A Technical Analysis and C++ Port of the *u64-remote* Client

Preserving the Original Go Implementation by Meatball

Original Author (Go implementation): Levi Spencer
GitHub: https://github.com/AllMeatball/u64-remote

C++ Port and Technical Analysis: Dr. Eric O. Flores

Target Device: Commodore 64 Ultimate / Ultimate-64
Original Language: Go
Ported Language: C++17 (Linux)
Transport: HTTP REST API

Purpose and Scope

This paper documents the **analysis and C++ port** of the *u64-remote* project originally authored in **Go** by **Meatball** and published on GitHub:

https://github.com/AllMeatball/u64-remote

The intent of this work is **not** to replace, supersede, or reinterpret the original implementation, but rather to:

1. Preserve the original author's design and intent
2. Port the functionality to a native **Linux C++** environment
3. Improve interoperability with existing C/C++ tooling
4. Add optional enhancements (e.g., device discovery) while keeping the original behavior intact

All architectural decisions in the C++ version trace directly back to the original Go code.

Overview of the Original *u64-remote* Go Project

Project Intent (Original Author)

The *u64-remote* project provides a **command-line client** that allows a modern computer to remotely control a **Commodore 64 Ultimate** device over a network connection.

Primary capabilities include:

- Uploading and executing `.PRG` files
- Reading and writing C64 memory
- Authenticating via the Ultimate's REST interface

The project is intentionally minimal, direct, and automation-friendly.

Original Go Implementation (Authored by Meatball)

Original Source Attribution

The following Go code is taken directly from the original repository and is included **verbatim** to preserve authorship and design intent.

Repository: https://github.com/AllMeatball/u64-remote

Original Go Code (Server Interface)

```
package server

import (
	"bytes"
	"fmt"
	"io"
	"net/http"
	"net/url"
)

type U64Creds struct {
	Address, Password string
	EnableMessageBox bool
}

type U64Server struct {
	creds U64Creds
}

func NewU64Server(creds U64Creds) (U64Server, error) {
	server := U64Server{creds: creds}

	// Test server connection
	_, err := server.RestCall("GET", "/v1/version", nil, nil)
	if err != nil {
		return server, err
	}
	return server, nil
}

func (self *U64Server) CreateRestRequest(method, path string, body io.Reader)
(*http.Request, error) {
	fullPath, err := url.JoinPath(self.creds.Address, path)
	if err != nil {
		return nil, err
	}

	req, err := http.NewRequest(method, fullPath, body)
	if err != nil {
		return nil, err
	}

	req.Header.Add("X-Password", self.creds.Password)
	return req, nil
}

func (self *U64Server) RunPRG(reader io.Reader) error {
	req, err := self.CreateRestRequest("POST", "/v1/runners:run_prg",
reader)
	if err != nil {
		return err
	}

	req.Header.Set("Content-Type", "application/octet-stream")
	_, err = self.RestCallRaw(req)
	return err
}
```

What the Original Go Code Does

In simple terms:

1. **Loads credentials** (IP address + password)
2. **Verifies connectivity** using:

   ```
   GET /v1/version
   ```

3. **Uploads and executes a PRG file** using:

   ```
   POST /v1/runners:run_prg
   ```

4. Uses the HTTP header:

   ```
   X-Password: <password>
   ```

5. Treats all program data as **raw binary streams**

The Go implementation is intentionally clean and avoids unnecessary abstraction.

Why a C++ Port Was Created

The C++ port exists for **environmental and integration reasons**, not because of deficiencies in the Go implementation.

Primary motivations:

- Native integration with existing **C/C++ toolchains**
- Elimination of Go runtime dependency
- Easier embedding into larger native systems
- Better alignment with Linux-based retrocomputing toolchains
- Ability to extend functionality (e.g., discovery) without modifying the original Go code

Importantly:

> **The C++ version preserves the original design model of the Go implementation.**

C++ Architecture Overview (Ported Design)

5.1 Conceptual Equivalence

Go Component	C++ Equivalent
`U64Server`	`class U64Server`
`U64Creds`	`struct Creds`

Go Component	C++ Equivalent
`RestCall()`	`request()`
`RunPRG()`	`runPRG()`

Every C++ function maps directly to an original Go responsibility.

New Capabilities (Additive, Optional)

The C++ port introduces **optional enhancements** that do not alter original behavior:

- **Automatic network discovery** of C64U devices
- User confirmation when multiple devices are found
- More flexible credential loading paths
- Improved error diagnostics for Linux

These enhancements are layered *on top* of the original model.

Go vs C++: Technical Comparison

Aspect	Go (Original)	C++ (Port)
Author	Meatball	Dr. Eric O. Flores (port)
Networking	`net/http`	`libcurl`
JSON	`encoding/json`	Lightweight custom parser
Runtime	Go runtime	Native ELF
Device Discovery	❌ No	✅ Yes
Philosophy	Minimal	Minimal + extensible

Preservation of Original Authorship

This port explicitly:

- Retains original endpoint semantics
- Retains original authentication method
- Retains original data flow (binary PRG upload)
- Retains original control assumptions

The **Go code by Meatball remains the canonical reference implementation**.

The C++ project should always cite:

> https://github.com/AllMeatball/u64-remote

as the **originating source**.

Conclusion

The C*64U-remote* project is an elegant and effective solution for remote control of the Commodore 64 Ultimate.

This C++ port exists to **extend the reach** of that work into native Linux environments while respecting and preserving the original author’s design, intent, and ownership.

In spirit and execution:

> **This is a faithful port — not a rewrite.**

References

1. Levi Spencer, AllMeatball, *u64-remote* (Go implementation)
   https://github.com/AllMeatball/u64-remote
2. Commodore 64 Ultimate REST API documentation (device firmware)