Determining a Cache Hit/Miss over RDMA – A NetCAT Replication

CS6465 – Final Project

Emerson Ford Calvin Lee

1 Overview

Timing-based cache side-channels have been a well documented threat since at least 2005 [1]. By measuring access times to a specific line of memory one can determine with reasonably high accuracy if the access was a cache hit or miss. Research have shown that this can be used by attackers to recover anything from bits of RSA secret keys to passwords received over the network [2] [3]. While further research has shown these types of side channels can bypass even VM isolation boundaries, it appeared these attacks were locally restricted to a host [4]. Recently, however, technologies such as Infiniband and DDIO have enabled sub-microsecond remote memory accesses with RDMA and given NICs access to a portion of the last-level cache. In 2019, VUSec claimed these technologies enable cache side-channels to be accessible through the network. They further claim they were able to use these to perform a remote SSH keystroke timing attack [5]. We seek to replicate their remote cache side-channel access results which is the basis to enable the rest of the paper. In particular, we wanted to replicate figure 1.1.

1.1 Test Hardware

We used the following hardware in our experiments:

1. Cloudlab Apt Cluster r320: 1 x Xeon E5-2450 processor (8 cores, 2.1Ghz), 16GB Memory (4 x 2GB RDIMMs, 1.6Ghz), 1 x Mellanox MX354A Dual port FDR CX3 adapter w/1 x QSA adapter

2. Cloudlab Apt Cluster c6220: 2 x Xeon E5-2650v2 processors (8 cores each, 2.6Ghz), 64GB Memory (8 x 8GB DDR-3 RDIMMs, 1.86Ghz), 1 x Mellanox FDR CX3 Single port mezz card

3. University of Utah Notchpeak Cluster notch008/notch010: 2 x Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz, 186GB Memory, EDR Infiniband

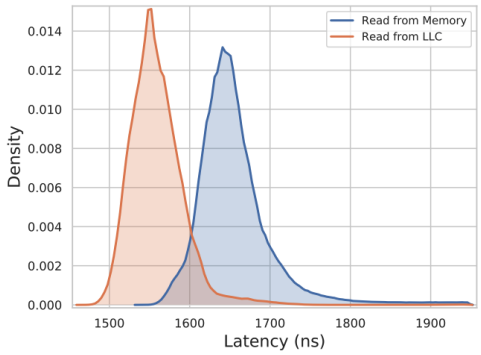

Figure 1: Read Timings from the Paper

As a rough benchmark, on the r320s an `ib_read_lat` took 1900ns on average with 50ns of standard deviation.

2 Code

2.1 Timing

On a given remote memory address x, we:

1. RDMA Read x (cache miss)

2. RDMA Write to x (pull into cache)

3. RDMA Read x (cache hit)

and timed the first read and second read. The following code was used for timing:

```
start_cycle_count = start_tsc();

rc = ibv_post_send(res->qp, &sr, &bad_wr);
if (rc)
    fprintf(stderr, "failed to post SR\n");
do {
    poll_result = ibv_poll_cq(res->cq, 1, &wc);
} while (poll_result == 0);

end_cycle_count = stop_tsc();
```

Where `start_tsc` is an lfence, rdtsc, lfence and `stop_tsc` is an rdtscp, lfence.

2.2 Read→Write→Read Methods

We were uncertain if RDMA writes triggered any cache prefetching. As such, we developed multiple ways to measure the read→write→read operations to overcome any potential prefetching.

1. Read-write-read a single address with a `clflush` between each iteration

2. Random access across an array, avoiding the cache line hit from the last iteration

3. Sequential access across an array with strides to (hopefully) overcome any prefetchers (64 byte msgs, columns count = 4, row count = 524288, ~134 MB total)

All of our code is available publicly at `https://github.com/emersonford/NetCAT-Replication`.

3 Results

Please see the presentation for graphs on each method.

3.1 `clflush` Method

Initially, we got seemingly impressive results from the `clflush` method with a clear separation between the two distributions – more so than figure 1.1. However, upon further investigation we found these timings are likely misleading. A read→read produced similar results as read→write→read albeit with a smaller distance between the two distributions. We expect it to produce two identical distributions.

We suspect two things may cause this. First, we use TCP socket communication to synchronize between the server and client. These result in syscalls that may result in page tables switches, cache evictions, etc that could cause timing anomalies. Second, there is little documention or research on the pathological behaviors of DDIO. It is possible there may be side effects from an interaction between DDIO and `clflush` that cause these timing anomalies.

Ultimately, we ruled this method out for reproducing figure 1.1.

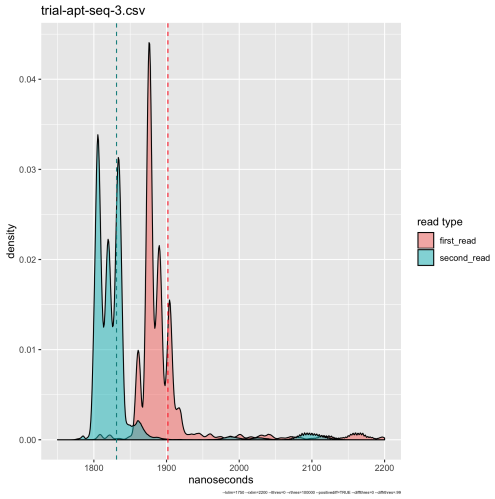

Figure 2: Results from a Run of the Sequential Access

3.2 Random Access Method

The data from random access also seemed promising at first. There appeared to be little noise and the graphs produced lined up closely with figure 1.1. However, further iterations produced unexpected results, such as the first reads being faster than the second reads. We suspect this might stems from mistuning of our random access parameters and it may be something we can fix in the future. However, we also ruled this method out for reproducing figure 1.1.

3.3 Sequential Access Method

The data obtained from sequential access consistently gave us expected results, albeit noiser than data from the random access method. The graphs generated seemed to line up with results from figure 1.1, aside from a few odd timing anomalies as seen in figure 2. Furthermore, read→read produces expected results with sequential access. Further investigations and improvements to the sequential access method are necessary to clean up data and account for timing anonamlies, but tenatively the data suggests the cache timing results are reproducable.

4 Challenges

We quickly found that timing at a nanosecond granularity is unexpectedly difficult on modern systems.

There are a number of factors that can influence the timing results and there is little documentation for how some parts of the system interact, such as DDIO and the NIC. Some of the challenges we faced and tried to investigate are:

4.1 Server Challenges

1. **NUMA**: it's possible the server's registered memory could be placed in a separate NUMA node than the NIC. This would affect timing results as main memory access would then also have to traverse the CPU fabric. We investigated how to put our registered memory in the correct NUMA node and what effects incorrect NUMA placement had on our timing.
2. **Prefetching**: it's likely there is some prefetching into cache done with an RDMA write, but we were unable to verify this with either data or documentation. If the prefetcher is smarter than we expect, it may severly interfere with timing. Further investigations of this are necessary.
3. **DDIO Pathologies**: we're uncertain what caused the misleading data from our `clflush` method. It is possible there may be some unexpected interaction with DDIO when `clflush` is used. Further testing of this is required.
4. **DDIO Ways Restriction**: DDIO, on Intel Xeon E5 CPUs, is restricted to only 2 of the 20 cache ways in the LLC. This limits the types of data we wish to PRIME+PROBE on.

4.2 Client Challenges

1. **CPU Frequency/Power Management**: we frequently found the client CPU to dramatic alter in its frequency. We're unsure if this contributed to the noise in our data and investigated ways to turn off frequency scaling for our tests to be on the safe side. We ended up learning quite a bit about how Intel manages CPU frequencies and the different power states a CPU can be in to solve this.

4.3 Network Fabric Challenges

1. **Infiniband Saturation**: it appears the more saturated an Infiniband fabric is, the higher the mean and variance for RDMA read latencies. This causes significant issues as the saturation of an Infiniband fabric is difficult to determine. In future tests, we may try to find a non-used Infiniband fabric to control for this.

There are still quite a number of graph anomalies we cannot reason about. These anomalies make it extremely difficult to do any statistical prediction on whether a read was a cache hit or miss. Further testing is required to account for these anomalies.

5 Next Steps

Continuing on the project, the next steps would be:

- Clean up data and reduce data anomalies
- Develop a statistical prediction on whether an RDMA read is a cache hit or miss
- Use this prediction to build an evicition set on the remote host
- Perform a PRIME+PROBE attack on the remote host

We suspect each task will be non-trivial based on papers that achieve these locally.

6 Conclusion

It appears it is likely one can determine if a remote memory access is served from LLC or main memory, but our data is not conclusive. While we may not have been able to replicate their cache timing results exactly, we did learn quite a bit about the various technologies that go into this paper. Given more time, we feel we could overcome or at least account for these challenges and fully develop a remote PRIME+PROBE attack on a remote host.

References

[1] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of aes," in *Cryptographers' track at the RSA conference*, Springer, 2006, pp. 1–20.

[2] M. Schwarz, M. Lipp, D. Gruss, S. Weiser, C. Maurice, R. Spreitzer, and S. Mangard, "Keydrown: Eliminating software-based keystroke timing side-channel attacks," 2018.

[3] Y. Yarom and K. Falkner, "Flush+ reload: A high resolution, low noise, l3 cache side-channel attack," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 719–732.

[4] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *2015 IEEE Symposium on Security and Privacy*, IEEE, 2015, pp. 605–622.

[5] M. Kurth, B. Gras, D. Andriesse, C. Giuffrida, H. Bos, and K. Razavi, "NetCAT: Practical Cache Attacks from the Network," in *S&P*, Intel Bounty Reward, May 2020. [Online]. Available: Paper=https://download.vusec.net/papers/netcat_sp20.pdf%20Web=https://www.vusec.net/projects/netcat.