# Cache Attacks in Modern/Multi-Socket x86 Systems (Work in Progress)

Guillaume Didier, Augustin Lucas, Thomas Rokicki

# Cache Attacks in Modern/Multi-Socket x86 Systems (Work in Progress)

Guillaume DIDIER[1,2,4], Augustin LUCAS[2,3], and Thomas ROKICKI[5]

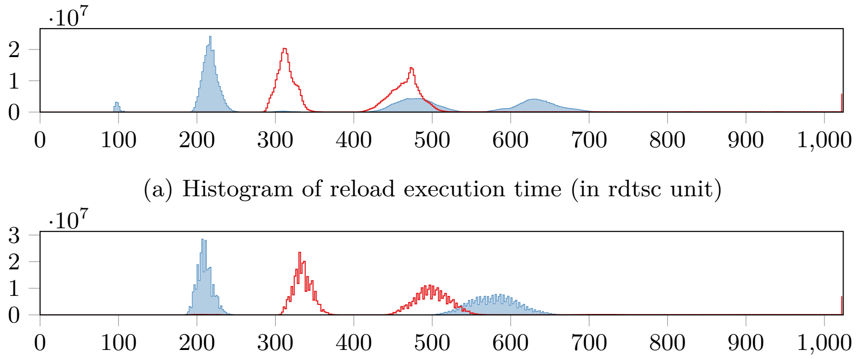
[1] Universität des Saarlandes
[2] Univ. Rennes, Inria, IRISA
[3] Département d'Informatique, ENS de Lyon
[4] Direction Générale de l'Armement
[5] CentraleSupélec, Inria, CNRS, IRISA, Université de Rennes, France
guillaume.didier@uni-saarland.de[0009−0007−9076−7318],
augustin.lucas@ens-lyon.fr, thomas.rokicki@irisa.fr[0000−0002−7041−4300]

**Abstract.** In this work in progress, we re-examine cache attacks that leverage the x86 `clflush` instruction — namely Flush+Reload and Flush+Flush — on modern hardware. In particular, we look at multi-socket systems in which cross-socket cache coherence between different sockets affects the attacks' accuracy and performance.
Our preliminary results show that on dual-socket systems, Flush+Reload exhibits complex timing patterns that usually render single-threshold approaches ineffective to classify cache hits and misses.On such systems, the reload step of Flush+Reload may even be *slower* for a cache hit than a miss. We pinpoint Non-Uniform Memory Access as the main source of this complexity, which both Flush+Reload and Flush+Flush thus need to account for. Properly accounting for this can lead to a more than tenfold reduction in error rate (e.g., on dual-socket Ice Lake SP, from a 26.0% to a 1.55% average error rate). We also show the Flush+Flush attack works on AMD CPUs, in single-socket and multi-socket systems alike.

**Keywords:** Cache Side Channel · NUMA · Microarchitectural attack.

## 1 Introduction

In the field of microarchitecture security, cache attacks are both amongst the oldest primitives [15,13] and the most used ones. For instance, most transient execution attacks rely on a cache channel to exfiltrate data from the transient domain into the permanent domain [17]. These primitives are also an important part of the toolbox to reverse engineer microarchitecture. On the x86 instruction set, the `clflush` instruction is available in user mode, obviating the need for eviction sets as long as the underlying target memory is shared between attacker and victim. This leads to the Flush+Reload [19] attack primitive, which is commonly used in x86 security research. Additionally, it has been noticed [8] that on Intel CPUs, `clflush` execution time depends on the state of the cache line to be flushed, leading to the Flush+Flush attack. This attack was deemed restricted



(a) Histogram of reload execution time (in rdtsc unit)

(b) Histogram of `clflush` execution time (in rdtsc unit)

Fig. 1: Histograms on `troll`, a 2×Intel Xeon Gold 5218 system (Cascade Lake X). Hits are in blue, filled; misses are in thick outline red.

in applicability, as it had only ever been demonstrated on Intel CPUs, but never on AMD ones [6]. Didier et al. [5] refined this later attack with a topology-aware calibration.They also noted an unexpected bimodal distribution of execution time on dual-socket Intel systems, but didn't investigate further.

We thus investigate the unusual behavior of such cache attacks on multi-socket systems, and more broadly review their behavior on newer Intel and AMD systems. *Our goal is to determine the sources of variability in Flush+Flush and Flush+Reload attacks on multi-socket systems, and the performance achievable under various attacker models.* Our main hypothesis is that accounting for these sources of variability greatly improves attack accuracies. We also hypothesize that Non-Uniform Memory Access (Numa) is the dominant source of variability, which our preliminary data supports. Under that hypothesis, these results can then improve the quality of data collected for further microarchitecture reverse-engineering works. These insights can also help better understand the behavior of Numa caches regarding cache attacks, benefiting defensive research.

Our empirical study finds that Numa-aware attacks are feasible in userland, and our Numa-aware Flush+Flush often shows a 10× improvement over a topology-unaware Flush+Reload. On a dual socket Ice Lake SP (`montcalm`), the naïve implementation of Flush+Reload and Flush+Flush exhibit average error rates of 26.0% and 25.0%, respectively. In contrast, the Numa-aware Flush+Reload achieves a reduced error rate of 4.86%, while the Numa-aware Flush+Flush further improves accuracy, with an error rate of 1.55%. Similarly, on a dual-socket AMD Zen 4 (Genoa) system (`musa`), the naïve Flush+Reload and Flush+Flush exhibit respective average error rates of 23% and 16.7%. The Numa-aware Flush+Reload then achieves a reduced error rate of 15%, while the Numa-aware Flush+Flush further improves accuracy, with an error rate under 1%.

We can already report the following findings on multi-socket systems:

1. Numa is a major contributor to load and `clflush` execution timing.

2. NUMA-awareness is key for accurate Flush+Reload and Flush+Flush attacks.
3. Flush+Flush is a viable attack on most modern x86 systems, both single- and multi-socket, and often more accurate than Flush+Reload.

The remainder of the paper is organized as follows: In Section 2, we review related work, then Section 3 shows how NUMA is the dominant contributor to the variability of cache-attack primitives on multi-socket systems. In Section 4, we present an extensive study of the accuracy of Flush+Reload and Flush+Flush on a wide array of machines, comparing topology unaware and NUMA-aware attacker models. We then describe our future plans (Section 5), for the full paper and later work. Lastly, we summarize our findings in Section 6. Our extensive results, on 32 machines, are available online, as described by Appendix 6.

## 2 Background and related work

*Non-Uniform Memory Access:* In modern computer systems, the physical distance between the processor and memory has become a critical factor. Longer signal paths greatly increase memory access latency and can significantly impede system performance. This issue is further exacerbated in architectures where multiple processors share the same signal path, leading to potential throughput bottlenecks that constrain overall efficiency. In particular, in multi-socket systems, having a single, unified memory can even create contention on the interconnect within an individual socket. To address this, hardware vendors introduced *Non-Uniform Memory Access* or *NUMA*, *i.e.*, different parts of a processor memory (or *NUMA nodes*) have different performance characteristics depending on the core accessing them. The most common approach on multi-socket computers is to attach parts of the memory directly to a specific socket. *NUMA* requires the Operating System (OS) to be aware of the separation of memory to optimize the speed of loads. Unix systems offer the `numactl` interface to let a user define the affinity of a program regarding NUMA memory. In these systems, the physical memory and cores in the system are partitioned into *NUMA nodes*: Each node contains a memory controller, its associated physical memory, and the cores closest to this controller. For instance, pinning the memory used by a process to a given NUMA node and its threads to cores within that node thus minimizes memory latency.

*Cache Timing Attacks:* CPU caches are small, fast memories, used to retain highly used data or instructions. This microarchitectural optimization is amongst the most targeted by attacks, and many primitives are known. Such primitives usually exploit the timing difference between a *cache hit*, *i.e.*, accessing a value in the cache, and a *cache miss*, *i.e.*, accessing a value not in the cache. In this work, we focus on two primitives: Flush+Reload and Flush+Flush. Both use the unprivileged `clflush` x86 instruction to evict a target cache line, which gets cached again after a victim access. They then measure an execution time to determine whether the target is cached, repeating these two steps as needed. These attacks require read-only physical memory sharing, a constraint that can be met for target addresses in shared library code and read-only sections[19,9,8], in cross-VM settings due to page deduplication [1], and in transient execution attacks within the same process or against the kernel [3,17].

Flush+Reload [19] uses a load instruction, and then flushes the line again.

Flush+Flush [8] leverages the fact that the execution time of `clflush` itself depends on the line state to remove the load. Although the timing difference is smaller—often just a few CPU cycles—making Flush+Flush more susceptible to noise, it improves stealth and speed by avoiding time-consuming cache misses.

*Numa-based side channels:* To the best of our knowledge, few works have studied Numa from a side-channel perspective. Yao et al. [18] exploited the cache access timing differences caused by Numa on a multi-socket system to build covert channels, *i.e.*, a stealthy exchange of information across isolation boundaries. The information is transmitted by distinguishing local hits—*i.e.*, data accessed from the same Numa node, from remote hits—*i.e.*, data accessed from another node. PCIe bus contention between different Numa nodes was leveraged by Tian et al. [16] to determine FPGA co-locality on cloud FPGA infrastructure.

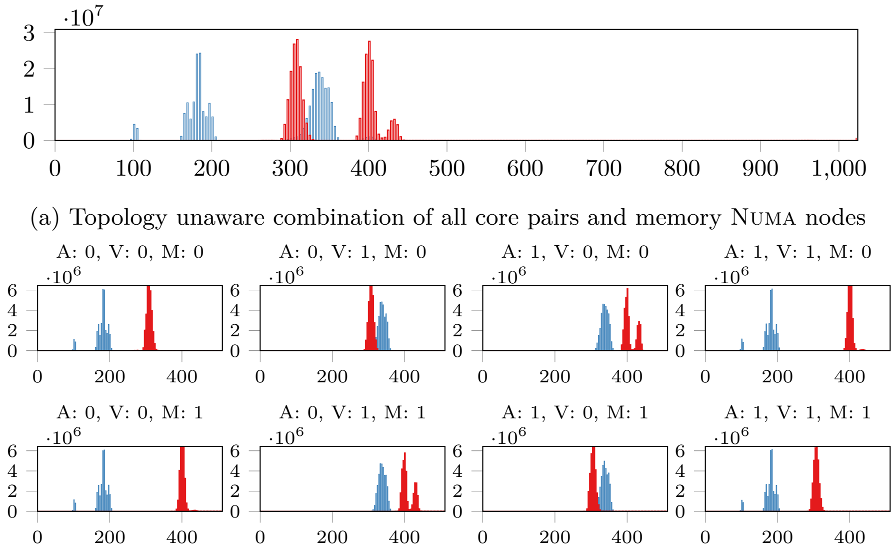
*Other cross-core related works:* The Last-Level Cache (LLC) of Intel CPUs is split into slices, each located next to a distinct core, and physical addresses are hashed to determine the slice they belong in [12]. Didier and Maurice [5] studied the impact of this slicing on Flush+Flush on client Intel CPUs. In particular, by accounting for the interconnect topology of these CPUs, they reduce the error rate of a standard Flush+Flush from 20% to a negligible amount, while also tripling its bandwidth. Pacagnella et al. [14] leveraged ring-interconnect contention to build cross-core primitives for covert-channel and side-channel attacks. Similarly, Dai et al. [4] exploited the mesh interconnect introduced in Skylake SP server CPUs to mount side-channel attacks. Dutta et al. [7] focused on multi-GPU systems, mounting a Prime+Probe attack on remote GPUs, for fingerprinting and hyperparameter extraction. Li et al. [10] proposed a new attack primitive across independent LLCs in AMD's *CoreComplex (CCX)* server CPUs.

## 3 NUMA contribution to timing in dual-socket systems

Figure 2a is a histogram of load execution time on a dual-socket system, and exhibits a strange behavior, as first reported by Didier and Maurice [5]: in some cases, a hit may be slower than a miss. Thus, a single threshold would here be insufficient to distinguish hits from misses. To investigate it, we first used an older, better understood, dual-socket Haswell EP machine, with 8 cores[6] and a single Numa node per socket. However, it first seemed that the slices associated with virtual addresses were changing during execution. This suggested the kernel was changing the underlying physical addresses, which was eventually explained by kernel Numa re-balancing as our threads migrated across sockets.

Using `numactl` to properly pin memory on each node, we measured hit and miss load execution times for each possible combination of the attacker core ($A$),

[6] This CPU uses the same linear hash functions [12,5] as other 8-core Intel systems.



(a) Topology unaware combination of all core pairs and memory NUMA nodes

(b) Distinguishing the NUMA nodes of the attacker $A$, victim $V$ and target memory $M$.

Fig. 2: Hit (blue) and miss (red) reload time histograms on `parasilo`, *cf.* Table 1.

victim core ($V$), and memory NUMA node ($M$). In Figure 2, we plot both the histograms for individual $(A, V, M)$ combinations, and the combined histogram. It confirms that NUMA is the major factor in the observed variability. Because NUMA relates to memory access latency, its large impact on the execution time of memory instruction is quite consistent. Figure 2b shows that the memory NUMA node has no impact on reload-hit timings, as expected for a cache hit. Meanwhile, it is a major source of variability for misses: for the same $A$ and $V$, a reload is much slower if $A$ and $M$ differ, which must thus be accounted for. Furthermore, Figure 2a proves that ignoring NUMA hampers the distinction of hits and misses.

Lastly, this figure underlines the symmetry of the system: what matters is the relative location of the node of each element. For instance, the configurations $A = 0, V = 1, M = 1$ and $A = 1, V = 0, M = 0$ raise similar histograms. This symmetry was used by Lucas [11] to further study the behavior of `clflush` and the topology of Intel server CPUs from Sandy Bridge to Haswell.

## 4 Extensive study on modern systems, NUMA awareness

### 4.1 Experiment

Given those insights, we undertook a broader study of Flush+Reload and Flush+Flush, on a wide range of microarchitectures of mostly dual-socket x86 systems.

We adapt to NUMA the calibration algorithm from Didier and Maurice [5]: For each combination of a NUMA node, to which our memory is pinned, a pair

Table 1: Subset of the machines used as an illustration of the work

| Name | Processors | $\mu$-arch | N | C/T | OS |
|---|---|---|---|---|---|
| `dahu` | 2× Intel Xeon Gold 6130 | Skylake SP | 2 | 2× 16/32 | Deb. 11 |
| `esterel41` | 2× Intel Xeon Gold 6426Y | Sapphire Rapids | 2 | 2× 16/32 | Deb. 11 |
| `grue` | 2× AMD EPYC 7351 | Zen 1 (Naples) | 8 | 2× 16/32 | Deb. 11 |
| `montcalm` | 2× Intel Xeon Silver 4314 | Ice Lake SP | 2 | 2× 16/32 | Deb. 11 |
| `musa` | 2× AMD EPYC 9254 | Zen 4 (Genoa) | 2 | 2× 24/48 | Deb. 11 |
| `parasilo` | 2× Intel Xeon E5-2630 v3 | Haswell EP | 2 | 2× 8/16 | Deb. 11 |
| `troll` | 2× Intel Xeon Gold 5218 | Cascade Lake SP | 2 | 2× 16/32 | Deb. 11 |
| `yeti` | 4× Intel Xeon Gold 6130 | Skylake SP | 4 | 4× 16/32 | Deb. 11 |
| `ARL` | Intel Core Ultra 7 265K | Arrow Lake | 1 | 8/8P+12/12E | Ub. 24.10 |
| `EMR` | Intel Xeon Silver 4514Y | Emerald Rapids | 1 | 16/32 | Ub. 24.10 |
| `Zen2` | AMD Ryzen 7 3700X | Zen 2 (Matisse) | 1 | 8/16 | Ub. 20.04 |
| `Zen4` | AMD Ryzen 5 7600X | Zen 4 (Raphael) | 1 | 6/12 | Ub. 22.04 |
| `Zen5` | AMD Ryzen 7 9700X | Zen 5 (Granite Ridge) | 1 | 8/16 | Ub. 24.10 |
| `ZenP` | AMD Ryzen 5 2600 | Zen+ (Pinnacle Ridge) | 1 | 6/12 | Ub. 18.04 |

of threads (attacker and victim), and a cache line within a 4 KiB page, we take 1024 measurements (after 128 warm-up iterations) of each *operation*.

Starting with the line in a known state, usually flushed, an *operation* consists of a victim execution, acting on the cache-line state, followed by the attacker execution, which measures the execution time of a load or `clflush` instruction, and resets the state. Fences and mutual exclusion are used to avoid races and ensure correct ordering. We then build a histogram of the execution times for each combination of the $(A, V, M)$ parameters (respectively, the Numa node of the Attacker, Victim and target Memory). For better reproducibility, we report the results obtained at a fixed frequency and with prefetchers disabled.

Hence, the experiment run-time and uncompressed result files both scale linearly in the number of Numa nodes, and quadratically in the number of cores[7].

### 4.2 Preliminary analysis and results

Having exhaustively sampled the variability space, we can combine all the 1024-point histograms into a single, *topology-unaware histogram.* We also combine those into a small set of histograms for a Numa-aware attack. In such a setup, the Numa node for the attacker and victim core are known and the victim memory is also pinned to a node. We thus have the same three parameters: $A$ the

[7] We iterate on *pairs of cores.* As a concrete example, the results for `yeti`, with 128 threads, took over 24h to obtain, and occupy over 80 GiB in memory.

Table 2: Error rate predictions (in %) on single-socket machines from Table 1. ST: Single Threshold, DT: Dual-Threshold.

| Machine | Flush+Reload | | Flush+Flush | |
|---|---|---|---|---|
| | ST | DT | ST | DT |
| ARL | 2.45 | 2.45 | 0.02 | 0.01 |
| EMR | 39.32 | 33.85 | 0.04 | 0.04 |
| Zen2 | 26.56 | 26.56 | 20.78 | 20.79 |
| Zen4 | $< 0.01$ | $< 0.01$ | 10.11 | 10.11 |
| Zen5 | 3.05 | 3.05 | 16.24 | 16.24 |
| ZenP | 28.82 | 25.87 | 9.27 | 9.27 |

attacker's node, $V$ the victim's node, and $M$, the memory's node. Usually, the cores of a single socket form a NUMA node[8]. Precisely, this model corresponds with knowing the NUMA node of the attacker thread, the victim thread and the memory targeted, but we will shorten this to NUMA-aware, as we have not yet considered other models (such as one where the kernel remains free to migrate the memory used across NUMA nodes). We plan to include more models in our full paper. If the attacker is able to choose, and not just know, the three NUMA nodes, the resulting error rates are the minimum rate from our experiments.

From the pair of the cache hit and miss histograms of a given attack, we compute a threshold minimizing the average error rate (*i.e.*, minimizing the sum of false hits and false misses). In cases where the distributions exhibit several peaks, we also consider an approach with two thresholds. In this case, we interpret an execution time between the two thresholds as one outcome (e.g., a hit), and a time outside the interval as the other outcome (respectively a miss). This approach is suggested by the bimodal distribution observed in Figure 1b, for instance. We applied this method to the topology-unaware attackers.

For topology-unaware setups we get a single threshold or a single pair of threshold, while for NUMA-aware ones, we get $N_{\text{NUMA}}^3$ thresholds. We can then break up the accuracy of the attacks on each NUMA-aware configuration and compute statistics: The minimum, maximum, average, and median error rates observed, along with the quartiles. We present the result on a small subset of multi-socket machines as box plots in Figure 3. We also present results for single socket machines in Table 2, evaluating both attacks across multiple generations of AMD CPUs. We remarked that AMD CPUs seem to have a coarser `rdtsc`, which degrades attack accuracy, with a greater impact on Flush+Flush.

Thus, we observe that NUMA-awareness strongly improves the attack accuracy and usually reduces the worst-case error rate significantly. Notably, under a topology-unaware model, an unlucky selection of NUMA nodes by the kernel for threads and memory can push error rates above 50%. For instance, this is the

[8] The one exception is the Zen 1 (Naples) system with 4 NUMA nodes per socket

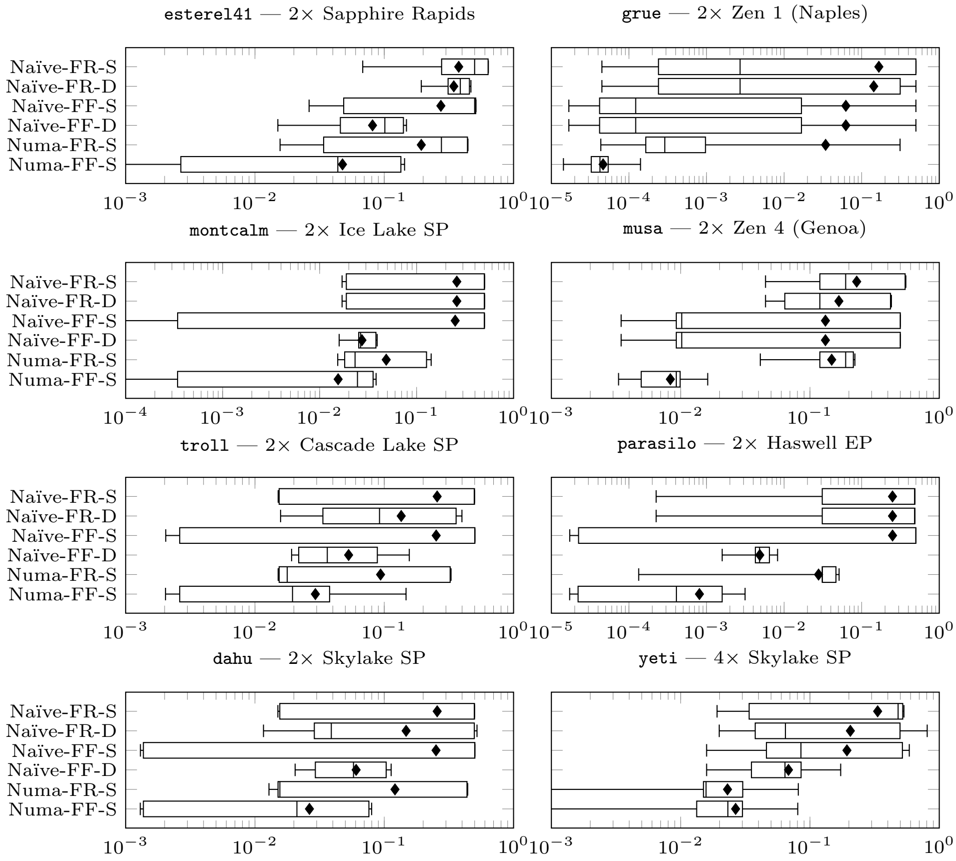


Fig. 3: Box plot of the error rates for attacks on the multi-socket machines in Table 1, with a *logarithmic* scale (lower is better).
The *Naïve*, topology-unaware, methods use the same single-threshold (-S) or pair of thresholds (-D) for all $(A, V, M)$ node combinations, while the Numa-aware methods use a threshold for each $(A, V, M)$ node combination. *Note that not all scales are identical.*

case for `esterel41` (2× Sapphire Rapids): The *median* error rate is 50.0% for the topology-unaware single-threshold Flush+Reload, the worst case 63.9%, the best case 6.8%, and three-quarters of configurations are above 25% However, the Numa-aware Flush+Flush attack has a worst case of 14%, an average and median error rates of 4%, and a quarter of the configurations have error rates under 0.3%. Overall, Numa-aware attacks greatly outperform topology unaware-attacks.

### 4.3 Impact of prefetchers and dynamic frequency and scaling

While disabling prefetchers and dynamic frequency scaling improves reproducibility of research, attackers usually cannot disable these, as it requires elevated privileges. Hence, we ran our experiments twice, once in the reproducible condition above, and once in the more realistic conditions. Comparing the results, the overall findings are consistent, with mostly slight increase in noise levels, except on some AMD machines. We include those results in the online appendix.

## 5 Further work

### 5.1 Planned work for the full paper

Our preliminary results only account for the NUMA nodes of the attacker, victim, and target memory. However, the impact of the exact location of both attacker and victim at the core granularity, and of their co-location has yet to be explored. We intend to further analyze our already-collected data to evaluate the impact of those parameters and the potential accuracy under additional attacker models. Previous single-socket work [5] has shown the impact of the slices and the core locations on the results of Flush+Flush. By combining our NUMA calibration with this in-socket topology-aware approach, we hope to further lower error rates.

In addition, we will extend our framework to include several attacker models. For instance, we want to study the case where an attacker is aware of the NUMA architecture but is unable to pin processes to a specific node. This would illustrate a scenario where `numactl` is unavailable, or an attacker-aware NUMA scheduler.

Lastly, the performance evaluation of side-channel primitives is a complex matter. While the error rate is a significant performance indicator, it does not totally reflect a primitive's performance. We will address this in the full paper by accounting for the attack speed in addition to its accuracy. To do so, we will thus integrate our improved calibration and the various attacker models into an end-to-end performance benchmark, and run it extensively on the same machines.

### 5.2 Lead for other work

This work focuses heavily on x86-based server-grade microarchitectures. Porting topology-aware methods to different architectures, which may not provide an unprivileged equivalent to `clflush` could provide similar gains to attacks on those platforms. On such platforms, the equivalent attack would be Evict+Reload, which relies on eviction sets instead, and would thus be the most attractive target. On the other hand, Prime+Probe attacks seem rather unlikely to be suitable for use in systems that do not share a single last-level cache, but verifying this assumption would be a simple extension on top of Evict+Reload. Lastly, the behavior of `rdtsc` on AMD machines seems to warrant further investigation.

## 6 Conclusion

We have carefully calibrated Flush+Flush and Flush+Reload attacks on recent, multi-socket and single-socket, Intel and AMD machines. We showed that cache attacks in multi-socket systems have to cope with major variability caused by

Non-Uniform Memory Access (Numa). This variability can significantly degrade the accuracy depending on the respective location of the attacker, victim and the target memory. Usually, the most unfavorable case is an attack in which both the target memory and the victim belong to a different node than the attacker.

In general, we show that attacks that do not account for these variables will not work reliably, with the kernel Numa re-balancing causing major interference.

On the other hand, we show that Flush+Flush can be a viable attack in many situations, including on AMD CPUs, whose vulnerability had not been identified before. Often Flush+Flush is even better than Flush+Reload, especially in the aforementioned remote victim and memory configuration. On average, topology awareness can reduce the error rate by at least an order of magnitude, e.g., on `kinovis`, a dual-socket Sapphire Rapids machine, naïve Flush+Reload and Flush+Flush alike have a 50% average error rate, while a Numa-aware Flush+ Flush only suffers from a 4.9% average error rate.

Overall, we conclude that topology awareness is essential on modern multi-socket systems. In our full paper, we intend to also account for the variability at both core granularity and cache-slicing granularity. We will also diversify the attacker models to cover more scenarios, and identify the minimal prerequisite an attacker needs to run a successful attack.

**Acknowledgments.** Multi-socket experiments used the Grid'5000 test-bed [2], supported by a scientific interest group hosted by Inria, including CNRS, RENATER, several universities, and other organizations (*c.f.* `https://www.grid5000.fr`).

This project started at IRISA, with support from the French DGA, and was then pursued at Saarland University. This work has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 101020415).

## References

1. Apecechea, G.I., Inci, M.S., Eisenbarth, T., Sunar, B.: Fine grain cross-VM attacks on Xen and VMware. In: International Conference on Big Data and Cloud Computing, BDCloud (2014)
2. Balouek, D., Carpen Amarie, A., Charrier, G., Desprez, F., Jeannot, E., Jeanvoine, E., Lèbre, A., Margery, D., Niclausse, N., Nussbaum, L., Richard, O., Pérez, C., Quesnel, F., Rohr, C., Sarzyniec, L.: Adding virtualization capabilities to the Grid'5000 testbed. In: Communications in Computer and Information Science (2013)
3. Canella, C., Bulck, J.V., Schwarz, M., Lipp, M., von Berg, B., Ortner, P., Piessens, F., Evtyushkin, D., Gruss, D.: A systematic evaluation of transient execution attacks and defenses. In: USENIX Security (2019)
4. Dai, M., Paccagnella, R., Gomez-Garcia, M., McCalpin, J., Yan, M.: Don't Mesh Around: Side-Channel Attacks and Mitigations on Mesh Interconnects. In: 31st USENIX Security Symposium (2022)
5. Didier, G., Maurice, C.: Calibration done right: Noiseless Flush+Flush attacks. In: Detection of Intrusions and Malware, and Vulnerability Assessment - 18th International Conference, DIMVA (2021)

6. Didier, G., Maurice, C., Geimer, A., Ghandour, W.J.: Characterizing prefetchers using CacheObserver. In: SBAC-PAD (2022)
7. Dutta, S.B., Naghibijouybari, H., Gupta, A., Abu-Ghazaleh, N., Marquez, A., Barker, K.: Spy in the GPU-box: Covert and Side Channel Attacks on Multi-GPU Systems. In: Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA) (2023). https://doi.org/10.1145/3579371.3589080
8. Gruss, D., Maurice, C., Wagner, K., Mangard, S.: Flush+Flush: A Fast and Stealthy Cache Attack. In: Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA) (2016). https://doi.org/10.1007/978-3-319-40667-1_14
9. Gruss, D., Spreitzer, R., Mangard, S.: Cache template attacks: Automating attacks on inclusive last-level caches. In: USENIX Security (2015)
10. Li, D., Zhu, Z., Shen, J., Zhang, Y., Shi, G., Meng, D.: Flush+Revisit: A Cross-CCX Side-Channel Attack on AMD Processors. In: IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) (2023). https://doi.org/10.1109/TrustCom60117.2023.00131
11. Lucas, A.: Internship report: Caractérisation de l'instruction clflush sur systèmes multi-socket (2024)
12. Maurice, C., Scouarnec, N.L., Neumann, C., Heen, O., Francillon, A.: Reverse engineering intel last-level cache complex addressing using performance counters. In: RAID (2015)
13. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: The case of AES. In: CT-RSA (2006)
14. Paccagnella, R., Luo, L., Fletcher, C.W.: Lord of the Ring(s): Side Channel Attacks on the CPU On-Chip Ring Interconnect Are Practical. In: 30th USENIX Security Symposium (2021)
15. Percival, C.: Cache missing for fun and profit. In: BSDCan (2005)
16. Tian, S., Giechaskiel, I., Xiong, W., Szefer, J.: Cloud FPGA Cartography using PCIe Contention. In: IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM) (2021). https://doi.org/10.1109/FCCM51124.2021.00035
17. Xiong, W., Szefer, J.: Survey of transient execution attacks and their mitigations. ACM Computing Surveys (2021)
18. Yao, F., Venkataramani, G., Doroslovački, M.: Covert Timing Channels Exploiting Non-Uniform Memory Access based Architectures. In: Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)) (2017). https://doi.org/10.1145/3060403.3060417
19. Yarom, Y., Falkner, K.: FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In: 23rd USENIX Security Symposium (2014)

## Appendix: Online supplementary materials

The online appendix at https://doi.org/10.5281/zenodo.16794924 contains:

- The list of all 25 multi-socket and 7 single-socket machines we studied.
- Detailed experimental results for all 32 machines, running at a fixed frequency and with prefetcher disabled when possible.
- The same results with variable frequency and default prefetcher configuration.
- Extended Figure 3 with all machines with fixed and variable frequency.