NOPROBE: A Fast Multi-strategy Probing Technique For x86 Dynamic Binary Instrumentation

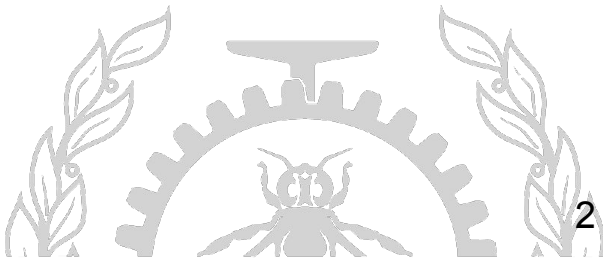
Anas Balboul

Mai 08, 2020

Polytechnique Montréal

Génie informatique et logiciel

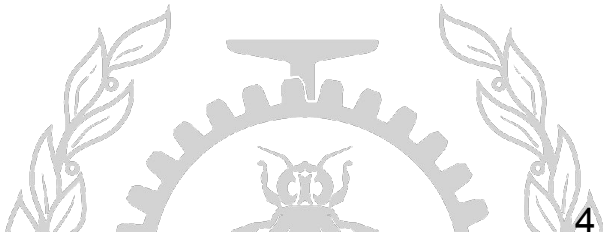
Summary



1. Introduction

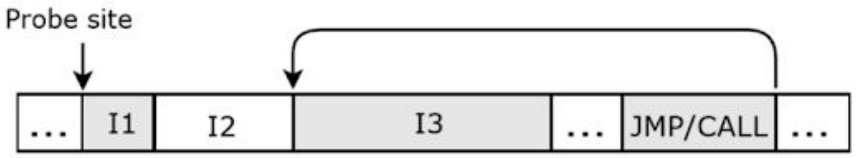
1. Introduction

- The what and why..
- Static:
  - Added before/during compilation
- Dynamic:
  - During the execution
  - Agent injected in process address space

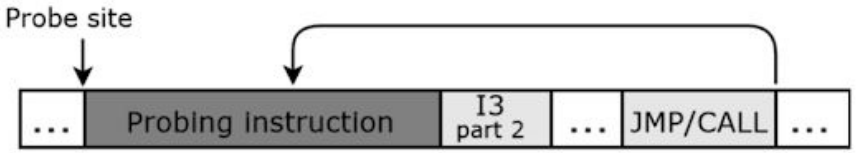


1. Introduction x86-ISA

- x86 variable instruction size
  - Reason: performance and compression.
- Dynamic Binary instrumentation (code injection with instr replacement)
  - branch (CALL / JMP) vs Trap (INT3)
- Instructions border changes:
  - Jump to an invalid border.
  - Return from preemption, interruption, or blocked state to an invalid border.



(a) Before overwritting.



(b) After overwritting.

2. Literature review

Dynamic binary instrumentation in user-space

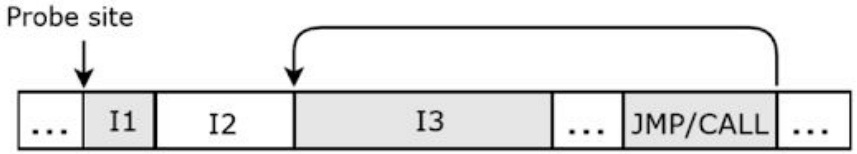
2. Revue de la littérature

- **Uprobe**: Trap instruction (flexible but too slow)
- **GDB**: Trap instruction (with ptrace => slower than ***uprobe***)
  - Fast tracepoint: A branch with a limit on the instrumented instr size
  - Seek quiescence by stopping the world
- **Dyninst:** Uses Control-Flow Graph (CFG) => flexible and very fast
  - Too much intrusion: high memory usage, stop the world (more than 5s in 64 cores machine), high computation during the insertion.
- **Valgrind and Pin:** Intermediate representation (IR) in a VM.
  - Very slow execution (up to x100 slower for Valgrind)
- **Liteinst and Dyntrace:** A branch probe that embed traps in it's offset
  - Fast in most cases, but could worst than trap-based probes in some cases (loop)
  - Fragment the memory. Badly supported in x86_32 (4Go addr space). ***Dyntrace***: Unsafe on-the-fly instrumentation.

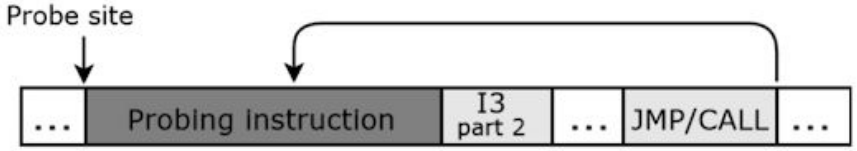
3. Problem definition

3. Problem definition

- Simultaneous execution problem:
  - Due to instruction borders changes



(a) Before overwritting.

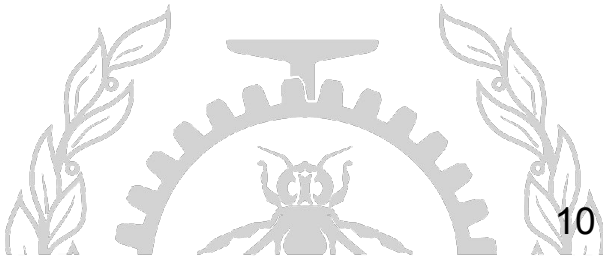


(b) After overwritting.

- Performance and memory usage

3. Problem definition

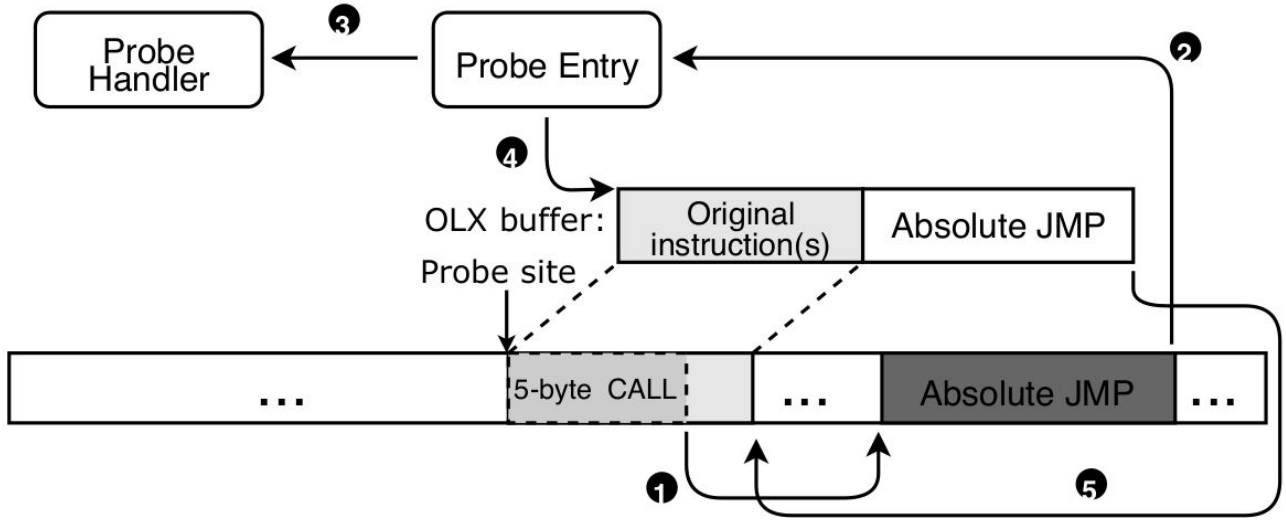
- Out-of-line execution: Support as many instruction relocation as possible (Target out of reach, invalid relative address).
- Intel Cross/Self Modification:
  - During execution, patching code that overlaps cache lines may not be atomic and can cause a GPF.
  - Intel errata: A core should execute a serializing instruction (CPUID, IRET, etc..) prior to new code execution.



4. Proposed Solution

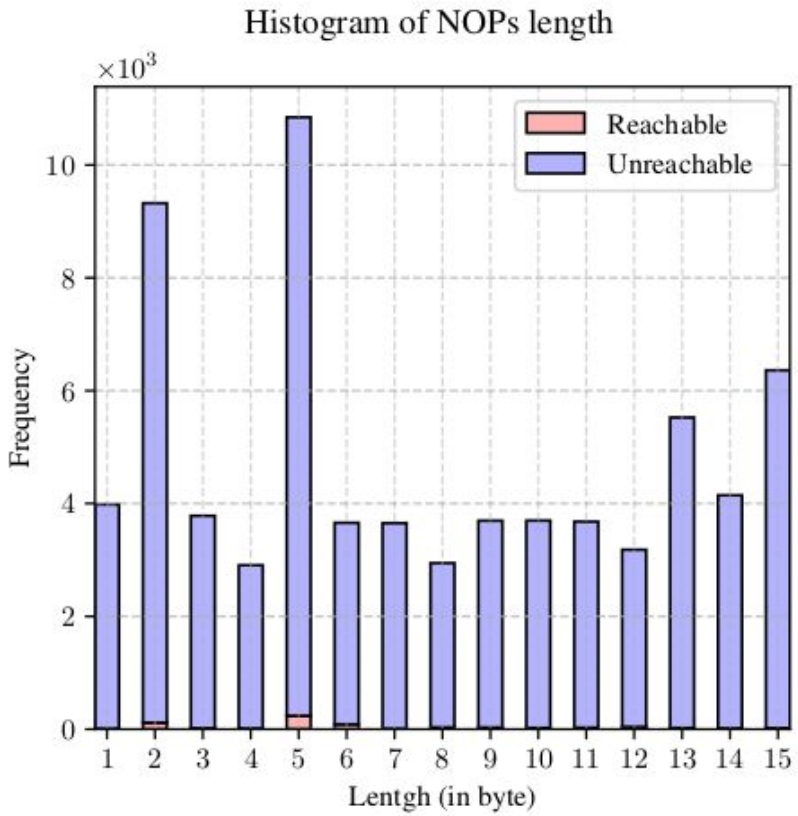
4. Proposed Solution

- NOProbe: A dynamic instrumentation solution
  - First strategy:

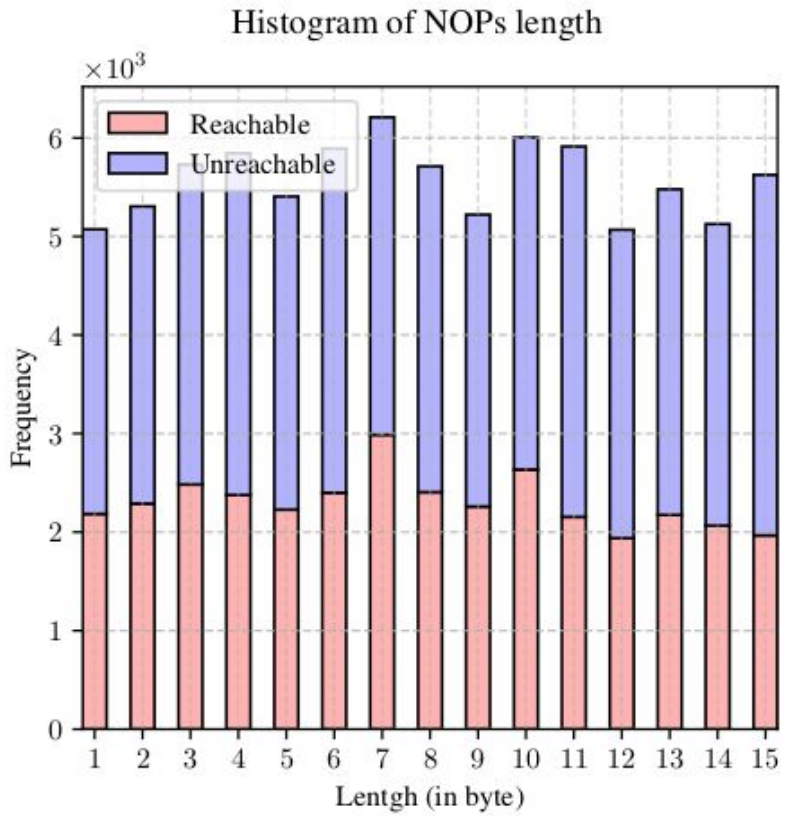


4. Proposed Solution

- **NOP-padding** added by default during compiler optimization starting from level 2 for GCC and level 3 for Clang.
- They are everywhere !
  - percentage of reachable and unreachable NOPs in two samples:



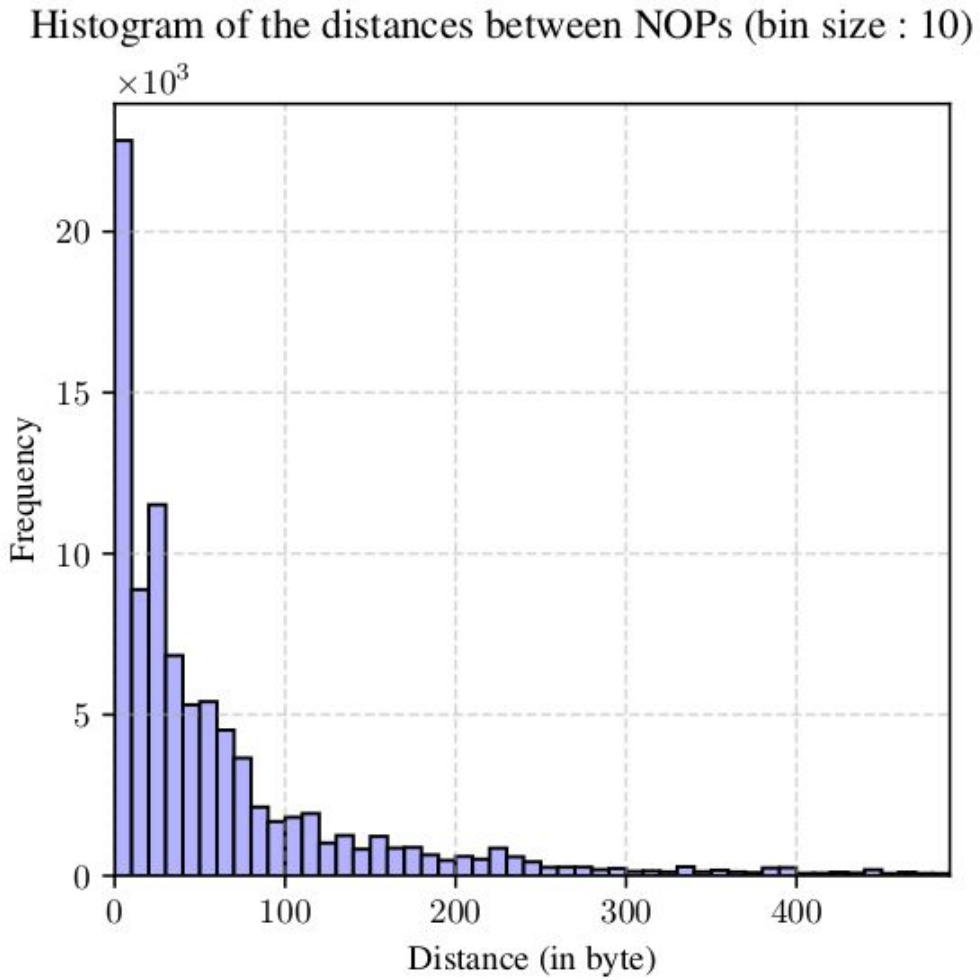
(a) *llvm-ar* compiled with *gcc -O2*.



(b) *clang* compiled with *clang -O3*.

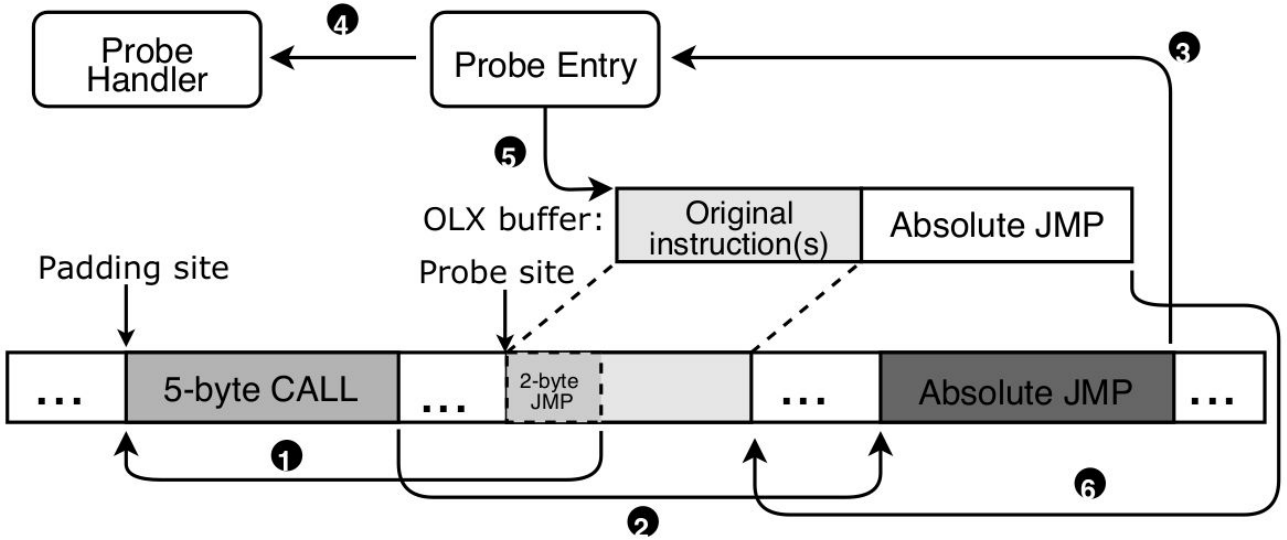
4. Proposed Solution

- NOP distribution analysis
  - Histogram of distances between NOPs:



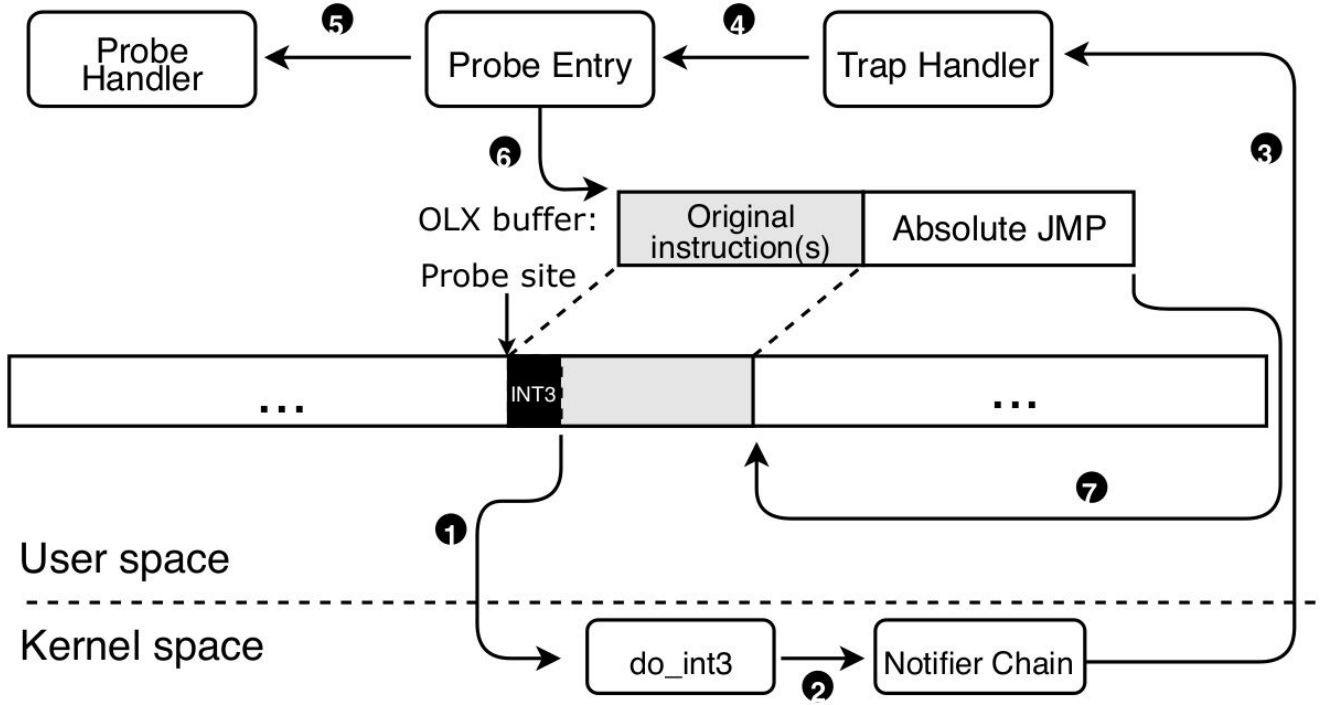
4. Proposed Solution

- NOProbe: A dynamic instrumentation solution
  - Second Strategy:



4. Proposed Solution

- NOProbe: A dynamic instrumentation solution
  - Third and last resort strategy:



4. Proposed Solution

- NOProbe: **Static Analysis**:
  - Which strategy to deploy:
    - Disassemble the probed function and check the target of each branch..
  - To find out if a NOP is reachable or not:
    - The function size in the ELF symbol table is used to find NOPs that align functions.
    - NOPs aligning jumps, loops and labels have a pattern:

```
0000000000027a00 <free_parsed_cmdline>:
                ...
27a09:  48 8b 7f f8        mov    -0x8(%rdi),%rdi
27a20:  f3 c3              repz retq
27a22:  0f 1f 40 00        nopl   0x0(%rax)
27a26:  66 2e 0f 1f 84     nopw   %cs:0x0(%rax,%rax,1)
        00 00 00 00 00
0000000000027a30 <absolute_dirname>:
                ...
```

```
0000000000015610 <do_replay>:
                        ...
15763:  0f 88 d7 00 00 00     js   15840 <do_replay
                       ...
15833:  89 30 ff ff ff        jns  15769 <do_replay
15839:  1f 80 00 00 00 00     nopl 0x0(%rax)
15840:  83 c4 01              add  $0x1,%r12d
```

4. Proposed Solution

Assembly (AT&T Syntax)	Byte Sequence (Hexadecimal)
nop	90
xchg %ax,%ax	66 90
nopl (%rax)	0f 1f 00
nopl 0x0(%rax)	0f 1f 40 00
nopl 0x0(%rax,%rax,1)	0f 1f 44 00 00
nopw 0x0(%rax,%rax,1)	66 0f 1f 44 00 00
nopl 0x0(%rax)	0f 1f 80 00 00 00 00
nopl 0x0(%rax,%rax,1)	0f 1f 84 00 00 00 00 00
nopw 0x0(%rax,%rax,1)	66 0f 1f 84 00 00 00 00 00
nopw %cs :0x0(%rax,%rax,1)	66 2e 0f 1f 84 00 00 00 00 00

- NOProbe: **Reachable NOP**:
  - The program could execute them
  - Carefully patch them..
  - We need 5 bytes for the trampoline

```
0:   0f 1f 80 00 00 00 00     nop     DWORD PTR [rax+0x0]
7:   48 89 c8                 mov     rax,rcx
```

```
 0:   eb 04                    jmp     0x6
 2:   e8 01 02 03 04           call    0x4030208
 7:   48 89 c8                 mov     rax,rcx
```

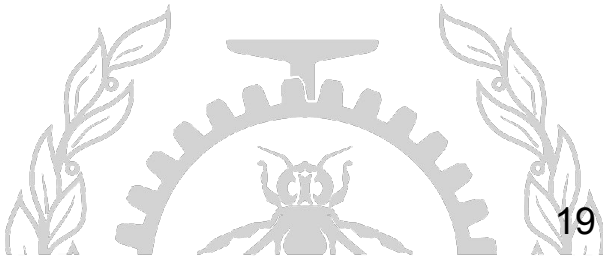
```
0:   0f 1f 84 00      nop   DWORD PTR [rax+rax*1+0x0]
     00 00 00 00
```

```
0:   0f 1f 84 e8      nop   DWORD PTR [rax+rbp*8+0x4030201]
     01 02 03 04
```

4. Proposed Solution

- NOProbe: **Assigning NOPs to probes**:
  - Optimal solution (Hungarian):
    - Complexity $O(n^3)$
  - Greedy algorithm:
    - A two bytes JMP has one byte offset and can go 128/127 bytes back and forth.
    - Prioritize the closed NOP to a probe.

	N1	N2
F1	(1)	0
F2	1	(1)



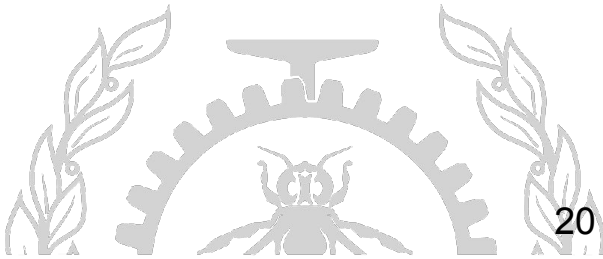
4. Proposed Solution

- NOProbe: **Out-of-Line eXecution (OLX):**
  - 5-byte JMP, JCC (conditional jump) and CALL have an offset of 4 bytes
  - They can only reach 2Go in both directions (problematic in x86_64)
  - They are relocated this way:

```
0:  e9 12 34 56 78        jmp     0x78563417
```

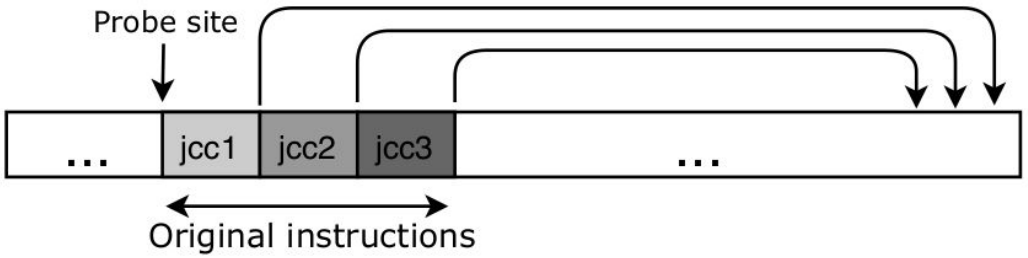
(a) Before relocation.

```
0:  ff 25 00 00 00 00     jmp     QWORD PTR [rip+0x0]
6:  00 00 00 00 17 34     data    target address
    56 78
```

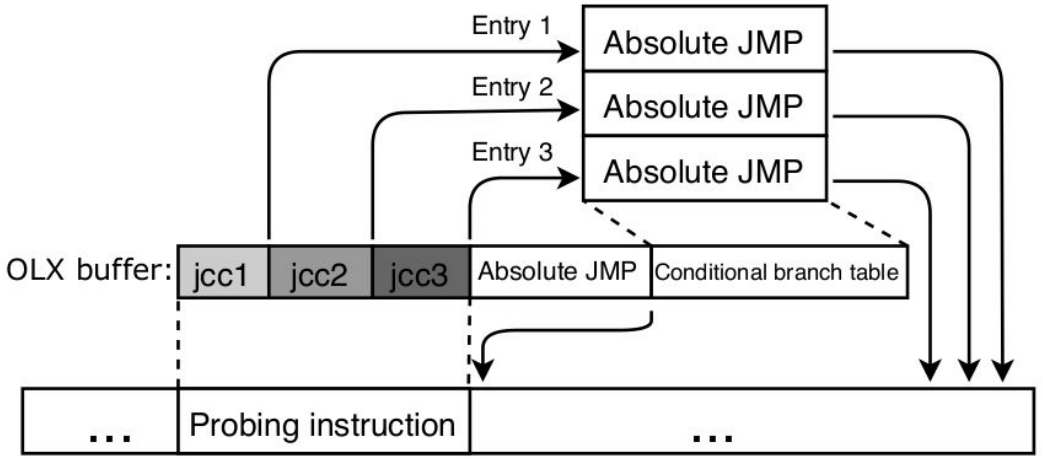


4. Proposed Solution

- NOProbe: **Out-of-Line eXecution (OLX):**
  - 2-byte JMP, JCC (conditional jump) have an offset of 1 byte
  - They can only reach 127/128 bytes in both directions
  - They are relocated this way:



(a) Before relocation.



(b) After relocation.

4. Proposed Solution

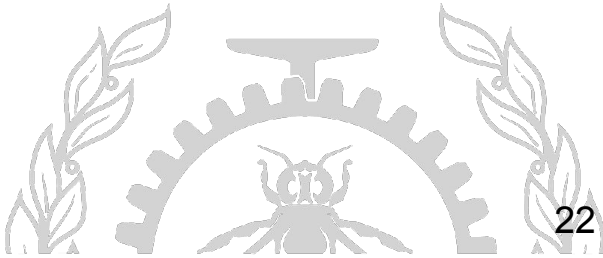
- NOProbe: **Out-of-Line eXecution (OLX):**
  - RIP relative addressing (added in x86_64)
  - Relative address is invalid after relocation:
  - They are relocated this way: (inspired from uprobe and adapted to user space)

```
site+0:   cmp     rdx, QWORD PTR [rip+0x5]
```

(a) Before relocation.

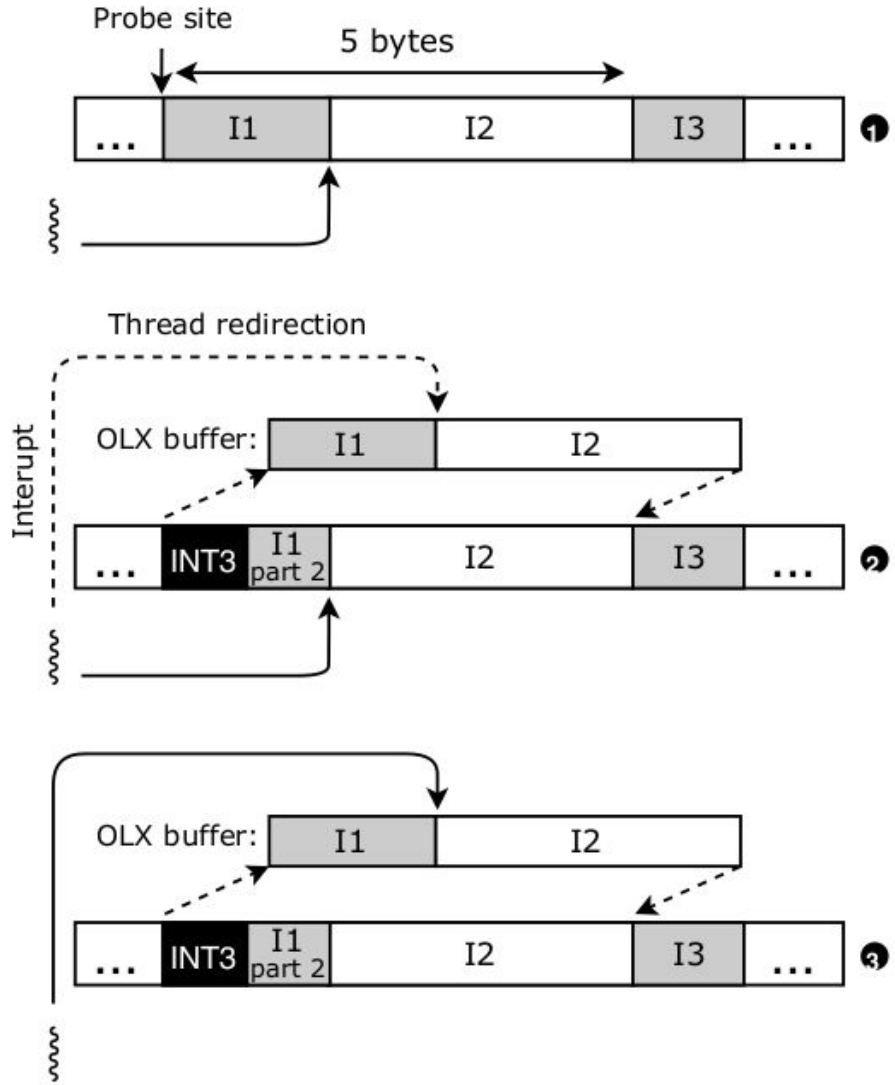
```
olx+0:   push    rax
olx+1:   mov     rax, 0x000000000000000F
olx+b:   cmp     rdx, QWORD PTR [rax]
olx+c:   pop     rax
```

(b) After relocation.



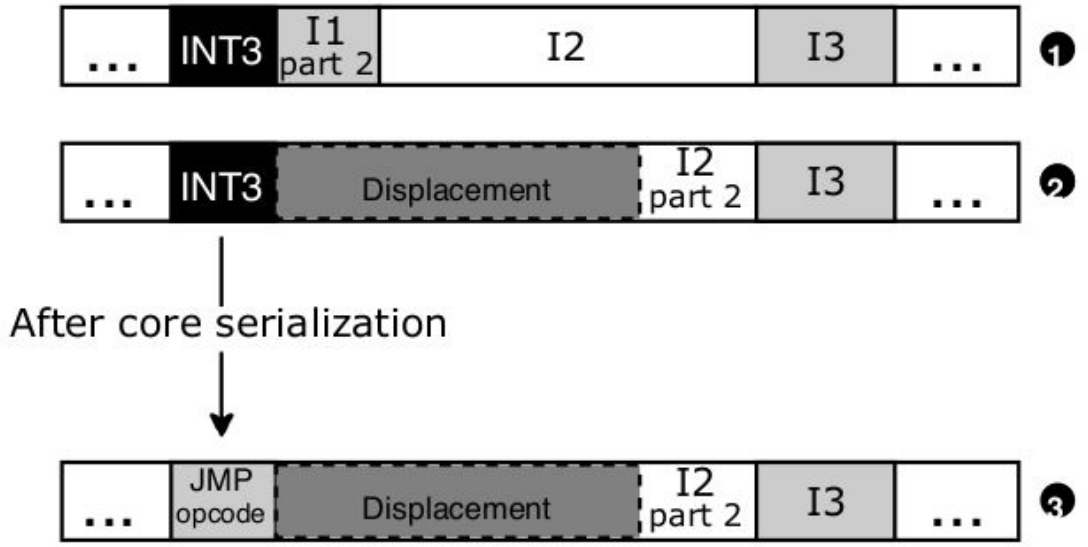
4. Proposed Solution

- NOProbe: ***Lock and Load***: On-the-fly patching without stopping the program
  - Two steps:
    - First step: **Lock and Redirect**:



4. Proposed Solution

- NOProbe: ***Lock and Load***: On-the-fly patching without stopping the program
  - Two steps:
    - Second step: **Load and Arm**:
      - When available: Core serialization with membarrier (the syscall)
      - Else: real-time signal to execute CPUID.



4. Proposed Solution

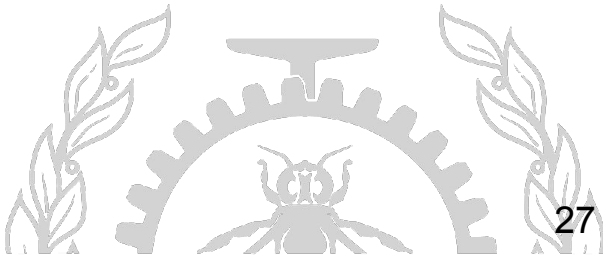
- NOProbe: **Signal Dispatching**:
  - We use two signals
    - SIGTRAP:
      - When locking the patching area
    - Real-time signal
      - When redirecting the threads outside the patching area
  - We avoid exclusive usage by sharing them
  - How we do it ? we Intercept sigaction(), signal(), et sigprocmask() :
    - Save the signal handler and signal masks in user space instead of registering them.
    - We register our own trap handler that dispatch the signal to the saved handler registered by the program or to the instrumentation signal handler (based on the TID for real-time signal, and based on the address of the raised trap for SIGTRAP).

5. Results

5. Results

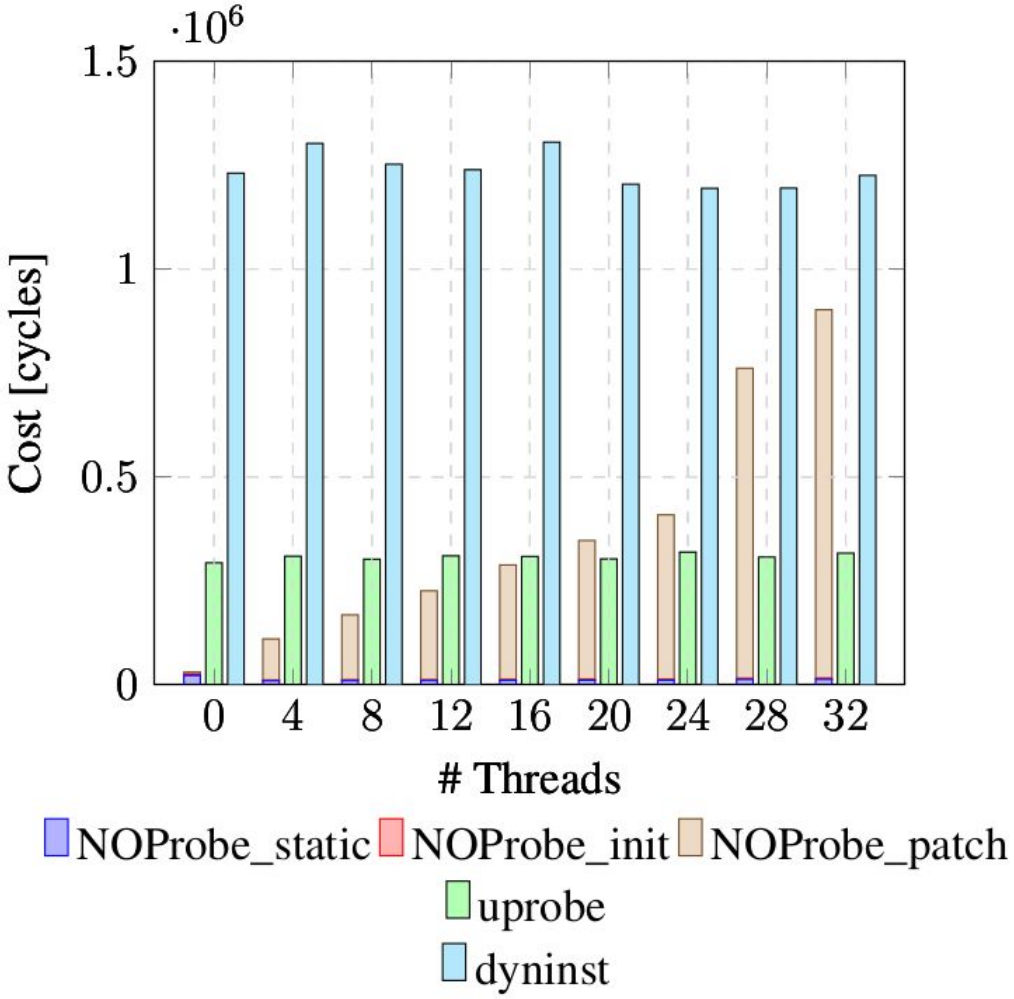
❑ Specification of the test machine:

Processor	4x Intel Xeon E7-8867 v3 @2.50GHz
# Cores	64 with Hyper-Threading disabled
Operating System	Debian GNU/Linux 9.0 and Linux 4.19.0-6-amd64
Compiler & Libraries	GCC/G++ 8.3.0 and GLIBC 2.28-10
Virtualization	Kernel-based Virtual Machine
Memory	256GB



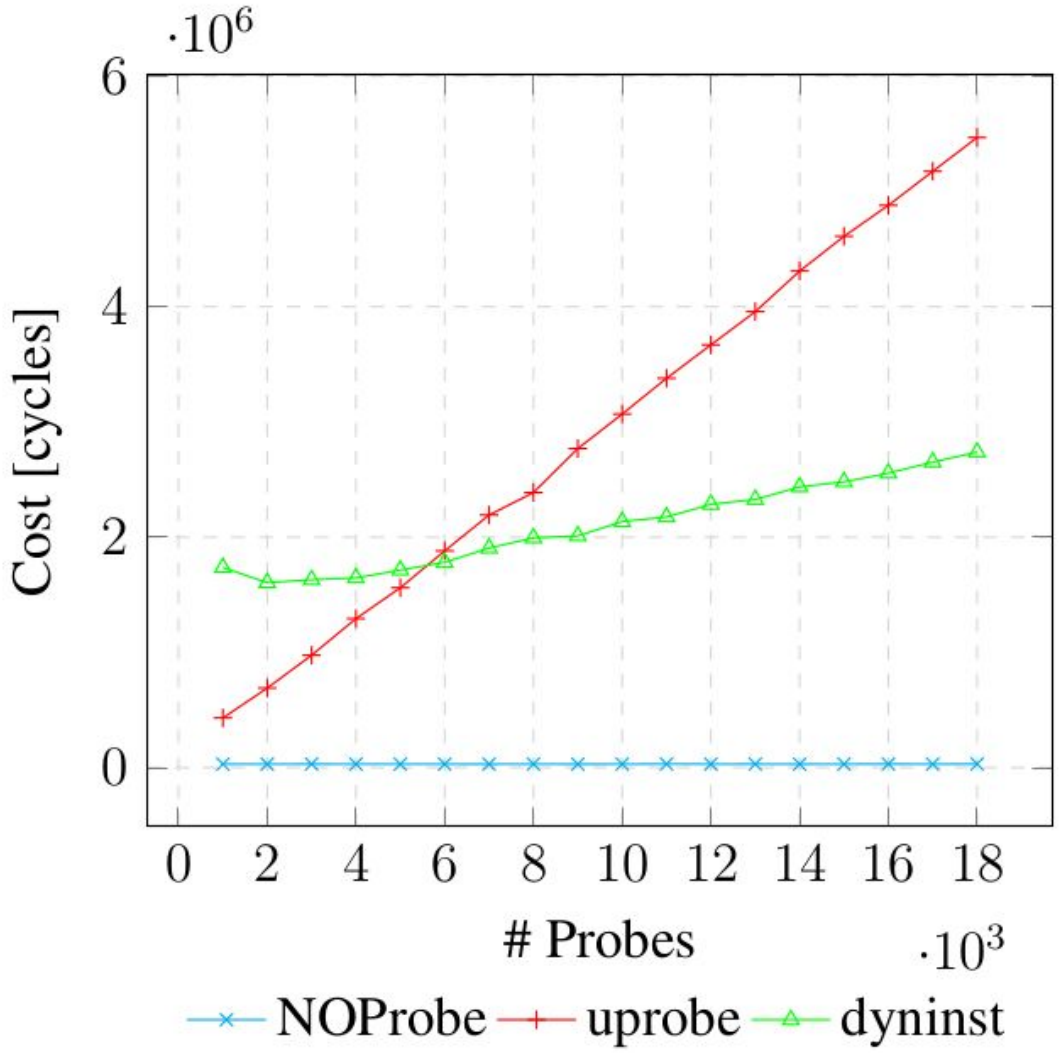
5. Results

- Instrumentation **Installation cost**:
  - VS the number of executing thread:



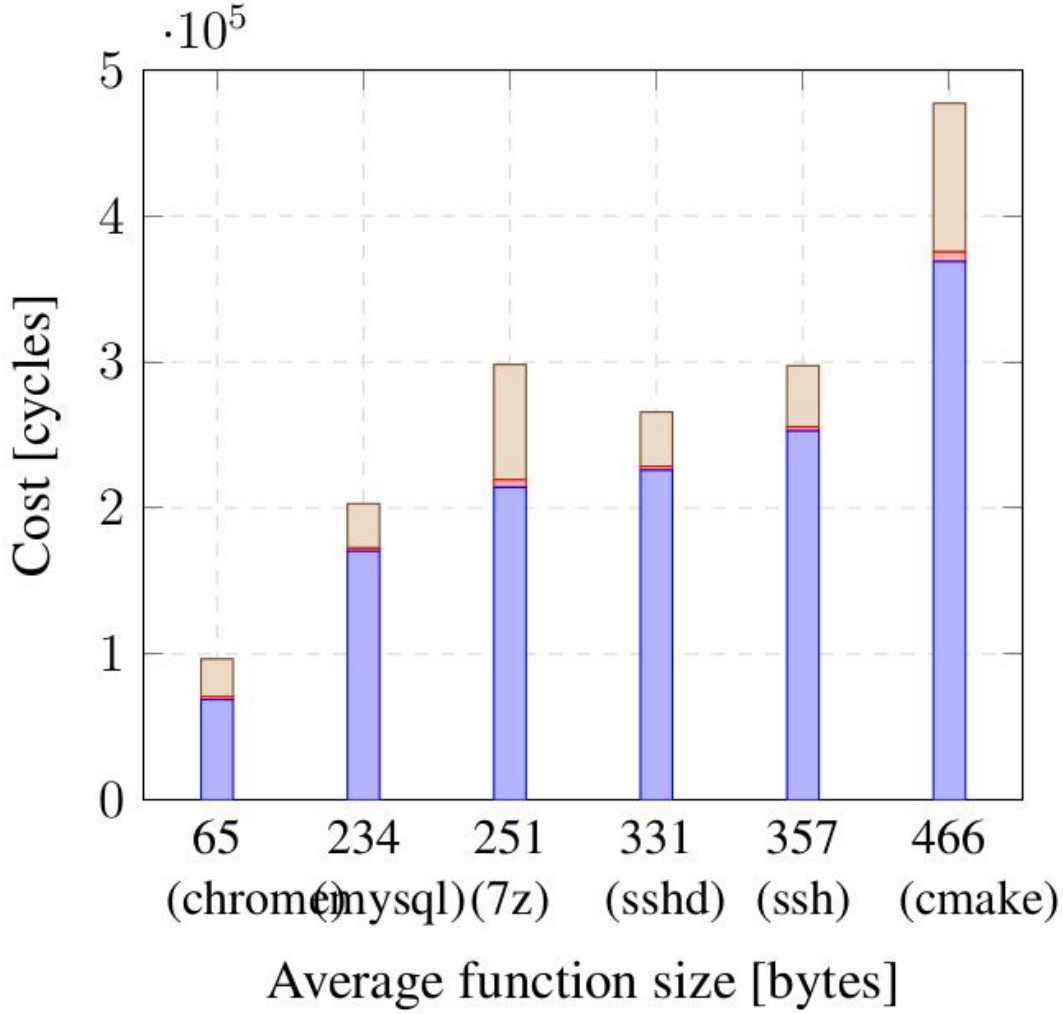
5. Results

- Instrumentation **Installation cost**:
  - VS the number of installed probes:



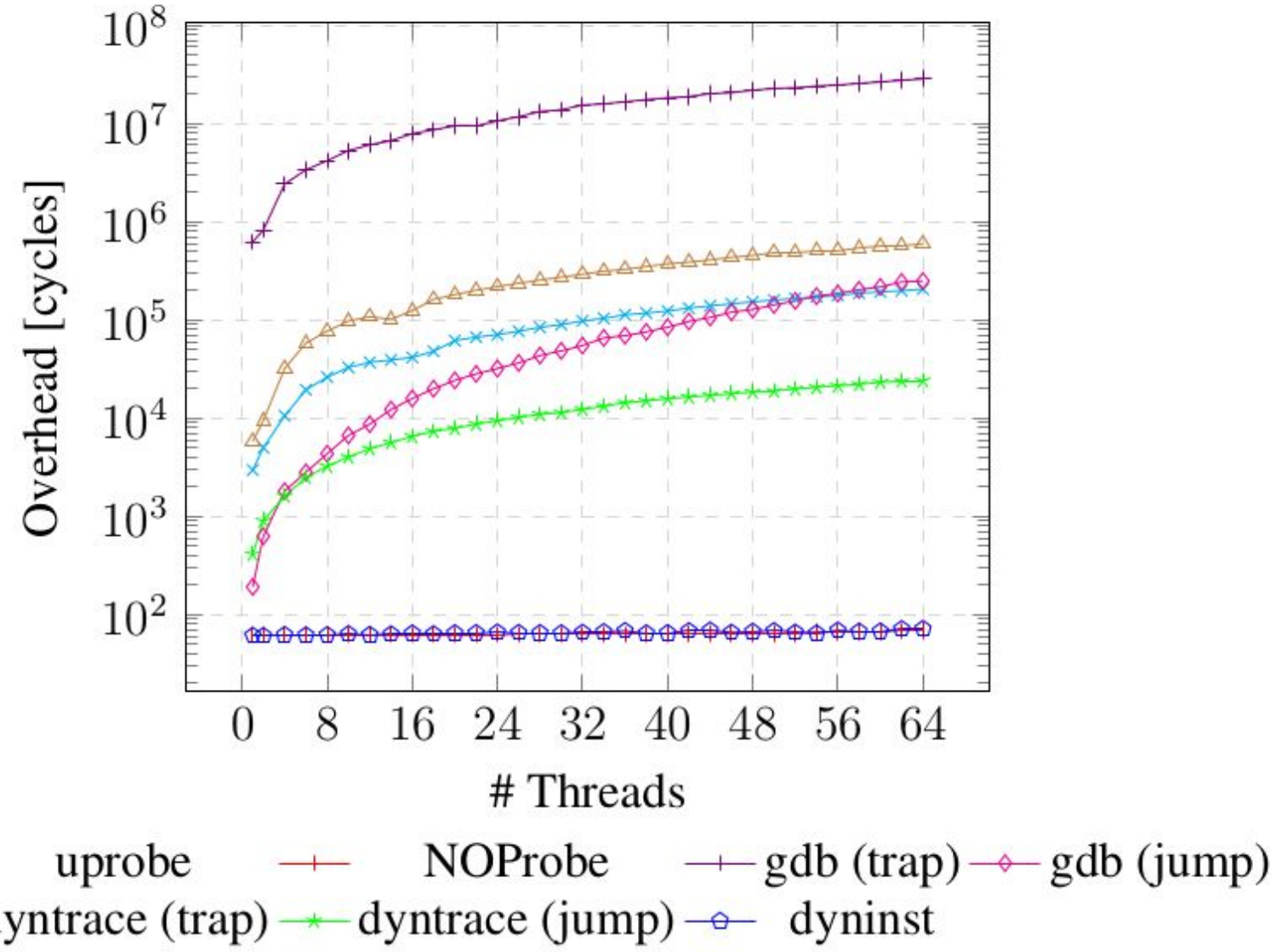
5. Results

- Instrumentation **Installation cost**:
  - VS the average function size:



5. Results

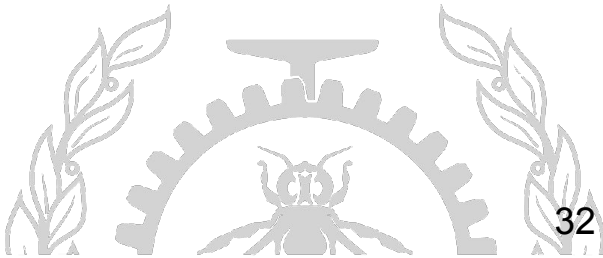
- **Execution cost:**
  - For probe insertion in a red and black tree, NOProbe avoid using costly locks and CAS (Compare And Swap) operations.



5. Results

- **Probe effectiveness**
  - Defined as: successful fast (branch) probes insertion ratio:

Bench	GDB			NOProbe			Dyntrace			Dyninst		
	Success	Total	Rate	Success	Total	Rate	Success	Total	Rate	Success	Total	Rate
chrome	28982	326205	8.88%	310497	326205	95.18%	-	-	-	320216	326205	98.16%
llvm-ar	523	19940	2.2%	19104	19940	95.80%	9318	19940	46.73%	75539	75539	100%
git	1	9901	0.01%	9815	9901	99.13%	4291	9900	43.34%	9908	9908	100%
vim	0	6396	0%	6279	6396	98.17%	5506	6396	86.08	6403	6403	100%
nginx	137	1148	11.93%	1109	1148	96.60%	1054	1148	91.81%	1154	1154	100%



6. Conclusion

6. Conclusion

- NOProbe: Fast, scalable, and less intrusive.
- Multiple strategy
  - 5-byte CALL
  - 2-byte JMP combined with a NOP-padding.
    - Two algorithms to assign NOPs to probes.
  - Last resort TRAP-based probe.
- Lock and Load: A protocol that can patch up to a basic block length safely without stopping the program.
- Future direction:
  - More strategies could be added.
  - For safe probe removal, URCU (User-space Read Copy Update) could be used.