JANUS: Enabling Expressive and Efficient ACLs in High-speed RDMA Clouds

Ziteng Chen[1,*], Menghao Zhang[2,*], Jiahao Cao[3,4], Xuzheng Chen[5], Qiyang Peng[2], Shicheng Wang[6], Guanyu Li[6], Mingwei Xu[4,3,1]
[1]Southeast University [2]Beihang University [3]Tsinghua University [4]Quan Cheng Laboratory
[5]Zhejiang University [6]Unaffiliated

***Abstract*—RDMA clouds are becoming prevalent, and ACLs are critical to regulate unauthorized network accesses of RDMA applications, services, and tenants. However, the unique QP semantics and high-speed transmission characteristics of RDMA prevent existing ACL expressions and enforcement mechanisms from comprehensively and efficiently governing RDMA traffic in a user-friendly manner. In this paper, we present JANUS, a tailored ACL system for RDMA clouds. JANUS designs specialized ACL expressions with QP semantics to identify RDMA connections, and provides a high-level policy language for expressing sophisticated ACL intents to govern RDMA traffic. JANUS further leverages DPUs with traffic-aware and architecture-specific optimizations to enforce ACL policies, enabling line-rate RDMA inspection and robust policy updates. We implement an open-source prototype of JANUS with NVIDIA BlueField-3 DPUs. Experiments demonstrate that JANUS provides sufficient expressivity for governing unauthorized RDMA accesses, and achieves line-rate throughput in a 200Gbps real-world RDMA testbed with $<$5μs latency.**

I. INTRODUCTION

Remote Direct Memory Access (RDMA) is getting increasing deployment in leading cloud companies for distributed applications, such as large language model (LLM) training, cloud storage, and remote procedure calls. This adoption is driven by RDMA's ability to deliver line-rate throughput and ultra-low latency (e.g., $\geq$100Gbps and $\leq$5μs), offering significant performance improvements over traditional TCP/IP networks [1], [2], [3], [4], [5], [6], [7], [8], [9]. Recently, a notable trend of RDMA is an expansion to cloud environments, where numerous users can access RDMA services to enjoy high-performance networking [10], [11], [12], [13].

Despite their promising network performance, RDMA-based clouds have recently attracted increasing security concerns. Several studies revealed RDMA-specific vulnerabilities and attacks, such as connection hijack [14], unauthorized memory access [14], resource exhaustion [12], [13], and bandwidth degradation [15], [16], posing serious threats to RDMA clouds. Network access control lists (ACLs) serve as a widely used mechanism to govern communications in clouds, as adopted by major providers such as Amazon [17], Google Cloud [18], Azure [19] and IBM Cloud [20]. ACLs explicitly specify `allow` or `deny` rules based on specific attributes, such as IP addresses and ports. Following a lightweight and stateless inspection philosophy, the header of each packet is examined against the ACL rules [21], [22], and only traffic that conforms to the defined policies is permitted, allowing operators to prevent unauthorized accesses to applications, services, and tenants. However, when introducing ACLs to RDMA clouds, the distinct characteristics of RDMA prevent existing ACLs [23], [24], [25], [26], [11], [27], [10], [28] from achieving the above objective.

To effectively govern the unique semantics and communication patterns of RDMA cloud traffic, existing ACLs fail to provide sufficient granularity and expressiveness. Traditional ACL expressions are mainly represented in a five-tuple format [23], [24], [25], but they fail to regulate the RDMA traffic due to its fundamentally different semantics from TCP/IP. Specifically, RDMA involves more sophisticated state management and finer-grained communicating types based on *queue pairs* (QPs), such as QP creation and destruction along with diverse QP operations on remote memory region (MR). Besides, RDMA traffic is disaggregated into control path for QP lifecycle maintenance, and data path for application data exchange. Each of them requires independent controls over distinct packet metadata and QP behaviors. Although recent studies [26], [11] attempt to impose control over certain QP states, their governance fails to cover the intricate QP semantics originated from different traffic paths.

Furthermore, existing ACL enforcement mechanisms are not well equipped to efficiently handle the full inspection for RDMA traffic in clouds. Traditional end-host ACLs, such as iptables [23] and Open vSwitch [24], are enforced in the OS kernel. However, RDMA data path traffic bypasses the kernel, preventing them from capturing the data path packets. Although microkernel-based RDMA solutions (e.g., Snap [27] and FreeFlow [10]) can govern RDMA traffic at a software shim layer, they incur significant CPU overhead and impose non-negligible performance penalty for traffic inspection. In-network hardware enforcement schemes (e.g., Bedrock [28]) can achieve line-rate ACL throughput. However, when inspecting intra-host traffic, they must redirect it to in-network ACL devices, introducing additional latency to RDMA communication.

*Equal contribution.

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.240721
www.ndss-symposium.org

We believe an ideal ACL paradigm for RDMA clouds should satisfy the following three properties: ① *Coverage*, which requires ACL expressions to capture fine-grained QP semantics, and complete enforcement mechanisms with full inspection for RDMA traffic arising from both intra-/inter-host and control/data paths. ② *Efficiency*, which needs to perform per-packet inspection without compromising RDMA's line-rate throughput and ultra-low latency, while minimizing the system overhead. ③ *Usability*, which demands a transparent and user-friendly approach to relax the policy definition and maintenance burden for diverse and elaborate QP behaviors.

In this paper, we present JANUS, a tailored and user-friendly ACL system for RDMA clouds that combines comprehensive RDMA-semantics expressions with efficient policy enforcement. We point out traditional five-tuple-based ACL expressions are insufficient to represent the identity of an RDMA connection, i.e., *request entities* for denoting the traffic source and destination, as well as *QP behaviors* indicating the associated QP operations, thereby requiring a dedicated expression to control the QP-semantics permission. JANUS identifies an RDMA connection by extracting the essential packet fields related to entity information and QP behaviors from complex and varying metadata in control and data path traffic. To simplify policy specification, JANUS introduces a high-level policy language that enables operators to express sophisticated ACL intents for controlling RDMA-native traffic and preventing unauthorized accesses.

JANUS uses a data processing unit (DPU) to efficiently enforce ACL policies at end-hosts. Located in the critical path, DPU provides comprehensive coverage on RDMA traffic, and programmability to offload the ACL enforcer with transparency and compatibility for RDMA applications. However, a naive offloading without careful considerations of RDMA traffic patterns, ACL enforcer characteristics and DPU architecture can lead to poor performance. Therefore, we carefully design DPU-specialized optimizations to enable efficient packet inspection and policy updates for high-speed RDMA traffic. For efficient usage of DPU's hierarchical memory subsystem, we design a cache-friendly data structure for the ACL ruleset, and develop a feasible usage strategy for heterogeneous DRAMs to significantly optimize the memory accesses in high frequency. To fully leverage and orchestrate DPU's high parallelism, we develop a dynamic and load-aware packet steering mechanism, along with batched processing of doorbell rings to remarkably enhance inspection efficiency. To ensure robustness during updates, we further optimize the coordination between the DPU control plane and data plane to deliver consistent, non-blocking and timely policy updates.

We implement a JANUS prototype on NVIDIA BlueField-3 DPUs, and make the source code publicly available at Github [29]. We conduct extensive experiments to evaluate the expressiveness and enforcement performance of JANUS. The case studies show that JANUS can effectively define sophisticated ACL policies in a QP granularity to govern unauthorized accesses proposed in existing works [14], [15], [12], [16] with minimal efforts. Specifically, JANUS achieves line-rate throughput and 4.33μs p99 latency with less than 4% additional resource overhead on DPUs in a 200Gbps real-world RDMA testbed. Moreover, JANUS significantly outperforms existing ACL approaches, achieving up to 8.63× higher throughput than the software-based FreeFlow [10] and up to 63.1% lower latency than the in-network Bedrock [28]. Ablation studies validate the effectiveness of our specialized optimizations in improving ACL performance, e.g., the optimized scheduling on heterogeneous DRAMs can increase the ACL throughput by up to 7.32×.

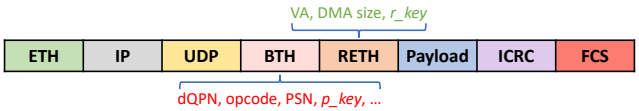


Fig. 1: RoCEv2 packet header.

To the best of our knowledge, JANUS is the first work to explore tailored ACL expressions for fine-grained QP regulation, coupled with an efficient DPU-based enforcement for RDMA clouds. The main contributions of this work include:

- We identify the limitations of traditional schemes, and design specialized ACL expressions, accompanied by a high-level policy language for RDMA semantics (§IV).
- We design specialized optimizations based on RDMA traffic patterns and DPU characteristics to achieve line-rate throughput and ultra-low latency for ACL (§V).
- We open-source and evaluate JANUS, with results demonstrating its strong expressivity and superior ACL performance in a real-world RDMA testbed (§VI).

II. BACKGROUND AND MOTIVATION

A. RDMA Preliminary

RDMA bypasses the TCP/IP stack in the operating systems and offloads data transmission to RDMA network interface cards (RNICs), enabling clients to directly access the remote memory of servers without CPU involvement, which delivers high-throughput and low-latency networking performance. RDMA traffic travels through two main paths. The control path manages QP operations such as creation, destruction, state transitions, and metadata exchange. The data path uses kernel-bypass techniques to transmit application data with one-sided (`READ`/`WRITE`), two-sided (`SEND`/`RECV`), and other atomic operations (`FAA`/`CAS`) on the target MR associated with QPs, which greatly improves data transmission performance.

Figure 1 illustrates the packet format of RDMA over Converged Ethernet version 2 (RoCEv2) [30]. In addition to basic fields of ETH, IP, UDP, payload and checksums, RoCEv2 traffic includes QP-semantics fields in BTH and RETH headers, including `dQPN` denoting the destination QP number (QPN), `VA` representing the target virtual memory address, and `opcode` for memory access type. Similar QP-semantics metadata apply to other RDMA implementations [31], [32], [33], [34], [35], despite their varying packet formats.

B. ACLs for RDMA

Why ACLs are necessary in RDMA clouds? As a fundamental network service and control mechanism in leading

production clouds [17], [18], [19], [20], [36], ACLs play a lightweight and effective role in managing the network access permissions. By defining policies for specific entities with `allow` or `deny` permissions, ACLs determine whether each RDMA packet can be transmitted or received, thereby regulating communications among applications, services and tenants [21], [22]. Compared to physical region partition for network tenants [37] and Virtual Extensible LAN (VXLAN) [38] for layer-2 tenant isolation, ACLs offer more precise, flexible and fine-grained network control based on the upper-layer metadata of each packet. ACLs can not only govern accesses on internal sensitive services of a single tenant, but also block potential communication between multiple tenants.

As for security aspect, recent studies revealed several distinct RDMA-specific vulnerabilities and attacks. For example, ReDMArk [14] unveiled the predictability of QP metadata (e.g., QPN and rkey), allowing attacker tenants to hijack a connection and conduct unauthorized accesses of benign tenants. Husky [12] and Harmonic [13] found a resource exhaustion attack where attackers can launch exhaustive atomic operations to consume RNIC computing resources. NeVerMore [15] and LoRDMA [16] pointed out that attackers can launch unauthorized QP disconnect requests at the control path, or periodical bursts to degrade the victim throughput at the data path. Fortunately, ACLs also serve as one of the feasible approaches to defend against the potential threats. They allow operators to define tailored policies that block potential malicious requests from adversaries, preventing benign users from being exposed to such attacks. With their capabilities for tenant isolation and security enhancement, ACLs are therefore essential in multi-tenant RDMA clouds.sary in multi-tenant RDMA clouds.

Why TCP/IP ACLs are insufficient in RDMA clouds? Although ACLs show promise in regulating unauthorized accesses, current expression and enforcement designed for TCP/IP networks fall short in addressing the unique requirements in RDMA clouds. Firstly, networking equipment typically adopts ACL expressions based on five-tuple format, i.e., <`sip`, `sport`, `dip`, `dport`, `protocol`>, which is tailored to TCP/IP traffic where each connection is identified by a five-tuple [39]. However, this format is coarse-grained and fails to govern complete RDMA connections, whose fundamental unit is the Queue Pair (QP) and its associated semantics. For example, consider a sender establishing an RDMA connection with a receiver. For security reasons, the operator wishes to restrict the `write` accesses to the receiver's memory region while still allowing `read` operations. If the ACL rule is defined using a five-tuple expression without awareness of QP semantics, it will indiscriminately block all outgoing RDMA traffic matching the rule—both target `writes` and innocent `reads`—since they share the same five-tuple metadata. In this example, five-tuple-based ACLs fail to support fine-grained regulation over specific memory access behaviors of an individual QP in RDMA scenarios. While certain programmable devices may provide field-extension capabilities, there has been no thorough exploration of complete, feasible and efficient ACL expressions tailored to RDMA environments. Therefore, simple extensions cannot effectively mitigate existing RDMA-specific threats [14], [15], [12], [16] without concise yet expressive policies that capture QP semantics.

TABLE I: Property analysis for existing ACLs. ●, ◑, and ○ denote "yes", "partially yes", and "no", respectively.

ACL system \ Property	**Coverage**	**Efficiency**	**Usability**
native RDMA [44], [45]	◑	N/A	○
iptables [23], open vSwitch [46], [24]	○	○	◑
VFP [47], AccelNet [48], Sirius [49]	○	◑	○
LITE [26], MasQ [11]	○	N/A	◑
Snap [27], FreeFlow [10]	◑	○	◑
Bedrock [28]	○	◑	◑

Secondly, current ACL enforcers widely used in production-level TCP/IP clouds are mainly designed to satisfy the TCP/IP performance with $\leq$40Gbps bandwidth and $\geq$500μs delay [40], [41]. However, they struggle to keep up with the stringent performance demands of high-speed RDMA clouds with $\geq$100Gbps throughput and $\leq$5μs latency. Moreover, many of these enforcers operate at the OS-kernel level to regulate traffic [1], [2], [42], [43]. This kernel-based enforcement model cannot capture kernel-bypass RDMA traffic on the data path, thereby failing to provide the per-packet inspection that ACLs are intended to deliver.

Expected properties of RDMA ACLs. To achieve access control on RDMA communication, we believe ideal RDMA ACLs should offer delicate expression and enforcement capabilities, guided by the following three key principles:

Coverage: RDMA ACLs need to fully cover RDMA semantics and traffic. An RDMA connection involves the request entities and associated QP behaviors, necessitating finer-grained ACL expressions to capture its identity and provide complete coverage on the unique QP semantics. In addition, the ACL enforcement should cover all RDMA packets across the control and data paths, ensuring no traffic is overlooked due to incomplete packet capture and malicious circumvention. Especially, we note only regulating the QP states (e.g., creation and destruction) instead of full coverage of RDMA traffic is not enough, as it falls short of meeting ACL's fundamental objective of per-packet inspection [21], [22]. As a result, adversaries can exploit this inspection gap to generate packet-level misbehavior that passes QP creation and destruction regulation, for example, fabricating arbitrary packet header fields to hijack a victim connection [14], [15].

Efficiency: RDMA ACLs should offer high efficiency. ACL enforcer inevitably introduces additional overhead due to per-packet inspection and frequent policy updates. As RDMA clouds are equipped with bandwidth over 100Gbps and latency within 5μs, efficient ACL enforcement must impose minimal overhead to keep pace with high-speed RDMA transmission. Substantial throughput or latency degradation, along with excessive resource use by ACLs, are undesirable. This is particularly critical because RDMA tenants often run performance-sensitive applications (e.g., LLM training, cloud storage) that leverage RDMA to accelerate distributed computing. Any substantial overhead or resource usage introduced by ACL

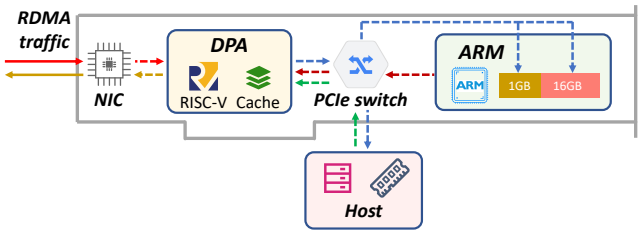


Fig. 2: NVIDIA BlueField-3 DPU architecture.

enforcement can significantly degrade the performance of both RDMA transmission and upper-layer applications, which is unacceptable in such environments.

Usability: RDMA ACLs must satisfy ease of use. An ACL enforcer should be transparent to developers and tenants, avoiding interference with typical development of RDMA applications. Given RDMA's more intricate semantics, moreover, the ACL system should be user-friendly for operators, relaxing the difficulties of policy formulation on the fine-grained QP-semantics traffic, and avoiding being overwhelmed by excessive policy expressions.

Existing ACLs for RDMA. Table I compares the properties of existing ACLs in RDMA scenarios. Native RDMA has simple access control supports, e.g., restricting MR access type in the control path [44], and revoking access privileges with memory window in the data path [45]. However, they only impose limited per-QP control over a few memory access permissions, and also fail to support per-packet inspection and zero-downtime policy updates. Iptables [23] and Open vSwitch [46], [24] are incapable of posing permissions on QP-semantics operations, and their kernel-based enforcement cannot provide efficient ACL inspection for kernel-bypass and high-speed RDMA traffic. VFP [47], AccelNet [48] and Sirius [49] allow offloading ACLs to smartNICs, but only support traditional TCP/IP expressions without consideration of the unique QP semantics in RDMA. LITE [26] and MasQ [11] adopt per-QP regulation to control whether a bound QP can be established in the kernel driver. Once permitted or denied, subsequent packets associated with the corresponding QP are no longer inspected. However, the lack of per-packet inspection violates the fundamental design philosophy of ACLs [21], [22] and leaves the door open for attackers to potentially circumvent the initial regulation and carry out malicious activities after the QP has been established. Snap [27] and FreeFlow [10] capture RDMA traffic at the software shim layer, but they consume substantial host resources and result in poor ACL performance. Bedrock [28] deploys ACLs in programmable switches to govern data path traffic, but does not account for control path traffic and introduces extra latency by "pulling" intra-host traffic to switches for inspection. In addition, all of them lack a user-friendly interface to relax the formulating burden of sophisticated ACL policies for diverse QP behaviors in RDMA clouds.

C. Data Processing Unit

DPU is a programmable NIC with offloading capabilities for network functions. Without loss of generality, we take

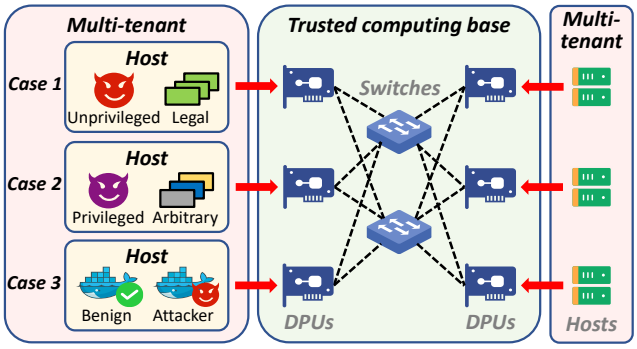


Fig. 3: JANUS threat model.

NVIDIA BlueField-3 DPU as an example to introduce the DPU architecture as shown in Figure 2. It consists of an on-path NIC core and a data-path accelerator (DPA), an off-path ARM CPU, a PCIe switch, and memory components. For compute resources, the DPA is a RISC-V processor offering 190 out of 256 user-available threads, each running at 1.8GHz computing power, which is suitable to accelerate parallelizable network functions. Regarding memory resources, the DPA features a hierarchical memory subsystem, including local L1, L2 and L3 caches, and 1GB exclusive memory & 16GB shareable memory at ARM, as well as additional host memory accessible via PCIe switch. Each tier in this hierarchy exhibits different access performance.

The reasons why we choose DPU for ACL enforcement are three-fold. Firstly, DPU locates in the critical traffic path, which not only incurs fewer hops for lower packet processing latency, but also inherently covers all RDMA packets from different dimensions, especially 1) kernel-bypassing traffic missed by traditional kernel-based ACLs for TCP/IP [23], [24]; 2) intra-host traffic that would require triangle routing in switch-based ACLs [28]. Secondly, ACL enforcers can be easily parallelized into independent and stateless packet inspection, and DPU has the high-parallelism capability to deliver efficient ACL performance. Finally, as an enhanced RNIC, DPU supports typical RDMA development with verbs, while the offloaded network functions can run agnostic to upper-layer applications without additional modification. These three reasons make DPU a feasible ACL vantage point with broad coverage, high efficiency and strong usability.

III. JANUS OVERVIEW

Threat model. As shown in Figure 3, we assume that switches and DPUs are trusted in a multi-tenant RDMA cloud. The ACL system aims to enforce packet-level RDMA access control in a stateless and high-performance manner, with the goal of preventing unauthorized access by allowing or dropping packets at the DPU. We consider attackers capable of crafting malicious RDMA packets or leveraging existing tools [14], [15], [12] to initiate RDMA connections. In a multi-tenant RDMA cloud, we consider three spatial attacker distributions following existing studies [28], [14]: (1) an unprivileged attacker with exclusive control of a host, capable of sending syntactically legal packets such as frequent QP creation and destruction; (2) a privileged attacker with root access on an

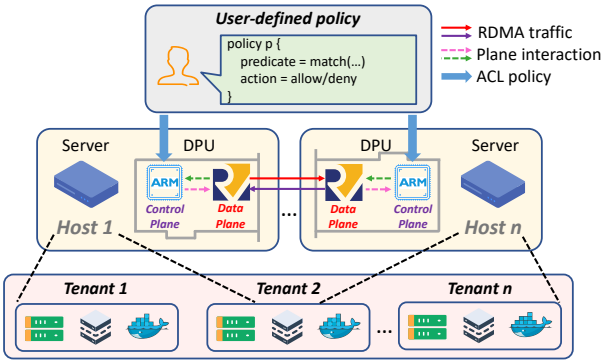


Fig. 4: JANUS overview.

isolated host, able to forge arbitrary RDMA packet metadata and inject fraudulent traffic, including forged congestion signals; and (3) an unprivileged attacker co-located with benign tenants, attempting to interfere with RDMA transmissions by exploiting shared host resources.

System overview. Figure 4 presents the JANUS overview, which includes a centralized controller and multiple distributed end-hosts, each equipped with a DPU. Similar to PortCatcher [50], JANUS works in a user-defined manner, allowing the trusted cloud operator to use JANUS expressions and policy language to define ACL policies for governing RDMA communications. The controller interprets the policies into specific rules, and pushes them to the target DPUs to enforce the policies. Each DPU runs the ACL enforcer, where the control plane governs the policy management and updates, while the data plane performs stateless and high-performance ACL inspection in a per-packet manner. For each sent and received RDMA packet, the DPU parses the header metadata and performs lookups to determine whether the packet should be allowed or rejected. If allowed, the packet proceeds with normal operations on the target MR; otherwise, it is discarded.

IV. JANUS EXPRESSION

This section unveils the shortcomings of applying five-tuple ACL expressions to RDMA scenarios. Based on the analysis of RDMA semantics, we propose RDMA-specific ACL expressions with necessary packet fields, and a high-level policy language to satisfy coverage and usability properties.

A. Why JANUS Needs New Expressions?

Why traditional expressions fail? Traditional ACL expressions typically use a five-tuple format designed for TCP/IP networks. However, due to limited expressivity, these expressions fall short of supporting comprehensive and fine-grained regulation for RDMA traffic. The fundamental limitation stems from RDMA's unique QP semantics. Different from TCP/IP using five-tuple to describe a connection, the basic transmission unit of RDMA is QP, i.e., *how a source QP at sender operates the target remote MR of a destination QP at receiver*. For example, data path packets are characterized by a sender QP's reads or writes towards a target MR of a receiver QP, whereas control path packets are specified by QP initialization and state management between a pair of sender and receiver.

TABLE II: Packet fields in JANUS expressions.

Entity fields		**QP-semantics fields**	
RoCEv2	Infiniband	Control path (CM)	Data path
`sIP`	`sLID`	`type`	`dQPN`
`dIP`	`dLID`	`lQPN`	`opcode`
`sport`	`sGID`	`dQPN`	`VA`
`dport`	`dGID`		

Therefore, the identity of an RDMA connection should be represented by *request entities* to indicate request initiator and responder, as well as associated *QP behaviors* defining the QP-relevant semantics. Due to the lack of QP expressivity, five-tuple-based expressions in TCP/IP scenario are insufficient for specifying fine-grained policies for diverse QP behaviors in RDMA traffic, thereby failing to impose access control over a complete RDMA connection.

Required expressivity and design challenges. Analogous to how a five-tuple identifies a TCP/IP connection, we point out that an ideal ACL expression should also be capable of identifying an RDMA connection. For one thing, JANUS expressions should cover the entities of request initiator and responder to localize the source and destination pair. For another, JANUS expressions are required to encompass the QP semantics—identifying what operations and MRs are performed by which QPs—enabling fine-grained access control over QP behaviors.

We encounter two primary challenges when satisfying the above expressivity requirements. The first challenge is to justify which RDMA packet fields should be included in JANUS expressions. Although an RDMA packet header contains many metadata serving different purposes and functions, only those fields representing the identity of an RDMA connection are essential for ACL governance, while the remaining fields fall outside the scope. This is similar to TCP/IP scenario, because many application-layer fields in TCP/IP packets, such as sequence number, TTL and TOS, do not denote the five-tuple identity of a TCP/IP connection, such that they are excluded from TCP/IP ACL expressions. Therefore, we need to justify a minimal yet complete set of header fields capturing the RDMA connection identity to be included in JANUS expressions.

The second challenge is to relax the usage of JANUS expressions. Given the complexity of QP semantics, it is non-trivial to define and delegate sophisticated intents with multiple policies to govern diverse and fine-grained QP behaviors across numerous applications, services and tenants in a large-scale RDMA cloud. Consequently, JANUS expressions should offer a user-friendly interface to lower the barrier for ACL policy definition.

B. Specialized Expression for RDMA

To address the first challenge, we thoroughly analyze the characteristics and justify packet metadata of RDMA traffic to determine the minimal yet complete header fields in JANUS expressions. For the second challenge, we develop a high-level policy language that simplifies policy formulation to achieve fine-grained access control for RDMA clouds.

1) Which Fields are Necessary?:

Predicate: $P ::= \texttt{match}(f \circ v) \mid \texttt{match}(f \in S) \mid \texttt{match}(f \in [v_1, v_2]) \mid P \ \& \ P \mid P \mid P \mid !P$

Action: $A ::= \texttt{allow} \mid \texttt{deny}$

Policy: $C ::= \texttt{policy}\ id\ \{\ \texttt{predicate} = P\ ;\ \texttt{action} = A\ \}$

Apply: $S ::= \texttt{apply}(p_1\ [,\ p_2,\ \cdots])$

Fig. 5: Language syntax of JANUS.

Control path traffic. RDMA control path manages the QP states, necessitating JANUS expressions to involve the packet fields related to specific QP connect and disconnect requests from which pair of nodes. Firstly, the entity fields denoting the source and destination information, such as IP addresses + ports in RoCEv2, and GIDs/LIDs in Infiniband, are necessary to express the entities of requester and responder. Secondly, the QP-semantics fields capturing the target QP and associated request types, such as `QPN` for indexing a QP, `(Dis)ConnectRequest` & `(Dis)ConnectReply` for QP management in CM protocol [51], denote the basic QP state management. These two kinds of fields together represent and constitute the identity of an RDMA connection from the control path.

We also note control path packets contain other fields irrelevant to its identity, such as `srq` to mark shared receive queues, initial packet sequence numbers (PSNs) for in-order transmission during QP creation. However, these metadata belong to application-layer information tightly coupled with RDMA applications, and they do not express the identity of RDMA connection. Therefore, they fall outside the scope of JANUS governance, which is similar to five-tuple expressions excluding the application-layer fields in TCP/IP packets.

Data path traffic. RDMA data path handles application data transmission, with packets specifying how a sender QP operates on a target MR address of receiver QP at the remote side. Similar to control path packets, the entity fields denoting the request source and destination are crucial to represent the entity information of a data path connection. Meanwhile, the QP-semantics fields, including `dQPN`, `opcode` and `VA` for memory address, denote how a particular QP operates on a specific MR address, which signify the QP behaviors of remote memory accesses. Other header fields, such as `rkey` for memory authentication, DMA size for correctness, PSN for in-order transmission, and checksums for data integrity, are mainly for application-level availability and reliability, rather than expressing the identity of RDMA connections, such that they are not incorporated in JANUS.

Required header fields. Based on traffic characteristics and semantics analysis, Table II summarizes the minimum yet complete set of packet header fields to express the identity of RDMA connections arising from control and data paths, respectively. They are classified into entity fields and QP-semantics fields, allowing JANUS to describe an RDMA connection and define policies for fine-grained permission regulation. We can prove by elimination that these packet fields are the minimum necessary set to identify an RDMA connection. For example, if we exclude `dQPN`, the expressions fail to index and localize a particular QP, resulting in loss of QP semantics. In Infiniband scenario, if `GID` is not included, the expressions fall short of denoting the request sender and receiver, leading to entity metadata loss. Similar analysis also applies for other packet fields in Table II.

2) High-level Policy Language:

The JANUS policy language. To facilitate the regulation and assignment of intricate ACL policies, we refer to existing efforts developing domain specific language to govern or secure particular scenarios [25], [52], [53], and propose the language syntax based on the included header fields. As shown in Figure 5, **`Predicate`** describes the logical conditions that determine whether an RDMA packet satisfies a combination of matching rules, including exact match of a specific value or membership checking, as well as range match within a given numerical range. Operators can involve multiple packet fields to be matched simultaneously, including entity fields and QP-semantics fields in the control and data paths. **`Action`** specifies the access control with `allow` and `deny` permissions. **`Policy`** defines the composition of a specific ACL policy, i.e., to-be-matched packet fields denoted by a target `predicate` and associated `action`, to determine the permission of matched condition. **`Apply`** activates the policy combination made by operators. In general, JANUS language largely adopts the similar design principles as existing efforts, but incrementally introduces a critical contribution that several RDMA-specific fields (in Table II) can be involved in a predicate. For example, JANUS empowers operators to regulate Infiniband networks with `sGID` and `dGID` in the entity fields, as well as comprehensive QP behaviors using QP-semantics fields, distinguishing JANUS from current expressions and languages that are mainly designed for TCP/IP networks without supports of RDMA semantics.

An example in JANUS language. Operators at the centralized controller can define and delegate their ACL intents in JANUS language to govern accesses of RDMA applications, services and tenants in a user-defined manner. Figure 6 illustrates an example to control RoCEv2 communication for a tenant using JANUS language. Suppose the operators only allow node 10.0.1.101 to access the RDMA service at node 10.0.1.105. In this case, an ACL intent with four policies can be implemented in three steps: ① For the control path, policy p_1 allows the request types of `ConnectRequest` for creation and `ConnectReply` for destruction for QPs indexed at 200 and 500 from 10.0.1.101 towards 10.0.1.105, while other QP creation requests within 10.0.1.0/24 are denied by policy p_2; ② For the data path, policy p_3 ensures only 10.0.1.101 with two specific QPs of 200 and 500 can read the restricted MR address range of 10.0.1.105, while other memory operation types from 10.0.1.101 and all requests from 10.0.1.0/24 are rejected by policy p_4; ③ Finally, we apply four policies to finish a high-level ACL governance. Other regulating examples for unauthorized accesses are provided in the Appendix due to space limitations.

```
# for RDMA control path
policy p1 {
  predicate = match(sip = 10.0.1.101) & match(dip = 10.0.1.105) &
              match(sport = any) & match(dport = 4791) &
              match(type in {ConnectRequest, ConnectReply}) &
              match(lQPN = any) & match(dQPN in {200, 500})
    action = allow
}

policy p2 {
  predicate = match(sip = 10.0.1.0/24) & match(dip = 10.0.1.105) &
              match(sport = any) & match(dport = 4791) &
              match(type in {ConnectRequest, ConnectReply}) &
              match(lQPN = any) & match(dQPN = any)
    action = deny
}

# for RDMA data path
policy p3 {
  predicate = match(sip = 10.0.1.101) & match(dip = 10.0.1.105) &
              match(sport = any) & match(dport = 4791) &
              match(dQPN in {200, 500}) & match(opcode in {READ}) &
              match(VA in [0x00001000, 0x00002000])
    action = deny
}

policy p4 {
  predicate = match(sip = 10.0.1.0/24) & match(dip = 10.0.1.105) &
              match(sport = any) & match(dport = 4791) &
              match(dQPN = any) & match(opcode = any) &
              match(VA in [0, inf])
    action = deny
}

# Activate four ACL policies
apply(p1, p2, p3, p4)
```

Fig. 6: An example of using JANUS language to regulate RoCEv2 communication. For both control and data path, operators can define an intricate intent consisting of multiple policy instances, with predicates involving multiple target fields in Table II and the values to be matched against, associated with an `allow` or `deny` action. Finally, operators can activate all policy instances.

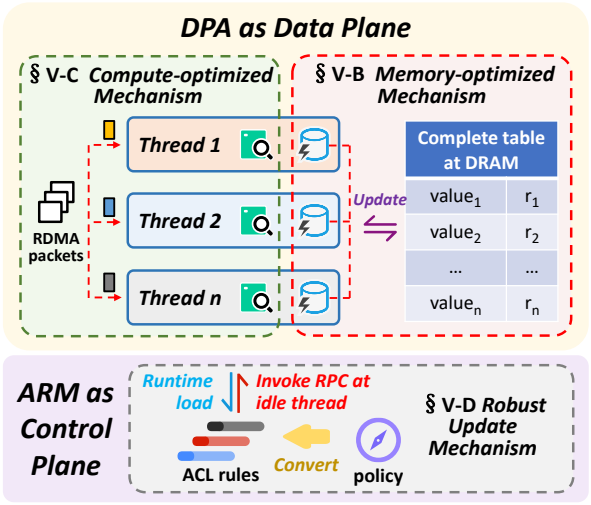


Fig. 7: JANUS enforcement overview.

V. JANUS ENFORCEMENT

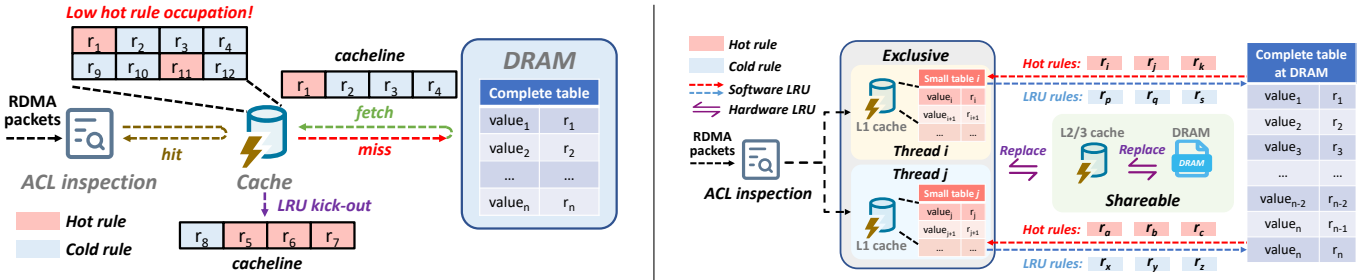
This section presents how to enforce ACLs with per-packet inspection and runtime updates on DPUs. Generally, RDMA clouds pose two basic requirements on ACL enforcement. Firstly, characterized by line-rate throughput and ultra-low latency, RDMA necessitates an efficient ACL enforcer to avoid degradation on high-speed RDMA transmission. Secondly, operators can generate frequent ACL updates for many RDMA applications, services and tenants, requiring us to ensure policy consistency, timely updates and uninterrupted inspection at runtime. To satisfy the requirements, JANUS enforces the ACL function on NVIDIA BlueField-3 DPUs [54] with several system optimizations in different aspects. We note JANUS prototype is implemented on NVIDIA BlueField-3 DPU for research purposes, but the design and optimizations are generalizable to other smartNICs (e.g., Intel E2200 IPU [55], Marvell OCTEON TX2 [56]) sharing a similar architecture with BlueField-3—an ARM CPU for policy management and a RISC-V accelerator for parallel computing.

A. Enforcement Overview

Figure 7 demonstrates JANUS enforcement overview. Given that an ACL enforcer can be inherently parallelized into multiple independent and lightweight per-packet checks, and that the DPA offers such parallelism, JANUS employs the DPA as the data plane to perform efficient per-packet inspection. In contrast, policy updates in the ACL enforcer are more complex, triggered less frequently but involving higher computational overhead. Therefore, JANUS delegates these updates to the ARM CPU, which serves as the control plane. After defining high-level ACL intents using JANUS expressions and language, the control plane on ARM first receives the latest policy from the centralized controller. Then, the policy is converted into fixed-size rules, waiting for insertion into the data plane with `insert`, `delete` and `modify` primitives. The DPA, a multi-thread RISC-V processor, examines each received RDMA packet according to the policies. The complete ruleset is stored in heterogeneous DRAM. Each received RDMA packet is steered to a specific thread for ACL lookup: packets failing the permission check are dropped, while permitted packets proceed to subsequent processing.

As we mentioned in §II-C, the NVIDIA BlueField-3 DPU is capable of offloading an ACL enforcer. However, meeting the requirements for efficient packet inspection and robust policy updates on the DPU remains non-trivial. Firstly, characterized by parallelized computation and heterogeneous memory, DPU exhibits varying packet lookup performance with different data structures, hardware usage schemes, and resource scheduling strategies. This necessitates a memory-optimized and compute-efficient ACL enforcement, tailored to DPU specifications and RDMA traffic patterns, to enable high-performance packet inspection. Secondly, completing an update on the DPU involves interactions between the control plane and data plane. However, due to non-negligible overhead during this process, update delays and policy inconsistencies may arise. More critically, a target thread on the DPA must pause ongoing packet lookups to communicate with the ARM core, resulting in unexpected suspensions. Therefore, a robust mechanism is required to ensure consistent and timely updates without interrupting ongoing ACL inspections. Last but not least, RDMA is highly sensitive to out-of-order delivery [57], [58], as it lacks the packet retransmission and reordering mechanisms widely adopted in TCP/IP networks [59]. Therefore, when utilizing and scheduling DPU resources, it is crucial to maintain in-order lookup for each RDMA packet without compromising high-speed RDMA transmission.

Based on the above analysis, JANUS introduces several



(a) Naive hash table

(b) Hierarchical hash table

Fig. 8: Cache-friendly hash table.

system-level optimizations tailored for DPUs. It incorporates a cache-friendly hash table along with efficient scheduling for heterogeneous DRAM to optimize frequent memory accesses (§V-B). To orchestrate the parallelized compute resources, JANUS adopts dynamic load-aware packet steering and batched doorbell rings (§V-C). JANUS also supports consistent, non-blocking and batched updates of ACL rules to ensure system robustness during policy changes (§V-D).

B. Memory-optimized Mechanism

NVIDIA BlueField-3 DPU has a hierarchical memory subsystem. The DPA contains local L1, L2, and L3 caches: each thread has an exclusive 1KB L1 cache, and all threads share the L2 and L3 caches with capacities of 1.5MB and 3MB, respectively. In addition, the DPA can access multiple DRAM components, including a 1GB DPA main memory, 16GB ARM memory, and massive host memory, each exhibiting diverse access performance [60]. For instance, the access latency from a DPA thread to the exclusive L1 cache, the shared L2/3 caches, the DPA main memory, and the ARM memory are 12.8ns, 40ns, 69ns and 318ns, respectively. Meanwhile, the memory read bandwidth from a DPA thread to the DPA, ARM, and host DRAMs are 9.8GB/s, 20GB/s and 7.2GB/s, respectively. Since ACL involves very frequent reads on the ruleset, designing a memory-optimized mechanism is essential with consideration on packet inspecting logic, RDMA traffic patterns, and DPU memory characteristics.

Cache-friendly hash table. Traffic locality, where bursty packets from the same RDMA connection arrive within a small period, is common in RDMA networks [1], [2], [61], [7], [58]. In this situation, certain hot ACL rules can be frequently accessed over a period of time, making it beneficial to place them in faster caches to improve memory access performance. BlueField-3 DPU has a hardware-native cache replacement strategy, i.e., the DPA cache automatically fetches frequently used content from main memory, and discards the stale content using a least-recently-used (LRU) approach. A naive data structure is to store all ACL rules in one complete table at main memory, and relies on the native cache replacement to update L1, L2, and L3 caches, but this naive scheme results in a low cache hitting rate. As illustrated in Figure 8(a), cache replacement works at the granularity of a cacheline (64B by default), while each ACL rule occupies much less capacity (16B in this example). In this case, when a hot rule r_1 is inserted into the cache, several address-adjacent but cold rules in main memory (r_2, r_3 and r_4) are also brought in as part of the same cacheline. This reduces the proportion of "real" hot rules in the DPA caches, since the cold rules that are not accessed frequently also account for a non-negligible share, ultimately lowering the cache hitting rate.

To improve the cache hitting rate, JANUS designs a cache-friendly data structure by disaggregating the hot & cold ACL rules with two-tier hash tables, which is shown in Figure 8(b). The complete table stores all rules in the shareable memory, and several small tables locate in the local thread memory, each of which is sized at 1KB and is exclusively accessed by the local thread for recording the local hot rules. To maintain the hotness of small tables, JANUS develops a software LRU algorithm to update each local small table: before a new rule is inserted, JANUS evicts the LRU rule to the complete table, whose position is replaced with the new hot rule. When inspecting a packet permission, JANUS follows a hierarchical table lookup order: if the relevant rule is found in the small table, the lookup stops; otherwise, it proceeds to search the complete table, then inserts the matched rule to the target local table. Under skewed traffic conditions, most ACL lookups can be finished at the small local tables with a low likelihood to access the shareable complete table. With frequent accesses on the small table memory, the hardware LRU strategy updates each L1 cache by fetching the contents in the local small table, with cacheline comprising of "real" hot rules maintained by software LRU algorithm. Therefore, each local L1 cache is populated with more "real" hot rules, thereby increasing the cache hitting rate.

Efficient scheduling on heterogeneous DRAMs. In addition to local cache at DPA threads, how to optimize the usage on heterogeneous DRAMs also remains essential for ACL lookup performance. Generally, two main data structures require careful DRAM scheduling: 1) ring buffer to send and receive packets, and 2) the complete hash table with numerous ACL rules. We consider their placement on heterogeneous DRAM by considering the ACL lookup and DPU memory characteristics. Firstly, we place the ring buffer for receiving packets at the ARM DRAM. The reason is that fetching massive data from the ring buffer is a bandwidth-sensitive operation, and ARM DRAM offers significantly higher memory bandwidth than the DPA memory and is located closer than host memory. Secondly, we place the complete hash table in the DPA DRAM. This is because ACL lookup requires very

timely lookup results with frequent memory reads, and the DPA DRAM has lower access memory latency than other DRAM components due to closer locations to DPA threads. With optimized scheduling of heterogeneous DRAMs, JANUS achieves high bandwidth to fetch data from the ring buffer, along with low access latency for responsive lookups for high-speed RDMA networks.

C. Compute-optimized Mechanism

ACL inspection involves frequent computation, including hashing operations and value matching of header fields against specific rules. It thus becomes necessary to orchestrate the DPU compute resources efficiently.

Dynamic load-aware packet steering. Due to the wimpy computing power of one single thread, a straightforward approach is to leverage DPA's parallelism to conduct packet inspection. However, under parallelism settings, with an incoming new packet, we need to determine which thread to inspect this packet efficiently. BlueField-3 DPU is equipped with a hardwired packet steerer, allowing the DPA to schedule an arrival packet to a specific thread according to its packet metadata. A naive strategy is to scatter each packet to a random thread, and with more arrival of packets, the load gradually converges and becomes balanced among threads. Nevertheless, this random approach can lead to out-of-order transmission [57], [58], where earlier-arriving packets being dispatched to overloaded threads, while later packets are assigned to less-loaded threads, resulting in the latter being processed ahead of the former. To avoid out-of-order RDMA transmission, one can make static configuration to steer packets of the same RDMA connection to a fixed thread and inspect them in a first-in-first-out manner. However, the static steering method is load unaware. In an extreme situation, many RDMA connections are steered to the same thread, resulting in skewed thread utilization and significant load imbalance, which disrupts the throughput and latency performance.

Therefore, we develop a dynamic and load-aware steering strategy to load-balance multiple threads by their runtime loads. Periodically, each thread in the DPA reports their runtime load to the control plane. The ARM CPU analyzes the traffic load distribution, and uses a heuristic approach to adjust the steering configuration for later incoming RDMA connections. In Figure 9, for example, with existing connections f_1, f_2 and f_3, thread k is overloaded with longest queue length, and we should avoid new arrival connections to be steered to thread k. For the new connection f_4, consequently, the control plane changes the steering configuration by greedily assigning f_4 to thread i with the least runtime load. With this approach, JANUS maintains load balance among multiple threads, optimizing queuing delay and overall inspection throughput.

Batching doorbell rings. In the naive implementation, when a packet is received in the ring buffer, the NIC triggers a hardware doorbell to inform the DPA to handle the packet. After the DPA processes the packet, it updates the record of software doorbells. However, it is very costly to update the software doorbell for each packet, especially with a traffic pattern dominated by bursty packets, leading to poor throughput. Therefore, JANUS adopts a batched approach: instead of a per-packet manner, JANUS triggers a software doorbell update only after the DPA completes inspections for n packet (customized by developers). The batching mechanism can greatly reduce doorbell rings and improve throughput performance.

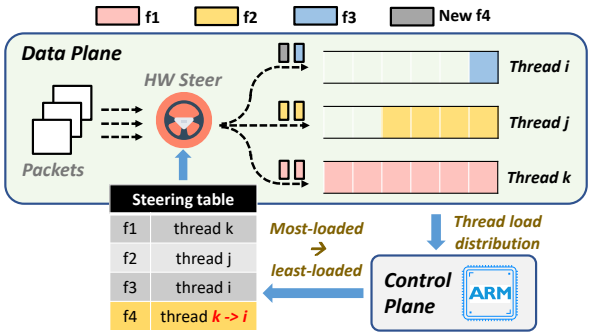


Fig. 9: Dynamic load-aware steering.

D. Robust Update Mechanism

In a large-scale RDMA cloud, policy updates can happen very frequently, necessitating a robust update mechanism with optimized coordination between the DPU control plane and data plane, in order to avoid policy inconsistency, update untimeliness and lookup suspension.

Consistent update. With multiple rules to be inserted, a naive update mechanism is a one-by-one approach that inserts then immediately activates each rule subsequently. However, this approach can generate transitional states because only a partial of rules are completed while the remaining are not, resulting in different versions of ACL intents and inconsistent inspection for RDMA traffic. To address this, we employ an atomic mechanism with version control for consistent update. We maintain a shadow hash table buffer with *version*. During an update, the data plane still sticks to the main hash table for inspection, and simultaneously modifies the shadow hash table buffer. Only when the update is completed will JANUS increment *version*, and let all threads turn to the shadow buffer to apply the latest ACL policies. This is useful to guarantee atomicity especially when a sophisticated ACL update is comprised of multiple policies.

Non-blocking batched update. To update the ruleset in the data plane, BlueField-3 DPU allows ARM to invoke a remote procedure call (RPC) to communicate with a thread in the DPA. A naive update mechanism would have ARM application invoke an RPC associated with the latest rule towards a DPA thread and modify the shareable hash table in the DPA. However, RPC invocation works in a blocking manner to preempt the target thread, such that the running packet inspection at this thread has to be suspended and waits for resume until the RPC communication finishes. The suspension period depends on RPC execution overhead, including context switch and coordination between ARM and the DPA, which typically lasts for tens of microseconds. The non-negligible suspension can aggravate the update latency and degrade ACL throughput.

TABLE III: JANUS enforcement settings.

Category \ Settings	**Naive**	**Optimized**
Parallelism	128	
Ruleset size	300K × 16B	
Traffic skewness	80%	
DRAM usage	ringbuf@DPA rule@ARM	ringbuf@ARM rule@DPA
Data structure	Simple	Cache-friendly
Packet steering	Static	Dynamic + load-aware
Doorbell batch size	1	8
Policy update	Blocking Inconsistent	Non-blocking Consistent
Update batch	1	100

To provide more timely and efficient updates, we develop a batched and non-blocking mechanism. Firstly, JANUS allows ARM to insert a batch of latest policies to the DPA hash table with one single RPC invocation, rather than invoking a new RPC per each update, thereby reducing the RPC invocation. Secondly, under parallelism configuration, we experimentally notice that fully utilizing all 190 available threads is not the optimal setting in a BlueField-3 DPU, which always exhibits poorer performance than partial parallelism (e.g., 128 threads shown in §VI-C). Based on this observation, we design a non-blocking mechanism: we reserve a subset of DPA threads only responsible for updating the complete table with RPC execution, and leave the remaining threads dedicated to running the ACL inspection without preemption by RPC invocation. This mechanism ensures that packet inspection and policy updates do not interfere with each other by RPC, meanwhile achieving the sweet point of ACL efficiency.

VI. EVALUATION

We evaluate JANUS to answer the following three key questions: (1) how expressive is JANUS to define ACL regulation with easy usage (§VI-B)? (2) how efficient is per-packet inspection and policy update performance (§VI-C)? (3) how necessary is the proposed optimizations for JANUS (§VI-D)?

A. Experimental Setups

Experiment environment. We develop JANUS with NVIDIA DOCA 2.5 [62] in the NVIDIA B3220 BlueField-3 DPU [54], which is equipped with two 200Gbps ports. Two DPUs are deployed on Dell R7525 servers, and are connected directly with a 200Gbps link. We generate RoCEv2 traffic in both control path and data path with two types of realistic workloads in RDMA clouds. The first workload is an LLM training job in RDMA clusters [7], [63], which is composed of elephant flows with large packets (payload $\geq$ 1KB). The second workload is a key-value (KV) query workload [64], which is a highly skewed traffic pattern with dynamic request sizes ranging from 88B to 1480B.

Baseline and design comparison. For ACL expressivity, we use open-sourced tools [14], [15], [12], [16] to generate malicious traffic, and evaluate JANUS for how to regulate various RDMA unauthorized accesses in JANUS language. For ACL enforcement, we refer to a public ACL ruleset collected from 12 real-world providers [65], and generate a ruleset with 300,000 synthetic rules with additional QP semantics, with each consuming 16 bytes and being inserted to DPUs for packet lookup. Before deploying rules to target DPUs, operators at the centralized controller can use existing network verifiers [36], [66], [67], [68] to resolve potential conflicts and ensure ACL correctness. Given that this issue falls outside the main scope of this work, we add a more detailed discussion in Section VII.

We compare the overall ACL performance and system overhead between the optimized JANUS, naive JANUS (i.e., without optimizations in §V) and non-JANUS (i.e., RDMA mode without ACLs) settings, as well as two ACL schemes of software enforcement (e.g., FreeFlow [10]) and in-network enforcement (e.g., Bedrock [28]). We also conduct several ablation studies to validate the effectiveness of each optimization proposed in §V. The enforcement settings are given in Table III with source codes available at Github [29].

B. Policy Expressiveness

Table IV compares JANUS with existing ACL schemes to reject six kinds of unauthorized accesses, and lists the policies and lines of code (LoCs) expressed in JANUS language. Compared to FreeFlow [10] that only regulates control path traffic at the shim layer, and Bedrock [28] that only governs the data path traffic at in-network switches, JANUS expressions with QP-semantics fields can block comprehensive misbehaving RDMA requests initiated from various sources, and enables definition of sophisticated ACL policies in a user-friendly manner, thus providing better coverage and usability than existing schemes. We give a brief introduction to three ACL intents as follows, with the remaining intent descriptions and all detailed policy codes in JANUS language given in the Appendix. The results prove JANUS can serve as one of the feasible solutions for security enhancement in RDMA clouds.

PU exhaustion by atomic verbs. It aims to exhaust the RNIC processing units by generating many atomic operations on the remote memory in the data path. To block the exhaustion requests, a feasible ACL intent can consist of three policies: 1) only allow the atomic verb traffic from trusted senders associated with QPNs; 2) drop all atomic verbs from other nodes by default; 3) reject QP creation requests applying for `CAS/FAA` operation types. The intent takes 32 LoCs as Figure 14 shows in Appendix.

QP connect exhaustion. It aims to exhaust the RNIC cache and file descriptors within kernel by issuing overwhelming `qp_connect` requests in the control path. To block them, we can: 1) deny the QP creation requests from blacklist entities; 2) allow the clients to create QPs by default. It takes 17 LoCs shown in Figure 15 in Appendix.

Unauthorized MR access. A tenant aims to conduct unauthorized access on a remote MR belonging to other tenants in the data path. The operators can reject the unauthorized traffic by: 1) dropping control and data path traffic from other tenants by their entity fields; 2) only allowing RDMA accesses from trusted entities and associated QP behaviors from the tenant's devices. This intent takes 28 LoCs in Figure 18 in Appendix.

TABLE IV: Comparison of existing ACLs against unauthorized accesses.

Unauthorized access	ACL schemes			
	FreeFlow [10]	Bedrock [28]	JANUS policy description	JANUS LoCs
PU exhaustion by atomic verbs [12]	✗	✓	1. allow atomic from whitelist entities and QPs 2. deny all traffic with atomic opcode by default 3. deny QP connect for `CAS/FAA` from blacklist	32
QP connect exhaustion [14]	✓	✗	1. deny QP connect requests from blacklist nodes 2. allow QP connect requests from whitelist	17
Fraud QP disconnect [15]	✗	✗	1. allow QP disconnect requests from whitelist 2. deny all QP disconnect requests by default	19
Packet injection [14]	✗	✗	1. allow matched packets of entities and QPNs 2. deny unmatched packets by default	26
Unauthorized MR access [14]	✗	✓	1. deny all traffic from other tenants 2. allow trusted transmission within the tenant	28

TABLE V: Comparison of performance and system overhead.

	Performance metrics		System overhead	
	Throughput (Gbps)	P99 latency (μs)	Compute utilization	Memory utilization
optimized JANUS	LLM: 195 KV: 102	LLM: 4.33 KV: 4.5	LLM: 97.71% (DPA) & 0.92% (ARM) KV: 99.53% (DPA) & 0.96% (ARM)	0.4% (DPA) & 0.3% (ARM)
non-JANUS	LLM: 197 KV: 121	LLM: 3.64 KV: 3.65	LLM: 93.65% (DPA) & 0.94% (ARM) KV: 99.24% (DPA) & 0.92% (ARM)	0.19% (DPA) & 0.1% (ARM)
FreeFlow [10]	LLM: 24.6 KV: 11.81	LLM: 64.93 KV: 57.56	Dedicated CPU core at 100%	N/A
Bedrock [28]	LLM: 198 KV: 19.2	LLM: 7.06 KV: 7.05	N/A	SRAM: 15.97% TCAM: 74.98%

C. Overall Enforcement Performance

Overall comparison. Table V compares the optimized JANUS with non-JANUS, as well as FreeFlow and Bedrock in terms of ACL performance and system overhead. Compared with non-JANUS in original RDMA, the additional ACL enforcer of optimized JANUS does not introduce obvious performance loss and system overhead to DPU hardware. In LLM workloads, for example, optimized JANUS exhibits almost same 200Gbps line-rate throughput as non-JANUS, and incurs only 4.33μs latency, satisfying the line-rate throughput and ultra-low latency requirements in RDMA. Meanwhile, optimized JANUS also introduces little overhead on the DPA and ARM for compute and memory resources, which only incrementally incurs about 4% DPA occupation and 0.3% memory consumption over the non-JANUS scheme, proving the deployability of enforcing ACLs at DPUs.

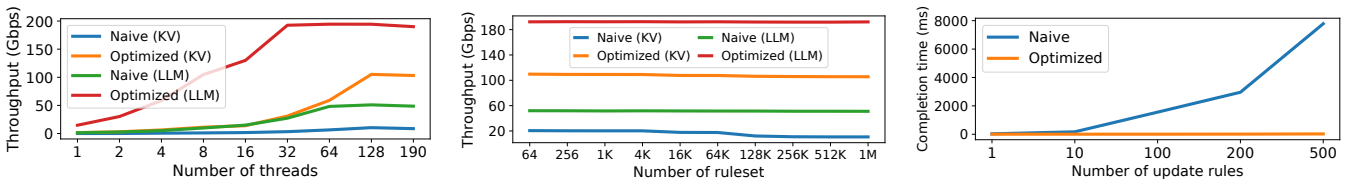
Compared with other ACL schemes, JANUS shows performance and overhead advantages. It is obvious FreeFlow achieves much lower throughput and higher latency than optimized JANUS, and dedicates a CPU core to capture RDMA traffic with 100% utilization, proving that software schemes are difficult to satisfy the performance and cost-efficiency requirements. Although Bedrock realizes line-rate inspection for LLM workload, it almost doubles the per-packet latency as it "pulls" the intra-host RDMA traffic to switches for permission checks, and delivers poor performance for KV workload dominated by small packets. Our results show that end-host DPUs provide superior, cost-effective ACL enforcement in RDMA clouds.

Parallelism degree. Figure 10(a) compares the ACL throughput of naive and optimized JANUS with different parallelism degrees. Generally, with increasing numbers of threads, the throughput of JANUS under different workloads rise correspondingly, and optimized JANUS performs much better than naive JANUS by up to 4.8×. Although difficult to achieve line-rate throughput with KV workload (this is because of BlueField-3's limited capability to process overwhelming small packets), our optimized JANUS still performs 12.52× better than the naive scheme without optimizations.

It is also noticeable that when parallelism degree reaches 190 (i.e., full usage of all available threads), the ACL throughput performance shows a subtle performance degradation compared to using 128 threads. This motivates us to design a non-blocking update mechanism mentioned in §V-D. For example, we can use 128 threads to achieve the optimal inspecting throughput, meanwhile reserve the remaining 62 threads to execute RPCs for policy updates, thereby delivering efficient packet inspection and policy updates simultaneously.

Ruleset size. Figure 10(b) illustrates the ACL throughput with different ruleset sizes. With more rules being inserted, the optimized JANUS shows a stable throughput with KV and LLM workloads, but the naive JANUS encounters a throughput degradation especially with KV workload. The reasons are two-fold: 1) due to the hash algorithm, both schemes need only $O(1)$ memory accesses to localize the matched rule, thereby maintaining a relatively stable lookup performance; 2) optimized JANUS with cache-friendly data structure makes most hot rules hit in the local L1 cache, while the naive scheme can encounter frequent cache misses and resorts to DRAM with worse access performance. The results prove JANUS's scalability to ACL ruleset sizes.

Policy update scale. Figure 10(c) compares the update com-



(a) Parallelism setting (b) Ruleset size (c) Policy update

Fig. 10: Overall ACL performance under different settings.

pletion time with various update scales. It is obvious that with more rules to update, naive JANUS exhibits a linear growth, while optimized JANUS maintains very low and stable update completion time. This is because optimized JANUS with batching mechanism can reduce RPC invocations from the control plane, while the naive scheme needs to invoke multiple RPCs equal to the numbers of inserted rules. This proves JANUS's adaptability to frequent and composite ACL updates in practical clouds.

D. Ablation Study

1) Memory-optimized Mechanism:

Cache-friendly hash table. Figure 11(a)-(c) show the throughput advantages of a cache-friendly hash table design in different settings. In Figure 11(a), as the traffic skewness rises, the cache-friendly design shows a higher throughput than the naive scheme without cache-friendly data structure. For instance, under a KV workload, when the traffic skewness increases from 50% to 95%, the ACL throughput with and without the cache-friendly design increases by 22.2% and 1.78%, respectively. It proves that despite additional maintenance (e.g., hierarchical tables with software LRU algorithm), a cache-friendly data structure leads to higher hitting rate of DPU caches and benefits more from higher traffic skewness.

Figure 11(b) explores the size of an exclusive small table for each local thread. When the small table size exceeds a threshold (e.g., 64 rules in our implementation), the ACL throughput starts to decline. Especially under a KV workload with much more frequent reads on the ruleset, the ACL throughput drops by 13.1%. This is because when the small table size is larger than the L1 cache size, the small table content cannot be entirely cached in each exclusive L1 cache, such that some hot rules are more likely to be placed at slower L2/3 cache, leading to a suboptimal L1 cache hitting rate. Therefore, an optimal size for each small table should not be larger than the size of local L1 cache.

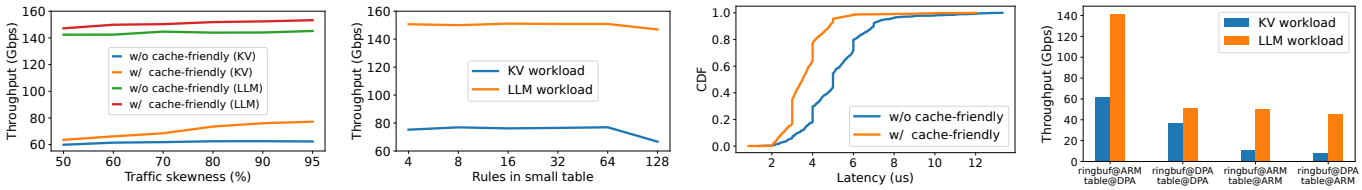
Figure 11(c) compares the cumulative distribution function (CDF) of per-packet latency with and without a cache-friendly hash table. It is obvious the latency performance with a cache-friendly design can outperform the naive scheme by 36.8% in the average case and 32.2% in the worse case. The reason is that a cache-friendly hash table can improve the proportion of hot rules within the L1 cache than the naive scheme, leading to a higher L1 cache hitting rate. Therefore, a cache-friendly data structure is helpful and necessary to reduce the lookup latency for each packet and satisfy the ultra-low latency requirement of RDMA. Meanwhile, we also note that JANUS still introduces $\sim 5\mu$s latency at the 99th percentile. While it satisfies the latency requirement of RDMA, there is still room for further optimization. We believe with further hardware optimization, JANUS's latency can be improved in future DPUs.

Heterogeneous DRAM scheduling. Figure 11(d) presents ACL throughput with different scheduling schemes of heterogeneous DRAMs for the ring buffer and ruleset. Obviously, when scheduling ring buffer at ARM's DRAM and ruleset at DPA's DRAM, the ACL throughput outperforms other scheduling schemes by at most 7.32× and 3.11× for KV workload and LLM workload, respectively. The reasons are: 1) fetching data from the ring buffer is bandwidth-sensitive operation, and ARM DRAM provides much higher memory bandwidth than the DPA DRAM; 2) the closer DPA DRAM can provide lower memory access latency with timely responsiveness for ACL lookup under very frequent ruleset reads. Therefore, a DRAM scheduling scheme of "ringbuf@ARM-ruleset@DPA" is more feasible in this scenario.

2) Compute-optimized Mechanism:

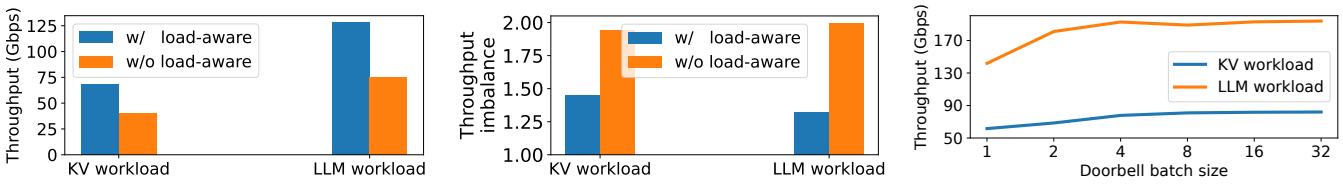
Dynamic load-aware steering. We use a static steering configuration where different RDMA connections are steered randomly to 128 threads at the DPA. Because the size of these connections are highly skewed, we find 60 out of 128 threads are heavily overloaded, while the other 68 threads are relatively idle. As optimization, our load-aware scheme can dynamically adjust the runtime steering configuration to load-balance multiple threads. Figure 12(a) compares the throughput with and without a load-aware steering. JANUS optimizes the ACL throughput by steering new arrival connections from overloaded threads to least-loaded threads according to their runtime load distribution, which outperforms the static scheme by 74.35% and 72.9% for KV workload and LLM workload, respectively. We also evaluate the load imbalance among multiple threads with and without load-aware steering, which is computed by $\frac{load_{max}-load_{min}}{\frac{1}{n}\sum_{i=1}^{n} load_i}$. Figure 12(b) shows that JANUS reroutes new connections from overloaded to least-loaded threads, reducing imbalance by 25.7 % for KV workloads and 34.2 % for LLM workloads. The results prove that a dynamic and load-aware packet steering can alleviate the load imbalance and improve the overall ACL throughput.

Batch doorbell rings. We evaluate the throughput with varying doorbell batch sizes as Figure 12(c) shows. Generally, a larger doorbell batch size is beneficial to higher ACL throughput by reducing the PCIe bandwidth consumption. For example, when the batch size reaches 32, the lookup throughput improves by 33.3% and 36.7% than no doorbell batches (i.e., batch size = 1) under KV workload and LLM workload, respectively. We also note it is not feasible to



(a) RDMA traffic skewness (b) Small table size (c) Packet processing latency (d) DRAM usage

Fig. 11: Ablation study on memory-optimized mechanism.



(a) Throughput (b) Throughput imbalance (c) Doorbell batch size

Fig. 12: Ablation study on compute-optimized mechanism.

set the batch size too large, as a larger batch size can lead to higher packet processing latency for each packet, thereby degrading the ACL inspection performance. To achieve the tradeoff between throughput and latency, a batch size with 8 is recommended in JANUS implementation.

3) Robust Update Mechanism:

Consistent update. With insertion of consecutive 100 rules to reject the target RDMA traffic, Figure 13(a) presents the runtime throughput with and without a consistent update mechanism. It is obvious they present varying throughput decline trends. Without consistent updates, there exist transitional states that only a partial of traffic is blocked but the remaining maintains permitted, such that the runtime throughput shows a slowly decreasing trend until all 100 rules are inserted completely. Instead, JANUS with consistent update ensures atomicity: only when all rules are updated will the new policy take effect, such that the runtime throughput suddenly drops to zero when all updates complete atomically, thereby eliminating any potential incomplete or delayed updates.

Non-blocking update. Figure 13(b) and (c) compare the runtime throughput of an inspecting thread during 100 and 200 consecutive updates, respectively. When the control plane invokes RPCs to insert rules into the data plane, the naive scheme without a non-blocking update mechanism has to suspend the target thread that performs packet lookup to execute the RPC process, making its throughput drop to nearly 0. Instead, with the non-blocking scheme by executing RPC at the reserved threads, the packet inspecting logic is not influenced and maintains stable lookup throughput during updates. With more updates, the performance advantage gets larger since the naive scheme needs linear-growing RPC invocations. These results confirm the non-blocking update mechanism's effectiveness, especially under frequent RDMA cloud updates.

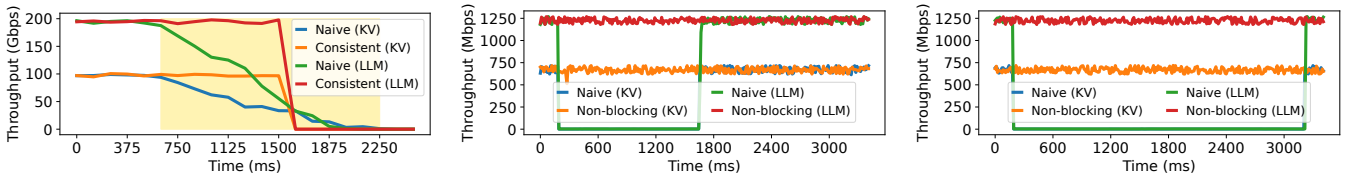
E. Security Analysis

Generally, JANUS achieves two security goals. Firstly, with ACL expressions in QP semantics and enforcing point at the critical path, JANUS delivers fine-grained governance on all traffic under RDMA semantics from various sources, including control path packets for QP state management and data path packets that are transmitted in a kernel-bypass manner. Therefore, the existing attacks related to unauthorized QP-granularity accesses can be blocked by JANUS with specified ACL rules. Secondly, with consideration of RDMA traffic pattern as well as tailored system-level optimizations on DPU memory and compute resources, JANUS achieves per-packet inspection and policy updates in 200Gbps throughput and $<5\mu$s latency, without compromising RDMA's superior performance. The analysis and experimental results prove JANUS's feasibility and practicality in real-world RDMA clouds.

As shown in the threat model (Figure 3), the JANUS enforcer is deployed on a trusted DPU to ensure that attackers cannot tamper with the ACL enforcement, thereby preventing potential circumvention or modification of the packet inspection logic. Furthermore, following the models in PortCatcher [50], JANUS only allows the trusted operators to make policies and deploy them to target DPUs at end-hosts, ensuring the reliability and security of ACL rules.

VII. DISCUSSION

Deployability of JANUS. Currently, we use programmable DPUs to enforce the ACL function. Though satisfying the coverage, efficiency and usability properties, the DPU-based enforcement scheme also introduces inevitable hardware deployment requirement to RDMA clouds. In fact, a better enforcement location is the RNIC itself: if the ACL enforcer is hardwired at RNIC processing units, it becomes a more easy-deployable and cost-efficient manner for large-scale deployment in cloud infrastructure. In addition to DPUs, we note there is other specialized hardware which has the potential to offload the ACL enforcer. For example, several smartNICs, such as Netronome Agilio series [69] and AMD Alveo series [70], allow developers to offload hardwired eBPF primitives, which hold the potentials to deliver performant ACL inspection, although they sacrifice the usability property of compatibility and transparency. While current enforcement still has some limitations, we hope the philosophy of JANUS



(a) Consistent updates (b) 100 consecutive updates (c) 200 consecutive updates

Fig. 13: Ablation study on robust update mechanism.

can inspire future development of next-generation RNICs with native ACL support.

ACLs for security enhancement. JANUS is designed to govern unauthorized accesses, which is empowered to block malicious requests aiming for disrupt the RDMA clouds. We highlight that ACLs serve as one of the most effective solutions for security enhancement, but are not the only scheme for addressing the security issues. Besides JANUS, we also note there are several works trying to improve the security of RDMA networks. For example, researchers secure the RNIC resource and performance isolation, such as cache and processing units [12], [13], to prevent a tenant's RNIC resource from illegal contention by malicious tenants. In fact, existing efforts tend to focus on specific security issues individually, and seek to resolve each of them at a hardware or system level. In contrast, JANUS is a general ACL system tailored for RDMA clouds, which enables operators to govern malicious request generation as needed. We believe both solutions can contribute to a safer environment, and a powerful combination with JANUS and each individual enhancement is promising for future multi-tenant RDMA clouds.

ACL correctness and management. Due to sophisticated expressions, JANUS imposes more scrutiny and overhead to design and update policies to achieve desired reachability, consistency and correctness in a production-scale RDMA cloud. We note that current TCP/IP-based network verifiers [36], [66], [67], [68] can be transferable to RDMA ACLs to address the above three challenges, since JANUS expressions are based on general operators with incremental QP-semantics fields to five-tuple. By analyzing the formalized relationship between ACL updates, therefore, network verifiers can detect and fix the violations when using JANUS expressions with specialized RDMA fields.

VIII. Related Work

Besides the most relevant works discussed in Section II-B, our work is also inspired by the following topics.

Expressions and policy languages. There are many domain-specific ACL expressions and languages to ease the policy regulation in various networking scenarios. For example, Ripple [52], Poseidon [25] and Jaqen [71] allow operators to customize DDoS defense policies with extension of Spark-like primitives in programmable switches in P4 programming abstract. Marple [72], Sonata [73] and Cerberus [74] provide high-level language with complicated primitives for operators to measure runtime anomaly within the network. PBS [53] and RDMI [75] define declarative languages to enable policies for BYOD and memory introspection tasks, respectively. Though providing effective and user-friendly regulation on traditional TCP/IP traffic, none of these works involve the RDMA-specific semantics, which motivates us to design a domain-specific expressions tailored for RDMA networks.

Offloading network functions to DPUs. There is an increasing trend of offloading network functions to DPUs to accelerate various network applications, e.g., distributed storage [76], [77], [78], [79], [80], AI training [81], [82], networking persistency & flexibility [83], [43], and multi-tendency support [84]. Although they unveil DPU's capability for different network functions, they lack specialized optimization on the unique characteristics and requirements of RDMA ACLs, and cannot be directly migrated to our scenarios with satisfactory transmission performance for RDMA.

IX. Ethical Consideration

No ethical consideration is raised in this paper. The experiments are conducted within our private RDMA testbed with all devices in our control. The malicious RDMA traffic is generated by open-sourced tools, including ReDMArk [14], NeVerMore [15], Husky [12] and LoRDMA [16].

X. Conclusion

This paper unveils why existing ACL expressions and enforcement fail in RDMA clouds, and proposes JANUS, an RDMA-tailored ACL system that satisfies the coverage, efficiency and usability properties. JANUS considers the RDMA-native semantics, and designs expressions as well as a policy language for easier regulation on the comprehensive QP behaviors. JANUS enforces ACLs in DPUs, and develops several optimizations on DPU resources for efficient packet inspection and policy updates. We implement JANUS, with experiment results showing JANUS's powerful expressivity for RDMA traffic and efficient enforcement with line-rate throughput and ultra-low latency in a 200Gbps RDMA testbed.

Acknowledgment

We thank the anonymous NDSS reviewers for their valuable comments. This work is supported in part by the National Natural Science Foundation of China (Grant 62221003, 62202260, 62402025), Shandong Provincial Natural Science Foundation, China (Grant ZR2024LZH011), the Research Project of Provincial Laboratory of Shandong, China (Grant SYS202201), the Research Project of Quan Cheng Laboratory, China (Grant QCL20250108), and the Fundamental Research Funds for the Central Universities. Jiahao Cao and Mingwei Xu are the corresponding authors.

REFERENCES

[1] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," in *SIGCOMM*, 2015.

[2] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "Rdma over commodity ethernet at scale," in *SIGCOMM*, 2016.

[3] Y. Gao, Q. Li, L. Tang, Y. Xi, P. Zhang, W. Peng, B. Li, Y. Wu, S. Liu, L. Yan *et al.*, "When cloud storage meets rdma," in *NSDI*, 2021.

[4] W. Bai, S. S. Abdeen, A. Agrawal, K. K. Attre, P. Bahl, A. Bhagat, G. Bhaskara, T. Brokhman, L. Cao, A. Cheema *et al.*, "Empowering azure storage with rdma," in *NSDI*, 2023.

[5] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong *et al.*, "Megascale: Scaling large language model training to more than 10,000 gpus," in *NSDI*, 2024.

[6] K. Qian, Y. Xi, J. Cao, J. Gao, Y. Xu, Y. Guan, B. Fu, X. Shi, F. Zhu, R. Miao, C. Wang, P. Wang, P. Zhang, X. Zeng, Z. Yao, E. Zhai, and D. Cai, "Alibaba hpn: A data center network for large language model training," in *SIGCOMM*, 2024.

[7] A. Gangidi, R. Miao, S. Zheng, S. J. Bondu, G. Goes, H. Morsy, R. Puri, M. Riftadi, A. J. Shetty, J. Yang *et al.*, "Rdma over ethernet for distributed training at meta scale," in *SIGCOMM*, 2024.

[8] W. Liu, K. Qian, Z. Li, F. Qian, T. Xu, Y. Liu, Y. Guan, S. Zhu, H. Xu, L. Xi *et al.*, "Mitigating scalability walls of rdma-based container networks," in *NSDI*, 2025.

[9] Z. Wang, X. Wei, J. Gu, H. Xie, R. Chen, and H. Chen, "Odrp: On-demand remote paging with programmable rdma," in *NSDI*, 2025.

[10] D. Kim, T. Yu, H. H. Liu, Y. Zhu, J. Padhye, S. Raindel, C. Guo, V. Sekar, and S. Seshan, "Freeflow: Software-based virtual rdma networking for containerized clouds," in *NSDI*, 2019.

[11] Z. He, D. Wang, B. Fu, K. Tan, B. Hua, Z.-L. Zhang, and K. Zheng, "Masq: Rdma for virtual private cloud," in *SIGCOMM*, 2020.

[12] X. Kong, J. Chen, W. Bai, Y. Xu, M. Elhaddad, S. Raindel, J. Padhye, A. R. Lebeck, and D. Zhuo, "Understanding rdma microarchitecture resources for performance isolation," in *NSDI*, 2023.

[13] J. Lou, X. Kong, J. Huang, W. Bai, N. S. Kim, and D. Zhuo, "Harmonic: Hardware-assisted rdma performance isolation for public clouds," in *NSDI*, 2024.

[14] B. Rothenberger, K. Taranov, A. Perrig, and T. Hoefler, "Redmark: Bypassing rdma security mechanisms," in *USENIX Security*, 2021.

[15] K. Taranov, B. Rothenberger, D. De Sensi, A. Perrig, and T. Hoefler, "Nevermore: Exploiting rdma mistakes in nvme-of storage applications," in *CCS*, 2022.

[16] S. Wang, M. Zhang, Y. Du, Z. Chen, Z. Wang, M. Xu, R. Xie, and J. Yang, "Lordma: A new low-rate dos attack in rdma networks," in *NDSS*, 2024.

[17] "Access control list (acl) overview," 2025. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/acl-overview.html

[18] "Access control policy," 2025. [Online]. Available: https://cloud.google.com/apigee/docs/api-platform/reference/policies/access-control-policy

[19] "Overview of azure network security," 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/security/fundamentals/network-overview#network-access-control

[20] "About network acls," 2025. [Online]. Available: https://cloud.ibm.com/docs/vpc?topic=vpc-using-acls

[21] M. Jethanandani, S. Agarwal, L. Huang, and D. Blair, "Yang data model for network access control lists (acls)," *RFC 8519*, 2019.

[22] M. Boucadair and T. Reddy.K, "Distributed denial-of-service open threat signaling (dots) data channel specification," *RFC 8783*, 2020.

[23] "iptables," 2025. [Online]. Available: https://www.netfilter.org/projects/iptables/index.html

[24] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *NSDI*, 2015.

[25] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," in *NDSS*, 2020.

[26] S.-Y. Tsai and Y. Zhang, "Lite kernel rdma support for datacenter applications," in *SOSP*, 2017.

[27] M. Marty, M. de Kruijf, J. Adriaens, C. Alfeld, S. Bauer, C. Contavalli, M. Dalton, N. Dukkipati, W. C. Evans, S. Gribble *et al.*, "Snap: A microkernel approach to host networking," in *SOSP*, 2019.

[28] J. Xing, K.-F. Hsu, Y. Qiu, Z. Yang, H. Liu, and A. Chen, "Bedrock: Programmable network support for secure rdma systems," in *USENIX Security*, 2022.

[29] "Janus enforcement on nvidia bluefield-3 dpu," 2025. [Online]. Available: https://github.com/czt8888/janus-bf3

[30] InfiniBand Trade Association, "InfiniBand Architecture Specification Release 1.2.1 Annex A17: RoCEv2," https://cw.infinibandta.org/document/dl/7781, 2014.

[31] "Infiniband trade association," 2024. [Online]. Available: https://www.infinibandta.org

[32] "Understanding iwarp: Delivering low latency to ethernet," 2018. [Online]. Available: https://www.intel.com/content/dam/support/us/en/documents/network/sb/understanding_iwarp_final.pdf

[33] L. Shalev, H. Ayoub, N. Bshara, and E. Sabbag, "A cloud-optimized transport protocol for elastic and scalable hpc," *IEEE Micro*, vol. 40, no. 6, pp. 67–73, 2020.

[34] D. Gibson, H. Hariharan, E. Lance, M. McLaren, B. Montazeri, A. Singh, S. Wang, H. M. Wassel, Z. Wu, S. Yoo *et al.*, "Aquila: A unified, low-latency fabric for datacenter networks," in *NSDI*, 2022.

[35] Y. Yuan, J. Huang, Y. Sun, T. Wang, J. Nelson, D. R. Ports, Y. Wang, R. Wang, C. Tai, and N. S. Kim, "Rambda: Rdma-driven acceleration framework for memory-intensive μs-scale datacenter applications," in *HPCA*, 2023.

[36] B. Tian, X. Zhang, E. Zhai, H. H. Liu, Q. Ye, C. Wang, X. Wu, Z. Ji, Y. Sang, M. Zhang *et al.*, "Safely and automatically updating in-network acl configurations with intent language," in *SIGCOMM*, 2019.

[37] "Security practices in aws multi-tenant saas environments," 2025. [Online]. Available: https://aws.amazon.com/jp/blogs/security/security-practices-in-aws-multi-tenant-saas-environments

[38] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks," *RFC 7348*, 2014.

[39] P. Matthews, I. van Beijnum, and M. Bagnulo, "Stateful nat64: Network address and protocol translation from ipv6 clients to ipv4 servers," *RFC 6146*, 2011.

[40] R. Kapoor, G. Porter, M. Tewari, G. M. Voelker, and A. Vahdat, "Chronos: Predictable low latency for data center applications," in *SoCC*, 2012.

[41] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen *et al.*, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *SIGCOMM*, 2015.

[42] Z. Yu, B. Su, W. Bai, S. Raindel, V. Braverman, and X. Jin, "Understanding the micro-behaviors of hardware offloaded network stacks with lumina," in *SIGCOMM*, 2023.

[43] C. Zhao, J. Min, M. Liu, and A. Krishnamurthy, "White-boxing rdma with packet-granular software control," in *NSDI*, 2025.

[44] "ibv_reg_mr," 2025. [Online]. Available: https://www.rdmamojo.com/2012/09/07/ibv_reg_mr

[45] "Optimized memory access," 2025. [Online]. Available: https://docs.nvidia.com/networking/display/mlnxofedv522230/optimized+memory+access

[46] "Open vswitch," 2025. [Online]. Available: https://www.openvswitch.org

[47] D. Firestone, "Vfp: A virtual switch platform for host sdn in the public cloud," in *NSDI*, 2017.

[48] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung *et al.*, "Azure accelerated networking: Smartnics in the public cloud," in *NSDI*, 2018.

[49] D. Bansal, G. DeGrace, R. Tewari, M. Zygmunt, J. Grantham, S. Gai, M. Baldi, K. Doddapaneni, A. Selvarajan, A. Arumugam *et al.*, "Disaggregating stateful network functions," in *NSDI*, 2023.

[50] C. Jung, S. Kim, R. Jang, D. Mohaisen, and D. Nyang, "A scalable and dynamic acl system for in-network defense," in *CCS*, 2022.

[51] C. Lever, "Rfc 8797: Remote direct memory access-connection manager (rdma-cm) private data for rpc-over-rdma version 1," 2020.

[52] J. Xing, W. Wu, and A. Chen, "Ripple: A programmable, decentralized link-flooding defense against adaptive adversaries," in *USENIX Security*, 2021.

[53] S. Hong, R. Baykov, L. Xu, S. Nadimpalli, and G. Gu, "Towards sdn-defined programmable byod (bring your own device) security." in *NDSS*, 2016.

[54] "Nvidia bluefield-3 dpu," 2025. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-3-dpu.pdf

[55] P. Fleming, C. Chang, D. Collier, A. Singhai, S. Doyle, E. Louzoun, D. Lee, V. Ayyavu, S. Livne, R. Hathaway *et al.*, "Intel® ipu e2200: Second generation infrastructure processing unit (ipu)," in *HCS*, 2025.

[56] "Marvell octeon tx2 cn913x product brief," 2025. [Online]. Available: https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-infrastructure-processors-octeon-tx2-cn913x-product-brief.pdf

[57] R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy, and S. Shenker, "Revisiting network support for rdma," in *SIGCOMM*, 2018.

[58] Z. Wang, X. Wan, L. Li, Y. Sun, P. Xie, X. Wei, Q. Ning, J. Zhang, and K. Chen, "Fast, scalable, and accurate rate limiter for rdma nics," in *SIGCOMM*, 2024.

[59] S. Floyd, H. Balakrishnan, and M. Allman, "Enhancing tcp's loss recovery using limited transmit," *RFC 3042*, 2001.

[60] X. Chen, J. Zhang, T. Fu, Y. Shen, S. Ma, K. Qian, L. Zhu, C. Shi, Y. Zhang, M. Liu, and Z. Wang, "Demystifying datapath accelerator enhanced off-path smartnic," in *ICNP*, 2024.

[61] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh *et al.*, "Hpcc: High precision congestion control," in *SIGCOMM*, 2019.

[62] "Doca documentation v2.5.0 lts," 2025. [Online]. Available: https://docs.nvidia.com/doca/archive/doca-v2-5-0/index.html

[63] X. Wang, Q. Li, Y. Xu, G. Lu, D. Li, L. Chen, H. Zhou, L. Zheng, S. Zhang, Y. Zhu, Y. Liu, P. Zhang, K. Qian, K. He, J. Gao, E. Zhai, D. Cai, and B. Fu, "Simai: Unifying architecture design and performance tunning for large-scale large language model training with scalability and precision," in *NSDI*, 2025.

[64] S. Li, H. Lim, V. W. Lee, J. H. Ahn, A. Kalia, M. Kaminsky, D. G. Andersen, O. Seongil, S. Lee, and P. Dubey, "Architecting to achieve a billion requests per second throughput on a single key-value store server platform," in *ISCA*, 2015.

[65] D. E. Taylor and J. S. Turner, "Classbench: A packet classification benchmark," *IEEE/ACM transactions on networking*, 2007.

[66] D. Guo, S. Chen, K. Gao, Q. Xiang, Y. Zhang, and Y. R. Yang, "Flash: fast, consistent data plane verification for large-scale network settings," in *SIGCOMM*, 2022.

[67] P. Zhang, A. Gember-Jacobson, Y. Zuo, Y. Huang, X. Liu, and H. Li, "Differential network analysis," in *NSDI*, 2022.

[68] Z. Li, P. Zhang, Y. Zhang, and H. Yang, "Ndd: A decision diagram for network verification," in *NSDI*, 2025.

[69] "Agilio cx smartnics," 2025. [Online]. Available: https://netronome.com/agilio-smartnics

[70] "Amd alveo sn1000 smartnic accelerator card," 2025. [Online]. Available: https://www.amd.com/en/products/accelerators/alveo/sn1000/a-sn1022-p4.html

[71] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar, "Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches," in *USENIX Security*, 2021.

[72] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim, "Language-directed hardware design for network performance monitoring," in *SIGCOMM*, 2017.

[73] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in *SIGCOMM*, 2018.

[74] H. Zhou and G. Gu, "Cerberus: Enabling efficient and effective in-network monitoring on programmable switches," in *S&P*, 2024.

[75] H. Liu, J. Xing, Y. Huang, D. Zhuo, S. Devadas, and A. Chen, "Remote direct memory introspection," in *USENIX Security*, 2023.

[76] J. Kim, I. Jang, W. Reda, J. Im, M. Canini, D. Kostić, Y. Kwon, S. Peter, and E. Witchel, "Linefs: Efficient smartnic offload of a distributed file system with pipeline parallelism," in *SOSP*, 2021.

[77] T. Kim, D. M. Ng, J. Gong, Y. Kwon, M. Yu, and K. Park, "Rearchitecting the tcp stack for i/o-offloaded content delivery," in *NSDI*, 2023.

[78] J. Zhang, H. Huang, L. Zhu, S. Ma, D. Rong, Y. Hou, M. Sun, C. Gu, P. Cheng, C. Shi *et al.*, "Smartds: Middle-tier-centric smartnic enabling application-aware message split for disaggregated block storage," in *ISCA*, 2023.

[79] Q. Zhang, P. Bernstein, B. Chandramouli, J. Hu, and Y. Zheng, "Dds: Dpu-optimized disaggregated storage," in *VLDB*, 2024.

[80] J. Shu, K. Qian, E. Zhai, X. Liu, and X. Jin, "Burstable cloud block storage with data processing units," in *OSDI*, 2024.

[81] Y. Xiao, D. Z. Tootaghaj, A. Dhakal, L. Cao, P. Sharma, and A. Kuzmanovic, "Conspirator: Smartnic-aided control plane for distributed ml workloads," in *ATC*, 2024.

[82] M. Khalilov, S. Di Girolamo, M. Chrapek, R. Nudelman, G. Bloch, and T. Hoefler, "Network-offloaded bandwidth-optimal broadcast and allgather for distributed ai," in *SC*, 2024.

[83] A. Psistakis, F. Chaix, and J. Torrellas, "Minos: Distributed consistency and persistency protocol implementation & offloading to smartnics," in *HPCA*, 2024.

[84] M. Khalilov, M. Chrapek, S. Shen, A. Vezzu, T. Benz, S. Di Girolamo, T. Schneider, D. De Sensi, L. Benini, and T. Hoefler, "Osmosis: Enabling multi-tenancy in datacenter smartnics," in *ATC*, 2024.

APPENDIX

```
############################ for RDMA control path ############################
policy p5 {
  predicate = match(sGID in {100, 200}) & match(dGID = 500) &
              match(type in {ConnectRequest, ConnectReply}) &
              match(lQPN = any) & match(dQPN = any)
    action = allow
}

policy p6 {
  predicate = match(sGID = any) & match(dGID = 500) &
              match(type in {ConnectRequest, ConnectReply}) &
              match(lQPN = any) & match(dQPN = any)
    action = deny
}

############################ for RDMA data path ############################
policy p7 {
  predicate = match(sGID in {100, 200}) & match(dGID = 500) &
              match(dQPN in {300, 400}) & match(opcode in {CAS, FAA}) &
              match(VA in [0x00001000, 0x00002000])
    action = allow
}

policy p8 {
  predicate = match(sGID = any) & match(dGID = 500) &
              match(dQPN = any) & match(opcode in {CAS, FAA}) &
              match(VA in [0, inf])
    action = deny
}

############################ Activate ACL policy ############################
apply(p5, p6, p7, p8)
```

Fig. 14: Govern atomic operations.

```
############################ for RDMA control path ############################
policy p9 {
  predicate = match(sip in 10.0.4.0/24) &
              match(type in {ConnectRequest, ConnectReply}) &
              match(lQPN = any) & match(dQPN = any)
    action = deny
}

policy p10 {
  predicate = match(sip = any) &
              match(type in {ConnectRequest, ConnectReply}) &
              match(lQPN = any) & match(dQPN = any)
    action = allow
}

############################ Activate ACL policy ############################
apply(p9, p10)
```

Fig. 15: Govern QP connect requests.

```
############################ for RDMA control path ############################
policy p11 {
  predicate = match(sip = 10.0.1.101) & match(sport = any) &
              match(dip = 10.0.1.105) & match(dport = 4791) &
              match(type in {DisconnectRequest, DisconnectReply}) &
              match(lQPN in {300, 400, 500}) & match(dQPN in {200, 600})
    action = allow
}

policy p12 {
  predicate = match(sip = 10.0.1.101) & match(sport = any) &
              match(dip = 10.0.1.105) & match(dport = 4791) &
              match(type in {DisconnectRequest, DisconnectReply}) &
              match(lQPN = any) & match(dQPN = any)
    action = deny
}

############################ Activate two ACL policies ############################
apply(policy11, policy12)
```

Fig. 16: Regulate QP disconnect requests.

```
############################## for RDMA data path ##############################
policy p13 {
  predicate = match(sip = 10.0.5.109) & match(sport = any) &
              match(dip = 10.0.1.10) & match(dport = 4791) &
              match(dQPN in {500, 550}) & match(opcode = WRITE) &
              match(VA in [0x00001000, 0x00002000])
    action = allow
}

policy p13 {
  predicate = match(sip = 10.0.5.104) & match(sport = any) &
              match(dip = 10.0.1.10) & match(dport = 4791) &
              match(dQPN = 600) & match(opcode = WRITE) &
              match(VA in [0x00003000, 0x00005000])
    action = allow
}

policy p14 {
  predicate = match(sip = 10.0.5.0/24) & match(sport = any) &
              match(dip = 10.0.1.10) & match(dport = 4791) &
              match(dQPN = any) & match(opcode = WRITE) &
              match(VA in [0, inf])
    action = deny
}

############################ Activate ACL policy ############################
apply(p12, p13, p14)
```

Fig. 17: Regulate packet injection.

```
############################## for RDMA data path ##############################
policy p15 {
  predicate = match(sip = 10.0.10.100) & match(sport = any) &
              match(dip = 10.0.10.10) & match(dport = 4791) &
              match(dQPN in {30, 50, 70}) & match(opcode = READ) &
              match(VA in [0x00002000, 0x00003000])
    action = allow
}

policy p16 {
  predicate = match(sip = 10.0.10.200) & match(sport = any) &
              match(dip = 10.0.10.10) & match(dport = 4791) &
              match(dQPN in {80, 120}) & match(opcode = WRITE) &
              match(VA in [0x00004000, 0x00005000])
    action = allow
}

policy p17 {
  predicate = match(sip = 10.0.10.0/8) & match(sport = any) &
              match(dip = 10.0.10.10) & match(dport = 4791) &
              match(dQPN = any) & match(opcode in {WRITE, READ}) &
              match(VA in [0, inf])
    action = deny
}

############################ Activate ACL policy ############################
Janus.apply(p15, p16, p17)
```

Fig. 18: Govern unauthorized MR access.

Fraud QP disconnect. It aims to disconnect a victim's QP by unauthorized QP disconnect requests with CM protocol. A feasible regulation strategy in the control path can be shown in Figure 16: 1) allow the QP disconnect requests from whitelist clients with associated legal QPNs; 2) deny the disconnect requests from untrusted endpoints and QPNs. The above policies take 19 LoC with JANUS.

Packet injection. It aims to inject unauthorized data to a destination MR of the server. A practical governance in the data path is illustrative in Figure 17: 1) allow the trusted client with QPs from whitelist to launch `WRITE` operations on specific MR address; 2) deny other `WRITE` requests towards the rest of the MR address. The above ACL intent takes 26 LoC with JANUS.