A first look at ROS 2 applications written in asynchronous Rust

Martin Škoudlil, **Michal Sojka**, Zdeněk Hanzálek

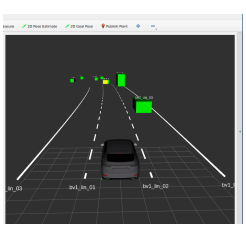
Czech Technical University in Prague, Czech Republic

ECRTS'25, July 9, 2025
Brussels, Belgium

T A
Č R

Motivation – ALKS

Automated Lane-Keeping System



- Advanced Driver Assistance System
- Drives on highways up to 130 km/h
- SAE Level 3 – **driver does not need to pay attention**
- EU regulation for ALKS approval

- University project with limited resources
- **Rust** chosen to focus on developing functionality instead of debugging programming errors

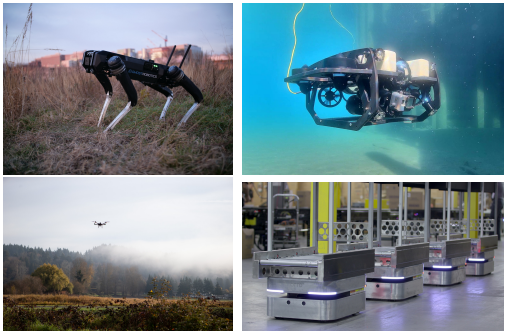
Rust

- Relatively new language (Rust 1.0: 2015)
- Statically typed, compiled, no runtime, expressive type system, (C++ successor?)
- **Memory safety**
- Lisp-like macros (metaprogramming)

- Suitable for **complex and real-time applications**

```rust
fn main() {
    // 1. Ownership & Borrowing
    let s = String::from("hello");  // s owns the String
    print_length(&s);               // passing by reference (borrowing)
    println!("Still usable: {}", s); // s is still valid

    // 2. Pattern Matching with `match`
    let number = Some(7);
    match number {
        Some(n) if n > 5 => println!("Greater than 5: {}", n),
        Some(n) => println!("Number: {}", n),
        None => println!("No number"),
    }

    // 3. Safe Concurrency (using threads and move)
    use std::thread;
    let v = vec![1, 2, 3];
    let handle = thread::spawn(move || {
        println!("Vector from thread: {:?}", v);
    });
    handle.join().unwrap();
}

fn print_length(s: &String) {
    println!("Length: {}", s.len());
}
```

ROS



Source: `https://arxiv.org/abs/2211.07752`

- Software framework for distributed robotic applications
- Open source, over 1500 packages
- ~~ROS~~, ROS 2
- **Linux**, Windows, macOS
- C++, Python

Rust & ROS

- Rust support in ROS is unofficial, supported only by the community.
- Several independent projects...

rclrs

`https://github.com/ros2-rust/ros2_rust`

- Rust client library modeled after its C++ counterpart `rclcpp`
- Accompanied by tools like message code generators

R2R

`https://github.com/sequenceplanner/r2r/`

- Glue between ROS and Rust asynchronous execution frameworks
- Flexible, more features than `rclrs`
- **How to use it for real-time applications?** Documentation and examples give no answer

Detailed comparison of rclcpp, R2R and rclrs

Group	Feature	rclcpp (C++)	R2R	rclrs
Communication	Message generation	✅	✅	✅
	Publishers and subscriptions	✅	✅	✅
	Loaned messages (zero-copy)	✅	✅	✅
	Tunable QoS settings	✅	✅	✅
	Clients and services	✅	✅	✅
	Actions	✅	✅	⚠[1]
	Dynamic type support	✅	✅	✅
Parameters	Parameter handling	✅	✅	✅
	Parameter ranges	✅	⚠	✅
	Parameter locking	none	per-node	per-parameter
	Derived parameters	❌	✅	❌
Time	Timers	✅	✅	❌
	Simulated time	✅	✅	✅
	Tracepoints	✅	☑[3]	❌
Executors	Single-threaded	✅	☑[4]	✅
	Multi-threaded	✅	☑[4]	❌
	Asynchronous programming style	❌	✅	❌
Other	Composable nodes	✅	❌	❌
	Lifecycle nodes	✅	❌	❌

Detailed comparison of rclcpp, R2R and rclrs

Group	Feature	rclcpp (C++)	R2R	rclrs
Communication	Message generation	✅	✅	✅
	Publishers and subscriptions	✅	✅	✅
	Loaned messages (zero-copy)	✅	✅	✅
	Tunable QoS settings	✅	✅	✅
	Clients and services	✅	✅	✅
	Actions	✅	✅	⚠[1]
	Dynamic type support	✅	✅	✅
Parameters	Parameter handling	✅	✅	✅
	Parameter ranges	✅	⚠	✅
	Parameter locking	none	per-node	per-parameter
	Derived parameters	❌	✅	❌
Time	Timers	✅	✅	❌
	Simulated time	✅	✅	✅
	Tracepoints	✅	☑[3]	❌
Executors	Single-threaded	✅	☑[4]	✅
	Multi-threaded	✅	☑[4]	❌
	Asynchronous programming style	❌	✅	❌
Other	Composable nodes	✅	❌	❌
	Lifecycle nodes	✅	❌	❌

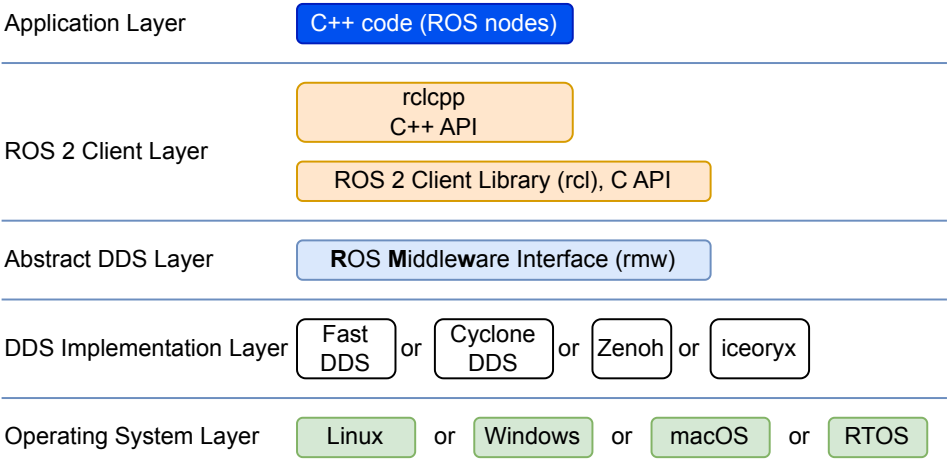
`https://github.com/skoudmar/Ros2TraceAnalyzer`

Goals of this work

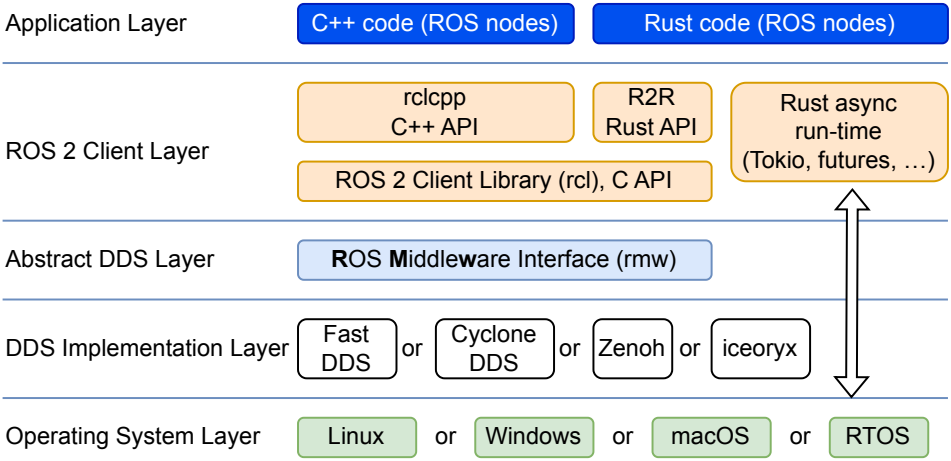
- Find out how to structure **R2R applications** to be suitable for **deterministic real-time operation**
- Investigate the execution model of Rust asynchronous runtimes
- Evaluate real-time properties of R2R on:
  - Simple synthetic application
  - Real-world autonomous driving application

Content

ROS architecture



ROS architecture



Asynchronous Rust

- A form of concurrent programming
- Scheduling decisions in user space, in cooperative manner
- Language support: `async`, `await`
- **Compiler creates multiple schedulable entities from linear code**
- Executors – schedule and execute async tasks

```rust
// Synchronous code
fn read_file_sync(path: &str) -> io::Result<String> {
    let mut file = File::open(path)?;
    let mut contents = String::new();
    file.read_to_string(&mut contents)?;
    Ok(contents)
}

// Asynchronous code
async fn read_file_async(path: &str) -> io::Result<String> {
    let mut file = File::open(path).await?;
    let mut contents = String::new();
    file.read_to_string(&mut contents).await?;
    Ok(contents)
}

fn main() {
  // Execute a single async task
  let res = executor::block_on(read_file_async("file.txt"));
}
```

Asynchronous Rust

- A form of concurrent programming
- Scheduling decisions in user space, in cooperative manner
- Language support: `async`, `await`
- **Compiler creates multiple schedulable entities from linear code**
- Executors – schedule and execute async tasks

```rust
// Synchronous code
fn read_file_sync(path: &str) -> io::Result<String> {
    let mut file = File::open(path)?;
    let mut contents = String::new();
    file.read_to_string(&mut contents)?;
    Ok(contents)
}

// Asynchronous code
async fn read_file_async(path: &str) -> io::Result<String> {
    let mut file = File::open(path).await?;
    let mut contents = String::new();
    file.read_to_string(&mut contents).await?;
    Ok(contents)
}

fn main() {
  // Execute a single async task
  let res = executor::block_on(read_file_async("file.txt"));
}
```

Executors

ROS, C++

- Single-threaded executor
- Multi-threaded executor
- Events executor

Asynchronous Rust

- Executors are provided by async runtimes:
  - Tokio – prefers throughput over time determinism
  - Futures – simpler, suitable for real-time
- Tokio executor
- Futures local executor (single-threaded)
- Futures thread-pool executor (multiple-threads)

Main difference

ROS executors combine

- event sampling and
- callback scheduling & execution.

Rust async executors are responsible only for

- async task (callback) scheduling & execution.

Executors

ROS, C++

- Single-threaded executor
- Multi-threaded executor
- Events executor

Asynchronous Rust

- Executors are provided by async runtimes:
  - Tokio – prefers throughput over time determinism
  - Futures – simpler, suitable for real-time
- Tokio executor
- Futures local executor (single-threaded)
- Futures thread-pool executor (multiple-threads)

Main difference

ROS executors combine

- event sampling and
- callback scheduling & execution.

Rust async executors are responsible only for

- async task (callback) scheduling & execution.

Executors

ROS, C++

- Single-threaded executor
- Multi-threaded executor
- Events executor

Asynchronous Rust

- Executors are provided by async runtimes:
  - Tokio – prefers throughput over time determinism
  - Futures – simpler, suitable for real-time
- Tokio executor
- Futures local executor (single-threaded)
- Futures thread-pool executor (multiple-threads)

Main difference

ROS executors combine

- event sampling and
- callback scheduling & execution.

Rust async executors are responsible only for

- async task (callback) scheduling & execution.

Content

Comparison of single-threaded ROS C++ executor and Rust R2R

C++

```cpp
auto node = make_shared<rclcpp::Node>();
node->create_subscription<UInt64>("/A", 10, cb_A);
node->create_subscription<UInt64>("/B", 10, cb_B);
rclcpp::spin(node);
```

R2R equivalent

```rust
let ctx = r2r::Context::create()?;
let mut node = r2r::Node::create(ctx, "example", "")?;

let stream_A = node.subscribe("/A", Qos::default())?;
spawner.spawn_local(stream_A.for_each(cb_A))?;
let stream_B = node.subscribe("/B", Qos::default())?;
spawner.spawn_local(stream_B.for_each(cb_B))?;

local_executor.run_until_stalled();       // init.
loop {
    node.spin_once(Duration::seconds(1)); // sampling
    local_executor.run_until_stalled();   // execution
}
```

Response-Time analysis:

[1] T. Blaß, D. Casini, S. Bozhko, and B. B. Brandenburg, "A ROS 2 Response-Time Analysis Exploiting Starvation Freedom and Execution-Time Variance," in 2021 IEEE Real-Time Systems Symposium (RTSS).

Comparison of single-threaded ROS C++ executor and Rust R2R

C++

```cpp
auto node = make_shared<rclcpp::Node>();
node->create_subscription<UInt64>("/A", 10, cb_A);
node->create_subscription<UInt64>("/B", 10, cb_B);
rclcpp::spin(node);
```

R2R equivalent

```rust
let ctx = r2r::Context::create()?;
let mut node = r2r::Node::create(ctx, "example", "")?;

let stream_A = node.subscribe("/A", Qos::default())?;
spawner.spawn_local(stream_A.for_each(cb_A))?;
let stream_B = node.subscribe("/B", Qos::default())?;
spawner.spawn_local(stream_B.for_each(cb_B))?;

local_executor.run_until_stalled();       // init.
loop {
    node.spin_once(Duration::seconds(1)); // sampling
    local_executor.run_until_stalled();   // execution
}
```

Response-Time analysis:

[1] T. Blaß, D. Casini, S. Bozhko, and B. B. Brandenburg, "A ROS 2 Response-Time Analysis Exploiting Starvation Freedom and Execution-Time Variance," in 2021 IEEE Real-Time Systems Symposium (RTSS).

Comparison of single-threaded ROS C++ executor and Rust R2R

C++

```cpp
auto node = make_shared<rclcpp::Node>();
node->create_subscription<UInt64>("/A", 10, cb_A);
node->create_subscription<UInt64>("/B", 10, cb_B);
rclcpp::spin(node);
```

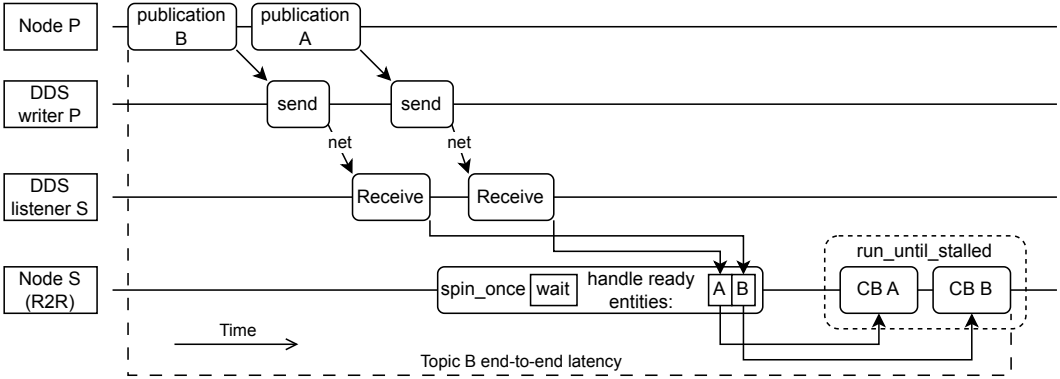
R2R equivalent

```rust
let ctx = r2r::Context::create()?;
let mut node = r2r::Node::create(ctx, "example", "")?;

let stream_A = node.subscribe("/A", Qos::default())?;
spawner.spawn_local(stream_A.for_each(cb_A))?;
let stream_B = node.subscribe("/B", Qos::default())?;
spawner.spawn_local(stream_B.for_each(cb_B))?;

local_executor.run_until_stalled();       // init.
loop {
    node.spin_once(Duration::seconds(1)); // sampling
    local_executor.run_until_stalled();   // execution
}
```

Response-Time analysis:

[1] T. Blaß, D. Casini, S. Bozhko, and B. B. Brandenburg, “A ROS 2 Response-Time Analysis Exploiting Starvation Freedom and Execution-Time Variance,” in 2021 IEEE Real-Time Systems Symposium (RTSS).

Example of execution chain from publisher *P* to Rust R2R subscriber *S*



The paper contains details and examples about R2R sampling and Rust async executor scheduling.

ROS C++ multi-threaded executor, callback groups

- Callback groups allow restricting callback execution (mutual exclusion, etc.)
- It is possible to emulate multi-threaded executor and callback groups in Rust R2R.
- Details in the paper.

- ROS C++ multi-threaded executor has various problems

In R2R one can do better...

ROS C++ multi-threaded executor, callback groups

- Callback groups allow restricting callback execution (mutual exclusion, etc.)
- It is possible to emulate multi-threaded executor and callback groups in Rust R2R.
- Details in the paper.

- ROS C++ multi-threaded executor has various problems

In R2R one can do better...

Proposed R2R application structure

```rust
fn main() -> Result<(), Box<dyn Error>> {
    set_thread_priority_and_policy(
        thread_native_id(),
        ThreadPriority::try_from(MAIN_PRIORITY)?,
        RealTime(Fifo),
    )?;

    let ctx = r2r::Context::create()?;
    let mut node = r2r::Node::create(ctx, "example", "")?;

    let subs = node.subscribe("/topic", Qos::default())?;
    let future = subs.for_each(move |msg: Msg| async move {
        // process msg
    });
    spawn_in_thread(future, CALLBACK_PRIORITY);

    loop {
        node.spin_once(SPIN_TIMEOUT); // ROS sampling
    }
}
```

1. Run the main thread with the highest priority
   - Inherited by DDS threads spawned from `r2r::Node::create()`
   - Used for ROS sampling (`node.spin_once()`), i.e. dispatching events from ROS to Rust asynchronous executors
2. Run Rust async executor(s) executing callbacks in lower priority thread(s)
   - 1 thread = 1 executor
   - 1 executor = 0..n callbacks

Proposed R2R application structure

```rust
fn main() -> Result<(), Box<dyn Error>> {
    set_thread_priority_and_policy(
        thread_native_id(),
        ThreadPriority::try_from(MAIN_PRIORITY)?,
        RealTime(Fifo),
    )?;

    let ctx = r2r::Context::create()?;
    let mut node = r2r::Node::create(ctx, "example", "")?;

    let subs = node.subscribe("/topic", Qos::default())?;
    let future = subs.for_each(move |msg: Msg| async move {
        // process msg
    });
    spawn_in_thread(future, CALLBACK_PRIORITY);

    loop {
        node.spin_once(SPIN_TIMEOUT); // ROS sampling
    }
}
```

1. Run the main thread with the highest priority
   - Inherited by DDS threads spawned from `r2r::Node::create()`
   - Used for ROS sampling (`node.spin_once()`), i.e. dispatching events from ROS to Rust asynchronous executors
2. Run Rust async executor(s) executing callbacks in lower priority thread(s)
   - 1 thread = 1 executor
   - 1 executor = 0..n callbacks

Proposed R2R application structure

```rust
fn main() -> Result<(), Box<dyn Error>> {
    set_thread_priority_and_policy(
        thread_native_id(),
        ThreadPriority::try_from(MAIN_PRIORITY)?,
        RealTime(Fifo),
    )?;

    let ctx = r2r::Context::create()?;
    let mut node = r2r::Node::create(ctx, "example", "")?;

    let subs = node.subscribe("/topic", Qos::default())?;
    let future = subs.for_each(move |msg: Msg| async move {
        // process msg
    });
    spawn_in_thread(future, CALLBACK_PRIORITY);

    loop {
        node.spin_once(SPIN_TIMEOUT); // ROS sampling
    }
}
```

1. Run the main thread with the highest priority
   - Inherited by DDS threads spawned from `r2r::Node::create()`
   - Used for ROS sampling (`node.spin_once()`), i.e. dispatching events from ROS to Rust asynchronous executors
2. Run Rust async executor(s) executing callbacks in lower priority thread(s)
   - 1 thread = 1 executor
   - 1 executor = 0..n callbacks

Proposed R2R application structure

```rust
fn main() -> Result<(), Box<dyn Error>> {
    set_thread_priority_and_policy(
        thread_native_id(),
        ThreadPriority::try_from(MAIN_PRIORITY)?,
        RealTime(Fifo),
    )?;

    let ctx = r2r::Context::create()?;
    let mut node = r2r::Node::create(ctx, "example", "")?;

    let subs = node.subscribe("/topic", Qos::default())?;
    let future = subs.for_each(move |msg: Msg| async move {
        // process msg
    });
    spawn_in_thread(future, CALLBACK_PRIORITY);

    loop {
        node.spin_once(SPIN_TIMEOUT); // ROS sampling
    }
}
```

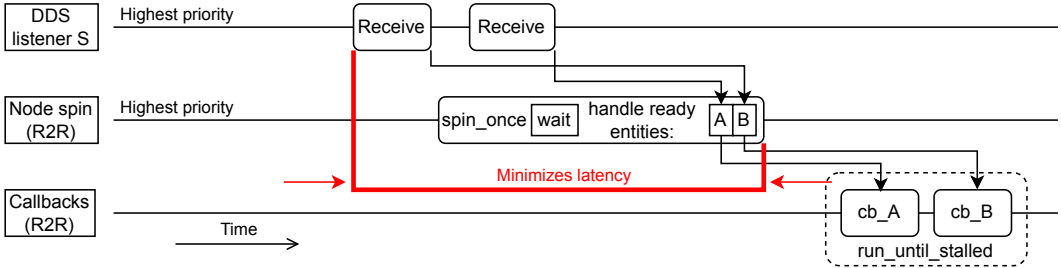
1. Run the main thread with the highest priority
   - Inherited by DDS threads spawned from `r2r::Node::create()`
   - Used for ROS sampling (`node.spin_once()`), i.e. dispatching events from ROS to Rust asynchronous executors
2. Run Rust async executor(s) executing callbacks in lower priority thread(s)
   - 1 thread = 1 executor
   - 1 executor = 0..n callbacks

Proposed R2R application structure

```rust
fn main() -> Result<(), Box<dyn Error>> {
    set_thread_priority_and_policy(
        thread_native_id(),
        ThreadPriority::try_from(MAIN_PRIORITY)?,
        RealTime(Fifo),
    )?;

    let ctx = r2r::Context::create()?;
    let mut node = r2r::Node::create(ctx, "example", "")?;

    let subs = node.subscribe("/topic", Qos::default())?;
    let future = subs.for_each(move |msg: Msg| async move {
        // process msg
    });
    spawn_in_thread(future, CALLBACK_PRIORITY);

    loop {
        node.spin_once(SPIN_TIMEOUT); // ROS sampling
    }
}
```

1. Run the main thread with the highest priority
   - Inherited by DDS threads spawned from `r2r::Node::create()`
   - Used for ROS sampling (`node.spin_once()`), i.e. dispatching events from ROS to Rust asynchronous executors
2. Run Rust async executor(s) executing callbacks in lower priority thread(s)
   - 1 thread = 1 executor
   - 1 executor = 0..n callbacks

The effect of running DDS & sampling with the highest priority

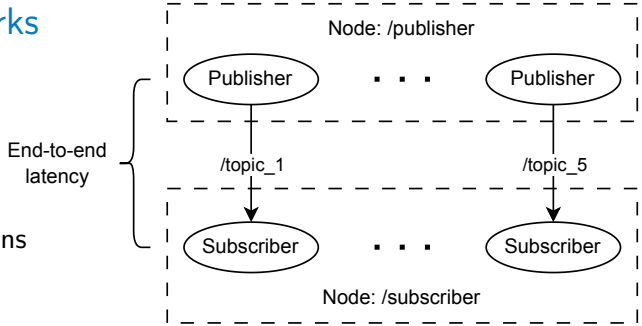


- Scheduling determined only by policies of Rust async executor and OS scheduler
- DDS & ROS sampling introduces just execution time overhead
- Similar effect as ROS events executor
- Simplest case: 1 callback per executor $\Rightarrow$ only OS scheduler policy is relevant

Content

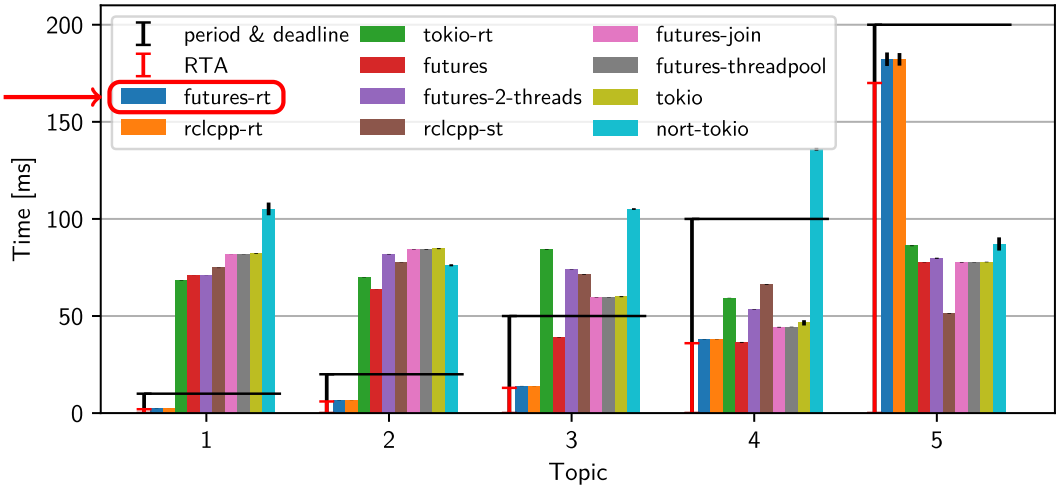
Evaluation on synthetic benchmarks

- Five topics
- C++ publisher
- Multiple subscriber implementations
- Linux, LTTng trace processing

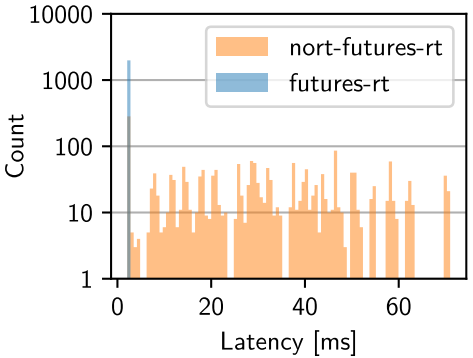


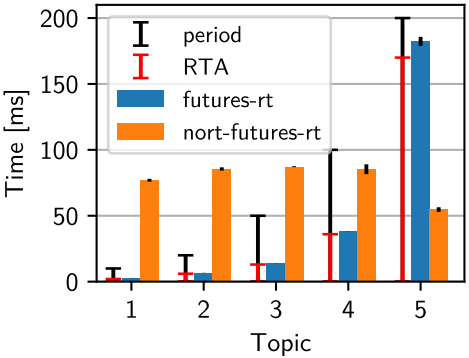
Topic #	1	2	3	4	5
Publisher period [ms]	10	20	50	100	200
Subscription callback execution time [ms]	2	4	5	15	50

End-to-end latency – comparison with response-time analysis



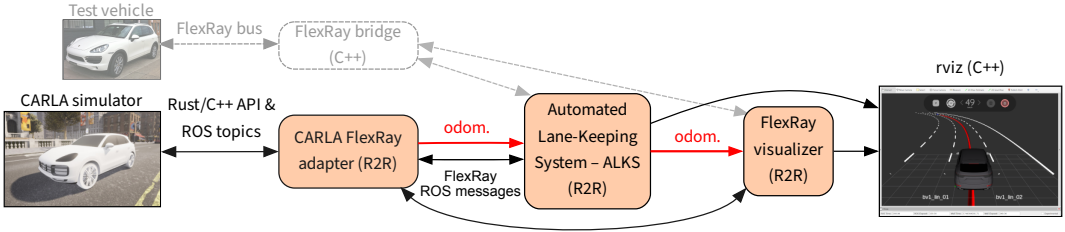
Failing to set high priority of DDS threads





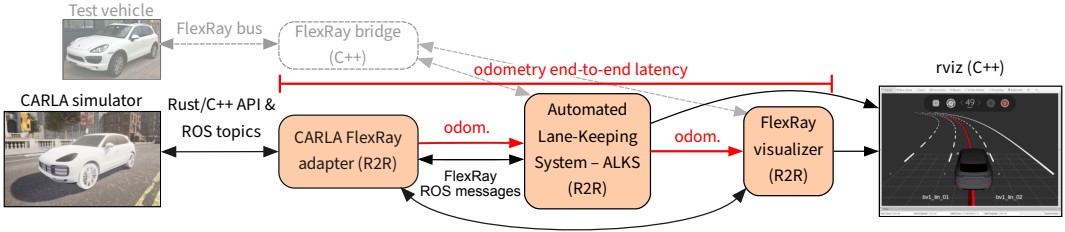
Complex autonomous driving case study

Simulation of Automated Lane-Keeping System (ALKS)



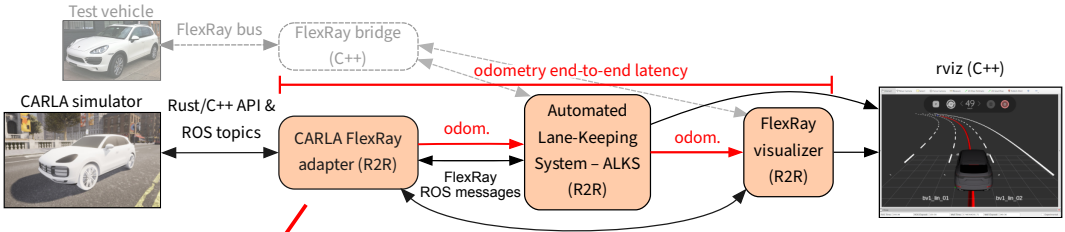
Complex autonomous driving case study

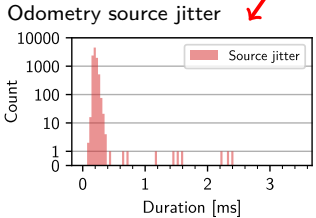
Simulation of Automated Lane-Keeping System (ALKS)



Complex autonomous driving case study

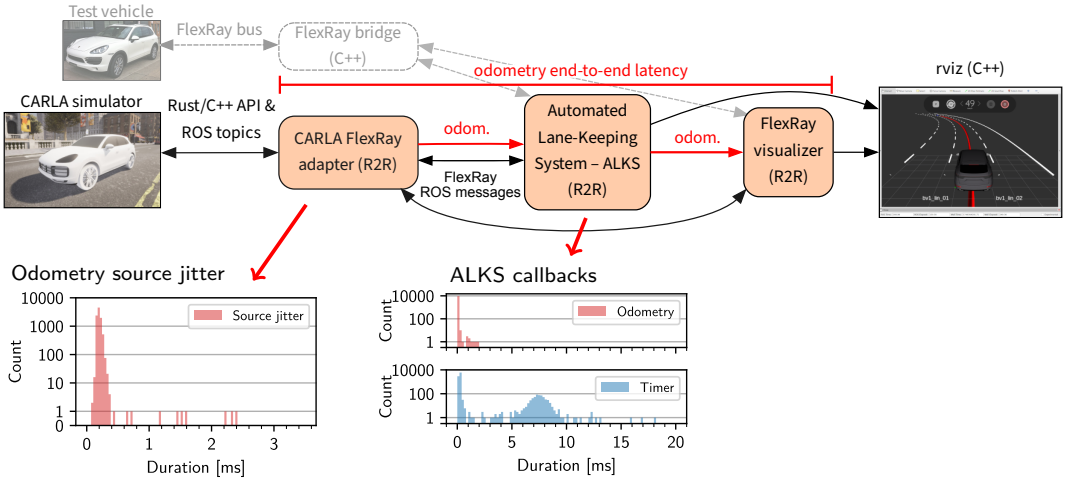
Simulation of Automated Lane-Keeping System (ALKS)





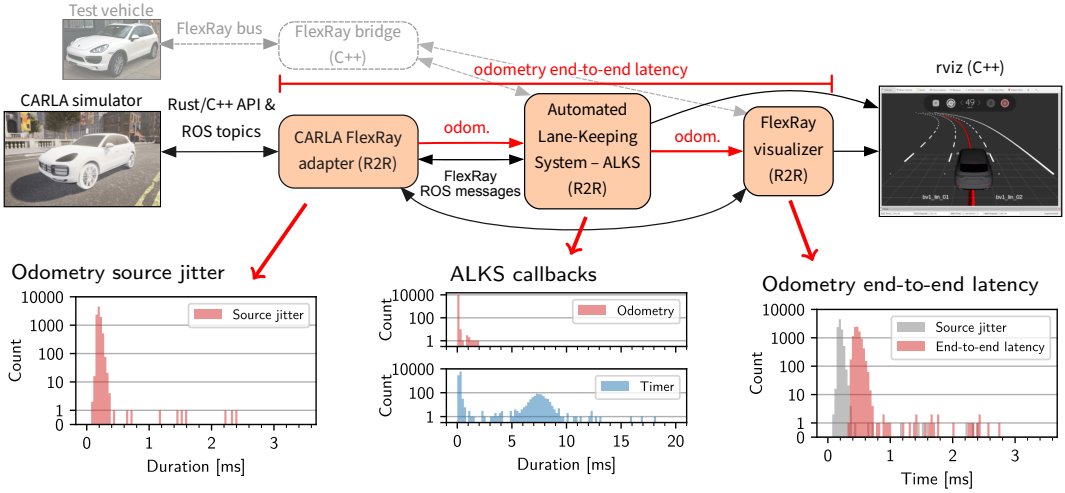
Complex autonomous driving case study

Simulation of Automated Lane-Keeping System (ALKS)

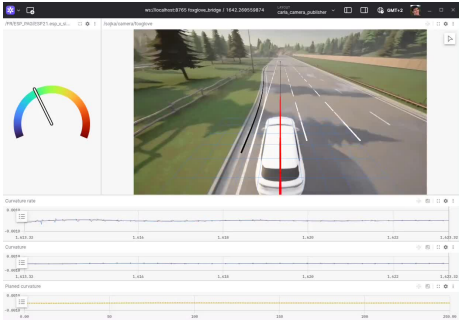


Complex autonomous driving case study

Simulation of Automated Lane-Keeping System (ALKS)



Videos



Conclusion

- We analyzed scheduling policies of Rust asynchronous runtimes
- Proposed structure of Rust R2R applications that can provide deterministic timing, comparable to C++ ROS applications
- Examples and benchmarks available →
- Evaluated on two case studies

Questions?

Conclusion

- We analyzed scheduling policies of Rust asynchronous runtimes
- Proposed structure of Rust R2R applications that can provide deterministic timing, comparable to C++ ROS applications
- Examples and benchmarks available →
- Evaluated on two case studies

Questions?