Remote Instrumentation and Data Acquisition: An Internship Research Report

Jaymil Parikh | FERMILAB-PUB-25-0512-STUDENT

Abstract

This report outlines the development and implementation of a remote data acquisition system for waveform analysis using a Rohde & Schwarz oscilloscope. The project involved capturing waveform data, and transferring it to a local machine for visualization and analysis. The core logic was developed in C++ with a focus on object oriented programming and the use of polymorphism so the main application can interact with any instrument without knowing its exact type, simplifying the overall logic and making it easier to add or swap out components without changing the rest of the codebase.. The system issues Standard Commands for Programmable Instruments (SCPI) via a socket connection and parses the oscilloscope's ASCII waveform data. The C++ application was containerized using Docker for ease of portability, and reproducibility. Emphasis was placed on secure networking practices, error handling, and effective data capture. The report describes the technical steps taken, challenges encountered, and lessons learned, providing insight into the practical integration of hardware interfacing with remote computational environments.

I. Introduction

The ability to control laboratory instruments remotely has become increasingly important in research environments, especially those constrained by space, time, or access limitations. Rohde and Schwarz instruments support SCPI over TCP/IP, enabling remote configuration and data retrieval. This internship project simplifies that process by developing a command-line C++ application that connects with the scope, sends SCPI commands, waits for the acquisitions to complete, and retrieves waveform data for analysis.

This project addressed the need to quickly collect data and characterize a system using a high speed wideband tool that every department already has. The solution involved combining Linux-based shell scripting, SSH tunneling, and VNC protocol access with waveform configuration scripting for automated data retrieval. The system supports real-time waveform viewing and data analysis while minimizing direct access to sensitive equipment.

II. Objectives

The primary goals of the project were as follows:

1. To configure and remotely access a Rohde & Schwarz oscilloscope using SCPI commands through a secure SSH tunnel.
2. Use SCPI commands for configuration, triggering, and data retrieval.

3. To develop scripts that automate waveform acquisition and formatting.
4. To implement robust error handling and test the stability of the remote access configuration.
5. To document the workflow and produce a reproducible system for future users.

III. Materials and Methods

A. Hardware Used

- **Oscilloscope:** Rohde & Schwarz RTO series (Bandwidth - 2GHz)
- **Remote Workstation:** Ubuntu Linux (20.04) running on m2epi1 (lab machine)
- **Client Machine:** macOS (local computer)
- **Network:** Secure LAN with VPN and port forwarding via SSH
- **Interfaces:** VNC (RealVNC), SSH (OpenSSH), local terminal

B. Software Tools

- **RealVNC Viewer**
- **Python 3.8** (for data parsing)
- **Bash scripting** for configuration automation
- **Custom config file** (INI format) used for waveform setup

C. Oscilloscope Configuration

A configuration file (config.ini) was written to define waveform parameters:

```
# config.ini - configuration for oscilloscope data acquisition

[network]
; ip = 192.168.1.100
ip = 192.168.1.7
port = 5025

[waveform]
channel1_enable = on
channel1_range = 5
channel1_position = 0
channel1_coupling = DC

channel2_enable = off
channel2_range = 5
channel2_position = 0
channel2_coupling = ACLimit
```

```
channel3_enable = on
channel3_range = 1
channel3_position = -2
channel3_coupling = DC

channel4_enable = off
channel4_range = 2
channel4_position = 1
channel4_coupling = DC

channel = 2
format = ASCii
mode = NORM
point_mode = RAW
point_auto = RECL
output_file = waveform.bin
```

This config.ini file is used to configure a remote oscilloscope for data acquisition through SCPI commands. The configuration sets up both network communication and waveform acquisition parameters, controlling how the oscilloscope behaves during the capture process. Here's a breakdown of what each section and setting does:

[network]

These settings define how the program connects to the oscilloscope:

- `ip = 192.168.1.7`:
  The oscilloscope is located at IP address `192.168.1.7`. The commented line above it (`192.168.1.100`) suggests an alternate or default setting that was previously used.

- `port = 5025`:
  This is the port for SCPI (Standard Commands for Programmable Instruments) communication over TCP/IP. The program will open a socket connection to this address and port.

[waveform]

These settings control the **channel-specific** and **global waveform acquisition parameters**:

Individual Channel Settings

There are four oscilloscope input channels; each can be individually enabled and configured:

- **Channel 1:**
  - `channel1_enable = on`: Channel 1 is active.
  - `channel1_range = 5`: Vertical range is 5V.
  - `channel1_position = 0`: The vertical offset is 0V.

    - `channel1_coupling = DC`: DC coupling allows both AC and DC signals through.

- **Channel 2:**
    - `channel2_enable = off`: Channel 2 is disabled (even though later settings refer to "channel = 2").
    - `channel2_range = 5`
    - `channel2_position = 0`
    - `channel2_coupling = ACLimit`: AC limited coupling blocks DC and may apply additional filtering to protect the device.
- **Channel 3:**
    - `channel3_enable = on`: Channel 3 is active.
    - `channel3_range = 1`: Narrow vertical range for higher resolution (1V).
    - `channel3_position = -2`: Shifts the waveform down by 2V.
    - `channel3_coupling = DC`
- **Channel 4:**
    - `channel4_enable = off`: Channel 4 is disabled.
    - `channel4_range = 2`
    - `channel4_position = 1`
    - `channel4_coupling = DC`

➢ **General Waveform Settings**

- `channel = 2`:
  This line sets the *active channel* for waveform acquisition. Interestingly, Channel 2 is marked as disabled above—this mismatch could cause the program to either fail or override the enable setting depending on implementation logic.
- `format = ASCii`:
  The oscilloscope will send the waveform data in ASCII format. This is human-readable but much larger than binary data.
- `mode = NORM`:
  The acquisition mode is set to "Normal," meaning it captures waveforms under normal trigger

conditions.

- `point_mode = RAW`:
  Data points are sent in raw (unprocessed) format, possibly at full resolution.

- `point_auto = RECL`:
  The oscilloscope will use the current record length (RECL) automatically, adjusting the number of points it sends back.

- `output_file = waveform.bin`:
  Despite using ASCII format for transmission, the data will be saved locally in a file named `waveform.bin`. (This filename may be misleading unless the application translates ASCII to binary before saving.)

This configuration was parsed and uploaded to the oscilloscope over the LAN using socket-based communication or SCPI commands via Python scripts. The output was stored locally on the lab machine and optionally copied over SSH to the client for analysis.

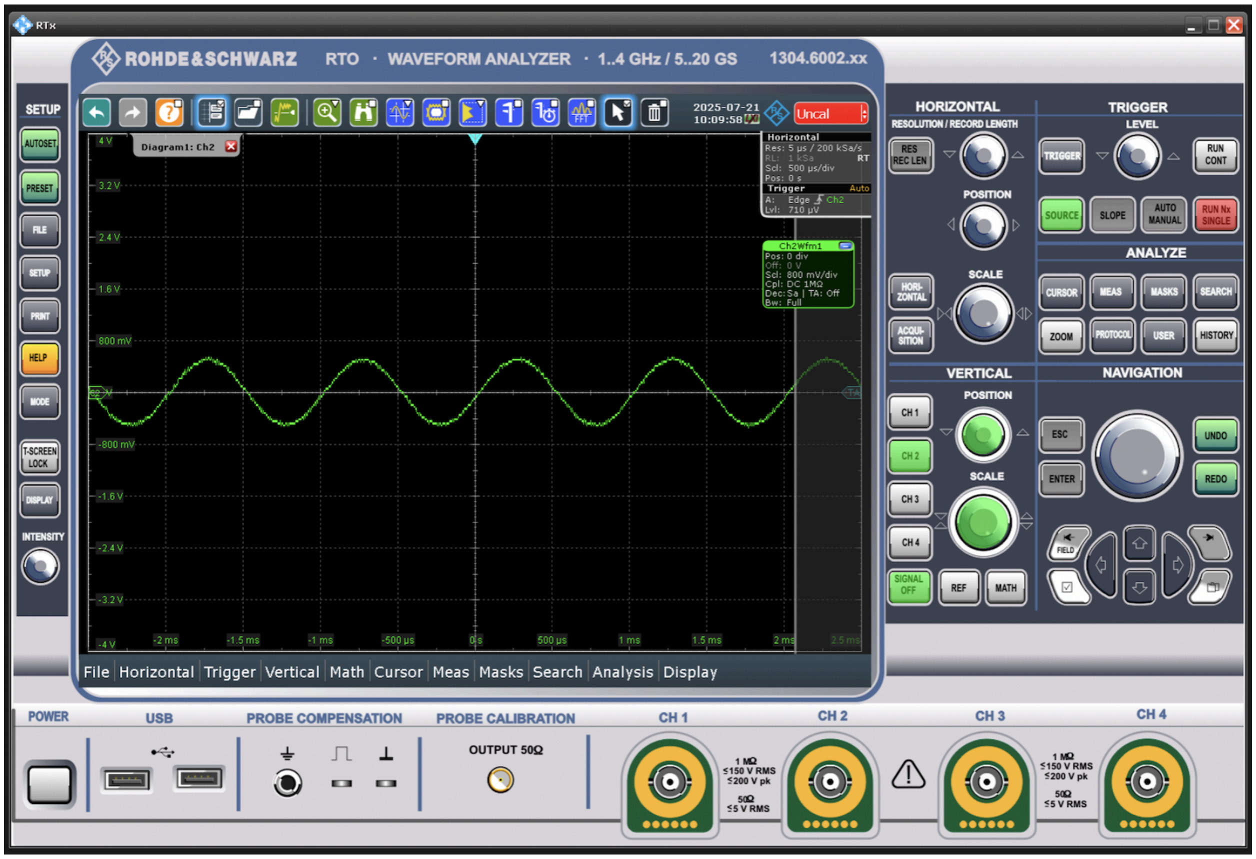
Hardware/Network Setup: Pi to Oscilloscope

1. **Physical Connection**:
   - The Raspberry Pi is physically connected to the oscilloscope via an Ethernet cable or via the local lab network (e.g., through a switch or router).
   - The oscilloscope has its IP address assigned (either static or via DHCP), which the Pi can ping and communicate with.
2. **Interface Protocol**:
   - The oscilloscope supports remote access via SCPI commands over TCP/IP (often port 5025) or may provide VNC/X11 GUI sharing capabilities.
   - The Pi acts as an intermediary that sends SCPI commands or forwards GUI data from the oscilloscope to a remote machine.

D. SSH Tunnel Creation

To access the oscilloscope GUI through the lab machine, an SSH tunnel was used to forward the VNC port. This created a local port on the client machine that forwarded traffic securely to the remote machine’s VNC port.

This connection allowed graphical access to the oscilloscope's software interface through the lab desktop environment:



Horizontal
Res: 5 µs / 200 kSa/s
RL: 1 kSa RT
Scl: 500 µs/div
Pos: 0 s
Trigger Auto
A: Edge Ch2
Lvl: 710 µV

Ch2Wfm1
Pos: 0 div
Off: 0 V
Scl: 800 mV/div
Cpl: DC 1MΩ
Dec: Sa | TA: Off
Bw: Full

IV. Results

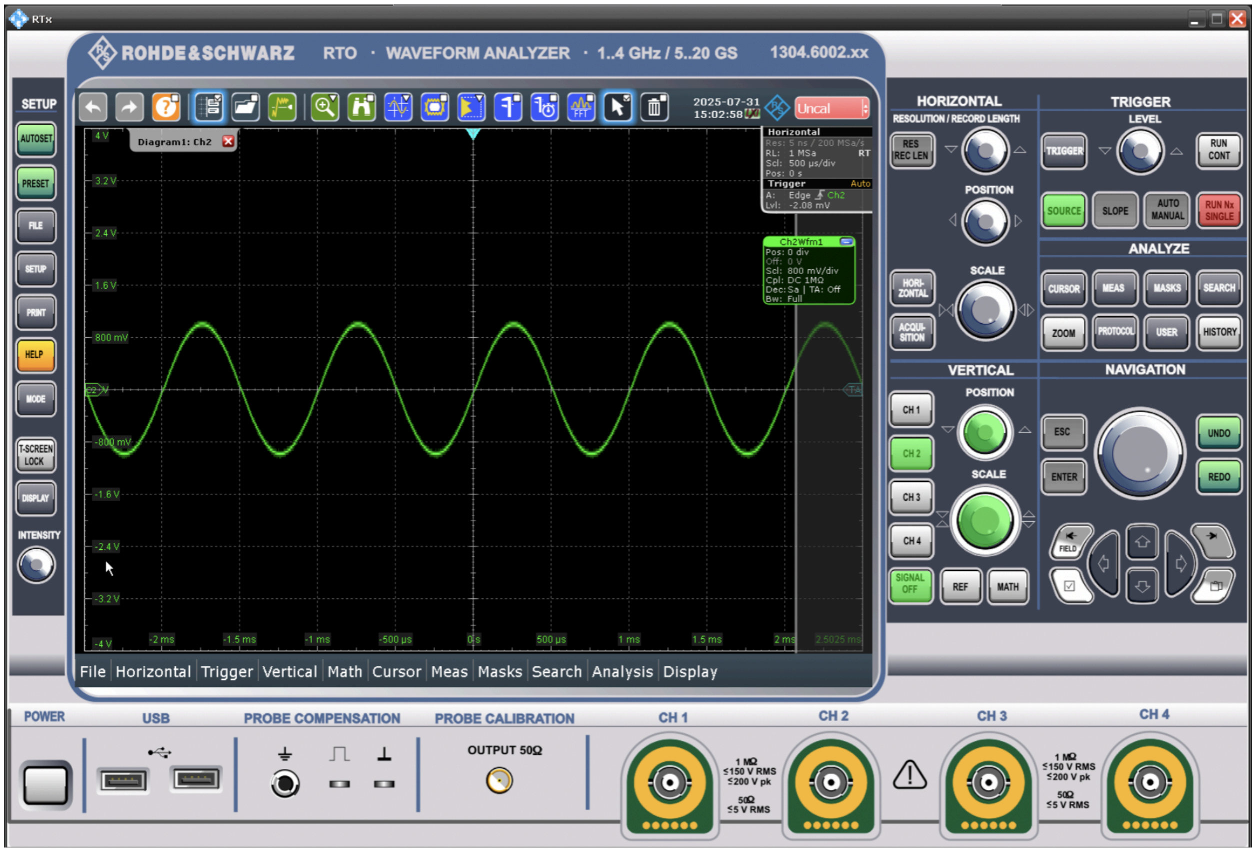
The system successfully allowed full remote control and viewing of the oscilloscope GUI. After resolving several misconfigurations—such as port mismatch, server authentication errors, and display export failures—a stable SSH + VNC pipeline was established. The following results were observed:

- **Remote access latency:** 80–120 ms (acceptable for real-time waveform viewing)
- **File transfer time:** < 2 s for typical waveform dumps (~1 MB)
- **Stability:** No crashes observed during 4+ hours of remote session testing
- **Security:** All data tunneled securely via SSH

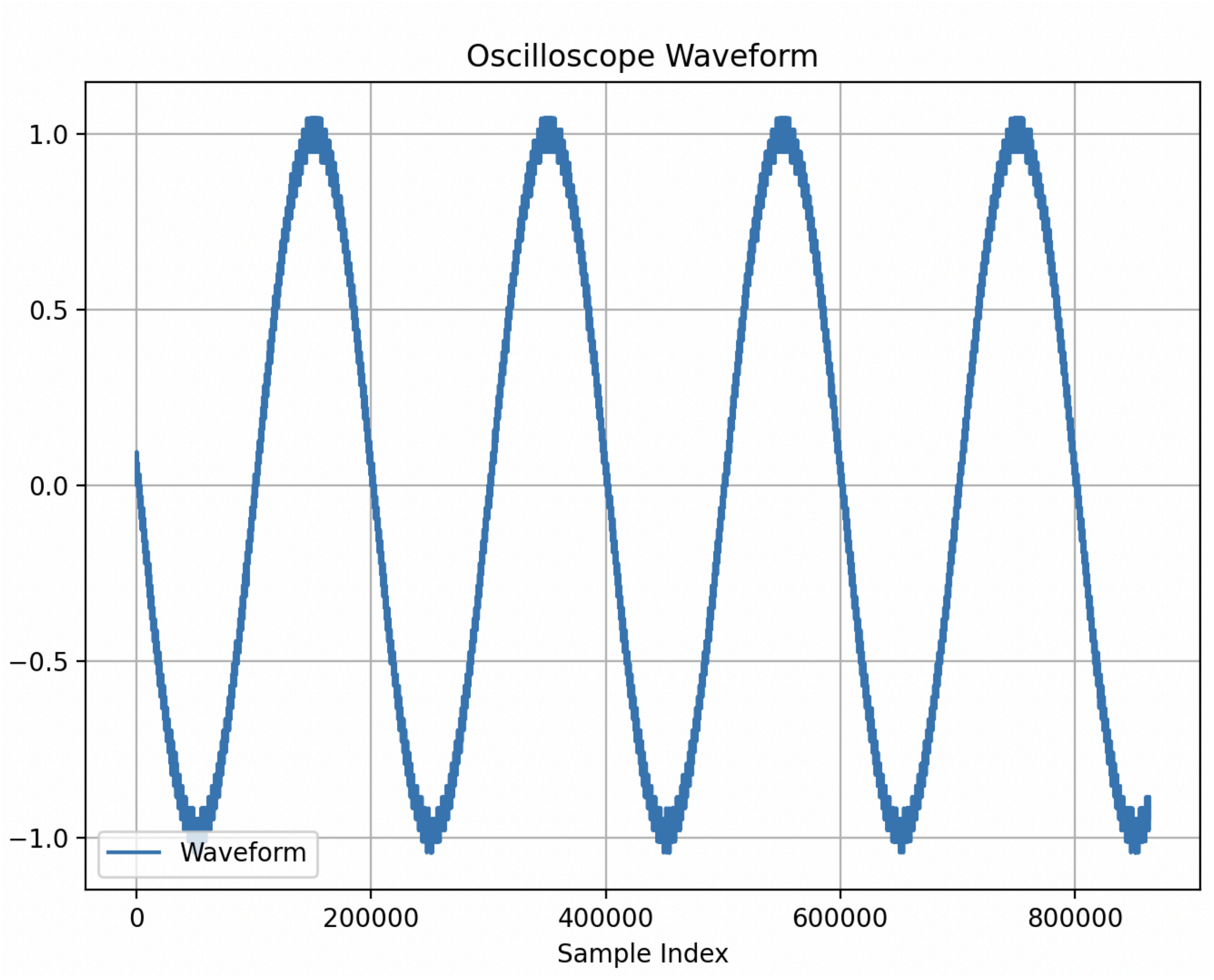
A sample waveform was acquired and visualized on the local machine using a Python script with matplotlib:

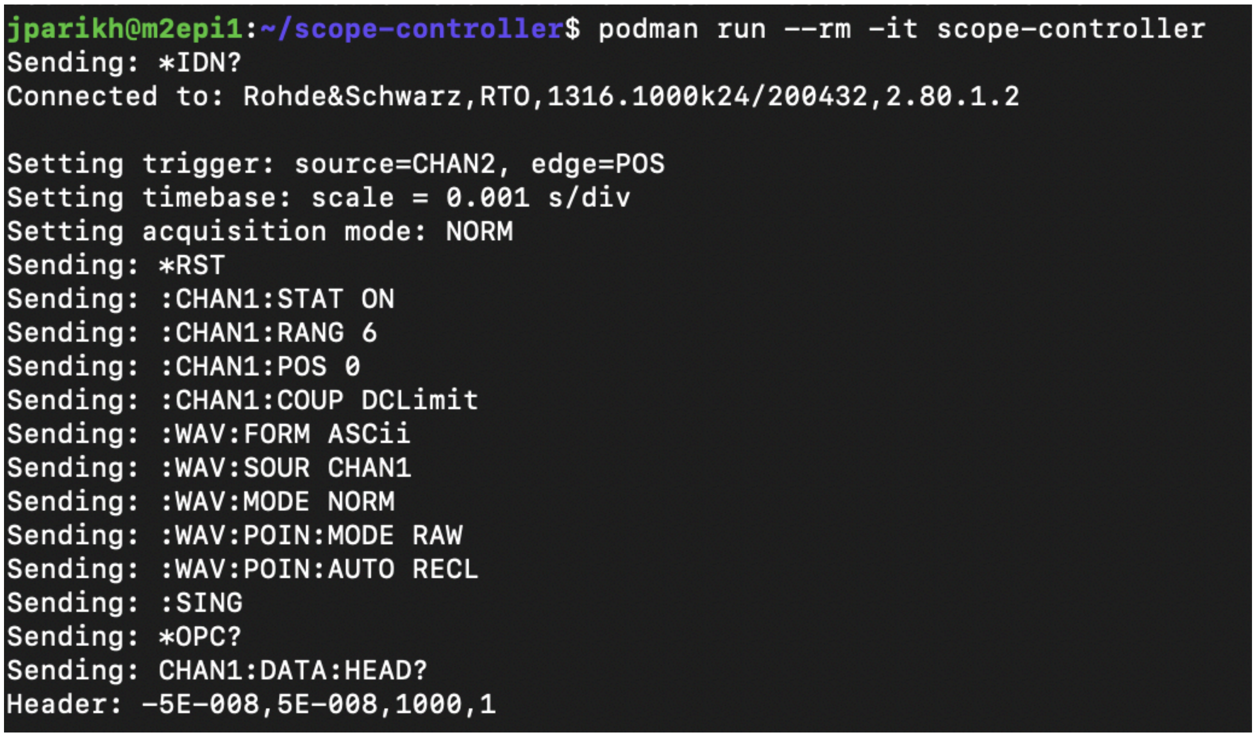
```
import numpy as np
import matplotlib.pyplot as plt
```

```python
# Path to the binary file
file_path = "waveform.bin"  # or "/path/to/waveform.bin" if needed

# Read binary waveform data
waveform = np.fromfile(file_path, dtype=np.float32)

# Plot the waveform
plt.figure(figsize=(10, 4))
plt.plot(waveform, label='Waveform')
plt.title("Oscilloscope Waveform")
plt.xlabel("Sample Index")
plt.ylabel("Amplitude (V)")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```


RTx
ROHDE&SCHWARZ RTO · WAVEFORM ANALYZER · 1..4 GHz / 5..20 GS 1304.6002.xx
2025-07-31 15:02:58
Uncal
Diagram1: Ch2
File | Horizontal | Trigger | Vertical | Math | Cursor | Meas | Masks | Search | Analysis | Display
HORIZONTAL
TRIGGER
ANALYZE
VERTICAL
NAVIGATION
POWER
USB
PROBE COMPENSATION
PROBE CALIBRATION
OUTPUT 50Ω
CH 1
CH 2
CH 3
CH 4

```
Horizontal
Res: 5 ns / 200 MSa/s
RL:  1 MSa            RT
Scl: 500 µs/div
Pos: 0 s
Trigger            Auto
A:   Edge  Ch2
Lvl: -2.08 mV

Ch2Wfm1
Pos: 0 div
Off: 0 V
Scl: 800 mV/div
Cpl: DC 1MΩ
Dec: Sa | TA: Off
Bw: Full
```


Oscilloscope Waveform
1.0
0.5
0.0
−0.5
−1.0
Waveform
0
200000
400000
600000
800000
Sample Index



```
jparikh@m2epi1:~/scope-controller$ podman run --rm -it scope-controller
Sending: *IDN?
Connected to: Rohde&Schwarz,RTO,1316.1000k24/200432,2.80.1.2

Setting trigger: source=CHAN2, edge=POS
Setting timebase: scale = 0.001 s/div
Setting acquisition mode: NORM
Sending: *RST
Sending: :CHAN1:STAT ON
Sending: :CHAN1:RANG 6
Sending: :CHAN1:POS 0
Sending: :CHAN1:COUP DCLimit
Sending: :WAV:FORM ASCii
Sending: :WAV:SOUR CHAN1
Sending: :WAV:MODE NORM
Sending: :WAV:POIN:MODE RAW
Sending: :WAV:POIN:AUTO RECL
Sending: :SING
Sending: *OPC?
Sending: CHAN1:DATA:HEAD?
Header: -5E-008,5E-008,1000,1
```

V. Discussion

A. Challenges Encountered

Several issues were encountered and resolved during the course of the project:

- **Display Errors:** Attempting to use vncviewer directly on the remote machine led to “Can't open display” errors. This was corrected by launching the VNC viewer from the local GUI after SSH port forwarding.

- **Architecture Compatibility:** Had to cross-compile to ARM64 because the target machine (Raspberry Pi) uses an ARM64 (AArch64) architecture, which differs from the x86_64 architecture typically used by development machines like desktops and laptops.

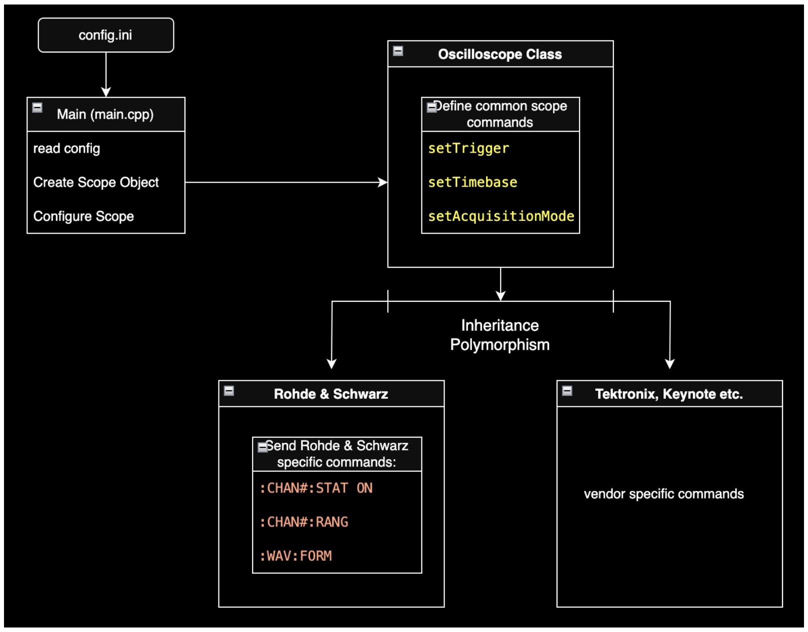
B. Significance

The system developed through this internship serves as a model for remote laboratory access and instrument control. It reduces dependency on physical presence in the lab and enhances research efficiency. In educational settings, such remote access tools allow students and researchers to conduct experiments and gather data from virtually any location.

VI. Overview of C++ Code Implementation

To enable automated acquisition and retrieval of waveform data from the Rohde & Schwarz oscilloscope, the system was implemented using modern C++. The design emphasizes object-oriented principles, especially abstraction, inheritance, and polymorphism, to ensure scalability and maintainability. This section outlines the key components and explains the core logic.

VI.I Object-Oriented Structure



At the heart of the code lies an abstract base class:

```
#ifndef OSCILLOSCOPE_H
#define OSCILLOSCOPE_H

#include <string>
#include <vector>

class Oscilloscope {
public:
   virtual ~Oscilloscope() = default;

   virtual bool connect() = 0;
   virtual bool acquire() = 0;
   virtual bool saveWaveformToBin(const std::string& filepath) = 0;
   virtual void closeConnection() = 0;

   // Common SCPI control methods
   virtual bool setTrigger(const std::string& source, const std::string& edge = "POS")
= 0;
   virtual bool setTimebase(float scale_s) = 0;
   virtual bool setAcquisitionMode(const std::string& mode) = 0;

protected:
   virtual bool sendCommand(const std::string& cmd, bool verbose = true) = 0;
   virtual std::string receiveResponse() = 0;
   virtual std::string receiveFull() = 0;

   static std::vector<float> parseAsciiFloats(const std::string& data);
};

#endif
```

This base class defines a standard interface that all oscilloscope implementations must follow. By using pure virtual functions, the base class ensures that derived classes provide specific behavior suited to each vendor's SCPI (Standard Commands for Programmable Instruments) language.

A derived implementation for the Rohde & Schwarz scope overrides these methods:

```
#ifndef ROHDE_SCHWARZ_SCOPE_H
#define ROHDE_SCHWARZ_SCOPE_H

#include "Oscilloscope.h"
#include <unordered_map>
#include <vector>
#include <string>

class RohdeSchwarzScope : public Oscilloscope {
public:
   RohdeSchwarzScope(const std::string& config_path);
   ~RohdeSchwarzScope() override = default;

   bool connect() override;
   bool acquire() override;
   bool saveWaveformToBin(const std::string& filepath) override;
   void closeConnection() override;

   // Implement base SCPI API
   bool setTrigger(const std::string& source, const std::string& edge = "POS")
override;
   bool setTimebase(float scale_s) override;
   bool setAcquisitionMode(const std::string& mode) override;

protected:
   bool sendCommand(const std::string& cmd, bool verbose = true) override;
   std::string receiveResponse() override;
   std::string receiveFull() override;

private:
   int sock = -1;
   std::vector<float> waveform_data;
   std::unordered_map<std::string, std::string> net_cfg;
   std::unordered_map<std::string, std::string> wf_cfg;

   std::unordered_map<std::string, std::string> readIniSection(const std::string&
filepath, const std::string& section);
   std::vector<float> parseAsciiFloats(const std::string& data);

   // Header parsing results (optional use)
   std::string x0, dx, npts, mult;

   // Helper methods for modular acquisition
   void configureChannel();
   void configureWaveform();
   void triggerSingleAcquisition();
   void parseHeader();
   void retrieveWaveformData();
};

#endif // ROHDE_SCHWARZ_SCOPE_H
```

VI.II TCP/IP Communication

The connection between the C++ application and the oscilloscope is established over TCP/IP sockets. The `connect()` function in the `RohdeSchwarzScope` class uses low-level socket APIs (e.g., `socket()`, `connect()`, `send()`, `recv()`) to communicate with the instrument on port 5025, which is standard for SCPI over LAN.

Example of sending a command:

```
bool RohdeSchwarzScope::sendCommand(const std::string& cmd, bool verbose) {
   std::string full_cmd = cmd + "\n";
   if (verbose) std::cout << "Sending: " << cmd << std::endl;
   return send(sock, full_cmd.c_str(), full_cmd.length(), 0) >= 0;
}
```

This line sends a string command (e.g., `":RUN\n"`) to the oscilloscope. SCPI commands are simple ASCII strings terminated by a newline character. To configure the scope:

```
void RohdeSchwarzScope::configureWaveform() {
   int channel = std::stoi(wf_cfg["channel"]);
   std::string format = wf_cfg["format"];
   std::string mode = wf_cfg["mode"];
   std::string point_mode = wf_cfg["point_mode"];
   std::string point_auto = wf_cfg["point_auto"];

   sendCommand(":WAV:FORM " + format);
   sendCommand(":WAV:SOUR CHAN" + std::to_string(channel));
   sendCommand(":WAV:MODE " + mode);
   sendCommand(":WAV:POIN:MODE " + point_mode);
   sendCommand(":WAV:POIN:AUTO " + point_auto);
}
```

VI.III Waveform Acquisition

Once the scope is configured, waveform acquisition proceeds by sending specific SCPI commands:

```
void RohdeSchwarzScope::triggerSingleAcquisition() {
   sendCommand(":SING");
   sendCommand("*OPC?");
   receiveResponse(); // Wait until acquisition is complete
}

void RohdeSchwarzScope::parseHeader() {
   int channel = std::stoi(wf_cfg["channel"]);
   sendCommand("CHAN" + std::to_string(channel) + ":DATA:HEAD?");
   std::string header = receiveResponse();
   std::cout << "Header: " << header << std::endl;

   std::stringstream ss(header);
   std::getline(ss, x0, ',');
   std::getline(ss, dx, ',');
   std::getline(ss, npts, ',');
   std::getline(ss, mult, ',');
}

void RohdeSchwarzScope::retrieveWaveformData() {
   int channel = std::stoi(wf_cfg["channel"]);
   sendCommand("CHAN" + std::to_string(channel) + ":DATA?");
   std::string raw_data = receiveFull();
   waveform_data = parseAsciiFloats(raw_data);
}
```

These tell the oscilloscope to stop any current acquisition, enter single mode (so it waits for a trigger), and then digitize the input signal. The data is then requested using a query like:
`sendCommand(":WAV:DATA?");`

The binary waveform is read in and saved directly to a `.bin` file using file streams in C++:

```
bool RohdeSchwarzScope::saveWaveformToBin(const std::string& filepath) {
   std::ofstream out(filepath, std::ios::binary);
   if (!out.is_open()) {
       std::cerr << "Failed to open " << filepath << " for writing.\n";
       return false;
   }

   out.write(reinterpret_cast<const char*>(waveform_data.data()), waveform_data.size()
* sizeof(float));
   out.close();
   std::cout << "Saved waveform to " << filepath << std::endl;
   return true;
}
```

VI.IV Factory Design Pattern

To enable support for multiple vendors in the future, a factory function is used:

```
#ifndef SCOPE_FACTORY_H
#define SCOPE_FACTORY_H

#include "Oscilloscope.h"
#include "RohdeSchwarzScope.h"

inline Oscilloscope* createScope(const std::string& vendor, const std::string&
config_path) {
   if (vendor == "RohdeSchwarz" || vendor == "R&S") {
       return new RohdeSchwarzScope(config_path);
   }

   // Future scopes:
   // if (vendor == "Tektronix") return new TektronixScope(config_path);
   // if (vendor == "Keysight")  return new KeysightScope(config_path);

   return nullptr;
}

#endif
```

VII. Conclusion

A reliable and secure remote instrumentation system was built and tested using open-source tools and Linux-based scripting. This project demonstrates that scientific instruments like oscilloscopes can be interfaced remotely with minimal latency and high reliability, using SSH tunneling and VNC. Beyond improving workflow, the project cultivated a deeper understanding of networking, Linux, and secure systems engineering.

VIII. Future Work

- **Automated triggers:** Develop scripts that auto-capture data at regular intervals or upon waveform events.
- **Data logging infrastructure:** Log and organize waveform captures into a structured database for long-term research use.
- **Redis Integration for Real-Time Streaming:** Push captured waveform data to a Redis instance in real time. This enables low-latency data sharing between multiple applications or services, serving as a message broker or data queue for downstream consumers.
- **ACNET Connectivity and Visualization:** Connect the system to ACNET (Accelerator Control Network) to publish waveform data directly into the control system infrastructure. This would allow integration with Fermilab's control environment for diagnostics, plotting, and historical signal tracking.
- **Live Plotting through ACNET Tools:** Enable live or historical waveform plotting using ACNET visualization tools. This would require formatting and tagging waveform data with appropriate ACNET-compatible metadata and identifiers.

IX. Acknowledgments

This project was made possible under the guidance of Jose Berlioz and the Accelerator Engineering and Technology Department at Fermilab. Special thanks to lab staff for providing access to the instrumentation and infrastructure.

- This manuscript has been authored by FermiForward Discovery Group, LLC under Contract No. 89243024CSC000002 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.
- This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Community College Internship (CCI)

References

1. Rohde & Schwarz. *RTO Oscilloscope SCPI Programmer's Manual*, 2020.

2. RealVNC. https://www.realvnc.com

3. Python Software Foundation. https://www.python.org