Toward Distributively Build Time-Sensitive-Service Coverage in Compute First Networking

Jianpeng Qi, Xiao Su, and Rui Wang

***Abstract*— Despite placing services and computing resources at the edge of the network for ultra-low latency, we still face the challenge of centralized scheduling costs, including delays from additional request forwarding and resource selection. To address this challenge, we propose SmartBuoy, a new computing paradigm. Our approach starts with a service coverage concept that assumes users within the coverage have high access availability. To enable users to perceive service status, we design a distributed metric table that synchronizes service status periodically and distributively. We propose coverage indicator updating principles to make the updating process more effective. We then implement two distributed methods, SmartBuoy-Time and SmartBuoy-Reliability, that enable users to perceive service capability directly and immediately. To determine the metric table update window size, we provide an analysis method based on user access patterns and offer a theoretical upper bound in a dynamic environment, making SmartBuoy easy to use. Finally, we implement the proposed methods distributively on an open-source edge computing simulator. Experiments on a real-world network topology dataset demonstrate the efficiency of SmartBuoy in reducing delays and improving the success rate.**

***Index Terms*— Edge computing, time-sensitive service, service discovering, service coverage, analysis.**

I. Introduction

IDC ESTIMATES that by 2025 41.6 billion devices will be interconnected, and data volume will reach 79.4 zettabytes (ZB) [1]. Current cloud computing architectures do not afford such an overwhelming amount of devices and data due to high latency, limited bandwidth, high carbon footprint, and poor security [2], leading to a high cost for maintaining many TSs (time-sensitive services). Thus, edge computing, an accelerator of cloud computing, affords better computing resources for users and thus gains widespread attention, making the end-to-end delays of the networked services smaller and smaller.

Typically, end-to-end delays of a networked service consist of two phases: (1) Data transmission, including forwarding and scheduling the user's request and, if needed, returning the corresponding result, and (2) service computing, i.e., running the service or function on the computing node. Many works aimed to reduce end-to-end delays belong to these two categories. In the conventional network, where the forwarding nodes are just doing the data transmission, and the computing resources are usually the terminals at the network edge, reducing the end-to-end delays is thus usually separable yet not optimal. Nevertheless, in today's ubiquitous computing architectures, where computing nodes are embedded in the network, end-to-end delays can be further reduced using these resources. Recently, many novel computing paradigms have become popular, such as the COIN (in-network computing or COmputing In the Network)[1] and its improved version CFN (Compute First/Force/Power Networking) [4], [5], [6].

Even though many novel computing paradigms have been proposed, unlike many other moderate services, TSs that need to finish within a hard delay are still suffering from many shortages. Two of the most noticeable steps during the data transmission phase are the additional *service discovering* and *centralized coordinator scheduling*. The former step aims to find where the requested service is and the latter to find the proper service node among all candidates. Obviously, these two steps introduce forwarding and scheduling latencies [7]. To solve this, many novel techniques and frameworks are proposed, such as DHT (Distributed Hash Table) [8], [9], NFaaS (Named Function as a Service) [10], and CFN [4].

Most importantly, according to our observation, when finding the proper service node, users backed by some of those techniques still need to directly or indirectly sense the *global searching space* (resources), which inevitably leads to a high searching cost. Even with some efficient indexing algorithms, synchronizing the resource status in the global searching space is still hard to promise, not to mention the single-point failure. Take the popular DHT technique as an example; the size of the hashing space is the total number of global resources. Indeed, many empirical practices prepare a larger size in case of address collision. This characteristic may cause by common sense that finding the optimal solution needs information on global resources. Fortunately, in [11], we propose a dynamic *stop condition* concept, a condition to identify the service coverage border within the forwarding path to stop searching resources early during the searching step while getting the optimal result without scanning all nodes.

In this paper, we still argue that the resource searching phase is useless when the request latency between the remote service

Manuscript received 14 November 2022; revised 19 March 2023 and 11 May 2023; accepted 23 June 2023; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor C. Peng. Date of publication 3 July 2023; date of current version 16 February 2024. This work was supported by the National Natural Science Foundation of China under Grant 62173158. *(Corresponding author: Rui Wang.)*

Jianpeng Qi is with the School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China, and also with the College of Computer Science and Technology, Ocean University of China, Qingdao, Shandong 266100, China (e-mail: jianpengqi@126.com).

Xiao Su and Rui Wang are with the School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China (e-mail: ustbsuxiao@163.com; wangrui@ustb.edu.cn).

Digital Object Identifier 10.1109/TNET.2023.3289830

[1]We invite the readers to check [3] published in IETF for more interesting use cases.

node and the users is higher than the given deadline. In other words, to save costs, users or schedulers should be blind to the remote service resources if their distance (or service latencies) are far away from the resources. Section III gives a more concentrated illustration of the centralized and decentralized service provisioning processes to explain the abovementioned challenges.

To avoid the forwarding latencies and reduce the global searching space, which is different from the previous works mentioned in II-A, we propose a *service coverage* concept in the compute first networking domain. It is like a fisherman declares his property by shaping a coverage identified by many buoys on the ocean. Then, we propose a novel framework, *SmartBuoy*, to distributively shape the service coverage. Within the coverage, users have a high promise to access the time-sensitive services successfully and are effectively and thoroughly blinded to the resources out of the coverage. Further, we propose two novel distributed indicators, time and reliability, and implement them in SmartBuoy. The significant contributions of this work are as follows:

- To fulfill users' time-sensitive service requirements, we propose a novel framework SmartBuoy to shape the service coverage. SmartBuoy pushes service states near the users and can avoid the additional forwarding latencies introduced by the centralized scheduler and reduce the service nodes searching space. Users within the coverage can quickly know the service status without a centralized coordinator.
- We list several simple design and usage principles of SmartBuoy. Based on these principles, we propose two distributed indicators, SmartBuoy-Time and SmartBuoy-Reliability, to shape the service coverage and to make the users sense the service as fast as possible.
- Under the Poisson process, we analyze the time window design of the service states updating and infer its upper bound. Analysis methodology to this bound are general and can help the current popular computing paradigm such as compute first networking design an effective synchronizing mechanism.
- We implement, evaluate and analyze the performance of SmartBuoy, involving numerous experiments conducted on a real-world ISP network topology dataset.

Section II presents an analysis of related works, with a particular focus on distributed compute-first networking. Section III provides the motivation and proposed framework. In addition, Section IV-A outlines several key design principles, while Sections IV-C and IV-D describe two distributed indicators for shaping service coverage: time and reliability, respectively. Section V presents an analysis of the time window size for updating indicators based on various probabilistic and statistical models, as well as the Age of Information (AoI). Section VI presents simulation results, which demonstrate the effectiveness of the proposed framework. In addition, Section VII discusses several interesting problems related to SmartBuoy's roadmap. Finally, the paper concludes in Section VIII.

II. Related Work

We classify related work into two categories: 1) Service discovering aims to tell users where the requested service is. 2) Reducing the resource searching space when organizing the service-resource mapping relations. We need to note that the second category is to reduce the scheduling cost by filtering unrelated nodes. In the following, we review work on decentralized technologies.

A. Service Discovering

Service discovering plays an important role in selecting an appropriate service or node and further fulfil the requester's demands [22].

Distributed Hash Table (DHT) [8], [9] is a classical way of directly finding the requested services, especially the data, which hashes the data name (key) and distributes the related identifiers into the distributed system. To support location-aware services, **Locality Sensitive Hashing** (LSH) [23] is further integrated. Recently, LSH as a useful tool accelerates the data accessing speed in **Information-Centric Networking** (ICN) [24].

ICN and, in particular, its prominent **Named Data Networking** (NDN) instantiation [25] that constructs the network based on the data name rather than IP provides a realistic solution. Specifically, the work based on NDN uses the function/service name as its routing rules to find the proper service node in a fully distributed manner. This strategy provides several scalable and robust services, e.g., serverless computing, function as a service, and in-network computing [26], [27], [28]. Routing function/service according to the name, such as **Named Function as a Service** (NFaaS) [10], edge cloud selection in **Cross-Service Communication** [21], **Compute-First Networking** (CFN) [4] and NDNe [29], can support many edge-native services.

To gather the runtime information (or status) of services and make the services distributively discovered across the Internet, many related fields are becoming hot, such as **Service-Centric Networking** (SCN) [30], **Cross-Service Communication** (CSC) [21]. Enabling techniques, such as Software Defined Networking (SDN) and NDN, are making them more practicable [32], [33]. A notable purpose of SCN or CSC is to solve the heterogeneous problems that existed in different edge clouds or service providers, therefore, one or more coordinators usually existed to make the service performance data circulate in the whole system.

Another direction is using the **Ad Hoc Network** to automatically select a global or a local coordinator, which significantly speeds up the scalability of the service management, such as [8] and [9]. Recently, a novel concept **Computing Power Network** [5] is proposed. It makes services tactile to Internet users by injecting the dynamic state of services into the routing protocol.

In contrast to the aforementioned research, our work only includes a subset of services in the network in our routing table. Moreover, we provide a theoretical upper bound for the status updating interval, which was not presented in the previous study.

TABLE I
THE COMPARISON OF NOTABLE RELATED WORKS WITH SMARTBUOY

Work	Original Space	Filtered Space	Final Space	Decentralized?
Yousefpour, et al. [12], [13]	$\mathcal{D}$	$\mathcal{D}$	$\mathcal{F} \subseteq \mathcal{D}$ (latency threshold)	$\surd$
Santos, et al. [14]	$\mathcal{G}$	$\mathcal{G}$	$\mathcal{F} \subseteq \mathcal{G}$ (resource requirements)	$\times$
Santos, et al. [15]	$\mathcal{G}$	$\mathcal{F} \subseteq \mathcal{G}$ (latency or location)	nodes satisfying the latency or physical distance requirements	$\times$
Caminero, et al. [16], Toka, et al. [17], Marchese, et al. [18]	$\mathcal{G}$	$\mathcal{G}$	nodes after being filtered by latency threshold	$\times$
Caminero, et al. [19], Qinglan, et al. [20]	$\mathcal{G}$	neighbors within k hops	nodes filtered by resource requirement threshold	$\surd$
Dimolitsas, et al. [21]	$\mathcal{G}$	$\mathcal{F} \subseteq \mathcal{G}$ (nodes after being filtered by the TTL value, i.e., # of hops)	$\mathcal{F}$	$\surd$
Jianpeng, et al. [11]	$\mathcal{L}$	$\mathcal{F} \subseteq \mathcal{L}$ (a inequation between computing, transmitting, and total delays)	two consecutive nodes (piggybacking way)	$\surd$
SmartBuoy (this paper)	$\mathcal{G}$	$\mathcal{F} \subseteq \mathcal{G}$ (latency, reliability, or others)	one, if not in overlapped area, overlapped count, otherwise	$\surd$

1. Space Definition. **Original Space**: The resources covered by the maximal resource searching border, representing for all nodes can be monitored, such as the nodes in a city; **Filtered Space**: The resources that can be scheduled by a coordinator, such as the nodes in a town that scheduled by an edge server. Though for some methods the node status may not be received by the coordinator, it does not mean that those nodes not respond; **Final Space**: The final resources that can be scheduled by a coordinator, such as the top-2 candidates.
2. Space Markers. $\mathcal{G}$: all nodes in the system; $\mathcal{D}$: manual specified domain; $\mathcal{L}$: nodes in the request forwarding path; $\mathcal{F}$: nodes filtered by indicators.

B. Reducing the Searching Space

Because the resources in edge computing are usually dynamic, huge, and various, gathering the status of these distributed resources is helpful in quickly forwarding users' requests to the suitable service node. However, achieving this is challenging because of the considerable searching or monitoring space. Therefore, reducing the resource searching space is essential.

To end this, many distributed **grid and tree-based indexing** schemas and algorithms are proposed. References [10] and [28] are the first cases adopting the tree-like naming schema. Each node of these works has a sketchy view or direction of the service node candidates. Further, CFN using this novel design achieves distributively gathering the status of the nodes [4]. Similarly, [34] also proposes a distributed resources monitoring method, CFN-dyncast, according to the load of each computing site and the network status. However, searching space can be significantly further improved in these works.

To limit the search space, researchers have proposed several methods. One such method, called DoSRA, is introduced in [20] and is based on a service coverage approach. DoSRA builds a resources pool with a fixed size k hops radius, centered around the end devices. However, this approach can add additional pressure to the end devices. In [13], the authors use a reachability table to monitor the status of fog nodes within a specific domain or area, considering factors such as round-trip and processing delays. This domain knowledge can be useful in various settings, such as smart homes or factories. Alternatively, [35] compares four different fog colony methods: centralized, independent, with communication, and with overlap. A fog colony comprises the edge nodes in a specific geographical region, and these methods differ in their coordination and communication strategies.

In some cases, distance can also be used to reduce the search space, as in the floating content (FC) method [36], [37] for opportunistic networking or ad hoc networks. FC provides a predefined shared area where content can be shared among mobile devices in the region based on communication range constraints. Similarly, computing nodes can also be selected based on distance, such as in the drop computing approach [38].

Table I presents several notable works that have focused on distributively finding suitable nodes to perform services. By employing two or three filtering steps, these methods can significantly reduce the original search space to a set of top-X candidates, thereby improving scheduling efficiency. However, there is still considerable room for improvement in the final search space used when making a scheduling decision. Many recent works that rely on fixed borders or centralized approaches may not be suitable for time-sensitive scenarios in which resource utilization is dynamic (as discussed in Section III).

In our previous work [11], we demonstrated that it is feasible to mine a dynamic border of the search space for finding a global optimal service node among forwarding nodes in a dynamic edge environment. In this work, we propose the service coverage concept and introduce two automatic and distributed methods for solving the problems of distributed service discovery and search space reduction. We also provide a theoretical bound and analysis steps for determining the optimal status updating window size, which makes our proposed approach, SmartBuoy, easy to use.

III. MOTIVATION AND SMARTBUOY FRAMEWORK

Many applications or services have relatively short deadlines that must be met [39]. To see a partial list of these deadlines,

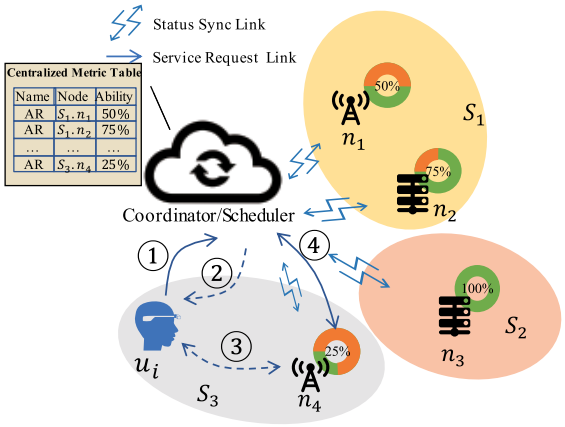


Fig. 1. Service provisioning processes: a centralized manner.

one can check the list we maintain on GitHub.[2] To ensure a good user experience, these applications and services typically have an end-to-end delay upper bound or deadline threshold that must be met.

In this section, we will use various deadline thresholds to indicate different services, which can be achieved by varying the deadline threshold T_s^{THR}. We will assume that the services are pre-installed on specific nodes, which we discussed in Section VII as a service placement problem. To make our motivation more clear, we compare two computing paradigms including the conventional centralized manner and our proposed decentralized SmartBuoy framework.

A. A Conventional Way

This example proves that centralized service discovering and scheduling may not be appropriate for TS applications. As shown in Fig. 1, Coordinator maintains and monitors four service nodes $\{n_1, n_2, n_3, n_4\}$ scattered in different areas. In order to provide a suitable service node for a user request, the Coordinator needs to establish a heartbeat link to receive the nodes' runtime status.

Take an AR service as an example, which includes many bandwidth- and/or computation-intensive components such as data "receiving", task "processing" and augmented image "displaying" [3]. "Processing" is of the most important steps to analyze the sensed data from the physical world and then add them to a live camera feed or situation [40]. Because of the critical time requirement, putting this task in the remote cloud is unrealistic. However, in a resource-constrained environment, finding a powerful node to process it is vital but challenging.

Therefore, popular ways usually follow three steps: (1) Service discovering. A user u_i first sends a resources request to the Coordinator maintaining the global resource status by a metric (or metadata) table. One of the most well-known cases is NameNode in Apache Hadoop. Then (2) Centralized coordinator scheduling. The Coordinator checks its metric table and compares the maintained service node candidates according to their ability, response delay, and location, and finally returns n_4 in area S_3 as her service node. In this step, Coordinator usually monitors and maintains numerous geographical devices' status through an unstable and changeable network, which makes the process unreliable and carbon footprint heavy. At last, (3) u_i sends her task to n_4 for further processing.

Another way of service discovering is regarding the coordinator as an information forwarding proxy. This way, (1) u_i sends her information (location, angle, and moving speed) to the coordinator, and (4) the coordinator forwards the information to n_4 after checking the candidate's status.

However, a constraint for services is that, like AR, end-to-end latencies usually need no more than 20 ms. Based on the network latency analysis in [41], even for the fastest 5G, the median RTT (Round-Trip Time) of the nearest edge site from the end-user is still 10.4 ms. That means there is almost no time left to analyze users' data after the coordinator found a powerful computing node, i.e., service discovering and scheduling cost a half time! Thus, we conclude that this centralized computing paradigm is not applicable to time-sensitive applications.

Dig deeper; we have discovered that, in addition to the forwarding delays caused by service discovery and scheduling, utilizing a metric table in this paradigm results in all candidate nodes being considered by the user. This ultimately leads to a significantly larger search space and a higher cost, even for idle remote nodes like n_3 in area S_2, which have no chance of being selected as the service node. This lack of efficiency is also evident in the recently popular "compute first networking" [4], [5]. Furthermore, the coordinator can become overwhelmed by the global performance metric synchronization, particularly when available network and computation resources are dynamically changing. In summary, this scenario results in unnecessary networking and computing resources being expended due to the large global search space.

B. SmartBuoy Framework

Aiming to avoid the additional forwarding delays and the global searching space, we propose a novel framework that shapes a service coverage to fulfill users' demands, especially for the TS applications, namely *SmartBuoy*. SmartBuoy also adapts to the dynamic environment, especially when the bandwidth and computing ability are changeable.

To get the design idea of SmartBuoy, one may be familiar with the "15-Minute City" scenario in a smart city where the locals can access all necessities at distances that would not take them more than 15 minutes by their convenient transportation [42]. In the scenario, local users are blinded to the remote infrastructures, e.g., a parking lot far 10 KM away from their home. Because for them having much information about these remote infrastructures is useless but a pressure, i.e., *"a relative afar is less important than a close neighbor."*

Similarly, a service node in this paper can be regarded as a center of its *service coverage*. Users within the coverage are the "infrastructures" that must be fulfilled, i.e., service-oriented. Fig. 2 depicts this imagination where each shaded ring area denotes the TS service coverage, i.e., S_1, S_2,

[2]Time requirements for services: https://github.com/qijianpeng/time-requirement-for-services.

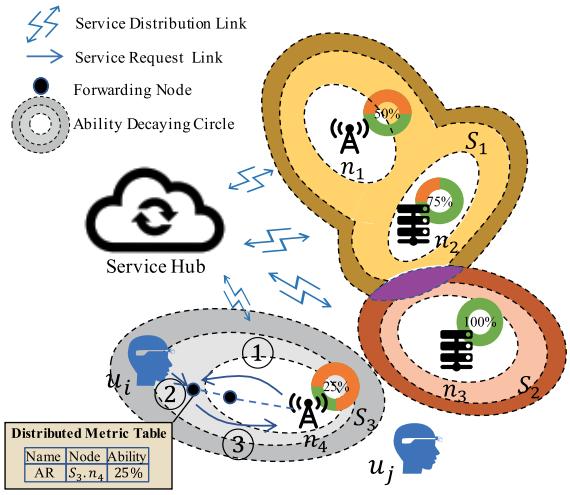


Fig. 2. Service provisioning processes: our decentralized imagination.

and S_3 for service nodes $\{n_1, n_2\}, \{n_3\}$, and $\{n_4\}$, respectively. Taking account of the deadline metric is mainly influenced by network conditions and computing power of the service nodes, service accessibility to users decreases as the distance from the center increases.

To let users know where the service node is, we design a distributed metric table (we discuss it latterly in Section IV-B) and assume it is stored on an AP (access point). In this way, (1) service node n_4 periodically sends its service status to AP nodes, then (2) user u_i in area S_3 can quickly and directly perceives the service status by checking the distributed metric table, and (3) accesses the appropriate service node or computing resources that cover her, i.e., node n_4. This paradigm does not need a coordinator to sense and search the global resources located in remote areas S_1 and S_2 and thus avoid the additional forwarding delays introduced by centralized service discovering and scheduling.

Note that because the geo-distributed, space- and resource-limited characteristics have broadly existed in edge computing, scalability near the users is thus hard to promise [43]. Therefore, resources in the area are usually countable and in a relatively small number, which means the size of the distributed metric table is also small. This is clearly different from the coordinator, which may be overwhelmed by numerous metric updates in a centralized paradigm.

Furthermore, if users are in the blinded areas, i.e., out of the coverage, or the requested service node reaches its maximum affordability, we assume two solutions or principles can be adopted, including (1) service vendors extend computing resources in the long term, and/or (2) accessing points near the users redirects their requests to the cloud in the short term, if in time before the deadline.

Here, we give the network topology description that we assumed in this paper. It is known that there exists a computing center or edge site in each city [44], [45]. These centers are linked by backbone networks and are managed by a cloud center. The network connections within each city are structured in multiple layers, where the highest layer is the computing center and the lowest layer can be the AP. Intermediate layers could include various components, such as micro-DCs, base stations, or forwarding devices with computing capability. See Fig. 11 for an topology example.

To make the concept clear, we define *service coverage* below (Def. 1).

Def. 1 (Service Coverage): Call a physical service circle a service coverage if it is identified by a specific ability, such as the reciprocal of latency or the service reliability. As the geo-distance is far away from the centric service node (e.g., micro-DC, base station), the quantities of the indicator decrease.

Thus, a natural question that arises is **"how to shape the service coverage dynamically and distributively?"** In Section IV, we design and implement two indicators to answer this question.

IV. Two Distributed Sensing Indicators: Time and Reliability

We first give several design principles and preliminaries when implementing SmartBuoy in Section IV-A. Section IV-B introduces the distributed metric table structure. Then we use two indicators, including time and reliability, to distributively shape the service coverage step by step in Section IV-C and Section IV-D, respectively. These two indicators are mainly stored and updated in the metric table, aiming to select the optimal service node in Fig. 2.

A. Design Principles and Preliminaries

We propose several principles and preliminaries to combat the service discovering latency, enormous resources searching space, and dynamically changed resources.

*1) **Distributed, incremental, and dynamic**:* Unlike many current works that hold a global view of the network status (i.e., users or coordinators are essentially sensible to all service nodes), SmartBuoy tries to create a service coverage distributively, incrementally, and dynamically. "Distributively" means the coverage border of a TS service can be identified without direct communication to the service node or a coordinator. "Incrementally" means performance indicators stored in *distributed metric table* can be updated hop-by-hop incrementally based on the current forwarding node status and the received status packets, i.e., memoryless. "Dynamically" means the coverage border is not fixed but changeable because of the dynamic bandwidth and computation resources.

Meanwhile, in many cases, the concurrency capability of the service may be constrained, and newly accessed users usually have a terrible impact on the already served sessions because of competition for resources. Given that the pre-allocated resources are usually stable, we also assume the computing delay is relatively fixed if the number of served sessions/users does not reach the maximum number of concurrent accesses. Otherwise, queueing theory should be introduced.

*2) **Proactively diffusion and/or piggybacking**:* In order to fulfill the service time constraints, we need to spread the service node status and update the metric table. In addition to proactively diffusing the service status, e.g., according to broadcasting, *piggybacking* can also be used [46] when there is no security issues. As an enhancement for proactive diffusion,

piggybacking is a way that allows us to inject the status as a field into the service packet flow. When the service results flow toward the user, each node on the path can extract the needed information.

In a natural production environment, broadcasting may be unaffordable, which can utilize a publish/subscribe message protocol. SmartBuoy assumes that network topology can be various, which means geospatial borders or domains are not essential.

*3) **QoS (Quality of Service) promises**:* Service coverage for the same services may be overlapped. It may happen rather often in the edge nodes that are densely deployed, such as UAV formation. Overlapped information of the services' coverage can be directly recorded on the distributed metric table in the way of one service with multiple accessible routes (or entries). To simplify the forwarding processes, we pick up the best record marked with an optimal value and forward the request to the corresponding next hop.

Furthermore, the overlapped area is also an improvement for QoS. Inevitably, users who want to request TS service may be out of the service coverage (e.g., user u_j as shown in Fig. 2). It can be further classified into two types: (1) The remote cloud can fulfill critical time requirements. In this scenario, given that the remote cloud is actually a remote powerful service edge node [47], the metric table near the users should also have that record if the time constraints are not broken. Thus, the service can be directly sent to the remote cloud. Another scenario is (2) the users are totally out of service coverage. We think the neglected users may need to proactively contact the infrastructure or service vendor, like setting up a network service by contacting the ISP.

Next, we describe the distributed metric table structure indicating where the service node is (Section IV-B). Unlike the centralized coordinator, the metric table in SmartBuoy holds only a local view of the available services. Then, we design and implement two indicators (Section IV-C and Section IV-D) to distributively shape the service coverage for TS service in edge computing.

B. Distributed Metric Table Structure

We assume each storing and accessing capable forwarding node holds a metric table in the structure shown in TABLE II so that users near the node can quickly find a proper service node. Each entry in the table contains at least three items: Service `Name` corresponding to the user requested, the `Next-Hop Node` toward the node providing the TS service, and service accessing `Ability` from the current node. This structure can provide much flexibility. `Ability` can be expressed in many ways, including the time consumed, the promised reliability, or any other scalar quantities. Following the principles, one can also integrate other indicators such as energy cost, privacy, and budget.

Instead of combining the `Name` and `Next-Hop Node` attributes to find the service node, a fixed routing pattern can be achieved through direct adoption of the Pending Interest Table (PIT) in Named-Data Networking [25] or by using tracing applications [11]. By broadcasting an Interest to all nexthops of the service node and using tracing-based solutions

TABLE II
DISTRIBUTED METRIC TABLE: PER-NODE PERSPECTIVE

Name	Next-Hop Node	Ability
VR	n_i	a_i
VR	n_j	a_j
Remote Surgery	n_k	a_k
Smart Venue	n_l	a_l
...	...	...

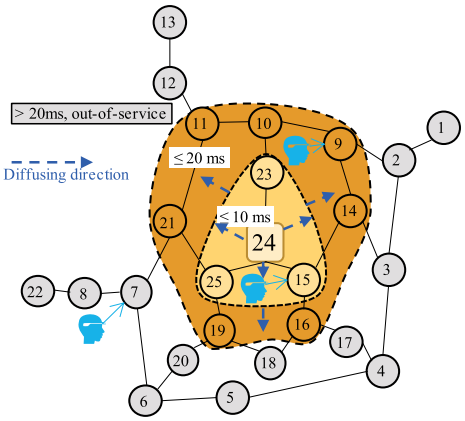


Fig. 3. Service coverage generated by SmartBuoy-time.

to create the inverse path in the PIT [48], a time-indicated service coverage can also be established. Additionally, in cases where techniques such as SDN are used, the address of the service node can also be used to simplify the process for the `Next-Hop Node`.

When a forwarding node or an AP is near a user who wants to access a VR service, the node first receives the request, then checks the distributed metric table and finds the target node(s) maintaining the VR service (e.g., n_i and n_j). Finally, the node forwards the request to the selected most suitable node (e.g., node n_i, if the ability $a_i > a_j$). Formally, in this paper, we define the service request as

Def. 2 (TS Service Request): The action of an end-user or client requesting a time-sensitive service or resource. The request requires bandwidth resources to transmit data of size D and computing resources, which can be converted to time, to complete the task. The service node must return the result within a latency threshold of T_s^{THR}.

Note that we mainly solve the TS service coverage in this paper. Although the number of kinds of services may be significant, their quantities maintained in a node are usually limited because of the resource-limited and geo-distributed characters, leading to a small number of records in the table. Meanwhile, the TS service coverage range is usually small because of the critical time constraints, which makes the forwarding path hops short.

C. SmartBuoy-Time: Shaping the Service Coverage by Time

This section answers the question of "how to **shape the service coverage identified by time dynamically and distributively**?"

To be clear, Fig. 3 depicts a service delay coverage demo. In this demo, n_{24} is a service node, whereas other nodes are

the forwarding nodes or the APs that provide a direct link to users. Take a service with no more than 20 ms response delay as an example. We can shape the first-level coverage including $\{n_{15}, n_{23}, n_{25}\}$ with the service time no more than 10 ms. Recall that service delays include two parts: computing and transmitting. Computing delay can be analyzed from a log file or prediction models derived from queueing theory and/or machine learning. Transmitting time can be adopted from instantaneous measuring network conditions or based on a network observer. Similarly, the second coverage can be shaped by the nodes with service delays of no more than 20 ms. Moreover, higher than 20 ms is judged to be out-of-service.

Denote by ability a_j the estimated service delay that started on the AP n_j and ended when the result returns. Therefore,

$$a_j = \frac{1}{T_s^{compt} + T_{j,s}^{trans}} \tag{1}$$

where T_s^{compt} is the computing time on node n_s and $T_{j,s}^{trans}$ the transmitting time of a data from n_j to n_s. Different from TCP/IP where transmitting data likes a flow, in store-and-forward network, e.g., named-data networking, delays of transmitting are cumulated hop-by-hop and

$$T_{j,s}^{trans} = T_{j,j+1}^{trans} + \cdots + T_{s-1,s}^{trans}$$

In many cases, such as identifying an image of a fixed resolution in computer vision, the raw data of size D_{raw} is usually fixed or known. Thus, transmitting time $T_{j,s}^{trans}$ can be easily evaluated inversely given a service, i.e., n_{24} proactively diffuse the updating action indicating its status to its neighbors; then, the neighbors calculate transmitting delays according to the received packet of size D_{recv} with its transmitting duration T_{recv} and update their metric table accordingly. To simplify, we omit the result transmitting time because of its small packet size. Thus, $T_{j,s}^{trans}$ on node n_j can be estimated in real-time by

$$T_{j,s}^{trans} = \frac{D_{raw}}{D_{recv}/T_{recv}} \tag{2}$$

where D_{recv}/T_{recv} is actually the observed bandwidth under a assumption that uplink and downlink bandwidth can be equal, which is natural in a decentralized network. However, there are also some useful tools to monitor the bandwidth when the bandwidth equative condition is not followed, such as iPerf3,[3] netperf,[4] in-band network telemetry and gRPC [49]. We also provide another method SmartBuoy-Reliability, instead of real-time monitoring.

To avoid endless forwarding and updating during the proactive diffusion processes, we derive a *stop condition* based on (1) as $a_j < 1/T_s^{THR}$.

Def. 3 (Time-based stop condition $a_j < 1/T_s^{THR}$): If the ability a_j to provide a user the TS service s is smaller than the given threshold $1/T_s^{THR}$, or the diffusion path forms a loop, or the node receives stale packets, the diffusion process can be stopped. We call this judgment criterion a time-based stop condition.

[3]iPerf3: https://iperf.fr.

[4]netperf: https://github.com/HewlettPackard/netperf.

Algorithm 1 SmartBuoy-Time. Shaping the Service Coverage Distributively: A Single Node View (on Node n_j)

Require: T_s^{compt}, the service computing time on n_s; T_s^{THR}, deadline threshold for service s.

observes and receives status packets to calculate $T_{j,s}^{trans}$
calculates ability a_j by (1)
if the time-based stop condition (Def. 3) is met **then**
    stop forwarding the metric-table-updating action
end if
if the service s in metric table exists **then**
    update a_j
else
    insert a new entry $\{s, n_s, a_j\}$ into metric table
end if
forwards the updating action to its next neighbors

Based on (1) and Def. 3, we propose Algorithm 1 on the perspective of a single node, e.g., n_j. This node receives the status packets from the service node s to update its metric table. After this process is done, a service coverage that contains all satisfied forwarding nodes or APs is generated. Given that we have already described the details, we omit the algorithm descriptions.

Complexity analysis for SmartBuoy-Time. In Algorithm 1, for a service node n_s, we first calculates a_j when receives a status packet, whose time complexity is $O(1)$. Then, we update the metric table if the same entry is founded, whose time complexity is proportional to the table size m and is $O(m)$ at the worst case.

For communication overhead, like [21], we define it as the percentage of utilized links. Under this definition, the length of the longest path indicating the maximal coverage radius can be denoted by $d_{s,radius} = T_s^{THR}/l_i$, where l_i is the lead time for link i. Suppose the distance between the service node n_s and the farthest node n_e on the network edge is $d_{s,e}$, the communication overhead is $min(d_{s,radius}, d_{s,e})/d_{s,e}$. Typically, $d_{s,radius} \leq d_{s,e}$. We suppose the average degree for each node n_j is $E_a = \frac{\#\ links}{\#\ nodes}$ [50]. Given an entire network, considering an active estimation of $T_{j,s}^{trans}$, the total communication overhead could be $min(E_a^{d_{s,radius}}, E_a^{d_{s,e}})/E_a^{d_{s,e}}$. See Fig. 4 for an example of the relationship between $d_{s,radius}$ and $d_{s,e}$ when $d_{s,radius} = 2$, $d_{s,e} = 4$, and $E_a = 3$. In this example, the communication overhead is $10/15 \simeq 66.7\%$. Therefore, when handling an extensive edge network, $d_{s,radius}$ as a filter is efficient, i.e., using delay threshold to shape the service coverage.

D. SmartBuoy-Reliability: Shaping the Service Coverage by Reliability

SmartBuoy-Time promises a distributed service coverage updating in real-time and a dynamic environment adapting. On the one hand, even if we are guided by the principle of "proactively diffusion and piggybacking", the metric table updating action may not be easy to judge. With a higher updating frequency, costs may be high. On the other hand,

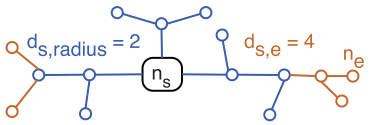


Fig. 4. Distance of $d_{s,radius}$ and $d_{s,e}$.

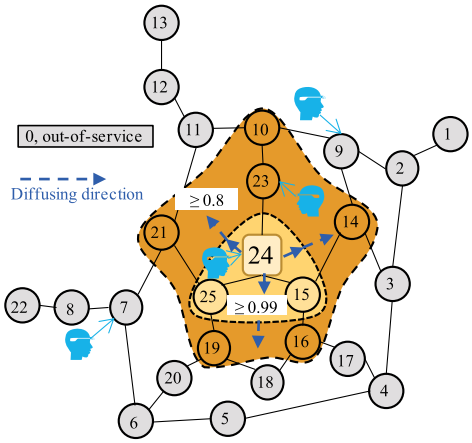


Fig. 5. Service coverage generated by SmartBuoy-reliability.

a lower updating frequency may lead to the records stored in the metric table being stale.

Therefore, inspired by service reliability, we design another distributed indicator for a_j, namely SmartBuoy-Reliability. SmartBuoy-Reliability promises a reliable, lightweight way to provide users with a TS service. Based on [51] and [52], TS service reliability definition is thus given in Def. Def. 3.

Def. 4 (TS Service Reliability): The ability of a service s to respond user's request within a given deadline threshold T_s^{THR},

$$a_j := R(t) = \Pr\{t < T_s^{THR}\} \tag{3}$$

where t is the service delay.

How to **shape the service coverage identified by TS service reliability dynamically and distributively?** Fig. 5 depicts an reliability coverage demo. Similar to Fig. 3, n_{24} is a service node, whereas other nodes are the forwarding nodes or the APs that provide a direct link to users. Instead of shaping the time coverage, after examining the networking and computing condition, we can shape the first coverage including $\{n_{15}, n_{25}\}$ with a promise of the service reliability $R(t) \geq 0.99$. Similarly, the nodes with $R(t) \geq 0.8$ can shape the second reliability coverage. And $R(t) < R_s^{THR} = 0.8$ that meets the reliability-based stop condition defined in Def. Def. 5 is judged to be out-of-service.

Def. 5 (Reliability-based stop condition given $a_j < R_s^{THR}$): Suppose the ability to provide the user a TS service s is smaller than a given reliability threshold R_s^{THR}, or the diffusion path formed a loop, or the node receives stale packets. In that case, the diffusion process can be stopped. We call this judgment criteria a reliability-based stop condition.

However, calculating $R(t)$ has been proved to be a challenging problem because not only the transmitting resources

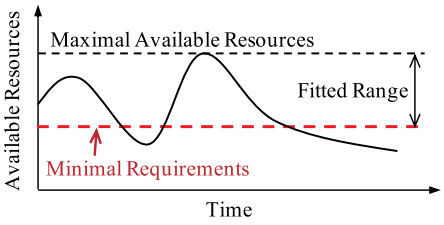


Fig. 6. Available resource curve.

vary with time but also the service node resources, which can be regarded as a dynamic multi-state edge computing network problem [53]. See Fig. 6, if the available resources drop below the minimum requirements of a service, then the service loses promised reliability. Popular ways to solve this problem are like calculating the minimal service requirements, including bandwidth and computing resources, to check whether they can be fitted into the changeable resources. For example, counting the feasible solutions' ratio to the total. However, current works mainly focus on a centralized way of computing $R(t)$ based on the globally collected networking data and performing the RSDP (Recursive Sum of Disjoint Products) algorithm [53], ignoring the service node states. Other methods, such as Monte-Carlo simulation [54], SVM [55], and DNN [56], also suffer from a centralized and huge computation problem.

Therefore, SmartBuoy-Reliability is more complex than SmartBuoy-Time when implementing it in a decentralized manner. In the next, we crack it in 3 steps: (1) examining $R(t)$, (2) computing $R(t)$ distributively, and finally, (c) proposing the distributed SmartBuoy-Reliability algorithm.

*1) **Examining R(t)**:* Service delay t, or $R(t)$, is typically affected by the changeable available resources, including computing and bandwidth. To simplify the processes, we assume the computing delay related to the service node is relatively fixed if the number of served sessions/users does not reach the maximum number of concurrent accesses. This assumption is fare enough, especially in today's VM/Docker/K8S-supported applications where the resources consumed by a container can be pre-allocated with a fixed demand. We use T_s^{THR} to denote the rest time after it subtracts the fixed computing delays.

Therefore, our goal now turns to find the bandwidth bounds $\mathcal{B}_{THR}$ satisfying $t < T_s^{THR}$ in a shared network. $R(t)$ can be further denoted by

$$R(t) = \Pr\{\mathcal{B} > \mathcal{B}_{THR}\} \tag{4}$$

where $\mathcal{B} = \{b_{s,2}, b_{2,3}, \cdots, b_{j,k}\}$ is a sequence of links representing the available bandwidth from the service node n_s to the forwarding or the AP node n_k, and $b_{j,k}$ the available bandwidth of two adjacent nodes n_j and n_k.

Like many networking reliability analyses, $b_{j,k}$ observed on n_k follows a discrete PDF (Probability Density Function) given a time window of size T. TABLE III depicts an example of the distribution. In practice, the distribution of available bandwidth can be measured by many popular tools and techniques, such as mentioned in Section IV-C. Note that we do not focus on solving the alignment of bandwidth units,

TABLE III
BANDWIDTH PDF

$p_{b_{j,k}}$ \ Range (Mbps)	**0 ∼ 1**	**1 ∼ 2**	**2 ∼ 3**	**3+**
$p_{b_{s,2}}$	0.2	0.2	0.5	0.1
$p_{b_{2,3}}$	0.1	0.3	0.5	0.1
···	···	···	···	···

TABLE IV
BANDWIDTH CDF $F(c, b_x)$ (THREE NODES IN TOTAL)

$F(c, b_x)$ \ Range (Mbps)	**≥ 0**	**≥ 1**	**≥ 2**	**≥ 3**	Node
$F(s, b_x)$	1	1	1	1	n_s
$F(2, b_x)$	1	0.8	0.6	0.1	n_2
$F(3, b_x)$	1	0.72	0.36	0.01	n_3
···	···	···	···	···	···

e.g., aligning Kbps, Mbps, and Gbps to Kbps. To make an alignment, one can check the work [53], [57].

*2) **Distributively Computing R(t)**:* We can compute the minimal bandwidth requirements by

$$b_{min} = \frac{D_{raw}}{T_s^{THR}} \tag{5}$$

where D_{raw} is the size of the raw data, which is known based on our previous assumption. Therefore, by (5) b_{min} can be calculated based on prior knowledge. Note that we use prior knowledge here to illustrate the idea more clearly. However, this prior knowledge can be removed when calculating the reliability.

Moreover, in order to ensure a stable routing path from the access node to the service node, SDN can be integrated. In this paper, we utilize a relatively stable network topology and employ a shortest path forwarding strategy to maintain measurability.

Then, by (4), we further derive

$$R(t) = \Pr\{\mathcal{B} \geq b_{min}\} \tag{6}$$

Eq. (6) means for each $b_{j,k} \in \mathcal{B}$ we have $b_{j,k} \geq b_{min}$. Therefore, $R(t)$ can be further denoted as

$$R(t) = \Pr\{\bigcap_{b_{j,k} \in \mathcal{B}} b_{j,k} \geq b_{min}\} \tag{7}$$

which means the available bandwidth from the n_k to n_s should be greater than b_{min}.

We assume the capacities of different edges are statistically independent, which is a typical assumption of reliability analysis on networking [53]. Then (7) can be further denoted by

$$R(t) = \Pi_{b_{j,k} \in \mathcal{B}} \Pr\{b_{j,k} \geq b_{min}\} \tag{8}$$

However, through (8), the globally available network status still needs to be known given b_{min}, i.e., each forwarding or AP node will maintain a whole TABLE III, leading to centralized gathering when calculating $R(t)$. To avoid this and to compute distributively, we define a CDF (Cumulative Distribution Function) $F(c, b_x)$ as

$$F(c, b_x) = F(c-1, b_x) \Pr\{b_{c-1,c} \geq b_x\} \tag{9}$$

where $c \in \{s = 1, 2, \cdots\}$ represents a node that currently receiving and processing the status packets, i.e., the current node being accessed, b_x the available bandwidth boundary (see the header of TABLE IV). Therefore, we have the reliability CDF

$$R(c, t) = F(c, b_{min}) \Pr\{s \ is \ available\} \tag{10}$$

where $R(c, t)$ represents the reliability of finishing the service s within time $t < T_s^{THR}$ on node c. $\Pr\{s \ is \ available\}$ is the availability of service s given a time period T on node n_s, which can be computed by analyzing the historical data.

By (9), a global view of networking in TABLE III can be further turned from a single node to a cumulative style as in TABLE IV. This table contains three nodes' CDF, and each row (record) in the table is the data that needs to be stored and updated on a specific node. Note that we assume the bandwidth of the service node to itself is infinite; thus, all the values in $F(s, b_x)$ are equal to 1. CDF $F(c, b_x)$ provides the ability to change b_{min} at various data size. In other words, services with arbitrary input data size can also be supported.

For example, given the data of size 3 Mb, a user request wants to be responded within 1 second. According to (10), by checking TABLE IV entry $F(3, b_x)$ on the forwarding node n_3, the produced reliability $a_j = R(3, 1s) = 0.01$. If we promise the user a predefined reliability of R_s^{THR} = 0.8, node n_3 as an access point around the user is really not acceptable.

*3) **Distributed SmartBuoy-Reliability Algorithm**:* Algorithm 2 depicts the steps of distributively shaping the service coverage guided by the reliability mentioned above computing processes. Given the bandwidth PDF $p_{b_{c-1,c}}$ observed on node n_c, CDF $F(c-1, b_x)$ received from node n_{c-1}, minimal bandwidth b_{min} calculated by (5), and the reliability threshold R_s^{THR} for service s. We first compute $F(c, b_x)$ given $F(c-1, b_x)$ and $p_{b_{c-1,c}}$ by (9) (line 4). To avoid the forwarding process endlessly, we perform the reliability-based stop condition defined in 5 (lines 1∼3, 5, and 6∼8). If the stop condition is not satisfied, we continually forward the updating action to its next neighbors (line 9).

Complexity analysis for SmartBuoy-Reliability. In Algorithm 2, we first calculates a_c when receives a status packet containing the CDF information (i.e., the value of $F(c-1, b_x)$) from the last hop n_{c-1}, whose time complexity is $O(1)$. In details, calculating $F(c, b_x)$ according to (9) costs $O(1)$ by checking the bandwidth CDF (i.e., TABLE IV) and PDF records of n_c neighbors. After that, calculating a_c according to (10) costs $O(1)$. However, maintaining CDF for a service and observing bandwidth PDF for n_c's neighbors might need $O(l + lh)$, where l and h are the number of bandwidth ranges (or columns) and neighbors, respectively. Therefore, the time complexity of SmartBuoy-Reliability is $O(lh)$. Then, similar to Algorithm 1, the time complexity of updating an entry in metric table is $O(m)$ at the worst case.

For communication overhead, it is still $min(E_a^{d_{s,radius}}, E_a^{d_{s,e}})/E_a^{d_{s,e}}$.

V. AGE OF THE METRIC TABLE ANALYSIS

Distributively synchronizing the service status, i.e., proactively updating the metric table, has a challenge that makes

Algorithm 2 SmartBuoy-Reliability. Shaping the Service Coverage Distributively: A Single Node View (on Node n_c)

Require: $p_{b_{c-1,c}}$, the PDF of the current node n_c; $F(c-1, b_x)$, the CDF for the last hop of n_c; b_{min}, the minimal bandwidth requirements given the TS service; R_s^{THR}, reliability threshold;

```
if action looped then
    stop forwarding the metric-table-updating action
end if
calculates F(c, b_x) by (9)
calculates a_c according to (10)
if the reliability-based stop condition (Def. 5) is met then
    stop forwarding the metric-table-updating action
end if
if the service s in metric table exists then
    update a_c
else
    insert a new entry {s, n_s, a_c} into metric table
end if
forwards F(c, b_x) to its next neighbors n_{c+1}
```

the updating and synchronizing time interval T hard to decide. On the one hand, a smaller T may cause communication a little suffering and energy cost. On the other hand, a larger T may cause the service capability to have a lagged effect, leading the users to read the outdated status. **How often should the metric table be updated? And does there exist an upper bound?**

One feasible solution is to adopt a feedback mechanism, e.g., ACK mechanism, to dynamically change T. However, that complicates the implementation and might not be suitable for an unstable environment. In this section, we provide a one-way propagation analysis, i.e., without feedback support. Background of the analysis in this section can refer to AoI [58]. Typically, AoI is an end-to-end performance metric used to describe the latency when updating the system's monitored status. We consider a single service node with its service coverage to derive T.

In SmartBuoy, as shown in Fig. 7, we divide the main process into three phases: (1) a service node (source node) triggers a metric-table-updating action and sends calculated updates to a network for delivery to the forwarding or AP nodes (AP for short), costing time T_1. (2) After the metric table has been updated, T_2 will be elapsed until the users access it. (3) Transmitting user's request costs T_3.

During phase (1), updates traverse a route consisting of forwarding nodes, each of which is also a destination. To simplify the model, we provide an end-to-end example. When users access the service through the AP, we use a M/D/1/2/N/D queueing system following Kendall's notation. In this system, user accesses follow a Poisson process with the parameter λ, while service computing delays have a deterministic time of D. A single service node in the service coverage serves two entities at a time from the front of the queue, according to a First-Come, First-Served (FCFS) discipline.

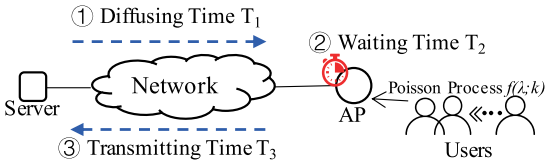


Fig. 7. Phases of the age of metric table.

*1) **Modeling the Service Provisioning to Be Atomic**:* Denote by $T_{age} := T_1 + T_2 + T_3$ the total time elapsed for the service provisioning process before the requests reach the server (see Fig. 7). From now on, our objective is to analyze T_{age} to ensure the metric table is not outdated. Keeping the metric table fresh means that during the period T_{age}, no other users are accessing the service concurrently except the allowed users; otherwise, T_{age} will be outdated. We can see that if we allow unexpected requests during T_1, the metric table for the expected users will be outdated. Similarly, if we allow unexpected requests during T_3, the expected users may be discarded due to the service being overloaded. In other words, we could assume *the process to be atomic*.

In order to get a more general result, we allow at most N users can access the service during T. N can also be regarded as the maximal concurrent accesses of the service. Keep in mind that this assumption regards phases 1, 2, and 3 as a locker, which excludes the condition where the server may be overloaded.

T_1 and T_3 are much more stable than T_2 because the latter is related to several random processes modeling users' behaviors. Next, we derive a more general model to estimate the length of T_2 by using several probabilistic and statistical models; then, we combine T_1, T_2, and T_3 to infer the synchronizing time window T.

*2) **Aggregating User Accessing Models & Estimating** T_2:* Recall that users access the service following a Poisson process and are independent of one another. Thereby, the aggregated pattern of the user requesting process still is the same, i.e.,

$$f(\lambda; k) = f(\lambda_1 + \lambda_2 + \cdots + \lambda_U; k) \tag{11}$$

where λ_i is the mean of user u_i's requesting frequency per time unit, $k \in \{0, 1, 2, \dots\}$ the number of occurrences, and U the total number of the users within the service coverage.

Following the Poisson process and the Erlang-k distribution, we thereby get the arrival time y of the kth occurrence PDF,

$$f_{Y_k}(y) = \frac{\lambda^k y^{k-1} e^{-\lambda y}}{(k-1)!}, 1 \le k \le N \tag{12}$$

And the PMF (Probability Mass Function),

$$F_{Y_k}(y) = Pr(Y_k \le y) = \frac{\gamma(k, \lambda y)}{\Gamma(k)} = \frac{\gamma(k, \lambda y)}{(k-1)!} \tag{13}$$

where $\gamma(\cdot)$ is the lower incomplete gamma function. Since we know that exactly k occurrences within a time interval y still follows Poisson distribution with a parameter λy, i.e., $Pr(k, y) = e^{\lambda y} \frac{(\lambda y)^k}{k!}$. Therefore, let $k = N + 1$, by (13) we

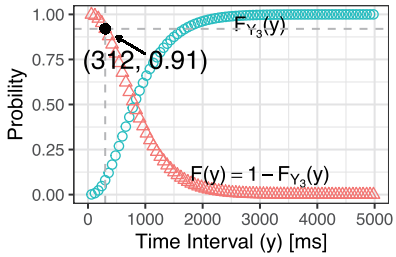


Fig. 8. Probabilities varying y.

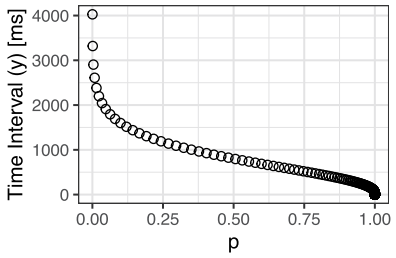


Fig. 9. Curve of $Q(p; 2, 200)$.

could derive the time distribution that no more than N (N is included) users accessing the service, write $F(y)$, is

$$F(y) = 1 - F_{Y_{N+1}}(y) \tag{14}$$

Fig. 8 depicts the probabilities varying y given $\lambda = 200$ and $N = 2$. To plot it, we assume $\lambda_i = 10$ times per minute each user accesses and $U = 20$ users are in the service coverage. Therefore, following (11), we have $\lambda = 200$. From now on, $T_2 = Q(p; N, \lambda)$ denotes the time consumed by U users given the specific probability p, where $\lambda = \sum_{i=1}^{U} \lambda_i$. For example, if we let $p = 0.91$ and $N = 2$, we could say we are 91% sure that we have a time interval of size $T_{Q_{0.91}} \approx 312$ms during which no more than two out of U users accessed the service.

In addition, to easily compute y, the quantile function can be further derived from (14), i.e.,

$$Q(p; N, \lambda) := F^{-1}(y) = \frac{1}{\lambda} P^{-1}(1 - p, N + 1), p \in [0, 1] \tag{15}$$

where P^{-1} is the inverse of the Erlang function (13). Fig. 9 shows the results given $N = 2$, $\lambda = 200$.[5] Obviously, the preferred time interval y decreases as the p increases. We can see that if we want a nearly 100% promise, the updating time interval will be smaller than 1ms.

*3) **Inferring the Synchronizing Time Window** T:* To keep the atomicity, let

$$T_{age} \geq T \tag{16}$$

which means during T at most N requests are allowed. To clearly understand (16), inversely, if we let $T_{age} \leq T$, more than N users may access the service during T, leading to the metric table outdated (records in the table are worthless) and eventually the server overloaded.

In the mentioned-above analysis, we have derived $T_2 = Q(p; N, \lambda)$, and hereafter we estimate T_1 and T_3.

For T_1, the record for each service only contains three items and is relatively small, so we can regard the time for transmitting it roughly as the propagation time. Suppose L hops in the path between the server and the user; therefore we can denote

$$T_1 = T_{hops} \times L \tag{17}$$

where T_{hops} is the minimal time consumed by transmitting a packet from one node to its neighbor. In practice, and in our experiments, T_{hops} is usually equal to 2ms. In SmartBuoy, L has a maximum L_{max}, which is usually easy to get. Thus $T_1 \leq T_{hops} \times L_{max}$.

For T_3, which is relative to the data size D_{raw}. However, for a time-sensitive service, because we have a specific deadline T_s^{THR}, we could limit

$$T_3 \leq T_s^{THR} - T_s^{compt} \leq T_s^{THR} \tag{18}$$

where T_s^{compt} is fixed based on our assumptions in Section IV-A, and $T_s^{compt} < T_s^{THR}$. (18) remove the impact of the dynamically changing bandwidth and gives a simple yet clear upper bound, simplifying the analysis processes. Meanwhile, the upper bound T_s^{THR} actually includes two parts: the transmitting and the computing delays. We draw another conclusion that T_s^{THR} is tight for the sum of phase 3 and the computing delay.

Therefore, substituting (15), (17), and (18) for (16), we have

$$T \leq T_{age} \leq T_{hops} \times L_{max} + Q(p; N, \lambda) + T_s^{THR} \tag{19}$$

which means the upper bound of the synchronizing time window T is $T_{age-bounds} := sup\{T_{hops} \times L_{max} + Q(p; N, \lambda) + T_s^{THR}\}$.

*4) **The Takeaway**:* In SmartBuoy, or similar scenarios such as computing first networking, where U users (each user's accesses follow a Poisson process with a parameter λ_i) access a service that supports at most N concurrences a time and with T_s^{THR} time guaranteeing, we have p sure to say that the updating time interval T of the metric table should be no more than $T_{age-bounds}$, higher than it will lead to a lagged effect.

VI. Experiments

Fig. 10 illustrates a panoramic view of our proposed SmartBuoy framework. We assume a service hub (e.g., the edge cloud) exists, allowing the selected edge nodes (service nodes) to pre-install the needed services. Based on the historical performance data, scheduler on service hub can make a service deployment strategy, such as with the objective of maximize the total services' coverage. See also service placement problem discussion in Section VII. Therefore, during the simulation, we assume the service nodes are given in advance.

We also assume that each node that can provide user network access (access point, AP) has a pre-installed lightweight application. This application can perform our proposed SmartBuoy framework and update indicators such as time and

[5]We thank the authors of `distributions3` (an R package including many useful distributions, https://cloud.r-project.org/web/packages/distributions3), their nice work accelerates our analysis intuitively.

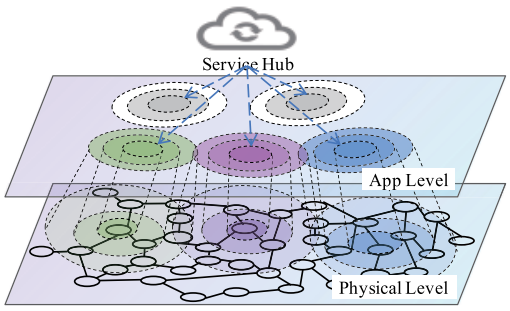


Fig. 10. A panoramic view of the generated service coverage.

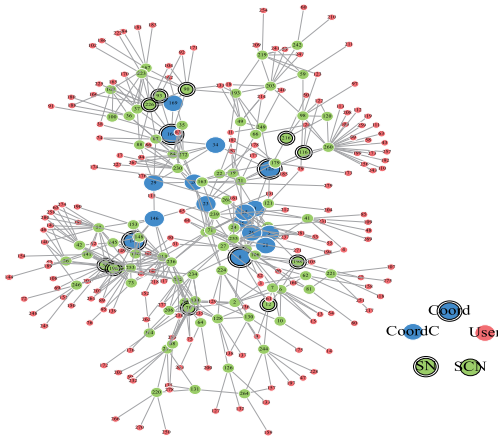


Fig. 11. Rocketfuel dataset.

reliability. In the logic layer, i.e., app-level, service coverage can be built based on the user-specified indicators and the stop condition. Then, the node updates the indicators and sends them to the physical level. At the physical level, we can use traditional communication protocols such as TCP/IP or UDP to recursively and distributively transmit the indicator to the node's neighbors.

We implement and test SmartBuoy distributively on an open-source simulator EasiEI [59]. EasiEI is a discrete-event edge computing simulator based on ns-3, which supports dynamic scenario simulation where the computing, bandwidth, and storage resources can be changeable with time.

A. Dataset and Indicators

a) ***Simulation Dataset and Parameters****:* The trade secret concern makes finding a suitable dynamic edge computing network dataset, including network topology and edge node computing status, complex. Therefore, we examine SmartBuoy by real-world network topology, i.e., rocketfuel [60], comprising 282 nodes, including 177 clients (red), 89 gateways (green), and 16 backbones (blue). The topology is illustrated in Fig. 11. We assume all gateways are service candidate nodes (SCNs) and randomly select ten service nodes (SN) from the candidates.

To get closer to the ground truth and make the computing and bandwidth dynamically change, we let the process of each user accessing the service follow the Poisson distribution

TABLE V
SIMULATION PARAMETERS

Parameter	Default Value	Explanation
$max(\mathcal{B})$	1.5Gbps	Maximum available bandwidth
$max(c_s)$	2	Maximum concurrent accesses of s
λ	10	10 times per minute
D	8Mb	User sends a file of a size D
T_s^{THR}	17 ms	Deadline threshold of service s
R_s^{THR}	0.96	Reliability threshold of service s
T	315 ms	Metric table synchronizing time window
T_s^{compt}	5 ms	Computing delay of service s
Hops delay	2 ms	Delay of hop-by-hop

$f(\lambda;k) = \frac{\lambda^k e^{-\lambda}}{k!}$ with the mean of λ, e.g., ten times per minute, where $k \in \{0, 1, 2 \dots\}$ is the number of occurrences. And the time interval $x \geq 0$ between two consecutive access for each user follows the Exponential distribution $f(x;\lambda) = \lambda e^{-\lambda x}$ with the mean of $1/\lambda$. According to the analysis of T on Section V, we set $T = 315$ms. We simulate for 100 seconds to get the results, which generate about 5,000 requests for each method by default settings.

Notable simulation parameters relating to the simulation dataset are shown in TABLE V.

b) ***Indicators****:* When a user accesses a service s, two statuses may obtain eventually: 1) *Success*, the user acknowledges that she can be successfully served before the deadline and eventually be satisfied. 2) *Failure*, the user perceives in advance at the first AP, or the coordinator, that she can successfully access the service but fails eventually. To evaluate SmartBuoy, we define

$$Success\ Rate = \frac{\#Success}{\#Total\ Requests} \tag{20}$$

as the ratio of users or service requests can be fulfilled within deadline threshold when they access the service.

c) ***Comparison Method****:* We compared SmartBuoy with a (semi-)centralized scheduling method whose idea is adopted from [35], i.e., fog colonies with communication. In this method, we assume the centralized scheduling nodes (coordinator candidate nodes, CoordC) are at least one hop away compared with service node candidates. Similarly, we randomly picked three nodes as the selected coordinators (Coord). Following the steps of the conventional way in Section III-A, users first access the coordinator and then are responded with the service nodes' locations. Second, users directly access the service node guided by the returned locations.

In our experiments, each coordinator monitors 2∼3 nearest (classified by several hops of the shortest path) service nodes. We use a network tool `NetworkX`[6] to extract these relations. Since our models cover effective coverage, users out of the coverage are blinded to the services and send no requests. This is different from the centralized method that provides a global discovering service.

B. Simulation Results

To clearly show the results, we use c_s-Method to denote the method, where c_s is the number of concurrences and

[6]https://networkx.org

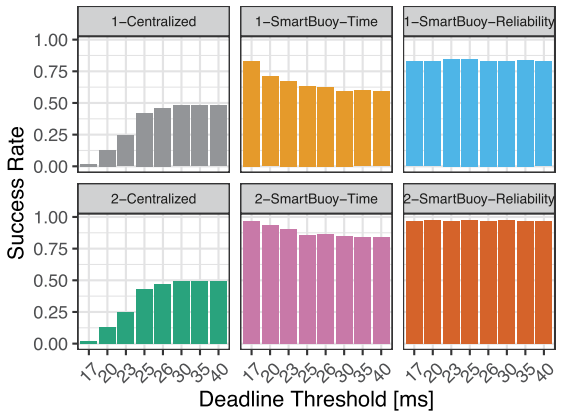


Fig. 12. Success rate w.r.t deadline threshold T_s^{THR}.

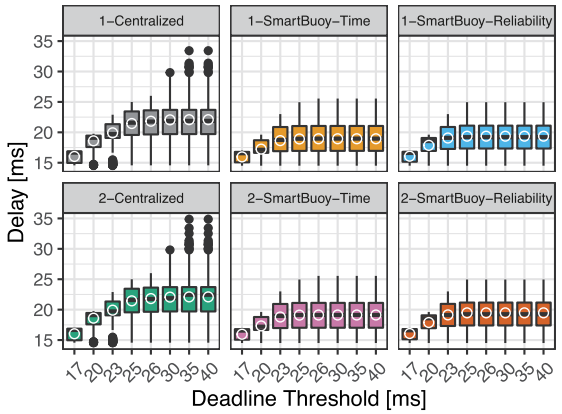


Fig. 13. Service delay w.r.t deadline threshold T_s^{THR}.

the Method is one of the Centralized, SmartBuoy-Time, or SmartBuoy-Reliability. For example, 1-Centralized means the service node can simultaneously provide a maximum of 1 request.

To verify the effectiveness of our methods, we compare the success rate of the three methods varying the deadline thresholds T_s^{THR} indicating different services. Fig. 12 shows the results. Overall, SmartBuoy-Time and SmartBuoy-Reliability under different concurrences have a significantly higher success rate than the Centralized method. Meanwhile, methods with a higher concurrency also have a success rate, indicating that increasing the computing power has a good effect. When we set the deadline threshold below 26 ms, the Centralized method is even worse than our models. The main reason is that the Centralized method introduces additional forwarding delays when handling users' requests. In addition, the success rate of SmartBuoy-Time decreases as the deadline threshold increases. The main reason is that as the deadline threshold increases, the service coverage also increases, leading to the number of served users in the service coverage increase. Due to the dynamic edge environment, SmartBuoy-Reliability shapes a soft and reliable service coverage, and filters many unacceptable requests in advance, making the success rate of SmartBuoy-Reliability significantly higher than both SmartBuoy-Time and Centralized methods.

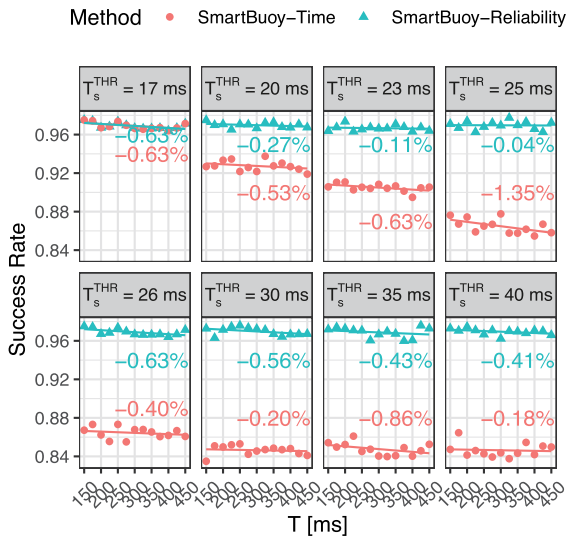


Fig. 14. Success rate w.r.t synchronizing window T.

We also examine the service delays of the successful requests (see Fig. 13). We use Boxplot, where each box summarizes the minimum, first quartile, median, third quartile, and maximum values. The white circle in the box is the mean value of the service delays. The service delays of SmartBuoy-Time and SmartBuoy-Reliability are apparently lower and steadier (no outliers or jitters) than the Centralized. The reason is that in the Centralized method, the number of nodes in the requests forwarding path is higher than in our methods, making the bandwidth more changeable and the queening delays higher, leading to the forwarding and scheduling latencies being higher and not stable. Combined with Fig. 12, SmartBuoy-Reliability has a higher success rate while maintaining the delays at a low level (same with SmartBuoy-Time). Another discovery is that after 26 ms, the success rate of the Centralized for different concurrences becomes steady and similar (slightly but not equal). This also occurred in Fig. 12. We think this is because the coordinators (Coord) become the bottleneck due to the hysteresis effect, limiting the scalability of the services.

Fig. 14 gives the success rate varying the window size T of updating the metric table and the deadline threshold (T_s^{THR}). To clearly show the success rate decreasing trend, we also plot the linear regression. Numbers marked on the figure are the drop difference. We can see that the success rate decreases slightly as T increases, implying that the gains of frequently updating the metric table, e.g., $T = 1$ms, are small. When $T_s^{THR} = 20$ ms, the success rate of SmartBuoy-Time is around 0.92, which matches the analysis on Section V, which is 0.91, see Fig. 8 (in our experiments, the number of the maximal users in Fig. 11 given a service node is about 20, and thus $\lambda = 200$). Another discovery is that the success rate after $T_s^{THR} = 25$ ms becomes steady. The reason is that the service coverage for a service node reaches the maximum and covers the whole network.

Finally, to evaluate the communication overhead of SmartBuoy, we adopt the average coverage radius as a metric, rather than relying on the percentage of utilized links. This

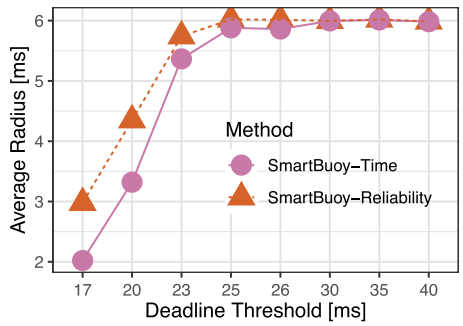


Fig. 15. Communication area w.r.t deadline threshold T_s^{THR}.

approach offers a more meaningful measure of the network's performance, as it directly captures the spatial extent of the network's coverage. Specifically, we calculate the average transmission delays of each request, as depicted in Fig. 15, and use this information to derive the coverage radius.

Our results show that the average coverage radius increases with the deadline threshold T_s^{THR}, reflecting the network's ability to cover a larger area and support a greater number of access points and users. Notably, we observe that SmartBuoy-Reliability exhibits a larger coverage radius than SmartBuoy-Time. This finding may initially seem counterintuitive; however, it is due to the fact that SmartBuoy-Time employs a hard delay to enforce a relatively rigid boundary, whereas SmartBuoy-Reliability utilizes a probabilistic approach to define a more flexible boundary. These results suggest that SmartBuoy-Time can be viewed as a special case of SmartBuoy-Reliability, corresponding to a 2-dimensional cross-section of the latter's 3-dimensional irregular spheres.

VII. Discussion

Conventional methods assume users or applications know the locations providing the services or facilitate a service discovering service. This way regards the users as the centric and may not be suitable for the time-sensitive services in a dynamic environment. SmartBuoy, to some extent, proactively regards the service as its centric and improves the forwarding latencies and the global searching space. However, some directions still need discussion.

*1) **Service Placement Problem (SPP)**:* Placing the services in a given geographical region significantly impacts latency and other QoS metrics. Typically, two categories can solve SPP, including centralized and decentralized methods. The former usually gives optimal results according to QoS but with poorer scalability, and the latter, supported by with or without inter-nodes communication and collaboration, can quickly fulfill QoS, however, with a relatively lower resources efficiency [8].

In our proposed SmartBuoy, size of the service coverage could be changeable due to the dynamic resources. Some methods could be the choices, such as proactively re-deploying the services based on the request prediction [61]. In this paper, we do not focus on solving SPP and assume the cloud or the service providers can solve it. Recently, we are examining a relative stable solution that uses reliability as a metric and the Linear Programing to get the optimal service locations. It can be formulated as a maximum set covering problem. Other decentralized methods to solve SPP may also be adaptable such as [7] uses the accessing popularity in the request forwarding path.

*2) **Dynamic Network Topology**:* In the context of dynamic network topology, network nodes may be intermittent, as is the case in opportunistic networks. Such intermittency can lead to the disruption of communication sessions established between users and service nodes, caused by changes in the session path, addition or removal of forwarding nodes, among other factors. If these disruptions occur while users are being serviced, the user experience may be negatively affected. However, techniques such as SDN can provide a stable and flexible routing path, even with the addition or removal of network nodes.

Moreover, in the event of a disruption during the metric table updating process, a new route can be created after at most a size T of an update window. This can be achieved by using a network flooding mechanism to update the metric table cached on access points, while ensuring global flooding is avoided by adhering to stop conditions proposed in the paper. Consequently, the information reflecting service states and their location in the metric table will become outdated in at most T time units. Therefore, selecting an appropriate T is essential to keep the service requester informed of the service information. To this end, the mobile patterns of wireless nodes may need to be further explored to decide on the update window size T. A smaller size that reflects the reconnection time interval or the disconnection expectation for wireless nodes should be considered. Presently, our framework is designed for a wired network, where computational nodes are stably connected. However, we plan to conduct further research on intermittent patterns in the future.

*3) **Variable Processing Times**:* To simplify the processes, we first assume that the computing time for a service is relatively fixed if the number of sessions/users served does not reach the maximum number $max(c_s)$ of concurrent accesses or the service capacities. This assumption may be reasonable, especially in today's VM/Docker/K8S-supported applications, where the resources consumed by a container can be pre-allocated with a fixed demand. Under this assumption, resources are not preempted.

For the variable times that are unrelated to computation, which may cause the request not to be responded to within a deadline threshold. In our SmartBuoy-Time method, according to the service access capability a_j on an AP node n_j (see Eq. 1), variable processing times might be further added to the computing delay T_s^{compt} using a predicting model. However, it might seem a little complicated. In our SmartBuoy-Reliability method, we focus on the availability $\Pr\{s\ is\ available\}$ of service s given a time period T on node n_s, which can be computed by counting the ratio of successfully served requests based on the historical data. The meaning of $\Pr\{s\ is\ available\}$ implies that failed requests due to virtualization and operating system scheduling are also included. In addition, we need to note that our proposed methods might not reach a global optimal but a relatively reliable result.

*4) **User Accessing Patterns**:* We need to note that our analysis of user accessing pattern on Section V is based on the Poisson process. In the scenarios where users obey other stochastic models, finding the time interval T_2 of two consecutive requests may not be easy. However, the analysis process mentioned in Section V is general.

VIII. Conclusion

In this paper, we have proposed a time-sensitive service coverage concept and a distributed framework SmartBuoy with several design principles. SmartBuoy aims to avoid forwarding delays by introducing a distributed metric table and uses several stop conditions to identify the coverage border. Intuitively, it put the metric table representing the service status into the network edge to let users quickly be responded. Most importantly, to determine the metric table updating frequency, we also provide an analysis methodology and a theoretical upper bound. Results show that SmartBuoy is significantly better than the conventional centralized method.

Future works include designing a self-adaptive T, integrating T with queueing theory, improving the service status diffusing efficiency, exploring inter-service nodes collaboration, examining mobility support, implementing a prototype based on NDN, and solving the service placement problem.

References

[1] F. Mass. (2019). *The Growth in Connected IoT Devices is Expected to Generate 79.4 Zb of Data in 2025, According to a New IDC Forecast*. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS45213219

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016, doi: 10.1109/jiot.2016.2579198.

[3] I. Kunze et al. (Mar. 2022). *Use Cases for In-Network Computing*. Internet Engineering Task Force. [Online]. Available: https://datatracker.ietf.org/doc/draft-irtf-coinrg-use-cases/02/

[4] M. Król, S. Mastorakis, D. Oran, and D. Kutscher, "Compute first networking: Distributed computing meets ICN," in *Proc. 6th ACM Conf. Information-Centric Netw.*, New York, NY, USA, Sep. 2019, pp. 67–77, doi: 10.1145/3357150.3357395.

[5] X. Tang et al., "Computing power network: The architecture of convergence of computing and networking towards 6G requirement," *China Commun.*, vol. 18, no. 2, pp. 175–185, Feb. 2021, doi: 10.23919/JCC.2021.02.011.

[6] B. Nour, S. Mastorakis, and A. Mtibaa, "Compute-less networking: Perspectives, challenges, and opportunities," *IEEE Netw.*, vol. 34, no. 6, pp. 259–265, Nov. 2020, doi: 10.1109/mnet.011.2000180.

[7] Z. Fan, W. Yang, F. Wu, J. Cao, and W. Shi, "Serving at the edge: An edge computing service architecture based on ICN," *ACM Trans. Internet Technol.*, vol. 22, no. 1, pp. 1–27, Oct. 2021, doi: 10.1145/3464428.

[8] I. Murturi and S. Dustdar, "A decentralized approach for resource discovery using metadata replication in edge networks," *IEEE Trans. Services Comput.*, vol. 15, no. 5, pp. 2526–2537, Sep. 2022, doi: 10.1109/TSC.2021.3082305.

[9] I. Murturi and S. Dustdar, "DECENT: A decentralized configurator for controlling elasticity in dynamic edge networks," *ACM Trans. Internet Technol.*, vol. 22, no. 3, pp. 1–21, Aug. 2022, doi: 10.1145/3530692.

[10] M. Król and I. Psaras, "NFaaS: Named function as a service," in *Proc. 4th ACM Conf. Information-Centric Netw.*, Sep. 2017, pp. 134–144, doi: 10.1145/3125719.3125727.

[11] J. Qi and R. Wang, "R2: A distributed remote function execution mechanism with built-in metadata," *IEEE/ACM Trans. Netw.*, vol. 31, no. 2, pp. 710–723, Apr. 2023, doi: 10.1109/tnet.2022.3198467.

[12] A. Yousefpour, G. Ishigaki, and J. P. Jue, "Fog computing: Towards minimizing delay in the Internet of Things," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jun. 2017, pp. 17–24, doi: 10.1109/ieee.edge.2017.12.

[13] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue, "On reducing IoT service delay via fog offloading," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 998–1010, Apr. 2018, doi: 10.1109/jiot.2017.2788802.

[14] J. Santos, T. Wauters, B. Volckaert, and F. D. Turck, "Towards network-aware resource provisioning in Kubernetes for fog computing applications," in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, Jun. 2019, pp. 351–359, doi: 10.1109/netsoft.2019.8806671.

[15] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards delay-aware container-based service function chaining in fog computing," in *Proc. IEEE/IFIP Netw. Operations Manag. Symp.*, Apr. 2020, pp. 1–9, doi: 10.1109/noms47738.2020.9110376.

[16] A. C. Caminero and R. Muñoz-Mansilla, "Quality of service provision in fog computing: Network-aware scheduling of containers," *Sensors*, vol. 21, no. 12, p. 3978, Jun. 2021, doi: 10.3390/s21123978.

[17] L. Toka, "Ultra-reliable and low-latency computing in the edge with Kubernetes," *J. Grid Comput.*, vol. 19, no. 3, pp. 1–31, Jul. 2021, doi: 10.1007/s10723-021-09573-z.

[18] A. Marchese and O. Tomarchio, "Network-aware container placement in cloud-edge Kubernetes clusters," in *Proc. 22nd IEEE Int. Symp. Cluster, Cloud Internet Comput. (CCGrid)*, May 2022, pp. 859–865, doi: 10.1109/ccgrid54584.2022.00102.

[19] U. Ambalavanan, D. Grewe, N. Nayak, L. Liu, N. Mohan, and J. Ott, "DICer: Distributed coordination for in-network computations," in *Proc. 9th ACM Conf. Information-Centric Netw.* New York, NY, USA: Association for Computing Machinery, Sep. 2022, pp. 45–55, doi: 10.1145/3517212.3558084.

[20] Q. Peng, C. Wu, Y. Xia, Y. Ma, X. Wang, and N. Jiang, "DoSRA: A decentralized approach to online edge task scheduling and resource allocation," *IEEE Internet Things J.*, vol. 9, no. 6, pp. 4677–4692, Mar. 2022, doi: 10.1109/jiot.2021.3107431.

[21] I. Dimolitsas, D. Dechouniotis, S. Papavassiliou, P. Papadimitriou, and V. Theodorou, "Edge cloud selection: The essential step for network service marketplaces," *IEEE Commun. Mag.*, vol. 59, no. 10, pp. 28–33, Oct. 2021, doi: 10.1109/mcom.211.2001056.

[22] R. Guerzoni et al., "Analysis of end-to-end multi-domain management and orchestration frameworks for software defined infrastructures: An architectural survey," *Trans. Emerg. Telecommun. Technol.*, vol. 28, no. 4, p. e3103, Sep. 2016, doi: 10.1002/ett.3103.

[23] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, " Multi-probe LSH: efficient indexing for high-dimensional similarity search," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 950–961, doi: 10.5555/1325851.1325958.

[24] M. W. Al Azad and S. Mastorakis, "Reservoir: Named data for pervasive computation reuse at the network edge," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. (PerCom)*, Mar. 2022, pp. 141–151, doi: 10.1109/percom53586.2022.9762397.

[25] L. Zhang et al., "Named data networking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, Jul. 2014, doi: 10.1145/2656877.2656887.

[26] C. Scherb, B. Faludi, and C. Tschudin, "Execution state management in named function networking," in *Proc. IFIP Netw. Conf. (IFIP Networking) Workshops*, Jun. 2017, pp. 1–6, doi: 10.23919/ifipnetworking.2017.8264867.

[27] S. Mastorakis, A. Mtibaa, J. Lee, and S. Misra, "ICedge: When edge computing meets information-centric networking," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4203–4217, May 2020, doi: 10.1109/JIOT.2020.2966924.

[28] M. Krol, K. Habak, D. Oran, D. Kutscher, and I. Psaras, "RICE: Remote method invocation in ICN," in *Proc. 5th ACM Conf. Information-Centric Netw.*, New York, NY, USA, Sep. 2018, pp. 1–11, doi: 10.1145/3267955.3267956.

[29] M. Amadeo, C. Campolo, and A. Molinaro, "NDNe: Enhancing named data networking to support cloudification at the edge," *IEEE Commun. Lett.*, vol. 20, no. 11, pp. 2264–2267, Nov. 2016, doi: 10.1109/lcomm.2016.2597850.

[30] P. Simoens et al., "Service-centric networking for distributed heterogeneous clouds," *IEEE Commun. Mag.*, vol. 55, no. 7, pp. 208–215, Jul. 2017, doi: 10.1109/mcom.2017.1600412.

[31] D. Griffin et al., "Service-centric networking," in *Handbook of Research on Redesigning the Future of Internet Architectures*. Hershey, PA, USA: IGI Global, 2015, pp. 68–95, doi: 10.4018/978-1-4666-8371-6.ch004.

[32] Z. Wang, D. Zhang, and H. Xia, "On the design of software-defined service-centric networking for mobile cloud computing," in *Proc. IEEE 25th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2019, pp. 1000–1005, doi: 10.1109/icpads47876.2019.00153.

[33] M. Gasparyan, T. Braun, and E. Schiller, "IaDRA-SCN: Intra-domain routing architecture for service-centric networking," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, May 2018, pp. 1–6, doi: 10.1109/iccw.2018.8403718.

[34] B. Liu, J. Mao, L. Xu, R. Hu, and X. Chen, "CFN-dyncast: Load balancing the edges via the network," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Mar. 2021, pp. 1–6, doi: 10.1109/wcncw49093.2021.9420028.

[35] Z. Á. Mann, "Decentralized application placement in fog computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 3262–3273, Dec. 2022, doi: 10.1109/tpds.2022.3148985.

[36] J. Ott, E. Hyytiä, P. Lassila, J. Kangasharju, and S. Santra, "Floating content for probabilistic information sharing," *Pervas. Mobile Comput.*, vol. 7, no. 6, pp. 671–689, Dec. 2011, doi: 10.1016/j.pmcj.2011.09.001.

[37] R. Yamamoto, A. Kashima, T. Yamazaki, and Y. Tanaka, "Adaptive contents dissemination method for floating contents," in *Proc. IEEE 90th Veh. Technol. Conf. (VTC-Fall)*, Sep. 2019, pp. 1–5, doi: 10.1109/vtcfall.2019.8891485.

[38] R.-I. Ciobanu, C. Negru, F. Pop, C. Dobre, C. X. Mavromoustakis, and G. Mastorakis, "Drop computing: Ad-hoc dynamic collaborative computing," *Future Gener. Comput. Syst.*, vol. 92, pp. 889–899, Mar. 2019, doi: 10.1016/j.future.2017.11.044.

[39] J. P. Champati, H. Al-Zubaidy, and J. Gross, "Transient analysis for multihop wireless networks under static routing," *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 722–735, Apr. 2020, doi: 10.1109/tnet.2020.2975616.

[40] M. E. Porter and J. E. Heppelmann, "How does augmented reality work?" *Harvard Bus. Rev.*, vol. 95, no. 6, p. 58, 2017. [Online]. Available: https://hbr.org/2017/11/how-does-augmented-reality-work

[41] M. Xu et al., "From cloud to edge: A first look at public edge platforms," in *Proc. 21st ACM Internet Meas. Conf.*, Nov. 2021, pp. 37–53, doi: 10.1145/3487552.3487815.

[42] C. Moreno, Z. Allam, D. Chabaud, C. Gall, and F. Pratlong, "Introducing the '15-minute city': Sustainability, resilience and place identity in future post-pandemic cities," *Smart Cities*, vol. 4, no. 1, pp. 93–111, Jan. 2021, doi: 10.3390/smartcities4010006.

[43] B. Cohen et al. *Cloud Edge Computing: Beyond the Data Center*. Accessed: May 16, 2022. [Online]. Available: https://www.openstack.org/use-cases/edge-computing/cloud-edge-computing-beyond-the-data-center/

[44] L. Corneo et al., "(How much) can edge computing change network latency?" in *Proc. IFIP Netw. Conf. (IFIP Networking)*, Jun. 2021, pp. 1–9.

[45] H. Jin, L. Jia, and Z. Zhou, "Boosting edge intelligence with collaborative cross-edge analytics," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2444–2458, Feb. 2021.

[46] M. Amadeo, G. Ruggeri, C. Campolo, and A. Molinaro, "IoT services allocation at the edge via named data networking: From optimal bounds to practical design," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 2, pp. 661–674, Jun. 2019, doi: 10.1109/tnsm.2019.2900274.

[47] S. Dustdar, V. C. Pujol, and P. K. Donta, "On distributed computing continuum systems," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 4092–4105, Apr. 2023, doi: 10.1109/tkde.2022.3142856.

[48] Y. Zhang, A. Afanasyev, J. Burke, and L. Zhang, "A survey of mobility support in named data networking," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2016, pp. 83–88, doi: 10.1109/INFCOMW.2016.7562050.

[49] L. Tan et al., "In-band network telemetry: A survey," *Comput. Netw.*, vol. 186, Feb. 2021, Art. no. 107763. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128620313396

[50] D. Bonchev and G. A. Buck, *Quantitative Measures of Network Complexity*. Boston, MA, USA: Springer, 2005, pp. 191–235, doi: 10.1007/0-387-25871-X_5.

[51] M. Rausand, A. Barros, and A. Hoyland, *System Reliability Theory: Models, Statistical Methods, and Applications*. Hoboken, NJ, USA: Wiley, 2020, ch. 4, pp. 117–118. [Online]. Available: https://www.wiley.com/en-cn/SystemReliabilityTheory:Models,StatisticalMethods,and Applications,3rdEdition-p-9781119373957

[52] A. Hazra, M. Adhikari, T. Amgoth, and S. N. Srirama, "A comprehensive survey on interoperability for IIoT: Taxonomy, standards, and future directions," *ACM Comput. Surv.*, vol. 55, no. 1, pp. 1–35, Nov. 2021, doi: 10.1145/3485130.

[53] C.-F. Huang, D.-H. Huang, and Y.-K. Lin, "Network reliability evaluation for multi-state computing networks considering demand as the non-integer type," *Rel. Eng. Syst. Saf.*, vol. 219, Mar. 2022, Art. no. 108226, doi: 10.1016/j.ress.2021.108226.

[54] J. E. Ramirez-Marquez and D. W. Coit, "A monte-carlo simulation approach for approximating multi-state two-terminal reliability," *Rel. Eng. Syst. Saf.*, vol. 87, no. 2, pp. 253–264, Feb. 2005, doi: 10.1016/j.ress.2004.05.002.

[55] M. L. Gámiz, F. J. Navas-Gómez, and R. Raya-Miranda, "A machine learning algorithm for reliability analysis," *IEEE Trans. Rel.*, vol. 70, no. 2, pp. 535–546, Jun. 2021, doi: 10.1109/TR.2020.3011653.

[56] C.-H. Huang, D.-H. Huang, and Y.-K. Lin, "A novel approach to predict network reliability for multistate networks by a deep neural network," *Qual. Technol. Quant. Manag.*, vol. 19, no. 3, pp. 362–378, Oct. 2021, doi: 10.1080/16843703.2021.1992072.

[57] L. Chen, J. Qi, X. Su, and R. Wang, "REMR: A reliability evaluation method for dynamic edge computing network under time constraint," *IEEE Internet Things J.*, vol. 10, no. 5, pp. 4281–4291, Mar. 2023, doi: 10.1109/JIOT.2022.3216056.

[58] R. D. Yates, Y. Sun, D. R. Brown, S. K. Kaul, E. Modiano, and S. Ulukus, "Age of information: An introduction and survey," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 5, pp. 1183–1210, May 2021, doi: 10.1109/jsac.2021.3065072.

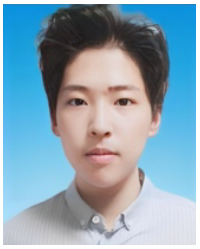
[59] X. Su. (2022). *EasiEI: A Simulator to Modeling Synthetical Edge Computing Environments*. [Online]. Available: https://gitlab.com/Mirrola/ns-3-dev/-/wikis/EasiEI-Simulator

[60] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with rocketfuel," *IEEE/ACM Trans. Netw.*, vol. 12, no. 1, pp. 2–16, Feb. 2004, doi: 10.1109/tnet.2003.822655.

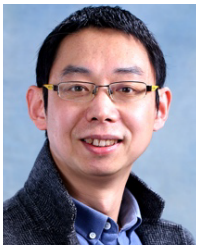
[61] L. Liu, H. Tan, S. H.-C. Jiang, Z. Han, X.-Y. Li, and H. Huang, "Dependent task placement and scheduling with function configuration in edge computing," in *Proc. Int. Symp. Quality Service*, Jun. 2019, pp. 1–10, doi: 10.1145/3326285.3329055.

Jianpeng Qi received the Ph.D. degree in computer science and technology from the University of Science and Technology Beijing, Beijing, China, in 2023.

He is currently a Post-Doctoral Researcher with the College of Computer Science and Technology, Ocean University of China, China. His research interests include edge computing, compute first networking, and distributed systems.



Xiao Su received the B.S. degree in information and computational science from Beijing Information Science and Technology University and the M.S. degree in computer technology from the University of Science and Technology Beijing, Beijing, China, under his advisor Prof. Rui Wang. His research interests include edge computing, simulation, and machine learning.



Rui Wang received the Ph.D. degree in pattern recognition and intelligent systems from Northwestern Polytechnical University, Xi'an, China, in 2007.

He is currently a Professor with the Department of Computer Science and Technology, School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, China. His research interests include edge intelligence, mobile and ubiquitous computing, and distributed systems.