parameters is also mentioned in [64], but it does not include specific details. Cost and delay are considered as QoS parameters in [65]. Although not explicitly labelled as "QoS", [18] uses the number of messages successfully exchanged as the basis of trust, which is an alternative objective metric that is closely related to the definition of reliability. SLA is used for the evaluation of trust in [54], which can contain constraints on QoS. Metrics such as latency are used in [49] as one of the basis to establish trust.

Several existing models do not rely on QoS metrics. For zero-trust-based models [38], [39], when authorising the access, the system will compare the compatibility between the requesting trusters and the information of the trustees. The requests and information includes IP address, port, ID, network namespace, etc. QoS is not required to establish trust. The control-based [67] authenticates trust based on the information saved in REST APIs, which does not involve any QoS metrics.

Compared to other trust models, Zero-Trust-based models have comprehensive trust bootstrapping processes and are more resilient to missing microservices. However, they have little tolerance towards failure, and tend not to give all microservices a chance to perform.

Socio-based models are more diverse than Zero-Trust-based models, and thus socio-based models tend to exhibit more of the identified qualities. However, they generally lack resistance towards missing microservices, computing trust individually, and giving all microservices a chance to perform.

Composition-based models combine QoS to trust computation, and can reflect recent trustworthiness. However, they are less strong in terms of the other qualities.

Control-based models are resistant to trust manipulation, can reflect recent trustworthiness, and can compute trust individually. However, their suitability in terms of the other qualities is less obvious. They generally lack a comprehensive trust bootstrapping process, resistance to missing microservices, giving all microservices a chance to perform, and objective QoS metrics.

Table 3 shows that different trust models can meet different qualities, but that none of them can fulfil all the qualities. In particular, trust models need to elevate their awareness on trust bootstrapping, resistance to missing microservices, individual trust computation, failure tolerance, and giving all microservices a chance to perform.

VI. CONCLUSION AND FUTURE WORK

The microservices architecture (MSA) is a dominant architectural style that has been widely adopted in industry. As with any other distributed systems architecture, trust is an important topic in MSA systems. The theory of MSA suggests that trust should be established during messaging, and therefore building a trust model will facilitate the operation of the microservices system without a trust attack occurring. In this paper, we have introduced some existing models and categorised them into four classes: Zero-trust-based, Socio-based, Composition-based, and Control-based.

Meanwhile, MSA has been acknowledged by some researchers and companies as a potential solution for open systems. Nevertheless, the exploration of microservices trust models still has a long way to go in this context.

In this paper, we studied existing microservices trust models, identified the characteristics of open systems, and examined the limitations of existing models with respect to the characteristics of open systems. Based on these limitations, we proposed some qualities that trust models should meet to support the openness of open microservices systems. The qualities include:

- Have a comprehensive process for trust bootstrapping;
- Be resilient to missing microservices;
- Be resistant to trust manipulation;
- Calculate the trust value of each microservice individually;
- Allow failures to some extent;
- Reflect recent trustworthiness;
- Give all microservices chances to perform; and
- Use objective QoS metrics in trust calculation.

We also assessed existing microservices trust models using these novel qualities. The study demonstrated that none of the existing microservices trust models can fulfil all these qualities, which means they cannot fully support open systems. Therefore, there remains a need to propose a novel trust model that can meet all the qualities that have been identified. Primarily, this will entail designing a comprehensive trust bootstrapping process which can make sure that the initial trust value of newcomer microservices is appropriate, in combination with a trust model that can handle the dynamism of open systems while computing trust in a manner that allows for the independence of microservices.

REFERENCES

[1] T. Cerny, M. J. Donahoo, and M. Trnka, "Contextual understanding of microservice architecture: Current and future directions," *ACM SIGAPP Appl. Comput. Rev.*, vol. 17, no. 4, pp. 29–45, Jan. 2018.

[2] V. Raj and S. Ravichandra, "Microservices: A perfect SOA based solution for enterprise applications compared to web services," in *Proc. 3rd IEEE Int. Conf. Recent Trends Electron., Inf. Commun. Technol. (RTEICT)*, May 2018, pp. 1531–1536.

[3] S. Lee, L. Chan, and E. Lee, "Web services implementation methodology for SOA application," in *Proc. IEEE Int. Conf. Ind. Informat.*, Aug. 2006, pp. 335–340.

[4] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: A decade's overview," *Inf. Sci.*, vol. 280, pp. 218–238, Oct. 2014.

[5] D. Neri, J. Soldani, O. Zimmermann, and A. Brogi, "Design principles, architectural smells and refactorings for microservices: A multivocal review," *Softw.-Intensive Cyber-Phys. Syst.*, vol. 35, nos. 1–2, pp. 3–15, Aug. 2020.

[6] O. Zimmermann, "Microservices tenets," *Comput. Sci., Res. Develop.*, vol. 32, nos. 3–4, pp. 301–310, Jul. 2017.

[7] P. D. Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption," in *Proc. IEEE Int. Conf. Softw. Archit. (ICSA)*, Apr. 2017, pp. 21–30.

[8] A. Hannousse and S. Yahiouche, "Securing microservices and microservice architectures: A systematic mapping study," *Comput. Sci. Rev.*, vol. 41, Aug. 2021, Art. no. 100415.

[9] P. Dasgupta, "Trust as a commodity," in *Trust: Making and Breaking Cooperative Relations*, D. Gambetta, Ed. Department of Sociology, Univ. of Oxford, 2000, ch. 4, pp. 49–72.

[10] N. B. Truong, H. Lee, B. Askwith, and G. M. Lee, "Toward a trust evaluation mechanism in the social Internet of Things," *Sensors*, vol. 17, no. 6, p. 1346, Jun. 2017.

[11] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *Proc. IEEE 9th Int. Conf. Service-Oriented Comput. Appl. (SOCA)*, Nov. 2016, pp. 44–51.

[12] J. Fritzsch, J. Bogner, S. Wagner, and A. Zimmermann, "Microservices migration in industry: Intentions, strategies, and challenges," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2019, pp. 481–490.

[13] L. Sun, Y. Li, and R. A. Memon, "An open IoT framework based on microservices architecture," *China Commun.*, vol. 14, no. 2, pp. 154–162, 2017.

[14] J. Parmar, S. Chouhan, V. Raychoudhury, and S. Rathore, "Open-world machine learning: Applications, challenges, and opportunities," *ACM Comput. Surv.*, vol. 55, no. 10, pp. 1–37, Oct. 2023.

[15] L. Baresi, E. D. Nitto, and C. Ghezzi, "Toward open-world software: Issues and challenges," *Computer*, vol. 39, no. 10, pp. 36–43, Oct. 2006.

[16] M. P. Singh and M. N. Huhns, *Service-Oriented Computing: Semantics, Processes, Agents*. Hoboken, NJ, USA: Wiley, 2005.

[17] B. Pourghebleh, K. Wakil, and N. J. Navimipour, "A comprehensive study on the trust management techniques in the Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 9326–9337, Dec. 2019.

[18] C. Skandylas, N. Khakpour, and J. Andersson, "Adaptive trust-aware decentralized information flow control," in *Proc. IEEE Int. Conf. Autonomic Comput. Self-Organizing Syst. (ACSOS)*, Aug. 2020, pp. 92–101.

[19] Y. Shoham, "Agent-oriented programming," *Artif. Intell.*, vol. 60, no. 1, pp. 51–92, 1993.

[20] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *Proc. Eur. Conf. Object-Oriented Program.* Berlin, Germany: Springer, 1997, pp. 220–242.

[21] L. Richardson and S. Ruby, *RESTful Web Services*. Sebastopol, CA, USA: O'Reilly Media, 2008.

[22] A. W. Brown and K. C. Wallnan, "Engineering of component-based systems," in *Proc. 2nd IEEE Int. Conf. Eng. Complex Comput. Syst. (ICECCS)*, Oct. 1996, pp. 414–422.

[23] R. W. Collier, E. O'Neill, D. Lillis, and G. M. P. O'Hare, "MAMS: Multi-agent microservices," in *Proc. World Wide Web Conf. (WWW)*, San Francisco, CA, USA, 2019, pp. 655–662.

[24] D. Lillis, R. W. Collier, M. Dragone, and G. M. P. O'Hare, "An agent-based approach to component management," in *Proc. 8th Int. Conf. Auto. Agents Multi-Agent Syst. (AAMAS)*. Budapest, Hungary: International Foundation for Autonomous Agents and Multiagent Systems, May 2009, pp. 529–536.

[25] K. B. Laskey and K. Laskey, "Service oriented architecture," *Wiley Interdisciplinary Rev., Comput. Statist.*, vol. 1, no. 1, pp. 101–105, 2009.

[26] T. Erl, *SOA Principles of Service Design*. Upper Saddle River, NJ, USA: Prentice-Hall, 2007.

[27] C. Ferris, D. Booth, D. Orchard, E. Newcomer, H. Haas, M. Champion, and F. McCabe. (Feb. 2004). *Web Services Architecture, W3C Note, W3C*. [Online]. Available: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[28] J. Bogner, J. Fritzsch, S. Wagner, and A. Zimmermann, "Microservices in industry: Insights into technologies, characteristics, and software quality," in *Proc. IEEE Int. Conf. Softw. Archit. Companion (ICSA-C)*, Mar. 2019, pp. 187–195.

[29] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*. Cham, Switzerland: Springer, 2017, pp. 195–216.

[30] E. Evans and E. J. Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Reading, MA, USA: Addison-Wesley, 2004.

[31] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Vienna, Austria: Springer, 2014.

[32] F. Ponce, J. Soldani, H. Astudillo, and A. Brogi, "Smells and refactorings for microservices security: A multivocal literature review," *J. Syst. Softw.*, vol. 192, Oct. 2022, Art. no. 111393.

[33] W. Abdelghani, C. A. Zayani, I. Amous, and F. Sèdes, "Trust management in social Internet of Things: A survey," in *Social Media: The Good, the Bad, and the Ugly*, Y. K. Dwivedi, M. Mäntymäki, M. N. Ravishankar, M. Janssen, M. Clement, E. L. Slade, N. P. Rana, S. Al-Sharhan, and A. C. Simintiras, Eds. Cham, Switzerland: Springer, 2016, pp. 430–441.

[34] G. Evan and B. Doug, *Zero Trust Networks: Building Secure Systems in Untrusted Networks*. Sebastopol, CA, USA: O'Reilly Media, 2017.

[35] X. Li, Y. Chen, Z. Lin, X. Wang, and J. H. Chen, "Automatic policy generation for inter-service access control of microservices," in *Proc. 30th USENIX Secur. Symp. (USENIX Security)*. Berkeley, CA, USA: USENIX Association, Aug. 2021, pp. 3971–3988.

[36] E. Shmeleva, "How microservices are changing the security landscape," M.S. thesis, School Sci., Aalto Univ., Espoo, Finland, Dec. 2020.

[37] S. Teerakanok, T. Uehara, and A. Inomata, "Migrating to zero trust architecture: Reviews and challenges," *Secur. Commun. Netw.*, vol. 2021, May 2021, Art. no. 9947347.

[38] C. DeCusatis, P. Liengtiraphan, A. Sager, and M. Pinelli, "Implementing zero trust cloud networks with transport access control and first packet authentication," in *Proc. IEEE Int. Conf. Smart Cloud (SmartCloud)*, Nov. 2016, pp. 5–10.

[39] Z. Zaheer, H. Chang, S. Mukherjee, and J. Van der Merwe, "EZTrust: Network-independent zero-trust perimeterization for microservices," in *Proc. ACM Symp. SDN Res.* New York, NY, USA: Association for Computing Machinery, Apr. 2019, pp. 49–61.

[40] G. Palmer, "De-perimeterisation: Benefits and limitations," *Inf. Secur. Tech. Rep.*, vol. 10, no. 4, pp. 189–203, Jan. 2005.

[41] C. K. Rudrabhatla, "Comparison of event choreography and orchestration techniques in microservice architecture," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 8, p. 18, 2018.

[42] A. Scoppetta, "Zero-trust architectures," M.S. thesis, Politecnico di Torino, Turin, Italy, Oct. 2022.

[43] S. Rodigari, D. O'Shea, P. McCarthy, M. McCarry, and S. McSweeney, "Performance analysis of zero-trust multi-cloud," in *Proc. IEEE 14th Int. Conf. Cloud Comput. (CLOUD)*, Sep. 2021, pp. 730–732.

[44] Y. He, D. Huang, L. Chen, Y. Ni, and X. Ma, "A survey on zero trust architecture: Challenges and future trends," *Wireless Commun. Mobile Comput.*, vol. 2022, Jun. 2022, Art. no. e6476274.

[45] Z. Hosseinifard, L. Shao, and S. Talluri, "Service-level agreement with dynamic inventory policy: The effect of the performance review period and the incentive structure," *Decis. Sci.*, vol. 53, no. 5, pp. 802–826, Oct. 2022.

[46] T. Zheng, X. Zheng, Y. Zhang, Y. Deng, E. Dong, R. Zhang, and X. Liu, "SmartVM: A SLA-aware microservice deployment framework," *World Wide Web*, vol. 22, pp. 275–293, Jan. 2019.

[47] S.-Y. Lin, L. Zhang, J. Li, L.-L. Ji, and Y. Sun, "A survey of application research based on blockchain smart contract," *Wireless Netw.*, vol. 28, no. 2, pp. 635–690, Feb. 2022.

[48] P. Kochovski, S. Gec, V. Stankovski, M. Bajec, and P. D. Drobintsev, "Trust management in a blockchain based fog computing platform with trustless smart oracles," *Future Gener. Comput. Syst.*, vol. 101, pp. 747–759, Dec. 2019.

[49] L. Ruan, S. Guo, X. Qiu, L. Meng, S. Wu, and R. Buyya, "Edge in-network computing meets blockchain: A multi-domain heterogeneous resource trust management architecture," *IEEE Netw.*, vol. 35, no. 5, pp. 50–57, Sep./Oct. 2021.

[50] S. Galizia, A. Gugliotta, and J. Domingue, "A trust based methodology for web service selection," in *Proc. Int. Conf. Semantic Comput. (ICSC)*, Sep. 2007, pp. 193–200.

[51] S.-G. Deng, L.-T. Huang, J. Wu, and Z.-H. Wu, "Trust-based personalized service recommendation: A network perspective," *J. Comput. Sci. Technol.*, vol. 29, no. 1, pp. 69–80, Jan. 2014.

[52] K. Papadakis-Vlachopapadopoulos, R. S. González, I. Dimolitsas, D. Dechouniotis, A. J. Ferrer, and S. Papavassiliou, "Collaborative SLA and reputation-based trust management in cloud federations," *Future Gener. Comput. Syst.*, vol. 100, pp. 498–512, Nov. 2019.

[53] F. Liu, L. Wang, L. Gao, H. Li, H. Zhao, and S. K. Men, "A web service trust evaluation model based on small-world networks," *Knowl.-Based Syst.*, vol. 57, pp. 161–167, Feb. 2014.

[54] M. Azarmi, B. Bhargava, P. Angin, R. Ranchal, N. Ahmed, A. Sinclair, M. Linderman, and L. B. Othmane, "An end-to-end security auditing approach for service oriented architectures," in *Proc. IEEE 31st Symp. Reliable Distrib. Syst.*, Oct. 2012, pp. 279–284.

[55] K. Kravari and N. Bassiliades, "StoRM: A social agent-based trust model for the Internet of Things adopting microservice architecture," *Simul. Model. Pract. Theory*, vol. 94, pp. 286–302, Jul. 2019.
[56] A. Caballero, J. A. Botia, and A. F. Gomez-Skarmeta, "A new model for trust and reputation management with an ontology based approach for similarity between tasks," in *Multiagent System Technologies*, K. Fischer, I. J. Timm, E. André, and N. Zhong, Eds. Berlin, Germany: 2006, pp. 172–183.
[57] J. Sabater and C. Sierra, "REGRET: Reputation in gregarious societies," in *Proc. 5th Int. Conf. Auto. Agents*. Montreal, QC, Canada: ACM Press, May 2001, pp. 194–195.
[58] H. T. Nguyen, W. Zhao, and J. Yang, "A trust and reputation model based on Bayesian network for web services," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2010, pp. 251–258.
[59] M. Mehdi, N. Bouguila, and J. Bentahar, "Trust and reputation of web services through QoS correlation lens," *IEEE Trans. Serv. Comput.*, vol. 9, no. 6, pp. 968–981, Nov. 2016.
[60] Z. M. Aljazzaf, M. A. M. Capretz, and M. Perry, "Trust-based service-oriented architecture," *J. King Saud Univ., Comput. Inf. Sci.*, vol. 28, no. 4, pp. 470–480, Oct. 2016.
[61] L. Lu and Y. Yuan, "A novel TOPSIS evaluation scheme for cloud service trustworthiness combining objective and subjective aspects," *J. Syst. Softw.*, vol. 143, pp. 71–86, Sep. 2018.
[62] Z. Malik and A. Bouguettaya, "RATEWeb: Reputation assessment for trust establishment among web services," *VLDB J.*, vol. 18, no. 4, pp. 885–911, Aug. 2009.
[63] M. Tang, X. Dai, J. Liu, and J. Chen, "Towards a trust evaluation middleware for cloud service selection," *Future Gener. Comput. Syst.*, vol. 74, pp. 302–312, Sep. 2017.
[64] A. A. Adewuyi, H. Cheng, Q. Shi, J. Cao, X. Wang, and B. Zhou, "SC-TRUST: A dynamic model for trustworthy service composition in the Internet of Things," *IEEE Internet Things J.*, vol. 9, no. 5, pp. 3298–3312, Mar. 2022.
[65] M. Azarmi, "End-to-end security in service-oriented architecture," Ph.D. thesis, Graduate School, Purdue Univ., West Lafayette, IN, USA, 2016.
[66] L. Huang, S. Deng, Y. Li, J. Wu, J. Yin, and G. Li, "A trust evaluation mechanism for collaboration of data-intensive services in cloud," *Appl. Math. Inf. Sci.*, vol. 7, no. 1L, pp. 121–129, Feb. 2013.
[67] C. Pasomsup and Y. Limpiyakorn, "HT-RBAC: A design of role-based access control model for microservice security manager," in *Proc. Int. Conf. Big Data Eng. Educ. (BDEE)*, Aug. 2021, pp. 177–181.
[68] Y. Liu and X. Tang, "A trusted model for service selection in trustworthy service composition," in *Proc. Int. Conf. Comput. Sci. Netw. Technol.*, Dec. 2011, pp. 927–930.
[69] K. Lee, J. Jeon, W. Lee, S.-H. Jeong, and S.-W. Park, "QoS for web services: Requirements and possible approaches," *W3C Work. Group Note*, vol. 25, no. 3, p. 119, 2003.
[70] Y. Wang and J. Vassileva, "A review on trust and reputation for web service selection," in *Proc. 27th Int. Conf. Distrib. Comput. Syst. Workshops (ICDCSW)*, Jun. 2007, p. 25.
[71] World Wide Web Consortium (W3C). *Quality of Service*. Accessed: Nov. 3, 2022. [Online]. Available: https://www.w3.org/Architecture/qos.html
[72] A. Huff, M. Hiltunen, and E. P. Duarte, "RFT: Scalable and fault-tolerant microservices for the O-RAN control plane," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2021, pp. 402–409.

ZHONGYI LU received the B.Sc. degree in software engineering from the Beijing University of Technology and the B.Sc. degree in software engineering from University College Dublin (UCD), where she is currently pursuing the Ph.D. degree with the School of Computer Science. Her research interests include microservices, trust management, service-oriented architectures, and microservices trust models.

DECLAN T. DELANEY received the Ph.D. degree in network analysis and design for IoT from the School of Computer Science, University College Dublin (UCD), Dublin, Ireland, in 2015. He worked with LMI Ericsson, Dublin, and has collaborated with SMEs on Horizon Europe funding proposals. He is currently an Assistant Professor with the School of Electrical and Electronic Engineering, UCD. He is an SFI Funded Investigator of the Project CONSUS (https://www.ucd.ie/consus), an SFI industry-funded collaboration focused on precision agriculture, a Principal Investigator of the EPA-funded SmartBOG Project (https://www.smartbog.com), and a Principal Investigator of the CAMEO Project (https://www.cameoplatform.ie). His research interests include network data analytics for adaptable programmable networks and infrastructure and data assurance for the IoT and sensor systems.

DAVID LILLIS (Senior Member, IEEE) received the Ph.D. degree in computer science from University College Dublin (UCD), Ireland, in 2012, and the degree in law and accounting from the University of Limerick. As a Fulbright Scholar, he was hosted by the University of New Haven Cyber Forensics Research and Education Group (UNHcFREG). He is currently an Assistant Professor at the School of Computer Science, UCD. He is a Guest Professor with the Data Mining and Security Laboratory, Beijing University of Technology (BJUT). He is a Principal Investigator of the TRANSPIRE Project (a RegTech-focused industry collaboration with Corlytics and Version 1 funded through the Irish Government's Disruptive Technologies Innovation Fund) and the CeADAR Centre for Applied AI (https://ceadar.ie) and an SFI Funded Investigator of the CONSUS Project (https://ucd.ie/consus). He has published more than 60 peer-reviewed research articles in a variety of subject areas related to computer science, including agent-oriented software engineering, component-based systems, natural language processing, information retrieval, wireless sensor networks, and digital forensics.