




Article

Status-Byte-Assisted RDMA Transmission Mechanism for Optimizing Multi-Task Video Streaming in Edge Computing

Donglei Xiao ¹, Huiyue Yi ^{2,*} , Wuxiong Zhang ²  and Wenhui Shen ^{1,*} 

¹ School of Communication and Information Engineering, Shanghai University, Shanghai 200444, China; dlxiao@shu.edu.cn

² Science and Technology on Microsystem Laboratory, Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, Shanghai 201899, China; wuxiong.zhang@mail.sim.ac.cn

* Correspondence: huiyue.yi@mail.sim.ac.cn (H.Y.); haomeni@163.com (W.S.)

Abstract: In the context of the rapid development of edge computing, optimizing data transmission and reducing latency is crucial for efficient collaborative processing among edge servers. Traditional TCP/IP protocols are hindered by high latency and low throughput, while RDMA (Remote Direct Memory Access) technology addresses these challenges by enabling direct memory access and bypassing the operating system kernel. However, the RDMA data transmission mechanism based on sliding windows requires frequent memory status exchanges in the order of memory blocks, which can limit its ability to handle multiple concurrent tasks within a single Queue Pair (QP). To address the limitations of the traditional sliding window transmission mechanism in multi-task environments, we propose a novel RDMA data transmission mechanism that utilizes status bytes to indicate memory block utilization, which utilizes stateless server connections, and multi-task shared QP transmission strategies. In the proposed mechanism, fine-grained control over memory blocks is achieved through the status byte, thereby enabling effective multi-task real-time video stream transmission. Experimental results show that, compared to the sliding window method, the proposed status-byte-assisted RDMA transmission mechanism provides higher throughput, lower latency, and reduced resource consumption, thus enhancing system scalability and reducing CPU utilization. Moreover, this mechanism achieves more stable throughput than the sliding window method when transmitting multiple real-time video streams in edge computing scenarios, making it particularly suitable for data transmission in such environments.

Keywords: RDMA; sliding window; shared QP; multitasking; status byte; data transmission



Citation: Xiao, D.; Yi, H.; Zhang, W.; Shen, W. Status-Byte-Assisted RDMA Transmission Mechanism for Optimizing Multi-Task Video Streaming in Edge Computing. *Appl. Sci.* **2024**, *14*, 7437. <https://doi.org/10.3390/app14177437>

Academic Editor: Christos Bouras

Received: 26 July 2024

Revised: 16 August 2024

Accepted: 21 August 2024

Published: 23 August 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

With the rapid growth in the number of Internet of Things (IoT) devices and the surge in data volume, edge computing has attracted increasing attention as a key paradigm for decentralizing computational tasks near the source of data generation, thereby reducing latency and improving bandwidth utilization [1,2]. Edge computing achieves this by offloading some computational tasks from central cloud servers to edge nodes closer to data sources. However, edge servers often face challenges related to limited computational and storage resources and the heterogeneity of computational resources (e.g., CPUs, GPUs, and FPGAs) complicates the ability of individual edge servers to independently handle complex business processes [3]. Consequently, collaborative processing among multiple edge servers becomes essential, which in turn necessitates large-scale data transmission among multiple servers.

The traditional TCP/IP protocol stack faces significant bottlenecks in this environment, including high transmission latency, low transmission rates, and high CPU resource consumption, which limit the overall performance and scalability of the system [4]. Remote Direct Memory Access (RDMA) technology enables direct memory access between network

nodes by bypassing the operating system kernel. This approach substantially reduces transmission latency and CPU resource consumption, making RDMA a powerful tool for addressing data transmission bottlenecks in edge computing.

Despite the notable advantages of RDMA in enhancing data transmission performance, the existing RDMA data transmission mechanisms still face several challenges in practical applications. Most current work primarily exploits RDMA's unilateral operation feature to accelerate data transmission without further optimization, and RDMA NICs face scalability issues [5–10]. Furthermore, the existing RDMA technologies show limitations in multitasking transmissions and resource management, particularly when supporting multiple tasks within a single QP. For example, the sliding window transmission mechanism proposed by Tu et al. [11] requires frequent exchanges of memory status during data transmissions to prevent memory overwriting or reading invalid data. Additionally, since it uses only two pointers to indicate the status of all memory blocks, it limits flexible manipulation of memory blocks and lacks robust support for multitasking transmissions within a single QP.

To further optimize resource scheduling and data processing in edge computing environments, container technology, along with Kubernetes (K8s) [12,13], has been introduced. Kubernetes is used to manage containers, automating their deployment, scaling, and operation. As the smallest unit of processing, containers can flexibly and efficiently manage and allocate computing resources while ensuring resource isolation, allowing edge servers to collaborate more effectively in handling complex business processes.

The contributions of this paper can be summarized as follows. Firstly, we propose the use of status bytes to indicate the status of all memory blocks and introduce a novel status-byte-assisted RDMA transmission mechanism that addresses the limitations of the traditional sliding window mechanism. The proposed transmission mechanism optimizes data transmission and provides synchronization methods for the transmission of multiple data streams, and thus provides higher throughput, lower latency, and reduced resource consumption. Moreover, this improvement enhances system scalability and reduces CPU utilization. Secondly, we design and implement an efficient multi-task shared QP transmission strategy, which achieves fine-grained control over memory blocks through status byte transmission. This mechanism provides more stable throughput compared to the sliding window method in multi-task real-time video streaming scenarios. Thirdly, in conjunction with our data transmission mechanism, we use containers as the smallest processing units to handle data and enable collaborative resource scheduling and allocation among edge servers, thereby enhancing the multitask processing and system scalability of edge computing.

The rest of the paper is organized as follows. Section 2 provides the background and motivation for our work. In Section 3, we detail the design and implementation of the status-byte-assisted RDMA transmission mechanism. Section 4 presents our experimental evaluation results to illustrate the effectiveness of the proposed status-byte-assisted RDMA transmission mechanism. Finally, Section 5 concludes the paper with a summary.

2. Related Works and Motivations

2.1. Edge Computing and RDMA Technology

Edge computing aims to deploy computing and storage resources closer to the data source to meet the demands for low latency and high bandwidth [14]. However, the computing and storage resources of edge servers are often limited, and the heterogeneous computing resources severely complicate the ability of a single server to complete complex business processes. When performing collaborative processing among edge servers, large volumes of data need to be transmitted, making it crucial to coordinate the allocation and scheduling of edge server resources to meet the requirements of multitask processing. The works [15–17] demonstrate that the traditional TCP/IP network protocol stack requires multiple copies of data between the kernel and user space during data transmission, resulting in numerous limitations such as high transmission latency, low transmission speed, low throughput, and significant CPU core resource consumption.

The RDMA technology is a high-performance data transmission technology that allows network nodes to directly access the memory of remote nodes. By bypassing the operating system kernel, it significantly reduces transmission latency and CPU resource consumption [18,19]. In the field of RDMA, many studies utilize RDMA's one-sided operation characteristics to improve system performance [20–22], while others focus on enhancing RDMA's scalability [23]. For example, Wang et al. [7] improve connection scalability by minimizing on-chip data structures and their memory requirements through protocol and architecture co-design. Huawei's StaR [24] adopts stateless server connection technology to reduce RDMA resource consumption on the server side, thereby alleviating connection scalability issues. Mellanox attempts to limit the number of active network connections through dynamic connection transport service technology [25], aiming to avoid frequently triggering cache miss events in the network interface. Chen et al. s' Scalable [26] effectively mitigates resource contention through connection grouping and virtualization mapping. However, in resource-limited situations, there is a need for improved memory management and data synchronization methods [27].

Moreover, to further optimize resource scheduling and data processing in edge computing environments, container technology has been introduced as the smallest processing unit. K8s is employed to manage these containers, automating deployment, scaling, and operations. Container technology can flexibly and efficiently manage and allocate computing resources while ensuring isolation, enabling edge servers to better collaborate in handling complex business processes.

2.2. Asymmetry in RDMA Unilateral Communication

The core of the RDMA protocol lies in its one-sided read-and-write capabilities, which provide the foundation for efficient data transmission. RDMA's one-sided operations can be categorized into active and passive operations [28]. In active operations, the server initiates RDMA operations to write data to or read data from the client. In passive operations, the client initiates requests for data reads or writes, and then the server merely responds to these requests passively. However, the resources consumed and latency incurred by actively initiating RDMA operations differ from those involved in passively responding to RDMA operations.

Due to the unique nature of RDMA communication, some data processing functions are directly integrated into the RDMA NIC (RNIC) hardware, allowing RDMA operations to bypass the operating system kernel and occur directly at the hardware level [29]. During one-sided RDMA transmissions, passive RDMA operations (receive operations) are entirely handled by the RNIC hardware. In contrast, active RDMA operations (initiate requests) involve interaction between hardware and software. This process includes sending a work request, waiting for an acknowledgment (ACK) from the remote RNIC, and generating a completion queue entry (CQE) upon request completion, which requires data tracking throughout to ensure successful operation.

This asymmetry, referred to as the asymmetry of RDMA one-sided communication, implies that during data transmission, the initiating side (active party) bears most of the processing burden, while the responding side (passive party) only needs to provide simple responses. Consequently, under the same conditions, the cost of processing passive RDMA operations is significantly lower than that of initiating active RDMA operations.

In edge computing, although servers work collaboratively, once the RDMA connection is established, the receiving server often does not need to send large amounts of data to the sending server. Therefore, in an RDMA-based data transmission system, the sending server uses active RDMA operations exclusively, while the receiving server exclusively uses passive RDMA operations. This approach not only conserves significant CPU and RNIC resources for the receiving server but also improves throughput and supports a higher number of RDMA connections.

2.3. Motivations for the Status-Byte-Assisted RDMA Transmission Mechanism

Currently, there are two primary transmission mechanisms: traditional RDMA data transmission and sliding window-based data transmission [11]. Traditional RDMA data transmission requires at least three network interactions: memory address notification, remote data reading/writing, and completion notification. This method incurs significant overhead in communication control and is more suitable for scenarios where transmissions are infrequent and latency is less critical.

As illustrated in Figure 1a, the sliding window technique proposed by Han et al. requires only two RDMA interactions per transmission. This technique consists of two phases: initialization and data transmission. In the initialization phase, the sender invokes the memory pool initialization interface (RmInit). RmInit is an interface that sets up the sliding window using the received memory pool information (address, length, and KEY). This function of RmInit is to initialize the memory pool for use in the RDMA communication and create a circular queue with n members, as shown in Figure 1b. The sender then uses the remote memory write interface (RmWrite) to perform data transmission. RmWrite is an interface that writes data to a specified remote memory location. The RDMA Write With Immediate (Write_with_imm) operation is utilized, which includes immediate data to notify the receiver about the data's starting position and length. After the data has been processed, the receiver sends an ACK notification to confirm receipt. Finally, the sender calls the memory release interface (RmFree). RmFree is an interface that updates the sliding window status by releasing the memory block and making it available for reuse.

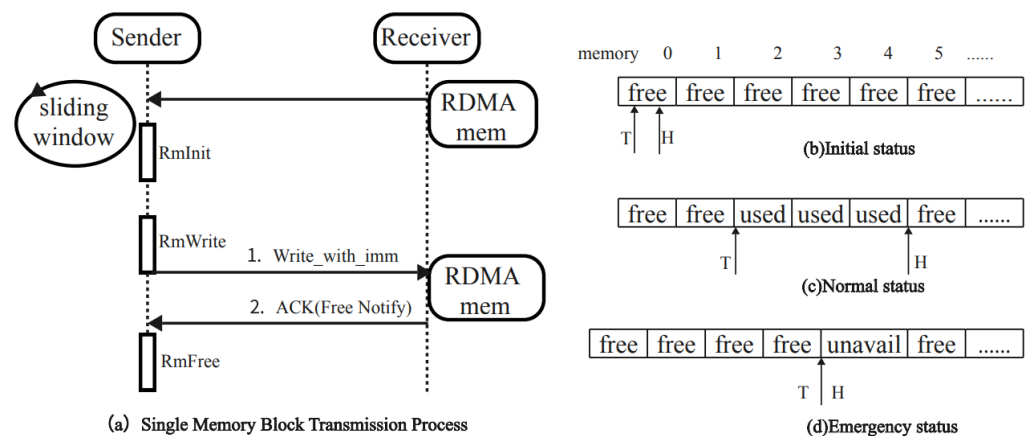


Figure 1. (a) Data transmission process based on sliding window mechanism; (b–d) Memory analysis of sliding window transmission mechanism facing special status.

However, this sliding window transmission mechanism has several limitations. As illustrated in Figure 1c, the sliding window tracks the usage status of all memory blocks with two pointers, which necessitates that memory block sending and releasing memory blocks follow a strict sequence, resulting in a lack of flexibility. During task transmission, as shown in Figure 1d, if an ACK confirmation for a memory block is not received promptly or the memory block at the receiver needs to remain in waiting status, the head pointer cannot advance and the window position occupied by this data block cannot be released for new data. When the tail pointer cycles back to this position, the system might misinterpret the situation as no available memory blocks, despite actual availability, leading to transmission delays or even crashes due to the logical constraints of the sliding window. Additionally, this transmission mechanism requires the receiver to actively return ACK notifications and the sender to periodically poll for ACK notifications. This method not only consumes the receiver's CPU resources but also results in poor real-time performance.

When processing the transmission of multiple data streams of various devices, the receiver may require multiple processing threads, which may result in varying sequences of ACK returns. Given that the release of sliding window memory blocks must follow

a strict order, this can easily lead to confusion in the release of memory blocks, making it difficult to handle the inputs from multiple devices effectively. If concurrency control mechanisms are employed to synchronize access to resources, it will severely impact RDMA performance, rendering it no better than single-threaded performance [30]. Thus, when transmitting data streams from multiple devices, the sliding window transmission mechanism typically requires establishing multiple RDMA connections, which consumes significant RDMA NIC resources [31,32], potentially leading to performance bottlenecks. To offer a comprehensive understanding of the key characteristics, advantages, and limitations of existing RDMA transmission mechanisms, Table 1 presents a comparative analysis of traditional RDMA transmission methods and the sliding window-based approach. This comparison highlights the inherent trade-offs and challenges associated with each mechanism, providing a foundation for discussing potential improvements.

Table 1. Comparison of various RDMA transmission mechanisms.

Method/Technique Name	Key Features	Advantages	Limitations
Traditional RDMA Transmission	Dynamic memory block allocation, requires at least three network interactions per transmission	Simple implementation, high flexibility	High network latency and communication overhead
Sliding Window-based RDMA Transmission	Controls data transmission using a sliding window mechanism, requiring only two RDMA interactions	Optimized for fixed window size, reduces network interactions	Memory blocks must be used in order, consuming excessive CPU and RNICA resources.

Building on this analysis, we propose to utilize status bytes to indicate the status of each memory block and introduce a status-byte-assisted RDMA transmission mechanism. By introducing the status byte, we achieve fine-grained control over memory blocks, reduce the need for frequent memory status exchanges, and enhance multi-task transmission efficiency and system scalability. Moreover, the proposed status-byte-assisted transmission mechanism better manages memory block status, thereby overcoming the limitations of the sliding window transmission mechanism and providing higher stability and scalability in multi-device environments.

3. Proposed Status-Byte-Assisted RDMA Transmission Mechanism

In this section, the overall system architecture of the proposed status-byte-assisted RDMA transmission system is first described. Then, the status-byte-assisted transmission mechanism is illustrated in detail, covering the processes involved in data filling, sending, receiving, synchronization, and processing. Next, this section details the specificity of the status-byte-assisted RDMA transmission mechanism, which offers fine-grained control over memory blocks. Finally, it explains the multi-device shared QP strategy to efficiently handle data streams from multiple devices.

3.1. Overall System Architecture

Figure 2 provides an overview of the architecture of our proposed system, which includes a cloud-edge collaborative global management platform and multiple edge servers. The cloud-edge collaborative global management platform is a cloud-based system that manages and allocates resources across all edge servers, ensuring efficient use of resources and coordination among them. Each edge server is connected to several sensor devices such as radar and cameras. When the data processing requirements of an edge server exceed its processing capacity, the server sends a resource scheduling request to the cloud-edge collaborative global management platform. The platform then responds with a resource scheduling result based on the resource status of each edge server and the requested amount of resources.

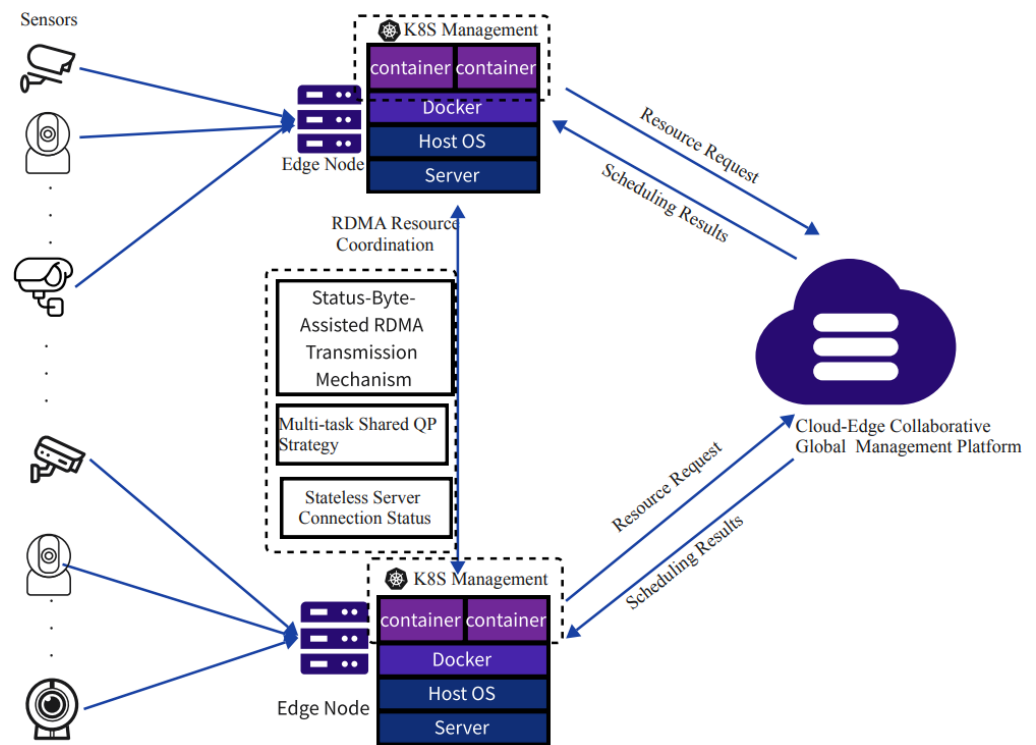


Figure 2. Overall architecture of the proposed status-byte-assisted RDMA transmission system.

Based on the resource scheduling results, the edge server that sends the request is designated as the data-sending end, while the edge server allocated for the collaborative processing function is the data-receiving end. The data transmission between the sending and receiving ends utilizes the proposed status-byte-assisted RDMA data transmission mechanism to accelerate data transmission and synchronization. Moreover, Docker and container technology, managed by K8s, are implemented at the data-receiving ends to achieve flexible and efficient data processing workflows. By enabling collaborative data processing among edge servers, the system can fully utilize the unique characteristics and resources of each edge server.

3.2. Design of the Status-Byte-Assisted RDMA Transmission Mechanism

3.2.1. Overview of the Status-Byte-Assisted RDMA Transmission Mechanism

In our proposed data transmission mechanism, the system is divided into two main parts: the data-sending end and the data-receiving end. The sending end is responsible for filling, sending, and synchronizing data, while the receiving end is responsible for receiving, storing, and processing data. Moreover, we propose a novel memory management approach that utilizes a ‘status byte’ to track and indicate the current state of memory blocks, which helps in managing data flow more effectively.

Initially, the sending end sets up status bytes to indicate the status of all memory blocks. When sending data, these status bytes are transmitted along with the data. The receiving end monitors for new data arrivals by checking the status bytes. The sending end also retrieves the status bytes from the receiving end to track their states. This system leverages status bytes to effectively manage synchronization between the sending and receiving ends.

Figure 3 illustrates the data transmission process based on the status-byte-assisted RDMA transmission mechanism. As shown in Figure 3, the data-sending end first calls the memory pool initialization interface (RmInit) to create the status byte. After that, it uses RDMA Write requests (via the RmWrite interface) to continuously send data. When no memory blocks are available, the sending end invokes the memory release interface

(RmFree) to issue the RDMA Read requests. These RDMA Read requests read the status bytes of all memory blocks on the receiving end at once and store them in the second status byte memory block. This method reduces communication overhead on the receiving end and improves real-time performance since the sending end manages the status updates.

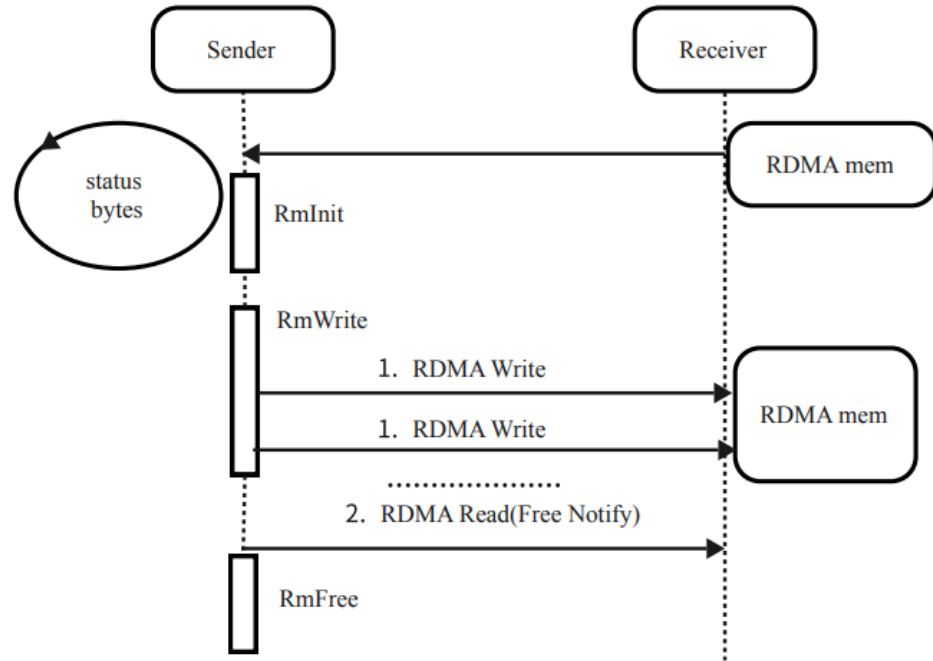


Figure 3. Data transmission process based on the status-byte-assisted RDMA transmission mechanism.

Figure 4 illustrates the relationship between the status byte and the status byte memory blocks on the sender and receiver ends. As shown in Figure 4, the data-sending end includes N sending memory blocks, the first status byte memory block for storing the usage status of the local memory blocks, and the second status byte memory block for storing the usage status of the receiving end’s memory blocks. The data receiving end includes N receiving memory blocks and the third status byte memory block for storing the usage status of the local memory blocks. Here, the status bytes are used to describe the usage status of memory blocks. The usage status of the N sending memory blocks is represented by the status bytes stored in the first status byte memory block, while the usage status of the N to receive memory blocks is represented by the status bytes stored in the third status byte memory block. The structures of the first, second, and third status byte memory blocks are shown in Figure 4.

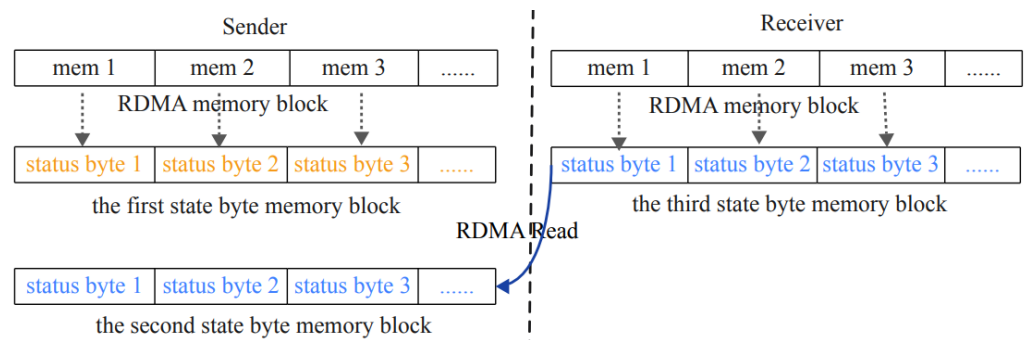


Figure 4. Relationship between status byte and status byte memory blocks on the sender and receiver end.

The status bytes for the sending memory blocks can have two statuses: 0 indicates that the sending memory block does not contain data, while 1 indicates that it contains data. The status bytes for the receiving memory blocks can have three statuses: 0 indicates that a receiving memory block does not contain data, 1 indicates that it has data, and 2 indicates that it is currently unavailable.

The structures of the sending memory block and receiving memory block used in the proposed transmission mechanism are shown in Table 2. Each memory block, whether sending or receiving, contains two sections: the data status section and the valid content section. The data status section includes one byte for the device number and two bytes for the data packet number, which facilitates sharing the QP and prevents data confusion.

Table 2. Structure of sending and receiving memory blocks.

device number (one byte)	packet number (two bytes)	valid data (fixed size)
--------------------------	---------------------------	-------------------------

3.2.2. Transmission Mode Selection

The sliding window transmission mechanism uses RDMA Write With Immediate to transmit data, utilizing immediate data to notify the receiver of the data's arrival. Although this method is a one-sided RDMA transmission, it still requires the receiver to issue a receive request, which generates a Completion Queue Entry (CQE). A CQE is a data structure used in RDMA to indicate the completion of an operation. The location of the transmitted data is determined by the immediate data in the CQE, which requires interaction with the receiver. To address this issue, we propose to utilize two RDMA Writes for data transmission: the first RDMA Write transmits the valid data and the second RDMA Write transmits the status byte of the corresponding memory block to the third status byte memory block on the receiving end. These two requests are sent using a linked list to minimize context switching. The first request does not generate a CQE, while the second request uses inline data. This approach effectively reduces the exchange between the RNIC and memory. Evidently, this approach achieves true one-sided transmission without interacting with the receiver, thereby enhancing transmission efficiency and reducing resource consumption.

3.2.3. Data-Filling Module

The data-filling module on the sender end is responsible for filling the data into the sending memory blocks. First, it checks the status byte stored in the first status byte memory block to determine whether there are available sending memory blocks. If available, the data-filling process is executed.

Figure 5 shows the flowchart of the data-filling process. Specifically, the steps of the data-filling process are as follows:

1. Check the status of the sending memory blocks: The filling thread first checks the first status byte memory block to determine the status of the i -th sending memory block. If the i -th memory block already contains data (status is 1), it needs to wait. If the i -th memory block has no data (status is 0), the filling thread proceeds with the data-filling operation;
2. Perform data-filling operation: During this operation, the filling thread writes the device number and packet number into predefined positions within the block;
3. Update the status byte: Finally, after completing the data-filling operation, the filling thread updates the status byte of the sending memory block in the first status byte memory block to indicate that it contains data.

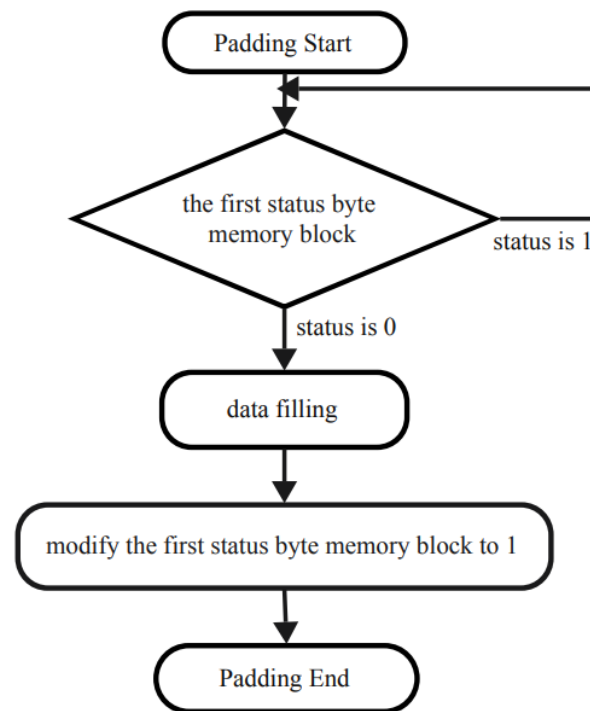


Figure 5. Flowchart of the data-filling process.

3.2.4. Data-Sending Module

The data-sending module first checks the status byte stored in the second status byte memory block to determine whether there are available receiving memory blocks. Next, it checks the first status byte memory block to determine whether there are available sending memory blocks, and then executes the data-sending process.

Figure 6 shows the flowchart of the data-sending process. Specifically, the steps of the data-sending process are as follows:

1. Check the status of the receiving memory blocks: The sending thread examines the second status byte memory block to determine the status of the receiving memory blocks. If the j -th receiving memory block contains data (status is 1), it means there are no available memory blocks, and synchronization operations must be performed until the j -th receiving memory block is empty (status is 0);
2. Check the status of the sending memory blocks: The sending thread verifies the first status byte memory block to determine the status of the j -th sending memory block. If the j -th sending memory block is empty (status is 0), it waits for the filling thread to fill the data. If the memory block contains data (status is 1), the sending thread initiates RDMA Write requests to transmit the data stored in the sending memory block to the j -th receiving memory block on the receiver end;
3. Update the status byte: The sending thread uses inline operations to update the status byte in the third status byte memory block, reflecting that the j -th receiving memory block now contains data. After the data sending is complete, the sending thread updates the corresponding status byte of the sending memory block in the first status byte memory block to indicate that it is empty. Additionally, the sending thread updates the status byte in the second status byte memory block to reflect that the j -th receiving memory block now contains data and waits for the receiver to process the data.

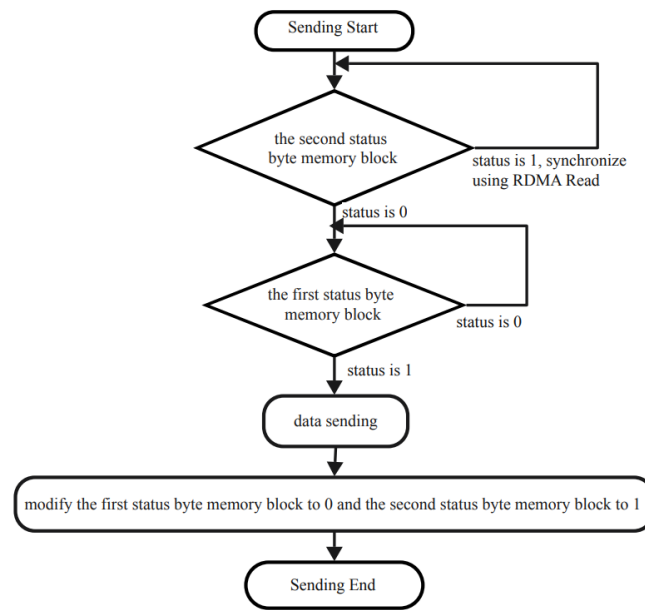


Figure 6. Flowchart of the data-sending process.

3.2.5. Data-Receiving Module

The data-receiving module determines whether there are available receiving memory blocks by checking the status byte stored in the third status byte memory block. If there are available receiving memory blocks, the data-receiving process is executed.

Figure 7 shows the flowchart of the data-receiving process. Specifically, the steps of the data-receiving process are as follows:

1. Check the status of the receiving memory blocks: The receiving thread checks the third status byte memory block to determine the status of the j -th receiving memory block. If the j -th receiving memory block is empty (status is 0), it means no data have arrived, and the thread sleeps for a fixed time before querying again;
2. Perform the data processing operation: If the j -th receiving memory block contains data (status is 1), the receiving thread performs the data processing operation. Before executing the data processing operation, the receiving thread extracts the contents of the first predetermined position in the j -th receiving memory block to obtain the device number and then extracts the contents of the second predetermined position to obtain the packet number;
3. Update the status byte: After completing the data processing operation, the receiving thread updates the status byte in the third status byte memory block to indicate that the j -th receiving memory block is empty.

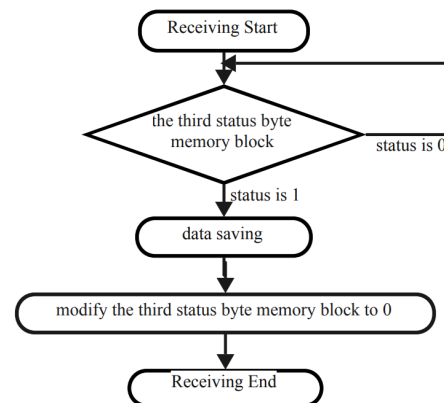


Figure 7. Flowchart of the data-receiving process.

3.2.6. Data Synchronization Module

The synchronization module performs operations when no available receiving memory blocks on the sender end. The specific process is as follows: the sender uses an RDMA Read request to obtain the status bytes of the receiving memory block from the receiver end and saves the retrieved status bytes in the second status byte memory block.

3.2.7. Data-Processing Module

During the data-receiving phase, the data from the same sensing device are stored in the same directory based on the device number and arranged sequentially by packet number. By using container volume mounting technology for data sharing, the data from different devices are mounted into separate containers for processing. By allocating different resources to different containers, the containers are treated as the smallest computing units for data processing.

To enhance the efficiency and scalability of container management, K8s is utilized to automate the deployment, scaling, and operations of containers, ensuring optimal resource utilization and fault tolerance. By managing the lifecycle of containers, K8s simplifies the orchestration of containerized data processing applications and thus allows the system to handle dynamic workloads more effectively. Thus, this approach facilitates collaborative data processing by the edge server and enhances the scalability of the system.

3.2.8. Stateless Connection at the Receiving End

Both the synchronization module and the sending module are initiated by the sender through RDMA operations, and the receiver passively receives the data. Since the receiver does not perform sending operations, it does not need to retain connection information or maintain the sender’s connection status, thereby freeing up RNIC resources on the receiver end.

3.3. Specificity of the Status-Byte-Assisted Transmission

In the proposed status-byte-assisted transmission mechanism, an innovative memory management solution is proposed by implementing the concept of status byte, as illustrated in Figure 8. The status byte indicates the usage status of a memory block: 0 signifies the memory block is available, 1 indicates the memory block contains data, and 2 means the memory block is currently unavailable. Compared with the sliding window transmission mechanism, the status-byte-assisted transmission can handle memory blocks more flexibly.

Memory	free	free	free	free	free	free
status byte	0	0	0	0	0	0

(a)Initial state

Memory	free	free	used	used	used	free
status byte	0	0	1	1	1	0

(b)Normal state

Memory	used	free	used	free	unavailab	free
status byte	1	0	1	0	2	0

(c)Emergency state

Figure 8. Memory status in the status-byte-assisted transmission mechanism.

In burst scenarios, if the current receiving memory block is unavailable, the status byte of the corresponding memory block can be marked as 2. This mechanism allows the sending thread to skip the unavailable memory block and continue sending other memory

blocks, thereby avoiding the blocking of threads. This flexibility makes the transmission mechanism more suitable for various edge-computing data transmission scenarios.

3.4. Multi-Device Shared QP Strategy

In the proposed status-byte-assisted RDMA transmission mechanism, the system scalability is improved by introducing device numbers and packet numbers to manage the memory blocks. Figure 9 provides a comparison of data transmission methods for different transmission mechanisms in the multitasking scenarios. As shown in Figure 9a, the existing sliding window RDMA transmission techniques struggle to support multi-threaded sending and receiving on both ends when transmitting multiple data streams from multiple devices. The sliding window approach establishes multiple RDMA connections, which significantly consumes the on-chip cache resources of the RDMA NIC. When the number of devices is large, low-speed PCIe channels may be required to retrieve connection status from main memory, which will lead to performance bottlenecks. Additionally, due to the uncertainty in the sending times of data blocks, multiple memory blocks may be sent simultaneously or not at all, resulting in significant fluctuations in the system’s instantaneous throughput and low bandwidth utilization.

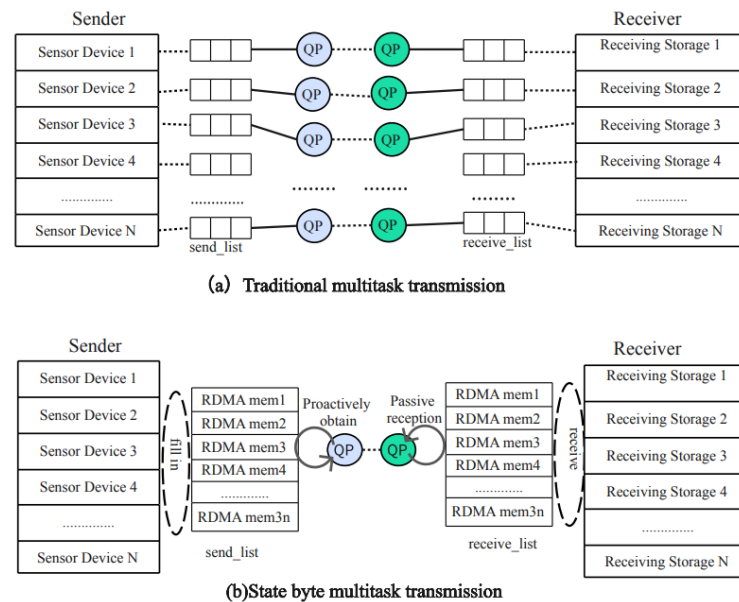


Figure 9. Comparison of data transmission methods for different transmission mechanisms facing multitasking.

To address this issue, as shown in Figure 9b, the proposed status-byte-assisted RDMA transmission technology enables multiple tasks to share a single RNIC QP, in contrast to existing multi-task RDMA transmission methods where each task uses a separate QP connection. Although the multi-task QP sharing can lead to parallel access to QP resources and result in resource contention, the status-byte-assisted RDMA transmission mechanism serializes the process of filling device data into the RNIC QP resource. During data filling, the filling thread adds device numbers and packet numbers to the memory blocks. The receiving end identifies data from different devices using device numbers and arranges the data in sequence based on the packet number to ensure data reliability.

Moreover, by utilizing container mounting technology to share transmitted data among containers for processing, a single-edge server can simultaneously control multiple devices. Additionally, K8s is employed to manage these containers, which automates the deployment, scaling, and operations. This approach addresses the issue of low resource utilization and poor scalability typically encountered with traditional RDMA technologies when transmitting data streams from multiple devices.

4. Results and Analysis

In this section, comprehensive test results are presented to demonstrate the effectiveness of the proposed status-byte-assisted RDMA transmission mechanism and compare it with existing RDMA transmission techniques.

4.1. Test Environment and Experimental Design

4.1.1. Test Environment

The experiments were conducted on two x86 servers, each equipped with dual Intel Xeon Gold 2650 CPUs. Both servers ran the Ubuntu 22.04 LTS operating system with kernel version 6.2.0-33-generic. Each server was fitted with a 100 Gbps Mellanox ConnectX-5 InfiniBand NIC and connected via 100 Gbps fiber.

4.1.2. Experimental Design

The experiments were carried out in the following two parts to evaluate and verify the performance and characteristics of the status-byte-assisted RDMA transmission mechanism:

1. Comparison of transmission mechanisms: This part compares the performance of the status-byte-assisted RDMA transmission mechanism and the sliding window transmission mechanism in terms of multiple metrics, including synchronization control latency, transmission latency, throughput, and CPU utilization.
2. Validation of transmission mechanism specificity: This part aims to verify the advantages of the status-byte-assisted RDMA transmission mechanism in terms of flexible memory block control and its application capabilities in multi-task data transmission scenarios.

The goal of the above experiments is to comprehensively evaluate the effectiveness of the status-byte-assisted RDMA transmission mechanism in an edge computing environment and provide a detailed performance comparison with the existing methods.

4.2. Comparative Testing of Transmission Mechanisms and Results Analysis

In the comparison tests, the procedures for data filling and processing were standardized in both transmission mechanisms, so the duration of these steps was not factored into the results. Each sender and receiver was allocated three memory blocks for data transmission and reception. This choice of three memory blocks was intended to simulate a realistic scenario where at least two blocks are necessary for continuous data transmission from the sender, with an additional block providing redundancy to address potential delays in data processing by the receiver. This configuration helps to balance sending and receiving activities. Furthermore, the data packet sizes varied from 64 B to 8 MB to cover a broad spectrum of transmission scenarios. In addition, the upper limit of 8 MB was chosen to ensure fairness in the tests. These considerations were made to thoroughly evaluate the proposed mechanism under practical conditions.

4.2.1. Synchronization Control Delay

For the sliding window transmission mechanism, only the time from when the receiver successfully polled to the start of the next poll was calculated, excluding the time for the sender to process the ACK. This is because the sender handles ACKs in real time using a dedicated ACK polling thread, which does not interrupt the sending thread. Since the measurement steps for synchronization control latency and the amount of transmitted data remain constant regardless of the number of memory blocks, the communication control latency for the sliding window transmission mechanism was measured only once. In contrast, the status-byte-assisted RDMA transmission mechanism blocks the sending thread during communication control operations. Therefore, the time for the receiver to modify the status byte is negligible, while the amount of communication control transmission data varies with the number of memory blocks. As a result, the communication control latency

for the status-byte-assisted RDMA transmission mechanism was remeasured each time the number of memory blocks changed.

Figure 10 compares the synchronization control latency of various transmission mechanisms with different numbers of memory blocks. As shown in Figure 10, the synchronization control latency of the proposed status-byte-assisted RDMA transmission mechanism is significantly lower than that of the sliding window transmission mechanism for different number of memory blocks. For instance, with three memory blocks, the control latency for the status-byte-assisted RDMA transmission mechanism is 5.1 μs , compared to 70 μs for the sliding window transmission mechanism. This discrepancy arises because the sliding window transmission mechanism employs RDMA Send to transmit synchronization data, a bidirectional operation that requires both sending and receiving ends to generate Completion Queue Entries (CQEs), resulting in increased latency. Moreover, the control latency of the sliding window transmission mechanism does not vary with window size as it immediately returns an ACK after processing a memory block. In contrast, the control latency of the status-byte-assisted RDMA transmission mechanism increases with the number of memory blocks because it retrieves the status byte for all memory blocks simultaneously.

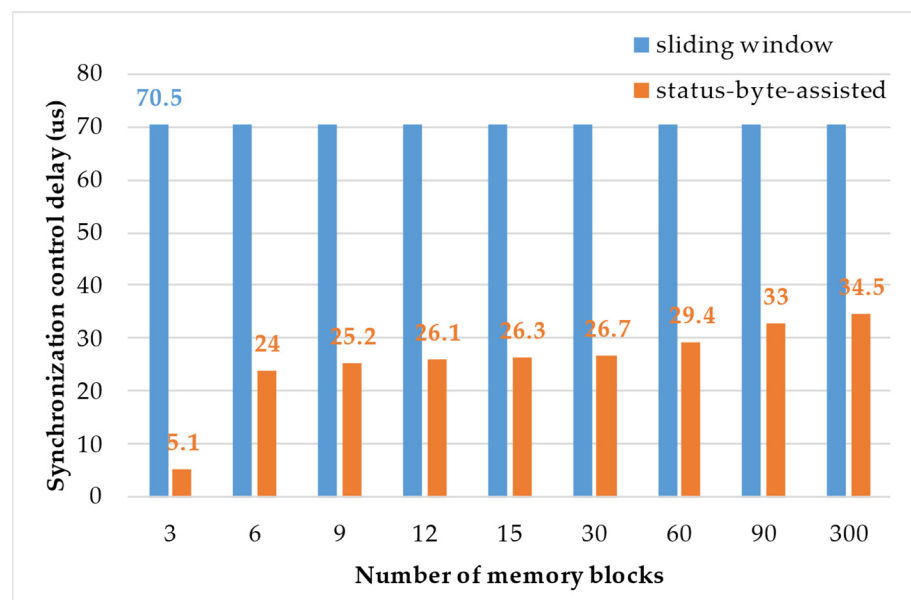


Figure 10. Comparison of synchronization control latency of various transmission mechanisms with different number of memory blocks.

4.2.2. Synchronization Control Delay Distribution

The status-byte-assisted RDMA transmission mechanism utilizes RDMA Read, which blocks the sending thread during the transmission of synchronization data. This approach results in relatively stable synchronization control latency. In contrast, the sliding window transmission mechanism uses RDMA Send, with the receiver sending synchronization data. This approach does not block the sending thread, and thus, it makes transmission latency susceptible to sender-induced fluctuations and leads to greater variance. Figure 11 illustrates the distribution of synchronization control latency versus the number of transmissions: (a) the sliding window RDMA transmission mechanism and (b) the status-byte-assisted RDMA transmission mechanism. As can be seen from Figure 11, under identical test conditions, the status-byte-assisted transmission mechanism exhibits consistent latency with significantly smaller fluctuations compared to the sliding window mechanism.

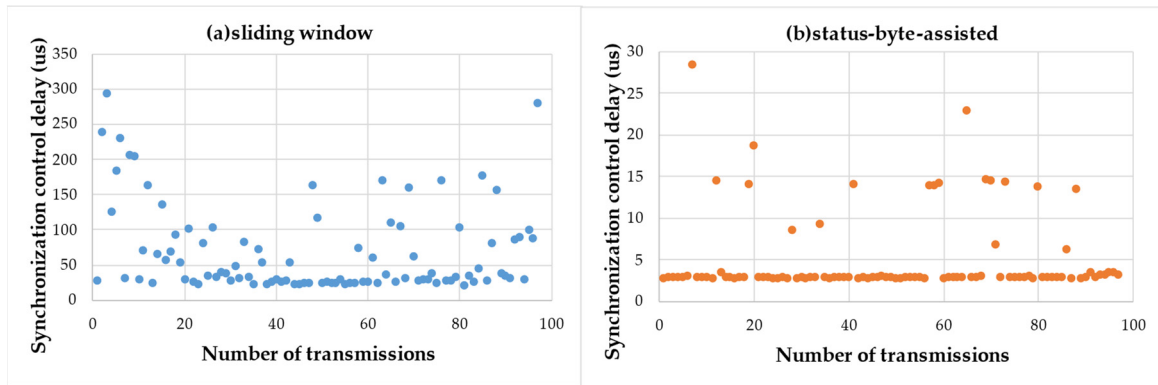


Figure 11. Distribution of synchronization control latency versus the number of transmissions: (a) the sliding window RDMA transmission mechanism and (b) the status-byte-assisted RDMA transmission mechanism.

4.2.3. Comparison of Transmission Delay

During the program initialization phase, the size of memory blocks used for transmission varies from 64 B to 8 MB. Subsequently, the latency of both the sliding window RDMA transmission mechanism and the status-byte-assisted RDMA transmission mechanism for transmitting a single memory block is measured. The transmission delay of each mechanism is recorded and averaged over 100 independent experiments.

Figure 12 compares the transmission delay of various RDMA transmission mechanisms under different packet sizes. As illustrated in Figure 12, the transmission latency of the status-byte-assisted RDMA transmission mechanism is reduced by 2 ns to 3.4 μ s compared to the sliding window RDMA transmission mechanism. The sliding window RDMA transmission mechanism uses RDMA Write With Immediate to transmit data, which requires the receiver to issue a receive request and generate a CQE. This results in significant latency due to interactions with the receiver. In contrast, the status-byte-assisted RDMA transmission mechanism performs two RDMA Write operations in a linked list: the first request does not generate a CQE and the second request uses inline data transmission. This approach reduces both NIC and memory exchanges, eliminates the need for receiver interaction, and thus achieves lower transmission latency due to its true unilateral nature.

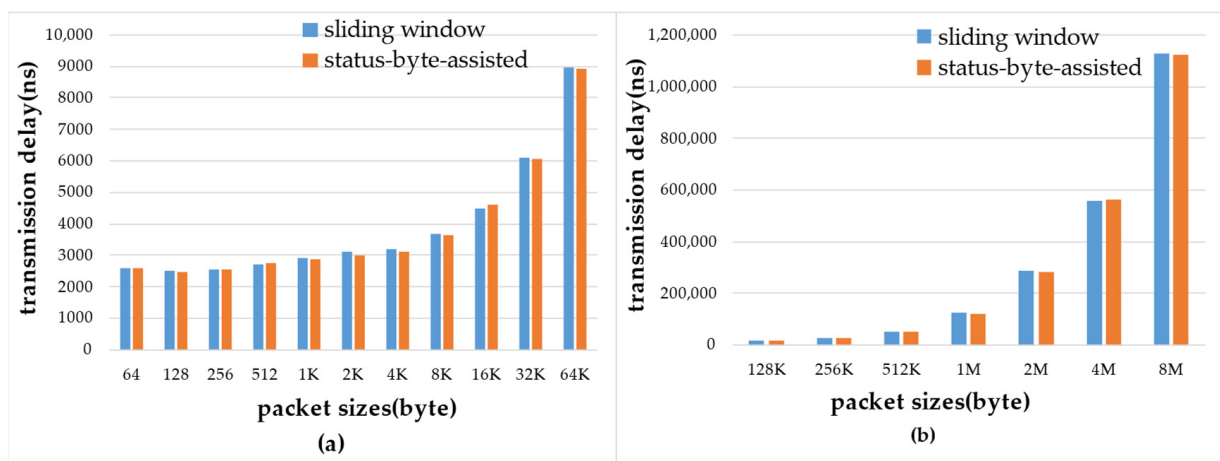


Figure 12. Comparison of the transmission delay for various RDMA transmission mechanisms under different packet sizes: (a) small data packet sizes (64 B to 4 KB) and (b) large data packet sizes (8 KB to 8 MB).

4.2.4. Throughput Test

For different packet sizes, the time taken to 1000 consecutive transmissions was measured for both RDMA transmission mechanisms. This measurement was repeated 10 times to calculate the average throughput. Figure 13 presents a comparison of throughput between the different RDMA transmission mechanisms. As illustrated in Figure 13, the throughput of the status-byte-assisted RDMA transmission mechanism is significantly higher than that of the sliding window RDMA transmission mechanism, especially for smaller packet sizes.

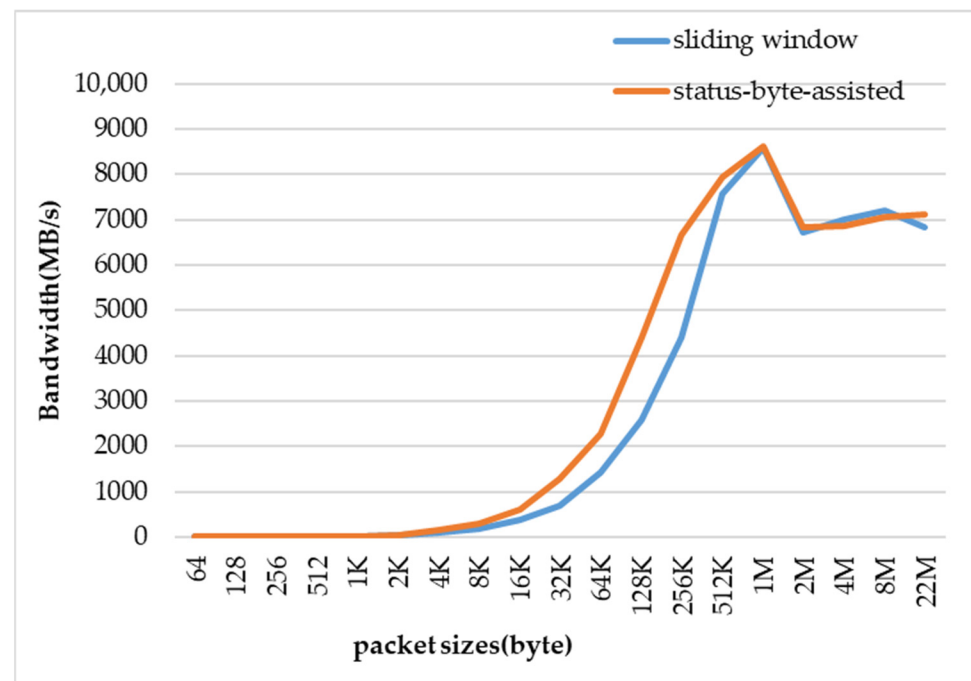


Figure 13. Comparison of the throughput of various RDMA transmission mechanisms for different packet sizes.

To illustrate throughput for different data packet sizes, Figure 14 compares the throughput of different RDMA transmission mechanisms: (a) for small data packet sizes (64 B to 4 KB) and (b) for large data packet sizes (8 KB to 22 MB). As shown in Figure 14, the throughput of the proposed status-byte-assisted RDMA transmission mechanism is notably higher than that of the sliding window RDMA transmission mechanism, especially for smaller packet sizes. For packets smaller than 1 KB, the sliding window RDMA transmission mechanism experiences high control latency that often exceeds the actual data transmission time, severely limiting its throughput. In contrast, the status-byte-assisted RDMA transmission mechanism shifts synchronization control to the sender and employs low-latency unilateral operations to query control information in real time, significantly reducing the control latency. This results in higher throughput for small packet sizes. As the packet size increases, the transmission time lengthens, which reduces the number of packets transmitted per unit of time. Consequently, the effect of communication control latency on throughput diminishes and the throughput of both transmission mechanisms converges. The throughput peaks when transmitting packets of 1 MB. However, for packets larger than 1 MB, the throughput tends to stabilize at approximately 7000 MB/s due to the increased overhead of retransmissions for excessively large packets.

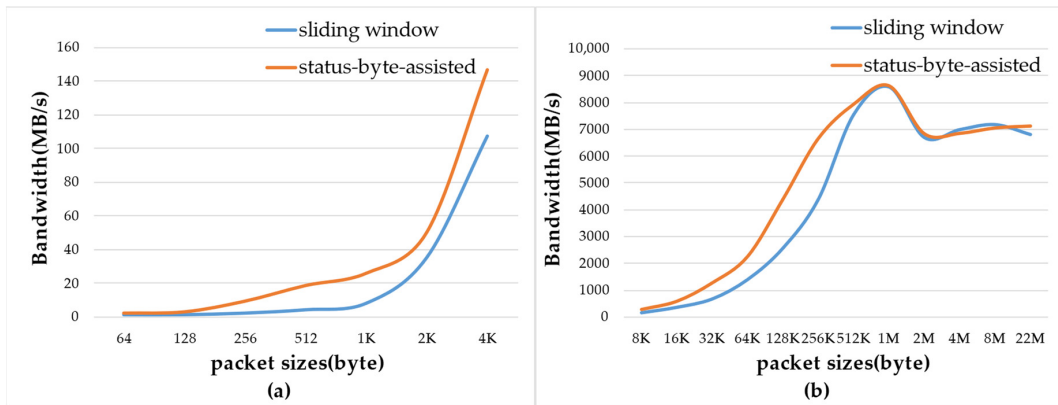


Figure 14. Comparison of the throughput of various RDMA transmission mechanisms for different packet sizes: (a) small data packet sizes (64 B to 4 KB) and (b) large data packet sizes (8 KB to 22 MB).

In summary, the status-byte-assisted RDMA transmission mechanism significantly outperforms the sliding window RDMA transmission mechanism for packet sizes smaller than 512 KB, and the throughput of both transmission mechanisms becomes relatively similar for packet sizes larger than 512 KB.

4.2.5. CPU Utilization Analysis

Optimizing CPU utilization is a core objective in improving the RDMA transmission mechanism’s efficiency, particularly under high-load conditions. Therefore, CPU utilization serves as a key indicator of system performance and resource management.

During the throughput test experiments, the CPU utilization on the sender side was recorded. Figure 15 compares CPU utilization for various RDMA transmission mechanisms at the sending end for different data packet sizes. As shown in Figure 15, the CPU utilization of the status-byte-assisted RDMA transmission mechanism is consistently lower than that of the sliding window RDMA transmission mechanism. This is because the sliding window RDMA transmission mechanism requires additional ACK receiving threads on the sender end to handle ACKs promptly, which results in higher CPU utilization. In contrast, the status-byte-assisted RDMA transmission mechanism eliminates the need for extra ACK receiving threads by leveraging one-sided operations, resulting in a 20% reduction in average CPU utilization.

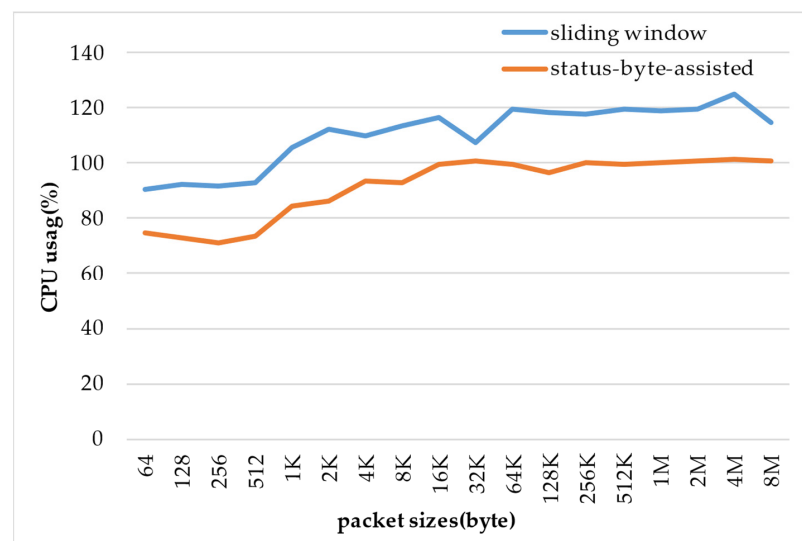


Figure 15. Comparison of CPU utilization for various RDMA transmission mechanisms at the sending end for different data packet sizes.

This reduction in CPU utilization directly supports our research objective of enhancing resource efficiency in RDMA transmission, especially in edge computing environments handling large-scale data transmissions. The findings demonstrate that the status-byte-assisted mechanism not only reduces computational overhead but also significantly enhances system performance under high-load conditions, thereby offering considerable potential for real-world applications.

4.2.6. NIC Resource Utilization Analysis

In terms of RDMA resource utilization, the status-byte-assisted RDMA transmission mechanism demonstrates high efficiency and significantly reduces RNIC resource usage on the receiver end. This is because the status-byte-assisted RDMA transmission mechanism utilizes one-sided operations and implements a stateless transmission strategy on the receiver end, which eliminates the need to maintain QP connection status and requires only maintaining locally registered memory block information. Consequently, it greatly reduces the consumption of RNIC on-chip resources on the receiver end.

Figures 16 and 17 compare the minimum RNIC resource consumption required during transmission for different RDMA transmission mechanisms. As shown in these figures, the RNIC resource consumption of the status-byte-assisted RDMA transmission mechanism is significantly lower than that of the sliding window RDMA transmission mechanism. In the sliding window transmission mechanism, to prevent the receiver from continuously sending ACKs, the sender initially issues three consecutive receive requests. Additionally, the sender may issue an RDMA Write With Immediate request and a receive request simultaneously, generating two CQEs. Consequently, the receiver might continuously receive three requests, so the minimum capacity required for both the receive queue and the completion queue is three. Overall, to reduce processing latency, the sender in the sliding window RDMA transmission mechanism initiates an ACK receiving thread and sets up two CQ queues to maintain continuous send and receive operations. Both sides issue multiple receive requests during initialization, which increases the consumption of RNIC on-chip resources. In contrast, in the status-byte-assisted RDMA transmission mechanism, the sender needs to issue two RDMA Write requests for each send operation, generating one CQE. An RDMA Read request, which also generates a CQE, is issued only when a status byte of 1 is encountered in the second status byte memory block. Since the sender serializes Write and Read requests, the minimum capacity of the sender's send queue is two and the completion queue's minimum capacity is one. The receiver does not explicitly send or receive any requests, so the capacity required for both the send queue and receive queue is zero. In addition, establishing an RDMA connection necessitates creating a completion queue with a minimum capacity of one.

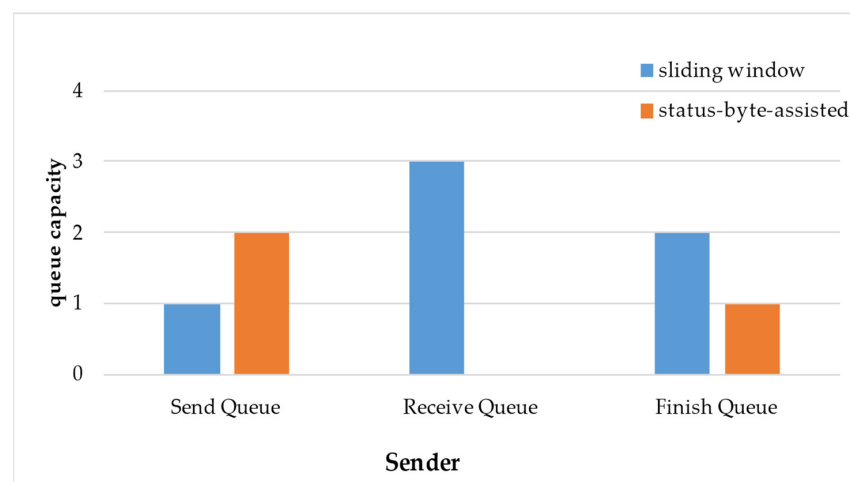


Figure 16. Comparison of the minimum RNIC resource consumption at the sending end for different RDMA transmission mechanisms.

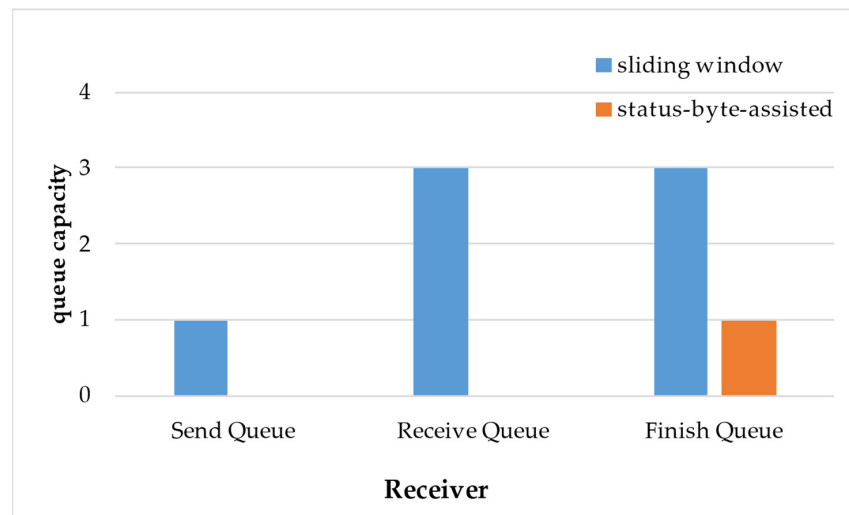


Figure 17. Comparison of the minimum RNIC resource consumption at the receiving end for different RDMA transmission mechanisms.

4.2.7. Special Features of the Status Byte in Memory Block Management

The status-byte-assisted RDMA transmission mechanism adds a status byte to each memory block, enabling operations on individual memory blocks. In contrast, the sliding window RDMA transmission mechanism requires sequential processing of memory blocks. In practical applications, the receiver might need to retain data within a memory block. The two transmission mechanisms handle this differently. To simulate this scenario, it is assumed that the receiver retains the data from the third memory block at the 100-ms mark and resumes processing at the 200-ms mark.

Figure 18 shows the throughput changes in the sliding window RDMA transmission mechanism and the status-byte-assisted RDMA transmission mechanism in response to specific demands. As depicted in Figure 18, the status-byte-assisted RDMA transmission mechanism experiences only a 12% decrease in throughput after the 100-ms mark and recovers at the 200-ms mark. In contrast, the sliding window RDMA transmission mechanism stops sending data at the 100-ms mark, resulting in zero throughput until the memory block is released at the 200-ms mark, after which the throughput recovers.



Figure 18. Throughput change in various RDMA transmission mechanisms in response to special demands.

The reason for this difference lies in the status-byte-assisted RDMA transmission mechanism's ability to set the status byte of a retained memory block to "unavailable" while continuing to process the next memory block. In this scenario, the sending thread skips over the retained memory block and continues processing until the device returns to normal. With only two memory blocks used for data transmission, the throughput only slightly decreases. Therefore, this flexible memory block handling capability of the status-byte-assisted RDMA transmission mechanism enhances the system's ability to manage various unexpected situations. In contrast, in the sliding window RDMA transmission mechanism, memory block sending and receiving must occur sequentially. Consequently, a subsequent memory block cannot be sent before the preceding one, and the ACK for a subsequent memory block cannot be sent before the preceding memory block's ACK. As a result, its throughput drops to zero.

One of the key advantages of the status-byte-assisted RDMA transmission mechanism in practical applications is its ability to maintain transmission continuity effectively. When network conditions change or delays occur, this mechanism dynamically adjusts the allocation of memory blocks, ensuring that the data transmission process is not interrupted. This continuity is particularly crucial in critical applications such as real-time video streaming and autonomous systems, where any pause or interruption could lead to serious consequences. By quickly adapting to changing transmission conditions, this mechanism significantly reduces the risk of system downtime, ensuring efficient and stable operation.

Although the status-byte-assisted RDMA transmission mechanism provides significant flexibility in memory block management, this flexibility also comes with some potential drawbacks. Specifically, this mechanism requires status byte checks during data transmission to flexibly manage the memory blocks. The status-byte-assisted mechanism does not need to check the memory block status when the memory block is available. However, it needs to perform an additional check to find an available memory block when a particular memory block is unavailable, which could increase additional CPU resource usage. Moreover, the CPU resource consumption of the status-byte-assisted transmission mechanism remains lower than that of the sliding window transmission mechanism as the data transmission of the status-byte-assisted RDMA transmission mechanism is handled by a single thread.

4.3. System Testing for the Multi-Tasking Video Streaming Delivery

The status-byte-assisted RDMA transmission mechanism supports multi-task processing because each memory block has an independent status byte indicating its usage status. In contrast, the sliding window RDMA transmission mechanism utilizes only two pointers, which require memory blocks to be used and released sequentially. This can lead to contention for pointer resources when multiple threads send data, and the order of returned ACKs can be inconsistent due to various factors when multiple threads receive data. Therefore, the sliding window RDMA transmission mechanism does not effectively support parallel multi-threading. For transmitting real-time video streams from multiple cameras, typically, multiple sliding window programs are needed for the sliding window RDMA transmission mechanism, whereas only one status-byte-assisted RDMA transmission program is necessary for the status-byte-assisted RDMA transmission mechanism.

In this experiment, 12 simulated cameras were used, each filling memory blocks with 25 frames of data per second; the size of each frame is $640 \times 480 \times 3$ bytes, and the system's throughput variation over time is recorded. Figure 19 shows the throughput variations in multitasking scenarios using both the sliding window RDMA transmission mechanism and the status-byte-assisted RDMA transmission mechanism. As shown in Figure 19, the overall throughput of the status-byte-assisted RDMA transmission mechanism remains relatively stable, concentrating around 7000 MB/s over a longer period. This is because the status-byte-assisted RDMA transmission mechanism sequentially queries the first status-byte memory block and sends memory blocks serially. The periods where throughput is zero indicate that no data need to be sent, during which the program enters a sleep

mode and periodically checks for new data to send. In contrast, the throughput of the sliding window RDMA transmission mechanism fluctuates significantly over time, with a broader range of transmission rates compared to the status-byte-assisted mechanism. This is because each sliding window RDMA transmission program’s transmission times are independent, and multiple memory blocks can sometimes be transmitted in parallel, causing substantial throughput variations and leading to unstable performance. Moreover, when transmitting multiple data streams over the same period, the status-byte-assisted RDMA transmission mechanism demonstrates more stable throughput than the sliding window RDMA transmission mechanism, thus enhancing the system’s stability and reliability while avoiding resource waste and bottlenecks.

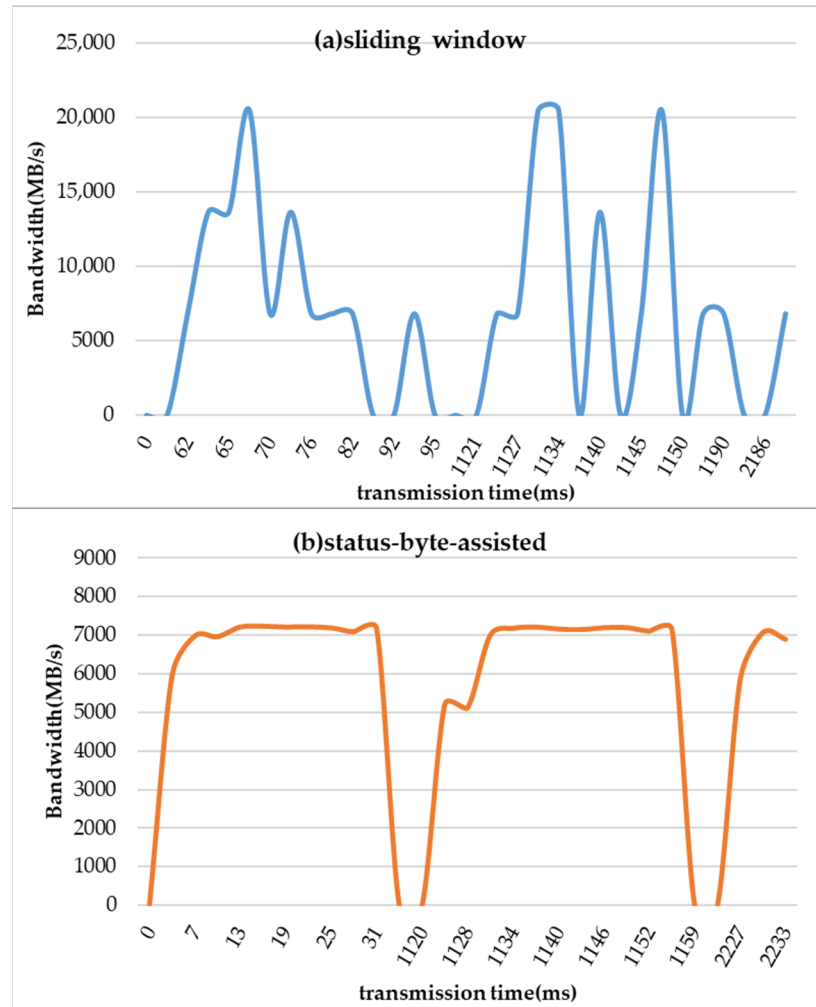


Figure 19. Throughput variation in multitasking: (a) the sliding window RDMA transmission mechanism and (b) the status-byte-assisted RDMA transmission mechanism.

During the transmission process, the average CPU utilization of the status-byte-assisted RDMA transmission mechanism is 106% and memory usage is 1.3%. In contrast, the sliding window RDMA transmission mechanism has an overall average CPU utilization of 168% and memory usage of 2.4%. In summary, the status-byte-assisted RDMA transmission mechanism maintains a smooth throughput curve by efficiently organizing tasks on the sending end and concentrating them into a single QP, while also saving CPU and memory resources and reducing RNIC resource consumption.

To summarize the key findings and facilitate a clear comparison between the two RDMA transmission mechanisms, the following Table 3 is provided. This table highlights

the main strengths and weaknesses of each approach, offering a concise overview of the performance, resource utilization, and specific features discussed above.

Table 3. Comparative analysis of RDMA transmission mechanisms.

Comparison Dimension	Sliding Window-Based Mechanism	Status-Byte-Assisted Mechanism
Synchronization Control Latency	High, especially due to delays introduced by ACK processing.	Low, using RDMA Read for unidirectional synchronization, significantly reducing control latency.
Transmission Latency	High, involves bidirectional operations and processing of ACKs.	Low, optimized transmission method using unilateral transmission.
Throughput	High, but performs poorly with small packets, limiting throughput.	High, especially for small packets, significantly improving throughput.
CPU Utilization	High, need to create and manage additional ACK receiving threads.	Low, reduces CPU utilization by using one-sided operations.
RNIC Resource Consumption	High, requires bidirectional communication.	Low, one-sided operation and stateless connections reduce resource consumption.
Memory Block Management Characteristics	Sequential processing, less suited for parallel data transmission.	Flexible handling, allows independent management of each memory block and supports parallel processing.
Limitations	Poor performance with small packets, limiting throughput.	Additional checks of status bytes may increase CPU resource consumption.
Applicable Scenarios	Suitable for general data transmission and supports basic transmission needs.	Suitable for multi-task data transmission scenarios, especially video streaming.

4.4. Future Work and Potential Improvements

Although the status-byte-assisted RDMA transmission mechanism demonstrates significant advantages over the existing methods in optimizing multi-task video streaming, there are several aspects that warrant further exploration and improvement.

One area for future research is optimizing status byte management. While the status-byte-assisted RDMA transmission mechanism effectively manages memory blocks and improves transmission efficiency, the handling and determination of status bytes might increase system overhead. Therefore, future research could explore more efficient algorithms for managing status bytes to reduce memory and computational resource consumption. For instance, investigating methods to dynamically adjust the frequency of status byte updates could optimize the system's performance. Additionally, integrating advanced technologies such as machine learning to predict and optimize status byte behavior may lead to the development of more intelligent and autonomous systems. This integration could not only enhance performance but also minimize the need for manual intervention in data transmission management.

5. Conclusions

With the development of edge computing, there is an increasing need for edge servers to collaboratively process data, accelerate data transmission, and reduce latency. This paper evaluates the strengths and weaknesses of the sliding window RDMA transmission mechanism, particularly in relation to the multitasking requirements of the current edge computing environment and the asymmetry inherent in RDMA's unilateral operations. To address the shortcomings of the existing RDMA transmission mechanism, we propose an RDMA data transmission mechanism based on status byte. The proposed status-byte-assisted RDMA transmission mechanism modifies the data transmission and synchronization methods of the sliding window RDMA transmission mechanism. By utilizing the unique properties of status-byte-assisted RDMA transmission to precisely control memory blocks, it provides a solution for multitasking processing and is well-suited

for transmitting real-time video streams. Finally, we provide extensive experimental results to validate the performance of the status-byte-assisted RDMA transmission mechanism.

Experimental analysis results demonstrate that the proposed status-byte-assisted RDMA transmission mechanism improves all performance metrics compared to the sliding window RDMA transmission mechanism. First, in terms of transmission latency and throughput, when transmitting data packets ranging from 64 B to 8 MB, the status-byte-assisted RDMA transmission mechanism reduces latency by 2 ns to 3.4 μ s compared to the sliding window scheme, and its throughput exceeds that of the sliding window scheme for packet sizes up to 1 MB. Notably, for 256 B packets, the status-byte-assisted RDMA transmission mechanism achieves a throughput 4.6 times higher than the sliding window scheme. Second, in terms of resource utilization, experimental results show that the status-byte-assisted RDMA transmission mechanism reduces average CPU utilization by 20% under the same workload, while minimizing NIC resource consumption. Lastly, the status-byte-assisted RDMA transmission mechanism exhibits exceptional flexibility in memory block transmission and support for multitasking, exhibiting greater stability and higher throughput in multitasking scenarios. Additionally, in conjunction with multitasking transmission, we utilize containers as the smallest computing units for data processing, with K8s managing these containers by automating deployment, scaling, and operations. This achieves more flexible resource coordination and allocation among edge servers. These improvements make the status-byte-assisted RDMA transmission mechanism a novel solution for efficient data transmission in edge computing environments, with broad application prospects.

Author Contributions: Conceptualization, D.X., H.Y. and W.S.; Funding acquisition, W.Z. and W.S.; Investigation, D.X. and H.Y.; Methodology, D.X.; Project administration, W.Z. and W.S.; Resources, W.Z.; Supervision, H.Y. and W.S.; Validation, D.X.; Writing—original draft, D.X.; Writing—review and editing, H.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partly funded by the Special Projects for Key R&D Tasks in the Autonomous Region of Xinjiang Grant No. 2022B01009.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge computing: Vision and challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
2. Zhou, X.; Ke, R.; Yang, H.; Liu, C. When intelligent transportation systems sensing meets edge computing: Vision and challenges. *Appl. Sci.* **2021**, *11*, 9680. [CrossRef]
3. Liu, P.-C.; Tseng, H.-E.; Yang, S.-K.; Kuo, F.-H. New Multi-Access Network Transmission Technology to Enhance Edge Computing. In Proceedings of the 2021 IEEE 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS), Tainan, Taiwan, 8–10 September 2021; pp. 9–12. [CrossRef]
4. A Quick Look at the Differences: RDMA vs. TCP/IP. Updated on 20 February 2023. Available online: <https://community.fs.com/article/roce-vs-infiniband-vs-tcp-ip.html> (accessed on 1 July 2024).
5. Mitchell, C.; Geng, Y.; Li, J. Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store. In Proceedings of the 2013 USENIX Annual Technical Conference (USENIX ATC 13), San Jose, CA, USA, 26–28 June 2013; pp. 103–114.
6. He, Q.; Gao, P.; Zhang, F.; Bian, G.; Zhang, W.; Li, Z. Design and optimization of a distributed file system based on RDMA. *Appl. Sci.* **2023**, *13*, 8670. [CrossRef]
7. Wang, Z.; Luo, L.; Ning, Q.; Zeng, C.; Li, W.; Wan, X.; Xie, P.; Feng, T.; Cheng, K.; Geng, X. SRNIC: A scalable architecture for RDMA NICs. In Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23), Boston, MA, USA, 17–19 April 2023; pp. 1–14.
8. Gil, M.-S.; Moon, Y.-S. SPinDP: A High-Speed Distributed Processing Platform for Sampling and Filtering Data Streams. *Appl. Sci.* **2023**, *13*, 12998. [CrossRef]

9. Mellanox ConnectX-5 Product Brief. 2020. Available online: <https://nvdam.widen.net/s/pkxbnmbgkh/networking-infiniband-datasheet-connectx-5-2069273> (accessed on 22 July 2024).
10. Mellanox ConnectX-6 Product Brief. 2020. Available online: <https://nvdam.widen.net/s/5j7xtzqfxd/connectx-6-infiniband-datasheet-1987500-r2> (accessed on 22 July 2024).
11. Tu, Y.; Han, Y.; Jin, H.; Chen, Z.; Zhao, Y. RDMA Based Performance Optimization on Distributed Database Systems: A Case Study with GoldenX. In Proceedings of the Wireless Algorithms, Systems, and Applications: 16th International Conference, WASA 2021, Nanjing, China, 25–27 June 2021; Part II 16. Springer: Berlin/Heidelberg, Germany; pp. 237–248. [CrossRef]
12. Use Containers to Build, Share and Run Your Applications. Available online: <https://www.docker.com/resources/what-container/> (accessed on 22 July 2024).
13. Kubernetes Documentation. Available online: <https://kubernetes.io/docs/home/> (accessed on 22 July 2024).
14. Wang, S.; Zhang, X.; Zhang, Y.; Wang, L.; Yang, J.; Wang, W. A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access* **2017**, *5*, 6757–6779. [CrossRef]
15. Taranov, K.; Rothenberger, B.; De Sensi, D.; Perrig, A.; Hoefler, T. NeVerMore: Exploiting RDMA Mistakes in NVMe-oF Storage Applications. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, Los Angeles, CA, USA, 7–11 November 2022; pp. 2765–2778. [CrossRef]
16. Shi, W.; Wang, Y.; Corriveau, J.-P.; Niu, B.; Croft, W.L.; Peng, M. Smart shuffling in MapReduce: A solution to balance network traffic and workloads. In Proceedings of the 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), Limassol, Cyprus, 7–10 December 2015; pp. 35–44. [CrossRef]
17. Wu, Y.; Ma, T.; Su, M.; Zhang, M.; Chen, K.; Guo, Z. RF-RPC: Remote fetching RPC paradigm for RDMA-enabled network. *IEEE Trans. Parallel Distrib. Syst.* **2018**, *30*, 1657–1671. [CrossRef]
18. About InfiniBand. Available online: <https://www.infinibandta.org/about-infiniband/> (accessed on 22 July 2024).
19. RDMA in Data Centers: Looking Back and Looking Forward. 2017. Available online: <https://conferences.sigcomm.org/events/apnet2017/slides/cx.pdf> (accessed on 22 July 2024).
20. Zhu, Y.; Yu, W.; Jiao, B.; Mohror, K.; Moody, A.; Chowdhury, F. Efficient user-level storage disaggregation for deep learning. In Proceedings of the 2019 IEEE International Conference on Cluster Computing (CLUSTER), Albuquerque, NM, USA, 23–26 September 2019; pp. 1–12. [CrossRef]
21. Abbasi, U.; Bourhim, E.H.; Dieye, M.; Elbiaze, H. A performance comparison of container networking alternatives. *IEEE Netw.* **2019**, *33*, 178–185. [CrossRef]
22. Cassell, B.; Szepesi, T.; Wong, B.; Brecht, T.; Ma, J.; Liu, X. Nessie: A decoupled, client-driven key-value store using RDMA. *IEEE Trans. Parallel Distrib. Syst.* **2017**, *28*, 3537–3552. [CrossRef]
23. Wang, Z.; Wan, X.; Zeng, C.; Chen, K. Accurate and Scalable Rate Limiter for RDMA NICs. In Proceedings of the 7th Asia-Pacific Workshop on Networking, Hong Kong, China, 29–30 June 2023; pp. 15–20. [CrossRef]
24. Wang, X.; Chen, G.; Yin, X.; Dai, H.; Li, B.; Fu, B.; Tan, K. StaR: Breaking the scalability limit for RDMA. In Proceedings of the 2021 IEEE 29th International Conference on Network Protocols (ICNP), Dallas, TX, USA, 1–5 November 2021; pp. 1–11. [CrossRef]
25. Graham, R. Dynamically Connected Transport. In Proceedings of the Annual OFA Workshop, The Hague, The Netherlands, 11 April 2014; pp. 13–14.
26. Chen, Y.; Lu, Y.; Shu, J. Scalable RDMA RPC on reliable connection with efficient resource sharing. In Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, 25–28 March 2019; pp. 1–14. [CrossRef]
27. Ziegler, T.; Nelson-Slivon, J.; Leis, V.; Binnig, C. Design Guidelines for Correct, Efficient, and Scalable Synchronization Using One-Sided RDMA. *Proc. ACM Manag. Data* **2023**, *1*, 1–26. [CrossRef]
28. RDMA Tutorial. 2018. Available online: <https://www.doc.ic.ac.uk/~jgiceva/teaching/ssc18-rdma.pdf> (accessed on 22 July 2024).
29. Kalia, A.; Kaminsky, M.; Andersen, D.G. Design guidelines for high performance RDMA systems. In Proceedings of the 2016 USENIX Annual Technical Conference (USENIX ATC 16), Denver, CO, USA, 22–24 June 2016; pp. 437–450.
30. Hemmatpour, M.; Montrucchio, B.; Rebaudengo, M. Communicating Efficiently on Cluster-Based Remote Direct Memory Access (RDMA) over InfiniBand Protocol. *Appl. Sci.* **2018**, *8*, 2034. [CrossRef]
31. Tsugami, R.; Fukui, T.; Narikawa, S. RDMA Transmission Control Method: Using Network Resource Allocation For Wide-Area Data Collection. In Proceedings of the 2024 IEEE 21st Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 6–9 January 2024; pp. 578–581. [CrossRef]
32. Zhang, Z.; Cai, D.; Zhang, Y.; Xu, M.; Wang, S.; Zhou, A. FedRDMA: Communication-Efficient Cross-Silo Federated LLM via Chunked RDMA Transmission. In Proceedings of the 4th Workshop on Machine Learning and Systems, Athens, Greece, 22 April 2024; pp. 126–133. [CrossRef]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.