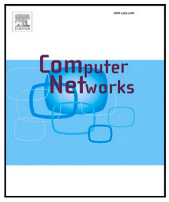
Computer Networks 250 (2024) 110541

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



SECM: Securely and efficiently connections setup using RDMA-CM

Xingyu Guo [a], Guo Chen [a,*], Xiaoning Zhan [a], Ting Qu [b], Zhaojiao Han [b]

[a] College of Computer Science and Electronic Engineering, Hunan University, China
[b] Huawei, China

ARTICLE INFO

Keywords:
RDMA
Data center network
Connection setup

ABSTRACT

Remote Direct Memory Access (RDMA) releases the potential of the data center as it bypasses the kernel. However, establishing and managing RDMA connections is a significant challenge due to the complex topology of RDMA networks. We present SECM, a secure and efficient RDMA Communication Management (CM) library, which can be used for distributed or resilient RDMA network connection setup. The main idea is to split the RDMA-CM connection setup phase, reuse resources between different connections, and perform multiple connection setups simultaneously in a pipelined manner. At the same time, adding message authentication codes to the CM protocol enables authentication of connection requests and prevents malicious connection requests. SECM establishes 16 connections in 1.42 times the time of 1 connection, tens or hundreds of times faster than verbs. SECM is compatible with existing commercial RNICs and provides services in a user-state-driven manner, with low CPU overhead, low memory usage, and no impact on the RDMA data level.

1. Introduction

Remote Direct Memory Access (RDMA), a high-performance network stack, is being widely utilized in data centers [1–3]. By offloading the network stack of the data path from the operating system to the RDMA Network Interface Card (RNIC), a node can directly access data in a remote node without the involvement of the operating system, avoiding the overhead incurred by the traditional TCP/IP on the kernel.

Before transmitting data using a high-speed RDMA connection, it is essential to create a connection between the communicating nodes [4,5]. Presently, RDMA supports two connection setup methods: out-of-band connection and connection setup based on Communication Management (CM) API [6]. The latter, being a native RDMA-supported method, is widely applicable across InfiniBand (IB) and RDMA over Converged Ethernet (RoCE) [7,8] protocols. As a result, it stands as the most widely adopted connection setup method in current data center applications [9]. Therefore, this paper primarily explores the connection setup method based on the CM API.

Unfortunately, RDMA has a slow and insecure connection setup. The latency of creating an RDMA connection (30 ms) is higher than its data path operation. Since the current demand for network latency in large-scale Key-Value Storages(KVS) services have reached the microsecond-scale [10–12], high connection times may significantly decrease the application performance. For instance, it may introduce delays when scaling resources to handle peak loads. Essentially, the current CM API's connection setup interface operates serially, thereby increasing the latency for subsequently created connections. In addition, the messages of RDMA-CM are transmitted in plaintext. This exposes the connection information to potential attackers, leading to the risk of sensitive data leakage or even connection crashes.

An intuitive approach to mitigate connection setup latency is to utilize multi-threading for creating RDMA connections. However, the creation of a thread incurs consumption of CPU and memory resources, which are particularly valuable in data centers. In extreme cases, where the number of threads exceeds the count of physical CPU cores, it may even deplete the CPU and memory resources of the data center. Second, maintaining a cache of all connections between all nodes takes up a lot of host resources (especially memory) [13–15]. This is unacceptable in a data center trying to use resources efficiently.

In this paper, we propose SECM, a **S**ecure and **E**fficient connection setup of **CM** API. It divides the connection setup into five sub-stages and uses pipelined parallel execution to avoid serial task execution and excessive resource consumption. Meanwhile, an authentication mechanism is integrated into the CM to ensure the security of connection setup. Note that all the work of SECM is done at the software layer and only enhancements are made to the control path, therefore SECM does not affect the data level transfer and has no impact on RDMA performance.

We implemented SECM in a user-state dynamic library with no additional hardware dependencies. Our experiments show that SECM

* Corresponding author.
E-mail address: guochen@hnu.edu.cn (G. Chen).

https://doi.org/10.1016/j.comnet.2024.110541
Received 21 May 2023; Received in revised form 10 December 2023; Accepted 24 May 2024
Available online 28 May 2024

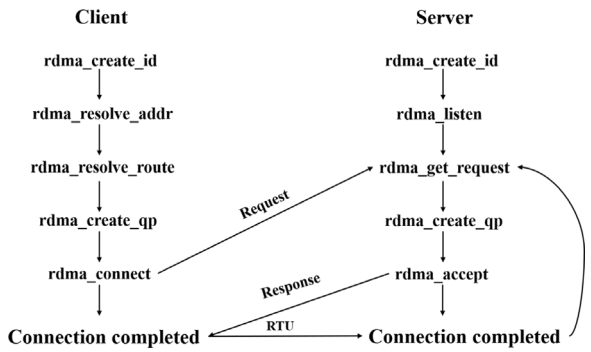


Fig. 1. An overview of the basic operation for the active (Client) and the passive (Server).

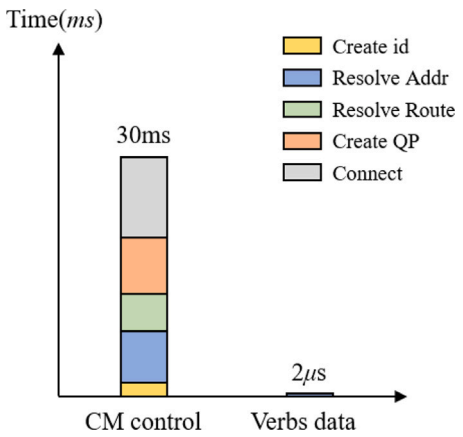


Fig. 2. Huge time difference between RDMA-CM connection control paths and Verbs data paths (issuing 8B READ).

can drastically reduce the time for large-scale connection setups, taking only 1.42 times the time of a single connection to establish 16 connections, especially when establishing connections with more than a thousand nodes, each connection takes only 200us on average, which is tens of times shorter compared to the time consumed in serial. At the same time, it can reduce the success rate of malicious connection requests to zero.

In summary, we make the following contributions.

- SECM optimizes the performance of CM in large-scale node scenarios and achieves RDMA-CM connection setup with low overhead and low latency. The experimental results show that SECM has excellent performance at the control level and does not have any impact on the data level.
- Design and implementation of secure connection setup. Without changing the existing CM protocol, message authentication codes are added to the connection setup packets to authenticate the requester.

In this paper, we first analyze the background of current RDMA connection establishment, the shortcomings of existing work and the motivation of our work. Secondly, we describe the scheme design and corresponding implementation, and conduct experimental proofs in subsequent chapters. Finally, we summarize the article and present our conclusions.

2. Background and motivation

2.1. RDMA-CM

Currently, commercial RDMA network cards support three connection modes, namely reliable connection (RC), unreliable connection (UC) and unreliable datagram (UD). Because RC mode supports all RDMA Verbs and can provide lossless networks for data center applications, RC mode is the most widely used mode in current data center applications. Therefore, this article mainly studies RDMA connection setup based on RC mode [16,17].

RDMA-CM is a connection setup method natively supported by RDMA, which has a set of customized message format, interaction process and user interface (*librdmacm*). The Fig. 1 shows the specific flow of creating an RDMA-CM connection. ① The application allocates an *rdma_cm_id* identifier, which is functionally similar to a socket and serves as the context of RDMA-CM (**Create ID**). ② *rdma_resolve_addr* obtains a local RDMA device to access the remote address (**Resolve Address**). ③ *rdma_resolve_route* determines a route to the remote address (**Resolve Route**), ④ *rdma_create_qp* allocates a queue pair (QP) for the communication (**Create QP**), and ⑤ *connect* exchange the QP information to the remote node using a handshake protocol (**Connect**). It is easy to use and has detailed implementations that have been proven in production environments, such as reliable transmission and timeout mechanisms [18]. However, RDMA-CM still has shortcomings, it is slow and insecure.

2.2. RDMA-CM disadvantages

RDMA-CM is slow. Fig. 2 shows that the latency to create a connection is higher than the latency to transfer data. To quantify the cost of RDMA-CM, we have segmented the connection setup time for detailed analysis. Consider a node that may need to initiate RDMA connections to multiple nodes simultaneously (e.g. KVS). With the current RDMA OFED (OpenFabrics Enterprise Distribution) dynamic library, which performs RDMA-CM operations serially, later connection operations must wait for the previous operation to complete, which can increase the transfer latency. A simple solution is to create multiple threads to perform connection setups in parallel. However, how many threads to create is a difficult trade-off issue. Data center resources are limited and valuable, creating too many threads will consume a large amount of CPU and memory resources, affecting the performance of other applications in the data center; creating too few threads will result in performance degradation when the number of connections is greater than the number of threads, as the CPU thread scheduling mechanism frequently switches contexts.

RDMA-CM is insecure. (1) RDMA-CM transfers data in plaintext [19,20]. This exposes the connection information to the network, and an attacker can modify or spoof the connection packets to corrupt the normal connection. (2) RDMA-CM lacks authentication. In a traditional TCP/IP network, it is possible to set up a whitelist (e.g., firewall) to deny access to unprivileged users [21–23], because TCP/IP passes through the kernel. But RDMA bypasses the kernel, so traditional firewalls and security groups are no longer available for it [24]. Currently, the passive side of RDMA-CM is in a completely open state, and any node with information about the server side can connect to it. As a result, the insecurity of RDMA-CM can cause losses to the users due to information leakage, and even cause the service to fail.

3. Design

The goal of SECM is to provide a secure and efficient method for establishing RDMA connections between nodes in large-scale clusters, reducing the time required to establish full network RDMA connections between nodes. Additionally, SECM aims to enhance the authentication mechanism for CM connection setup, ensuring the legitimacy and security of the established connections. In this section, we present the system architecture of the SECM design, describing its key components and functions in detail.

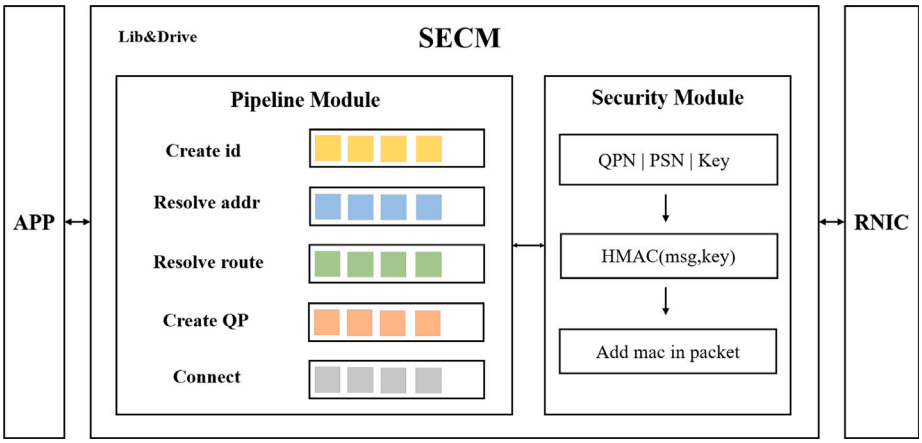


Fig. 3. An overview of SECM Architecture (The square represents tasks awaiting processing at each stage).

3.1. Overview

SECM is designed based on the IB specification and is compatible with existing RDMA hardware devices. SECM provides connection establishment services in the form of APIs, so that users do not need to understand the complex RDMA establishment mechanism, but only need to provide the information and initialization parameters of the peer node, and then SECM performs the creation of QPs, CQs, and other resources, as well as the necessary information exchange. On this basis, SECM provides a connection authentication mechanism that can authenticate connection requests to ensure that all RDMA connections are legitimate (see Fig. 3).

3.2. Pipeline

By analyzing the previous chapters, it can be seen that RDMA-CM provides default interfaces which are invoked synchronously in order to simplify the complexity of API usage. Inside the RDMA-CM dynamic library provides a mode of asynchronous API invocation, which includes the more time-consuming stages of resolving addresses, resolving routes, and requesting connections, which means that we can split the CM connection establishment process and design it hierarchically to achieve faster and more efficient connection establishment.

When establishing an RDMA-CM connection, a series of stages is required, and there is a strict back-and-forth correlation between these stages, and we must handle the order of function calls correctly. The tasks of these sub-stage are fixed and are performed in the following order: creating the connection ID, resolving the remote address, resolving the route, creating the QP, and initiating the connection request. When a connection completes the connection ID creation stage, it enters the address resolution process. Eventually all the stages are executed in order to complete the connection. If there are multiple connections, each connection follows the same process. This processing is similar to the behavior of the CPU when executing instructions, so we designed the method to process multiple RDMA-CM connection establishment in a pipelined manner.

We use threads and work queues to design the pipeline. We create work queues for each step, as well as a corresponding work thread for it. Each thread is only responsible for handling the relevant interface calls for this stage, interacting with the kernel state through the user-state interface. Since the asynchronous calling mode is enabled, the program does not block here after issuing the relevant command to the hardware device, but continues to execute the later statements. As to when the hardware device returns the relevant information, SECM uses event handler functions to process these callback events. When the function call that initiates the current connection is made, the creation of the next connection is started directly without waiting for a response

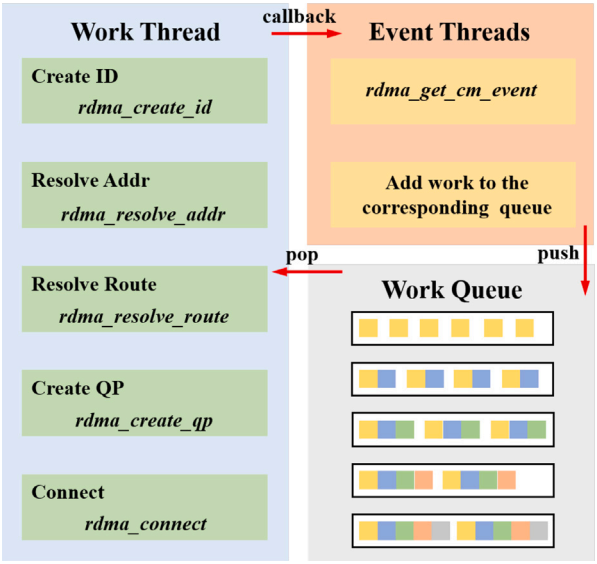


Fig. 4. How SECM use pipeline to execute RDMA-CM (different colored blocks represent different execution steps of RDMA-CM).

from the hardware device or the remote device. The time it takes to call a kernel function to initiate a command to the hardware is much less than the time it takes for the device to respond. For example, calling the rdma_connect function takes only microseconds, under 10us in most cases. However, this part of waiting for the callback event of rdma_connect to return is relatively long and consists of the far-end processing time and the RTT. The far-end processing delay is around tens of milliseconds, and the RTT time depends on the network. In particular, this time may be longer when the network topology is very complex and large. Therefore, using asynchronous calls, it is possible to quickly send establishment requests for multiple connections, and then wait for successful establishment responses for multiple connections at the same time (see Fig. 4).

Since the function is called asynchronously, we need to handle the callback function correctly. When the result of one stage is returned, we need to process it in some way and then add it to the processing queue of the next stage, and so on, to complete all stages of the function call.

3.3. Security

RDMA Connection Manager receives requests through a reserved QP of type UD(Unreliable Datagram) in the kernel [25], so it can

receive packets from any node. All CM packets headers and payloads are transmitted in plaintext and there is no authentication mechanism. Only one mechanism is used for packet integrity checking, each packet contains a 32-bit ICRC checksum. However, the algorithm and initial seed for this ICRC checksum are publicly available, which means that the ICRC can be recalculated and does not do anything to protect packets. Since RDMA packets bypass the kernel, traditional authentication mechanisms (security groups, firewalls) cannot be applied to RDMA. This means that an attacker can modify or forge packets established by a CM connection to establish an illegal connection or disrupt a normal connection. SECM proposes the use of Message Authentication Codes (MAC) [26] for packet authentication and integrity checks to ensure that CM packets cannot be modified or forged.

Message Authentication Code (MAC) generates a fixed length code by performing cryptographic hash operation on a message, which is used to verify the integrity and authenticity of the message to ensure that the message has not been tampered with or forged during transmission. The HMAC [27] is a hash-based message authentication code, its algorithm formula is:

$$HMAC(k, m) = H(k' \oplus opad, H(k' \oplus ipad, m)) \quad (1)$$

H is the hash function, k is the key, m is the message being authenticated, *ipad* and *opad* are the padding constants, and k' is the key after padding.

HMAC requires two inputs, the protected message and the key. By analyzing CM data messages, QP Number and starting packet sequence number (PSN) are necessary and unique values for a connection. To protect this critical information, the SECM takes the QPN and the starting PSN as inputs to the MAC message and computes them using a predefined MAC algorithm. This ensures that these fields are not tampered with or forged during transmission and provides identification for authentication.

MAC supports a variety of hash algorithms, and different algorithms can be selected depending on the security level required. Different hash algorithms may require different key lengths and key distribution mechanisms. Nodes can select appropriate hash algorithms based on their security needs and capabilities and provide the corresponding keys to the SECM for generating the MAC.

4. Implementation

We implemented SECM in the software driver of *MLNX_OFED_LINUX -5.8-3.0.7.0* [28] version, and since SECM is implemented in the pure software layer, it can be easily migrated to other versions. We first present the parallel connection establishment implementation and then the security enhancement implementation.

4.1. Pipeline connection setup

We split the CM connection setup into five steps, each with a processing unit responsible for only one of the stages. Multiple processing units enable the setup of multiple connections to run as a pipeline.

(1) Resource Creation

In order to be able to handle multiple stages at the same time, we use a threaded library. For each of the five steps of connection setup, *rdma_create_id*, *rdma_resolve_addr*, *rdma_resolve_route*, *rdma_create_qp*, and rdma_connect, we created a work thread for each of them, and each of them corresponds to a work queue containing the pending work requests. We designed the *rdma_pipeline_create* API, which is responsible for creating these worker threads and worker queues, as well as creating a private context and mounting these worker threads and queues to this context.

(2) Parameter Assignment

In a distributed training network scenario, the configuration of each connection may be different, it may have unique QP attributes, destination addresses, or connection properties, therefore, SECM needs

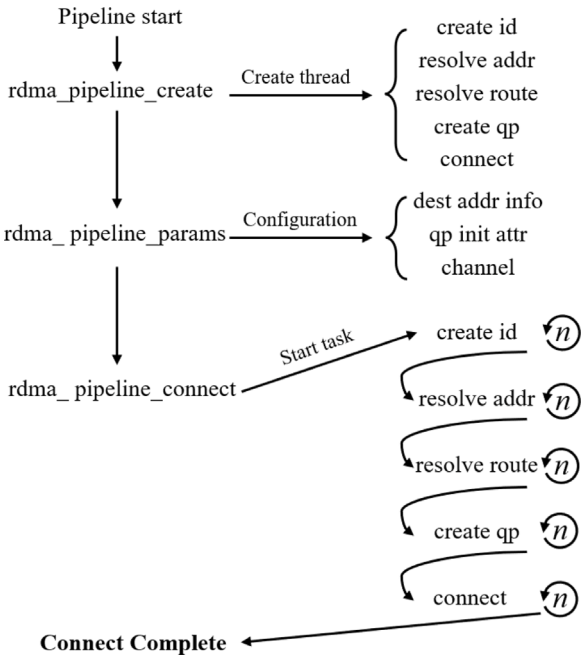


Fig. 5. High-level overview of functional interface in SECM.

to provide parameter customization as much as possible. We designed the *rdma_pipeline_params* API, which is responsible for accepting various configuration parameters passed in by the user, and detecting the legitimacy of their passing, and returning an error message to the user if an unreasonable configuration parameter is detected. The parameters that pass the detection are finally mounted on the parallel context for use by subsequent stage functions.

(3) Connection Setup

After making the preparation, the user needs to initiate the request for parallel connection setup. We designed the *rdma_pipeline_connect* API, which is responsible for initiating all the connection requests, and SECM will send all these tasks to the work queue of *rdma_create_id*, and notify the work thread of *rdma_create_id* to start processing the work tasks here by means of signaling. When the CREATE_ID phase of the first connection is processed, SECM adds the CM ID created in this phase to the work queue of RESOLVE_ADDR in the next phase and notifies the RESOLVE_ADDR thread to start processing the work request. And so on, each connection will go through the five phases of *rdma_create_id*, *rdma_resolve_addr*, *rdma_resolve_route*, *rdma_create_qp*, and rdma_connect until the callback event of the call to *rdma_connect* is returned, and then, a connection is established. Overall, the five steps work in a pipelined manner, requiring only fixed thread resources, which can achieve efficient results.

(4) Connection Disconnection

Considering that when a node dynamically exits the cluster, it needs to disconnect all previously established connections, we designed the *rdma_pipeline_disconnect* API, which is responsible for releasing all occupied resources and sending disconnect messages to the remote end to quickly and efficiently disconnect all established connections.

Event Handle: Since each worker thread uses asynchronous calls when calling the corresponding stage function, we need threads to handle these asynchronous returned results. This thread gets the asynchronous results returned by the function in real time by listening to the *rdma_get_cm_event* function. Depending on the type of event in the result, the result is returned to a different handler function, which will send the work request to the next stage of the work queue. Since the event handler function needs to handle all the event callbacks of the connection, we will dynamically adjust the number of threads here

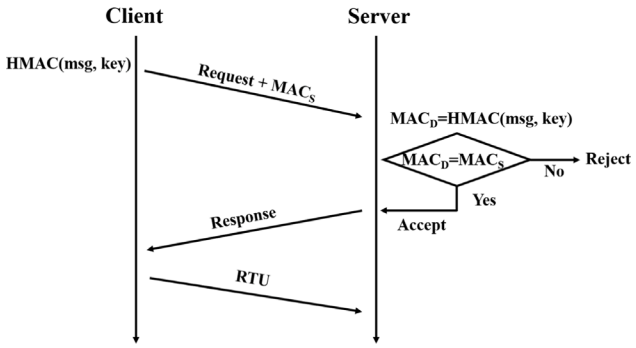


Fig. 6. Detailed process for establishing a secure connection with MAC.

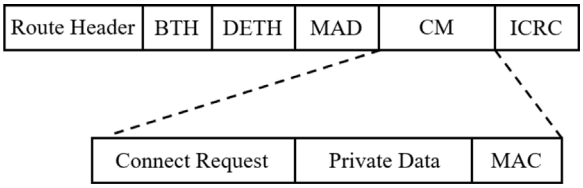


Fig. 7. SECM secure packet format.

according to the number of connections as well as the number of CPU cores to handle these callback events to speed up the connection setup.

Error Handle: Due to unreachable remote nodes, incorrect local connection configuration, etc., the connection may enter an error state, so we need to handle these corrupted and unestablished connections correctly. SECM adds an error flag bit to the connection ID, when an unrecoverable error is encountered, error flags the error state, and at the same time, interrupts the pipeline execution, and jumps to the end of the connection setup state. The user can determine whether the connection is successfully established based on this flag bit.

4.2. Security module

SECM provides security enhancements to CM. we verify the identity of the connection initiator by adding MAC information to the CM connection packet. Fig. 6 shows the detailed flow of the secure connection.

(1) Include the MAC when sending a connection request.

The MAC needs to be sent together with the message in order to play its role, so the connection initiator needs to embed the MAC information in the connection message. How to carry MAC information without destroying the original packet information becomes a tricky problem.

By analyzing the CM packet fields, SECM selects the *Private Data* field to carry the MAC information. The *Private Data* field is an opaque piece of data specified in the CM protocol, which can be user-defined. The available size of the *Private Data* field is 56 bytes, which is enough to hold the complete MAC information(see Fig. 7). At the same time, *Private Data* being a field that can be assigned a value in the user state means that the SECM does not need to make any changes to the RNIC to pass the MAC to the receiver along with the request packet.

(2) Verification of MAC at the receiver side.

When a node receives a connection request, it needs to verify the MAC in the request. SECM takes QPN, and starting PSN as the message, so it needs to take out this information from the request packet first, compose the message, and then compute it with a key to get the MAC. The node then compares the computed MAC with the MAC carried in the *Private Data*, and if the match passes, then it is delivered to the upper layer to If the match passes, the request will be delivered to the upper layer for request acceptance; otherwise, it will be regarded as an illegal connection request and discarded, and the connection information will be written to the illegal access log for subsequent review by the user.

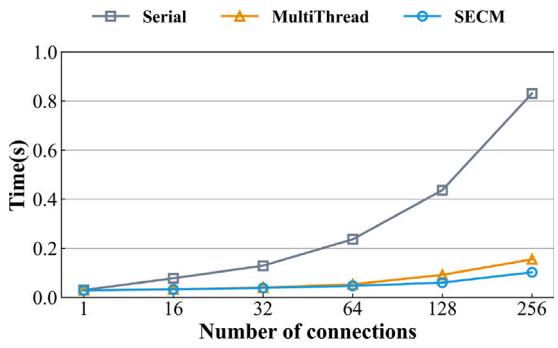


Fig. 8. A comparison of time overhead on connections.

5. Evaluation

We built a small cluster containing 10 physical machines, each equipped with a Mellanox ConnectX-6 DX [29]. The 10 physical machines are connected to a 100G switch over 100G fiber. Each host OS is Ubuntu 20.04.5 LTS with a kernel version of 5.15.0-102-generic, and the RDMA driver version is *MLNX_OFED_LINUX-5.8-3.0.7.0*, and all of them use RoCEv2 protocol.

5.1. Connection setup test

In our experiments, we tested the performance of SECM and analyzed it in comparison with two other different connection establishment methods, namely serial connection setup and multi-threaded connection setup. The main goal of our experiments is to compare the time consumption of these three methods for different number of connections. Fig. 5 shows the detailed process.

Serial connection setup is a sequential way of establishing connections one by one, i.e., one connection is established before the next connection is established. The advantage of this approach is that it is simple and intuitive, but it can lead to long waiting times in the case of a large number of connections. In contrast, multi-threaded connection setup allows for multiple connections to be established at the same time by opening a separate thread for each connection to be processed in parallel. This approach can be effective in increasing the speed of connection setup. However, it may also introduce some additional overheads such as thread management and resource contention. Especially when there is a large number of connections to be established..

In order to closely compare the performance differences of these three methods, we set different numbers of connections in our experiments and recorded the elapsed time required for each method to complete connection setup. By comparing the elapsed time under different numbers of connections, we can evaluate the performance of these three methods under different load conditions.

As can be seen in Fig. 8, the serial connection setup method shows a sharp increase in time consumption as more connections are created, especially in the case of a larger number of connections, this method may take several seconds, and has the worst performance. The multi-threaded connection setup method has a much better performance in terms of time consumption, and the SECM method has the best time consumption performance in the experimental results, with the increase of the number of connections, the time consumption curve is very smooth, and the time consumption is one order of magnitude less than that of the traditional serial method, and at the same time, it is better than the multi-threaded method, and on average, it only takes about 200us to complete the connection for each connection.

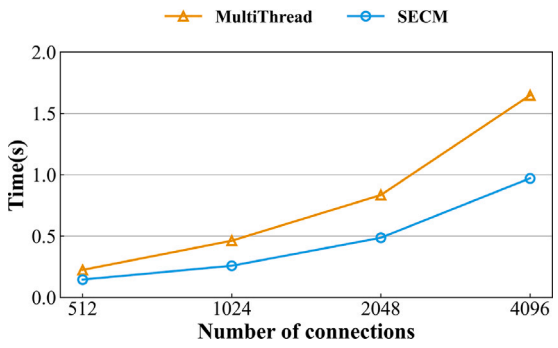


Fig. 9. A comparison of time overhead on large amount connections. As the number of connections increases to a larger number, SECM shows better performance.

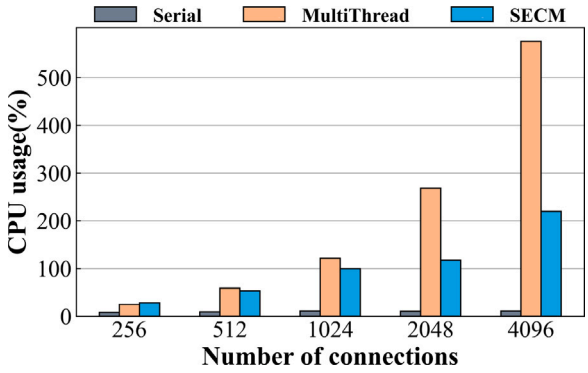


Fig. 10. A comparison of CPU usage on connections: Multi-thread takes up more CPU resources.

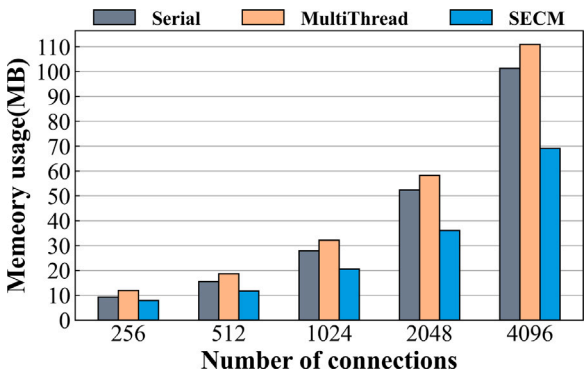


Fig. 11. A comparison of memory usage on connections: SECM allocates few threads, while Multi-thread allocates more.

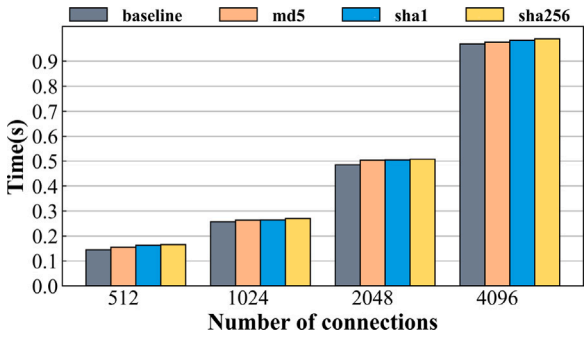


Fig. 12. Time overhead of security modules implemented based on different hash algorithms for creating different number of connections. SECM can create secure and reliable connections without significant time overheads.

Fig. 9 reflects the time-consumption of the multi-threaded approach and SECM as the number of connections increases. In the case of a sharp increase in the number of connections, the time-consumption of SECM is much better than that of the multi-threaded approach, the main reason being that the multi-threaded approach creates a large number of threads as the increase in the number of connections, which results in frequent context switching and thus longer execution times and poorer performance. SECM, on the other hand, uses only a limited number of threads and is not affected by the number of connections, resulting in excellent performance.

In addition to the time-consumption analysis, we also need to consider the resource usage. We conducted tests to compare the CPU occupancy and memory usage when using the SECM method, the serial approach and the multi-threaded approach under different numbers of connections. Through our experiments, we found that SECM has lower CPU occupancy and memory usage when dealing with a large number of connections, showing better resource utilization efficiency.

Fig. 10 shows the CPU usage. As the number of connections increases, the CPU utilization of the serial approach stays at a stable level, mainly because the serial program can only handle one connection at a time and the CPU resources consumed to create a connection are relatively fixed, so the CPU occupancy of the single-threaded serial approach stays at a stable level. On the other hand, the CPU occupancy of the multi-threaded approach increases dramatically, especially when creating connections to 4096 nodes, the multi-threaded approach's CPU occupancy is more than 40 times that of the benchmark. The main reason is that this approach needs to create a thread for each connection, which brings frequent context switching, and the switching between threads requires saving and restoring the context information of the threads, a process that requires the CPU to perform frequent switching operations and consumes a large amount of CPU resources. In contrast, the SECM method has a relatively low CPU utilization and changes slowly as the number of connections grows, e.g., only one-third of the multi-threaded method at 4096 connections.

Fig. 11 shows the memory usage. In terms of memory occupancy, SECM shows a clear advantage. As the number of connections increases, the memory footprint of all approaches starts to increase, the main reason being the need to allocate the appropriate hardware and software resources for each connection, including QP, CQ, FD, and so on. Among them, SECM has the slowest memory growth rate and the multi-threaded approach has the fastest growth rate. As SECM has optimized the performance for large-scale connections in *librdmacm* dynamic library, such as optimizing the event callback function, multiplexing the event channel to accept the callback events of multiple connections, and so on. As a result, SECM exhibits a lower memory footprint compared to the serial approach. On the other hand, the multi-threaded approach needs to allocate a certain amount of memory space for each thread to store the thread's stack space and execution context and other information, which will take up more memory resources than the serial approach when the number of threads increases.

5.2. Security connection setup test

In our security tests, we tested the impact of different HMAC algorithms on connections.SECM provides three hash functions by default, MD5, SHA, and SHA256, and we measured the time-consuming impact of these HMAC algorithms on connections.

Fig. 12, the introduction of the security module has some impact on the connection setup time, but the impact is very minor. As expected, SHA256 has the highest latency and is the most secure and expensive algorithm. The computation of HMAC introduces a latency of about 5us or so for each connection, but it provides a secure authentication mechanism that enables selective acceptance of connection requests by the passive side to prevent unauthenticated connection requests from being established.

6. Related work

DCT [30]: Currently NVIDIA added DC type of QP in mlx5 driver, DC combines the advantages of UD and RC, supports read/write unilateral semantics, and does not need to create connection explicitly at the remote end. DC maintains a pool of responders (DCRs) internally, which is able to process the request messages quickly. Since the pooled resources are limited, when there are insufficient resources in the pool, DCQP will keep switching among different remote addresses, which will cause a large number of connection messages to appear, and in extreme cases the number of data messages may be equal to the number of connection messages, which will lead to a continued degradation of application performance.

KRCORE [14] proposes a scheme to share RC and DC type QPs in the kernel state, by pooling a large number of RC and DC QPs in the kernel to provide services to the user in a virtual QP scheme, but it plunges all the paths of RDMA into the kernel, which reduces the latency of the connection setup, but increases the time consumed at the data level. The main reason is that each data sending needs to be plunged into the kernel to transform the request from the virtual QP in the user state to the physical QP shared by the kernel. Also, due to the shared QP, any wrong operation may damage the QP, which affects all the virtual QPs in the user-state based on this physical QP and may have an impact on the data transfer.

Lee et al. [31,32] suggested replacing the ICRC field with MAC to implement an authentication mechanism for RDMA packets. However, existing devices such as NICs and routers may consider the MAC information as wrong ICRC and discard the packet. This approach can lead to incompatibility of RDMA packets with existing hardware devices [20]. Therefore it is not used in the existing RDMA environment.

7. Conclusion

This paper presents SECM, a control-plane enhancement for RDMA that accelerates RDMA-CM connection setup. It provides high performance, low-overhead connection setup and adds an authentication mechanism to CM. SECM reduces the control path costs of RDMA-CM without incurring data path costs. SECM is compatible with existing RDMA hardware and software resources. Experimental results confirm the performance and security of SECM.

CRediT authorship contribution statement

Xingyu Guo: Conceptualization, Formal analysis, Investigation, Methodology, Software, Writing – original draft. **Guo Chen:** Conceptualization, Funding acquisition, Resources, Supervision, Writing – review & editing. **Xiaoning Zhan:** Data curation, Writing – original draft. **Ting Qu:** Conceptualization, Supervision. **Zhaojiao Han:** Conceptualization, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors do not have permission to share data.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under Grant 62222204 and Grant 62172148, and in part by the National Key Research and Development Program of China under Grant 2023YFB3002203, and in part by the Major special project of Changsha science and technology plan under Grant kh2401005.

References

[1] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, Ameya Bhagat, Gowri Bhaskara, Tanya Brokhman, Lei Cao, Ahmad Cheema, et al., Empowering azure storage with RDMA, in: 20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 23, 2023, pp. 49–67.

[2] Shin-Yeh Tsai, Yiying Zhang, Lite kernel rdma support for datacenter applications, in: Proceedings of the 26th Symposium on Operating Systems Principles, 2017, pp. 306–324.

[3] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, Marina Lipshteyn, RDMA over commodity ethernet at scale, in: Proceedings of the 2016 ACM SIGCOMM Conference, 2016.

[4] Haoran Zhang, Adney Cardoza, PeterBaile Chen, Sebastian Angel, Vincent Liu, Fault-tolerant and transactional stateful serverless workflows, in: Operating Systems Design and Implementation,Operating Systems Design and Implementation, 2020.

[5] Zhipeng Jia, Emmett Witchel, Boki: Stateful Serverless Computing with Shared Logs.

[6] NVIDIA, RDMA CM API, 2013, https://docs.nvidia.com/networking/display/rdmaawareprogrammingv17/rdma_cm+api.

[7] InfiniBand Trade Association, Infiniband architecture specification release 1.2.1 annex a16: RoCE, 2010.

[8] InfiniBand Trade Association, Infiniband architecture specification release 1.2.1 annex a17: RoCEv2, 2014.

[9] Bongjae Kim, TSL-RDMA: Thread-safe and lightweight RDMA API for InfiniBand-based cluster computing systems, Int. Inf. Inst. (Tokyo). Inf. 19 (11A) (2016) 5281.

[10] Xingda Wei, Rong Chen, Haibo Chen, Binyu Zang, Xstore: Fast rdma-based ordered key-value store using remote learned cache, ACM Trans. Storage (TOS) 17 (3) (2021) 1–32.

[11] Xiaoyi Lu, Dipti Shankar, Dhabaleswar K. Panda, Scalable and distributed key-value store-based data management using RDMA-memcached, IEEE Data Eng. Bull. 40 (1) (2017) 50–61.

[12] Xingda Wei, Rong Chen, Haibo Chen, Fast RDMA-based ordered key-value store using remote learned cache, in: Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation, 2020, pp. 117–135.

[13] Teng Ma, Tao Ma, Zhuo Song, Jingxuan Li, Huaixin Chang, Kang Chen, Hai Jiang, Yongwei Wu, X-RDMA: Effective RDMA middleware in large-scale production environments, in: 2019 IEEE International Conference on Cluster Computing, CLUSTER, 2019.

[14] Xingda Wei, Fangming Lu, Rong Chen, Haibo Chen, KRCORE: A microsecond-scale RDMA control plane for elastic computing, in: 2022 USENIX Annual Technical Conference, USENIX ATC 22, 2022, pp. 121–136.

[15] Xizheng Wang, Guo Chen, Xijin Yin, Huichen Dai, Bojie Li, Binzhang Fu, Kun Tan, StaR: Breaking the scalability limit for RDMA, in: 2021 IEEE 29th International Conference on Network Protocols, ICNP, IEEE, 2021, pp. 1–11.

[16] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al., HPCC: High precision congestion control, in: Proceedings of the ACM Special Interest Group on Data Communication, 2019, pp. 44–58.

[17] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, Marina Lipshteyn, RDMA over commodity ethernet at scale, in: Proceedings of the 2016 ACM SIGCOMM Conference, 2016, pp. 202–215.

[18] Tarick Bedeir, Building an RDMA-capable application with IB verbs, in: Technical report, HPC Advisory Council, 2010.

[19] Benjamin Rothenberger, Konstantin Taranov, Adrian Perrig, Torsten Hoefler, ReDMArk: Bypassing RDMA security mechanisms, in: USENIX Security Symposium, 2021, pp. 4277–4292.

[20] Konstantin Taranov, Benjamin Rothenberger, Adrian Perrig, Torsten Hoefler, sRDMA: efficient NIC-based authentication and encryption for remote direct memory access, in: Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference, 2020, pp. 691–704.

[21] Trisha Datta, Nick Feamster, Jennifer Rexford, Liang Wang, spine: Surveillance protection in the network elements, in: 9th USENIX Workshop on Free and Open Communications on the Internet, FOCI 19, 2019.

[22] Roland Meier, Petar Tsankov, Vincent Lenders, Laurent Vanbever, Martin Vechev, NetHide: Secure and practical network topology obfuscation, in: 27th USENIX Security Symposium, USENIX Security 18, 2018, pp. 693–709.

[23] Jiarong Xing, Wenqing Wu, Ang Chen, Architecting programmable data plane defenses into the network with FastFlex, in: Proceedings of the 18th ACM Workshop on Hot Topics in Networks, 2019, pp. 161–169.

[24] Jiarong Xing, Kuo-Feng Hsu, Yiming Qiu, Ziyang Yang, Hongyi Liu, Ang Chen, Bedrock: Programmable network support for secure RDMA systems, in: 31st USENIX Security Symposium, USENIX Security 22, 2022, pp. 2585–2600.

[25] InfiniBand Trade Association, Infiniband architecture specification release 1.2.1, 2014.

[26] Mihir Bellare, Joe Kilian, Phillip Rogaway, The security of the cipher block chaining message authentication code, J. Comput. System Sci. 61 (3) (2000) 362–399.

[27] James M. Turner, The keyed-hash message authentication code (hmac), Federal Inf. Process. Stand. Publ. 198 (1) (2008) 1–13.
[28] NVIDIA, MLNX_OFED_LINUX-5.8-3.0.7.0., 2023, https://network.nvidia.com/products/infiniband-drivers/linux/mlnx_ofed/. (Accessed 9 July 2023).
[29] NVIDIA, Mellanox ConnectX-6 DX, 2015, https://www.nvidia.com/content/dam/en-zz/Solutions/networking/ethernet-adapters/connectX-6-dx-datasheet.pdf. (Accessed on 2015).
[30] OFED, Dynamically connected transport, 2013, https://www.openfabrics.org/images/eventpresos/workshops2014/DevWorkshop/presos/Monday/pdf/05_DC_Verbs.pdf.
[31] Manhee Lee, Security Enhancement in Infiniband Architecture, IEEE, 2005.
[32] Manhee Lee, Eun Jung Kim, A comprehensive framework for enhancing security in infiniband architecture, IEEE Trans. Parallel Distrib. Syst. 18 (10) (2007) 1393–1406.

Xingyu Guo received the B.S. degree from Hunan University, in 2022. He is currently pursuing the master's degree with Hunan University, China. His research interests include computer networking.

Guo Chen received the Ph.D. degree from Tsinghua University in 2016. He was a Researcher with Microsoft Research Asia from 2016 to 2018. He is currently a Professor with Hunan University. His current research interests include networked systems and with a special focus on data center networking. He has published more than 40 papers including those on top conferences/journals like NSDI, USENIX ATC, INFCOM, ToN, JSAC. His research has been adopted in Huawei Kunpeng CPU, Tencent switch, Tencent CDN and Baidu wireless search, etc.

Xiaoning Zhan received the B.S. degree from Hunan University, in 2020. He is currently pursuing the master's degree with Hunan University, China. His research interests include computer networking.