Pythia: Remote Oracles for the Masses

Shin-Yeh Tsai, ***Purdue University;*** **Mathias Payer,** ***EPFL;*** **Yiying Zhang,** ***Purdue University***

https://www.usenix.org/conference/usenixsecurity19/presentation/tsai

This paper is included in the Proceedings of the 28th USENIX Security Symposium.

August 14–16, 2019 • Santa Clara, CA, USA

978-1-939133-06-9

Open access to the Proceedings of the 28th USENIX Security Symposium is sponsored by USENIX.

Pythia: Remote Oracles for the Masses

Shin-Yeh Tsai
Purdue University

Mathias Payer
EPFL

Yiying Zhang
Purdue University

Abstract

Remote Direct Memory Access (RDMA) is a technology that allows direct access from the network to a machine's main memory without involving its CPU. RDMA offers low-latency, high-bandwidth performance and low CPU utilization. While RDMA provides massive performance boosts and has thus been adopted by several major cloud providers, security concerns have so far been neglected.

The need for RDMA NICs to bypass CPU and directly access memory results in them storing various metadata like page table entries in their on-board SRAM. When the SRAM is full, RNICs swap metadata to main memory across the PCIe bus. We exploit the resulting timing difference to establish side channels and demonstrate that these side channels can leak access patterns of victim nodes to other nodes.

We design Pythia, a set of RDMA-based remote side-channel attacks that allow an attacker on one client machine to learn how victims on other client machines access data a server exports as an in-memory data service. We reverse engineer the memory architecture of the most widely used RDMA NIC and use this knowledge to improve the efficiency of Pythia. We further extend Pythia to build side-channel attacks on Crail, a real RDMA-based key-value store application. We evaluated Pythia on four different RDMA NICs both in a laboratory and in a public cloud setting. Pythia is fast (57 μs), accurate (97% accuracy), and can hide all its traces from the victim or the server.

1 Introduction

Direct Memory Access (DMA) allows a machine's peripherals like storage and network devices to access its main memory directly without involving CPU, vastly increasing I/O performance and reducing CPU utilization. Inspired by DMA, Remote Direct Memory Access, or *RDMA*, is a technology that allows remote hosts to directly access (exported) memory of a node without having to go through its CPU. RDMA enables high throughput and low latency data transfers and largely reduces CPU utilization in clusters.

In recent years, major cloud vendors like Microsoft Azure [73] and Alibaba Cloud [6] have adopted RDMA in their datacenters to speed up processing and to reduce cost of accessing large amounts of data. As the underlying protocols prosper [32], more and more servers leverage the RDMA protocol to speed up processing. The use of RDMA has revolutionized data sharing in cloud environments with implementations for efficient key-value stores [24, 25, 40, 56, 57], in-memory databases and transactional systems [19, 83, 88], and graph processing systems [68, 85].

There is a plethora of research work on RDMA, but the focus so far was all on performance, usability, and network protocols. Security has been largely overlooked in RDMA research and production[1]. With the rise of information leaks through memory-based side-channels [17,26,39,41,42,86,87], we have set out to evaluate the side-channel resistance of existing RDMA implementations.

In a scenario where multiple nodes connect to a server that provides remote access to its local memory through RDMA (*e.g.*, for a key-value store), a malicious node may want to learn what data was accessed by benign nodes. We assume that the server is trusted and that the attacker tries to learn what data was accessed by the victim nodes through a *remote* side channel which leaks access patterns. Figure 1 illustrates this environment.

We discovered a new side channel that prevails across all RDMA hardware that we know of. NICs that support RDMA, or *RNIC*s, cache metadata such as page table entries in their on-board SRAM so that they can perform all operations needed to access main memory on their own without involving CPU. However, the on-board SRAM size is limited and the RNIC can only cache hot metadata while leaving the rest in main memory. When an RDMA request's metadata is not cached, the RNIC takes extra time to fetch the metadata from main memory to its SRAM. We observe and characterize different side channels that can lead to this timing difference

[1] We made an initial exploration of security issues and opportunities in one-sided communication in a recent workshop [77].

on three generations of RNIC devices.

Based on our findings, we designed *Pythia*, a set of side-channel attacks that can be launched completely from the network through RDMA. The basic idea is to issue RDMA network requests to the server to fill its RNIC SRAM, eventually evicting the metadata of the target data. Then the attacker reloads the target data with an RDMA request and based on the time it takes, predict if the victim has accessed the data.

Although the basic idea is similar to the EVICT+RELOAD CPU cache side-channel attack [30], designing Pythia presents many new challenges. The first challenge is the difficulty in achieving good eviction performance. Existing CPU-cache based side-channel attacks leverage cache associativity to reduce the eviction set size, thereby improving eviction performance. However, RNICs are vendor owned and are complete black boxes to public knowledge. To confront this challenge, we reverse engineered the memory architecture of the Mellanox ConnectX-4 RNIC [48], the type of RNIC that is used in all major datacenter RDMA deployment. We successfully discovered the internal architectural organization of RNIC SRAM and leverage this knowledge to achieve low-latency eviction.

The second challenge is in the reload and prediction process. Because of our environment of being in a shared datacenter network, the latency of an RDMA request can vary with different network state. The traditional approach of using a static threshold to differentiate cache hit from cache miss is not a good fit for our environment. We take an adaptive approach to dynamically train a hit/miss classifier based on RDMA access latency at the time of attack and use the trained classifier to statistically predict victim accesses [22, 28].

We evaluated Pythia in our lab environment and in a public cloud [65] with four different types of RNICs. Pythia completes one EVICT+RELOAD cycle (across the network) in as low as 57 μs with 97% accuracy[2] Moreover, Pythia effectively hides its traces from the server and victims because it performs all its attack using RDMA operations from a separate machine.

We further built three variations of Pythia to attack a real RDMA-based system, the Apache Crail key-value store system [7, 70]. On a real application like Crail, it is more challenging to establish a strong side-channel attack because of limited application interface and noise coming from application performance overhead. After improving Pythia to accommodate these difficulties, we successfully launched a side-channel attack solely from a separate client machine using the unmodified Crail client interface. This attack is efficient and can accurately learn a victim's key-value pair access patterns.

The contributions of this paper are:

1. Discovery of new side channels in RDMA-based systems that leak client RDMA access patterns;

[2]The definition of accuracy throughout the paper is the percentage of successful guesses over total guesses.

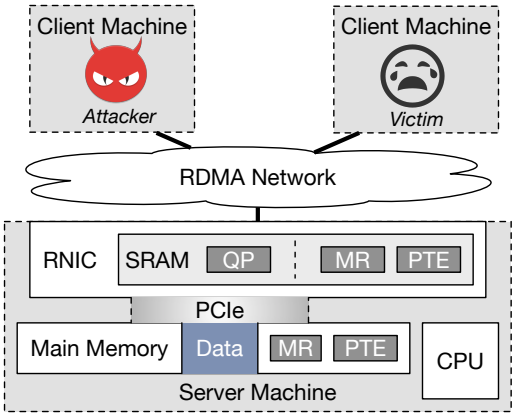

Figure 1: **Attack Environment and RNIC Architecture.** *The attacker and the victim are both clients that can access data in the server machine's memory throuth RDMA.*

2. Reverse engineering of the most widely used RNIC hardware architecture, which can be leveraged in designing efficient side channels;
3. Design, implementation, and evaluation of a set of Pythia side-channel attacks, which are fast, accurate, and can be launched solely from a separate machine across the network;
4. A case study of Pythia in a real-world setting;
5. Discussion of possible mitigations, most of which are uniquely applicable to RDMA systems.

Pythia is the first work that explores side-channel vulnerabilities in RDMA and exploits the vulnerabilities to launch attacks on RDMA-based datacenter systems. With today's datacenters all having robust defenses against direct sniffing or hijacking of network traffic, side channels are more feasible attack mechanisms and we believe that our work raises serious security concerns in a young but already widely-adopted network technology.

We have responsibly disclosed the weaknesses to Mellanox and Crail. Our implementation of Pythia is publicly available at https://github.com/Wuklab/Pythia.

2 Background on RDMA

2.1 RDMA Basics

Remote Direct Memory Access, or RDMA, is a network technology designed to offer remote low-latency, low-CPU-utilization access to exported memory regions. RDMA supports both *one-sided* and *two-sided* communication. One-sided RDMA operations directly access memory at a remote node without involving the remote node's CPU, similar to DMA on a single machine. Two-sided RDMA operations involve both sender and receiver processing, similar to `send`/`recv` in traditional network messaging.

RDMA improves performance along several dimensions. First, one-sided RDMA requests bypass the CPU of the receiver. Second, applications issue RDMA requests directly from user space, bypassing kernel and avoiding kernel trap cost. Third, RDMA avoids memory copying (a technique called *zero-copy*). As a result, RDMA achieves low-latency, high-throughput performance.

There are three implementations of RDMA: InfiniBand (IB) [10, 11], RoCE [8, 9], and iWARP [63]. All implementations follow the standard RDMA protocol [64]. Among them, RoCE, or *RDMA over Converged Ethernet*, implements the RDMA protocol over standard Ethernet (RoCEv1) and UDP (RoCEv2), and is the preferred technology in existing datacenters [32].

One-sided RDMA is the key area where significant performance and CPU utilization improvements over other network technologies happen. Thus, we focus on one-sided RDMA. To perform a one-sided RDMA operation, an application process at a receiver node needs to first allocate a consecutive virtual memory space and then use the virtual memory address range to register a *memory region*, or *MR*, with the RNIC. An application can register multiple MRs over the same or different memory spaces. The RNIC will assign a pair of local and remote protection keys (called *lkey* and *rkey*) to each MR. This application then conveys the virtual address of the MR and its rkey to processes running on other nodes. After building connections between these other nodes (senders) and the node that the MR-registering application runs on (receiver), these processes can use 1) a virtual memory address that falls in the MR's virtual memory address range, 2) a size, and 3) the rkey of the MR to perform one-sided RDMA `read` and `write`. In RDMA's term, a connection is called a *Queue Pair*, or *QP*.

2.2 RDMA NICs

RDMA NICs, or *RNIC*s, are where most RDMA functionalities are implemented. They usually contain complex hardware logic that implements the RDMA protocol and some SRAM to store metadata, and they are often connected to the host's PCIe bus (allowing the card access to main memory through DMA). Because of the need to bypass the kernel and receiver's CPU, most RDMA functionalities and data structures have to be offloaded to the RNIC hardware.

An RNIC's on-board SRAM stores three types of metadata. First, it stores metadata for each QP in its memory. Second, it stores lkeys, rkeys, and virtual memory addresses for all registered MRs. Third, it caches page table entries (PTEs) for MRs to obtain the DMA address of an RDMA request from its virtual memory address. RNICs have a limited amount of on-board SRAM which can only hold metadata for hot data. When the SRAM is full, an RNIC will evict its cached metadata to the main memory on the host machine, and on a future access, fetch the evicted metadata from the host main memory back through the PCIe bus. The timing difference between an RDMA access whose metadata is in RNIC SRAM and one that is not is what we exploit in our side-channel attack. The SRAM architecture is vendor-specific and not disclosed or specified in the RDMA standard. We reverse engineer the SRAM architecture of the state-of-the-art RNIC in Section 4.4.

2.3 RDMA-Based Applications

RDMA was originally designed for high-performance computing environments, and it has been a popular choice of network system in these environments for the past two decades [34, 44, 54]. In recent years, major datacenters and public clouds adopted RDMA for its low CPU utilization and superior performance. For example, Microsoft Azure [73] and Alibaba [6] have deployed RDMA with RoCE at large, production scale.

Many datacenter systems and applications have been ported to or rebuilt with RDMA. These include in-memory key-value stores [7, 24, 25, 40, 56, 57, 70], in-memory databases and transactional systems [19, 83, 88], graph processing systems [68, 85], distributed machine learning systems [15], consensus implementations [60, 80], distributed non-volatile memory systems [45, 67, 93], and remote swap systems [5, 31]. Most of these applications use both one-sided and two-sided RDMA operations, with some being pure one-sided [14, 88]. Our work is applicable to all RDMA-based applications that use one-sided RDMA (but not necessary purely one-sided).

3 Threat Model

In our attack, there are three parties: the server which hosts data in its main memory for other client machines to access (*e.g.*, an in-memory database or an in-memory key-value store), the victim who accesses the server's in-memory data through RDMA, and the attacker who tries to infer the victim's accesses and access patterns. The attacker and the victim are both normal clients that can access the data store service the server provides, and they run on separate machines. Following the threat models of related work that introduces and evaluates side channels, we assume that the attacker does not have direct control over the victim. As victim and attacker execute on different machines and communication to the server happens through the network, we assume that the attacker *cannot* observe the victim's network packets (as otherwise, the attacker could directly infer the accessed addresses and values as RDMA is currently not encrypted). This assumption is reasonable as sniffing victim's packets would require an attacker to have `root` access on either the victim's machine or the server's machine [52, 72, 74] or to launch man-in-the-middle attack to the network, both of which are well defended in cloud datacenters. We also assume that the server cannot *directly* observe memory accesses of either the victim or the attacker as both victim and attacker interact with the server through one-sided RDMA operations, not involving the server's CPU.

4 Side-Channel Attacks on RDMA

RDMA exposes node-internal memory to external hosts. Due to best practices of optimizing accesses and caching, current RDMA hardware is vulnerable to a variety of timing side channels. We present an overview of RDMA-based side channels, two basic attacks, refined attacks with our reverse engineered knowledge of RNIC internals, and evaluation results of the attacks. Unless otherwise stated, all our experiments use three machines, each equipped with a Mellanox ConnectX-4 100 Gbps network adapter [48], two Intel Xeon E5-2620 2.40GHz CPUs, and 128 GB main memory. They are connected with a Mellanox SB7700 100 Gbps InfiniBand switch [50]. One machine is used as the server that serves in-memory data through RDMA. The other two machines are the victim client machine and the attacker client machine, both of which can perform RDMA operations to access data on the server. In all the experiments in this section, the victim has a 50% chance of accessing the targeted data that the attacker tries to infer accesses on.

4.1 Attack Overview

The basic idea of our side-channel attacks is to exploit two weaknesses in RNIC: 1) the RNIC caches metadata in its SRAM, RDMA accesses whose metadata is not in SRAM must wait until that data is fetched from main memory, and 2) all RDMA accesses from all applications share the RNIC SRAM. As explained in Section 2.2, RNICs store three types of metadata in their SRAM: QP information, MR information, and PTEs. An RDMA access involves all three types of metadata: upon receiving a network request, an RNIC needs to locate which QP the request belongs to, which MR it falls into, and which page it is accessing. If any of these metadata is not in the RNIC SRAM, the RNIC will fetch it from the host memory, stalling the request until the required data arrives. By exploiting this timing difference, we can launch side-channel attacks to know which QP, which MR, and which PTE the victim has accessed.

Traditional CPU-cache-based side-channel attacks take three major forms: *prime*-based (*e.g.*, PRIME+PROBE [1–4, 42, 59, 75, 90]), *flush*-based (*e.g.*, FLUSH+RELOAD [86]), and *evict*-based (*e.g.*, EVICT+RELOAD [30]). Because RNICs do not provide any interface to flush their SRAM, all flush-based side-channel attacks such as FLUSH+RELOAD [86] and FLUSH+FLUSH [29] are incompatible with RDMA. All the operations that are needed in prime-based and evict-based attacks can be implemented through RDMA network requests that an attacker performs over the network. Attackers can hide their traces during the attacks, since one-sided RDMA reads are oblivious to the server or other clients. Hardware performance counters [35] may help servers track DMA traffic, but it is challenging to associate traffic with RDMA accesses or attacks. Even if the server suspects that an attack is happening, it is still hard, in practice, to attribute an attack based on traffic counters.

For the rest of the paper, we focus on RDMA-based EVICT+RELOAD attacks. PRIME+PROBE attacks are also possible on RDMA and we briefly discuss them in Section 7.

4.2 Unique Advantages and Challenges

There are three unique advantages for attackers in RDMA systems. First, RDMA's one-sided communication pattern allows the attacker to hide her traces, since the receiving node is unaware of any one-sided accesses. Second, the RDMA network is much faster both in latency and in bandwidth than traditional datacenter networks. The latest generation of RDMA switches and RNICs can sustain 200 Gbps bandwidth and under 0.6 μs latency [53]. RDMA's superior performance enables fine-grained, high-throughput, timing-based side-channel attacks over the network. Finally, RDMA's one-sided communication bypasses the sender's OS and does not involve CPU at the receiver, both of which help reduce disturbance to timing-based attacks.

At the same time, attacking the RNIC presents several novel challenges that no CPU-cache-based side-channel attack experiences. First, it is hard to discover efficient side channels in RNIC hardware. Unlike CPU caches, there is no public knowledge of how RNICs organize or use their SRAM. RNICs store different types of information in SRAM compared to a linear layout of CPU caches. Second, we set a strict threat model where attacks are launched from a separate machine that is different from the victim's machine and the server machine. This goal means that attacks have to be performed using RDMA network requests only. Finally, since our side channels are established over the network, noise in the network could potentially increase difficulties for timing-based attacks.

4.3 Basic Attack

Before presenting our side-channel attacks, we first discuss the type of victim information we choose as our attack target and the type of metadata we use to perform the eviction phase. Notice that these two dimensions are orthogonal and both have three options: QP, MR, and PTE.

Among these three types of information, knowing which QP the victim accesses leaks little information about the victim and usually is not useful in real attacks. MRs and PTEs can both leak more information. Using PTE as the attack target unit will reveal memory page (in virtual memory) accesses. All OSes use 4 KB as the default page size. The MR size is decided by the application that creates and registers it. For performance reasons [55], most RDMA-based applications choose to use large MRs. Thus, we choose PTE as the target of our attack. However, most of our ideas and techniques can be used to perform attacks that target MRs.

After choosing the attack target, we must decide what metadata to use to evict RNIC SRAM. To answer this question, we tried to evict SRAM using the three types of metadata and

Algorithm 1: MR-based eviction

```
Input  : a target victim virtual memory address
Output :

if No access to sufficient amount of MRs then
    start process at server to create MR_set;
else
    foreach MR that the attacker has access to do
        if MR ≠ victim's MR then
            insert MR into MR_set;
        end
    end
end

foreach MR in MR_set do
    perform 8-byte RDMA-read to MR;
end
```

Algorithm 2: Naive PTE-based eviction

```
Input  : a target victim virtual memory address
Output :

VictimVPN ← victim_address >> 12;
generate eviction_set using VictimVPN;

foreach VPN in eviction_set do
    perform 8-byte RDMA-read to address VPN << 12;
end
```

Algorithm 3: Reload and predict

```
Input  : a target victim virtual memory address
Output : prediction of if the victim has accessed the target
         address

determine Threshold according to network status;

start timer;
RDMA-read victim_address;
end timer;
time ← elapsed_time;

if time < Threshold then
    output accessed;
else
    output not_accessed;
end
```

reload our targeted information (*i.e.*, PTE). We can successfully evict a PTE with PTEs, no matter whether or not the PTEs we use to evict belong to the same MR as the target PTE. We can also evict an MR with other MRs. We further find that when an MR is evicted, PTEs of all the pages belonging to this MR will also be evicted. But we can only use QPs to evict QP. This behavior implies that RNICs isolates the SRAM used for QPs and for MRs and PTEs. We present the evaluation results in Section 4.5. From this initial test, we discovered that we can evict a PTE by either evicting the MR it belongs to (using a large number of MRs) or by evicting the PTE directly using a large number of other PTEs.

4.3.1 Eviction by MRs

We now present our attack that evicts SRAM with MRs, *PythiaMR*. Algorithm 1 presents the pseudocode of *PythiaMR*.

Because the MR-based attack requires the attacker to use many MRs to evict the server's RNIC SRAM, the attacker requires access to a sufficient amount of MRs. If the number of MRs is restricted, the attacker may resort to a (hypothetical) MR gadget that allows her to register multiple MRs. One approach is to launch a process on the server that allows her to register multiple MRs (see Section 5.1 for details). Since RDMA provides the functionality of registering multiple MRs with the same memory space, the attacker process at the server only needs to allocate a small memory space (of arbitrary size) and register it multiple times. This process then needs to send the rkeys corresponding to these registered MRs to the attacker running on a client machine.

In the eviction phase, the attacker performs one-sided RDMA reads from a client machine to the MRs it has access to at the server (except for the MR that the victim PTE is in). Since the server's RNIC needs to fetch and store metadata for each MR when the MR is accessed, its SRAM will eventually be filled with MR metadata that the attacker accessed.

4.3.2 Eviction by PTEs

Alternatively, we can use PTEs to establish a side channel. Compared to MR-based attacks, PTE-based attacks can often be performed entirely from a client machine. Algorithm 2 presents the pseudocode of our PTE-based attack.

To perform PTE-based eviction, the attacker issues one-sided RDMA reads to a sufficiently big memory space (1 GB to 4 GB for the RNICs we study). Most RDMA-based applications are services that provide in-memory data storage and use a large amount of memory, thus meeting the requirements of PTE-based attacks.

Accessing different memory pages will cause the RNIC to fetch PTEs to its SRAM. Because all accesses to the same memory page will hit the same PTE, we only need to perform one RDMA read (with the smallest RDMA operation size of 8 bytes) in every 4 KB virtual memory address range. To avoid loading the PTE that the victim accesses, we skip the memory addresses that are close to the victim address.

4.3.3 Reload and Predict

After the eviction phase (either by MRs or by PTEs), the attacker reloads the targeted victim data. If the reload time is smaller than a threshold, the attacker determines that the data has been accessed by the victim. Algorithm 3 presents the pseudocode of the reload and prediction process.

The threshold used at the reload time directly affects the result of an attack. Different from traditional CPU-cache side-

channel attacks, our attacks are in a distributed environment where network status can vary with other workloads in the datacenter. Thus, instead of a fixed threshold, we adapt the threshold dynamically according to the network status. To adjust the threshold, the attacker periodically measures the latency of an RDMA operation which hits the RNIC SRAM and the latency of one that misses the SRAM. The threshold can be set as a value in the middle of these two latencies.

In addition to using an average threshold, other more advanced methods can also be used to determine the reload result. In fact, we design a statistical method to determine the reload result for our real-application attacks, as will be presented in Section 5.

4.4 Finding PTE Eviction Sets

We call the attack that uses the eviction phase presented in Section 4.3.2 the basic PTE-based attack, or $PythiaPTE_{Basic}$. In order to reduce the time to perform eviction and improve the efficiency of $PythiaPTE_{Basic}$, we search for a smaller eviction set that achieves similar accuracy as $PythiaPTE_{Basic}$. Specifically, we perform a set of experiments to systematically reverse engineer the internal organization of RNIC SRAM and use our learned knowledge to construct a minimal PTE eviction set.

Reverse engineering RNIC SRAM organization is significantly harder than reverse engineering traditional CPU caches because there is no public knowledge of the internal organization of any RNIC. All we know is that the RNIC caches three types of metadata (PTEs, MRs, and QPs) in its SRAM. Moreover, reverse engineering the RNIC involves network operations which add noise compared to a well-isolated CPU cache environment.

First attempt in finding index bits. We initially guess that RNICs organize their in-SRAM PTE caches as set-associative caches, similar to how CPUs organize their caches. To validate this guess, we assume that the PTE cache is organized as a fixed number of sets (*e.g.*, 2, 4, 8) and different number of bits (*e.g.*, 1, 2, 3) are used to calculate the index into these sets. Since PTEs are identified by virtual page number (*VPN*), we can ignore the lowest 12 bits (with page size being 4 KB). We then use the lowest K bits of *VPN*s to calculate the index into one of the 2^K cache sets. These K bits correspond to the 12th to the $(11+K)$th bits of the full virtual memory address (we call the lowest bit the 0th bit and count upwards to higher bits).

We call the *VPN* of a victim memory address *VictimVPN*. The eviction set of a *VictimVPN* is formed by setting the same K bits as the *VictimVPN* and varying other bits in *VPN*s. To put it another way, for every 2^K pages, we pick one *VPN* to add to the eviction set. We keep adding distinct *VPN*s in this way until the number of *VPN*s in the set reaches an *eviction set size*. Algorithm 4 presents the pseudocode of how we form an eviction set. This is our straw-man approach and we call the PTE-based attack that uses this approach of forming eviction sets $PythiaPTE_{Straw}$.

Algorithm 4: Forming Eviction Set - Strawman

Input : *VictimVPN*, eviction set size, num of index bits
Output: an eviction set targeting *VictimVPN*

```
eviction_set ← {};
mask = VictimVPN & (1 << num_index_bits − 1);
for i = 0 to evict_set_size do
    VPN ← i << num_index_bits + mask;
    if VPN ≠ VictimVPN then
        insert VPN into eviction_set;
    end
end
output eviction_set;
```

Figure 2 plots the attack accuracy when we vary K from 0 to 15. We use two groups of attacks on *VictimVPN* 0. In the first group (the solid line), each eviction set has $2^7 = 128$ operations. The accuracy keeps improving until K is 13 and then flattens out. This result hints at the possibility of the RNIC using a set-associative cache with $2^{13} = 8192$ sets. It is because when K is smaller than 13, part of the operations in the eviction set will fall into a different cache set as the *VictimVPN*'s set, making the eviction set too small to evict the whole victim's cache set.

To verify this guess, we perform a second group of attacks (the dashed line). In this group, we double the eviction set size every time when we decrease K by 1. For example, we set the eviction set size to 128 when K is 13 and to 256 when K is 12. If the RNIC uses 8192 cache sets, then when K is 13, all of the eviction set will fall into the *VictimVPN*'s set, and when K is 12, half of the eviction set will fall into the *VictimVPN*'s set. These two attacks will then have the same effect in evicting the *VictimVPN*. Our results confirm this assumption. When K is less than 13, the attack accuracy is similar to when K is 13. However, when K is larger than 14, the accuracy drops. This is because we only use 64 and 32 eviction set size when K is 14 and 15, and these eviction sets are not large enough to evict a whole cache set.

From this set of experiments, we suspect that virtual memory address bits 12 to 24 are used in calculating the PTE cache set index and that each PTE cache set has 128 entries (*i.e.*, a 128-way cache).

Discovering prefetching behavior. Our guess above uses 13 bits as the index into the PTE cache and assumes that the PTE cache has 8192 sets. If this guess is correct, then an eviction set whose (*VPN*%8192) is different from (*VictimVPN*%8192) should fall completely into a different cache set from the victim's. To verify this assumption, we perform another set of attacks to the *VictimVPN* 0. In the nth attack, we construct its eviction set using *VPN*s where (*VPN*%8192 $= n$), and we change n from 0 to 8191.

Figure 3 plots the accuracy of the first 64 attacks (the rest of

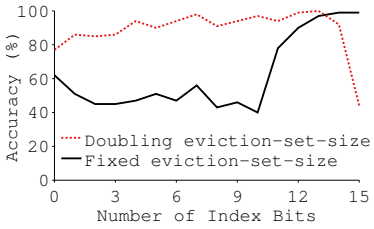

Figure 2: **Effect of Number of Index Bits.**

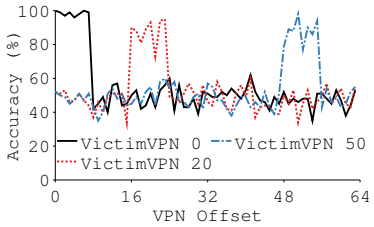

Figure 3: **Effect of Eviction Set Offset.** *X axis represents the first VPN in an eviction set (i.e., the "offset" of an eviction set").*

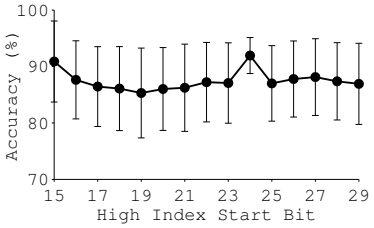

Figure 4: **Effect of Secondary Index.** *Error bars show the standard deviation across 1000 VictimVPNs.*

the attacks have the same pattern and we omit them from the figure). The black solid line shows the result of *VictimVPN* 0. Surprisingly, not only the 0th attack ($VPN\%8192 = 0$) has high accuracy, but also the 1st to the 7th attacks. To further understand this effect, we perform the same set of attacks on another two *VictimVPN*s, 20 and 50. With these *VictimVPN*s, the accuracy is high from 16th to 23rd attacks and 48th to 55th attacks respectively. These results imply that evicting any *VPN* within an eight-*VPN* range of ($VictimVPN - VictimVPN\%8$) to ($VictimVPN - VictimVPN\%8 + 7$) has the same effect.

We suspect that the RNIC prefetches eight *VPN*s at a time. This finding implies that instead of using 13 bits as cache index bits as in $PythiaPTE_{Straw}$, only the higher 10 bits (bits [15:24]) are used for index and the lower 3 bits (bits [12:14]) are used for prefetching. The RNIC cache thus only has $2^{10} = 1024$ sets.

Discovering secondary index. Our attack strategies so far work well ($> 90\%$ accuracy) with the *VictimVPN*s we tested (*e.g.*, 0, 20, 50). However, when we test the same attack strategy on some other *VictimVPN*s (*e.g.*, 6195, 30950), the accuracy can sometimes drop to around 75%. A dropped accuracy means that our eviction set cannot evict the *VictimVPN*, *i.e.*, the *VictimVPN* is in another cache set whose index is not the same as the index calculated using bits [15:24].

We thus suspect that there exists another 10 bits that are used to calculate where out of the 1024 sets a *VictimVPN* can fall into. To answer this question, we use a moving window of 10 bits, from bits [15:24] to bits [29:48], in the victim's virtual memory address. We randomly pick 1000 *VictimVPN*s and attack each of them by forming an eviction set with an index calculated with the moving window of 10 bits and an index calculated with bits [15:24] (64 operations under each index). We use two indices in this experiment because our alternative experiment of using just the index calculated by the moving window does not yield good accuracy. Figure 4 plots the average attack accuracy across 1000 *VictimVPN*s and their standard deviation (in error bars). Bits [24:33] yields the best average accuracy and smallest deviation. Thus, we believe that these bits are used as a second index into the PTE cache. Note that when the moving window is bits [15:24], the accuracy deviation is high, indicating that using bits [15:24] alone is not good enough.

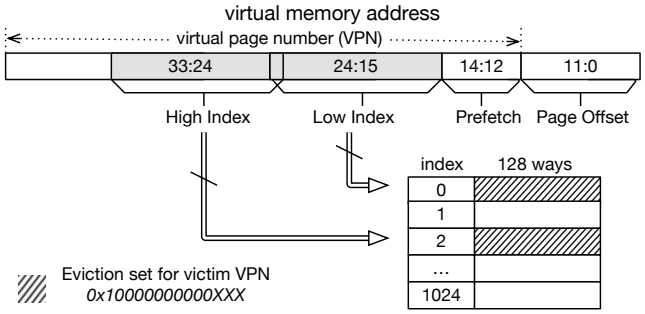

Figure 5: **Reverse-Engineered PTE Cache Organization.**

Complete algorithm. Figure 5 presents the final PTE cache architecture we speculate RNICs use based on our reverse engineering results. The PTE cache has 1024 sets and 128 ways. A PTE can be cached at one of the two cache sets. Two groups of bits are used to calculate the index of these two cache sets. The first group is bits [15:24], and the second group is bits [24:33]. We call them *low index bits* and *high index bits*. From our observation, both the high and the low index bits can decide which cache set will be used to cache a PTE. A PTE will be cached in either the cache set calculated by the high index bits or the cache set indicated by the low index bits. Every time when a PTE is accessed, its neighboring PTEs will also be fetched to the same set and bits [12:14] determine the 8 PTEs that will be prefetched.

Based on this reverse-engineered architecture, we present the final PTE-based EVICT+RELOAD attack, $PythiaPTE_{Full}$. We form half of the eviction set of a *VictimVPN* with *VPN*s that have the same low index bits as the *VictimVPN* and another half with *VPN*s that have the same high index bits as the *VictimVPN*. Algorithm 5 presents the complete algorithm.

4.5 Evaluation Results

We now present our evaluation results with the attacks described above.

4.5.1 Isolated Environment

We performed a set of experiments with three machines as described in the beginning of this section in our lab environment

Algorithm 5: Forming Eviction Set - Full

Input : *VictimVPN*, eviction set size
Output : an eviction set targeting the victim virtual memory address

eviction_set ← {}; *prefetch_bits* ← 3; *index_bits* ← 10; *high_index_start* ← 12;
low_index ← (*VictimVPN* >> *prefetch_bits*)&(1 << *index_bits* − 1); *mask_low* ← *low_index* << *prefetch_bits*;
high_index ← (*VictimVPN* >> *high_index_start*)&(1 << *index_bits* − 1); *mask_high* ← *high_index* << *prefetch_bits*;

for *i* = 0 *to evict_set_size*/2 **do**
    VPN ← *i* << (*index_bits* + *prefetch_bits*) + *mask_low*;
    if *VPN* ≠ *VictimVPN* **then**
        insert *VPN* into *eviction_set*;
    end
end

for *i* = 0 *to evict_set_size*/2 **do**
    VPN ← *i* << (*index_bits* + *high_index_start*) + *mask_high*;
    if *VPN* ≠ *VictimVPN* **then**
        insert *VPN* into *eviction_set*;
    end
end

output *eviction_set*;

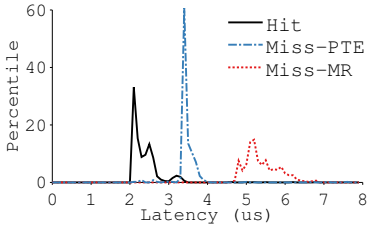

Figure 6: **Timing Differences.** *Each line presents the timing differences of each case over 1000 trials.*

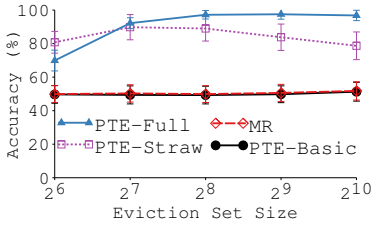

Figure 7: **Accuracy of Attacks.** *Error bars show the standard deviation across 1000 VictimVPNs.*

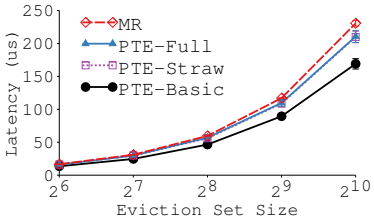

Figure 8: **Latency of Attacks.**

without any other network traffic.

Timing differences. Our side channels are based on timing differences between consecutive loads of the same memory address. To measure miss latency, we evict the RNIC SRAM using either MRs or PTEs and then issue an RDMA operation. To measure hit latency, we simply repeatedly issue the same RDMA operation. The measurements in Figure 6 show a clear timing difference between hit and miss latency. When we use MRs to evict SRAM, both the victim's MR and all the PTEs under this MR will be evicted. An RDMA operation afterwards will need to fetch both the PTE and the metadata for the MR containing the page from host main memory through the PCIe bus. On the other hand, using PTEs to evict will only evict the victim's PTE and reloading will only fetch the PTE. This explains why the miss latency of MR-based eviction is higher than that of PTE-based eviction.

Attack accuracy and latency. Figures 7 and 8 plot the accuracy and latency of four attack strategies: *PythiaMR*, $PythiaPTE_{Basic}$, $PythiaPTE_{Straw}$, and $PythiaPTE_{Full}$, as we change the eviction set size. As expected, with the same eviction set size, the time to perform these four attacks is similar, since they all use the same amount of RDMA operations. With bigger eviction sets, all attacks become slower.

$PythiaPTE_{Full}$'s accuracy is the highest: it can achieve 97% accuracy with only 57 μs per attack (when the eviction set size is 256). *PythiaMR* and $PythiaPTE_{Basic}$ have low accuracy, although we do observe *PythiaMR*'s accuracy improves significantly as the eviction set size increases (*PythiaMR*'s accuracy reaches 90% with 2^{15} eviction set size). This result demonstrates the benefit of using our reverse engineering findings.

Another observation is that the accuracy of $PythiaPTE_{Full}$ peaks when the eviction set size is 256 and remains the same when increasing the size further. This implies that the PTE cache has 128 ways, since we construct two cache sets with 256 entries in total.

Evaluation with different RNICs. All our experiments so far are performed with the Mellanox ConnectX-4 RNIC (most RDMA deployments in real datacenters use ConnectX-4 [32, 47]). We further validate our attacks on Mellanox ConnectX-5 [49] and ConnectX-3 [46] RNICs. ConnectX-5 is the latest generation of RNICs from Mellanox and ConnectX-

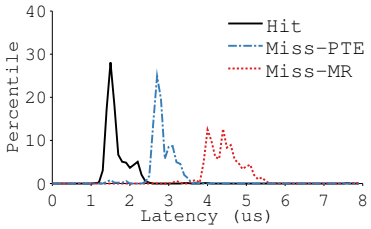

Figure 9: **Timing Differences in ConnectX-5.** *Each line presents the timing differences of each case over 1000 trials.*

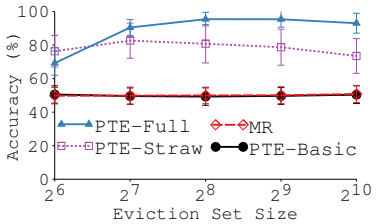

Figure 10: **Accuracy of Attacks in ConnectX-5.** *Error bars show the standard dev of 1000 VictimVPNs.*

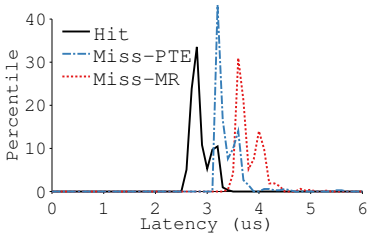

Figure 11: **Timing Differences in ConnectX-3.** *Each line presents the timing differences of each case over 1000 trials.*

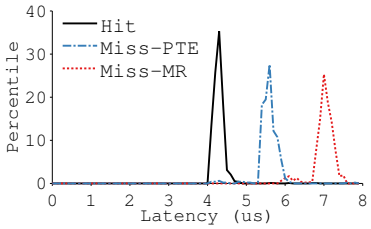

Figure 12: **Timing Differences in CloudLab.** *Each line presents the timing differences of each case over 1000 trials.*

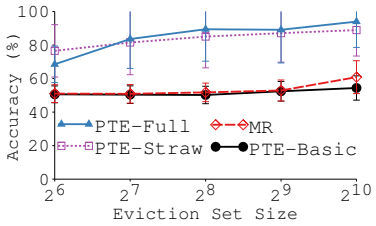

Figure 13: **Accuracy of Attacks in CloudLab.** *Error bars show the standard deviation across 1000 VictimVPNs.*

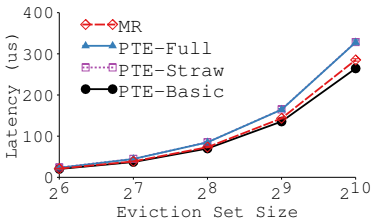

Figure 14: **Latency of Attacks in CloudLab.**

3 is the previous generation of ConnectX-4.

Figure 9 plots the timing results of SRAM hits and misses (due to eviction by MRs and by PTEs) on ConnectX-5. ConnectX-5's performance is better than ConnectX-4 on all cases. A clear timing difference between misses and hits remains, and misses caused by MR-based eviction are slower than by PTE-based eviction. Figure 10 plots the accuracy of the four types of attacks. The accuracy results are similar to ConnectX-4. Attack latency is also similar to ConnectX-4 and we omit the latency figure. Thus, we can confirm that ConnectX-5 uses a similar SRAM architecture as ConnectX-4, and it has the same side channels as ConnectX-4. We can launch the same attacks on ConnectX-5 with high accuracy and low latency.

We then perform the same set of experiments on ConnectX-3, see Figure 11. The hit latency with ConnectX-3 is longer than ConnectX-4. As hardware evolves, its internal performance often improves, which can explain why hit latency improves over generations of Mellanox RNICs. Surprisingly, the miss latency due to MR-based eviction is shorter on ConnectX-3 than on ConnectX-4 and ConnectX-5. Misses in RNIC SRAM involve the RNIC fetching metadata from the host main memory. We suspect the reason why miss performance drops in newer generations is because RNICs add more metadata for each data entry in newer generations, requiring longer time to fetch more metadata. As a result, the timing difference between miss and hit for ConnectX-3 is small.

Comparing ConnectX-3, ConnectX-4, and ConnectX-5, the three generations of RNICs from Mellanox, we found that as hardware RNICs evolve, their performance improves quickly, while the PCIe bus and host memory speed improve very slowly. As a result, the discrepancy between hit performance and miss performance becomes larger and we believe that this trend will continue in the future.

4.5.2 Public Cloud Environment

CloudLab [65] is a public cloud that has close to 15,000 cores distributed across three sites in the United States. We evaluated our attacks on a cluster that is connected with RoCE switches. Each machine in this cluster equips two Mellanox ConnectX-4 25 Gbps adapters. These RNICs are of the same product generation as our lab's ConnectX-4 RNICs, with the difference that our RNICs are 100 Gbps adapters. Both types of adapters can be configured for Ethernet (RoCE) and for InfiniBand. We configure ours for InfiniBand, and CloudLab's are configured for RoCE. Apart from RNIC differences, CloudLab is used concurrently by many different users; it has a more complex, hierarchical network topology; and it uses a RoCE network instead of InfiniBand. At the time of our test, 129 out of 199 physical machines in the cluster were in use.

We repeat the same set of experiments as Section 4.5.1. Similar to our lab's experiments, we use three machines, a server, a victim client, and an attacker client. Figure 12 plots the timing difference of RNIC SRAM hit and misses (due to MR-based eviction and PTE-based eviction). Similar to our isolated environment results, misses caused by MR-based eviction are slower than misses caused by PTE-based eviction, and both types of misses are slower than hits. In CloudLab's shared network and shared machine environments, the latencies of all accesses are longer than in our lab environment,

but the timing differences are still clear.

Figures 13 and 14 plot the accuracy and latency of the four types of attacks in CloudLab. Similar to the results in Figures 7 and 8, With the same eviction set size (and thus similar latency), *PythiaPTE*$_{Full}$ and *PythiaPTE*$_{Straw}$ have higher accuracy than *PythiaMR* and *PythiaPTE*$_{Basic}$. However, these attacks have larger variation in accuracy compared to attacks in our lab's environment because of the more dynamic environment in CloudLab.

5 Attacking Real RDMA-Based Systems

To demonstrate the feasibility of launching side-channel attacks on real RDMA-based applications, we design and perform a set of attacks on *Crail* [7, 70], an open-source RDMA-based key-value store written in Java. A Crail system consists of several roles: a server which stores key-value pairs, a namenode which stores metadata and manages the control path, and clients which issue key-value pair *get*s and *set*s to the server via a Crail-provided API. We install each component on a separate machine and connect all of them with RDMA. This section presents our design and evaluation of attacks on Crail.

5.1 Attacks

Based on the attack primitives described in Section 4, we designed three attacks on Crail. All these attacks have the same goal: knowing whether or not the victim Crail client accesses a specific key-value pair.

MR-based attack (*PythiaCrail*$_{MR}$). Our first attack uses MR-based eviction as described in Section 4.3.1. This attack requires three attacker processes. The first is a Crail client process (P_c). The second and the third processes run our attack code, with the second one running on the Crail server machine (P_s) and the third one running on any other machine (P_a) (it can be the same machine as the one where P_c runs). In the preparation phase, P_s registers a large number of MRs. In the eviction phase, P_a issues one-sided RDMA reads to these MRs. Finally, P_c performs a Crail *get* operation to reload the victim key-value pair. *PythiaCrail*$_{MR}$ requires P_s and P_a because MR-based attacks need to access many MRs but by default Crail only registers a small set of MRs.

PTE-based attack (*PythiaCrail*$_{PTE}$). The second attack uses PTE-based eviction. In this attack, we require three processes as in the MR-based attack: P_c, P_s, and P_a. In the preparation phase, we first use P_s to allocate a big chunk of memory and register it with an MR. In the eviction phase, P_a performs one-sided RDMA reads to different VPNs in the allocated memory space. Afterwards, P_c issues a Crail *get* request to the victim key-value pair.

Our reverse engineering results in Section 4.4 can be leveraged to reduce the eviction set size in *PythiaCrail*$_{PTE}$. However, to form the eviction set, we need to know the index of the SRAM cache set(s), which is calculated by the virtual memory address. Without modifying the source code of Crail which is written in Java, it is difficult to directly know the virtual memory address of a target key-value pair. Instead, we use a "learning" phase before launching the actual attack to determine the eviction set to use for a target key-value pair. Specifically, we let P_c access the target key-value pair and let P_a try all 1024 different cache sets for eviction. After 1024 trials, we pick the cache set that yields the best accuracy as our attack eviction set.

Client-only attack (*PythiaCrail*$_{Client}$). Our last attack on Crail is launched exclusively from a regular Crail client process and requires no other privileges or resources. The attacker (as a normal Crail client) issues Crail *get* requests to different key-value pairs during the eviction phase. After the eviction phase, it performs a Crail *get* operation to the victim key-value pair.

Our initial design of *PythiaCrail*$_{Client}$ randomly picks key-value pairs to access during the eviction phase. However, we soon discovered two issues with this naive approach. First, it needs a large number of key-value pairs to effectively evict the target key-value pair. Doing so not only makes the attack slow but also requires the Crail system under attack to already be storing many key-value pairs. Second, we found that the Crail system becomes slower and unstable as the server processes more client requests. We suspect this to be caused by Crail's own (memory) management overhead. Unstable access latency makes our timing-based attack harder and prohibits an accurate prediction during the reload phase. We improve our initial design with the following optimization. We selectively choose a small set of key-value pairs as the eviction set. We make the assumption that key-value pairs are sequentially allocated in chunks of memory and pick the pairs that are likely to be in the same RNIC SRAM cache set as the victim key-value pair. After reducing the eviction set size, our attack runs very fast. Instead of continuously launching the attack in loops, we add some sleep time between eviction and reloading so that we do not issue too many Crail requests to make Crail unstable.

Probabilistic prediction. Under real workloads and noisy network environments, we found that a simple threshold as used in Section 4.3 cannot accurately determine if the victim has accessed the target data. Thus, we use a more dynamic and adaptive approach to predict the outcome of the attack. Similar to the approach used in TLBleed [28], we perform a learning phase to train a classifier of operation latency with KNN [22] before the attack. We use the trained model to predict the probability of a reload latency implying a victim access (*i.e.*, a hit).

5.2 Results

We evaluated *PythiaCrail*$_{MR}$, *PythiaCrail*$_{PTE}$, and *PythiaCrail*$_{Client}$ using both controlled tests and work-

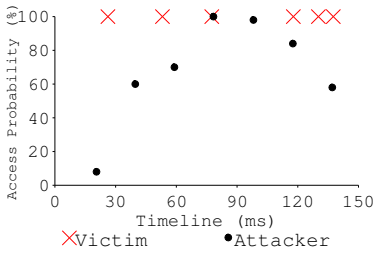

Figure 15: *PythiaCrail*$_{MR}$

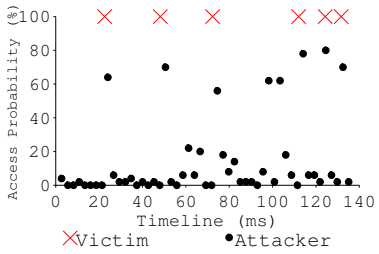

Figure 16: *PythiaCrail*$_{PTE}$

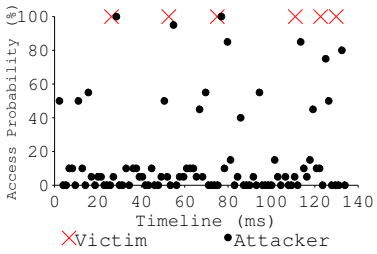

Figure 17: *PythiaCrail*$_{Client}$

loads that model real datacenter key-value stores.

5.2.1 Controlled Test

We first compare the latency of a Crail client key-value pair *get* operation that hits RNIC SRAM, a client *get* that misses RNIC SRAM after the eviction phase in *PythiaCrail*$_{MR}$, in *PythiaCrail*$_{PTE}$, and in *PythiaCrail*$_{Client}$. In these controlled tests, the victim client has a 50% chance of accessing the targeted key-value pair that the attacker tries to infer accesses on. Figure 18 plots these four types of latencies, each performing 1000 trials. All the three types of misses take longer than hits, with the timing difference of *PythiaCrail*$_{MR}$ the biggest and *PythiaCrail*$_{Client}$ the smallest. The timing difference implies that it is easiest to separate hits and misses with *PythiaCrail*$_{MR}$.

We launch the *PythiaCrail*$_{MR}$, and *PythiaCrail*$_{PTE}$, and *PythiaCrail*$_{Client}$ attacks by first performing their respective eviction phases. Next, we let victim access or not access the target key-value pair. Finally, we measure the time to reload the key-value pair and compare it with a threshold we determined from the timing difference testing phase. As expected, *PythiaCrail*$_{MR}$ gives the best accuracy. The accuracies of *PythiaCrail*$_{MR}$, *PythiaCrail*$_{PTE}$, and *PythiaCrail*$_{Client}$ are 96%, 85%, and 79% respectively, and the time to perform these attacks are 19*ms*, 0.1*ms*, and 0.3*ms*.

5.2.2 Macro-benchmark Results

Workloads. To evaluate how our attacks perform with real datacenter key-value store workloads, we construct a macro-benchmark with the Yahoo! Cloud Serving Benchmark (YCSB) [21] and statistics reported by Facebook in their production key-value store [12]. YCSB provides key-value *get/set* access pattern but no inter-arrival time between requests. Facebook provides the inter-arrival time of requests received at a server in its cluster, which includes requests from all the clients to this server. We set each key-value pair size to be 1 KB, the average key-value pair size reported by Facebook.

Attack environment setup. In this experiment, the victim (on a Crail client machine) executes our macro-benchmark to access key-value pairs on a Crail server machine using the Crail APIs. Since Facebook only provides aggregated request inter-arrival time across clients and does not reveal how many clients there are, we use one client machine to model the aggregated effect of all clients with the provided inter-arrival time. We run an attacker process as a normal Crail client on another machine. A fourth machine serves as the Crail namenode. While the victim process executes the macro-benchmark, we repeatedly perform *PythiaCrail*$_{Client}$, *PythiaCrail*$_{MR}$, or *PythiaCrail*$_{PTE}$ to detect if the victim accesses a target key-value pair.

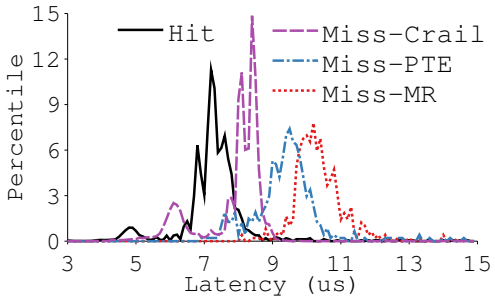

Figure 18: **Timing Difference in Crail.** *Each line presents the timing differences of each case over 1000 trials.*

Results. Figures 15, 16, and 17 present the timeline of the victim accessing the target key-value pair (red crosses) and the attacker's prediction (black dots with values as access probability). All three attacks can capture most if not all victim accesses. Among them, *PythiaCrail*$_{MR}$ is the worst in attack accuracy. This is because each attack in *PythiaCrail*$_{MR}$ takes 19*ms*, which is much longer than the Facebook inter-arrival time. As a result, *PythiaCrail*$_{MR}$ misses victim accesses that happen more frequent than its attack length. Both *PythiaCrail*$_{PTE}$ and *PythiaCrail*$_{Client}$ run very fast and capture all victim accesses. In fact, these two attacks run so fast that we add a sleep time of 1*ms* between evict and reload to avoid issuing too many Crail requests and making Crail's performance unstable. Comparing *PythiaCrail*$_{PTE}$ and *PythiaCrail*$_{Client}$, *PythiaCrail*$_{PTE}$'s predictions are of low access probabilities and *PythiaCrail*$_{Client}$ has more predictions of around 50% access probabilities. The attacker can set a threshold accordingly to determine the final set of victim accesses (*e.g.*, those with probabilities $>$ 60% for *PythiaCrail*$_{Client}$).

Overall, we believe *PythiaCrail*$_{Client}$ to be the most effec-

tive attack, since it predicts victim accesses with high confidence and it requires the least amount of attacker resources: *PythiaCrail*$_{Client}$ can be launched exclusively from a separate client machine through the unmodified Crail client interface. If attackers can run modified Crail clients, they can launch more efficient side-channel attacks by forming the eviction set with known virtual addresses.

6 Mitigation Techniques

Defending against RDMA-based side-channel attacks is possible and feasible. We discuss both mitigations for current hardware as well as those for future hardware.

Huge virtual memory page or no virtual memory. PTE-based attacks are only possible when RNICs cache PTEs and when the attacker can form an effective eviction set. One way to prevent PTE-based attacks is to force all RDMA registrations and operations to directly use physical memory addresses. When physical memory addresses are used, RDMA does not need to access or cache PTEs, thereby preventing PTE-based attacks. Registering physical memory addresses is a privileged operation that RNICs allow the kernel [76] and privileged users to perform [51]. However, using physical memory addresses loses all the benefits of virtual memory and introduces new security concerns.

Another method to defend against PTE-based attacks is to use huge memory pages [24]. Using huge pages (*e.g.*, 1 GB pages) introduces two types of difficulties for attackers. First, the attacker can only guess victim accesses at coarse granularity (*e.g.*, 1 GB). Second, the attacker will need to have access to a huge memory space to form an eviction set with enough PTEs.

Isolate server's resource. Our experience with Crail demonstrates that attacking Crail is difficult when the attacker can only use Crail's interface without the access to a large number of PTEs or MRs and without knowing Crail's data layout in the virtual memory address space. Our experiments show that for *PythiaCrail*$_{MR}$ and *PythiaCrail*$_{PTE}$ to work, an attacker needs to run a process, P_s, on the server machine. Otherwise, the attacker would not be able to launch those attacks (although *PythiaCrail*$_{Client}$ still works). Thus, a server that hosts RDMA service can prohibit normal users from running any processes to help defend against side-channel attacks. Various address randomization techniques can also complicate attacks.

Separate protection domains. When we disclosed the attacks in this paper to Mellanox, the Mellanox engineers stated that separating *Protection Domain*s (*PD*s) between different clients and connections can potentially mitigate the attacks. We evaluated this mitigation by moving the attacker to a different PD and found that doing so mitigates Pythia attacks. Unfortunately, all existing RDMA applications that we are aware of [7, 24, 83] use only one PD for higher performance. Using multiple PDs results in low throughput and high latency overhead (15% throughput reduction and 21% latency overhead with 256 PDs in our experiments). We plan to further investigate both attack and defense mechanisms when separating PDs across clients.

Introduce noise. Our side channels are established on timing differences at the microsecond or sub-microsecond level. Attacking Crail running real workloads is more difficult than attacking raw RDMA accesses mainly because of Crail's non-deterministic performance overhead. Therefore, an effective countermeasure is to introduce random latency overhead at an RDMA-based application or in the datacenter RDMA network, which, however, could impact application performance.

Detect and throttle attacker's network traffic. Our attackers can hide their attacks because one-sided RDMA operations are completely hidden from the receiver CPU (the server in our case). To detect these attacks, the server can deploy traffic sniffing tools to sniff all incoming RDMA network requests. If the sniffer detects heavy network activity from a client, it can raise a flag that this client may be malicious. If it further detects an access pattern that matches eviction sets described in Section 4.4, this client is more likely to be an attacker. This defense comes with the same drawbacks of other heuristic-based defenses that an attacker may stay under the detection threshold.

A further countermeasure is to throttle the maximum bandwidth allowed at every client. If an attacker cannot issue enough operations to evict RNIC SRAM, its attack accuracy will drop significantly. However, throttling client bandwidth can hurt normal clients' performance.

Better hardware design. All existing RNICs share their SRAM across all users and across all connections. Because of this, an attacker can evict a victim's PTE and MR even when the attacker and the victim have different connections to the server. If RNICs can partition their SRAM to different isolated domains for different connections, then attackers can never evict victim's PTEs or MRs. However, isolation resources at hardware level will inevitably hurt performance and increase hardware complexity, which gives little incentive for RNIC vendors to change their hardware design.

7 Discussion

We now briefly discuss the implications, impact, and limitations of Pythia, and some other attacks on RDMA that can be designed based on Pythia.

7.1 RDMA Vulnerabilities

We discovered new vulnerabilities in RDMA systems that are fundamental to the design of RDMA and not specific to just one RDMA device. RNICs cache metadata as a result

of RDMA's design philosophy of *one-sided* network communication. Because one-sided operations cannot involve host CPUs, RNICs have to handle and serve RDMA requests on their own, which involves accessing various types of metadata. With limited on-board SRAM, RNICs cannot store all the metadata and have to move metadata between their SRAM and the host machine's main memory through the PCIe bus. As a result, there exists timing difference between RDMA operations that hit or miss SRAM, and this timing difference keeps increasing as RNICs evolve over generations.

We demonstrated the feasibility of exploiting the above vulnerability to launch side-channel attacks on RDMA-based systems. Pythia attacks are fast and accurate, and they can be performed completely from the network. Moreover, attackers can hide their traces because the attack uses one-sided network requests.

Both the RNIC side channels we discovered and our attacks' unique advantages are fundamental to one-sided network communication. One-sided communication offers many performance and cost benefits that are attractive for datacenter systems. However, it also raises new security concerns [77], as we demonstrate in Pythia. Our work can inspire future security researchers in discovering and defending more vulnerabilities in RDMA.

7.2 Attacking Real Applications

We demonstrated that it is feasible to launch Pythia attacks on Crail, a real RDMA-based system developed by the Crail team. $PythiaCrail_{Client}$, the attack that is launched by performing Crail-provided client APIs only, can successfully infer victim's access patterns under real workloads.

We believe that Pythia can similarly attack other RDMA-based applications as well. Pythia only requires two features from an RDMA-based application: the application uses one-sided RDMA operations and allocates regular paged memory. Many applications meet these requirements, such as the NAM-DB RDMA-based in-memory database [88], the Pilaf RDMA-based key-value store [56], and the Wukong RDMA-based graph system [68]. Unfortunately, most of these systems are not available publicly.

7.3 Attack Limitations

Although our side-channel attacks are fast, accurate, and can be launched entirely from the network, they do have several limitations. First, the granularity of Pythia attacks (and therefore information leakage of accesses) is a memory page. Pythia currently cannot differentiate between two victim accesses that access the same target page. Second, Pythia can only predict if a data entity at the server has been accessed, but not which client machine(s) accessed it. Third, our MR-based attacks require access to a large number of MRs, and PTE-based attacks require access to large memory spaces. Finally, our attacks consume network bandwidth and can be detected by sniffing the network.

7.4 Other RDMA-Based Attacks

Pythia serves as a starting point for designing other types of RDMA-based attacks. For example, similar to Pythia EVICT+RELOAD attacks, it is possible to launch PRIME+PROBE attacks from the network by exploiting the MR or the PTE side channels.

The MR and the PTE side channels we established can also be used as covert channels. We implemented a naive covert-channel attack of using one EVICT+RELOAD cycle to transmit one bit and it could reach a sending rate of 20 Kbps with $PythiaPTE_{Full}$.

8 Related Work

Single-node side-channel attacks. In recent years, a host of attacks that exploit various hardware features to establish side channels have been proposed. CPU-cache-based side-channel attacks such as PRIME+PROBE [1–4, 42, 59, 75, 78, 90], EVICT+RELOAD [30], FLUSH+RELOAD [86], and FLUSH+FLUSH [30, 78] can leak victim's memory access patterns at fine granularity. CPU-cache-based side channels are also the key enabling factors in attacks like Meltdown [41], Spectre [39], and Foreshadow [17]. Other than CPU caches, TLB [28] or port contention [13] also expose hardware-based side channels. Side-channel attacks brought key concerns in cloud environments where one tenant can steal information of other tenants when they share the same physical resource [79, 89–91] or the same service [33]. But all these single-node side-channel attacks require the attacker to run on the same machine as the victim. Pythia is a remote side-channel attack that can be launched completely from a separate remote machine.

Remote side-channel attacks. Several remote side-channel attacks exploit TCP sequence numbers to hijack connections [18, 61, 62]. Another line of work relies on traffic analysis to exploit sensitive information [27, 37]. Brumley *et al.* perform a timing-based attack on OpenSSL's ladder implementation to obtain the private key of a TLS server [16]. Cock *et al.* present an empirical study of remote timing channels on microkernel [20]. NetSpectre [66] presents an access-driven remote EVICT+RELOAD cache attack. Weinberg *et al.* combined CSS and JavaScript to remotely sniff victims' browsing patterns [84]. Different from all previous remote side-channel attacks, Pythia targets the RDMA network. Moreover, Pythia exploits RNIC hardware features to establish timing-based side channels, while previous remote side channels exploit network protocols or software features. As far as we know, Throwhammer [71] is the only other attack related to RDMA. However, it simply uses RDMA network requests to launch a Rowhammer attack and does not explore or exploit vulnerabilities in the RDMA

technology itself.

Mitigations to side-channel attacks. Various defense mechanisms have been proposed to combat CPU cache side-channel attacks in both hardware [23, 36, 58, 81, 82] and software [38, 43, 69, 92, 94]. Unfortunately, none of the existing defense mechanisms can be directly applied to RDMA-based side-channel attacks. We propose a set of new mitigations that target RDMA-based side-channel attacks.

9 Conclusion

This paper presents Pythia, the first set of side-channel attacks on RDMA-based systems. We reverse engineer the internal data structures of current RDMA systems and leverage this information to improve our attack. Pythia can be launched completely from a normal client machine to steal access patterns of victims on other machines. We evaluate Pythia in laboratory settings to showcase the capabilities of the attack, on real software such as Crail, and on real data centers to show real-world impact. Pythia is fast, accurate, and can hide its trace from victims and the server.

Acknowledgments

We would like to thank the anonymous reviewers for their tremendous feedback and comments, which have substantially improved the content and presentation of this paper. We are also thankful to Ahmad Atamlh, Noam Bloch, Brandon Hathaway, Yuval Itkin, Ariel Levanon, Alex Polak, Randy Splinter, and Patrick Stuedi for their feedback during our responsible disclosure to Mellanox and the Crail team.

This material is based upon work supported by the National Science Foundation under the following grant: NSF 1719215. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or other institutions.

References

[1] Onur Aciiçmez. Yet another microarchitectural attack: Exploiting i-cache. In *Proceedings of the 2007 ACM Workshop on Computer Security Architecture (CSAW '07)*, Fairfax, VA, USA, November 2007.

[2] Onur Aciiçmez, Billy Bob Brumley, and Philipp Grabher. New results on instruction cache attacks. In *Proceedings of the 12th International Conference on Cryptographic Hardware and Embedded Systems (CHES '10)*, Santa Barbara, CA, USA, August 2010.

[3] Onur Aciiçmez, Çetin Kaya Koç, and Jean-Pierre Seifert. On the power of simple branch prediction analysis. In *Proceedings of the 2Nd ACM Symposium on Information, Computer and Communications Security (ASIACCS '07)*, Singapore, March 2007.

[4] Onur Aciiçmez and Werner Schindler. A vulnerability in rsa implementations due to instruction cache analysis and its demonstration on openssl. In *Proceedings of the 2008 The Cryptopgraphers' Track at the RSA Conference on Topics in Cryptology (CT-RSA '08)*, San Francisco, CA, USA, April 2008.

[5] Marcos K. Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Stanko Novakovic, Arun Ramanathan, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, Rajesh Venkatasubramanian, and Michael Wei. Remote regions: A simple abstraction for remote memory. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference (ATC '18)*, Boston, MA, USA, July 2018.

[6] Alibaba Cloud. Super computing cluster. https://www.alibabacloud.com/product/scc, 2018.

[7] Apache. Crail: High-performance distributed data store. https://crail.incubator.apache.org/, 2018.

[8] InfiniBand Trade Association. InfiniBand Architecture Annex A 16: RoCE. https://cw.infinibandta.org/document/dl/7148, April 2010.

[9] InfiniBand Trade Association. InfiniBand Architecture Annex A 16: RoCEv2. https://cw.infinibandta.org/document/dl/7148, September 2014.

[10] InfiniBand Trade Association. InfiniBand Architecture Volume 1 – Architecture Specification, Release 1.3. https://cw.infinibandta.org/document/dl/7859, March 2015.

[11] InfiniBand Trade Association. InfiniBand Architecture Volume 2 – Architecture Specification, Release 1.3.1. https://cw.infinibandta.org/document/dl/8125, November 2016.

[12] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. Workload Analysis of a Large-scale Key-value Store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '12)*, London, United Kingdom, June 2012.

[13] Atri Bhattacharyya, Alexandra Sandulescu, Matthias Neugschwandtner, Alessandro Sorniotti, Babak Falsafi, Mathias Payer, and Anil Kurmus. Smotherspectre: exploiting speculative execution through port contention. https://arxiv.org/abs/1903.01843, 2018.

[14] Carsten Binnig, Andrew Crotty, Alex Galakatos, Tim Kraska, and Erfan Zamanian. The End of Slow Networks: It's Time for a Redesign. *Proceedings of the VLDB Endowment*, 9(7):528–539, 2016.

[15] Rajarshi Biswas, Xiaoyi Lu, and Dhabaleswar Panda. Accelerating tensorflow with adaptive rdma-based grpc. In *25th IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC '18)*, Bengaluru, India, December 2018.

[16] Billy Bob Brumley and Nicola Tuveri. Remote timing

attacks are still practical. In *Proceedings of the 16th European Conference on Research in Computer Security (ESORICS '11)*, Leuven, Belgium, September 2011.

[17] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Conference on Security Symposium (SEC '18)*, Baltimore, MD, USA, August 2018.

[18] Yue Cao, Zhiyun Qian, Zhongjie Wang, Tuan Dao, Srikanth V. Krishnamurthy, and Lisa M. Marvel. Off-path tcp exploits: Global rate limit considered dangerous. In *Proceedings of the 25th USENIX Conference on Security Symposium (SEC '16)*, Austin, TX, USA, August 2016.

[19] Yanzhe Chen, Xingda Wei, Jiaxin Shi, Rong Chen, and Haibo Chen. Fast and general distributed transactions using rdma and htm. In *Proceedings of the Eleventh European Conference on Computer Systems (EUROSYS '16)*, London, UK, April 2016.

[20] David Cock, Qian Ge, Toby Murray, and Gernot Heiser. The last mile: An empirical study of timing channels on sel4. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*, Scottsdale, Arizona, USA, November 2014.

[21] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC '10)*, New York, New York, June 2010.

[22] Thomas. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, September 1967.

[23] Leonid Domnitser, Aamer Jaleel, Jason Loew, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. *ACM Transactions on Architecture and Code Optimization*, 8(4):35:1–35:21, January 2012.

[24] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. FaRM: Fast Remote Memory. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI '14)*, Seattle, WA, USA, April 2014.

[25] Aleksandar Dragojević, Dushyanth Narayanan, Edmund B. Nightingale, Matthew Renzelmann, Alex Shamis, Anirudh Badam, and Miguel Castro. No compromises: Distributed transactions with consistency, availability, and performance. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP '15)*, Monterey, CA, USA, October 2015.

[26] Daniel Genkin, Lev Pachmanov, Eran Tromer, and Yuval Yarom. Drive-by key-extraction cache attacks from portable code. In *Applied Cryptography and Network Security - 16th International Conference (ACNS '18)*, Leuven, Belgium, July 2018.

[27] Xun Gong, Nikita Borisov, Negar Kiyavash, and Nabil Schear. Website detection using remote traffic analysis. In *Privacy Enhancing Technologies - 12th International Symposium (PETS '12)*, Vigo, Spain, July 2012.

[28] Ben Gras, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks. In *Proceedings of the 27th USENIX Conference on Security Symposium (SEC '18)*, Baltimore, MD, USA, August 2018.

[29] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. Flush+flush: A fast and stealthy cache attack. In *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA '16)*, San Sebastián, Spain, July 2016.

[30] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. Cache template attacks: Automating attacks on inclusive last-level caches. In *Proceedings of the 24th USENIX Conference on Security Symposium (SEC '15)*, Washington, D.C., USA, August 2015.

[31] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. Efficient memory disaggregation with infiniswap. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation (NSDI '17)*, Boston, MA, USA, March 2017.

[32] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. Rdma over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*, Florianopolis, Brazil, August 2016.

[33] Danny Harnik, Benny Pinkas, and Alexandra Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *IEEE Security and Privacy*, 8(6):40–47, November 2010.

[34] Wei Huang, Gopalakrishnan Santhanaraman, Hyun-Wook Jin, Qi Gao, and Dhabaleswar K. Panda. Design of High Performance MVAPICH2: MPI2 over InfiniBand. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID '06)*, Rio de Janeiro, Brazil, May 2006.

[35] Intel. Intel Performance Counter Monitor, 2012. http://www.intel.com/software/pcm.

[36] Intel. Improving Real-Time Performance by Utilizing Cache Allocation Technology , 2015. https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cache-allocation-technology-white-paper.pdf.

[37] Rob Jansen, Marc Juarez, Rafa Galvez, Tariq Elahi, and Claudia Diaz. Inside job: Applying traffic analysis to measure tor from within. In *25th Annual Network and*

Distributed System Security Symposium (NDSS '18), San Diego, CA, USA, February 2018.

[38] Taesoo Kim, Marcus Peinado, and Gloria Mainar-Ruiz. STEALTHMEM: System-level protection against cache-based side channel attacks in the cloud. In *Proceedings of the 21st USENIX Conference on Security Symposium (SEC '12)*, Bellevue, WA, USA, August 2012.

[39] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP '19)*, San Francisco, CA, USA, May 2019.

[40] Bojie Li, Zhenyuan Ruan, Wencong Xiao, Yuanwei Lu, Yongqiang Xiong, Andrew Putnam, Enhong Chen, and Lintao Zhang. Kv-direct: High-performance in-memory key-value store with programmable nic. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*, Shanghai, China, October 2017.

[41] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *Proceedings of the 27th USENIX Conference on Security Symposium (SEC '18)*, Baltimore, MD, USA, August 2018.

[42] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. Oblivm: A programming framework for secure computation. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy (SP '15)*, San Jose, CA, USA, May 2015.

[43] Fangfei Liu, Qian Ge, Yuval Yarom, Frank Mckeen, Carlos Rozas, Gernot Heiser, and Ruby B Lee. Catalyst: Defeating last-level cache side channel attacks in cloud computing. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA '16)*, Barcelona, Spain, March 2016.

[44] Jiuxing Liu, Jiesheng Wu, and Dhabaleswar K. Panda. High Performance RDMA-based MPI Implementation over infiniBand. *International Journal of Parallel Programming*, 32(3):167–198, 2004.

[45] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. Octopus: an rdma-enabled distributed persistent memory file system. In *2017 USENIX Annual Technical Conference (ATC '17)*, Santa Clara, CA, USA, July 2017.

[46] Mellanox. Mellanox ConnectX-3 VPI Card, 2017. http://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-3_Pro_Card_VPI.pdf.

[47] Mellanox. Mellanox Network Adapters for 25G RoCE Ethernet Cloud Deployed in Alibaba, 2017. http://www.mellanox.com/page/press_release_item?id=1964.

[48] Mellanox. Mellanox ConnectX-4 VPI Card, 2018. http://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-4_VPI_Card.pdf.

[49] Mellanox. Mellanox ConnectX-5 VPI Card, 2018. http://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-5_VPI_Card.pdf.

[50] Mellanox. Mellanox InfiniBand EDR 100Gb/s Switch, 2018. http://www.mellanox.com/related-docs/prod_ib_switch_systems/pb_sb7700.pdf.

[51] Mellanox. Physical Address Memory Region, 2018. https://community.mellanox.com/s/article/physical-address-memory-region.

[52] Mellanox. RDMA/RoCE Solutions. https://community.mellanox.com/s/article/rdma-roce-solutions, 2018.

[53] Mellanox. Mellanox ConnectX-6 VPI Card, 2019. http://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-6_VPI_Card.pdf.

[54] Mellanox Technologies. InfiniBand Now Connecting More than 50 Percent of the TOP500 Supercomputing List. https://tinyurl.com/yy2ualhg, 2015.

[55] Frank Mietke, Robert Rex, Robert Baumgartl, Torsten Mehlan, Torsten Hoefler, and Wolfgang Rehm. Analysis of the memory registration process in the mellanox infiniband software stack. In *Proceedings of the 12th International Conference on Parallel Processing (Euro-Par '06)*, Dresden, Germany, September 2006.

[56] Christopher Mitchell, Yifeng Geng, and Jinyang Li. Using one-sided rdma reads to build a fast, cpu-efficient key-value store. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference (ATC '13)*, San Jose, CA, USA, June 2013.

[57] Christopher Mitchell, Kate Montgomery, Lamont Nelson, Siddhartha Sen, and Jinyang Li. Balancing cpu and network in the cell distributed b-tree store. In *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference (ATC '16)*, Denver, CO, USA, June 2016.

[58] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. Varys: Protecting SGX enclaves from practical side-channel attacks. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference (ATC '18)*, Boston, MA, USA, July 2018.

[59] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of aes. In *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology (CT-RSA '06)*, San Jose, CA, USA, February 2006.

[60] Marius Poke and Torsten Hoefler. Dare: High-performance state machine replication on rdma networks. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '15)*, Portland, OR, USA, June 2015.

[61] Zhiyun Qian and Z. Morley Mao. Off-path tcp sequence number inference attack - how firewall middleboxes reduce security. In *Proceedings of the 2012 IEEE Sym-*

posium on Security and Privacy (SP '12), San Francisco, CA, USA, May 2012.

[62] Zhiyun Qian, Z. Morley Mao, and Yinglian Xie. Collaborative tcp sequence number inference attack: How to crack sequence number under a second. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*, Raleigh, NC, USA, October 2012.

[63] RDMA Consortium. iWARP, Protocol of RDMA over IP Networks, 2009. http://www.rdmaconsortium.org/.

[64] Recio, R., Metzler, B., Culley, P., Hilland, J., and D. Garcia. A Remote Direct Memory Access Protocol Specification, 2007. https://tools.ietf.org/html/rfc5040.

[65] Robert Ricci, Eric Eide, and the CloudLab Team. Introducing cloudlab: Scientific infrastructure for advancing cloud architectures and applications. *The USENIX Magazine*, 39(6):36–38, December 2014.

[66] Michael Schwarz, Martin Schwarzl, Moritz Lipp, and Daniel Gruss. Netspectre: Read arbitrary memory over network, 2018. http://arxiv.org/abs/1807.10535.

[67] Yizhou Shan, Shin-Yeh Tsai, and Yiying Zhang. Distributed shared persistent memory. In *Proceedings of the 8th Annual Symposium on Cloud Computing (SOCC '17)*, Santa Clara, CA, USA, September 2017.

[68] Jiaxin Shi, Youyang Yao, Rong Chen, Haibo Chen, and Feifei Li. Fast and concurrent rdf queries with rdma-based distributed graph exploration. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI '16)*, Savannah, GA, USA, November 2016.

[69] Jicheng Shi, Xiang Song, Haibo Chen, and Binyu Zang. Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring. In *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSNW '11)*, Hong Kong, China, June 2011.

[70] Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, Radu Stoica, Bernard Metzler, Nikolas Ioannou, and Ioannis Koltsidas. Crail: A high-performance i/o architecture for distributed data processing. *IEEE Bulletin of the Technical Committee on Data Engineering*, 40:40–52, March 2017. Special Issue on Distributed Data Management with RDMA.

[71] Andrei Tatar, Radhesh Krishnan Konoth, Elias Athanasopoulos, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Throwhammer: Rowhammer attacks over the network and defenses. In *Proceedings of the 2018 USENIX Annual Technical Conference (ATC '18)*, Boston, MA, USA, July 2018.

[72] Mellanox Technologies. Mellanox OFED for Linux User Manual. http://www.mellanox.com/related-docs/prod_software/Mellanox_OFED_Linux_User_Manual_v3.1-1.0.0.pdf.

[73] Tejas Karmarkar. Availability of linux rdma on microsoft azure. https://azure.microsoft.com/en-us/blog/azure-linux-rdma-hpc-available, 2015.

[74] The Tcpdump Group. tcpdump - Dump Traffic on A Network. https://www.tcpdump.org/manpages/tcpdump.1.html, 2018.

[75] Eran Tromer, Dag Arne Osvik, and Adi Shamir. Efficient cache attacks on aes, and countermeasures. *Journal of Cryptology*, 23(1):37–71, January 2010.

[76] Shin-Yeh Tsai and Yiying Zhang. LITE Kernel RDMA Support for Datacenter Applications. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*, Shanghai, China, October 2017.

[77] Shin-Yeh Tsai and Yiying Zhang. A double-edged sword: Security threats and opportunities in one-sided network communication. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '19)*, Renton, WA, USA, July 2019.

[78] Stephan van Schaik, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Malicious management unit: Why stopping cache attacks in software is harder than you think. In *Proceedings of the 27th USENIX Conference on Security Symposium (SEC '18)*, Baltimore, MD, USA, August 2018.

[79] Luis M. Vaquero, Luis Rodero-Merino, and Daniel Morán. Locking the sky: A survey on iaas cloud security. *Computing*, 91(1):93–118, 2011.

[80] Cheng Wang, Jianyu Jiang, Xusheng Chen, Ning Yi, and Heming Cui. Apus: Fast and scalable paxos on rdma. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC '17)*, Santa Clara, CA, USA, September 2017.

[81] Zhenghong Wang and Ruby B. Lee. Covert and side channels due to processor architecture. In *Proceedings of the 22Nd Annual Computer Security Applications Conference (ACSAC '06)*, Miami Beach, FL, USA, December 2006.

[82] Zhenghong Wang and Ruby B. Lee. A novel cache architecture with enhanced performance and security. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '41)*, Lake Como, ITALY, November 2008.

[83] Xingda Wei, Zhiyuan Dong, Rong Chen, and Haibo Chen. Deconstructing rdma-enabled distributed transactions: Hybrid is better! In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI '18)*, Carlsbad, CA, USA, October 2018.

[84] Zachary Weinberg, Eric Y. Chen, Pavithra Ramesh Jayaraman, and Collin Jackson. I still know what you visited last summer: Leaking browsing history via user interaction and side channel attacks. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy (SP '11)*, Oakland, CA, USA, May 2011.

[85] Ming Wu, Fan Yang, Jilong Xue, Wencong Xiao, Youshan Miao, Lan Wei, Haoxiang Lin, Yafei Dai, and

Lidong Zhou. Gram: Scaling graph computation to the trillions. In *Proceedings of the Sixth ACM Symposium on Cloud Computing (SoCC '15)*, Kohala Coast, HI, USA, August 2015.

[86] Yuval Yarom and Katrina Falkner. Flush+reload: A high resolution, low noise, l3 cache side-channel attack. In *Proceedings of the 23rd USENIX Conference on Security Symposium (SEC '14)*, San Diego, CA, USA, August 2014.

[87] Yuval Yarom, Daniel Genkin, and Nadia Heninger. Cachebleed: A timing attack on openssl constant time RSA. *Journal of Cryptographic Engineering*, 7(2):99–112, 2017.

[88] Erfan Zamanian, Carsten Binnig, Tim Harris, and Tim Kraska. The End of a Myth: Distributed Transactions Can Scale. *Proceedings of the VLDB Endowment*, 10(6):685–696, 2017.

[89] Yinqian Zhang, Ari Juels, Alina Oprea, and Michael K. Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy (SP '11)*, Oakland, CA, USA, May 2011.

[90] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-vm side channels and their use to extract private keys. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*, Raleigh, NC, USA, October 2012.

[91] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Cross-tenant side-channel attacks in paas clouds. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*, Scottsdale, Arizona, USA, November 2014.

[92] Yinqian Zhang and Michael K. Reiter. Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and communications Security (CCS '13)*, Berlin, Germany, November 2013.

[93] Yiying Zhang, Jian Yang, Amirsaman Memaripour, and Steven Swanson. Mojim: A Reliable and Highly-Available Non-Volatile Memory System. In *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '15)*, Istanbul, Turkey, March 2015.

[94] Ziqiao Zhou, Michael K. Reiter, and Yinqian Zhang. A software approach to defeating side channels in last-level caches. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*, Vienna, Austria, October 2016.