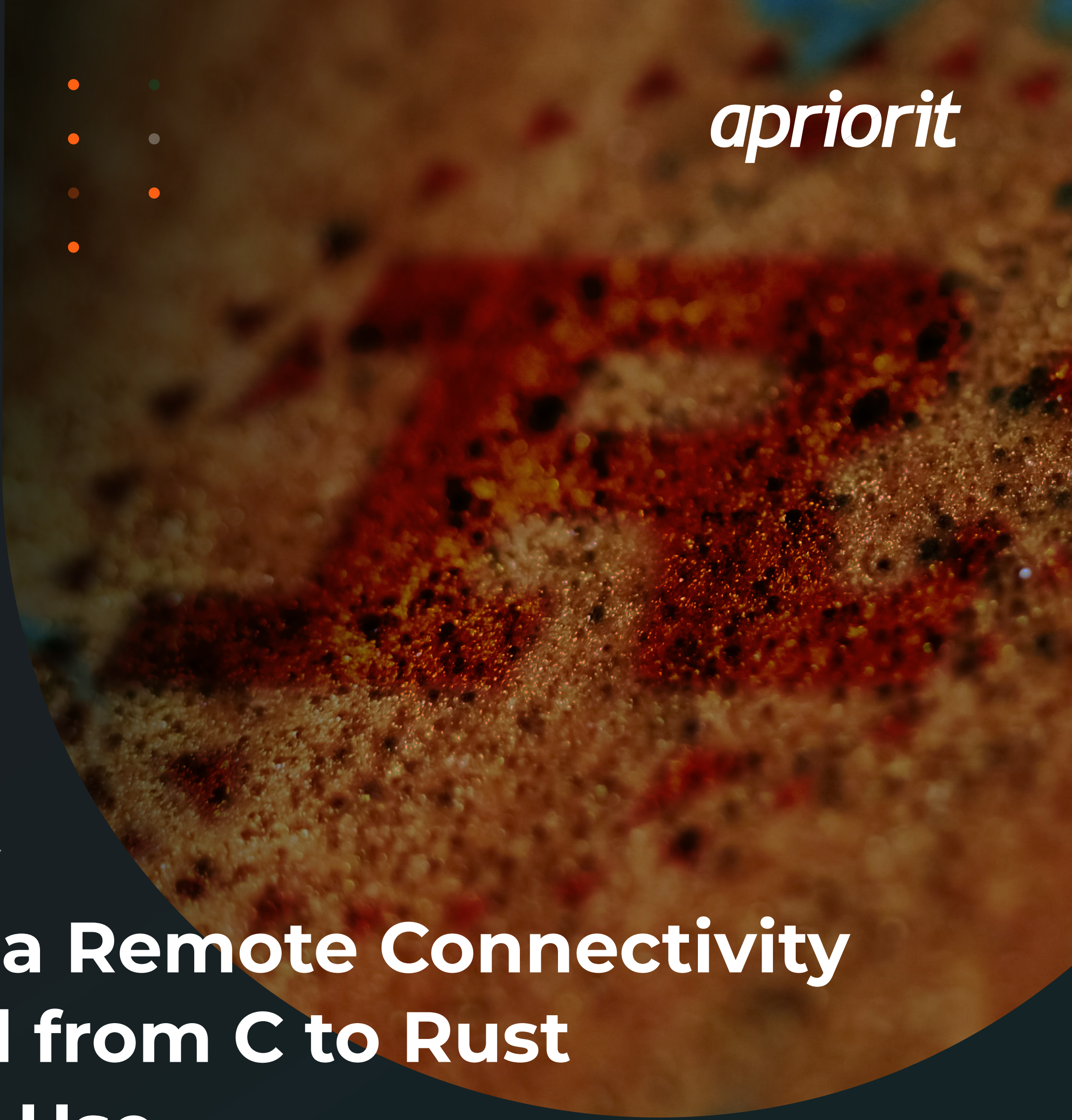


apriorit

CASE STUDY

Porting a Remote Connectivity Protocol from C to Rust for Web Use

SOFTWARE DEVELOPMENT

Background

As part of our long-term collaboration with Devolutions, Apriorit was tasked with modernizing our client's VNC protocol by porting it from C to Rust. The goal was to enable WebAssembly (WASM) support for browser-based remote access.

Additionally, Devolutions asked us to port an MVS codec, which is a part of the Apple Remote Desktop (ARD) VNC protocol extensions. Originally developed by Apple, this codec was reverse-engineered by Devolutions and used to build their own custom codec that was adapted for Windows and Linux. Our task was to port it to Rust as well, as a part of their IronVNC project.

The client

Devolutions is a Canadian cybersecurity company that provides industry-leading solutions for credentials and remote desktop access management. Their products are used in 140+ countries by over 1 million cybersecurity specialists.

Client:
Devolutions

Location:
Quebec, Canada

Industry:
Cybersecurity

Collaboration with Apriorit:
Long-term and ongoing

Services we provided:
- Rust development
- Quality assurance
- Project management

The challenge

Devolutions approached Apriorit with a request to port their IronVNC protocol from C to Rust and add more features from the third-party Virtual Network Computing (VNC) protocol. The aim was to achieve a more secure, reliable, and feature-rich solution.

Devolutions defined these specific project goals for the Apriorit team:

- Build a new implementation of the IronVNC protocol in Rust that is compatible with WASM and can be integrated into web applications
- Port the MVS codec from C to Rust while maintaining its expanded support for Windows and Linux
- Enable support for multiple VNC extensions including RFB, TightVNC, UltraVNC, and Apple Remote Desktop (ARD)
- Ensure I/O independence so the protocol can support WASM and native platforms (macOS, Linux, Windows)
- Improve performance, maintainability, and stability over the original C implementation
- Add new features from Apple's VNC (ARD) to improve the user experience, such as TightVNC file transfer and extended clipboard

The rewritten protocol would become part of a new Devolutions product called Devolutions Gateway — a web-based tool for remote desktop access across RDP, VNC, and other protocols.

Project details

Apriorit team

Project manager · Rust developer · QA

Tech stack

LANGUAGES

C · Rust

TESTING TOOLS

Proptests · rtest · libFuzzer · Arbitrary

BENCHMARKING TOOLS

Flamegraph · Callgrind · Hyperfine

TARGET PLATFORMS

Windows · Linux · Browser (via WASM) · macOS

DELIVERED SOLUTION

A WASM-compatible VNC protocol in Rust

Need to modernize your software?

Our team of 400+ developers has the skills and experience necessary to improve your product without disrupting its operations.

CONTACT US →

The result

> We delivered a new implementation of our client's VNC in Rust, optimized for performance, and made it compatible with both native platforms and browsers via WASM.
>
> Our implementation offers unmatched compatibility with macOS devices due to its support for ARD extensions, including the MVS codec. IronVNC is now part of Devolutions Gateway, enabling fast, browser-based VNC and ARD connectivity for Devolutions users.

Our solution

Our work focused on rewriting and extending the VNC protocol implementation while ensuring compatibility with web environments via WASM.

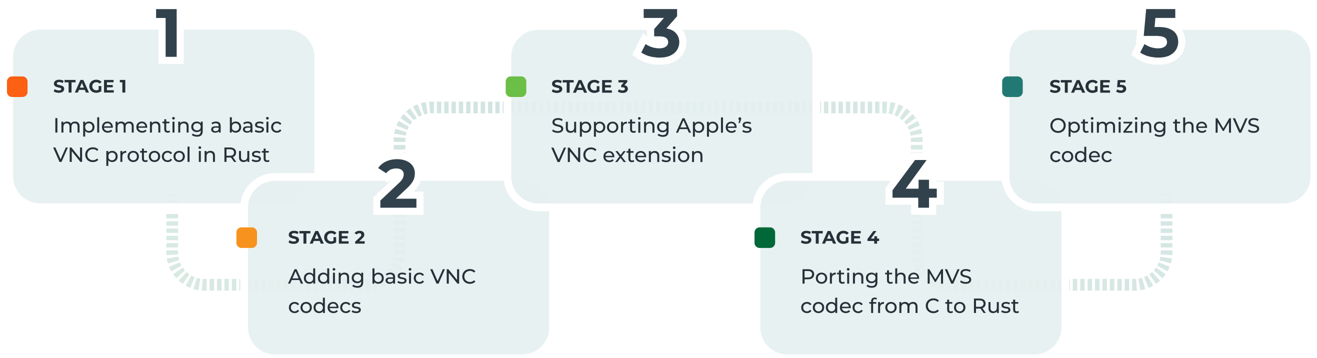
We used Rust as the core language for its safety and performance advantages and relied on automated testing, property-based testing, and fuzzing to validate correctness. To optimize performance, we leveraged tools including flamegraph, callgrind, and hyperfine.

How we did it

To meet the client's needs, Apriorit assembled a team consisting of a Rust developer, a QA engineer, and a project manager.

After discussing goals and priorities with Devolutions, we proposed a structured development approach broken into four stages.

5 stages of implementing IronVNC in Rust


1
STAGE 1
Implementing a basic VNC protocol in Rust
2
STAGE 2
Adding basic VNC codecs
3
STAGE 3
Supporting Apple's VNC extension
4
STAGE 4
Porting the MVS codec from C to Rust
5
STAGE 5
Optimizing the MVS codec

1. Implementing the basic VNC protocol

We started by analyzing the RFC6143 standard and existing VNC specifications. Devolutions had already prepared a foundational architecture suitable for WASM, so we reviewed it and integrated it into our workflow.

Our first step was to implement the core of the VNC protocol, enabling basic connections to remote systems. At this stage, the application could establish a session but didn't yet support screen updates or user interaction.

Because our IronVNC implementation had to support both WebAssembly and native platforms (Windows, Linux, macOS), we had to account for several restrictions during architecture and protocol development:

- **Avoid direct I/O operations**. Since browsers don't allow TCP connections, we implemented most modules using a sans-I/O approach. Instead of working with a transport layer directly, modules receive raw byte streams as input.
- **Use dependency injection for runtime data.** To ensure platform independence, we avoided system calls (e.g., `gethostname`) in core modules. Instead, any necessary runtime information is injected from outside, allowing the core logic to remain portable.
- **Ensure `no_std` compatibility wherever possible.** The Rust standard library depends on system APIs, which are unavailable in browsers. To support WebAssembly, we limited core modules to use only Rust's `core` library, which avoids system calls and improves portability across platforms.

2. Adding basic VNC codecs

Next, we implemented essential image codecs (Raw and ZLIB), which enabled screen rendering and user interaction. With these codecs in place, we could successfully view and interact with remote desktops.

// 3. Supporting Apple's VNC extension

To enable full compatibility with macOS machines, we implemented Apple's VNC extension. To achieve this, our team had to:

- Implement authentication similar to Apple's current functionality
- Enable support for macOS screen configurations
- Implement keyboard and cursor handling

This functionality allowed Devolutions users to connect to and interact with macOS devices, though performance was initially limited due to the use of basic codecs.

// 4. Porting the MVS codec from C to Rust

To improve performance on macOS, we ported the MVS codec — a proprietary and fast image codec that the client's in-house team had reverse engineered and previously implemented in C.

This stage was particularly challenging due to the codec's complexity.

To verify correctness, we wrote extensive unit tests using real session payloads. We also benchmarked the code to measure execution time. In doing this, we discovered that while the initial Rust implementation worked functionally, it was slower than expected.

// 5. Optimizing the MVS codec

After benchmarking and profiling the new Rust-based MVS codec, we identified several bottlenecks. Our team optimized the code by:

- Reducing memory allocations
- Introducing SIMD for pixel processing
- Improving caching mechanisms

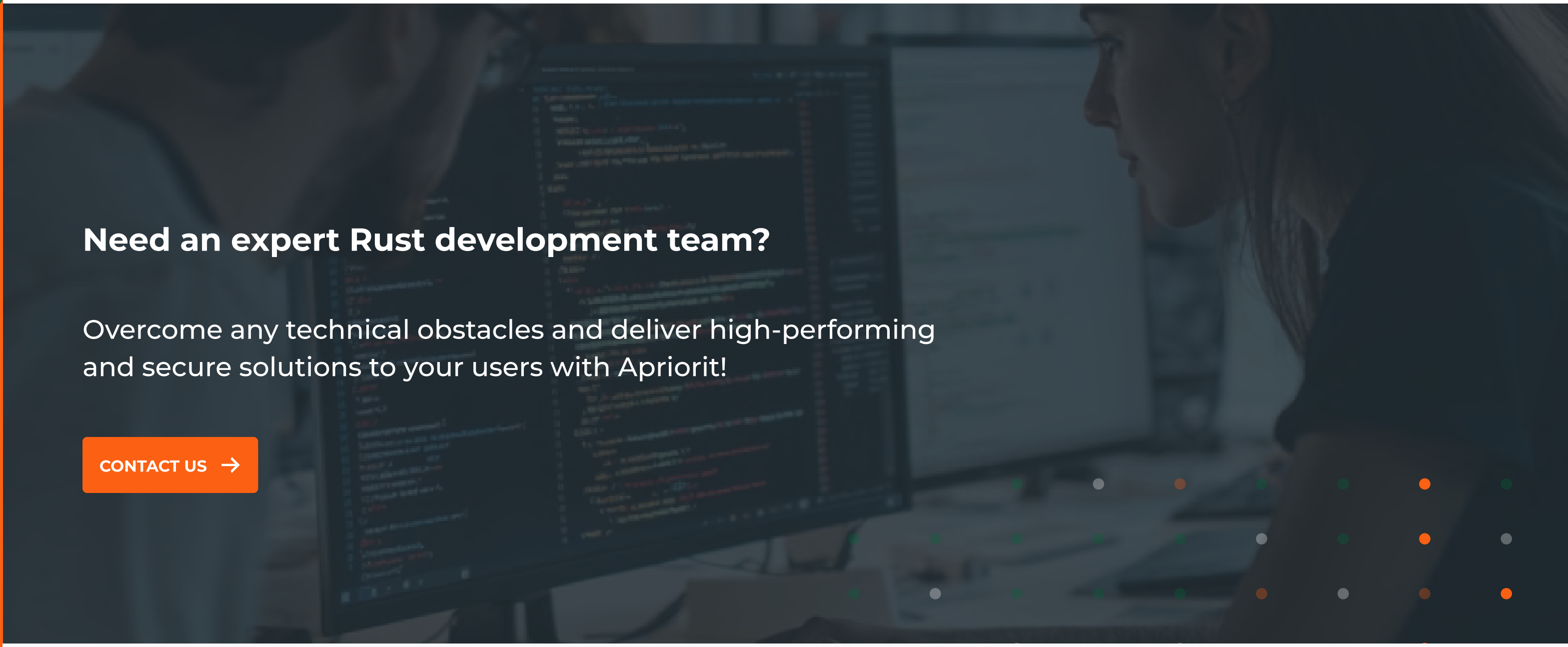
These changes resulted in a **4.5x performance improvement** compared to the initial version of IronVNC.

Impact

By rewriting the VNC protocol in Rust and making it compatible with WebAssembly, we helped our client modernize a legacy connectivity component and enable secure remote access directly in the browser. This upgrade allowed Devolutions to deliver a safer and more flexible cross-platform remote management solution.

Thanks to our optimization strategy, the new implementation was faster than the original, significantly improving performance. The client can now reuse the new VNC protocol across environments, reduce memory-related risks, and support web-first use cases more effectively.

Our Rust-based upgrade enabled Devolutions to offer fast, secure, and platform-independent remote access — including seamless browser support — as part of their modern remote management solution.



Need an expert Rust development team?

Overcome any technical obstacles and deliver high-performing and secure solutions to your users with Apriorit!

CONTACT US →