CS773 Project Checkpoint 2

ClepsydraCache – Preventing Cache Attacks with Time-Based Evictions

Ananya Rao, Deeksha Dhiwakar
creativitycachemiss
22b0980@iitb.ac.in, 22b0988@iitb.ac.in

Problem statement

Cache-based side-channel attacks, such as PRIME+PROBE and PRIME+PRUNE+PROBE, exploit cache contention to leak sensitive data. Existing defenses, like partitioning and index randomization, either reduce flexibility or remain vulnerable to advanced eviction set construction. The challenge is to prevent eviction set formation and eliminate timing-based leakage while maintaining high performance. We propose an implementation based on the paper **ClepsydraCache**, which combines time-based evictions and index randomization to achieve strong security with minimal overhead.

Project Goals

1) Implement a functional model of CLEPSYDRACACHE in ChampSim.

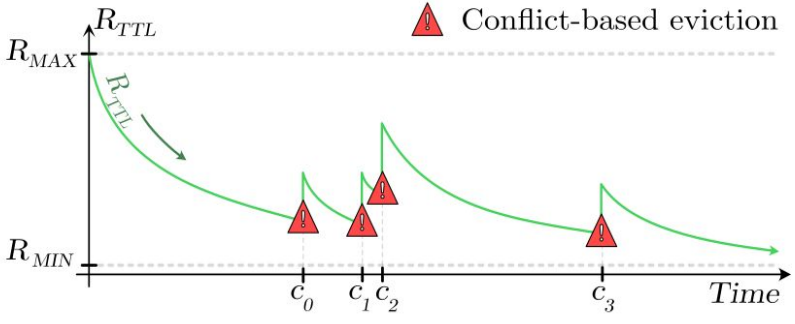
- Setup of ChampSim implementation environment
- Modification of the cache line eviction policy
- Implementation of hashing method (PRINCE)
- Implementation of randomization technique of indexing

2) Evaluation of CLEPSYDRACACHE against default ChampSim
- Performance metrics
- Security metrics

Tasks Completed

TTL based eviction policy

- Time-To-Live (TTL) uniformly randomly assigned to cache line on insertion/access.
- All TTLs decreased by $Rate_{TTL}$ at each cache access.
- $Rate_{TTL}$ adjusted dynamically every 100000 CPU cycles.
- $Rate_{TTL} = Rate_{TTL} * 2$ on conflict
- $Rate_{TTL} = Rate_{TTL} - 1$ otherwise
- Line evicted when TTL = 0



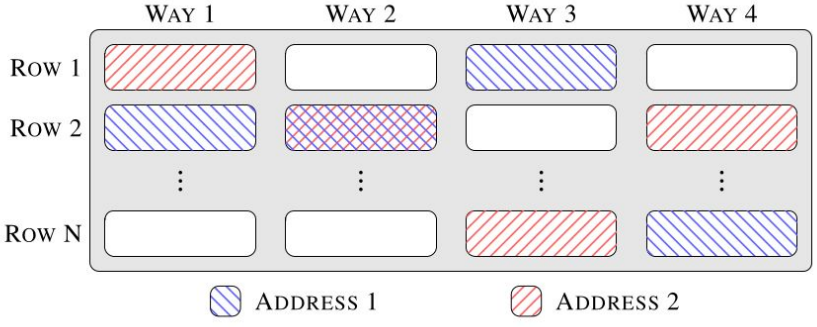
Variation of Rttl with time

Image credits: https://www.usenix.org/conference/usenixsecurity23/presentation/thoma

Tasks Completed

Index randomization

- Each way is assigned unique 128 bit key.
- PRINCE block cipher with per-way keys generates ciphers from addresses.
- Each address maps to a deterministic index per way — the “dynamic set.”
- On cache fill: if any dynamic set index is free, allocate it; else evict a random one.
- On lookup: check for tag match at all indices in the dynamic set.



Randomized address mapping

Image credits: https://www.usenix.org/conference/usenixsecurity23/presentation/thoma

How does this help?

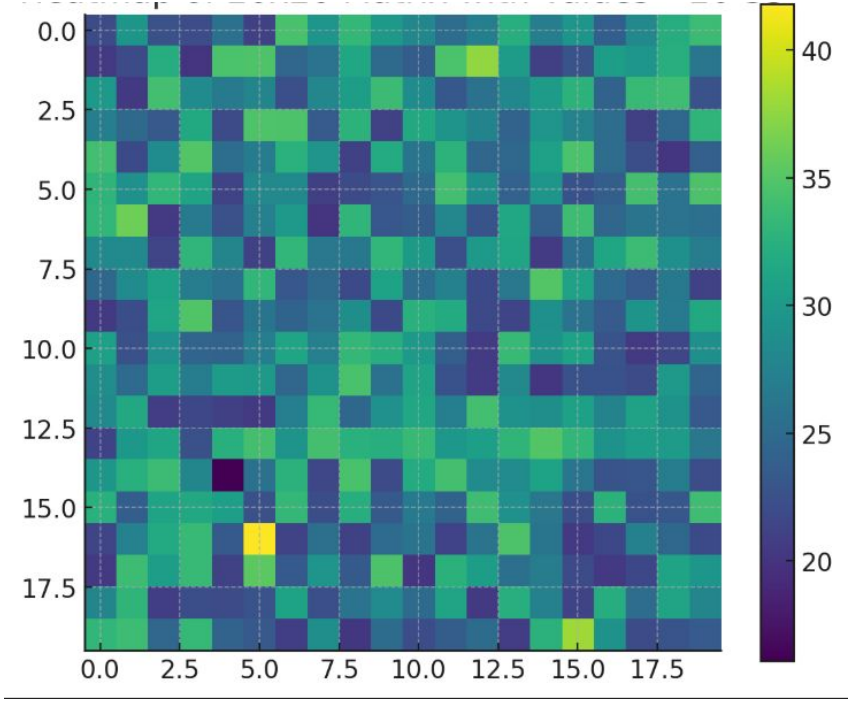
TTL based eviction - Attacker cannot exploit conflict based eviction to infer victim data!

Index randomization - Each address maps to different pseudorandom indices for each cache way; unpredictable cache layout!

Attacker cannot generate eviction sets!

Validation of our Implementation

In order to ensure that index randomization is indeed successful, we plot a matrix where the x axis represents the incoming index and the y axis represents the evicted index. We observe that there is no correlation between the evicted index and the target index. This guarantees that the attacker cannot create eviction sets reliably.



Heatmap for a subset of the matrix. Each cell represents no of times x axis index evicts y axis index

Experiments

Parameters to vary

- Initial value of $Rate_{TTL}$ ($Rate_{TTL0}$)
- TTL sampling range (L_{TTL} to U_{TTL})
- Bounds on $Rate_{TTL}$ (L_{Rate} to U_{Rate})
- $Rate_{TTL}$ update frequency

Metrics of Interest

- Clock cycles
- IPC
- LLC MPKI
- Avg Miss Latency

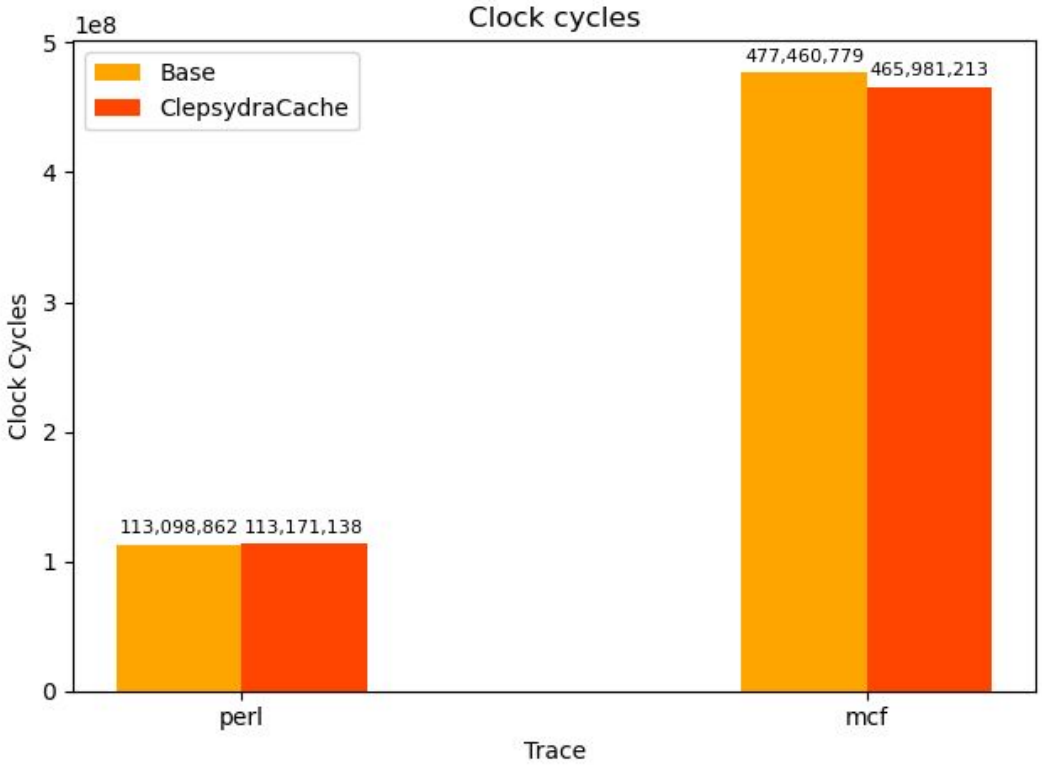
Results - 2 CPUs - perl + mcf traces

L_{TTL}	U_{TTL}	$Rate_{TTL0}$	L_{Rate}	U_{Rate}	IPC (avg)	LLC MPKI 0	LLC MPKI 1
NUM_WAY * NUM_SET / 2	3 L_{TTL}	NUM_SET/10	1	U_{TTL}	0.3737	0.911	31.48
NUM_WAY * NUM_SET / 2	3 L_{TTL}	NUM_SET/2	1	U_{TTL}	0.3831	0.98	19.12
NUM_WAY * NUM_SET / 2	5 L_{TTL}	NUM_SET	NUM_SET/10	U_{TTL} / 5	0.3272	1.90	27.04
BASELINE ORIGINAL VALUES					0.3894	1.47	4.90

In all experiments above, $Rate_{TTL}$ is updated every 100000 CPU cycles

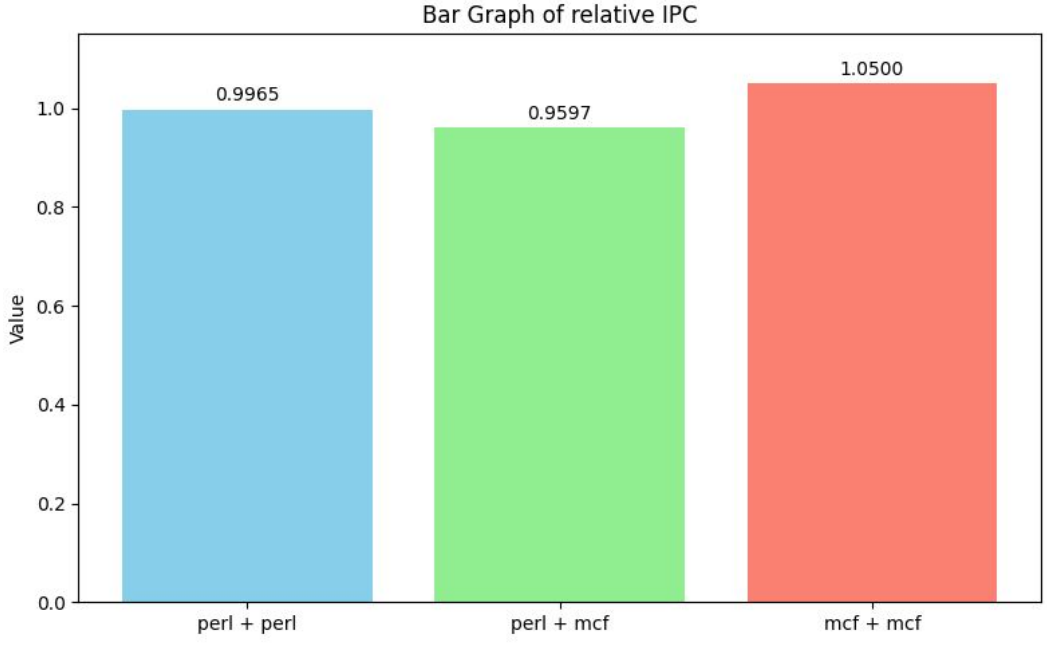
Clock cycles plot for Parameter Set 1

Observations: Performance comparable to baseline cycles. Reasoning: As we will see, ClepsydraCache lowers avg miss latency, which compensates for increased miss rate.

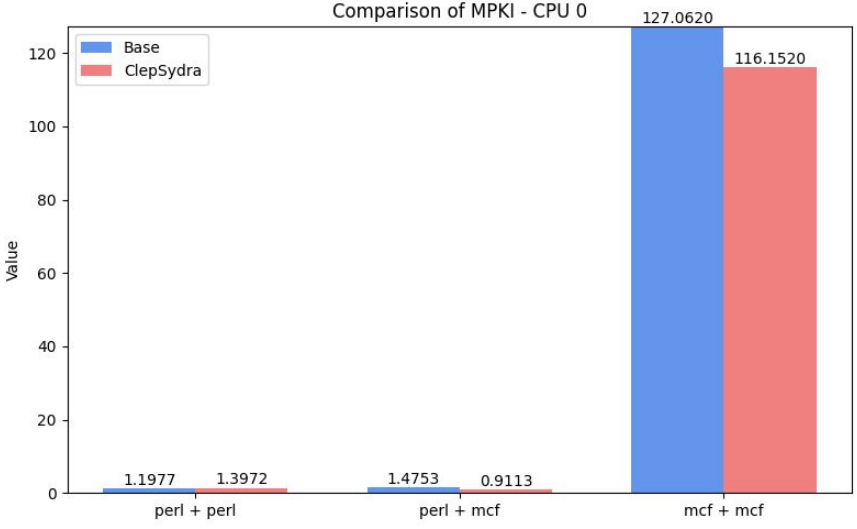


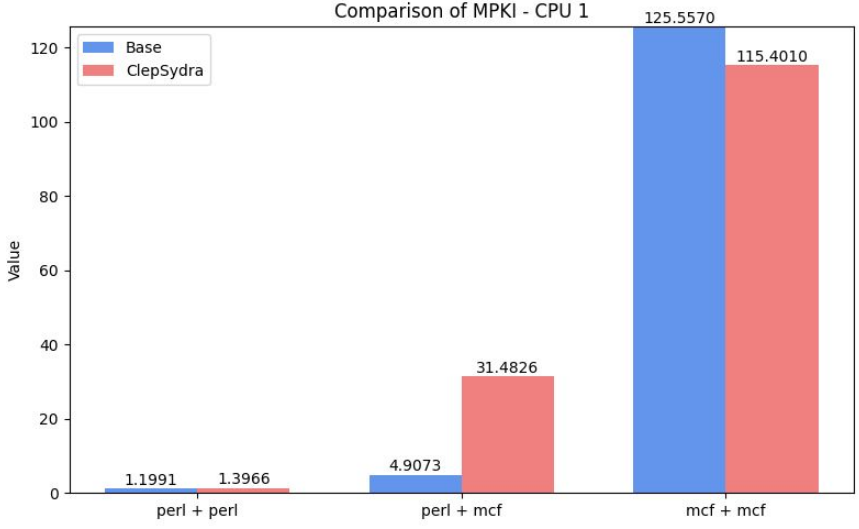
IPC plot for Parameter Set 1

Observations: IPC values comparable to baseline values. Small overhead due to hashing for index randomization.



LLC MPKI plots for Parameter Set 1

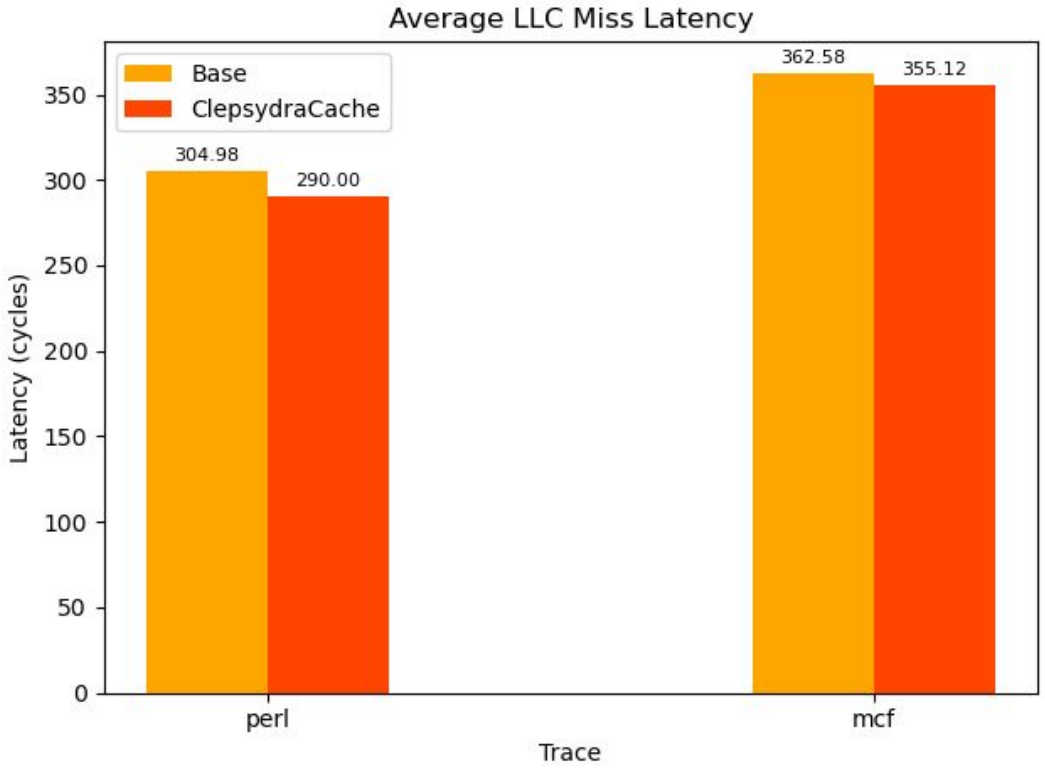




Observations: MPKI values comparable to baseline values. Slight variations depending on nature of addresses generated by trace.

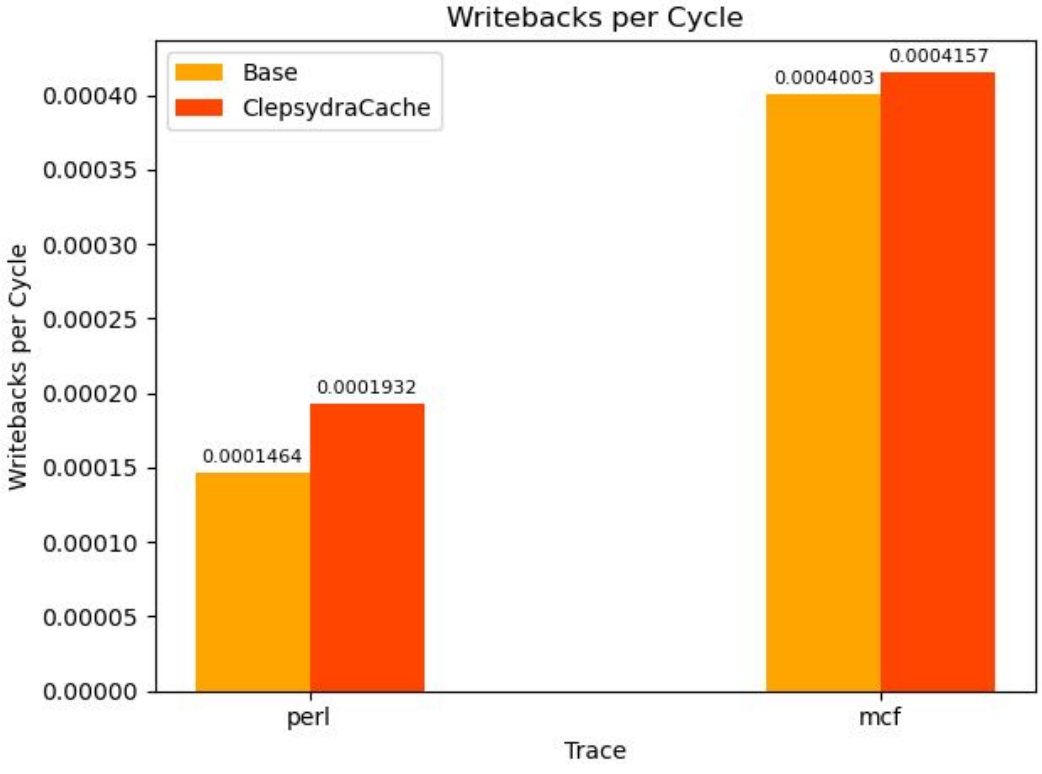
Avg miss latency plots for Parameter Set 1

Observations: Miss latency values slightly lower than baseline values.
Reasoning: Traditional cache performs writebacks on conflicts, but ClepsydraCache does it based on timing => independent of cache misses



Writebacks per cycle for Parameter Set 1

Observations: Writebacks per cycle increases slightly, since more entries are written back using ClepsydraCache (cache entries are invalidated more frequently due to TTL)



Security Metrics Evaluation

Methodology:

- Run ChampSim with 2 cores, with a PRIME+PRUNE+PROBE attacker running on one core and the victim on the other.
- Both attacker and victim implemented within the ChampSim simulator (not trace simulations)
- Attacker keeps generating address accesses attempting to evict victim cache blocks.
- Track no of conflict based evictions with and without ClepsydraCache
- Expect much lower conflict based evictions with ClepsydraCache

Conclusion

We have successfully implemented ClepsydraCache on ChampSim, which does not cause any significant performance overhead, and does not degrade cache efficiency (as evidenced by the comparable IPC and MPKI values).

Github link

- https://github.com/ananyarao23/ClepsydraCache-CS773.git