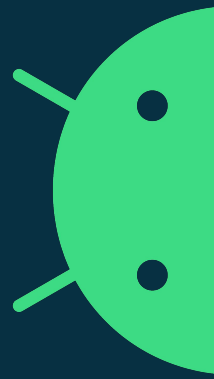
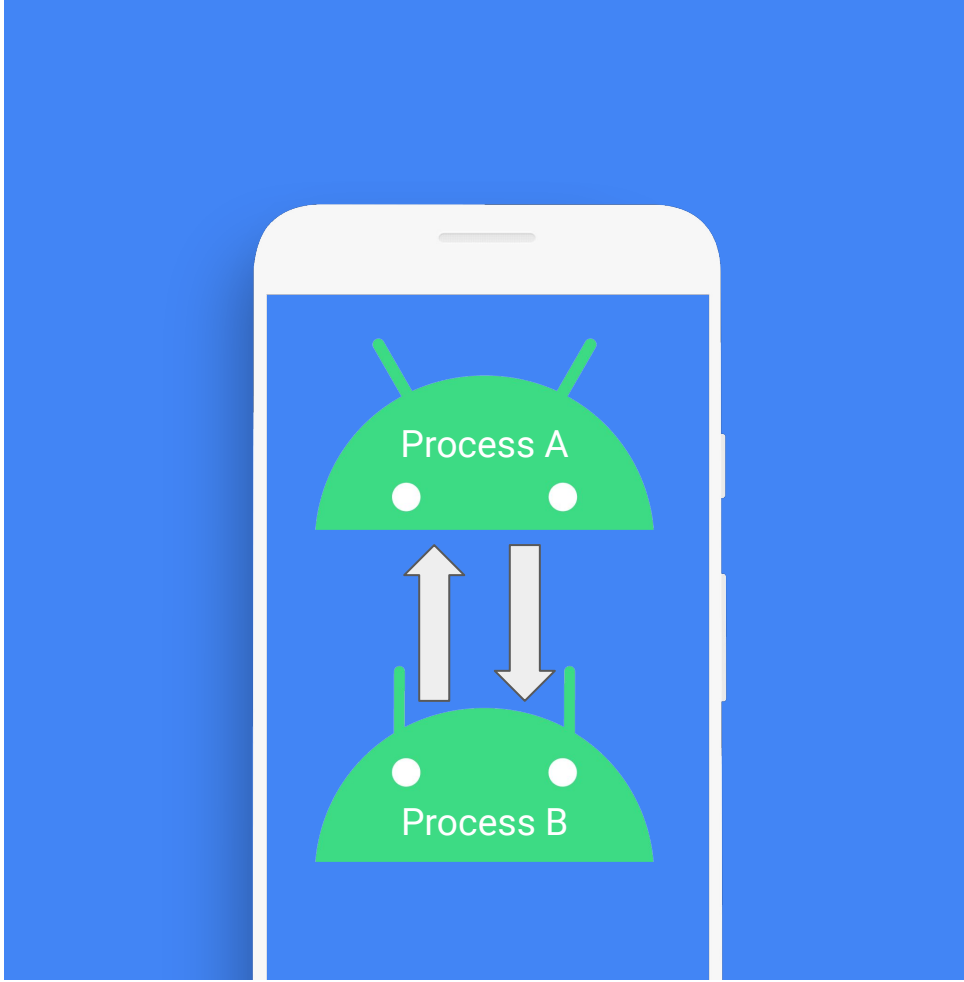
Setting up Binder for the future

Alice Ryhl and Carlos Llamas



What is Binder?

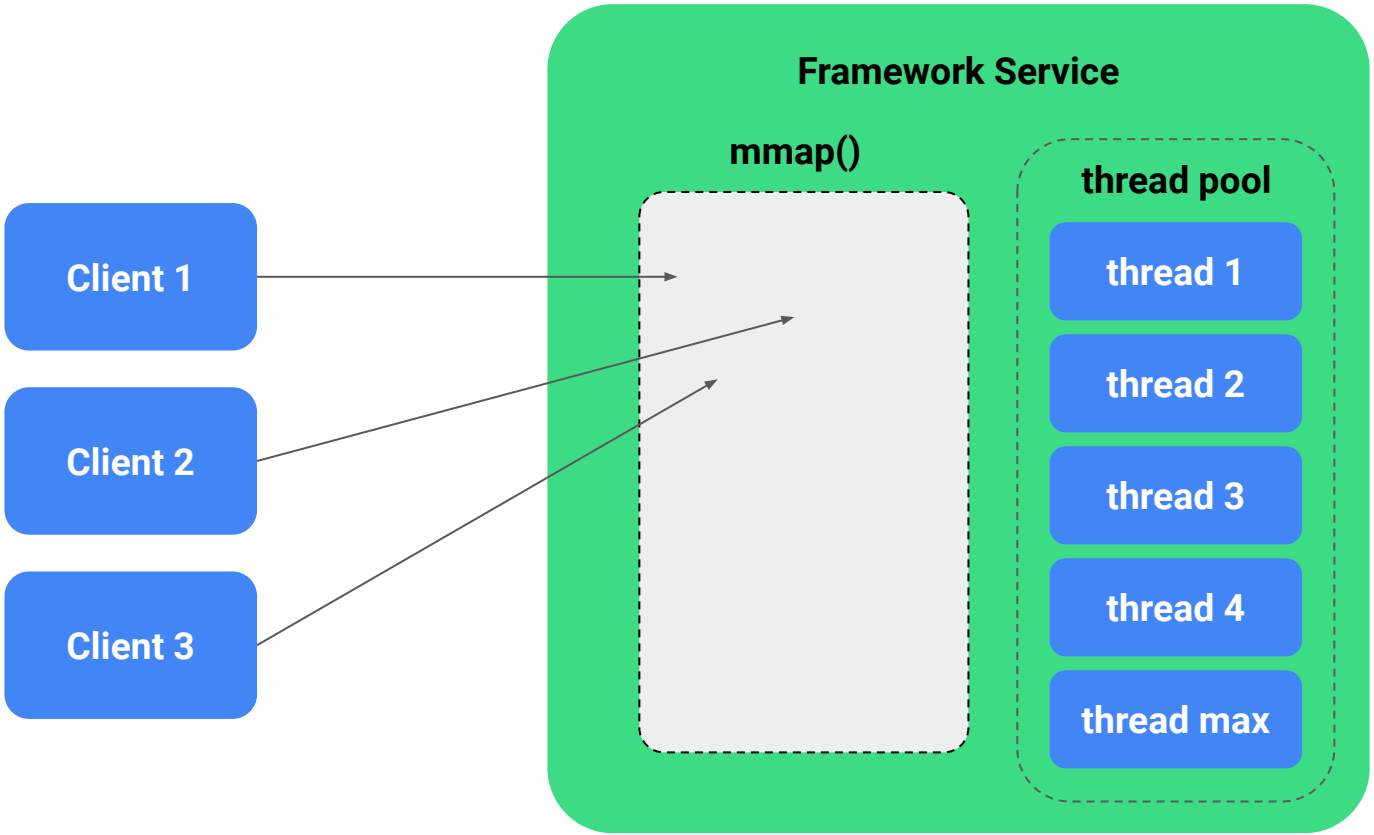
Binder is used for communication between processes in Android



What is binder?

- Yet another **IPC/RPC**
- BeOS -> PalmOS -> **Android**
- Used **extensively** in Android
- Provides access to framework services
- Composed of **libbinder** (userspace) and binder **driver**

Client-Server Model

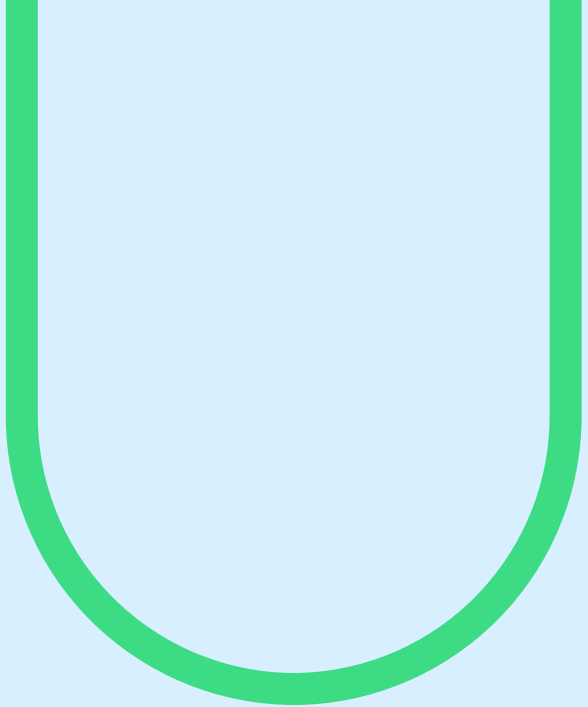


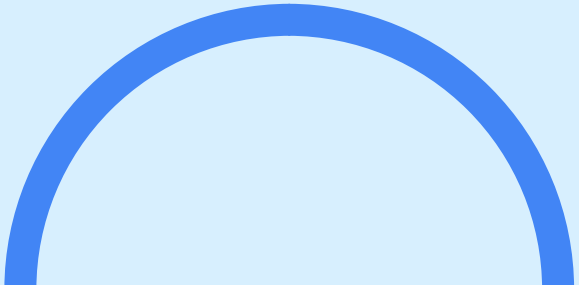
android

Why binder driver?

- Zero in-kernel buffering
- Write and read in a single ioctl()
- Priority Inheritance
- Share file descriptors
- Remote object management
- Weak/Strong reference counting
- Death notifications
- and many more…

Why rewrite Binder?





Challenges that Binder faces

High complexity

Accumulated technical debt

Security issues

High complexity makes it difficult to resolve tech debt without causing security issues.

Security issues in Binder

1

High vulnerability density

Binder's density is around 3.1 vulnerabilities per kLOC.

2

Not getting better

Binder has averaged ~3 high/critical severity vulnerabilities per year over the past 6 years.

3

Risk is not theoretical

We are aware of exploits for about half of the vulnerabilities in Binder.

4

Security critical

Even Android's most de-privileged sandboxes have direct access to Binder.

Binder has high complexity

- Binder is full of **complex** features
- It must do all of this as fast and **efficiently** as possible.
- Minor **performance** regressions can cause a noticeably degraded user experience.

Things to improve

Thousand line functions

Error-prone error handling

Improving the code is risky

Not a unique example

Things to improve

Thousand line functions

Error-prone error handling

Improving the code is risky

Not a unique example

```
err_dead_proc_or_thread:
        binder_txn_error("%d:%d dead process or thread\n",
                thread->pid, proc->pid);
        return_error_line = __LINE__;
        binder_dequeue_work(proc, tcomplete);
err_translate_failed:
err_bad_object_type:
err_bad_offset:
err_bad_parent:
err_copy_data_failed:
        binder_cleanup_deferred_txn_lists(&sgc_head, &pf_head);
        binder_free_txn_fixups(t);
        trace_binder_transaction_failed_buffer_release(t->buffer);
        binder_transaction_buffer_release(target_proc, NULL, t->buffer,
                                                buffer_offset, true);
        if (target_node)
                binder_dec_node_tmpref(target_node);
        target_node = NULL;
        t->buffer->transaction = NULL;
        binder_alloc_free_buf(&target_proc->alloc, t->buffer);
err_binder_alloc_buf_failed:
err_bad_extra_size:
        if (secctx)
                security_release_secctx(secctx, secctx_sz);
err_get_secctx_failed:
        kfree(tcomplete);
        binder_stats_deleted(BINDER_STAT_TRANSACTION_COMPLETE);
err_alloc_tcomplete_failed:
        if (trace_binder_txn_latency_free_enabled())
                binder_txn_latency_free(t);
        kfree(t);
        binder_stats_deleted(BINDER_STAT_TRANSACTION);
err_alloc_t_failed:
err_bad_todo_list:
err_bad_call_stack:
err_empty_call_stack:
err_dead_binder:
err_invalid_target_handle:
        if (target_node) {
                binder_dec_node(target_node, 1, 0);
                binder_dec_node_tmpref(target_node);
        }

        binder_debug(BINDER_DEBUG_FAILED_TRANSACTION,
                     "%d:%d transaction %s to %d:%d failed %d/%d/%d, size %lld-%lld line %d\n",
                     proc->pid, thread->pid, reply ? "reply" :
                     (tr->flags & TF_ONE_WAY ? "async" : "call"),
                     target_proc ? target_proc->pid : 0,
                     target_thread ? target_thread->pid : 0,
                     t_debug_id, return_error, return_error_param,
                     (u64)tr->data_size, (u64)tr->offsets_size,
                     return_error_line);

        if (target_thread)
                binder_thread_dec_tmpref(target_thread);
        if (target_proc)
                binder_proc_dec_tmpref(target_proc);

        {
                struct binder_transaction_log_entry *fe;

                e->return_error = return_error;
                e->return_error_param = return_error_param;
                e->return_error_line = return_error_line;
```

android

Things to improve

Thousand line functions

Error-prone error handling

Improving the code is risky

Not a unique example

CVE-2020-0041 was introduced during refactoring

The fix:

```
diff --git a/drivers/android/binder.c b/drivers/android/binder.c
index e9bc9fc..b2dad43 100644
--- a/drivers/android/binder.c
+++ b/drivers/android/binder.c
@@ -3310,7 +3310,7 @@
            binder_size_t parent_offset;
            struct binder_fd_array_object *fda =
                to_binder_fd_array_object(hdr);
-           size_t num_valid = (buffer_offset - off_start_offset) *
+           size_t num_valid = (buffer_offset - off_start_offset) /
                       sizeof(binder_size_t);
            struct binder_buffer_object *parent =
                binder_validate_ptr(target_proc, t->buffer,
@@ -3384,7 +3384,7 @@
                t->buffer->user_data + sg_buf_offset;
            sg_buf_offset += ALIGN(bp->length, sizeof(u64));

-           num_valid = (buffer_offset - off_start_offset) *
+           num_valid = (buffer_offset - off_start_offset) /
                     sizeof(binder_size_t);
            ret = binder_fixup_parent(t, thread, bp,
                         off_start_offset,
```

Things to improve

Thousand line functions

Error-prone error handling

Improving the code is risky

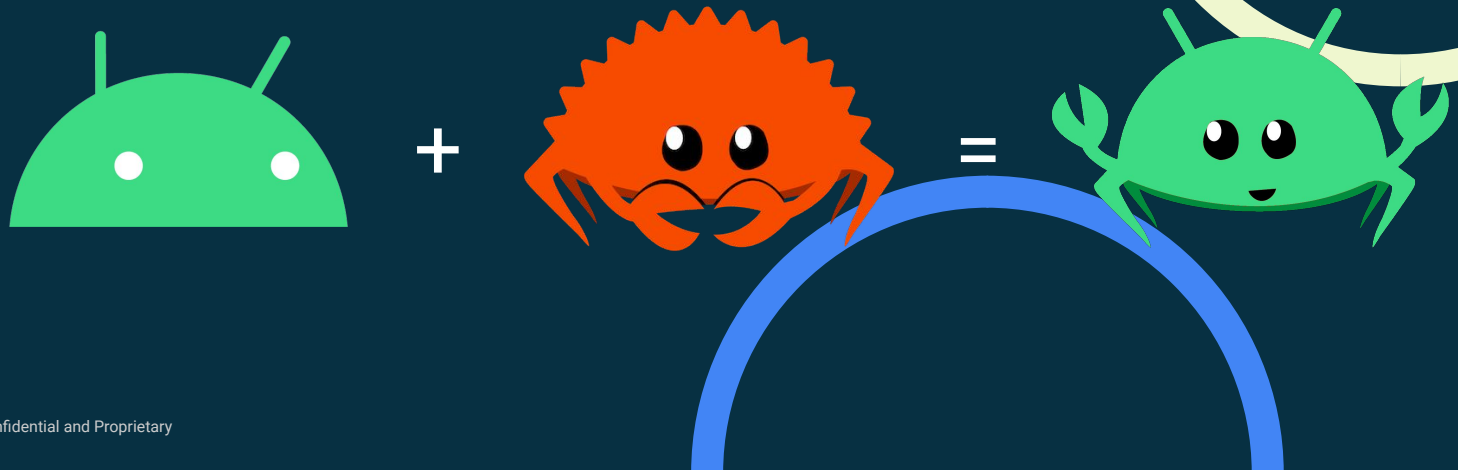
Not a unique example

```
* bde4a19fc04f ("binder: use userspace pointer as base of buffer space”)
* 16981742717b ("binder: fix incorrect calculation for num_valid")
```

```
* 44d8047f1d87 ("binder: use standard functions to allocate fds")
* 32e9f56a96d8 ("binder: don't detect sender/target during buffer cleanup")
* bdc1c5fac982 ("binder: fix UAF caused by faulty buffer cleanup")
```

```
* a0c2baaf81bd ("android: binder: Don't get mm from task")
* da1b9564e85b ("android: binder: fix the race mmap and alloc_new_buf_locked")
* a43cfc87caaf ("android: binder: Stop saving a pointer to the VMA")
* b0cab80ecd54 ("android: binder: fix lockdep check on clearing vma")
* 44e602b4e52f ("binder_alloc: add missing mmap_lock calls when using the VMA")
* 1da52815d5f1 ("binder: fix alloc->vma_vm_mm null-ptr dereference")
* 3ce00bb7e91c ("binder: validate alloc->mm in ->mmap() handler")
```

Rust Binder



Why is Rust a good fit?

Rust makes ownership visible in the type system and enforces it

In C

```
/*
 * The caller must have taken a
 * temporary ref on the node.
 */
BUG_ON(!node->tmp_refs);
```

In Rust

```
// This does not compile if caller
// has not taken a temporary ref on
// self.
fn my_fn(self: &Arc<Self>) {
```

Why is Rust a good fit?

Rust makes ownership visible in the type system and enforces it

In C

```
/*
 * @from, @to_proc, and @to_thread can be set
 * to NULL during thread teardown
 */
struct binder_transaction {
     struct binder_work work;
     struct binder_thread *from;
     struct binder_transaction *from_parent;
     struct binder_proc *to_proc;
     struct binder_thread *to_thread;
     struct binder_transaction *to_parent;
     struct list_head fd_fixups;
     /* + other fields */
}
```

Read the code to find out whether these own a ref-count

Compiler does not check this

In Rust

```
pub(crate) struct Transaction {
    node_ref: Option<NodeRef>,
    stack_next: Option<Arc<Transaction>>,
    from: Arc<Thread>,
    to: Arc<Process>,
    file_list: SpinLock<List<Box<FileInfo>>>,
    // + other fields
}
```

Exclusive ownership

Lock protects only this field

Owns a ref-count, never null

Owns a ref-count but nullable

Why is Rust a good fit?

You can’t forget to clean up

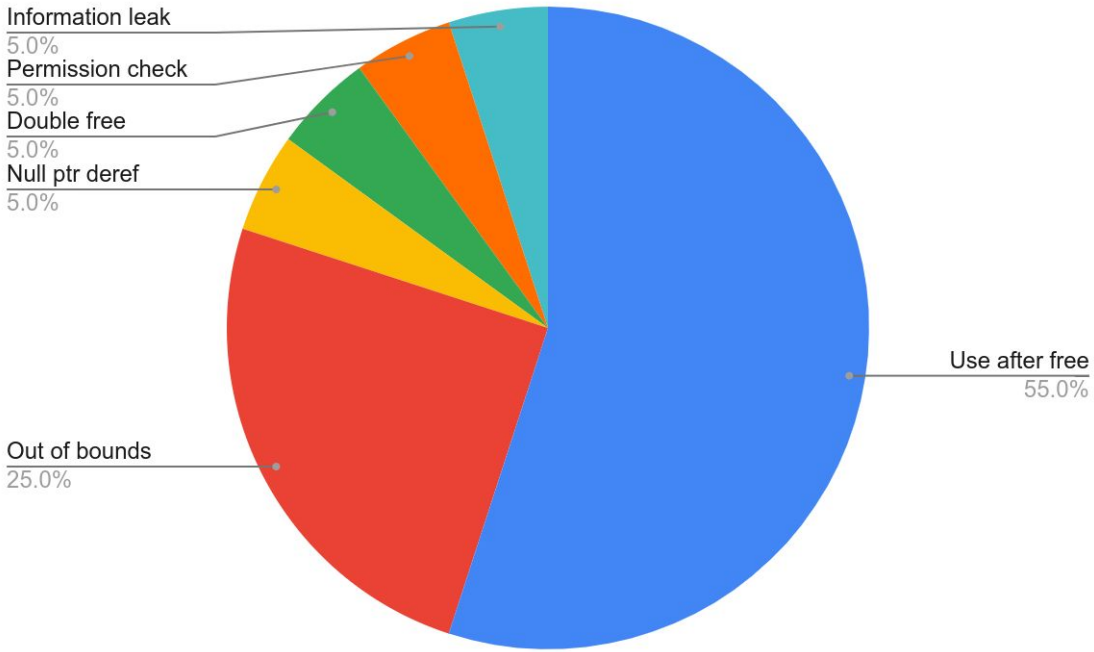
In C

```
err_translate_failed:
err_bad_object_type:
err_bad_offset:
err_bad_parent:
err_copy_data_failed:
         binder_cleanup_deferred_txn_lists(&sgc_head, &pf_head);
         binder_free_txn_fixups(t);
         trace_binder_transaction_failed_buffer_release(t->buffer);
         binder_transaction_buffer_release(target_proc, NULL, t->buffer,
                                                 buffer_offset, true);
         if (target_node)
                  binder_dec_node_tmpref(target_node);
         target_node = NULL;
         t->buffer->transaction = NULL;
         binder_alloc_free_buf(&target_proc->alloc, t->buffer);
err_binder_alloc_buf_failed:
err_bad_extra_size:
         if (secctx)
                  security_release_secctx(secctx, secctx_sz);
err_get_secctx_failed:
         kfree(tcomplete);
         binder_stats_deleted(BINDER_STAT_TRANSACTION_COMPLETE);
err_alloc_tcomplete_failed:
         if (trace_binder_txn_latency_free_enabled())
                  binder_txn_latency_free(t);
         kfree(t);
         binder_stats_deleted(BINDER_STAT_TRANSACTION);
err_alloc_t_failed:
err_bad_todo_list:
err_bad_call_stack:
err_empty_call_stack:
err_dead_binder:
err_invalid_target_handle:
         /* it keeps going … */
```

In Rust

```
}
```

Rust prevents almost all vulnerabilities in Binder



android

Rust Binder

Feature parity

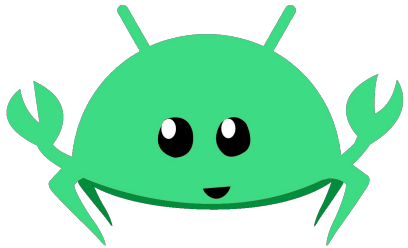
Implements all features in C Binder.

(except for some debugging facilities)

Passes tests

Passes all Binder tests in aosp.

Can boot a device and run a variety of apps without issues.

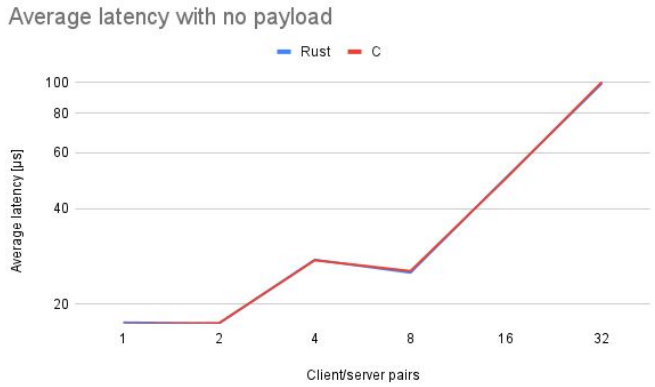


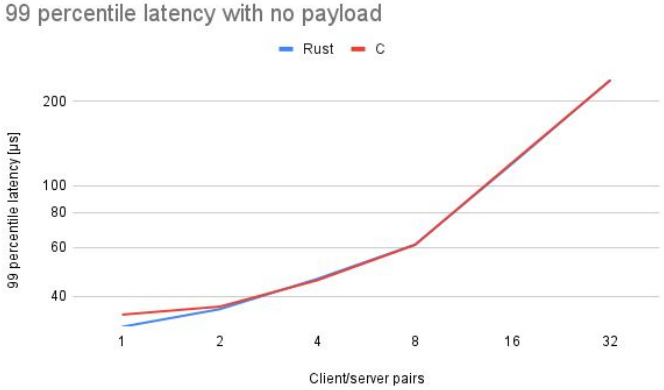
Promising performance

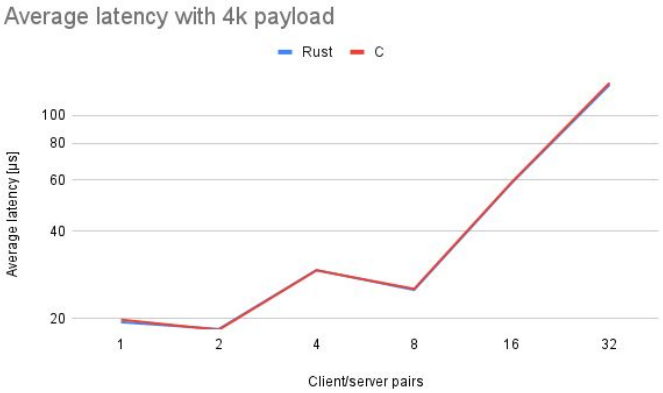
On a simple benchmark, drivers have similar performance.

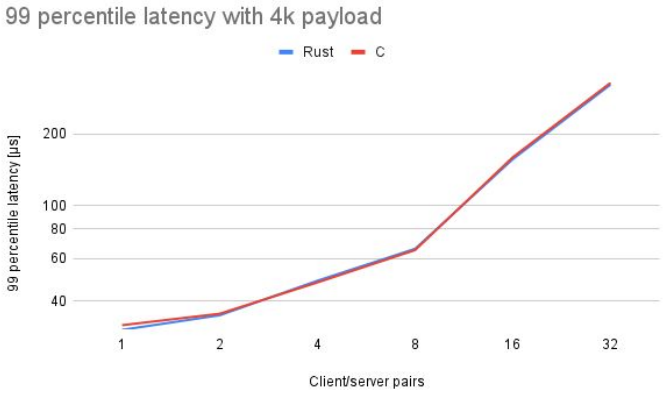
Still a lot of work to do.

Performance looks promising









Code size

Rust spends less code on handling errors.

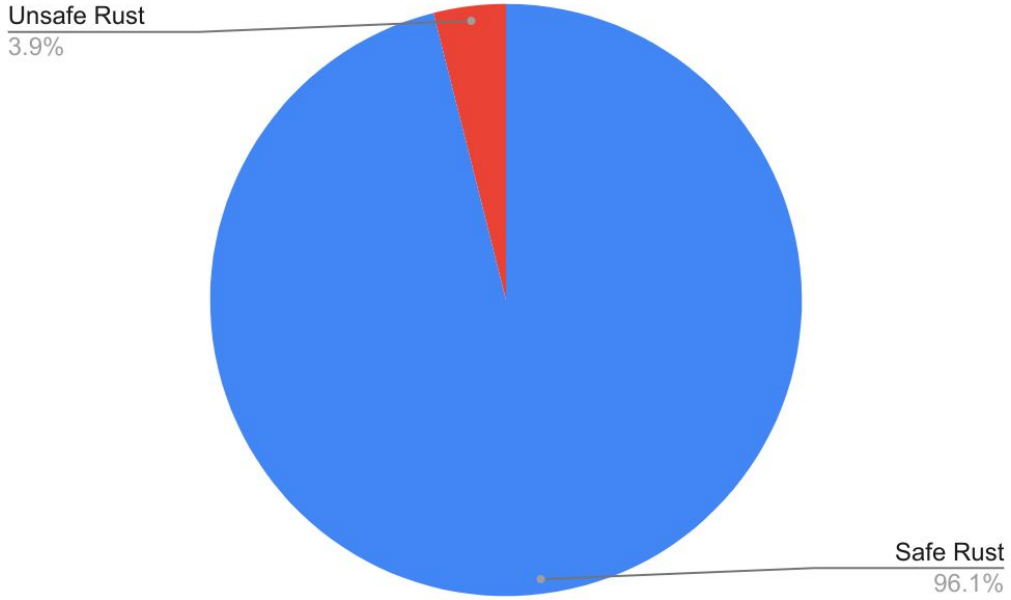
- Rust Binder
- C Binder

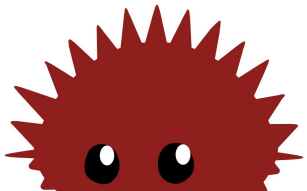
android

What about unsafe?

The majority is due to binderfs.

- Safe Rust in Binder
- Unsafe Rust in Binder

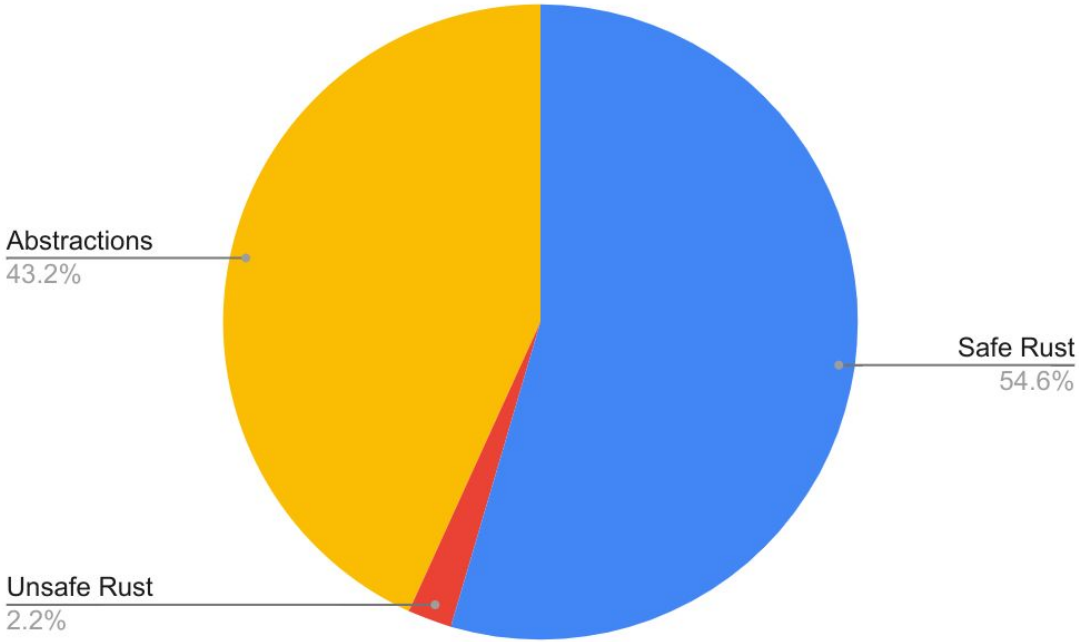


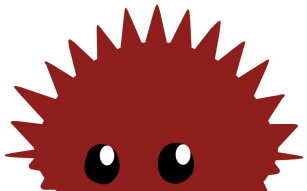


What about abstractions?

You only have to get them right once, across all drivers.

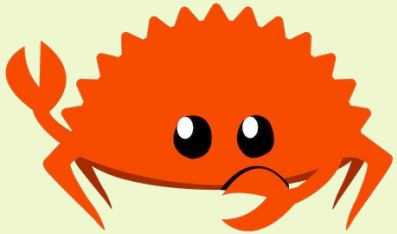
- Safe Rust in Binder
- Unsafe Rust in Binder
- Abstractions





"If you can implement Binder in Rust, you can implement any driver in Rust"

— Ricardo Ribalda (ChromeOS kernel engineer)



Thank you for your attention!