`clflush`-based Attacks on Modern Cloud Servers

Guillaume Didier[1,2], Augustin Lucas[2,3], Jasper Quirk[4], and Thomas Rokicki[5]

[1] Universität des Saarlandes
[2] Univ. Rennes, Inria, IRISA
[3] Département d'Informatique, ENS de Lyon
[4] Ruhr-Universität Bochum
[5] CentraleSupélec, Inria, CNRS, IRISA, Université de Rennes, France
guillaume.didier@uni-saarland.de[0009−0007−9076−7318],
augustin.lucas@ens-lyon.fr, jasper.quirk@rub.de[0009−0001−1712−8831],
thomas.rokicki@irisa.fr[0000−0002−7041−4300]

Abstract. Modern clouds usually rely on server CPUs, often deployed in multi-socket systems. Flush-based cache attacks have been extensively studied and leveraged, but mostly on single-socket systems, with client CPUs. In multi-socket servers, cache and memory latencies depend on the physical layout of cores, caches, and Numa nodes. Hence, system topology strongly influences the accuracy of such attacks. We thus investigate the impact of this growing complexity on the effectiveness of such attacks. We present a large-scale, topology-aware study of Flush+Reload and Flush+Flush across 36 single- and multi-socket Intel and AMD servers. We show topology-induced contributions dominate the latency variations. Notably, Numa's contribution on multi-socket systems makes topology-unaware attacks unreliable. Our topology-aware calibration accounts for such parameters as attacker/victim cores, memory Numa node, and target address, improving error rates and attack viability. We show Flush+Flush applies widely to AMD CPUs, and is often more accurate than Flush+Reload on modern x86 targets. Lastly, we introduce Load/Flush+Reload, a covert-channel that compares invalid loads to shared loads, with twice the true capacity of Flush+Reload/Flush+Flush. Our results show that topology awareness is required for reliable cache attacks on recent platforms, and provide practical guidance for attackers and defenders.

1 Introduction

Cache side and covert channels are among the oldest and most exploited microarchitectural leakage primitives [20,19], and remain essential to offensive and defensive research: e.g., to exfiltrate data out of the transient domain in transient execution attacks [25], or help reverse engineering modern microarchitectures [1,6].

The unprivileged `clflush` x86 instruction evicts cache lines from all levels of the cache hierarchy, enabling fine-grained observation of memory accesses within (read-only) memory shared by the attacker and victim, with the Flush+Reload attack [27]. `clflush`'s latency also depends on cache lines' coherence state, enabling the Flush+Flush attack on Intel [9], and AMD Zen2 [14] CPUs.

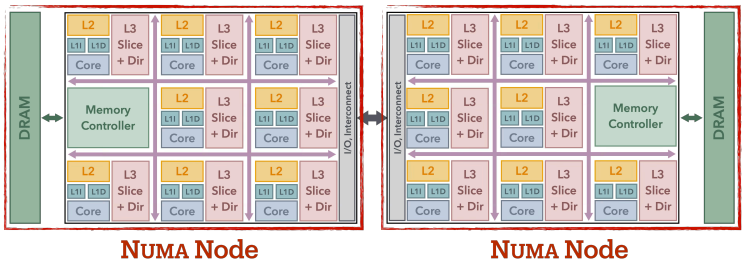
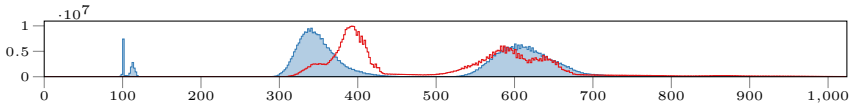
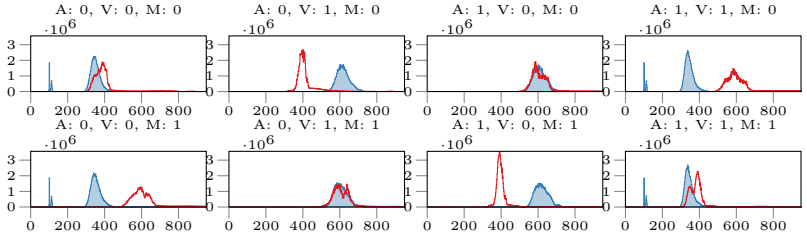


Fig. 1: Topology of a Intel-like NUMA multi-socket server.



(a) Topology-unaware histogram, equally weighing all possible configurations



(b) NUMA-aware histograms, under the NUMA-AVM model.

Fig. 2: Flush+Reload timing histograms on `esterel41` (2× Sapphire Rapids); cache misses (I) outlined in red; victim cache hits (victim-E), filled in blue.

Cloud servers have grown increasingly complex: Figure 1 shows such systems have several levels of caches, with complex last-level caches, on-chip and off-chip interconnect, and, in multi-socket systems, multiple NUMA nodes. Hence, the memory latency strongly depends on the relative placement of cores, cache slices, and NUMA nodes within the system, *cf.* Figure 2, where reading from a remote node's memory incurs a higher latency (600 cycles) than reading from the local node's memory (only 400). However intuitive, the impact of this behavior on the accuracy, reliability, and timing characteristics of cache-based side-channel attacks has yet to be quantified. This gap motivates our study and our main research question: **To what extent does the growing complexity of x86 server systems affect the effectiveness of flush-based cache attacks?**

We investigate which topological factors most strongly impact the variability of Flush+Flush and Flush+Reload, and how attacker awareness of them improves accuracy and bandwidth. We analyze memory latency across all relevant topological dimensions—attacker and victim core placement, cache slice, and NUMA node—to quantify their respective impact on timing variability and error rates. Our benchmark of the bandwidth and error rates of covert channels assesses these primitives' practical speed and robustness, further quantifying the gains from topology awareness. Our wide survey of **36** Intel and AMD systems, including 25 multi-socket servers, ensures our conclusions generalize across platforms.

We find that topology awareness, starting with NUMA, may reduce error rates by an order of magnitude over topology-unaware attacks: on `esterel41`, a two-socket Intel server, it reduces the Flush+Flush average error rate from 27.55% to 1.53%, and even below 0.01% if the attacker can select the best configuration. We also present Load/Flush+Reload, a fast and accurate covert-channel, with a true capacity of 4.98 Mbit/s, on a 1× Zen 5 machine, a 2× gain over Flush+Flush (1.95 Mbit/s) and Flush+Reload (2.27 Mbit/s). Our contributions are:

— We show that topology is a major contributor to load and `clflush` timing, with NUMA being the most significant factor in multi-socket systems.
— We show that, on modern systems, accurate Flush+Reload and Flush+Flush attacks require topology awareness.
— We evaluate the impact of topology on attack performance, on 36 systems.
— We show that Flush+Flush is widely applicable on x86, including multi-socket servers, and may be more accurate than Flush+Reload in many cases.
— We present a more accurate covert-channel primitive, Load/Flush+Reload, whose bandwidth can reach 2× that of Flush+Reload and Flush+Flush.

2 Background and Related Work

2.1 Modern System Memory Hierarchy

Cache hierarchy and structure Modern CPUs use several levels of caches to hide memory latency: per-core private L1 (instruction and data) and L2 caches, and a shared last-level cache (LLC) serving all cores on a chip. In multi-socket systems, each socket has a separate LLC [11]. In newer multi-chiplet designs, the LLC in each CPU is divided into multiple smaller caches, typically matched to separate chiplets, to improve cache performance. AMD calls such divisions Core Complexes (CCX) [23], while Intel calls them Sub-NUMA Clusters (SNC) [12]. Caches store 64-byte *cache blocks*, a block and its metadata form a *cache line*. *Inclusive* caches keep copies of all line in lower level caches [11]. Intel LLCs used to be inclusive, but server CPUs since Skylake-SP use non-inclusive ones [13].

Cache coherence In multicore systems with private and shared caches, cores might observe stale values in private cache. Modern CPUs enforce *cache coherence* to make all cores agree on the sequence of values taken by a memory block, using stateful protocols, with Intel and AMD using variants of the classical MESI [11]:
M: A **modified** copy — must be written back before other cores may access it[6].
E: An **exclusive** copy — the core can upgrade to the M state for free.
S: One of many **shared** copies — other copies must be invalidated before writing.
I: An **invalid** line contains no valid copy of any memory block.
Single-socket systems can maintain coherence with a global directory, but multi-socket ones need cross-socket exchanges. Inclusive LLCs often act as directories for private caches, but modern non-inclusive LLCs need separate directories [26]. Intel uses one directory per socket [17], while AMD uses per-CCX ones [7].

[6] Our attacks, as they target read-only memory, do not involve the M state.

Multi-socket systems and Numa In modern multi-socket systems, each socket has its own memory controllers, and associated DRAM. Memory latencies depend on which controller owns the requested location. A local controller fulfills requests faster than a remote one, whose requests must traverse intra-socket and inter-socket interconnects, causing *Non-Uniform Memory Access* (Numa). To optimize the performance, such systems are divided into Numa-nodes: Each Numa-node groups physical cores with similar memory latencies, and the closest memory controller(s), *i.e.*, with the best memory latency. Hence, operating systems often support thread and memory placement requests from processes. The Linux kernel also dynamically rebalances Numa-nodes, optimizing performance by migrating virtual pages from a node to another.

Coherency between LLCs in separate sockets is handled by a cache-coherence-Numa (ccNuma) directory stored in main memory [16]. This directory stores the remote state of lines and whether a coherency operation is required on read and write requests, only write requests, or on no requests. Each CPU also contains a memory directory cache which is used to reduce the performance impact of frequently checked lines. On AMD CPUs with a number of separate LLCs within each socket, coherency is handled by a cross-CCX coherency directory located on the central I/O die [24]. Accesses missing in the local CCX but marked as residing in another CCX are served from the remote CCX instead of main memory.

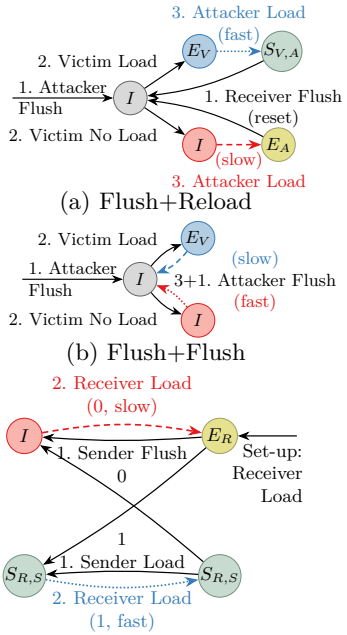


Fig. 3: Cache coherence transitions of the 3 primitives. Distinct attacker/victim physical cores; fast transitions dotted; slow transitions dashed; arrow colors match histograms. I: *Invalid*; E_C: *Exclusive* on core C; $S_{C,D}$: *Shared* on C, D.

2.2 Micro-architectural Attacks using `clflush`

Micro-architectural attacks violate the confidentiality guarantees intended by process isolation, by exploiting the ISA's implementation via timing or performance counters. We distinguish *side channels i.e.*, an attacker measuring data leaked in the micro-architectural state by unsuspecting process; from *covert channels i.e.*, two attackers cooperating to stealthily communicate across isolation boundaries.

The *unprivileged* `clflush` x86 instruction evicts a cache line from the whole coherence domain, *Invalidating* it [13]. It facilitates cache attacks that monitor a victim's cache block accesses within shared, usually read only, memory.

In **Flush+Reload** [27], the attacker first flushes the target cache line, and waits for the victim to, possibly, read the line. If the victim accesses the line, it transitions into a Victim-Exclusive state, otherwise the line remains Invalid. The attacker then times a load to that line, distinguishing the two possible coherence

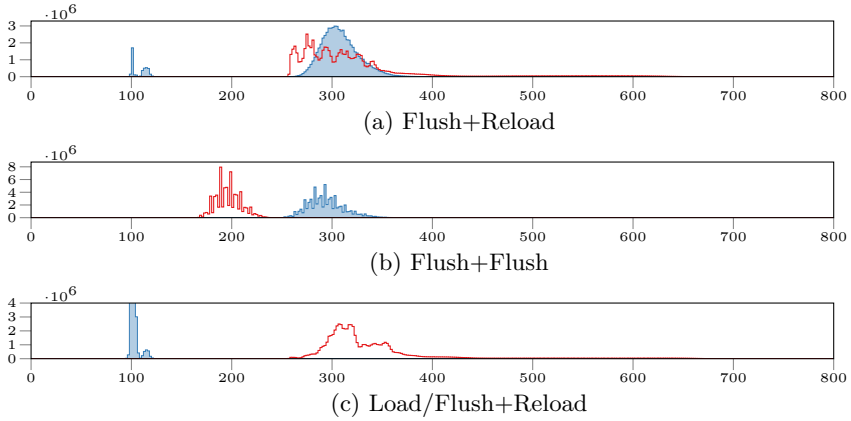


Fig. 4: All 3 topology-unaware histograms, on `EMR`, (1× Emerald Rapids).

states. For distinct attacker and victim, the line transitions respectively into a Shared or an Attacker-Exclusive state. The attacker then repeats the process. Figure 3a presents the coherence transitions for this attack.

Flush+Flush [9] exploited `clflush`'s execution time dependency on the state of its target cache line, on Intel CPUs. This avoids the Reload step of Flush+Reload, by timing the `clflush` instruction, measuring and resetting the state at the same time. The target line state thus alternates between Invalid and Victim-Exclusive, the latter after a victim memory access, as shown by Figure 3b. Single-socket AMD Zen 2 client CPUs have recently been found vulnerable [14].

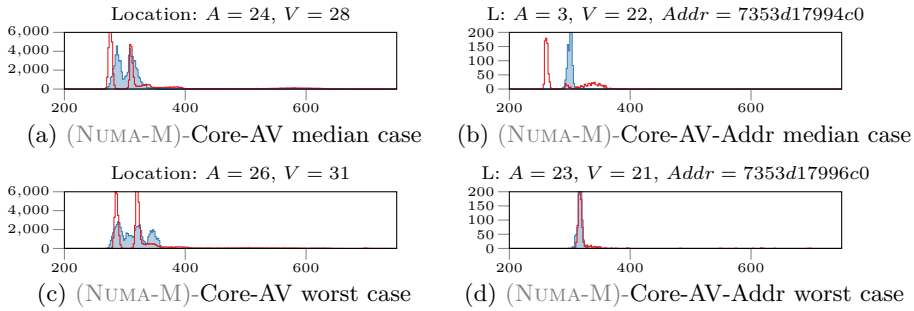
2.3 Calibration of Timing Attacks

Cache-based side and covert channels rely on timing measurements to infer whether data was cached. In both cases, the execution time must be classified into one of the possible outcomes. The simplest method is to compare the measured execution time to a threshold: intuitively, a cache hit is fast and a miss is slow. However, `clflush` is often faster when its operand is not cached.

In more complex systems, the timing distribution may not be cleanly separable, and classification may benefit from using a range defined by two thresholds rather than a single one. To select optimal thresholds, we collect representative timing data for each outcome and analyze the corresponding histograms of execution times. When multiple factors influence timing, such as which cores the attacker and victim execute on, or the NUMA node to which the target memory belongs, classification must account for each configuration independently. This approach, which we call *topology-aware calibration*, assigns separate classifiers (thresholds) to each relevant attack configuration to minimize misclassification rates.

3 Methodology

Our goal is to characterize how memory hierarchy topology affects cache attacks on modern servers by exploring all parameters influencing latencies. One such



(a) (Numa-M)-Core-AV median case (b) (Numa-M)-Core-AV-Addr median case

(c) (Numa-M)-Core-AV worst case (d) (Numa-M)-Core-AV-Addr worst case

Fig. 5: Flush+Reload (ST) worst and median case on `EMR`, for two attacker models.

parameter is the memory controller that serves cache misses in Numa systems. In a two-socket system, accessing a remote memory controller adds latency due to crossing the inter-chip interconnect and traversing two on-chip networks.

We call *topology* the set of all the variability sources induced by the different paths in the memory hierarchy. This covers the respective physical cores on which the **attacker** (A) and **victim** (V) processes run, the **memory** controller (M) hosting the target data, and the latency caused by the internal organization of caches, e.g., the cache slice on Intel CPUs. We use the virtual **address** ($Addr$) of the target cache block as a proxy to this factor, as the hashing functions used by the LLCs of our machines are not generally known, and attackers usually cannot access the required physical addresses. Three research questions thus follow from our goal: *Which attack is applicable on each micro-architecture?* (**RQ1**) *What is the impact of topology on attack accuracy, and what gains can be obtained with topology awareness?* (**RQ2**) *What covert-channel true capacity can be achieved using the best trade-off between accuracy and attack speed?* (**RQ3**)

We define a *configuration* as a tuple, $(A, V, M, Addr)$, of the attacker and victim physical core and socket, target memory, and virtual address. Our experiments iterate over the entire set of possible configurations, and make a series of measurements for each. To answer these questions, we run two sets of experiments: The first one measures latencies for hits and misses in each primitive, building histograms such as Figure 2, to answer **RQ1** and **RQ2**. The second one evaluates the performance of covert channels, in an end-to-end setting, answering **RQ3**.

3.1 Attack Primitives

Table 1: Operations in the *Calibration* experiment

Operation	Primitive use	Victim	Attacker	
Load I	F+R & LF+R (miss)	—	⏱ Load	`clflush`
Load E_V	F+R (hit)	Load	⏱ Load	`clflush`
Load $S_{A,V}$	LF+R (hit)	Load	⏱ Load	—
`clflush` I	F+F (miss)	—	⏱ `clflush`	—
`clflush` E_V	F+F (hit)	Load	⏱ `clflush`	—

We consider three attack primitives: First, the classical Flush+Reload (FR) and Flush+Flush (FF), already described previously (Section 2.2). In addition, we introduce Load/Flush+Reload (LFR), a covert-channel primitive which rearranges the elements of Flush+Reload, resulting in a different set of cache coherence transitions. While Flush+Reload compares the execution time of a load from

an Invalid state (I) to a load from an Exclusive state on the victim/sender core (E_V), this primitive compares the execution time of an Invalid load (I), with that of a load from a Shared state ($S_{R,S}$) between the sender and the receiver. The receiver only issues loads, while the sender encodes 1s as loads and 0s as flushes, an unlikely behavior for innocent victims, hence the limitation to covert-channel settings. Since the receiver has previously loaded, the sender's action brings the cache line into either the Invalid or the Shared state. The attacker's loads leave Shared states unchanged, and turn Invalid states into a Receiver-Exclusive states. This results in a much faster attack, makes transmitting series of ones inexpensive, and increases the timing difference between the two transitions.

3.2 Attacker Models

Table 2: Attacker Models in this paper.

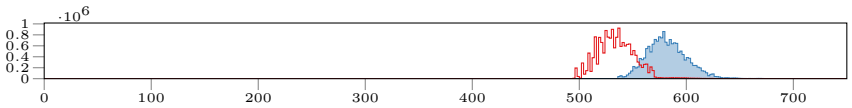
Attacker Model	Memory		Attacker (A)		Victim (V)	
	NUMA node (M)	Address ($Addr$)	NUMA node	Core	NUMA node	Core
Topology-Unaware (TU)	U ❓	U ❓	U ❓	U ❓	U ❓	U ❓
NUMA-AVM	K ✏	U ❓	K ✏	U ❓	K ✏	U ❓
NUMA-AVM-Addr	K ✏	K ✏	K ✏	U ❓	K ✏	U ❓
NUMA-M-Core-AV	K ✏	U ❓	K ✏	K ✏	K ✏	K ✏
NUMA-M-Core-AV-Best	C ✅	U ❓	C ✅	C ✅	C ✅	C ✅
NUMA-M-Core-AV-Addr (TA)	K ✏	K ✏	K ✏	K ✏	K ✏	K ✏
NUMA-M-Core-AV-Addr-Best (Best-TA)	C ✅	C ✅	C ✅	C ✅	C ✅	C ✅

We identify above several parameters that influence the latencies of hits and misses in cache attack primitives. The attacker aims to distinguish two *outcomes*: *hits* and *misses*. Hence, she wants the distribution of timings for hits and misses to be well separated. However, if she accounts for none of these topology parameters, the distributions get wider and separation may worsen. The attacker thus wants to account for as many parameters as possible to narrow those distributions, hopefully making *classification* easier. We consider two classifiers: *thresholds*, and *ranges*, respectively denoted **ST** (*single threshold*) and **DT** (*dual threshold*). A *threshold* maps values below to one outcome, and above to the other, while a *range* maps values within its two thresholds, to one outcome, and those outside to the other.

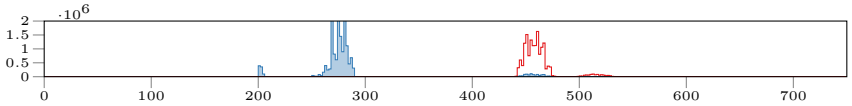
Depending on the knowledge and control the attacker has over these parameters, we define various *attacker models*, where each parameter can be *Unknown* U ❓ (the same classifier is used for all possible values of the parameter), *Known* K ✏ (a distinct classifier is used for each value), or *Chosen* C ✅, (the attacker selects the parameter's value to get the best average result).

On Linux, the NUMA node of the target memory (M) can be pinned using `libnuma`, which provides an ease-of-use interface on top of the *unprivileged* Linux system calls, a feature thus available to attackers. Distinct threshold values per address ($Addr$) can be used when known. Covert-channel attackers may choose $Addr$, while side-channel attackers usually cannot[7]. The attacker and victim [22] cores can be selected with the `sched_setaffinity` system call, a more powerful interface than the attacker and victim NUMA-node selection from `libnuma`. Additionally, `/proc/pid` also exposes the victim's core affinity and the

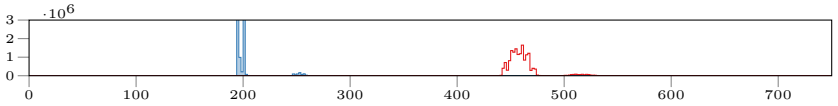
[7] Future work could try using repeated NUMA migrations to select a better address.



(a) Flush+Flush (`rdpru`)

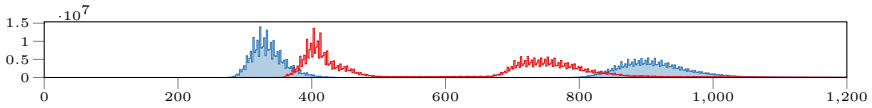


(b) Flush+Reload (`rdpru`)

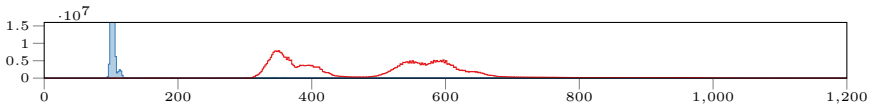


(c) Load/Flush+Reload (`rdpru`)

Fig. 6: All 3 topology-unaware histograms on `Zen5` (1× Granite Ridge)



(a) Flush+Flush



(b) Load/Flush+Reload

Fig. 7: Extra topology-unaware histograms on `esterel41` (2× Sapphire Rapids)

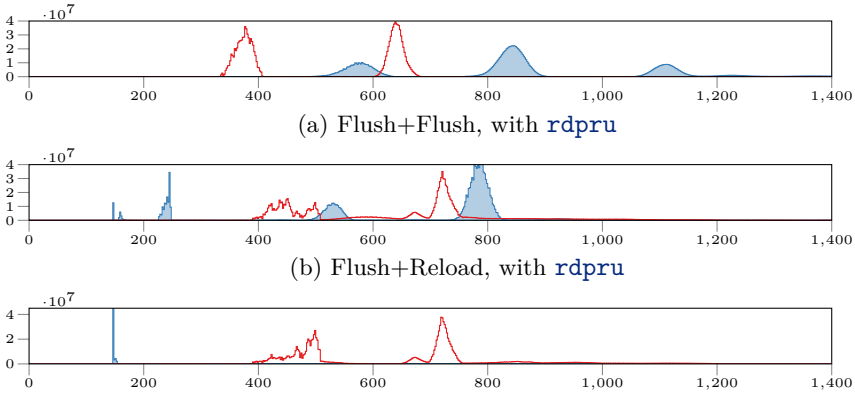
last core used. Consequently, all these parameters can be realistically controlled by an attacker. We thus define the attacker models in Table 2.

3.3 Experiment 1: Calibration

This first experiment collects latency data for each primitive's hits and misses. It iterates over the space of $(A, V, M, Addr)$ configurations. We successively pin our process's memory to each NUMA-node M, and iterate over all (A, V) pairs of cores, and all cache lines in a 4 KiB page, using `sched_setaffinity` and `libnuma`. For each such configurations, we then take measurements for a set of *operations*, *i.e.*, a victim execution to set up the cache state followed by the attacker execution of the primitive. Three primitives and two outcomes each should give six different operations, but, as Flush+Reload and Load/Flush+Reload both have a load to an invalid line as their miss, we only need five operations, listed in Table 1, with *Load I* shared by these two primitives. For each operation, we take 1024 measurements after 128 discarded warm-up iterations.

This results in a large set of histograms, with one 1024-sample histogram of the latencies per operation and configuration[8]. Combining the hit and miss

[8] We truncate very tall and narrow peaks in plots so the remainder can be read.



(a) Flush+Flush, with `rdpru`

(b) Flush+Reload, with `rdpru`

(c) Load/Flush+Reload, with `rdpru`

Fig. 8: All 3 topology-unaware histograms on `musa` (2× Zen 4 Genoa)

operations results in per primitive histograms, used to estimate the error rate of classifiers, and identify the best one. Following the attacker model, we select and sum up all histograms sharing a classifier, and compute the average (*Avg.*) error along with the minimum (*Min.*), maximum (*Max.*), median (*Med.*), and quartiles (*Q1* and *Q3*) of the error rates of the underlying series of histograms.

These predictions give an optimistic estimate of the error observed in a real attack: Our measurements, executed with no other activity on the system benefit from the DRAM row remaining open, speeding up cache misses. In a real attack, this might not be the case, and the branch predictor state could also vary and induce more variability. This limitation motivates the second experiment, which evaluates the resulting classifiers in an end-to-end attack.

3.4 Experiment 2: Covert Channel Benchmark

To answer **RQ3**, this experiment evaluates end-to-end attacks, measuring the practical bandwidth and error rates of covert channels built with each primitive. We transmit 1024 bits between two threads, over a channel using n cache lines in separate pages, in a round-robin fashion. This is repeated 16 times per configuration, giving an average and a standard deviation. Exploring values of n from 1 to 10, we found 3 lines to be the best tradeoff between bandwidth and accuracy.

We use a generic implementation, parametrized with the primitive. It uses the first experiment's algorithm to self-calibrate and select the classifier used. Because of that in-situ calibration, it requires a longer runtime than the first experiment. Due to the longer runtime, we cover a smaller set of attacker models, with the following shorthands: **TU** (Topology-Unaware), **TA** (Topology-Aware) **Best-TA** (Best Topology-Aware), see Table 2. The Best-TA model selects its configuration only to minimize the error rate, which may result in a lower bandwidth.

To speed up experiments, we use the systems' symmetries to reduce the number of configuration to be tested, pinning memory to a single NUMA-node and deduplicating equivalent hyper-threads. Section 4.3 verifies these assumptions.

To answer **RQ3**, Section 5 presents error rates, p, raw bandwidths, C, and true capacities, T, defined as [18]: $T = C \times (1 + (1-p)\lg(1-p) + p\lg(p))$

3.5 Experimental Setup

We run our experiments on 36 machines, described in Appendix A, with multi-socket ones on Debian 11, and single-socket ones on a mix of Ubuntu releases.

To allow reproduction, unless specified, results were obtained dynamic frequency scaling (DVFS) and prefetchers disabled. We include a second set of results with those features enabled in their default configuration in our online supplement *cf.* Appendix A. Lastly, fences and mutual exclusion prevent race conditions to enforce correct ordering of operations.

In this section, we have thus described, our three research questions, **RQ1**, **RQ2**, and **RQ3**, and described the two experiments we use to answer them. Section 4 thus answers **RQ1** and **RQ2**, while Section 5 answers **RQ3**.

4 Results for Experiment 1: Calibration

This section studies **RQ1**: "Which attack is applicable on each micro-architectures?" and **RQ2**: "What is the impact of topology in attack accuracy, and what gains can be obtained with topology awareness?"To do so, we measured the latency distributions of hits and misses for each primitive (Section 3.1) under each $(A, V, M, Addr)$ configuration, and build histograms under each attacker model (Table 2). Table 9, in Appendix A, includes a summary of the average predicted error rates for all 36 machines under the topology unaware (TU), the Numa-M-Core-AV-Addr (TA), and the Numa-M-Core-AV-Addr-Best (Best-TA) models. We first discuss the results of single-socket server Intel CPUs. Then, after verifying the symmetry assumptions needed for Experiment 2, we discuss results on two-socket Intel servers and two-socket AMD servers, illustrated with the most recent matching machines. (Appendix B shows four-socket Flush+Reload).

On single-socket systems, Numa is not a concern, the benefits of topology awareness were demonstrated for Flush+Flush on Coffee Lake-R[9] client CPUs [5]. We thus focused on the latest Intel server architecture, Emerald Rapids, and sampled more AMD micro-architectures, as they had much less Flush+Flush coverage. Appendix A shows the error rate predictions for all 36 machines.

4.1 Calibration of Intel Server CPUs

Emerald Rapids (2024) is Intel's latest server CPU, a minor evolution of Sapphire Rapids. Figure 4 shows the topology-unaware histograms for all three primitives. For loads, cache misses overlap with exclusive hits in another core (250–400 cycles), making topology-unaware Flush+Reload ineffective. Flush+Flush exhibits good separation, and Load/Flush+Reload an even better one.

[9] Skylake 4$^{\text{th}}$ refresh; we confirm Arrow Lake still behaves similarly *cf.* Appendix A.

Table 3: Error rates (%) for EMR (1× Emerald Rapids), -ST and -DT specified when results differ.

Model	Avg.	Min.	Q1	Med.	Q3	Max.
TU-FF	0.05	< 0.01	< 0.01	< 0.01	< 0.01	50.05
TU-FR-ST	41.34	9.13	25.54	38.82	50.00	100.00
TU-FR-DT	33.67	1.27	18.80	29.88	42.24	99.80
TU-LFR	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	2.44
(Numa-M)-Core-AV-FF	0.02	< 0.01	< 0.01	< 0.01	< 0.01	45.90
(Numa-M)-Core-AV-FR-DT	19.74	< 0.01	6.93	16.89	28.12	90.09
(Numa-M)-Core-AV-FF-Best	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
(Numa-M)-Core-AV-FR-Best	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.05
(Numa-M)-Core-AV-LFR-S-Best	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
(Numa-M)-Core-AV-Addr-FF-ST	0.01	< 0.01	< 0.01	< 0.01	< 0.01	23.78
(Numa-M)-Core-AV-Addr-FF-DT	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	3.12
(Numa-M)-Core-AV-Addr-FR-DT	7.79	< 0.01	1.66	5.86	11.91	43.41
(Numa-M)-Core-AV-Addr-FR-Best	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01

Table 3 presents results under various attacker models where the added parameters make a difference. Flush+Flush and Flush+Reload improve with more detailed models, but Flush+Reload remains much noisier than alternatives. Figure 5 shows examples of (Numa-M)-Core-AV and (Numa-M)-Core-AV-Addr median and worst case histograms, exhibiting the issue. Here Flush+Reload is noisier than Flush+Flush, itself noiser than Load/Flush+Reload.

Overall, all three attacks work on single-socket Intel servers (**RQ1**). Flush+Flush sees small improvements with topology awareness, while Flush+Reload sees significant ones, going from 41% to 7.8% under (Numa-M)-Core-AV-Addr. Selecting the best configuration renders the error rate negligible. Topology awareness improves Flush+Reload majorly on single-socket Intel servers (**RQ2**).

4.2 Calibration of AMD CPUs

On AMD CPUs, `rdtsc` has a coarse granularity, we use the more accurate `rdpru` instruction with the `APERF` MSR [8,15], when available. Figure 6 shows topology-unaware histograms for all three primitives. Table 4 shows Flush+Reload and Flush+Flush both work, with measurable error rates, while Load/Flush+Reload is much more reliable.

Table 4: Error rates (%) for Zen5 (1× Granite Ridge), single threshold

Model	Avg.	Min.	Q1	Med.	Q3	Max.
TU-FF	8.67	< 0.01	< 0.01	0.05	0.63	50.34
TU-FR	3.19	< 0.01	< 0.01	0.10	0.20	50.10
TU-LFR	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.15
(Numa-M)-Core-AV-FF	5.54	< 0.01	< 0.01	0.05	0.20	50.24
(Numa-M)-Core-AV-FR	3.14	< 0.01	< 0.01	0.10	0.20	61.04
(Numa-AVM)-Addr-FF	0.98	< 0.01	< 0.01	< 0.01	0.05	43.21
(Numa-AVM)-Addr-FR	3.19	< 0.01	< 0.01	0.10	0.20	50.10
(Numa-M)-Core-AV-FF-Best	0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.15
(Numa-M)-Core-AV-FR-Best	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.10
(Numa-M)-Core-AV-LFR-Best	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
(Numa-M)-Core-AV-Addr-FF	0.38	< 0.01	< 0.01	< 0.01	0.05	27.59
(Numa-M)-Core-AV-Addr-FR	3.01	< 0.01	< 0.01	0.10	0.20	49.76
(Numa-M)-Core-AV-Addr-FF-Best	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
(Numa-M)-Core-AV-Addr-FR-Best	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01

Overall, Flush+Flush works on all AMD systems, and on some micro-architecture, such as Zen 2, Flush+Flush is more reliable than Flush+Reload, *cf.* Appendix A. The accuracy of both primitives tends to be correlated: some micro-architecture exhibit high error rates for both, others low error rates for both; the differences between primitives are smaller than between micro-architectures.

Table 5: Results for for `esterel41` (2× Sapphire Rapids)

(a) Error rates (%)

Model	Avg.	Min.	Q1	Med.	Q3	Max.
TU-FF-DT	7.69	< 0.01	0.05	1.27	11.91	100.00
TU-FR-DT	31.86	< 0.01	6.05	33.40	50.00	100.00
TU-LFR-ST	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	23.05
Numa-M-Core-AV-FF-ST	3.80	< 0.01	< 0.01	< 0.01	2.25	78.71
Numa-M-Core-AV-FR-ST	16.48	< 0.01	< 0.01	1.51	35.60	100.00
Numa-M-Core-AV-LFR-ST	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	6.98
Numa-M-Core-AV-FF-Best	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
Numa-M-Core-AV-FR-Best	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
Numa-M-Core-AV-LFR-Best	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
Numa-M-Core-AV-Addr-FF	1.53	< 0.01	< 0.01	< 0.01	0.15	41.06
Numa-M-Core-AV-Addr-FR	6.57	< 0.01	< 0.01	0.49	8.94	47.75

(b) Covert Channel Results

Model	Primitive	p (%)	σp	C (Mbit/s)	σC	T (Mbit/s)	σT
TU	FF	26.67	23.95	1.32	0.29	0.65	0.70
	FR	30.89	20.59	1.47	0.39	0.51	0.63
	LFR	0.19	0.08	2.54	0.81	2.49	0.80
TA	FF	5.53	10.22	1.32	0.30	1.05	0.47
	FR	23.34	21.42	1.50	0.39	0.70	0.73
	LFR	0.18	0.08	2.56	0.84	2.51	0.83
Best-TA	FF	0.00	0.00	1.28	0.00	1.28	0.00
	FR	0.00	0.00	2.40	0.00	2.40	0.00
	LFR	0.05	0.00	1.67	0.00	1.66	0.00

We thus find single-socket AMD CPUs to be susceptible to all 3 attacks (**RQ1**). Flush+Flush benefits more from topology awareness than Flush+Reload, going from the noisiest to significantly more accurate, against the usual assumptions for Flush+Flush on AMD, denoting the benefits of topology awareness (**RQ2**).

Overall, all 3 primitives work on recent x86 single-socket systems (**RQ1**), and often benefit from topology awareness, with micro-architectural nuances (**RQ2**).

4.3 Are Numa Systems Symmetric?

Experiment 2 assumes symmetric system behavior with respect to Numa nodes, *i.e.*, swapping all Numa nodes (A, V, M) leaves results unchanged. Figure 2 (page 2) shows results on a 2× Sapphire Rapids system, the topology-unaware (Figure 2a) and the 8 Numa-AVM (Figure 2b) histograms. The topology-unaware one exhibits quite a complex behavior, with several wide humps. This is not surprising: Numa systems have, by definition, large memory latency variations. Figure 2b shows the memory Numa-node (M) has no impact on hit timings, as expected, while Numa is a major source of variability for misses: e.g., with $A = V$, a load is much slower if $A \neq M$ than when $A = M = V$, which must thus be accounted for in attacks. This complex behavior thus makes topology-unaware classifiers ineffective. The hump width, and the hit-miss overlap, e.g., $(A, V, M) \in \{(0, 1, 0), (1, 0, 1)\}$, hampers Flush+Reload attacks, and motivates our fine-grained topology approach, *i.e.*, one considering more than simply Numa.

Figure 2b shows exchanging all Numa nodes, e.g., from $(A, V, M) = (0, 1, 1)$ to $(1, 0, 0)$, produces similar histograms, which verifies experiment 2 assumptions. All behaviors can thus be explored with only $M = 0$, speeding up experiment 2.

4.4 Calibration of Two-socket Intel Systems

Figure 7 shows `esterel41`'s Flush+Flush and Load/Flush+Reload topology-unaware histograms, as a complement to the Flush+Reload ones already shown in Figure 2. Table 5a shows the resulting error rates under different attacker models.

While all three primitives' error rates can be reduced satisfyingly under the Numa-M-Core-AV-Best attacker model, Flush+Flush is usually slightly better than Flush+Reload. Models without choice remain exposed to excessively noisy worst cases. Load/Flush+Reload, unsurprisingly, is much more reliable.

All 3 primitives apply to Intel two-socket systems (**RQ1**), and topology awareness leads to large improvements (**RQ2**).

4.5 Calibration of Two-socket AMD Systems

For multi-socket AMD systems, Figure 8 shows the topology-unaware histograms and Table 6, a subset of the error predictions, where the parameters make a difference. Load/Flush+Reload remains the most accurate, as expected. Flush+Reload's and Flush+Flush's accuracies both benefit from topology awareness, with the average error rates respectively going from 18% and 12% in topology unaware, down to 2.1% and 0.12% with Numa-M-Core-AV, and 0.9% and 0.03% under Numa-M-Core-AV. Choosing cores, in the Numa-M-Core-AV-Best model, leads to negligible error rates. Flush+Reload is also more accurate with ranges (-DT) instead of thresholds (-ST).

Table 6: Error rates (%) for `musa` (2× Zen 4 Genoa)

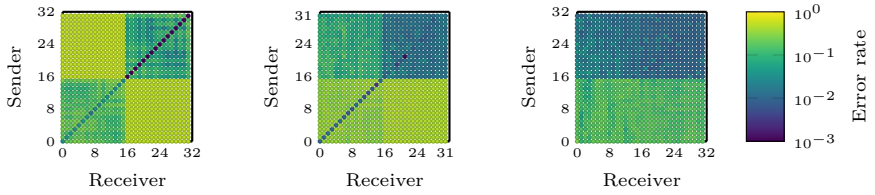
Model	Avg.	Min.	Q1	Med.	Q3	Max.
TU-FF	12.66	< 0.01	< 0.01	< 0.01	49.85	51.51
TU-FR-DT	18.01	< 0.01	6.25	8.89	22.31	90.48
TU-LFR	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	0.34
Numa-M-Core-AV-FF-ST	0.12	< 0.01	< 0.01	< 0.01	< 0.01	50.05
Numa-M-Core-AV-FR-ST	7.61	< 0.01	4.64	6.79	9.18	87.55
Numa-M-Core-AV-FR-DT	2.10	< 0.01	0.20	0.68	1.95	68.80
Numa-M-Core-AV-FF-Best	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
Numa-M-Core-AV-FR-Best	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
Numa-M-Core-AV-LFR-Best	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
Numa-M-Core-AV-Addr-FF-ST	0.03	< 0.01	< 0.01	< 0.01	< 0.01	49.56
Numa-M-Core-AV-Addr-FR-ST	6.55	< 0.01	3.86	6.54	8.94	48.39
Numa-M-Core-AV-Addr-FR-DT	0.93	< 0.01	0.05	0.39	0.93	47.36

We conclude Flush+Flush works on multi-socket AMD systems, and is worth using instead of Flush+Reload, while Load/Flush+Reload is the best covert channel (**RQ1**). Topology awareness remains needed for accurate attacks (**RQ2**).

Conclusion We have shown that all 3 primitives apply to single- and multi-socket systems (**RQ1**). Depending on the micro-architecture and attacker model, which of Flush+Flush and Flush+Reload is the most accurate may vary. However, Flush+Flush often has an edge over Flush+Reload, while Load/Flush+Reload covert channels remain the most accurate. We also observe that, overall, topology-aware approaches usually reach negligible error rates for at least one of Flush+Reload or Flush+Flush, and provide significant accuracy improvements (**RQ2**).

5 Results for Experiment 2: Covert Channel

This section answers **RQ3**, "What covert-channel true capacity can be achieved using the best trade-off between accuracy and attack speed?". We ran our generic covert channel benchmark on 3 attacker models: topology-unaware **TU**, topology-aware **TA** (Numa-M-Core-AV-Addr), and the best choice topology-aware **Best-TA** (Numa-M-Core-AV-Addr-Best). For each machine, we report p, the error rate; C, the bandwidth; and T, the *true capacity*, *cf.* Section 3.4.



(a) Flush+Reload, TU (b) Flush+Reload, TA (c) Flush+Flush, TA

Fig. 9: Core to core covert channel error rates for `esterel41` (log. scale)

Table 7: Other results for covert channels

Machine	Attack	Attacker Model	p (%)	C (Mbit/s)	T
`ARL` 1× Arrow Lake	FF	TU	0.00	3.20	3.20
	FR	TU	0.01	2.73	2.73
	LFR	TU	0.01	5.43	5.42
`Zen5` 1× Zen 5 (Granite Ridge)	FF	TU	24.58	2.22	0.46
	FF	TA	19.73	2.21	0.64
	FF	Best-TA	0.67	2.07	1.95
	FR	TU	0.56	2.30	2.23
	FR	TA	1.12	2.29	2.13
	FR	Best-TA	0.00	2.27	2.27
	LFR	TU	0.04	4.89	4.87
	LFR	Best-TA	0.00	4.98	4.98
`EMR` 1× Emerald Rapids	FF	TU	0.06	2.07	2.06
	FR	TU	55.47	1.68	−−
	FR	TA	34.97	1.74	0.42
	FR	Best-TA	0.00	2.20	2.20
	LFR	TU	0.19	3.28	3.22
`musa` 2× Zen 4 (Genoa)	FF	TU	43.67	1.13	0.03
	FF	Best-TA	42.33	1.88	0.03
	FR	TU	36.85	1.11	0.13
	FR	Best-TA	0.00	1.88	1.88
	LFR	TU	0.20	2.13	2.08
	LFR	Best-TA	0.00	2.92	2.92

Table 5b shows results for a two-socket Intel server. Flush+Flush and Flush+Reload benefit steadily from more detailed models, with the former being more accurate, respectively from 26.67%, 0.65 Mbit/s, and 30.89%, 0.51 Mbit/s, to 1.28 Mbit/s and 2.40 Mbit/s, with negligible error. Load/Flush+Reload beats both in accuracy and capacity, with only marginal improvement with topology awareness (2.51 Mbit/s). It also reflects how our *Best* model is chosen for accuracy and not true capacity, hence the capacity decrease in the Best-TA model, as the accuracy gains fail to compensate the bandwidth loss. Overall here, we see that Flush+Flush performs better than Flush+Reload, while Load/Flush+Reload performs best. Figure 9c shows the accuracy of the covert channel depending on the respective physical cores of the sender (victim) and the receiver (attacker), with memory pinned on the first Numa node (lowest numbered cores). It highlights the topology effect, with the sibling-hyperthread case showing up as a diagonal. It also shows how Flush+Flush is the best cross-socket attack on Intel servers.

Table 7 shows that topology awareness usually improves Flush+Flush and Flush+Reload attacks significantly, but Flush+Flush is not as competitive on AMD platforms. Load/Flush+Reload stays the best covert channel, even doubling, without topology awareness the best bandwidth of alternatives, e.g., on `Zen5` 4.9 Mbit/s, vs. Flush+Flush's 1.95 Mbit/s and Flush+Reload's 2.3 Mbit/s. It

also shows AMD machines have more complex behaviors for Flush+Reload and Flush+Flush. Figure 10 exhibits for Flush+Reload, a complex structure which reflects the multi-chip structure of this 2× AMD Zen 4 system, with the diagonal blocks reflecting the CCX structures. However, Flush+Reload cross-socket attacks remain possible, in the $(A = 1, V = 0, M = 0)$ quadrant.

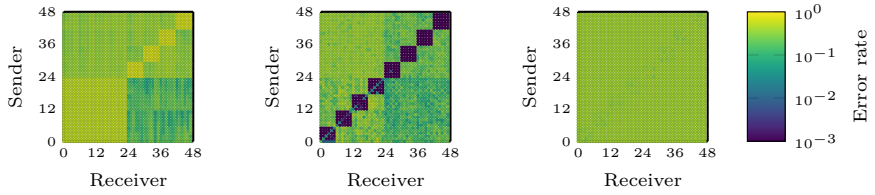
The observed error rates tend to exceed experiment 1's predictions, which are optimistic bounds, but remain consistent: machines with high predicted error rates also have high error rates (and even unexploitable topology-unaware Flush+Reload for `EMR`), while machines where we expect low error rates produce low error rates (e.g., `ARL`). However, our best error rates reported tend to be below the uncertainty of the measure, given the number of samples we have, which sometimes explain a slight increase in error rate when increasing the model quality. One further source of noise, is that the transmission uses several cache lines in parallel, to maximize the bandwidth. This could interfere with the individual latencies, and contribute to the degradation in accuracy.

Overall, Load/Flush+Reload is an effective covert channel, with a capacity around 5 Mbit/s for single socket, or above 2.5 Mbit/s for multi-socket systems. Topology awareness improves other primitives' accuracy, but may reduce the raw bandwidth, resulting in machine-dependent trade-offs. Some topology awareness is needed to make attacks viable, but the model with the best predicted error rate might not be the one with the best true capacity. We conclude to **RQ3** that while the tradeoffs vary depending on the system, an attack bandwidth in the Mbit/s range is usually achievable with at least one of Flush+Flush and Flush+Reload with suitable topology awareness, and that Load/Flush+Reload covert channels usually outperform either of these, by a factor of ×1.5 or even ×2.

6 Discussion

6.1 Limitations

While we study a wide range of architectures, several limitations remain. First, the measured covert-channel bandwidths do not always match our timing-based predictions, possibly due to limited sampling in some configurations. Additional measurements would be needed to refine error-rate estimates. Second, our calibration process tends to keep a DRAM row open, which may lower observed miss



(a) Flush+Reload, TU (b) Flush+Reload, TA (c) Flush+Flush, TA

Fig. 10: Core to core covert channel error rates for `musa` (log. scale)

latencies. It also executes on a system with very little memory traffic, which reduces the accuracy under higher memory traffic settings, such as our multi-cache line covert channel. Future work should develop sampling methods that mitigate these biases. Finally, the applicability of topology-aware calibration must be considered in the context of the attacker model. In native scenarios, an attacker can often control its own Numa node or core placement and influence the victim through the scheduler. In contrast, such control may be limited or unavailable in other threat models, including virtualized environments, or web browsers.

6.2 Future Work

More elaborate classification strategies than threshold and ranges could be investigated. Rauscher et al. [21] introduced metrics to quantify the quality of attack primitives, but these only account for topology-unaware approaches, with a single global threshold. Future work could adapt such metrics to more complex classifiers and apply them to complex systems that require topology awareness.

Our data also show complex behavior on AMD CPUs with multiple compute dies, which justifies further reverse engineering of AMD's cache hierarchy and topology, which is not as well known as Intel's [4,1].

Generalizing this work to other isas, e.g., ARM, assessing topology awarene attacks would be valuable, applied to Evict+Reload and Prime+Probe, and even, with Risc-V's *Zicbom* extension, Flush+Reload and Flush+Flush [2].

Lastly, future work could evaluate Evict+Reload, Prime+Probe and attacks based on `prefetchw` [10] on more recent and multi-socket systems. Prime+Probe is unlikely to work in systems sharing neither a llc nor a cache directory, such as multi-socket systems, but this assumption should be verified.

7 Conclusion

We evaluated the performance of Flush+Flush and Flush+Reload on different 36 systems, both from AMD and Intel, over a large range of micro-architectures, mostly recent ones. We proved Flush+Flush and Flush+Reload to be widely applicable to all of them, including multi-socket and AMD systems, but that accurate attacks may require topology awareness. In particular, on Numa-system, Numa-node rebalancing must be avoided to get reliable attacks, and Numa-awareness is critical to accurate attacks. In addition, we presented Load/Flush+Reload, a new fast and accurate covert-channel primitive that achieves a 1.5× true capacity improvement over Flush+Reload and Flush+Flush. Overall, topology awareness is essential for reliable cache attacks on modern x86 architectures.

References

1. Andreas Abel and Jan Reineke. nanobench: A low-overhead tool for running microbenchmarks on x86 systems. In *ISPASS*, 2020.

2. Cédrick Austa, Jan Tobias Mühlberg, and Jean-Michel Dricot. Systematic assessment of cache timing vulnerabilities on RISC-V processors. In *ESORICS*, 2025.
3. Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. Adding virtualization capabilities to the Grid'5000 testbed. In *Communications in Computer and Information Science*. 2013.
4. Miles Dai, Riccardo Paccagnella, Miguel Gomez-Garcia, John McCalpin, and Mengjia Yan. Don't Mesh Around: Side-Channel Attacks and Mitigations on Mesh Interconnects. In *USENIX Security*, 2022.
5. Guillaume Didier and Clémentine Maurice. Calibration Done Right: Noiseless Flush+Flush Attacks. In *DIMVA*, 2021.
6. Guillaume Didier, Clémentine Maurice, Antoine Geimer, and Walid J. Ghandour. Characterizing Prefetchers using CacheObserver. In *SBAC-PAD*, 2022.
7. Mark Evers, Leslie Barnes, and Mike Clark. The AMD Next-Generation "Zen 3" Core. *IEEE Micro*, 42:7–12, 05 2022.
8. Stefan Gast, Jonas Juffinger, Martin Schwarzl, Gururaj Saileshwar, Andreas Kogler, Simone Franza, Markus Köstl, and Daniel Gruss. SQUIP: exploiting the scheduler queue contention side channel. In *IEEE S&P*, 2023.
9. Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. Flush+Flush: A Fast and Stealthy Cache Attack. In *DIMVA*, 2016.
10. Yanan Guo, Andrew Zigerelli, Youtao Zhang, and Jun Yang. Adversarial prefetch: New cross-core cache side channel attacks. In *IEEE S&P*, 2022.
11. John L. Hennessy and David A. Patterson. *Computer Architecture - A Quantitative Approach*. Morgan Kaufmann, 6th edition edition, 2019.
12. Intel Corporation. *Intel 64 and IA-32 Architectures Optimization Reference Manual: Volume 1*. Intel Corporation, April 2024.
13. Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual*, 2025.
14. Danping Li, Ziyuan Zhu, Jiao Shen, Yusha Zhang, Gang Shi, and Dan Meng. Flush+Revisit: A Cross-CCX Side-Channel Attack on AMD Processors. In *IEEE TrustCom*, 2023.
15. Moritz Lipp, Daniel Gruss, and Michael Schwarz. AMD prefetch attacks through power and time. In *USENIX Security*, 2022.
16. Kevin Loughlin, Stefan Saroiu, Alec Wolman, Yatin A. Manerkar, and Baris Kasikci. Moesi-prime: preventing coherence-induced hammering in commodity workloads. In *ISCA*, 2022.
17. Clémentine Maurice, Nicolas Le Scouarnec, Christoph Neumann, Olivier Heen, and Aurélien Francillon. Reverse engineering intel last-level cache complex addressing using performance counters. In *RAID*, 2015.
18. Hamed Okhravi, Stanley Bak, and Samuel T. King. Design, implementation and evaluation of covert channel attacks. In *IEEE International Conference on Technologies for Homeland Security (HST)*, 2010.
19. Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of AES. In *CT-RSA*, 2006.
20. Colin Percival. Cache missing for fun and profit. In *BSDCan*, 2005.
21. Fabian Rauscher, Carina Fiedler, Andreas Kogler, and Daniel Gruss. A systematic evaluation of novel and existing cache side channels. In *NDSS*. The Internet Society, 2025.

22. Thomas Rokicki, Clémentine Maurice, Marina Botvinnik, and Yossi Oren. Port contention goes portable: Port contention side channels in web browsers. In *ASIA CCS*, 2022.
23. David Suggs, Mahesh Subramony, and Dan Bouvier. The amd "zen 2" processor. *IEEE Micro*, 2020.
24. Stavros Volos, Cédric Fournet, Jana Hofmann, Boris Köpf, and Oleksii Oleksenko. Principled microarchitectural isolation on cloud cpus. In *ACM CCS*, 2024.
25. Wenjie Xiong and Jakub Szefer. Survey of transient execution attacks and their mitigations. *ACM Computing Surveys*, 2021.
26. Mengjia Yan, Read Sprabery, Bhargava Gopireddy, Christopher W. Fletcher, Roy H. Campbell, and Josep Torrellas. Attack directories, not caches: Side channel attacks in a non-inclusive world. In *IEEE S&P*, 2019.
27. Yuval Yarom and Katrina Falkner. Flush+Reload: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *USENIX Security*, 2014.

Acknowledgments

Multi-socket experiments used the Grid'5000 test-bed [3], supported by a scientific interest group hosted by Inria, including CNRS, RENATER, several universities, and other organizations (*cf.* `https://www.grid5000.fr`). 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), and from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972. We thank also Stefan Gast for the results on the `lab32` (Zen4c) machine, and Jan Reineke for his advice.

A List of All Machines and Detailed Results

Table 8 and 9 lists the 36 machines we studied, and the average predicted error rates under the TU, TA, and Best-TA ST attacker models, for all primitives. An online supplement on Zenodo (`https://doi.org/10.5281/zenodo.20807693`), presents the full results of calibration and covert-channel benchmark, under all attacker models, with both fixed frequency and variable frequency settings. It includes a PDF with detailed results and plots for every machine, the raw experimental data set ((>20 GiB)), both licensed under CC-BY-SA v4.0 or later, and the source code, licensed under GPL v3.0 or later, also published at `https://codeberg.org/GuillaumeDIDIER/flush-based-cache-attacks-modern-x86`.

B Additional Results on a Four Socket System

Figure 11 shows similar but more complex results, as extra mode arise when V, A, M are all distinct sockets, which is impossible with only two sockets. **RQ1**, **2**, and **3** conclusions remain unchanged, *cf.* online supplement.

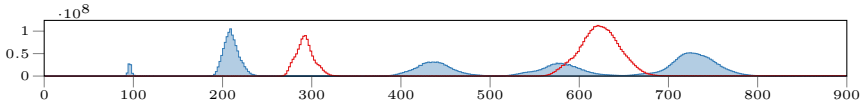


Fig. 11: Flush+Reload TU histogram for `yeti` (4× Skylake SP)

Table 8: Machines used in this work

Name	Processors	μ-arch	N	C/T	OS
abacus25	2× AMD EPYC 7413	Zen 3 (Milan)	2	2× 24/48	Deb. 11
chiclet	2× AMD EPYC 7301	Zen 1 (Naples)	2	2× 16/32	Deb. 11
chifflot	2× Intel Xeon Gold 6126	Skylake SP	2	2× 12/24	Deb. 11
chirop	2× Intel Xeon Platinum 8358	Ice Lake SP	2	2× 32/64	Deb. 11
dahu	2× Intel Xeon Gold 6130	Skylake SP	2	2× 16/32	Deb. 11
econome	2× Intel Xeon E5-2660	Sandy Bridge EP	2	2× 8/16	Deb. 11
ecotype	2× Intel Xeon E5-2630L v4	Broadwell EP	2	2× 10/20	Deb. 11
esterel41	2× Intel Xeon Gold 6426Y	Sapphire Rapids	2	2× 16/32	Deb. 11
graffiti	2× Intel Xeon Silver 4110	Skylake SP	2	2× 8/16	Deb. 11
grappe	2× Intel Xeon Gold 5218R	Cascade Lake SP	2	2× 20/40	Deb. 11
grele	2× Intel Xeon E5-2650 v4	Broadwell EP	2	2× 12/24	Deb. 11
grue	2× AMD EPYC 7351	Zen 1 (Naples)	8	2× 16/32	Deb. 11
kinovis	2× Intel Xeon Gold 6442Y	Sapphire Rapids	2	2× 24/48	Deb. 11
mercantour2	2× Intel Xeon E5-2650 v2	Ivy Bridge EP	2	2× 8/16	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
nova	2× Intel Xeon E5-2620 v4	Broadwell EP	2	2× 8/16	Deb. 11
paradoxe	2× Intel Xeon Gold 5320	Ice Lake SP	2	2× 26/52	Deb. 11
parasilo	2× Intel Xeon E5-2630 v3	Haswell EP	2	2× 8/16	Deb. 11
petitprince	2× Intel Xeon E5-2630L	Sandy Bridge EP	2	2× 6/12	Deb. 11
roazhon5	2× Intel Xeon E5-2660 v3	Haswell EP	2	2× 10/20	Deb. 11
sagittaire	2× AMD Opteron 250	K8	2	2× 1/1	Deb. 11
servan	2× AMD EPYC 7352	Zen 2 (Rome)	2	2× 24/48	Deb. 11
troll	2× Intel Xeon Gold 5218	Cascade Lake SP	2	2× 16/32	Deb. 11
uvb	2× Intel Xeon X5670	Westmere EP	2	2× 6/12	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
CFL	Intel Core i7-8700K	Coffee Lake	1	6/12	Ub. 24.04
EMR	Intel Xeon Silver 4514Y	Emerald Rapids	1	16/32	Ub. 24.10
lab32	AMD EPYC 8024P	Zen 4c (Sienna)	1	8/16 (2× 4/8)	Ub. 22.04
Ripper	AMD Threadripper 5995WX	Zen 3 (Chagall)	1	64/128 (8× 8/16)	Ub. 24.04
Zen2	AMD Ryzen 7 3700X	Zen 2 (Matisse)	1	8/16	Ub. 20.04
Zen3	AMD Ryzen 5 5600X	Zen 3 (Vermeer)	1	6/12	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

Ripper and lab32 are multi-die modules, whose number of dies and of core/threads per die are indicated between brackets after the total number of core/thread in the system.

Table 9: Summary of results for all 36 machines, with fixed frequency and single-threshold classifier

Machine	Flush+Flush			Flush+Reload			Load/Flush+Reload		
	TU	TA	Best-TA	TU	TA	Best-TA	TU	TA	Best-TA
abacus25	15.74	0.39	< 0.01	22.54	3.96	< 0.01	< 0.01	< 0.01	< 0.01
chiclet	6.02	< 0.01	< 0.01	16.69	0.86	< 0.01	< 0.01	< 0.01	< 0.01
chifflot	25.00	1.00	< 0.01	25.01	6.10	< 0.01	< 0.01	< 0.01	< 0.01
chirop	25.01	1.11	< 0.01	27.71	2.72	< 0.01	< 0.01	< 0.01	< 0.01
dahu	25.00	1.06	< 0.01	25.00	7.39	< 0.01	< 0.01	< 0.01	< 0.01
econome	25.00	< 0.01	< 0.01	24.35	0.29	< 0.01	< 0.01	< 0.01	< 0.01
ecotype	25.01	2.16	< 0.01	25.02	11.16	< 0.01	< 0.01	< 0.01	< 0.01
esterel41	27.55	1.53	< 0.01	34.15	6.57	< 0.01	< 0.01	< 0.01	< 0.01
graffiti	25.00	1.26	< 0.01	25.01	8.56	< 0.01	< 0.01	< 0.01	< 0.01
grappe	25.00	1.15	< 0.01	25.00	2.07	< 0.01	< 0.01	< 0.01	< 0.01
grele	25.00	1.61	< 0.01	25.04	11.55	< 0.01	< 0.01	< 0.01	< 0.01
grue	6.17	< 0.01	< 0.01	16.62	0.93	< 0.01	< 0.01	< 0.01	< 0.01
kinovis	27.66	2.29	< 0.01	34.40	8.08	< 0.01	< 0.01	< 0.01	< 0.01
mercantour2	25.00	11.56	< 0.01	24.02	10.86	< 0.01	< 0.01	< 0.01	< 0.01
montcalm	25.00	1.19	< 0.01	25.52	0.99	< 0.01	< 0.01	< 0.01	< 0.01
musa	12.66	0.03	< 0.01	24.25	6.55	< 0.01	< 0.01	< 0.01	< 0.01
nova	25.00	1.89	< 0.01	25.01	10.05	< 0.01	< 0.01	< 0.01	< 0.01
paradoxe	25.00	1.14	< 0.01	25.62	0.64	< 0.01	< 0.01	< 0.01	< 0.01
parasilo	25.00	< 0.01	< 0.01	24.18	0.31	< 0.01	< 0.01	< 0.01	< 0.01
petitprince	25.00	< 0.01	< 0.01	20.29	1.44	< 0.01	< 0.01	< 0.01	< 0.01
roazhon5	25.00	< 0.01	< 0.01	25.00	0.78	< 0.01	< 0.01	< 0.01	< 0.01
sagittaire	25.00	24.98	< 0.01	38.90	37.37	3.91	< 0.01	< 0.01	< 0.01
servan	35.65	6.09	< 0.01	37.96	9.48	< 0.01	< 0.01	< 0.01	< 0.01
troll	25.00	1.27	< 0.01	25.00	2.32	< 0.01	< 0.01	< 0.01	< 0.01
uvb	3.38	0.02	< 0.01	24.31	1.04	< 0.01	< 0.01	< 0.01	< 0.01
yeti	19.19	1.66	< 0.01	33.78	1.46	< 0.01	< 0.01	< 0.01	< 0.01
ARL	< 0.01	< 0.01	< 0.01	0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
CFL	18.73	14.46	< 0.01	0.04	5.12	< 0.01	< 0.01	< 0.01	< 0.01
EMR	0.05	0.02	< 0.01	41.34	0.01	< 0.01	< 0.01	< 0.01	< 0.01
lab32	0.75	< 0.01	< 0.01	25.16	1.96	< 0.01	< 0.01	< 0.01	< 0.01
Ripper	0.34	0.25	< 0.01	10.20	0.06	< 0.01	< 0.01	< 0.01	< 0.01
Zen2	0.74	0.52	< 0.01	16.97	0.26	< 0.01	< 0.01	< 0.01	< 0.01
Zen3	4.38	2.84	< 0.01	0.03	0.01	< 0.01	< 0.01	< 0.01	< 0.01
Zen4	0.87	0.32	0.01	0.08	0.04	< 0.01	< 0.01	< 0.01	< 0.01
Zen5	8.67	5.54	0.01	3.19	0.38	< 0.01	< 0.01	< 0.01	< 0.01
ZenP	10.69	7.79	0.25	28.95	7.51	< 0.01	< 0.01	< 0.01	< 0.01