Electronic Research Archive

http://www.aimspress.com/journal/era

ERA, 32(2): 1160–1190.
DOI: 10.3934/era.2024056
Received: 26 September 2023
Revised: 20 December 2023
Accepted: 08 January 2024
Published: 29 January 2024

Research article

A novel node selection method for wireless distributed edge storage based on SDN and a maldistributed decision model

Yejin Yang[1], Miao Ye[2,3,*], Qiuxiang Jiang[2] and Peng Wen[1]

[1] School of Information and Communications, Guilin University of Electronic Technology, China
[2] Guangxi Key Laboratory of Wireless Wideband Communication and Signal Processing, Guilin University of Electronic Technology, Guilin 541004, China
[3] Ministry of Education Key Laboratory of Cognitive Radio and Information Processing, Guilin University of Electronic Technology, Guilin 541004, China

* **Correspondence:** Email: yemiao@guet.edu.cn.

Abstract: In distributed edge storage, data storage data is allocated to network edge devices to achieve low latency, high security, and flexibility. However, traditional systems for distributed edge storage only consider individual factors, such as node capacity, while overlooking the network status and the load states of the storage nodes, thereby impacting the system's read and write performance. Moreover, these systems exhibit inadequate scalability in widely adopted wireless terminal application scenarios. To overcome these challenges, this paper introduces a software-defined edge storage model and a distributed edge storage architecture grounded in software-defined networking (SDN) and the Server Message Block (SMB) protocol. A data storage node selection and distribution algorithm is formulated based on a maldistributed decision model that comprehensively considers the network and storage node load states. A system prototype is implemented in combination with 5G wireless communication technology. The experimental results demonstrate that, in comparison to conventional distributed edge storage systems, the proposed wireless distributed edge storage system exhibits significantly enhanced performance under high load conditions, demonstrating superior scalability and adaptability. This approach effectively addresses the scalability limitation, rendering it suitable for edge scenarios in mobile applications and reducing hardware deployment costs.

Keywords: distributed edge storage; software-defined networking; node selection algorithm; 5G wireless communication

1. Introduction

With continuous advancements in edge computing [1], edge storage [2], intelligent transportation, and Internet of Things (IoT) technologies, an increasing number of intelligent network devices are being deployed at the edge. This surge in deployment has led to a substantial increase in the demand for edge computing and edge storage technologies in edge scenarios, resulting in a dramatic increase in both traffic volume and the amount of data requiring processing at the edge [3]. The conventional practice of uploading data to the cloud for processing and storage presents challenges, including risks to user privacy and heightened data transmission delays [4]. To address these issues, distributed edge storage technology has emerged as a compelling solution [5,6]. This technology employs a distributed approach in which data are stored across multiple edge devices positioned close to the user. Subsequently, data are selectively transmitted to a cloud center after encoding, effectively mitigating concerns related to data security when transmitting sensitive information to the cloud [7]. Simultaneously, this approach alleviates the burden on network bandwidth, enhances the data access speed, reduces the data transmission latency, and diminishes the risk of a single point of failure.

Additionally, most edge devices are mobile and access the network wirelessly [8]. The generated data must be flexibly deployed in the edge storage nodes, requiring that both the edge network devices and the edge storage nodes have wireless communication capabilities for interaction. The emergence of wireless 5G technology [9] has significantly decreased the latency of wireless communication and increased data transmission bandwidth. Thus, the development of wireless 5G technology has made it possible to combine distributed edge storage with wireless technology, resulting in a wireless distributed edge storage system (WDES).

In summary, it is of great theoretical and practical significance to research ways to enhance the overall performance of WDES systems to cope with the rapid growth of edge traffic. Such a system can assist in alleviating the bandwidth and storage burdens of a cloud center while safeguarding private user data.

In a WDES system, data are distributed on multiple nodes. Due to the diversity of the hardware and software structures available for edge storage nodes, such nodes are usually heterogeneous and have different computing and storage capabilities. When data need to be stored at the edge and a storage node is selected, there is a possibility that the computing and storage capabilities of the selected storage node may be insufficient, network bandwidth resources may be scarce, or the node may be faulty or under heavy load; in such a case, data storage and processing efficiency will deteriorate. Therefore, node selection is a key issue in WDES technology [10] that will directly affect the efficiency of data processing and storage and the overall performance of the system.

Presently, several common approaches are employed for node selection in distributed edge storage systems, including distance-based node selection [11], data type-based node selection [12], and load balancing-based node selection [13]. Distance-based node selection involves selecting the nearest node for data processing and storage based on proximity to the request source. This approach reduces data transmission delays and minimizes network bandwidth utilization. However, it overlooks node computing and storage capabilities, leading to imbalances among storage nodes and affecting system performance.

In data type-based node selection, the most appropriate node is selected for processing and storing specific data types. This approach, however, introduces complexity and computational overhead into the system, as it requires data type analysis. Additionally, it disregards the real-time storage node load status and network conditions. When there are significant disparities in data types among nodes, some nodes may become overloaded, resulting in reduced system reliability, performance, and service life.

In node selection based on load balancing, the aim is to identify the optimal node by considering multiple factors, such as the load status, computing capacity, storage capacity, and network conditions. While this approach can mitigate the limitations of the previous approaches, traditional distributed edge storage systems require complex and resource-intensive configurations to measure the status information of the storage nodes and the network [14,15]. Moreover, real-time monitoring of storage node load status and computing capacity often requires additional monitoring nodes, increasing the deployment cost.

To overcome these limitations and develop an efficient edge storage node selection strategy, it is essential to consider both the states of the edge storage node devices and the network status of the edge network. This requires flexible configuration and management of the edge network. Therefore, the widely adopted Server Message Block (SMB) protocol [16] is adopted as the primary communication protocol in this paper, and a multiattribute decision method is employed to design and implement a WDES node selection algorithm and prototype system based on software-defined networking (SDN) [17].

As an innovative network architecture, SDN breaks the tight coupling between the control plane and the data plane in traditional network devices. Its programmability significantly facilitates the measurement of network system states [18]. Simultaneously, by cleverly utilizing the mechanisms of SDN, it is possible to achieve real-time monitoring of the storage node load status by the controller without the need to add additional hardware for monitoring the nodes. Applying SDN technology in a WDES system enables optimization of the system with a simpler hardware configuration and reduced measurement overhead, enhancing system performance and resource utilization. When clients perform file read and write operations, the system proposed in this paper comprehensively considers the current network state and storage node load status in the system as weighting factors for node selection. Through the designed multiattribute decision model, the weight of each storage node is calculated, ultimately allowing a ranking of the edge storage nodes to be obtained and data to be stored accordingly.

The main contributions of this paper are as follows:

1) In contrast to traditional distributed edge storage systems, which use a single data writing method and have poor scalability of storage capacity, this paper presents the design of a distributed edge storage architecture based on SDN and the SMB protocol that is designed to support wireless 5G communication modes; the design and implementation of a data storage node selection algorithm to address the performance and scalability problems of data writing; and the physical implementation of a prototype WDES system that can better cope with complex edge scenarios and has the advantages of flexibility in deployment in edge scenarios, low costs for installation and maintenance, and ease of expansion.

2) In contrast to traditional distributed edge storage systems that consider only a single factor in storage node selection, this paper considers a multiattribute decision model for the storage node selection problem and presents the design of an improved ideal point solution algorithm. The designed multiattribute decision model comprehensively considers the real-time network status and the real-time load status of the storage nodes, enabling effective enhancement of the write performance for data in distributed edge storage systems.

3) In contrast to the conventional edge storage approach, which neglects the network status, this paper presents a novel software-defined edge storage mode by integrating SDN technology into the edge network. Leveraging the flexibility of SDN in dynamically measuring the network status, we

devise a self-reporting mechanism for monitoring the load status of storage nodes within the SDN architecture. This mechanism enables load status collection without requiring additional dedicated hardware nodes, enhancing system configuration flexibility and reducing the overall system cost.

The remainder of this paper is structured as follows: Section II provides an overview of previous research on distributed edge storage systems. Section III describes the data storage process in edge computing, outlines the challenges of node selection, and describes the foundation for the contributions presented in this study. Section IV comprehensively elucidates the architecture of the proposed WDES system. This section also introduces the proposed storage node selection algorithm and the implementation details of the proposed storage node load self-reporting mechanism. The algorithms and implementations of all these designs are based on the principle of measuring the network state in an SDN-based architecture. In Section V, the paper reports the construction and application of a physical system to serve as an experimental platform to validate both the effectiveness of the proposed algorithm and the reliability of the overall system through relevant experiments. Finally, Section VI provides a conclusion, highlighting the challenges encountered and presenting potential avenues for future research.[1]

2. Related work

This section reviews previous research pertaining to distributed edge storage systems. The principal distinctions between distributed edge storage systems and distributed cloud-centered storage systems lie in their storage locations and usage scenarios. Distributed cloud-centered storage systems are optimal for large-scale data storage and processing, whereas distributed edge storage systems excel in scenarios at the edge where low latency, high reliability, and offline support are imperative.

Rashid [19] et al. designed a distributed edge storage system called EdgeStore. By means of a game-theoretic resource incentive framework, this system addresses the problem of inconsistent storage system availability because edge devices tend not to share storage resources and the challenge of configuring all edge devices in a scalable way to integrate with the storage system. The whole system is divided into a device allocation module, a quality of service (QoS) regulation module, and a failure recovery module. However, the design of this distributed edge system does not consider complex mobile edge scenarios; instead, it relies on wired deployment, which is not only costly but also insufficiently flexible. Moreover, it considers only storage capacity as the single basis for selecting storage nodes, which is not conducive to load balancing of the system. Makris [20] et al. proposed a lightweight hybrid distributed edge storage framework in which data are migrated close to the farthest user to improve the quality of experience (QoE) for the farthest user, thereby reducing data transfer latency and network utilization. The proposed Edge Storage Component (ESC) utilizes a dynamic lifecycle framework to provide containerized applications with transparent and automated access to remote workloads. The effectiveness of the ESC was evaluated in terms of numerous resource utilization and QoS metrics. Nevertheless, this system is also designed based on a wired scenario, and consequently still suffers from a high deployment cost and a lack of flexibility. Sonbol [21] et al. designed a decentralized distributed storage system named EdgeKV for deployment at the network

[1] Our work reported in this paper won the first prize in South China and the third prize in the national final of the CHINA C4-NETWORK TECHNOLOGY CHALLENGE competition in 2023 (http://net.c4best.cn/newsDetail?id=5600&type=show), as well as the third prize in the national final of the China Future Light-Future Network Technology Innovation Competition in 2023 (www.gfnds.com/comprtition_list/125.html).

edge. Because of the distributed and heterogeneous nature of the edge and its limited resources, cloud-centric systems and frameworks for distributed storage are not applicable at the edge; consequently, EdgeKV outperforms cloud-centric storage in terms of both local and global data access. Moreover, it provides fast and reliable storage by virtue of its location-transparent and interface-based design, utilizing strong consistency features to ensure data replication. However, this work does not consider how to improve the edge storage performance under the influence of mobile nodes in wireless network scenarios, although this system can support better scalability. Xing [22] et al. proposed a distributed multilevel model for dynamic alternative edge computing storage, in which the storage level corresponds to the device level at the edge; when a node's storage space is insufficient, some of the data are deleted from the current node using the multiple-factors least frequently used (mLFU) algorithm and are uploaded to a higher storage level. This system is designed using a multilevel storage model, which necessitates data migration when nodes have insufficient storage space; however, the wired network approach restricts the flexibility of storage node expansion and increases costs. Qiao [23] et al. proposed a reinforcement learning-based trusted distributed edge storage architecture to solve the problems of heterogeneous transportation network coordination arising in intelligent transportation systems (ITSs). This architecture uses reinforcement learning based on trustworthiness and popularity to dynamically store data, thereby improving resource scheduling and storage space allocation. These authors also proposed an authentication protocol based on trapdoor hashing to protect access to the transportation network, effectively solving the problem of data transmission efficiency in ITSs. However, this system does not consider the heterogeneity of the storage nodes or the load imbalance that may result when selecting data storage nodes for distributed storage. Li [24] et al. employed an iterative technique to analyze malicious signals in edge wireless sensor networks. Li [25] et al. provides a detailed exposition of key aspects within the field of systems and control. Kontodimas [26] et al. developed a mechanism to perform resource allocation at the edge using a mixed integer linear programming approach. This mechanism considers the different characteristics of edge and cloud resources to decide whether to store data in either edge or cloud resources; additionally, corrective censoring techniques are utilized to enhance the system availability and lifetime. The system mainly considers the locations of applications and edge storage nodes for data placement to increase the access bandwidth and reduce the latency for edge data; however, this method does not consider the real-time load states of the storage nodes or the status information of the network. Moreover, the use of only location information for making data storage decisions can lead to problems of load imbalance and overconsumption among the nodes. Wu [27] et al. designed a new decentralized distributed edge storage system, DSPR, based on the provision of cryptographic proofs for stored data. This system can proactively find corrupted data and recover previously unrecoverable data in vulnerable environments. Li [28] et al. address the rapidly growing edge network storage and management problem by proposing an approach that utilizes both data popularity (for optimal data access performance) and data similarity (for optimal storage space efficiency) to jointly solve the node selection and data storage problems of distributed edge storage systems. The proposed algorithm was prototyped using Cassandra, an open-source distributed storage system, which effectively reduces the service request response time. Nevertheless, this method of selecting storage nodes using data types does not comprehensively consider the load states of storage nodes or the real-time network status and thus is not conducive to load balancing of the system.

Storage node selection and load balancing are highly important for a distributed edge storage system, and the results of storage node selection will directly affect the overall performance of the system and the lifetimes of individual devices [29]. The WDES system designed in this paper uses SDN technology to measure the real-time network state and node load states in the system, thereby providing more comprehensive decision-making information for node selection and data placement and reducing data access latency. Moreover, wireless 5G technology is adopted to enable smart devices and storage nodes to interact wirelessly with the system, greatly enhancing the system's flexibility and scalability. Thus, the overall performance of the WDES system is improved, and the service life of the equipment is extended.

3. The designed distributed edge data storage process based on the SMB protocol and the storage node selection problem

This section delineates the devised process for distributed data storage at the network edge, leveraging the SMB protocol. It expounds upon the research contributions presented in this paper, encompassing the selection of nodes for distributed storage at the edge, the state assessment mechanism, and the system architecture. Given the contextual framework of edge storage, the primary emphasis lies in addressing the storage node selection dilemma. In this context, the wireless storage system designed in this study segments a file into multiple smaller entities and subsequently allocates them to distinct storage nodes in a nonredundant manner. It is imperative to emphasize that this approach does not impede the applicability of the present findings to scenarios involving multiple replicas or error-correcting code-based distributed storage configurations.

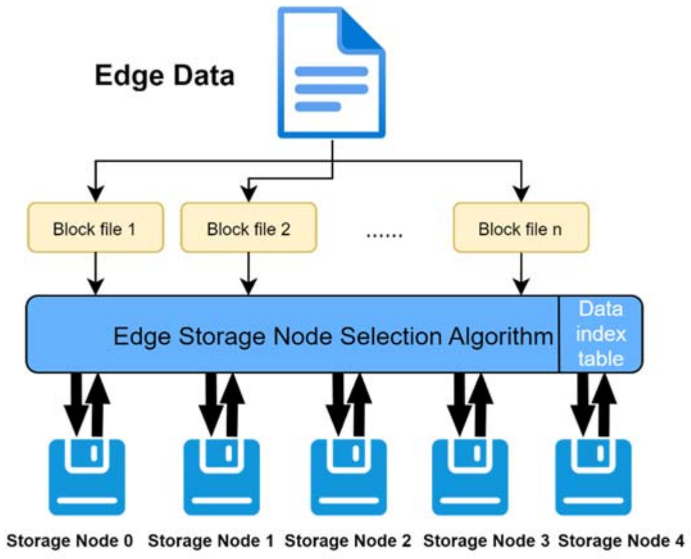


Figure 1. Process of distributed edge data storage.

In conventional distributed edge storage systems, the primary factor considered when selecting storage nodes is their storage capacity. This factor is employed to determine the final priority ranking of storage nodes. However, relying solely on storage capacity as the basis for node selection can give rise to various issues. For instance, the disparities in storage capacity among nodes can be significant, and some nodes may experience heavier loads than others. Such an imbalance has adverse repercussions on system performance, reliability, and service life, resulting in prolonged periods during which certain nodes' storage capacity remains underutilized. This idle state not only wastes storage resources, but also diminishes storage efficiency. Furthermore, neglecting to account for the actual network state and the load status of storage nodes can have detrimental consequences. In scenarios with heightened system loads, such as during client data read/write operations or system-wide data migration, when the network is congested and the storage nodes are under heavy loads, persisting in prioritizing storage node selection based primarily on capacity can severely impact the system's read/write performance and overall storage efficiency.

To address these shortcomings, both network state and storage node load factors are integrated into the node selection method in this study. The comprehensive process of distributed edge data storage is illustrated in Figure 1.

Under the premise that the generated edge data constitute a sizable file, the system initially divides this file into a series of smaller units. Subsequently, in accordance with the prescribed node selection algorithm, these smaller files are allocated to selected storage nodes possessing sufficient remaining storage capacity to meet the stipulated criteria.

Let M be the size of the large file F, and let the set of suitable storage nodes with available remaining capacity be expressed as $S = \{s_1, s_2, \cdots s_n\}$, where n denotes the number of storage nodes. The large file must be divided into n smaller files, where the size of small file f_i is represented by m_i, such that $m_1 + m_2 + \cdots + m_n = M$. To determine a reasonable file division method and the storage locations of the data blocks obtained after file division, it is necessary to consider the load on the host where each storage node is located and the network loads associated with those hosts; corresponding decisions cannot be made by considering only the remaining available storage space on each storage node. For instance, when a storage node is not performing well and the network state is poor, this node should be allocated less chunked file storage. Conversely, when a storage node is well loaded and the network is in a good state, this node should be allocated larger file chunks for storage. When two storage nodes have similar network states and load states, data chunks of similar size should be allocated to them for storage. This not only allows the load on the system to be as balanced as possible to reduce the time it takes to write data, but also keeps the spatial distribution of the data as even as possible.

The node selection problem involves determining the size m_i of each small file f_i, as well as the storage node s on which each small file is to be stored, that is, the design of a reasonable node mapping mechanism map. This mapping mechanism is expressed as follows (Eq (1)):

$$Node\_list = map(network\_state, node\_state, S) \tag{1}$$

Here, $node\_list$ represents the list of all storage nodes included in the results of node selection. The mapping function map is designed to obtain the node selection results by considering multiple factors related to the nodes and will be explained in a subsequent section. S denotes the collection of storage nodes that satisfy the remaining storage capacity requirements. $network\_status$ and $node\_status$ are two key factors used in the implementation of the node selection algorithm. $network\_status$ encompasses three aspects, namely, bandwidth, latency, and packet loss rate, while $node\_status$ includes four aspects, namely, disk I/O load, the remaining storage capacities of the nodes, CPU utilization, and memory utilization. By comprehensively considering the real-time network status and the real-time load status of the storage nodes, a final ranking of the storage nodes is obtained. Based on the relative superiority and inferiority among all participating storage nodes, the proportions into which the input storage file should be divided are determined for subsequent storage at the corresponding storage nodes. Accordingly, the sizes of the file blocks and the locations for storing the segmented files are determined. With this information, the indices of the segmented files can be saved, and when file retrieval is needed, it is sufficient to access the corresponding index table. Considering multiple factors in this way can alleviate the shortcomings of traditional distributed edge storage in terms of node selection, enhance system load balancing, and reduce data response times. The specific storage node selection algorithm is described in detail in Section 4.2 of this paper.

In addition, in traditional distributed edge storage system architecture, it is very cumbersome to perform $network\_status$ and $node\_status$ measurements, and such measurements consume considerable resources and incur considerable overhead. By comparison, the emergence of SDN technology has greatly reduced the difficulty and overhead of obtaining this information; therefore, this paper considers an SDN-based approach for the fast acquisition of $network\_status$ and $node\_status$ information at a low cost.

Upon completion of the node selection process, the system proceeds to transmit the data blocks of the file to be stored to the chosen nodes via the network. For reliable data transmission, conventional network transmission protocols such as TCP/IP [30] may be employed. Due to the architectural redesign, however, several previously utilized data storage communication protocols are no longer applicable. As preliminary work for this study, a range of currently prevalent file transfer protocols were meticulously examined, and the SMB protocol [31] was ultimately chosen as the foundational communication protocol for the system devised herein, with the specific implementation using Samba. Once a data block reaches its designated node, it is stored on the node's local storage device, which may be a disk, flash card, or other storage medium. Concurrently, the system maintains a data index to log each file block's location and pertinent details. This ensures swift and efficient retrieval of the necessary data blocks when the file needs to be accessed, representing the culmination of the distributed edge storage process.

One notable advantage of the WDES system presented in this study lies in its scalability. Traditional distributed edge storage systems tend to prioritize system reliability while inadvertently neglecting scalability. As such a system approaches full capacity or the complexity of the edge environment increases, the original hardware and deployment approach may no longer suffice. In such a case, the system's adaptability to the evolving edge scenario greatly depends on its scalability. A primary impediment to scalability often arises from reliance on wired deployment methods. To address these concerns, this paper introduces a wireless communication protocol within the novel architecture of the distributed edge storage framework. This innovation enables storage nodes and their associated data to interact with the system wirelessly. In instances where the number of storage nodes needs to be expanded, new nodes can be seamlessly integrated using wireless 5G technology. This enhancement significantly improves the system's scalability and alleviates the complexities associated with system deployment and maintenance.

4. Design and implementation of the proposed wireless distributed edge storage system and edge storage node selection algorithm

This section is divided into four parts. First, the overall architecture of the WDES system designed in this paper is discussed. Next, the storage node selection algorithm, which is designed based on a multiattribute decision-making model and incorporates a self-reporting mechanism for storage node load status, is introduced. Finally, the specific implementation of the network state measurement function is described in detail.

4.1. Design of the system architecture

The architecture encompasses several essential components, including the control and decision plane, the monitoring plane, the data forwarding plane, the data storage plane, and client software. A comprehensive diagram depicting this structure is presented in Figure 2, detailing each of these components.

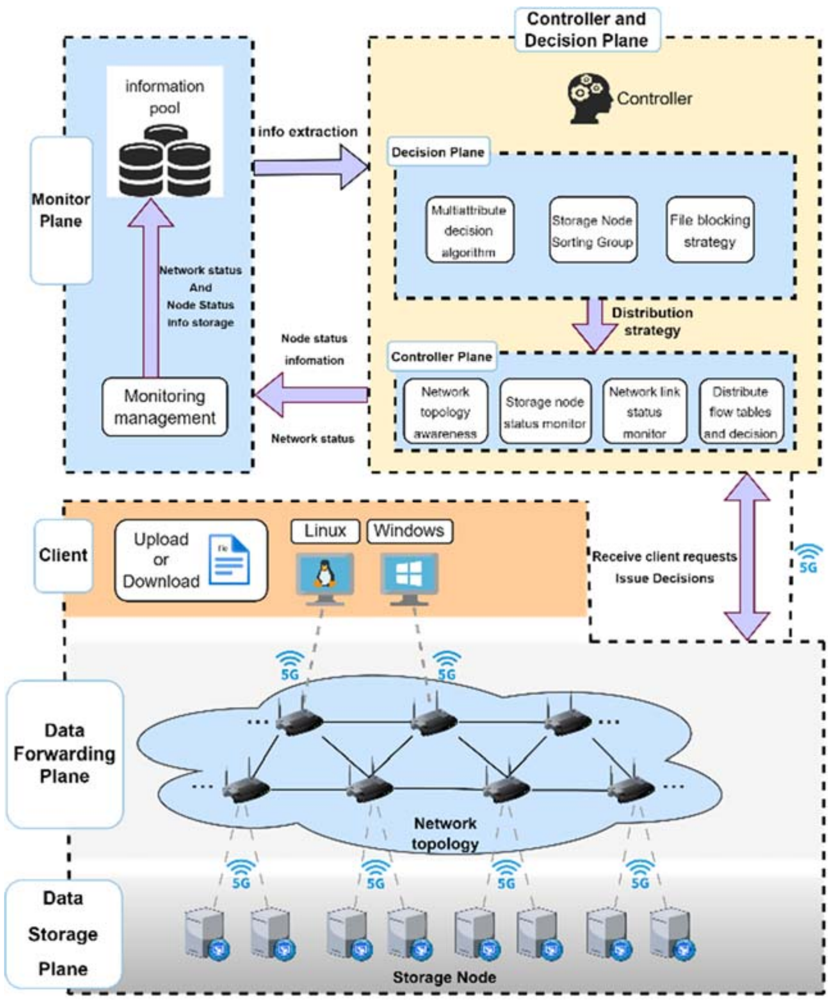


Figure 2. Overall framework diagram of the WDES system.

4.1.1. Control and decision plane

The core component of the system framework is the control and decision plane (CDP). This component assumes a central role in facilitating network topology discovery [32], retrieving the global network link states and global storage node load states, making determinations on client requests, and overseeing the operation of the entire network. To achieve comprehensive management, this plane leverages the characteristics of decoupling and centralized control offered by SDN technology in both the control and data planes, accordingly utilizing a controller for global oversight.

The controller's role encompasses the measurement and acquisition of comprehensive network link state information. Different from previous studies focusing solely on a single network state indicator, this paper comprehensively addresses three pivotal indicators reflecting the link state: the remaining bandwidth, bw_{remain}; the link latency, $delay$; and the link packet loss rate, $loss$. These factors collectively influence the state of a link from a data source to a storage node. When a storage request is initiated by a client, the system will execute the node selection algorithm based on multiattribute decision-making, as discussed in Section 4.2.

Merely obtaining network state information falls short of addressing the inherent limitations of traditional distributed edge storage systems. Thus, the controller is defined in this paper as a central intelligence that is responsible not only for gathering network link state information, but also for aggregating critical load data from storage nodes within the monitoring scope. In this design, the controller does not directly measure the vital load data of the storage nodes; instead, these data are autonomously assessed by the storage nodes themselves and communicated through the load self-reporting mechanism proposed in this study. Further details about this mechanism are presented in Section 4.3.

4.1.2. Monitoring plane

The monitoring plane comprises a set of functions dedicated to monitoring and overseeing both the performance of the SDN network and the states of the node devices in the storage plane. It facilitates real-time monitoring and management of the SDN network's status, performance, traffic, and individual storage plane devices.

In this paper, the monitoring plane is effectively derived from a segment of the CDP. Comprehensive information regarding the network link status and storage node load status is regularly and consistently gathered. These data are stored in the monitoring plane's information repository, establishing a cohesive and continuous dataset. The monitoring plane is responsible for processing and retaining this information. When decision-making calls for the controller's intervention, this information is retrieved from the repository and supplied to the controller, facilitating appropriate decision-making and the realization of corresponding functions.

4.1.3. Data forwarding plane

The data forwarding plane assumes the crucial role of processing and forwarding data, with its central components being switches. The programmability of SDN fundamentally hinges on the programmability of the data plane, as exemplified by OpenFlow switches [33], which serve as the foundation for a versatile and programmable data plane. This universal programmable data plane empowers users to use software programming to flexibly define data plane operations, encompassing packet parsing, processing, and additional functionalities. The data forwarding plane interfaces with both the controller above and the storage nodes below. The controller, situated within the CDP, administers the SDN switches (data forwarding plane) through diverse management modes, including in-band and out-of-band management [34].

The in-band management mode entails the amalgamation of management data and service data on a singular physical link within the network. This eliminates the necessity for a dedicated management channel, streamlining network management. In contrast, in the out-of-band management mode, a separate physical link is employed for transmitting management data. In the in-band mode, management and service data traverse the same physical link but remain segregated at the software level. This dichotomy ensures separate transmission times, precluding interference. The primary advantage of the in-band mode lies in its substantial reduction in practical deployment costs for WDES systems. Notably, this reduction becomes more pronounced with increasingly complex network topologies. If management data traffic is negligible in comparison to business data, its effect on overall network performance can be disregarded. This scenario aligns well with edge deployment.

Based on the above considerations, the controller of the WDES system designed in this paper employs the in-band approach to oversee the entire data forwarding plane.

4.1.4. Data storage plane

The data storage plane is responsible for file storage and retrieval operations. Additionally, the proactive reporting of critical load status information by each individual storage node occurs in this plane.

In the system proposed in this paper, the SMB protocol is employed for data storage, implemented using dedicated Samba software. Once a client has initiated a file storage request and received the decision outcome from the controller, the file undergoes chunking. Subsequently, the fragmented file interfaces with the relevant storage nodes through Samba, utilizing multiple concurrent threads to facilitate efficient storage completion.

Due to the SDN-based configuration, the storage nodes are incapable of direct communication with the controller. Thus, the conventional approach for collecting the load states of the storage nodes involves deploying a host device as a monitoring node. Periodically, this monitoring node establishes connections to the storage nodes, retrieves their load state information through remote commands, and subsequently stores these data within an information pool. Upon completion of data retrieval, the corresponding connection is terminated. When the controller must make a decision based on the storage node load states, the necessary information is retrieved from the information pool to facilitate decision-making. However, this conventional approach has several limitations:

1) The deployment of a dedicated monitoring node substantially increases the system deployment cost and enlarges the system's physical footprint.

2) Frequent interactions between the monitoring and storage nodes necessitate continuous controller involvement in sensing and transmitting associated flow tables, consuming a nonnegligible amount of network resources.

3) In the absence of hardware alterations, the storage node load data remain confined to the monitoring node. Consequently, the controller is unable to access this information. When researchers require these load data for designing decision-making algorithms, such algorithms must be deployed on the monitoring node. This disperses functionality and diminishes system cohesion.

To address the aforementioned problems, the packet_in mechanism of the controller [35] is innovatively leveraged in this paper to devise an improved solution. This approach circumvents both the issue of network resource consumption for downstream flow tables and the concern of escalating costs due to hardware augmentation. Furthermore, it empowers the controller to access information concerning the load states of storage nodes. This integration facilitates the uniform implementation of diverse researcher-designed algorithms at the controller, thereby enhancing system cohesion. The details of this mechanism are expounded upon in Section 4.3.

4.1.5. Client software

The client software, also referred to as the client for brevity, serves as the actual operating software of the WDES system and is provided to users for their usage. In this study, the proposed client software has been adapted to different operating systems, ensuring its normal operation on both Linux and Windows. The client software primarily accomplishes two functionalities: file storage and file retrieval. Users need to use only the client software for storage and retrieval operations. The client software simply executes the results obtained by the controller when running the node selection algorithm and does not participate in the algorithm's execution. The specific implementation details of the client are discussed in Section 5.1.4.

4.2. Edge storage node selection algorithm based on multiattribute decision-making

The foremost advantage of the WDES system lies in its distributed file storage capabilities. The distributed approach not only ensures high reliability, scalability, and performance, but also provides fault tolerance by dispersing data across multiple nodes. However, in conventional distributed edge storage systems, node selection is based solely on a single factor, namely, the storage capacity. This approach, while ensuring the spatial distribution uniformity of the data, overlooks the crucial influence of both the storage nodes' internal states and the network conditions on the storage performance. In scenarios with strongly heterogeneous nodes, some nodes may possess substantial remaining capacity but suffer high load and exhibit subpar performance due to their internal states. Nevertheless, their large capacity alone may cause them to be prioritized for selection, resulting in a pronounced drop in overall storage system performance.

This section introduces the edge storage node selection algorithm based on multiattribute decision-making that is developed in this study to rectify the shortcomings of the node selection process in conventional distributed edge storage systems. The ultimate goal is to enhance the system's write performance.

4.2.1. Selection of decision variables in the multiattribute decision model

A standard edge storage node essentially operates as a computer. Consequently, the assessment of elements impacting the performance and state of a storage node is analogous to the evaluation of factors affecting a computer's performance and operational load. To accurately gauge the performance and load status of a storage node, this paper employs several metrics to represent the factors influencing the node's performance and load state. The specific names of these metrics and the rationales for their selection are listed as follows:

1) L, the disk I/O load, represents the level of input and output operations being performed on the storage node's disk. A high disk I/O load signifies intensive read and write operations. An elevated load can result in delayed response times to client data requests and, in extreme cases, could lead to failure of the storage node, resulting in data loss. Given that the primary hardware component engaged in client file interaction is the disk, along with other storage devices, the disk I/O load is the most significant factor influencing the process of data read and write operations.

2) V, signifying the available disk capacity, reflects the amount of data that the node can accommodate. When the remaining capacity is approaching zero, persisting in file storage may lead to

data loss, severe performance decline, and potential disk damage. This aspect aligns with considerations in conventional distributed storage systems, and the available storage capacity is consequently retained as a pivotal factor in this study.

3) C, denoting the CPU utilization rate, serves as a crucial gauge of the available performance capacity of the storage node. Excessive background program operations can lead to a notable surge in the CPU utilization rate, resulting in sluggish data processing. In cases of excessively high utilization rates, system crashes and failures may occur.

4) R, represents memory utilization. While the memory utilization rate does not have a direct impact on client read/write performance, excess memory occupation can impede the data caching speed on the storage node. Additionally, memory and CPU operations are interdependent. In scenarios involving files with high bandwidth transfer demands, storage nodes with greater memory capacity and lower utilization may be needed.

The aforementioned influencing factors were chosen based on a comprehensive assessment of the intrinsic load state of a storage node. Moreover, since data must ultimately traverse the network, neglecting to account for network conditions can also impact read/write performance under specific circumstances. For instance, during data migration between storage nodes, there may be an increase in the disk I/O load. However, assessing solely this rise in load while neglecting the real-time transmission bandwidth may not adequately reflect the situation. The increase in the disk I/O load might be relatively minor, yet the link bandwidth for network transmission could be approaching its limit. Given rapid advancements in hardware, the read and write bandwidths of storage devices typically surpass the network transmission capacity. Therefore, in this study, both a storage node's load state and the associated network conditions are considered in the node selection process. Key metrics for evaluating the state of a network link typically include the residual bandwidth bw_{remain}, $delay$, and the packet loss rate $loss$. Therefore, these three factors are also included among the input factors for the multiattribute decision model, ensuring a comprehensive evaluation.

In a complex network topology, numerous potential paths exist from a data source to a given storage node, necessitating a decision on which transmission path to employ. To address this issue, in the method proposed in this paper, all link states in the network are initially acquired. Subsequently, the Dijkstra [36] routing algorithm is applied considering parameters such as the residual link bandwidth bw_{remain}, the link delay $delay$, and the link packet loss rate $loss$ to identify the optimal route from the data source to the target storage node, which is then taken as the data transmission path to this node. The remaining link bandwidth bw_{remain}, link delay $delay$, and packet loss rate on this path are extracted. Finally, the values of these three factors are summed to determine the network state of the storage node, as specified in Eq (2). Notably, among these factors, the remaining link bandwidth serves as a positive indicator, with a higher value being more desirable. In contrast, the link delay and packet loss rate serve as negative indicators, with smaller values being preferable, and a minimum value of zero.

$$P = scale(bw) - scale(delay) - (loss) \tag{2}$$

where scale(x) denotes a function used to normalize the variable x.

The five influencing factors selected in this study offer a comprehensive evaluation of the overall state of a storage node and its network environment. This approach is particularly pertinent in cases of node heterogeneity, where it effectively addresses disparities among storage nodes. Through a thorough assessment of their respective impacts on the performance of the WDES system, the weight rankings of the five indices, in descending order, can be determined as follows: disk I/O load, network state P, remaining disk capacity, CPU utilization, and memory utilization. Additionally, a threshold is established for the remaining disk capacity; should it fall below 5%, the corresponding node will be excluded from the storage process. Failing to do so may adversely affect read/write performance, potentially resulting in data loss.

4.2.2. Implementation of the node selection algorithm based on multiattribute decision-making

Once the five indicators representing the load status of the storage nodes and the network state have been accurately identified, the subsequent step is to perform a comprehensive evaluation based on the attribute values of each storage node. This evaluation entails sorting the nodes based on their states and performance, essentially framing the issue as a multiattribute decision-making problem within the domain of mathematical integration methods. Multiattribute decision-making is firmly rooted in systems engineering, and the Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) [37] method has emerged as a particularly effective approach. TOPSIS involves establishing both positive and negative ideal solutions from a normalized raw data matrix; then, the distances between each solution and these benchmarks can be calculated to form a basis for evaluation.

The primary computational steps of a multiattribute decision-making algorithm are as follows: Initially, the considered indicators undergo normalization and are assigned weights based on their respective importance, yielding a weighted normalization matrix. Subsequently, positive and negative ideal schemes are established by identifying the maximum and minimum values of the indicator parameters within the weighted normalization matrix. Next, the distances to both the positive and negative ideal schemes are calculated to ascertain relative proximity. Finally, the results are sorted accordingly.

In this study, out of the five factors chosen for assessing storage node status, the remaining disk capacity V and the network state P are considered positive indicators, with higher values signifying better performance. On the other hand, the I/O load, CPU utilization, and memory utilization serve as negative indicators, with lower values indicating higher storage node performance. Within the established multiattribute decision model, a negative sign is employed to denote negative indicators, as detailed in Algorithm 1. In the subsequent discussion, the algorithm is elucidated based on the provided diagram.

Step 1 consists of the operations on lines 1–14, which serve primarily to construct the decision matrix M for the storage node weight factors. This matrix is formulated as depicted in Eq (3), where V represents the residual capacity, P denotes the network state, L signifies the disk I/O load, C represents the CPU utilization rate, and R refers to the memory utilization rate. Subsequently, the normalized decision matrix M' is derived through the normalization process outlined in Eq (4). Here, f_{ij} represents the element in the ith row and jth column of the matrix M, and n represents the total number of storage nodes. The next step involves assessing the number of participating nodes and evaluating their remaining capacity. If the remaining capacity of a storage node falls below 5%, this node is immediately eliminated from consideration. In a case in which only one node remains that can participate in storage, further calculations are unnecessary, and the storage node is directly returned.

$$M = \begin{bmatrix} V_1 & P_1 & -L_1 & -C_1 & -R_1 \\ V_2 & P_2 & -L_2 & -C_2 & -R_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ V_n & P_n & -L_n & -C_n & -R_n \end{bmatrix} \tag{3}$$

$$M' = \frac{f_{ij}}{\sqrt{\sum_{i=1}^{n} {f_{ij}}^2}} \quad i = 1, 2 \cdots n; \ j = 1, 2, 3, 4, 5 \tag{4}$$

In Step 2, spanning lines 15–25, the five factors influencing storage node performance are assigned varying weights, with the residual capacity and I/O load bearing considerable significance. To this end, suitable weighting coefficients w are determined, and a normalized weighted decision matrix Z is formulated in accordance with Eq (6). The specific values of the weights in Eq (5) are typically established through a series of experimental iterations.

$$W = \begin{bmatrix} W_V & W_P & W_L & W_C & W_R \end{bmatrix} \tag{5}$$

$$Z = W_j \times M_{ij}' \ \ i = 1, 2 \cdots n; \ j = 1, 2, 3, 4, 5 \tag{6}$$

Step 3, encompassing lines 26–30, is dedicated to determining the positive and negative ideal solutions from the weighted decision matrix Z, as outlined in Eqs (7) and (8).

$$Z^+ = (Z_1^+, Z_2^+, Z_3^+, Z_4^+, Z_5^+) = \max_i \{Z_{ij} \mid j = 1, 2, 3, 4, 5\} \tag{7}$$

$$Z^- = (Z_1^-, Z_2^-, Z_3^-, Z_4^-, Z_5^-) = \min_i \{Z_{ij} \mid j = 1, 2, 3, 4, 5\} \tag{8}$$

Step 4, consisting of lines 31–34, involves computing the distances D^+ and D^- from each storage node to the positive and negative ideal solutions, as expressed in Eq (9).

$$\begin{aligned} D^+ &= (D_1^+, D_2^+, \cdots, D_n^+) & \qquad D^+ &= (D_1^+, D_2^+, \cdots, D_n^+) \\ D_i^+ &= \sqrt{\sum\nolimits_{j=1}^{5} (Z_{ij} - Z_j^+)^2} & \qquad D_i^- &= \sqrt{\sum\nolimits_{j=1}^{5} (Z_{ij} - Z_j^-)^2} \end{aligned} \tag{9}$$

Step 5, spanning lines 35–41, involves the computation of the relative closeness of each storage node to the optimally composed ideal storage node, denoted by C_i^+, as shown in Eq (10). A higher closeness value signifies better performance of the corresponding storage node.

$$C_i^+ = \frac{D_i^-}{D_i^+ + D_i^-} \ \ i = 1, 2 \cdots n \tag{10}$$

Through the steps outlined above, a hierarchical ranking of each storage node is ultimately established within the storage plane.

Algorithm 1 Multiattribute decision-making for node selection

```
Input:
   remaining capacity $V$; disk I/O load $L$; CPU usage rate $C$;
   memory usage rate $R$; weighting coefficients $W_V$, $W_L$, $W_C$, and $W_R$.
Output:
   relative closeness[ ] - A list of the relative closeness values
   between each storage node and the ideal node possessing
   each optimal decision factor value.
1  initialize $W_V$, $W_L$, $W_C$, $W_R$;
2  initialize $nodes\_all\_stats\_list$[];
3  initialize
   $Check\ if\ the\ remaining\ capacity\ of\ each\ storage\ is\ less\ than\ 5\%\ and\ remove\ it\ from\ consideration\ if\ so$;
4  initialize $participate\_in\_storage\_nodes\_list$[];
5  if $Client\ has\ sent\ a\ request\ to\ store\ a\ file\ ==\ true$ then
6        if $node\_list.length() == 1$ then
7            return 1;
8        end
9        do $Using\ V, L, C, and\ R, construct\ a\ weighted\ factor\ decision\ matrix\ M$;
10       for $each\_host\_stats$ in $M$ do
11           for $each\_stats$ in $each\_host\_stats$ do
12               $M'_{ij} = \sqrt{\sum each\_stats^2}$;
13           end
14       end
15       $\boldsymbol{Obtain}\ normalization\ matrix\ M'$;
16       for $each\_host\_stats$ in $M'$ do
17           for $each\_stats$ in $each\_host\_stats$ do
18               $Z_{ij} = M'_{ij}\ x\ W_j$;
19               disk_load_list.append(load_value);
20               remain_capacity_list.append(capa_value);
21               cpu_utilization_list.append(cpu_value);
22               mem_utilization_list.append(mem_value);
23           end
24       end
25       $\boldsymbol{Obtain}\ weighted\ decision\ matrix\ Z$;
26       load_best = max(disk_load_list);
27       remain_capa_best = max(disk_load_list);
28       cpu_uti_best = min(disk_load_list);
29       mem_uti_best = min(disk_load_list);
30       for $each\_host\_stats$ in $Z$ do
31           $D^+ = \sqrt{\sum_j^{stats\_num}(Z_{ij} - Z_j^+)^2}$;
32           $D^+ = \sqrt{\sum_j^{stats\_num}(Z_{ij} - Z_j^+)^2}$;
33       end
34       $\boldsymbol{Create\ all\ hosts'\ relative\_closeness\_list}$[];
35       for $each\_host\_$ in $nodes\_list$ do
36           $C_i^+ = \frac{D_i^-}{D_i^+ + D_i^-}$;
37           Relative_closeness_list.append($c_i^+$);
38       end
39       return relative_closeness_list;
40 end
```

4.3. Design and implementation of the self-reporting mechanism for storage node load status

In the system proposed in this paper, the global network link status is periodically obtained through requests sent by the controller and subsequent measurements. This is an active acquisition method. In contrast, the load status information of all storage nodes is self-measured and reported to the controller by the storage nodes themselves. Communication between the controller and SDN switches is facilitated by the OpenFlow protocol, but direct communication between the controller and the storage nodes connected to the SDN switches is not possible. As mentioned earlier, deploying a separate monitoring node would result in many negative consequences.

To address these issues, this paper presents a mechanism for storage nodes to autonomously report their load status. This mechanism innovatively utilizes the packet_in functionality of the controller. When a storage node needs to report its load status to the controller, it measures its own real-time disk I/O load, remaining disk space, CPU usage, and memory usage through its own commands. It then constructs an empty data packet and encapsulates these load metrics within this packet, which is subsequently sent with its destination IP pointing to the controller. Since there are no matching flow tables in the SDN switches for this packet, it triggers the packet_in mechanism, which directly sends the packet to the controller. With the repetition of this process at regular intervals, the controller can periodically obtain load status information from each storage node device, thereby achieving autonomous load status reporting for the storage nodes. Additionally, this enables the implementation of the multiattribute decision-making algorithm at the controller. The operation of the autonomous reporting mechanism is illustrated in Algorithm 2, and below we provide a further explanation of this mechanism based on the algorithm’s flow.

1) Lines 2–7 iterate through all storage nodes, and each storage node autonomously retrieves its four key load status indicators via a local terminal window.

2) On line 8, using the Scapy library [38], an empty TCP packet is constructed, and the values of the four key status indicators are appended to this packet as its payload.

3) On lines 9 and 10, the storage node’s IP address and the controller’s IP address are established as the source and destination IP addresses, respectively, for the constructed packet, followed by packet transmission.

4) As described on lines 11 and 12, the packet cannot be matched with any preexisting flow tables at the switches and thus immediately triggers a packet_in event, causing the packet to be forwarded to the controller. The controller, at this point, does not initiate flow table actions but rather stores the packet.

5) Lines 13–17 describe the controller-side process of parsing the packet to extract the payload data containing the values of the four key status indicators. After a 3-second delay, the algorithm proceeds to the next data acquisition cycle, continuing in an infinite loop, thus completing the autonomous reporting process for the storage nodes.

Should the controller need to make decisions and issue decision results, the reverse approach is used. An empty TCP packet is constructed, with the decision result appended as the payload. The controller then uses the packet_out functionality to specify the SDN switch port connected to the client requesting file operations, facilitating the forwarding of the decision result to that client.

Algorithm 2 Storage node self-reporting mechanism

```
   Input: All storage node objects
   Output: Load status information for all storage nodes sent to the controller
1  While $True$ do:
2      for $each\_node$ in $all\_nodes\_list$ do
3          Storage node: get remaining capacity $V$, disk I/O load $L$, CPU usage rate $C$,
           memory usage rate $R$;
4          Storage node: Terminal(df -lm, obtain $V$);
5          Storage node: Terminal(iostat -x 1 -t 3, obtain $L$);
6          Storage node: Terminal(top -bn 1 -i -c, obtain $C$);
7          Storage node: Terminal(cat /proc/meminfo, obtain $R$);
8          Storage node: construct an empty TCP packet, loading $V$, $L$, $C$, and $R$;
9          Storage node: set packet.src_ip = nodes_ip and packet.dst ip = Controller_ip;
10         Storage node: send(TCP packet);
11         Controller: trigger Controller.ofp_event.EventOFPPacketIn;
12         Controller: save TCP packet;
13         Controller: analyze TCP packet and extract load data;
14         Controller: obtain this node's $V$, $L$, $C$, and $R$;
15     end
16
17     delay 3 s;
18 end
```

4.4. Implementation of node load and network status measurement functionality

The storage node operating system used in this study is Linux. The measurement of the four indicators of node load is relatively straightforward. The disk I/O load can be obtained by continuously reading with the 'iostat' tool for 3 seconds. The remaining disk capacity can be acquired in Linux using the 'df -lm' command. Similarly, the CPU usage and memory usage can be obtained by using the 'top -bn 1 -i -c' and 'cat /proc/meminfo' commands, followed by the application of regular expressions.

For the measurement of the residual bandwidth on a link, it is assumed that the current need is to measure the link bandwidth between switch swA and switch swB. Let the current moment in time be denoted by $t-1$. The controller simultaneously issues a request to both swA and swB. Upon receiving the request, swA and swB each immediately return their respective total sent byte count tx and total received byte count rx, as well as the timestamp $time_{t-1}$ corresponding to the current time $t-1$ to the controller. After waiting for approximately 1 second, i.e., at time t, the controller issues another request for the retrieval of tx, rx, and $time_t$. To calculate the bandwidth utilized by swA during this time interval from $time_{t-1}$ to t, the total bytes transmitted and received at time $tx+rx$ are subtracted from the corresponding values at $time_{t-1}$, yielding the total traffic for swA during this period. Then, the utilized bandwidth of swA can be determined in accordance with Eq (11). As indicated in Eq (12), the remaining bandwidth for swA is then calculated by subtracting the utilized bandwidth from the maximum link bandwidth of the switch port. The remaining bandwidth for swB can be determined in a similar manner. In this study, a rigorous approach in which the link bandwidth between swA and swB is determined as the minimum value between the two, as per Eq (12), is employed for all data calculations.

$$bw_{used} = \frac{(tx_t + rx_t) - (tx_{t-1} + rx_{t-1})}{time_t - time_{t-1}} \tag{11}$$

$$bw_{remain} = bw_{max} - bw_{used} \tag{12}$$

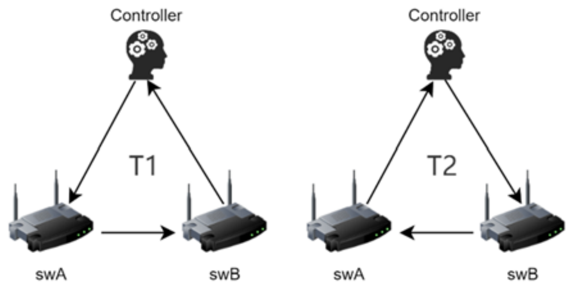


Figure 3. The controller sends packet out packets.

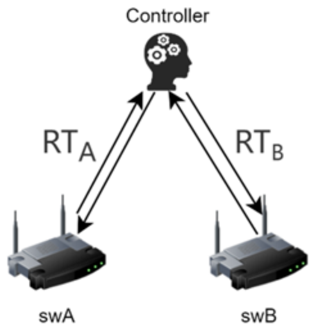


Figure 4. The controller sends Echo request packets.

For the measurement of link latency, the controller issues a packet_out message to swA. The data segment of the message contains the timestamp at the time the controller issued the message. The action specified in the packet_out message instructs the switch to either flood the network with this message or forward it to a specific port. When swB subsequently receives the data packet from swA, no corresponding flow table entry can be matched, triggering the sending of a corresponding packet_in packet to the controller. Upon receiving this data packet, the controller calculates the time difference T_1 by subtracting the current time from the timestamp in the received packet. This time difference is approximately equal to the latency from the controller to swA, plus the latency from swA to swB, and finally the latency from swB back to the controller, as illustrated in Figure 3. Similarly, the controller performs the reverse operation to obtain the time difference T_2, as also shown in Figure 3. The controller then sends Echo request messages to swA and swB, each containing the current timestamp. Upon receiving these messages, the switches immediately reply with reply packets that carry the same timestamp. By subtracting the timestamps in the reply packets from the times of packet reception, the controller can determine the round-trip latencies RT_A and RT_b between the controller and swA and between the controller and swB, respectively, as depicted in Figure 4. Under the assumption that these round-trip latencies are equal, Equation (13) can be used to calculate the link latency $Delay_{AB}$ between swA and swB.

$$delay_{AB} = \frac{T_1 + T_2 - RT_A - RT_B}{2} \tag{13}$$

The measurement method for the link packet loss rate is similar to that for the residual bandwidth. Taking swA as an example, the controller periodically issues requests to obtain the total number of bytes sent (swA_{tx}) and received (swA_{rx}) by swA during the time interval $T_2 - T_1$. The total numbers of sent bytes (swB_{tx}) and received bytes (swB_{rx}) for swB can be determined similarly. Let $loss_{AB}$ denote the packet loss rate in the direction from swA to swB. The final link packet loss rate between swA and swB can be calculated in accordance with Eq (14).

$$loss_{AB} = max(1 - \frac{swB_{rx}}{swA_{tx}}, 1 - \frac{swA_{rx}}{swB_{tx}}) * 100\% \tag{14}$$

5. Implementation and testing of an edge distributed wireless storage system

5.1. Experimental platform design and realization

To validate the efficacy of the suggested multiattribute decision-based node selection algorithm integrated with the node load self-reporting mechanism within the context of the WDES, this section assesses the influence of the devised system and algorithm on the system's read and write performance using a real-world experimental platform.

In contrast to conventional distributed edge storage systems, this study strategically selects and deploys hardware to suit the edge-end scenario, thereby distinctively differentiating all hardware components from other systems. This emphasis on alignment with the edge-end scenario is pivotal. The prototype system encompasses one controller, three storage nodes, and fourteen switches. The software ecosystem includes OpenWrt [39] version 22.03, Open vSwitch [40] version 2.1.3, Ryu version 4.8, OpenFlow version 1.3, and Samba version 4.13. The controller operates on Ubuntu 20.04 LTS, while the storage nodes run on Debian 10. The subsequent section offers detailed insights into the hardware design and implementation, along with an analysis of the conclusive experimental outcomes.

5.1.1. Controller hardware design and implementation

In SDN, controllers have two primary control methods for switches: single-controller control and multi-controller control. In this paper, the focus is primarily on single-controller control. However, considering the system's subsequent upgrade to multi-controller control, the enhancement algorithm amplifies system performance. Consequently, emphasis is bestowed upon multi-port hardware devices. Moreover, given the edge positioning of the system, compact device dimensions are paramount, while performance remains a prerequisite.

Considering the aforementioned criteria, this study ultimately opts for the N5105 six-network port soft routing host from the Smooth Network Micro-Controller Company to serve as the hardware platform for the controller. The soft routing host comes equipped with six 2.5G wired ports and wireless WiFi functionality, with compact dimensions, measuring only 17.8 cm (length) * 12.5 cm (width) * 5 cm (height). This aligns seamlessly with the edge-end system design paradigm. The physical depiction of the host is illustrated in Figure 5.

Figure 5. Physical drawing of N5105 soft route mainframe.

5.1.2. SDN Switch hardware design and implementation

The primary function of an SDN switch is to oversee data forwarding within the physical system, and its performance attributes significantly impact the overall network performance. Presently, SDN switches available in the market cater primarily to expansive cloud storage centers due to their size, rendering them less suitable for mobile applications. Furthermore, the cost of these SDN switches is substantial, often reaching tens of thousands of dollars per unit. As a result, this study confines the hardware selection for SDN switches to commonplace home routers. These routers are compact in design and generally cost-effective, aligning effectively with real-world edge scenarios. Nonetheless, it is important to note that, while home routers are used, they do not inherently possess the capabilities of SDN switches. To harness SDN functionality, a transformation process is requisite.

OpenWrt, a Linux-based open-source embedded operating system, is specifically designed for routers and embedded devices. It offers a flexible, customizable, and scalable platform, serving as a replacement for stock firmware to enhance functionality and control for routers and embedded devices. Open vSwitch (OVS), an open-source virtual switch software, serves as a tool to construct and oversee networks within virtualized environments. It furnishes an extensive array of network features and management utilities for establishing connections with physical networks. The combination of OpenWrt and Open vSwitch constitutes pivotal components in the process of converting routers into SDN switches.

To convert a router using the aforementioned components, several factors must be considered. First, the router should fall within the spectrum of models supported by the OpenWrt system. Second, in real-world distributed edge storage scenarios, SDN switches contend with substantial real data traffic and data forwarding demands. Following the router's transformation, the original dedicated data forwarding chip might encounter performance limitations, or, in some cases, become unusable. Finally, upon integration with Open vSwitch, the router's background resource consumption experiences a significant surge. If the chosen router possesses inadequate performance capabilities, potential consequences include tardy data forwarding responses, excessive resource usage, heightened heat generation, and even system crashes.

To address potential issues arising from router modifications, this study has selected the Xiaomi AX6000 router as the physical SDN switch for this system, as depicted in Figure 6. This router boasts a robust 2GHz CPU frequency, generous RAM and ROM capacities, as well as a compact form factor, aligning seamlessly with the system's edge-end positioning and design philosophy. They provide the critical performance metrics of the router in Table 1. The transformation of the router into an SDN switch is achieved via successive stages, including OpenWrt source code modification, firmware compilation, overriding the original system with burn-in procedures, and installing the Open vSwitch plug-in. This process adeptly equips the router to fulfill SDN switch functions.

Table 1. Xiaomi AX6000 router performance parameters.

model number	CPU Model	CPU frequency	Flash size
AX6000	MT7986A	2 GHz	128 MB

Figure 6. Xiaomi AX6000 router.

5.1.3. Edge storage node hardware design and implementation

Hardware design for storage nodes should align with authentic edge-side scenarios. In practical settings, storage nodes often exhibit heterogeneity, a key distinction from simulation systems, attributed to the following factors:

1) Within identical equipment types, inherent process variations result in certain performance disparities; complete uniformity of performance cannot be attained.

2) Divergent installation sites for storage node devices, varying environmental conditions, and external factors such as heat dissipation lead to progressive performance divergence among these devices over time.

3) In a deployed edge distributed storage system, which may undergo equipment updates and iterations, new and existing equipment coexists within the same distributed storage system, giving rise to a situation of storage node heterogeneity.

Based on the preceding analysis, the heterogeneous nature of storage nodes aligns better with real-world edge scenarios. Consequently, this paper adopts a heterogeneous design for the storage nodes. When selecting hardware, it becomes imperative to opt for devices featuring varying levels of performance to achieve this heterogeneity.

After a thorough comparison, three devices—Asus Tinker Board 2S (No.1), Raspberry Pi 4B (No.2), and Raspberry Pi 3B+ [41] (No.3)—have been chosen as the designated hardware platforms for the storage nodes in the edge-distributed wireless storage system proposed in this paper. The devices are ranked in order of performance: No.1, No.2, and No.3. Their essential parameters are tabulated in Table 2, while they depict their physical embodiments in Figure 7.

Table 2. Comparison of performance parameters of three embedded single-board computers.

Equipment Model	CPU Model	CPU frequency	RAM type	RAM size
Tinker Board 2S	RK3399	2 GHz	DDR4	4 GB
Raspberry 4B	BCM2711	1.5 GHz	DDR4	2–4 GB
Raspberry 3B+	BCM2837	1.4 GHz	DDR2	1 GB

Figure 7. Physical diagrams of the three storage nodes.

5.1.4. Network topology design and physical deployment

The client software interface design is illustrated in Figure 8. Below is a detailed and comprehensive description of the file storage process:

1) Users initiate the relevant client software on their Linux or Windows devices, establishing a wireless connection to the SDN switch through a 5G WiFi connection.

2) Users opt for the storage function, inputting the complete file path (e.g., video file) within the storage window.

3) The client software automatically computes both the file size and filename. Subsequently, it constructs a packet and appends the file size and filename data to the packet as load data before transmitting it.

4) Given that none of the switches within the network topology possesses a flow table entry that aligns with the packet, the packet corresponds to the lowest table miss flow table. As a consequence, the controller side initiates a packet_in event, leading to the packet being conveyed to the controller side. Subsequently, an unpacking procedure is undertaken by the controller to extract the file size and file name, thereby culminating in the comprehensive reporting of the information requested by the client for storage purposes.

5) Upon reception of the storage request from the client, the controller commences the execution of the node selection algorithm (elaborated upon in Section 4.2). Ultimately, the controller attains a result encompassing the file's requisite partitioning into chunks, the individual chunk sizes, and the pertinent IP addresses of the designated storage nodes. Once more, the controller formulates a packet, integrating the obtained result as load data within the packet. This packet is subsequently dispatched back to the client utilizing the packet_out function of the controller.

6) Upon receiving the decision result from the controller, the client promptly initiates the relevant file chunking program. This program generates a chunking information file in TXT format, designed for future file retrieval, and subsequently activates a multi-threaded concurrent transmission program to dispatch the corresponding chunking file to the designated storage node. At this juncture, the switch lacks a flow table. The controller identifies the connection request between the client and the storage node, promptly accessing comprehensive network link state data within the information pool. Employing the Dijkstra routing algorithm, the controller determines an optimally efficient routing and

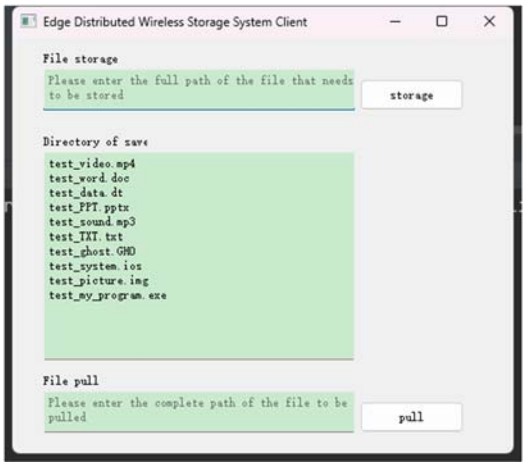
forwarding path, facilitating data transmission from the client (source) to the storage node (destination). The execution of the flow table assignment operation follows this. Sequential storage of the individual chunk files culminates in the successful execution of the complete storage function.

Retrieving stored files involves the following comprehensive process:

1) The client inputs the complete path of a segmentation information file. This file contains details about the file to be retrieved, including the file name, total file size, as well as the names, sizes, and corresponding IP addresses of the segmented files on storage nodes.

2) Upon clicking the pull button, the client autonomously initiates the reading program, which endeavors to establish connections with corresponding storage node hosts for sequential file retrieval. During this process, the switch lacks a relevant flow table, prompting the triggering of the controller's packet_in mechanism. Subsequently, the controller promptly accesses comprehensive network link state information stored in the information pool. Employing the Dijkstra routing algorithm based on these link states, the controller ascertains the optimal short route for data forwarding, spanning from the client (source) to the storage node (destination). This culminates in the execution of the operation to dispatch the requisite flow table.

3) Upon retrieving the chunk file, the file merge program is executed, considering the order of file names (e.g., video_1.mp4, video_2.mp4). The file operation function is utilized to sequentially read each chunk file in binary mode and write it into a new file. Eventually, the complete file name provided within the chunk file's information serves as the designation for the pulled file, ensuring accurate sizing to achieve successful file retrieval.



Please enter the full path of the file that needs to be stored

test_video.mp4
test_word.doc
test_data.dt
test_PPT.pptx
test_sound.mp3
test_TXT.txt
test_ghost.GHO
test_system.ios
test_picture.img
test_my_program.exe

Please enter the complete path of the file to be pulled

Figure 8. Client software interface.

5.1.5. Network topology design and physical deployment

To enhance complexity and replicate a highly realistic edge-end setting, this study configures 14 switch nodes, thereby establishing the comprehensive network framework illustrated in Figure 9. Interconnecting the devices, all SDN switches establish wired connections with each other to bolster reliability. Moreover, while they equipped the physical SDN switches with hardware provisions for wireless networking capabilities, direct utilization is unfeasible. Additionally, an examination of the topology reveals the presence of multiple loop-formed network links, which could potentially trigger broadcast storms [42].

This paper addresses the aforementioned issues by implementing Virtual Local Area Network (VLAN) isolation technology [43]. VLAN divides the physically established LAN into distinct logical subnets, isolating data link layer broadcast messages within these subnets to create individual broadcast domains. Each logical subnet constitutes a VLAN. Terminal devices accessing the VLAN are assigned to specific VLANs, preventing direct communication between devices in different VLANs through the data link layer. By utilizing VLAN technology, broadcast message transmission is confined, mitigating the impact of broadcast storms and enhancing network security. Employing VLAN, the wireless network card within the physical SDN switch can be partitioned into separate VLANs and integrated into the SDN switch, enabling devices beyond the SDN switch to join the distributed edge wireless storage system via wireless network connections. Each connected device operates within its own distinct LAN, preventing mutual interference and effectively mitigating broadcast storm issues within the network.

In the implemented system outlined in this paper, the controller devices, storage nodes, and client devices establish connections to the SDN switch through WiFi wireless 5G connections. This design choice brings the system in closer alignment with edge-end environments, enhancing system scalability. The tangible manifestation of the implemented configuration is depicted in Figure 10.

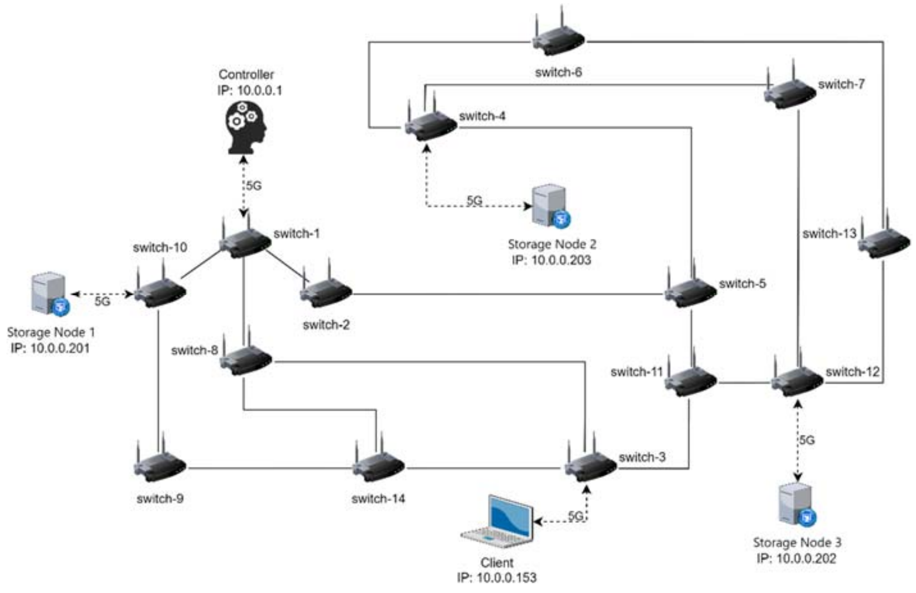


Figure 9. Network topology.

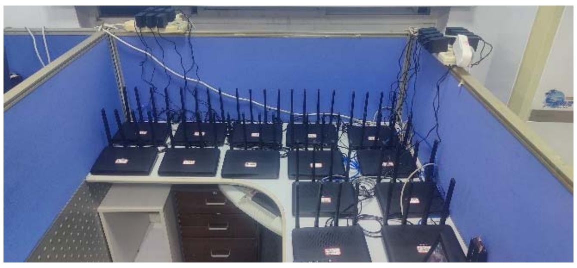


Figure 10. Overall physical picture.

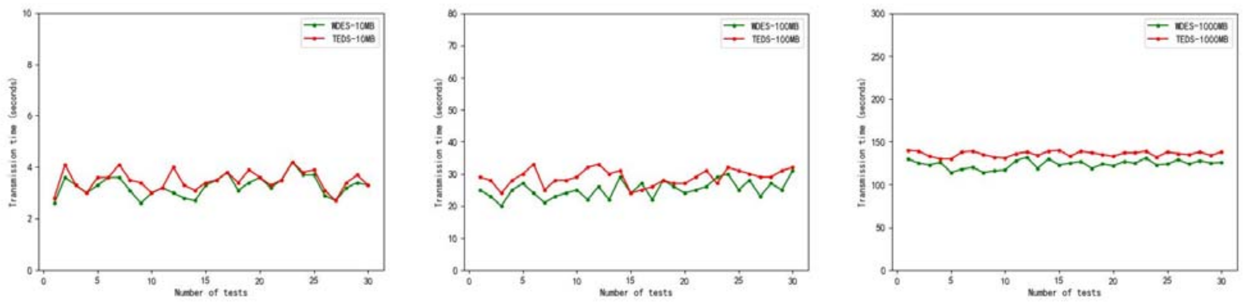
5.2. Performance testing and analysis of results

To evaluate the system's read and write performance for objects of different sizes, the experiment used a comparative method [44] and incorporates three file sizes: 10, 100, and 1000 MB, subjected to storage testing. Each file size undergoes 30 rounds of storage testing. The initiation of storage was marked by the moment the client clicked the storage button, and the completion of file storage marked the end time. The total storage time for each file was obtained by subtracting the start time from the end time. A comparative analysis of the test results was conducted between the wireless distributed edge storage system (WDES) developed in this study and the traditional Edge Distributed Storage System (TEDS).

As this paper is built upon the Samba communication protocol, the performance comparison is conducted within the framework of the Samba protocol [45] to validate the performance of the designed node selection algorithm.

The overall experimental phase is divided into two parts. The first part involves comparing storage times for different file sizes when the network is uncongested and the nodes are in the normal state. The second part includes comparing storage times for different file sizes when the network bandwidth resources are reduced and some of the nodes are in a high load state. This testing approach comprehensively reflects the system's performance in both normal and stressed states.

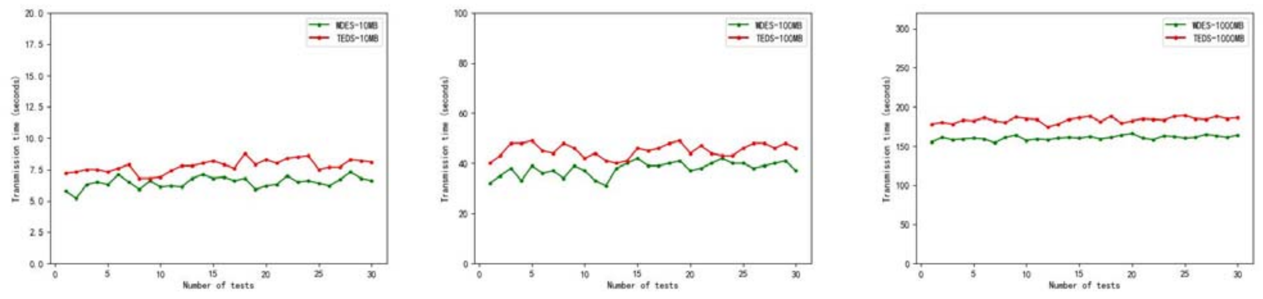
The first is the test with normal network state and node load, where the horizontal coordinate of Figure 11 is the number of tests and the vertical coordinate is the total storage consumption time. Figure 11(a) is a comparison chart of the storage time tested in two different systems under the condition of file size of 10 MB respectively, where the green line is the test result of the system designed in this paper, and the red line is the test result of the traditional edge-distributed storage (without SDN). It can be seen from the experimental results that, in terms of the storage of small files, although there is not a big difference in the time effect of the two comparisons, the overall view of the system designed in this paper's storage time is lower than that of the traditional edge distributed storage system. Figure 11(b) is a comparison chart of the storage time tested in two different systems under the condition of the file size of 100 MB, from which it can be seen that the storage time of the system designed in this paper is lower than that of the traditional edge-distributed storage system for the first 15 times, and from the 15th time onwards there is be an occasional increase in time, which is because the transformed SDN switch does not mobilize the dedicated data forwarding chip aggressively and the long-running time leads to heat generation, which makes the switch processor downclocked and slows down the processing data resulting in longer time. Figure 11(c) illustrates a comparative graph of storage times under the condition of a file size of 1000 MB in two distinct systems. Upon examining the three plots, it becomes evident that, with the increase in storage file size, the storage time curves of the two systems gradually diverge. This implies that, as storage involves progressively larger files, the advantages of the wireless distributed edge storage system designed in this study become more pronounced. This phenomenon arises due to the extended operation of certain physical switches in the network, resulting in heat generation, frequency reduction, and, consequently, temporary processing slowdowns in specific switch nodes. During such instances, the controller measures the bandwidth, latency, and packet loss rates for all links, identifying this phenomenon. Subsequently, based on the measured data, the controller reevaluates the situation, issues new flow tables, and directs data transmission through more optimal links, thereby mitigating the performance degradation associated with hardware issues.



(a) Comparison of storage latency in a 10 MB file (b) Comparison of storage latency in a 100 MB file (c) Comparison of storage latency in a 1000 MB file

Figure 11. Comparison of storage latency in a 10, 100, and 1000 MB file.

The second part of the experiment is a test in the case of poor network status and increased load on certain nodes. here a program will be written to implement a looped video playback in storage node No.2 (Raspberry Pi 4B) to increase the CPU and memory usage of this storage node. During the test, separate file storage is performed in storage node No.2 through other clients to increase the disk IO load of this node and reduce the state of the network. Figure 12 demonstrates the experimental result graphs for this condition.



(a) Comparison of storage latency in a 10 MB file (b) Comparison of storage latency in a 100 MB file (c) Comparison of storage latency in a 1000 MB file

Figure 12. Comparison of storage latency in a 10, 100, and 1000 MB file.

As can be seen from the figure, when the network state is poor and the load of storage node No.2 is large, the storage time of the three different file sizes increases to a certain extent, but the system designed in this paper still demonstrates better performance and a shorter storage time than the traditional edge-distributed storage system, and the advantage is more obvious than that of the case where the network and the storage node are well loaded. This is because the edge-distributed wireless storage system designed in this paper does not only take the remaining capacity of the storage node as an index when selecting the storage node, but also considers the state of the network and the load state of the storage node comprehensively. When the performance of the No.2 storage node and the state of the network decreases, the controller obtains the network state and the load state of the No.2 storage node before doing the file chunking and the selection of the storage node, and comprehensively judges that this storage node state is poor, at which point the decision of splitting files will allocate more file size to storage nodes 1 and 3, reducing the file storage size of storage node No.2 and increasing the overall system performance. The performance of the distributed edge wireless storage system designed in this paper is verified by the above two parts of the experiments.

6. Conclusions

In this paper, a distributed edge wireless storage system architecture based on SMB and software-defined networking is designed and a prototype physical system is fabricated. The system utilizes software-defined networking technology to sense and measure the real-time network link state, and at the same time obtains the real-time load state of the nodes through the storage node load state self-reporting mechanism, combines the two information to establish a multiattribute decision model, and completes the selection of storage nodes by solving the model. Through the write performance test of the actual system, compared with the traditional edge distributed storage system, the system designed in this paper has obvious improvement in write operation performance, and compared with the edge distributed storage system that needs to deploy monitoring nodes individually, it has the advantages of being lightweight and sporting a low deployment cost. It is believed that the work in this paper can help promote the development of distributed edge storage systems.

Currently, the architecture of our edge distributed wireless storage system in the data forwarding plane utilizes a wired approach, representing a limitation in our current work. In subsequent research, we plan to explore the deployment of the data forwarding plane using wireless Mesh, thereby achieving comprehensive wireless connectivity for the system. On another note, the adoption of wireless Mesh connectivity allows for convenient changes in network topology. Strategically placing and connecting these SDN switches can maximize the system’s transmission efficiency, making it more suitable for certain industrial physical network applications.

Finally, while the primary focus of this paper is at the edge, future work could involve exploring collaborative data storage at the cloud-edge interface. These aspects represent the directions that our future research endeavors will take.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work was supported in part by National Natural Science Foundation of China (Nos. 62161006, 62172095), the subsidization of Innovation Project of Guangxi Graduate Education (No. YCBZ2023134), Key Laboratory of Cognitive Radio and Information Processing, Ministry of Education (Guilin University of Electronic Technology) (No. CRKL220103), and Guangxi Key Laboratory of Wireless Wideband Communication and Signal Processing (Nos. GXKL06220110, GXKL06230102).

Conflict of interest

The authors declare there is no conflict of interest.

References

1. K. Cao, Y. Liu, G. Meng, Q. Sun, An overview on edge computing research, *IEEE Access*, **8** (2020), 85714–85728. https://doi.org/10.1109/ACCESS.2020.2991734
2. J. Xia, G. Cheng, S. Gu, D. Guo, Secure and trust-oriented edge storage for Internet of Things, *IEEE Internet Things J.*, **7** (2019), 4049–4060. https://doi.org/10.1109/JIOT.2019.2962070
3. M. U. A. Siddiqui, F. Qamar, M. Tayyab, M. N. Hindia, Q. N. Nguyen, R. Hassan, Mobility management issues and solutions in 5G-and-beyond networks: a comprehensive review, *Electronics*, **11** (2022), 1366. https://doi.org/10.3390/electronics11091366
4. P. Yang, N. Xiong, J. Ren, Data security and privacy protection for cloud storage: A survey, *IEEE Access*, **8** (2020), 131723–131740. https://doi.org/10.1109/ACCESS.2020.3009876
5. J. Wu, Y. Li, F. Ren, B. Yang, Robust and auditable distributed data storage with scalability in edge computing, *Ad Hoc Networks*, **117** (2021), 102494. https://doi.org/10.1016/j.adhoc.2021.102494
6. M. Legault, A practitioner's view on distributed storage systems: Overview, challenges and potential solutions, *Technol. Innovation Manage. Rev.*, **11** (2021), 32–41. https://doi.org/10.22215/timreview/1448
7. W. Liu, Research on cloud computing security problem and strategy, in *2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, **8** (2012), 1216–1219. https://doi.org/10.1109/CECNet.2012.6202020
8. G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, K. Huang, Toward an intelligent edge: Wireless communication meets machine learning, *IEEE Commun. Mag.*, **58** (2020), 19–25. https://doi.org/10.1109/MCOM.001.1900103
9. J. Thompson, X. Ge, H. C. Wu, R. Irmer, H. Jiang, G. Fettweis, et al., 5G wireless communication systems: Prospects and challenges, *IEEE Commun. Mag.*, **52** (2014), 62–64. https://doi.org/10.1109/MCOM.2014.6736744
10. W. Li, Q. Li, L. Chen, F. Wu, J. Ren, A storage resource collaboration model among edge nodes in edge federation service, *IEEE Trans. Veh. Technol.*, **71** (2022), 9212–9224. https://doi.org/10.1109/TVT.2022.3179363
11. C. Roy, S. Misra, J. Maiti, M. S. Obaidat, DENSE: Dynamic edge node selection for safety-as-a-service, in *2019 IEEE Global Communications Conference (GLOBECOM)*, **23** (2019), 1–6. https://doi.org/10.1109/globecom38437.2019.9014180
12. M. Abd-El-Malek, W. V. Courtright II, C. Cranor, G. R. Ganger, J. Hendricks, A. J. Klosterman, et al., Ursa minor: Versatile cluster-based storage, *FAST*, **5** (2005), 5.
13. D. Puthal, R. Ranjan, A. Nanda, P. Nanda, P. P. Jayaraman, A. Y. Zomaya, Secure authentication and load balancing of distributed edge datacenters, *J. Parallel Distrib. Comput.*, **124** (2019), 60–69. https://doi.org/10.1016/j.jpdc.2018.10.007
14. Y. Zhang, S. Debroy, P. Calyam, Network measurement recommendations for performance bottleneck correlation analysis, in *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, **12** (2016), 1–7.
15. R. G. Clegg, M. S. Withall, A. W. Moore, I. W. Phillips, D. J. Parish, M. Rio, et al., Challenges in the capture and dissemination of measurements from high-speed networks, *IET Commun.*, **3** (2009), 957–966. https://doi.org/10.1049/iet-com.2008.0068
16. S. E. Engineer, A. Engineer, *Structure and Interpretation of the SMB Protocol*, Springer, 2018.

17. K. Kirkpatrick, Software-defined networking, *Commun. ACM*, **56** (2013), 16–19. https://doi.org/10.1109/sta.2019.8717234
18. T. D. Nadeau, K. Gray, *SDN: Software Defined Networks: An Authoritative Review of Network Programmability Technologies*, O'Reilly Media, Inc., 2013.
19. M. T. Rashid, D. Zhang, D. Wang, Edgestore: Towards an edge-based distributed storage system for emergency response, in *2019 IEEE 21st International Conference on High Performance Computing and Communications*, **31** (2019), 2543–2550. https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00356
20. A. Makris, E. Psomakelis, T. Theodoropoulos, K. Tserpes, Towards a distributed storage framework for edge computing infrastructures, in *Proceedings of the 2nd Workshop on Flexible Resource and Application Management on the Edge*, **54** (2022), 9–14. https://doi.org/10.1145/3526059.3533617
21. K. Sonbol, Ö. Özkasap, I. Al-Oqily, M. Aloqaily, EdgeKV: Decentralized, scalable, and consistent storage for the edge, *J. Parallel Distrib. Comput.*, **144** (2020), 28–40. https://doi.org/10.1016/j.jpdc.2020.05.009
22. J. Xing, H. Dai, Z. Yu, A distributed multi-level model with dynamic replacement for the storage of smart edge computing, *J. Syst. Archit.*, **83** (2018), 1–11. https://doi.org/10.1016/j.sysarc.2017.11.002
23. F. Qiao, J. Wu, J. Li, A. K. Bashir, S. Mumtaz, U. Tariq, Trustworthy edge storage orchestration in intelligent transportation systems using reinforcement learning, *IEEE Trans. Intell. Transp. Syst.*, **22** (2020), 4443–4456. https://doi.org/10.1109/TITS.2020.3003211
24. W. Li, J. Ji, L. Huang, L. Zhang, Global dynamics and control of malicious signal transmission in wireless sensor networks, *Nonlinear Anal. Hybrid Syst.*, **48** (2023), 101324. https://doi.org/10.1016/j.nahs.2022.101324
25. W. Li, J. Ji, L. Huang, Z. Cai, Periodic orbit analysis for a delayed model of malicious signal transmission in wireless sensor networks with discontinuous control, *Math. Methods Appl. Sci.*, **46** (2023), 5267–5285. https://doi.org/10.1002/mma.8831
26. K. Kontodimas, P. Soumplis, A. Kretsis, P. Kokkinos, E. Varvarigos, Secure distributed storage on cloud-edge infrastructures, in *2021 IEEE 10th International Conference on Cloud Networking*, **93** (2021), 127–132. https://doi.org/10.1109/CloudNet53349.2021.9657156
27. C. Wu, Y. Chen, Z. Qi, H. Guan, DSPR: Secure decentralized storage with proof-of-replication for edge devices, *J. Syst. Archit.*, **125** (2022), 102441. https://doi.org/10.1016/j.sysarc.2022.102441
28. S. Li, T. Lan, HotDedup: Managing hot data storage at network edge through optimal distributed deduplication, in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, **144** (2020), 247–256. https://doi.org/10.1109/infocom41043.2020.9155233
29. K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, A. Akella, Presto: Edge-based load balancing for fast datacenter networks, *ACM SIGCOMM Comput. Commun. Rev.*, **45** (2015), 465–478. https://doi.org/10.1145/2785956.2787507
30. C. Hunt, *TCP/IP Network Administration*, O'Reilly Media, Inc., 2002.
31. R. Sharpe, Just what is SMB?, *Oct*, **8** (2002), 9.
32. S. Khan, A. Gani, A. W. A. Wahab, M. Guizani, M. K. Khan, Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art, *IEEE Commun. Surv. Tutorials*, **19** (2016), 303–324. https://doi.org/10.1109/COMST.2016.2597193

33. A. Bianco, R. Birke, L. Giraudo, M. Palacin, Openflow switching: Data plane performance, in *2010 IEEE International Conference on Communications*, **18** (2010), 1–5.
34. A. Jalili, H. Nazari, S. Namvarasl, M. Keshtgari, A comprehensive analysis on control plane deployment in SDN: In-band versus out-of-band solutions, in *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, **89** (2017), 1025–1031. https://doi.org/10.1109/KBEI.2017.8324949
35. F. Hu, Q. Hao, K. Bao, A survey on software-defined network and openflow: From concept to implementation, *IEEE Commun. Surv. Tutorials*, **16** (2014), 2181–2206. https://doi.org/10.1109/COMST.2014.2326417
36. M. Noto, H. Sato, A method for the shortest path search by extended Dijkstra algorithm, in *SMC 2000 Conference Proceedings. 2000 IEEE International Conference on Systems, Man and Cybernetics*, **3** (2000), 2316–2320. https://doi.org/10.1109/ICSMC.2000.886462
37. S. Syamsudin, R. Rahim, Study approach technique for order of preference by similarity to ideal solution (TOPSIS), *Int. J. Recent Trends Eng. Res.*, **3** (2017), 268–285.
38. R. Rohith, M. Moharir, G. Shobha, SCAPY-A powerful interactive packet manipulation program, in *2018 International Conference on Networking, Embedded and Wireless Systems (ICNEWS)*, **52** (2018), 1–5. https://doi.org/10.1109/ICNEWS.2018.8903954
39. C. E. Palazzi, M. Brunati, M. Roccetti, An OpenWRT solution for future wireless homes, in *2010 IEEE International Conference on Multimedia and Expo*, **74** (2010), 1701–1706. https://doi.org/10.1109/ICME.2010.5583223
40. B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, et al., The design and implementation of open vSwitch, in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, **261** (2015), 117–130.
41. W. Donat, W. Donat, Introducing the Raspberry Pi, in *Learn Raspberry Pi Programming with Python: Learn to Program on the World's Most Popular Tiny Computer*, **16** (2018), 1–26.
42. S. Vidya, R. Bhaskaran, ARP storm detection and prevention measures, *Int. J. Comput. Sci. Issues*, **8** (2011), 456.
43. V. Rajaravivarma, Virtual local area network technology and applications, in *Proceedings the Twenty-Ninth Southeastern Symposium on System Theory*, **18** (1997), 49–52. https://doi.org/10.1109/SSST.1997.581577
44. W. Li, J. Ji, L. Huang, Global dynamics analysis of a water hyacinth fish ecological system under impulsive control, *J. Franklin Inst.*, **359** (2022), 10628–10652. https://doi.org/10.1016/j.jfranklin.2022.09.030
45. A. Tridgell, Samba protocol, Available from: https://www.samba.org/.