Libpatch

Dynamic patching of binaries in userspace

Olivier Dion

Polytechnique Montréal
Dorsal laboratory

Summary

1. About Libpatch
2. Libpatch's concepts
3. Problem domain
4. Results
5. Conclusion

Summary

1. About Libpatch
2. Libpatch's concepts
3. Problem domain
4. Results
5. Conclusion

What is Libpatch?

C library

- Specializes in insertion of probes at runtime.
- Minimizes probes insertion and runtime overheads.
- Maximizes coverage (probe placement).

What is Libpatch?

C library

- Specializes in insertion of probes at runtime.
- Minimizes probes insertion and runtime overheads.
- Maximizes coverage (probe placement).

Supported architectures

- x86-64
- x86 (planned)
- arm (planned)
- aarch64 (planned)

What is Libpatch?

C library

- Specializes in insertion of probes at runtime.
- Minimizes probes insertion and runtime overheads.
- Maximizes coverage (probe placement).

Supported architectures

- x86-64
- x86 (planned)
- arm (planned)
- aarch64 (planned)

Dependencies

- capstone
- elfutils (libdw)
- libunwind or libsframe
- liburcu (x86-64 only)

What is Libpatch? (continuation)

Kernel features

- membarrier(2) expedited sync core (4.14)
- PTRACE_SEIZE and PR_SET_PTRACER (3.4)
- procfs(5)

What is Libpatch? (continuation)

Kernel features

- membarrier(2) expedited sync core (4.14)
- PTRACE_SEIZE and PR_SET_PTRACER (3.4)
- procfs(5)

Toolchain

- Only glibc (for now)
- *gcc* ≥ 8
- *clang* ≥ 8

What is Libpatch? (continuation)

Kernel features

- membarrier(2) expedited sync core (4.14)
- PTRACE_SEIZE and PR_SET_PTRACER (3.4)
- procfs(5)

Toolchain

- Only glibc (for now)
- *gcc* ≥ 8
- *clang* ≥ 8

Signal ownership (x86 only)

- SIGTRAP (optional)
- SIGILL (don't mask it!)

Libpatch's API

```
/* Library management. */
patch_err patch_init(const patch_opt *options, size_t options_count);
patch_err patch_fini(void);
patch_err patch_configure(const patch_opt *option);

/* Patches manipulation. */
patch_err patch_queue(uint64_t flags, patch_op *op);
patch_err patch_cancel(uint64_t cookie);
patch_err patch_commit(patch_result **results, size_t *results_count);
patch_err patch_uninstall_all(void);

/* Memory cleanup. */
patch_err patch_gc(uint64_t what, uint64_t policies);
patch_err patch_drop_results(patch_result *results, size_t results_count);

/* Thread specific actions. */
patch_err patch_make_coroutine(patch_coroutine_t *handle);
patch_err patch_drop_coroutine(patch_coroutine_t handle);
patch_err patch_switch_coroutine(patch_coroutine_t handle);
patch_err patch_unwind(size_t at);
```

Summary

1. About Libpatch
2. Libpatch's concepts
3. Problem domain
4. Results
5. Conclusion

Some definitions

Probe
: A user defined procedure to be called for instrumentation purpose.

Some definitions

Probe
: A user defined procedure to be called for instrumentation purpose.

Probe site
: Virtual address desired to be instrumented by the user. Also called the patch origin.

Some definitions

Probe
: A user defined procedure to be called for instrumentation purpose.

Probe site
: Virtual address desired to be instrumented by the user. Also called the patch origin.

Trampoline
: Intermediate instructions for jumping further in the program.

Some definitions

Probe
: A user defined procedure to be called for instrumentation purpose.

Probe site
: Virtual address desired to be instrumented by the user. Also called the patch origin.

Trampoline
: Intermediate instructions for jumping further in the program.

Handler
: Procedure that saves the context of the current computation, calls a probe, restores the context and returns to the computation continuation.

Some definitions (continuation)

Out of Line Execution (OLX) buffer

Relocated instructions from the probe site. Also called the computation continuation.

Some definitions (continuation)

Out of Line Execution (OLX) buffer
: Relocated instructions from the probe site. Also called the computation continuation.

Patch
: A patch is a divertion of the current computation, for calling a user defined callback and thereafter executing the patched instructions, before continuing the computation.

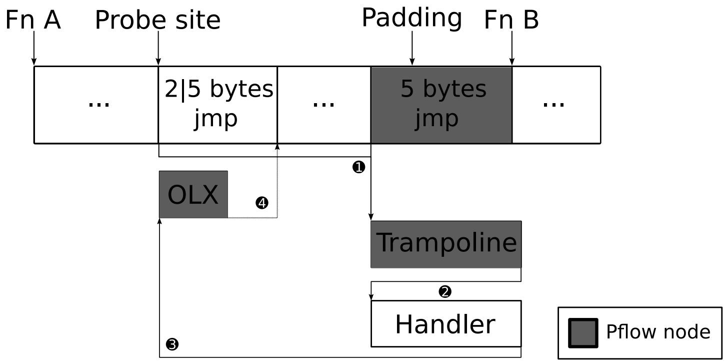
Some definitions (continuation)

Out of Line Execution (OLX) buffer
: Relocated instructions from the probe site. Also called the computation continuation.

Patch
: A patch is a divertion of the current computation, for calling a user defined callback and thereafter executing the patched instructions, before continuing the computation.

Pflow
: A thread is in a patch flow (pflow) from the moment it executes the first jump at the probe site, until it exits the pflow when it executes a branching inside the OLX buffer to the outside.

Patch control flow (pflow)



Summary

1. About Libpatch
2. Libpatch's concepts
3. Problem domain
4. Results
5. Conclusion

Instructions relocation

Problem

Instructions can be relative to the program counter.

- Relative branching.
- Anything with `rip`.

Instructions relocation

Problem

Instructions can be relative to the program counter.

- Relative branching.
- Anything with `rip`.

Solution A

Recompute the displacements.

- Kprobe does that.
- No overhead.
- Memory allocation restriction.

Instructions relocation

Problem

Instructions can be relative to the program counter.

- Relative branching.
- Anything with `rip`.

Solution A

Recompute the displacements.

- Kprobe does that.
- No overhead.
- Memory allocation restriction.

Solution B

Rewrite the instructions.

- Libpatch does that.
- Some overhead.
- No memory allocation restriction.

Instructions relocation examples

Generic case

```
OP DISP(%rip), R1

;; becomes
push -0x8(%rsp)
mov R2, -0x80(%rsp)
movabs RIP, R2
OP DISP(R2), R1
mov -0x80(%rsp), R2
pop -0x8(%rsp)

;; can become with register analysis
movabs RIP, R2
OP DISP(R2), R1
```

Instructions relocation examples (continuation)

Read RSP

```
OP %rsp, DISP(%rip)

;; becomes
lea -0x80(%rsp), %rsp
push R1
push R2
movabs RIP, R1
lea 0x90(%rsp), R2
OP R2, DISP(R1)
pop R2
pop R1
lea 0x80(%rsp)

;; can become with register analysis
movabs RIP, R1
OP %rsp, DISP(R1)
```

Instructions relocation examples (continuation)

Indirect jump

```
jmp *DISP(%rip)

;; becomes
lea -0x80(%rsp), %rsp
push %rax
movabs RIP, %rax
mov DISP(%rax), %rax
xchg %rax, (%rsp)
ret $0x80

;; can become with register analysis
movabs RIP, %rax
jmp *DISP(%rax)
```

Instructions relocation (continuation)

- Mostly solved problem.
- Previous algorithms were not respecting the ABI (red zone).
- Register analysis could help reduce overhead.

Trampoline anatomy

```c
struct trampoline_descriptor {
   u8           has_post_probe:1
   u8           k;
   patch_probe  probe;
   void        *user_data;
   uintptr_t    olx_addr;
   uintptr_t    real_pc;
} __packed;

/* Not a real C struct! */
struct trampoline {
   u8 instructions[];
   u8 padding[];
   struct trampoline_descriptor desc;
};
```

Trampoline anatomy

```
;; Skip red-zone (leaf functions only)
lea -128(%rsp), %rsp

;; Load trampoline descriptor at %rip + DISP1
push %rdi
lea DISP1(%rip), %rdi

;; Call handler stored at %rip + DISP2 in pool
call *DISP2(%rip)

;; Some padding
;; ...

;; Trampoline descriptor (referenced by DISP1 and DISP2)
;; ...
```

Trampoline location

- A trampoline is allocated from a pool.
- Every trampoline is within its own cache line.
- At the end of each pool, handlers addresses are stored.
- Pools are by default around a thousand of pages in size
- Around 64 000 trampolines per pool
- Pools are placed at various strategic places in the program.
  - Before the program.
  - In the heap.
  - Near libraries.
  - Top of stack.

Trampoline location (continuation)

Example of a program with -pie

```
555555164000-555555554000 ... /memfd:libpatch:null-trampoline (deleted)
555555554000-555555555000 ... /home/old/projects/libpatch/a.out
555555555000-555555556000 ... /home/old/projects/libpatch/a.out
555555556000-555555557000 ... /home/old/projects/libpatch/a.out
555555557000-555555558000 ... /home/old/projects/libpatch/a.out
555555558000-555555559000 ... /home/old/projects/libpatch/a.out
555555559000-5555555e3000 ... [heap]
...
7ffff6ded000-7ffff6df2000 ... /home/old/projects/libpatch/a.out
7ffff6df2000-7ffff71e2000 ... /memfd:libpatch:lib-trampoline (deleted)
...
7ffff7fc8000-7ffff7fcc000 ... [vvar]
7ffff7fcc000-7ffff7fce000 ... [vdso]
...
7ffff7fff000-7ffff83ef000 ... /memfd:libpatch:stack-trampoline (deleted)
7ffffffde000-7ffffffff000 ... [stack]
```

Trampoline searching

- Finding a trampoline is hard.
- Find a jump offset from the probe site to a free trampoline.
- The jump offset is actually a pattern of bytes.
- Many algorithms used for more patterns.

Fit algorithm

- Instruction at the probe site is at least 5 bytes.
- Works for around 40% of the instructions.
- Maximum performance.
- 256^4 patterns.

Alias algorithm

- The first bytes of the next instruction(s) become part of the offset.
- Maximum performance.
- Between 1 and 256^3 patterns.

Punning algorithm

- The next instruction(s) are hijacked.
- Traps (or illegal opcodes) have to be placed on basic block entries.
- If there's a indirect branching, assume that all instructions are basic block entries.
- Possibly lower performance if traps are hit.
- Between 23^4 and 256^4 patterns.

NOP padding algorithm

- 2 bytes jump to a padding area between two functions emitted at -O1.
- The padding is transformed into a mini-trampoline.
- Lower performance dues to additional jump.
- 256^4 patterns.

Trampoline searching (continuation)

- Search for trampoline is heavily optimized.
- Could benefit from the unification of the patterns.

A word about W^X protection

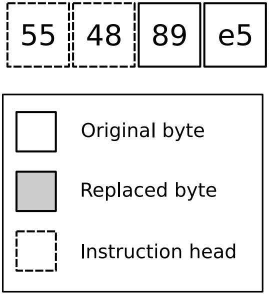
- There is support for write xor execute.
- Uses two virtuals mapping to same physical mapping.
- Breaks GDB and libdw when enabled.
- Not thoroughly tested!

Patching flow

- Punning algorithm was choose.
- There is a membarrier(2) between each step.

Original instructions

```
push %rbp          ;; 55
mov %rsp %rbp      ;; 48 89 e5
sub $0x10, %rsp    ;; 48 83 ec 10
```

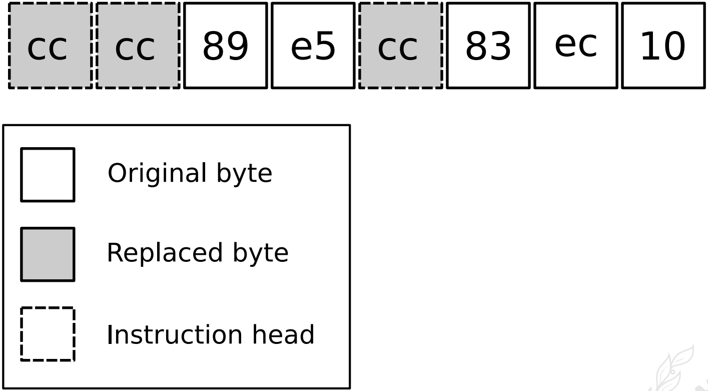


55	48	89	e5	48	83	ec	10

- □ Original byte
- ▨ Replaced byte
- ⬚ Instruction head

Patching flow (continuation)

Step 1 - Lock patches



cc cc 89 e5 cc 83 ec 10

□ Original byte

▨ Replaced byte

⬚ Instruction head

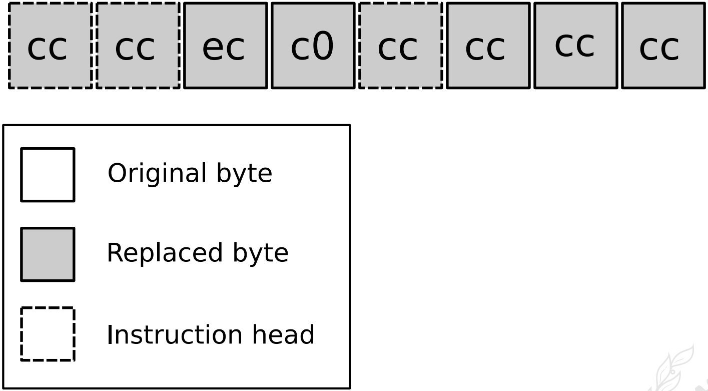
Patching flow (continuation)

Step 2

- Iterate over every thread with PTRACE_SEIZE.
- Check if the thread program counter is in a patching region.
- Check if the thread is in a signal handler and will return to a patching region.
- Move the thread to the corresponding OLX instruction.

Patching flow (continuation)

Step 3 - Replace bodies

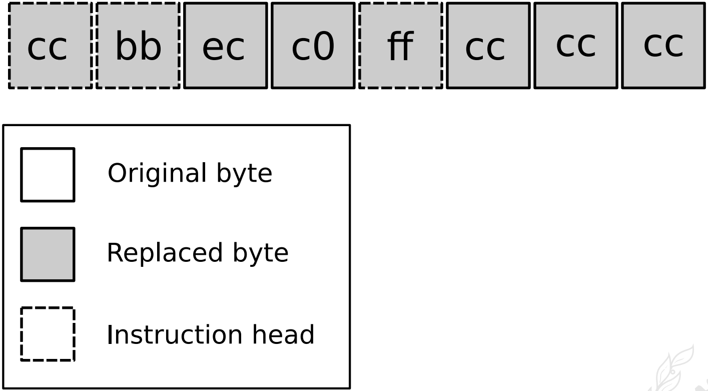


cc cc ec c0 cc cc cc cc

- □ Original byte
- ▨ Replaced byte
- ⬚ Instruction head

Patching flow (continuation)

Step 4 - Replace instruction's head



cc bb ec c0 ff cc cc cc

□ Original byte

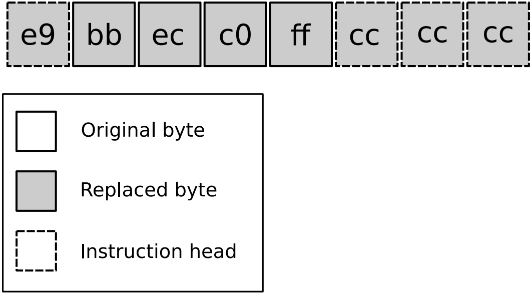
▨ Replaced byte

⬚ Instruction head

Patching flow (continuation)

Step 5 - Unlock patches

```
jmp 0xffc0ecbb ;; e9 bb ec c0 ff
int3           ;; cc
int3           ;; cc
int3           ;; cc
```



e9	bb	ec	c0	ff	cc	cc	cc

□ Original byte

▨ Replaced byte

⬚ Instruction head

Unpatching flow

- Same steps as patching.
- No need for moving threads out of critical regions.

Patch reclamation

- Restoring the bytes at the probe site is not enough.

Patch reclamation

- Restoring the bytes at the probe site is not enough.
- Any number of threads can already be in a patch flow of a removed patch.

Patch reclamation

- Restoring the bytes at the probe site is not enough.
- Any number of threads can already be in a patch flow of a removed patch.
- How can we reclaim the trampoline(s) and the OLX buffer?

Patch reclamation

- Restoring the bytes at the probe site is not enough.
- Any number of threads can already be in a patch flow of a removed patch.
- How can we reclaim the trampoline(s) and the OLX buffer?
- Libpatch does not use mutex nor reference counter.

Patch reclamation

- Restoring the bytes at the probe site is not enough.
- Any number of threads can already be in a patch flow of a removed patch.
- How can we reclaim the trampoline(s) and the OLX buffer?
- Libpatch does not use mutex nor reference counter.
- We know however that no new thread can enter a removed pflow.

Patch reclamation

- Restoring the bytes at the probe site is not enough.
- Any number of threads can already be in a patch flow of a removed patch.
- How can we reclaim the trampoline(s) and the OLX buffer?
- Libpatch does not use mutex nor reference counter.
- We know however that no new thread can enter a removed pflow.
- A naive solution would be to wait a grace period.

Patch reclamation

- Restoring the bytes at the probe site is not enough.
- Any number of threads can already be in a patch flow of a removed patch.
- How can we reclaim the trampoline(s) and the OLX buffer?
- Libpatch does not use mutex nor reference counter.
- We know however that no new thread can enter a removed pflow.
- A naive solution would be to wait a grace period.
- Libpatch solution is to unwind the thread stack and check for the addresses of the pflow nodes.

Patch reclamation

- Restoring the bytes at the probe site is not enough.
- Any number of threads can already be in a patch flow of a removed patch.
- How can we reclaim the trampoline(s) and the OLX buffer?
- Libpatch does not use mutex nor reference counter.
- We know however that no new thread can enter a removed pflow.
- A naive solution would be to wait a grace period.
- Libpatch solution is to unwind the thread stack and check for the addresses of the pflow nodes.
- Multiple policies.
  - Busy loop until the threads are out of the pflow.
  - Do not free the pflow if any thread is in it. Wait until next garbage collection.

Summary

1. About Libpatch
2. Libpatch's concepts
3. Problem domain
4. Results
5. Conclusion

Coverage

Coverage result with a corpus of 124 binaries from 50 packages

Binary	Instruction success rate	Fentry success rate
average	0.990202	0.997514
deviation	0.004104	0.006845
weighted average	0.995122	0.991632
weighted deviation	0.002905	0.014164
geometric average	0.990193	0.997490
geometric deviation	1.004170	1.006961

Coverage (continuation)

Algorithm chosen (sorted by attempt)

Algorithm	Amount	Proportion [%]
fit	8538178	40.9899
nop	3054066	14.6619
alias	38335	0.1840
punning	9199380	44.1642
total	20829959	100

Coverage (continuation)

Coverage result with a corpus of 124 binaries from 50 packages for functions that are more than 64 bytes

Binary	Instruction success rate	Fentry success rate
average	0.988211	0.992155
deviation	0.004929	0.030895
weighted average	0.993113	0.983421
weighted deviation	0.003781	0.022606
geometric average	0.988198	0.991601
geometric deviation	1.005036	1.035259

Coverage (continuation)

Algorithm chosen (sorted by attempt)

Algorithm	Amount	Proportion [%]
fit	8346893	40.1657
nop	2782318	13.3887
alias	42558	0.2048
punning	9609383	46.2409
total	20781152	100

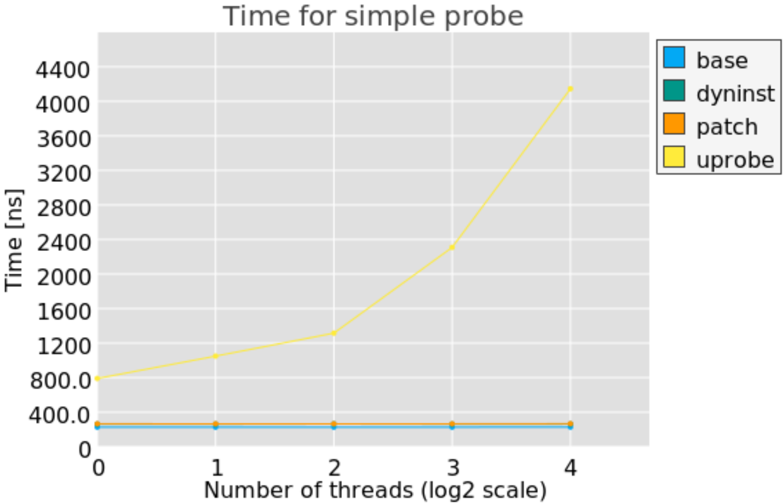
Micro-benchmark

- AMD Ryzen 9 5950X
- SMT (Hyper-threading) disabled
- Frequency boosting disabled
- processor.max_cstate=1
- idle=poll
- Stack depth is more than 128 call frames
- 10^7 loops

Micro-benchmark (continuation)

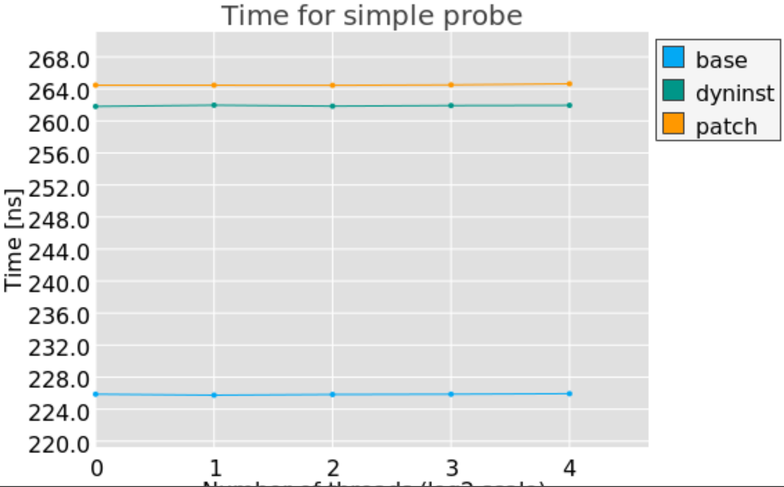
```
void not_a_leaf(void) { }

uint64_t powmod(uint64_t b, uint64_t e, uint64_t m)
{
        uint64_t c;

        not_a_leaf();

        c = 1;
        for(size_t i=0; i<e; ++i) {
                c = (c * b) % m;
        }

        return c;
}
```

Micro-benchmark (continuation)

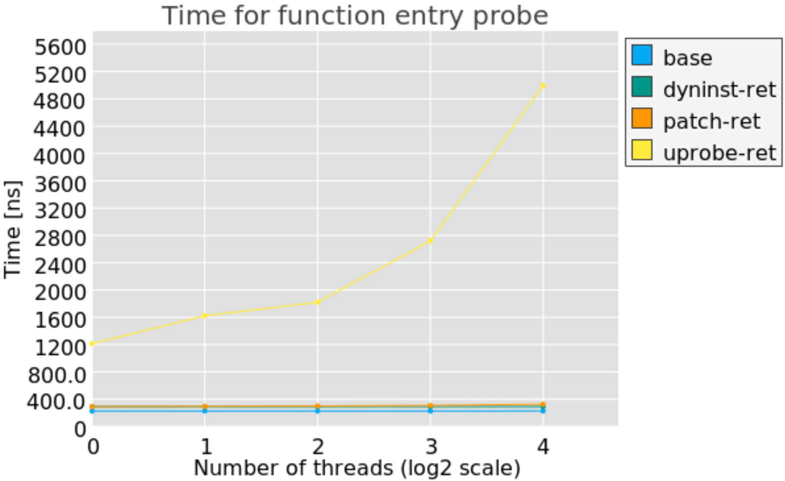


Micro-benchmark (continuation)

Without uprobe

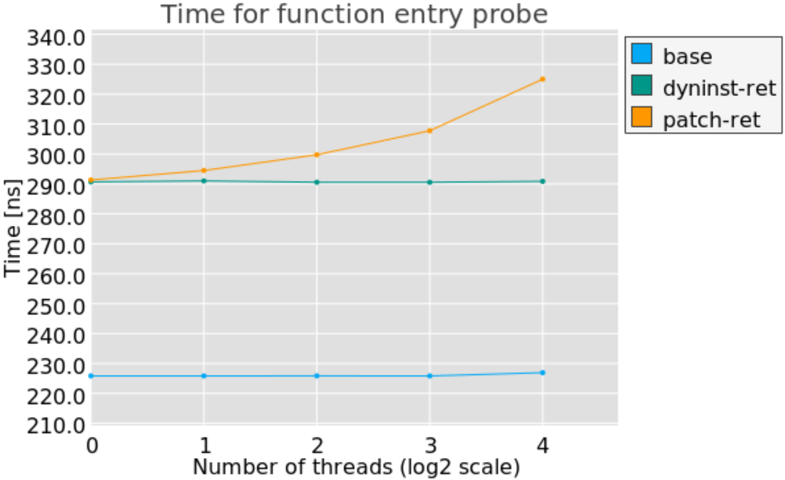


Micro-benchmark (continuation)

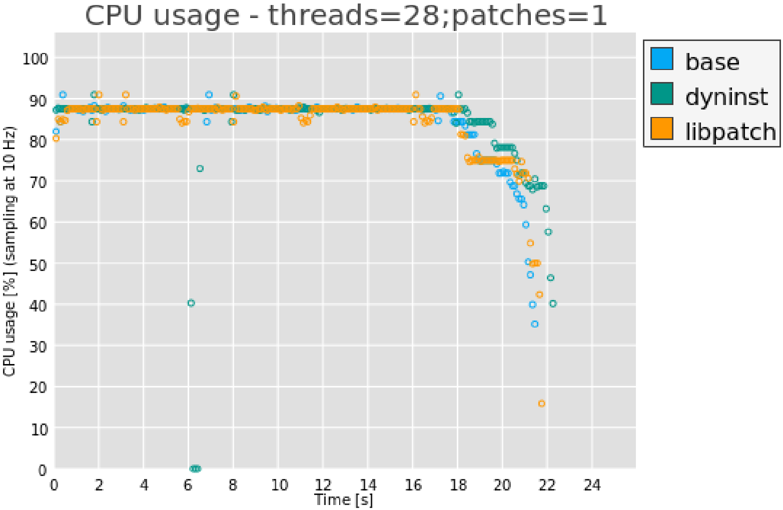


Micro-benchmark (continuation)

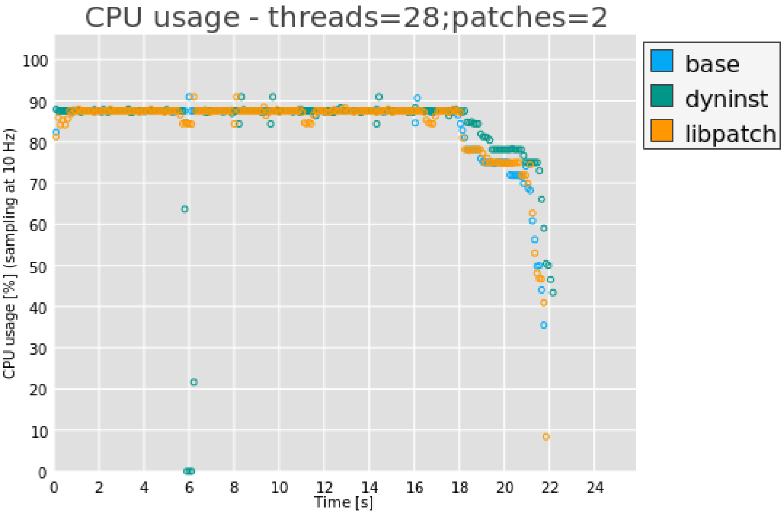
Without uprobe



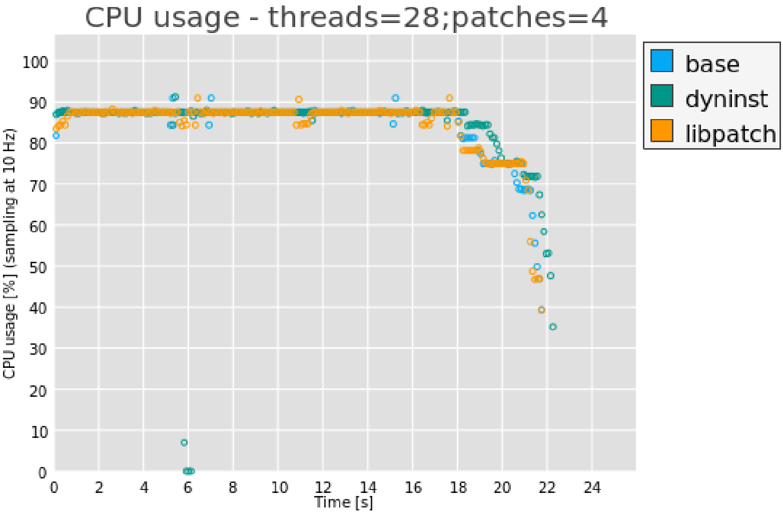
Toggle-benchmark



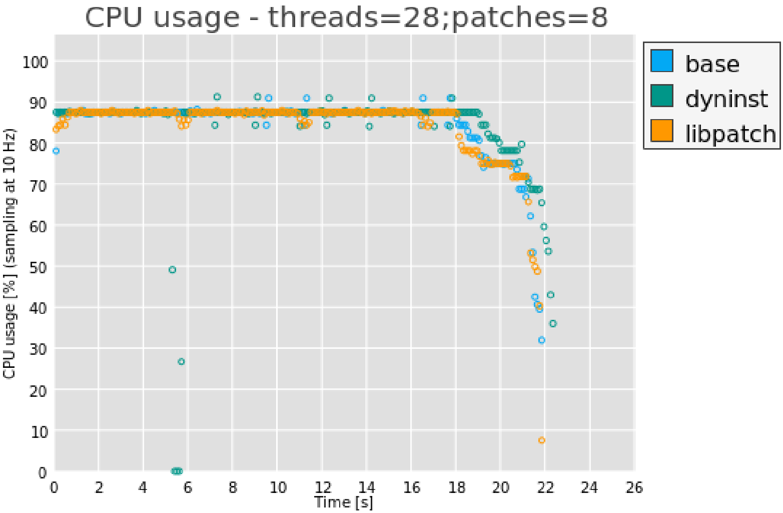
Toggle-benchmark (continuation)



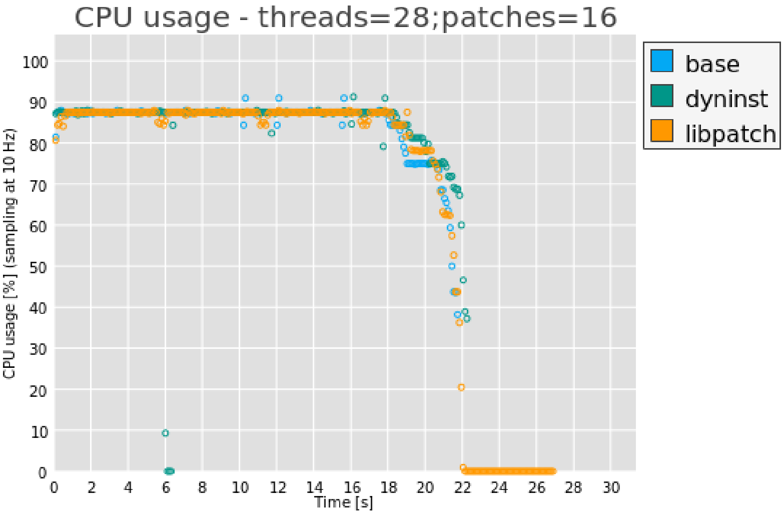
Toggle-benchmark (continuation)



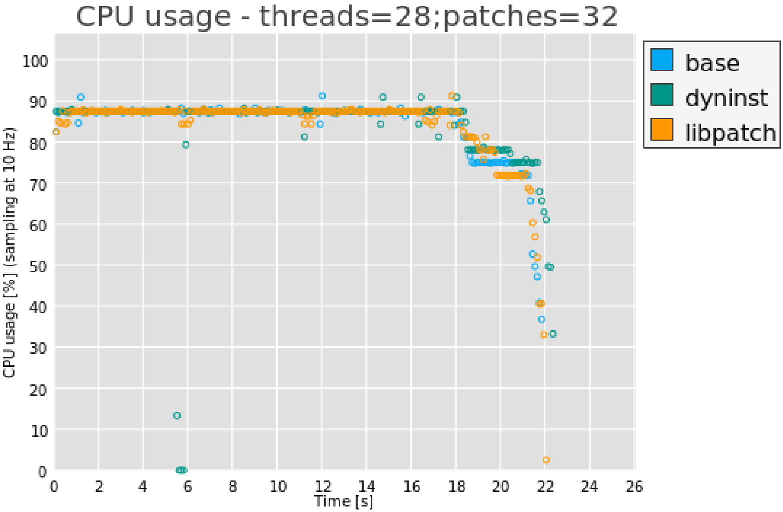
Toggle-benchmark (continuation)



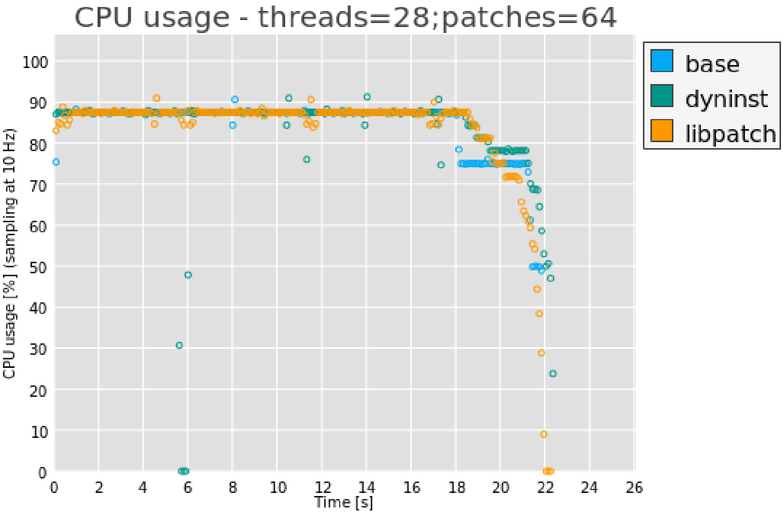
Toggle-benchmark (continuation)



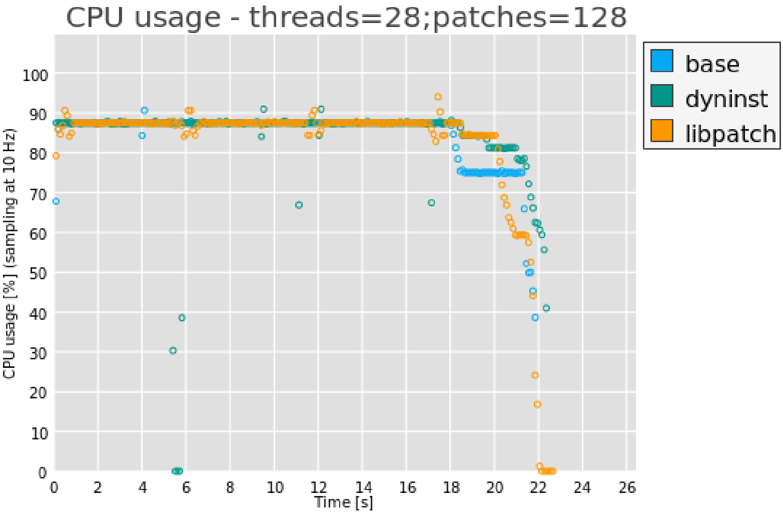
Toggle-benchmark (continuation)



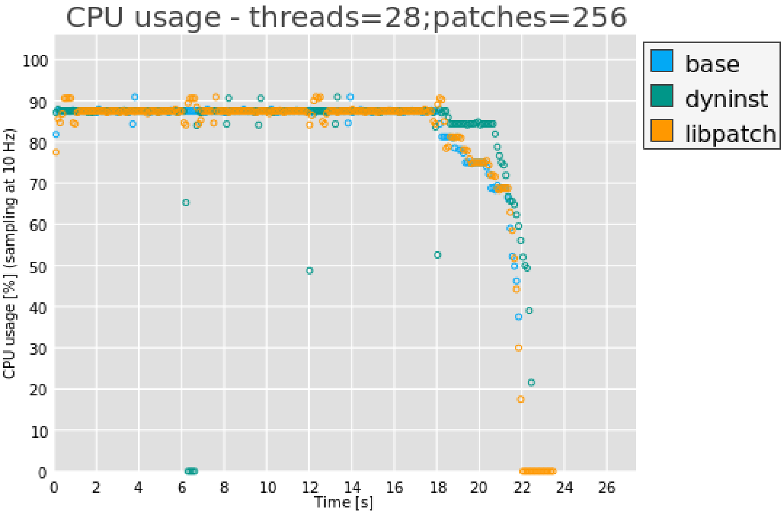
Toggle-benchmark (continuation)



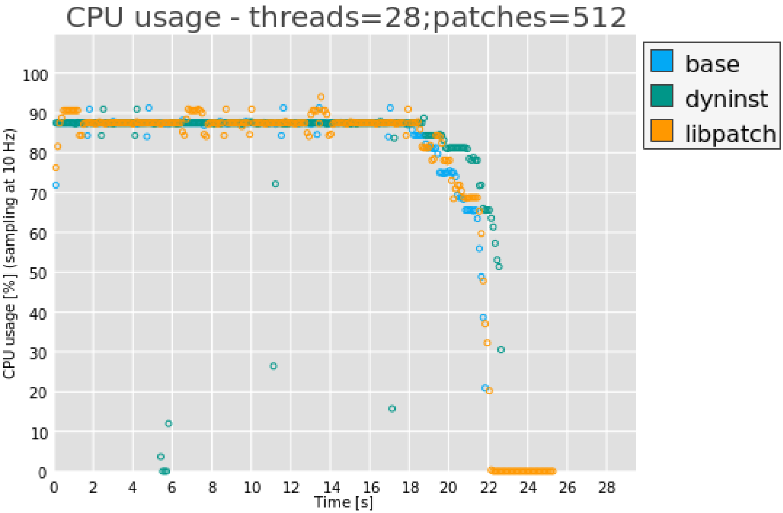
Toggle-benchmark (continuation)



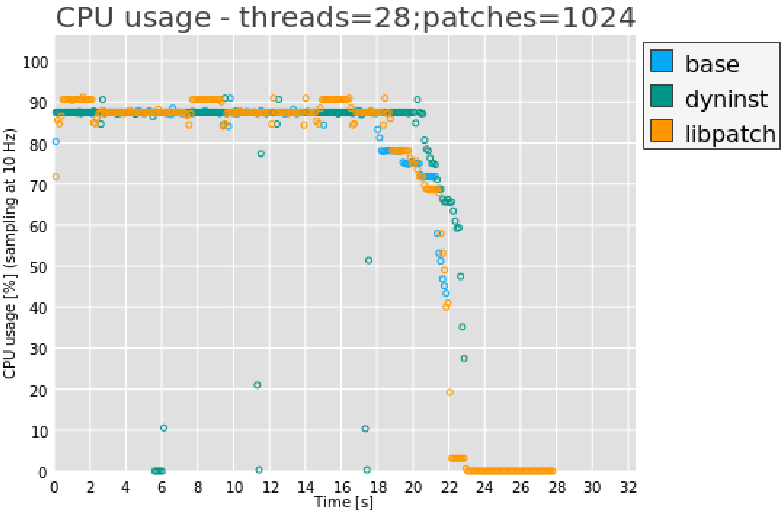
Toggle-benchmark (continuation)



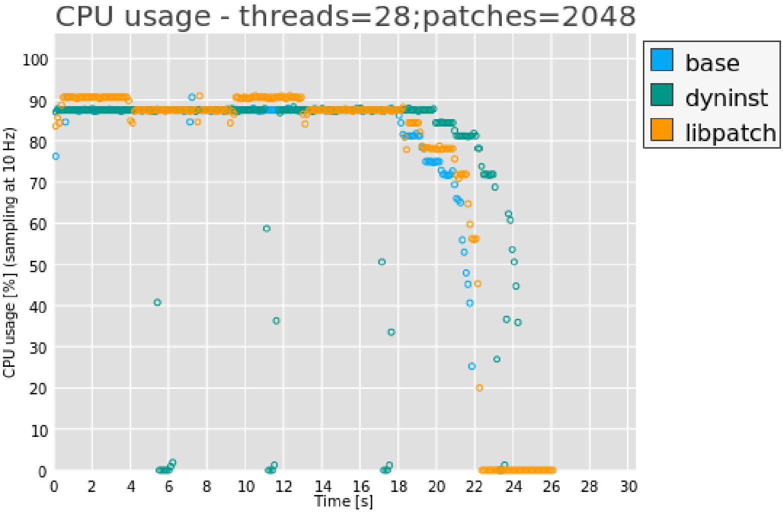
Toggle-benchmark (continuation)



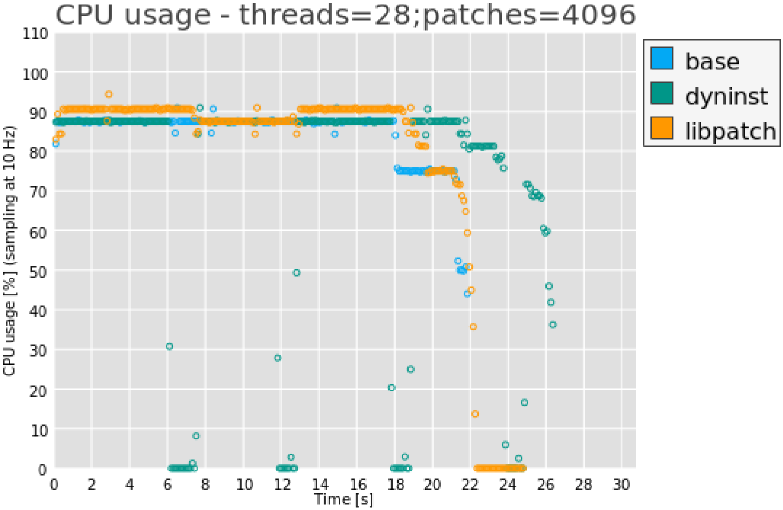
Toggle-benchmark (continuation)



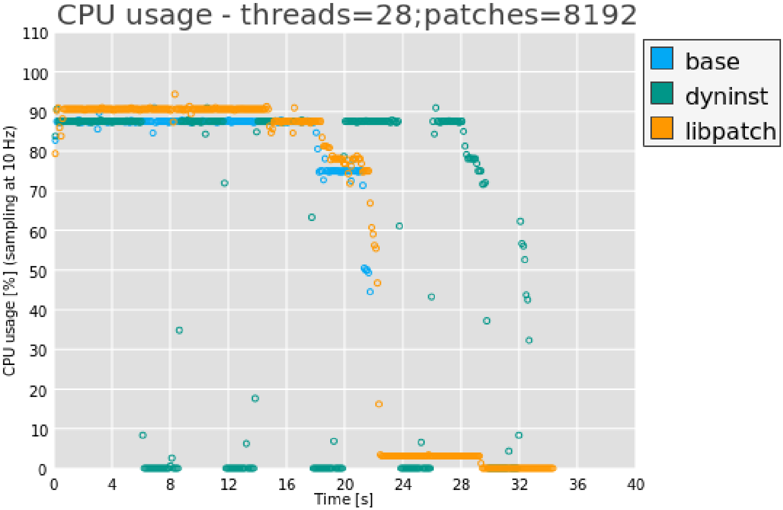
Toggle-benchmark (continuation)



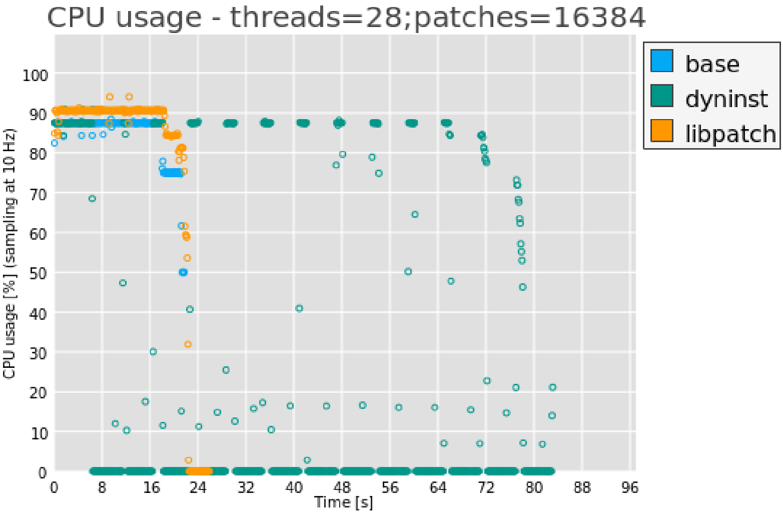
Toggle-benchmark (continuation)



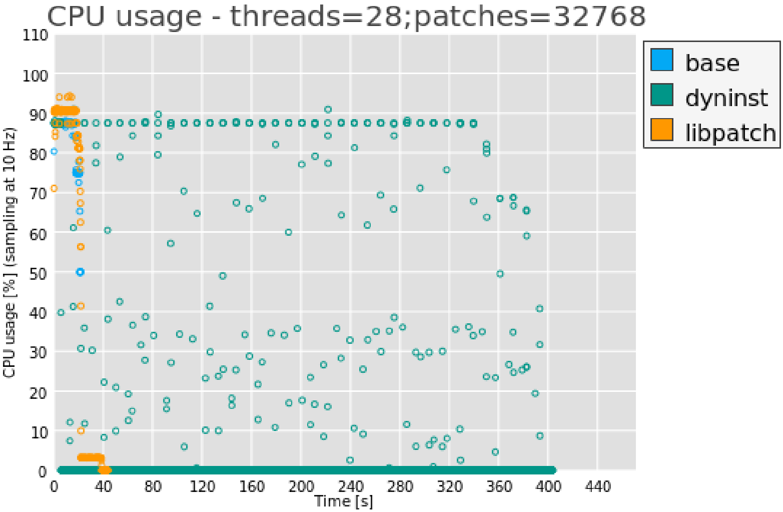
Toggle-benchmark (continuation)



Toggle-benchmark (continuation)



Toggle-benchmark (continuation)



Summary

1. About Libpatch
2. Libpatch's concepts
3. Problem domain
4. Results
5. Conclusion

Discussion

- DWARF basic blocks
  - There is an old patch for gcc.
  - But it never got merged fault of use case.
  - Would make Libpatch less conservative about indirect branches.

Discussion

- DWARF basic blocks
  - There is an old patch for gcc.
  - But it never got merged fault of use case.
  - Would make Libpatch less conservative about indirect branches.
- Need feedbacks for
  - Scenarios where the overhead of the GC is too high.
  - Memory usage.
  - The public API.
  - Error reporting.
  - W^X systems.

Conclusion

- Libpatch is currently only supported for x86-64.
- Support for ARM in the future.
- Almost two 9 for global coverage and already two 9 for function entry.
- Runtime overhead is close to Dyninst without its program analysis time and memory overhead.
- Insertion overhead scales well.

Questions

Questions?