1RMA: Re-envisioning Remote Memory Access for Multi-tenant Datacenters

Arjun Singhvi[†‡] Aditya Akella[†‡] Dan Gibson[‡] Thomas F. Wenisch[‡] Monica Wong-Chan[‡]
Sean Clark[‡] Milo M. K. Martin[‡] Moray McLaren[‡] Prashant Chandra[‡] Rob Cauble[‡]
Hassan M. G. Wassel[‡] Behnam Montazeri[‡] Simon L. Sabato[*] Joel Scherpelz[°]
Amin Vahdat[‡]
[†]*University of Wisconsin - Madison* [‡]*Google Inc* [*]*Lilac Cloud* [°]*Unaffiliated*

Abstract

Remote Direct Memory Access (RDMA) plays a key role in supporting performance-hungry datacenter applications. However, existing RDMA technologies are ill-suited to multi-tenant datacenters, where applications run at massive scales, tenants require isolation and security, and the workload mix changes over time. Our experiences seeking to operationalize RDMA at scale indicate that these ills are rooted in standard RDMA's basic design attributes: connection-orientedness and complex policies baked into hardware.

We describe a new approach to remote memory access – One-Shot RMA (1RMA) – suited to the constraints imposed by our multi-tenant datacenter settings. The 1RMA NIC is connection-free and fixed-function; it treats each RMA operation independently, assisting software by offering fine-grained delay measurements and fast failure notifications. 1RMA software provides operation pacing, congestion control, failure recovery, and inter-operation ordering, when needed. The NIC, deployed in our production datacenters, supports encryption at line rate (100Gbps and 100M ops/sec) with minimal performance/availability disruption for encryption key rotation.

CCS Concepts

• **Networks → Network design principles**; *Data center networks*.

Keywords

Remote Memory Access; Connection Free; Congestion Control

ACM Reference Format:

Arjun Singhvi, Aditya Akella, Dan Gibson, Thomas F. Wenisch, Monica Wong-Chan, Sean Clark, Milo M. K. Martin, Moray McLaren, Prashant Chandra, Rob Cauble, Hassan M. G. Wassel, Behnam Montazeri, Simon L. Sabato, Joel Scherpelz, Amin Vahdat. 2020. 1RMA: Re-envisioning Remote Memory Access for Multi-tenant Datacenters. In *Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication (SIGCOMM '20), August 10–14, 2020, Virtual Event, NY, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3387514.3405897

1 Introduction

The scale, diversity, and performance requirements of modern datacenter applications, such as search, ads serving, video transcoding, and machine learning, demand networks that support high bandwidth and operation (op) rates while achieving low tail latencies. Remote Direct Memory Access (RDMA) is an attractive option for such distributed systems because of the latency and op-rate benefits provided by one-sided reads and writes, as these ops involve no remote CPU and thus offer performance limited only by hardware [1, 8, 12, 13, 17, 19–21, 24, 29, 41].

Industry-standard RDMA evolved from supercomputer environments and has been challenging to deploy in commercial datacenters [17, 41]. RDMA assumes low-latency, reliable, ordered networks and supercomputing fabrics deliver on these expectations via switch-enforced lossless link-level flow control, which allows an RDMA-capable NIC (RNIC) to implement naive congestion control and loss recovery schemes to react to congestion and drops. These fabrics are commonly single-tenant (or statically partitioned), and RDMA solutions for authorization, access control, fault recovery, and privacy reflect single-tenant expectations.

In contrast, modern hyperscale datacenters are characterized by multi-tenancy, wherein uncoordinated large-scale distributed applications share common infrastructure. A diverse, time-varying application mix induces rapidly changing network traffic patterns. Strong privacy and authentication are needed. These requirements lead to tension with standard RDMA's design choices:

- Standard RDMA offers *connections in hardware*, an abstraction that aligned well with early RDMA applications, but one that places fundamental limits on at-scale isolation, performance, and fault-tolerance. With modern serving and storage systems [2–4, 6, 7, 10, 15, 37] operating beyond ten-thousand-server scale, per-connection hardware resources are easily exhausted. Workarounds (*e.g.*, connection sharing) lead to broken isolation, which is further exacerbated under failures (§2).
- In our experience, congestion control algorithms need constant iteration in response to deployment and application considerations. Standard RNICs (and switches) bake significant portions of congestion response into hardware; this leaves little opportunity to adapt post-deployment.
- As applications and infrastructure are mutually-untrusting, multi-tenancy calls for line-rate encryption and application support to manage provenance of encryption keys. Although modern RNICs provide encryption, practical challenges arise: encryption is intrinsically tied to the notion of connections; applications must trust lower levels of the stack to manage keys; and there is no support for security-related management operations, such as encryption key rotation.

Responsibility	RDMA	1RMA
Inter-op ordering	NIC	Software
Failure recovery	NIC	Software
Flow and congestion control	NIC and Fabric	NIC and Software
Security ops (*e.g.*, Rekey)	None	NIC

Table 1: Division of responsibilities in standard RDMA and 1RMA. Moving a subset of functionality to software simplifies hardware and enables more flexibility/rapid iteration.

We take a new approach to remote memory access (RMA) to better match the constraints of our consolidated, multi-tenant datacenters. Our approach delivers the performance advantages of standard one-sided RDMA—high bandwidth, high op rate, and low latency—while also providing predictable tail performance, scalability, fault tolerance, isolation, security features, and amenability to rapid post-deployment iteration.

We achieve these goals by dividing responsibilities between hardware and software, in a manner that represents a stark departure from standard RDMA (Table 1). Our NIC hardware is extremely simple, focused exclusively on fast, fixed-function primitives. We offer no illusion of infinite resources (unlike RDMA; §2) and instead manage the explicitly-finite hardware resources in software. To facilitate rapid iteration, software implements fault recovery, congestion control, and ordering when needed.

Our clean-slate design – One-Shot RMA (1RMA) – embraces several design idioms to achieve our objectives:

(1) No connections: 1RMA is *connection-free*. Hardware state does not grow with the number of endpoint pairs. Freed from connection semantics, the NIC can treat each op as independent of other ops, leaving software to handle inter-op ordering when needed. 1RMA assigns to software the duties of per-op retry and fault recovery (hence, the name "One-Shot"), and instead provides simpler *fail-fast behavior*: 1RMA hardware ensures timely completions ($< 50\mu s$) and delivers fast op failure notifications directly to applications, with clean semantics.

(2) Small-sized ops, with solicitation as the basis for all transfers: Each 1RMA op transfers at most a 4KB payload, which enables isolation and prioritization. Furthermore, all data transfers are *solicited*: 1RMA does not initiate a transfer unless it is assured to land the data in on-NIC SRAM [36]. Hardware-enforced solicitation prevents formation of large incasts by design. Solicitation, paired with small ops, enables responsive congestion control and bounds in-network queueing.

(3) Software-driven congestion control: 1RMA hardware *assists* software congestion control, in contrast to contemporary proposals that bake congestion control algorithms into hardware [25, 31, 41]. 1RMA provides per-op delay information and explicit fail-fast notification for congestion events, allowing software to easily distinguish between local and remote congestion and take precise actions to minimize congestion, timeouts, and drops.

(4) Software-defined resource allocation: 1RMA turns its explicitly-finite resource pools to advantage via software resource allocation that assigns resources based on business priority, and places explicit bounds on the work applications can offer at a time. These bounds, coupled with small ops, prevent low-priority applications from monopolizing the network. Crucially, planning for operation within the explicitly-finite resource pools simplifies hardware, as there is no need to provide the illusion of near-infinite resources.

(5) First-class security: All 1RMA transfers are encrypted and signed in hardware with line-rate AES-GCM [14], designed in concert with 1RMA's connection-free architecture. 1RMA allows applications to directly manage encryption keys, without requiring extending trust to infrastructure software, and enabling frequent encryption key rotation with minimal availability disruption.

We evaluate 1RMA using a 40-node testbed and with simulations against state-of-the-art alternatives. 1RMA offers predictable latency even at high load and in failure cases. It ensures that high-priority applications are not impacted by low priority ones; lack of isolation in alternatives leads to >10× slowdown. 1RMA's congestion control converges to fair bandwidth shares in the presence of competing applications almost immediately ($25\mu s$); separately reacting to local congestion improves convergence speed by 20×. First-class support for security reduces the unavailability period during encryption key rotations to $<1\mu s$. These gains come at a minimal cost of 0.5 cores to drive 100 Gbps line-rate, as 1RMA chunks large ops into 4KB ops, and implements congestion control and op management in software.

	READ	ATOMIC	WRITE	SEND/RECV
RC	✓	✓	✓	✓
UC	✗	✗	✓	✓
UD	✗	✗	✗	✓

Table 2: Standard RDMA ops supported by each transport type. In RC transport, the RNIC is responsible for retransmissions following a loss.

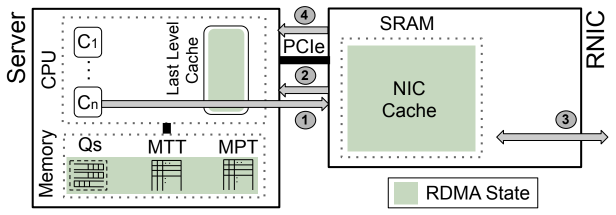

Figure 1: In standard RDMA (not 1RMA), (1) ops begin when software posts a request to a send queue. (2) The RNIC DMAs any data required for transfer and (3) sends it to the remote RNIC, which looks up the memory in question, validates that the connection belongs to the same protection domain, performs the op, and (4) acknowledges.

2 Background and Motivation

Standard RDMA offers three different transport types, each of which supports a different subset of ops (Table 2). The prevalent transport is RC, or "reliable connected".

Queue Pairs (QP), Connections. An application establishes connected *queue pairs* (QPs) between application-pairs via out-of-band exchange of tokens (*e.g.*, via RPC or `librdmacm`). The server-side determines access control rules by binding connections to logical protection domains. A QP consists of a send queue and a receive queue, and is bound to a completion queue (CQ). Connected transports (RC and UC) offer one-to-one communication between QPs (called *connections* hereafter).

Op Execution Flow. Applications post ops (called work queue entries, or WQEs) to the relevant queue via a user-space library (`libibverbs`). Figure 1 shows the steps in the remaining execution flow. To improve performance, RNICs cache active connections' state as well as the memory translation table (MTT) and memory protection table (MPT). These tables contain metadata related to memory available for remote access [33]; they reside authoritatively in host memory.

2.1 Challenges with Standard RDMA

Standard RDMA's architecture leads to several fundamental challenges for use in multi-tenant datacenters:

1. Connection exhaustion. Caching host-resident state allows RNICs to provide an illusion of unbounded connection counts. However, RNIC caches can be overwhelmed by large workloads, leading to performance cliffs as the cache is filled on demand [9, 12, 18–21, 33]. Worse still, allocation policies oblivious to business priorities can strand critical applications without connections, *e.g.*, when lower-priority applications have previously consumed all available connections. Recent works [16, 19–21, 28] explore using Unreliable Datagram (UD) to overcome scalability limitations, often implementing connection abstractions in software. However, UD is two-sided, and re-introduces CPU bottlenecks on the receiving host, falling short of one-sided op rate and latency.

2. Induced ordering. Systems may attempt to address connection exhaustion by multiplexing several independent workloads on the same QP. However, standard RDMA requires FIFO execution of ops of the same type within a single QP, thereby inducing false ordering constraints among unrelated ops. Thus, this approach can lead to priority inversion due to head-of-line blocking between large, background ops and small, latency-sensitive ops [39]. Likewise, op failures may cause connection teardown, imposing shared fate for ops unluckily sharing a connection. Furthermore, the ordered-execution requirement places a de facto near-in-order packet delivery requirement on the network, significantly complicating deployment of modern high-performance network capabilities, such as adaptive routing [26]. And yet, the RNIC itself must still implement order recovery in hardware [23, 31], exacerbating complexity.

3. Poor semantics. We have found that there is little semantic utility from RDMA's notions of ordering. In particular, it is not possible to reason about RDMA write side effects under connection teardown cases: standard RDMA's write op may still cause new side effects (*e.g.*, mutating memory at the destination) even *after* its failure has been reported (*e.g.*, via retransmit timeout at the initiator; see Figure 2). Such side effects make building reliable distributed algorithms more difficult.

4. Connection-centric security. Standard RDMA ties access control to connections; a system that eliminates connections must solve access control by other means. Although recent RNICs provide line-rate encryption, these approaches *also* rely on connections, and do not provide ready means to manage encryption keys. Applications that seek to rotate encryption keys in standard RDMA are obliged to reconnect, a costly and complex operation.

5. Rigid congestion control. RDMA over Converged Ethernet (RoCE) allows RDMA traffic to coexist with traffic from other protocol stacks, but requires fabric switches to use Priority Flow Control (PFC) to provide a near-lossless substrate. As is well known, using

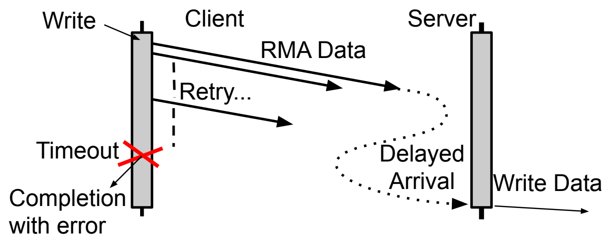

Figure 2: Delayed writes can cause future mutations after the client receives a completion event with an error.

PFC is untenable in commercial scale networks [17, 41] due to head-of-line blocking, poor at-scale failure isolation [30, 38, 41], and risk of deadlock [17, 38]. Recent hardware congestion control schemes reduce reliance on PFC [25, 41], but such techniques have limited applicability in our environments, in which congestion control algorithms are routinely updated and customized per deployment.

6. Firmware slow-paths. RNICs may rely on firmware to handle corner cases that arise in congestion control and other sources of complexity; at scale, firmware traps can create invisible and intolerable bottlenecks, leading to goodput collapse in our datacenters. For example, firmware's handling of connection teardown can compete with its ability to handle loss and maintain ordering.

3 1RMA Overview

1RMA overcomes the challenges of datacenter-scale remote memory access via a design philosophy that: (1) delegates to software all actions whose full and correct realization requires application intervention, such as ordering and failure recovery, as these fundamentally cannot be realized solely in the NIC; (2) implements in hardware functionality that precludes software intervention, notably DMA capabilities, host-wide incast bounding, authentication, and encryption.

The resulting 1RMA NIC is simple, and focuses on providing performant one-sided read, write, and management primitives implemented entirely in hardware. Send and receive are not directly supported, as these are easy enough to implement in software.

In 1RMA, software handles op pacing, congestion management, and policy choices on when/how to retry failed ops (*e.g.*, those that time out). The 1RMA NIC assists software by providing timely op completion ($< 50\mu s$) including and especially in failure cases, and by providing early indications of congestion build-up through precise delay measures in each completion.

3.1 Example: 1RMA Read Op Execution

We illustrate the overall operation of 1RMA by means of an example 2KB-sized RMA read (see Figure 3).

Prior to executing any RMA op, the initiating client performs an out-of-band RPC to obtain the necessary information to access the remote memory region in question (① – ②). These include an encryption key, K_d (i.e., a cryptographically-secure key bound to the initiating client, difficult to guess and non-transferrable §4.2), and a `RegionId`—the architectural name of the memory to be accessed—established at memory-registration–time on the server. All protocol messages are signed using a message authentication code generated from K_d. Similarly, all data is encrypted using K_d.

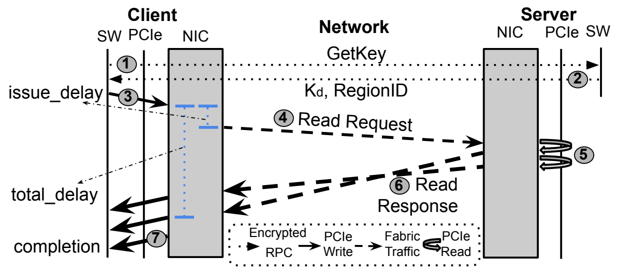

Figure 3: Execution of a 2KB Read op in 1RMA. (1 & 2) Client performs out-of-band communication to obtain information to access remote memory region. (3) The client initiates the 2KB read op by writing a command into a command slot on the 1RMA NIC. (4) The op is sent over the network subject to 1RMA's solicitation rules. (5) The request reaches the server-side 1RMA NIC which reads the requested data via PCIe and (6) streams read data as individual network responses. (7) Finally, once all the data arrives, a successful op completion is written to the client software.

Having obtained the necessary information for remote access, the client initiates its desired RMA op—in this example, a 2KB read—by writing a command over PCIe with write-combining MMIO stores into an on-NIC *command slot*, which enqueues the operation for service (③). The request awaits execution, subject to 1RMA's *solicitation rules*. To enter service, an op requires 4KB of free space in the on-chip *solicitation window*, which is an SRAM buffer that lands inbound payloads. Arbitrating for 4KB regardless of op size prevents small ops from starving large ops.

Once the op enters service, the NIC debits the solicitation window by the actual size (2KB), signs the read request using K_d provided in the RMA command (§4.4), and sends it on the network (④).

Commonly, the request arrives at the server-side 1RMA NIC shortly thereafter, which consults a fixed-size, on-chip table to look up key information for the `RegionId` included in the request, derives K_d, and authenticates the inbound packet. The NIC then reads the requested data via PCIe (⑤), and streams read completions as individual network responses (⑥). Each response is encrypted and signed with the previously derived key information. These responses traverse the network, perhaps arriving out-of-order due to adaptive routing.

Upon reaching the initiator, each response is individually authenticated and decrypted, then streamed via PCIe writes to the initiating host's memory, each at an offset encoded in the inbound response. To tolerate unordered responses, the NIC tracks byte arrivals, and once all bytes arrive (in error-free cases), a successful op completion is written to the initiating software (⑦). The completion also includes *hardware delay measures*, indicating how long it took to execute the operation (`total_delay`) and how long it took for the request to enter service at the initiator (`issue_delay`).

Whereas we have described the failure-free case, the example operation above can experience a variety of failures. For instance, the read operation may not enter service at the initiator due to heavy local congestion (*e.g.*, due to a large number of queued ops). When the request arrives at an overloaded server, the serving NIC might drop or "`NACK`" the request (*e.g.*, due to an over-long inbound request queue; §4.5). Finally, responses from the server to the client may be dropped

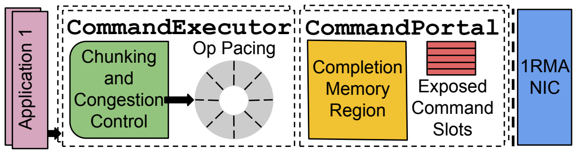

Figure 4: 1RMA Software consists of two components: (1) `CommandPortal` provides a familiar command/completion queue construct and (2) `CommandExecutor` provides support for arbitrary large transfers and congestion management.

or delayed in the network. 1RMA generates precise fast feedback in the form of explicit failure codes that indicate to the initiating software which failure mode manifested (§4.5). Upon encountering such failures, application software can take the appropriate action, such as immediately retrying the operation, perhaps to a different backend server replica in the scope of a broader system.

Our example focused on a small 2KB op. For larger transfers (*e.g.*, a 4MB read), a 1RMA per-client software module called `CommandExecutor` (Figure 4) breaks large transfers into small up-to-4KB ops (§4.4), and then batches, pipelines and paces these ops at a rate determined by a software-based congestion control algorithm. Congestion control relies on hardware delay measures and precise failure outcomes reported with completions to modulate request rate to avoid both local and network/remote congestion while keeping utilization high (§6).

4 1RMA Design In Depth

This section describes the key ideas of 1RMA, its security model, hardware components, and software abstractions.

4.1 Key Ideas

There are five key elements to 1RMA's design:

- **Connectionless security**: 1RMA embraces security requirements as first-order. We use a novel, connection-free protocol that binds both encryption and authentication to the specific memory region being accessed; and to the accessing process/host pair.
- **Solicitation**: 1RMA relies on solicitation to limit the severity of sudden, transient incasts, because software congestion control cannot react instantaneously. 1RMA uses fine-grained per-operation admission control, via a solicitation window maintained at each initiating NIC, which bounds the dynamic number of inbound bytes. New requests will stall until sufficient window capacity can be allocated. Window capacity frees as prior ops complete.
- **Writes via request-to-read**: 1RMA implements write operations as a request-to-read: the writer asks a remote NIC to retrieve data via a read operation. Although this approach adds a round-trip, it unifies security and solicitation for reads and writes and underpins our solutions to provide precise write failure semantics, replay protection, and incast avoidance.
- **Explicitly-managed hardware resources**: 1RMA offers no illusions of unlimited resources. Instead, it leverages higher-level resource allocation to apportion its finite hardware resources according to application-level priority. This approach explicitly bounds the work applications can offer at a time. Importantly, offering no illusions of unlimited resources enables 1RMA to do

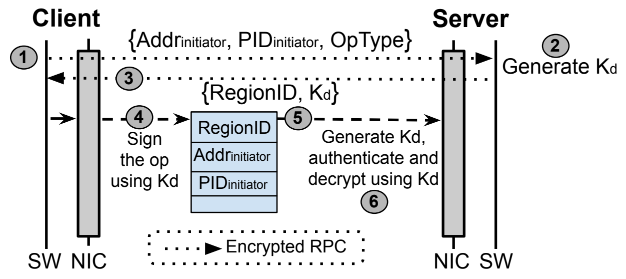

Figure 5: Derived region key generation and distribution. (1) Client sends its address, PID, and OpType to the server via an encrypted RPC. (2) Server software generates a derived region key, K_d, using the information provided by the client and the region key, K_r, corresponding to the memory region in question and (3) sends K_d to the client. (4) The client-side 1RMA NIC signs the op using the obtained K_d and (5) sends the request to the server. (6) On receiving the request, the serving 1RMA NIC regenerates the key and uses it for authentication and decryption.

away with on-NIC caches and the associated cache consistency between the host and NIC, leading to simpler hardware and avoiding performance pathologies of cache-oriented designs.

- **Fast completions with precise feedback**: Building on the predictability offered by solicitation and explicit resource management, the 1RMA NIC imposes tight timeouts on op completions. 1RMA aggressively times out delayed ops—slow ops are converted to failures—and provides unified timeout and failure semantics. Fail-fast behaviors simplify reasoning about congestion, ensure that operations do not consume the solicitation window for an inordinate amount of time, bound the worst-case operation latency, and allow applications to handle failures responsively and in an application-appropriate way.

4.2 1RMA Security

Given that we target deployments with mutually untrusting endpoints and an untrusted network fabric, security is built into and tied to all basic aspects of 1RMA. Before delving into the details, we first discuss the two attacks vectors 1RMA addresses:

A1 A malicious user process that attempts to access remote memory regions owned by other tenants. Such an attacker can freely initiate 1RMA ops of their choice (which must fail).

A2 An attacker with full access to network links and switches (*e.g.*, root exploit of network control plane or physical/side-channel observation). Such an attacker can observe, corrupt or inject ciphertext in transit, and also inject arbitrary packets.

Similar to standard RDMA, 1RMA bootstraps remote access between a client and a server with an out-of-band encrypted RPC that includes secure exchange of a key (Figure 5, ① – ③). Specifically, on memory registration, a region is assigned a `RegionId`. An application specifies a region key, K_r, that protects the corresponding memory region. K_r is a 128-bit value from which derived region keys, K_d, are computed, and these form the basis of 1RMA's security and protect individual transfers. Neither K_r nor K_d are ever sent over the network as a part of a 1RMA op or its response.

Derived keys (Figure 5). K_d is used to generate a message authentication code to sign all protocol messages, and to encrypt all data pertaining to a transfer (④). It is computed as follows:

$$K_d = AES(Key = K_r, Contents = Address\_Initiator,$$
$$PID\_Initiator, OperationType)$$

This function computes a key that is specific to each initiating process (identified by the host address and the PID of the process) and op type, and yet is rooted in a single, shared K_r that easily fits in on-NIC memory in a per-region table. When an application allocates *command slots* (§4.3)—used to initiate RMAs—the 1RMA driver stores the associated PID_Initiator immutably in the slots' configuration. 1RMA hardware always includes PID_Initiator in request packets along with the `RegionId` (⑤). Therefore, the serving 1RMA NIC can derive the assigned K_d for each inbound request (⑥). Also, key derivation can be easily performed in server-side software (which possesses K_r) — in the context of an authenticated RPC, server software can compute K_d for any potential initiator and communicate K_d to that initiator in the RPC response, without the need to retain K_d in any on-host or on-NIC tables (②). 1RMA salts the encryption process with per-NIC ascending message counters, which protects key integrity and guards against replays, covered in detail in Appendix A. Critically, on-NIC security-related state does not grow with the number of communicating endpoint pairs.

Returning to the two attack vectors: In A1, the attacker can easily guess `RegionId` but has difficulty in acquiring a K_d matching host and process, absent a root-level exploit. In A2, the attacker can observe ciphertext in transit, but not easily decrypt it without also subverting a participating host.

Of note, 1RMA also protects against replay attacks: though 1RMA does not prevent replayed read requests from being admitted, such attacks generate freshly-salted (by virtue of the server's ascending counter) ciphertext, so that an attacker still requires the correct K_d to decrypt the resulting responses (§5 discusses replay attacks on mutating commands).

1RMA's security model does not meaningfully change the behavior of a host with a root-level compromise; a root-level attacker can impersonate the processes on the host, and therefore can authenticate by whatever means it chooses as one of those users. The primary defenses against this threat are to accelerate detection, for example, by *enacting frequent encryption key rotation*, forcing repeated authentication steps, which can be logged and inspected. 1RMA provides explicit support for encryption key rotation to aid this objective (§5).

Authentication failures: 1RMA NICs drop responses that fail their authentication steps. In contrast, inbound *requests* failing authentication are sent an immediate failure notification in response, signed with a well-known reserved key, and with outcome `REMOTE_AUTHENTICATION_FAILURE`. It would be stronger to drop such requests, thereby forcing an attacker to face a timeout rather than a timely negative response. However, doing so penalizes non-attack cases, as the `REMOTE_AUTHENTICATION_FAILURE` error code is easily recognizable as a side effect of encryption key rotation, and therefore recovery steps are obvious to client software. In contrast, a dropped request manifests as a `TIMEOUT`, which does not immediately indicate that an encryption key rotation may have occurred.

Line-rate operation. Because 1RMA ops are independent and unordered, we are able to deploy multiple copies of AES-GCM hardware encryption blocks in our 100Gbps implementation of 1RMA

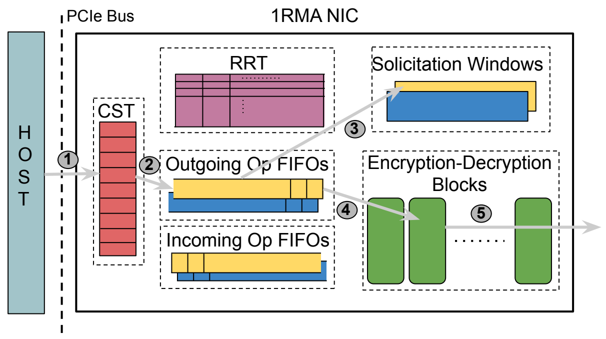

Figure 6: 1RMA NIC hardware components. (1) The client initiates an op, it is written to a command slot in the CST, and (2) the op is enqueued in a hardware FIFO, awaiting its turn for dispatch. (3) The queue head waits for capacity in the solicitation window after which the request is (4) signed with K_d and (5) sent to the remote side.

and freely load-balance among them. We assign work to these encryption pipelines as they become free, irrespective of order, as the underlying protocol and semantics allow this optimization.

4.3 Managing Hardware Resources

Given the inherent complexity, scalability, and performance limitations associated with cache-oriented NIC designs, 1RMA builds its finite resources into its execution model. The four main 1RMA NIC components (Figure 6) are:

- **Registered Region Table (RRT):** The RRT is 1RMA's memory translation table. Unlike standard RDMA, this table is maintained in fixed-size, on-NIC SRAM. A `RegionId` indexes this table to indicate a specific host memory range and all corresponding metadata—region key K_r, PCIe address, bounds, permissions, etc. Every time 1RMA accesses host memory, it does so on behalf of precisely one memory region in the RRT. Memory regions are managed via memory registration, similar to `ibv_reg_mr()` in standard RDMA.
- **Command Slots and Command Slot Table (CST):** Command slots are 1RMA analogues of RDMA WQEs; each slot represents a single in-flight operation and can be reused once the op completes. The CST consists of a fixed-number of slots in on-NIC SRAM. Each slot is uniquely identified by its `CommandSlotId`, which the NIC encodes in the op completion. This token indicates to the controlling software which command has completed, because ops may complete out-of-order.
- **SRAM Solicitation Window:** The solicitation window is allocated to inbound transfers and is the means to ensure that solicited data is not dropped due to, for example, transient PCIe backpressure. Because it is shared among outstanding operations, the solicitation window is sized proportional to bandwidth-delay product, with slack to account for jitter or RTT variation.
- **FIFO Arbiters:** Capacity in the solicitation window is shared dynamically by a pair of FIFO arbiters—one for each internal 1RMA class of service—which select among ready commands in the CST. When an application issues an op to a command slot, the CPU performs an MMIO write across PCIe to corresponding hardware registers [35] (1 in Figure 6). This write causes the 1RMA hardware pipeline to enqueue the op in a hardware FIFO based on its class of service (2), awaiting its turn for dispatch. The queue head waits for capacity in the solicitation window (3) before sending a request to the remote side, designated by address information in the command itself (4 – 5).

Each hardware structure's capacity is implementation specific and can change across 1RMA device generations. Capacity in the RRT and CST is centrally managed for each datacenter application, reflecting application needs. Unlike connections, the number of memory regions for RMA use cases is typically proportional to tasks, not task-pairs, and hence is manageable in finite resources. Similarly, the number of command slots allocated to a process is centrally managed, and caps how many outstanding operations a process may initiate, effectively bounding its burst potential in the network (*e.g.*, for lower-priority applications). Applications request command slot allocations from the host driver via an `ioctl`.

1RMA's solicitation rules can cause ops to queue in the initiating NIC while waiting for capacity in the solicitation window. Because the number of command slots is bounded, these queues, too, can be finitely sized and do not spill to the host. Because we desire highly responsive ops, these queues shed load eagerly by timing out ops that have been waiting too long to enter service. Timing out delayed ops helps to mitigate head-of-line blocking and provides a useful congestion signal to software congestion control (§6).

4.4 1RMA Software

Software layers: 1RMA software provides large-transfer abstractions and congestion management (see Figure 4). At the lowest layer, a `CommandPortal` object manages a collection of command slots and a memory region configured to accept completions, providing a familiar command/completion queue construct. Because command slots are memory-mapped registers in the NIC, the `CommandPortal` handles the details of `mmap()` to insert these registers into application memory; and provides routines to issue properly-formatted commands via MMIO stores to the correct portion of the 1RMA NIC's PCIe BAR (details in Appendix B).

Building on `CommandPortal`, the next layer in the software stack, `CommandExecutor`, provides support for arbitrary-sized transfers, transparently chunked into up-to-4KB-sized operations and subject to software pacing for congestion control (§6). Application software layers above `CommandExecutor` bear no responsibility for chunking, pacing, or congestion control. Applications can elect to manage failures according to individual needs. For example, consider a replicated key/value service in which a read op to a particular backend fails; rather than retrying for an inordinate amount of time under hardware control, the client software can opt to quickly redirect traffic to another backend server replica.

Commands: As the 1RMA NIC retains no per-destination state, each individual command fully encodes all metadata needed for the op. Apart from the op type and size, each command includes:

- Remote host address and `RegionId`, which uniquely identify the remote memory targeted by the operation.
- Two local `RegionId`s, one to source/sink data and another to serve as a completion queue.
- The derived region key K_d, used to sign the request and to decrypt responses.

Op Status Code	Op Outcome
`OK`	Op completed successfully and data was transferred.
`REMOTE_AUTHENTICATION_FAILURE`	Command did not provide the correct derived region key K_d.
`NACK`	Op reached the remote 1RMA device, discovered the inbound request queue above configured depth, and was NACKed, which signals congestion at the specific remote 1RMA device. Receipt of NACK immediately refunds capacity in the solicitation window.
`TIMEOUT`	Op timed out without transferring any data. `TIMEOUT` could arise due to a programming error (*e.g.*, wrong destination address), a network partition, a drop, or congestion.
`DISPATCH_TIMEOUT`	The request was queued too long locally, awaiting capacity in the solicitation window, and the op did not enter service. This outcome signals congestion local to the initiating NIC.

Table 3: Op status codes and rationale.

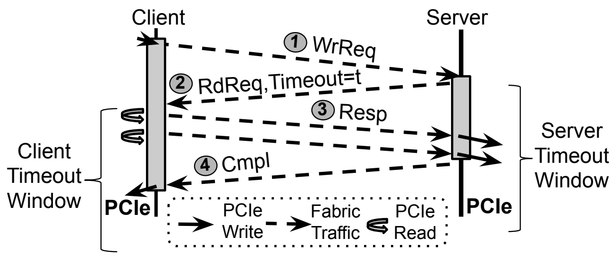

Figure 7: RMA write op execution flow. (1) Client sends a write request to server-side remote NIC. (2) Remote NIC sends a read request back to the client-side initiating NIC when 1RMA solicitation rules are satisfied. (3) The initiating NIC responds as though it received a read request and (4) the remote NIC sends a completion message.

4.5 Outcomes

Upon completion the 1RMA NIC returns a completion indicator to the initiating command's designated completion region. The completion includes the `CommandSlotId` of the completing slot, two hardware delay measures, `issue_delay` and `total_delay`, and an op status code, which conveys the nature of any errors (Table 3). Notable status codes include `NACK`s, which signal remote congestion, `DISPATCH_TIMEOUT`s, which signal congestion at the initiating NIC, and `TIMEOUT`s, which signal other errors that cause the op to fail.

5 Other 1RMA Ops

Apart from RMA reads (exemplified in §3), 1RMA also offers: (1) RMA writes, and (2) *Rekey* for use in regular rotation of encryption keys with minimal availability impact. Unlike reads, writes and Rekey use a four-hop protocol, thereby obeying solicitation and providing resilience against failures and replay attacks.

5.1 RMA Write

1RMA implements RMA writes as remote-request-to-read: each write solicits the remote node to read the initiator's memory. The sequence of steps (Figure 7) are: ① send write request to remote NIC, ② remote NIC sends a read request back to the initiating NIC when 1RMA solicitation rules are satisfied, ③ the initiating NIC responds as though it received a read request (following the steps in §3), and ④ the remote NIC sends a completion acknowledgement message. Both the remote and local 1RMA NICs can time out independently if messages are dropped or experience overlong delay.

Implementing write as request-to-read incurs the downside of an additional RTT, but offers several benefits:

(1) 1RMA writes obey solicitation, thereby providing incast burst protection for writes as well as reads, even when they occur concurrently.
(2) The additional protocol messages allow the server-side to include its own entropy as a salt, curried into the third- and fourth-step signatures, which protect 4-hop transactions against replay attacks.
(3) 4-hop write transactions have strong timeout semantics, like their RMA read counterparts, and unlike those provided by standard RDMA (§2.1). Figure 7 depicts the behavior of timeouts in 4-hop transactions. After receiving the second-phase "read request" (②), the initiating NIC *resets* its own local timeout, to a new, fixed time in the future, which is identical to the receiving-side's read timeout. Causality then guarantees that, if a timeout condition occurs, the initiator will necessarily time out *after* the receiver. Because the initiator always times out last, all 4-hop transactions provide a guarantee that a client timeout happens *after* any side effects in remote memory.

5.2 Rekey

1RMA offers a management operation, Rekey, to cheaply, and possibly remotely, install a new region key K_r in the RRT. Rekey is unique to 1RMA: no existing RMA implementation provides first-class support for encryption key rotation. We built Rekey because it is difficult to construct encryption key rotation primitives using standard RDMA without introducing either high transient connection usage (to pre-establish new connections with new encryption keys) or bursts of connection failure (which occurs when the server abruptly closes a connection when rotating keys). Coupled with 1RMA's connectionless nature, Rekey vastly simplifies encryption key rotation and minimizes performance side effects.

Systems built above 1RMA can leverage Rekey by initiating an RMA operation, which is no more expensive than an RMA write. In the simplest case, there is no need to proactively notify remote users of an upcoming rotation. Such users' ops simply begin to fail with outcome `REMOTE_AUTHENTICATION_FAILURE`, but only those ops targeting the intended `RegionId` are affected; those ops targeting unrelated regions see no performance impact, and there are no connections to fail. This makes Rekey an attractive means to unobtrusively install new encryption keys, without additional system-level coordination needed to prevent disruptive access errors. Instead of *avoiding* them, such errors are *tolerated* in 1RMA. More sophisticated rotation implementations can proactively notify users of upcoming Rekeys and new K_ds, which can reduce the period of unavailability due to a Rekey to a single RTT.

Because it is treated as a normal RMA operation, Rekey even allows a *remote* third party, in possession of the appropriate derived region key K_d, to change a region key K_r. Remote Rekey enables control planes to manage pools of remote memory with no local CPU involvement.

6 Congestion Control

1RMA implements congestion control (CC) policies in software (`CommandExecutor`), assisted by NIC support for hardware measured delays and precise failure outcomes, delivered with each op's completion. CC reacts to these signals by modulating offered load, specifically the rate at which software issues up-to-4KB commands to the 1RMA hardware.

Our objectives for CC are fourfold: (1) enable rapid iteration on policy, (2) avoid wasted bandwidth (*e.g.*, which occurs when reads time out due to congestion), (3) allocate bandwidth fairly, and (4) converge quickly in dynamic environments, as load comes and goes from bursting applications. Our first objective is met by virtue of 1RMA leaving all policy decisions to software. We meet the second objective by assigning target delays and modulating op issue rate such that hardware delay measures hover near our targets. For the remaining objectives, we leverage 1RMA's load-shedding outcomes to rapidly change op issue rates.

1RMA CC differs from traditional, packet-oriented schemes in two critical ways. First, 1RMA's software controls *when individual ops are initiated*, not the timing of packet-send. Second, all CC decisions are made on the initiating side only, because 1RMA is purely one-sided. 1RMA itself has no connections, so the `CommandExecutor` tracks congestion state per remote 1RMA NIC and per direction (inbound reads, outbound writes). Unlike hardware-based CC, such state resides in relatively cheap host DRAM and is never accessed or cached by the NIC.

Similar to Swift [22] and TIMELY [30], we use delay as a congestion signal and implement a software control loop to react to changes in delay. Delays are measured by hardware, broken down into two components (§4.5): `issue_delay` and `total_delay`. The difference between these two, which we call `remote_delay`, represents delay contribution from network congestion and remote queuing. The `issue_delay` signals local congestion at the initiating 1RMA NIC.

1RMA software uses the above delays to react separately to local and remote congestion; contemporary schemes [5, 23, 31, 41] cannot distinguish between these forms of congestion. 1RMA tracks a congestion window (CWND) for each remote destination/direction pair, and uses a single CWND for local congestion (because local congestion affects all local transfers). CC assigns op rates based on the more restrictive of the two CWNDs. Rising or falling delay prompts CC to adjust each CWND using a simple additive-increase-multiplicative-decrease control loop. Op failures trigger a more substantial CWND decrease. Notably, a `DISPATCH_TIMEOUT` triggers a 10× reduction in the *local CWND* (only), as this is a precise indicator of *local* congestion.

Algorithm 1 describes how 1RMA CC reacts to local congestion. The control loop probes for the correct congestion window, adjusting

Algorithm 1: 1RMA CC reaction to local congestion.

```
Input: issue_delay, RTT
Output: cwnd_local
On Successful Op Completion
  $cwnd\_local\_old \leftarrow cwnd\_local$
  if $issue\_delay < TARGET\_DELAY$ then
    ▷ Additive Increase (AI)
    if $cwnd\_local \geq 1$ then
      $cwnd\_local \leftarrow cwnd\_local + \frac{AI}{cwnd\_local}$   ▷ $AI = 0.25$
    else
      $cwnd\_local \leftarrow cwnd\_local + AI$
    $cwnd\_local \leftarrow min(cwnd\_local, CWND\_MAX)$
  else
    ▷ Multiplicative Decrease (MD)
    if no decrease in the last RTT time then
      $delta \leftarrow issue\_delay - TARGET\_DELAY$;
      $cwnd\_local \leftarrow$   ▷ $MD = 0.5, \beta = 0.8$
        $max(1 - \beta \cdot (\frac{delta}{issue\_delay}), MD) \cdot cwnd\_local$;
      $cwnd\_local \leftarrow max(cwnd\_local, CWND\_MIN)$;
On Dispatch Timeout
  if no decrease happened in the last RTT time then
    $cwnd\_local \leftarrow$   ▷ $TIMEOUT\_DECR = 0.1$
      $TIMEOUT\_DECR \cdot cwnd\_local$;
    $cwnd\_local \leftarrow max(cwnd\_local, CWND\_MIN)$
```

CWND up additively (down multiplicatively) when `issue_delay` is beneath (above) a target value. We choose the reduction factor to be proportional to the deviation between `issue_delay` and our target delay, bounded by half (0.5). `DISPATCH_TIMEOUT` causes a more drastic reaction, accelerating convergence when new local transfers begin (§7.4). The policy to control the CWND corresponding to remote congestion is similar, although `remote_delay` is compared against a distinct delay target, and `TIMEOUT` and `NACK` cause a 10× reduction in *remote* CWND, as these explicitly signal congestion specific to the network or a remote NIC.

1RMA uses the more restrictive of the two CWNDs to: (1) set a limit on the number of outstanding ops in the network, and (2) compute the ops' issue rate as $\frac{OpSize \times CWND}{RTT}$, which 1RMA enforces through the `CommandExecutor` in Figure 4. We are yet to investigate how 1RMA interacts with other transports. As such, we assign 1RMA its own traffic class on our production fabric.

6.1 Parameter choices

1RMA has several key parameters that impact CC effectiveness as well as 1RMA's overall performance and CPU overhead.

(1) Op Size: The 1RMA NIC bounds the size of individual ops. A small size ensures frequent feedback to, and precise reaction from, 1RMA CC. But, too small a size imposes high CPU overhead. Our benchmarks indicate that a single core can drive 6M ops/sec with our full software stack, which is twice what is needed for 100Gbps line rate with 4KB-sized ops. For large transfers, 4KB-sized chunks strike a balance between CPU overhead and CC responsiveness.

(2) Dispatch Timeout: An overly-full solicitation window can lead to a `DISPATCH_TIMEOUT`, signalled when an op cannot enter service locally within a bounded time interval. To software, this outcome signals local congestion. We tune the timeout value to 10μs, so that it is comparable to the block/wake time of a CPU thread. This timeout ensures local feedback is delivered immediately, and it enables timely CC responsiveness.

(3) Timeout: Delays and drops due to congestion trigger a more generic `TIMEOUT` outcome on affected operations. The bounded size of the solicitation window calls for shorter timeouts to help quickly reclaim window capacity. On the other hand, longer timeouts help ensure that delayed transfers are accepted at the receiving side and do not waste the sender's bandwidth. Because CC is effective at making timeouts rare, we tune these timeouts to 4-5× the fabric RTT. Timing out early is tolerable in 1RMA, because small op size bounds wasted work in the event of a false positive.

(4) NACK Threshold: The 1RMA NIC generates `NACK`s when the depth of inbound request queues cross above a configured threshold. We set the threshold to $\frac{(TIMEOUT-RTT-DISPATCH\_TIMEOUT)}{Bandwidth}$, so that `NACK`s are generated when a data response is unlikely to reach the initiator prior to a `TIMEOUT`, thereby preventing bandwidth waste at the serving NIC.

7 Evaluation

We evaluate 1RMA's distinctive features via testbed experiments and simulations. We report absolute performance (*e.g.*, 100Gbps) for context, but raw throughput is mostly a function of hardware generation. We designed our prototype hardware while 100Gbps network speeds were state of the art. We expect our approach will generalize to higher speed networks. To highlight the architectural choices in the 1RMA NIC, much of our evaluation focuses on choices relating to stability, utility, and predictability. We show that:

- 1RMA offers excellent performance in common cases, but remains stable and predictable under less-common (but important) failure cases (§7.1).
- 1RMA's intrinsic support for isolation and prioritization via independent, small-sized ops and software-driven resource allocation effectively prevents applications from monopolizing the network (§7.2).
- 1RMA's hardware support for encryption key rotation minimizes client-observable disruption (§7.3).
- Supported by hardware, 1RMA's CC converges to fair bandwidth shares in the presence of competing applications almost immediately (§7.4).
- 1RMA's solicitation rules prevent goodput loss due to transients (sudden dynamic changes), reacting at hardware speeds as software CC converges (§7.5).

Baselines. We compare against standard RDMA and Pony Express (Pony) [28]. With RDMA, our purpose is to highlight the implications of the *required and standard* behaviors of any compliant RNIC, with Mellanox NICs as examples. Pony, Google's software-defined NIC, represents a state-of-the-art datacenter networking alternative, most similar to 1RMA in its objectives. Pony supports one-sided ops by means of a userspace networking stack on the hosts. In all our experiments, we limit each networking stack to (at most) a single host CPU for network transport processing; this tends to limit Pony to 40Gbps maximum throughput (bidirectional). With larger CPU allocations, Pony performance scales commensurately.

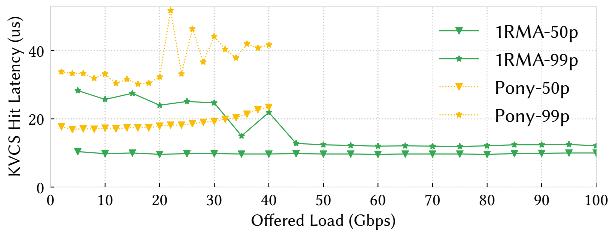

Figure 8: Latency vs. offered load for KVCS.

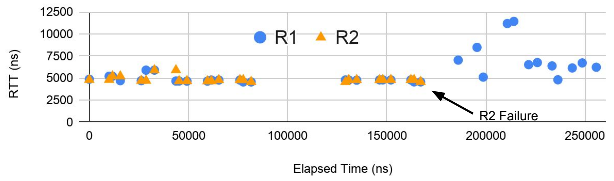

Figure 9: Observed latency during KVCS replica failure.

Testbed. Our 1RMA and Pony testbed consists of 40 Intel Skylake-based servers connected via the lowest-latency switches available to us. To study standard RDMA, we use 40Gbps Mellanox CX-3s for at-scale and 100Gbps CX-5s for point-to-point experiments. Our 40-node CX-3 testbed is from an older production generation, based on Intel Haswell. RDMA experiments run with PFC between the NIC and top-of-rack switch. Our evaluation uses a 1RMA hardware implementation that uses region keys, not derived region keys, for encryption and authentication.

Simulation. We augment our testbed findings with simulations covering various behaviors otherwise difficult to isolate.

7.1 Applications and Workloads

As performance is typically the goal of systems built atop RMA infrastructures, we first evaluate the extent to which 1RMA improves performance of (a) a key/value caching system and (b) a synthetic uniform random workload.

In-Memory Key/Value Caching Service (KVCS). We modified a Google production caching system (similar to [29]) to use 1RMA and Pony for cache lookups. Although the production KVCS operates at larger scales, for this controlled study we constrain it to ten nodes and evaluate its performance under varying load. The servers in this setting are outnumbered by the clients, such that the bottleneck is the serving-side NIC. Figure 8 plots performance per server, which ramps to the maximum practical throughput of each underlying RMA implementation.

1RMA outperforms (two-sided) Pony; with 1RMA we even observe tail improvement with load due to warming effects in client CPUs. 1RMA benefits from an all-hardware serving path, whereas software eventually bottlenecks Pony in this workload. The zero-load mark also highlights the effects of 1RMA design choices for command issue. Because 1RMA uses MMIO, it skips several PCIe RTTs (~400ns each) that software solutions built on UD [21] incur on the critical path (see Appendix B).

To demonstrate how 1RMA's fail-fast behavior can support fast application-level failure response, Figure 9 reflects the system under duress by injecting a failure into one of the KVCS server replicas, which we label R2. To KVCS, this manifests as an immediate op failure (i.e., not delayed by even a network transport timeout). In response, KVCS immediately shifts load to replica R1. Increased load

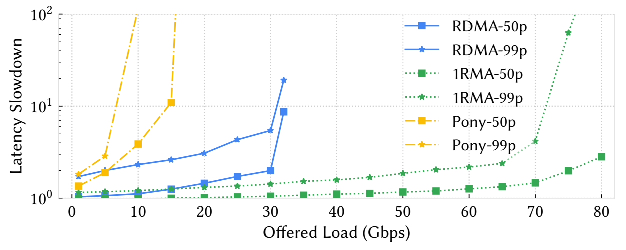

Figure 10: Latency vs. offered load for a uniform random traffic pattern.

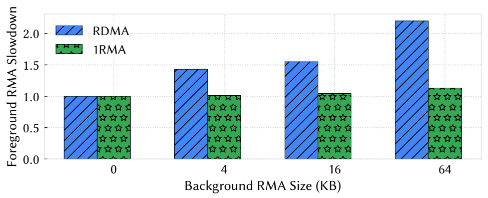

Figure 12: Impact of head-of-line blocking for a point-to-point workload.

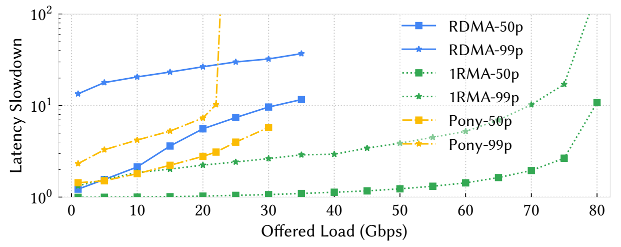

Figure 13: Impact of head-of-line blocking for a uniform random traffic pattern.

causes growth in latencies of ops serviced by R1 as it absorbs the new traffic, stabilizing in $< 100\mu s$ and without a period of unavailability.

Synthetic Workloads. Figure 10 plots *latency slowdown* in a synthetic, 40-node uniform random traffic pattern. Each node initiates 4KB ops according to a Poisson arrival distribution and randomly selects a destination node for the initiated op. The arrival distributions are configured to generate the required network load. For each baseline, slowdown is the ratio of the actual round trip latency of an op divided by the best possible latency for an op of that size on an unloaded network (slowdown of one is ideal). As before, one-sided RDMA and 1RMA have a performance advantage. At this scale, the RNIC operates well within its connection cache.

Figure 11a plots accepted load for 1RMA reads and writes, in a same-rack, point-to-point configuration. RMA writes diverge from reads near peak throughput as 1RMA begins to enforce solicitation rules, an effect also evident for reads at larger scale (§7.5). 4-hop writes consume solicitation window capacity longer than do 2-hop reads; hence RMA writes begin to saturate earlier, by design.

7.2 Application Isolation Benefits

1RMA builds isolation and prioritization into its design through transfer chunking and software-defined resource management.

Impact of independent, small-sized ops. Consider a simple point-to-point workload consisting of small (64B) foreground ops, driven

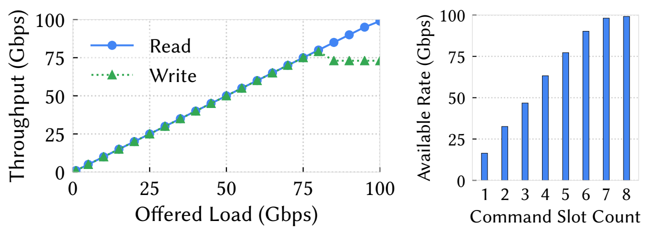

Figure 11: (a) Achieved throughput vs. offered load for reads and writes; (b) Load bounding via command slot allocation.

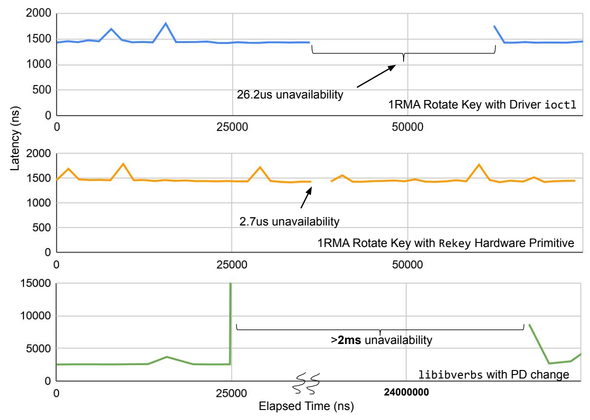

Figure 14: Client-observed unavailability during an encryption key rotation.

by 10K Poisson arrivals/sec, and one outstanding large transfer, sharing the connection in the case of RDMA (to forestall connection exhaustion). Ideally, the small ops should not be delayed by the large transfer. Figure 12 plots the latency slowdown of the small ops as we vary the size of the competing background op. Because 1RMA software chunks large transfers, the smaller ops experience a delay comparable to the service time of a single 4KB chunk. In contrast, the RNIC delay is proportional to the service time of a background op, and potentially severe. Such delays will occur on all RNICs, as they arise from RDMA's induced ordering.

The same behavior holds in uniform random patterns. Figure 13 plots slowdown experienced by small operations in a heavy-tailed workload. Ops are bimodally distributed among 4KB and 200KB-sized Reads (i.e., 85% of ops are 4KB, but 90% of bytes are from 200KB-sized operations) and follow a Poisson arrival distribution configured to generate the required network load. 1RMA slows the smaller ops minimally, compared to baselines. Even at moderate 50% load (i.e., 50 Gbps for 1RMA and 20 Gbps for the baseline), 1RMA median and tail slowdowns remain 6×-10× smaller than the other two baselines.

Software-defined resource allocation. 1RMA assigns finite hardware resources—command slots and memory regions—to applications according to their business priorities, a practice not possible with dynamically-allocated connections in RDMA. Figure 11b shows the effect: the transfer rate (measured within a rack) available to an application scales with the number of command slots it may allocate.

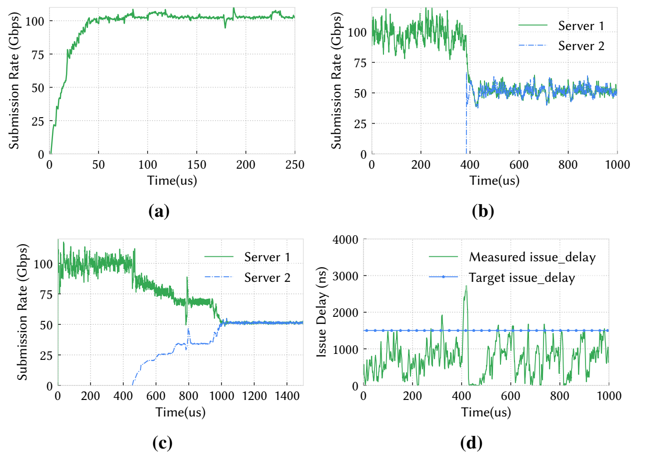

Figure 15: (a) Fast convergence of 1RMA CC - 0 to 100Gbps in 40us; (b) Fast convergence from single flow at full rate to two flows at fair-share rates; (c) Slower convergence when we configure 1RMA to not separate local congestion from remote; (d) `issue_delay` over time as reported by the NIC for (b).

7.3 1RMA Support for Encryption Key Rotation

We next consider the performance disruption of routine encryption key rotation, as in systems like KVCS. We arrange for a client process to continually probe (via RMA read ops) a server as the latter enacts an encryption key rotation. Ideally, performance and availability are impacted minimally. Figure 14 plots latency timelines under three rekeying design variants.

Standard RDMA provides no mechanism to rotate keys on an open connection; clients must switch connections to change keys. Consequently, upon server encryption key rotation, client connections break, typically via a timeout. Whereas connection re-establishment delay varies by implementation, it necessarily involves out-of-band communication, and disrupts the user (>2ms, bottommost line). In contrast, encryption key rotation via 1RMA's driver (topmost) reduces the unavailability period to only ~27μs. During this period, the client observes fast op failures with `REMOTE_AUTHENTICATION_FAILURE` outcome. Clients react by switching to new, pre-distributed keys. 1RMA's op independence property ensures rapid resumption of service. Compared to the driver-based mechanism, 1RMA's Rekey op (middle line) accomplishes the same rotation in a handful of hardware cycles ($< 1\mu$s) without needing to extend trust to the driver to manage the new key.

7.4 1RMA Congestion Control Efficacy

We now focus on 1RMA congestion control (CC). We use crafted workloads to answer the following questions: (a) How quickly does 1RMA respond (*e.g.*, saturate a 100Gbps network link for a point-to-point transfer)? (b) Do applications quickly reach a fair share of bandwidth? and (c) How important is response to both local and remote congestion?

Ramping to line rate. We inspect 1RMA's ramp-up behavior using a single client that initiates 4MB read transfers from a single server. Figure 15a plots the achieved op submission rate over time. With an idle RTT of 5μs in the testbed fabric, 1RMA is able to quickly converge to and maintain 100Gbps (with a CWND of 15 outstanding operations). Convergence is reached in ~8 RTTs, without incurring any non-`OK` outcomes.

Bandwidth Sharing. Using one client and two servers, each connected by a 100Gbps link, we initiate a single long-running transfer comprised of 4MB reads between the client and one server. At around T=400μs (Figure 15b), the client initiates another transfer with the second server, creating an incast condition at the client. 1RMA immediately detects the increase in local congestion, as signalled by a sharp increase in `issue_delay` (Figure 15d) and adjusts the local CWND appropriately, leading to a quick convergence (~5 RTTs) of both transfers to a fair share.

Local congestion reaction. To show the importance of reacting to local congestion specifically, we repeat the same experiment as above, but modify CC to react only to `total_delay`, plotted in Figure 15c. Failure to react specifically to local congestion leads to 20× slower convergence.

7.5 Benefits of Solicitation

To provide software control loops ample time to react, 1RMA's hardware-enforced solicitation mechanisms forestall drops and timeouts during transients (dynamic sudden changes). Remote NICs emit `NACK`s when inbound queues are full. This ensures that ops fail quickly under extreme congestion, rather than occupy scarce hardware resources for long durations.

Because software CC works well, pathological outcomes like `TIMEOUT` are rare in prior experiments. To study the benefits of solicitation and `NACK`s in 1RMA without reaction from CC, we evaluate 1RMA in hypothetical network conditions not reflected in our testbed using simulations. We study specifically the performance effects of RTT and jitter on solicitation. We also quantify the importance of `NACK`s.

Unless otherwise specified, read size is 4KB, RTT = 5μs, `DISPATCH_TIMEOUT` = 2 * RTT, and `TIMEOUT` = 4 * RTT.

1RMA's solicitation rules ensure that 1RMA does not initiate a transfer unless it is assured to land the data in the solicitation window. Crucially, the solicitation window size decides the number of outstanding ops (and thereby governs the achievable goodput). In an ideal network (no jitter or drops), sizing the window to the bandwidth delay product (BDP) would suffice. In practice, jitter is unavoidable. Figure 16a plots goodput as a function of jitter, for a point-to-point workload. The simulated client initiates ops at a static rate of 100Gbps. We plot three window sizes; 48KB (less than BDP), 64KB (equal to BDP), and 96KB (greater than BDP). Goodput drops with jitter, but larger-sized solicitation windows are generally more tolerant.

Critically, 1RMA's solicitation *sheds load eagerly rather than risking wasted bandwidth*. Figure 16b plots the rates at which ops are shed under varying network conditions, again with CC response disabled. `DISPATCH_TIMEOUT`—local load shedding—begins to manifest as the solicitation window is taxed by unpredictable RTT. When latency becomes unpredictable, some portion of ops time out, indicating that some of the sender's bandwidth was wasted (as all response bytes were sent, but late-arriving bytes are dropped). Importantly, solicitation and shedding work in concert to keep goodput high in all but the most pathological cases.

Similarly, `NACK`s defend servers and clients alike during sudden load shifts. First, by `NACK`ing inbound requests that are likely to

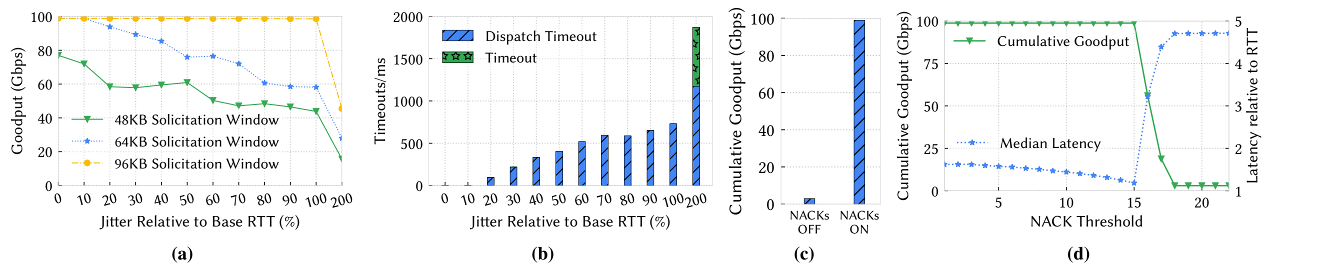

Figure 16: Impact of jitter and solicitation window size - (a) Goodput Vs. jitter for different window sizes and (b) Timeout rate for 64KB solicitation window; (c) Impact of NACKs on goodput; (d) NACK threshold tuning.

time out at the initiator, we conserve server bandwidth. Second, NACK eagerly refunds solicitation window capacity at the client, in 1 RTT instead of awaiting a TIMEOUT, allowing other ops to enter service. To illustrate, Figure 16d plots goodput as a function of the NACK threshold, wherein two clients each greedily demand 100Gbps from a single server (i.e., total demand 200Gbps), irrespective of congestion signals. The server can only provide 100Gbps. Such behavior can occur transiently when load shifts. When NACKs are not timely (i.e., the threshold is too high), goodput collapses, as the server's request queue grows without bound. To defend against this behavior, NACKs shed the portion of load that is not sustainable at the server-side. NACKed ops see higher apparent latency, because NACKed ops may be retried—thus, at lower NACK thresholds, more ops may see retries, leading to higher median latency but no loss of goodput.

8 Related Work

Several past works attempt to improve RDMA congestion control and loss recovery [23, 25, 27, 31, 41], both key issues that determine the performance of RDMA operations at scale. For example, DCQCN [41] implements a simple ECN-based congestion control protocol on the NIC. HPCC [25] relies on in-band telemetry to directly estimate the flow rate to use. RoGUE [23] is a software CC protocol that uses delay-based congestion window adaptation, while delegating loss recovery to the RNICs' existing strategies (i.e., go-back-N). IRN [31] (and, similarly, MELO [27]) advocates using selective ACKs in hardware (as opposed to go-back-N) for loss recovery; congestion control continues to rely on hardware-supported protocols such as DCQCN. While these techniques explore layering incremental software or hardware-based congestion control and loss recovery mechanisms on standard RDMA, 1RMA derives substantial benefits from completely refactoring the hardware-software division-of-labor.

iWARP [34] offloads all TCP stack functionality, including connection management, flow control, congestion control, loss recovery etc., to the NIC. This unfortunately makes the NIC design complex, and requires intricate translation between higher-level ops and NIC TCP actions.

All the above techniques are connection-oriented, and thus face at least some of the challenges of standard RDMA (§2).

To mitigate connection scalability issues, Mellanox has introduced DCT [11], which dynamically (and transparent to the applications) closes and opens connections to avoid queue pair exhaustion. DCT may cause latency increase due to frequent connection flips [21].

FaSST [21] and Scalable Connectionless RDMA [16] advocate using connectionless (UD) RDMA, while Homa [32] champions software-based solicitation and connectionless RPCs. Like Pony, such designs enable rapid evolution in software, but ultimately yield two-sided performance. Homa handles only the last-hop congestion, while [16, 21] rely on near-lossless fabrics based on PFC pause frames, leading to well-known issues [31, 41].

eRPC [18] is a fast RPC library designed for datacenter networks. Similar to [16, 21], eRPC uses the UD transport to mitigate connection scalability issues and is fundamentally two-sided. Thus, eRPC—like all RPC abstractions and Pony—involves software on both sides yielding two-sided performance. 1RMA instead focuses on one-sided primitives.

Recent efforts explore performance anomalies when multiple RDMA applications coexist and discover that they are caused by head-of-line blocking in RNICs [39, 40]. To address the performance anomalies, Justitia [40] adopts a software solution that uses shaping, rate limiting, and pacing at the senders. However, it can only provide latency guarantees in a best-effort manner as it enforces isolation via sharing incentive. Moreover, Justitia is still connection-oriented and does not address the broader set of issues of standard RDMA (§2).

9 Conclusion

This paper presents 1RMA, a ground-up rearchitecture of remote memory access aimed at multi-tenant datacenters and rooted in a principled division of labor between software and hardware. 1RMA's connection-free hardware treats each RMA operation independently; and aids software by offering fine-grained delay measurements and fast failure notifications. 1RMA software handles congestion control, and applications handle failure recovery and inter-operation ordering as needed. 1RMA's connection-free design supports confidentiality, authentication, and integrity at line rate with minimal performance/availability disruption for management actions such as encryption key rotation. This work does not raise any ethical issues.

Acknowledgments

We would like to thank Jeff Mogul, Nandita Dukkipati, Philip Wells, the anonymous SIGCOMM reviewers and our shepherd, Costin Raiciu, for providing valuable feedback. We thank the production, serving, and support teams at Google for their contributions to the work and the platform—including but not limited to the Pony Express and KVCS teams. We also gratefully thank our hardware development and verification teams at large.

References

[1] 2020. Alibaba Super Computing Cluster. https://www.alibabacloud.com/product/scc.

[2] 2020. Amazon Elastic Block Store. https://aws.amazon.com/ebs/.

[3] 2020. Amazon S3. https://aws.amazon.com/s3/.

[4] 2020. Microsoft Bing. https://www.bing.com/.

[5] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *Proceedings of the 2010 Conference of the ACM Special Interest Group on Data Communication*. 63–74.

[6] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks* 30 (1998), 107–117.

[7] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, and Leonidas Rigas. 2011. Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. 143–157.

[8] Wei Cao, Zhenjun Liu, Peng Wang, Sen Chen, Caifeng Zhu, Song Zheng, Yuhui Wang, and Guoqing Ma. 2018. PolarFS: An Ultra-low Latency and Failure Resilient Distributed File System for Shared Storage Cloud Database. *Proceedings of the Very Large Databases Endowment* 11, 12 (2018), 1849–1862.

[9] Youmin Chen, Youyou Lu, and Jiwu Shu. 2019. Scalable RDMA RPC on Reliable Connection with Efficient Resource Sharing. In *Proceedings of the Fourteenth European Conference on Computer Systems*. 1–14.

[10] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2013. Spanner: Google's Globally-Distributed Database. *ACM Transactions on Computer Systems* 31, 3 (2013), 1–22.

[11] Diego Crupnicoff, Michael Kagan, Ariel Shahar, Noam Block, and Hillel Chapman. 2012. Dynamically-Connected Transport Service. US Patent 8,213,315.

[12] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. 2014. FaRM: Fast Remote Memory. In *Proceedings of the Eleventh USENIX Symposium on Networked Systems Design and Implementation*. 401–414.

[13] Aleksandar Dragojević, Dushyanth Narayanan, Edmund B Nightingale, Matthew Renzelmann, Alex Shamis, Anirudh Badam, and Miguel Castro. 2015. No Compromises: Distributed Transactions with Consistency, Availability, and Performance. In *Proceedings of the Twenty-Fifth Symposium on Operating Systems Principles*. 54–70.

[14] Morris Dworkin. 2007. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. https://csrc.nist.gov/publications/detail/sp/800-38d/final.

[15] Franz Färber, Sang Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner. 2012. SAP HANA Database: Data Management for Modern Business Applications. *ACM Special Interest Group on Management of Data Record* 40, 4 (2012), 45–51.

[16] Ryan E Grant, Mohammad J Rashti, Pavan Balaji, and Ahmad Afsahi. 2015. Scalable Connectionless RDMA over Unreliable Datagrams. *Parallel Comput.* 48 (2015), 15–39.

[17] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over Commodity Ethernet at Scale. In *Proceedings of the 2016 Conference of the ACM Special Interest Group on Data Communication*. ACM, 202–215.

[18] Anuj Kalia, Michael Kaminsky, and David Andersen. 2019. Datacenter RPCs can be General and Fast. In *Proceeding of Sixteenth USENIX Symposium on Networked Systems Design and Implementation*. 1–16.

[19] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2014. Using RDMA Efficiently for Key-Value Services. In *Proceedings of the 2014 Conference of the ACM Special Interest Group on Data Communication*. 295–306.

[20] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2016. Design Guidelines for High Performance RDMA Systems. In *Proceedings of 2016 USENIX Annual Technical Conference*. 437–450.

[21] Anuj Kalia, Michael Kaminsky, and David G Andersen. 2016. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs. In *Proceedings of Twelfth USENIX Symposium on Operating Systems Design and Implementation*. 185–201.

[22] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Mike Ryan, David Wetherall, and Amin Vahdat. 2020. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *Proceedings of the 2020 Conference of the ACM Special Interest Group on Data Communication*.

[23] Yanfang Le, Brent Stephens, Arjun Singhvi, Aditya Akella, and Michael M Swift. 2018. RoGUE: RDMA over Generic Unconverged Ethernet. In *Proceedings of the ACM Symposium on Cloud Computing*. 225–236.

[24] Feifei Li. 2019. Cloud-Native Database Systems at Alibaba: Opportunities and Challenges. *Proceedings of the Very Large Databases Endowment* 12, 12 (2019), 2263–2272.

[25] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: High Precision Congestion Control. In *Proceedings of the 2019 Conference of the ACM Special Interest Group on Data Communication*. ACM, 44–58.

[26] Yuanwei Lu, Guo Chen, Bojie Li, Kun Tan, Yongqiang Xiong, Peng Cheng, Jiansong Zhang, Enhong Chen, and Thomas Moscibroda. 2018. Multi-Path Transport for RDMA in Datacenters. In *Proceedings of Fifteenth USENIX Symposium on Networked Systems Design and Implementation*. 357–371.

[27] Yuanwei Lu, Guo Chen, Zhenyuan Ruan, Wencong Xiao, Bojie Li, Jiansong Zhang, Yongqiang Xiong, Peng Cheng, and Enhong Chen. 2017. Memory Efficient Loss Recovery for Hardware-based Transport in Datacenter. In *Proceedings of the First Asia-Pacific Workshop on Networking*. 22–28.

[28] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Erik Rubow, Michael Ryan, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. 2019. SNAP: A Microkernel Approach to Host Networking. In *Proceedings of the Twenty-Seventh ACM Symposium on Operating Systems Principles*. 399–413.

[29] Christopher Mitchell, Yifeng Geng, and Jinyang Li. 2013. Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store. In *Proceedings of 2013 USENIX Annual Technical Conference*. 103–114.

[30] Radhika Mittal, Terry Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. In *Proceedings of the 2015 Conference of the ACM Special Interest Group on Data Communication*. 537–550.

[31] Radhika Mittal, Alexander Shpiner, Aurojit Panda, Eitan Zahavi, Arvind Krishnamurthy, Sylvia Ratnasamy, and Scott Shenker. 2018. Revisiting Network Support for RDMA. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 313–326.

[32] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. 2018. Homa: A Receiver-Driven Low-Latency Transport Protocol Using Network Priorities. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. Association for Computing Machinery, 221–235.

[33] Stanko Novakovic, Yizhou Shan, Aasheesh Kolli, Michael Cui, Yiying Zhang, Haggai Eran, Boris Pismenny, Liran Liss, Michael Wei, Dan Tsafrir, and Marcos Aguilera. 2019. Storm: A Fast Transactional Dataplane for Remote Data Structures. In *Proceedings of the Twelfth ACM International Conference on Systems and Storage*. 97–108.

[34] Renato Recio, Bernard Metzler, Paul Culley, Jeff Hilland, and Dave Garcia. 2007. A Remote Direct Memory Access Protocol Specification. RFC 5040.

[35] Steven L Scott. 1996. Synchronization and Communication in the T3E multiprocessor. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*. 26–36.

[36] Kai-Yeung Siu and Raj Jain. 1995. A Brief Overview of ATM: Protocol Layers, LAN Emulation, and Traffic Management. *ACM Special Interest Group on Data Communication Computer Communication Review* 25, 2 (1995), 6–20.

[37] Swaminathan Sivasubramanian. 2012. Amazon DynamoDB: A Seamlessly Scalable Non-Relational Database Service. In *Proceedings of the 2012 Conference of the ACM Special Interest Group on Management of Data*. 729–730.

[38] Brent Stephens, Alan L Cox, Ankit Singla, John Carter, Colin Dixon, and Wesley Felter. 2014. Practical DCB for improved data center networks. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 1824–1832.

[39] Yiwen Zhang, Juncheng Gu, Youngmoon Lee, Mosharaf Chowdhury, and Kang G Shin. 2017. Performance Isolation Anomalies in RDMA. In *Proceedings of the Workshop on Kernel-Bypass Networks*. 43–48.

[40] Yiwen Zhang, Yue Tan, Brent Stephens, and Mosharaf Chowdhury. 2019. RDMA Performance Isolation With Justitia. *arXiv preprint arXiv:1905.04437* (2019).

[41] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion Control for Large-Scale RDMA Deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 523–536.

Appendices

Appendices are supporting material that has not been peer-reviewed.

A Initialization Vectors for AES-GCM

AES-GCM requires a non-repeating Initialization Vector (IV) as further input to cryptographic operations, as AES-GCM is subject to attack when the combination of encryption key and IV repeats for distinct payloads. IVs are commonly derived from connection sequence numbers, which 1RMA lacks. However, unique IVs can nonetheless be provided by maintaining a counter, per 1RMA NIC, of all 1RMA protocol messages ever exchanged. Therefore, the combination of K_d+Counter+SenderAddress never repeats. We therefore use Counter+SenderAddress to seed the IV in request packets, which satisfies uniqueness for requests. Unlike sequence numbers, IVs need not be *contiguous*; only *uniqueness* is required.

When generating responses, serving 1RMA NICs also increment and include their own Counter value and RMA offset to further salt the IV, with the previous IV curried along as *additional authenticated data* (AAD), which ties all protocol messages together in sequence. The combination of currying and server-supplied re-seeding ensures that mutations and other 4-hop transactions are not vulnerable to replay attacks.

IVs and the various forms of AAD are memoized in command slot metadata, not readable by software, such that they remain in use for the duration of the command in question, and are then discarded.

B 1RMA Command Issue

Command slots correspond to a range of the 1RMA NIC's PCIe BAR, which is mapped into application virtual memory. Using memory-mapped registers in this fashion both simplifies and optimizes the hardware. To access these registers, applications use MMIO writes from the CPU to store commands directly into assigned slots, rather than relying on an on-NIC DMA-based command fetch mechanism. Conventional wisdom suggests that CPU-initiated writes across PCIe should be used sparingly because of their performance side effects. While true for non-write combining doorbells, we specifically architected 1RMA to leverage write-combining MMIO stores, which have significantly improved performance on modern CPUs over traditional doorbell writes. Such an operation is possible because 1RMA does not guarantee ordering between operations, and because we are willing to constrain software to issue commands using only carefully curated primitives (in our most performant library, four 16-byte SSE2 stores, in sequence). Our implementation achieves up to 87M commands/sec using eight Skylake CPU cores.

MMIO-based command issue also offers a latency benefit: commands are never fetched from host memory. Such fetches would incur a PCIe round-trip (hundreds of nanoseconds) on the critical path, which is a non-trivial latency adder [20].