




# SafeVerbs

---

**SafeVerbs** implements a memory-safe RDMA API in Rust. It leverages Rust's type system to ensure safety properties at compile time, simplifying the development of high-performance RDMA applications.

-  **Memory-Safe:** Prevents common RDMA programming pitfalls such as premature memory reuse, resource mismanagement, and invalid memory access.
-  **User-Friendly:** Encapsulates complex RDMA operations into intuitive, high-level abstractions.
-  **High Performance:** Provides zero-cost abstractions wherever possible and exposes unsafe APIs for advanced usage scenarios requiring maximum performance. Supports asynchronous programming for scalable applications.

## Getting Started

1. Clone the repo and its submodules.

```
git clone git@github.com:crazyboyjcjr/safeverbs.git --recursive
```

2. Install required packages and the Rust toolchain.

```
sudo apt update
sudo apt install libclang-dev libnuma-dev librdmacm-dev libibverbs-dev
cmake
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
cargo version
```

3. Build the codebase.

```
cargo build --release
```

## Devcontainer

You can alternatively set up the environment using the provided devcontainer for an isolated and pre-configured development setup.

## Run Examples

SafeVerbs includes example applications to demonstrate its usage. Ensure you have at least one machine (preferably two) with RDMA-capable NICs (RNIC) for running these examples.

### Greeter

A simple "Hello, World" example to verify that your setup and code are working as expected.

```
cd examples/greeter
cargo run --release --bin greeter_server
cargo run --release --bin greeter_client
```

The client will connect the server on the local machine. On a successful run, both the client and server should print the work completion and a message "Hello, SafeVerbs" should be printed on the server.

## UChan

A benchmark for RDMA Write operations on UC (unreliable connection). Results are comparable to [perftest](#).

```
$ cd examples/uchan
$ cargo run --release -- --help
UC Write Bandwidth Test.

Usage: uchan [OPTIONS]

Options:
  -c, --connect <CONNECT>    The address to connect, can be an IP address
                             or domain name. If not specified, the binary runs as a server that listens
                             on 0.0.0.0
  -p, --port <PORT>          The port number to use [default: 5000]
  -d, --device <DEVICE>      The IB device to use [default: mlx5_0]
  -t, --tx-depth <TX_DEPTH>  Send queue depth (max_send_wr) [default:
128]
  -n, --num-iters <NUM_ITERS> Total number of iterations [default: 5000]
  -w, --warmup <WARMUP>      Number of warmup iterations [default: 100]
  -s, --size <SIZE>          Message size [default: 65536]
  -h, --help                  Print help
```

Start a receiver (server) on one machine.

```
cargo run --release
```

Start a sender (client) on another machine.

```
cargo r --release -- -s 65536 -c <server_addr> --num-iters 1000000
```

## Code Structure

SafeVerbs is inspired by [rust-ibverbs](#), which provides simple abstractions for RDMA but leaves data path operations marked as unsafe. SafeVerbs builds upon and enhances this foundation by introducing memory safety and usability improvements. The rust-ibverbs crate is included as a submodule with minimal modifications stored in the [safeverbs branch](#).

- [src/](#): Core implementation of SafeVerbs.
- [examples/](#): Sample applications demonstrating the use of SafeVerbs in real-world scenarios.
- [demo/](#): Lightweight examples showcasing the key differences and improvements over libibverbs in C.
- [rust-ibverbs/](#): Submodule containing the base implementation of rust-ibverbs, with SafeVerbs-specific modifications.

## Design Highlights

RDMA programming with C libraries like [libibverbs](#) is powerful but prone to errors due to implicit contracts and manual memory management. SafeVerbs addresses these challenges:

### Data Path

- **Type-Safe Memory Regions:** Prevents data corruption by ensuring only valid types can be used in RDMA operations.
- **Memory Segments:** Supports multiple readers and multiple **non-overlapping** writers on the same memory region with memory safety guarantees. This design addresses a common issue in C, where [post\\_send](#) and [post\\_recv](#) operations on the same memory region can create data races. In SafeVerbs, such behavior is explicitly forbidden, as writable memory segments are guaranteed never to overlap with any other memory segments. While this safety enforcement cannot be made with zero cost, unsafe unchecked versions are available for high-performance use cases where safety checks can be bypassed by the user.
- **Asynchronous Operations:** [post\\_send](#) and [post\\_recv](#) are represented as [Futures](#), enabling seamless integration with Rust's async ecosystem.
  - All resources are [Send + Sync](#).
  - The [Future](#) representing an RDMA operation implements [Send + Sync + 'static](#), making it compatible with async libraries.
  - Expose unsafe functions for send requests that do not require NIC to generate a completion notification.

### Control Path

- **Typestate Pattern:** Connection setup uses the typestate pattern to enforce correct ordering and only necessary arguments at compile time.
- **Builder Pattern:** Simplifies creating reliable connections (RC) and unreliable connections (UC).
- **Resource Dependency Management:** Resource types implement reference counting, eliminating verbose lifetime annotations without sacrificing performance.

## License

SafeVerbs is released under the BSD 3 License. See the [LICENSE](#) file for details.